ns-3 NETWORK SIMULATOR

ICPS

Published by ACM

WNS3 2016

Workshop on ns-3

Seattle, Washington, USA

June 15-16, 2016

ICST.ORG

EAI European Alliance for Innovation

ns-3 NETWORK SIMULATOR

ICPS Published by ACM

WNS3 2016

General Chair
Thomas Henderson

Technical Program Committee Co-Chairs
Brian Swenson and Hajime Tazaki

Proceedings Chair
Eric Gamess

Workshop on ns-3

Seattle, Washington, USA
June 15-16, 2016

ISBN: 978-1-4503-4216-2

ICST.ORG

EAI European Alliance for Innovation

W

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, New York 10121-0701

ACM ISBN: 978-1-4503-4216-2

Program Committee

General Chair

Thomas Henderson
University of Washington, USA

Technical Program Committee Co-Chairs

Brian Swenson	Hajime Tazaki
Georgia Institute of Technology, USA	IIJ Innovation Institute, Japan

Proceedings Chair

Eric Gamess
Universidad Central de Venezuela, Venezuela

Technical Program Committee Members

Alexander Afanasyev	University of California at Los Angeles, USA
Ramón Agüero	Universidad de Cantabria, Spain
Peter D. Barnes Jr.	Lawrence Livermore National Laboratory, USA
Bow-Nan Cheng	Massachusetts Institute of Technology, USA
Sebastien Deronne	Alcatel-Lucent Bell, Belgium
David Ediger	Georgia Institute of Technology, USA
Eric Gamess	Universidad Central de Venezuela, Venezuela
Lorenza Giupponi	Centre Tecnològic Telecomunicacions Catalunya, Spain
Thomas Henderson	University of Washington, USA
Sam Jansen	StarLeaf, United Kingdom
Kevin Jin	Illinois Institute of Technology, USA
Mathieu Lacage	Alcmeon, France
Jason Liu	Florida International University, USA
Margaret Loper	Georgia Institute of Technology, USA
Marco Miozzo	Centre Tecnològic Telecomunicacions Catalunya, Spain
Carlos Moreno	Universidad Central de Venezuela, Venezuela
David Nicol	University of Illinois, USA
Luis Perrone	Bucknell University, USA
Ken Renard	Army Research Lab, USA
Manuel Ricardo	INESC Porto, Portugal
Damien Saucez	Inria, France
Brian Swenson	Georgia Institute of Technology, USA
Mohit Tahiliani	National Institute of Technology Karnataka, India
Cristiano Tapparello	University of Rochester, USA
Hajime Tazaki	IIJ Innovation Institute, Japan
Thierry Turletti	Inria, France

Preface

The 2016 Workshop on ns-3 (WNS3 2016) is the eighth edition of an annual series of workshops around the discrete-event network simulator known as “ns-3”. The workshop aims to gather ns-3 users and developers, together with networking simulation practitioners and users, and developers of other network simulation tools, to discuss the ns-3 simulator and related activities. ns-3 is a tool used for performance evaluation in computer networks, and this workshop offers a venue for those involved with extending or testing the tool itself to publish original work in this regard. The workshop is sponsored by the ns-3 Consortium and organized as part of a full week of activities including also training sessions, the Consortium Annual Meeting, and developer discussions.

WNS3 2016 was hosted by the Electrical Engineering department of the University of Washington in Seattle, Washington, and held on 15-16 June, 2016. Thomas R. Henderson served as General Chair of this edition of the workshop, and Eric Gamess, with the Central University of Venezuela, acted as the Proceedings Chair. The technical program committee, co-chaired by Dr. Brian Swenson and Dr. Hajime Tazaki, included 26 international reviewers from prestigious universities, laboratories, and manufacturers with a large experience in ns-3 and discrete-event network simulation research in general. The workshop was organized with technical cooperation of the Association for Computing Machinery (ACM), the European Alliance of Innovation (EAI), and the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (ICST), to which we extend our gratitude and acknowledgment.

The organizers received 28 submissions from around the world, out of which 18 papers were selected for presentation and publication. Each contribution submitted to the workshop was reviewed by at least two technical program committee members, with most papers receiving three reviews. Authors of accepted contributions were encouraged to provide links to code and scripts that would allow future readers to reproduce the results of their work. Accepted papers can be categorized into one of three themes: (1) twelve papers focused on the design, implementation process, and performance evaluation for extensions to ns-3 models, (2) three papers were primarily concerned with performance analysis, improvements, and testing of ns-3 itself and its Direct Code Execution environment, and (3) three papers described frameworks for integrating ns-3 with other software frameworks or libraries providing network coding, power grid planning and modeling, and software-defined networking.

We thank the WNS3 2016 members of the program committee and the organization committee, invited lectures and speakers, authors, teachers and students, and in general, to all those who contributed with their valuable support to carry out the successful completion of this important responsibility, keeping at the highest levels the relevance of WNS3.

Brian Swenson	Hajime Tazaki	Thomas R. Henderson
Georgia Tech Research Institute	IIJ Innovation Institute	University of Washington
TPC Co-Chair	TPC Co-Chair	General Chair

Table of Contents

Table of Contents

Design and Implementation of the Traffic Control Module in ns-3

Pasquale Imputato, Stefano Avallone
Università degli Studi di Napoli "Federico II"
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Via Claudio 21, 80125, Napoli
{pasquale.imputato, stefano.avallone}@unina.it

ABSTRACT

The Linux networking subsystem relies on the Traffic Control infrastructure to process both the incoming and the outgoing packets. One of the most important components of the Traffic Control is the queueing discipline, whose role is to store packets waiting for transmission and select the next packet to pass to the network interface. The Linux Traffic Control enables to perform scheduling, shaping of the egress traffic, policing of the ingress traffic, and dropping of both ingress and egress traffic.

In this paper, we present the design and implementation of the Traffic Control layer as an additional module in ns-3. This layer sits in between the netdevices and the network layer. We also present the design and implementation of the base class introduced to model a queueing discipline. Finally, we report a preliminary validation of our work, consisting in a number of tests that properly compare the new stack to the previous one.

CCS Concepts

•**Networks** → **Network simulations;** *Network performance modeling;*

Keywords

ns-3, Traffic Control, Queueing Discipline, Active Queue Management, Explicit Congestion Notification

1. INTRODUCTION

The Traffic Control infrastructure [1] of the Linux kernel enables to perform a number of actions on both outgoing packets, before they are handed to the netdevice for transmission, and incoming packets, before they are processed by the network layer protocols. In this paper, we focus on the transmission path taken by packets. Once the output interface and the next hop for an outgoing packet have been selected, the packet is enqueued into a queueing discipline (queue disc), which determines how the packet will be treated. A number of queue discs have been implemented in the Linux kernel, including simple FIFO (First In First Out) schedulers, such as pfifo_fast, fair-queuing schedulers, such as Deficit Round Robin (DRR) [7], Stochastic Fairness Queuing (SFQ) [5] and Quick Fair Queuing (QFQ+) [2], and Active Queue Management (AQM) algorithms, such as Random Early Drop (RED) [3] and Controlled Delay (CoDel) [6]. A queue disc stores the packets waiting for transmission and decides which packet to pass to the network interface when it is requested to dequeue a packet.

When a queue disc is requested to dequeue a packet depends on the implemented flow control mechanisms. Basically, enqueuing a packet into a queue disc triggers a number of consecutive requests of dequeuing a packet. This process can be halted by the netdevice driver (or by the network stack itself) by putting its netdevice queue into a *stop* state. The netdevice driver usually stops its transmission queue when it is full or its occupancy is above a given threshold. When the netdevice is able to receive packets again, the driver can *start* its transmission queue. Additionally, the netdevice driver can *wake* a queue disc, i.e., request it to dequeue a packet, when its transmission queue is empty or its occupancy is below a given threshold.

Currently, ns-3 is lacking an equivalent of the Linux Traffic Control infrastructure. No flow control mechanism is implemented and packets are only stored in the netdevice transmission queues. Consequently, AQM algorithms such as RED and CoDel can only manage the packets stored in the netdevice queues, which is not what happens in Linux. This paper presents the work done to introduce an equivalent of the Linux Traffic Control infrastructure into ns-3. We believe that our work will allow researchers to carry out more realistic simulations and to evaluate AQM algorithms more precisely.

The remainder of this paper is organised as follows. In section 2 we provide an overview of the Linux Traffic Control and of the current status of ns-3. Section 3 describes the model and design of the proposed Traffic Control module for ns-3. Section 4 presents the Traffic Control helper and some usage examples. Section 5 describes the experiments we performed with the new architecture and the results we obtained. In section 6 we conclude our work.

2. BACKGROUND

In this section, we describe the Linux Traffic Control infrastructure and the ns-3 queue system.

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915382

2.1 Linux Traffic Control

The Traffic Control is a component of the network subsystem of the Linux kernel. This component supports multiple operations needed to provide Quality of Service (QoS), including:

- shaping and scheduling of egress traffic;
- policing of ingress traffic;
- dropping of both ingress and egress traffic.

The Traffic Control relies on three fundamental components to perform the above mentioned operations:

- queue discs;
- classes;
- filters.

A queue disc can be added to every network interface in the Linux kernel and determines how packets outgoing from the interface are treated. A simple queue disc is fifo, which does no processing and is a pure FIFO queue with queue limit expressed in packets or bytes. It stores a packet when the network interface cannot handle it immediately. The default queue disc in Linux is pfifo_fast, which consists of a three band queue acting as a priority queue. The priority assigned to each packet may depend on the Type of Service (ToS) value or Diffserv Codepoint (DSCP) carried by the packet. More complex queue discs are available in the Traffic Control component. A typical taxonomy divides queue discs in classful (i.e., support classes) and classless (i.e., do not support classes). A classful queue disc can contain multiple classes, each of which has a child queue disc attached. Each class can be configured with distinct parameter values, so as to reserve a distinct treatment to different traffic classes. For instance, the prio queue disc is a container for a configurable number of classes which are served in order of priority. A packet filter can be used by a classful queue disc to classify packets based on different criteria. The most advanced filter available is u32 that can use anything in the header for classification.

More recently, after the appearance of multi-queue netdevices (such as Wifi), some multi-queue aware queue discs have been introduced. Multi-queue aware queue discs handle as many queues (or queue discs – without using classes) as the number of transmission queues used by the netdevice on which the queue disc is installed. An attempt is made, also, to enqueue each packet in the "same" queue both within the queue disc and within the netdevice.

2.2 ns-3 Queue System

In this section, we analyze ns-3 for what concerns the queuing system adopted at the network and netdevice layers.

The network layer has no queuing system. In case the IPv4 stack is employed (the same holds for the IPv6 stack), packets generated from the upper layers are passed to the `Ipv4L3Protocol`, which determines the right `Ipv4Interface` for packet forwarding. Then, the `Ipv4Interface` passes the packet to the corresponding netdevice. Thus, it is not possible to differentiate traffic at this layer and hold packets whose transmission to the netdevice failed (and are therefore dropped). Netdevices store the packets waiting for transmission in a queue. Such a queue contains packets with the data link header already added and is modelled through the base class `Queue`. Derived classes are `DropTailQueue`, `RedQueue` and `CoDelQueue`. `DropTailQueue` is a classical first in first out limited queue. The two models of AQM, `RedQueue` and `CoDelQueue`, both presently derive from class `Queue` and can be only installed on the Csma and PointToPoint netdevices but not on Wifi and LTE netdevices. The reason is that Wifi and LTE do not use subclasses of the base class `Queue` to implement their netdevice queues.

The ns-3 netdevices do not support any form of flow control between the network and netdevice layers. Indeed, netdevices have no means to request the network layer to stop sending packets and all the packets passed to the netdevices when their queues are full are inevitably lost.

The base class `Queue` does not provide easy visibility of the IP and transport headers. The non trivial access or modification of the IP header have hindered the addition of the Explicit Congestion Notification (ECN) support in ns-3 [8]. The ns-3 ECN support should remove the L2 header (of different length for a different netdevice) then remove the L3 header and apply the ECN policy. Also, the non simple access to the transport header, e.g. TCP, has hindered the ns-3 support to the recent internet aware queue discs such as FlowQueue-CoDel (FQ-CoDel). Those queue discs need access to the 5-tuple of IP protocol, source and destination IP addresses and port numbers.

3. DESIGN AND IMPLEMENTATION

In this section, we describe the model of the Traffic Control module, its design and several challenges that we encountered during its implementation.

3.1 Model Description

In order to add support for the features described in the previous section, we decided to introduce a new layer that sits above the netdevice and below the IP forwarding layer. The main consequence is that it requires flow control between the new Traffic Control layer and each of the netdevice queues. For each netdevice queue, it is necessary to keep a status bit which indicates if further packets can be passed to the netdevice for the transmission. The netdevice (or the network layer) can stop the passing of further packets when a resource becomes unavailable (e.g. the netdevice queue is full) and wake up the upper layer when the resource becomes available again.

Packets received by the Traffic Control layer for transmission to a netdevice can be passed to a queue disc to perform scheduling and policing. A netdevice can have a single (root) queue disc installed on it. Installing a queue disc on a netdevice is not mandatory. If a netdevice does not have a queue disc installed on it, the Traffic Control layer sends the packets directly to the netdevice. This is the case, for instance, of the loopback netdevice.

As in Linux, a queue disc may contain distinct elements:

- queues, which actually store the packets waiting for transmission;
- classes, which allow to reserve a different treatment to different packets;

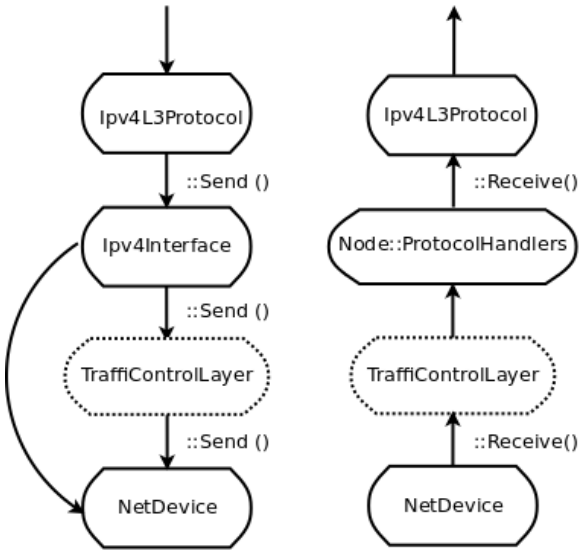


Figure 1: The send and receive path on internet enabled nodes after the introduction of the Traffic Control layer (IPv4 case).

- filters, which determine the queue or class which a packet is destined to.

Notice that a child queue disc must be attached to every class and a packet filter is only able to classify packets of a single protocol. Also, while in Linux some queue discs (e.g., FQ-CoDel) use an internal classifier and do not make use of packet filters, in ns-3 every queue disc including multiple queues or multiple classes needs an external filter to classify packets (this is to avoid having the Traffic Control module depend on other modules such as internet).

The Traffic Control layer interacts with a queue disc in a simple manner: after requesting to enqueue a packet, the Traffic Control layer requests the queue disc to "run", i.e., to dequeue a set of packets, until a predefined number ("quota") of packets is dequeued or the netdevice stops the queue disc. A netdevice may stop the queue disc when its transmission queue(s) is/are (almost) full. Also, a netdevice may wake the queue disc when its transmission queue(s) is/are (almost) empty. Waking a queue disc is equivalent to make it run.

3.2 Design

The new internet enabled node stack with Traffic Control is illustrated in Figure 1 (for the IPv4 case, the IPv6 case is similar). A `TrafficControlLayer` object is aggregated to every internet enabled node. The new layer intercepts packets that transit both in the input and output directions. Currently, scheduling of outgoing packets is supported, while policing of incoming packets is not supported (since the equivalent of the Linux ingress queue disc has not been implemented yet). The IPv{4,6} interfaces uses the aggregated `TrafficControlLayer` object to send packets down, instead of calling `NetDevice::Send()` directly. After the analysis and the process of the packet, when the flow control mechanism allows it, `TrafficControlLayer` will call the `Send()` method on the right netdevice. The IPv{4,6} interfaces call the `NetDevice::Send()` directly only in the case of packets destined to the loopback interface. To receive packets, instead, the callback chain, that (in the past) involved the node protocol handlers and the netdevice, is extended to involve `TrafficControlLayer`.

A `TrafficControlLayer` object holds a reference (smart pointer) to the objects representing the queue discs installed on each netdevice of the node. An abstract base class, class `QueueDisc`, is subclassed to implement specific queue discs. A subclass is required to implement the following methods:

- `bool DoEnqueue (Ptr<QueueDiscItem> item)`: enqueue a packet;
- `Ptr<QueueDiscItem> DoDequeue (void)`: dequeue a packet;
- `Ptr<const QueueDiscItem> DoPeek (void) const`: peek a packet;
- `bool CheckConfig (void) const`: check if the configuration is correct.

The base class `QueueDisc` implements:

- methods to add/get a single queue, class or filter and methods to get the number of installed queues, classes or filters;
- a Classify method which classifies a packet by processing the list of filters until a filter able to classify the packet is found;
- methods to extract multiple packets from the queue disc, while handling transmission (to the netdevice) failures by requeuing packets.

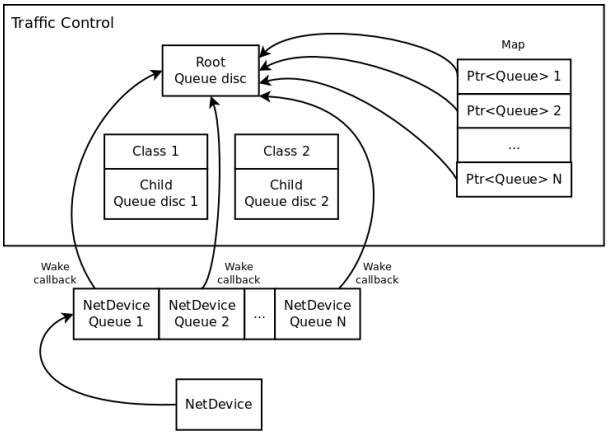
The base class `QueueDisc` holds the list of attached queues, classes and filter by means of three vectors accessible through attributes (InternalQueueList, QueueDiscClassList and PacketFilterList).

Internal queues are implemented as (subclasses of) `Queue` objects. A Queue stores `QueueItem` objects, which consist of just a `Ptr<Packet>`. Since a queue disc has to store at least the destination address and the protocol number for each enqueued packet, a new class, `QueueDiscItem`, is derived from `QueueItem` to store such additional information for each packet. Thus, internal queues are implemented as `Queue` objects storing `QueueDiscItem` objects. Also, there could be the need to store further information depending on the network layer protocol of the packet. For instance, for IPv4 and IPv6 packets it is needed to separately store the header and the payload, so that header fields can be manipulated, e.g., to support ECN. To this end, `Ipv4QueueDiscItem` and `Ipv6QueueDiscItem` are derived from `QueueDiscItem` to additionally store the packet header and provide protocol specific operations such as ECN marking.

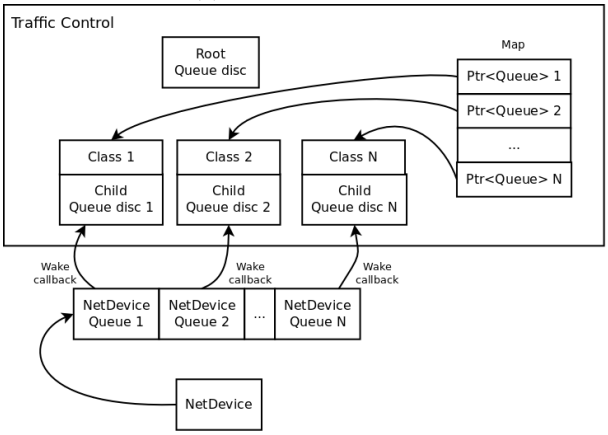
Classes are implemented via the `QueueDiscClass class`, which just consists of a pointer to the attached queue disc. Such a pointer is accessible through the queue disc attribute. Classful queue discs needing to set parameters for their classes can subclass `QueueDiscClass` and add the required parameters as attributes.

An abstract base class, `PacketFilter`, is subclassed to implement specific filters. Subclasses are required to implement two virtual private pure methods:

- `bool CheckProtocol (Ptr<QueueDiscItem> item) const`: check whether the filter is able to classify packets of the same protocol as the given packet;



(a) Classful queue disc



(b) Multi-queue aware queue disc

Figure 2: Queue discs in Traffic Control.

- `int32_t DoClassify (Ptr<QueueDiscItem> item) const`: actually classify the packet.

`PacketFilter` provides a public method, `Classify`, which first calls `CheckProtocol` to check that the protocol of the packet matches the protocol of the filter and then calls `DoClassify`. Specific filters subclassed from `PacketFilter` should not be placed in the Traffic Control module but in the module corresponding to the protocol of the classified packets.

In Linux, information about the status of a transmission queue of a netdevice is stored in the struct `netdev_queue`, which includes a `qdisc` field that is mainly used to solve the following problems:

- if a netdevice transmission queue is (almost) empty, identify the queue disc to wake;
- if a packet will be enqueued in a given netdevice transmission queue, identify the queue disc which the packet must be enqueued into.

The latter problem arises because Linux attempts to determine the netdevice transmission queue which a packet will be enqueued into before passing the packet to a queue disc. This is done by calling a specific function of the netdevice driver, if implemented, or by employing fallback mechanisms (such as hashing of the addresses) otherwise. The identifier of the selected netdevice transmission queue is stored in the `queue_mapping` field of the struct `sk_buff`, so that both the queue disc and the netdevice driver can get the same information. In ns-3, such identifier is stored in the `m_txq` member of the `QueueDiscItem` class.

Concerning the `qdisc` field of the struct `netdev_queue` in Linux, such a field cannot be similarly stored in a object `NetDeviceQueue`, because it would make the network module depend on the Traffic Control module. Instead, this information is stored in the `TrafficControlLayer` object aggregated to each node. In particular, a `TrafficControlLayer` object holds a map which stores, for each netdevice, a vector of `Ptr<QueueDisc>`. The size of such a vector is the number of netdevice transmission queues and each element of this vector is a pointer to the queue disc to activate when the above problems occur. The `SetRootQueueDiscOnDevice` method takes care of configuring such a map, based on the wake mode of the root queue disc. If the wake mode of the root queue disc is `WAKE_ROOT`, then all the elements of the vector are pointers to the root queue disc. If the wake mode of the root queue disc is `WAKE_CHILD`, then each element of the vector is a pointer to a distinct child queue disc. This requires that the number of child queue discs matches the number of netdevice queues. It follows that the wake mode of a classless queue disc must necessarily be `WAKE_ROOT`. These two configurations are illustrated in Figure 2.

Finally, we mention that the queue disc installed on a netdevice, along with the associated packet filters, classes and internal queues, can be removed by calling the method `DeleteRootQueueDiscOnDevice` of the `TrafficControlLayer` class.

3.3 Implementation Issues

The `Ipv{4,6}Interface` add the IP header to the packet before passing the packet to the underlying layer. Receving a packet with the IP header already attached makes it inefficient for the Traffic Control layer to manipulate the header, e.g., to perform ECN markings. For this reason, we changed the behavior of the internet stack so that the IP header and the IP payload of a packet are sent separately to the Traffic Control layer. This required modifications both to IPv4 (L3 protocol, ARP cache, ARP L3 protocol) and IPv6 (L3 protocol, extensions, ICMPv6, NDisc cache). The IP header is now added to the packet after the packet is dequeued from the queue disc.

The Traffic Control module cannot depend on the internet module, in order to avoid that future, alternative to internet, L3 modules have to depend on internet (through the dependency on Traffic Control) and to avoid a circular dependency (given that internet depends on Traffic Control). As a consequence, the Traffic Control layer cannot manipulate IP headers, which is necessary, e.g., to perform ECN marking, or filter packets based on the content of the IP header. As described earlier, this problem has been solved by enqueuing packets as (pointer to) `QueueDiscItem` objects which are actually either `Ipv4QueueDiscItem` or `Ipv6QueueDiscItem` objects. Likewise, using an abstract `PacketFilter` class allowed us to define protocol specific packet filters in the respective modules instead of in the Traffic Control module.

Other minor issues needed to be addressed. For instance, incorrect packet drops may be traced because the queue discs requeues packets whose transmission to the netdevice failed. Thus, if a netdevice drops a packet because, e.g., its queue is full, such a packet is traced as lost while it is actually

requeued by the queue disc and retransmitted as soon as the netdevice is ready to receive packets again. A workaround for this issue is to compute the number of packets that have been actually dropped as the difference between the number of dropped packets as reported by the netdevice drop trace and the number of requeued packets.

4. HELPER

A Traffic Control helper has been designed and implemented which allows to build even complex configurations (classful queue disc with multiple filters and child queue discs) and install them on a number of netdevices. Also a queue disc container has been implemented which allows to store the queue discs associated with the netdevices.

By default, the `InternetStackHelper` aggregates a `TrafficControlLayer` object to every node. The `Ipv{4,6}AddressHelper`, besides creating a `Ipv{4,6}Interface`, installs the default queue disc, `PfifoFastQueueDisc`, on every netdevice for which it creates an `Ipv{4,6}Interface`, unless a queue disc has been already installed on the netdevice. Thus, netdevices get the default queue disc installed even if they are added to the node after the internet stack has been installed on the node.

To install a queue disc other than the default one, it is necessary to install such queue disc before an IP address is assigned to the netdevice. Alternatively, the default queue disc can be removed from the netdevice after assigning an IP address, by using the convenient `Uninstall` method of the `TrafficControlHelper` class, and then installing a different queue disc on the netdevice. Clearly, it is also possible to have no queue disc installed on a netdevice.

The `TrafficControlHelper` class offers multiple methods:

- `AddInternalQueues` to add an internal `Queue` object to the `QueueDisc`;
- `AddPacketFilter` to add a `PacketFilter` to the `QueueDisc`;
- `AddQueueDiscClasses` to add a `QueueDiscClass` to the `QueueDisc`.

The `QueueDisc::CheckConfig` is called when the first packet is enqueued in the queue disc to verify that the queue disc is correctly configured.

A typical usage pattern is to create a Traffic Control helper object and to configure type and attributes of queue discs, queues, classes and filters in a top-down manner (i.e., the root queue disc is defined first). For example, the default `PfifoFastQueueDisc` can be configured as follows:

```
TrafficControlHelper tch;
uint16_t handle = tch.SetRootQueueDisc
    ("ns3::PfifoFastQueueDisc");
tch.AddInternalQueues (handle, 3,
    "ns3::DropTailQueue", "MaxPackets",
    UintegerValue (1000));
tch.AddPacketFilter (handle,
    "ns3::PfifoFastIpv4PacketFilter");
QueueDiscContainer qdiscs = tch.Install
    (devices);
```

With the above configuration, the config path of the root queue disc installed on the j-th netdevice of the i-th node is

```
/NodeList/[i]/$ns3::TrafficControlLayer/
RootQueueDiscList/[j]
```

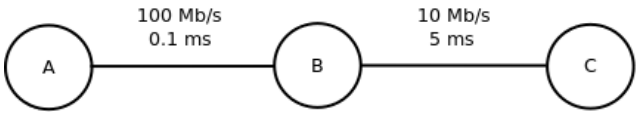


Figure 3: The network topology used for the validation tests.

5. RESULTS

5.1 Simulation Settings

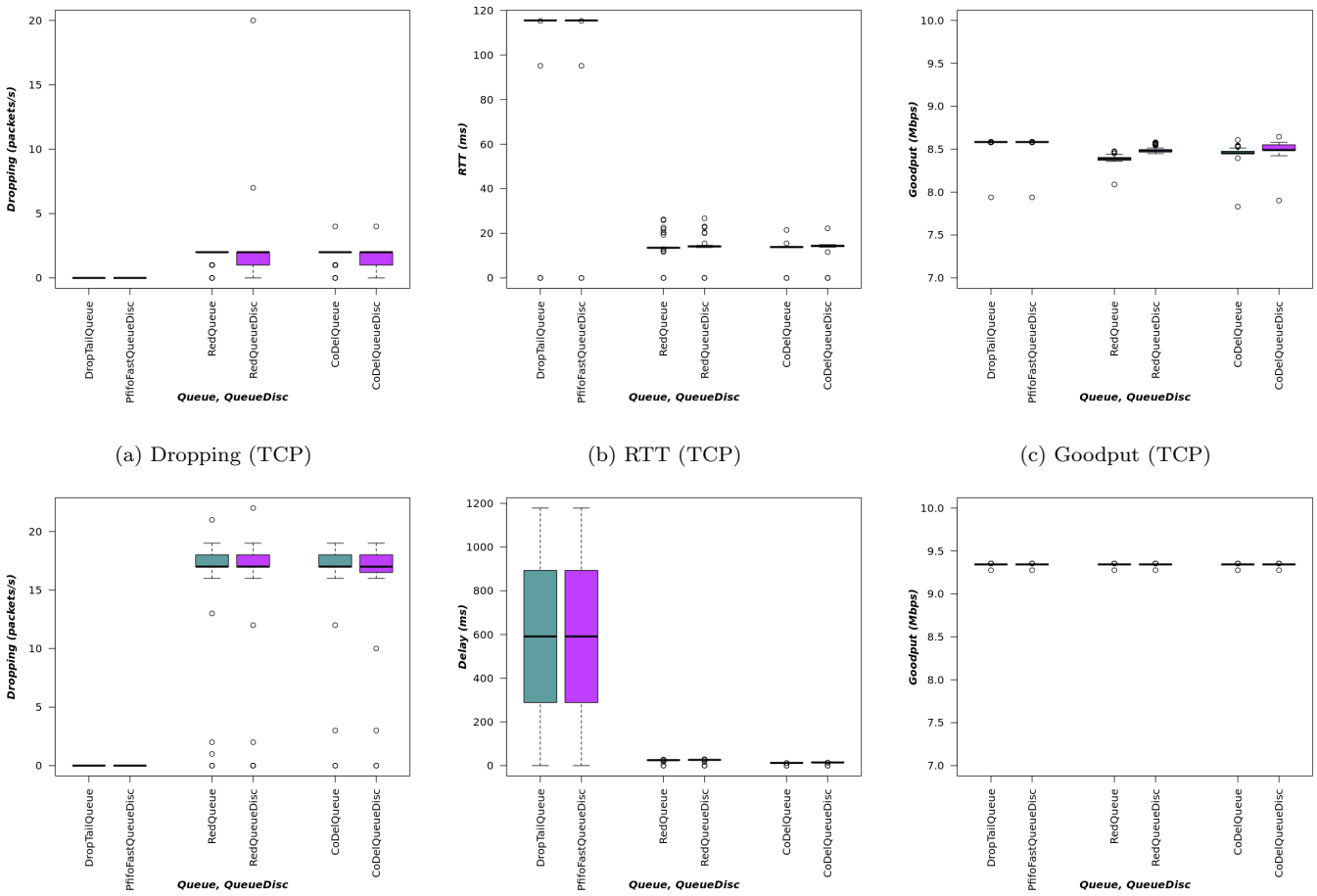
For all of the experiments described hereinafter, the simple three node topology reported in Figure 3 was used. Nodes A and B are connected by means of a point-to-point link having a data rate of 100 Mb/s and a delay of 0.1 ms. Nodes B and C are connected by means of a point-to-point bottleneck link having a data rate of 10 Mb/s and a delay of 5 ms. Two scenarios have been considered, each of which compares the current ns-3 stack with the new stack featuring the Traffic Control layer:

- the first scenario aims, to some extent, to validate the proposed Traffic Control layer by comparing the results obtained with the current and the new stacks in similar configurations. In particular, the current stack is evaluated by using netdevice queues having a size of 1000 packets, while the new stack is evaluated by using queue discs having a size of 1000 packets and netdevice queues having a size of 1 packet;
- the second scenario aims to highlight that queuing at the netdevice layer has a non negligible impact on the performance of AQM algorithms like RED and CoDel and that such an effect cannot be observed with the current ns-3 stack. The current stack is evaluated by using netdevice queues having a size of 100 packets, while the new stack is evaluated by using queue discs having a size of 1000 packets and netdevice queues having a size of 100 packets.

An OnOff traffic generator is installed on node A, while a packet sink is installed on node C. The OnOff data rate is 100 Mb/s in the TCP simulations and 10 Mb/s in the UDP simulations. The TCP version is New Reno. The packets size is 1458 bytes. The generated traffic is not marked with any QoS information. Three configurations are compared. For the current stack, we consider DropTail, RED and CoDel as the types of netdevice queues. For the new stack, we consider PfifoFast, RED and CoDel as the types of queue discs and DropTail as the type of netdevice queues. RED and CoDel are configured with the same parameter values when comparing the current and the new stack. RED is configured by setting the LinkBandwidth and LinkDelay attributes to the corresponding values of the bottleneck link, MeanPacketSize to the packet size, MinTh to 5 packets, MaxTh to 15 packets, the Gentle parameter to true. CoDel is configured by setting Interval to 100 ms and Target to 5 ms.

5.2 First Scenario

To evaluate the effects of the introduction of the Traffic Control module, the current stack (with a netdevice queue size of 1000 packets) is compared to the new stack (with



(a) Dropping (TCP) (b) RTT (TCP) (c) Goodput (TCP)

(d) Dropping (UDP) (e) Delay (UDP) (f) Goodput (UDP)

Figure 4: Plots of the first scenario.

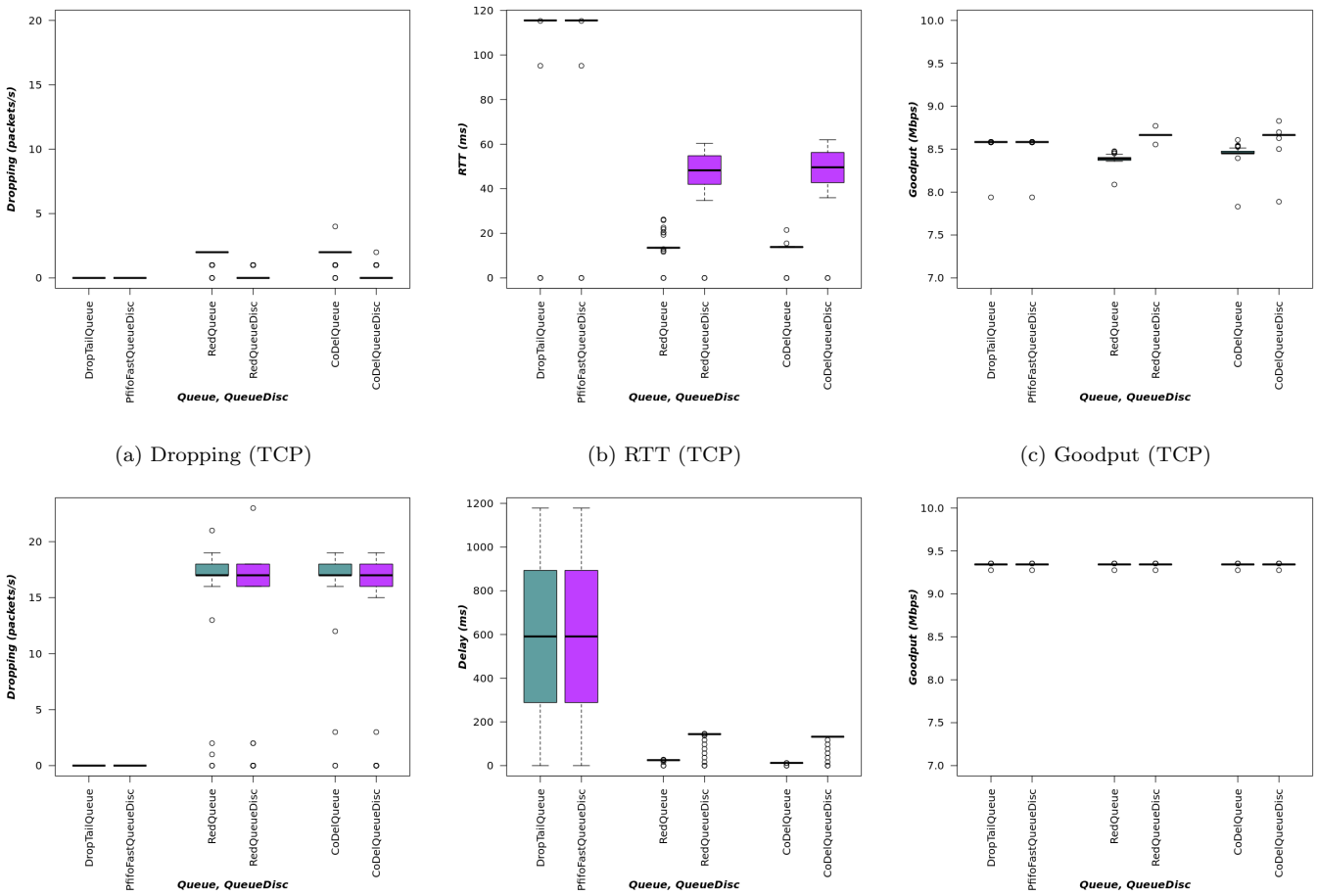
a netdevice queue size of 1 packet and a queue disc size of 1000 packets). When evaluating the new stack, the netdevice queue size is set to 1 packet in order to reduce the netdevice queueing delay, which is outside the control of the AQM algorithm, while focusing on the effectiveness of the AQM algorithm and the interactions between the Traffic Control layer and the netdevice. Note that this is equivalent to turn off the hardware offload feature and set the kernel Byte Queue Limits (BQL) to a maximum of one packet in a real system [4]. This case, occurring in a real system, could not be currently modeled in ns-3.

The results are reported in Figure 4. As noted below, when evaluating the new stack, the queue discs take advantage of the opportunity to deliver two packets to the netdevices, one immediately transmitted and another queued in netdevice queue.

In the case of TCP, the presence of the netdevice queue leads to a minor dropping activity in the queue discs in the cases of RED and CoDel (Figure 4a). We also noticed that the time elapsed between two consecutive packet droppings is higher when using the new stack and the difference between these values for the current and the new stack grows with the progress of the simulation. The dropping appears smoother in the new stack. Also, the Round Trip Time (RTT), which takes into account the netdevice queueing delay, is slightly greater when adding the Traffic Control layer, in the cases of RED and CoDel (Figure 4b). The minor dropping activity with the Traffic Control layer enables to achieve higher goodput than the current stack, in the cases of RED and CoDel (Figure 4c). When using PfifoFast/Droptail, there is no noticeable difference between the current and the new stacks.

In the case of UDP, the presence of the netdevice queue being able to accommodate one additional packet makes no difference, because, contrarily to TCP, UDP does not adapt its sending rate based on the estimated RTT. The dropping activity remains substantially unchanged in all three cases (Figure 4d). We note that no dropping activity occurs due to the netdevice queue (current stack) or the queue disc (new stack) being full, because the dropping is null in the PfifoFast/DropTail case. The delay also remains substantially unchanged in all three cases (Figure 4e). When using RED, the delay remains unchanged and is equal to about 20 ms; when using CoDel, the delay remains unchanged, too, and is equal to about 10 ms. The goodput remains constant in all three cases (Figure 4f).

The obtained results show that the new stack, in this scenario, behaves very similarly to the current one.



(a) Dropping (TCP) (b) RTT (TCP) (c) Goodput (TCP)

(d) Dropping (UDP) (e) Delay (UDP) (f) Goodput (UDP)

Figure 5: Plots of the second scenario.

5.3 Second Scenario

The second scenario aims to evaluate the effectiveness of some AQM algorithms in the common scenario in which a netdevice queue introduces a non negligible delay. The queue management algorithm is unaware of the time spent in the underlying netdevice queue, which can limit the effectiveness of the AQM algorithm. This is the case of all the netdevices for which BQL is not available and the sizing of the netdevice queue is difficult. For instance, this is the case of Wifi netdevices [4]. This case could not be evaluated in the current ns-3 stack.

The results are reported in Figure 5. In this case, the queue disc can send downwards 100 packets (queued in the netdevice queue) in addition to the packet being transmitted.

In the case of TCP, the netdevice queue limits the benefits of using an AQM algorithm. The dropping activity reflects the ability to deliver 100 packets to the netdevice. With the new stack, the dropping activity of the AQM algorithms (RED and CoDel) is reduced with respect to the current stack (Figure 5a). Consequently, the limited effectiveness of the AQM algorithms leads to a higher RTT, with respect to the current stack, in the cases of RED and CoDel (Figure 5b). In these cases, the RTT is about 50 ms. Given the reduced dropping compared to the current stack, the goodput improves and it is slightly greater than that achieved by PfifoFast/DropTail (Figure 5c). When using PfifoFast/-Droptail, there is no noticeable difference between the current and the new stacks.

In the case of UDP, the dropping activity tends to be slightly less and more smooth with the new stack, in the cases of RED and CoDel, while remains unchanged in the PfifoFast/DropTail case (Figure 5d). We note that no dropping activity occurs due to the netdevice queue (current stack) or the queue disc (new stack) being full. The delay is affected by queueing in the netdevice (Figure 5e). In this scenario, the netdevice queue introduces a non negligible delay. With the new stack, the delay is about 140 ms and 130 ms when using RED and CoDel, respectively. The goodput remains substantially unchanged in all three cases (Figure 5f)

The results obtained, in this scenario, show a behavior which cannot be observed with the current ns-3 stack. Such behavior is encountered in real systems where the netdevice queue introduces a non negligible delay that limit the effectiveness of the AQM algorithms.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the design, implementation and a preliminary validation of a Traffic Control module for ns-3. Our code has been integrated into ns-3 starting from the ns-3.25 release. We believe that our work will allow researchers to carry out more realistic simulations and to evaluate AQM schemes more precisely. One of the advantages of our Traffic Control infrastructure is that AQM schemes can now be tested on any netdevice, including, e.g., Wifi and LTE. However, changes to netdevices are still required to make them Traffic Control aware and support features such as flow control. Currently, the PointToPoint netdevice is the only Traffic Control aware netdevice. Work is in progress to make Wifi support flow control and the other features of the Traffic Control module.

Future work includes the implementation of additional queue discs such as FQ-CoDel and packet filters and of advanced features such as ECN and BQL, which is used by Linux to dynamically control the size of the netdevice transmission queues. Additionally, we plan to validate our ns-3 implementation of the Traffic Control layer against real systems, such as Linux, by exploiting the availability of powerful tools such as DCE (Direct Code Execution) or container-based platforms such as Docker.

7. ACKNOWLEDGMENTS

We wish to thank Natale Patriciello for the initial implementation of the Traffic Control layer and Tom Henderson and Tommaso Pecorella for their helpful suggestions and review of our code.

8. REFERENCES

[1] W. Almesberger, J. Salim, and A. Kuznetsov. Differentiated services on linux. In *Proceedings of the Global Telecommunications Conference (Globecom)*, volume 1B, pages 831–836. IEEE, 1999.

[2] F. Checconi, L. Rizzo, and P. Valente. Qfq: Efficient packet scheduling with tight guarantees. *IEEE/ACM Transactions on Networking*, 21(3):802–816, June 2013.

[3] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[4] T. Høiland-Jørgensen, P. Hurtig, and A. Brunstrom. The good, the bad and the wifi: Modern aqms in a residential setting. *Computer Networks*, 89:90–106, 2015.

[5] P. McKenney. Stochastic fairness queueing. In *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM)*, pages 733–740. IEEE, June 1990.

[6] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. Controlled Delay Active Queue Management. draft-ietf-aqm-codel-02, IETF, December 2015.

[7] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996.

[8] B. P. Swenson and G. F. Riley. Implementing explicit congestion notification in ns-3. In *Proceedings of the 2014 Workshop on ns-3*, WNS3 '14, pages 1–8. ACM, 2014.

Implementation and Evaluation of Proportional Integral Controller Enhanced (PIE) Algorithm in ns-3

Shravya K.S, Smriti Murali and Mohit P. Tahiliani
Wireless Information Networking Group (WiNG)
NITK Surathkal, Mangalore, India, 575025
shravya.ks0@gmail.com, m.smriti.95@gmail.com, tahiliani@nitk.ac.in

ABSTRACT

This paper proposes a new ns-3 model and presents the evaluation results for Proportional Integral controller Enhanced (PIE), a recently designed Active Queue Management (AQM) mechanism to address the problem of *bufferbloat*. The problem of bufferbloat arises due to the presence of large unmanaged buffers in routers. This leads to high queuing latency and significantly degrades the performance of time-sensitive and interactive traffic. AQM mechanisms that aim to address the problem of bufferbloat try to achieve an optimal trade-off between high link utilization and low mean queue length. PIE is a lightweight AQM mechanism that tries to achieve the same. To our knowledge, ns-3 network simulator does not have a model for simulating PIE. Hence, in this paper, we implement a ns-3 model for PIE, and show that the results obtained from it are in line with those obtained from the ns-2 model of PIE, implemented by its authors.

CCS Concepts

• **Networks** → **Network simulations;** • **Computing methodologies** → **Model development and analysis;**

Keywords

ns-3, Proportional Integral controller Enhanced, Active Queue Management, Bufferbloat

1. INTRODUCTION

Network buffers play a pivotal role in ensuring proper link utilization and smoothening of internet traffic, especially when there are intermittent bursts. Over the period of time, there has been a sharp increase in the size of network buffers due to the reduced memory cost and the need to handle large amount of bursts. The importance of managing these buffers has long been emphasized by researchers because unmanaged buffers (or passive buffers) lead to large number of packet drops and increase the queuing latency. Other known issues with passive buffers are: global synchronization [1], lock-out [2] and bufferbloat [3]. Active Queue Management (AQM) mechanisms are targeted to solve these issues by continuously monitoring and managing the router queues. From an initial goal of *avoiding congestion* to the recent most focus on *controlling queue latency*, AQM mechanisms have come a long way in the past two decades. One among the recent AQM mechanisms is Proportional Integral controller Enhanced (PIE), a lightweight mechanism to control the mean queue delay to a desired value [4].

In this paper, we make two contributions: (i) propose a new model for PIE algorithm in ns-3 network simulator [5], and discuss its design and implementation. The implementation presented in this paper is based on the PIE model in ns-2 [6], implemented by the authors of PIE (ii) validate the proposed ns-3 PIE model by comparing its results to those obtained from ns-2 PIE model. Moreover, the results presented in this paper can be easily reproduced (see appendix).

The rest of this paper is organized as follows: Section 2 provides a brief theoretical overview of the PIE algorithm. Section 3 discusses the design and implementation of ns-3 PIE model in detail. Section 4 presents the validation of ns-3 PIE model by comparing the results obtained from it to those obtained from the ns-2 PIE model. Lastly, Section 5 summarizes and concludes the paper.

2. BACKGROUND OF PIE

PIE algorithm uses the classic Proportional Integral controller [7] to control the queuing latency, and further extends it by auto-tuning the control parameters based on the level of congestion. Moreover, PIE combines the advantages of two AQM mechanisms: Random Early Detection (RED) [1] and Controlled Delay (CoDel) [8]; it is simple to deploy like RED and considers queuing delay as a measure of congestion like CoDel. Adding further, it uses the trends in latency values i.e., increase or decrease in latency, to determine the level of congestion.

PIE is designed to improve the performance of time-sensitive and interactive traffic while maintaining high link utilization and ensuring network stability. The latter is guaranteed by adapting its control parameters in small increments, thereby avoiding the large oscillations that lead to instability. PIE algorithm comprises following four components:

Random Dropping: PIE algorithm randomly drops packets depending on the value of drop probability, p which is

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915385

calculated at a regular interval. Like RED, drop probability calculation is a separate component in PIE.

Drop Probability Calculation: Drop probability is calculated at a regular interval which is denoted by *Tupdate*. The parameters involved in the calculation of drop probability are:

- *cur_del*: queuing delay during the current sample. It is estimated using Little's law [9].
- *old_del*: queuing delay during the previous sample.
- *ref_del*: desired queuing delay.
- *avg_drate*: average departure rate of the queue. It is a separate component in PIE, and its calculation is discussed later in the paper.
- α and β: scaling factors.

The difference between *cur_del* and *old_del* is used to determine whether the queuing delay is increasing or decreasing. The scaling factor α decides how much this impacts the drop probability.

Similarly, the difference between *cur_del* and *ref_del* helps to determine whether drop probability has stabilized (a stable state is when *cur_del* is equal to *ref_del*). The scaling factor β decides whether any adjustments are required to stabilize the drop probability.

Average Departure Rate Estimation: In order to account for inaccuracies that may arise due to short intermittent bursts of packets, the *avg_drate* is calculated only when the queue is in a measurement cycle i.e., when the amount of data in the queue is greater than a predetermined value, *dq_threshold*. Once a queue has more than *dq_threshold* amount of data, the samples of current departure rate (*dq_rate*) are obtained and exponentially averaged with an averaging constant (ϵ) to determine the *avg_drate*. The time interval between two samples of *dq_rate* is represented by *dq_int* and the amount of data that departed during *dq_int* is stored in *dq_count*.

Burst Allowance Calculation: This component of PIE ensures that short bursts of packets are allowed to pass successfully through the buffers. Two parameters are required to achieve this: (i) *burst_allow*; to decide whether burst should be allowed and (ii) *max_burst*; maximum time till which the burst should be allowed. If *burst_allow* is false, packets are randomly dropped with probability p; otherwise they are enqueued. Initially, *burst_allow* is set to *max_burst* and then decremented by *Tupdate* in every sample. It is reset to *max_burst* when drop probability is 0, and *cur_del* and *old_del* are less than half of *ref_del*. *max_burst* is a user configurable parameter, similar to the *interval* parameter in CoDel or Adaptive RED (ARED) [10].

3. PIE MODEL IN NS-3

This section provides the details of our implementation of PIE algorithm in ns-3. Figure 1 illustrates the interaction between the base `Queue` class in ns-3 and the derived `PieQueue` class. In our implementation, the entire PIE algorithm is contained in `PieQueue` and covers all four components of PIE described in Section 2.

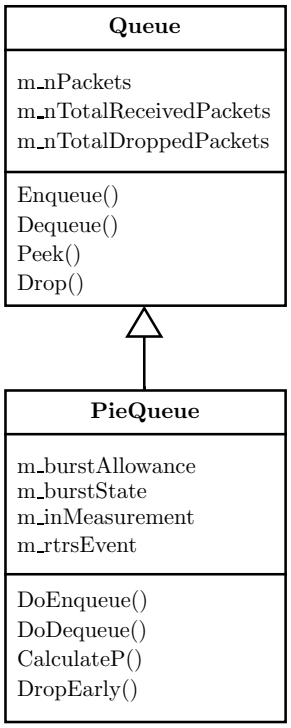


Figure 1: **Class diagram for PIE model in ns-3.**

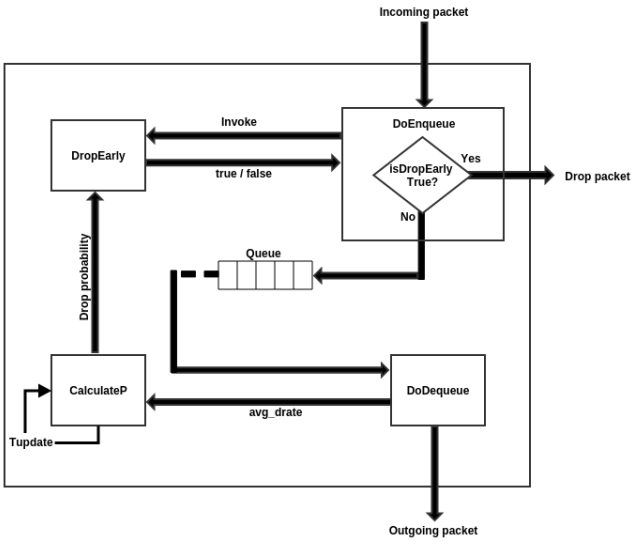


Figure 2: **Interaction between the core methods of PIE.**

The four private methods of class `PieQueue`, named `DoEnqueue`, `DoDequeue`, `DropEarly` and `CalculateP` implement the required functionality of PIE. `DoEnqueue` and `DoDequeue` are invoked on every packet arrival and departure, respectively. `DropEarly` is used to decide whether the incoming packet should be enqueued or dropped, and `CalculateP` calculates drop probability at a regular interval. Figure 2 depicts an interaction diagram between these four methods. Going forward, we provide the implementation details of every PIE component individually.

3.1 Dropping Packets Randomly

PIE drops the incoming packets randomly during the *enqueue* time and hence, this component has been implemented in `DoEnqueue` method. The decision to enqueue or drop the packet is arrived at by invoking `DropEarly` method, which returns a boolean value; false indicates enqueue and true indicates drop. Internally, `DropEarly` compares the drop probability, p with a value u, obtained from `UniformRandomVariable` class available in ns-3, and returns false if p is smaller than u; else, returns true. Other conditions when `DropEarly` method returns false include: (i) *cur_del* is less than the half of *ref_del* and (ii) the queue length is less than twice the mean packet size.

3.2 Calculation of Drop Probability

This component is implemented in `CalculateP` method and it includes: estimating *cur_del* by using Little's law, auto-tuning the scaling factors, calculating drop probability, p and updating *old_del*. All are implemented based on a set of equations provided in Section III.B of [4], respectively. Table 1 provides a list of PIE parameters required to calculate the drop probability and maps them to their corresponding variable names in ns-3. Among these parameters, *Tupdate* and *ref_del* can be configured by the user whereas rest of the parameters are internally set and updated by the PIE algorithm.

Table 1: **PIE parameters to calculate *p*.**

PIE parameter	ns-3 variable
Tupdate	m_tUpdate
cur_del	m_qDelay
old_del	m_qDelayOld
ref_del	m_qDelayRef
α	m_a
β	m_b
avg_drate	m_avqDqRate

3.3 Estimation of Average Departure Rate

This component is implemented in `DoDequeue` method. As mentioned in Section 2, the *avg_drate* is calculated only when the queue is in a measurement cycle. Table 2 provides a list of PIE parameters required to estimate average departure rate of a queue and maps them to their corresponding variable names in ns-3. All listed parameters are internally set and updated by the PIE algorithm.

Table 2: **PIE parameters to estimate *avg_drate*.**

PIE parameter	ns-3 variable
qlen	m_packets / m_bytesInQueue
dq_threshold	m_dqThreshold
dq_count	m_dqCount
start	m_dqStart
dq_int	tmp
ϵ	fixed to 0.5

3.4 Calculation of Burst Allowance

The calculation of burst allowance has a direct impact on the calculation of drop probability. Thus, the implementation of this component is coupled with the calculation of drop probability in `CalculateP` method. Table 3 provides a list of PIE parameters required to calculate burst allowance and maps them to their corresponding variable names in ns-3. Apart from these parameters, three states of the burst are also tracked; `NO_BURST` indicates that the probability based random dropping should be bypassed, `IN_BURST_PROTECTING` indicates that an active burst should be allowed to pass till `burst_allow` reaches zero and `IN_BURST` indicates that random dropping should be enabled on the passing burst.

Table 3: **PIE parameters to calculate *burst_allow*.**

PIE parameter	ns-3 variable
burst_allow	m_burstAllowance
max_burst	m_maxBurst
Tupdate	m_tUpdate
cur_del	m_qDelay
old_del	m_qDelayOld
ref_del	m_qDelayRef

3.5 Implementation Issues

Besides pro-actively dropping packets, AQM mechanisms are targeted to work with Explicit Congestion Notification (ECN) [11] and mark the packets, instead. Our implementation of PIE does not support marking of packets due to the unavailability of ECN model in ns-3. Even though there is an implementation of ECN available for ns-3 [12], it has not been included in the main distribution yet.

4. MODEL EVALUATION

To evaluate the performance of our PIE implementation, we first provide a test suite in ns-3. Further, we simulate the PIE algorithm in ns-3 and ns-2 by configuring scenarios depicted in the original paper of PIE [4]. The results obtained from both the tools are then compared to validate the implementation of our PIE model in ns-3. The performance metrics used for comparison are:

- instantaneous queue delay,
- throughput, and
- the number of packet drops.

Instructions to reproduce the results presented in this paper are provided in Appendix.

4.1 Model Verification

As a part of the PIE test suite, we run tests to ensure the compatibility of our PIE model with ns-3. Simultaneously, we also verify for the appropriate setting of attributes like Queue Limit, parameters α and β, burst allowance, etc. We consider a wide range of traffic, ranging from 8 packets to 3,000,000 packets to confirm the correct functionality of enqueue and dequeue methods. We also vary the burst allowance to check if the packet drop behavior changes accordingly.

4.2 Functional Verification

This verification is required to confirm that while minimizing the queuing delay, PIE does not affect the overall link utilization or increase the packet drop rate. We use the results obtained from the PIE model of ns-2 (implemented by the authors of PIE) and compare them with the results obtained from the PIE model of ns-3 to verify the functionality of our implementation. The simulation setup details are listed in Table 4, and are similar to the ones described in [4] except that, we use TCP Newreno "without" SACK because ns-3 does not have a model for simulating SACK. Further, our simulations consist of four different scenarios as suggested in [4]: (i) light TCP traffic, (ii) heavy TCP traffic, (iii) mix TCP and UDP traffic, and (iv) bursty UDP traffic.

Table 4: **Simulation setup.**

Parameter	Value
Topology	Dumbbell
Bottleneck RTT	100ms
Bottleneck buffer size	200KB
Bottleneck bandwidth	10Mbps
Bottleneck queue	PIE
Non-bottleneck RTT	10ms
Non-bottleneck bandwidth	10Mbps
Non-bottleneck queue	DropTail
Mean packet size	1000B
TCP	NewReno
ref_del	20ms
Tupdate	30ms
α	0.125Hz
β	1.25Hz
dq_threshold	10KB
max_burst	100ms
Application start time	0s
Application stop time	99s
Simulation stop time	100s

Scenario 1: Light TCP Traffic
This scenario consists of 5 TCP flows that pass through a bottleneck link in the dumbbell topology. We start all our TCP sources at the same time to validate the functionality of PIE in terms of controlling queue delay around a reference value and retaining proper link utilization. Other simulation parameters are same as described in Table 4.

We observe that the results obtained from ns-2 and ns-3 are similar. Fig. 3 presents the instantaneous queue delay obtained from the PIE model of ns-2 and ns-3. In both ns-2 and ns-3 results, there is a sharp increase in the queue delay initially because our TCP flows start at the same time and lead to a sudden burst of traffic. However, we observe that PIE algorithm successfully brings down the queue delay to a reference value of 20ms within 1 or 2 seconds and maintains it performance for the rest of the simulation.

The instantaneous throughput results obtained from ns-2 and ns-3 are presented in Fig. 4. We observe that link throughput drops significantly at the beginning of the simulation because PIE aggressively drops packets to bring down the queue delay from 160ms to a reference value of 20ms (see Fig. 3). For the rest of the simulation, we observe a TCP sawtooth behavior sometimes because the number of TCP senders in this scenario is less and hence, when a few TCP senders reduce their congestion window (*cwnd*), the bottleneck link utilization reduces.

Scenario 2: Heavy TCP Traffic
This scenario consists of 50 TCP flows that pass through a bottleneck link in the dumbbell topology. Like in previous scenario, we start all our TCP sources at the same time to validate the functionality of PIE in terms of controlling queue delay and retaining proper link utilization. Other simulation parameters are same as described in Table 4.

We observe that the results obtained from our PIE model in ns-3 are similar to those obtained from the PIE model in ns-2. Fig. 5 presents the instantaneous queue delay obtained from the PIE model of ns-2 and ns-3. We observe that increasing the amount of traffic has negligible impact on the performance of PIE as it successfully controls the queue delay around the reference value, except for the initial burst caused by starting all the TCP flows at the same time.

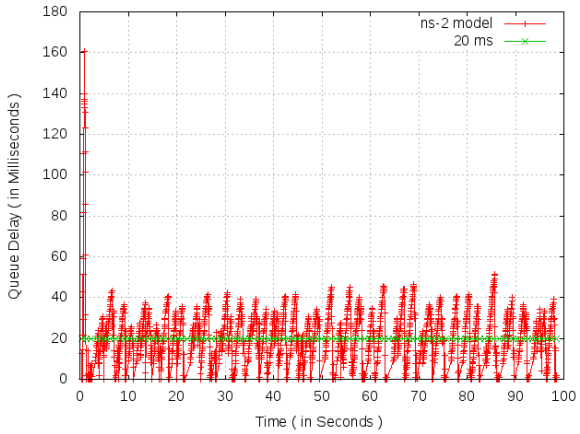
The results for instantaneous throughput obtained from ns-2 and ns-3 are presented in Fig. 6. Like in Fig. 4, the link throughput drops at the beginning of the simulation because of aggressive behavior of PIE. But, for the rest of the simulation in this scenario, the bottleneck link is fully utilized because the number of TCP senders are more and hence, when a few TCP senders reduce their *cwnd*, other TCP senders utilize the unused bottleneck link bandwidth. This also confirms that PIE algorithm does not affect the bottleneck link utilization while trying to control the queue delay.

Scenario 3: Mix TCP and UDP Traffic
This scenario consists of 5 TCP and 2 UDP flows that pass through a bottleneck link in the dumbbell topology. Like in previous scenarios, all TCP and UDP sources start at the same time. UDP sources send the data at a rate of 10 Mbps. Other simulation parameters remain same as described in Table 4.

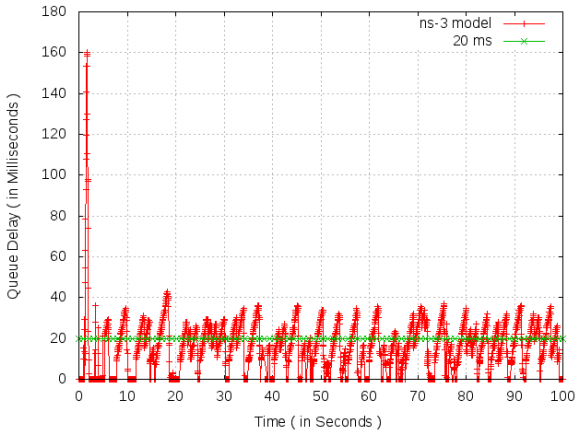
Fig. 7 shows that the performance of PIE remains unaffected in the presence of unresponsive UDP traffic, and the results obtained from our PIE model of ns-3 are inline with those obtained from the PIE model of ns-2. Similarly, Fig. 8 confirms that PIE successfully regulates the traffic to keep bottleneck link fully utilized.

Scenario 4: Bursty UDP Traffic
This scenario consists of one UDP flow that passes through a bottleneck link in the dumbbell topology and generates the traffic at a rate of 25 Mbps. The main purpose of this scenario is to functionally verify PIE's ability to tolerate bursts by varying the value of *max_burst* parameter. We consider two values of *max_burst*: (i) 0ms, (ii) 100ms while keeping the burst length fixed to 200ms. The start time of the flow is 1s and stop time is 1.2s. Mean packet size used in this scenario is 500B. Other parameters are configured as mentioned in Table 4.

Fig. 9 shows plots of the number of packet drops as a function of simulation time, and confirms that the results obtained from ns-2 and ns-3 are similar. Further, we can observe that when *max_burst* is set to 0ms, PIE starts dropping packets from the beginning of the simulation (i.e., 1.04s)

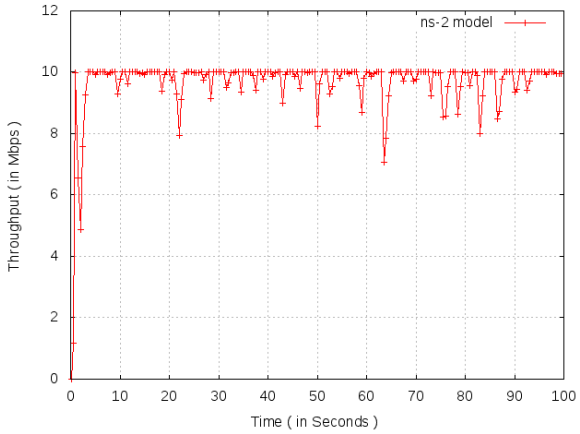


(a) ns-2

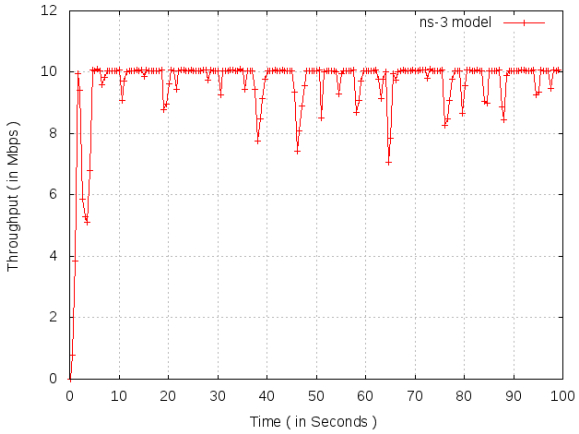


(b) ns-3

Figure 3: Queue delay with light TCP traffic.

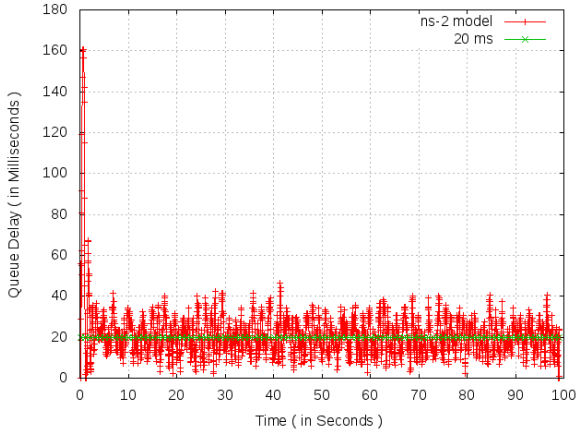


(a) ns-2

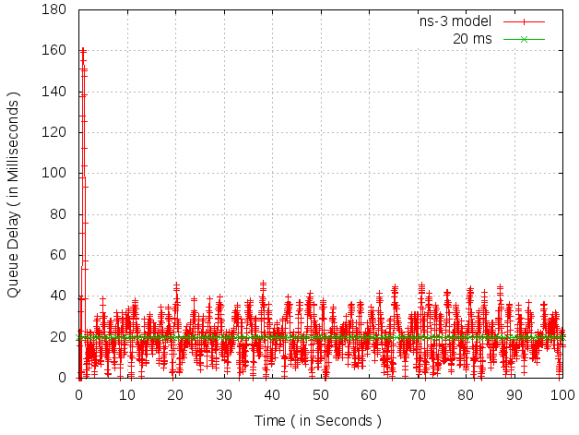


(b) ns-3

Figure 4: Link throughput with light TCP traffic.

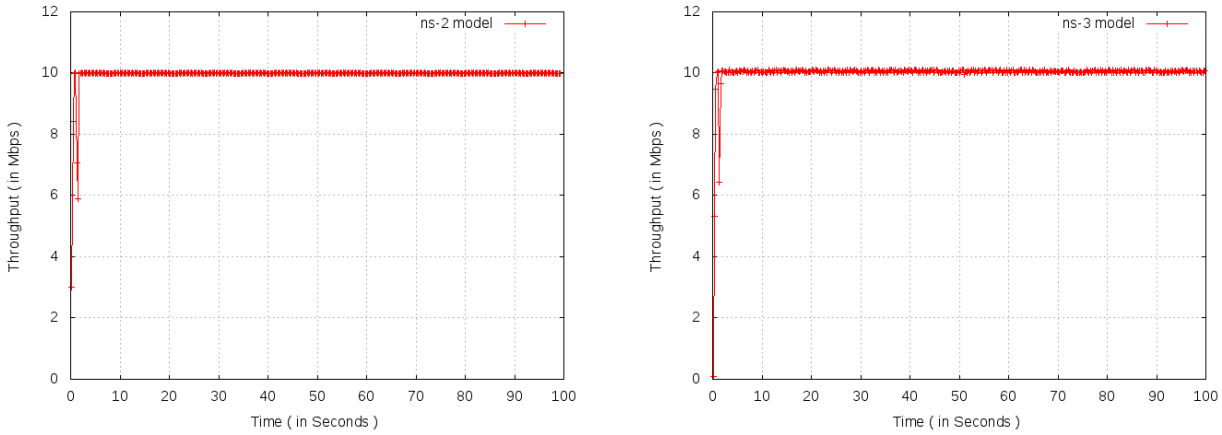


(a) ns-2



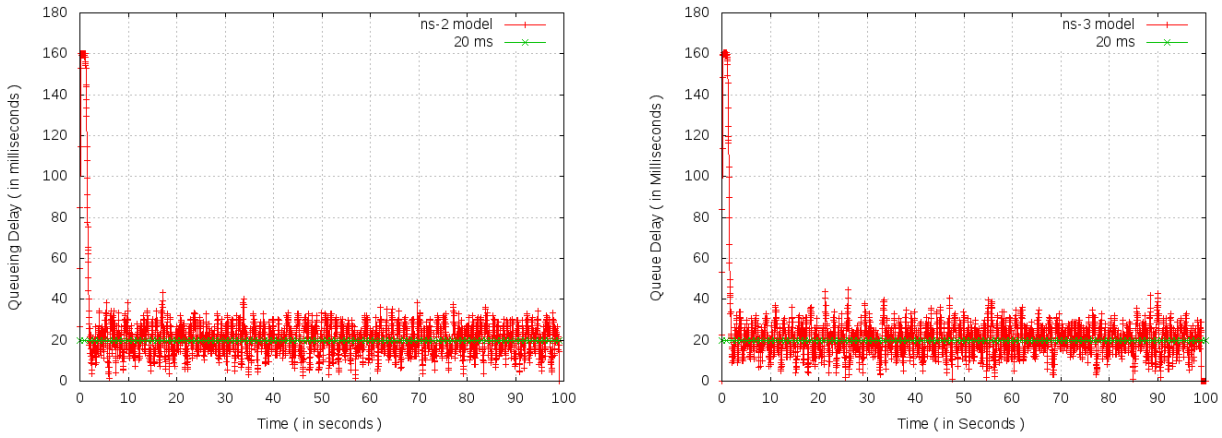
(b) ns-3

Figure 5: Queue delay with heavy TCP traffic.



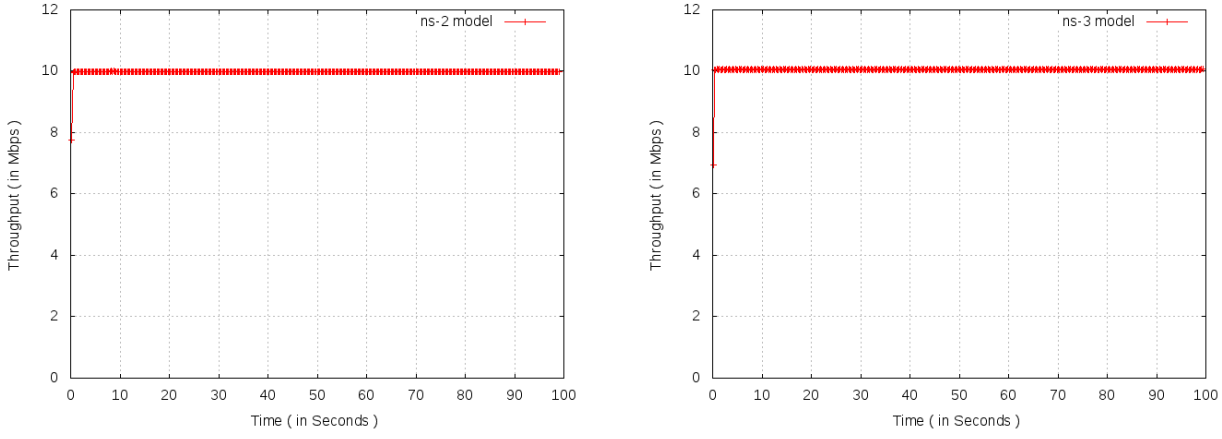
(a) ns-2 (b) ns-3

Figure 6: Link throughput with heavy TCP traffic.



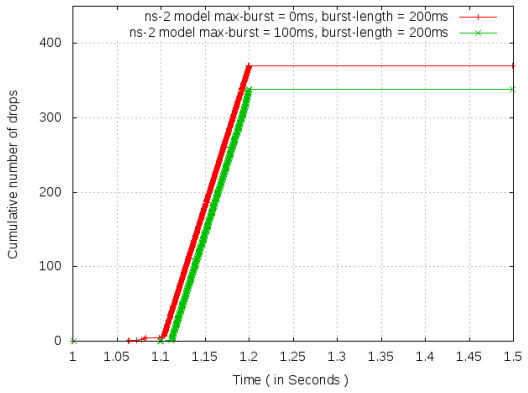
(a) ns-2 (b) ns-3

Figure 7: Queue delay with mix TCP and UDP traffic.

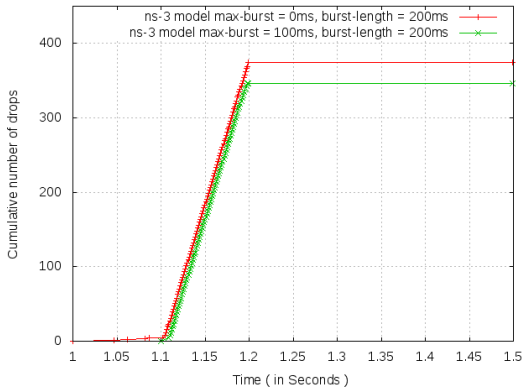


(a) ns-2 (b) ns-3

Figure 8: Link throughput with mix TCP and UDP traffic.



(a) Burst allowance : ns-2



(b) Burst allowance : ns-3

Figure 9: **PIE's burst control with short lived UDP traffic.**

whereas when *max_burst* is set to 100ms, it drops the packets after 1.1s i.e., it allows the burst for 100ms and then starts dropping the packets.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we present an implementation of PIE AQM mechanism in ns-3. We provide details for the implementation of every component of PIE algorithm. Additionally, we also provide the interactions among different methods used for implementing PIE. We validate our PIE implementation in a variety of simulation scenarios that include: light TCP traffic, heavy TCP traffic, mix TCP and UDP traffic and bursty UDP traffic. The results show that PIE effectively controls the queuing delay without affecting the bottleneck link utilization. Moreover, we compare the results obtained from our ns-3 PIE model to those obtained from ns-2 PIE model, and show that they are similar. Our PIE implementation is currently under review. We plan to further extend our PIE implementation to work with the traffic control layer which will be available in ns-3 soon, and finally merge it to the main distribution of ns-3.

6. ACKNOWLEDGMENTS

We would like to acknowledge Tommaso Pecorella for reviewing our PIE implementation in ns-3 and helping us in resolving few bugs. We highly appreciate the support of Hajime Tazaki and all reviewers, who helped us to revise the paper. Further, we would like to acknowledge Ankit Deepak and Virang Vyas for helping in the implementation of few simulation scenarios used for evaluation, and Radhesh Anand and Ayush Agarwal for correcting parts of this paper.

7. REFERENCES

[1] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1:397–413, 1993.

[2] M. Hassan and R. Jain. *High performance TCP/IP networking*, volume 29. Prentice Hall, 2003.

[3] J. Gettys and K. Nichols. Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55(1):57–65, 2012.

[4] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, pages 148–155. IEEE, 2013.

[5] Network Simulator 3. https://www.nsnam.org, 2011.

[6] Network Simulator 2. http://www.isi.edu/nsnam/ns, 1995.

[7] C. V. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1726–1734. IEEE, 2001.

[8] K. Nichols and V. Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[9] J. D. C. Little and S. C. Graves. Little's law. In *Building intuition*, pages 81–100. Springer, 2008.

[10] S. Floyd, R. Gummadi, S. Shenker, et al. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, 2001.

[11] K. Ramakrishnan, S. Floyd, D. Black, et al. The addition of explicit congestion notification (ECN) to IP, 2001.

[12] B. P. Swenson and G. F. Riley. Implementing explicit congestion notification in ns-3. In *Proceedings of the 2014 Workshop on ns-3*, page 5. ACM, 2014.

APPENDIX

In this section we provide additional details about reproducing the simulation scenarios described in this paper.

The latest version of ns-3 at the time of writing this paper is ns-3.24 and the same has been used for implementing the PIE algorithm. Some initial reviews on our code can be found here[1]. For ns-2 simulations, we have used ns-allinone-2.36.rc1 which contains a PIE model implemented by the authors of PIE. Our source code for ns-2 and ns-3 simulations can be found here[2].

Our ns-3 PIE model is implemented in `pie-queue {.h, .cc}` and the test suite can be found in `pie-queue-test-suite.cc`. Similarly, we modified the existing PIE model

[1]https://codereview.appspot.com/277610043
[2]https://github.com/mohittahiliani/reproduce-pie-paper

of ns-2 to enable tracing of queue delay parameter. The modified code is available in `pie {.h, .cc}`. Further, the simulation scripts used to obtain results in this paper are provided in the directories named ns-2 and ns-3 (see the github link mentioned above). A README file is also provided in each directory, which includes stepwise details to reproduce our simulation results.

An Implementation of Scalable, Vegas, Veno, and YeAH Congestion Control Algorithms in ns-3

Truc Anh N. Nguyen*, Siddharth Gangadhar*, Md Moshfequr Rahman*,
and James P.G. Sterbenz*†§
*Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045, USA
‡School of Computing and Communications (SCC) and InfoLab21
Lancaster University, LA1 4WA, UK
§Department of Computing
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
{annguyen, siddharth, moshfequr, jpgs}@ittc.ku.edu

ABSTRACT

Despite the modern advancements in networking, TCP congestion control is still one of the key mechanisms that ensure the stability of the Internet. Given its principal role, it is a popular research topic revisited every time TCP and its variants are studied. Open-source network simulators such as ns-3 are important tools used by the research community to gain valuable insight into existing TCP congestion control algorithms and to develop new variants. However, the current TCP infrastructure in ns-3 supports only a few congestion control algorithms. As part of the ongoing effort to extend TCP functionalities in the simulator, we implement Scalable, Vegas, Veno, and YeAH based on the original literature and their implementations in the Linux kernel; this paper presents our implementation details. The paper also discusses our validation of the added models against the theories to demonstrate their correctness. Through our evaluation, we highlight the key features of each algorithm that we study.

CCS Concepts

•**Networks** → **Transport protocols; Network simulations;**

Keywords

TCP Vegas, NewReno, Veno, YeAH, Scalable, transport protocols, ns-3 network simulator, performance evaluation, congestion control, loss-based, delay-based, hybrid, Future Internet

1. INTRODUCTION

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915386

TCP has been proven to be a crucial component of the Internet due to its ability to sustain its performance (although not optimal) with the evolution of networking during the past decades. One of TCP elements that is principal in ensuring the Internet stability and widely studied with numerous research is its congestion control algorithm. TCP congestion control is revisited on almost every attempt to study the Internet transport layer. With the modern advancements in networking, comes the emergence of new network environments such as Gigabit Ethernet or satellite links with challenging characteristics: high bit-error rate, long propagation delay, high link capacity, and asymmetric channels. Standard TCP has been enhanced for the Future Internet, resulting in many variants such as those studied in our paper (Scalable, Vegas, Veno, and YeAH). The current effort to employ load balancing at the transport layer by incorporating multi-path feature into TCP producing MPTCP [8] also requires a thorough understanding of the standard congestion control. This enables design of new algorithms that can operate efficiently in a more complex system, one with multiple coexistent, but heterogeneous subflows simultaneously transferring data through a bottleneck with high congestion probability.

The study of existing TCP algorithms and the development of any new enhancements gain substantial benefits through the use of open-source network simulators such as ns-3 [2]. However, the current ns-3 standard release only consists of NewReno (default), Westwood, Westwood+, Hybla, and HighSpeed congestion control algorithms. In order to extend the supported ns-3 TCP functionalities, we implement additional protocols, including Scalable, Vegas, Veno, and YeAH. This paper presents our implementation details of the added models. The paper also discusses our validation of these contributions against the original papers to demonstrate the correctness of the models. Through our evaluation, we highlight the key features of each algorithm.

The remainder of the paper is organized as follows: Section 2 briefly provides the theoretical background of the congestion control algorithms studied in our paper followed by a short survey on related work. Section 3 explains the implementations and how the new models interact with the rest of the TCP framework in ns-3. In Section 4, the correctness of our implementations is verified. Finally, Section 5 concludes

our paper with directions for future work.

2. BACKGROUND AND RELATED WORK

This section provides the theoretical background of different congestion control algorithms studied in our paper, including the standard NewReno, Scalable, Vegas, Veno, and YeAH, followed by a brief survey of related work.

2.1 TCP Congestion Control Algorithms

TCP congestion control algorithms can be classified into four categories: loss based, delay based, hybrid, and explicit notification [5]. Loss-based algorithms treat the occurrence of a packet loss as an indication of congestion. Delay-based algorithms infer congestion based on the increasing delay due to queue build-up when traffic load exceeds network capacity. Hybrid algorithms take advantage of both loss- and delay-based mechanisms, while explicit congestion notification (ECN) relies on explicit signals from network elements to learn about congestion. Our paper covers the first three groups, in that Scalable is a loss-based algorithm, Vegas is delay-based, and Veno and YeAH are hybrid. The loss-based NewReno (default TCP in ns-3) is used in our paper as the baseline for comparison.

Most TCP congestion control variants are derivatives of the standard defined in RFC 5681 [4], which is known as the Reno algorithm introduced by Jacobson [13] as a revision of his original Tahoe [12]. The standard specifies four intertwined algorithms that together play a principal role in the stabilization of the Internet and the prevention of congestion collapse: slow start, congestion avoidance, fast retransmit, and fast recovery. The implementation of these algorithms requires the definition and maintenance of three state variables: cwnd, rwnd, and ssthresh. Congestion window (cwnd) determines the amount of data a sender can transmit before it receives an ACK to prevent network overflow. The receiver window (rwnd) indicates the amount of data a receiver is willing to accept. The actual sending window is the minimum of cwnd and rwnd. Slow start threshold (ssthresh) provides the transition point between slow start and congestion avoidance phases.

The slow start allows TCP to gradually probe for network bandwidth when TCP starts its data transmission or after an expiration of its retransmission timer. The slow start algorithm prevents TCP from suddenly throttling the network with a large burst of traffic. In addition, slow start initiates TCP ACK clocking that determines when new data should be placed into the network to sustain equilibrium during a connection lifetime. During slow start, cwnd is incremented by 1 for every new ACK received, resulting in an exponential increase of the sending rate until a loss happens as shown in Equation 1.

$$\text{cwnd} = \text{cwnd} + 1 \tag{1}$$

The congestion avoidance algorithm continues to allow TCP to increase its sending rate (cwnd), but at a slower speed than when it is in the slow start phase to prevent congestion after the sending rate reaches ssthresh. Specifically, cwnd is incremented by 1 for every RTT, resulting in a linear increase over time until the experience of a loss. This is equivalent to the cwnd modification per Equation 2 upon a new ACK receipt.

$$\text{cwnd} = \text{cwnd} + \frac{1}{\text{cwnd}} \tag{2}$$

The fast retransmit algorithm is responsible for promptly detecting and recovering lost data by observing the number of received duplicate ACKs (dupACKs), with the arrival of three dupACKs signifying the loss of a segment. Fast retransmit was developed as an alternative to the original retransmission timer in detecting packet losses. The fast recovery governs data transmission after fast retransmit until a new ACK arrives informing the recovery of the loss. The occurrence of a loss requires Reno to halve its slow-start threshold and sending rate according to Equations 3 and 4, respectively.

$$\text{ssthresh} = \frac{\text{cwnd}}{2} \tag{3}$$

$$\text{cwnd} = \text{ssthresh} + 3 \tag{4}$$

2.1.1 NewReno

NewReno [11] modifies the Reno fast recovery algorithm explained above by introducing a mechanism for responding to *partial acknowledgments* to enhance TCP's ability by recovering more efficiently from multiple losses occurring in a single sending window. NewReno defines an additional state variable named recover to keep track of the highest sequence number transmitted before the sender enters fast retransmit, and it only leaves its fast recovery state upon the receipt of a full ACK, which is an ACK that acknowledges all sent data up to and including recover. In case a partial ACK arrives with acknowledgment number less than recover, the algorithm remains in fast recovery trying to retransmit the next in-sequence packet while sending a new segment if cwnd and rwnd allow. The NewReno algorithm is an alternate solution for multiple data loss recovery in the absence of TCP selective acknowledgment (SACK) [16].

2.1.2 Scalable

Scalable (STCP) [14] improves TCP performance for bulk transfers in high-speed wide-area networks that are characterized by long delay and high link bandwidth, by altering TCP congestion window update algorithm. The goal is to shorten TCP recovery time following a transient congestion by using a different additive increase and multiplicative decrease factors from those employed in Reno. While operating in congestion avoidance phase, STCP increments its cwnd by 0.01 for every new ACK received until a loss occurs as shown in Equation 5. On its detection of a congestion, the ssthresh value is reduced by a factor of 0.125 as in Equation 6 instead of 0.5 as in Reno (Eq. 3).

$$\text{cwnd} = \text{cwnd} + 0.01 \tag{5}$$

$$\text{ssthresh} = \text{cwnd} - \lceil 0.125 \times \text{cwnd} \rceil \tag{6}$$

2.1.3 Vegas

Vegas [6] implements a proactive congestion control algorithm that tries to prevent packet drops by keeping the backlog at the bottleneck queue small. During a connection lifetime, Vegas continuously samples the actual throughput rate and measures the RTT since these metrics reflect the

network condition when it approaches congestion. The actual sending rate is computed using Equation 7. The difference diff (Equation 9) between this throughput value and the expected throughput calculated in Equation 8 reflects the number of packets enqueued at the bottleneck, i.e. the amount of extra data sent because the Vegas sender has been transmitting at a rate higher than the available network bandwidth. Vegas tries to keep this diff value between two predefined thresholds, α and β by linearly increasing and decreasing cwnd during its congestion avoidance phase. The diff value and another predefined threshold γ are used to determine the transition between slow-start and linear increase/decrease mode.

$$\text{actual} = \frac{\text{cwnd}}{\text{RTT}} \tag{7}$$

$$\text{expected} = \frac{\text{cwnd}}{\text{BaseRTT}} \tag{8}$$

$$\text{diff} = \text{expected} - \text{actual} \tag{9}$$

In Equation 8, BaseRTT represents the minimum RTT observed during a connection lifetime.

2.1.4 Veno

TCP Veno [9] enhances Reno algorithm to deal with random loss in wireless access networks by employing the Vegas algorithm for estimating the current network condition to identify the actual cause of a loss. Specifically, Veno does not use the estimated number of packets enqueued (backlog N calculated in Equation 10) at the bottleneck to proactively detect congestion, but to distinguish between a corruption-based loss and a congestion-based loss. When Veno learns that a loss is non-congestive, instead of halving ssthresh (Eq. 3), it reduces ssthresh by a smaller amount using Equation 11. Veno also refines the Reno congestion avoidance algorithm by increasing the sending rate by 1 every 2 RTTs if the backlog exceeds its predefined threshold β, allowing it to operate longer in the stable state during which network capacity is fully utilized.

$$N = \text{actual} \times (\text{RTT} - \text{BaseRTT}) = \text{diff} \times \text{BaseRTT} \tag{10}$$

$$\text{ssthresh} = \text{cwnd} \times \frac{4}{5} \tag{11}$$

2.1.5 YeAH

Yet Another Highspeed (YeAH) [5] is a heuristic aimed to fully exploit the capacity of high bandwidth-×-delay product (BDP) networks with a small number of induced congestion events, while trying to balance among various constraints such as fair competition with standard Reno flows and robustness to random losses. YeAH-TCP operates between its *Fast* and *Slow* modes. While in the Fast mode when the network link is not yet fully utilized, YeAH increases its cwnd according to STCP rule (Eq. 5). After full link utilization is achieved, it switches to Slow mode and behaves as Reno with a precautionary decongestion algorithm. The transition between the two modes is determined based on the backlog at the bottleneck queue calculated in Equation 12 and the estimated level of network congestion shown in Equation 13. Note that Equation 12 is basically the same as Equation 10.

$$Q = \text{RTT}_{\text{queue}} \times G = (\text{RTT}_{\min} - \text{RTT}_{\text{base}}) \times \frac{\text{cwnd}}{\text{RTT}_{\min}} \tag{12}$$

$$L = \frac{\text{RTT}_{\text{queue}}}{\text{RTT}_{\text{base}}} \tag{13}$$

To fairly compete with Reno flows, YeAH ensures that it only executes its decongestion algorithm if its current cwnd is greater than the cwnd of the competing Reno flows that it estimates, denoted by $\text{count}_{\text{reno}}$ in the algorithm.

Upon the receipt of three dupACKs, YeAH adjusts its ssthresh based on the current value of Q as in Equation 14 if it is not competing with Reno flows. Otherwise, ssthresh is halved as in Reno.

$$\text{ssthresh} = \min\{\max\{\frac{\text{cwnd}}{8}, Q\}, \frac{\text{cwnd}}{2}\} \tag{14}$$

2.2 Related Work

There are three ns-3 research works that are most relevant to our paper. The first implements TCP Westwood and Westwood+ protocols [10] and compares their performance against existing variants, including Tahoe, Reno, and NewReno under some selected network conditions. The second presents an implementation of TCP CUBIC [15], which is the default congestion control algorithm in the Linux kernel. The authors validate their implementation by comparing their model with the one in Linux using Network Simulator Cradle (NSC) and the corresponding implementation in ns-2 [1]. In the most recent work, Window Scaling and Timestamp Options together with other congestion control algorithms including Hybla, Highspeed, BIC, CUBIC, and Noordwijk are introduced into ns-3 TCP infrastructure [7].

3. IMPLEMENTATIONS

In this section, we first explain the TCP congestion control classes and their main operations in the new ns-3 TCP framework. We follow this with the implementation details of STCP, Vegas, Veno, and YeAH algorithms.

3.1 TCP Congestion Control Classes in ns-3

TCP implementation in ns-3 resides in the Internet module and consists of multiple classes interacting with each other to perform the supported TCP functionalities. The current standard release contains multiple TCP variants including NewReno as the default congestion control algorithm, Hybla, Highspeed, Westwood, and Westwood+. They are pluggable components implemented as child classes of `TcpNewReno`, which is in turn derived from the congestion control abstract class `TcpCongestionOps`. The main methods currently utilized in the base classes are described in ns-3 documentation [3] and summarized below. An extended version of this paper with class diagram is available on our ResiliNets wiki [17] [1].

- `TcpCongestionOps::GetSsThresh()` and `TcpNewReno::GetSsThresh()`: These methods compute ssthresh after a loss event.
- `TcpCongestionOps::IncreaseWindow()` and `TcpNewReno::IncreaseWindow()`: These methods determine the current congestion phase by comparing cwnd and ssthresh and call the corresponding functions.
- `TcpNewReno::SlowStart()`: This method adjusts cwnd during slow-start phase.

[1] http://www.ittc.ku.edu/resilinets/reports/Nguyen-Gangadhar-Rahman-Sterbenz-2016-extended.pdf

- `TcpNewReno::CongestionAvoidance()`: This method modifies cwnd during congestion avoidance phase.
- `TcpCongestionOps::PktsAcked()`: This method manipulates timing information carried by received ACKs.

3.2 STCP

`TcpScalable` is a derived class of `TcpNewReno` that inherits `TcpNewReno::IncreaseWindow()` and `TcpNewReno::SlowStart()`. Because STCP modifies NewReno additive increase and multiplicative decrease factors used in the congestion avoidance and fast retransmit modes, respectively, `TcpScalable` replaces `TcpNewReno::CongestionAvoidance()` and `TcpNewReno::GetSsThresh()`. The implementation of these methods requires three class members to be declared: m_aiFactor that represents the increase factor with a default value of 50, m_mdFactor that represents the decrease factor with a default value of 0.125, and m_ackCnt that keeps track of the number of segments acknowledged. Following the Linux implementation of STCP, we use 50 for m_aiFactor instead of 100 suggested in the literature to account for delayed ACKs. Listing 1 shows a code snippet of `TcpScalable::CongestionAvoidance()`.

```
uint32_t w = std::min (cwndInSegments,
    m_aiFactor);
if (m_ackCnt >= w)
  {
    cwndInSegments += m_ackCnt / w; }
cwnd = cwndInSegments * m_segmentSize;
```

Listing 1: TcpScalable::CongestionAvoidance().

`TcpScalable::GetSsThresh()` updates ssthresh after the receipt of three dupACKs following STCP rule as shown in Listing 2.

```
uint32_t ssThresh = std::max (2.0,
    cwndInSegments * (1.0 - m_mdFactor)) *
    m_segmentSize;
```

Listing 2: TcpScalable::GetSsThresh().

3.3 Vegas

The key components of our Vegas implementation in `TcpVegas` are its RTT sampling performed in `TcpVegas::PktsAcked()` method upon the receipt of an ACK, its calculation of the diff rate based on the RTT measurements in `PktsAcked()`, and the implementation of its linear increase/decrease mode. The latter is performed inside `TcpVegas::IncreaseWindow()` method, in which a code snippet is presented in Listing 3.

```
if (diff > beta)
  {
    /* We are going too fast, we need to
    slow down by linearly decreasing cwnd
    for the next RTT */
  cwnd = cwnd - m_segmentSize;  }
else if (diff < alpha)
  {
    /* We are going too slow, we need to
    speed up by linearly increasing cwnd
    for the next RTT */
    cwnd = cwnd + m_segmentSize; }
else
  {
    /* We are going at the right speed,
     * cwnd should not be changed */ }
```

Listing 3: TcpVegas::IncreaseWindow().

3.4 Veno

Similar to Vegas, `TcpVeno` requires an implementation of the `PktsAcked()` method to perform RTT sampling needed for the calculation of backlog N at the bottleneck queue. `TcpVeno::IncreaseWindow()` modifies cwnd following Veno additive increase rule as shown in Listing 4, where m_inc is a boolean variable that is only set to True every other RTT.

```
if (N < beta)
  {
    /* Available bandwidth is not fully
    utilized, we increase cwnd by 1 every
    RTT as in NewReno */
     TcpNewReno::CongestionAvoidance (tcb,
    segmentsAcked); }
else
  {
    /* Available bandwidth is fully
    utilized, we increase cwnd by 1 every
    other RTT */
     if (m_inc)
       {
         TcpNewReno::CongestionAvoidance (
    tcb, segmentsAcked); }
      else
       {
         m_inc = true; } }
```

Listing 4: TcpVeno::IncreaseWindow().

`TcpVeno::GetSsThresh()` implements the Veno multiplicative decrease algorithm as shown in Listing 5.

```
if (N < beta)
  {
    /* Random loss is most likely to have
    occurred, we reduce cwnd by only 1/5 */
    return std::max (cwnd * 4 / 5, 2 *
    m_segmentSize); }
else
  {
    /* Congestion-based loss is most likely
     to have occurred, we reduce cwnd by
    1/2 as in NewReno */
    return std::max (cwnd / 2, 2 *
    m_segmentSize); }
```

Listing 5: TcpVeno::GetSsThresh().

3.5 YeAH-TCP

`TcpYeah` also implements the `PktsAcked()` method to measure the RTT values required for its calculation of Q, which is used by `TcpYeah::IncreaseWindow()` to determine YeAH's operation mode (Fast or Slow) during its congestion avoidance phase. A code snippet of `TcpYeah::IncreaseWindow()` is presented in Listing 6. Following the Linux kernel implementation of YeAH, we use 80 and 8 as the default values of the two thresholds maxQ and phy, respectively.

```
if (Q < maxQ & L < (1 / phy))
  {
```

```
    // We are in Fast mode; cwnd is
    incremented based on STCP rule
     TcpScalable::CongestionAvoidance (tcb,
    segmentsAcked); }
else
  {
    /* We are in Slow mode, determine if we
     need to execute the precautionary
    decongestion algorithm */
     if (Q > maxQ && cwndInSegments >
    renoCount)
       {// Precautionary decongestion
         cwndInSegments -= Q;
         cwnd = cwndInSegments *
    m_segmentSize; } }
```

Listing 6: TcpYeah::IncreaseWindow().

In `TcpYeah::GetSsThresh()`, the reduction of `cwnd` depends on whether YeAH competes with Reno flows as shown in Listing 7, where the threshold `rho` is the minimum number of RTTs required to consider the presence of Reno flows.

```
if (doingRenoNow < rho)
  {// YeAH does not compete with Reno flows
    return std::min (std::max (cwnd / 8,Q),
     cwnd / 2); }
else
  {// YeAH competes with Reno flows
    return std::max (cwnd / 2, 2 *
    m_segmentSize); }
```

Listing 7: Yeah::GetSsThresh().

4. VERIFICATION AND VALIDATION

In addition to writing various unit tests that are mandatory in ns-3 to ensure the correctness of our new models before they can be merged into the standard release, we also try to simulate them under various network conditions and validate their performance against the corresponding literature. Given that the testing scenarios in the original papers are varied, the exact parameters used are not explicitly described in some sources, and to be consistent in our paper, we use the dumbbell topology illustrated in Figure 1 to fulfill our validation purpose. The goal of this section is to demonstrate that despite the simulation topology that we use, our models still exhibit the key characteristics of the protocols. Our simulations are conducted with ns-3.24-dev.

4.1 Simulation Topology

At each edge of the dumbbell topology in Figure 1 are two nodes serving as the sources on one end and the sinks at the other end. The endpoints communicate through a single bottleneck link that connects two network routers. All traffic across the network are generated using `BulkSendApplication` with an MTU size of 1500 bytes. Drop Tail queues are used at the bottleneck link with size set to the bandwidth-×-delay product. Each of the access link that connects an endpoint with one of the two routers has a bandwidth of 10 Mb/s with a negligible delay of 0.1 ms. The bandwidth and delay of the bottleneck link are varied depending on the simulated scenarios. The one-way delay value of this link ranges from 50 ms to 300 ms to cover the different delays for various network environments. Given that the protocols studied in this paper focus on improving the standard NewReno linear increase and multiplicative decrease phases, we set both the initial congestion window and slow start threshold to 15 packets to eliminate the slow start phase. Timestamps and window scaling options are both enabled. The duration of each simulation is 200 to 400 seconds. We use NewReno as the baseline for all of our comparisons. Simulation parameters are summarized in Table 1.

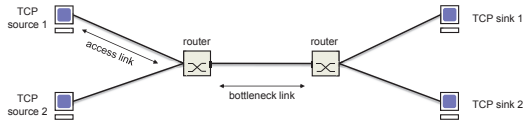


Figure 1: Simulation topology.

Table 1: Simulation parameters.

Parameter	Values
Access link bandwidth	10 Mb/s
Bottleneck link bandwidth	varied
Access link delay	0.1 ms
Bottleneck link delay	varied
Packet MTU size	1500 B
Delayed ACK count	2 segments
Application type	Bulk send application
Queue type	Drop tail
Queue size	BDP
Simulation time	200 s – 400 s

4.2 Robustness to Random Loss

To study the impact of random packet losses on the congestion control algorithms, we set the bottleneck bandwidth to 10 Mb/s and delay to 100 ms. Using `ns3::RateErrorModel`, we introduce a packet error rate (PER) of 10^{-3} into the unreliable bottleneck link. For a clearer presentation of the plots, each simulation is run for 200 seconds. We show the congestion window dynamics of a single connection with one sender and receiver on each network edge.

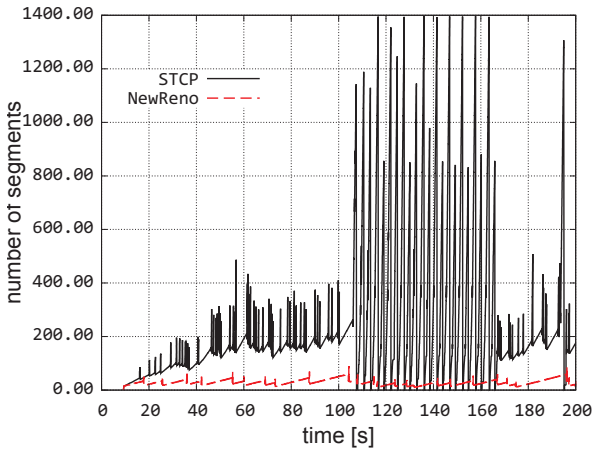


Figure 2: cwnd dynamics of STCP and NewReno.

Figure 2 shows the `cwnd` of STCP in comparison with NewReno's as they evolve over time. Overall, STCP's `cwnd` values are much higher than NewReno's, resulting in about

7 times more average throughput than the standard algorithm. As seen in the plot, NewReno takes $C/2$ seconds to recover where C is the cwnd that NewReno reaches following a loss, while the packet loss recovery time for STCP is independent of the connection's window size. In addition, NewReno halves its cwnd upon the receipt of three dupACKs. STCP, on the other hand, only reduces its window by $b \times C$ with b equal to 0.125. However, similar to other loss-based congestion control algorithms that were designed for high BDP network environments, the high throughput achieved by STCP comes at a price of higher chance of experiencing multiple retransmission timer timeouts (RTO) due to its aggressive increasing rule, as shown in the plot at time between 110 seconds and 160 seconds.

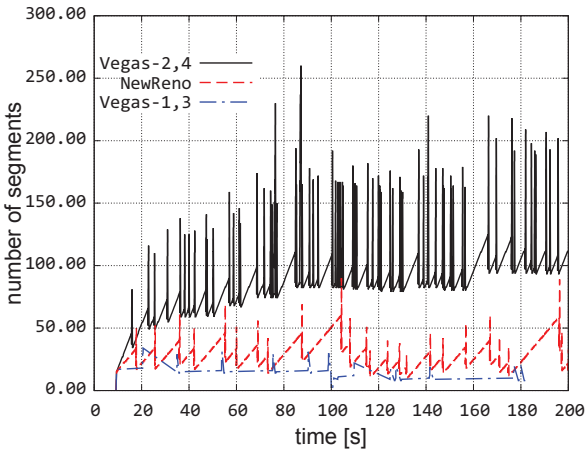


Figure 3: cwnd dynamics of Vegas and NewReno.

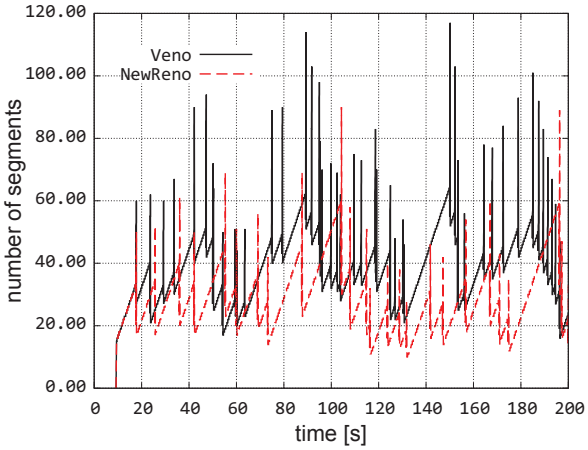


Figure 4: cwnd dynamics of Veno and NewReno.

Figure 3 shows the cwnd evolution of two versions of Vegas against NewReno. Following the notation used in the original Vegas paper, Vegas-1,3 sets α to 1 and β to 3, while Vegas-2,4 uses 2 and 4 for α and β, respectively. Overall, with our setup for this simulation, Vegas-2,4 outperforms Vegas-1,3, which even has a lower sending rate than the standard NewReno. Vegas-2,4 is able to achieve a throughput of 4.6 Mb/s, which is 3 times higher than the 1.5 Mb/s throughput achieved by NewReno. On the other hand, Vegas-1,3 is unable to utilize the available network capacity, resulting in a throughput of only 0.7 Mb/s. When we use 1 and 3 for α and β as the default values, most of the time, the calculated diff rate falls in between the two thresholds, causing Vegas to remain sending at the same rate without modifying its cwnd. For Vegas-2,4, the cwnd is increased by 1 segment size every RTT when the diff value is less than α, resulting in a linear increase until three dupACKs are received due to the random packet loss we introduce into the channel, which causes Vegas to reduce its cwnd by the maximum of the current ssthresh and 1/4 of congestion window. The linear increase/decrease mode then governs the sending rate after the reduction of cwnd. In our other simulations, we use Vegas-2,4 for a better throughput of Vegas, and this is also the default Vegas version in the Linux kernel.

Figure 4 shows the cwnd dynamics of Veno and NewReno. In this case, when the protocols are unable to fully utilize the available network bandwidth due to packet corruptions, the Veno increase rule is the same as NewReno. The only difference is its decreasing algorithm when a loss is detected. Since Veno is able to distinguish between congestive and non-congestive losses, for most of the time, Veno only reduces its cwnd by 1/5, resulting in a better throughput than NewReno.

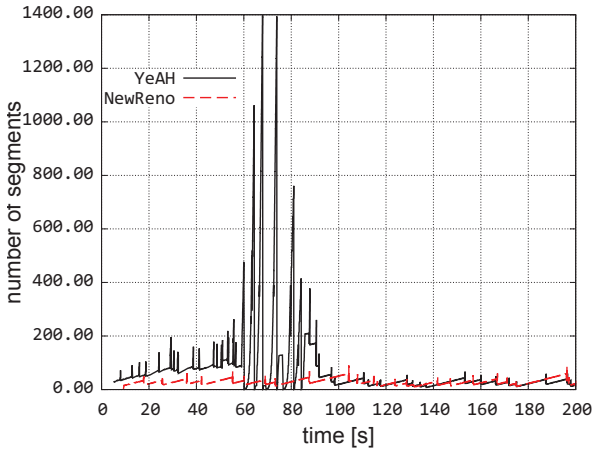
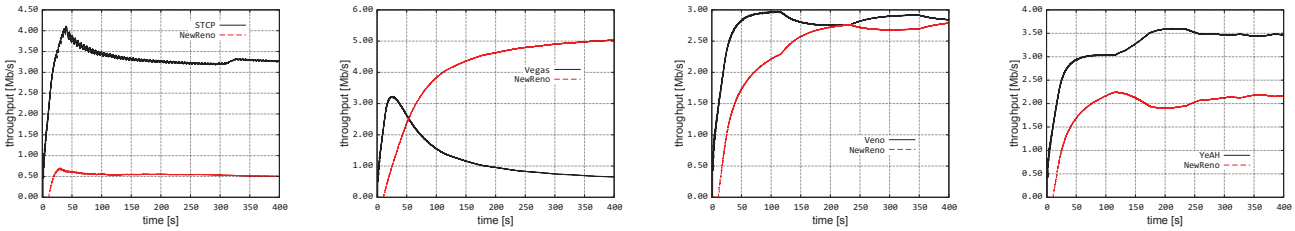


Figure 5: cwnd dynamics of YeAH and NewReno.

Figure 5 shows the cwnd dynamics of YeAH and NewReno. The congestion avoidance phase of YeAH is the switching between its Fast and Slow mode. With our simulation parameters and the default value of maxQ set to 80 segments, YeAH does not execute its precautionary decongestion algorithm because Q is less than maxQ although it detects that it does not compete with "greedy" Reno flows. Thus, the cwnd is only updated when YeAH enters its Fast mode. The increment rule during Fast mode follows STCP, but at a slower speed than the result for STCP presented in Figure 2 because we set m_aiFactor to 100 for YeAH implementation. Upon the detection of a loss through the receipt of three dupACKs before the occurrence of several RTOs, YeAH reduces its window by 1/8. The RTOs trigger a false alarm of the presence of Reno flows, which causes YeAH to halve its window afterward. All of these factors result in a low throughput of YeAH when comparing with STCP.

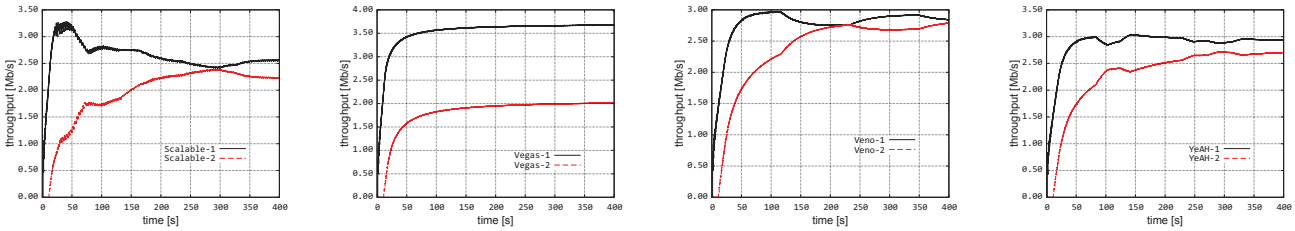
4.3 Friendliness to NewReno

For a new TCP congestion control algorithm to be widely deployed in practice, it must be friendly with the standard Reno traffic. When it shares the network capacity with Reno



(a) STCP (b) Vegas (c) Veno (d) YeAH

Figure 6: Instantaneous throughput with NewReno traffic.



(a) STCP (b) Vegas (c) Veno (d) YeAH

Figure 7: Instantaneous throughput with second flow using same algorithm.

flows, it should avoid starving the competing flows while being able to exploit the link bandwidth. So, in our second scenario, we study the friendliness of STCP, Vegas, Veno, and YeAH. We use the same dumbbell topology, but with two senders and two receivers at each network edge. Each 400-second simulation generates two traffic flows; one of them is NewReno while the other is one of the protocols studied in the paper. The bottleneck link has a bandwidth of 6 Mb/s and a delay of 100 ms with no random losses. The NewReno flow starts 10 seconds later than the other one.

Figure 6 shows the instantaneous throughput of each variant against NewReno's. Overall, Veno is the most TCP-friendly among all algorithms. The aggressive STCP puts the NewReno flow in starvation with the ratio of STCP throughput to NewReno throughput being 3.26:0.5 Mb/s. Basically, STCP is a NewReno derivative with higher increase, but smaller decrease factor than the standard algorithm's. Vegas exhibits its well-known behavior of a pure delay based algorithm as being the least aggressive among all protocols studied in our paper. While NewReno continues to increase its cwnd until a packet loss occurs, Vegas executes its proactive window adjustment that tries to send data at a moderate rate to prevent any packet drops at the bottleneck queue. As soon as the Reno flow enters the network, Vegas throughput starts to reduce. NewReno quickly obtains the same throughput as Vegas about 40 seconds after it starts and continues to steal the network bandwidth away from Vegas. Unlike Vegas, Veno does not use the extra number of packets at the bottleneck queue to control its sending rate. When the available bandwidth is not fully utilized and no random loss presents, Veno behaves exactly as NewReno during its additive increase and multiplicative decrease phase, resulting in a fair share of network capacity at 200 seconds. Because YeAH does not execute its precautionary decongestion control algorithm due to the presence of NewReno flow, it behaves like STCP during congestion avoidance. The ratio of YeAH to NewReno throughput is 3.47:2.17, which is smaller than the ratio of STCP to NewReno due to the higher m_aiFactor used in YeAH implementation as explained previously.

4.4 Intra Fairness

In addition to TCP friendliness, a congestion control algorithm is required to be internally fair: it should be friendly to itself. We study intra fairness of the protocols by simulating the same scenario as in Section 4.3, except that the second flow uses the same TCP variant as the first flow.

Figure 7 plots the instantaneous throughput of each variant in the competition with a second flow. While STCP, Veno, and YeAH try to converge to a fair share of the network resource after some time, Vegas maintains a constant gap between the throughput values of its two flows throughout the whole simulation period. This is because both Vegas flows have the tendency of attempting to prevent any queue drops. The first flow has the advantage of entering the network 10 seconds before the other, so it can obtain more bandwidth. By the time the second flow starts, it just attempts to utilize the remaining capacity.

4.5 Impact of Channel Delay

In this scenario, we study the impact of link delay on the performance of our congestion control algorithms. Each simulation generates a single flow of traffic through the bottleneck link that has a bandwidth of 6 Mb/s and delay varying from 50 ms to 300 ms. No random losses are introduced into the link.

Figure 8 plots the average throughput achieved by each algorithm when the bottleneck delay is varied. Overall, all variants are affected by high link delay, resulting in a decreasing of throughput with increasing RTT. STCP exhibits the most interesting behavior as it initially performs worse

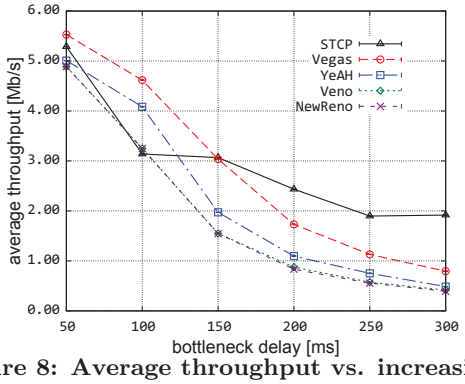


Figure 8: Average throughput vs. increasing delay.

than Vegas and YeAH, but starts to improve the throughput at the 150-second delay. Considering Vegas and its variants (Veno and YeAH), Vegas performs the best while Veno performs the worst. As explained above, with no random loss and available bandwidth not fully utilized (the high delays prevent all protocols from efficiently exploiting network capacity), Veno behaves exactly like NewReno. The non-aggressive behavior of the pure delay-based Vegas is an advantage in this case as its sending rate is more stable, resulting in fewer RTO occurrences.

5. CONCLUSIONS

We have presented our implementations of STCP, Vegas, Veno, and YeAH congestion control algorithms in ns-3 and studies of their behavior under various network conditions using a variety of metrics (robustness to random loss, TCP friendliness, intra-protocol fairness, and the impact of link delay) while verifying the correctness of the models. The results show that STCP is the most robust to random bit errors, and Vegas outperforms Veno and YeAH in the presence of non-congestive packet drops. While STCP and YeAH are the most aggressive algorithms, Vegas is the least when they have to share the bottleneck link capacity with a standard NewReno flow. The proactive congestion control mechanism that Vegas employs also prevents it from achieving intra fairness, although the same mechanism helps Vegas to better sustain its throughput than other protocols in a high propagation delay network.

For future work, we plan to study these variants under additional network scenarios to have a more complete picture of the algorithms' characteristics. We are also interested in experimenting with different values for the thresholds used in our implementations as we have seen from Section 4 that the different default values for α and β affect Vegas performance. In addition, we plan to continue to contribute to the ns-3 community with more TCP models, inclduing CTCP.

6. ACKNOWLEDGMENTS

We would like to acknowledge the members of the ResiliNets research group for their useful discussions and suggestions that helped us with this implementation. We would like to thank the anonymous reviewers for their helpful feedback on this paper. Finally, we would also like to thank Tom Henderson, Natale Patriciello, and the ns-3 development team for their timely responsiveness to guidance and issues with the ns-3 platform. This work was funded in part by NSF grant CNS-1219028 (Resilient Network Design for Massive Failures and Attacks).

7. REFERENCES

[1] The Network Simulator: ns-2. http://www.isi.edu/nsnam/ns, December 2007.

[2] The ns-3 Network Simulator. http://www.nsnam.org, July 2009.

[3] The ns-3 Network Simulator Doxygen Documentation. http://www.nsnam.org/doxygen, July 2012.

[4] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), Sept. 2009.

[5] A. Baiocchi, A. P. Castellani, and F. Vacirca. YeAH-TCP: yet another highspeed TCP. In *Proc. PFLDnet*, volume 7, pages 37–42, 2007.

[6] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, 1994.

[7] M. Casoni, C. A. Grazia, M. Klapez, and N. Patriciello. Implementation and Validation of TCP Options and Congestion Control Algorithms for ns-3. In *Proceedings of the 2015 Workshop on ns-3*, WNS3 '15, pages 112–119, New York, NY, USA, 2015. ACM.

[8] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), Jan. 2013.

[9] C. P. Fu and S. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 21(2):216–228, 2003.

[10] S. Gangadhar, T. A. N. Nguyen, G. Umapathi, and J. P. Sterbenz. TCP Westwood Protocol Implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, Cannes, France, March 2013.

[11] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Standards Track), 2012.

[12] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, 1988.

[13] V. Jacobson. Modified TCP congestion avoidance algorithm, April 1990.

[14] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(2):83, Apr. 2003.

[15] B. Levasseur, M. Claypool, and R. Kinicki. A TCP CUBIC Implementation in ns-3. In *Proceedings of the 2014 Workshop on ns-3*, WNS3 '14, pages 3:1–3:8, New York, NY, USA, 2014. ACM.

[16] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), Oct. 1996.

[17] T. A. N. Nguyen, S. Gangadhar, M. M. Rahman, and J. P. Sterbenz. An Implementation of Scalable, Vegas, Veno, and YeAH Congestion Control Algorithms in ns-3 (extended). ITTC Technical Report ITTC-FY2016-TR-69921-04, The University of Kansas, Lawrence, KS, April 2016.

TCP Evaluation Suite for ns-3

Dharmendra Kumar Mishra, Pranav Vankar and Mohit P. Tahiliani
Wireless Information Networking Group (WiNG)
NITK Surathkal, Mangalore, India, 575025
dharmendra.nitk@gmail.com, pranavvankar442@gmail.com, tahiliani@nitk.ac.in

ABSTRACT

Congestion Control (CC) algorithms are essential to quickly restore the network performance back to stable whenever congestion occurs. A majority of the existing CC algorithms are implemented at the transport layer, mostly coupled with TCP. Over the past three decades, CC algorithms have incrementally evolved, resulting in many extensions of TCP. A thorough evaluation of a new TCP extension is a huge task. Hence, the Internet Congestion Control Research Group (ICCRG) has proposed a common TCP evaluation suite that helps researchers to gain an initial insight into the working of their proposed TCP extension.

This paper presents an implementation of the TCP evaluation suite in ns-3, that automates the simulation setup, topology creation, traffic generation, execution, and results collection. We also describe the internals of our implementation and demonstrate its usage for evaluating the performance of five TCP extensions available in ns-3, by automatically setting up the following simulation scenarios: (i) single and multiple bottleneck topologies, (ii) varying bottleneck bandwidth, (iii) varying bottleneck RTT and (iv) varying the number of long flows.

CCS Concepts

•**Networks** → **Transport protocols; Network simulations;** •**Computing methodologies** → *Simulation environments;*

Keywords

ns-3, Congestion Control, TCP Evaluation

1. INTRODUCTION

Congestion Control (CC) algorithms implemented in TCP play a vital role in ensuring proper functioning of the Internet. Over the period of time, as CC algorithms continue to evolve, a lot of new TCP extensions are frequently proposed. Evaluating the performance of new TCP extensions is not trivial because there is a lack of agreed set of performance metrics, because of which, each study highlights only a particular aspect of TCP while leaving some of the most important ones. Due to a high volume of research being carried out in this area, there is a need for systematically screening every new TCP extension and identifying the suitable ones for detailed evaluation.

To address this problem, the Internet Congestion Control Research Group (ICCRG) [1] provides information regarding a common test suite for evaluating new TCP extensions. This suite is not framed to *exhaustively* evaluate a new TCP extension; instead, it focuses to help the researchers to *easily* and *quickly* derive initial results of their work.

In this paper, we present an implementation of TCP evaluation suite in ns-3 [2]. We also highlight some additions that were required in existing models of ns-3 to successfully implement the suite. Our implementation can be used to automate the work cycle from setting up the simulation environment to collecting results. Moreover, our implementation provides support for automatically configuring many Internet-like scenarios such as scenarios with single and multiple bottleneck links, scenarios with different traffic mix and scenarios with varying bottleneck attributes. Nevertheless, our implementation is only a part of the evaluation suite designed by ICCRG [1]. Implementing the entire suite requires significant changes to existing models in ns-3 and is beyond the scope of this paper.

The rest of this paper is organized as follows: Section 2 provides a review of similar benchmark proposals for TCP evaluation and their implementation details. Section 3 discusses the design choices and our proposed architecture of TCP evaluation suite in ns-3. Section 4 demonstrates the usage of our suite to compare existing TCP extensions in ns-3 by automatically configuring different simulation scenarios. Lastly, Section 5 summarizes and concludes the paper.

2. RELATED WORK

The design of TCP evaluation suite dates back from a paper in 2007 to an internet draft in 2014, as shown in Table 1. All proposals have been implemented using ns-2 [7]. Although both, proposal 1 and 3, are targeted towards evaluating High-Speed TCPs, each adopts a different approach for implementing it. Proposal 4 is an enhancement of proposal 2, both being the internet drafts. Moreover, they adopt a similar approach for implementation in ns-2; in fact, proposal 4 extends the implementation of proposal 2.

The design and implementation of TCP evaluation suite presented in this paper is partially adopted from the approach followed by proposals 2 and 4. We found that the implementation of TCP evaluation suite in ns-3 is relatively simpler, thanks to the topology helper classes provided.

3. DESIGN AND IMPLEMENTATION

TCP evaluation suite is implemented as a separate model called `tcp-eval`, under the `src` directory in ns-3. This sec-

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915388

Table 1: Different implementations of TCP Evaluation Suite.

No.	Details of proposals	Tools used
1	Shimonishi, Hideyuki, M. Y. Sanadidi, and Tutomu Murase. "Assessing Interactions among Legacy and High-Speed TCPs." PFLDnet 2007 (2007).	ns-2. Download link [3]
2	Internet draft: "An NS-2 TCP Evaluation Tool", Gang wang, Yong Xia, David Harrison (April 2007)	ns-2. Code was released in two versions. Download link of second version [4]
3	Li, Yee-Ting, Douglas Leith, and Robert N. Shorten. "Experimental evaluation of TCP protocols for high-speed networks." Networking, IEEE/ACM Transactions on 15.5 (2007): 1109-1122.	ns-2. Download link [5]
4	Internet draft: "Common TCP Evaluation Suite", D. Hayes, D. Ros, L. Andrew, S. Floyd (July 2014)	ns-2. Download link [6]

tion describes the core design decisions made for the development of the TCP evaluation suite model, along with several additions that were required in the existing models of ns-3. Table 2 highlights different elements that are supported in our implementation of `tcp-eval`. Figure 1 depicts the interactions among different classes implemented in `tcp-eval`, and with other existing classes of ns-3.

Table 2: Different elements supported by `tcp-eval`.

Topologies	Types of traffic	Performance metrics
Dumbbell (single bottleneck)	Long-lived FTP (TCP)	Aggregate link utilization
Parking lot (multiple bottleneck)	Streaming video (UDP)	Mean queue length
	Interactive voice (UDP)	Packet drop rate

The `tcp-eval` model comprises three primary classes:

3.1 ConfigureTopology

This class is used for configuring the simulation parameters such as setting bottleneck bandwidth and bottleneck delay, setting the parameters for Random Early Detection (RED) algorithm [8], etc. It acts as a base class to configure parameters for dumbbell and parking lot topologies.

3.2 CreateTraffic

This class generates different traffic patterns for the simulation, as listed in Table 2.

Long-lived FTP traffic: It runs on top of TCP and is generated by using the `BulkSend` Application provided in ns-3. It generates a stream of packets, each of size 512 bytes, till the end of simulation. Moreover, this traffic can be generated in forward, reverse and cross directions. Cross FTP traffic is generated only for parking lot topology.

Streaming video traffic: It runs on top of UDP and is generated by using the `OnOff` Application provided in ns-3. The default packet size is set to 840 bytes and the streaming rate is set to a default value of 640 Kbps [4]. The packet size and streaming rate can be explicitly configured by the user through command line arguments. This traffic can be generated in forward and reverse directions.

Interactive voice traffic: This traffic also runs on top of UDP and is generated by using the `OnOff` Application provided in ns-3. The default packet size is set to 172 bytes and the data rate is set to a default value of 64 Kbps [4]. This is a two-way traffic between the caller and the callee.

3.3 EvalStats

This class is used for collecting post simulation results and store them in files, which are later used to plot graphs. All the performance metrics listed in Table 2 are calculated in this class.

Aggregate Link Utilization: This metric specifies the ratio of current network traffic to the maximum available bandwidth. It is important for a new TCP extension to maximize the bandwidth utilization, while ensuring fairness with other TCP flows.

Mean Queue Length: Maintaining a steady queue length is important to avoid variations in delay. Large variations in delay affect the user perceived application behaviour, especially in the case of interactive voice applications and streaming applications. This metric is important to analyse the stability of a new TCP extension.

Average Packet Drop Rate: This metric is crucial to analyse the performance of TCP in the presence of bursty background traffic. High packet drop rate hurts the performance of time sensitive traffic like Google search, etc.

3.4 Other Classes in `tcp-eval`

TrafficParameters: This class provides setters and getters for configuring the traffic related parameters like:

- Number of forward FTP flows
- Number of reverse FTP flows
- Number of cross FTP flows
- Number of two-way voice flows
- Number of forward streaming flows
- Number of reverse streaming flows

DumbbellTopology: This class sets up a dumbbell simulation scenario, and is placed in a file called `dumbbell-topology.cc` in `tcp-eval` model. First, it creates a dumbbell topology by invoking `PointToPointDumbbellHelper` class which is already available in ns-3. Next, it obtains the simulation parameters from `ConfigureTopology`, generates the traffic using `CreateTraffic`, and finally calculates and stores the results using `EvalStats`. Figure 2 shows user's interaction with `tcp-eval` model for simulating dumbbell

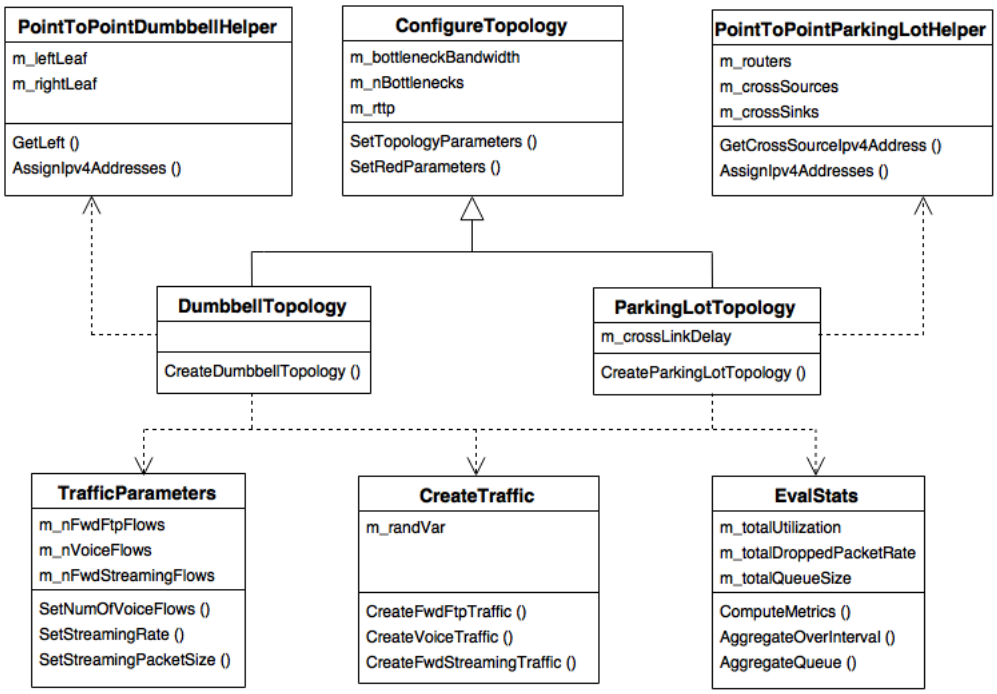


Figure 1: Class diagram of TCP Evaluation Suite in ns-3.

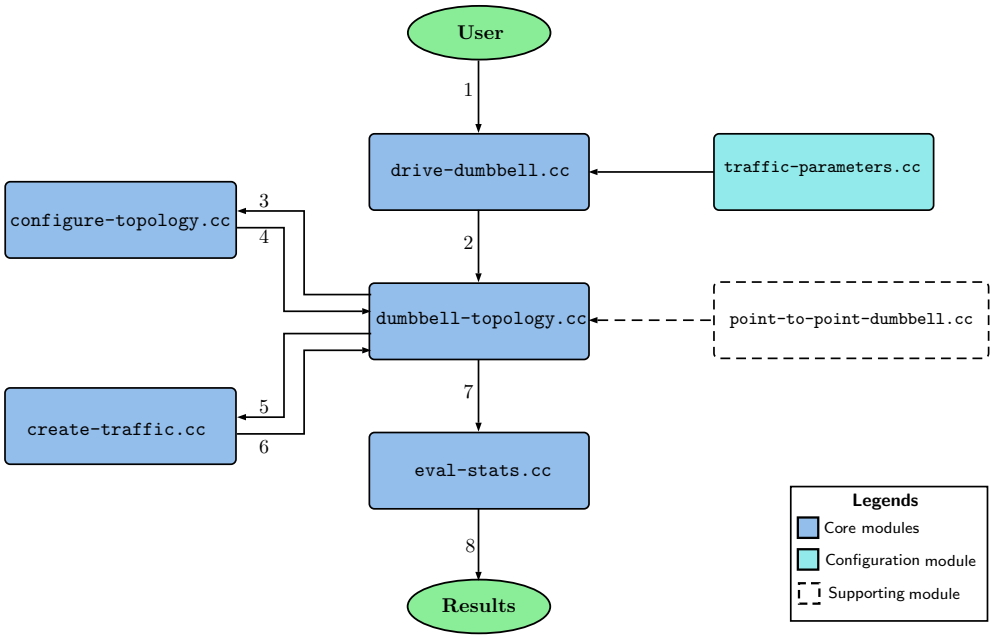


Figure 2: User interaction diagram of `tcp-eval` for dumbbell scenario.

scenarios. `drive-dumbbell.cc` creates an object of `TrafficParameters`, before creating an object of this class.

ParkingLotTopology: While implementing this class we found that a helper for creating multiple bottleneck topology, like parking lot topology, is not available in ns-3. Hence, we implemented a class called `PointToPointParkingLotHelper` and included it in the `point-to-point-layout` model of ns-3. This helper is a stand-alone implementation and not closely linked to `tcp-eval` model. It can be used to simulate parking lot scenarios, even without the `tcp-eval` model.

The functionality of `ParkingLotTopology` class is equivalent to that of `DumbbellTopology`, except that it is responsible to set up a parking lot simulation scenario. This class is placed in a file called `parking-lot-topology.cc` in `tcp-eval` model. Figure 3 shows user's interaction with `tcp-eval` model for simulating parking lot scenarios. `drive-parking-lot.cc` creates an object of `TrafficParameters`, before creating an object of this class.

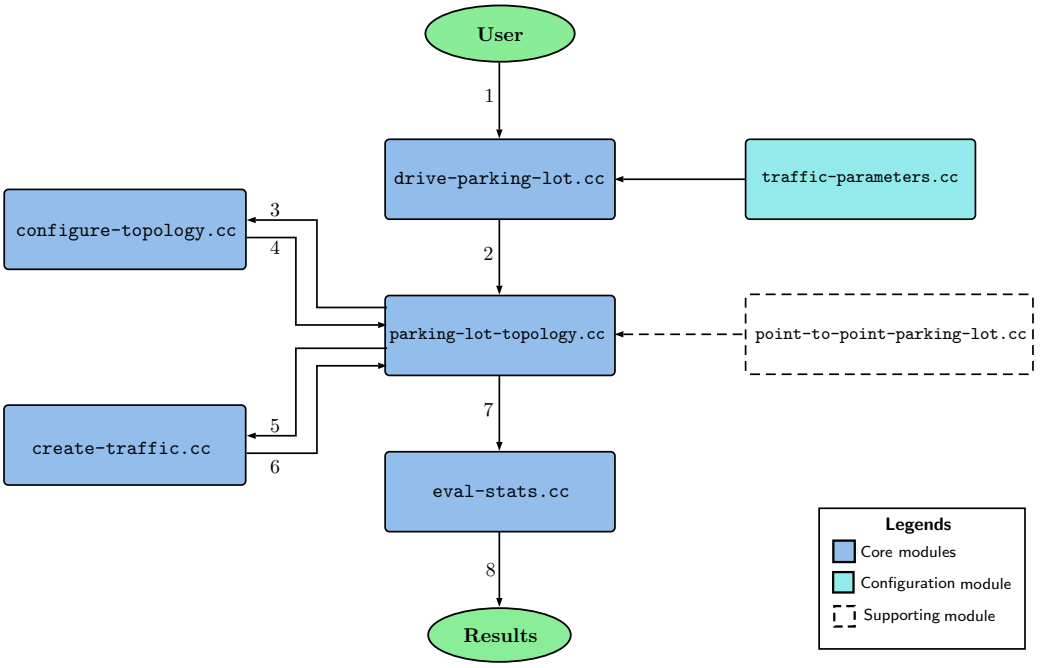


Figure 3: **User interaction diagram of `tcp-eval` for parking lot scenario.**

4. NS-3 TCP EVALUATION USING TCP-EVAL

In this section, we compare the performance of five TCP extensions available in ns-3 by using our implementation of the suite. We provide a set of six shell scripts that run a series of simulations by varying the following three parameters: bottleneck bandwidth, bottleneck RTT and the number of forward FTP flows in two topologies: dumbbell and parking lot. Once the results are collected in respective files for every scenario, shell scripts convert those textual results into graphical form. Further, a PDF file containing graphs for each scenario is created automatically if LaTex is installed.

As suggested by ICCRG, we randomize the start times of all traffic flows to model the real traffic behaviour. Moreover, our implementation provides support for simulating RED algorithm [8] at bottleneck routers. We have used the traditional droptail mechanism in our scenarios, however, to keep the analysis simple. The bottleneck routers can be easily configured to use RED algorithm by passing command line arguments. No further changes are required.

Table 3: **Default simulation parameters.**

Simulation Parameters	Values
Bottleneck bandwidth	10 Mbps
Round Trip Time	80 ms
Number of forward FTP flows	5
Number of reverse FTP flows	5
Number of voice flows	5
Number of forward streaming flows	5
Number of reverse streaming flows	5
Simulation time	100 seconds
Streaming rate	640 Kbps
Streaming packet size	840 bytes

4.1 Varying Bottleneck Bandwidth

In this scenario, the bottleneck bandwidth is varied from 1 Mbps to 100 Mbps. Other parameters shown in Table 3 remain fixed. For every TCP extension, simulation runs till the specified duration for a collection of bandwidth values.

Varying the bottleneck bandwidth provides an estimate of TCP's performance under the same traffic load, but different bottleneck capacity. Figure 4 and 5 depict the simulation results obtained by varying bottleneck bandwidth in dumbbell and parking lot topologies, respectively. It can be observed that the bottleneck link occupancy in both the figures is close to 100% for all TCP extensions when the bandwidth values are relatively low, and it gradually decreases with the increase in the bandwidth. It is important for any TCP extension to adapt its congestion window to ensure that bottleneck bandwidth remains fully utilized.

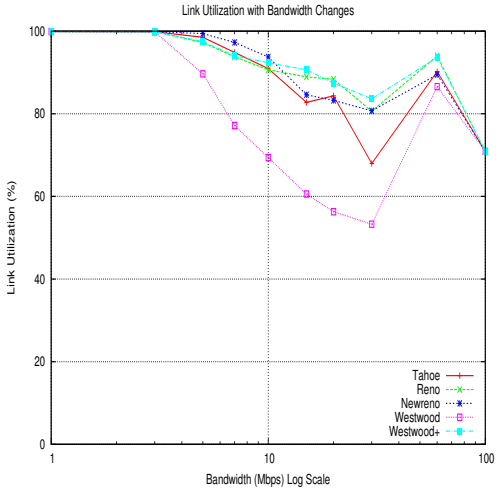
4.2 Varying RTT

In real internet scenarios, the hosts can be distributed over the large span of geographical area. Such hosts can be modelled by varying the propagation delays between the nodes in the simulation environment. Hence, in this scenario, we change the RTT values between the bottleneck routers from 10 milliseconds to 1 second, while keeping other values fixed as shown in Table 3.

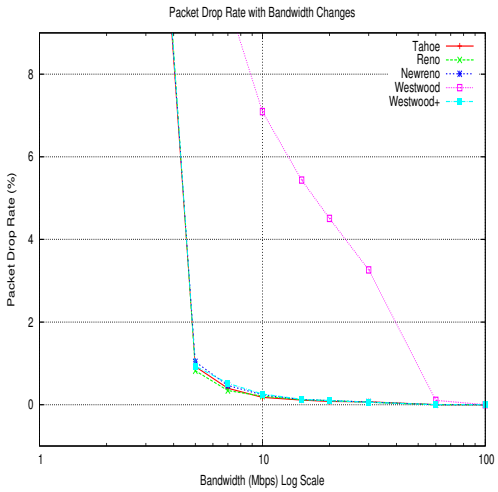
Figure 6 and 7 depict the simulation results obtained by varying the RTT values between the bottleneck routers in dumbbell and parking lot topology. The results clearly depict the performance of each TCP extension, and how its behavior differs from other TCP extensions.

4.3 Varying the Number of FTP Flows

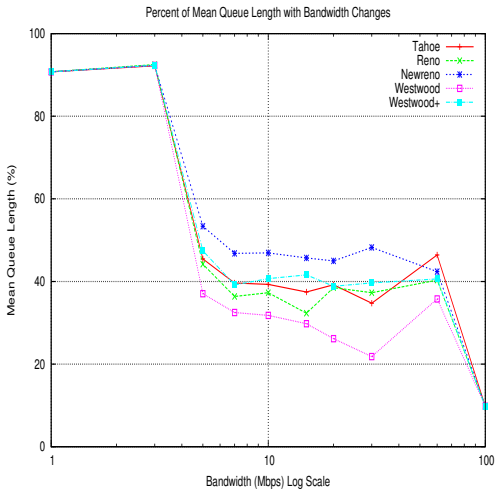
This scenario is designed to test the performance of TCP extensions by varying the traffic load. We run a series of simulations by varying the number of forward FTP flows from 1 to 100. Other simulation parameters listed in Ta-



(a) **Aggregate bandwidth utilization.**

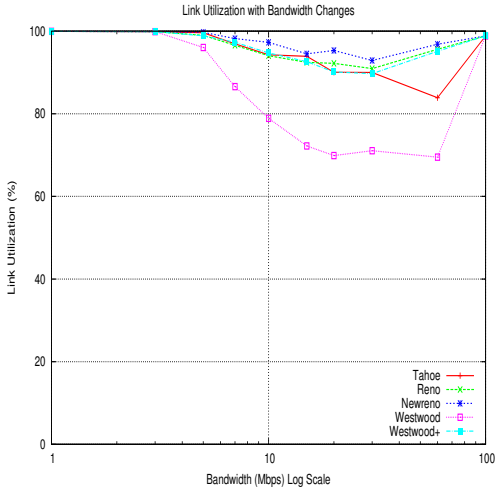


(b) **Packet drop rate.**

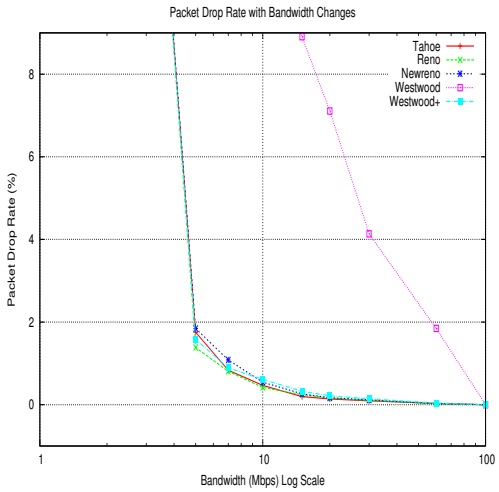


(c) **Mean queue length.**

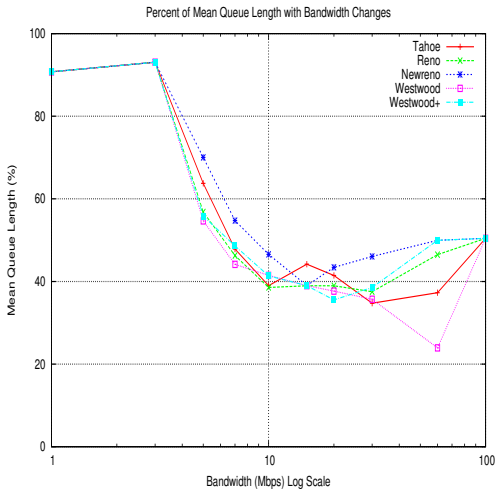
Figure 4: TCP performance results for changing bandwidth for dumbbell topology.



(a) **Aggregate bandwidth utilization.**

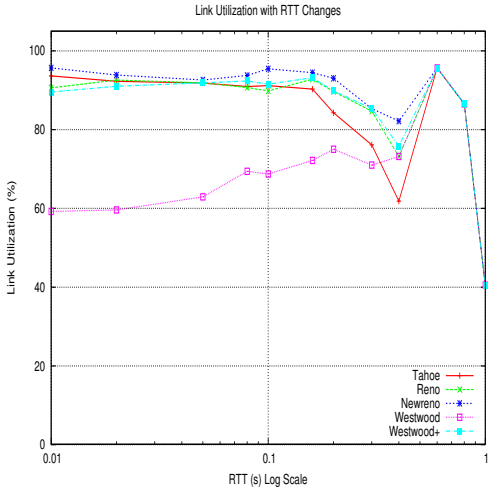


(b) **Packet drop rate.**

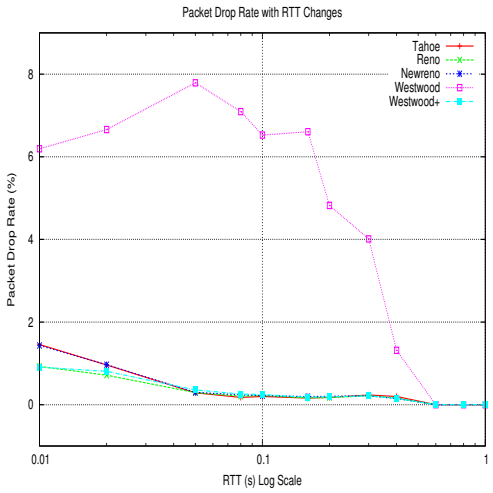


(c) **Mean queue length.**

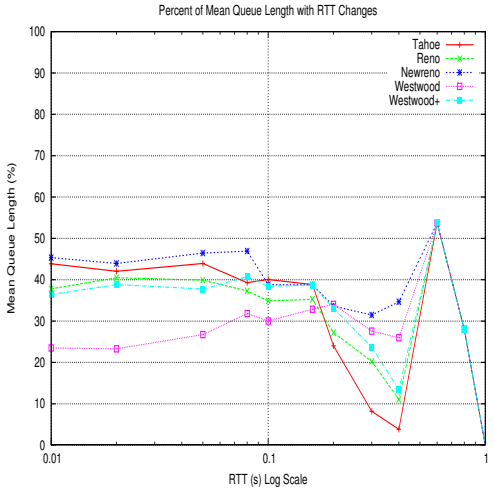
Figure 5: TCP performance results for changing bandwidth for parking lot topology.



(a) **Aggregate bandwidth utilization.**

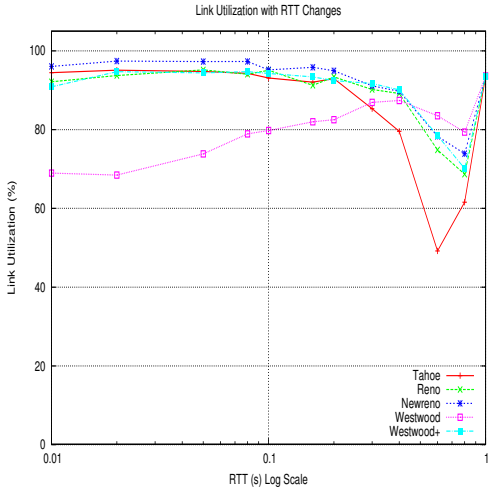


(b) **Packet drop rate.**

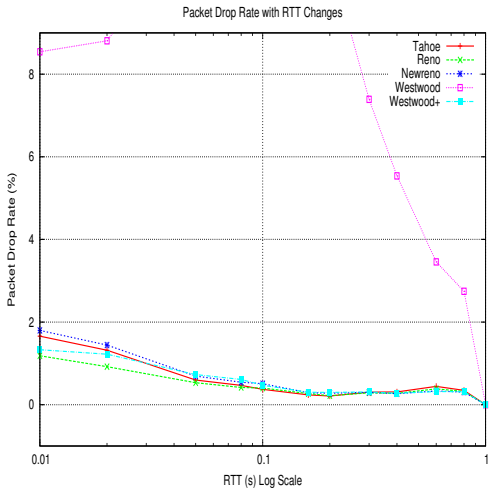


(c) **Mean queue length.**

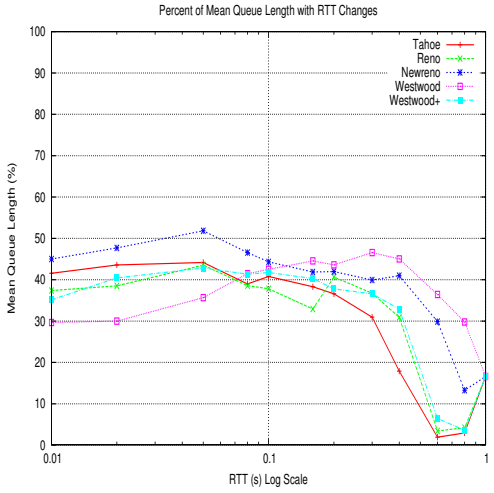
Figure 6: TCP performance results for changing RTT for dumbbell topology.



(a) **Aggregate bandwidth utilization.**

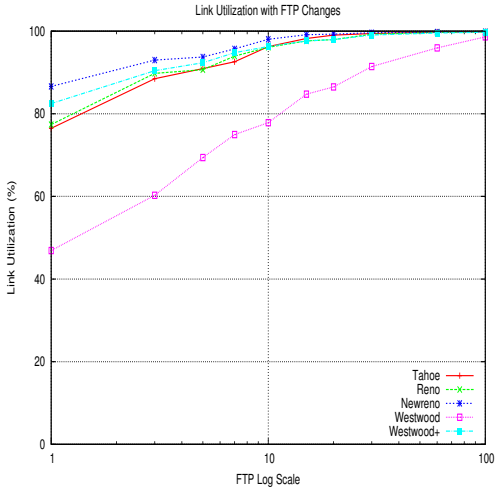


(b) **Packet drop rate.**

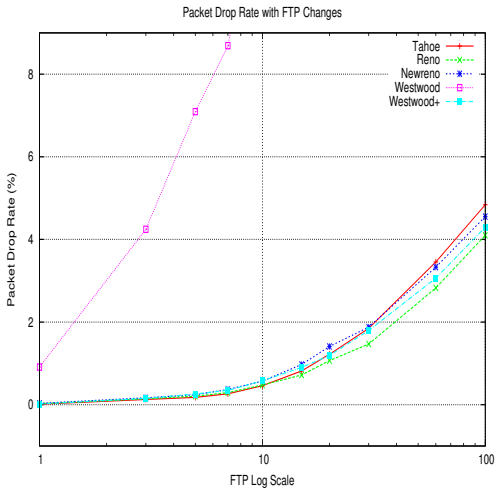


(c) **Mean queue length.**

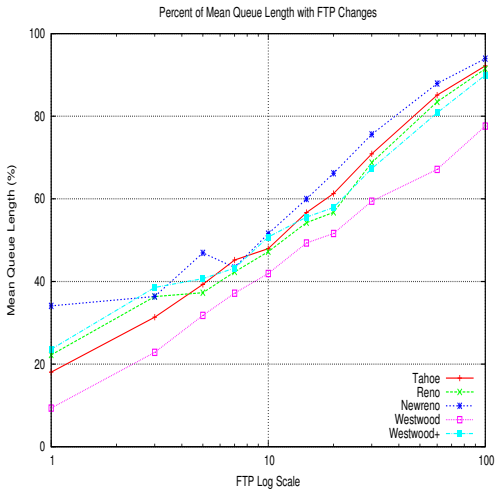
Figure 7: TCP performance results for changing RTT for parking lot topology.



(a) **Aggregate bandwidth utilization.**

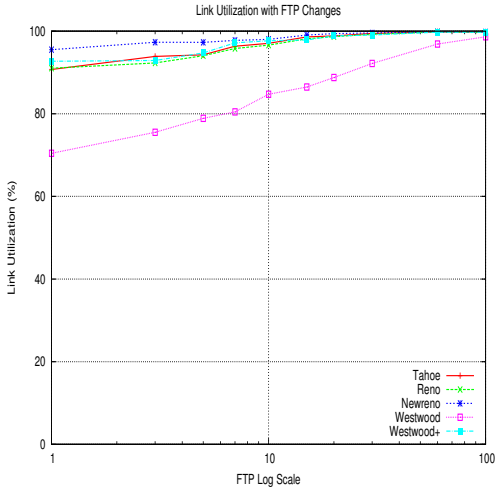


(b) **Packet drop rate.**

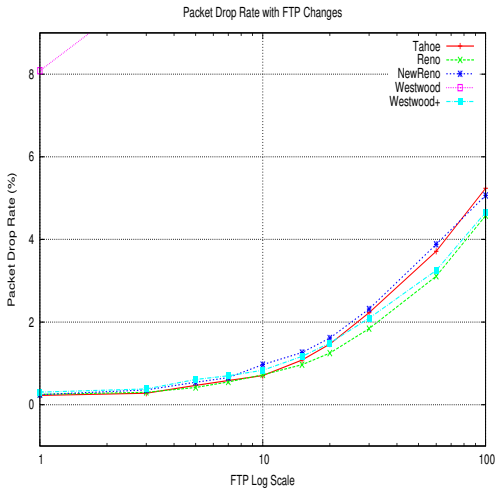


(c) **Mean queue length.**

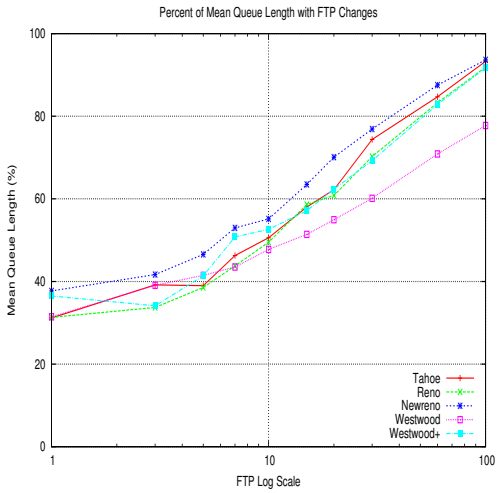
Figure 8: TCP performance results for changing FTP flows for dumbbell topology.



(a) **Aggregate bandwidth utilization.**



(b) **Packet drop rate.**



(c) **Mean queue length.**

Figure 9: TCP performance results for changing FTP flows for parking lot topology.

ble 3 remain fixed. Figure 8 and 9 depict the simulation results obtained for dumbbell and parking lot topology respectively. The bottleneck link utilization remains close to 90% for small number of FTP flows, and as expected, gradually increases with the increase in traffic load.

All graphs presented for this scenario and previous two scenarios are automatically generated using the shell scripts. Instructions to reproduce these results are provided in the Appendix.

5. CONCLUSION AND FUTURE WORK

In this paper, we present an implementation of TCP evaluation suite as a separate model in ns-3. We provide the implementation details for every class and highlight the interactions among them. Additionally, we demonstrate the usage of our TCP evaluation model by comparing five TCP extensions available in ns-3 in benchmark scenarios stated in the internet drafts. The results are collected for aggregate link utilization, mean queue length and packet drop rate, and the statistics are generated in textual and graphical formats.

We plan to further extend our implementation and add other features suggested in the draft of ICCRG. We have completed porting Tmix traffic generator to work with the latest version of ns-3. The next step would be to integrate it with our implementation and provide flexibility in terms of using realistic traffic patterns for evaluating the performance of TCP extensions.

6. ACKNOWLEDGMENTS

We would like to acknowledge the support of Gopika Pai, H. J. Bhargav, Pratheek B., Sourabh S. Shenoy for helping in the implementation of EvalStats class. Further, we would like to acknowledge Radhesh Anand for correcting the parts of this paper.

7. REFERENCES

[1] D. Hayes, D. Ros, L. L. H. Andrew, and S. Floyd. Common TCP Evaluation Suite. Internet-Draft draft-irtf-iccrg-tcpeval-01, Internet Engineering Task Force, January 2015. Work in Progress.

[2] Network Simulator 3. https://www.nsnam.org, 2016.

[3] H. Shimonishi, M. Y. Sanadidi, M. Gerla, C. Marcondes, and P. Vasu. TCP Evaluation Suite. http://nrlweb.cs.ucla.edu/tcpsuite/index.html, 2007. Evaluating New Congestion Control Schemes and Its Impact on Standard TCP NewReno.

[4] An NS2 TCP Evaluation Tool. https://sourceforge.net/projects/tcpeval, 2007.

[5] Y. Li, D. Leith, and R. N. Shorten. Hamilton Institute TCP Evaluation Suite. http://www.hamilton.ie/net/eval/hi2005.htm, 2007.

[6] D. Hayes, D. Ros, L. Andrew, and S. Floyd. Common TCP Evaluation Suite. https://bitbucket.org/hayesd, July 2014.

[7] Network Simulator 2. http://www.isi.edu/nsnam/ns, 2016.

[8] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.

APPENDIX

In this section, we provide additional details about reproducing the simulation scenarios described in this paper.

The latest version of ns-3 at the time of writing this paper is ns-3.24 and the same has been used for implementing the TCP evaluation suite. The modified ns-3.24 that contains our TCP evaluation suite implementation can be obtained from here[1]. All the results obtained from our suite get stored in a new directory called `tcp-eval-results` in `ns-3.24` directory.

TCP evaluation suite is implemented in `src/tcp-eval` and its the classes can be found at `src/tcp-eval/model`. The helper for point to point parking lot topology is located at `src/point-to-point-layout/model/point-to-point-parking-lot{.h, .cc}`. `drive-dumbbell.cc` and `drive-parking-lot.cc` are available at `src/tcp-eval/examples`. Several command line arguments can be passed to both; more details can be found in the respective files. Lastly, six shell scripts are provided in `ns-3.24` directory that reproduce the graphs presented in Section 4 of this paper. Each shell script produces three graphs, and if LaTex is found installed on the machine, it places the graphs in respective PDF files on successful completion.

[1]https://github.com/dharmendra-mishra/wns3-2016

OFSwitch13: Enhancing ns-3 with OpenFlow 1.3 Support

Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira
Computer Networks Laboratory, Institute of Computing
University of Campinas (Unicamp), Brazil
luciano@lrc.ic.unicamp.br, {islene, edmundo}@ic.unicamp.br

ABSTRACT

The world is witnessing the rapid evolution of communication technologies, and meeting current market requirements is virtually impossible with traditional network architectures. Many works point to the use of Software Defined Networking (SDN) paradigm and the OpenFlow protocol as enabling solutions to overcome current limitations. Despite the fact that the Network Simulator 3 (*ns-3*) already has a module that supports OpenFlow simulations, it is possible to note that the available implementation provides a very outdated protocol version (0.8.9). As many new major features were introduced up to the latest versions, it is interesting to have some of them available for use. In this context, this paper presents the `OFSwitch13`: a module to enhance the *ns-3* with OpenFlow 1.3 technology support. This module provides both an OpenFlow 1.3 switch device and a controller application interface. Details about module design and implementation are discussed throughout this paper, and a case study scenario is used to illustrate some of the available `OFSwitch13` module features.

CCS Concepts

•**Computing methodologies** → **Model development and analysis; Simulation support systems;** *Simulation environments;* Discrete-event simulation; •**Networks** → *Network simulations;*

Keywords

SDN, OpenFlow 1.3, Network Simulator 3 (*ns-3*)

1. INTRODUCTION

As stated by the Open Networking Foundation (ONF), network operators are facing challenges as the number of connected devices increases [14]. Meeting current market requirements is virtually impossible with traditional network architectures, where the vendor dependence and lack of open interfaces limit the ability of network operators to tailor the network to their individual environments. In this context, SDN has emerged as a network architecture where network control is decoupled from packet forwarding, enabling more agile and flexible networks [14]. The OpenFlow protocol [12] is the first standard technology designed specifically for SDN and has been adopted by a number of networking vendors and by the research community.

As is known, it is very costly to deploy a complete testbed containing multiple networked computers, routers, switches and data links to validate and verify a certain network protocol or a specific network algorithm. In these circumstances, software tools save a lot of money and time in accomplishing this task. When talking about OpenFlow, the straightforward software tool is Mininet [6]. It is an open-source emulator that provides a quick and easy way to prototype and evaluate SDN networks. It creates user-space or kernel full-compliant OpenFlow switches, allowing the use of real controllers and softwares. However, Mininet suffers from the maximum link bandwidth limited by hardware processing power and no time dilation, which prevents it from carrying out emulations when computational demand is higher than the real-time processing capacity. When it comes to the experimentation of the OpenFlow protocol in wireless networks, these shortcomings become more worrisome [3].

A reasonable choice would be using a simulated environment to this end, such as the Network Simulator 3 (*ns-3*) [7]. It is a discrete-event simulator, targeted primarily for research and educational use, and distributed as free software. *ns-3* simulations can model OpenFlow switches via the existing OpenFlow module [4], which relies on an external OpenFlow switch library linked to the simulator. This module implements a very outdated OpenFlow protocol (version 0.8.9 [16]), and too many new major features were introduced up to the latest versions. Among these new features, it is possible to cite multiple tables in the pipeline, group tables, virtual ports, extensible match support, IPv6 support, per flow meters, auxiliary connections, and support for multiple controllers.

To overcome this shortage, this paper introduces the `OFSwitch13` module for *ns-3* [8]. This module provides support for OpenFlow protocol version 1.3 [17], bringing both a switch device and a controller application interface to the simulator, as depicted in Figure 1. With this module, it is possible to interconnect *ns-3* nodes to send and receive traffic using the existing Carrier Sense Multiple Access (CSMA) network devices and channels. To orchestrate the network, the controller application interface can be extended to implement any desired control logic. The communication be-

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915381

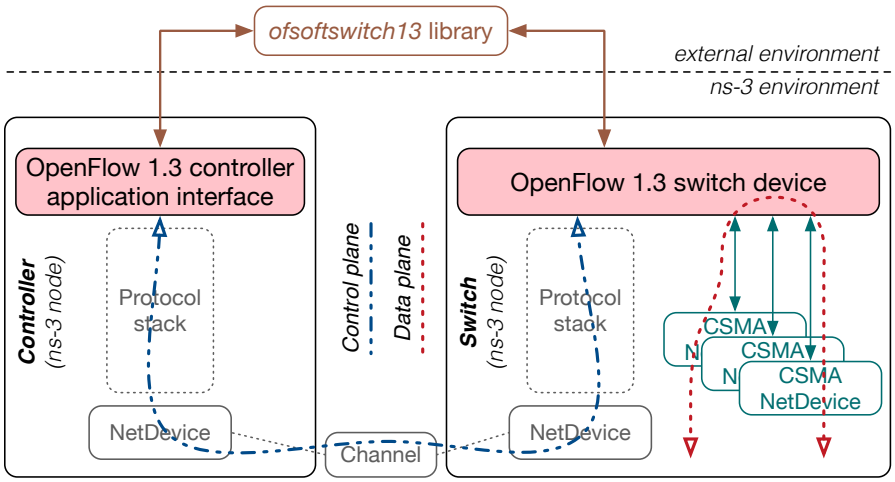


Figure 1: The *OFSwitch13* module overview.

tween the controller and the switch is realized over standard *ns-3* protocol stack, devices and channels. The module also relies on an external library (`ofsoftswitch13`) [11] that provides the switch datapath implementation, the `dpctl` utility tool used to create OpenFlow messages from command lines with simple syntax, and the code for converting OpenFlow messages to and from wire format.

The remainder of this paper describes the SDN paradigm, the module design and implementation, and an illustrative case study scenario. It is organized as follows: Section 2 examines the SDN paradigm and the OpenFlow protocol while Section 3 describes the module design and implementation. Section 4 presents a case study scenario that is used to demonstrate some of the OpenFlow 1.3 module features. Section 5 concludes by summarizing the work described in this paper and looking toward future endeavors.

2. SOFTWARE-DEFINED NETWORKING

Software Defined Networking (SDN) is a paradigm that has been designed to enable more agile and cost-effective networks. In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. By centralizing network intelligence, decision-making is facilitated based on a global (or domain) view of the network, as opposed to today's networks, which are built on an autonomous system view where nodes are unaware of the overall state of the network [15]. To better understand this paradigm, Subsection 2.1 presents the classical SDN architecture while Subsection 2.2 describes some of the main OpenFlow protocol features.

2.1 SDN Architecture

The ONF is taking the lead in SDN standardization and has defined the architecture depicted in Figure 2. This SDN architecture consists of three distinct layers that are accessible through open Application Program Interfaces (APIs):

- The *Application Layer* consists of the end-user applications that consume the SDN communications services. The boundary between the Application Layer and the Control Layer is traversed by the northbound API.

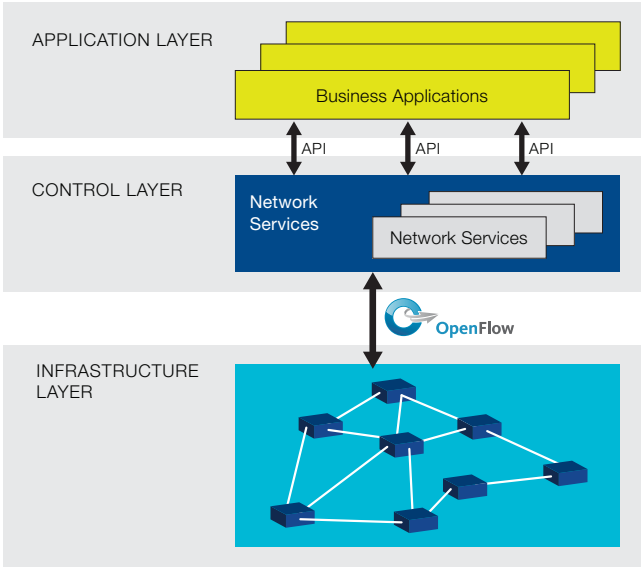


Figure 2: The ONF/SDN architecture [15].

- The *Control Layer* provides the consolidated control functionality that supervises the network forwarding behavior through the southbound API.
- The *Infrastructure Layer* consists of the simplified network elements and devices that provide packet switching and forwarding.

Network intelligence is centralized at control layer, in a software-based controller. The controller can maintain a network global view, providing real-time information, fast optimized routing, etc. By centralizing network state, SDN gives network managers the flexibility to configure, manage, secure, and optimize network resources via dynamic, automated SDN programs. Such programmability enables network configuration to be automated, influenced by the rapid adoption of the cloud. By providing open APIs for applications to interact with the network, SDN networks can achieve unprecedented innovation and differentiation [15].

2.2 OpenFlow Protocol

The OpenFlow protocol [12] is the first southbound interface designed for SDN networks, providing high-performance and granular traffic control across multiple network devices from different vendors. OpenFlow uses the concept of flows to identify network traffic based on predefined match rules that can be programmed by the SDN controller. The OpenFlow protocol is implemented on both sides of the interface between infrastructure devices and the SDN control software. The current specification covers the components and the basic functions of the switch, and the OpenFlow protocol to manage a switch from a remote controller. Figure 3 shows the main components of an OpenFlow switch.

The *OpenFlow channel* is the interface that connects each OpenFlow switch to an OpenFlow controller. Through this interface, the controller configures and manages the switch, receives events and sends packets out to the network. The control channel of the switch may support a single OpenFlow channel with a single controller, or multiple OpenFlow channels enabling multiple controllers to share management of the switch. All OpenFlow channel messages must be formatted according to the OpenFlow protocol. The OpenFlow channel is usually encrypted using Transport Layer Security (TLS), but may be run directly over plain Transmission Control Protocol (TCP).

The OpenFlow switch datapath consists of a pipeline of one or more *flow tables*, which perform packet lookups and forwarding, based on flow entries configured by the controller. Each flow entry consists of OpenFlow eXtensible Match (OXM) Type-Length-Value (TLV) fields to identify the flow, counters, and a set of instructions to apply to matching packets. The matching starts at the first pipeline flow table, following order of priority, and may continue to next tables. If a matching entry is found, the instructions associated with the specific flow entry are applied. Instructions can modify the pipeline processing, sending the packet to one of the following tables, or can contain actions that describe packet forwarding, packet modification, and group table processing. The most common action is the output, which forwards the packet to an output port. Another action is the group action, which directs the packets to another switch datapath element that is called *group table*. Groups represent sets of actions for flooding, as well as more complex forwarding semantics. The switch can also send unmatched packets to the controller, or simply drop the packet.

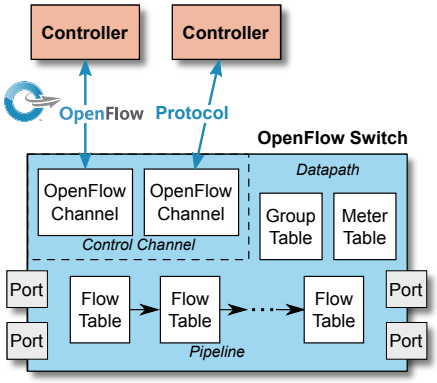


Figure 3: The OpenFlow switch architecture [18].

Another OpenFlow switch datapath element is the *meter table*, consisted of entries defining per-flow meters. Per-flow meters enable OpenFlow to implement rate-limiting, a simple Quality of Service (QoS) operation constraining a set of flows to a chosen bandwidth. A switch can also optionally have one or more queues attached to a specific output port, and in many cases, those two features can be combined to implement complex work conserving QoS frameworks, such as Differentiated Services (DiffServ). The reader can refer to the OpenFlow Switch specification [18] for details on the protocol operation. Note that the Appendix B of the specification contains the release notes highlighting the main changes between major versions of the OpenFlow protocol.

3. *OFSWITCH13* MODULE

This section describes the `OFSwitch13` module design and implementation. Subsection 3.1 brings the description of the switch device, followed by the controller application interface at Subsection 3.2. The OpenFlow channel, used for interconnecting the controller to the switches, is described in Subsection 3.3. The external `ofsoftswitch13` library is presented in Subsection 3.4, and the current module limitations are listed in Subsection 3.5. For details in classes design, please refer to the code API documentation [9].

3.1 OpenFlow 1.3 Switch Device

The OpenFlow 1.3 switch device (hereinafter referred to as switch device) can be used to interconnect *ns-3* nodes using the existing CSMA network devices and channels. Figure 4 shows the internal switch device structure. The switch device takes a collection of ports, each one associated with a *ns-3* underlying CSMA network device. It acts as the intermediary between the ports, receiving a packet from one port and forwarding it to another. The OpenFlow switch datapath implementation (flow tables, group table, and meter table) is provided by the `ofsoftswitch13` library. For this reason, packets entering the switch are sent to the library for OpenFlow pipeline processing before being forwarded to the correct output port(s). Messages received from the controller are also sent to the library for datapath configuration.

A packet enters the switch device through a new *OpenFlow receive callback* in the CSMA network device that is invoked for packets successfully received by the network device. This is a promiscuous receive callback, but in contrast to a promiscuous protocol handler, the packet sent to this callback also includes the Ethernet header, which is necessary for pipeline processing. This is the only required modification to the *ns-3* source code for `OFSwitch13` integration.

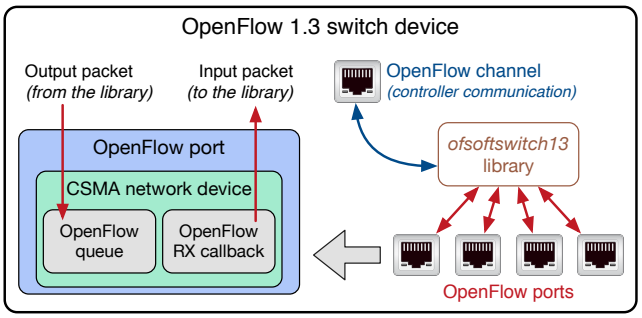


Figure 4: The *OFSwitch13* switch device structure.

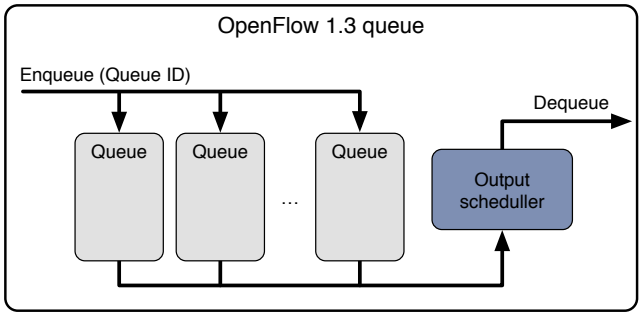


Figure 5: The *OFSwitch13* queue structure.

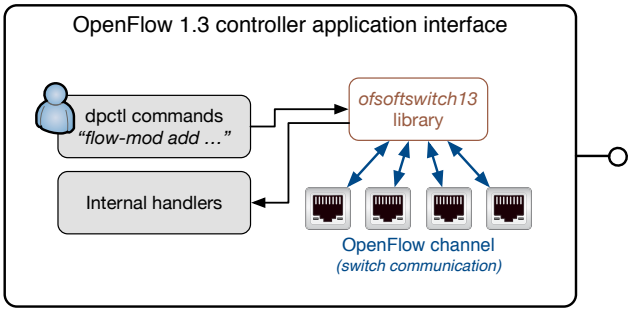


Figure 6: The *OFSwitch13* controller structure.

To model OpenFlow hardware operations, the `OFSwitch13` module considers the concept of *virtual Ternary Content-Addressable Memory (TCAM)* to estimate the average flow table search time [4]. This search time is used to postpone the pipeline processing at the library. To provide a more realistic delay, the module considers that real OpenFlow implementations use sophisticated search algorithms for packet classification and matching. As most of these algorithms are based on binary search trees, the equation $k * log_2(n)$ is used for delay estimation, where k is the constant attribute set to the time for a single TCAM hardware operation and n is the current number of flow entries in the pipeline.

Packets coming back from the library for output action are sent to the *OpenFlow queue*. An OpenFlow switch provides limited QoS support by means of a simple queuing mechanism, where one or more queues can attach to a port and be used to map flow entries on it. The OpenFlow queue extends the *ns-3* queue interface to allow compatibility with the CSMA network devices. Figure 5 shows its internal structure. It can hold a collection of other queues, each one identified by a unique ID. Packets sent to the OpenFlow queue for transmission are expected to carry a *queue tag*, which is used to identify the internal queue that will hold the packet. Then, the output scheduling algorithm decides from which queue to get packets during dequeue procedures.

3.2 OpenFlow 1.3 Controller Interface

The *OpenFlow 1.3 controller application interface* (hereinafter referred to as controller interface) provides the basic functionalities for controller implementation. As illustrated in Figure 6, it can manage a collection of OpenFlow switches. For constructing OpenFlow messages and sending them to the switches, the controller interface relies on the `dpctl` utility provided by the `ofsoftswitch13` library. With a simple command-line syntax, this utility can be used to add flows to the pipeline, query for switch features and status, and change other configurations. For OpenFlow messages coming from the switches, the internal collection of handlers are used to deal with the different types of messages. Some handlers can not be modified, as they must behave as already implemented. Other handlers can be overridden by derived controllers to implement the desired control logic.

The `OFSwitch13` module brings a *learning controller* that implements the controller interface to work as a learning bridge device without the Spanning Tree Protocol (STP) implementation [5]. It instructs OpenFlow switches to forward incoming unicast frames from one port to the correct output port whenever possible (as in the `BridgeNetDevice`).

3.3 OpenFlow Channel

The *OpenFlow channel* is the interface that connects each switch to an OpenFlow controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch. In the `OFSwitch13` module, the controller interface can manage the switch devices remotely over a separate dedicated network (out-of-band controller connection). It is possible to use standard *ns-3* channels and devices to create an OpenFlow channel using a single shared channel or individual connections between the controller interface and each switch device. This model provides realistic control plane connections, including communication delay and, optionally, error models. It also simplifies the OpenFlow protocol analysis, as the *ns-3* tracing subsystem can be used for outputting PCAP files to be read by third-party software.

Considering that the OpenFlow messages traversing the OpenFlow channel follow the standard wire format, it is also possible to use the *ns-3* `TapBridge` to allow an *external OpenFlow controller*, running on the local machine, to interact with the simulated environment over this control channel. This would allow the use of real controller implementations to manage simulated OpenFlow switches. However, note that this use case has not been validated yet.

3.4 *ofsoftswitch13* Library

The `OFSwitch13` module was designed to work together with the `ofsoftswitch13` user-space software switch compiled as a library [2, 10]. The original implementation was forked and slightly modified for proper integration with the `OFSwitch13` module [11]. The code does not modify the datapath implementation, which is currently maintained in the original repository and regularly synced to the modified one.

Figure 7 shows the library architecture and highlights the `OFSwitch13` interconnection points. The library provides the complete OpenFlow switch datapath implementation, including input and output ports, the flow-table pipeline for packet matching, the group table, and the meter table. It also provides the `OFLib` library that is used for converting internal messages to and from OpenFlow 1.3 wire format, and the `dpctl` utility for converting text commands into internal messages. The `NetBee` library [13] is used for packet decoding and parsing, based on the *NetPDL* XML-based language for packet header description [19].

For proper *ns-3* integration, the switch ports were set aside, and the library was modified to receive and send packets directly to the *ns-3* environment. To accomplish this task, all library functions related to sending and receiving

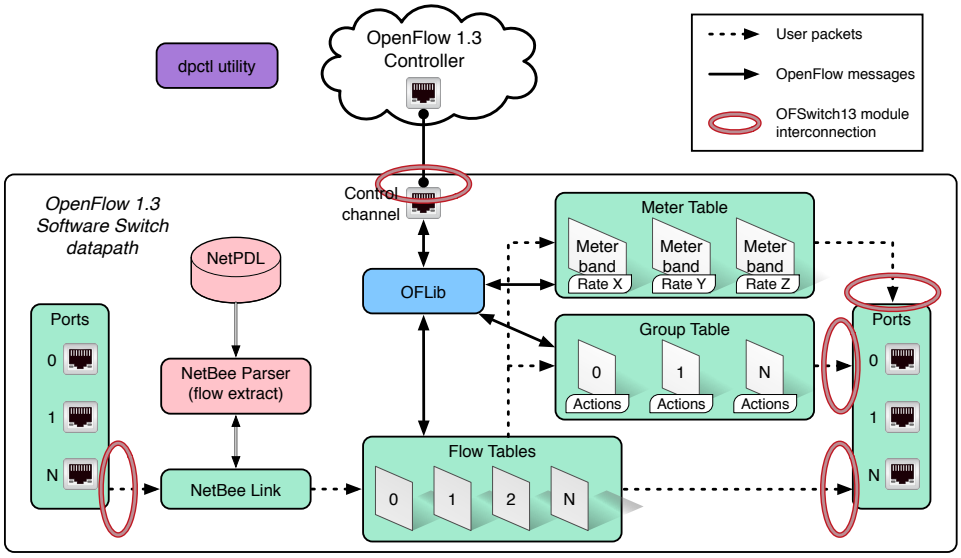


Figure 7: The *ofsoftswitch13* library architecture (adapted from [2]).

packets over ports were annotated as *weak symbols*, allowing the module to override them at link time. This same strategy was used for overriding time-related functions, ensuring time consistency between the library and the simulator. The integration also relies on *callbacks*, which are used by the library to notify the module about internal packet events, like packets dropped by meter bands, packet content modifications by pipeline instructions, packets cloned by group actions, and buffered packets sent to the controller.

One potential performance drawback is the conversion between the *ns-3* packet representation and the serialized packet buffer used by the library. This is even more critical for empty packets, as *ns-3* provides optimized internal representation for them. To improve the performance, when a packet is sent to the library for pipeline processing, the module keeps track of its original *ns-3* packet. For packets processed by the pipeline without content changes, the switch device forwards the original *ns-3* packet to the specified output port. In the face of content changes, the switch device creates a new *ns-3* packet with the modified content (eventually copying all packet and byte tags from the original packet to the new one). This approach is more expensive than the previous one but is far more simple than identifying which changes were performed in the packet by the library to modify the original *ns-3* packets.

3.5 Current Limitations

These are the major limitations of the available module in current implementation:

- *Platform support*: The module implementation is only available for GNU/Linux platforms, and must be compiled with the GNU Compiler Collection (GCC).
- *Auxiliary connections*: Only a single connection between the switch and the controller is available. According to the OpenFlow specifications, auxiliary connections could be created by the switch and are helpful to improve the switch processing performance and exploit the parallelism of most switch implementations.
- *Multiple controllers*: Each switch can only be managed by a single controller. According to the OpenFlow specifications, having multiple controllers would improve reliability as the switch can continue to operate if one controller or controller connection fails.
- *OpenFlow channel encryption*: The switch and controller may communicate through a TLS connection to provide authentication and encryption of the connection. However, as there is no straightforward TLS support on *ns-3*, the OpenFlow channel is implemented over a plain TCP connection, without encryption.
- *In-band control*: The OpenFlow controller manages the switches remotely over a separate dedicated network (out-of-band controller connection), as the switch port representing the switch's local networking stack and its management stack is not implemented.

4. CASE STUDY SCENARIO

The network topology proposed for this case study scenario is described in Subsection 4.1, and it is used to demonstrate how some of the OpenFlow 1.3 module features can be exercised to improve network management. Specifically, a link aggregation, a load balancing, and QoS per-flow metering solutions for this network topology are detailed in Subsections 4.2, 4.3, and 4.4, respectively.

4.1 Network Topology

Figure 8 shows the network topology used for this case study scenario. It represents the internal network of an organization, where servers and client nodes are located far from each other (e.g. in separated buildings). The "long-distance" connection between the sites is via two links of 10 Mbps each, while all the other local connections are 100 Mbps. On the server side, the *OpenFlow border switch* acts as a border router element: it is responsible for handling connection requests coming from the clients, and redirecting them to the appropriate internal server. On the client side, the *OpenFlow client switch* is used to interconnect all clients in a star

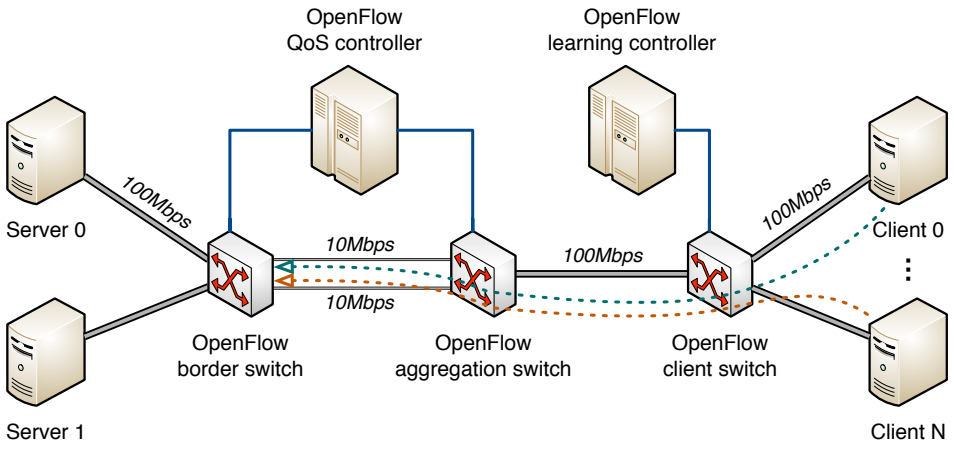


Figure 8: The network topology for the case study scenario.

topology. Between these two switches, there is the *OpenFlow aggregation switch*, located at the border of the client side and used to provide long-distance improved communication. The default learning controller is used to manage the client switch, whereas the new *OpenFlow QoS controller* is used to manage the other two switches. The latter controller implements some QoS functionalities exploiting OpenFlow 1.3 features, as described in the following subsections.

For this case study scenario, a total number of 4 client nodes was used during simulations. Each client opens a single TCP connection with one of the 2 available servers, and sends packets in uplink direction as much as possible, trying to fill the available bandwidth. TCP segment size is set to 1400 bytes and the length of the simulation is set to 100 seconds. The *ns-3* simulation scripts for this case study scenario are available under the `examples/qos-controller/` directory at the `OFSwitch13` source code.

4.2 Link Aggregation

The link aggregation can be used to combine multiple network connections in parallel in order to increase throughput beyond what a single connection could sustain. To implement the link aggregation, the OpenFlow group table can be used to split the traffic.

OpenFlow groups were introduced in OpenFlow 1.1 as a way to perform more complex operations on packets that cannot be defined within a flow alone. Each group receives packets as input and performs any OpenFlow actions on these packets. The power of a group is that it contains separate lists of actions, and each individual action list is referred to as an OpenFlow bucket. There are different types of groups, and the *select* group type can be used to perform link aggregation. Each bucket in a select group has an assigned weight, and each packet that enters the group is sent to a single bucket. The bucket selection algorithm is undefined and is dependent on the switch's implementation (the `ofsoftswitch13` library implements the weighted round robin algorithm).

In the network topology, the QoS controller configures both the border and the aggregation switches to perform link aggregation over the two narrowband long-distance connections, providing a 20 Mbps connection between servers and clients. Each OpenFlow bucket has the same weight in the select group, so the load is evenly distributed over the links.

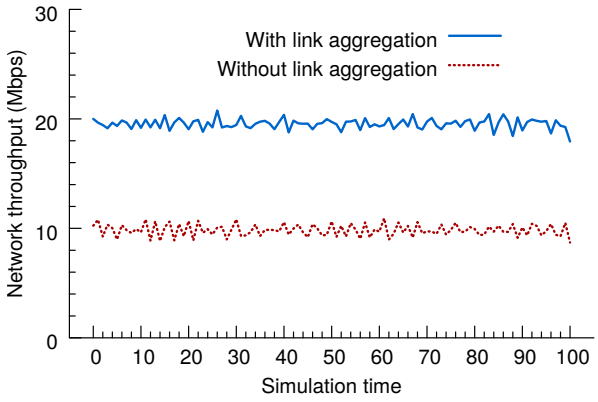


Figure 9: Link aggregation mechanism.

Figure 9 shows the network aggregated throughput, measured at the aggregation switch, for simulations with and without link aggregation. In the case of no link aggregation, only one of the two long-distance links are used, limiting the available bandwidth to 10 Mbps. With the link aggregation, TCP connections can fill all the available bandwidth for both links, coming to 20 Mbps throughput.

4.3 Load Balancing

A load balancing mechanism can be used to distribute workloads across multiple servers. Among many goals, it aims to optimize resource use and avoid overload of any single server. One of the most commonly used applications of load balancing is to provide a single Internet service from multiple servers, sometimes known as a server farm.

In the network topology, the OpenFlow QoS controller configures the border switch to listen for new requests on the Internet Protocol (IP) and port where external clients connect to access the servers. The switch forwards the new request to the controller, which will decide which of the internal servers must take care of this connection. Then, it install the match rules into border switch to forward the subsequent packets from the same connection directly to the

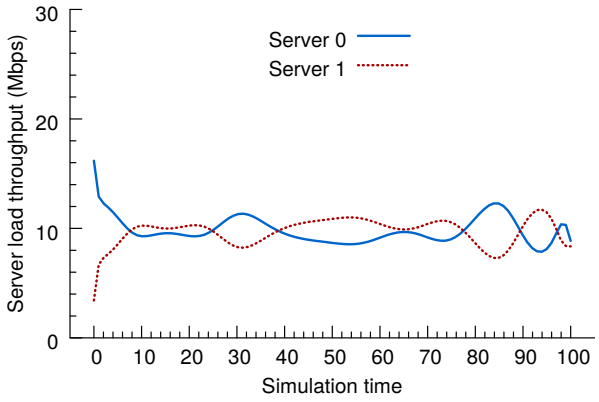


Figure 10: Server load balancing mechanism.

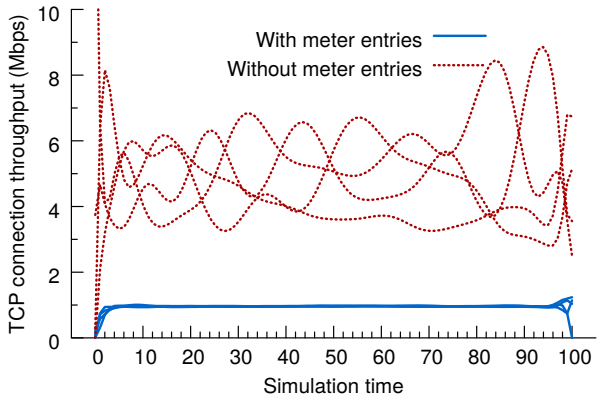


Figure 11: Per-flow metering mechanism.

chosen server. All this happen without the client ever knowing about the internal separation of functions.

To implement this load balancing mechanism, the QoS controller depends on the extensible match support introduced in OpenFlow 1.2. Prior versions of the OpenFlow specification used a static fixed length structure to specify matches, which prevents flexible expression of matches and prevents the inclusion of new match fields. The extensible match support allows the switch to match Address Resolution Protocol (ARP) request messages looking for the server IP address and redirect them to the controller, which will create the ARP reply message and send it back to the network. The set-field action is used by the border switch to rewrite packet headers, replacing source/destinations IP addresses for packets leaving/entering the server farm.

Figure 10 compares the server load balancing in terms of data arrival throughput from uplink TCP connections. In this case study, the QoS controller uses the round robin algorithm to perform load balancing among the two available servers, redirecting half of TCP connections to each one. The link aggregation was enabled during the simulation.

4.4 Per-Flow Meters

OpenFlow meter table, introduced in OpenFlow 1.3, enables the switch to implement various simple QoS operations. A meter measures the rate of packets assigned to it and enables controlling the rate of those packets. The meter triggers a meter band if the packet rate or byte rate passing through the meter exceeds a predefined threshold. If the meter band drops the packet, it is called a rate limiter.

To illustrate the meter table usage, the OpenFlow QoS controller can optionally limit each connection throughput to a predefined data rate threshold, installing meter rules at the border switch along with the load balancing flow entries.

Figure 11 compares the smoothed throughput of all 4 TCP connections for simulations with and without per-flow meter entries. The link aggregation was enabled during the simulation, and all meter bands were configured to limit each individual connections at 1 Mbps. It is possible to observe that TCP connection throughput grows up to the available bandwidth for simulations without per-flow meter entries (near 5 Mbps for each connection). With meter entries, the throughput of all TCP connections is limited to 1 Mbps, regardless of the available bandwidth.

5. CONCLUSIONS AND FUTURE WORK

SDN represents an important paradigm shift that will enable future networks to be more simple and flexible. Targeting scientific research in this area, this paper introduced the `OFSwitch13`: a module to enhance the *ns-3* simulator with OpenFlow 1.3 technology support. The module design and implementation were presented in the paper, highlighting the available features and listing current limitations. The `OFSwitch13` module is available as free software, and its usage requires minimal changes to the *ns-3* source code. Interested users can visit the project homepage at http://www.lrc.ic.unicamp.br/ofswitch13, which contains links to the code repository and module documentation.

A case study scenario was used to illustrate some of the OpenFlow 1.3 module features: group tables used to perform link aggregation; extensible match support used for fine-grained packet matching, server farm implementation, and load balancing solution; and meter table used for implementing QoS rate-limiting mechanism. The `OFSwitch13` module was also used for *ns-3* simulations integrating OpenFlow and Long Term Evolution (LTE) networks, which are described in another work of the same authors [1].

As future work, it is intended to improve the module with new features to overcome current limitations. Especial attention will be dedicated to the support of auxiliary connections and multiple controllers, which are considered as important features of OpenFlow version 1.3. In addition, it is intended to create a set of *ns-3* tests to endorse the module validation, and also to evaluate its scalability in terms of the size of the simulated topology.

As `OFSwitch13` is free software, contributions can also be made by interested developers and users. Note that the module offers support to OpenFlow 1.3 as a result of the `ofsoftswitch13` library datapath implemented version. Once the library is updated to a newer OpenFlow protocol version (the latest version is 1.5.1 [18]), it would be possible to update de module to support the recent features.

6. ACKNOWLEDGMENTS

The authors would like to thank Vítor Marge Eichemberger and Eder Leão Fernandes for their contributions to module design and implementation. They also would like to thank Capes and CNPq - Brazil, for the financial support (process 118198/2014-9).

7. REFERENCES

[1] L. J. Chaves, V. M. Eichemberger, I. C. Garcia, and E. R. M. Madeira. Integrating OpenFlow to LTE: some issues toward Software-Defined Mobile Networks. In *International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, Paris, France, July 2015.

[2] E. L. Fernandes and C. E. Rothenberg. OpenFlow 1.3 software switch. In *Brazilian Symposium on Computer Networks and Distributed Systems (SBRC)*, pages 1021–1028, Florianópolis, Brazil, May 2014.

[3] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg. Mininet-WiFi: Emulating software-defined wireless networks. In *International Conference on Network and Service Management (CNSM)*, pages 1–6, Barcelona, Spain, November 2015.

[4] GSoC 2010 OpenFlow. Available at: http://www.nsnam.org/wiki/GSOC2010OpenFlow.

[5] IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges. IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998), IEEE Computer Society, 2004.

[6] Mininet: An instant virtual network on your laptop (or other PC). Available at: http://www.mininet.org.

[7] *ns-3* Network Simulator. Available at: http://www.nsnam.org.

[8] OpenFlow 1.3 module for *ns-3*. Available at: http://www.lrc.ic.unicamp.br/ofswitch13.

[9] OpenFlow 1.3 module for *ns-3* API documentation. Available at: http://www.lrc.ic.unicamp.br/ofswitch13/doc/html/index.html.

[10] OpenFlow 1.3 software switch. Available at: http://cpqd.github.io/ofsoftswitch13/.

[11] OpenFlow 1.3 software switch for *ns-3*. Available at: https://github.com/ljerezchaves/ofsoftswitch13.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.

[13] The NetBee library. Available at: http://www.nbee.org/doku.php.

[14] Open Networking Foundation. Software-Defined Networking: The new norms for networks. White Paper, 2012.

[15] Open Networking Foundation. OpenFlow enabled mobile and wireless networks. Solution Brief, 2013.

[16] OpenFlow 0.8.9. OpenFlow switch specification v0.8.9. ver. 0.8.9, Open Networking Foundation, 2008.

[17] OpenFlow 1.3.5. OpenFlow switch specification. OpenFlow Spec v1.3.5, Open Networking Foundation, 2015.

[18] OpenFlow 1.5.1. OpenFlow switch specification. OpenFlow Spec v1.5.1, Open Networking Foundation, 2015.

[19] F. Risso and M. Baldi. NetPDL: An extensible XML-based language for packet header description. *Computer Networks*, 50(5):688–706, 2006.

Analysis of Programming Language Overhead in DCE

Jared S. Ivey
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA
j.ivey@gatech.edu

George F. Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA
riley@ece.gatech.edu

ABSTRACT

In the context of network simulation, directly executing the code for new protocols and applications can add a sense of realism to the simulation while saving time that would have been required for development and validation of sufficiently representative models. Direct Code Execution (DCE) in ns-3 provides this functionality for debugging and analyzing new network applications directly in simulation. However, DCE currently requires that these applications are executed as native code that has been compiled from source code written in C or C++. This work exploits the fact that languages such as Python and Java launch their applications through native code that ultimately translates their source code into instructions that the underlying system understands. The efforts of this study provide a framework for allowing applications written in Python or Java to be executed within the simulated environment similarly to how DCE currently allows applications written in C or C++ to be introduced into simulation. This framework is tested on multiple simulated topologies for a variety of applications, and its overhead is examined in the context of memory usage and wallclock execution time.

CCS Concepts

•**Computing methodologies** → **Discrete-event simulation;** *Simulation evaluation;* •**Software and its engineering** → *Object oriented languages;*

Keywords

Network Simulation, ns-3, Direct Code Execution, Programming Languages, C/C++, Java, Python

1. INTRODUCTION

The field of network communications is constantly evolving and expanding with the introduction of new technologies and protocols aimed at providing greater quality, resiliency, and security to the vast amounts of data traversing current networks. In this ever-changing field, modeling and simulation provide an avenue for examining the traffic within new or existing network topologies. In simulating a network, characteristics and metrics of the topology may be derived without interfering with the existing framework or incurring an immediate hardware or software cost. Popular network simulators such as ns-3 are effective tools for studying these network behaviors. However, adequate coverage of new network programs and protocols within simulation requires that models of these new applications are ported by developers into the simulators. These efforts require a significant amount of time in terms of simply writing the code but also in testing its validity against real-world behaviors. Instead, a mechanism for directly deploying real-world network applications within a simulated environment can alleviate these issues while supplying an air of realism. The Direct Code Execution (DCE) framework in ns-3 can deliver this functionality. It allows real-world protocols and applications to be installed and executed directly on the simulated nodes of a topology created in ns-3.

The concept of direct code execution is primarily constrained to network applications and protocols written in the programming language of the employed simulator. For DCE in ns-3 and some similar network simulators, prior work has demonstrated their effective use for applications written in C or C++, which is conveniently accomplished because these simulators are written in C++. The Python-based flow simulator *fs* includes extensions that can connect it to external applications simply because those applications are also written in Python[1]. The efforts of this work introduce a framework for allowing DCE to accommodate and execute code for network applications written in languages other than C or C++, specifically Python and Java. The executables that launch applications written in these languages are simply native code binaries that are built from C/C++ source code. This work exploits this fact in order to launch Python and Java applications within the DCE environment, allowing their source code to be interpreted line by line from within the context of the simulation. In this way, entirely new sets of network applications can be conveniently examined through the DCE environment without source code modification or the need for a translated port to the languages that DCE understands.

The remainder of this paper is organized into the following sections. Section 2 briefly describes ns-3 and provides detailed information on DCE. The programming languages C/C++, Python, and Java as well as considerations required for their usage in DCE are discussed in section 3. Section

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915383

4 details the applications, topologies, and experiments examined for each programming language. The results are examined and discussed in section 5. Section 6 outlines prior work and use cases of DCE and similar libraries. Section 7 concludes the work and provides potential directions for future research.

2. NETWORK SIMULATOR NS-3

This work uses the network simulator ns-3, a popular discrete event network simulator written in C++. It is used for a variety of educational and research-oriented purposes within the field of computer networking. ns-3 provides simulation and emulation frameworks for developing network topologies and analyzing their network characteristics. In creating network simulations, users of ns-3 have the choice of accessing its libraries by programming in C++ or through Python bindings. Stock or user-created applications may be installed on the nodes of simulated topologies in ns-3, generating and monitoring packets within the simulated network. Data may be collected from the artifacts of these network simulations in order to better understand and gauge the performance of existing and proposed real-world networks.

2.1 Direct Code Execution

DCE is an additional module for ns-3 that can test real-world network applications within the ns-3 environment. It provides the capability to execute userspace and kernelspace network protocols and applications directly within an ns-3 simulation to provide additional realism to the simulated network. These real-world applications are installed on specified nodes within the simulated topology in a similar manner to the stock applications in the ns-3 baseline. The applications themselves typically require no modifications, but they must be rebuilt such that they can act as dynamic binaries rather than static executables. This procedure simply requires that some additional configuration flags are added to the compile and link instructions. Once installed on a node, the applications can interact with the rest of the simulated network through either the ns-3 simulated network stack or the Linux kernel stack. DCE interacts with the installed binary similarly to how an actual operating system would. DCE applications are executed within the simulated environment under a single-process model such that all applications inside their own simulated processes are managed by a single process on the underlying system. This single-process approach provides a convenient debugging alternative to complex, distributed debuggers.

DCE is composed of three layers: the core, the kernel, and a POSIX layer. The *core* layer provides virtualization mechanisms to coordinate the actions of the simulated processes within the context of the ns-3 event scheduler. At this layer, global variables are loaded in physical pages on the underlying single process to be shared by the multiple simulated processes. Threads, stack space, and the heap for each particular process are managed in the core layer as well. The *kernel* layer connects the real-world Linux TCP/IP stack to the simulated physical layer (L2) of ns-3. Traffic from the installed applications can then traverse this hybrid simulated stack from application-level socket function calls.

The POSIX layer of DCE replaces the real GNU C Library (*glibc*), the standard library for the C programming language. Calls to *glibc* functions are caught by this layer to determine how they should be handled. In many cases, DCE does not need to manage the interactions and results of certain *glibc* calls within the context of the simulation. Calls to functions in `string.h` or `math.h`, for example, can simply be passed to the *glibc* of the underlying system. Time-related functions return information about the simulated time rather than the wallclock time. Socket function calls are effectively wrappers to the simulated sockets in relation to the ns-3 stack. Subprocess and threading functions are also handled by this layer to appropriately manage their contexts. Local files are managed in the POSIX layer relative to specific file space for each node running DCE-configured applications. The file space holds all output generated by the application and may also be used to hold any runtime file dependencies[2, 3, 4].

3. PROGRAMMING LANGUAGES

This section provides a simple historical and technical overview of the programming languages examined in this work. It begins by examining the languages for which DCE was originally intended to be compatible, namely C and C++. Through extensions for DCE performed in this work, two additional languages have been made operable within the ns-3/DCE environment. The interpreted language Python and its predominant runtime library CPython are described. Additionally, the Java programming language and its runtime environment, the Java Virtual Machine (JVM), are discussed.

3.1 C/C++

The C programming language is considered one of the first mainstream, general-purpose programming languages. It was developed by Dennis Ritchie from 1969 to 1973 at Bell Telephone Laboratories (now AT&T Bell Labs). It is a statically typed, procedural language that has been adopted for use in a variety of systems. It is one of the "lower" high level languages available with many newer languages, including C++, Java, and Python, employing it as an intermediary at some point in their respective compiler/runtime pipelines. As stated in section 2.1, the standard library for the C programming language, referred to as *glibc*, provides a substantial level of functionality. String and memory manipulation, mathematical functions, system time information, file and socket handling, parallel processing, and a variety of other capabilities are available in *glibc*.

The C++ programming language, created by Bjarne Stroustrup at AT&T Bell Labs in 1983, was originally intended as an object-oriented enhancement to C. In addition to similar features as C, C++ enables class creation complete with abstraction, encapsulation, inheritance, and polymorphism, templates, and operator overloading. A standard API is provided by the C++ Standard Library, which *glibc* currently supports. Both C and C++ are compiled to native code that an underlying computer system can recognize equivalently. This characteristic results in them being effectively identical from the viewpoint of the DCE environment.

To operate as DCE applications, programs written in C or C++ must be recompiled such that DCE recognizes them as dynamic libraries rather than static executables. In this way, DCE can load the `main` function of a particular program as if it was just another addressed symbol in the memory space of the dynamically loaded library. Compiling the source code into position-independent code with the `fPIC` flag allows it to be loaded similarly to a shared object library. Subse-

quently, linking with `pie` produces a position-independent executable and `rdynamic` ensures that all of its symbols will be loaded into the dynamic symbol table. Regarding operability within DCE, the addition of these flags to the compile and link steps is generally the only modification required for building a target application for execution in DCE. However, additional considerations may be necessary if certain functions used by the application are not currently implemented in the *glibc* coverage of the DCE baseline.

3.2 Python

The Python programming language is a multi-paradigm, general-purpose, high-level language that aims to be more readable than other general-purpose, object-oriented languages, such as C++ or Java. It was developed by Guido van Rossum and initially released in 1991. Today, programming in Python is available in two versions: Python 2 (most recently 2.7) and the backwards-incompatible Python 3 (currently 3.5). The reference implementation for both versions is CPython, whose source is written in C. CPython acts as a source code interpreter more so than a compiler. When Python programs (or *scripts*) are provided as inputs to the `python` command, the code will be read and directly executed by the Python runtime. This fact suggests that, with the proper configuration, the `python` command can be built to operate in the DCE environment. In this way, when it is provided a Python script, the source lines will be interpreted, and the underlying *glibc* calls can be handled by DCE. The focus of this work is the CPython implementation of Python 2.7.

The CPython source code did not require any modifications. In building it into the `python` executable and libraries, it required the typical flags such that DCE could recognize and load it, i.e. `fPIC` for compile and `pie` and `rdynamic` for linking. Similarly to *glibc*, Python is equipped with the Python Standard Library (PSL), a set of objects and APIs written in Python that provide a standardized baseline of Python programming capabilities. Because the PSL is written as Python scripts and not a "built" library in the same sense as *glibc*, it is loaded in a different way. A Python script is generally "imported" into the Python runtime through the `python` command. In the context of DCE, this step occurs after the `python` executable is loaded and executed. In this way, the effective kernel space has transitioned into simulation. At this point, the location of the user-defined Python application and baseline Python scripts in the PSL such as `os`, `socket`, and `string` must be placed where the DCE-enabled node can "see" them. To make the files visible to the simulated node, the user-defined script is copied into the filespace for the node, and the PSL is symbolically linked under this filespace as well.

Integrating CPython into the DCE framework is relatively straightforward and follows the typical process for installing and testing new programs on DCE-enabled nodes in an ns-3 topology. Code within new programs that is currently not recognized by the POSIX layer of DCE can be marked to use the system *glibc* library with the `NATIVE` macro. A number of `math.h` functions are needed within the Python library such as `exp`, `log`, `pow`, `__isnan`, `__isinf`, etc. Some *pthread* attribute functions are set to execute natively as well such as `pthread_attr_init`, `pthread_attr_setscope`, and `pthread_attr_destroy`. Since these functions only handle pthread attributes and not the actual threads, they can be manipulated natively with relatively low risk. For a function that cannot be run natively, the `DCE` macro can be utilized to either override the function behavior or wrap it within the context of the simulation. Conveniently, the Python library required relatively few functions needing the `DCE` macro. One example is `sendfile` which is intended to move data between file descriptors more quickly than a combination of `read` and `write`. In the context of simulation, this efficiency is less of a requirement, and as such, its DCE version can simply be the `read-write` combination. Another function – `__rawmemchr` – is only available in the binary standard rather than the source standard in the *glibc* base. Hence, DCE cannot recognize it natively. Instead, `__rawmemchr` must be forward-declared and wrapped around a simpler call to `strchr` as a workaround. Furthermore, a patch providing `epoll` support is incorporated as well from [5].

3.3 Java

Java is a general-purpose, object-oriented programming language similar in some ways to C++. It was created by James Gosling and released in 1995 through Sun MicroSystems (acquired by Oracle). Java applications are compiled but not to native code. Instead, they are converted to Java bytecode which can be executed on the JVM. The JVM along with the standard Java Class Library (JCL) comprises the Java Runtime Environment (JRE) which provides the APIs and executable environment for running Java programs. When running a compiled Java program, the JRE will initialize the environment, and then the JVM will interpret the provided bytecode into native code that the underlying system can understand. One version of the JVM, HotSpot, provides performance optimizations such as adaptive compilation as well as efficient heap management and garbage collection. Development of Java programs is enabled through the Java Development Kit (JDK), which allows the applications to be compiled and packaged.

The OpenJDK library, an open-source implementation of the Java Standard Edition (SE) Platform, provides a configurable mechanism to build an interface between DCE and the Java programming language. The JRE and JDK provided by Oracle is only distributed as binaries such that they are only available in specific build configurations. In contrast, OpenJDK is available as source code that can be built – compiled and linked – with the position-independent and dynamic flags that allow DCE to load its programs into simulation. The "program" of specific interest is the `java` command. The source code that produces the command and the JRE and JVM libraries that it calls to configure and execute the Java environment are all written in C. In this way, applications written in Java that are compiled to class files that the `java` command will accept will ultimately be interpreted to *glibc* symbols. These symbols can then be loaded and executed by the DCE environment. The implementation of OpenJDK used by this work is OpenJDK 8.

The JRE, designed around the JVM, requires additional considerations beyond simply addressing *glibc* symbols that the DCE baseline has yet to include. This process did require the inclusion of additional symbols, some related to determining the process location for newly created Java threads within virtual memory. However, accommodating the JVM also needed to address determining networking interfaces for the DCE-enabled nodes. Under the baseline OpenJDK source code, information about network interfaces

and next-hop routes for a system running the JVM is gathered from the `/proc/net/if_inet6` and `/proc/net/ipv6_route` files, respectively. (The JVM handles translating addresses between IPv4 and IPv6 schemes.) The information in these files is relatively easy to gather. However, an interesting issue presents itself in the form of buffer limitations for standard I/O methods such as `sscanf` within the DCE environment. Limitations are implicitly imposed by DCE on the applications it manages simply due to the nature of handling virtual kernel space inside of a simulation. To model the environment the JVM expects as closely as possible, filespace for a DCE-enabled node is created with the `/proc/net` directory in place. However, to accommodate the noted buffer issues, modified versions of the `if_inet6` and `ipv6_route` files are written that only hold the information that the JVM needs. For `if_inet6`, this information is the hexadecimal representations of the IPv6 addresses for a node as well as their indexes and device names. The `ipv6_route` file that is responsible for configuring routes for the node holds the base IPv6 addresses as 32-character hexadecimal values, the hexadecimal prefix length (similar to the IPv4 subnet mask), configuration flags, and the device names.

4. EXPERIMENTS

This section describes three sets of experiments that have been performed to confirm – and in the case of C/C++ applications reaffirm – the relative range of functionality DCE provides for a variety of applications. Since both C and C++ programs would compile to roughly similar native code representations, only C programs have been examined. Alternatively, the choice of compiler between the GNU C Compiler (GCC) and the Clang frontend for LLVM is introduced for examination to gauge potential differences in compiler optimizations. In one set of experiments, a single node is tasked to perform some simple applications written in each language. In the second set of experiments, a simple dumbbell topology tests basic networking functionality for client/server-style applications written in each language. The final experiments task a host to ping every other end host in a ring topology multiple times to examine the scalability of the simulations when handling multiple DCE applications. All experiments have been performed using the baseline ns-3.24.1 and a modified version of DCE 1.7 that implements the described updates from section 3. Performance is examined against the unoptimized builds of ns-3 and DCE as they are the defaults and typically provide the lowest risk for interference with DCE applications.

4.1 Performance Benchmarks

The first set of experiments installs programs written in C, Java, and Python on a single DCE-enabled node. These programs provide computational workload through some simple algorithms to confirm basic language capability in terms of data types and operators. They also test simple functionality in threading and local (loopback) networking, two relatively common features in networking applications.

4.1.1 Matrix Multiplication and Digits of Pi

The matrix multiplication applications for each language create a 1000-by-1000 array of integers and populate all columns in each row with the row index, e.g. array[0] = [0 0 ... 0], array[1] = [1 1 ... 1], etc. The program then performs matrix multiplication of the array with itself and inserts the corresponding values in a separate 1000-by-1000 array. Although it is not algorithmically complex, the matrix multiplication example demonstrates successful memory allocation, basic operand functionality, and simple control statement behavior in each language within the context of DCE. Furthermore, while the matrix multiplication applications in C and Java simply use nested `for` loop constructs, the Python application administers array allocation and multiplication "Pythonically" using the following construct:

```
[[x for y in range(0,1000)] for x in range(0,1000)]
```

An application to display 100,000 digits of π is written for each language based on the spigot algorithm in [6]. This application tests control statements similarly to the matrix multiplication program. However, additional mathematical complexity is administered, and string formatting is tested as well.

4.1.2 Simple Threading

A threaded application is examined to confirm successful import and usage of threads in Python and Java. The application written for this benchmark creates 500 threads. When each thread executes, it simply iterates 1,000 times printing a string. Threading capabilities using *pthreads* in C are already available in the DCE baseline. Since Python and Java threading functionality ultimately encapsulates calls to *pthreads* methods, it is expected that similar functionality can be achieved in these languages, most likely with some processing overhead. Java implements a `Thread` class for handling thread-based parallelism. In Python, threads are created and managed by importing the `threading` library.

4.1.3 Local Ping

An application to test local networking functionality performs an Internet Control Message Protocol (ICMP) echo request 1,000 times to the *localhost* (127.0.0.1) address and determines if ICMP echo replies are returned. For C, the source code for the original PING application[7] is recompiled for DCE. In Java, an `InetAddress` object is created using the `getLocalHost` method. Then, the `isReachable` method is called to determine the reachability of the address. For the Python application, ping functionality uses a library that mimics the C version of PING through raw sockets sending ICMP echo requests[8].

4.2 Simple Topology

A simple dumbbell topology is constructed to confirm successful networking capabilities are achieved in C/C++, Java, and Python. The dumbbell consists of two inner nodes acting as routers that are connected by a point-to-point link with a data rate of 100Mbps and a speed-of-light delay of 1ms. The outer 2 nodes each connect to one router with a 10Mbps link with 1ms delays. Additionally, network interfaces for the links are configured on different 255.255.255.0 subnets (CIDR /24) to confirm packet transmission occurs through successful L3 routing and not simply ARP requests.

Both end hosts in the described topology are enabled for installation of DCE applications. One end host is installed with a DCE application that acts as a client pushing data to the other end host. The client will create a TCP socket, establishing a connection with the other end host. Following successful connections, the client allocates 65,536 bytes for transmission. This amount is selected to ensure that

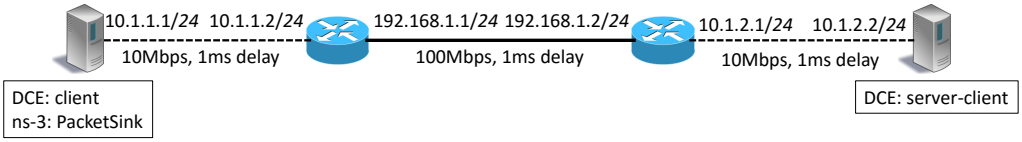


Figure 1: The simple dumbbell topology is used to confirm successful socket handling and data transmission and reception for C/C++, Java, and Python. The topology consists of 4 nodes connected linearly with the outer 2 nodes acting as end hosts and the inner 2 acting as routers.

the socket connection will accommodate multiple segments (based on the ns-3 default segment size of 536 bytes) without encountering congestion control, a process of the simulated stack rather than the examined program. These allocated bytes will be transmitted iteratively multiple times to examine the processing overhead of socket writes for the different programming languages. Upon completion of the specified number of transmission iterations, the client program will exit.

At the other end host, a hybrid server/client program is installed. The server aspect of the program binds a TCP socket to the host and sets it to listen on the same port that the client on the other host will attempt to connect. When the client makes a connection to the server, the server program will accept the connection and reads bytes from the socket until the client closes the connection. Upon closing the initial socket connection, the server program will store the number of received bytes, increment the port number, and relay the same amount of bytes back to the first end host on a new TCP connection. Ready to accept packets on this new port is an ns-3 PacketSink application ready to call the `Simulator::Stop` method when it receives the expected number of bytes. The server-turned-client will iteratively transmit 1,024 bytes until it has transmitted as many bytes as it originally received.

4.3 Ping Ring

A ring topology is simulated to examine the scalability of ns-3 when it must handle multiple DCE applications. The ring consists of a variable number of routers connected to one another in a circle. Point-to-point links connect the routers with a data rate of 100Mbps and 1ms delays. Each router is connected to an end host through a separate point-to-point link with 10Mbps rates and 1ms delays. Network interfaces for all links are configured similarly to those in the dumbbell topology using different 255.255.255.0 subnets (CIDR /24) to again confirm packet transmission occurs through successful L3 routing and not simply ARP requests.

Within the topology, one end host is selected for DCE application installations. The applications are the same applications described in 4.1.3. However, instead of testing for the *localhost* address, the end host attempts to reach every other end host in the topology. Pings are sent 1,000 times for each end host to inflate the amount of processing required for each DCE application.

5. RESULTS AND DISCUSSION

All of the described experiments have been performed on a system running 64-bit Linux Mint 17.3 with a 2.5GHz dual-core Intel i5-3210M processor with 8GB of memory. Simulations are executed 10 times for each set of parameters to obtain the average total memory usage in MB and average wallclock execution time. Gathering memory usage is accomplished through the `ps_mem` Python utility, which determines the core memory usage for a running process[9]. During each simulation run, this tool is called periodically to record the maximum usage realized by the ns-3/DCE simulation at any point in execution. This statistic is important as insufficient memory any time during the life of a program would inhibit successful simulation completion.

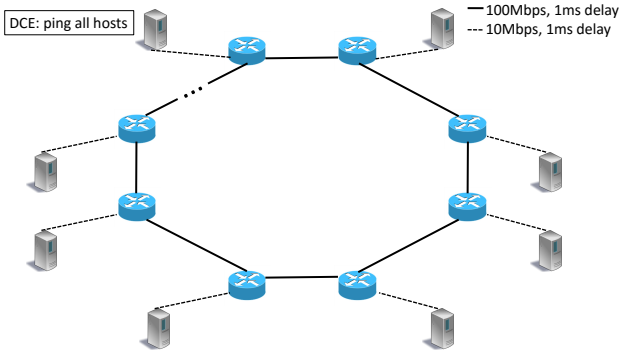


Figure 2: The ring topology is used to examine the scalability of ns-3 simulations when handling multiple DCE applications. The network is comprised of a variable number of router nodes connected in a ring with an end host connected to each one. One host is tasked to ping every other end host a specific number of times.

The benchmark results are shown in Figures 3 and 4 alongside results of each application executed natively in the real Linux environment rather than the ns-3/DCE simulated environment. (The local ping benchmarks are not compared against native versions due to timing differences in the application designs in each programming language.) Java is run both regularly and with the `-Xint` flag to allow it to run in interpreted mode without some of the "performance benefits" of the HotSpot JVM. The programs compiled with GCC and Clang required roughly the same amount of memory and produced similar time results. Based on the programs used in the benchmark simulations, the resource results are as expected. Most of the variables in the C programs had been allocated with stack memory and relatively few variables were utilized, providing little room for any significant compiler optimizations. Interestingly, the Clang-compiled executable for matrix multiplication produced a slight timing improvement over its GCC-built counterpart. This result may suggest that Clang realized some compiler optimizations such as loop unrolling that GCC may not have attempted. Even so,

the timing discrepancies between the two compiler versions are still relatively negligible when compared to the Python and Java versions of the applications. Based on the notion that both Python and Java are effectively invoking C/C++ calls within their underlying libraries, a certain amount of computational overhead is expected. In most of the benchmark tests, a tradeoff appears between lower memory usage with higher wallclock times in Python versus higher resource requirements with quicker execution performance in Java. The printing of the digits of π as well as the threading test produced a relatively significant discrepancy between resource usage and execution timing. However, this tradeoff is actually not realized for the matrix multiplication program. This result suggests that dynamically typed `list` allocation and assignment for the Python arrays incurs both a resource and performance cost compared to the statically typed integer array in Java. Additionally, within the DCE environment, the "performance benefits" of the HotSpot JVM appear to provide some level of speed-up to interpreted-only Java. However, these benefits come with increased memory usage in some cases. Threading appears to be one of the few cases that benefit in terms of memory and time from full access to the JVM. The local ping results are relatively flat compared to the other benchmarks most likely implying that the overall program ultimately lacked a significant amount of processing.

Simple dumbbell topology results are graphed in Figures 5 and 6. For the applications written in C, memory remains approximately constant as address space is simply and explicitly reused while performance intuitively requires more processing time as more processing is required. The Java versions of the programs had to be run in interpreted mode in order to complete. Full usage of the HotSpot JVM in this context is prevented due to its alternation between periods of optimization and deoptimization for profiling and debugging. In testing the framework, it was determined that this feature actually presented conflicts for the addressable space that DCE maintains. Again, an overhead is noted for the interpreted Java and Python programs compared to the C versions. However, interpreted Java memory usage appears to approach a limit suggesting potential memory reuse and adequate garbage collection. On the other hand, Python continues to require more memory with more transmission iterations. One possible reason behind this result may be that Python will continue to dynamically allocate memory as its applications transmit and receive data without reaching a point where garbage collection is deemed appropriate by the interpreter.

Figures 7 and 8 display the results of the ring topology experiments. Starting multiple ping applications for the native code did not require significant overhead in terms of memory or time. In this set of experiments, Python produced wallclock times between Java and interpreted-only Java. However, its memory usage was significantly lower than the dumbbell topology results. The Java program benefited from the `-Xint` flag in this set of experiments, producing lower wallclock times than the Python version with a slight memory usage improvement over the full JVM. However, both experimental runs of the Java program produced significantly higher resource usage than the other programs. This memory usage trend may have been a consequence of starting and stopping the JVM multiple times.

During testing of the OpenJDK within DCE, multiple applications running simultaneously within the limited address space of DCE made it difficult for the JVM to find free space for its code heap. Two Java applications running at the same time as in the dumbbell topology experiments appeared to be the stable limit while more than two simultaneous applications produced inconsistent successful results. DCE applications within the ring topology experiments were given staggered start times such that one would complete before another began. When Java applications had dedicated individual access to the Java runtime as in the ring topology experiments, the JVM resources could be successfully launched and deconstructed without interference. Testing for potential solutions to the simultaneous usage issue is ongoing. However, sufficient use cases are available to advocate for usage of the current framework.

6. RELATED WORK

DCE has previously been used in numerous experiments to enhance the realism of network simulations. Demonstrations of DCE in literature have primarily focused on the fields of mobile and wireless networks where updated protocols are introduced frequently as the technology improves. Rather than constantly porting and modifying these procedures into simulation, it can be much quicker to simply introduce these protocol implementations into simulation via DCE. In [10], a comparison of the ns-3 Optimized Link State Routing (OLSR) model with an actual OLSR daemon in DCE uncovered deficiencies in both programs. Addressing tuning issues in the simulated model and the daemon program allowed them to be updated to better fit the OLSR RFC. Demonstrations of content-centric networking (CCN) over mobile ad-hoc networks (MANETs) and multipath TCP over LTE and wireless in [11, 12] provide examples of additional use cases for DCE that required no modifications to the original implementations. However, these examples are limited to software binaries that originated from C/C++ source code whereas this work has provided a proof of concept and framework for testing protocols in Java and Python in addition to C/C++.

The rise of software-defined networking (SDN) in the network communications field over the past few years has provided another valuable avenue for utilizing DCE. In separating the control and forwarding planes of a network, SDN employs a separate controller process for handling the routing decisions of a network. These controller processes are implemented as libraries, such as NOX (C++), POX and Ryu (Python), Floodlight and OpenDaylight (Java), etc. NOX and the Open vSwitch virtual switch kernel have already been demonstrated successfully in [13] for use cases in both wired and wireless networks. The framework discussed in this work will surely benefit the testing of controller applications written for some of these other libraries as well.

The Open Network Emulator (ONE) [14] is another network simulator providing direct code execution functionality. It provides a compiler framework that converts real-world network applications into modules that can be integrated into simulated network stacks. The source code for these applications requires no modification prior to compilation within the ONE framework. Applications compiled for ONE are built with LLVM into the LLVM intermediate representation (IR)[15]. This notion provides the claim that ONE modules can be language and architecture independent. However, the published literature on ONE and its dis-

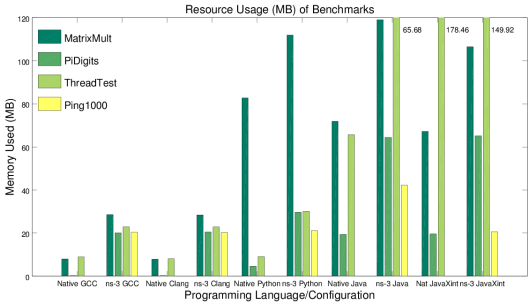


Figure 3: Performance benchmark total memory usage in MB for a single DCE-enabled node and native equivalents for the various programming languages, compilers, and configurations. All data have standard errors that are less than 2% of their respective reported values.

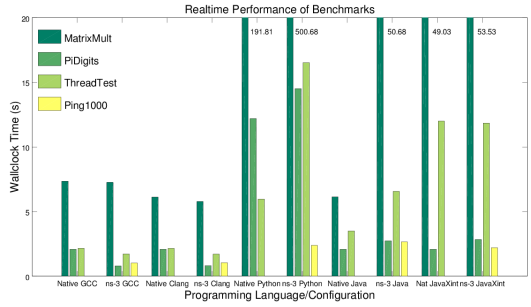


Figure 4: Performance benchmark wallclock execution time for a single DCE-enabled node and native equivalents for the various programming languages, compilers, and configurations. All data have standard errors that are less than 2% of their respective reported values except the GCC and Clang versions of the simple thread test. Their standard errors are 6% of their reported values.

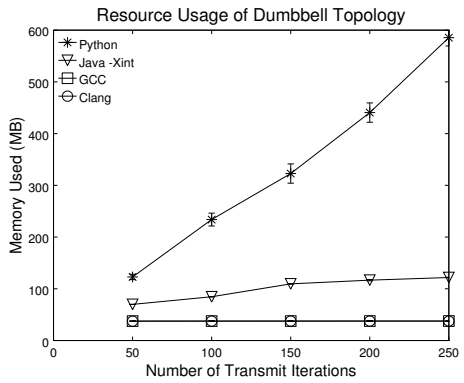


Figure 5: Total memory usage in MB for the simple dumbbell topology for the programming languages, compilers and configurations. Standard error bars are displayed for each data point.

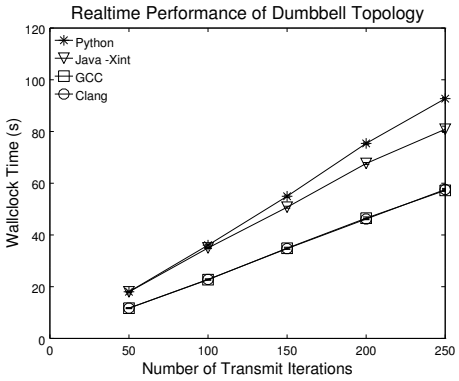


Figure 6: Wallclock execution time for the simple dumbbell topology for the programming languages, compilers and configurations. Standard error bars are displayed for each data point and noted as negligible.

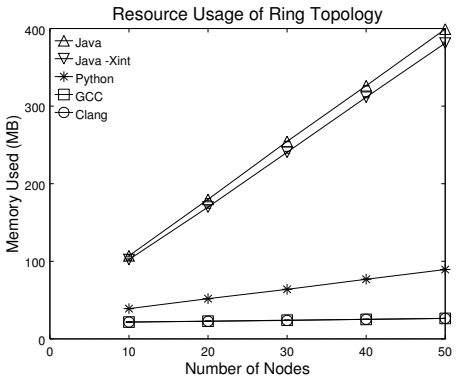


Figure 7: Total memory usage in MB for the ring topology for the programming languages, compilers and configurations. Standard error bars are displayed for each data point and noted as negligible.

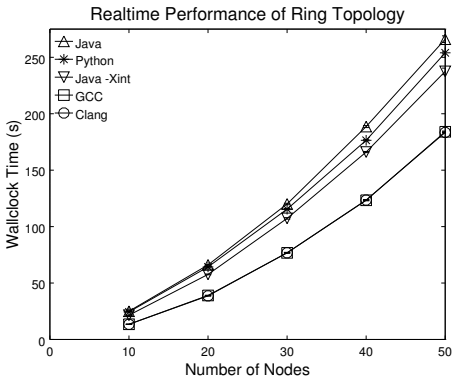


Figure 8: Wallclock execution time for the ring topology for the programming languages, compilers and configurations. Standard error bars are displayed for each data point and noted as negligible.

tributed descendant[16] only examine applications written in C and C++. Based on the subprojects for LLVM[17], these applications are most likely compiled with Clang. For other languages, the DragonEgg plugin allows LLVM to interface with GCC parsers. One such parser, GCJ[18], allows Java source code to be compiled directly to native code. However, it utilizes segmentation fault signals to determine the limits of addressable memory, making it difficult to integrate with DCE.

7. CONCLUSIONS AND FUTURE WORK

This work has described and analyzed a framework using DCE for examining network applications written in Python or Java within simulated topologies in ns-3. This framework provides a convenient way to test entirely new sets of applications written in Python or Java that were not previously compatible to the DCE environment. The framework itself exploits the fact that Python and Java applications are launched and translated ultimately via native code with which DCE can interface. Parts of the underlying language libraries that required modification for successful DCE interaction have been discussed. Multiple applications on topologies of varying complexity have been studied to confirm successful operation of the framework. The resource usage and wallclock execution time of multiple applications written in C, Java, and Python have been gathered and studied to gauge the overhead required to run these applications within simulation.

Future work will continue to enhance the framework to automate some of the discussed environment and library modifications. Efforts to test SDN libraries such as POX and Ryu within the DCE framework have already had preliminary success. Testing will be expanded to determine how well Java controller libraries such as Floodlight and OpenDaylight integrate into the discussed framework. Furthermore, attempts to address the GCJ issue discussed in section 6 will continue in order to better gauge the benefits and shortcomings of the described framework against simply compiling the applications to native code.

8. REFERENCES

[1] M. Gupta, J. Sommers, and P. Barford. Fast, accurate simulation for SDN prototyping. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 31–36, New York, NY, USA, 2013. ACM.

[2] D. Camara, H. Tazaki, E. Mancini, T. Turletti, W. Dabbous, and M. Lacage. DCE: Test the real code of your protocols and applications over simulated networks. *Communications Magazine, IEEE*, 52(3):104–110, March 2014.

[3] H. Tazaki, F. Urbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous. Direct code execution: Revisiting library OS architecture for reproducible network experiments. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 217–228, New York, NY, USA, 2013. ACM.

[4] Inria. Direct code execution (DCE) manual. Website, 2013. https://www.nsnam.org/docs/dce/release/1.5/manual/html/index.html.

[5] H. Tazaki. dce-pre.patch. Website, 2014. https://gist.github.com/thehajime/b2efe96e797cd131cac1.

[6] S. Rabinowitz and S. Wagon. A spigot algorithm for the digits of π. *The American Mathematical Monthly*, 102(3):195–203, 1995.

[7] M. Muuss. The story of the ping program. Website, 1999. http://ftp.arl.army.mil/~mike/ping.html.

[8] J. Diemer. python-ping. Website, 2011. https://pypi.python.org/pypi/python-ping.

[9] P. Brady. ps_mem 3.6: Python package index. Website, 2015. https://pypi.python.org/pypi/ps_mem/3.6.

[10] E. Bikov and P. Boyko. Direct execution of OLSR MANET routing daemon in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 454–461, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[11] S. Kwon, K. Hasan, M. Lee, and S. Jeong. Comparative analysis of real-time video performance in the CCN-based LTE networks. In *Information and Communication Technology Convergence (ICTC), 2015 International Conference on*, pages 509–511, Oct 2015.

[12] H. Tazaki, E. Mancini, D. Camara, T. Turletti, and W. Dabbous. MSWIM demo abstract: Direct code execution: Increase simulation realism using unmodified real implementations. In *Proceedings of the 11th ACM International Symposium on Mobility Management and Wireless Access*, MobiWac '13, pages 29–32, New York, NY, USA, 2013. ACM.

[13] E. Mancini, H. Soni, T. Turletti, W. Dabbous, and H. Tazaki. Demo abstract: Realistic evaluation of kernel protocols and software defined wireless networks with DCE/ns-3. In *Proceedings of the 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '14, pages 335–337, New York, NY, USA, 2014. ACM.

[14] V. Duggirala and S. Varadarajan. Open network emulator: A parallel direct code execution network simulator. In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, PADS '12, pages 101–110, Washington, DC, USA, 2012. IEEE Computer Society.

[15] C. Lattner and V. Adve. LLVM: a compilation framework for lifelong program analysis transformation. In *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pages 75–86, March 2004.

[16] V. Duggirala, C. Ribbens, and S. Varadarajan. Distributed ONE: Scalable parallel network simulation. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, pages 10–16, ICST, Brussels, Belgium, Belgium, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[17] C. Lattner. The LLVM compiler infrastructure project. Website, 2016. http://llvm.org.

[18] GCC Team. GCJ: The GNU compiler for Java. Website, 2016. https://gcc.gnu.org/java.

Implementation and Validation of an IEEE 802.11ah Module for ns-3

Le Tian*, Sébastien Deronne†, Steven Latré*, Jeroen Famaey*

* Department of Mathematics and Computer Science, University of Antwerp – iMinds, Belgium
{le.tian, steven.latre, jeroen.famaey}@uantwerpen.be

† Nokia, Belgium
sebastien.deronne@gmail.com

ABSTRACT

IEEE 802.11ah or HaLow is a new Wi-Fi standard for sub-1Ghz communications, aiming to address the major challenges of the Internet of Things: connectivity among a large number of power-constrained stations deployed over a wide area. Existing research on the performance evaluation of 802.11ah is generally based on analytical models, which does not accurately represent real network dynamics and is hard to adjust to different network conditions. To address this hiatus, we implemented the 802.11ah physical and MAC layer in the ns-3 network simulator, which, compared to analytical models, more closely reflects actual protocol behavior and can more easily be adapted to evaluate a broad range of network and traffic conditions. In this paper, we present the details of our implementation, including a sub-1Ghz physical layer model and several novel MAC layer features. Moreover, simulations based on the implemented model are conducted to evaluate performance of the novel features of IEEE 802.11ah.

CCS Concepts

•Networks → Wireless access networks; •Computing methodologies → Model development and analysis; Simulation evaluation;

Keywords

IEEE 802.11ah, Wi-Fi HaLow, ns-3, Fast Association, Restricted Access Window

1. INTRODUCTION

The Internet of Things (IoT) consists of millions of constrained devices addressable over the Internet. Existing low-power wireless network technologies for connecting such devices can be categorized into two groups: (i) wireless personal area networking (WPAN) technologies that provide connectivity up to tens of meters (e.g., ZigBee, Bluetooth Low Energy) and (ii) low-power wide-area networking (LPWAN) technologies that offer an extended range up to several kilometers but sacrifice in terms of throughput (e.g., LoRaWAN, Sigfox, DASH-7). As such, a gap still exists for a low-power communication technology that offers both extended range and higher throughput.

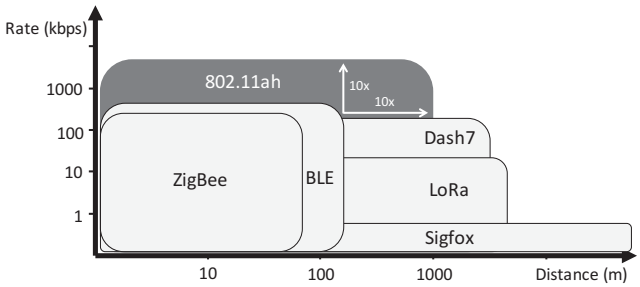


Figure 1: Position of IEEE 802.11ah compared to existing WPAN and LPWAN technologies, promising considerably extended range compared to WPAN and higher bitrate than LPWAN.

The new IEEE 802.11ah standard, marketed as Wi-Fi HaLow, fills this gap, combining the advantages of Wi-Fi and low-power communication technologies (cf. Figure 1). IEEE 802.11ah is a wireless communication PHY and MAC layer protocol that operates in the unlicensed sub-1Ghz frequency bands (e.g., 863–868 Mhz in Europe and 902–928 Mhz in North-America). It was designed to provide communications at a range of up to 1 kilometer while maintaining a data rate of 150 Kbps, offering a much greater coverage than existing WPAN and considerably higher throughput than LPWAN technologies. In the MAC layer, 802.11ah introduces mechanisms such as hierarchical organization, short MAC header, fast association, restricted access window (RAW), traffic indication map (TIM) segmentation and target wake time (TWT) to support densely deployed energy-constrained stations.

Even though the IEEE 802.11ah standard has only been officially released since January 2016, researchers have been investigating it already for a few years. Several recent studies investigate physical layer aspects of IEEE 802.11ah specifically and sub-1Ghz communications generally [5, 3, 2, 4]. Hazmi et al. [5] study link budget, achievable data rate and

WNS3, June 15 - 16, 2016, Seattle, WA, USA

ACM 978-1-4503-4216-2/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2915371.2915372

Table 1: 802.11ah MCSs for 1, 2 MHz, NSS=1, GI=8 μs.

MCS Index	Modulation	Coding rate	Data rate (Kbps)	
			1 Mhz	2 Mhz
0	BPSK	1/2	300	650
1	QPSK	1/2	600	1300
2	QPSK	3/4	900	1950
3	16-QAM	1/2	1200	2600
4	16-QAM	3/4	1800	3900
5	64-QAM	2/3	2400	5200
6	64-QAM	3/4	2700	5850
7	64-QAM	5/6	3000	6500
8	256-QAM	3/4	3600	7800
9	256-QAM	5/6	4000	Not valid
10	BPSK	1/2 with 2x repetition	150	Not valid

the influence of packet size in different scenarios. Moreover, a few studies of the IEEE 802.11ah physical layer have been performed using an implementation based on Software Defined Radios (SDRs) [3, 2, 4]. Current research on the MAC layer of 802.11ah mainly focuses on performance of the RAW mechanism [11, 7, 16, 12, 13, 9, 8]. Nearly all of this research is based on analytical modelling of the saturated network state. Such models do not accurately capture real network behavior and are hard to adapt to non-saturated network conditions. Only Raeesi et al. [12] performed actual simulations using OMNeT++.

To address this hiatus, we have implemented the 802.11ah standard in the ns-3 network simulator. Our implementation builds upon existing 802.11 implementations in ns-3 and extends them with a physical layer model for sub-1Ghz radio communications and a subset of the new MAC layer features of the standard. The implementation is modular, allowing it to be easily extended with additional 802.11ah-specific features. Moreover, it has been made available as open source for other researchers to experiment with[1].

Our simulation model has several benefits compared to the OMNeT++ model implemented by Raeesi et al. [12]. Specifically, their RAW implementation does not support grouping. Moreover, we have implemented the fast association and two-stage back-off mechanisms. Finally, their implementation, in contrast to ours, is not publicly available.

The contributions of this paper are twofold. First, we outline our implementation, its features, and its integration into ns-3. Second, we have conducted an in-depth simulation study to validate our implementation.

The remainder of this paper is structured as follows. Section 2 provides an overview of the most prominent 802.11ah features, both on the PHY and MAC layer. Details on the ns-3 implementation model are provided in Section 3. In Section 4, we validate both the physical and MAC layer of the implemented model. The planned future extensions to our implementation are detailed in Section 5. Finally, conclusions are discussed in Section 6.

2. OVERVIEW OF THE IEEE 802.11AH

IEEE 802.11ah operates at sub-1GHz, supporting long distance transmission, 8192 nodes connected to a single AP and high energy efficiency. These features make it an attractive standard for long-range IoT applications, such as sensor-based monitoring, smart meters and home automation. Throughout this section we highlight aspects of the standard that are important for our implementation. For a more detailed overview of the standard, the reader is referred to existing literature [6, 1].

2.1 PHY Layer

The IEEE 802.11ah PHY layer inherits characteristics from IEEE 802.11ac and adapts them to sub-1Ghz frequencies. Its channel bandwidths range from 1 to 16 MHz, with 1 and 2 MHz support being mandatory. Operating at low frequency and narrow bandwidth allows it to transmit at longer ranges (up to 1 km or more in theory) with considerably less power consumption than traditional Wi-Fi technologies, which use frequencies in the 2.4 and 5 GHz bands.

For different data rates and bandwidths, 802.11ah utilizes different sets of modulation and coding schemes (MCSs), number of spatial streams (NSS) and duration of the guard interval (GI). Coding schemes include binary conventional coding (BCC), which is mandatory, as well as the optional low density parity check (LDPC). Table 1 lists data rates and their MCSs when GI and NSS are 8 μs and 1 respectively for 1 and 2 MHz bandwidth.

2.2 MAC Layer

The MAC layer of IEEE 802.11ah consists of several novel features, such as fast association and authentication, RAW, TIM segmentation and TWT, aiming to address the requirements of dense IoT networks. Details of the those features are described as follows.

2.2.1 Fast Authentication and Association

When an AP is deployed or after a power outage, a large number of stations are simultaneously trying to associate, this process could take a long time due to collisions. Two more effective fast authentication and association control mechanisms (i.e., centralized and distributed), are proposed for IEEE 802.11ah. In centralized authentication, the AP sets a threshold in authentication control elements attached to a beacon. When a station is initialized, it generates a random value from the interval $[0, 1022]$ and send authentication/association requests to the AP if the random value is smaller than the threshold obtained from the received beacon, otherwise postpone authentication/association until the next beacon. The threshold should be adjusted dynamically by the AP to allow all stations to get associated eventually. Distributed authentication is based on the truncated binary exponential back-off, each beacon interval is divided into slots of equal duration, stations randomly select one slot to send their association request.

2.2.2 Restricted Access Window (RAW)

The RAW mechanism aims to reduce collisions and improve throughput when hundreds or even thousands of stations are simultaneously contending for channel access. It restricts the number of stations that can simultaneously access the channel by splitting them into groups and only allowing stations that belong to a certain group to access the channel at specific times. Figure 2 schematically depicts how RAW works. Specifically, the airtime is split into intervals,

[1]https://www.uantwerpen.be/en/rg/mosaic/projects/ieee-802-11ah/

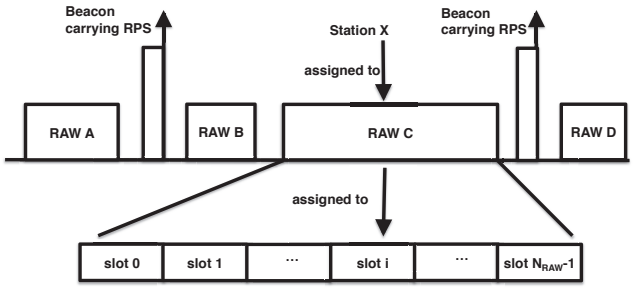


Figure 2: Schematic representation of the RAW mechanism.

each of which is assigned to one RAW group. Each interval is preceded by a beacon that carries a RAW parameter set (RPS) information element that specifies the stations that belong to the group, as well as the interval start time. Moreover, each RAW interval consists of one or more slots, over which the stations in the RAW group are split evenly. As such, the RPS also contains the number of slots, slot format and slot duration count sub-fields which jointly determine the RAW slot duration as follows:

$$D = 500\ \mu s + C \times 120\ \mu s \quad (1)$$

Where C represents slot duration count sub-field, which is either $y = 11$ or $y = 8$ bits long if the slot format sub-field is set to respectively 1 or 0. The number of slots field is $14 - y$ bits long.

Different from previous IEEE 802.11 technologies, each station uses two back-off states of enhanced distributed channel access (EDCA) to manage transmission inside and outside its assigned RAW slot respectively. The first back-off function state is used outside RAW and the second is used inside RAW slots. For the first back-off state, the station suspends its back-off at the start of each RAW and stores the back-off states, then restores them and resumes back-off at the end of the RAW. For the second back-off state, stations start back-off with initial back-off state inside their own RAW slot, and disregard the back-off state at the end of their RAW slot. As shown in Figure 3, station 1 is inside the RAW group and assigned to slot 1, while station 2 is not included in this RAW group. Therefore, station 1 uses the first back-off state outside its RAW slot period and the second back-off state inside its RAW slot, while station 2 only uses the first back-off state outside the RAW group period and goes into a sleep state inside the RAW group period.

2.2.3 Power Saving Mechanisms

In the current 802.11 standards, beacons trigger power saving (PS) station contention for the channel, which is the bottleneck of the whole power management framework since stations have to wake up to listen to every beacon. IEEE 802.11ah introduces the TIM segmentation mechanism to split information transmitted in the TIM into several segments and transmit them separately. In addition to using TIM beacons for station-level signaling, as shown in Figure 4, it also uses delivery traffic indication map (DTIM) beacons for TIM group-level signaling. The AP uses the DTIM beacon to broadcast to stations which TIM segments have pending data, stations only need wake up to listen to

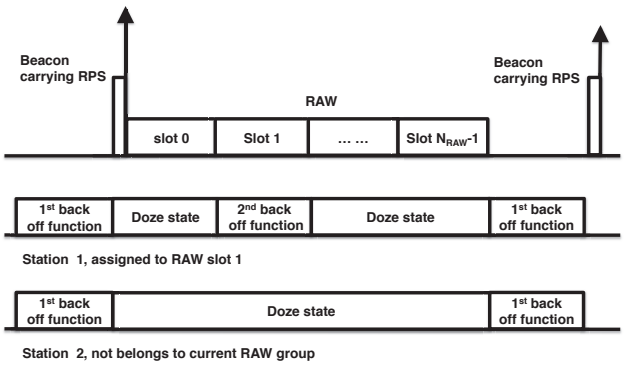


Figure 3: Back-off procedure for the IEEE 802.11ah RAW mechanism.

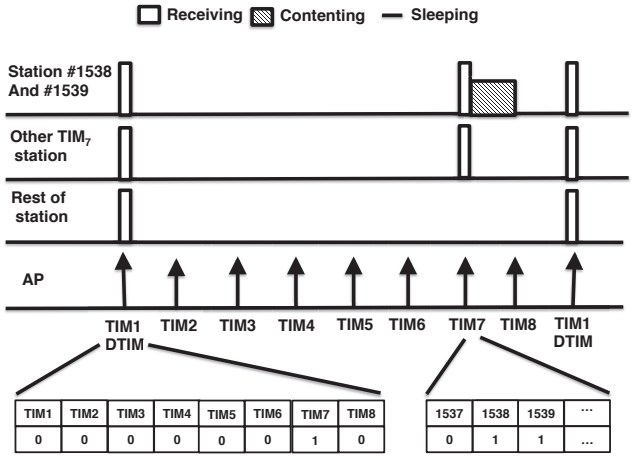


Figure 4: Example of the TIM segmentation mechanism (source: Adame et al. [1]).

their corresponding TIM beacon, thus they can maintain a longer power-saving state. Power consumption can be further reduced by TWT for stations transmitting data rarely. TWT stations can negotiate a time slot with the AP when they should wake up to exchange frames, therefore they can stay in a power-saving state for very long periods of time during their TWT intervals.

3. IMPLEMENTATION

Currently, ns-3 comes with support for several IEEE 802.11 standards, including 802.11a, 802.11b, 802.11g, 802.11n and since recently 802.11ac. As Figure 5 shows, the components of the ns-3 Wi-Fi PHY and MAC models consist of 4 main components:

- WifiChannel: An analytical approximation of the physical medium over which data is transmitted (i.e., the air in case of Wi-Fi), consisting of propagation loss and delay models.
- WifiPhy: The PHY part of the protocol, takes care of sending and receiving frames and determining loss due to interference.
- MacLow: Implements RTS/CTS/DATA/ACK trans-

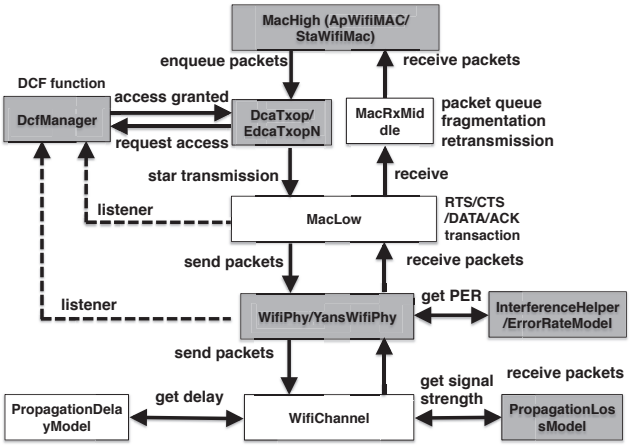


Figure 5: ns-3 Wi-Fi models and interactions; grey background denotes changed components in our implementation.

actions, the distributed coordination function (DCF) and enhanced distributed channel access (EDCA), packet queues, fragmentation, re-transmission and rate control.

- MacHigh: Implements management functions such as beacon generation, probing, association.

The remainder of this section outlines how we adapted the above models to support 802.11ah. Our implementation is based on ns-3 version 3.23.

3.1 PHY Layer

Our work in implementing 802.11ah PHY model focused on the components marked in Figure 5: InterferenceHelper, ErrorRateModel, WifiPhy, YansWifiPhy and PropagationLossModel.

3.1.1 WifiPhy and YansWifiPhy

In the WifiPhy class, the modulation and coding schemes MCS0 to MCS9 for channel bandwidths 1, 2, 4, 8 and 16 Mhz (cf. Table 1) are defined, while MCS10 is currently not supported by our implementation. Moreover, header and preamble formats of IEEE 802.11ah as well as the way of calculating sending/receiving duration of the preamble, header and payload are implemented. The `ConfigureStandard()`, `WifiModeToMcs()`, and `McsToWifiMode()` functions in YansWifiPhy are modified to support the 802.11ah configuration and the conversion between Wi-Fi mode and MCS.

3.1.2 InterferenceHelper and ErrorRateModel

Since 802.11ah defines new header formats, the `CalculatePlcpPayloadPer()` and `CalculatePlcpHeaderPer()` functions in the InterferenceHelper class are revised to calculate the length of received 802.11ah packets, which is required for the packet error rate calculation. Additionally, the NistErrorRate and YansErrorRate classes are modified to be capable of calculating the packet error rate of 802.11ah with support for the QAM256 modulation scheme. It is worth noting that YansErrorRate mode is known to be too optimistic, while NistErrorRatemodel better matches experimental results according to Pei and Henderson [10].

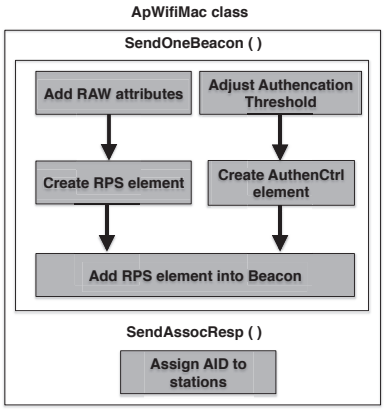


Figure 6: New funcationality in ApWifiMac class.

3.1.3 PropagationLossModel

This model determines the signal strength in the wireless medium based on the distance between sender and receiver. We implemented the indoor and outdoor propagation loss models for 802.11ah developed by Hazmi et al. [5]. They proposed two different outdoor models, one for macro deployments and one for pico or hotzone deployments. The macro deployment model assumes an antenna height of 15 meters above a rooftop, with the propagation loss (in dB) given as follows:

$$L\left(d\right) = 8 + 37.6\log_{10}\left(d\right) + 21\log_{10}\left(\frac{f}{900\text{MHz}}\right) \quad (2)$$

The pico deployment model is assumed to be at rooftop height, with the propagation loss (in dB) given as follows:

$$L(d) = 23.3 + 36.7\log_{10}(d) + 21\log_{10}\left(\frac{f}{900\text{MHz}}\right) \quad (3)$$

Finally, the indoor propagation loss model is the same as that of 802.11n, and consists of the free space loss (FSL) with a slope of 2 up to a breakpoint distance and a slope of 3.5 after this breakpoint:

$$L(d) = \begin{cases} 20\log_{10}\left(\frac{4d\pi f}{c}\right) & d \leq d_{BP} \\ 20\log_{10}\left(\frac{4d\pi f}{c}\right) + 35\log_{10}\left(\frac{d}{d_{BP}}\right) & d > d_{BP} \end{cases} \quad (4)$$

With d, f, c and d_{BP} the transmit-receive distance in meters, carrier frequency, speed of light and breakpoint distance respectively.

3.2 MAC Layer

Among the new features of IEEE 802.11ah listed in Section 2, our simulator currently supports fast association and RAW, the implementation mainly focuses on the MacHigh, DcaTxop, EdcaTxopN and DcfManager components. The implementation of the power-saving features (e.g., TIM segmentation and TWT) is planned as future work.

3.2.1 MacHigh

As shown in Figures 6 and 7, new functionality is added in both ApWifiMac and StaWifiMac in order to support fast association and RAW.

For the ApWifiMac class, RAW related attributes, as listed in Table 2, are added in order to allow user configuration.

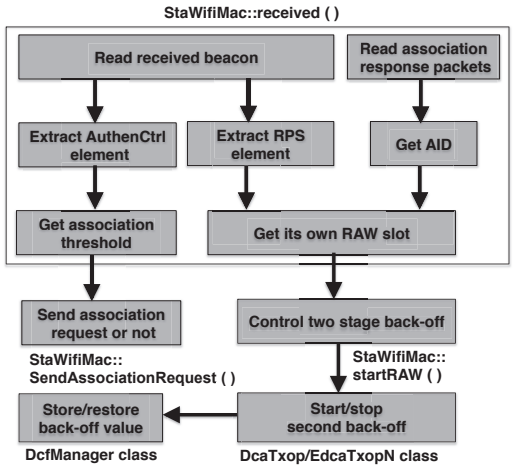


Figure 7: New funcationality in StaWifiMac class.

Table 2: 802.11ah attributes in ApWifiMac class.

Parameter	Description
NRawGroupStas	Stations per RAW group
NRawStations	Stations supporting RAW
SlotFormat	Slot format
SlotDurationCount	Slot duration count
NRawSlotNum	Slots per RAW group

The RPS and AuthentCtrl elements, which carry RAW group and association information, are defined in a new classes, and are generated and attached to beacons based on the values of the newly defined attributes. A mechanism for adjusting the association threshold is implemented as well, to allow changing the threshold dynamically. Besides that, the AID assignment scheme is added in the `SendAssocResp()` function to allows the AP to assign an AID to stations during the association exchange.

For the StaWifiMac class, the station receives an association threshold and RAW information from the RPS and AuthentCtrl elements carried by the received beacon, and gets an AID from the association response packet in the `received()` function. Based on the association threshold, the station determines to send an association request or not. RAW information and the AID jointly determine the RAW slot stations belong to and control the two stage back-off, making sure stations are allowed to access the channel during their own RAW slot.

3.2.2 DcaTxopN, EdcaTxopN and DcfManager

The two stage back-off mechanism was implemented in the DcaTxop, EdcaTxopN and DcfManager classes, supporting both Quality of Service (QoS) and non-QoS data transmissions. The start and termination of the two stage back-off is managed by the DcaTxop and EdcaTxopN classes by sending instructions to the DcfManager class, which is modified to be able to store and restore back-off related values. Based on the instructions from the StaWifiMac class, stations start to contend for the channel in the appropriate slot with the corresponding back-off state. Additionally, the WifiRemoteStationManager class is modified as well to store and restore the re-transmission counter of the two back-off states.

Table 3: Default physical layer parameters used in our experiments.

Parameter	Value
Frequency	900 Mhz
Transmission power	0 dBm
Transmission gain	0 dB
Reception gain	3 dB
Noise Figure	3 dB
Coding method	BCC
Propagation loss model	Outdoor, macro [5]
Error Rate Model	NistErrorRate, YansErrorRate

Table 4: Default MAC layer parameters used in our experiments.

Parameter	Value
CWmin	15
CWmax	1023
AIFSN	3
Traffic access categories	AC_BE
Payload size	256 bytes
MAC header type	legacy header
RTS/CTS	not enabled
Beacon Interval	0.1 s
Cross slot boundary	enabled
Station distribution	randomly
Wi-Fi mode	MCS8, 2 Mhz
Rate control algorithm	constant

4. EVALUATION

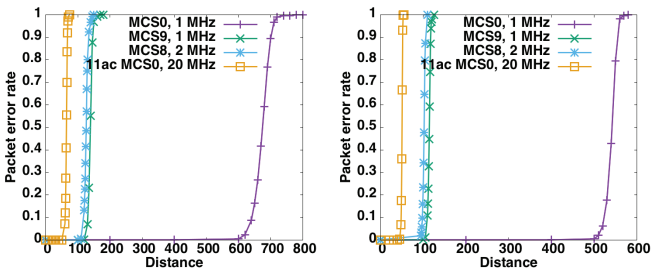
In this section we present and discuss our results obtained using the ns-3 implementation of IEEE 802.11ah. The simulation setup is described, and the implementation of the physical layer, fast association and RAW is validated.

4.1 Simulation Setup

We consider an IoT sensor-based monitoring scenario where a large number of battery-powered sensors send measurements to a back-end server (through the AP) at specific time intervals. Stations are deployed randomly within a 400 meter diameter around the AP. The default PHY and MAC layer parameters are shown in Tables 3 and 4 respectively. Note that these are the default parameter values, and some of them take different values in specific experiments (e.g., Wi-Fi mode), which is explicitly mentioned. Given the low-power nature of battery-powered sensors, transmission power is limited to 0 dBm. Like 802.11ac, 802.11ah uses forward error correction (FEC) schemes to improve transmission range. With the same physical parameters, we also test Wi-Fi mode MCS0 of 802.11ac, which has a data rate of 6.5 Mbps for bandwidth of 20 MHz.

4.2 Physical Layer

This section evaluates the physical layer packet error rate as a function of distance of different 802.11ah Wi-Fi modes for outdoor macro deployments, using the parameters defined in Table 3. Figure 8 shows the transmission range of different Wi-Fi modes. When YansErrorRateModel is used, IEEE 802.11ah can transmit over distances up to 640 m with a packet error rate below 10% and up to 670 m with an error rate below 50% using MCS0 with 1 Mhz bandwidth, which achieves data rates up to 300 kbps. Results also clearly show that using modes that provide higher data rate signifi-



(a) YansErrorRate Model (b) NistErrorRate Mode

Figure 8: Packet error rate as a function of uplink distance with different ErrorRate Model for various IEEE 802.11ah Wi-Fi modes and compared to IEEE 802.11ac.

cantly reduces the maximum transmission distance. For example, MCS9 with 1 Mhz bandwidth and MCS8 with 2 Mhz bandwidth are capable of achieving data rate up to 4 Mbps and 7.8 Mbps respectively. They allow error-free transmission up to 130 and 110 m respectively. For comparison, IEEE 802.11ac only has a range of up to 60 m with a data rate of 6.5 Mbps and an error rate of 10% (at an equal 0 dBm transmission power). As indicated in Section 3.1, the transmission range becomes shorter when NistErrorRateModel is applied, around 150, 20 and 10 m are lost for Wi-Fi modes MCS0 with 1 Mhz, MCS9 with 1 Mhz and MCS8 with 2 Mhz bandwidth. It should be noted that the 1 km range promised by 802.11ah could potentially be realized using the MCS10 mode together with LDPC coding, since it is capable of bridging higher distances. This mode's implementation is left for future work.

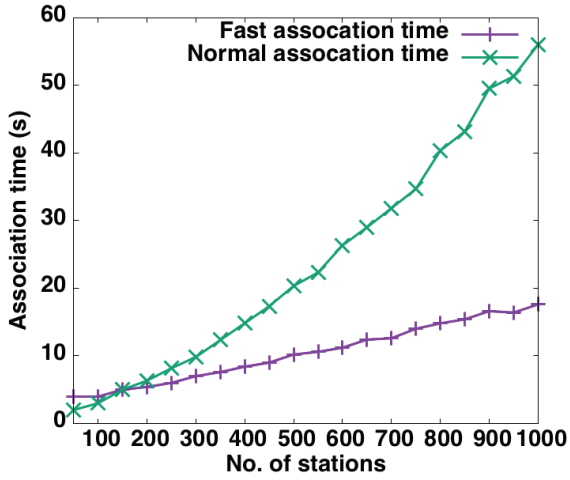
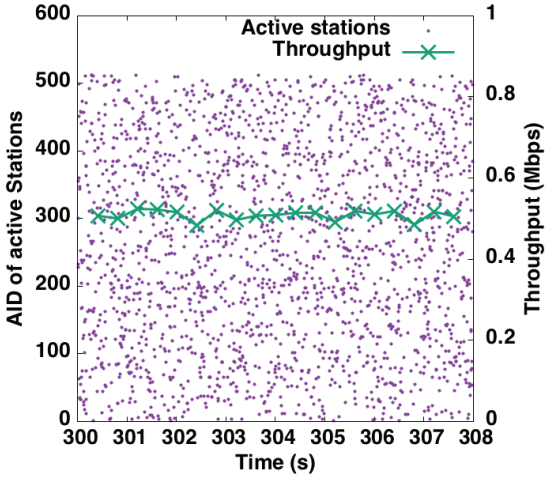


Figure 9: Association time comparison between normal association and fast association.

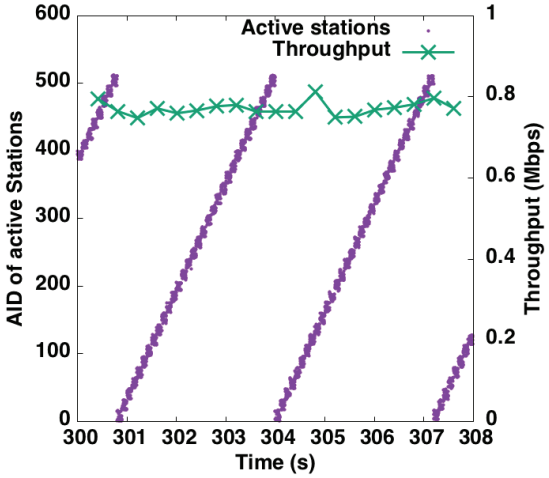
4.3 Fast Association Mechanism

In this section, we evaluate the fast association mechanism for a varying number of stations. A threshold adaptation mechanism proposed by Wang [15] is adopted in this simulation, in which the AP adjusts the association threshold dynamically based on its sending queue size. This results in increasing the threshold by 50 when the queue is smaller than 10, otherwise decreasing the threshold by 50. The result is depicted in Figure 9, which clearly reveals that fast association substantially decreases association time, especially for a large number of stations. It is quite straightforward since fewer stations are allowed to send association requests simultaneously, reducing the collision probability. Association performance could be further improved with more advanced threshold adaptation algorithms.

4.4 RAW Mechanism



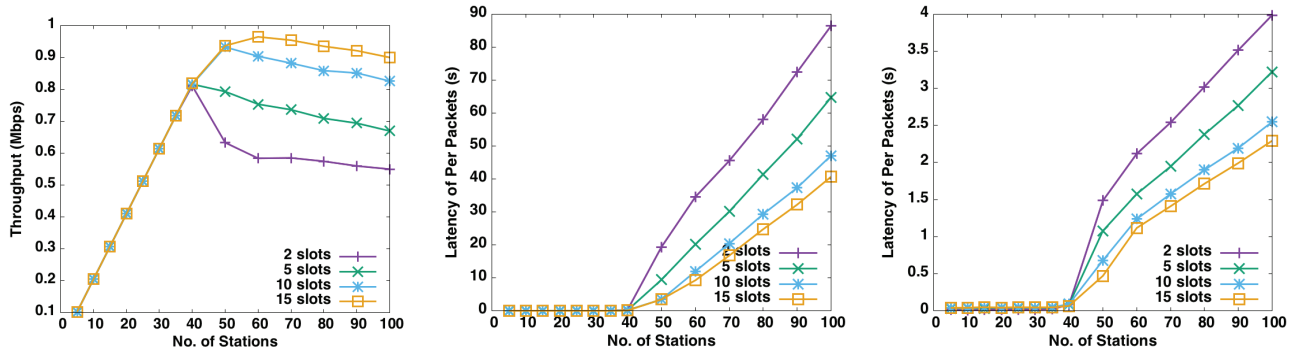
(a) Without RAW



(b) With 32 RAW groups

Figure 10: Number of active stations and instantaneous throughput for a 512 station network.

Figure 10 depicts the active stations (by AID) and throughput at each instant in time with and without the use of RAW. Figure 10a clearly shows that the channel is randomly utilized by stations all the time without the use of RAW. Due to the density of the network this results in many collisions and an average throughput of around 0.5 Mbps. In contrast,



(a) Throughput (b) Latency(infinite queue) (c) Latency(queue of 10-packets)

Figure 11: Throughput and latency for varying number of RAW slots.

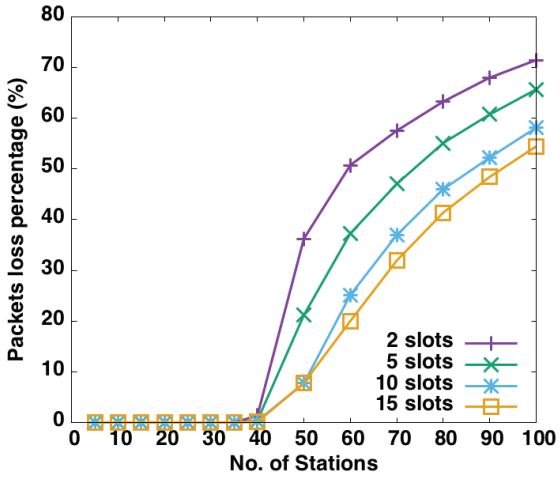


Figure 12: Packets loss with different number of RAW slots when the length of send queue is 10 packets.

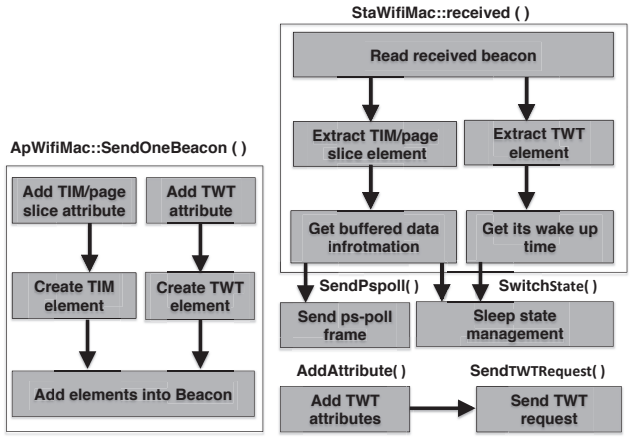


Figure 13: Future planned implementation of TIM segmentation and TWT.

Figure 10b clearly shows that the RAW mechanisms results in a more controlled channel access manner, only allowing contention among a limited number of stations. As a result, collision probability drops and the average throughput increases by 50% and becomes 0.75 Mbps.

Figure 11 aims to assess the influence of the number of RAW slots per group, as well as transmit buffer length on throughput and latency. The reader is referred to our previous work [14] for the impact of the number of RAW groups. There is a single RAW group, and each station sends one packet every 0.1 seconds to the AP with a random starting time. Due to the relatively small payload size, the maximum throughput that can be achieved using MCS8 with 2 Mhz bandwidth is around 1 Mbps. The network becomes congested when there are more than 50 stations transmitting. In the case of an infinite buffer, no packets are dropped, while packet loss for the case with a 10 packet buffer is shown in Figure 12.

Figure 11 clearly shows that the RAW slot parameter has no effect on performance for a small number of stations. However, higher throughput and reduced latency can be achieved for more stations with a larger number of slots. This is because splitting stations into RAW slots reduces contention, and in turn collision probability and retransmissions. However, it should be noted that very many slots could make the duration of slots too short to be enough for one packet to be successfully sent.

The use of an infinite transmit buffer obviously results in a considerable latency increase as congestion increases. On the other hand, a limited buffer space causes a considerable amount of packet loss. Finally, the fact that a higher number of RAW slots results in significantly less packet loss, again confirms that RAW reduces contention.

5. FUTURE IMPLEMENTATION PLANS

Our aim is to implement TIM segmentation and TWT in ns-3 in the near future. As shown in Figure 13, the implementation will focus on ApWifiMac and StaWifiMac. For the ApWifiMac class, TIM and TWT elements will be defined and attached to beacons according to the related attributes. For the StaWifiMac class, functionality such as reading TIM and TWT elements from received beacons,

sending ps-poll frames and managing sleep state indicated by the TIM and TWT elements, is going to be implemented. Since the TWT setup can be launched by both stations and the AP, TWT related attributes and sending TWT requests will also be implemented in the StaWfiMac class. Moreover, support for MCS10 Wi-Fi mode will be added by implementing LDPC coding with 2x repetition. On the longer term, we aim to support other features of 802.11ah, such as Relays and hierarchical RAW. Finally, work is in progress to include 802.11ah support in a future ns-3 official release.

6. CONCLUSION

This paper presents an implementation of the IEEE 802.11ah physical and MAC layer protocol for the ns-3 network simulator. The implementation and its integration into ns-3 are described in detail. Moreover, the implementation was evaluated, validated through simulation based on the implemented model. The evaluation confirmed a correct behavior of the fast association and RAW MAC-layer mechanisms. Moreover, 802.11ah is shown to be a promising wireless technology for densely deployed IoT applications, due to its ability to limit contention using RAW groups and slots.

7. REFERENCES

[1] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver. IEEE 802.11ah: the WiFi approach for M2M communications. *IEEE Wireless Communications*, 21(6):144–152, 2014.

[2] S. Aust and R. V. Prasad. Advances in wireless M2M and IoT: Rapid SDR-prototyping of IEEE 802.11ah. In *IEEE Local Computer Networks Conference*, 2014.

[3] S. Aust, R. V. Prasad, and I. G. M. M. Niemegeers. Performance study of MIMO-OFDM platform in narrow-band sub-1 GHz wireless LANs. In *11th International Symposium on Modeling & Optimization in Mobile, Ad Hoc & Wireless Networks (WiOpt)*, pages 89–94. IEEE, 2013.

[4] R. A. Casas, V. Papaparaskeva, R. Kumar, P. Kaul, and S. Hijazi. An IEEE 802.11ah programmable modem. In *IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2015.

[5] A. Hazmi, J. Rinne, and M. Valkama. Feasibility study of IEEE 802.11ah radio technology for IoT and M2M use cases. In *2012 IEEE Globecom Workshops*, pages 1687–1692. IEEE, 2012.

[6] E. Khorov, A. Lyakhov, A. Krotov, and A. Guschin. A survey on IEEE 802.11ah: An enabling networking technology for smart cities. *Computer Communications*, 58:53–69, 2015.

[7] K. Ogawa, M. Morikura, K. Yamamoto, and T. Sugihara. IEEE 802.11ah based M2M networks employing virtual grouping and power saving methods. *IEICE Transactions on Communications*, E96-B(12):2976–2985, 2013.

[8] C. W. Park, D. Hwang, and T.-J. Lee. Enhancement of IEEE 802.11ah MAC for M2M communications. *IEEE Communications Letters*, 18(7):1151–1154, 2014.

[9] M. Park. IEEE 802.11ah: Energy efficient MAC protocols for long range wireless LAN. In *2014 IEEE International Conference on Communications (ICC)*, pages 2388–2393. IEEE, 2014.

[10] G. Pei and T. R. Henderson. Validation of ofdm error rate model in ns-3. *Boeing Research Technology*, pages 1–15, 2010.

[11] M. Qutab-ud din, A. Hazmi, B. Badihi, A. Larmo, J. Torsner, and M. Valkama. Performance analysis of IoT-enabling IEEE 802.11ah technology and its RAW mechanism with non-cross slot boundary holding schemes. In *IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2015.

[12] O. Raeesi, J. Pirskanen, A. Hazmi, T. Levanen, and M. Valkama. Performance evaluation of IEEE 802.11ah and its restricted access window mechanism. In *IEEE International Conference on Communications Workshops (ICC)*, pages 460–466.

[13] O. Raeesi, J. Pirskanen, A. Hazmi, J. Talvitie, and M. Valkama. Performance enhancement and evaluation of IEEE 802.11ah multi-access point network using restricted access window mechanism. In *IEEE International Conference on Distributed Computing in Sensor Systems*, pages 287–293, 2014.

[14] L. Tian, J. Famaey, and S. Latré. Evaluation of the ieee 802.11ah restricted access window mechanism for dense iot networks. In *IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Accepted.

[15] H. Wang. *Supporting Authentication/Association for Large Number of Stations, 2012.*, 2012. https://mentor.ieee.org/802.11/dcn/12/11-12-0112-04-00ah-supporting-of-the-authentication-association-for-large-number-of-stations.pptx.

[16] L. Zheng, L. Cai, J. Pan, and M. Ni. Performance analysis of grouping strategy for dense IEEE 802.11 networks. In *2013 IEEE Global Communications Conference (Globecom)*, pages 219–224, 2013.

Implementation and Evaluation of a WLAN IEEE 802.11ad Model in ns-3

Hany Assasa
IMDEA Networks Institute and
Universidad Carlos III de Madrid, Spain
hany.assasa@imdea.org

Joerg Widmer
IMDEA Networks Institute
Madrid, Spain
joerg.widmer@imdea.org

ABSTRACT

The IEEE 802.11ad amendment to the 802.11 standard for multi-gigabit communication at 60 GHz was published several years ago, but to date, no precise simulation model for networking in this band is available. In this paper, we present a model for IEEE 802.11ad implemented in the network simulator ns-3. We model new techniques that are essential for IEEE 802.11ad operation such as beamforming training and steering, relay support, and fast session transfer. We then evaluate by simulation the performance of IEEE 802.11ad as well as the gains obtained through the aforementioned techniques. The code for our simulation model is publicly available.

CCS Concepts

•**Networks** → **Network simulations; Wireless local area networks;**

Keywords

Millimeter Wave, IEEE 802.11ad, 60 GHz, ns-3

1. INTRODUCTION

With the proliferation of mobile devices with data hungry applications, existing mobile networks and wireless local area networks (WLAN) technologies are becoming increasingly congested and overloaded. As a result, mobile network operators and telecommunications equipment vendors are considering leveraging the underutilized radio spectrum available between 30 GHz and 300 GHz, the so called the millimeter wave (mmWave) band, for next generation wireless networks. Wireless communication in this band is highly appealing since it provides extremly high capacity and thus allows for a several-fold increase in data rates and lower latencies. However, transmission in this band has specific signal propagation characteristics compared to existing technologies working in lower bands and thus requires major design changes for both medium access control (MAC) and physical (PHY) layers. Both the Wireless Gigabit Alliance (WiGig) and the Wi-Fi Alliance took the initiative to leverage this wide spectrum and provide multi-gigabit per second communication in the 60 GHz unlicensed band. They introduced the WLAN IEEE 802.11ad amendment [2, 9] which provides very high throughput of up to 7 Gbps for short range communication for local area networks. This allows for a range of new high-rate applications, such as wireless docking stations, wireless storage, and instant file synchronization. Compared to IEEE 802.11ac [3] which is capable of supporting multi-gigabit throughput by employing high modulation and coding schemes (MCSs) and advanced physical layer technologies such as multi-user-multiple-input and multiple-output (MIMO), IEEE 802.11ad achieves multi-gigabit throughput by utilizing only the wide channels of 2.16 GHz available at the 60 GHz band.

Experimental evaluation of networking in this band is extremely costly and available hardware has very limited capabilities. Current research studies deduce network performance from individual 60 GHz links [7, 4, 8] but cannot evaluate the behavior of an entire network. In such cases, resorting to network simulation is a very useful alternative which abstracts implementation details while providing a good grade of realism. However, there are no publicly available simulation tools supporting IEEE 802.11ad in the mmWave band. For these reasons, we provide in this paper a concrete model for simulating IEEE 802.11ad with its novel techniques such as channel access periods, beamforming training, relay operation, and fast session transfer.

The paper structure is as follows. In Section 2 we provide background on IEEE 802.11ad accompanied by a survey of the existing simulation models. Section 3 presents our IEEE 802.11ad implementation in ns-3 and Section 4 presents the evaluation results for the proposed model for different scenarios. Finally, Section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

In the following subsections, we provide background on the WLAN IEEE 802.11ad amendment and survey existing work on network level simulators for mmWave technologies.

2.1 Background on IEEE 802.11ad

Wireless communication in the 60 GHz band has different characteristics compared to IEEE 802.11 devices operating in the 2.4 GHz and 5 GHz bands. In the following paragraphs, we provide a brief description of the major design changes for both MAC and PHY layers in IEEE 802.11ad and the intuition behind them.

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915377

2.1.1 802.11ad Physical Layer

Emerging applications using the IEEE 802.11ad multi-gigabit capability have different constraints and requirements in terms of power consumption, data rates, processing capabilities and antenna design complexity. For these reasons, IEEE 802.11ad introduces four different types of PHY layers to cope with these requirements. Each PHY layer supports a set of specific MCSs.

- **Control PHY**: This PHY layer (MCS0) is dedicated to low Signal-to-Noise Ratio (SNR) operation with low throughput communication (27.5 Mbps). It is mainly used during the Beamforming Training (BF) phase.
- **OFDM PHY**: This PHY layer (MCS 13-24) provides the highest data rates of up to 6.76 Gbps. It adopts Orthogonal Frequency Division Multiplexing (OFDM) technology which is very efficient in multi-path environments. However, its implementation is complex and therefore it targets devices with less stringent power and design constraints, such as docking stations and wireless streaming devices.
- **Single Carrier (SC) PHY**: Power limited and low complexity devices adopt this physical layer which provides a good trade-off between average throughput and energy efficiency compared to the OFDM PHY. Mobile phones and tablet devices will most likely adopt this PHY layer. SC PHY defines MCS 1-12, of which MCS 1-4 are mandatory modes to be implemented in all devices for interoperability.
- **Low Power (LP)-SC PHY**: This PHY layer with MCS 25-31 is similar to the SC PHY layer, but allows for further power reduction by using low-density parity check (LDPC) codes instead of Reed-Solomon codes.

Figure 1 depicts the IEEE 802.11ad frame structure. The frame starts with the typical IEEE 802.11 fields such as short training field (STF) and channel estimation field (CEF) which are used for detecting the type of the PHY layer. These fields are followed by the PHY header which includes information such as payload length in bytes and index of the MCS used in the payload part. This field together with the MAC header and the MAC payload are protected by a Cyclic redundancy check (CRC). Finally, IEEE 802.11ad appends optionally two fields named automatic gain control (AGC) and training (TRN). These new fields are used during the BF phase which we describe in section 2.1.3.

2.1.2 DMG Channel Access

IEEE 802.11ad organizes the access to the medium in so-called Beacon Intervals (BIs). Each BI is further subdivided into different access periods. An access period has different access rules and provides certain functionalities to nearby directional multi-gigabit (DMG) stations (STAs). Figure 2 illustrates a typical BI consisting of Beacon Header Interval (BHI) and Data Transmission Interval (DTI). The BHI compromises the following three sub-intervals:

- **Beacon Transmission Interval (BTI)**: In this sub-interval, multiple DMG Beacon frames are transmitted across different sectors by the DMG personal basic service set control point (PCP)/access point (AP) to announce the network and provide transmit sector training towards nearby stations. DMG Beacons are transmitted using MCS 0 to reach large distances.
- **Association Beamforming Training (A-BFT)**: The A-BFT is used mainly by DMG STAs to train their transmit antenna sectors towards the DMG PCP/AP in a contention based manner.
- **Announcement Transmission Interval (ATI)**: This sub-interval is used mainly for management frame exchange between the PCP/AP and beam-trained STAs. Since communication takes place with beam-trained stations, stations can use high MCSs during the ATI for more efficient communication.

In the DTI period, DMG STAs exchange data frames either in the contention-based access period (CBAP) or the scheduled service period (SP). During the CBAP, DMG STAs contend for the channel access using the IEEE 802.11 Enhanced Distributed Coordination Function (EDCF), whereas in SP, DMG STAs access the channel in a contention-free manner where the channel is reserved for communication between two dedicated DMG STAs.

2.1.3 Beamforming Training Mechanism

Propagation conditions at 60 GHz band are worse compared to the lower bands due to oxygen absorption [10], high attenuation, weak signal reflectivity, and quasi-optical propagation behavior [11]. For these reasons, IEEE 802.11ad provides a mechanism to establish a directional link through a beamforming training process to compensate for signal quality degradation. In this process, stations focus their energy towards the intended receivers only, which increases antenna gain and may result in reduced interference, allowing for high spatial reuse. The beamforming training process in IEEE 802.11ad is divided into the following two phases:

- **Sector Level Sweep (SLS) Phase**: In this phase, a DMG STA selects a coarse grain antenna sector for the initial communication. The phase can be used in two ways: 1) as transmit sector sweep (TXSS) where a DMG STA tries to select the best transmit antenna sector towards a particular DMG STA by sending Sector Sweep (SSW) frames via each of its antenna sectors or 2) as a receive sector sweep (RXSS), where a DMG STA trains its receive antenna sector by requesting a peer DMG STA to transmit SSW frames using a fixed antenna pattern while the former is sweeping across its receive antenna sectors.
- **Beam Refinement Protocol (BRP) Phase**: IEEE 802.11ad defines multiple optional mechanisms to refine the sectors obtained in the SLS phase. The most important mechanism is the beam refinement mechanism, which is an iterative process where two DMG STAs exchange a special BRP packet ending with either transmit training (TRN-T) or receive training (TRN-R) fields. Additionally, the amendment defines a Beam Tracking (BT) option to keep a track of signal quality during an ongoing data transmission by adding the previous TRN fields to the PHY frames.

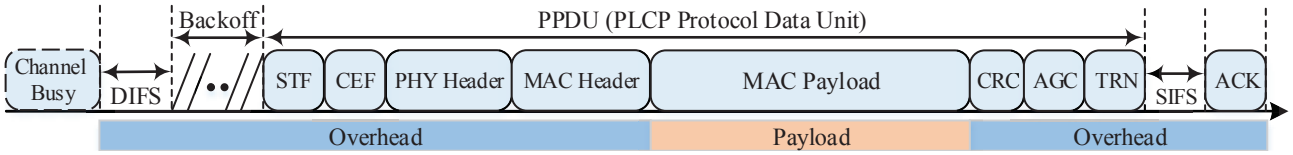


Figure 1: IEEE 802.11ad frame structure.

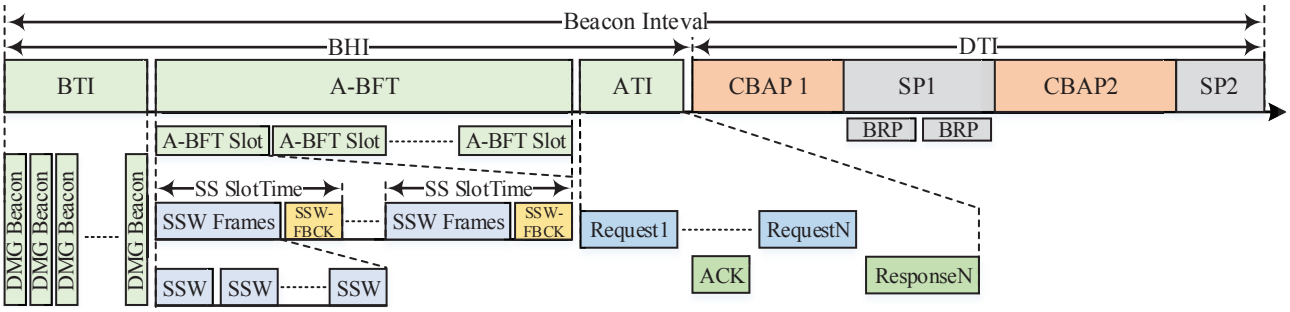


Figure 2: IEEE 802.11ad beacon interval with different access periods.

2.1.4 *Fast Session Transfer Technique*

Since communication in the 60 GHz band is limited in range and suffers high penetration loss in case of obstacles, IEEE 802.11ad included a fast session transfer (FST) technique. With this, an IEEE 802.11 capable device can change seamlessly its operational band from 60 GHz to 2.4/5 GHz. As a result, a device can extend its coverage area and maintain its current sessions. As an example of this technique, a user may stream an Ultra High Definition (UHD) video on his device from a wireless docking station over the 60 GHz band when he is in the proximity of the docking station. As the user starts to move a way from the docking station, signal quality starts to degragade so the docking station decides to transfer the session to a lower band but continues video streaming using lower video encoding techniques.

2.1.5 *DMG Relay Operation*

IEEE 802.11ad also introduces a relay mode. In this mode, two DMG STAs named source Relay Endpoint DMG STA (REDS) and destination REDS can communicate with each other with the assistance of a Relay DMG STA (RDS) which results in coverage area extension, improved link resilience against interruptions, and persistent multi-gigabit throughput. IEEE 802.11ad defines the following two types of relay operation modes:

- **Link Switching Type**: In this type, the source REDS maintains two links to the destination REDS: a direct link and a relay link through RDS. If the direct link is disrupted, the source REDS switches its transmission to the relay link. Communication over the disrupted link can resume once the direct link is recovered. Under this type, the RDS can operate either in full-duplex amplify-and-forward (FD-AF) mode or in half-duplex decode-and-forward (HD-DF) mode. In FD-AF mode, RDS amplifies the received frames and forwards them directly to the destination DMG STA. For this reason, the RDS should include at least two RF chains for sending and receiving frames at the same time. In contrast, for the HD-DF mode the RDS receives multiple frames from the source REDS in one SP and forwards them to the destination REDS in the following SP.

- **Link Cooperating Type**: Contrary to the previous mode, in this mode the source REDS utilizes both direct link and relay link simultaneously to improve received signal quality at the destination REDS. Operating in this mode requires the source DMG to be aware of propagation delays over each link.

2.2 Existing Work

Although wireless networking in the mmWave band is a hot topic, limited work has been done to provide system level simulators, particularity for IEEE 802.11ad. For example, authors in [7] propose a 5G module in ns-3 based on the LTE protocol stack. The module provides a channel model based on extensive channel measurements in the 28 GHz band. However, it does not employ any algorithms for establishing directional links nor steering antennas arrays. Authors in [4] utilize a IEEE 802.11ad PHY layer to establish multi-gigabit links in data centers using ns-3. In their implementation, they use data rates provided for both SC and OFDM PHY layers in the amendment. Additionally, since the topology of the data center is stable and known, they steer their antennas geometrically i.e. they do not simulate any of the beamforming procedures established in the standard. Finally, authors in [6] provide an architecture for simulating IEEE 802.11ad in ns-3 with a general description on modeling various BF procedures provided in the amendment. However, the implementation is not publicly available and in the validation section the author does not take into account the overhead imposed by different access periods in the BI. All of the previous works simplify the implementation and do not model essential techniques for MAC and PHY operation in the mmWave band.

3. IMPLEMENTATION

In the following section, we provide an overview of the IEEE 802.11 model in ns-3 and how we augmented it to adhere to the IEEE 802.11ad amendment. The model implementation is available on GitHub [1].

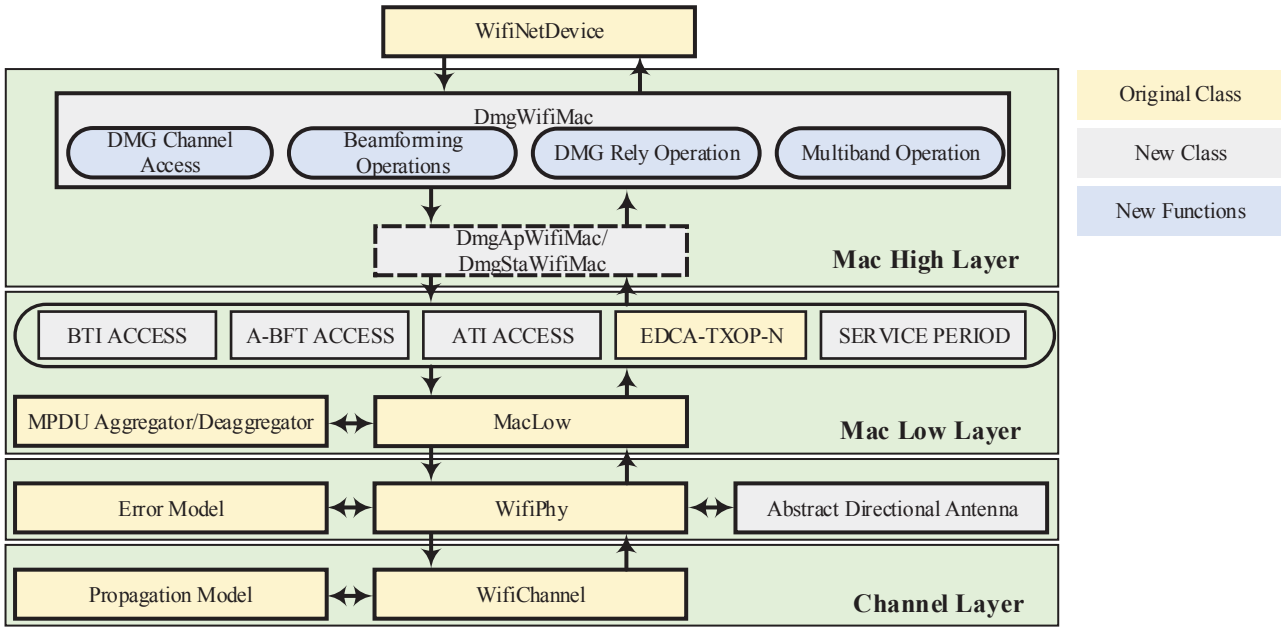


Figure 3: Our implemented IEEE 802.11ad architecture in ns-3.

3.1 IEEE 802.11 Model in ns-3

The current Wifi model in ns-3 supports different IEEE 802.11 specifications such as a/b/e/g/n/ac with an accurate implementation of the MAC layer. The model can be divided into the following four layers:

- **MAC High Layer**: It provides some Mac layer management entity (MLME) functionalities depending on the underlying network it supports, such as Infrastructure basic service set (BSS) or Independent BSS.
- **MAC Low Layer**: This layer takes care of Ready to Send (RTS)/Clear to Send (CTS)/DATA/Normal Acknowledgment (ACK)/BlockACK transmission using the distributed coordination function (DCF) and Enhanced DCF channel access (EDCA) functions. Additionally, it is provides both MAC service data unit (MSDU)/MAC protocol data unit (MPDU) aggregation and deaggregation capabilities.
- **Physical Layer**: It is a simplified model of the real Wifi PHY layer. This layer handles packet transmission and reception over the underlying channel. It calculates interference among different STAs and provides some probabilistic error model for packet reception.
- **Channel Layer**: This layer interconnects different PHY layers of different wireless STAs. Additionally, it simulates and models propagation effects that wireless signals encounter in real environments.

The IEEE 802.11 model in ns-3 is well suited for wireless technologies that use a carrier sense multiple access with collision avoidance (CSMA/CA) scheme with omni-directional transmission and reception. However, IEEE 802.11ad has characteristics that require some major changes to this model. Figure 3 shows the existing ns-3 IEEE 802.11 architecture together with the new blocks for the mechanisms introduced in IEEE 802.11ad. An abstract *DmgWifiMac* class provides common capabilities and techniques for DMG operation. From this class, we derive two classes to represent different BSS types. The first class *DmgStaWifiMac* implements procedures specific to DMG STA such as TXSS in A-BFT, responding to request frames in the ATI period, and the association state machine. The second class, *DmgApWifiMac*, represents the DMG AP and provides DMG Beaconing, BRP Setup Subphase, and BRP transaction initiation.

The following subsections provide an in-depth description of implementation and design assumptions for each block. Since ns-3 provides packet-level granularity, it was important to provide an accurate implementation of the newly introduced MAC frames and Wifi Information Elements to support various procedures defined in IEEE 802.11ad. Furthermore, representing the actual frame structure facilitates packet flow analysis using any network protocol analyzer that supports the IEEE 802.11ad extension.

3.2 DMG PHY Layer

ns-3 provides a simple PHY layer for the operation of IEEE 802.11. In this layer, the reception of the Physical Protocol Data Unit (PPDU) frame is modeled as simulation delay corresponding to the transmission time of this frame plus propagation delay. To model the multi-gigabit throughput of IEEE 802.11ad, we provide all the mathematical equations required for the calculation of PHY frame transmission time including preamble, header and payload using either control, SC, or OFDM PHYs.

3.3 DMG Access Periods

The *DmgApWifiMac* class organizes medium access by initiating BI through transmission of DMG Beacons across all its antenna sectors. The remaining time for each access period is announced in the duration field of the MAC header. This allows DMG STAs to synchronize their clocks with the DMG AP clock. During BTI, the DMG AP ensures the medium is free before it starts DMG beacon transmission.

For this reason, the value of the duration field is calculated once the DMG AP is granted access to the channel. A DMG STA that receives at least one DMG beacon from the DMG AP schedules an event to start the A-BFT access period at the end of the current BTI. The DMG APs divides the A-BFT into slots, where the duration of each slot is calculated based on the number of SSW frames to be transmitted. DMG STAs choose one of these slots randomly using a uniform distribution. If two DMG STAs select the same slot, they will collide and do not receive an SSW-Feedback (FBCK) frame within a pre-determined period of time. These two DMG STAs then have to select a new slot while ensuring that they do not exceed the duration of the A-BFT. This period is followed by the ATI access period, where the DMG AP initiates management frame transmission. Currently, we use this period to perform the BRP setup phase and exchange BRP transactions. Any packet that arrives during the previous access periods is queued for transmission until the beginning of the DTI.

3.4 Directional Antenna Pattern

Unlike the previous IEEE 802.11 specifications which are able to exploit omni-directional communication, IEEE 802.11ad requires a directional communication link towards its intended receiver, and thus a directional antenna pattern model. We provide a generic directional antenna model named *AbstractDirectionalAntenna* which divides the 2D plane into a user defined number of virtual sectors with equal apertures and coverage range. Concrete antenna models are inherited from this base model. In this derived model it is possible to define the attributes of the radiation pattern such as maximum gain, side lobe gain, and the gain based on the selected sector and the geometric angle between the transmitter and the receiver. In our implementation, we use the antenna model provided in [5] for evaluating IEEE 802.11ad in a conference room. In this antenna model, the authors provide a simple mathematical model to characterize a directional antenna with averaged side lobe. For frame transmission, a DMG STA should be using one of the predefined antenna transmit sectors. For reception, the antenna can be either in omni receive mode or directional receive mode depending on the current access period.

3.5 DMG Beamforming Operation

We provide a generic implementation for both SLS and BRP phases in the *DmgWifiMac* class. The implementation can be used either as part of the initial BT between DMG AP and DMG STAs or as a scheduled SP between two DMG STAs. The *DmgWifiMac* class has two data structures: one for storing and mapping antenna sector configurations for each received frame from a peer DMG STA with its corresponding SNR and another data structure for storing the best transmit and receive antenna sectors towards a particular station. The latter is updated at the end of each BF operation. In the current implementation, all decisions regarding the best transmit and receive antenna sectors are based on SNR measurements. The *MacLow* class uses the second data structure to select the antenna sector based on the receiver address in the MAC header. In the current implementation, we assume DMG STAs perform TXSS in the A-BFT. In addition, the BRP phase will be utilized to train antenna receive sectors for all DMG STAs instead of refining the selected antenna transmit sector. All BF frames are transmitted using MCS 0.

To model TRN field transmission in ns-3, we modify the current Physical Layer Convergence Protocol (PLCP) transmission model which handles PHY preamble, header and payload transmission and reception only. For example, a MAC frame that requires TRN fields to be appended to its end pass this information in the TxVector together with the length and type of the TRN fields to be appended (either TRN-T or TRN-R). Since each TRN field corresponds to a unique antenna sector pattern, we schedule the transmission of each TRN field separately to allow DMG to change its active sector. At the end of each TRN field transmission, the receiver calculates the received SNR value for this particular field and reports it to the *DmgWifiMac*. Once all TRN fields are received, the *DmgWifiMac* determines the best antenna sector.

3.6 Fast Session Transfer

IEEE 802.11ad supports multi-band operation for fast session transfer (FST). FST operation can be either in transparent or non-transparent mode. In transparent mode, all MAC sub-layers in the STA expose a single MAC-Service Access Point (SAP) to the upper layers, i.e., a single MAC address. In non-transparent mode, each MAC sub-layer exposes its own MAC-SAP to the higher layers which adds more complexity. In our implementation, we use the transparent mode where we design a new *NetDevice* named *MultiBandNetDevice*. This new *NetDevice* encapsulates different IEEE 802.11 technologies as depicted in Figure 4. For each technology, a user defines a *WifiMac*, *WifiPhy*, *WifiRemoteStation* and *WifiChannel* object. One technology should be active at any point for any pair of devices. A STA that supports multi-band operation should announce this in its Beacon, Association Request, Association Response, Probe Request, Probe Response, and DMG Beacon a MultiBand Information Element.

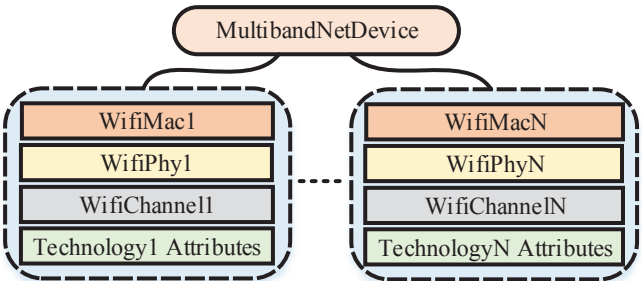


Figure 4: MultiBandNetDevice implementation.

Figure 5 illustrates various states a STA goes through to establish a unique fast session transfer session (FSTS) ID with a peer STA. At the beginning, each STA is in the **INITIAL_STATE** where they communicate in the old band/channel. A station that wishes to set-up a FSTS is called FST Initiator and the peer station is FST Responder. To proceed to the **SETUP_COMPLETION_STATE** and obtain a unique FSTS ID, both Initiator and Responder have to exchange FST Setup Request/Response frames successfully. In this new state, STAs keep communicating in the old band/channel. However, depending on the value of the link loss timeout (LLT) field in the Session Transfer Information Element, both STAs shall either transfer their current session to the new band/channel immediately if the value of

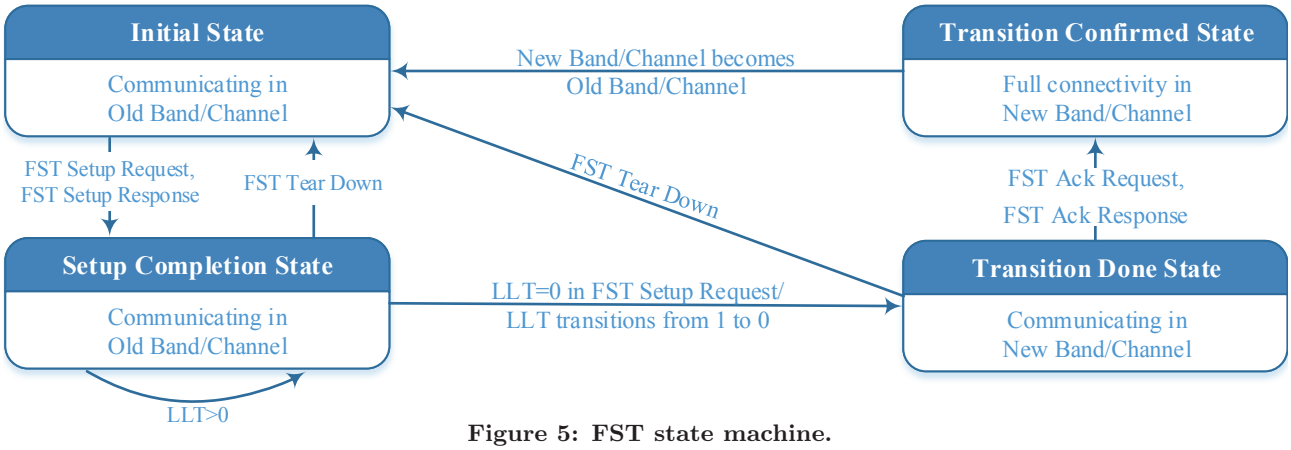


Figure 5: FST state machine.

LLT is equal to zero, or they shall start a Link Loss countdown equal to LLT $* 32\mu$s if LLT > 0. The LLT defines the amount of time that has to elapse since the initiating STA received an MPDU frame from the responding STA until the initiating STA should perform FST. Once the value of LLT reaches zero, both Initiator and Responder move to the **TRANSITION_DONE_STATE** and start communicating in the new band/channel. If the two STAs exchange normal MPDU frames or FST ACK Request/Response in the new band/channel successfully, the two STAs move to the **TRANSITION_CONFIRMED_STATE**, otherwise the two STAs move to the **INITIAL_STATE** and resume communication in the old band/channel.

3.7 DMG Relay Operation

We implement link switching type relay operating in FD-AF mode. Figure 6 summarizes different procedures to establish a relay link with a destination Relay Endpoint DMG STA (REDS) in a DMG BSS. A STA should acquire the DMG capabilities of the DMG STA it wishes to establish a relay operation with before it initiates any relay setup operation. This is done by sending an Information Request frame to the DMG AP after the DMG STA completes its association with the DMG AP.

- **RDS Discovery Procedure**: In this phase, a source REDS searches for candidates Relay DMG STAs (RDSs) in the DMG BSS. The DMG AP informs both source REDS and destination REDS about the available REDS in the network with their DMG capabilities.

- **RDS Selection Procedure**: At this point, the DMG AP schedules several SPs for BF training between all the available RDSs together with source REDS and destination REDS consecutively. After that, the source REDS request for channel measurements with the candidates RDSs. Later on, the DMG AP schedules a SP for BF between source REDS and destination REDS. After finishing the BF, the source REDS requests destination REDS to send channel measurements with the available RDSs. As a result, the source REDS will be aware of all channel states in the network. Based on this information, the source REDS selects the best RDS for relaying. In our implementation, we select the RDS which receives frames from both source REDS and destination REDS with the highest SNR.

- **Relay Link Setup (RLS) Procedure**: In this phase, the source REDS decides to forward its current transmission through the selected REDS in the previous phase. Thus, it sends an RLS Request frame to the selected RDS. The selected RDS in return forwards this frame to the destination REDS. At this point, the destination REDS replies back to the selected RDS with an RLS Response with a status equal to Success if it accepts to communicate through the relay link. The selected REDS forwards this frame to the source REDS with a status equal to Success if it accepts to act as relay. If both destination REDS and RDS accept to switch the link, the source REDS sends an announcement frame to the DMG AP regarding the newly established relay link in the network.

- **Relay Teardown Procedure**: If the source REDS decides to terminate its relay link through the selected RDS, it shall transmit an RLS Tear Down frame to the selected RDS, destination REDS and DMG AP.

4. MODEL EVALUATION

In this section we provide some evaluation results for our new IEEE 802.11ad model. In all the experiments, we assume all DMG STAs and DMG AP have one antenna array with 8 sectors. We use a Friis propagation loss model to calculate received signal strength (RSS) and UDP as transport protocol. All STAs support both MSDU and MPDU aggregation, and data transmission is done in CBAP mode.

4.1 Evaluating 802.11ad Beamforming Overhead and Achievable Throughput

In this experiment, we calculate the amount of time it takes to establish directional communication between two DMG STAs. We also demonstrate the obtained throughput for different MCSs for both SC and OFDM PHY layers.

The setup compromises two nodes: one DMG AP and one DMG STA. These nodes are spaced 2m apart from each other. The DMG STA generates a flow of User Datagram Protocol (UDP) messages towards the DMG AP. The announced A-BFT by the DMG AP consists of 8 sector sweep (SS) slots where each slot contains 8 SSW frames.

From simulations, we find that the two nodes spends almost 572 μs to complete an SLS phase for TXSS. The RXSS is performed during the BRP which takes around 396 μs. Figure 7 depicts the obtained throughput for two dif-

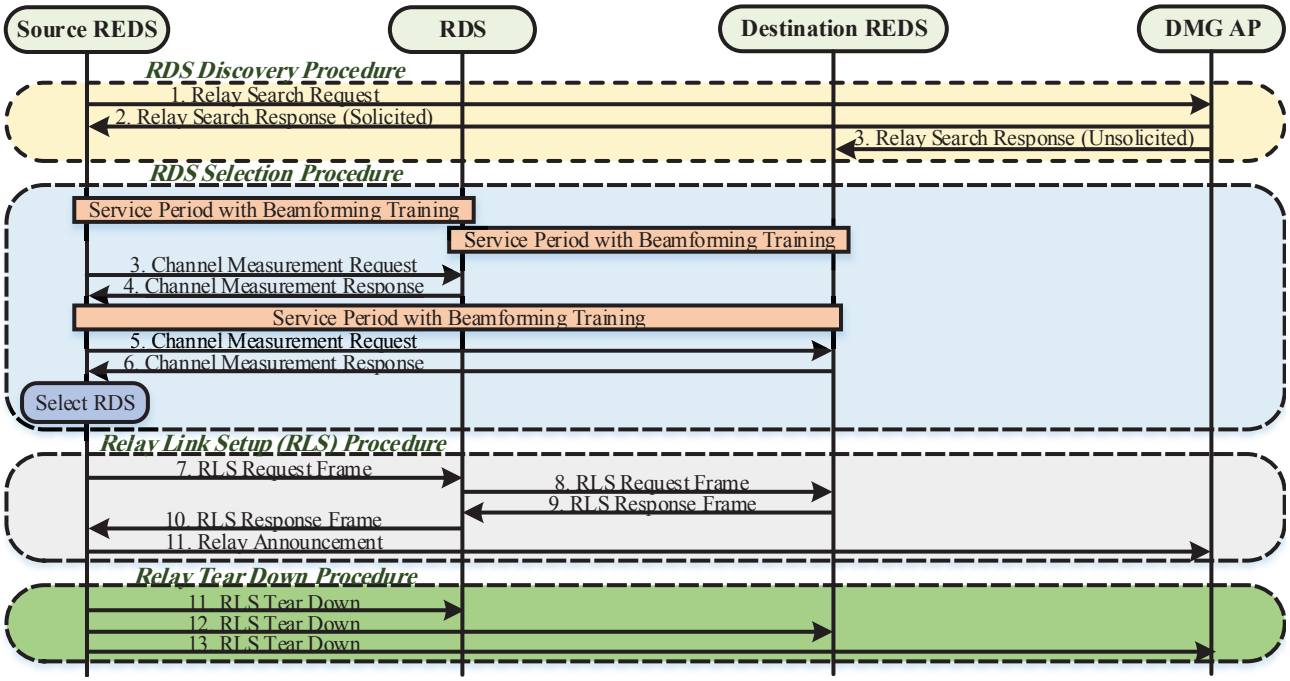


Figure 6: Relay signaling procedure.

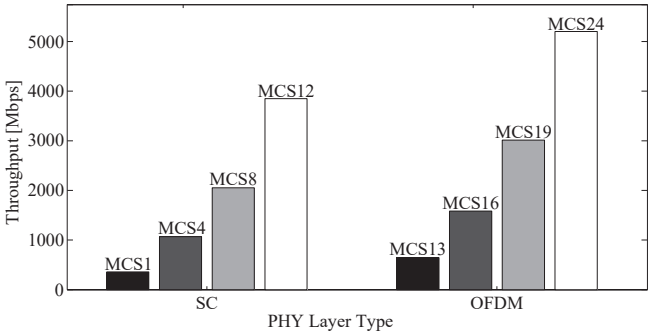


Figure 7: Throughput for different MCSs.

ferent sets of MCSs. The highest throughput achieved for SC is almost 4 Gbps and for OFDM is 5.2 Gbps. However, the achieved throughput for OFDM is 1.5 Gbps less than the theoretical maximum IEEE 802.11ad throughput of 6.72 Gbps. This is mainly due to the overhead imposed by the CBAP access mechanism. Besides that, the data rate reported in the standard assumes a continues stream of OFDM symbols without any PHY and MAC overhead and any Interframe Space (IFS).

4.2 Evaluating DMG Relay Operation

Here, we measure how much time it takes to switch a directional communication link between two REDSs and use an alternative path through an RDS operating in full-duplex amplify-and-forward (FD-AF). Additionally, we calculate the throughput gain we obtain by using this alternative path compared to the case where we do not have any available RDS in the DMG network.

The simulation setup shown in Figure 8 contains one DMG AP with 3 DMG STAs. Two DMG STAs act as REDS and one DMG STA supports RDS. During the DTI acess period, all DMG STAs communicate using MCS 24.

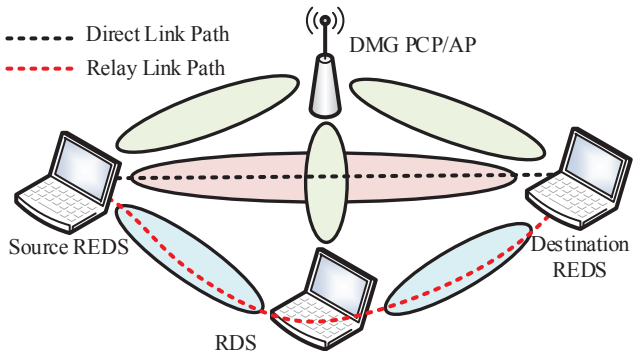


Figure 8: Relay test setup topology.

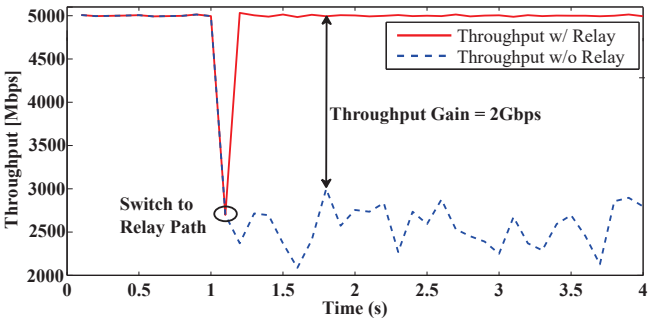


Figure 9: Relay setup results.

At the beginning, the two REDSs are able to communicate with each other over the direct link. At a certain point in the simulation, we introduce a blockage in the direct link.

This blockage does not result in a complete link failure. The source REDS starts to miss some ACKs from the destination REDS and the achievable application throughput is halved. As a counter measure, the source REDS decides to perform an RLS procedure with the selected RDS to obtain a better link quality with the destination REDS. Figure 9 shows the received throughput with and without relay support. From the simulations, we find that it takes around 117 μs to switch from the direct link to the relay link. In addition, using the relay link results in a throughput gain of 2.5 Gbps.

4.3 Evaluating Fast Session Transfer

In this experiment, we demonstrate the capability of transferring an on-going data session smoothly from the 60 GHz band to the 2.4 GHz band. The simulation setup is similar to the one in Section 4.1 with the addition that the nodes can communication in the 2.4 GHz band using IEEE 802.11n. We set the value of LLT to 1000 which corresponds to a link loss countdown value of 32 ms. After the nodes establish the directional link, they setup a unique FSTS between each other.

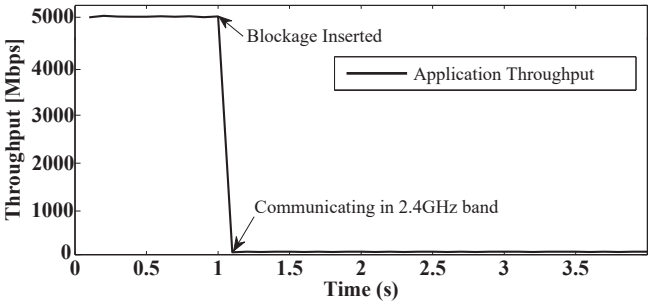


Figure 10: FST setup results.

A the beginning of the simulation, the nodes communicate with each other normally and the achieved throughput is around 5 Gbps as shown in Figure 10. After one second, we introduce a blockage in the link of -45 dBm. This blockage breaks the link so the nodes starts a link loss countdown. When the timer expires, the two nodes switch to the 2.4 GHz band and continue their session. We notice the degradation in the achieved throughput around 60 Mbps due to the limited capacity available in the lower frequency band.

5. CONCLUSION AND FUTURE WORK

We provide an architecture for modeling WLAN IEEE 802.11ad with its various enhancements in ns-3. We implement beamforming training and steering, relay operation, and fast session transfer. We discuss a range of implementations details and how the model integrates into ns-3. In the evaluation section, we demonstrate the overhead required to establish a directional communication link and the average throughput achieved for different MCSs for both SC and OFDM PHY layers. Then, we show that swapping a transmission from a direct link to an alternative link through a relay takes around 117 μs and the throughput gain is almost double. Finally, we demonstrate the capability of fast session transfer of maintaining an on-going session alive. The source code of our implementation is publicly available.

IEEE 802.11ad provides many mechanisms to either avoid communication disruption for certain cases or improves the achievable data rate. The current implementation considers only CBAP access period in the DTI and lacks a hybrid MAC model which provides dynamic channel allocations for applications with strict Quality of Service (QoS) requirements. In our current DMG relay implementation, we model full-duplex amplify-and-forward relay operation mode only and we omit the half-duplex decode-and-forward mode. Additionally, the standard defines a new type of frame aggregation named PPDU aggregation. In this aggregation mode, a DMG STA transmit two or more PPDUs without any separation. Next steps in the further development of our model are to implement these missing features. The current implementation provides all MAC frames and Information Elements necessary for modeling missing MAC sublayer functionalities. Besides, our implementation can also serve as a base line for implementing and evaluating the next generation 60 GHz WLAN IEEE 802.11ay amendment.

6. ACKNOWLEDGMENTS

This work is partially supported by the European Research Council grant ERC CoG 617721, the Ramon y Cajal grant from the Spanish Ministry of Economy and Competitiveness RYC-2012-10788, and the Madrid Regional Government through the TIGRE5-CM program (S2013/ICE-2919).

7. REFERENCES

[1] GitHub Repository for IEEE 802.11ad model. https://github.com/hanyassasa87/ns3-802.11ad.git.

[2] IEEE 802.11ad, Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band. *IEEE 802.11 Working Group*, 2012.

[3] IEEE 802.11ac, Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz. *IEEE 802.11 Working Group*, 2013.

[4] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting Data Center Networks with Multi-gigabit Wireless Links. In *Proc. of the ACM SIGCOMM 2011 Conference*, 2011.

[5] R. Maslennikov and A. Lomayev. Implementation of 60 GHz WLAN Channel Model. Technical report, IEEE, 2010.

[6] M. May. Modelling and Evaluation of 60 GHz IEEE 802.11 Wireless Local Area Networks in ns-3, 2014.

[7] M. Mezzavilla, S. Dutta, M. Zhang, M. R. Akdeniz, and S. Rangan. 5G MmWave Module for the Ns-3 Network Simulator. 2015.

[8] T. Nitsche, G. Bielsa, I. Tejado, A. Loch, and J. Widmer. Boon and Bane of 60 GHz Networks: Practical Insights into Beamforming, Interference, and Frame Level Operation. 2015.

[9] T. Nitsche, C. Cordeiro, A. Flores, E. Knightly, E. Perahia, and J. Widmer. IEEE 802.11ad: directional 60 GHz communication for multi-Gigabit-per-second Wi-Fi [Invited Paper]. *Communications Magazine, IEEE*, 2014.

[10] P. Smulders. Exploiting the 60 GHz band for local wireless multimedia access: prospects and future directions. *Communications Magazine, IEEE*, 2002.

[11] H. Xu, V. Kukshya, and T. Rappaport. Spatial and temporal characteristics of 60-ghz indoor channels. *Selected Areas in Communications, IEEE Journal on.*

Investigation and Improvements to the OFDM Wi-Fi Physical Layer Abstraction in ns-3

Hossein-Ali Safavi-Naeini
University of Washington
safavi@u.washington.edu

Farah Nadeem
University of Washington
farahn@u.washington.edu

Sumit Roy
University of Washington
sroy@u.washington.edu

ABSTRACT

This work presents results based on a critical re-examination of the current physical layer abstractions for IEEE 802.11 OFDM WLAN in the ns-3 network simulator motivated by a) the need to improve fidelity of the Layer-1 abstraction essential for a network level simulator, and b) setting the stage for desired new features (not currently implemented) in the future. We implement a *multi-stage packet reception* that addresses a shortcoming in the current WLAN receiver model, making it closer to existing hardware, and lays the groundwork for improvements such as packet capture. Next, we consider the frame error rate (FER) model in ns-3 which relies on analytical bounds on the bit error probability (BER) at the output of the convolutional decoder. We demonstrate key issues with the current approach through detailed link level simulations using a newly developed Wi-Fi link simulator and look forward to forthcoming fixes being considered.

CCS Concepts

•**Computing methodologies** → **Discrete-event simulation;** *Modeling methodologies;* •**Networks** → **Network simulations**.

Keywords

Wi-Fi, Network Simulator 3 (*ns-3*)

1. INTRODUCTION

In recent years, simulation tools have played a critical role in wireless research and development. Standards organizations such as the 3rd Generation Partnership Project (3GPP) and the IEEE have relied heavily on simulation scenarios to guide the development of cellular and Wi-Fi standards and evaluate ideas proposed for inclusion. Likewise, the research community has made extensive use of simulation tools to accompany mathematical analysis. In fact, in many cases, mathematical analysis is intractable or otherwise cumbersome and simulation tools such as ns-3 have become the primary method for evaluating new designs and performance in specified use-case scenarios. To that end, ensuring the accuracy and performance of these tools is crucial to their credibility and ultimately, wider acceptability. One such tool widely used in academia is the network simulator 3 (ns-3) [2]. ns-3 is an open source packet level simulator, with the intent to support a growing number of wireless and wired network stacks. In this paper, we focus exclusively on the current Layer-1 error model abstractions for ns-3 OFDM Wi-Fi (802.11n); based on a critical examination, highlight some shortcomings.We first propose some architecture modifications that are consistent as a foundation for future extensions. We then implement a new *multi-stage packet reception* framework and evaluate its impact.

WNS3, June 15 - 16, 2016, Seattle, WA, USA

ACM 978-1-4503-4216-2/16/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2915371.2915387

1.1 The Simulation Workflow

Network simulation integrates two categories of simulation tools:

1. **Link Simulators** These signal level simulators (typically built in MATLAB) are designed to closely mimic the physical layer operation of a wireless modem incorporating all details of the digital baseband sections - modulation, demodulation, coding, channel emulation/estimation, and the effect of analog components such as gain control and digitization. The goal is to evaluate receiver algorithms (typically as a function of SNR) to derive bit and ultimately, frame error rates in a single Wi-Fi link. The run-time complexities of these simulators limit their suitability to single link simulations.

2. **System Simulators** System level simulators are designed to abstract away the effects of the physical layer into simple quantities such as frame error rate as a function of SNR. By design, these operate at Layer-2 (frame-in/frame out) and thus require the Link Simulator to provide a frame/packet based link abstraction (i.e. a table look-up for the FER as a function of all relevant system parameters, including SNR). By operating at this level of abstraction, system simulators offer the ability to simulate networks with hundreds of nodes with reasonable run-time. Hence, they are an excellent tool for evaluating the full protocol stack including the effect of application, transport and network layers as well as network topologies.

One important aspect in the usage of system simulators is the choice of the physical layer abstractions (also referred to as the link-to-system mapping). The accuracy of this

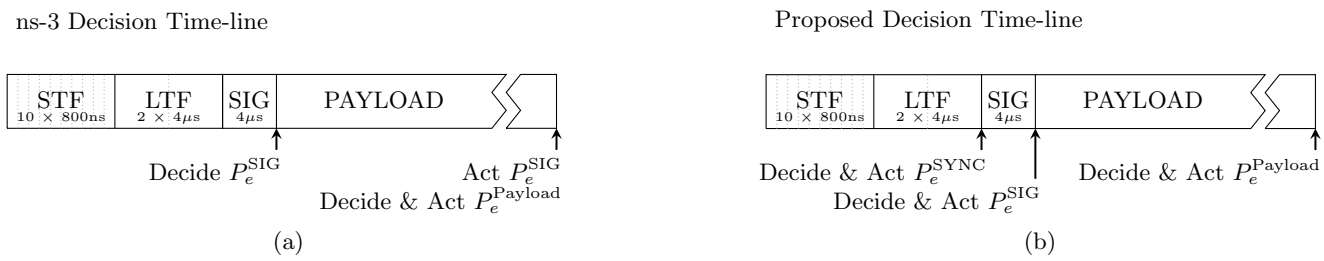


Figure 1: (a)The current implementation of Wi-Fi in ns-3 uses a single action point at the end of the frame. (b)Our newly proposed decision process makes the appropriate decisions at the relevant intermediate points.

mapping (and hence the credibility of the system simulator) reflect how closely the simulator models effects that occur in real networks. In the case on ns-3, the Wi-Fi model at present lacks the ability to accurately depict packet capture [10]. Also, concerns have been raised over the validity of the two error models in use; the YANS error model [13] has been found to be too optimistic for AWGN channels, while the NIST error model [16] has been found to be too conservative [12, 16]. These concerns motivate detailed look at the current physical layer abstraction employed in ns-3.

1.2 Preliminaries

We begin by describing the structure of a Wi-Fi packet as it pertains to the reception process. In this paper, we consider a Wi-Fi system with a channel width of 20MHz according to the 802.11 standard [6]. Since the 802.11g release of Wi-Fi, OFDM has served as the underlying air interface technology for a multitude of reasons pertaining to favorable performance characteristics (e.g. resistance to multi-path delay spread, ease of equalization, etc.) At the physical layer, a Wi-Fi frame consists of four major portions[1] (Figure 1):

Short Training Field (STF): This portion of the frame consists of an 800ns signal repeated 10 times. The periodic nature of the signal is meant to assist in frame synchronization and detection using a delayed autocorrelation method [19]. At the start of the frame, correlating the incoming signal with a version of itself delayed by 800ns presents strong peaks that allow the receiver to locate the start of the frame. In addition, during the STF, the Automatic Gain Control (AGC) at the receiver is adjusted in order to obtain good quantization performance (dynamic range) for the remainder of the frame.

Long Training Field (LTF): This portion of the frame consists of a $4\mu s$ OFDM symbol repeated twice. The purpose of this portion of the frame is to assist in channel estimation, coarse carrier frequency offset estimation, IQ imbalance estimation and other signal processing tasks requiring a reference signal. Errors in the estimate can dramatically impact decoding of the payload. Typical estimators such as MMSE or DFT are employed [8].

L-SIG Field: This portion of the frame is modulated as BPSK with a rate 1/2 convolutional code. It is responsible for carrying information on how the data field can be decoded including the modulation scheme, the coding rate, and the duration of the data. The BPSK symbols are converted into OFDM symbols with the same properties as the data portion.

Payload: The data field is transmitted using 64 sub-carriers with a carrier separation of 312.5 kHz. Some sub-carriers are unused (e.g. DC sub-carrier, 6 on the left and 5 on the right due to the guard bands) and 4 of the modulated sub-carriers are allocated as pilots. The remaining 48 data sub-carriers all use the same modulation which can range from BPSK/QPSK through 64-QAM (802.11a/n) or 256-QAM (802.11ac or later). The data is encoded using a convolutional encoder and decoded at the receiver using a Viterbi decoder. Each OFDM symbol used for data transmission is of duration $4\mu s$ (which includes an $800ns$ cyclic prefix). For Modulation Coding Schemes (MCSes) addressed in this work, Wi-Fi employs a constraint length 7 convolutional encoder with rate 1/2 which can be punctured to achieve other desired rates [6].

1.3 Motivation

The current ns-3 error model for payload reception is based on analytical results for bit error probabilities (note that the L-SIG is nothing but a short duration payload). Specifically, once an expression for P_b (the bit error rate) as a function of SNR is obtained [14], the payload error rate P_e is computed as follows:

$$P_e = 1 - (1 - P_b)^N, \tag{1}$$

where N is the number of data bits in the payload.

In examining the current physical layer abstraction and error model in ns-3, we will consider two aspects:

1. Sources of packet/frame error
2. Accuracy of error probabilities as a function of SNR and payload length

To address the first aspect, we note that in the current implementation of ns-3, a successful packet entails a computation at the end of the L-SIG for correct reception of the header, followed by an error rate computation at the end of the frame to decide on correct reception of the payload. Although failure of the L-SIG is considered at the end of the L-SIG field, the model defers any change of state notification (i.e. the receiver remains in an RX state) until the end of the frame. This does not conform to existing hardware Wi-Fi systems. Lastly, errors during STF/LTF are not even considered.

But first, we begin our examination with the second facet of the abstraction by considering the analytical bit/packet

[1]We omit specifically addressing additional header fields introduced in the HT modes in this work and leave it as a subject of future work

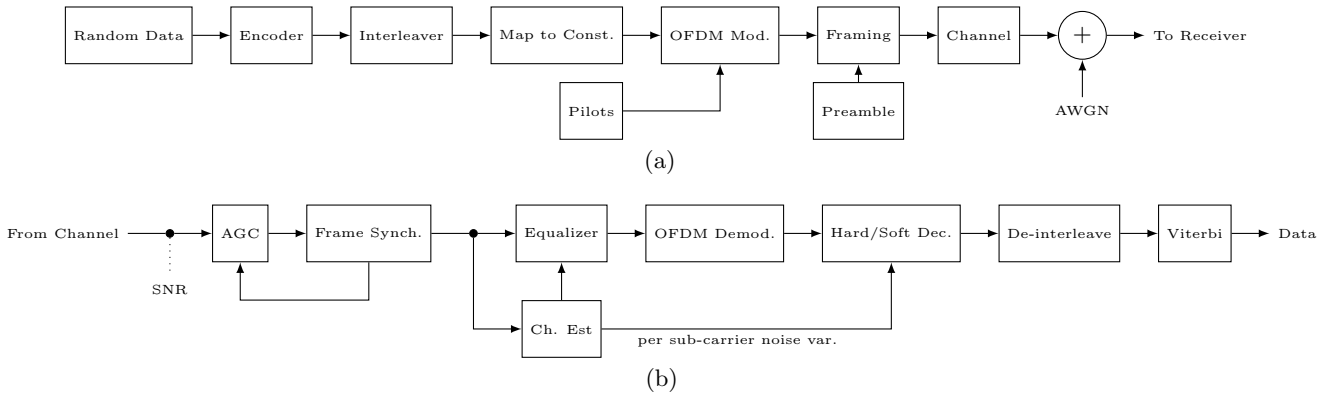


Figure 2: (a) Block diagram for a single antenna Wi-Fi transmitter. (b) Receiver block diagram for single-antenna Wi-Fi receiver link simulations.

Table 1: Link Simulation Parameters.

Antenna	1x1 (SISO)
Sampling Rate	100 MHz
AGC	Logarithmic Loop
ADC	Ideal 12bit
Synchronization	Delayed Auto-correlation
Channel Estimation	Ideal (AWGN)
Demodulator	Soft Decision (8bit quantization)
Decoder	Viterbi (128 Traceback)
Noise Figure	0dB
Iterations	> 2048

error rates used within ns-3. In the next section, using the published results from 802.11 standardization as a guide [15], we take a closer look at these error rates and consider any factors that may explain the large discrepancies.

2. AWGN PHY LAYER SIMULATIONS

In order to take a critical look at the ns-3 physical layer error model, we employ a MATLAB based link simulator [5] and focus on a SISO Wi-Fi system as a natural first step. Figure 2 shows a block diagram describing operations carried out in our link simulator. Additionally, the current incarnation of ns-3 operates under the assumption of an AWGN channel exclusively, hence the results we produce make the same assumption (we hope to address frequency selective channel models such as TGn Channel D in the future).

On the transmitter side, after the application of a convolutional error correcting code [6], the bits are interleaved and modulated according to the desired constellation (BPSK through 64QAM) before being transmitted through the channel. The parameters used for the link simulations are shown in Table 1. We emphasize that *noise figure* is set to 0dB for all the results in this section ensuring a fair comparison. Non-zero noise figure can easily be accounted for as a simple additive SNR shift for all results.

The currently employed ns-3 model for frame errors is described in 1.3 and in particular in (1). The two issues with this approach which we aim to study in this section:

1. The bit error rate bounds are accurate at higher SNRs, yet the lower MCSes are typically used at lower SNRs (where the bound is loose).
2. Computing the frame error rate using (1) makes the *critical* assumption that bit errors occur independently

There are two other discrepancies when comparing ns-3 to our simulations as well as those from IEEE Task Group n (TGn). Firstly, modern WLAN receivers typically use soft decoding in place of hard decoding due to the 1-2dB gain[2] afforded as a result. The 802.11 standard does not require any specific decoding mechanism, though usage of soft decoders has become standard practice.

Another contributing factor in mismatch between ns-3 and simulations lies in the computation of the SNR (Figure 2b shows where SNR is measured in the system). Currently, ns-3 considers noise power over 20MHz while in reality the noise of interest is limited to the occupied sub-carriers (i.e. 52/64), hence an additional SNR shift must be accounted for:

$$SNR_{\text{dB}} = 10\log_{10}\frac{P_{TX}}{N_0 20\text{MHz}} + 10\log_{10}\frac{52}{64} \qquad (2)$$

In Figure 3a, we compare our link simulation results to those produced by the IEEE's 802.11n task group [15] during standardization for MCS0, NCS3 and MCS7 (selected for simplicity, though all MCSes exhibit the same behavior). While our link simulation results for frame errors closely match those of IEEE TGn, ns-3 displays a far more pessimistic error rate. We can observe that this disparity is not merely an SNR shift since the slope of the graphs also differs (ns-3 exhibits a sharper transition).

Even if we were to adjust for the SNR offset described by (2), ns-3 retains a 2-3dB gap across the range. This can be attributed in part to the assumption of independence regarding bit errors which in effect spreads errors amongst multiple frames when they are actually more concentrated/localized. In our opinion, this intrinsic assumption makes models that rely on bit errors less suitable for use in ns-3.

2.1 Packet Error Rates in ns-3: Issues

A possible remedy is for the ns-3 physical layer to move to a packet error based model that uses link simulation results

[2]The real system gain is less than the theoretical one due to quantization and truncation of the log-likelihood ratios

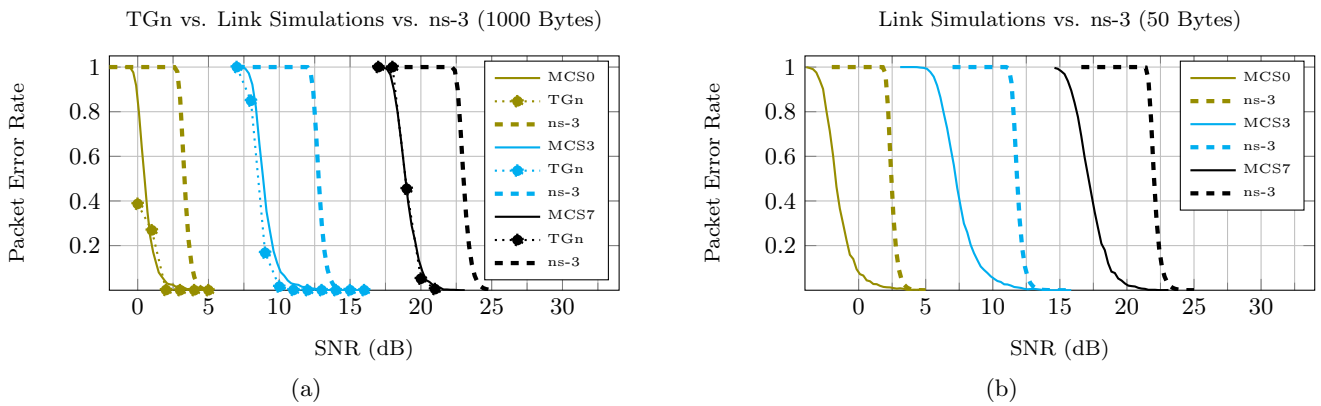


Figure 3: (a) Comparing simulation and TGn results (see [15] Figure 2-1) to ns-3 reveals a large gap. The transition is also more rapid in the case of ns-3. (b) The SNR gap between ns-3 results and those of the link simulator widens for smaller payloads. No TGn results exist for this payload size.

to decide on packet errors directly. In deriving frame errors from analytical bounds on bit error rates for coded bits, we encounter complicating factors that warrant a thorough examination of mappings from SNR to error rates:

- Effect of frame length on error rates
- The impact of coding on isolated bit errors

It is this second shortcoming which is most problematic. It is well known within the literature that bit errors occur in bursts at the output of the Viterbi decoder [9]. Hence, using the Viterbi bit error rate under the iid assumption (1) results in a very loose upper bound. While this equality holds for an uncoded packet, for coded bits, a tighter *upper bound* can be obtained by using the first error probability P_e in (1) instead of the *bit error* probability [17].

Figure 3b compares the error rates for a shorter frame duration (50 bytes) that could represent something like a TCP acknowledgment message. Clearly, the gap has increased and the slope has diverged even more in comparison to the 1000 byte packets examined in 3a. Likewise, in the case of the L-SIG (a 3 byte payload at MCS0), the difference is even more stark. In Table 2, we show the required SNR to decode the L-SIG with various success rates. To achieve a 50% success rate in decoding the L-SIG, ns-3 error models indicate a required SNR of 1.7dB while our link simulator can do so at -3.5dB, a difference of more than 5dB.

Then, we must also consider the effect of varying SNR during a frame which is currently tackled in ns-3 by splitting the frame across bit boundaries (potentially introducing additional inaccuracies). And finally, we note that ns-3's current treatment of interference via a unified SINR (whereby interference is treated as additive noise) has it's own limitations; but changes to this is deferred to future. We conclude this section by stating that the need for an improved physical layer abstraction is clear, and through the proposed improvements, we set the stage for subsequent improvements in ns-3 WLAN/OFDM Phy abstractions in future.

3. WI-FI RECEPTION PROCESS IN NS-3

The existing Wi-Fi reception process in ns-3 works based on the received SNR at the physical layer [1] and frame

Table 2: Required SNR to achieve given success rates for SYNC and L-SIG.

	ns-3 (SNR)		LinkSim (SNR)	
Success Rate	50%	95%	50%	95%
SYNC		-	-10 dB	0.21 dB
SIG	1.7 dB	2.4 dB	-3.5 dB	0.3 dB

length. We propose to modify the existing decision process to the one shown in Figure 1b, with the ability to "drop" a frame at any of the decision stages, bringing it closer to actual WLAN implementations [7]. The motivation behind this modification is two fold: i) helping to account for the lack of capture effect in existing ns-3 reception, and ii) anticipated low SINR scenarios in future co-existence simulations within ns-3. Splitting up the reception process requires error results for the STF/LTF preamble sync, which we generate from link simulations.

A selection of results for sync are tabulated in Table 2. A natural question to ask is: what is the significance of the added stages? At SNRs higher than 5dB, synchronization is almost always successful, however, keep in mind that ns-3 uses SNR in lieu of SINR, hence it incorporates interference. While periodically low SINRs at the start of the frame are expected to occur more frequently in co-existence studies, they can occur even in the current Wi-Fi only simulations when accounting for high node densities or hidden nodes (see Table 3).

As an example, accounting for drops at the LTF/STF stage can lead to a 5.5% drop in throughput in the case of 10 flows on the network. Conversely, if we consider a hidden node scenario (described in [3]), we observe that partial collisions within the STF/LTF occur frequently. If the receiver is locked onto a weak signal through the end of a frame, other subsequent stronger frames will be ignored. By dropping a weak packet early, running the same simulation scenarios yields a throughput gain of up to 16% at high offered loads.

Given this motivation, we modify the Wi-Fi reception in ns-3. It is pertinent to mention here that inclusion of accu-

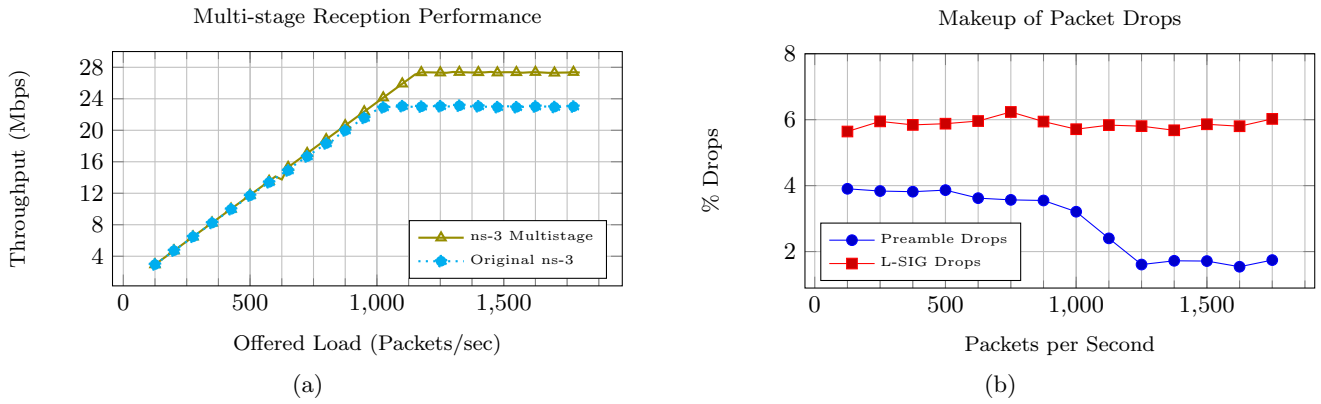


Figure 4: (a) Throughput increase in the hidden node scenario due to multi-stage reception (b) Percentage of frame drops occurring before the payload captured by multistage reception.

Table 3: Likelihood of Low SNR During Synchronization for 25 Node Ad-Hoc Network [4].

	Frequency of Occurrence			
Flows	$< 2dB$	*SINR*	$< 5dB$	*SINR*
2	1.5%		2.4%	
5	2.47%		4.04%	
10	3.56%		5.71%	

rate sync error models adds to the fidelity of the reception process as opposed to relying only on the L-SIG error rates and assuming perfect synchronization based only on *received power*. This will have a greater impact in the case of multiple possibly heterogeneous interference sources.

We now go on to describe the simulations used to obtain the synchronization error rates, and the impact of the multistage reception on existing ns-3 Wi-Fi only simulations.

3.1 Multistage Reception

We use link simulations to provide a synchronization failure rate (P_e^{SYNC}) for the newly added stage in ns-3. Aside from the trivial case of not detecting the start of a frame altogether, any timing error in the synchronization can propagate through the remainder of the reception process (so called "soft effects"). In particular, the effective SNR loss due to synchronization error has some impact on payload demodulation especially for higher modulation orders (e.g. 64QAM).

We do not propose keeping track of such soft effects at this time due to the added implementation complexity that it would entail (though we are considering them for inclusion in the future). Instead, in order to arrive at a number for P_e^{SYNC}, we set an acceptable synchronization error threshold of less than 1 cyclic prefix (i.e. $800ns$) within the LinkSim. If the initial synchronization estimate is within this window, we consider the timing acquisition (and frame detection) to have been successful[3]. Table 2 shows some synchronization results as a function of SNR for the AWGN channel. Note that if there is significant SNR variation between the synchronization stage and the payload, the soft effects of synchronization will become more apparent, however, we leave this case for future work.

The implementation has been modified to separate the preamble and header reception process, with the ability to drop the packet at either of these stages and, potentially, commit to another incoming packet. The preamble reception is compliant with the standard [6, 15.3.6.2]. At the start of reception, if the PHY is either IDLE or CCA_BUSY, and there is currently no SYNC being attempted, an end of preamble event is scheduled. At the LTF, the success rate of the preamble sync is looked up from LinkSim based results, and an event for L-SIG header reception is scheduled. If the preamble has synced successfully, the reception process moves onto header decode, otherwise the frame is dropped and the PHY reverts from the reception state. If the header is decoded successfully, this is followed by payload decode. Again, if the header fails, the frame is dropped with a corresponding change in the PHY state. This model necessitates the addition of a state to indicate whether there is a preamble sync being attempted, and to ensure that a header decode is preceded by a successful preamble sync.

[3]For higher MCSes, a fine grained timing synchronization step can follow the initial coarse estimate [18].

4. NS-3 SIMULATION RESULTS

To study the impact of multi-stage reception on Wi-Fi simulations, a canonical hidden node scenario was studied, both with and without RTC/CTS enabled [3]. Two nodes, hidden from each other, transmit to a common access point placed in the center. The L-SIG was transmitted at MCS0, with the payload at MCS7 for 1472 byte frames. Constant bit rate traffic was generated, and the number of frames per second ranged from 125 ($1.47Mbps$) to 1800 ($21.19Mbps$) for *each* transmitting station. The default NIST error rate model was used for both cases, with SYNC results from LinkSim incorporated for the multi-stage reception.

The original reception process gives lower throughput, since incoming frames with a high (average) SNR can be dropped due to the AP's commitment to receive a frame from the other station that partially overlaps the incoming frame. With modified reception, the throughput improves, with increased effect at higher traffic loads as seen in Figure 4a. To further illustrate the impact of frame drops at the

preamble and header stages, the percentage of total drops occurring at these two stages were tabulated 4b. In the hidden node scenario, 8-10% of the total drops happen before the payload stage. For these cases, the AP is now free to receive the next incoming frame, instead of being tied to the last received erroneous frame. In both implementations we observe that the throughput levels off at increasing offered load due to built-in MAC layer overheads.

The same experiment was repeated with RTC/CTS enabled. Since the data packet collisions decrease significantly, the throughput for both the multi-stage and original reception line up. There is still a small difference in the throughput ($0.3Mbps$) at the maximum offered load (1800 packets per second), due to the collision of RTS frames.

When the same simulation scenario is repeated with both stations visible to each other, the throughput is identical for both cases. For simulations with multiple flows with no hidden nodes, the throughput with multi-stage reception was lower than the original ns-3 implementation. One example scenario was an ad-hoc grid [4], where the throughput for multistage is 0.3% lower for 2 flows, and decreases by 5.5% for 10 $4Mbps$ flows. This difference results from an overall greater number of frame drops, a direct consequence of the newly added sync stage.

The results indicate that in current scenarios, multistage reception has an impact on the simulation results, albeit in a limited fashion. However, while our current work does not explicitly model frame capture, the additional stages we have introduced set the stage for inclusion of capture effect in the future which will lead to even greater changes in throughput (see [11]).

5. CONCLUSION AND FUTURE WORK

In this paper, we evaluate the ns-3 physical layer abstraction to improve fidelity and accuracy. We developed a link simulation tool and show that its results closely match those published by the IEEE task group charged with Wi-Fi standardization. We then used our link simulator to examine possible sources of error in the ns-3 physical layer error model that support a move away from the default analytical model. Then, with a eye towards future inclusion of capture effects in ns-3 as well conforming to existing hardware implementation, we used results from our link simulator to introduce a multi-stage reception model for ns-3 with inclusion of preamble sync error rates. Finally, we studied the effect of our modifications in some canonical scenarios (hidden nodes and RTS/CTS) to ensure that the behavior conforms to expectations. In this context, we are validating and developing both analytical error models, and link-sim results, as well as changes to the physical layer abstraction to augment the fidelity and range of applicability of ns-3.

Although not finalized yet for submission to the ns-3 main tree, we have developed modifications to the YansWifiPhy class to enable the third stage of reception and to provide a link simulation-based error model for the PLCP preamble fields. We have also developed a framework that permits the loading of link simulation-based error models that are expressed as text files, to replace the analytical error models. We plan to investigate further with our link simulator how best to handle situations in which the SNR varies during the duration of the received frame, including occurrences of Wi-Fi signal overlap.

6. ACKNOWLEDGMENTS

The authors would like to thank Thomas Henderson from the University of Washington for his invaluable guidance and help during the course of this work. The authors would also like to thank Benjamin Cizdziel for his contributions in developing early prototypes of the multistage reception process.

7. REFERENCES

[1] ns-3 Design Documentation: Wi-Fi Module.
[2] ns-3 Network Simulator.
[3] ns-3 Wi-Fi Example: wifi-hidden-terminal.
[4] ns-3 Wi-Fi Example: wifi-simple-adhoc-grid.
[5] University of Washington Wi-Fi Link Simulator.
[6] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Nov 2012.
[7] Broadcom. *WLAN Chipset BCM2050.*
[8] Y. S. Cho, J. Kim, W. Y. Yang, and C.-G. Kang. *MIMO-OFDM Wireless Communications with MATLAB*. Wiley, 1st edition, 2010.
[9] L. Deutsch and R. Miller. Burst Statistics of Viterbi Decoding. Technical Report TDA Progress Report 42-64, NASA, May 1981.
[10] P. Fuxjaeger and S. Ruehrup. Validation of the NS-3 Interference Model for IEEE 802.11 Networks. In *Wireless and Mobile Networking Conference (WMNC), 2015 8th IFIP*, October 2015.
[11] X. Ge, Dongyan, and Y. Zhu. Throughput Model of IEEE 802.11 Networks with Capture Effect. In *Wireless Communications, Networking and Mobile Computing, 2006. WiCOM 2006.International Conference on*, pages 1–4, Sept 2006.
[12] C. Hepner, A. Witt, and R. Muenzner. In Depth Analysis of the ns-3 Physical Layer Abstraction for WLAN Systems and Evaluation of its Influences on Network Simulation Results. *SINCOM 2015*, page 46, 2015.
[13] M. Lacage and T. R. Henderson. Yet Another Network Simulator. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 12. ACM, 2006.
[14] L. Miller. Validation of 802.11 a/uwb coexistence simulation. Technical report, national institute of standards and technology (NIST), WCTG white paper, 2003.
[15] S. A. Mujtaba. TGnSync Proposal PHY Results. Technical Report IEEE 802.11-04/891r5, Agere Systems, July 2005.
[16] G. Pei and T. R. Henderson. Validation of ofdm error rate model in ns-3. Technical report, 2010.
[17] J. G. Proakis and M. Salehi. Digital conununicarions, 2007.
[18] T. M. Schmidl and D. C. Cox. Robust Frequency and Timing Synchronization for OFDM. *Communications, IEEE Transactions on*, 45(12):1613–1621, 1997.
[19] K. Wang, M. Faulkner, J. Singh, and I. Tolochko. Timing synchronization for 802.11a wlans under multipath channels. In *Proc. ATNAC*, volume 2004, 2003.

LL SimpleWireless: A Controlled MAC/PHY Wireless Model to Enable Network Protocol Research

Patricia Deutsch
MIT Lincoln Laboratory
Lexington, MA 02420
patricia.deutsch@ll.mit.edu

Leonid Veytser
MIT Lincoln Laboratory
Lexington, MA 02420
veytser@ll.mit.edu

Bow-Nan Cheng
MIT Lincoln Laboratory
Lexington, MA 02420
bcheng@ll.mit.edu

ABSTRACT
Wireless network protocol research typically requires evaluating performance over a set of controlled wireless link conditions. Although ns-3 provides wireless models like WiFi and WiMax, they have dozens of parameters that affect performance and it is difficult to control link rate, latency, error, and other attributes. To mitigate this issue, we extended a basic, range-based ns-3 model called SimpleWireless. Features that were added to this model include transmission delay, configurable queues that enforce a data rate, support for differentiation of control and data traffic, support for several configurable error models, support for directional networking, support for fixed contention and finally, support for PCAP packet capture. The goal of these additional features is to provide network protocol researchers with a basic yet feature rich wireless model that enables evaluating their protocol in a controlled wireless environment. In this paper, we describe the base SimpleWireless model and each feature that has been developed to enhance that model and create the so called "LL SimpleWireless" model. Additionally, we provide information on performance evaluation of the LL SimpleWireless model to verify functionality of the added features.

CCS Concepts
• **Networks → Network performance modeling** • **Computing methodologies → Model development and analysis;** *Discrete event simulation; Simulation support systems;*

Keywords
Modeling; Simulation; ns-3; Wireless

1. INTRODUCTION
Network Simulator Version 3 (ns-3) [6] is a discrete-event network simulator that provides several models for simulating wireless environments such as WiFi [11] and WiMax [2][12]. However, these models do not necessarily lend themselves to use for studying new network protocols in a wireless environment because they couple medium access control (MAC) protocol specifics with PHY channel effects. WiFi has limited transmission range and does not support a large number of nodes. WiMax overcomes these limitations, but with both wireless technologies, there are dozens of parameters that affect network performance and in many situations a user may want a more controlled wireless environment. For example, when testing routing protocols, a user may want a wireless environment that has limited delay and guaranteed delivery. For these users, a SimpleWireless model is available as an add-on to ns-3 [10]. Though this model solves the complexity issues of using WiFi or WiMax, it is perhaps a little too simple as it only provides propagation delay, receive error and transmission range features. To enable wireless network protocol research with tightly controlled MAC/PHY effects, we enhanced the existing SimpleWireless model to provide a more realistic and configurable wireless environment while maintaining the simplicity that makes it a better choice than WiFi and WiMax for certain simulation applications. In this paper, we present LL SimpleWireless (LLSW) which extends SimpleWireless to include transmission delay, configurable queues that enforce a data rate, support for differentiation of control and data traffic, support for several configurable error models including range based error, support for directional networking, fixed contention, and finally, PCAP [5] packet capture.

Key contributions of the work include:

1. LL SimpleWireless: A simple and configurable ns-3 MAC model to enable network protocol research
2. Evaluation of the key features and discussions on the applicability of LL SimpleWireless

The rest of the paper is organized as follows: Section 2 describes the original SimpleWireless model while Section 3 focuses on the key enhancements (LL SimpleWireless). Section 4 presents evaluation results of the enhancements and discusses implications. Finally, Section 5 concludes the paper and highlights some future work.

2. SIMPLE WIRELESS MODEL
The ns-3 SimpleWireless model [10] is available as an add-on to ns-3. It provides a simple range-based on/off model for the transmission of packets and models a network interface controller that is not based on any specific protocol but provides a simple protocol for wireless communication (hence its name). The model consists of two parts: SimpleWirelessChannel and SimpleWirelessNetDevice. Figure 1 shows the base SimpleWireless model.

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915376

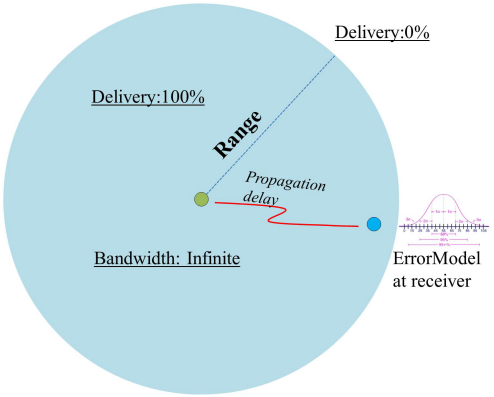


Figure 1. Base ns-3 SimpleWireless Model.

The SimpleWirelessChannel is the physical layer model and is used to transmit packets within a user specified transmission range with a propagation delay based on speed of light. When the SimpleWirelessChannel receives a packet from the SimpleWirelessNetDevice to transmit, it iterates over all devices attached to the channel to determine if the packet should be sent to each of these devices. The decision is quite simple: if the destination is within the specified transmission range of the sender then delivery of the packet to the destination device is scheduled in the future with a delay based on propagation time of distance times the speed of light. This transmission range is all or nothing – within the range yields 100% delivery and outside the range yields 0% delivery. Besides the range check, there are no other limitations on sending the packet to the destination.

The SimpleWirelessNetDevice is the device model and is used to receive packets based on a user specified error model. When a packet is received from the channel of a sending node, the user specified error model is applied to the packet and if the result is a packet in error, the packet is dropped. Otherwise, the packet is considered a successful reception. The error model uses ns-3's ErrorModel [7] class and thus supports several types of error models: rate error (error based on a random distribution), list error (error based on a list of packet IDs), and burst error (error in random bursts). The available ErrorModel types allow errors to be applied on a per-packet, per-byte, or per-bit basis.

With this understanding of the SimpleWireless model, its limitations begin to appear. Though there is support for packet errors, it is a fixed random distribution. Within the transmission range, every packet has the same probability of error without regard to distance from the sender. In reality, a wireless transmission will have an error rate that is based on distance, that is, the closer the receiver is to the source, the less errors will occur and the error rate grows as the distance increases. Thus the error modeling of the SimpleWireless model is not necessarily representative of wireless transmission in which distance can affect the amount of errors a packet experiences. As well, errors are only applied on the receive side but applying an error to packets on the send side would reduce simulation overhead because a packet that is deemed not deliverable is dropped at the source instead of being retained through to reception. Another limitation of the SimpleWireless model is that there is no concept of data rate thus the channel has effectively infinite bandwidth. Finally, all nodes in the network are considered potential receivers so there is no support for directional networks.

3. LL SIMPLE WIRELESS

In seeking to overcome the limitations of the SimpleWireless model and create a more robust model, several features were added to the SimpleWireless model to create the LL SimpleWireless (LLSW) model. The following features were added and are described in the sections that follow:

- Error Models (including range based error)
- Packet Queuing (including control and data sub-queues)
- Transmission delay
- Fixed contention
- Simulated directional networking
- PCAP packet capture

With the exception of the transmission delay, all of these features are optional and are not required to be enabled. Thus if a user wants the behavior of the original SimpleWireless model, this can be accomplished by not enabling the new features and setting the data rate to a large enough value that the transmission delay is negligible. Figure 2 shows the LLSW model with some of these features.

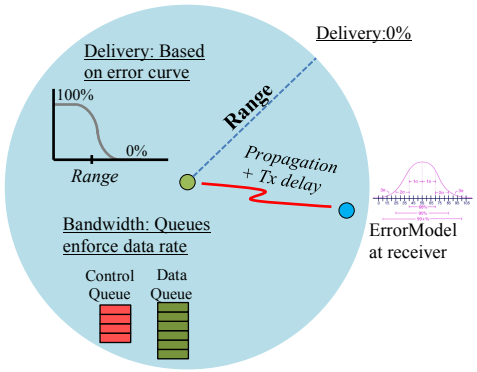


Figure 2. LLSimpleWireless Model.

3.1 Error Models

In order to enhance the error modeling capabilities and potentially reduce simulation overhead, LLSW implements several error models on the send side: Constant Error Rate, Packet Error Rate Curve, and Stochastic.

The Constant Error Rate model applies a uniform random distribution for dropping packets. When a packet is to be sent, a uniform random selection is performed and if the returned value (which is between 0 and 1) is less than the user configured error rate then the packet is dropped and not sent. This is quite a simple error model and in fact has fewer features than the error models available on the receive side. This error model was added to the send side simply to provide a quick and easy method for dropping packets at the source and improving simulation speed.

The Packet Error Rate Curve model applies a user specified curve of distance versus error for dropping packets which provides a more realistic transmission distance-based error. As part of creating a scenario, the user builds this error curve using a set of <distance, error rate> pairs. When a packet is sent, the distance between the sender and receiver is used to determine a value from this curve, linearly interpolating between points on the error curve to provide the acceptable error rate for the given distance. If a uniform random selection (values between 0 and 1) yields a value that exceeds this error rate, the packet is considered in error and is not sent. Figure 3 shows an example of this type of error model. Because the user has full control over creating the error curve, this

error model is very flexible in providing a realistic transmission range based error.

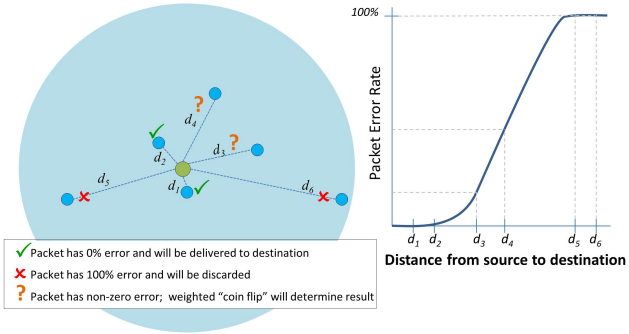


Figure 3. LL SimpleWireless PER Curve Error.

While these two newly added error models are packet based, that is, they are used to determine if any one packet is in error, the third error model, Stochastic, is used to disable the entire channel between specific sender/receiver pairs. The Stochastic error model [15] uses two exponential distributions with user specified means – an "up" distribution and a "down" distribution – to toggle the state of the channel between each sender/receiver pair in the network. When the channel is down, all packets for the destination from the sender are dropped. When the channel is up, all packets for the destination from the sender are delivered. Figure 4 shows an example of this type of error model.

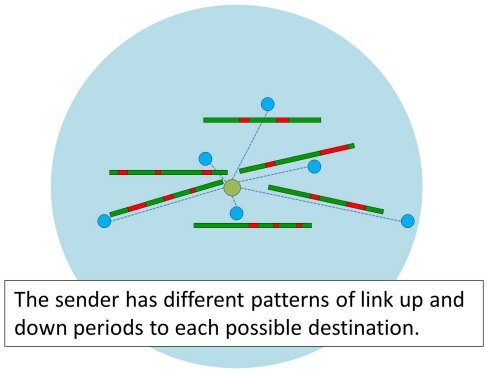


Figure 4. LL SimpleWireless Stochastic Error.

The state of the channel is continuously altered between up and down states based on the randomly selected state durations. The selection of the channel state occurs when a packet is sent and is not performed using timers to manage the states. This keeps simulation overhead to a minimum because only those source/destination channels that are being used have their states maintained. Figure 5 shows an example of the channel state selection. At initialization, the channel state is set to ON and the random duration of this state is selected. Each time a packet is sent, the current time is compared to the state end time and if the state end time has passed, the next state duration is randomly selected until the end time of the new state is greater than the current time.

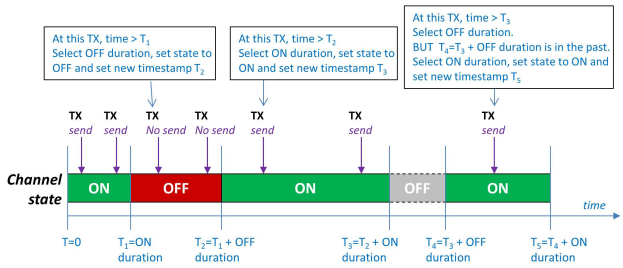


Figure 5. LL SimpleWireless Stochastic Error State Selection.

It is important to note that these error models have all been added to the send side so that any packet that would not arrive at the destination were it to be sent is not sent. This reduces simulation overhead by deleting packets earlier in the processing chain.

3.2 Packet Queuing

In order to provide support for a device data rate, LLSW implements queuing of outbound packets at the SimpleWirelessNetDevice. Queues are always First-In-First-Out (FIFO) but the method for dropping packets on a full queue is configurable to be either drop head or drop tail. In addition, there is support for priority queues which implements separate control and data sub-queues, each of which can be set independently as drop head or drop tail. When a priority queue is used, a PCAP string filter can be configured and is used to differentiate control and data packets. The control sub-queue operates as strict priority, meaning it is always serviced first before the data sub-queue.

When queues are used, the SimpleWirelessNetDevice maintains a transmit state flag to indicate if the device is currently transmitting or is idle. Figure 6 shows an example of the use of this transmit state flag. When the SimpleWirelessNetDevice receives a packet from the upper layer to transmit, it places the packet into the queue and if currently idle, immediately transmits the packet and sets the flag to busy. After the transmission is complete, it sets the flag back to idle and checks the queue to see if there is another packet waiting to be sent. The transmission time of a packet is based on the packet size and the user configured data rate.

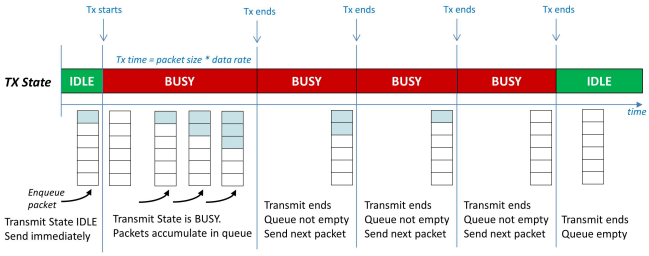


Figure 6. LL SimpleWireless Queue Transmit State.

It is important to note that the busy state is set and maintained at the SimpleWirelessNetDevice without regard to what happens to the packet when transmitted via the SimpleWirelessChannel. As indicated previously, the SimpleWirelessChannel does not actually send a packet that would be delivered in error at the destination node. Such packets are dropped and not sent over the channel. However, the device does not know about this and will still be considered "busy" for the amount of time required to transmit the packet even if the channel does not actually send it. This is an important aspect of maintaining a data rate, and despite packets being dropped instead of being transmitted, the SimpleWirelessNetDevice still considers the time it would have

taken to transmit the packet as not available to transmit any other packet.

Enabling the use of queues is optional and when not used the behavior is that of the original simple wireless model. The SimpleWirelessNetDevice immediately sends a packet when it receives it and does not maintain a busy flag.

3.3 Transmission Delay

Transmission delay is simply the size of the packet times the data rate of the device. This delay is added to the propagation delay when the packet delivery is scheduled in the SimpleWirelessChannel. The data rate is a newly added attribute on the SimpleWirelessNetDevice thus the device actually provides the transmission time value to the channel when a packet is sent. The data rate is defined on the device because it is also used to support queuing which is discussed in the previous section.

3.4 Fixed Contention

Fixed contention is a feature used to account for contention on the channel. Each device uses the queuing mechanism to maintain the specified data rate but in reality this bandwidth may actually need to be shared amongst neighboring nodes. That is, each neighbor node cannot have 100% of the available bandwidth. The fixed contention feature is used to simulate the sharing of the channel among neighbors. Figure 7 shows an example. Each time a transmission occurs, the model counts the number of neighbor nodes that could cause contention on the channel and uses that number to adjust the data rate to a value that would be the effective data rate based on contention on the channel. A node is considered to cause contention if it is within a user configurable "contention range". This value is a separate value from the transmission range of the channel but if it is not specified explicitly by the user and the feature is enabled, the transmission range will be used as the contention range. Once the count of neighbors in contention is determined, it is used to reduce the data rate available to a sending node. The data rate becomes the original data rate divided by the number of contention neighbors. The device uses this new, lower data rate when it determines the transmit time which is used for setting how long to keep the device's busy flag set and for transmission time on the channel. The manner in which the number of neighbors is counted is as follows: the count first includes the sending device itself. Then when the channel transmits a packet, it loops over all devices on the channel. During this loop the number of destination devices that are within the user specified contention range is counted and this is the count that is used for the next time that a packet arrives on the device to be sent. Note that this information is not the exact number of nodes within contention range at the instant when a packet is queued but is based on the previous transmit. The neighbor count could be taken at the start of each transmit but because the channel already loops over all the devices as part of its sending, the fixed contention feature was designed to simply re-use this loop instead of adding a second loop and introducing overhead to a simulation.

Is it important to note that this method of simulating contention is best used in scenarios with high volume traffic because that is when the full number of neighbor nodes would actually cause contention. When the volume of traffic is low and nodes are able to transmit when no other neighbor is transmitting, this contention model would falsely reduce the data rate as though the transmissions were contested when in fact they were not.

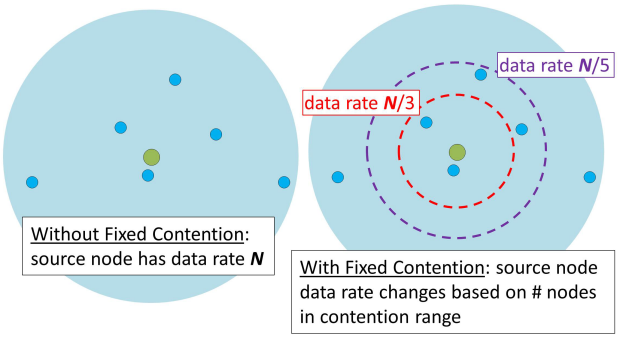


Figure 7. LL SimpleWireless Fixed Contention.

3.5 Simulated Directional Networking

Although commercial wireless technologies are generally omni-directional systems, there has been a large push in recent years to leverage highly directional systems for increased capacity, range, and reliability [1][3]. Implementation of highly directional radio systems adds significant complexity due to the need to track neighbor positions, orientation, and antenna patterns. To provide basic directional effects to network protocol researchers, LLSW implements a simple version of directional networks with the use of a "neighbor list" in the SimpleWirelessNetDevice. In short, the "neighbor list" is a fixed configuration which identifies the neighbors that are in view of the node as though there was a directional antenna. The goal is not to simulate the full details of directional radio systems, but to give a high level effect to network layer protocol researchers. Topology management schemes can be built on top of this feature to dynamically elect neighbors to connect if necessary. If the directional networking feature is disabled then all nodes are within view of the node. If this feature is enabled, then only the nodes listed in the node's neighbor list are considered in view of the node. When directional networking is enabled for the device, the SimpleWirelessNetDevice enqueues outbound packets as follows:

- A broadcast packet is duplicated and enqueued once for each directional neighbor.
- A unicast packet is enqueued only if the destination is a directional neighbor.

Note that when the directional network feature is enabled, packets are enqueued for a specific destination so that when the SimpleWirelessChannel receives the packet to send, there will be only one destination device to which the packet will be sent. This differs from the case without a directional network when only one packet is enqueued. As an example, suppose a node has nine neighbors, five of which are considered directional neighbors. If directional networks are enabled, the device will enqueue and serially transmit five packets – one for each directional neighbor. It is assumed that there is just one radio so the transmissions are serial and not simultaneous. When the channel receives a packet to send, it will only deliver it to one destination device. This is equivalent to five serial transmissions using a directional antenna. The only devices that receive the packet are the five that are considered directional neighbors; the four neighbors that are not do not receive the data. If directional networks are not enabled, the device enqueues one packet. When the channel receives the packet to send, it will deliver it to all devices in range which includes the five directional neighbor devices as well as the four other nodes. This latter case is equivalent to a single transmission using an omnidirectional antenna. Figure 8 shows the difference

in LLSW when using the directional network feature to send a broadcast packet.

The list of directional neighbors is built by the user in the ns-3 scenario. The user needs to add nodes as directional neighbors in the scenario and if desiring a non-static set of directional neighbors the user must write the appropriate code to maintain the list. LLSW provides the methods to add and remove directional neighbors as well as the infrastructure to use the list once built but it does not provide an automatic building and maintenance of a directional neighbor list.

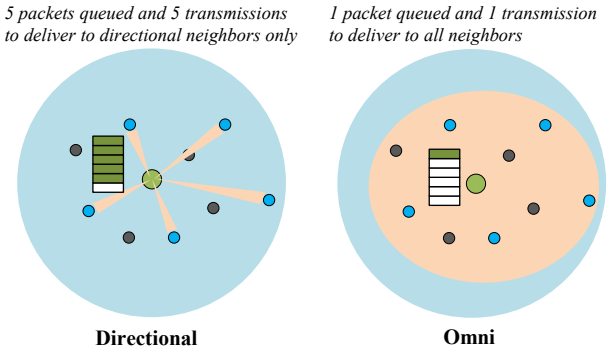


Figure 8. LLSimpleWireless Model Directional Network Example for Sending a Broadcast Packet.

3.6 PCAP Packet Capture

PCAP packet capture is a feature that exists on other ns-3 device models including WiFi and WiMax but was not part of the base SimpleWireless model. This packet sniffing was added to the LLSimpleWireless model to capture packets sent and received at the device. This allows the simple wireless model to be consistent with other ns-3 models which provide a packet capture capability.

4. EVALUATION

The new features of LLSW were evaluated for correct behavior and performance. This section describes the testing that was performed including a description of the scenarios used for testing, and test results.

4.1 Error Model Test Case

The scenario to test the Error Model had 101 nodes in total placed within a circle of radius 100. Node 0 was placed at the center of the circle and nodes 1 through 100 were randomly placed within the circle. Figure 9 shows the node placement. Node 0 sent broadcast traffic at a rate of 1Mbps using the ns-3 OnOffApplication [9]. There were no queues used so the bandwidth of the network was essentially infinite and did not hinder successful packet delivery. Thus the only reason a packet would not be received was packet error.

This scenario was executed using the Packet Error Rate (PER) Curve for the error model with the error curve comprised of the <distance, error rate pairs> shown in Table 1.

Table 1. Distance, Error Rate Pairs.

Distance	Error Rate
0	0%
10	0%
20	5%
30	7%
40	12%
50	15%
60	50%
70	60%
80	70%
90	80%
100	100%

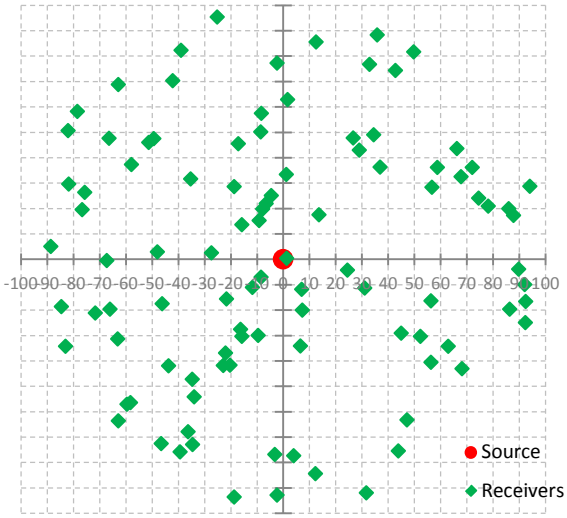


Figure 9. Error Model Test Scenario Node Placement.

Figure 10 shows the results of the simulation in terms of the % of packets dropped versus distance of the destination node from the source node. These results are overlaid on the PER curve specified per Table 1. As can be seen from the graph, the % of packets dropped based on distance matches the PER curve specified.

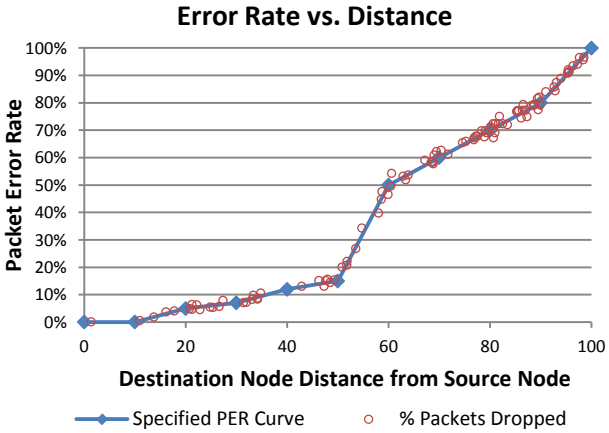


Figure 10. PER Curve Test Results: Measured packet errors correctly match the user specified PER curve.

The scenario was executed a second time but with a reduction in the number of nodes to 21 total and using the Stochastic error model with an average up time of 15 seconds and an average down time of 5 seconds. Figure 11 shows the results of the simulation in terms of packet reception. The x-axis is the simulation time and the y-axis is receiving node id. Each green mark represents when a node received a packet. There are gaps in the reception and these gaps correspond to when the channel between the source node (Node 0) and the destination node id (y-axis value) was disabled as part of the stochastic error. The average error was purposely set to a high value of 5 seconds so that gaps in reception could be seen on the graph. Note that each destination has a unique pattern for when the channel is up/down

which shows how this stochastic error is applied to each specific source/receiver pair.

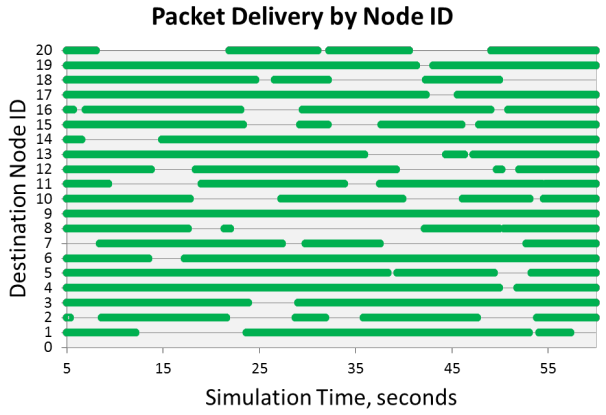


Figure 11. Stochastic Error Test Results: Each destination node has a unique random error pattern.

4.2 Queue Test Case

The scenario to test the queue implementation had two nodes. Node 0 sent data to Node 1 at a rate of 1Mbps using the OnOffApplication. Optimized Link State Routing (OLSR) [8][14] was used as the routing protocol which provided control packets. The data rate of the device was varied to show how the queues enforce a device data rate. The queue type was also varied to demonstrate how the queue latency is affected by the type of queue used. Figure 12 shows the scenario for the queue test.

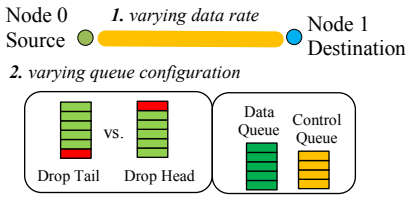


Figure 12. Queue Test Scenario.

Figure 13 shows the results of the simulations run over a range of device data rates from 0.1Mbps to 1.1Mbps using DropHead and DropTail queues. The graph shows two sets of data: percent packets received and queue latency. DropHead and DropTail queues have nearly identical packet delivery rates so the values from the DropHead and DropTail simulations were averaged to show just one line on the graph. As the data rate increases, the packet receive rate also increases, reaching 100% at 1.1Mbps, which is just above the OnOffApplication traffic rate of 1Mbps. At a device data rate of 1Mbps, 96% of the data packets are received instead of 100% because some of the bandwidth is needed for the OLSR overhead. The graph also shows the queue latency for DropHead and DropTail queues. As expected, the DropHead queue has a lower latency than DropTail because DropHead will drop the oldest packet (highest latency) when there is no room in the queue to enqueue a new packet while the DropTail will drop the newest packet (lowest latency). Dropping the newest packet means that the oldest packets remain in the queue and thus increases the queue latency.

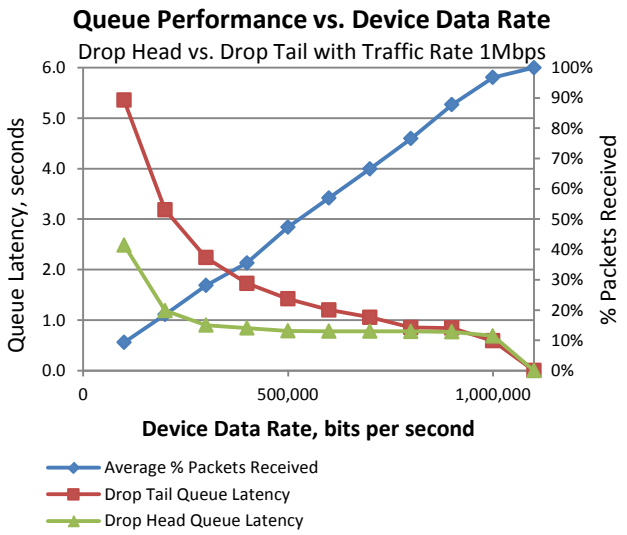


Figure 13. Queue Test Results for Drop Head vs Drop Tail: Delivery rate increases and queue latency decreases as device data rate increases.

The scenario was used for a second set of simulations to test the use of the priority queues to differentiate control and data traffic. Once again, the device data rate was varied from 0.1Mbps to 1.1Mbps and Node 0 sent 1.0Mbps of data traffic but using a PriorityTail queue configuration which separates control traffic and data traffic and only sends data traffic when the control sub-queue is empty. Figure 14 shows the results of the simulations. The graph shows two sets of data for control and data traffic: % packets received and queue latency. The graph shows that control packets are given preference over data packets. Control packets have a 100% reception rate and zero queue delay while the data packets have reception rate and queue delay that vary with the data rate. At a device data rate of 0.1Mbps, only 10% of the data packets are delivered and the queue latency is high.

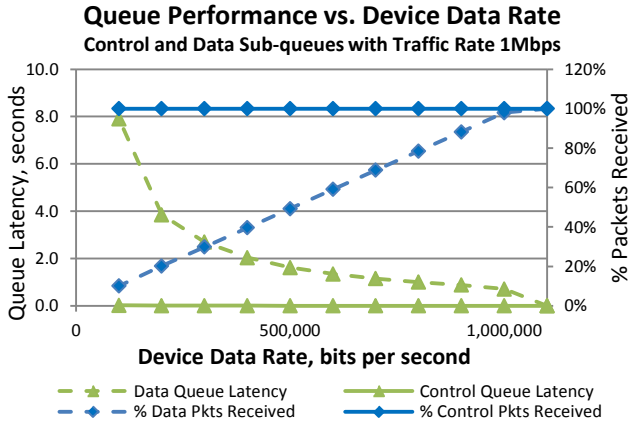


Figure 14. Queue Test Results for Control and Data Sub-queues: Control traffic is appropriately given priority over data traffic.

4.3 Fixed Contention Test Case

To test the fixed contention feature, a scenario similar to that used for testing error models was used. A total of 101 nodes were placed in a circle of radius 100. Node 0 was placed at the center

and the other 100 nodes were randomly placed. The node placement is identical to that shown in Figure 9. Node 0 sent broadcast traffic at a rate of 1Mbps using the OnOffApplication. A DropHead queue was used to limit the device data rate to 10Mbps so that the effect of contention could be observed. The contention range was varied so that the number of contention neighbors affecting Node 0's transmissions varied. Figure 15 shows the results. The graphs shows the percentage of packets received and the percentage of packets dropped at Node 0's queue due to queue overflow. Increasing the number of contention neighbors increases the transmission time of each packet. This in turn reduces the effective device data rate resulting in poor throughput and queue congestion as the number of contention neighbor increases.

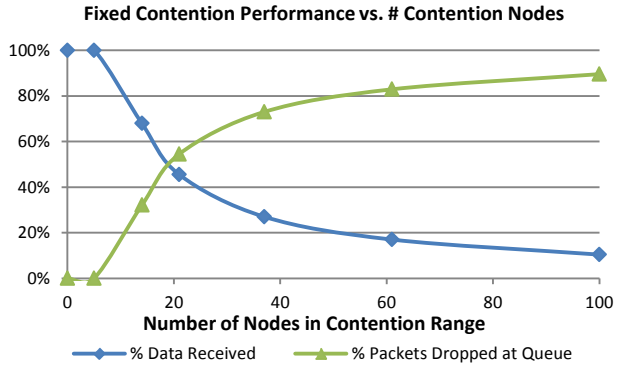


Figure 15. Fixed Contention Test Results: % Data received decreases and % packets dropped at queue increases as contention increases.

4.4 Directional Networking Test Case

The scenario to test the directional networking had 13 nodes. Node 0 was placed at the center of a circle of radius 50 and the remaining 12 nodes were placed on the perimeter of the circle in a clockwise fashion. Figure 16 shows the node placement. The scenario was configured such that Node 0 had 6 directional neighbors: Nodes 1, 3, 4, 7, 10, 11. All other nodes were in range of Node 0 but were not directional neighbors.

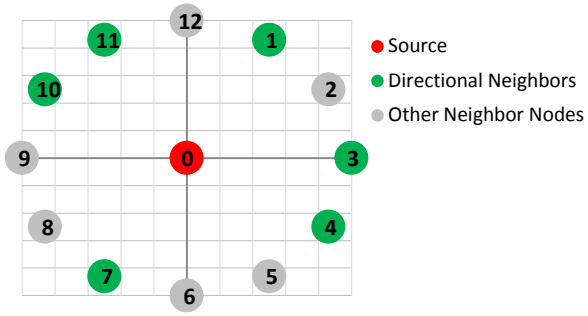


Figure 16. Directional Network Test Scenario Node Placement.

A DropHead queue was used with a device data rate of 10Mbps. In the first simulation run using this scenario, Node 0 was the source of broadcast traffic at a rate of 1Mbps using the OnOffApplication. As indicated previously, when a directional network is enabled, a broadcast packet is duplicated and enqueued once for each directional neighbor. Figure 17 shows the results of the simulation in terms of the number of packets in the queue versus simulation time for a small, 0.1 second interval. The time scale on this graph is small so that the dots on the graph can be seen. On a larger time scale, it is difficult to see the individual dots as they appear as a solid line due to their density. Each time a broadcast packet is sent, the queue occupancy peaks at five packets then decreases to zero as each packet is transmitted. The reason there are never six packets in the queue (the number of directional neighbors) is because the channel is idle when the first packet is sent and thus it is sent immediately and does not appear in the queue occupancy statistics. This graph shows how a unique packet is enqueued for each directional neighbor.

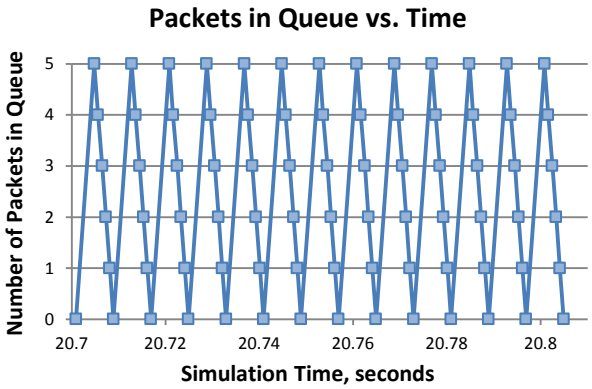


Figure 17. Directional Networking Queueing: Each packet sent correctly results in five packets enqueued.

Figure 18 shows the packet delivery statistics for the scenario with and without the directional network feature enabled. When the directional networking feature is not enabled, all nodes receive 100% of the broadcast traffic but when directional networking is enabled only those nodes that are designated as directional neighbors receive the broadcast traffic.

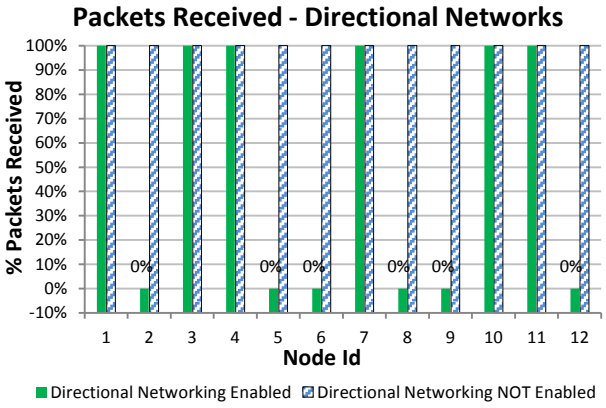


Figure 18. Directional Networking Packet Delivery: Only directional neighbors receive traffic when directional networking is enabled.

5. CONCLUSION

ns-3 provides several models that can be used to simulate wireless networks. However, for protocol research and development, the WiFi and WiMax models can be complex to configure and difficult to control in terms of packet delivery and performance. The optional SimpleWireless model overcomes this complexity but is a bit too simple and has limited features. The LLSimpleWireless model adds several important features to the base SimpleWireless model to provide a more realistic and yet

controlled wireless environment for ns-3 simulations. These additional features provide queues for enforcing a data rate, range based error modeling, transmission delay, support for simulating channel contention and directional networks, and PCAP packet capture. The goal of the effort is to provide wireless network protocol researchers a MAC/PHY model that enables tight controls of MAC/PHY effects to limit uncertainty when developing network protocols.

Though this new set of features is extensive, there are several enhancements that could be made as future work, specifically:

- Add full ErrorModel support on the send side. LLSW includes three types of error models on the send side. The random error that was added to LLSW only supports uniform distribution of errors applied to entire packets. This could be modified to support the ErrorModel class so that additional random distributions beside uniform can be used and the error rate can be applied at the bit or byte level in addition to packet level.
- Add support to the priority queue for user configurable number of sub-queues. LLSW includes support for queues but only supports the use of a single queue to hold all packets or a priority queue with two sub-queues for control and data packets. The priority queue implementation could be expanded to allow the user to define N sub-queues each with a PCAP string filter as well as a user specified de-queueing order.
- Add support for multiple radios for use in directional networks. LLSW assumes that there is just one radio and therefor the directional transmissions are serial. Support for a multiple radio directional system could be added to the model to allow for simultaneous directional transmissions.
- Add automatic support for directional networks including automatic neighbor detection for mobility. LLSW includes support for directional networking but the user must create and maintain the list of directional neighbors for each node. The simplest method is to statically configure this in the scenario. If mobility is used, then the user must write the appropriate code in their scenario to manage the list of directional neighbors. Support for creating and maintaining lists of directional neighbors based on current neighbors could be added to the model so that the user does not have to manually create and maintain the list of directional neighbors in their scenario.
- Add support for interference. LLSW includes the support for dropping packets based on an expected error rate at a given range with the implementation of the PER curve error model. This type of PER curve would be built from known or expected performance of the network which could but may not include the effects of packet interference. Support for discrete modeling of packet interference could be added to the model and could be based on existing interference models such as the YANS WiFiPhy model [4][13].
- Add support for distance based data rate. LLSW includes a user configurable data rate but this is a constant data rate. Support for a distance-based data rate could be added to the model for simulating adaptive coding techniques which lower the data rate as distance increases in order to improve bit error rates.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency. Approved for public release; distribution unlimited.

7. REFERENCES

[1] R. R. Choudhury, X. Yang, R. Ramanathan, and N. Vaidya. 2002. Using directional antennas for medium access control in ad hoc networks. In *Proceedings of the 8th annual international conference on Mobile computing and networking* (MobiCom '02). ACM, New York, NY, USA, 59-70.

[2] J. Farooq and T. Turletti. 2009. An IEEE 802.16 WiMAX module for the ns-3 simulator. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques* (Simutools '09). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, , Article 8, 11 pages.

[3] Y.-B. Ko, V. Shankarkumar, and N. Vaidya. 2000. Medium access control protocols using directional antennas in ad hoc networks. In *Proceedings Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2000). IEEE, New York, NY, USA, 13-21, Vol. 1.

[4] M. Lacage and T.R. Henderson. 2006. Yet another network simulator. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator* (WNS2 '06). ACM, New York, NY, USA, Article 12.

[5] LibPCAP, http://www.tcpdump.org.

[6] Network Simulator 3 (ns-3), http://www.nsnam.org.

[7] ns-3 ErrorModel Code, http://www.nsnam.org/doxygen/group__errormodel.html

[8] ns-3 OLSR Code, http://www.nsnam.org/doxygen/group__olsr.html

[9] ns-3 OnOffApplication Code, http://www.nsnam.org/doxygen/group__onoff.html

[10] ns-3 Simple Wireless, http://code.nsnam.org/tomh/ns-3-simple-wireless

[11] ns-3 Wifi Code, http://www.nsnam.org/docs/models/html/wifi-design.html

[12] ns-3 Wimax NetDevice, http://www.nsnam.org/docs/release/3.12/models/html/wimax.html

[13] ns-3 Yans WifiPhy Code, http://www.nsnam.org/doxygen/classns3_1_1_yans_wifi_phy.html

[14] Optimized Link State Routing, RFC 3626, https://www.ietf.org/rfc/rfc3626.txt

[15] M. Smith. 2005. Channel Characterization and Modeling for Satellite Communications on the Move. In *Proceedings of Military Communications Conference 2005* (MILCOM 2005). IEEE, New York, NY, USA, 821-827 Vol. 2.

A Realistic MAC and Energy Model for 802.15.4

Vishwesh Rege
Robert Bosch Centre for Cyber Physical Systems
Indian Institute of Science (IISc)
Bangalore, India
vishwesh.rege@cps.iisc.ernet.in

Tommaso Pecorella
Dpt. of Information Engineering (DINFO)
Università di Firenze
Firenze, Italy
tommaso.pecorella@unifi.it

ABSTRACT

The IEEE 802.15.4 standard defines the physical and media access control layers for LR-WPANs (Low-Rate Wireless Personal Area Networks), and is one of the enabling technologies for Wireless Sensor Networks (WSNs) as well as the emerging Internet of Things (IoT) applications. The ns-3 network simulator offers support for simulating LR-WPANs as specified by the IEEE standard 802.15.4 (2006). However, only the ad-hoc mode is currently supported and many important features of the MAC such as radio duty cycle management are missing from the implementation.

Moreover, at the moment ns-3 does not support simulating the energy consumption of LR-WPAN devices. Since energy efficiency is an important consideration for WSN and IoT applications, support for accurate energy modeling is highly desirable in order to develop energy-aware protocols for such applications. In this paper, we present the models developed for simulating the energy consumption of nodes in LR-WPANs. Further we implement the ContikiMAC radio duty cycling protocol in order to provide a realistic 802.15.4 compliant MAC layer which supports sleep/wake mechanisms.

CCS Concepts

• **Computing methodologies** → **Model development and analysis;** Discrete-event simulation; **Networks** → *Network simulations*

Keywords

ns-3, energy consumption, IEEE 802.15.4, IoT, WSN, MAC, duty cycling

1. INTRODUCTION

Wireless sensor networks (WSNs) consist of a large number of battery-powered wireless sensor nodes required to operate for years without any human intervention. The Internet of Things (IoT) is the future of wireless sensor network applications where any device can be connected to and operated through the Internet. Since the wireless network nodes in WSN and IoT applications are typically battery-powered, they have a limited amount of energy available. As a result, efficient utilization of the available energy is essential for continuous operation over long periods of time. Wireless sensor network research attempts to design network protocols to meet these energy constraints and increase network lifetime.

IEEE 802.15.4 is an important standard for wireless sensor network and Internet of Things applications. The standard specifies the physical layer and media access control for low-rate wireless personal area networks (LR-WPANs), and is the underlying protocol used in a majority of sensor network deployments with applications such as remote sensing, surveillance and monitoring. The standard provides an energy efficient and cost effective option for low latency and high accuracy communication required in these scenarios, with the ability to survive on battery or harvested power for extended periods of time.

A number of custom and industry standard networking protocols have been developed using the services provided by 802.15.4. It is the basis for standards like ZigBee, WirelessHART, and the IPv6 adaptation protocol 6LoWPAN [4] which further extend it by specifying the upper layers which are not defined. When combined with 6LoWPAN, it can be used alongside the standard Internet protocols to build a wireless embedded Internet and facilitate the Internet of Things vision.

The 802.15.4 standard is the basis for the LRWPAN model in ns-3. The model allows simulation of WSN scenarios, though it is severely limited in its current capabilities. There is no support for simulating the energy consumption of LRWPAN nodes in the model. Since, energy efficiency is one of the major concerns for researchers looking to test their WSN protocols, this is a serious limitation that needs to be addressed.

Furthermore, the MAC (Media Access Control) layer doesn't incorporate sleep/wake mechanisms. As a result, the radio is always ON, which doesn't represent the actual deployment situation. Most WSN nodes when deployed, use some sort of radio duty cycling mechanism in order to turn off the radio when not in use. As the radio is the main source of energy consumption on a node, this is one of the primary methods used to increase energy efficiency and consequently the network lifetime. Lack of radio duty cycling therefore prevents users from running realistic simulations. On one side this increases the performance (delay, throughput, etc.), and on the other side overestimates the energy consumption for receiving packets and underestimates the transmission energy.

WNS3, June 15 - 16, 2016, Seattle, WA, USA

ACM 978-1-4503-4216-2/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2915371.2915379

A simulator model that supports modeling energy consumption of 802.15.4 devices will allow studying the energy consumption of nodes and thus the energy efficiency and network lifetime of 802.15.4 based protocols. A MAC model that incorporates radio duty cycling will allow researchers to test their network protocols under realistic conditions. Thus the developed models are expected to greatly simplify the future development and evaluation of protocols for WSN/IoT applications.

In this paper, we present our contributions to ns-3 which are as follows:

- PHY layer support for modeling energy consumption of 802.15.4 radios
- LRWPAN device energy model that represents the energy consumption of 802.15.4 devices
- A MAC model that incorporates radio duty cycling, in order to realistically simulate WSN scenarios in ns-3

The rest of the paper is organized as follows: Section 2 gives an overview of the IEEE 802.15.4 standard and the existing implementation in ns-3. Section 3 presents the in detail design and implementation of the proposed ns-3 energy and MAC models. Section 4 includes the validation of the developed models. Section 5 discusses possible future work and Section 6 concludes the paper.

2. IEEE 802.15.4 AND LRWPAN SIMULATION MODEL IN NS-3

In this section, we give a brief overview of IEEE 802.15.4 and introduce the relevant features of the standard. We also discuss the LRWPAN simulation model in ns-3 which is based on the standard.

2.1 PHY Layer

The 802.15.4 PHY layer provides two services:

1. The PHY data service, which enables the transmission and reception of PHY protocol data units (PPDUs) across the physical medium.
2. The PHY management service interfacing to the physical layer management entity (PLME) service access point (SAP) (known as PLME-SAP).

The features of the 802.15.4 PHY are activation and deactivation of the radio transceiver, transmission and reception of packets across the wireless channel, as well as performing additional tasks that may be required by the upper layers, such as Energy Detection (ED), Clear Channel Assessment (CCA) and Link Quality Indicator (LQI) measurement for received packets. ED and CCA operations are required in the Carrier Sense Multiple Access Collision Avoidance (CSMA-CA) functionality of the MAC layer.

In ns-3, the Phy model is based on the specification described in section 6 of IEEE Std 802.15.4-2006. It models the PHY service specifications, PPDU formats, PHY constants and PIB attributes.

2.2 MAC Layer

The medium access control (MAC) layer enables transmission of frames through the physical channel.

The 802.15.4 MAC layer provides two services:

1. The MAC data service, which enables the transmission and reception of MAC protocol data units (MPDUs) across the PHY data service.
2. The MAC management service interfacing to the MAC layer management entity (MLME) service access point (SAP) (known as MLME-SAP).

The features of the 802.15.4 MAC layer are beacon management, channel access, GTS management, frame validation, acknowledged frame delivery, association, and disassociation.

The MAC model in ns-3 is based on the specification described in section 7 of IEEE Std 802.15.4-2006. The standard allows three different MAC modes: beacon-enabled, non beacon-enabled (star and cluster tree modes) and beaconless (ad-hoc mode). However, the current MAC implementation lacks some features foreseen by the standard. In particular, there is no active support for coordinators, association, disassociation and beacon management, required for beacon-enabled and non beacon-enabled modes of operation. Only channel access (using unslotted CSMA/CA) and the basic data transfer API along with acknowledged frame delivery is currently implemented. Thus, the only mode supported is ad-hoc mode (i.e., beaconless).

Further, the current ns-3 MAC does not use low power features, forcing the radio always in an ON state. As a result, it is impossible to simulate realistic scenarios in which the nodes are assumed to be duty-cycled spending most of the time with their transceivers switched off. It must be stressed that the standard specifies only the sleep management for beacon-enabled MAC, while for the beaconless case the standard does not give any strict rule.

Due to the lack of beacons and centralized timing in the beaconless case, the main challenge is to achieve a local synchronization between two different nodes, because the local clocks drift does not allow to keep the synchronization if not for small periods. As a consequence, it is not guaranteed to know when the receiver node will be awake.

In order to save energy in beaconless mode, many alternative 802.15.4 compliant MAC protocols such as S-MAC, B-MAC, X-MAC, WiseMAC [9] have been proposed in the literature. Any of these duty cycling schemes can be used in optimizing the radio power consumption as required by the application. In our proposed model, we implement ContikiMAC [10], which is the default radio duty cycling mechanism in the Contiki OS [11].

2.3 ns-3 Energy Framework

The current ns-3 simulator (since ns-3.9) provides a basic framework for modeling energy consumption in wireless networks [5]. The model consists of 2 major components:

1. The Energy source, and
2. The Device energy model

The Energy Source represents the energy supply of a node in the simulation, while the Device energy model is used to represent the energy consumption characteristics of a specific device, such as an 802.11 radio on a node.

The energy framework focuses on modeling radio energy consumption because the radio is assumed to consume the most power in a wireless node. The framework also assumes a state-based model, i.e., the radio is assumed to be in one of several states

(Receive, Transmit, Idle, etc.) with a corresponding load current associated with each state. The energy model and energy source attached to each node keep track of the time spent in a particular state and the total energy consumed by the node respectively.

The most difficult part in using the energy model is to define the device states and the transitions, in order to mimic the real device energy characteristics.

3. DESIGN AND IMPLEMENTATION

3.1 LRWPAN Energy Model

The LRWPAN energy model extends the basic energy framework in order to model energy consumption of LRWPAN nodes. As noted in the previous section, it follows a state based approach, modeling the nodes' energy consumption as a function of the states of the radio transceiver. We base our model on the AT86RF231 [14], which is a typical 802.15.4 compliant radio transceiver.

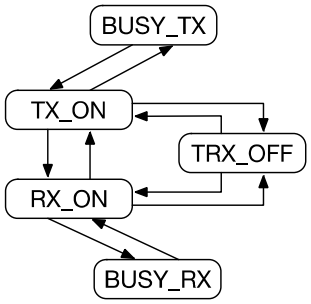


Figure 1: State transition diagram.

According to the model, the radio can be in one of the following states:

- **TRX_OFF** - Transceiver disabled
- **TX_ON** - Transmitter enabled
- **RX_ON** - Receiver enabled
- **BUSY_TX** - Transmitting
- **BUSY_RX** – Receiving

However, there is no difference between the (TX/RX) ON and corresponding (TX/RX) BUSY states with respect to the radio transceiver circuitry. The energy consumed in these states is approximately the same. Hence, the node's energy consumption can be modeled as a function of only three states TRX_OFF, TX_ON and RX_ON.

Most transceiver chips also have an additional shutdown (sleep in case of AT89RF231) state in which the chip is completely deactivated. This state can only be entered through external interrupts and isn't modeled. The rest of the states can be entered by sending appropriate commands to the PHY layer through the PLME. BUSY_RX and BUSY_TX are entered automatically during frame reception and transmission respectively. Figure 1 illustrates the corresponding radio transceiver states and the allowed transitions between them. All the transitions are triggered by MAC events, except for the BUSY states, which correspond to actual PHY layer events (e.g. preamble detection causes the transition from RX_ON to BUSY_RX).

A PHY Listener is registered with the LrWpan PHY in order to notify concerned objects of every PHY state transition.

Energy update algorithm:

The PHY, on each successful transition notifies the Energy model through the PHY Listener. Correspondingly, the LrWpan-RadioEnergyModel notifies the EnergySource object to update its energy. The EnergySource object queries the energy model for the current draw of the state and uses that to calculate the energy consumed using the formula:

$$stateDuration \times stateCurrent \times supplyVoltage$$

where, *stateDuration* represents the time spent in the state, *stateCurrent* represents the state's current draw, and *supplyVoltage* is the node's attached energy source's supply voltage.

The energy source is then updated with the new value of remaining energy. When the energy is completely depleted, the LrWpanRadioEnergyModel is informed by the EnergySource, using the EnergyDepletionCallback (defined in LrWpanRadio-EnergyModel).

The node's attached energy source is also updated periodically to keep track of the energy consumed even when there are no radio state transitions.

In most cases, the state transitions in radio transceivers aren't immediate. There is a finite time difference between the PHY receiving a request to change the state, and then issuing a confirm primitive indicating that the state change is accepted. This transceiver switching time not only affects the MAC operations, but also has a significant impact on the total energy consumption as a result of the very low radio duty cycle in wireless sensor networks [7]. Hence, it is important to also take into account the transition time and energy between the states. In the Three States Model described, six different transitions are possible:

- TRX_OFF -> TX_ON and vice versa.
- TRX_OFF -> RX_ON and vice versa.
- TX_ON -> RX_ON and vice versa.

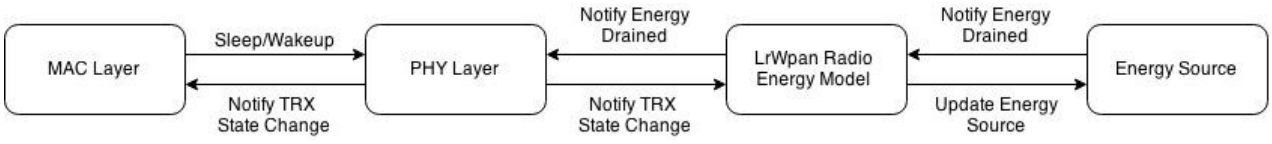


Figure 2: Energy model flow diagram.

Table 1: State transition timing.

Transition	Transition Time
TRX_OFF → TX_ON	110 us
TX_ON → TRX_OFF	Immediate
TRX_OFF → RX_ON	110 us
RX_ON → TRX_OFF	Immediate
TX_ON → RX_ON	8/12 symbol periods* (128/192 us)
RX_ON → TX_ON	8/12 symbol periods* (128/192 us)

* defined by the standard

The transition times have been modeled according to the values given in the datasheet. Table 1 shows the transition timings of the AT89RF231 radio transceiver.

When a state change is requested by the MAC through the PLME, the state machine is put in the transition state. On receiving confirmation from the PHY that the target state is operative, the state machine is finally updated with the new state. The state transition energy is calculated by multiplying the transition time by the power in the target state. This is a worst-case assumption since the transition energy has been shown to depend on the power in the state the transceiver is switching *to*, multiplied by a parametric constant (less than 1) [8].

As noted before, the current *LrWpanEnergyModel* is modeled based on Atmel's AT89RF231 transceiver. However, other 802.15.4 compliant transceivers such as TI CC2420 [15] can be simulated by changing the current draw to appropriate values. For the CC2420, TRX_OFF, TX_ON and RX_ON correspond to the chip's Idle, Transmit and Receive states respectively.

The PHY enumeration values defined in *LrWpanPhyEnumeration* are used to represent the different states of the radio transceiver, with IEEE_802_15_4_PHY_UNSPECIFIED being used for the TRANSITION state.

3.2 LRWPAN ContikiMAC Model

ContikiMAC is the default duty cycling mechanism in the Contiki OS. It is based on the concept of periodic wake-ups in which nodes sleep most of the time and wake up at regular intervals to check for radio activity. It uses an asynchronous sender-initiated radio duty cycling mechanism, i.e. it doesn't depend on any a priori synchronization between nodes, and the sender initiates communication by repeatedly sending the same packet until a link layer acknowledgment is received. It is simple to implement, since no signaling messages or additional packet headers are used. All

ContikiMAC packets are ordinary link layer messages already being used by the existing 802.15.4 standard.

The primary features of ContikiMAC can be summarized as follows:

- *Periodic sleep/wake mechanism* defined by the channel check rate.
- *Stable wake-up interval* unaffected by node's radio activity.
- *Two consecutive CCAs* after wake-up to detect packet - stay awake to receive the packet if energy is detected on the channel, else sleep if channel is found idle.
- *Link layer acknowledgment* followed by sleep on packet reception (transmitter repeatedly sends the frame until an ACK is received, after which it too returns to sleep).
- *Broadcast*: Continuous transmission of the data packet by the transmitter for full length of sleep period – since the transmission must wake up and deliver the packet to all the node's neighbors.
- *Fast sleep optimization* – To optimize the power at the receiver side by letting potential receivers go to sleep earlier if the CCA caused a node to stay awake due to spurious radio noise instead of an actual packet transmission.
- *Transmission Phase-Lock* - sender side optimization to reduce the power at the transmitter by minimizing the number of retransmissions.

Note that the retransmissions by the ContikiMAC Radio Duty Cycle (RDC) is independent of the MAC layer retransmissions which refers to the number of times a frame is given to the RDC layer for transmission.

The MAC and RDC mechanisms in the ContikiMAC model could be split into two separate classes, like in Contiki OS. However, in order to support multiple MAC protocols and reuse the code, the ns-3 implementation follows a different approach.

The first choice has been to split the MAC functionalities into a base, abstract class (LrWpanMac), implementing only the APIs defined by the 802.15.4 standard and some common elements. The particular MAC behavior is implemented in two subclasses: LrWpanNullMac and LrWpanContikiMac. LrWpanNullMac implements the current behavior, as is the always-on radio with no sleep or duty cycle. The LrWpanContikiMac is responsible for duty cycle, sleep wake, and node synchronization. All ContikiMAC features, with the exception of the Fast sleep optimization, have been implemented and tested in the proposed model. Both MAC layers use the LrWpanCsmaCa module to detect any activity on the channel.

It should be remarked that the new modular architecture enables an easy integration of new MAC protocols. This is an important feature, as the future IoT systems could be hindered by the use of 802.15.4-compliant (but mutually incompatible) protocols. As a matter of fact, even if two protocols are formally following the standard, different radio duty cycle management policies can make them very inefficient, leading to high energy consumption.

The LrWpanContikiMac API is identical to LrWpanMac. Additionally, the user can specify attributes such as the sleep interval, the interval between the two CCAs, the interval between each packet retransmission, and the maximum number of retries by the RDC.

4. VALIDATION AND RESULTS

In this section, validation results of the proposed LRWPAN energy and MAC models are presented. First, the energy consumed by the LRWPAN energy model is compared against expected values obtained through manual calculations. Next, in order to evaluate the ContikiMAC model, the energy consumption values of simulated nodes are compared with those obtained from running the same scenario in the Cooja simulator.

Energy model evaluation

In order to verify the correctness of the energy model, a simple scenario involving 2 nodes is considered. Node 1 sends a packet of size 50 bytes to Node 2 and receives an acknowledgement. The

simulation is run for 10s with both the nodes having their radios turned on during the first 5s and turned off for the next 5s.

The time spent by both the nodes in each state is shown in Table 2. The currents in TX, RX and OFF states are taken to be 19.5mA, 21.8mA and 1800µA respectively. For a supply voltage of 3.3V, the energy consumption comes out to be 0.3894 Joules for Node 1 and 0.3895 Joules for Node 2 which agrees with the simulation results.

Table 2: State timings for simulation scenario.

States	Node 1 (sender)	Node 2 (receiver)
TX_ON/BUSY_TX	0.002400	0.000544
*to TX_ON**	0.000192	0.000192
RX_ON/BUSY_RX	4.997106	4.998962
*to RX_ON**	0.000302	0.000302
TRX_OFF	5.000000	5.000000
*to TRX_OFF**	0.000000	0.000000

* from any arbitrary state

ns-3 vs. Cooja Comparison

Cooja [12] is a sensor network simulator for the Contiki OS. The Cooja energy estimation mechanism consists of keeping track of the time the radio spends in the different transceiver states and multiplying these times with corresponding power levels to obtain a rough estimate of the energy consumed.

Cooja also separately keeps track of the CPU power consumption. Apart from the regular CPU mode used during communication, the nodes can also operate in a Low Power Mode (LPM) during periods of inactivity. The total consumed energy is then calculated as the sum of the total time spent in all the states – similar to how the energy is calculated in our developed model in ns-3. The state transition energy however isn't taken into account. Thus the energy consumed by a node at the end of the simulation is given by:

$$E = PrxTrx + PtxTtx + PslpTslp + PlpmTlpm = IrxVrxTrx + ItxVtxTtx + IslpVslpTslp + IlpmVlpmTlpm$$

where *rx*, *tx*, *slp* and *lpm* refer to the transmit, receive, sleep and LPM states respectively.

The amount of time spent in each state can be tracked using the Powertrace [13] tool provided with Cooja. Powertrace outputs four values: *cpu*, *lpm*, *transmit*, and *listen*. These values correspond to the time spent in each of the four states. *transmit* and *listen* correspond to the "transmitter enabled" and "receiver enabled" states in our model respectively. The *cpu* state includes the time spent in *transmit* and *listen* states as well as the "transceiver off" state. Therefore, the time when TRX is off, but CPU is in its normal mode is given by *cpu* – (*transmit* + *listen*). *lpm* refers to the state when TRX is off and the node is in its low power mode. Since, our model only takes into account energy consumed by the transceiver, we consider *cpu* with transceiver off and *lpm* as one state by using the same current draw for both states.

For the comparison, we consider a two node scenario with Node 1 sending a packet to Node 2 periodically every 2.5 seconds over a 60 second interval. The channel check rate is set to be 8 Hz, which corresponds to a sleep interval of 125 ms. The energy consumption of both the sender and receiver nodes using the original ContikiMAC implementation and the ns-3 ContikiMAC implementation is compared in Table 3, while Table 4 compares the respective NullMAC implementations.

Table 3: ContikiMAC energy comparison (in Joules).

Protocol	Contiki OS ContikiMAC	ns-3 ContikiMAC
Sender	0.47945	0.44100
Receiver	0.37694	0.38750

Table 4: NullMAC energy comparison (in Joules).

Protocol	Contiki OS NullMAC	ns-3 NullMAC
Sender	4.04262	4.28037
Receiver	4.17223	4.28071

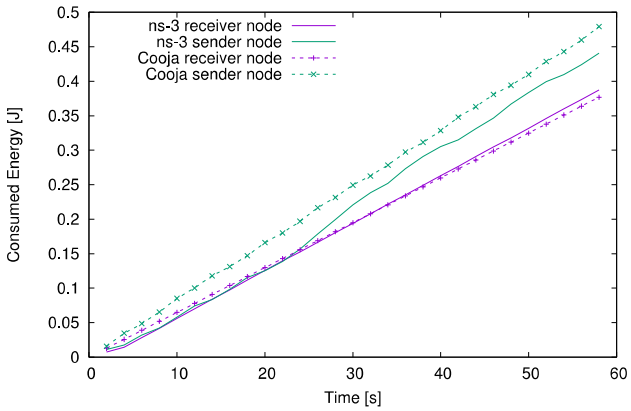


Figure 3: ContikiMAC energy comparison.

In Figure 3 it is evident that the energy consumed by the receiver nodes in both the implementations is in close agreement, while there is a slight disagreement in the energy consumption values of the sender nodes. This can be attributed to the difference in time both the nodes spend in transmit mode which varies due to the CSMA-CA random backoff algorithm.

5. FUTURE WORK

In this section, we discuss possible extensions to the proposed models.

Local Clocks Drift Simulation

One major improvement toward the model precision is to be done as part of a different ns-3 development: per-node local clocks. LR-WPAN energy efficiency is highly dependent on how nodes can keep (or lose) a mutual synchronization. But ns-3 clock is global, preventing a real evaluation of clock drift effects.

CPU Energy Consumption

Currently, the LRWPAN energy model only considers energy consumed by the radio transceiver and doesn't take into account the energy consumption of the CPU. As described in [5], states representing computation tasks available on the node can be used to incorporate CPU energy consumption information into the simulations.

Another possible development goal is to upgrade the current model to 802.15.4-2011 standard.

6. CONCLUSION

In this paper, we presented an energy model for 802.15.4 radio transceivers, which will enable users to measure the energy consumption of 802.15.4 nodes in the network and thus allow them to develop energy-efficient protocols for WSNs using this information. We also developed a modular and easily extensible MAC model and implemented the ContikiMAC radio duty cycling protocol in order to enable realistic simulations of WSN scenarios. The code is currently under review to be merged into ns-3.

7. ACKNOWLEDGMENT

This work was initiated as a project in the 2015 Google Summer of Code (GSoC) program, funded by Google.

8. REFERENCES

[1] GSOC2015LrWpanMac. 2015. http://www.nsnam.org/wiki/GSOC2015LrWpanMac

[2] ns-3 lr-wpan. http://www.nsnam.org/wiki/index.php/Lr-wpan

[3] IEEE Std 802.15.4-2006, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).

[4] J. Hui and P. Thubert. Compression format for IPv6 datagrams over IEEE 802.15.4-Based Networks. RFC6282, IETF Sep. 2011.

[5] H. Wu, S. Nabar, and R. Poovendran. 2011. An energy framework for the network simulator 3 (ns-3). In Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools '11). Brussels, Belgium, March 2011.

[6] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. 2007. Software-based on-line energy estimation for sensor nodes. In Proceedings of the 4th workshop on Embedded networked sensors (EmNets '07). New York, USA, June 2007.

[7] B. Bougard, F. Catthoor, D.C. Daly, A. Chandrakasan, W. Dehaene. 2005. Energy efficiency of the IEEE 802.15.4 standard in dense wireless microsensor networks: modeling and improvement perspectives. In Proceedings of the conference on Design, Automation, and Test in Europe (DATE '05). Munich, Germany, March 2005.

[8] P. Hurni, B. Nyffenegger, T. Braun, and A. Hergenroeder. 2011. On the accuracy of software-based energy estimation techniques. In Proceedings of the 8th European conference on Wireless sensor networks (EWSN'11). Bonn, Germany, February 2011.

[9] A. Bachir, M. Dohler, T. Watteyne, and K. K. Leung. 2010. MAC Essentials for Wireless Sensor Networks. *Commun. Surveys Tuts.*

[10] A. Dunkels. The ContikiMAC radio duty cycling protocol. Technical Report T2011:13 , Swedish Institute of Computer Science Decembers 2011.

[11] A. Dunkels, B. Gronvall, and T. Voigt. 2004. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN '04). Tampa, USA, November 2004.

[12] F. Osterlind. A sensor network simulator for the Contiki OS. Swedish Institute of Computer Science (SICS), Tech. Rep. T2006-05, Feb. 2006.

[13] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes. Powertrace: Network-level power profiling for low-power wireless networks. Technical Report T2011:05, Swedish Institute of Computer Science, Mar. 2011.

[14] AT86RF231 Datasheet. http://www.atmel.com/images/doc8111.pdf

[15] CC2420 Datasheet. http://www.ti.com/lit/ds/symlink/cc2420.pdf

A Framework for End-to-End Evaluation of 5G mmWave Cellular Networks in ns-3

Russell Ford, Menglei Zhang, Sourjya Dutta
Marco Mezzavilla, Sundeep Rangan
New York University
Brooklyn, New York, USA
{russell.ford,menglei,sdutta}@nyu.edu
{mezzavilla,srangan}@nyu.edu

Michele Zorzi
University of Padova
Padova, Italy
zorzi@dei.unipd.it

ABSTRACT

The growing demand for ubiquitous mobile data services along with the scarcity of spectrum in the sub-6 GHz bands has given rise to the recent interest in developing wireless systems that can exploit the large amount of spectrum available in the millimeter wave (mmWave) frequency range. Due to its potential for multi-gigabit and ultra-low latency links, mmWave technology is expected to play a central role in 5th Generation (5G) cellular networks. Overcoming the poor radio propagation and sensitivity to blockages at higher frequencies presents major challenges, which is why much of the current research is focused at the physical layer. However, innovations will be required at all layers of the protocol stack to effectively utilize the large air link capacity and provide the end-to-end performance required by future networks.

Discrete-event network simulation will be an invaluable tool for researchers to evaluate novel 5G protocols and systems from an end-to-end perspective. In this work, we present the first-of-its-kind, open-source framework for modeling mmWave cellular networks in the ns-3 simulator. Channel models are provided along with a configurable physical and MAC-layer implementation, which can be interfaced with the higher-layer protocols and core network model from the ns-3 LTE module to simulate end-to-end connectivity. The framework is demonstrated through several example simulations showing the performance of our custom mmWave stack.

CCS Concepts

•**Networks** → **Network performance evaluation; Network simulations; Mobile networks;** •**General and reference** → *Evaluation; Design; Performance;*

Keywords

mmWave; 5G; Cellular; Channel; Propagation; PHY; MAC.

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915380

1. INTRODUCTION

Millimeter Wave (mmWave) communications promise to be highly disruptive for both cellular and wireless LAN technologies due to the potential for multi-gigabit wireless links, which make use of the gigahertz of contiguous bandwidth available at mmWave frequencies in combination with high-dimension antenna arrays for high-gain directional transmission. Although the mmWave channel is known to suffer from poor high-frequency propagation loss, advances in physical layer technology such as adaptive smart antennas, along with recent work on channel measurements and modeling, have paved the way for achieving sufficient range and coverage in these networks [1, 2]. Nevertheless, before mmWave technology can be effectively realized in 5G cellular networks, there are numerous challenges to be addressed, not only at the physical layer, but at higher layers of the radio stack and in the core network as well. For instance, the extreme susceptibility of mmWave links to shadowing from blockages will require frequent, near instantaneous handovers between neighboring cells, fast link adaptation, and a TCP congestion control algorithm that can utilize the large capacity when available but adapt quickly to rapid channel fluctuations to avoid congestion. Therefore, the constraints and characteristics of the mmWave physical layer will require novel solutions throughout the 5G network and across all layers of the stack.

Discrete-event network simulators have, for long, been one of the most powerful tools available to researchers for developing new protocols and simulating complex networks. The ns-3 network simulator [3] currently implements a wide range of protocols in C++, making it especially useful for cross-layer design and analysis.

In this work, we present the current state of the millimeter wave module for ns-3, first introduced in [4], which can now be easily interfaced with the LTE LENA module [5] radio stack and core network in order to evaluate cross-layer and end-to-end performance of 5G mmWave networks. We provide an overview of the module and discuss a number of enhancements and added features since the first version, such as improved statistical channel model derived from 28 GHz channel measurements as well as a new ray tracing-based model. Custom implementations of an "LTE-like" Physical (PHY) and Medium Access Control (MAC) layer are also provided, which follow the LENA module architecture. The PHY and MAC classes are parameterized and highly customizable in order to be flexible enough for testing different

designs and numerologies without major modifications to the source code.

The rest of the paper is organized as follows. In Section 2, we introduce the overall architecture of the mmWave module framework. We then take a closer look at each component, starting with the PHY layer and channel models in Sections 3. Section 4 follows with a discussion on the MAC layer, which includes several scheduler classes as well as support for Adaptive Modulation and Coding (AMC) and Hybrid Automatic Repeat Request (HARQ). In Section 5, to demonstrate how the framework can be used for cross-layer and end-to-end evaluation, we provide some example simulations showing (i) the capacity of a TDMA mmWave cell with multiple users and (ii) the performance of TCP for a single user under varying channel conditions. Finally, we discuss future work and conclude the paper in Section 6.

2. MMWAVE FRAMEWORK OVERVIEW

Presently, the ns-3 mmWave module is targeted for simulating LTE-style cellular networks and is based heavily on the architecture and design patterns of the LTE LENA module. The main enhancement introduced in this latest version of the module is the implementation of Service Access Points (SAPs) for interfacing with the existing LENA classes, which enables mmWave classes to leverage the robust suite of LTE/EPC protocols the LENA module provides.

In Figure 1, we show the high-level composition of the `MmWaveEnbNetDevice` and `MmWaveUeNetDevice` classes, which respectively represent the mmWave eNodeB (eNB) base station and User Equipment (UE) radio stacks. A more detailed UML class diagram is given in Figure 1 of [4] and details on each layer will be given in their respective sections. The `MmWaveEnbMac` and `MmWaveUeMac` MAC layer classes implement the LTE module Service Access Point (SAP) *provider* and *user* interfaces, which allow them to interoperate with the LTE Radio Link Control (RLC) layer. Support for RLC Saturation Mode (SM), Unacknowleged Mode (UM) and Acknowledged Mode (AM) is built into the MAC and scheduler classes (i.e., `MmWaveMacScheduler` and derived classes). The MAC scheduler also implements a SAP for configuration by the LTE RRC layer (`LteEnbRrc`). Therefore, all the components required for Evolved Packet Core (EPC) connectivity are available.

The `MmWavePhy` classes handle directional transmission and reception of the downlink (DL) and uplink (UL) data and control channels based on control messages from the MAC layer. Similar to the LTE module, each PHY instance communicates over the channel (i.e., `SpectrumChannel`) via an instance of the `MmWaveSpectrumPhy` class, which is shared for both DL and UL (instead of separating such objects as in LTE LENA). Instances of `MmWaveSpectrumPhy` encapsulate all PHY-layer models including those for interference calculation
(`MmWaveInterference`), Signal to Interference and Noise Ratio (SINR) calculation (`MmWaveSinrChunkProcessor`), the Mutual Information (MI)-based error model (`MmWaveMiErrorModel`), which computes packet error probability, as well as the Hybrid ARQ PHY-layer entity (`MmWaveHarqPhy`) for performing soft combining.

A more detailed exposition of the procedures and interactions of these classes is given in [4]. Since the structure, high-level functions and naming scheme of each class closely follows the LTE LENA module, the reader is also referred to the LENA project documentation [6] for more information.

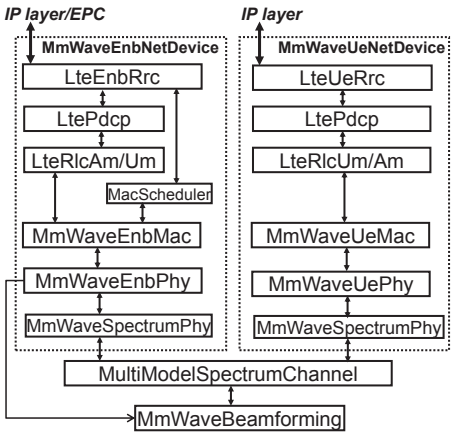


Figure 1: Simplified class diagram for the mmWave module.

3. PHYSICAL LAYER

In this section, we discuss the key features of the mmWave PHY layer. Specifically, we have implemented a TDD frame and subframe structure, which has similarities to TD-LTE, but allows for more flexible allocation and placement of control and data channels within the subframe and is suitable for the *variable Transmission Time Interval (TTI)* MAC scheme described in Section 4. Significant improvements have also been made to the channel model since the version introduced in [4]. The 28 GHz statistical path loss model can now be combined with the *building obstacle* model to simulate a realistic shadowing environment. A ray tracing-based path loss and fading model, which makes use of paths generated by third-party ray tracing software, has also been added. Additionally, we have modified the LENA error model and Hybrid ARQ model to be compatible with our custom mmWave PHY and numerology (for instance, to support larger TB and codeword sizes as well as multi-process stop-and-wait HARQ for both DL and UL).

3.1 Frame Structure

It is widely agreed that 5G mmWave systems will target Time Division Duplex (TDD) operation because it offers improved utilization of wider bandwidths and the opportunity to take advantage of channel reciprocity for channel estimation [1, 7, 8]. In addition, shorter symbol periods and/or slot lengths have been proposed in order to reduce radio link latency [9, 10]. The ns-3 mmWave module therefore implements a TDD frame structure which is designed to be configurable and supports short slots so as to be useful for evaluating different potential designs and numerologies. These parameters, shown in Table 1, are accessible through the common `MmwavePhyMacCommon` class, which stores all user-defined configuration parameters used by the PHY and MAC classes.

The frame and subframe structures share some similarities with LTE in that each frame is subdivided into a number of subframes of fixed length. However, in this case, the user is allowed to specify the subframe length in multiples of Or-

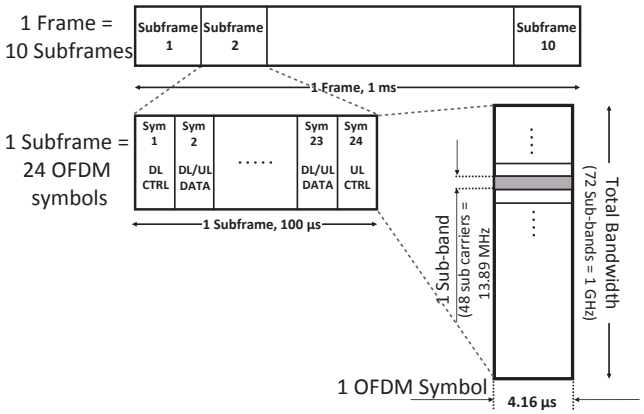


Figure 2: Proposed mmWave frame structure.

thogonal Frequency-Division Multiplexing (OFDM) symbols.[1] Within each subframe, a variable number of symbols can be assigned by the MAC scheduler and designated for either control or data channel transmission. The MAC entity therefore has full control over multiplexing of physical channels within the subframe, as discussed in Section 4. Furthermore, each variable-length time-domain data slot can be allocated by the scheduler to different users for either DL or UL.

Figure 2 shows an example of the frame structure with the numerology taken from our proposed design in [10]. Each frame of length 1 ms is split in time into 10 subframes, each of duration 100 μs, representing 24 symbols of approximately 4.16 μs in length. In this particular scheme, the DL and UL control channels are always fixed in the first and last symbol of the subframe, respectively. A switching guard period of one symbol is introduced every time the direction changes from UL to DL. In the frequency domain, the entire bandwidth of 1 GHz is divided into 72 sub-bands of width 13.89 MHz, each of which is composed of 48 sub-carriers. It is possible to assign UE data to each of these sub-bands, as is done with Orthogonal Frequency-Division Multiple Access (OFDMA) in LTE, however only TDMA operation is currently supported for reasons we shall explain shortly.

3.2 PHY Transmission and Reception

The `MmWaveEnbPhy` and the `MmWaveUePhy` classes model the physical layer for the mmWave eNodeB and the UE, respectively, and encapsulate similar functionality to the `LtePhy` classes from the LTE module. Broadly, these objects (i) handle the transmission and reception of physical control and data channels (analogous to the PDCCH/PUCCH and PDSCH/PUSCH channels of LTE), (ii) simulate the start and the end of frames, subframes and slots and (iii) deliver received and successfully decoded data and control packets to the MAC layer.

In `MmWaveEnbPhy`/`MmWaveUePhy`, calls to `StartSubFrame()` and `EndSubFrame()` are scheduled at fixed periods, based on the user-specified subframe length, to mark the start and the end of each subframe. The timing of variable-TTI slots, controlled by scheduling the `StartSlot()` and `EndSlot()` methods, is dynamically configured by the MAC via the MAC-PHY SAP method `SetSfAllocInfo()`, which enqueues a `SfAllocInfo` allocation element for some future subframe index specified by the MAC. A *subframe indication* to the MAC layer triggers the scheduler at the beginning of each subframe to allocate a future subframe. For the UE PHY, `SfAllocInfo` objects are populated after reception of Downlink Control Information (DCI) messages. At the beginning of each subframe, the current subframe allocation scheme is dequeued, which contains a variable number of `SlotAllocInfo` objects. These, in turn, specify contiguous ranges of OFDM symbol indices occupied by a given slot, along with the designation as either *DL* or *UL* and control (*CTRL*) or data (*DATA*).

The data packets and the control messages generated by the MAC are mapped to a specific subframe and slot index in the *packet burst map* and *control message map*, respectively. Presently, in our custom subframe design, certain control messages which must be decoded by all UEs (such as the DCIs) are always transmitted in fixed PDCCH/PUCCH symbols as the first and last symbol of the subframe, but this static mapping can easily be changed by the user. [2] Other UE-specific control and data packets are dequeued at the beginning of each allocated TDMA data slot and are transmitted to the intended device.

To initiate transmission of a data slot, the eNB PHY first calls `AntennaArrayModel::ChangeBeamformingVector()` to update the transmit and receive beamforming vectors for both the eNB and the UE. In the case of control slots, no beamforming update is applied since we currently assume an "ideal" control channel. For both DL and UL, either the `MmWaveSpectrumPhy` method `StartTxDataFrame()` or `StartTxCtrlFrame()` is then called to transmit a data or control slot, respectively. The functions of `MmWaveSpectrumPhy`, which are similar to the corresponding LENA class, are as follows. After the reception of data packets, the PHY layer calculates the SINR of the received signal in each sub-band, taking into account the path loss, MIMO beamforming gains and frequency-selective fading. This triggers the generation of Channel Quality Indicator (CQI) reports, which are fed back to the base station in either UL data or control slots. The error model instance is also called to probabilistically compute whether a packet should be dropped by the receiver based on the SINR and, in the case of a HARQ retransmission, any soft bits that have been accumulated in the PHY HARQ entity (see Section 4.3). Uncorrupted packets are then received by the `MmWavePhy` instance, which forwards them up to the MAC layer SAP.

3.3 Channel Models

The mmWave module allows the user to choose between two channel models. The first, implemented in the `MmWavePropagationLossModel` and `MmWaveBeamforming` classes, is based on our previous code in [4], which is derived from ex-

[1] Although many waveforms are being considered for 5G cellular, OFDM is still viewed as a possible candidate. Therefore we adopt OFDM, which allows us to continue to leverage the existing PHY models derived for OFDM from the LTE LENA module.

[2] As in [9, 10], we assume that either FDMA or SDMA-based multiple access would be used in the control regions. However, we do not currently model these modulation schemes nor the specific control channel resource mapping explicitly. We intend for this capability to be available in later versions, which will enable more accurate simulation of the control overhead.

Table 1: Parameters for configuring the mmWave PHY.

Parameter Name	Default Value	Description
SubframePerFrame	10	Number of subframes in one frame
SubframeLength	100	Length of one subframe in µs
SymbolsPerSubframe	24	Number of OFDM symbols per slot
SymbolLength	4.16	Length of one OFDM symbol in µs
NumSubbands	72	Number of sub-bands
SubbandWidth	13.89×10^6	Width of one sub-band in Hz
SubcarriersPerSubband	48	Number of subcarriers in each sub-band
CenterFreq	28×10^9	Carrier frequency in Hz
NumRefScPerSymbol	864 (25% of total)	Reference subcarriers per symbol
NumDlCtrlSymbols	1	Downlink control symbols per subframe
NumUlCtrlSymbols	1	Uplink control symbols per subframe
GuardPeriod	4.16 µs	Guard period for UL-to-DL mode switching
MacPhyDataLatency	2	Subframes between MAC scheduling request and scheduled subframe
PhyMacDataLatency	2	Subframes between TB reception at PHY and delivery to MAC
NumHarqProcesses	20	Number of HARQ processes for both DL and UL

tensive MATLAB® simulation of the 28 GHz channel presented in [2]. This model has now been combined with the built-in `BuildingsObstaclePropagationLossModel` for simulating mobility. The second model, the `MmWaveChannelRaytracing` class, uses data obtained from third-party ray tracing software. The details of each model are as follows.

3.3.1 Simulation-Generated Statistical Model

A full discussion on the simulation based channel model is available in [4]. We have strengthened this model by incorporating a building obstacle propagation model which enables simulation of UE mobility through a shadowing environment. For each simulation, instances of the building class (built into ns-3) are used to simulate obstacles. The channel state is updated based on the relative position of the transmitter, the receiver and the buildings as the UE moves through the environment. A virtual line is drawn between the transmitter and the receiver. If this line intersects any building we assign the channel state as NLoS; otherwise, we assign a LoS channel.

After selecting the channel state, the propagation loss can be computed as

$$PL(d)[dB] = \alpha + \beta 10\log_{10}(d) + \xi, \quad \xi \sim N(0, \sigma^2), \quad (1)$$

where ξ represents shadowing, d is the distance from receiver to transmitter and the values of the parameters α, β, and σ for each channel state are given in [2]. In this model, we consider the channel to be in outage if the distance between the transmitter and the receiver exceeds a predefined threshold.

Due to lack of information about the effects of diffraction in the mmWave channel, we assume the propagation loss increases or decreases suddenly without a transition period when the channel state changes. We plan to model diffraction more accurately in our future work.

Channel matrix. Following the method in [2], to compute the long-term statistical characterization of the mmWave channel, we model it as a combination of clusters, each composed of several subpaths. The channel matrix is described by the following equation,

$$H(t,f) = \sum_{k=1}^{K}\sum_{l=1}^{L_k} g_{kl}(t,f)\mathbf{u}_{rx}(\theta_{kl}^{rx},\phi_{kl}^{rx})\mathbf{u}_{tx}^{*}(\theta_{kl}^{tx},\phi_{kl}^{tx}) \quad (2)$$

where K is the number of clusters, L_k the number of subpaths in cluster k, $g_{kl}(t,f)$ the small-scale fading over frequency and time, $\mathbf{u_{rx}}(\cdot)$is the spatial signature of the receiver and $\mathbf{u_{tx}}(\cdot)$ the spatial signature of the transmitter.

As given in [2], the small-scale fading is generated by

$$g_{kl}(t,f) = \sqrt{P_{kl}}e^{2\pi i f_d cos(\omega_{kl})t - 2\pi i \tau_{kl} f} \quad (3)$$

where P_{kl} is the power of subpath kl, f_d is the maximum Doppler shift, ω_{kl} is the angle between subpath kl and the direction of motion of the receiver, τ_{kl} gives the delay spread and f is the carrier frequency.

Beamforming. The two methods implemented in the `MmWaveBeamforming` class to compute beamforming vectors are the *power iteration method* and the *swipe sector method*. For the power iteration method, we assume that the BS knows the channel matrices and can compute the largest singular value and singular vector associated with the strongest path (i.e., it uses non-codebook based beamforming with perfect Channel State Information (CSI)). Therefore, the optimal set of TX/RX beamforming vectors will always be selected to maximize the antenna array gain for transmission between a given BS-UE pair.

The swipe sector method implements a basic cell search/ synchronization technique where the cell is divided into fixed sectors. The BS and UE scan all sectors and select the beam with maximum gain based on these pre-stored beamforming vectors (i.e., they perform codebook-based beam switching). This method does not require the CSI to be known a priori, but takes additional time to scan the cell. We provide this code as a basis for implementing more advanced cell search and synchronization protocols.

Channel Configuration. For both methods, the channel matrices and optimal beamforming vectors are pre-generated in MATLAB® to reduce the computational overhead in ns-3. At the beginning of each simulation we load 100 instances of the spatial signature matrices, along with the beamforming vectors.

In order to simulate realistic channels with large-scale fading, the channel matrices are updated periodically for NLoS channels but remain constant for LoS links as they are inherently more stable. Currently, no results are available for how

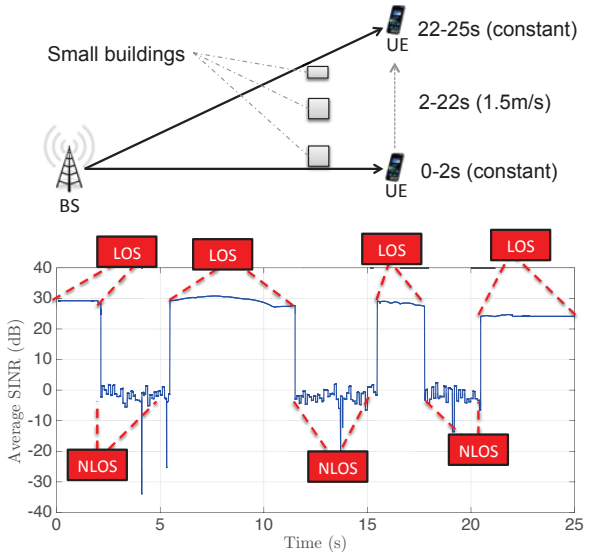


Figure 3: Average SINR plot for simulated route.

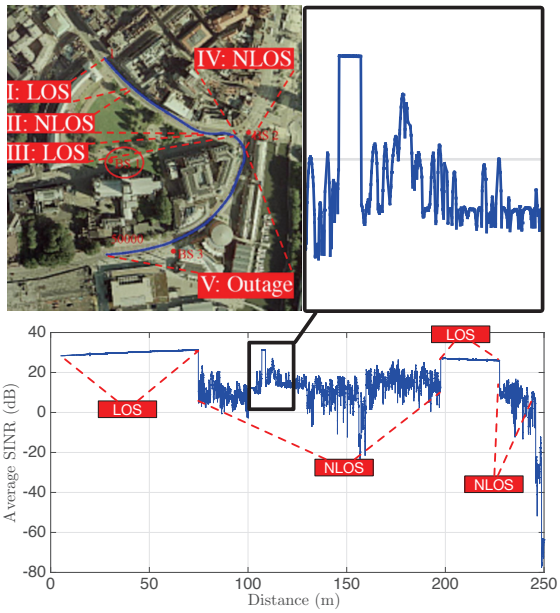


Figure 4: Average SINR plot for raytracing route.

the large-scale statistics of the mmWave channel change over time for a mobile user. We therefore implement a form of large-scale *block fading*, where we update the channel by randomly selecting one of the 100 channel matrix-beamforming vector combinations after some interval. The large-scale parameters of the channel are thus independent in each interval. The update time can be some fixed interval specified by the `LongTermUpdatePeriod` attribute of the `MmWaveBeamforming` class. We also provide the option to update after some time drawn from an exponentially-distributed random variable (i.e., a Poisson process) with mean also defined by the update period attribute. It should be noted that the accuracy of this method is not validated at this time.

The small-scale fading is calculated for every transmission based on Equation (3) where we obtain the speed of the user directly from the mobility model. The remaining parameters depending on the environment are assumed to be constant over the entire simulation time.

From Figure 3 we observe the average SINR trend obtained in a scenario where 3 buildings are distributed between BS and UE. The number of antennas at the BS and the UE is 64 and 16, respectively. The user starts moving at a speed of 1.5 m/s 2 seconds after the start of the simulation and stops after 20 seconds. As expected, the SINR is constant over time while the user is static (0-2 s and 22-25 s). However, the SINR varies over time when the user is in motion. The sudden SINR jumps result from the switching of the channel state (as discussed, the channel matrices are updated after a fixed 100 ms interval for the NLoS channel and remain unchanged for LoS transmissions).

3.3.2 Ray Tracing-Generated Model

In order to better characterize the time dynamics of the mmWave channel, we have added a ray tracing-generated channel model that computes the channel matrices in ns-3. The input is traces generated by third-party ray-tracing software, which simulates the physics of radio propagation for a specific environment. [3] This channel model offers more flexibility to customize parameters, such as transmit and receiver antenna elements. Figure 4 shows the average SINR generated in ns-3 over a given ray tracing route, with 64 transmit and 16 receive antennas. Propagation loss and channel matrices are computed according to Equation (2), where parameters are obtained from the ray-tracing data containing 5000 samples within a 500 meter-long route. Each sample includes the following fields:

- Number of paths
- Propagation loss per path
- Delay per path
- Angle of arrival (elevation and azimuth plane) per path
- Angle of departure (elevation and azimuth plane) per path

As the user moves, the channel matrices are updated accordingly. For example, if the current location of the user is 10.1 meters from the BS, channel matrices are computed using the data corresponding to this distance. The beamforming vectors are generated using the power method discussed earlier.

Figure 4 plots the average SINR indicating both LoS intervals and NLoS channel states. The SINR has a sudden change when the channel state switches. We note that the SINR curve within LoS matches our simulation generated model, but for NLoS the ray tracing based model introduces more random variation.

3.3.3 Interference

Albeit potentially less significant for directional mmWave signals, which are generally assumed to be power-limited, there are still some cases where interference is non-negligible. For instance, although intra-cell interference (i.e., from devices of the same cell) can be neglected in TDMA or FDMA-mode operation, it does need to be explicitly calculated in the case of SDMA/Multi-User MIMO, where users are multiplexed in the spatial dimension but operate in the same time-frequency resources. As mentioned, only TDMA transmission is currently supported, however we anticipate that other multiple access schemes, including SDMA, will be added in

[3]The ray tracing data is provided by the Communication Systems and Networks Group, University of Bristol, UK.

later versions. Therefore, we include interference computation in the `MmWaveInterference` class that takes into account the beamforming vectors associated with each link. More details on the relevant computation can be found in Section 3.4 of [4].

4. MAC LAYER

The high-level structure and functions of the mmWave MAC layer are now introduced. In particular, the two scheduler classes, multi-process stop-and-wait Hybrid ARQ (for both the DL and the UL), along with some minor modifications that have been made to the LENA module AMC and MI error model classes, are described in the following sections.

4.1 MAC Scheduling

We now present the implementation of two scheduler classes, which are based on a variable-length or *flexible* TTI, Time-Division Multiple Access (TDMA) scheme.

TDMA is widely assumed to be the de-facto scheme for mmWave access because of the dependence on analog beamforming, where the transmitter and receiver align their antenna arrays to maximize the gain in a specific direction (rather than in a wide angular spread or omni-directionally, like conventional antennas). Many early designs and prototypes have been TDMA-based [7, 8], with others incorporating SDMA for the control channel only [9]. While SDMA or FDMA schemes (like in LTE) are possible with *digital beamforming*, which would allow the base station to transmit and receive in multiple directions within the same time slot, they may not be practical for mmWave systems due to high power consumption from requiring Analog-to-Digital Converters (ADCs) on each antenna element [10]. It should also be noted that FDMA with analog beamforming is possible with wider beam widths, but this approach comes at the cost of some of the directional gain.

Furthermore, one of the foremost considerations driving innovation for the 5G MAC layer is latency. The Key Performance Indicator (KPI) of 1 ms over-the-air latency has been proposed as one of the core 5G requirements by such standards bodies as the ITU as well as recent studies such as those carried out under the METIS 2020 project [11]. However, a well-known drawback of TDMA is that fixed slot lengths or TTIs can result in poor resource utilization and latency, which can become particularly severe in scenarios where many intermittent, small packets must be transmitted to/received from many devices.

Based on these considerations, variable TTI-based TDMA frame structures and MAC schemes have been proposed in [9, 10, 12]. This approach supports slot sizes that can vary according to the length of the packet or Transport Block (TB) to be transmitted and are well-suited for diverse traffic since they allow both bursty or intermittent traffic with small packets as well as high-throughput data like streaming and file transfers to be scheduled efficiently and without significant under-utilization.

4.1.1 Round-Robin Scheduler

The `MmWaveFlexTtiMacScheduler` class is the default scheduler for the mmWave module. It implements a variable TTI scheme previously described in Section 3 and assigns OFDM symbols to user flows in Round-Robin (RR) order. Upon being triggered by a subframe indication, any HARQ retransmissions are automatically scheduled using whatever OFDM symbols are available. While the slot allocated for a retransmission does not need to start at the same symbol index as the previous transmission of the same TB, they do need the same number of contiguous symbols and Modulation and Coding Scheme (MCS), since an adaptive HARQ scheme has not yet been implemented.

Before scheduling new data, Buffer Status Report (BSR) and Channel Quality Indicator (CQI) messages are first processed. The MCS is then computed by the AMC model for each user based on the CQIs for the DL or SINR measurements for the UL data channel. The MCS and the buffer length of each user are used to compute the minimum number of symbols required to schedule the data in the user's RLC buffers.

To assign symbols to users, the total number of users with active flows is calculated. Then the total available data symbols in the subframe are divided evenly among users. If a user requires fewer symbols to transmit its entire buffer, then the remaining symbols (i.e., the difference between the available and required slot length) are distributed among the other active users.

One also has the option to set a fixed number of symbols per slot by enabling *fixed TTI* mode. However, utilization and latency are likely to suffer in this case, depending on the traffic pattern.

4.1.2 Max-Weight Scheduler

The `MmWaveFlexTtiMaxWeightMacScheduler` class is similar to the RR scheduler but is intended to provide various priority queue policies. Currently only an Earliest Deadline First (EDF) policy is implemented, which weighs flows by their relative deadlines for packet delivery, with weights determined by the delay budget of the QoS Class Indicator (QCI) configured by the RRC layer. The EDF scheduler can be used to evaluate the delay performance of various radio frame configurations, although the results of such analysis are outside the scope of this paper. Other weight-based disciplines, such as Proportional Fair (PF) scheduling, will be added in future versions.

4.2 Adaptive Modulation and Coding

The `MmWaveAmc` class reuses most of the code from the corresponding LENA module class. Some minor modifications and additional methods were necessary to accommodate the dynamic TDMA MAC scheme and frame structure. For instance, the `GetTbSizeFromMcsSymbols()` and `GetNumSymbolsFromTbsMcs()` methods are used by the scheduler to compute the TB size from the number of symbols for a given MCS value, and vice-versa. Also the `CreateCqiFeedbackWbTdma()` method is added to generate wideband CQI reports for variable-TTI slots.

Figure 5 shows the results of the test case provided in *mmwave-amc-test.cc*. This simulation serves to demonstrate the performance of the AMC and CQI feedback mechanisms for a single user in the UL (although a multi-user scenario could easily be configured as well). The default PHY/MAC parameters in Table 1 are used along with the default scheduler and default parameters for the statistical path loss, fading and beamforming models (i.e., `MmWavePropagationLossModel` and `MmWaveBeamforming`).

We compute the rate versus the average SINR over a period of 12 seconds (long enough for the small-scale fading to

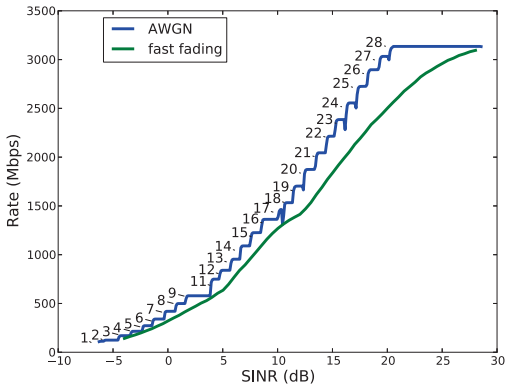


Figure 5: Rate and MCS vs. SINR for single user under AWGN and fast-fading mmWave channels

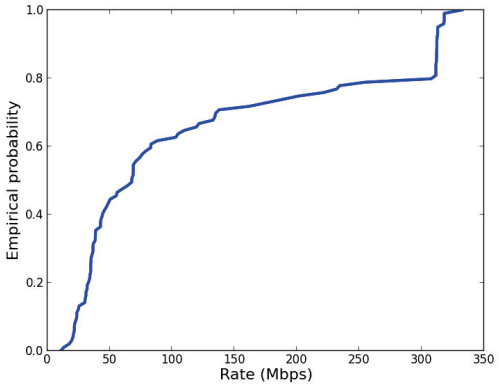


Figure 6: Empirical CDF of DL user rates for 10 users with RR scheduling.

average out), after which we artificially increase the path loss while keeping the UE position fixed. The average PHY-layer rate is then computed as the average sum size of successfully decoded TBs per second. As the SINR decreases, the MAC will select a lower MCS level to encode the data. The test is performed for the AWGN case as well as for small-scale fading. Although the UE position relative to the base station is constant, we can generate time-varying multi-path fading through the `MmWaveBeamforming` class by setting a fixed speed of 1.5 m/s to artificially generate Doppler, which is a standard technique for such analysis. Also we assume that the long-term channel parameters do not change for the duration of the simulation.

If this plot is compared to the one generated from a similar test in Figure 3.1 of the LENA documentation [6], we notice that the AWGN curve from the mmWave test is shifted by approximately 5 dB to the left, indicating that the LENA version is transitioning to a lower MCS at a much higher SINR. This is because the LENA test is using the more conservative average SINR-based CQI mapping, whereas we use the Mutual Information-Based Effective SINR (MIESM) scheme with a target maximum TB error of 10% in order to maximize the rate for a given SINR [13].

4.3 Hybrid ARQ Retransmission

Full support for HARQ with soft combining is now included in the mmWave module. The `MmWaveHarqPhy` class along with the functionality within the scheduler are based heavily on the LENA module code. However, multiple HARQ processes per user in the UL are now possible. The number of processes can also be configured through the `NumHarqProcesses` attribute in `MmWavePhyMacCommon`. Additional modifications were needed to support larger codeword sizes in both the HARQ PHY methods and the error model.

5. EXAMPLE SIMULATIONS

We now present some example simulations to show the utility of the framework for the analysis of novel mmWave protocols and testing higher-layer network protocols, such as TCP, over 5G mmWave networks. The simulations in this section are all configured with basic PHY and MAC parameters as in Table 1, with other notable parameters given in the sequel.

5.1 Multi-User Throughput Simulation

The purpose of this experiment, which one can reproduce by running the *mmwave-tdma* example, is to simulate the DL throughput of 10 UEs in a 1 GHz mmWave cell under the variable TTI/TDMA MAC scheme and round-robin scheduling policy. UEs are placed at uniformly random distances between 20 and 200 meters from a single eNodeB. As explained in Section 4.2, users are stationary but are modeled as having a constant speed of 1.5 m/s and are thus subject to small-scale fading. The long-term channel parameters are updated based on the exponentially-distributed update time with a mean of 100 ms (see Section 3.3). Rates are computed from the average size of RLC PDUs delivered to each UE and therefore reflect the performance of the stack up to and including the RLC layer. We assume full-buffer traffic.

The simulation is performed for 10 runs or *drops* of the 10 UEs, where for each drop they are placed at different distances and assigned different channel matrices. The average system throughput at the RLC layer for this scenario over all drops is found to be about 1.2 Gbps. We observe in the plot of the empirical distribution function in Figure 6 how UEs with LoS links all have roughly the same average rates around 325 Mbps. Also a significant number of NLoS users achieve rates over 100 Mbps, and even the worst 5% of users at the cell edge can get between 10 and 20 Mbps.

5.2 TCP Performance over mmWave

Here we run the *mmwave-tcp-building* example to analyze the performance of TCP flows over a mmWave link. TCP data packets are sent from a remote host to the UE at a rate of 1 Gbps. The New Reno algorithm is used for this experiment. The delays for the point-to-point link between the remote host and PDN-Gateway (PGW), as well as that from the PGW to the BS, are set to 10 ms. Thus, the contribution to the total Round Trip Time (RTT) from the core network is 40 ms and any additional latency is due to the radio link, which, under stable queue conditions, is observed to be less than 10 ms. The size of the RLC-AM buffer is adjusted to 10 Megabytes to avoid overflow. The TCP buffer size is set to 5 Megabytes and the slow start threshold is 6000 segments (about 3 MB).

Figure 7 plots the SINR, transport layer throughput, congestion window size (CWnd) and RTT. As shown, the transport layer throughput matches the sending rate for the LoS

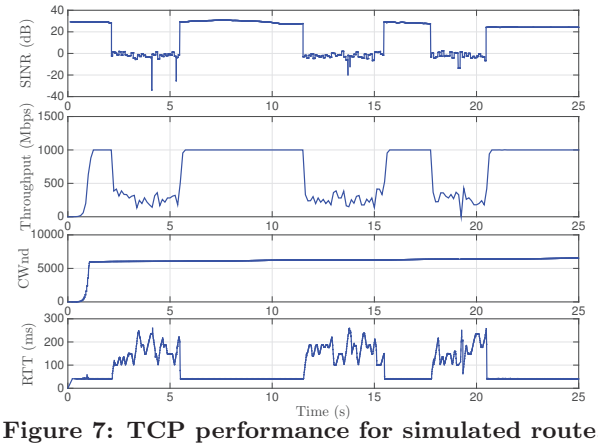


Figure 7: TCP performance for simulated route.

channel, but is reduced when the channel is in the NLoS state. This is attributed to the MAC-layer AMC model adapting to the change in capacity. From the congestion window plot we see that there is no TCP timeout or retransmission since the packet loss events are handled by lower layer retransmissions, i.e. MAC layer HARQ and RLC ARQ. Moreover, we see that RTT is around the baseline of 40 ms for the LoS channel, but goes above 150 ms for the NLoS case. It is clear that decreased channel capacity and more frequent RLC retransmission events cause the RLC buffer to become backlogged, which explains the increase in RTT. This result suggests the need for a more advanced congestion control mechanism, perhaps aided by feedback or control from lower layers, to prevent large spikes in latency under rapid channel fluctuations.

6. CONCLUSIONS & FUTURE WORK

In this paper, the current state of the ns-3 framework for simulation of mmWave cellular systems has been presented. The code, which is publicly available at GitHub [14], is highly modular and customizable to facilitate researchers to experiment with novel 5G protocols. The code includes implementations of a mmWave eNodeB and User Equipment stack, including the MAC layer, PHY layer and channel models. Some example simulations have been given, which show how the framework may be used for analysis of custom mmWave PHY/MAC protocols as well as higher-layer network protocols over a mmWave stack and channel.

As part of our future work, we have targeted several new features for channel modeling, including a more accurate model for large-scale fading for mobile users as well as channel matrix generation and beamforming computation within ns-3 to support experimentation with adaptive beamforming algorithms. Future enhancements to the MAC layer include support for other multiple access schemes, relay devices, and additional scheduling algorithms. Also, although a number of configurable example scripts are currently included, which may be used for testing, a complete test framework is not yet been provided. Thus, we intend to include a set of test scripts in a later release.

7. REFERENCES

[1] S. Rangan, T. S. Rappaport, and E. Erkip, "Millimeter-wave cellular wireless networks: Potentials and challenges," *Proc. IEEE*, vol. 102, no. 3, pp. 366–385, Mar. 2014.

[2] M. Akdeniz, Y. Liu, M. Samimi, S. Sun, S. Rangan, T. Rappaport, and E. Erkip, "Millimeter wave channel modeling and cellular capacity evaluation," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1164–1179, June 2014.

[3] "ns-3 Network Simulator," Available at `http://www.nsam.org`, Feb. 2012.

[4] M. Mezzavilla, S. Dutta, M. Zhang, M. R. Akdeniz, and S. Rangan, "5G mmwave module for the ns-3 network simulator," in *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '15)*, Nov. 2015, pp. 283–290.

[5] "LTE-EPC Network Simulator," Available at `http://iptechwiki.cttc.es/LTE-EPC_Network_Simulator_(LENA)`.

[6] "The LENA ns-3 LTE Module Documentation," Available at `http://iptechwiki.cttc.es/LTE-EPC_Network_Simulator_(LENA)`.

[7] Z. Pi and F. Khan, "System design and network architecture for a millimeter-wave mobile broadband MMB system," in *Proc. IEEE Sarnoff Symposium*, May 2011.

[8] A. Ghosh, T. A. Thomas, M. C. Cudak, R. Ratasuk, P. Moorut, F. W. Vook, T. S. Rappaport, J. G. R. MacCartney, S. Sun, and S. Nie, "Millimeter-wave enhanced local area systems: A high-data-rate approach for future wireless networks," *IEEE J. Sel. Areas in Comm.*, vol. 32, no. 6, pp. 1152–1163, June 2014.

[9] T. Levanen, J. Pirskanen, and M. Valkama, "Radio interface design for ultra-low latency millimeter-wave communications in 5G era," in *Proc. IEEE Globecom Workshops (Gc Wkshps)*, Dec. 2014, pp. 1420–1426.

[10] S. Dutta, M. Mezzavilla, R. Ford, M. Zhang, S. Rangan, and M. Zorzi, "Frame structure design and analysis for millimeter wave cellular systems, to appear in *Proceedings of the European Conference on Networks and Communications (EuCNC 2016)*," Jun. 2016.

[11] P. Popovski, V. Brau, H.-P. Mayer, P. Fertl, Z. Ren, D. Gonzales-Serrano, E. G. Ström, T. Svensson, H. Taoka, P. Agyapong *et al.*, "EU FP7 INFSO-ICT-317669 METIS, D1.1: Scenarios, requirements and KPIs for 5G mobile and wireless system," 2013.

[12] P. Kela, M. Costa, J. Salmi, K. Leppanen, J. Turkka, T. Hiltunen, and M. Hronec, "A novel radio frame structure for 5G dense outdoor radio access networks," in *Proc. IEEE 81st Vehicular Technology Conference (VTC Spring)*, May 2015, pp. 1–6.

[13] M. Mezzavilla, M. Miozzo, M. Rossi, N. Baldo, and M. Zorzi, "A lightweight and accurate link abstraction model for the simulation of lte networks in ns-3," in *Proceedings of the 15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '12)*, Oct. 2012, pp. 55–60.

[14] "ns-3 module for simulating mmwave-based cellular systems," Available at `https://github.com/mmezzavilla/ns3-mmwave`.

ns-3 Web-Based User Interface - Power Grid Communications Planning and Modeling Tool

Kurt Derr
Idaho National Laboratory
2525 Freemont Avenue
Idaho Falls, Idaho 83415
kurt.derr@inl.gov

ABSTRACT
All utilities have invested in some level of engineering tools and qualified staff required to modernize the power grid. These engineering tools vary in their level of complexity and fidelity. Most of the existing tools are either application specific or have not been designed to work together to assist in assembling comprehensive first order smart grid build-out cost estimations and planning, especially with regard to communications. There is a need for software to assist utility owners, operators, engineers, and consultants with smart grid communications planning and engineering. The development of a web-based Power Grid Communications Planning and Modeling Tool using ns-3 is in process that will address these needs. This tool models use cases for power grid communications, abstracting away the details of programming in ns-3. The tool will be made open source at the completion of the initial version.

CCS Concepts
• Computing methodologies → Model development and analysis
• Networks → Network performance evaluation

Keywords
Smart Grid, Network Simulation, Use Case, OMF

1. INTRODUCTION
The Power Grid Communications Planning and Modeling Tool (PGCPMT) is a power grid communications network planning tool for the smart grid. The smart grid is characterized by a set of objectives specified in the U.S. Energy Independence and Security Act of 2007. The goal of the PGCPMT is to perform a comprehensive grid communications analysis that will evaluate a utility's existing architecture and/or proposed enhancements and help identify optimal implementation strategies that will align with the utility's business roadmap and constraints.

The tool interacts with the user via a web-based geospatial user interface coupled with other screens for specialized subsystems such as modeling, simulations, cyber security, cost analysis, etc. This interaction will occur in several different ways, including creating, converting, manipulating, and analyzing or visualizing the data and analysis results.

Understanding that users will be applying this tool with various levels of data fidelity available to them for their specific system, the tool will accommodate planning and design using a range of data from basic "rough" estimations to highly detailed network information.

By employing a library of grid component use cases that identify not only the specific properties of each device but also the recommended communication pathways and integration scenarios, the tool will assist users in determining a confidence level in their proposed configurations. This capability will aid in identifying where more information and what detail is needed to improve the overall design. The tool is intended to answer the question, "will my communications network support the smart grid hardware and applications I want to run?" The tool will also guide the user in entering data required for preparing graphs and tables that summarize the costs and benefits of the project.

The concept behind this tool development is to create an application that blends geospatial analysis, standards application, component database, and communication network modeling/simulation with a graphical front end and menu driven guidance that can adapt to meet the users skill level from a data and expertise standpoint. The focus is to develop a tool to assist in developing the communications and networking systems for the smart grid, and not for developing the power delivery infrastructure of the smart grid. The source code for PGCPMT will be made available in the near future at the discretion of the Department of Energy.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 provides an overview of the PGCPMT. Section 4 presents the architecture of PGCPMT. Section 5 presents the PGCPMT implementation. Section 6 presents the smart grid use case implementations with ns-3. Section 7 discusses the cyber security module that provides guidance for cyber secure communications. Section 8 presents conclusions and future work.

2. RELATED WORK
PGCPMT is designed to provide a graphical user interface (GUI) front end to ns-3 and support the analysis of smart grid use cases. The author is not aware of any other effort that provides a GUI front end to ns-3 focused on the network modeling needs of the utility community for smart grid deployment. Related efforts describing the integration of ns-3 with other toolsets are subsequently described.

CORE is the Common Open Research Emulator that controls virtual machines and a network emulation subsystem [1]. The emulated networks may be on one or more machines and can be connected to live networks. A CORE GUI tool may be used to create diagrams of network topologies. ns-3 networks can only be instantiated from a Python script or from the GUI hooks facility in the most current version of CORE. The WiFi model is supported

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915373

and experimentation is continuing with the WiMAX and 3GPP LTE models.

Radio Mobile (RM) [2-3] is a freeware tool that predicts the performance of a radio system by using digital terrain elevation data. A user may create a graphical view of their network topology on top of a rendered topography via the RM user interface. A python script is used to translate the output results of RM into a form suitable for import into an ns-3 simulation. Other tools such as Gnuplot and net-measure may then be used to create and view the performance of the simulation.

The Network Experiment Programming Interface (NEPI) [4] is an experimental life-cycle management tool for heterogeneous experiments with PlanetLab and Emulated nodes seamlessly integrated with ns-3. NEPI is a general solution for providing life-cycle control support for non-specific platform resources. NEPI supports the design, execution, control, and results sequence of steps in an experiment workflow. The experiment description is written in XML providing an environment-independent application programming interface (API) for the user. NEPI supports ns-3, NETNS emulator, and PlanetLab. The two ways of using NEPI are the Network Experimentation Frontend (NEF) GUI or a python script. NEF allows dragging and dropping and interconnecting of boxes in a canvas. Complex experiments may be also designed and executed as a single python script.

The Universidad de los Andes has performed modeling and simulation of an AMI network implemented with Long Term Evolution (LTE) and WiFi technologies in ns-3 [5]. This work includes the Device Language Message Specification (DLMS) and the COmpanion Specification for Energy Metering (COSEM). The authors have made the source code available for the AMI network that includes multiple elements; a data concentrator model; meter data management application; a demand response (DR) application for a utility control center implemented without the DR mechanism; LTE, WiFi, and CSMA technologies in the DLMS/COSEM application.

None of these efforts provides a GUI exposing the capabilities of ns-3 that is specifically targeted to support multiple smart grid use cases and networking technologies, such as wireless mesh, WiFi, LTE, WiMAX, and wired links. The next section presents an overview of the PGCPMT.

3. TOOL OVERVIEW

ns-3 does not have a graphical user interface. Simulation code is written in C++ or python to run an ns-3 simulation. Integrating the ns-3 network simulator with GUIs is a large effort [6]. The PGCPMT implementation is factoring this integration of GUIs and ns-3 into manageable phases and building needed capabilities over time. PGCPMT utilizes the ns-3 C++ bindings, rather than the python bindings, to maximize the network simulation capabilities that may be made available to the user.

PGCPMT will provide a capability to compare the performance of multiple network models. Cost and performance criteria may be used to evaluate each model.

The PGCPMT is comprised of the subsystems shown in Figure 1. The implementation of each of these sub systems is in varying states of progress.

The geospatial interface enables the user to develop a location based layout of the communications network for power grid network communications simulation and analysis. The interface includes svg-edit and OpenLayers v3.12.1. svg-edit is an opensource web-based JavaScript-driven SVG drawing editor that works in a browser. The INL has customized svg-edit for the visual graphical objects to contain metadata. The metadata describes attributes of the real world network modeling objects represented by the graphical elements, such as routers, switches, and data concentrators. The logical network diagramming subsystem provides a drawing tool for creating or editing logical network diagrams that also utilizes this customized version of svg-edit. OpenLayers is an opensource JavaScript library to load, display and render maps from multiple sources on web pages.

The cybersecurity subsystem provides users with a systematic, repeatable approach for assessing the cybersecurity posture of their grid network components. A user may select the standards for evaluating their network architecture that will be used to generate a question-answer set. The questions and answers are used to collect facility specific system information for evaluating the cyber security posture of a user's network.

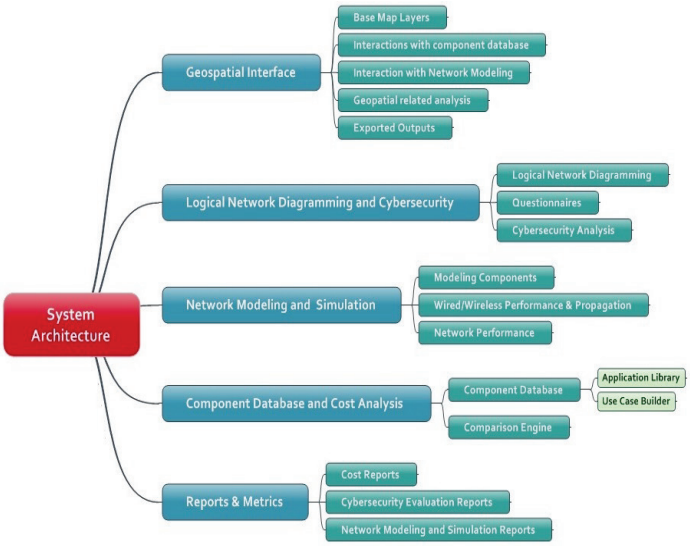


Figure 1. System overview.

The network modeling and simulation subsystem enables utilities to quickly evaluate design alternatives, or different configurations, from both a performance and cost standpoint. Users may easily compare the same or similar topologies using different networking technologies and operating parameters.

The component database and cost analysis subsystem database includes networked devices such as meters, gateways, and distributed generation controllers, as well as different communication channel types. This subsystem will enable users and vendors to add component elements to the system in the future.

The reports and metrics subsystem includes reports/data on network simulation results and the results of the cybersecurity evaluation. Cost reports are a future capability dependent upon component database data. Metrics for evaluating network performance must be specified before running a network simulation. Examples of metrics include packet delivery ratio, latency, packet loss, throughput, and average delay.

The next section presents the tool's architecture that provides a framework for the implementation of the PGCPMT subsystems.

4. ARCHITECTURE

The high-level architecture of the PGCPMT is shown in Figure 2. The Open Modeling Framework (OMF) is an analytics platform for simulating the behavior of the electric grid [7] for smart grid cost-benefit analysis. OMF, developed by the Cooperative Research Network™ (CRN), is based on Flask, a python-based web-development environment. CRN is the technology research arm of the National Rural Electric Cooperative Association (NRECA). NRECA members may log into the interface hosted in the cloud, and accessible via a web browser, to perform power distribution system simulation and analysis. NRECA members may make their data and results sharable with other users, or retain the privacy of this information. Developers may download and install a working version of OMF from the GitHub web site. The goal of OMF is to make advanced power distribution systems models usable in the electric cooperative community.

The PGCPMT application is built within the OMF framework and extends OMF with new GUIs, an ns-3 code generator, and access to the ns-3 simulator. PGCPMT allows a user to create a new communications model representing their utilities communications infrastructure, or to select among a variety of use cases representing network topologies and editing those topologies to address their communications requirements.

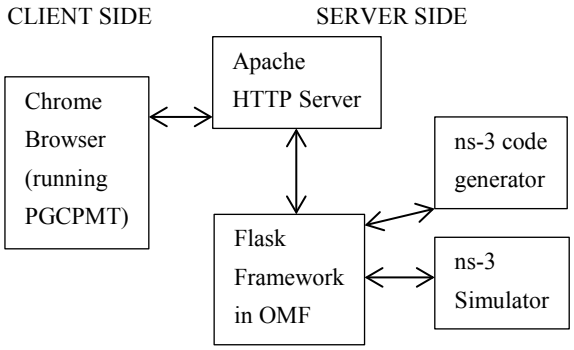


Figure 2. PGCPMT architecture.

OMF uses the D3 JavaScript library for producing dynamic, interactive data visualizations. Any browser such as Chrome, or others, that supports D3 may be used to run the OMF and the PGCPMT.

The ns-3 code generator uses XSD (XML Schema Definition) to specify how to formally describe the elements in an Extensible Markup Language (XML) document. The XML document presented to the ns-3 code generator describes the network topology, operating parameters of the requested network simulation and the desired output results for analysis by the end user. The ns-3 code generator is an Idaho National Laboratory (INL) enhanced version of the ns-3 topology generator tool originally written by Pierre Weiss and Sebastien Vincent, University of Strasbourg.

The implementation of the PGCPMT tool based on this architecture is described next.

5. PGCPMT IMPLEMENTATION

The OMF is targeted towards power system modeling. The INL has extended OMF via additional GUI screens to support network communications modeling. This section briefly describes the OMF power distribution systems modeling screens with subsections providing a more detailed explanation of the new network communications modeling capability now incorporated in the OMF.

Network communications modeling for the smart grid with the PGCPMT requires a number of steps starting from creating a model to analyzing the output results of a simulation.

1. Identify what smart grid use case to model. A user may alternatively create a model that is not one of the pre-defined use cases.
2. The PGCPMT initially displays a logical network model diagramming view. The user may alternatively select a Geographical Information System (GIS) modeling view for placing network components at specific locations.
3. Create the network model/topology. Alternatively a user may open a previously created topology.
4. Specify the applications that will run on each of the network components in the diagram. Smart grid use cases, described in Section 6, will have specialized applications.
5. Specify the operating parameters for the network if this is a new model or, if desired, adjust existing parameters. Operating parameters include attributes such as packet sizes, intervals between packets, maximum number of packets, data rate, error rate and delay of communications line/channel, mobility model, positions of mobile devices, and wireless propagation model.
6. Specify the desired output results that enable the user to evaluate the performance of the network. The options include packet capture (pcap) and ascii trace file data, as well as packet delivery ratio, latency, packet loss, throughput, and average delay for different elements of the network topology.
7. Specify the length of time to run the simulation if applicable.
8. Run the simulation. Distributed simulations are not supported.
9. Review the output results of the simulation. A user may want to compare network simulation results from multiple models, adjust operating parameters and rerun a simulation. When a user is satisfied with the network model, the user may export or print the results of the modeling effort.

A walkthrough of the PGCPMT screens for performing network modeling is described next.

5.1 Logon User Interface

The initial screen of the PGCPMT that is viewable in a browser is the logon screen shown in Figure 3.

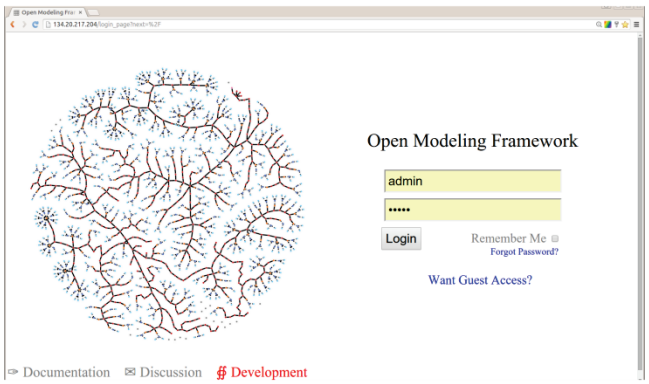


Figure 3. OMF logon screen.

This is the standard OMF login screen. The user is then presented with the power systems modeling interface after logging into OMF.

5.2 Power Systems Modeling User Interface

A user may next select power system models, feeders, or communications systems modeling. The user is initially presented with the models screen, shown in Figure 4, for power systems modeling.

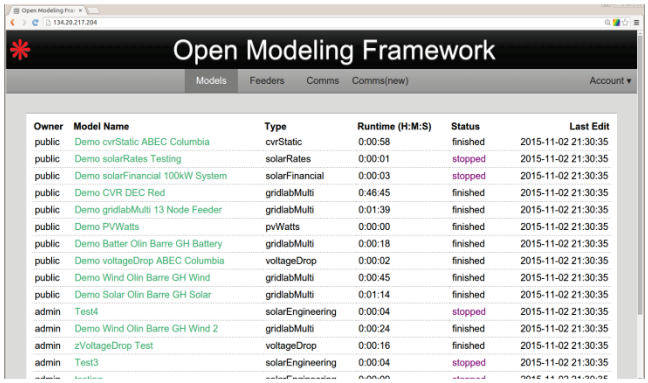


Figure 4. Power distribution system models screen.

A user may import feeder data or select a feeder for power systems modeling as shown in Figure 5.

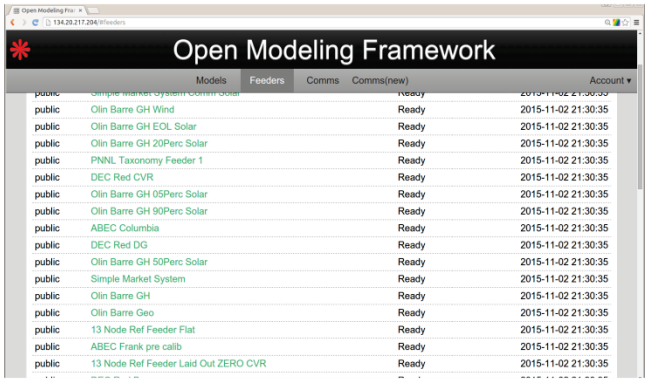


Figure 5. Power distribution system feeders screen.

OMF utilizes GridLAB-D, a power distribution system simulation and analysis tool for users who design and operate distribution systems. GridLAB-D simulates the interdependent behavior of a multitude of devices in the power system. GridLAB-D is not part of, or integrated with, the OMF communications modeling using ns-3. GridLAB-D may be used to evaluate new power systems technologies and control systems such as microgrid operation and control, distributed energy storage and generation, and feeder reconfiguration and automation [8]. GridLAB-D is packaged with OMF for developer download.

5.3 Network Communications Modeling User Interface

The network communications modeling screen, shown in Figure 6, is accessed by selecting the Comms tab. The modeling screen has both horizontal and vertical toolbars, a drawing/viewing pane, and a layers pane on the right side of the screen.

The horizontal toolbar across the top of the modeling screen has icons for SVG code viewing, wireframe mode, show/hide grid, xml code viewing, analysis, cyber, GIS lock, GIS view, undo button, and redo button. The GIS view icon allows the user to easily switch from a logical network diagramming view to a GIS view to place icon and communications lines at specific geographic locations.

The vertical toolbar on the left side of the modeling screen has icons, in top to bottom order, for selection, drawing lines, selecting node types, selecting link types, text for annotating the diagram, and a search button. The node and link buttons allow the designer to create network components and specify the communications link between those components representative of their topology. The layers pane allows the user to select one of their created models for modeling and analysis.

A user may create a network topology diagram as:

1. a logical network diagram or
2. network diagram in a GIS view.

A logical network model is useful when a user wishes to get an idea of the performance of a network under ideal conditions (no reflections, scattering, or propagation delays of wireless signals) without specifying network component locations. The GIS network modeling view allows a user to place networking components at their desired locations and take environmental and propagation delay factors into account.

A user may create a logical network diagram, shown in Figure 6, or open a previously created logical network diagram, to analyze the performance of a proposed network architecture. Network components such as routers, switches, data concentrators, and computers may be dragged and dropped onto the diagram. The connections between these components, such as a wired or wireless link, may be specified by selecting a link type and then the components for this connection. A user may save the topology through the Topology drop down menu in the left corner of the modeling tool screen.

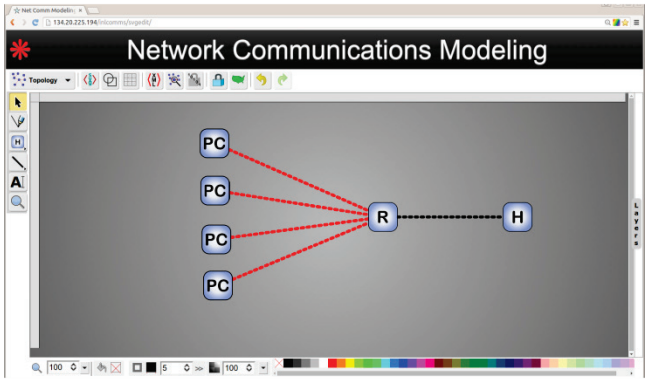


Figure 6. Network communications modeling screen.

A user may associate this topology data with geography and location by selecting the GIS icon. The network is then visualized in a GIS map view. The user may next zoom into the GIS map at the appropriate level and visually move the network elements to their desired locations as shown in Figure 7. Selecting the GIS lock icon will then associate GIS coordinates for each of the network elements at their current locations.

A "layers" pane is displayed in the right side of the modeling screen. The user may quickly switch from one model to another to compare and contrast network topologies and simulation results.

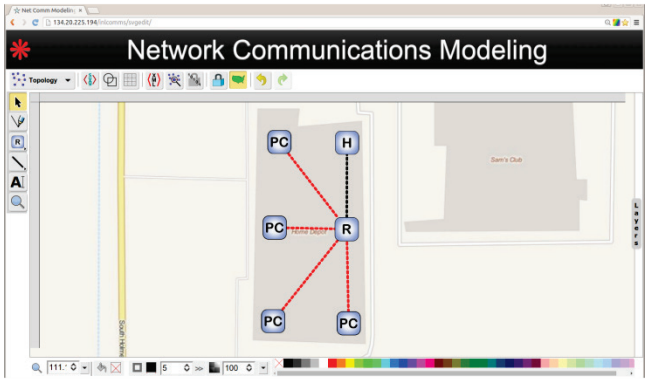


Figure 7. GIS network diagramming screen.

Alternatively a user may create a new diagram in the GIS view without using the logical modeling screen. A future capability will enable a user to import network topology data in XML, CSV, or other formats to expedite the modeling process.

A user also has the capability to select a predefined smart grid *use case* with a specific topology as shown in Figure 8. This allows the user to quickly develop a new network diagram that is representative of their utilities communications infrastructure. Each use case is associated with appropriate smart grid application(s) that will be simulated using ns-3.

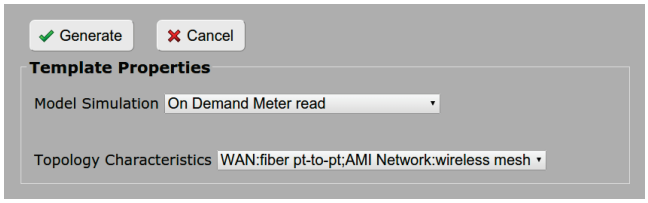


Figure 8. Smart grid use case and topology selection.

The operating parameters for network components and communications lines may be specified by the user by editing the properties of a component. Default operating parameters are provided for every network element.

The network communications GUI enables the user to create and view a network to be modeled or simulated. The network diagram is represented in scalable vector graphics format (SVG). Any network diagram may be modified, saved, and reused as needed by the user. After creating a model of the network the user next decides what data should be collected to evaluate the performance of the network and the value of the model.

5.4 Simulation Execution and Desired Output Results

Figure 9 shows the performance measurements and data that may be collected and reported between nodes connected through a path in the network topology. Multiple metrics, previously noted, may be used to evaluate the performance of the network. The expected reliability and latency performance measures, such as for the smart meter (SM) application and the communications link protocols connecting a smart meter to a data concentrator (DC), may be entered by the user. Reliability is the probability that an operation will complete without failure over a specific period or amount of time [9]. Latency is the time from when an application at an origin node sends a packet to an application at a destination node which receives the packet.

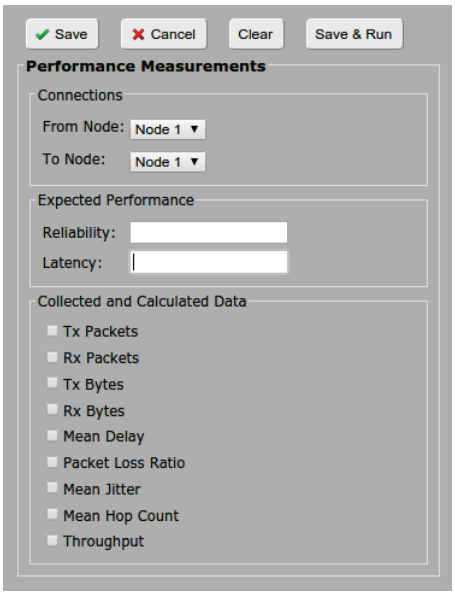


Figure 9. Performance measurements selection.

A user selects a sending and receiving node, optionally specifies expected reliability and latency requirements, indicates the data to collect and calculate, and saves the entered information. These performance measurements may be specified between two nodes directly connected through a wired/wireless link or between two nodes through which a communications path exists over multiple network hops.

When a user runs a simulation of a model by clicking on the analysis icon in the horizontal toolbar, the SVG representing the drawn network topology in the GUI is translated into XML. The XML describes the network topology, operating parameters, and desired simulation results specified by the user. The XML is passed to the server (see Figure 2) and translated into C++ ns-3 code by the ns-3 code generator program. The simulation is then run by the server. At the completion of the simulation the modeling results are then translated into user interface changes representing the output results requested by the user. The simulation output results may be textual, or ascii and/or pcap trace files. Graphs and plots of network performance data will be provided in the future.

The results of the simulation are stored in a database allowing the user to analyze data over multiple trials (simulation runs with a variety of operating parameters and/or topologies). Future improvements to the PGCPMT will include additional features to compare and contrast the results of multiple simulation runs, and to request a copy of the generated C++ ns-3 simulation code. Advanced users may desire a copy of the generated simulation code to customize and run directly in ns-3 without the PGCPMT.

PGCPMT supports a number of pre-defined use cases presented in the next section. A user may alternatively create a topology that is not defined within these use cases.

6. SMART GRID USE CASES IN PGCPMT

The current use cases [9] built into PGCPMT are meter reading for the Advanced Metering Infrastructure (AMI). These use cases may be implemented with multiple technologies such as wireless mesh, power line communications, WiFi, cellular Long Term Evolution (LTE), dedicated wire lines, and WiMAX. Each of these technologies is supported in ns-3 or available as an add on

developed by the user community. Users may create and save their own network models that implement use cases other than meter reading for the AMI via the modeling interface shown in Figure 6.

6.1 AMI Use Cases

The communications (T1 thru T4) and NAN technologies supported by each of the AMI use cases are:

T1: WAN:fiber pt-to-pt NAN: wireless mesh
T2: WAN:LTE NAN: wireless mesh
T3: WAN:fiber pt-to-pt NAN: PLC
T4: WAN:LTE NAN: PLC

where
LTE → Long Term Evolution
NAN → Neighborhood Area Network
PLC → Power Line Communications
WAN → Wide Area Network

The use cases (UC) initially being implemented are:

UC1: On Demand Meter read
Description: Meters may be read on demand to retrieve missing information and terminate or start up new customer accounts.

UC2: On Demand Meter read failure
Description: Meters are hardware devices that may fail during an on-demand meter reading request. Communication errors or meter reading failures may occur requiring a failure notification to be sent to an upstream device.

UC3: On Demand Meter interval period read
Description: Utilities typically acquire customer consumption data at least once per day, typically at midnight for validating meter intervals.

UC4: Normal Meter Reading Operations
Description: Utilities may acquire consumption data 4 to 6 times per day with each acquisition obtaining interval usage information from 15 minutes to 1 hour in duration.

UC5: Bulk Meter Interval data read
Description: A MDMS (Meter Data Management System) may initiate a large bulk meter data request for billing purposes. This will include data from on-demand or normal meter readings.

The ns-3 code generator is a key component of the PGCPMT. The ns-3 code generator must support each of these use case technologies for generating the appropriate simulation code.

6.2 AMI Networking Diagrams

An AMI networking diagram applicable for these use cases is shown in Figure 10. The Head End (HE) computer communicates with the DC and MDMS over an IP connection. Meter data exchanges, commands and responses may occur between the HE and the MDMS. The MDMS may communicate to a smart meter only through the HE computer. The DC communicates with the meters and collects periodic measurements and alarms as well as sending commands from the HE to a meter(s).

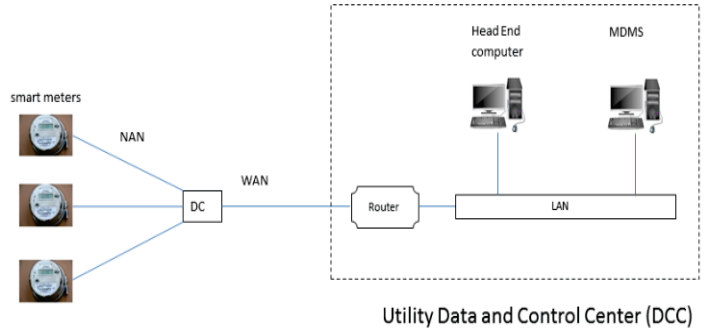


Figure 10. AMI networking.

A logical network diagram for this AMI network is shown in Figure 11. The text tool A has been used to annotate and clarify the components in the diagram. Possible technologies for the WAN include fiber point-to-point link, WiMAX, or LTE technology and for the NAN include wireless mesh, PLC, or WiFi technology.

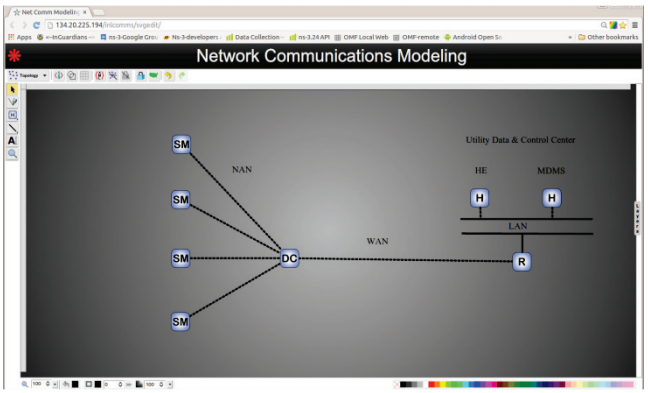


Figure 11. Logical network model for AMI network.

The implementation of the AMI network builds upon the DLMS/COSEM package in [5] for the gathering of meter data. This package includes helper classes for a data concentrator application, meter data management application, demand response application, and smart meter to data concentrator communications.

The DC must support both WiFi and LTE technologies – WiFi for communicating with the smart meters and LTE for a communications path to the Data and Control Center (DCC) computers. The router will interface with the Packet Data Network Gateway (PGW) of the LTE network. The PGW provides connectivity to external packet data networks via a point-to-point link. DR and MDMS applications may run on the HE, MDMS as well as other computers that are part of the Utility DCC.

Several applications are part of this AMI network simulation:

- DLMS/COSEM
- Data Concentrator
- Meter Data Management
- Demand Response

DLMS/COSEM is currently based on the IEC 62056 standard for electricity metering data exchange. The Cosem application server and client, based on the User Datagram Protocol, are installed on each smart meter and DC, respectively. The DLMS/COSEM application will be modified in the future to support ANSI C12.22

– the ANSI standard protocol used in two-way communications with a meter in North America.

A Data Concentrator application is installed on the Data Concentrator as well. This application enables the DC to communicate with each smart meter.

Meter Data Management (MDM) and DR applications are installed on the DCC computers on the LAN. The MDM application enables the DCC to send requests to the AMI network's DC and to receive consumption data from the DC.

The DR application enables sending data to the DC and receiving data from the MDMS. Demand response is the managing of increased demand by reducing demand and/or increasing supply. The DR application uses the DLMS/COSEM protocol application and is dependent upon the meter data collected by the MDMS. Control/curtailment commands and updated pricing rates may be sent to the meters.

6.3 AMI Simulation Output Results

Meter reading and the AMI network have latency and reliability requirements as well as payload sizes and timed events for meter reading. Users may have specified these latency and reliability requirements as well as measurements and data to be collected and reported on over the links between nodes, as previously shown in Figure 9.

Possible simulation output results to assist the user in determining if the planned network meets these latency and reliability requirements are shown in Table 1 for a normal meter reading use case. The performance indicator is the performance requirement relevant to the smart grid topology. The planned performance is the performance expected over the WAN, from smart meters to the end of the AMI network such as the HE computer, and the reliability and latency requirements from the smart meter to the DC. The planned performance data are examples from the Smart Grid Network Systems Requirement Specification [9]. The actual average performance is the average of these measurements over a given time period.

The green, yellow, and red status indicators in the Table are simple visual indicators to give the user a quick view of whether the planned network predicted performance is within pre-specified bounds. Green indicates the actual performance matches the expected performance, yellow less than expected performance and red significantly less than expected performance. The boundary values for these visual indicators may be specified by the user.

Table 1. Network and meter performance data.

Performance Indicator	Planned Performance	Actual Average Performance	Status
End-to-end (ETE) reliability	>98%	100%	
ETE latency	<15 sec	17 sec	
WAN reliability	>98%	100%	
WAN latency	>98%	100%	
SM reliability	<15 sec	10 sec	
SM latency	<15 sec	24 sec	

The PGCPMT uses the ns-3 Flow Monitor to evaluate the network model based on the previously described performance metrics (see Section 5.4) chosen by a user. Flow Monitor tracks the packets exchanged by nodes and measures a number of parameters. An example output for a network flow is as follows.

UDP 10.0.0.10/49153 ----> 10.0.1.1/999
Tx bitrate:+9.17979797979798103100 kbps
Rx bitrate:+9.18220427381494926376 kbps
Mean delay:6.31181 ms
Packet Loss ratio: 0%
First Tx Packet: 1.1 secs.
First Rx Packet: 1.11526 secs.
Last Tx Packet: 25.85 secs.
Last Rx Packet: 25.8588 secs.
Delay Sum: 0.631181 secs.
Jitter Sum: 0.200328 secs.
Last Delay: 0.0087716 secs.
Tx Bytes: 28400
Rx Bytes: 28400
Tx Packets: 100
Rx Packets: 100
Lost Packets: 0
Times Forwarded: 100
Throughput: 8.96147 Kbps

A differentiating capability built into the PGCPMT is the cyber security module presented in the next section.

7. CYBER SECURITY MODULE

PGCPMT includes a cybersecurity module that provides users with a systematic and repeatable approach for assessing the cybersecurity posture of their grid network components. The Cyber Security Evaluation Tool (CSET) [10] is being used as a model for this implementation.

Users are guided through a step by step process using a question/answer approach to collect specific system information that addresses topics such as hardware, software, administrative policies, and user obligations. The rigors of the questions are determined by the criticality of the components themselves in addition to the confidentiality, integrity, and availability data specified in the diagram. The user may select from a list of standards to apply to their network as shown in Figure 12. The list of generated questions will be dependent upon the standard selected by the user.

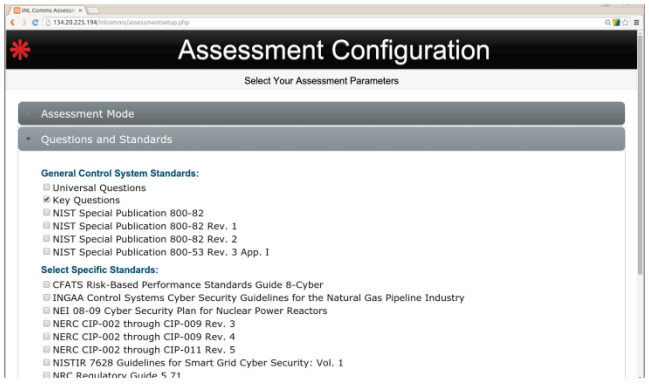


Figure 12. Assessment configuration screen.

The tool compares user entered information to selected relevant security standards and regulations, assesses overall compliance, and provides appropriate recommendations for improving the system's cybersecurity posture. Recommendations are pulled

from a database of the best available cybersecurity practices that have been adapted specifically for application to control system networks and components. Where appropriate, recommendations are linked to a set of actions to remediate specific security vulnerabilities.

The primary objective of the questionnaires is to reduce the risk of cyber-attacks by identifying current vulnerabilities within power-grid architectures. This offers the following benefits:

1. Comprehensive evaluation and comparison to existing industry standards and regulations.
2. Identification of potential vulnerabilities in the system design and security policies.
3. Access to a centralized repository of cybersecurity requirements.

The results of the questionnaire and evaluation may be reviewed through an analysis screen where charts present both summary and detailed information. By drilling down, the application will open new screens that show a finer level of detail until reaching the list of actual questions and answers as shown in Figure 13. All missed questions will be ranked in the order of recommended priority.

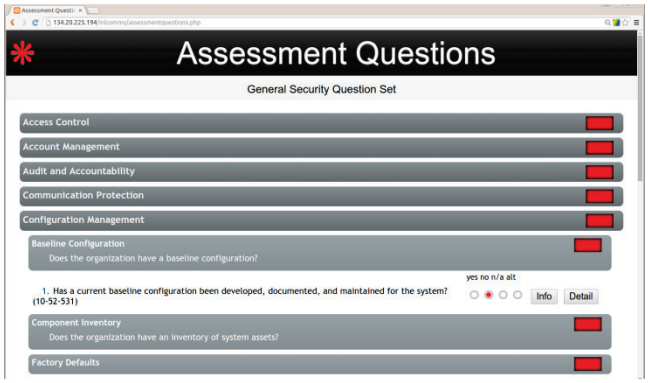


Figure 13. Drill Down into questions screen.

A summary report screen, Figure 14, is displayed when the user has completed answering the security questions. The summary report may then be downloaded to the user's computer.

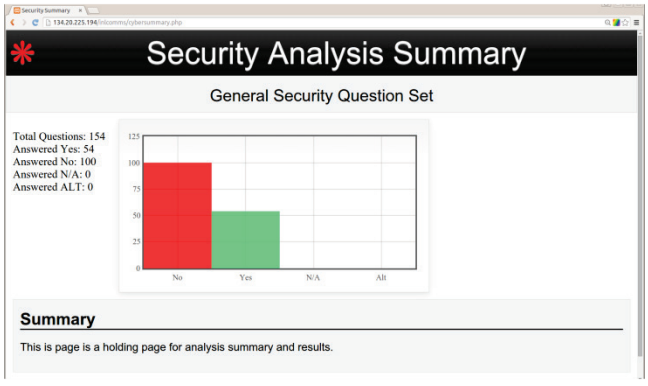


Figure 14. Security analysis summary.

8. CONCLUSION AND FUTURE WORK

PGCPMT provides a much needed capability for ns-3 focused on utility communications – an easy way to create and evaluate the predicted performance of smart grid network models without programming. PGCPMT abstracts away the details of programming in ns-3. Features of the tool are in different stages of implementation and require future work, such as cost estimation and planning, component database, additional network performance reporting options, use case support for new domains, and simulation run comparisons.

This paper describes the vision of the evolving capabilities of PGCPMT. Suggestions and recommendations for tool features may be sent to the author. PGCPMT will be made open source when the initial version is fully implemented and tested. The goal is to make PGCPMT extensible by other developers for adding capabilities and making those capabilities available to the open source community so that all users and developers may benefit from their efforts.

9. ACKNOWLEDGMENTS

The Department of Energy Office (DOE) of Electricity Delivery and Energy Reliability (OE) has the vision of providing tools to assist utilities in accelerating the deployment of smart grid technologies. The Idaho National Laboratory development team thanks the DOE OE for supporting the development of the PGCPMT.

10. REFERENCES

[1] CORE Documentation, Release 4.8, core-dev, June 5, 2015.

[2] A. Leclrec and M. Crosby, Test and Evaluation of WiMAX Performance Using Open Source Modeling and Simulation Software Tools, ITEA Journal, 31, pp. 518-524, 2010.

[3] I. Brown, Radio Mobile – What Can It Do For You?, antenneX issue no. 147, July 2009.

[4] A. Quereilhac, NEP Network Experiment Programming Interface, INRIA Sophia Antipolis, France.

[5] R. Bustamente and J. Aranda, Modeling and Simulation of AMI Network Implemented under LTE and WiFi Technologies, Universidad de los Andes, Electric and Electronic Engineering Department, December 7, 2012.

[6] T. Henderson, C. Dowell, J. Ahrenholz, T. Goff, and B. Adamson, Virtual Machines and ns-3.Workshop on ns-3, March 2010.

[7] D. Pinney, Open Modeling Framework software, https://github.com/dpinney/omf.

[8] D. Chassin, J. Fuller, GridLAB-D, A Unique Tool to Design the Smart Grid, November 2012.

[9] OpenSG SG-Network 119 Task Force Core Development Team, Smart Grid Networks System Requirements Specification, Release Version 5.

[10] Cyber Security Evaluation Tool, ICS-CERT, Industrial Control Systems Cyber Emergency Response Team, https://ics-cert.us-cert.gov/.

Getting Kodo: Network Coding for the ns-3 Simulator

Néstor J. Hernández M.
Steinwurf ApS, Aalborg University
Aalborg, Denmark
nestor@steinwurf.com

Morten V. Pedersen
Steinwurf ApS
Aalborg, Denmark
morten@steinwurf.com

Péter Vingelmann
Steinwurf ApS
Dunaújváros, Hungary
peter@steinwurf.com

Janus Heide
Steinwurf ApS
Aalborg, Denmark
janus@steinwurf.com

Daniel E. Lucani
Aalborg University
Aalborg, Denmark
del@es.aau.dk

Frank H. P. Fitzek
Techn. Universität Dresden
Dresden, Germany
frank.fitzek@tu-dresden.de

ABSTRACT

Network Coding (NC) has been shown to improve current and upcoming communication systems in terms of throughput, energy consumption and delay reduction. However, today's evaluations on network coding solutions rely on homegrown simulators that might not accurately model realistic systems. In this work, we present for the first time the steps to use Kodo, a C++11 network coding library into the ns-3 simulator and show its potential with basic examples. Our purpose is to allow ns-3 users to use a flexible and reliable set of network coding functionalities together with the technologies simulated in ns-3. Therefore, in this paper we (i) show how to set up the Kodo library with ns-3, (ii) present the underlying design of the library examples, and (iii) verify the performance of key examples with known theoretical results.

CCS Concepts

•**Networks** → **Network simulations;** *Packet-switching networks;* •**Mathematics of computing** → **Coding theory;** •**Computing methodologies** → **Simulation tools;** •**Software and its engineering** → **Software libraries and repositories;**

Keywords

Network Coding, C++, ns-3, simulator

1. INTRODUCTION

Since its inception, network coding [14] has been a disruptive technology that allows intermediate network nodes to combine packets, instead of just routing them, resulting in increased throughput, reliability, and lower delay. NC implementations have also corroborated these promised gains under specific scenarios [15, 23, 24, 27, 29, 30].

In most previous implementations, the Kodo C++11 network coding library [31] was used. Kodo is intended to make network coding implementations available to both researchers and commercial entities, in particular those developing protocols. Kodo provides fast implementations of finite field arithmetics and the encoding, decoding and recoding fnctionalities for a variety of network codes, including Random Linear Network Coding (RLNC) [22], Perpetual [21] and Fulcrum network codes [28]. The library is continuously tested to support a large number of operating systems, compilers and architectures with hardware acceleration (SIMD) [6]. Hence, Kodo has been designed to ensure performance, testability and flexibility.

An important part of the evaluation process for these protocols is the simulation stage that aids developers to verify analytical results, rethink the modeling process by including unobserved system effects or proceed with a given design. Through the research community, the ns-3 project [8] aims to develop and establish an open network simulation environment for research. Among the project's goals are: simulation of standard technologies, simple usage and debugging, code testing and documentation that caters to the needs of the simulation workflow. Although there has been various initiatives to develop simulations tools in the network coding environment, [1, 9, 12, 18], most of these simulators: (i) may not be continuously maintained and tested, (ii) may rely on former functionalities of its components and (iii) are hard to integrate with standard technologies. Thus, to date there are no accurate network coding libraries that are well-tested and maintained to interact with deployable network simulation environments. Hence, in this work we provide for the first time, a set of examples compliant with ns-3 using Kodo as an external library for network coding where we verify know and expected results from the NC literature.

Our work is organized in the following way: Section 2 provides the theoretical aspects regarding the encoding, decoding and recoding of RLNC packets indicating some application scenarios. Section 3 shows how to get the Kodo library for ns-3 in an easy and rapid fashion. Section 4 describes the design and implementation details of our examples. Section 5 provides known verifiable results in the NC literature using several ns-3 simulations to validate the examples. Final conclusions of our work are drawn in Section 6.

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915389

2. NETWORK CODING BASICS

Kodo implements core functionalities of intra-session NC (i.e., where data packets from a single flow are combined with each other). In this type of network coding, the original data $P_j, j \in [1, g]$, each of B bits, is used to create coded packets. In the following subsections, we describe the basic functionalities of RLNC [22], namely encoding, decoding and recoding. Later, we mention applications that could potentially benefit from including RLNC as a coding scheme. More complex code variants available in Kodo are described in more detail in [3].

2.1 Encoding

With RLNC, each coded packet is a random linear combination of the original set of packets. Hence, a linearly independent (l.i.) set of g coded packets, $C_i,\ i \in [1, g]$ is required in order to get the original information. Each original packet is considered as a concatenation of elements from a Galois Field (GF) of a given size q, which we denote $GF(q)$. To create a coded packet, a coding coefficient $v_{i,j}$, is chosen at random from $GF(q)$ for every packet P_j and multiplied and added following the respective GF arithmetics. In this way, a coded packet is:

$$C_i = \bigoplus_{j=1}^{g} v_{i,j} \otimes P_j,\ \forall i \in [1, g] \tag{1}$$

To indicate which packets were used to generate a coded packet, one form is to append its coding coefficients. In this case, the overhead included for $C_i,\ \forall i \in [1, g]$ by the coding coefficients is given by:

$$|v_i| = \sum_{j=1}^{g} |v_{i,j}| = g \times \lceil \log_2(q) \rceil \text{ [bits]} \tag{2}$$

2.2 Decoding

To perform decoding, we define $\mathbf{C} = [C_1 \dots C_g]^T$ and $\mathbf{P} = [P_1 \dots P_g]^T$. Then, decoding reduces to solve the linear system $\mathbf{C} = \mathbf{V} \cdot \mathbf{P}$ using Gaussian elimination [19]. Here, the coding matrix $\mathbf{V}$ contains any set of g linearly independent packets C_i as rows as follows:

$$\mathbf{V} = \begin{bmatrix} \overline{v_1} \\ \vdots \\ \overline{v_g} \end{bmatrix} = \begin{bmatrix} v_{1,1} & \dots & v_{1,g} \\ \vdots & \ddots & \vdots \\ v_{g,1} & \dots & v_{g,g} \end{bmatrix} \tag{3}$$

The decoder begins to compute and remove the contributions from each of the pivot elements, e.g. leftmost elements in the main diagonal of (3), to reduce $\mathbf{V}$ to reduced echelon form. In this way, it is possible to recover the original set of packets.

2.3 Recoding

Network coding allows intermediate nodes in a network to recombine (or recode) packets from their sources whether they are coded or not. In general, a recoded packet should be indistinguishable from a coded one. Thus, we define a recoded packet as R_i and its corresponding encoding vector as w_i with coding coefficients $[w_{i,1} \dots w_{i,g}]$, as follows:

$$R_i = \bigoplus_{j=1}^{g} w_{i,j} \otimes C_j,\ \forall i \in [1, g] \tag{4}$$

$$\tag{5}$$

In (4), $w_{i,j}$ is the coding coefficient that multiplies C_j, uniformly and randomly chosen from $GF(q)$. Any decoder that collects $R_i, i \in [1, g]$ linearly independent coded packets, with their respective w_i, will be able to decode the data as mentioned before.

2.4 Network Coding Applications

There are numerous situations where NC provides benefits over conventional routing schemes. Basic gain descriptions and practical use cases for network coding can be found in [19, Sections 3,4] covering various areas. Among different benefits for communications, network coding can achieve the capacity for networks with multicast flows [26], improve content distribution in peer-to-peer networks [20] or enhance throughput in conventional Transmission Control Protocol (TCP) protocols for reliable communication [25]. For distributed storage systems, network coding has found applications in scenarios where it could incur less redundancy for data protection than simple replication [16].

3. GETTING KODO FOR NS-3

In this section, we explain how to get the Kodo up and running. The procedure helps to quickly add new coding functionalities in ns-3. The project with the examples is available in [4] under a GPLv2 license and it tracks the latest stable revision of the ns-3 development repository, `ns-3-dev`, to get the most recent changes. For research purposes, Kodo uses a free research license detailed in [10].

A more detailed setup guide can also be found at [4]. A descriptive tutorial for the project is available at [5]. We strongly encourage any developer to follow the setup guide. As a reference for this guide, we assume that the ns-3 project is in the `~/ns-3-dev` folder on the developer's system.

1. To get access to Kodo, it is necessary to submit a request at [11] for a research license.
2. Build the local ns-3 repository with its examples since the Kodo examples need the ns-3 binaries in order to build itself. Execute in the local `ns-3-dev` folder:
   (a) `python waf configure --enable_examples`
   (b) `python waf build`
3. Go to ~ and clone the `kodo-ns3-examples` git repository. At this point, a confirmed license is necessary to get the Kodo dependencies.
4. Go to the new `kodo-ns3-examples` folder and configure with `python waf configure` (Kodo also uses the `waf` [13] build system) to set and compile the project and its dependencies.
5. Build the `kodo-ns3-examples` and install all the needed files for ns-3 in the `~/ns-3-dev/examples/kodo` folder with `python waf build install --ns3_path="PATH"`. In this case, `"PATH"` would be `~/ns-3-dev`.

6. Get back to ns-3 folder and build the local ns-3 project with `python waf build`. At this point, the examples should be available to run as any ns-3 simulation.

4. KODO EXAMPLES FOR NS-3

In this section we describe our design implementation and criteria for creating the examples, an overview of what do the examples simulate and the design of two helpers that provide the coding operations for the system represented in the examples. The helpers function is to serve as an interface between ns-3 and the Kodo C++ bindings [2]. These are high level wrappers for the core functionalities of Kodo.

4.1 Examples Implementation

To create our examples, we consider an approach where we perform intra-session network coding, between the application and transport layer of the User Datagram Protocol (UDP) / Internet Protocol (IP) model as shown in Fig. 1. Although other approaches apply coding between the transport and Medium Access Control (MAC) layer [15,24,27,32], we implement it below the application layer to not alter other layers within the protocol stack and keep the implementation simple.

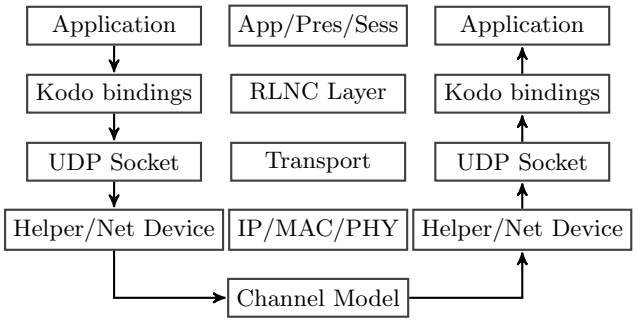


Figure 1: ns-3 + Kodo Implementation Protocol Stack based in a simple UDP/IP model.

In Fig. 1, we consider an application that generates a batch of g packets of some content. For practical reasons, we consider Hyper Text Transfer Protocol (HTTP) traffic and represent it by sending RLNC coded packets through port 80 of a UDP socket from the `UdpSocketFactory`. To encode, recode (if necessary) and decode NC packets, we employ the Application Programming Interface (API) provided by the bindings. We employ UDP datagrams because we consider best effort traffic. For the IP layer, we employ IPv4. For address assignment and routing tables, we use the `InternetStackHelper`, the `Ipv4AddressHelper` and the `Ipv4GlobalRoutingHelper` from ns-3. The details of the MAC, Physical Layer (PHY) and channel models depend on the considered example as we will see.

4.2 Examples Description

With a defined protocol stack, we describe the networks implemented in the examples to evaluate NC performance providing the details for the layers not described previously.

4.2.1 kodo-wifi-broadcast

This example, shown in Fig. 2, simulates a source broadcasting a generation of RLNC packets with generation size g and field size q to N sinks with an IEEE 802.11b WiFi ad-hoc channel. For the MAC we regard it without Quality of Service (QoS) implemented through `NqosWifiMacHelper`. We pick a WiFi MAC without QoS since in principle we are simulating connectionless best-effort traffic. Thus, the ns-3 net devices are constructed through the `WifiHelper`. Also, we turn off unnecessary MAC parameters, namely: frame fragmentation for frames larger than 2200 bytes and RTS / CTS frame collision protocol for the less than 2200 bytes. Although not required within the example, these parameters need to be included in order for the WiFi MAC to work.

For the PHY of this example, we use the `YansWifiPhyHelper`. The considered PHY includes a channel model that accounts for channel delay, path loss and receiver signal strength in dBm. We employ the `FixedRssLossModel` where the receiver signal (`rss`) is set to a fixed value. We set the broadcast data rate to be the same as unicast for the given `phyMode`. As a transmission policy, the sender keeps transmitting coded packets until all the receivers have g l.i. coded packets, even if some receivers are able to decode the whole generation.

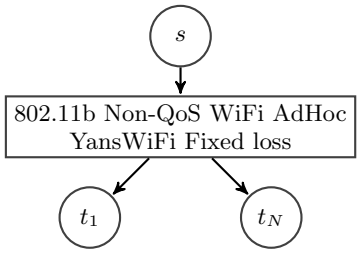


Figure 2: kodo-wifi-broadcast example network.

4.2.2 kodo-wired-broadcast

The example shown in Fig. 3, is similar as in Fig. 2 but instead, we evaluate a basic time-slotted wired system where a node either transmits or receives a single packet in a given time slot with the aid of the `PointToPointHelper`. To model a network with erasures, we consider the `RateErrorModel` for the PHY and channel model. In this case, packets sent from the transmitter could be lost or useless before arriving at the receiver. To control the amount of losses, an `ErrorRate` attribute is included at the `ReceiveErrorModel` attribute of the `RateErrorModel` to indicate the frequency of erasures within a given channel. The resulting topology is a basic representation for packet erasure networks which is akin for network coding applications. The transmission policy is the same as before. For simplicity, all devices are assumed to have the packet erasure rate, $0 \leq \epsilon < 1$. The erasure rate can be introduced as a command-line argument to set the `ErrorRate` attribute from the wired topology as we will see.

4.2.3 kodo-recoders

This example shows the gain of RLNC with recoding in a 2-hop line wired network consisting of a source, N recoders and a sink with different erasure rates. All the links between the sender and the recoders have the same packet erasure rate, $0 \leq \epsilon_{S \to R} < 1$. Equivalently, the packet erasure rate for the links between the recoders and the receivers is the

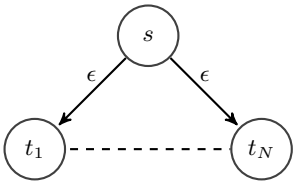


Figure 3: kodo-wired-broadcast example network.

same, $0 \leq \epsilon_{R \to D} < 1$. Again, both recoding and the erasure rates can be modified by command-line parsing. The transmission policy for this case, is as follows: First, packets are sent to each of the recoders. The transmitter stops if the decoder or all the recoders are full rank, e.g. have g l.i. coded packets. Second, a recoder retransmits packets in another scheduled time slot if l.i. packets to transmit and it stops only if the decoder is full rank.

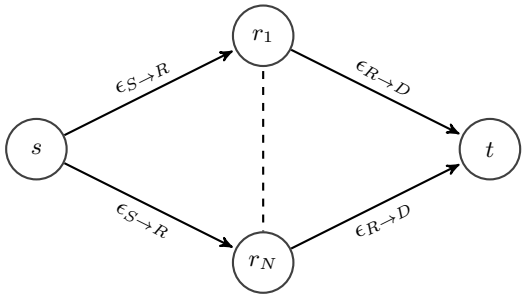


Figure 4: kodo-recoders example network.

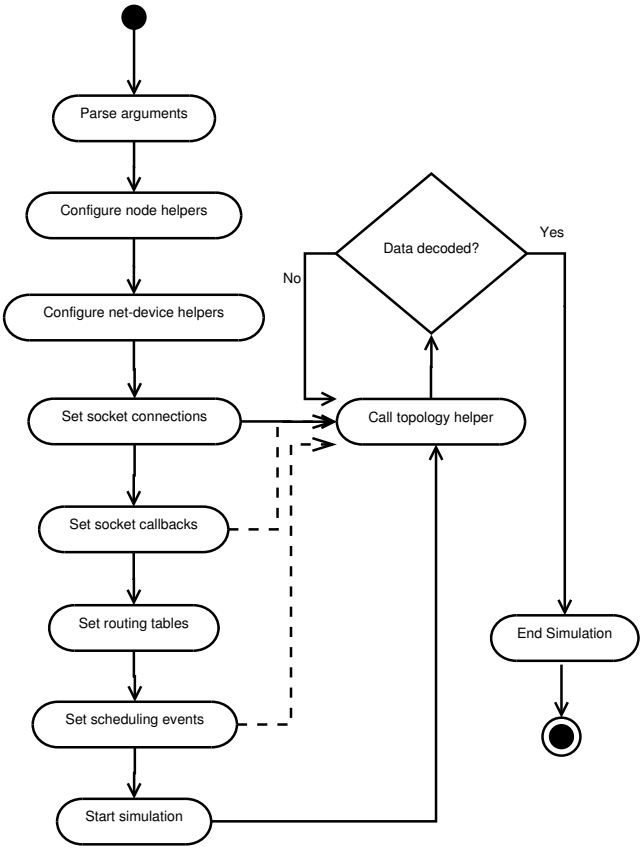


Figure 5: Examples Simulations workflow.

4.3 Simulation Workflow and Helpers

In Fig. 5, we show the workflow of the example's simulation source program. This workflow is standard for ns-3 simulations and consists in defining the network (nodes, net-devices) with ns-3 helpers according the required layer functionality described in Section 4.1. Once the socket connections are defined, we call the topology helper which provides the application and coding layers. A receive callback is set to trigger an action whenever a packet is received in a decoder socket. When an encoding or decoding action has been performed, a new event is scheduled through the `ns3::Simulator` class. Events are scheduled until a generation originated in the source is decoded by the sink(s) in the evaluated example.

At the core of each example implementation resides a topology helper which contains all the encoding, recoding and decoding parameters and functionalities of the RLNC layer, the transmission policy and eases the socket connections made in each source file. The helpers are classes that serve as interfaces between the bindings and ns-3. To accomplish this, the helpers are included in ns-3, but its basic elements are objects from the Kodo C++ bindings. For our case, we use two helpers. For `kodo-wifi-broadcast` and `kodo-wired-broadcast`, we use the `Broadcast` topology helper . For the `kodo-recoders` example, we utilize the `Recoders` topology helper. In this section we present the API of these helpers in order to show the interface between Kodo and ns-3. To do so, we elaborate an Unified Modeling Language (UML) class diagram to visualize the relationships between our bindings and the helpers. We make this review only for the `Broadcast` topology helper, since the analysis for the `Recoders` topology would be similar.

Fig. 6 shows the UML class diagram for the `Broadcast` topology. We have indicated the most important classes that have a type of dependency with the bindings. Also, we employ the UML package notation to indicate the namespace where all the bindings reside. We describe the topology members where the links with `kodocpp` occur. Later, we give an overview of other members whose type are natively contained in the C++ standard library or ns-3. We list the members with dependency on `kodocpp` according to their functionality.

4.3.1 Code Parameter Members

First, `m_codeType` stands for the type of erasure correcting codes utilized. In our implementation, an instance of `kodocpp::codec` is passed to the source program. The available codecs in the bindings are: `full_vector`, `on_the_fly`, `sliding_window`, `sparse_full_vector`, `seed`, `sparse_seed`, `perpetual`, `fulcrum` and `reed_solomon`. A complete description of each codec can be found in the overview section of the Kodo documentation [3]. Second, `m_field` indicates the finite field of the coding scheme. An instance of `kodocpp::field` is passed to the source program. For the available fields: `binary`, `binary4` and `binary8` represent $GF(2)$, $GF(2^4)$ and $GF(2^8)$ respectively.

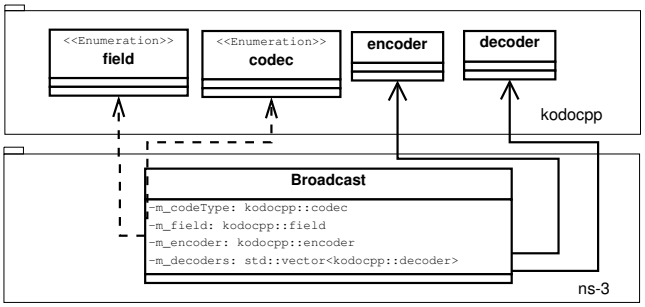


Figure 6: UML class diagram for the Broadcast topology helper interface.

4.3.2 Encoder / Decoder Object Members

The encoder data type is `kodocpp::encoder`, which is provided by the bindings. However, the encoder is not aware of the topology on its own, thus the uni-directional association link to indicate this in Fig. 6. The encoder class is a child class of the more general `kodocpp::coder` abstract class. In this way, the encoder class contains both own and general functionalities, inherited from `kodocpp::coder`, to configure its basic parameters and generate coded data. Similarly, we employ `std::vector<kodocpp::decoder>` to get a local decoder instance for each sink socket. As before, it contains functionalities to configure itself, read coded data and signal when to stop transmissions.

4.3.3 Sockets and Transmission-State Members

For packet transmissions and receptions, we use the native `ns3::Ptr<ns3::Socket>` class. Only the policies for packet transmissions/receptions are implemented through the methods `SendPacket` and `ReceivePacket`. Both of them receive the intended socket for tranmission or reception. In case of the transmitter, the packet interval time (`ns3::Time pktTime`) is also given because this will indicate the transmitter the scheduling time for next transmissions. Finally, other members like the number of users to serve, generation, packet sizes and storage buffers are considered too as standard types.

5. SIMULATIONS

To verify the accuracy of the results provided by the examples, we execute a set of ns-3 simulations to observe the behavior of RLNC in well-known scenarios. For the simulations, we compute the distribution of the number of transmissions required to decode a set of g packets with RLNC.

We only consider this metric since, typically, the time cost for the encoding and decoding operations is much lower when compared to the time spent in conveying the information from a transmitter to a receiver. Still, information regarding encoding and decoding speeds for RLNC can be easily obtained by running the benchmarks in [7] for a given platform. Similar benchmarks exist for other codes as well in their respective repositories. In our scenarios, we consider that an ideal feedback scheme is employed, where the source is aware when any destination has acknowledged all its required coded packets. To get this information, we simply call the bindings API required functions in the topology helpers.

We obtain the distribution in two scenarios. First, we consider the case of one transmitter-receiver pair. Second, we review the scenario of single-hop broadcast for N receivers. We examine these scenarios under two conditions, without packet erasures and with packet erasures. Hence, for the broadcast case, we regard the packet erasure distribution of receiver $j \in [1, N]$ as $Bernoulli(1 - \epsilon_j)$ where ϵ_j is the packet erasure probability. For evaluation purposes, we compute the distribution under a homogeneous packet erasure for all the receivers, $\epsilon_j = \epsilon \; \forall j$.

To accomplish this, we run the `kodo-wired-broadcast` example and get the number of transmissions required to decode the data in 10^4 runs. To get independent runs, the pseudo-random number generator is set to use the default seed and the `RngRun` parameter is changed in the `RngSeedManager` class by command line parsing.

For the single transmitter and receiver, we use the following parameters: `users` $= 1$, `generationSize` $= 30$, `errorRate` $= 0, 0.1$, and `field` $=$ `binary`, `binary8`. For the broadcast case, we evaluate with `users` $= 10$, `generationSize` $= 50$, `errorRate` $= 0.1, 0.2$, and `field` $=$ `binary`, `binary8`. To verify our simulations, we compare the practical results with analytical ones. To do so, we compute the Probability Mass Function (pmf) as [33, Eqs. 11-12] for the single receiver and [17, Eq. 3, Sec. III-B] for the broadcast case. Then, we plot the pmf of the analytical distributions against the simulation results.

5.1 RLNC Probability Mass Function

Fig. 7 shows the result fors the pmf of the single receiver for the evaluated parameters. We present the results for $g = 30$, $\epsilon = [0,\ 0.1]$ with $GF(2)$ and $GF(2^8)$ to observe the effect of linear independence in packets transmissions. We also evaluate the consequences of packet erasures in the number of transmissions required for decoding. In all the results, it can be clearly seen that the analytical calculations matches the simulations obtained from ns-3. For the case of no erasures, employing RLNC with $GF(2)$ requires more transmissions compared with $GF(2^8)$ since the possibilities for selecting the coding coefficients are much reduced for the last packets. For the erasure case, the transmissions alos increase given that packets might be lost regardless of linear dependency, but still are less that when employing a higher field size.

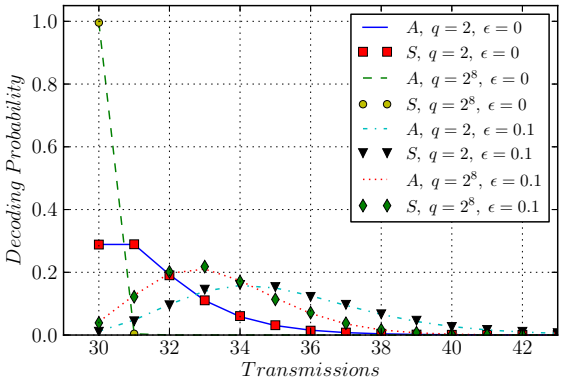


Figure 7: Analytical (A) vs. Simulation (S) for Unicast with 1 receiver and 30 packets.

5.2 RLNC Broadcast Probability Mass Function

Fig. 8 shows the result for the pmf of broadcast with RLNC for the case of 10 receivers and the evaluated parameters. In this scenario, $g = 50$. The selected fields are $GF(2)$ and $GF(2^8)$. We present the results for two erasure rates in all the links, $\epsilon = [0.1, 0.2]$.

Again, we observe the theoretical computations fit the simulations results. A difference that can be noticed with the single receiver case is the number of transmissions required to decode increases much more. Excluding the field and the erasure effects, the difference arises from all the receivers being required to get g l.i. coded packets in order to be able to decode. This is the main reason why the pmfs do not start to show a significant non-zero probability of decoding at g transmissions and shortly afterwards.

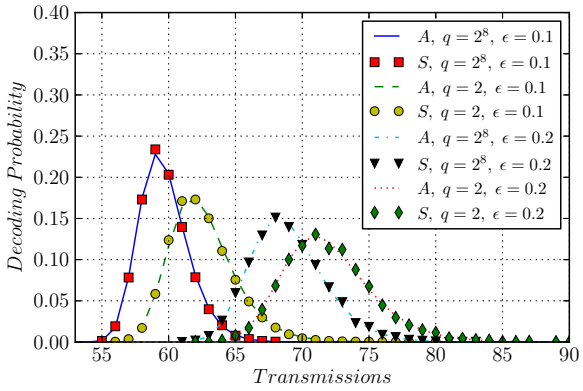


Figure 8: Analytical (A) vs. Simulation (S) for Broadcast with 10 receivers and 50 packets.

6. CONCLUSIONS

Given the increasing amount of NC applications from both academia and industry, we introduced a framework for using the Kodo library with ns-3. We hope that our contribution helps to cover the need for NC simulation capabilities in ns-3. With a set of examples where NC provides known gains, we show that our library complies with the expected results. Although the examples are made for particular topologies, the deployment of different topologies or scenarios could be easily extended by the user as detailed in [4]. Future work will be to simulate RLNC with other technologies, such as Long Term Evolution Advanced (LTE-A) within ns-3.

7. ACKNOWLEDGMENTS

This research has been financed by the CROSSFIRE MITN Marie Curie project (317126) from the European Comission FP7 framework, the Green Mobile Cloud project (Grant No. DFF - 0602 - 01372B) and the TuneSCode project (Grant No. DFF - 1335-00125) both granted by the Danish Council for Independent Research.

8. REFERENCES

[1] Inter-session network coding simulator for matlab. http://www.mathworks.com/matlabcentral/fileexchange/53750-network-coding-simulator.

[2] Kodo c++ bindings git repository. https://github.com/steinwurf/kodo-cpp.

[3] Kodo documentation read-the-docs codecs overview. http://kodo-docs.steinwurf.com/en/latest/overview.html.

[4] Kodo examples for the ns-3 simulator git repository. https://github.com/steinwurf/kodo-ns3-examples.

[5] Kodo-ns3-examples documentation read-the-docs tutorial. http://kodo-ns3-examples.readthedocs.org/en/latest.

[6] Kodo platform support. http://steinwurf.com/kodo-specifications.

[7] Kodo-rlnc git repository. https://github.com/steinwurf/kodo-rlnc.

[8] ns-3 website. https://www.nsnam.org.

[9] Software related to network coding. http://www.ifp.illinois.edu/~koetter/NWC/Software.html.

[10] Steinwurf research license. http://steinwurf.com/research-license.

[11] Steinwurf research license webpage. http://steinwurf.com/license.

[12] Universidad de cantabria network coding implementation on ns-3.13. https://github.com/dgomezunican/network-coding-ns3.

[13] Waf. the metabuild system webpage. https://waf.io.

[14] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000.

[15] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. *SIGCOMM Comput. Commun. Rev.*, 37(4):169–180, 2007.

[16] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inf. Theor.*, 56(9):4539–4551, 2010.

[17] A. Eryilmaz, A. Ozdaglar, M. Médard, and E. Ahmed. On the delay and throughput gains of coding in unreliable networks. *Information Theory, IEEE Transactions on*, 54(12):5511–5524, 2008.

[18] D. Ferreira, L. Lima, and J. Barros. Neco: Network coding simulator. ICST, 5 2010.

[19] C. Fragouli, J.-Y. Le Boudec, and J. Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68, 2006.

[20] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE INFOCOM*, number MSR-TR-2004-80, page 12, 2005.

[21] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and M. Médard. Perpetual codes for network coding. *CoRR*, abs/1509.04492, 2015.

[22] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *Information Theory, IEEE Transactions on*, 52(10):4413–4430,

2006.

[23] M. Hundebøll, J. Leddet-Pedersen, J. Heide, M. Pedersen, S. Rein, and F. Fitzek. *CATWOMAN: Implementation and Performance Evaluation of IEEE 802.11 based Multi-Hop Networks using Network Coding*, pages 1–5. IEEE Press, 9 2012.

[24] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: Practical wireless network coding. *IEEE/ACM Trans. Netw.*, 16(3):497–510, 2008.

[25] M. Kim, T. Klein, E. Soljanin, J. Barros, and M. Médard. Modeling network coded tcp: Analysis of throughput and energy cost. *Mobile Networks and Applications*, 19(6):790 – 803, December 2014.

[26] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Netw.*, 11(5):782–795, 2003.

[27] J. Krigslund, J. Hansen, M. Hundebøll, D. Lucani, and F. Fitzek. *CORE: COPE with MORE in Wireless Meshed Networks*, pages 1–6. IEEE, United States, 2013.

[28] D. E. Lucani, M. V. Pedersen, J. Heide, and F. H. P. Fitzek. Fulcrum network codes: A code for fluid allocation of complexity. *CoRR*, abs/1404.6620, 2014.

[29] A. Paramanathan, P. Pahlevani, S. Thorsteinsson, M. Hundebøll, D. Lucani, and F. Fitzek. Sharing the pi: Testbed description and performance evaluation of network coding on the raspberry pi. In *2014 IEEE 79th Vehicular Technology Conference*, 2014.

[30] A. Paramanathan, M. Pedersen, D. Lucani, F. Fitzek, and M. Katz. Lean and mean: Network coding for commercial devices. *IEEE Wireless Communications Magazine*, 20(5):54 – 61, 2013.

[31] M. Pedersen, J. Heide, and F. Fitzek. Kodo: An open and research oriented network coding library. In *Networking 2011 Workshops*, volume 6827 of *Lecture Notes in Computer Science*, pages 145–152. Valencia, Spain, 2011.

[32] H. Seferoglu, A. Markopoulou, and K. K. Ramakrishnan. I2nc: Intra- and inter-session network coding for unicast flows in wireless networks. In *INFOCOM*, pages 1035–1043. IEEE, 2011.

[33] O. Trullols-Cruces, J. M. Barcelo-Ordinas, and M. Fiore. Exact decoding probability under random linear network coding. *Communications Letters, IEEE*, 15(1):67–69, 2011.

Improving ns-3 Emulation Performance for Fast Prototyping of Network Protocols

Helder Fontes
INESC TEC and
Faculdade de Engenharia
Universidade do Porto
Portugal
heldermf@gmail.com

Tiago Cardoso
INESC TEC and
Faculdade de Engenharia
Universidade do Porto
Portugal
ei10066@fe.up.pt

Manuel Ricardo
INESC TEC and
Faculdade de Engenharia
Universidade do Porto
Portugal
mricardo@inesctec.pt

ABSTRACT

A common problem in networking research and development is the duplicate effort of writing simulation and implementation code of network protocols. This duplication can be avoided through the use of fast prototyping development processes, which enable reusing simulation code in real prototyping and in production environments. Although this functionality is already available by using ns-3 emulation, there are still limitations regarding the additional packet processing that emulation introduces, which degrades the node's performance and limits the amount of network traffic that can be processed.

In this paper we propose an approach to reduce the performance problem associated with fast prototyping that consists in migrating data plane operations processing to outside of ns-3. In a well-designed network, most of the traffic should be data. By moving the data plane operations outside of ns-3 the overhead associated with this kind of traffic is greatly reduced, while control plane protocols may still be reused.

In order to validate our proposed solution, we extended the Wireless Metropolitan Routing Protocol (WMRP) and Optimized Link State Routing (OLSR) protocols to use the developed architecture, tested their performance in real environments, and verified the amount of code reuse between the simulator and the real system.

CCS Concepts

•Networks → Network protocols; •Computing methodologies → Real-time simulation; •Software and its engineering → Reusability;

Keywords

ns-3; Network Simulation; Network Emulation; Fast Prototyping

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915374

1. INTRODUCTION

The development of new protocols for communication systems generally involves four phases: design, evaluation in a network simulator, evaluation in a real testbed, and deployment.

The first phase consists in the design of a protocol concept that is intended to solve a given problem. From this concept, the researchers can implement a protocol model to be used in simulation. Network simulation is a tool of great value in the evaluation of communication protocols because the variables that influence the network scenario can be easily controlled to create specific and reproducible test conditions; It is also an environment that is easily observable and much cheaper to deploy than a testbed with real hardware. However, evaluation on a testbed must still be done, eventually to test how the protocol behaves in real-world conditions which gives more accurate and credible results. Finally, the protocol can be deployed in the desired systems to solve the problem for which it was initially designed.

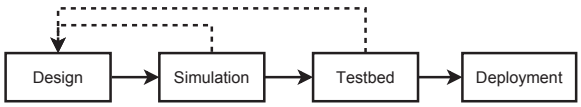


Figure 1: Development process of protocols for communication systems.

This development process is usually iterative (Figure 1). For example, findings in one of the evaluation phases may force the researchers to go back to the design phase and reevaluate some aspects of the protocol. This iteration is necessary to improve the protocol but it is a drawback related to how the simulation model and the protocol testbed implementation are usually developed.

Traditionally, a simulation model of the protocol is created from the concept designed in the first phase. Multiple simulations are run and the results are analyzed and used to improve the protocol. When the simulation results are acceptable, development can continue to the next phase. A prototype of the protocol is thus implemented in a real system and run in a testbed to be validated. If the results do not meet the expectations then the protocol needs to be changed (first phase) and new simulations carried out (second phase). At this point, two implementations of the protocol are being maintained: the simulation model and the implementation prototype. This situation leads to duplication of effort when a change needs to be made in the

protocol which in turn increases both the development time and the chance of introducing errors in the implementations of the protocol.

Researchers have been trying to find ways to alleviate the problem of having to create and maintain two implementations of protocols that have to be evaluated both in simulation and in testbeds [12]. The solution is **shared protocol implementation**, which consists in developing a single implementation of a protocol that can run both in a simulator and in a real system.

The motivation for this work is thus the need for a shared protocol implementation that allows researchers to reuse the same code in ns-3 simulations and in testbeds while at the same time enabling the use of the full functionality provided by the simulator and providing good performance in both the simulation and the real environment.

A proposed solution for this problem was fast prototyping, a protocol development process in which the simulation code is reused in the testbed with the help of a feature of ns-3 that enables emulation [4]. Emulation allows nodes in a simulation to communicate with the outside world and thus enables them to participate in testbeds. The SITMe [1] testbed, composed by 11 fast prototyped ns-3 nodes installed in buses, was a successfull example of using fast prototyping via ns-3 emulation in complex real world networks [6]. However, it was found that this approach is not scalable to nodes that need to process a large amount of real traffic. The problem with this approach is that emulation introduces overhead to packet processing which degrades the node's performance limiting the amount of network traffic that can be handled.

The goal of this work is thus to propose an architecture that improves the performance of emulated nodes while still allowing most of the code developed for ns-3 to be reused. With improved performance, fast prototyping can represent a viable complementary approach to existing shared protocol implementation solutions such as Direct Code Execution (DCE)[23], but from the ns-3 starting point rather than from the existing implementation starting point. This is especially relevant when creating new routing protocols, where developers may start by an easier to code ns-3 implementation, taking also advantage of the ns-3 tracing capabilities.

This document is structured as follows. In Section 2 we introduce the related work, where the state-of-the-art solutions to shared protocol implementation are presented and analyzed. Then, in Section 3 we present the details of our contribution to improve the Emulation Performance where we introduce three optimizations to run the data plane outside of simulation. In Section 4 we validate each of the proposed optimizations both by analysing the performance gains and the level of code reuse. Finally, in Section 5 we draw the conclusions about the proposed optimizations and the obtained results, and we also introduce some examples of future work to further improve the results obtained in this work.

2. RELATED WORK

In this section we present and analyze the state-of-the-art solutions related to shared protocol implementation.

RapidNet is a toolkit that enables the development of network protocols for simulation and implementation [16]. The protocols are specified in a declarative paradigm using Network Datalog (NDlog), a recursive query language. RapidNet can compile the protocol specification into ns-3 code which can then be used for simulation. The same specification can also be used in a real environment by running the ns-3 code in emulation mode. One of the problems of RapidNet is that it uses a declarative programming paradigm which is unusual in protocol development and results in a steep learning curve. Another problem is that the generated ns-3 code depends on the RapidNet library an it cannot be submitted for inclusion into ns-3 as a new module.

The Protean Protocol Prototyping Library (Protolib) is a toolkit that provides a cross-platform C++ API for the development of network protocols. The developed protocol can run on various systems (including Linux, MacOS, Windows, and FreeBSD among others) and on the ns-2 and OPNET simulators [25]. Some interesting classes provided by Protolib include the following: ProtoSocket, which abstracts the underlying systems TCP and UDP sockets; ProtoTimer, a generic timer class; ProtoRouteMgr, which provides a common interface to the systems routing tables; and ProtoCap, used for raw MAC-layer packet capture. The main disadvantage of Protolib is that it currently only supports ns-2, not ns-3. However, even if it did, it is unclear if it could provide all the features of ns-3 since some of those features are not available on all systems that Protolib also supports.

The Click Modular Router (or just Click) is a flexible software architecture for developing configurable routers [11]. A router can be created by combining small elements in a graph-like architecture. Each element has a defined number of input and output ports and implements some simple functionality such as decrementing a packet TTL or looking up an IP route. The combination of these elements can create routers with complex functionality providing a flexible platform for researchers to experiment with new protocols. Click can run in several systems such as Linux and FreeBSD and there are also tools that integrate it with ns-2 (nsclick [17]) and ns-3 (ns-3-click [22]). The ns-3 simulation can even become more efficient in some cases by using Click to perform the layer 3 functionality, although at the cost of increased memory consumption. Click also has a steep learning curve since it uses a flow-based programming paradigm [14] which is different from the more usual discrete event simulation used in ns-3. In addition, researchers also have to learn the configuration syntax that is used to describe the connections between the elements of a router.

The ns-3 Direct code execution (DCE) module simulator provides an environment that is able to execute Linux protocols in simulation without source code changes [23]. Both userspace and kernelspace protocols or applications can be used, so it is possible, for instance, to use the real ping program or the real Linux networking stack in simulation. There are a number of problems when using the DCE module as a way to achieve a shared protocol implementation. First, there is a loss of performance in the simulation resulting from having to virtualize the protocol [23]. It is also more time consuming to develop protocols in Linux than in ns-3 [4], resulting in a longer development time. Finally, not all of the functionality provided by ns-3 can be used since there is no equivalent in Linux. In some cases an alternative is available, for example, it is possible to add tracing support to a real protocol by using aspect based programming [13], but this is not as straightforward as simply using the ns-3 tracing facilities.

Before the development of ns-3 DCE, the Network Simu-

lation Cradle (NSC) [10] was another framework that could execute a kernel stack in ns-3; however it was hard to maintain because it relied on source code transformations. The ns-3 DCE module can however replace most of the functionality provided by NSC [24].

Simulator-agnostic ns–3 applications are applications that can run both in simulation and in real systems [3]. They are developed as traditional ns-3 applications, but when needed they can communicate with the real environment via real sockets from the underlying system instead of simulated sockets. The actual socket that is used is transparent to the application so its code does not need to be rewritten. This can be done by implementing the real socket functionality as classes derived from the base Socket class of ns-3 and using the factory pattern to create the actual socket as needed (real or simulated). This approach is very efficient [4] and works well for applications; however it is not enough for the needs of a routing protocol. These protocols need more than simple UDP or TCP sockets, they also need to access and alter routing tables. Many protocols also do not use TCP or UDP, but instead use IP datagrams directly (e.g. OSPF [15]). Others may even work on top of layer 2 protocols (e.g. WMRP [20]). Simulator-agnostic ns–3 applications are thus a good complement to other approaches such as the fast prototyping [4], allowing the control plane of a layer 3 protocol developed in ns-3 (operating at a specific TCP or UDP port) to communicate using real system sockets. Although, on their own they can not offer a complete shared protocol implementation solution as they lack the data forwarding capabilities of fast prototyping which processes all the network traffic, including the data plane.

3. DATA PLANE OUTSIDE OF SIMULATION

In a network node there are two planes of operation: the control plane and the data plane. The control plane is responsible, for instance, for discovering and maintaining network routes and ensuring connectivity. The data plane uses the routing information generated by the control plane to forward network packets. In a well designed network, most of the traffic corresponds to data and processing large amounts of packets in ns-3 is less efficient than doing it in userspace or kernelspace. Thus, by moving the data plane operations outside of simulation the overhead associated with this kind of traffic can be greatly reduced.

We propose two alternatives for doing this: executing the data plane in userspace or executing it in kernelspace. These approaches are described in the next sections.

3.1 Data Plane in Userspace

The first approach consists in running the data plane outside of ns-3 in userspace. The control plane is still executed in ns-3 and communicates with the real environment through emulation, but the data plane is executed in a userspace process to avoid the overhead of processing the large amount of data traffic in the simulator (Figure 2).

The two planes are executing in different processes so interprocess communication (IPC) is needed to allow them to exchange information. For example, the control plane needs to update the routing table information in the data plane so that data packets can be forwarded correctly. The data plane may also need to send feedback to the control plane

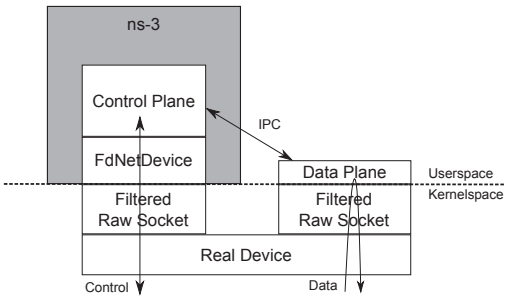


Figure 2: Data plane in userspace.

or request a route in the case of reactive protocols.

The classification of the packets that should be delivered to the control and data planes is done through filters applied to the raw sockets using the Linux Socket Filter (LSF) facility [8]. Listing 1 shows an example filter that accepts only IPv4 packets. At line 0 we load the half word at position 12 in the frame (the EtherType field). Line 1 compares the previously loaded half word to 0x800 (the IPv4 EtherType). If both are equal we jump to line 2 which returns 65535. This will result in the first 65535 bytes of the frame being accepted and delivered to the userspace process. If the comparison of line 2 fails, we jump to line 3 which returns 0. This means that no bytes of the frame are accepted and so it is dropped.

A simple way to obtain the code of socket filters is to run `tcpdump` with the `-d` option. For example, running `tcpdump -i eth0 -d ip` will return the code in Listing 1.

```
0 ldh      [12]
1 jeq      #0x800           jt 2  jf 3
2 ret      #65535
3 ret      #0
```

Listing 1: Linux Socket Filter code that only accepts IP packets.

Socket filters are configured in userspace but they are executed by the kernel on received frames so they are very efficient because packets not accepted by the filter are not delivered to userspace. In this way, the control and data planes only receive their respective traffic, avoiding the overhead caused by the processing of unneeded packets.

Using this architecture, the control plane code can be reused between simulation and testbed but the data plane code needs to be rewritten for the real system. However, because the data plane is usually much simpler than the control plane, only a relatively small amount of code needs to be ported.

3.2 Data Plane in Kernelspace

It is possible to specialize the previous architecture for the case of L3 routing protocols. For protocols of this type, whose objective is to forward IP packets, we can update the kernel routing tables with the information gathered by the control plane. In this way the kernel will be forwarding the traffic it receives (Figure 3) which is even more efficient than doing it in userspace.

In this architecture, a route updater process receives the

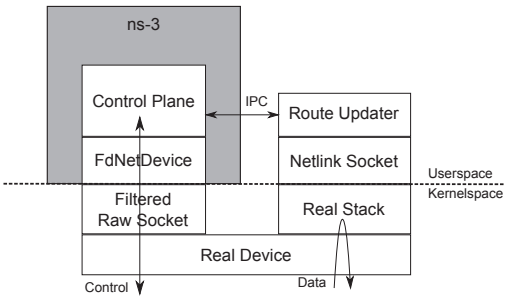


Figure 3: Data plane in kernelspace.

routing information from the control plane and updates the kernel routing tables with that information. To do this, the route updater process uses a netlink socket [21]. Netlink is a communication channel between kernelspace and userspace and, among other functionalities, allows user processes to update and retrieve routing information. Root access is needed to use this functionality so, in order to avoid running the whole simulation as root, the route updater is executed in its own process as root and communicates with ns-3 trough IPC to receive the routes generated by the control plane.

As in the previous architecture, the raw socket used by the control plane needs to be filtered to avoid the overhead of processing unnecessary network traffic.

This approach is more efficient than the previous one since the forwarding is done in the kernel instead of userspace. In addition, there is even less code that needs to be rewritten because the route updater code can be reused by all the protocols that use this architecture. The downside is that this approach is only applicable to proactive L3 protocols, where the control plane is more independent from the data plane, as the kernel may not provide the feedback needed. For example, reactive protocols such as AODV need to know if a route was not found for a given packet in order to initiate the route request procedure [19]. To support this type of protocols the proposed architecture could be extended with a kernel module that listens on the desired events and reports them back to the control plane.

3.2.1 *Real Routing* Module

Most of the data plane in kernelspace architecture can stay the same across different protocols, namely the route updater and the interprocess communication. The only part that is specific to each protocol is the socket filter used. This is unlike the data plane in userspace architecture, which requires the data plane (which can be very different between protocols) to be ported to the real system. To enable researchers to quickly use the data plane in kernelspace architecture, an implementation was developed for ns-3, the *real routing* module [2].

The *real routing* module implements a route updater (*RtNetlinkRouteUpdater*) that uses Netlink sockets to update the kernel routing tables and that runs in a privileged userspace process to avoid having to execute the whole simulation with root privileges. Inside ns-3, the class *RealRouteUpdater* was created to spawn and configure the *RtNetlinkRouteUpdater* process, receive the routes from the routing protocols and send them to the created process, which in turn updates the kernel. The interprocess communication, used to send the routes from the simulator to the route updater, is implemented with Unix domain sockets.

In order for ns-3 routing protocols be able to update the routes of the system without having to be coupled to the new *real routing* module, three new optional callbacks were added to *Ipv4RoutingProtocol*, the base class of all IPv4 protocols (support for IPv6 was not created but it can be easily added when needed). The new callbacks are: RouteAddedCallback (called when a the protocol creates a new route), RouteRemovedCallback (called when a protocol deletes a route), and RoutingTableUpdatedCallback (called when the protocol updates the whole routing table at once). Protocols that use these callbacks will work with the *real routing* module. The *RealRouteUpdater* will register its route updating functions with each of the protocol's callbacks. Thus, every time one of the callbacks is called, the corresponding function in the *RealRouteUpdater* is also called and the received information is sent to the *RtNetlinkRouteUpdater* process which, in turn, updates the kernel routing tables (Figure 4).

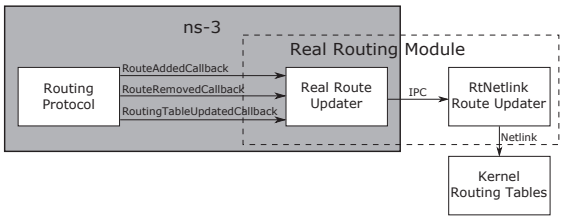


Figure 4: Real Routing module architecture.

Finally, two other new features were added to the *EmuFdNetDeviceHelper* in the *fd-net-device* module to support the data plane in kernelspace architecture and support network namespaces: setting a socket filter in the raw socket and selecting a network namespace for the socket.

With the *real routing* module implemented, a researcher must take the following steps to use the kernelspace architecture in a new protocol. When implementing the routing protocol in ns-3, the new callbacks must be called as appropriate. Then, when using the protocol in real environments, the simulation script needs the following changes: (1) set the socket filters in the *EmuFdNetDeviceHelper* and, if using network namespaces, set the network namespace as well; (2) in the emulated routing node create an instance of the *RealRouteUpdater* and configure it with the routing protocol being used, the network namespace of the node, and the mapping between the ns-3 interfaces and the real interfaces of the host. With this configuration done, the created scenario will then be executed using the data plane in kernelspace architecture. All the patch files needed to add the Real Routing module to ns-3.21 source code can be downloaded in [2]. The patch files have to be applied by the order they are numbered.

4. VALIDATION

We evaluated the proposed solutions to make sure that they achieve the stated goals of compatibility with ns-3, scalability in simulation and real systems, and code reuse.

There is no need to validate the compatibility with ns-3 and scalability in simulation because the protocols are developed directly in ns-3 and so are as compatible and scalable

as any other ns-3 protocol as long as they are correctly implemented. Since the proposed changes to network protocols are only needed in testbeds, our validation focus on scalability in real environments and in the amount of new code that needs to be developed to use the proposed architectures.

The scalability in real environments requirement was validated by selecting routing protocols already implemented in ns-3 and extending them to use the new architectures. The performance of both implementations was then compared in a real scenario.

The requirement of code reuse was validated by analyzing the number of lines of code that were added to the implementations of the protocols in ns-3 in order to make them use the proposed architectures.

4.1 Data Plane in Userspace

4.1.1 Performance Validation

To validate the data plane in userspace architecture we used the Wireless Metropolitan Routing Protocol (WMRP), an ad hoc, proactive routing protocol that operates over layer 2 [20][5][7].

The scalability in real environments will be validated by comparing the performance of WMRP running completely in ns-3 emulation with the performance of WMRP when the data plane is executing in userspace. We then analyze the performance gains that were obtained with the new architecture.

The two scenarios have the same network topology (Figure 5), which is composed of four real nodes connected by 100 Mbit/s Ethernet links: the source **S** uses the `iperf` tool to generate UDP traffic destined to **D** and at the same time it also pings **D** to measure the round-trip time (RTT); the destination **D** runs an `iperf` server; the routing bridges **RB1** and **RB2** forward the frames between **S** and **D**.

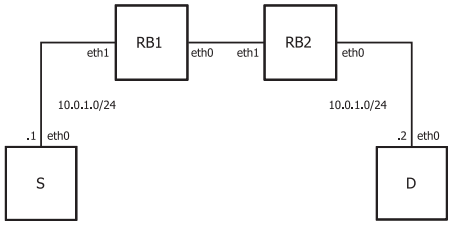


Figure 5: Network topology of the test scenarios.

There were two scenarios tested. In the first scenario (*ns-3 emulation*), the two RBridges are implemented using traditional ns-3 emulation. In the second scenario (*userspace forwarding*), the proposed architecture for executing the data plane in userspace is implemented in both RBridges.

For each scenario a series of tests was executed, starting with **S** generating 1 Mbit/s of UDP traffic to **D** and then increasing that value to 2, 4, 8, 16, 32, 64, and 70.8 Mbit/s. Each test had a duration of 30 seconds and was repeated 5 times. The payload of the UDP packets was 160 bytes, thus inducing a network load representing a worst case scenario where the network had to process a large number of packets per second and maxed out the CPU processing capabilities. This was especially important due to the absence of Gigabit Ethernet cards in the test machine. The maximum offered data rate was of 70.8 Mbit/s because that is the maximum throughput that can be achieved in 100 Mbit/s Ethernet links for UDP packets with 160 bytes of data.

For each of these tests, four performance metrics were measured: the received data rate in **D**; the packet loss ratio; the average round round-trip time; and the average CPU load in **RB2**.

The machine running **RB2** has an Intel Atom N270, which is a single core processor but that uses the Hyper-Threading technology so it behaves as having two logical cores. The operating system used was Ubuntu 14.04 LTS.

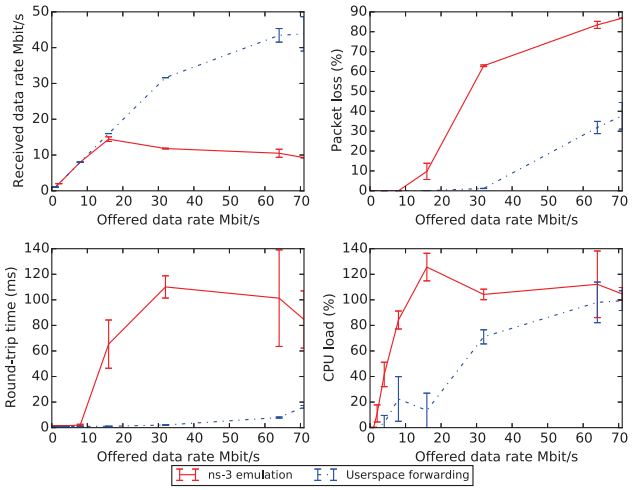


Figure 6: Data plane in userspace performance validation results for UDP traffic with payload of 160 bytes.

The results are shown in Figure 6, where the lines represent the average and the error bars represent the standard deviation. The traditional ns-3 emulation implementation is able to forward packets at the offered rate until 8 Mbit/s. After that, performance starts degrading and when the offered rate hits 70.8 Mbit/s, ns-3 emulation has 87% packet loss, can only forward at 9 Mbit/s, and the round trip time (RTT) also increases to 85 ms. The reason why ns-3 emulation goes above 100% load is that the CPU of the bridge has two logical cores (due to Hyper-Threading) and ns-3 uses a dedicated thread for reading operations.

The data plane in userspace implementation is able to forward packets at the offered rate until 32 Mbit/s while keeping RTT low (1 ms until 16 Mbit/s and 2 ms at 32 Mbit/s). For higher data rates performance starts degrading but at 70.8 Mbit/s, it can forward at around 44 Mbit/s (4.9 times more than traditional emulation) while keeping RTT at 16 ms (5.3 times lower than traditional emulation). Additionally, while the offered data rate is lower than 64 Mbit/s the CPU load of the data plane in userspace implementation is lower than that of ns-3 emulation. For higher data rates both implementations present similar loads, but the userspace implementation is processing more traffic for that same load.

From our experience on using ns-3 emulation in varied scenarios such as the tests performed in [4], we know that the processing bottleneck is mostly related to the number of packets processed, rather than the size of the packets forwarded by the node. This is easily understandable as "bulk" memory copy operations are usually much faster than the multiple function calls and processing involved with the

handling of each packet object, such as the interpretation of the header and routing/forwarding tables lookups. Based on this information, it is expected near a ten-fold performance increment in the forwarding data-rates if the node were to be handling 1500 bytes Ethernet frames in Gigabit Ethernet links.

4.1.2 Code Reuse Validation

To validate that using the data plane in userspace architecture does not require writing a large amount of new code we counted the lines of code that were developed to implement the proposed architecture on top of the existing ns-3 implementation of WMRP. In this process we discarded the blank and comment lines.

The original implementation of WMRP has 5057 lines of code, while the same protocol implementation including userspace forwarding has 5608 lines of code. This concludes that only around 11% more lines of code were written to implement the data plane of WMRP in userspace. This is in part because the implemented data plane was simplified and assumes that the topology of the network does not change. Implementing the full operations of the data plane would increase the number of lines of code but in general, the data plane is much simpler than the control plane so only a relatively small amount of code needs to be ported.

Ultimately, the researchers developing the protocol have to decide if the extra amount of code that has to be written is worth the performance gains that will be obtained.

4.2 Data Plane in Kernelspace

4.2.1 Performance Validation

The data plane in kernelspace architecture was validated using the Optimized Link State Routing Protocol (OLSR), a link state, proactive, L3 protocol [9].

The scalability in real environments was validated by comparing the performance of three different implementations: OLSR using traditional ns-3 emulation; OLSR using the real routing module; and olsrd, an implementation of OLSR for real systems [18].

The three scenarios have the same network topology (Figure 7), which is composed of three nodes connected by 100 Mbit/s Ethernet links: the source **S** runs OLSR and uses the `iperf` tool to generate UDP traffic destined to **D** while at the same time it also pings **D** to measure the RTT; the destination **D** runs OLSR and an `iperf` server to receive the UDP traffic from **S**; **R** also run OLSR and forwards the packets between **S** and **D**.

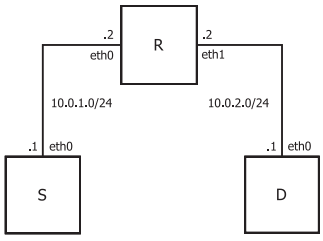


Figure 7: Network topology of the test scenarios.

There were three scenarios tested. In the first (*ns-3 emulation*), **R** runs a traditional ns-3 emulation of OLSR. In the second scenario, (*ns-3 real routing*) **R** runs a ns-3 emulation of OLSR using the real routing module that was developed. In the last scenario (*olsrd*), **R** runs olsrd. In all scenarios, **S** and **D** run olsrd. The router **R** runs on an Intel Atom N270 with Ubuntu 14.04 LTS x86.

For each scenario, a series of tests was executed, following the same procedure and parameters used to validate the performance of the data plane in userspace architecture (Section 4.1.1).

For each of these tests, four performance metrics were measured: the received data rate in **D**; the packet loss ratio; the average RTT; and the average CPU load in **R**.

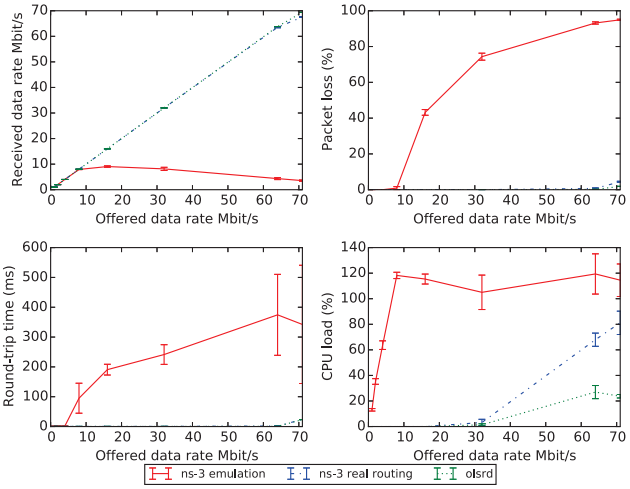


Figure 8: Data plane in kernelspace performance validation results for UDP traffic with payload of 160 bytes.

The results are shown in Figure 8, where the lines represent the average and the error bars represent the standard deviation. The traditional ns-3 emulation implementation is able to forward packets at the offered rate until 8 Mbit/s. After that, performance starts degrading and when the offered rate hits 70.8 Mbit/s, ns-3 emulation has 95% packet loss and can only forward at 3.5Mbit/s. The round trip time (RTT) also increases to 342 ms, while until 8 Mbit/s it was around 1 ms. The reason why ns-3 emulation goes above 100% load is that the CPU of the router has two logical cores (due to Hyper-Threading) and ns-3 uses a dedicated thread for reading operations.

The real routing implementation and olsrd are able to forward packets at the offered rate at all rates except at 70.8 Mbit/s. At this highest rate there is packet loss of 4% and 2% for real routing and olsrd, respectively, and they forward traffic at 67.5 Mbit/s and 69 Mbit/s, respectively. Both implementations provide very low round trip times, less than 0.5 ms until 32 Mbit/s, around 1 ms at 64 Mbit/s and around 23 ms at 70.8 Mbit/s. This means that, at the highest rate, real routing provides an increase in throughput of around 19 times when compared to traditional emulation as well as a decrease in RTT of around 14 times.

The reason that real routing performs slightly worse than olsrd is because it incurs a higher CPU load (around 3 times more at 70.8 Mbit/s). This can be attributed mainly to the socket filters, which must be executed for every packet that is received in the system. One way to reduce the overhead incurred by the filters is to enable them to be JIT (just-

in-time) compiled, which means that instead of the kernel interpreting them for each packet, it will compile them to the underlying processor architecture just once and then run the compiled code for the received packets. However, we could not enable this optimization in the system we were using because it is a x86 machine and Ubuntu 14.04 only configures the kernel to allow JIT compilation of filters in x64 systems.

As explained in the performance results for the userspace implementation using WMRP, in this OLSR scenario it is also expected near a ten-fold performance increment in the forwarding data-rates if the router were to be handling 1500 bytes Ethernet frames in Gigabit Ethernet links.

4.2.2 Code Reuse Validation

To validate that using the data plane in kernelspace architecture only requires writing a small amount of new code we counted the lines of code that were developed to use the real routing module with the existing ns-3 implementation of OLSR. In this process we discarded the blank and comment lines.

The existing ns-3 implementation of OLSR has 5067 lines of code, while the same protocol implementation including the extension to work with the real routing module has 5140 lines of code. The lines of code of the real routing module itself are not counted because it is a generic module that can be used with any proactive L3 protocol without having to be reimplemented. This concludes that only around 1.4% more lines of code were written to use the real routing module. Most of the new lines implement the code that calls the real routing module callbacks to update the routing table and the rest of the lines correspond to code that sets the socket filters and initializes the real routing module. We can thus conclude that the increase in code is negligible, especially when compared to the performance gains that are obtained when using the data plane in kernelspace architecture.

5. CONCLUSIONS AND FUTURE WORK

The main goal of this work was to propose an approach to shared protocol implementation that allows users of ns-3 to use their simulation code in real environments. Our proposed solution is to use fast prototyping with ns-3 emulation but move the data plane outside of the simulator. Since most of the traffic of typical networks is related to data, this solution improves the network nodes efficiency at the cost of having to port the data plane code. This is not a big problem, however, because the data plane is much simpler than the control plane resulting in a relatively small amount of code that needs to be ported. We proposed two architectures to implement this solution: executing the data plane in userspace and executing it in kernelspace. The former is more generic and works for any protocol while the latter is more efficient and allows more code reuse but is only applicable to proactive L3 protocols.

In the case of the data plane in kernelspace architecture, since most of it can be reused between different protocols, we created a new ns-3 module called *real routing* which implements the generic parts of the architecture. This allows researchers to easily extend their ns-3 protocols to execute the data plane in kernelspace.

In order to validate the proposed solutions, we extended the WMRP and OLSR protocols to work with the developed architectures. Then, we compared the performance of the new architectures against traditional ns-3 emulation running multiple tests with different configurations. The summarized interpretation of the obtained results is the following: emulating an ns-3 node, the maximum throughput can be improved by as much as 4.9 times in userspace and 19 times in kernelspace, while having the RTT lowered by 5.3 and 14 times, respectively. With these results in consideration, we can conclude that the proposed architectures (userspace and kernelspace) allow much better data plane performance, when compared against traditional ns-3 emulation. The kernelspace architecture has the best performance, obtaining results very close to a real protocol implementation. Thus, when possible, the kernelspace architecture should be used to obtain the best emulation performance results. This enables the use of fast prototyping for network protocols in more traffic demanding scenarios or in real nodes with limited processing power.

The amount of code reuse obtained was also verified by counting the lines of code needed for each implementation. We found that implementing the data plane in userspace only required the development of around 11% additional lines of code on top of traditional emulation implementations. In the case of executing the data plane in kernelspace, that value can be reduce to 1.4% by using the *real routing* module.

As a result of this work, ns-3 users will be able to reuse their simulation code in real networks, now with reduced packet processing overhead, enabling them to reap the benefits of the fast prototyping process without the previously associated negative performance impact.

In terms of future work, the *real routing* module can be proposed for integration in the main ns-3 distribution in order to reach more protocol developers and researchers. It can also be made more easy to use, for example, it could allow socket filters to be specified as a string with the tcpdump filter syntax instead of an array of machine code that must be manually generated by users.

To increase the usefulness of the *real routing* module, it can also be extended to support reactive protocols. To do this, a kernel module or other facility would have to be created that would listen to the events generated in the kernel during packet processing (e.g. failure to find a route) and send the relevant information back to the simulator so the appropriate control plane operations can be executed.

It would be also interesting to test the performance gains that can be obtained with the data plane in kernelspace architecture on machines that support JIT compilation of socket filters.

Finally, it would be interesting to merge the benefits of the simulator agnostic applications approach with the real routing module, by allowing to run the control plane implementation using real system sockets, thus complementing the routing/forwarding funcionality already provided by the real routing module. This can further reduce the ns-3 performance footprint in a scenario where we are trading ns-3 logging and tracing functionalities for a better performing prototype.

6. ACKNOWLEDGMENTS

The authors would like to thank the support from the Portuguese Foundation for Science and Technology (FCT) under the fellowship SFRH/BD/69051/2010.

The authors would also like to express their deepest grati-

tude to Tom Henderson, for his invaluable input to this work, during both the development of the Real Routing module and the review of this paper.

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project "POCI-01-0145-FEDER-006961", and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013.

7. REFERENCES

[1] Sitme's project website. http://www.sitme.org. Accessed: 2016-02-12.

[2] Source code for real routing module - sequency of patches to ns-3.21. http://telecom.inesctec.pt/~hfontes/real_routing_patches.zip, Feb. 2016.

[3] J. Abraham and G. Riley. Simulator-agnostic ns-3 applications. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 391–396, Sirmione-Desenzano, Italy, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[4] G. Carneiro, H. Fontes, and M. Ricardo. Fast prototyping of network protocols through ns-3 simulation model reuse. *Simulation Modelling Practice and Theory*, 19(9):2063–2075, Oct. 2011.

[5] G. J. a. A. M. Carneiro. *Transparent metropolitan vehicular network: design and fast prototyping methodology*. PhD thesis, Universidade do Porto, Porto, 2012.

[6] H. Fontes, R. Campos, and M. Ricardo. Improving ns-3 emulation support in real-world networking scenarios. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 261–266, Athens, Greece, 2015. ICST.

[7] H. M. Fontes. Multi-technology router for mobile networks: layer 2 overlay network over private and public wireless links, 2010. MSc Thesis, MIEIC, FEUP, Universidade do Porto.

[8] G. Insolvibile. Kernel korner: Linux socket filter: Sniffing bytes over the network. *Linux J.*, 2001(86):8–, June 2001.

[9] P. Jacquet and T. Clausen. Optimized link state routing protocol (OLSR). RFC 3626, IETF, Oct. 2003.

[10] S. T. Jansen. *Network Simulation Cradle*. Thesis, The University of Waikato, 2008.

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[12] G. Kunz, O. Landsiedel, and G. Wittenburg. From Simulations to Deployments. In K. Wehrle, M. Günes, and J. Gross, editors, *Modeling and Tools for Network Simulation*, pages 83–97. Springer, New York, 2010 edition edition, June 2010.

[13] M. Lacage. *Experimentation Tools for Networking Research*. Ph.D., Universite de Nice-Sophia Antipolis, 2010.

[14] J. P. Morrison. *Flow-Based Programming, 2nd Edition: A New Approach to Application Development*. CreateSpace Independent Publishing Platform, Unionville, Ont., 2 edition edition, May 2010.

[15] J. Moy. OSPF version 2. RFC 2328, IETF, Apr. 1998.

[16] S. C. Muthukumar, X. Li, C. Liu, J. B. Kopena, M. Oprea, and B. T. Loo. Declarative toolkit for rapid network protocol simulation and experimentation. In *ACM SIGCOMM Conference on Data Communications (demo)*, Barcelona, Spain, Aug. 2009.

[17] M. Neufeld, A. Jain, and D. Grunwald. Nsclick: Bridging Network Simulation and Deployment. In *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '02, pages 74–81, New York, NY, USA, 2002. ACM.

[18] OLSR.org. OLSRd. http://www.olsr.org. Accessed June 01, 2015.

[19] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, IETF, July 2003.

[20] M. Ricardo, G. Carneiro, P. Fortuna, F. Abrantes, and J. Dias. WiMetroNet a scalable wireless network for metropolitan transports. In *2010 Sixth Advanced International Conference on Telecommunications (AICT)*, pages 520–525, Barcelona, Spain, May 2010.

[21] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Linux netlink as an IP services protocol. RFC 3549, IETF, July 2003.

[22] P. L. Suresh and R. Merz. ns-3-click: click modular router integration for ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 423–430, Barcelona, Spain, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[23] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 217–228, New York, NY, USA, 2013. ACM.

[24] H. Tazaki, F. Urbani, and T. Turletti. DCE cradle: Simulate network protocols with real stacks for better realism. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, pages 153–158, Cannes, France, 2013. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[25] U.S. NRL Networks and Communications Systems Branch. The Protean Protocol Prototyping Library (Protolib). http://www.nrl.navy.mil/itd/ncs/products/protolib. Accessed January 16, 2015.

ns-3 Based Framework for Simulating Communication Based Train Control (CBTC) Systems

Abdulhalim Dandoush*, Alina Tuholukova, Sara Alouf, Giovanni Neglia
Inria, Sophia Antipolis, France
name.surname@inria.fr

Sebastien Simoens, Pascal Derouet, Pierre Dersin
Alstom Transport, France
name.surname@transport.alstom.com

ABSTRACT

In a Communication Based Train Control System (CBTC), a central zone controller server (ZC) exchanges signaling messages with on-board carborne controllers (CC) inside the trains through a wireless technology. The ZC calculates and sends periodically to each train its Limit of Movement Authority (LMA), i.e. how far the train can proceed. A CC triggers an emergency break (EB) if no message is received within a certain time interval to avoid collision. Clearly, it is not desired to have an EB due to signaling messages losses (called spurious EB) and not to real risks for the trains. Quantifying the rate of spurious EBs and predicting correctly CBTC system performance are hard tasks with important industrial relevance.

This work aims at filling this gap using simulation to better predict CBTC system performance and avoid extra provisioning before deployment. A typical CBTC system implementation for metro by Alstom Transport is considered. New ns-3 modules (CBTC protocol, Video traffic generator, multi-channel scanning mechanism, 3D antennas patterns) are developed and a piece of existing code is enhanced. The simulation is also used to investigate the dimension of the radio access networks in a realistic environment (specific modems and access point antennas, radio frequencies, train and track models), another aspect also ignored in the previous literature. Last, our approach can be useful to validate some analytical works.

CCS Concepts

•Computing methodologies → **Discrete-event simulation; •General and reference** → *Validation;* **•Networks** → *Network simulations; Wireless local area networks;* **•Applied computing** → *Transportation;*

Keywords

Communication Based Train Control, Signaling System, ns-3, Video Streaming Generator, Directional Antennas, Performance Evaluation

1. INTRODUCTION

In the traditional train control systems track lines are divided into fixed blocks. Each of them can only contain one train at a time to avoid collision. Improving the traditional system is needed due to the increasing demand for efficient mass transit transport. In other words, we have to utilize train lines more efficiently taking advantage of the new on-board processing and communication technologies. In fact, the moving-block control allows to safely increase the number of trains traveling over a track line as it takes into account the real-time information about train position rather than the information provided by the fixed blocks of large enough length. The moving-block control is being deployed as CBTC for urban mass transit system.

In moving-block systems, the zone controller (ZC) computes the Limit of Movement Authority (LMA) for each train based on the information received for all the trains in its zone. Then, it sends the End-of-Authority (EOA) control message carry the LMA to the on-board carborne controllers (CC) in the train through a wireless technology. Currently, Alstom uses WiFi technology. However, the use of 4G/5G may be considered for the future. To increase the system reliability against losses and delays, the messages can be sent redundantly through two separated networks called red and blue network. To avoid the risk of collision, CC will stop automatically the train if no valid EOA is received during a given time interval by triggering an Emergency Break (EB). Clearly, it is not desired to have an EB due simply to signaling messages losses (called spurious EB) when there is no eventual risk for the train. For this reason, the so-called performance based contracts (similar to service level agreements for network operators) can bind rail transport companies to specify the maximum number of spurious emergency brakes over a given period of time.

Therefore quantifying the rate of spurious EBs and predicting correctly CBTC system performance to avoid extra provisioning before deployment are important industrial and

*This author is now with ESME Sudria, Paris Sud, France, email: dandoush@esme.fr.

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915378

research problems. Until now this was only done through analytical approaches that ignore many important aspects such as packet losses, handovers and congestion due to bursty traffic. The more reliable solution at the moment is based on costly field measurements of deployed systems. For that reason we decided to develop a framework that helps the evaluation of the system performance and the prediction of spurious EBs.

To that end, we have chosen the network simulator ns-3 [4] because it is a GPLv2 licensed and an open source discrete-event network simulator supported by Inria and the University of Washington.[1] It is highly modularized and thus can be easily modified or extended. In addition, for our industrial project, ns-3 includes many useful modules for the wireless channel, modeling co-channel interference, transmission error and the propagation, as well as mobility models and the TCP/IP protocol stack.

However, the current ns-3 version is missing some functionalities needed to simulate train-trackside communications. Thus, the need for implementing new modules and enhancing some existing ones. Some modules cannot be shared at this stage for confidentiality reasons. The current shared code of this work is available at [5, 6].

The paper is organized as follows. In Section 2 we introduce the new CBTC and MPEG video traffic generator modules. We describe the details of the CBTC protocol, the Train and Track objects and discuss their implementation and usage in Section 2.1. Section 2.2 presents the design and the implementation of the new video streaming generator. A brief description of the 3D Antenna module is introduced in Section 2.3. Selected results to illustrate some issues that can be addressed using our ns-3 based simulation are presented and discussed in Section 3. Section 4 reviews some related works and Section 5 concludes the paper. Last, the acronyms used in the paper are listed in Appendix A.

2. NEW AND ENHANCED MODULES

In this section, we describe briefly the new modules and enhancements we have integrated within ns-3 in order to correctly simulate the train systems.

First, there are no ns-3 modules for train and track, and—as expected—no implementation of a moving block control system in ns-3. Second, there is no configurable and flexible video traffic generator that can generate traffic based on particular values of the basic characteristic of MPEG-compatible cameras such as the group of pictures (GOP) length, number of frames per second, average bit rate, etc. In fact, trains and ground controllers broadcast two video traffic profiles. The Platform TV flow is a video flow in the ground-to-train direction (e.g. the control monitor in the driving cabin). Closed-circuit television (CCTV) flow is the second type of video flow in the train-to-ground direction (e.g. video surveillance systems). Platform TV and CCTV are MPEG-4 video traffic and the traffic shape can be changed from camera to another one due to different resolution, frame rate, compression method, motion degree, and image quality settings. It is particularly important for the application we are considering to simulate these video traffic flows because they interfere with the critical signaling messages on the wireless link and they require large bandwidth.

[1]A main objective of Inria in the Inria-Alstom Transport lab is to apply modeling and simulation tools in industry.

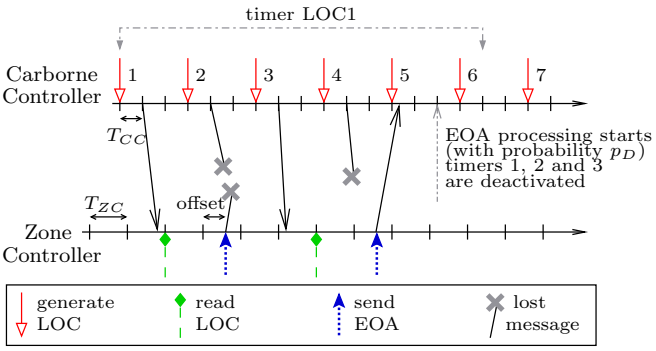


Figure 1: Illustration of LOC-EOA exchanges.

Also, the current version is missing some functionalities that we need to simulate train-trackside communications. ns-3 considers only omni-directional antennas for the Yans-WiFi model even if some theoretical directional antenna models are implemented for the simulation of other wireless technologies. However, the gain of the transmitter/receiver antennas is a key parameter for the train system because it changes the received signal strength (rss) and then the signal to noise (SNR) ratio. SNR affects the packet loss probability, the behavior of the handover process, and many of other parameters such as the distance between two APs on a track. Moreover, ns-3 does not yet implement a multi-channels scan mechanism of beacon signals' strength transmitted by access points (APs) operating on different frequencies, which is necessary for a realistic layer-2 handover implementation in WiFi. In addition, ns-3 does not implement a switch object because the impact of layer 2 switching operations on the network performance is assumed to be negligible. Hence, changing routes after a handover at the MAC level as done in the real implementation, that we consider, is tricky. Using the layer-3 routing algorithms (e.g. OLSR, AODV) as a workaround is not an option because the convergence to the correct path may require a few seconds causing the loss of several control messages (major limitation for CBTC).

2.1 New CBTC Modules

In this section we describe the detailed operation of a moving block system considering as reference the specific CBTC implementation by Alstom Transport.[2]

Figure 1 shows a typical messages' exchange between the on board controller (the CC) and the ground controller (the ZC). Observe that both the controllers operate in discrete time on the basis of clock periods of hundreds of milliseconds. This is due to the fact that they are actually e-out-of-f voting systems where different processors perform in parallel the same calculations and a time-slotted operation simplifies the synchronism of the processors. The clock periods at the ZC and at the CC (respectively T_{ZC} and T_{CC}) are in general different because the subsystems are provided by different vendors and also because they have different computational loads during one period.

The most important CBTC messages are location reports (LOC) and end-of-authority ones (EOA). A LOC is a mes-

[2]The parameters' values have been slightly changed and some specific implementation details are hidden to protect the industrial know-how of the company.

sage periodically transmitted from the on board CC through the Data Communication Sub-System (DCS) to the ZC. The message is actually sent twice to increase reliability through the blue and the red radio networks (i.e. two separated WiFi networks with separated Access Points APs and two different on-board modems OBMs inside each train). We denote by TRE (Track-side Radio Equipment) a pair of redundant APs, one blue and another red AP that are installed at the same location along a track. The first LOC arriving at the ZC is processed. Each LOC is acknowledged by an EOA message in the reverse direction (again sent through the two networks). The EOA indicates to the CC how far the train can advance. For each generated LOC, a timer is activated. The timer expires when its value exceeds a validity duration (TM) before it can be acknowledged and therefore an emergency break (EB) is triggered to avoid collision. The details of the protocol is described hereafter.

2.1.1 Implementation Overview

We created new modules/Objects as follows: (i) a separate module called "cbtc-manager" lives in the directory "src/" and (ii) new application level modules "LocEoaClient" and "LocEoaServer" live in "src/applications". The cbtc-manager module consists of the following objects:

1. Train: It contains mainly two OBMs (red/front and blue/back OBM) with a CC server (i.e. Nodes, NodContainers, NetDevices, Interfaces, smart pointers, etc.). A train has several properties such as ID, length, initial position, velocity, and direction.
2. Track: It defines the required number of TREs along a track. The code is optimized such that referring initially only to 5 TREs (10 APs) nodes, and then reuses/cycles them to extend the track length as the train moves. This is important to be able to simulate long track without any "out of memory" problem or execution time limitation. The attributes of this object are the number of the required TREs, a flag to activate the reuse/recycle feature, the position of the first TRE, the distance between two TREs, the distance between twp APs of the same TRE, and the velocity of the train (for recycling the APs at appropriate times). The object will change the APs positions using their mobility model smart pointer.
3. TrainCcServer: It handles the received EOAs by LocEoaClient application (sent by the LocEoaServer application), manages the clock and the timers for the CC, activates or deactivates an EB and calculates the EB rate.
4. CbtcManager: The objective of this object is to simplify the creation of the simulation scenarios by automatically creating the networks topologies (on-board LAN, radio network, ground LAN). It has member functions to create all the networks elements, interconnect them and install the applications on. In particular, it will do the following: (i) create all the nodes in the scenario (e.g. the Zc server, APs and OBMs of trains); (ii) create the wired and wireless channels that interconnect the nodes (e.g. Ethernet, a ppp link and a WiFi connection); (iii) create the Net devices and attache them to the corresponding nodes and channels; (iv) identify the PHY/MAC properties of TREs/APs and OBMs and install the right antennas; (v) create and install a mobility model, internet stack and routing instances for the created nodes. Then, in order to run CBTC protocol the application LocEoaServer should be created and installed on the node that corresponds to the ground ZC (ground server) and the application LocEoaClient needs to be installed on the nodes that correspond to the OBMs of each train (back and front OBM). A TrainCcServer object is then created to refer to the OBMs and the LocEoaClient applications installed on them, and to handle the EOA messages sent by the ZC server (LocEoaServer) and received by the front and back OBMs (LocEoaClients). An example of the usage is given in the next subsection.

The LocEoaClient, LocEoaServer modules with their helpers represent the CBTC message generators. Table 1 depicts their attributes with the default values. The message exchange between these two applications and the TrainCcServer instance works as follows:

1. a LOC is generated at the CC every T_{LOC}, multiple of the CC clock period T_{CC},
2. the LOC (say LOC k) is ready to be emitted and passed to the DCS after a processing delay equal to T_{CC} since its generation,
3. the delivery delay introduced by the DCS is random and depends on the network conditions,
4. at the ZC the LOC is available for computing at the next tick of the local clock,
5. the computing time at the ZC required to process the LOCs from all the trains in the zone and generate the corresponding EOAs is T_{ZC},
6. the EOA k is emitted within the next cycle of the ZC at an offset O depending on the train (the ZC sends sequentially the EOAs to all the trains in the zone),
7. the EOA is delivered to the CC after a random delay depending on the network conditions,
8. at the CC the EOA gets in a processing queue, at the next tick of the CC clock the most recent EOA present in the queue is processed unless there are higher priority tasks arrived during the same CC clock period (which happens with probability p_D); in any case an EOA processing is not delayed more than an additional CC period,
9. the EOA k is actually processed only if it remains valid until the end of the current CC clock; once processing starts, all the pending timers for older LOCs (i.e. LOC h for $h \leq k$) are deactivated,
10. if the timer of a LOC is not deactivated before its expiration, the EB procedure is triggered.

2.1.2 Building Scenarios

We show in this subsection how easy it is to create a scenario script using the cbtc-manager rather than implementing a scenario from scratch (i.e. creating nodes for TREs and Trains, Containers, Network topology, channels, Antennas, mobility, internet stack, interfaces, etc).

We first describe briefly the creation of a default scenario template that can be modified easily according to the needs. The default scenario template has one train that moves along the track that is simply a line. The track consists of 5 TREs. In order to create this scenario we need to create the object

Table 1: Attributes of LocEoaClient, LocEoaServer and CbtcManager.

Name of the attribute	Description
Attributes of LocEoaClient	
LocProcessingDelay (200ms)	The time to wait after generation of the LOC and before its emission
InputProcDelay (5ms)	The time to process the input
ClockPeriod (200ms)	The period of the clock
Locfrequency (3)	The number of ticks to wait before generating new LOC message
ProbStartProcEoa (0.99)	Probability to start the processing of the EOA on the next tick
TimeOutValue (5s)	The time during which the generated LOC is valid
OutputDelayMin (5ms)	Minimal output delay
OutputDelayMax (40ms)	Maximum output delay
ObmId	The ID of the OBM
TrainCC	Train CC server object
Attributes of LocEoaServer	
InputProcDelay (12ms)	The time to process the input
ClockPeriod (275ms)	The period of the clock
ProcessingDelay (275ms)	Time to process the received LOC
Offset (40ms)	The offset delay for sending the EOA response
PacketLoss (0.1)	Packet loss Probability
OutputDelayMin (5ms)	Minimal output delay
OutputDelayMax (40ms)	Maximum output delay
Attributes of CbtcManager	
NumberOfTre (5)	The number of the TREs along the track
NumberOfChannels (5)	Number of channels that are used for TREs
SimulationTime	The total time of the simulation
EnergyDetection-Threshold (-80dbm)	Energy detection threshold
CcaMode1Threshold (-90dbm)	CCA Mode 1 Threshold
ObmTxPower (16dbm)	OBM transmission power
ApTxPower (16dbm)	AP transmission power

of CbtcManager, then create the Train object and pass its reference to the CbtcManager instance. After that, we call the method DefaultScenario() of the CbtcManager as follows:

```
Ptr<CbtcManager> manager = CreateObject<CbtcManager>();
Ptr<Train> train = CreateObject<Train>();
  manager->CreateTrainTopology(train);
  manager->DefaultScenario();
```

In Table 1 the attributes of CbtcManager and their default values are presented.

Now, we can easily change the default parameters values using Config::SetDefault() or SetAttibute() methods.

To add CBTC traffic it is enough to call the method SetUpLocEoaApplication() that will create and install the LocEoaClient/Sertver applications. The easiest way to change the default behaviour of CBTC protocol is to modify the values of the attributes of the LocEoaClient and LocEoaServer applications depicted in Table 1.

An important point is to activate the TREs reuse/cycle in order to decrease the memory usage and optimize the ns-3 execution time as we explained in the Track object. This can be done easily by setting the corresponding boolean property to true as follows:

```
Config::SetDefault ("ns3::Track::DoCycle", BooleanValue
    (true));
```

Note that when we have several trains that have different directions on the same track, we cannot activate this property.

Last, we can add MPEG video traffic generator and a UDP server applications to the ZC server and the OBMs nodes.

During the simulation, if we want to print some information such as the current EB rate, or the association/disassociation moments with train positions, we simply call the corresponding member function of the cbtc-manager.

```
manager->PrintTime (periodToPrint);
manager->PrintEbRate (periodToCountEB);
manager->PrintAssocInfo();
```

A complete example is located at (/src/cbtc-manager/examples/cbtc-manager-example.cc).

2.2 New Video Traffic Generator Module

The goal of the video generator module is to provide a flexible configurable packet generator of MPEG-4-like UDP traffic. The source code for the new module called (MpegPktGenClient) lives in the directory "src/applications". A helper module is also implemented to facilitate its use. We can change the video streaming traffic shape, i.e. the group of pictures (GOP) structure, by simply changing the key parameters values of the simulated camera explained hereafter. The Video generator client can work with any UDP based server like the already implemented UdpServer. An example of usage, expected results, and a brief analysis of the output pcap trace are presented in the next subsections. Table 2 summarizes these attributes with their typical values.

2.2.1 Implementation Overview

The methodology to create video streaming is: (i) to build

Table 2: MpegPktGenClient attributes.

GopLength	number of frames (I + P or B) per Burst period	15f
ImageRate	number of frames per Second fps created by the Camera	12fps
IframeSize	size of the I-frames in the GOP in kbits	1200kbits
AvBitRate	average Bit rate in mbps	2mpbs
PeakRate	peak Bit rate at which we send the I-Frames in mbps	10mbps
MaxPacketSize	the maximum size of a packet in Bytes without the headers	1468B
VideoFilename	name of the text file that will contain GOP calculated structure	mpeg-4.dat
RemotePort	The port on which the server is listening	-

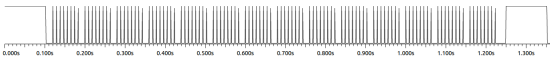


Figure 2: An example of a generated GOP.

the GOP structure (e.g. the size and the type of frames in the GOP and the spacing between frames) based on the attributes values and save the characteristics in the text file pointed by the user or in the mpeg-4.dat if no name was provided by the user were; (ii) calculate the number of packets in each frame and the inter arrival time between two packets of the same frame type; (iii) schedule the sending events of the generated packet at the appropriate moments; (iv) repeat the process until the stop time of the application.

Detailed calculation.

We determine the moments of the sending events as follows[3].

- Calculate the duration of a GOP "burstPeriod" in seconds by dividing the values of gopLength over the imageRate.
- Compute the burst duration of the I-frame: burstDuration = iFrameSize/peakBitRate.
- The amount of data in each GOP is computed as follows: avDataSizeInGop = avBitRate * burstPeriod.
- Subtract the I-Frame-Size from the total amount of data in a GOP to get the summation of the P/B-Frames sizes (sum_frames_P_sizes).
- The size of each P/B-Frame and the bitRate at which we send them are calculated as follows:
  pFrameSize = sum_frames_P_sizes / (gopLength - 1) (Bytes)
  pFRate = sum_frames_P_sizes / (burstPeriod - burstDuration- interval_P_frames).
- Given the burstDuration (duration of the I-Frame in a GOP), we calculate the inter-P-Frame durations as follows:
  interval_P_frames = (burstPeriod - burstDuration) / (gopLength -1)
- Then we can organize the Frames in each GOP. An example of typical GOP can be found in the auto generated .dat file. Figure 2 illustrates the structure of a GOP obtained with the default values of attributes.
- Our next step is to transfer the characteristics into a packets generator while respecting the key attributes (e.g. average bit rate and packet size). Thus, we have to calculate the number of packets of each frame and then the interval between two consecutive packets of I-frames and P/B-Frames by sending packets of I-frame with "peakBitRate" and with "pFRate" for packets of frames P/B. In other words, the key parameter "avBitRate" for transmitting each GOP's data must still be verified.

  Therefore, the next time to send a packet is (maxPacketSize / dataRate), where dataRate is either the PeakBitRate or the pFRate according to the frame type.
- Using this calculated intervals, we can schedule sending events for packets of all the frames in each GOP, and we repeat the iteration again and again until the stop time of the application.

[3]We will delete the $m\_$ from the beginning of all variables for readability.

2.2.2 Usage and Validation

The module usage is extremely simple. The helper will take care of most things. Default values can be changed easily as shown in the code below.

The typical use is to create a UDP server Application that is listing on a particular port, and then to create the MPEG Packet generator client application using the desired values of its attributes as follows:

```
/*Create one udpServer applications on node one to
    receive the video packets from our generator*/
UdpServerHelper server (port);
ApplicationContainer apps = server.Install (n.Get (1));
apps.Start (Seconds (1));
apps.Stop (Seconds (100));
/*Create one MpegPktGenClient application on node zero*/
std::string fn = "mpeg.dat";
MpegPktGenClientHelper client (serverAddress, port,"");
client.SetAttribute ("MaxPacketSize", UintegerValue
    (1460));
client.SetAttribute ("VideoFilename", StringValue (fn));
client.SetAttribute("GopLength",UintegerValue (15));
client.SetAttribute("ImageRate",UintegerValue (12));
client.SetAttribute("IframeSize",UintegerValue (1200));
client.SetAttribute("AvBitRate",DoubleValue (2.0));
client.SetAttribute("PeakRate",DoubleValue (10.0));
apps = client.Install (n.Get (0));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (100.0));
```

A complete example is located in examples/mpeg-gen-pkt-client.

To validate the application, we present and discuss snapshots from the exchanged packets between the client and the

No.	Time	Source	Destination	Protocol	Length
1	0.000000	192.168.1.1	192.168.1.2	UDP	1490
2	0.001000	192.168.1.1	192.168.1.2	UDP	1490
3	0.002000	192.168.1.1	192.168.1.2	UDP	1490
4	0.003000	192.168.1.1	192.168.1.2	UDP	1490
5	0.004000	192.168.1.1	192.168.1.2	UDP	1490
6	0.005000	192.168.1.1	192.168.1.2	UDP	1490

Figure 3: Received packets of 1st frame (I-Frame) in the GOP.

No.	Time	Source	Destination	Protocol	Length
101	0.100000	192.168.1.1	192.168.1.2	UDP	1490
102	0.101000	192.168.1.1	192.168.1.2	UDP	1490
103	0.102000	192.168.1.1	192.168.1.2	UDP	1110
104	0.158000	192.168.1.1	192.168.1.2	UDP	1490
105	0.167000	192.168.1.1	192.168.1.2	UDP	1490
106	0.176000	192.168.1.1	192.168.1.2	UDP	1490
107	0.185000	192.168.1.1	192.168.1.2	UDP	1490
108	0.194000	192.168.1.1	192.168.1.2	UDP	1490
109	0.203000	192.168.1.1	192.168.1.2	UDP	1490
110	0.212000	192.168.1.1	192.168.1.2	UDP	1490
111	0.221000	192.168.1.1	192.168.1.2	UDP	1417
112	0.235000	192.168.1.1	192.168.1.2	UDP	1490

Figure 4: Received packets of 2ed frame (P-Frame) in the GOP.

server applications. The presented data are extracted from the pcap captured files. A graphical presentation of the captured data is displayed in Figures 2, 3 and 4, that are drawn using the packet analyzer program "wireshark".

We notice from Figure 2 that we have 15 frames, the first one is the I-frame that lasts for 0.102 s (burst duration) over the 1.25 s of the GOP duration (burst period), where the rest are the P/B frames. The first I-Frame consists of 103 IPv4 (IPv6 is supported) packets with an interval around 0.001 s between them because they are sent at the peak rate (e.g. 10 Mbps) as shown from Figure 4. The other frames consists of 8 IPv4 packets (7 of size 1468 B and one of size 1389 B as shown from packet N104 to N111 in the Figure 4) with inter packet arrival time around 0.009 s because they are sent with pFRate ($<$ avBitRate) as explained in the previous subsection. We remark as well that the first packet of each new frame starts at the corresponding starting time of the given frame as calculated in the GOP structure (the file mpeg.data). Also, from the statistics information obtained from wireshark and depicted in Figure 5, we see that the average bit rate is verified (e.g. 2 Mbps).

2.3 New 3D Antennas Patterns Modules

To run a realistic simulation, it is crucial to model the real radiation patterns of the different types of antennas that are used in the real CBTC implementation. In the real system, two types of antennas are used. A track-side antenna 'Ground antenna' and an on board shark antenna 'OBM antenna'. The first one consists of two back-to-back

Traffic	Captured	Displayed	Displayed %
Packets	2483	2483	100.000%
Between first and last packet	14,039 sec		
Avg. packets/sec	176,864		
Avg. packet size	1484 bytes		
Bytes	3683795	3683795	100.000%
Avg. bytes/sec	262397,251		
Avg. MBit/sec	2,099		

Figure 5: Statistics from the pcap file generated by wireshark.

antennas pointing in the direction tangent to the track and connected by a coupler (this yields to some db loss, usually 3db). The on board shark antenna is physically installed on the rooftop of the train, and screwed onto a horizontal metal ground plane. As described in Section 2.1, for increasing reliability, two OBMs are installed inside a train. Each OBM is attached to an OBM antenna. One antenna is pointing to the movement direction of the train (front antenna) and the second one is pointing at the opposite direction (back antenna). Both antennas should be oriented parallel to the track.

We implemented a new antenna model that imitates the radiation pattern of real antennas. In order to use this module, we need to provide the desired radiation pattern as an input text file as the one provided by the H&S antennas vendor [1].[4] The vendors have usually a tool via their web site that generate the data file for a given antenna model. Usually, the text file consists of the three columns that correspond to the azimuth, elevation angles and the gain values. The angles are used to define the apparent position of an OBM antenna or a Ground antenna in the polar coordinate system used by H&S. The module performs the mapping with the ns-3 coordinate system.

2.4 Handover and Fast Rerouting After it

Enhancing handover. We added a multi-channel scanning mechanism and modified the related objects at Channel, Physical, and MAC level, based on the received signal strength of beacons and the active probe response messages. An OBM will scan for some fixed time the current channel and the x next and previous channels, after switching to a new frequency, where x is typically 2. The OBM will then select the best channel to switch to and the corresponding AP to associate with.[5]

Fast rerouting after handovers. We have modified the static routing module and added some methods to the MAC layer objects to statically and immediately update the routing tables of the nodes affected by the association/disassociation events.

3. SELECTED EXPERIMENT RESULTS

The new modules help in understanding the impact of many factors on the performance of the system. In a real implementation of the CBTC system, the engineers have to adapt different system parameters (e.g. transmission power levels, distance between TREs, change Antenna's model, timer values, ...) when deploying a new line. The cost of these experiments in terms of time and equipment is a main difficulty for the rail transport companies. In particular, the estimation of spurious EBs rate through experimental measurements when important changes are done on a track is very costly, because we want to estimate events that occur at most once every 10^5 hours. An alternative solution is to use a detailed simulation. For illustrative purpose of the use of simulation for engineering the system, we will show visual presentation of selected results of three scenarios. In the first one we will see how to check the radio frequency (RF) model and study the impact of different antennas gain,

[4]One of the largest constructors of antennas for CBTC systems.

[5]Sharing the details and the code of this part is not authorized for confidentiality reasons.

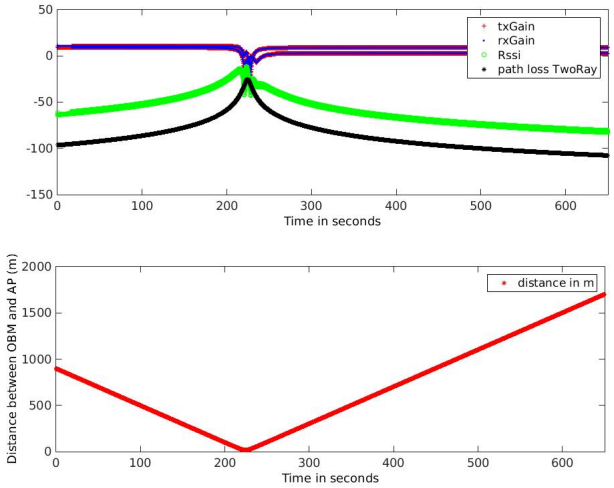


Figure 6: Evolution of RSSI, antenna gains and path loss versus train position around one AP.

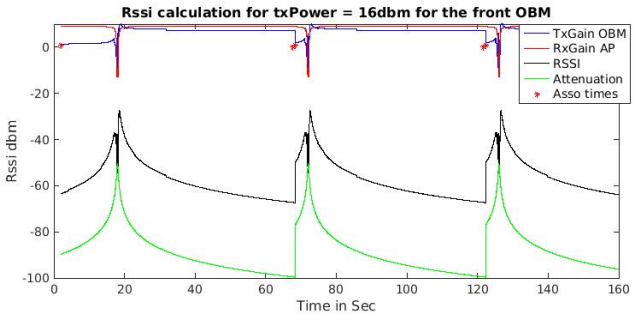


Figure 7: Evolution of RSSI, antenna gains and path loss versus train position with mobility and multible TREs.

transmitter/receiver power, the mobility and propagation models. We will show in the second one the behavior of the RF with handovers. We can study the impact of some important factors such as train's length as well as its speed and the distance between TREs. Last, in the third one, we show the exploitation of our modules for the estimation of the spurious EBs with a particular configuration. In fact, our ultimate goal is a fast estimator based on normal wireless network and traffic conditions. However, the simulation approach is not computationally efficient to evaluate a very rare event in normal conditions. Instead, we can evaluate the EB rate using simulation in degraded and critical scenarios (e.g. a high packet loss rate and a large latency due to interference between several trains of different speed and direction, handovers, different inter-TREs distance, different antennas gain, variable transmission power levels, and bursty multimedia traffic). For normal conditions, our approach to evaluate the EB rate is analytical. The robust developed analytical tool is a first step in this direction. We advocate to use ns-3 for validating EB rate estimations under degraded conditions and network provisioning.

First Experiment:

An important parameter to be observed and checked before tracing the CBTC messages is the received signal strength indicator (RSSI). In Figure 6, we depict the evolution of the RSSI with respect to the evolution of the antennas gain and the train position while assuming a single traveling train rightward at constant speed (e.g. 30 km/h) along a track that has only one AP. In fact, the RSSI determines when the amount of radio energy is above the modem sensitivity or the threshold power that must be received to correctly decode the packet (e.g. -79 dbm for data transmission rate of 36 mbps). It is impacted by the transmitter power, the transmission/receiving antenna gain, the orientation of antennas and the propagation model. In Figure 6, the two extreme points at initial and final time correspond to the front or back lobe for OBM antenna and a main front lobe for the Ground antenna model. Note that the former represents physically one single antenna whereas the latter represents two back-to-back antennas pointing in the direction tangent to the track as explained in Section 2.3. Notice that the AP transmits with the max gain and the OBM receives by the front lobe in the left part of the figure (i.e. before crossing the AP at time 250 seconds) and then it receives by the back lobe after crossing the AP. Also, when the train is close to the AP, the gain of both antenna's types degrades. However, when it is too close to the AP, the attenuation is less important and then the rssi remains above the threshold of the handover, consequently the OBM keeps its association with the current AP.

Second Experiment:

Figure 7 reports the evolution of the same metrics as in Figure 6 while considering many TREs with some distance between them (e.g. 400 m). The train will associate with the first AP, then when the RSSI level decreases below some value or after loosing some consecutive number of beacons, it starts a handover process to associate with a new AP after the scanning and the authentication times (e.g. some tens to hundreds milliseconds). Here we can monitor many important factors again as the distance between the TREs in addition to the transmission power, etc. We want to verify that the time to re-associate is close to what we observe in a real implementation. Note that the probability of loss during the handovers is 1. The loss probability before the handover phases depends on the rssi value and other factors, and then we aim at reducing this probability by correctly choosing the handover threshold, the antenna's type, their orientation, the transmission power and the distance between TREs.

Third Experiment:

Figure 8 reports the EB rate with different values of the loss probability out of the handovers periods. In this simple scenario, we fix the loss probability rather than using the calculated value by ns-3 (based on the SNR, etc.) for two reasons: (i) to simulate simply a degraded scenario without consider many trains with congested traffic and interference, and (ii) to present a new use of the simulation for our project; the validation of a mathematical formula that estimates the EB rate. From the figure, we see that for a constant packet loss out of the handovers the analytical results are within the confidence interval of the simulated results.

4. RELATED WORK

Studying train control systems has attracted considerable attention recently such as [7, 3, 2] that considered the abstraction of ETCS level 3 specifications. However, most of

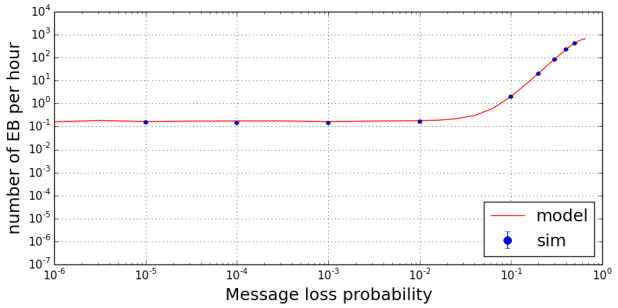


Figure 8: Number of emergency brakes per hour.

the work has been based on high level models using for instance the stochastic Petri Networks without a validation with realistic simulations. Some works developed Monte-Carlo simulation or numerical results of the Petri Networks. In both cases the dependence on the system parameters is hidden (e.g. handovers and track model). The proposed new modules aim at filling this gap.

5. CONCLUSION AND FUTURE WORK

The proposed CBTC simulation modules are useful for driving experiments in intelligent transport domain. Some developed modules, i.e. Video generator and directional Antennas, can be used in a wide range of research areas. It is worth mentioning that we enhanced some modules and discovered some bugs (e.g. some mishandled packets in CSMA, and fixed OFDM transmission rates for some control messages (e.g. cts) in the constant rate WiFi manager model).

Using ns-3 with the new modules permits to explicitly consider the impact of many important factors that are the cause of losses or delay and then contribute to triggering EBs. More precisely, we jointly study the impact of trains mobility, the real directional antennas pattern, 802.11e/WMM-style QoS support, and the bursty traffic with the handovers on the performance. This aspect was ignored in the previous literature. Our framework can be used for validating analytical approaches.

Using our ns-3 implementation we could answer the following questions: How to utilize train lines more efficiently? What is the QoS level of the CBTC messages and video traffic perceived by the system in a particular case?

The remaining barrier to enhancing our tool in the future is the consideration of fast fading models and more elaborate track models.

6. ACKNOWLEDGMENTS

This work is partially funded by the Inria-Alstom Transport joint lab.

7. REFERENCES

[1] HUBER+SUHNER official site. http://www.hubersuhner.com.

[2] L. Carnevali, F. Flammini, M. Paolieri, and E. Vicario. Non-markovian performability evaluation of ERTMS/ETCS level 3. In *Computer Performance Engineering (Proc. of EPEW 2015)*, volume 9272 of *Lecture Notes in Computer Science*, pages 47–62. 2015.

[3] F. Flammini, S. Marrone, M. Iacono, N. Mazzocca, and V. Vittorini. A multiformalism modular approach to ERTMS/ETCS failure modeling. *International Journal of Reliability, Quality and Safety Engineering*, 21(1):1450001 (29 pages), 2014.

[4] ns-3 official site. https://www.nsnam.org.

[5] ns-3 codereview issue of the cbtc module. https://codereview.appspot.com/289110043.

[6] ns-3 codereview issue of the video generator. https://codereview.appspot.com/286160043.

[7] A. Zimmermann and G. Hommel. Towards modeling and evaluation of ETCS real-time communication and operation. *Journal of Systems and Software*, 77(1):47–54, 2005.

APPENDIX

A. LIST OF ACRONYMS

The acronyms used in the paper are listed in below.

AP	Access Point
CBTC	Communication Based Train Control
CC	Carborne Controller
DCS	Data Communication Sub-System
EB	Emergency Brake
EOA	End-Of-Authority
ETCS	European Train Control System
LMA	Limit of Movement Authority
LOC	Location report
OBM	On Board Modem
RSSI	Received Signal Strength Indicator
TM	validity duration Timer of a LOC
TRE	Trackside Radio Equipment
ZC	Zone Controller

Implementation of 3D Obstacle Compliant Mobility Models for UAV Networks in ns-3

Paulo Alexandre Regis, Suman Bhunia and Shamik Sengupta
Department of Computer Science and Engineering
University of Nevada, Reno
Reno, NV, USA 89557
{pregis, sbhunia}@nevada.unr.edu, ssengupta@unr.edu

ABSTRACT

UAV networks are envisioned to play a crucial role in the future generations of wireless networks. The mechanical degrees of freedom in the movement of UAVs provides various advantages for tactical and civilian applications. Due to the high cost of failures in system-based tests, initial analysis and refinement of designs and algorithms for UAV applications are performed through rigorous simulations. Current trend of UAV specific simulators is mainly biased towards the mechanical properties of flying. For network-centric simulations, the intended measurements on the performance of protocols in mobile scenarios are conventionally captured from general-purpose network simulators, which are not natively equipped with comprehensive models for 3D movements of UAVs. To facilitate such simulations for UAV systems, this paper presents different mobility models for emulation of the movement of a UAV. Detailed description of three mobility models (random walk, random direction, and Gauss-Markov) are presented, and their associated movement patterns are characterized. This characterization is further extended by considering the effect of large obstacles on movement patterns of nodes following the three models. The mobility models are prepared as open-source add-ons for ns-3 network simulator.

CCS Concepts

•**Networks** → **Network simulations;**

Keywords

UAV, 3D, mobility model, obstacle, ad hoc network, ns-3

1. INTRODUCTION

With the advancement in the unmanned aerial vehicles (UAV) technologies, three-dimensional mobile networks are gaining popularity as 3D mesh networks. In addition to movement in a horizontal plane as in conventional mobile ad hoc networks (MANET), UAVs enable nodes to move in the vertical direction in 3D as well. The autonomous movement of UAVs in all directions makes it suitable for a wide range of operations, such as border surveillance, disaster monitoring, firefighter networks, relay for ground vehicles, tactical networks, imagery and sensing in rough terrain areas, and so forth [7]. Infrastructure networks are conventionally used to establish links with UAVs. But as the spatial diversity increases, the deployed infrastructure is not adequate to always associate UAVs with fixed access points. As a result, multi-hop mesh networks are gaining popularity. Data can be relayed over multiple UAVs to reach the destination without direct communication, either due to the link being blocked by a large obstacle, or the end nodes being out of communication range.

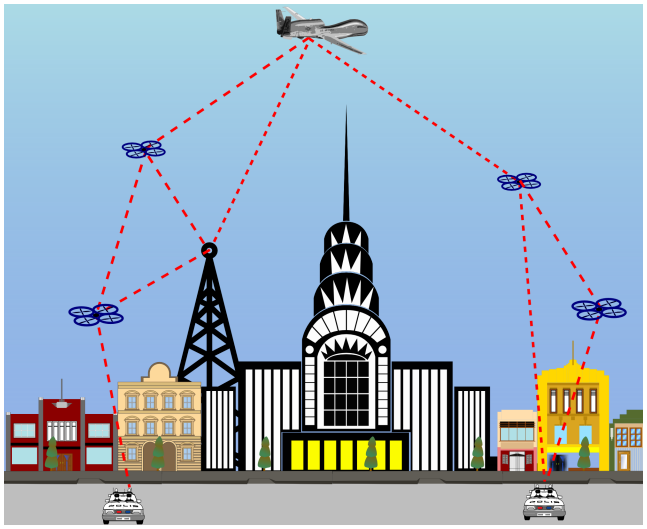


Figure 1: An example of a UAV 3D mesh network in a urban environment with obstacles.

Testing with UAVs is a very costly, time-consuming event which requires a lot of manpower, since failure in communication or any other mechanical malfunction may cause fatal accidents. Also, in a real test environment, there are many parameters involved making the comparison of two different algorithms very difficult to perform. For example, the signal propagation characteristics change throughout the day, or even with the local humidity. In addition to all these factors, flying UAVs in outdoor spaces is prohibited due to the risk of accidents with civilians. However, since 3D mesh network is gaining its importance in scenarios such as first responders in disaster situations or firefighter networks, the

WNS3, June 15-16, 2016, Seattle, WA, USA

DOI: http://dx.doi.org/10.1145/2915371.2915384

required network protocols must be tested intensively before deployment. Simulations are used widely to model the networks that can emulate the behavior of UAVs. Simulation can reproduce the exact same environment when comparing two different protocols. Another huge problem arises when testing the scalability of protocols, i.e. the reaction of a protocol when a larger number of nodes is present in the network. After performing rigorous simulations, a protocol or a model is tested with real hardware. To obtain realistic performance, some networks with mobility should be tested in an environment with large obstacles, the deployment of which can be financially infeasible. Federal Aviation Administration restricts flying and testing with UAVs to mostly rural areas such as in the state of Nevada [1]. Since testing UAV networks in urban areas is not possible, simulation of UAV networks should be done in a reliable network simulator that provides seamless integrity of mobile nodes (MNs) with a wide range of upper layer protocols, and also provides the capability of 3D mobility with obstacles.

An example of 3D UAV mesh can be seen in Figure 1. Different classes of UAVs are depicted at different layers. One hop links between two neighbors fluctuate due to multipath signal propagation through reflection at large objects. The upper layer network protocols are affected by the mobility of MNs as well as the corresponding environment and terrain [15]. For example, in a city with multistoried buildings or in an irregular terrain, the link between MNs may break when a building is between them. Since the application scenarios are not limited, UAV mobility is desired to be integrated with simulator where the application and other protocols are implemented. The movements of MNs are created using standard mobility models. These models are now incorporated into the networks simulator to describe the migration of MNs. The mobility of an MN can be represented in a simulator in two ways: *traces* and *synthetic models*. Traces are used to provide real-life recorded data as a time-series position information. Traces are useful to simulate accurate systems, but in multiple MN scenarios with long time simulation it is space consuming. Synthetic models are widely used in simulators to mimic the movement of MNs movement in MANET for performance evaluation. Mobility models characterize how MNs change their velocity and direction over time. A wide range of mobility models are presented in literature such as *random walk, random waypoint, random direction, Gauss-Markov, column formulation*, and *nomadic community*. A detailed description of the mobility models commonly found in network simulators can be found in [9] where a comprehensive study on the mathematical properties of many models is detailed.

This paper introduces three extended mobility models to allow obstacles in the environment. The models were developed for ns-3 [2] using the existing implementations when possible. We show preliminary network performance results, and also provide free access to the source code[1]. The main contributions of the paper are:

- Description of three-dimensional mobility models
- Algorithms for how the models behave with obstacle avoidance
- Open-source implementations of the models in ns-3

[1]http://www.cse.unr.edu/~regis/ns-3

The remainder of the paper is organized as follows: Section 2 provides an overview of the existing systems and their weaknesses. Sections 3, 4, and 5 discribe the implementation of the new models and features. A brief demonstration and performance comparison of these models is presented in Section 6. Finally Section 7 concludes the paper.

2. MOBILITY MODELS AND SIMULATORS

Mobility model is a set of rules that governs how a mobile node moves. It rules how the velocity, acceleration, and location of a node change over time. These mobility models are necessary for simulation purposes when investigating new communications protocols and techniques. Currently many mobility models have been proposed, but their movements are inspired by specific applications and scenarios.

For example, the Manhattan Mobility model [17] uses a grid-like topology of paths in which a node is constrained to move. This model aims to imitate the mobility of vehicles in a urban city environment, and is particularly useful in simulating vehicular ad hoc networks (VANETs). However, the movements of UAVs are not limited by the street paths of a city. If a node has the ability to fly with high enough altitude, its movements may not be physically limited at all. In these cases, more suitable mobility models should be used to simulate the trajectory. Here is where random models come into the picture, to simulate the movements of a node without the knowledge of its mission or task. The Random Walk mobility model defines a sequence of random steps in which a node traverses a certain area. This model establishes how long and in which direction the node should move before changing its direction once again. The problem with this model is that the node density in the simulation is more concentrated in the center of the scenario and less near the boundaries. The Random Direction mobility model is a little different from the random walk, because it forces the node to travel until it reaches the border before changing direction, not limiting the trajectory by a certain amount of time. Another branch of the random models are the temporal dependency mobility models. A popular example is the Gauss-Markov model, in which the direction and speed of a node depends on its previous direction, thus limiting these parameters within a certain range [9].

Authors in [12] proposed a new MANET mobility model called Realistic Mobility Model, in which the node velocities and directions are based on probability distributions that imitate real user mobility behavior. The results create trajectories that resemble real mobility traces. The authors in [11] presented a new VANET mobility model. The movements are limited to the streets in a city, and the speed of a node changes based on its neighboring nodes, similar to a real situation in the streets. In [13] two different mobility models for common applications of fixed wings aircrafts are proposed. The first is a simple random model with dependency on the last action taken by the aircraft: turn right, turn left or straight ahead. The second is a pheromone repel model, where the probability of taking an action depends on the other aircraft's movement as well. The models however were created to simulate cooperative reconnaissance applications. [8] proposed a mobility model for UAVs based on the most common movement patterns executed by commercialized products such as the paparazzi. The node selects one

of the predefined patterns based on the probability of a pattern being chosen. Despite the complexity and resemblance of the models to the real world, there is a lack of proposed models for three-dimensional scenarios, and even less when it comes to obstacles. In an effort to fill this gap, this paper presents two mobility models that are both obstacle and 3D compliant, imitating the movements of modern UAVs.

Simulation plays an important role when creating new networking techniques and protocols. Before building the hardware and software that implements a new feature, the simulations help to decide the feasibility of the new technology; if it does not perform well, it most likely will not perform well in the real world. The mobility of nodes is an important aspect of a network. It directly influences the topology of the network, the links formation, and link failures. Therefore, it is critical to build reliable simulation models to assist the development of more advanced technologies. A network simulator consists of a series of different models that imitate certain behaviors and protocols, glued together to analyze the performance of the system. There are some network simulators available for use that implement these mobility models. OPNET [5] is a commercial platform specialized in telecommunication transmission products for access networks. OMNeT++ [3] is a simulation framework that relies on different modules to provide functionalities rather than having an extensive core library. Its modules are developed independent of each other, following their own development schedule. ns-3 [2] is a discrete event network simulator, created to overcome the weaknesses of its ancestor ns-2. It is designed to facilitate the creation and integration of new models into the core of the simulator. The project is maintained by a community of researchers affiliated with reliable educational institutions, and main libraries receive constant scheduled updates. It offers a means to integrate third-party tools, such as Simulation of Urban MObility (SUMO) [6]. Since the primary purpose of this simulator is to serve the educational and research communities, its modules and source codes are kept under open-source license. Due to its easy integration and support from the community, ns-3 was chosen as the development platform for the work presented in this paper. ns-3 assumes the Cartesian coordinate system represented by the tuple (x, y, z), while the mobility models use the spherical coordinate system composed by (r, θ, ϕ). In this paper the radius r can be referred to as v when representing speed. Figure 2 shows the coordinate systems.

There are many different applications for a UAV network: surveillance, border control, first response in emergency situations, and so on. The movement patterns of each mission can be very different from the other. In the same network for example, different nodes may perform various tasks, but still serve as relay nodes in the communication network. Current simulators lack the ability to imitate the movements enabled by the new UAV technologies. UAVs can, for example, change rapidly its direction in both two and three dimensions. They can also share the space in civilian environments with obstacles like buildings and houses. This paper aims to fill this gap by providing new mobility models that allow the simulation of obstacles, and implementing it in a reliable network simulator to be used by the research and educational communities.

The set of libraries contained in the core of ns-3 platform provide the means to simulate not only the common Internet

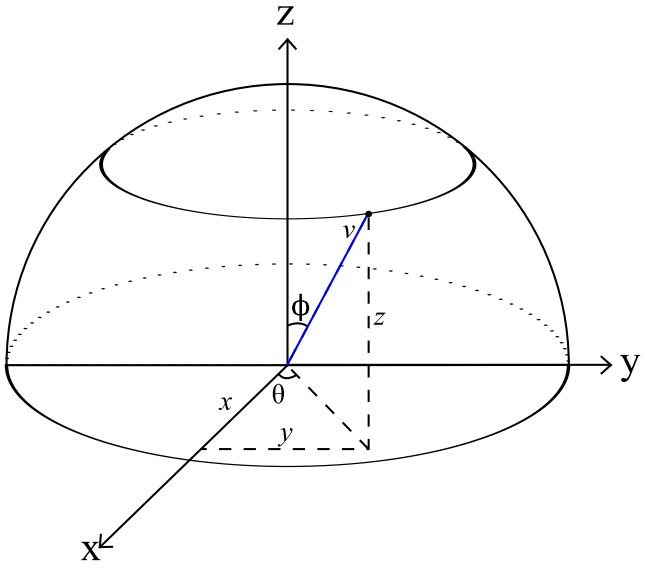


Figure 2: Visualization of coordinates in both cartesian and spherical systems.

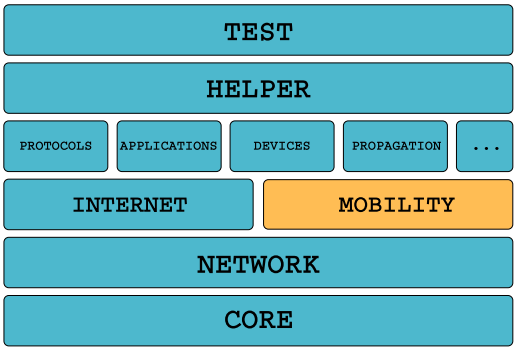


Figure 3: ns-3 block diagram.

but also other networking protocols. The mobility models relevant to this work already implemented and integrated into the ns-3 core libraries are:

- Random Walk (2D)
- Random Direction (2D)
- Gauss-Markov

ns-3 also implements different propagation loss models that accounts for the effect in transmission when buildings are present in the environment. The buildings module provides different physical characteristics of the construction. Commercial buildings, for example, affect differently than residential ones based on their common building materials. This model can be easily reused to construct obstacle compliant mobility models.

An overview of the building blocks and how the simulator works is shown in Figure 3. The core block is where the simulator implements classes with functionalities to facilitate the development of the actual models, like smart-pointers, tracing, logging, and event scheduler. The network

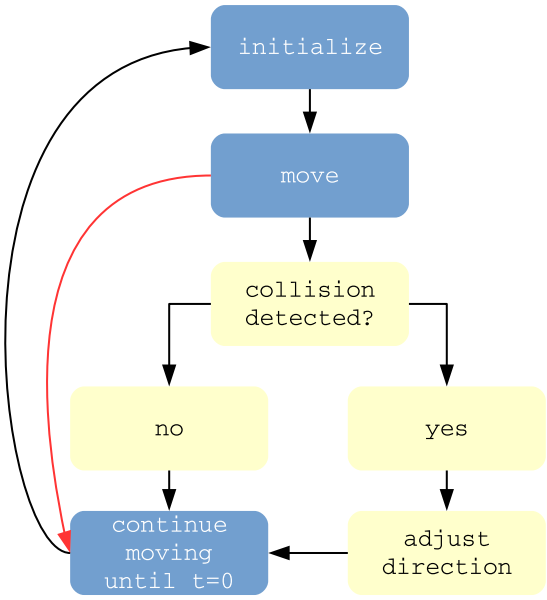


Figure 4: Simplified block diagram of collision avoidance.

block provides abstract base classes for common objects such as packets. These modules are independent of specific networks, and comprise the core that may be used by the upper modules to implement more sophisticated features in a network, not exclusively Internet related. Modules depend only on the modules beneath them. The mobility module, for example, only uses components implemented by the core, while propagation modules depend on the mobility model beneath. The focus of this work is on the mobility module; inside this block resides all the other mobility models implemented in the simulator. Currently the mobility models available to the community do not consider obstacles, they simple assume the node moves in a straight line for a certain period of time.

The flowchart in Figure 4 illustrates where the contributions in the mobility models reside. The adaptations verify if the node collides with the obstacles and adjust the attributes accordingly. Previously the models simply moved from one point to another until the end of the current cycle, denoted by time t in the figure. The new models presented in this paper can be easily used in conjunction with other models to simulate a UAV network. A simple example is the use of SUMO to emulate the movements of ground vehicles while utilizing the 3D models to imitate UAVs, all in the same network. Street maps and building information can be obtained from open sources, such as OpenStreetMap [4], and used with SUMO to generate traces for VANETs [10].

3. RANDOM WALK 3D

Originally the Random Walk model was created to imitate the behavior of particles in physics, but it was later used to simulate movements of nodes in mobile networks. The original Random Walk model implemented in ns-3 defines the movements of a node by the random variables θ, that changes the direction in which the node moves, and speed $v \in [V_{min}, V_{max}]$ contained in a predefined speed interval. This model has two modes: *time mode* and *distance mode*. The first one explicitly decides for how long a node will keep its current speed and direction before choosing new values. The later also decides the time before new values are chosen, but based on the current speed of the node and the predefined *distance* value: $time = distance/speed$. If the mobile node reaches the boundary of the simulation before the resetting process is executed, the node simply bounces back by changing its direction in the corresponding axis and continues moving for the remaining time. For example, if the x-axis is the boundary limit (equivalent to $y = 0$), then the direction in y of the velocity vector is inverted by multiplying it with -1. Similarly, if y-axis is the limit ($x = 0$) the x component of the velocity is inverted.

To enable this model to function in a three-dimensional world, a new random variable is introduced, the aforementioned $\phi \in [0, \pi]$. In addition to the speed and direction, now the node also selects a new pitch at every cycle. Instead of having lines as limits, now the model assumes planes. If the node reaches the limit $xy - plane$ for example, the z component of the velocity vector is inverted to bounce the node back into the scenario $\mathbb{S}$.

Realistic scenarios such as urban environments are not simply empty places where nodes move freely. These scenarios actually introduce certain constraints to the mobility. Even when they move in a 2d plane that is lower than a skyscraper, obstacles must be considered. The implementation of the 3D mobility model in ns-3 provides the feature to simulate movements in the presence of obstacles. The obstacles are assumed to be boxes and axis aligned, meaning their faces are parallel to the axis planes. The effect caused is similar to the bouncing velocity vector: if the node collides with an obstacle it changes the corresponding component of the velocity, depending which side of the object was hit.

In each cycle of the Random Walk model the node chooses new speed, direction, and pitch. These parameters basically define the velocity vector of the node. Then it calculates the next position based on the *time* it is supposed to move with those parameters. If during this period any obstacle is encountered, the orthogonal component of the velocity vector is inverted as mentioned. After the inversion it continues to move with the new values for the remaining time of the cycle, after which a new iteration takes place.

Algorithm 1 shows the pseudo-code of the random walk 3D with obstacles. The implementation required new methods to be added in original classes of the source-code: namely the base classes *ns3::Box* and *ns3::Rectangle*. These modifications are used in all new mobility models. The new method *IsInside(position)* simply verifies if a *position* is inside the object. A method called *WillCollide(position,velocity)* is introduced to the *ns3::Box* class, it verifies if the trajectory of a node will intersect with the box and also returns the intersection point. The new *ns3::RandomWalk3dMobilityModel* class is similar to the original 2d model, except by the new angle variable and the *AddObstacle(obstacle)* method. The *Rebound()* function simply multiplies some of the velocity vector components (*x,y,z*) by -1, depending on which surface of the object collision occurred.

4. RANDOM DIRECTION 3D

In [16] this model was first proposed with the objective of eliminating the concentrated node density problem that occurs in the Random Waypoint Model. In this model the nodes randomly select a point inside the limited area and

Algorithm 1: Random Walk 3D

Input: Boundary ($\mathbb{S}$), set of obstacles ($\mathbb{O}$), default distance ($default\_distance$) or $default\_time$, and Speed limits $[V_{min}, V_{max}]$
Output: List of Time and Position pairs

repeat
  if $mode = time$ **then**
    $default\_distance \leftarrow default\_time * v$;
  $\theta \leftarrow \mathcal{U}(0, 2\pi]$;
  $\phi \leftarrow \mathcal{U}[0, \pi]$;
  $v \leftarrow \mathcal{U}[V_{min}, V_{max}]$;
  $distance \leftarrow default\_distance$;
  $collides \leftarrow false$;
  for $O_i \in \mathbb{O}$ **do**
    if `WillCollide`$(x, y, z, \theta, \phi, O_i)$ **then**
      if $\Delta d >$ `DistToObstacle`$(x, y, z, \theta, \phi, O_i)$ **then**
        $\Delta d \leftarrow$ `DistToObstacle`$(x, y, z, \theta, \phi, O_i)$;
        $collides \leftarrow True$;
  if $collides$ **then**
    $Rebound(\Delta d)$;
  $x \leftarrow x + sin(\theta)cos(\phi)\Delta d$;
  $y \leftarrow y + sin(\theta)sin(\phi)\Delta d$;
  $z \leftarrow z + sin(\phi)\Delta d$;
until $simulation\ ends$;

move there with a certain speed. Since the probability of choosing a point closer to the boundaries is lower than in the middle, the nodes in the scenario eventually end up concentrating in the center. The Random Direction model mitigates this problem by forcing the node to travel until it reaches the boundary of the delimited area.

In the Random Direction model, three variables govern the trajectory of a mobile node: pause P, direction θ, and speed v. Similar to the Random Walk model, these attributes are constrained in their limits. First the node chooses a speed and direction in which it will move. Then it continues moving until it reaches the boundary, standing by that position for P amount of time before reseting the parameters and beginning the cycle once again. Intuitively, the node will only move again if the new direction points towards the middle of the limited area, otherwise it will remain in the paused state.

The movements in Z-axis are allowed by the introduction of the pitch ϕ in the system. In the main loop of the algorithm the model selects the values for the random variables ϕ, θ, and v. Then it calculates the intersection of the new trajectory path of the node with each obstacle inside the scenario, and the scenario boundaries itself. The next position where the node will randomize the values again is the intersection with the smallest distance to travel. If an object is not in the path of the node, the intersection distance returns infinity. After the node reaches the destination, it pauses for a random amount of time, constrained within the interval $P \in [P_{min}, P_{max}]$.

Algorithm 2 shows the pseudo-code of the random direction 3D. The implementation utilizes the same new methods included in base classes *ns3::Box* and *ns3::Rectangle* in the source-code. The class *ns3::RandomWalk3dMobilityModel* also adds a new angle to allow the movement in z-axis. To keep consistent, the same *AddObstacle(obstacle)* method was implemented.

Algorithm 2: Random Direction 3D

Input: Boundary ($\mathbb{S}$), set of obstacles ($\mathbb{O}$), Pause time limits (P_{min}, P_{max}) and Speed limits $[V_{min}, V_{max}]$
Output: List of Time and Position pairs

repeat
  $Pause(\mathcal{U}[P_{min}, P_{max}])$;
  $\theta \leftarrow \mathcal{U}(0, 2\pi]$;
  $\phi \leftarrow \mathcal{U}[0, \pi]$;
  $v \leftarrow \mathcal{U}[V_{min}, V_{max}]$;
  $\Delta d \leftarrow$ `DistanceToBoundary`(x, y, z, θ, ϕ);
  for $O_i \in \mathbb{O}$ **do**
    if `WillCollide`$(x, y, z, \theta, \phi, O_i)$ **then**
      if $\Delta d >$ `DistToObstacle`$(x, y, z, \theta, \phi, O_i)$ **then**
        $\Delta d \leftarrow$ `DistToObstacle`$(x, y, z, \theta, \phi, O_i)$;
  $\Delta t \leftarrow \Delta d / v$;
  $x \leftarrow x + sin(\theta)cos(\phi)v\Delta t$;
  $y \leftarrow y + sin(\theta)sin(\phi)v\Delta t$;
  $z \leftarrow z + sin(\phi)v\Delta t$;
until $simulation\ ends$;

5. GAUSS-MARKOV

The movements of an object in real world may be limited and constrained by the laws of physics. Hence, the change in velocity and direction may depend on the previous values of these attributes. This correlation between the speed/direction in different iteration time slots is not considered when Random Models are used. In random approaches, the nodes' movements and speed may vary abruptly from one instant to another. But when considering real objects like cars, helicopters, and airplanes, even though some may be able to change direction faster than others, these movements are not as quick as the random models imply. The Gauss-Markov mobility model, on the other hand, adds this temporal dependency characteristic when the mobile node is moving around the scenario, enabling the limitations of real world movements to be emulated.

This model assumes the velocity of a mobile node is correlated over time. The changes in velocity are modeled as a Gauss-Markov stochastic process. The new value is constrained within a limited range, with a mean and standard deviation. The model also has a tunning parameter $\alpha \in [0, 1]$, known as the memory level. This parameter dictates the importance of old values when calculating the new ones. This tunning parameter gives the ability to imitate the behavior of other models. For example, if α is low, the velocity is determined by random variables and not the previous values, making the behavior closer to the ones of random models.

The Gauss-Markov model also assumes a bounding box to limit the movements of the mobile node. The velocity and directions of the node are changed after a predefined amount of time. During this time interval the node moves with fixed speed and direction. When it finishes moving for the interval, new values are chosen based on the previous

ones and the system defined parameters. The system parameters define, for example, the deviation the new speed will have from the previous one. If the node moves towards the limits of the simulation, the direction is shifted by 180 degrees (πrad), forcing the node to stay within the boundaries. More details of the original model can be found in [14].

The implementation of obstacles in this model follow the same principle as when the node gets closer to the bounding limits. If the mobile node is moving towards an obstacle, its direction and pitch are shifted, forcing it to bounce back inside the simulation limits. The trajectory appears to be smoother than the random models, giving the simulations a more realistic behavior of a mobile node's movement. Since the 3D Gauss-Markov model is already implemented in ns-3, the only modifications made were to allow the addition of obstacles with *AddObstacle(obstacle)* and the collision verification, which is basically the same as the one to verify if the node is out of bounds, and adjust the new directions accordingly.

6. DEMONSTRATION AND RESULTS

To illustrate the behavior of the enhanced models, a sample trajectory for each of the described methods is shown in Figure 5. In Figures 5a and 5b the Random Walk models are configured in different modes: *time* and *distance* respectively. Figure 5c shows an example of the Random Direction model, while the trajectory in Figure 5d is defined by the Gauss-Markov model. Only one obstacle is placed in this illustration to facilitate the visualization of the sampled trajectory. The object is centered in $(50, 50)$ and has dimensions of $100m$ width and length, and $200m$ height.

The simulated scenario has dimensions of $300 \times 300 \times 200(m)$. The speed of the node is set to an average of $25m/s$, and the simulation runs for $200s$. In the *distance* mode of the Random Walk, the node travels for $200m$ (equivalent to *time* mode with $200/25 = 8s$ interval) before changing direction, while in the *time* mode it travels for $10s$ before resetting.

Table 1: Obstacle dimensions.

Parameter (meters)	Obstacle 1	Obstacle 2
Center coordinates	(50,50)	(150,150)
Width	100	100
Length	100	100
Height	100	200

To provide some insights on the behavior of an ad hoc network, we executed preliminary simulations comparing the performance in two cases: with and without obstacles. When simulating with obstacles, two objects are placed in the simulated scenario. Both obstacles have the characteristics of solid *stone blocks*, and the detailed position and dimensions are depicted in Table 1. We traced the packet delivery ratio of control packets (CPDR) of the Optimized Link State Routing (OLSR) protocol. The general parameters for the simulations are as follows.

By varying the number of nodes (5 to 50), it is possible to see the degradation in the performance when obstacles are considered. For example, the Gauss-Markov model has CPDR of 80.9636% and 83.9625% when simulating 50 nodes with and without obstacles, respectively. The same trend can be observed for the other models as well, and the performance is also influenced by the mobility model utilized.

Table 2: Simulation parameters.

Parameter	Value
No. of nodes	20
Simulation time	200 s
Traffic generator	OnOff Application
Transport protocol	UDP
Routing protocol	OLSR
WiFi standard	IEEE 802.11b
Datarate	1 Mbps
Speed	5 to 25 m/s (25 default)
Propagation loss model	Okumura-Hata

From the graph in Figure 6 we see that Gauss-Markov model results in a deteriorated performance compared to the other random models.

Figure 7 illustrates the performance of the network. The traffic was generated by half of the nodes, the other half acts as the receiver (i.e. there are 10 information flows with distinct sources and destinations). We compare the results in both cases, with and without obstacles. We can see that the number of reflections increases in a linear trend according to the speed (when there is no obstacle, there is no reflection), as it would be expected. A counter-intuitive result is the hop count, which is in general lower when there are obstacles in the scenario. This happens because the nodes are more restricted in their movements, forcing them to stay closer to each other. Hence, if there is no obstacle, they have more room to move, and the probability of augmenting the distance is higher. Following the same logic, if the number of hops increases, the end-to-end delay and jitter also increase if there is no obstacle.

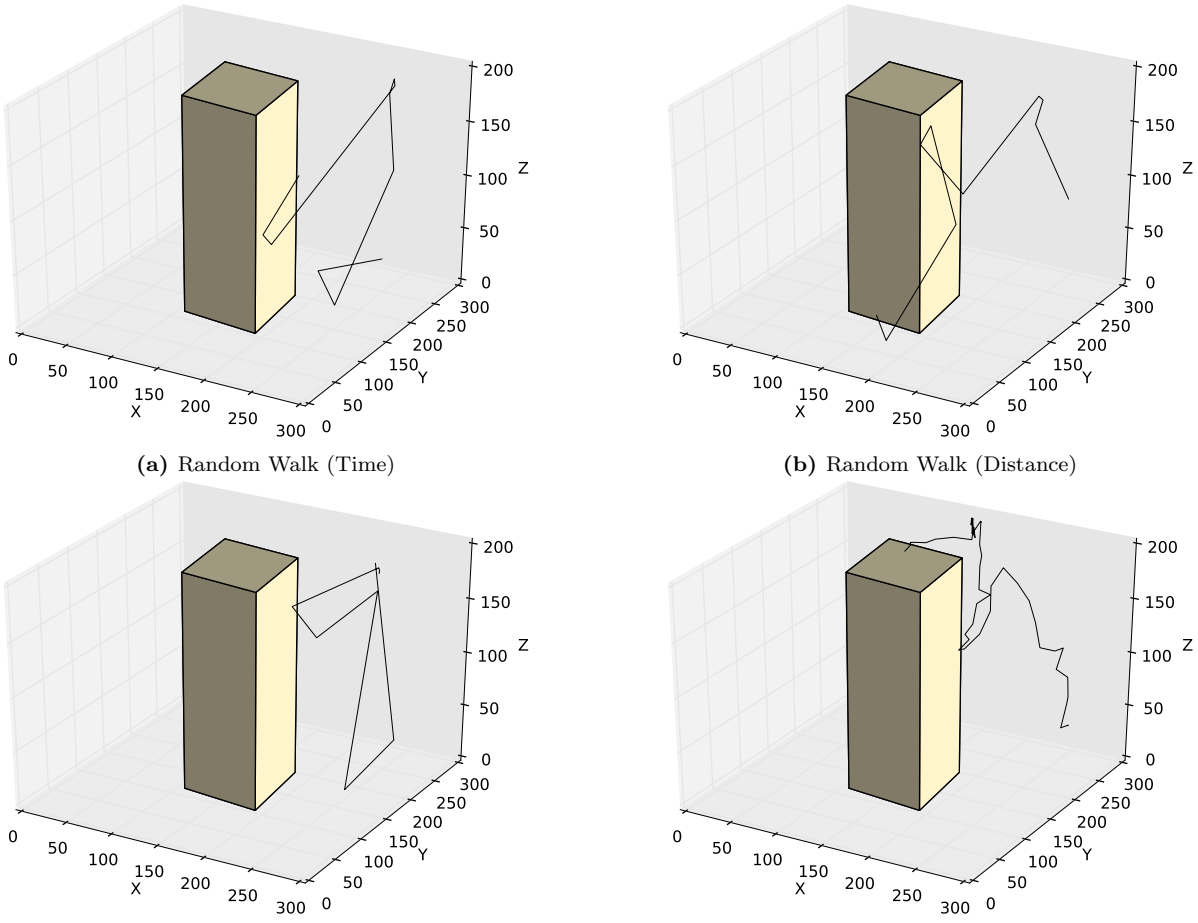
7. CONCLUSION AND FUTURE WORK

In this paper we implement the three different mobility models in a reliable network simulator framework to emulate a three-dimensional world. One requirement for the development of the models is to make them obstacle compliant, to meet the requirement when simulating realistic environments such as urban areas with buildings or mountains. The models are implemented in the network simulator ns-3, due to its open-source nature, and the focus in research and educational communities. The models were demonstrated in a simple scenario, resulting in the expected degradation of performance when the obstacles are present in the simulation.

In the future, more complex algorithms for obstacle avoidance can be implemented to help simulate in case the path of the node is known, like in a particular mission known to the system developer. Such algorithms already exist and are constantly being studied by the robotics research community. Integrating these two remains a challenge to be explored. This work provides the means to anyone interested in quickly deploy simulations of UAV networks, without the need of external tools to obtain the movements of the nodes in the network.

8. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation, Partnership for Innovation Program, Grant No. 1430328 and CAPES Brazil 13184-13-0.



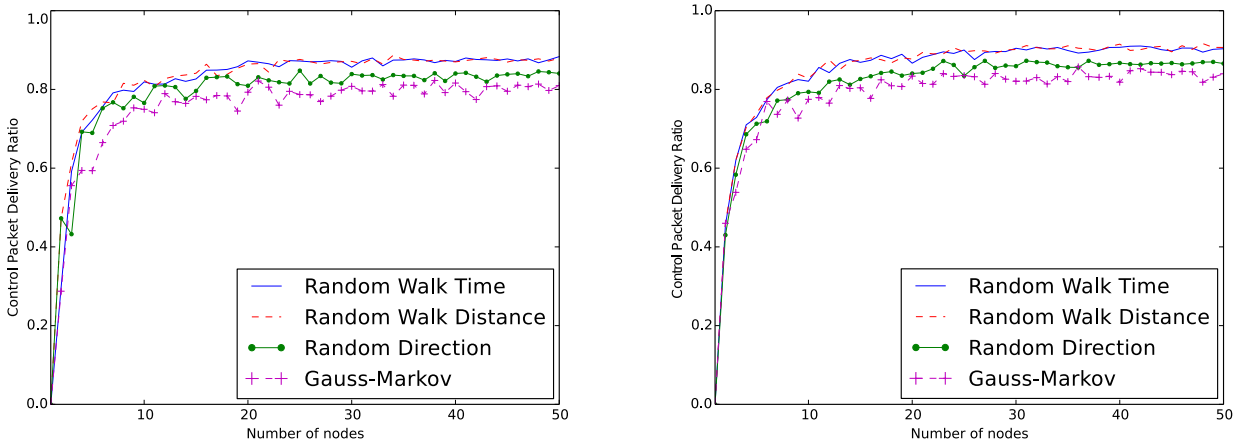
(a) Random Walk (Time)

(b) Random Walk (Distance)

(c) Random Direction

(d) Gauss-Markov

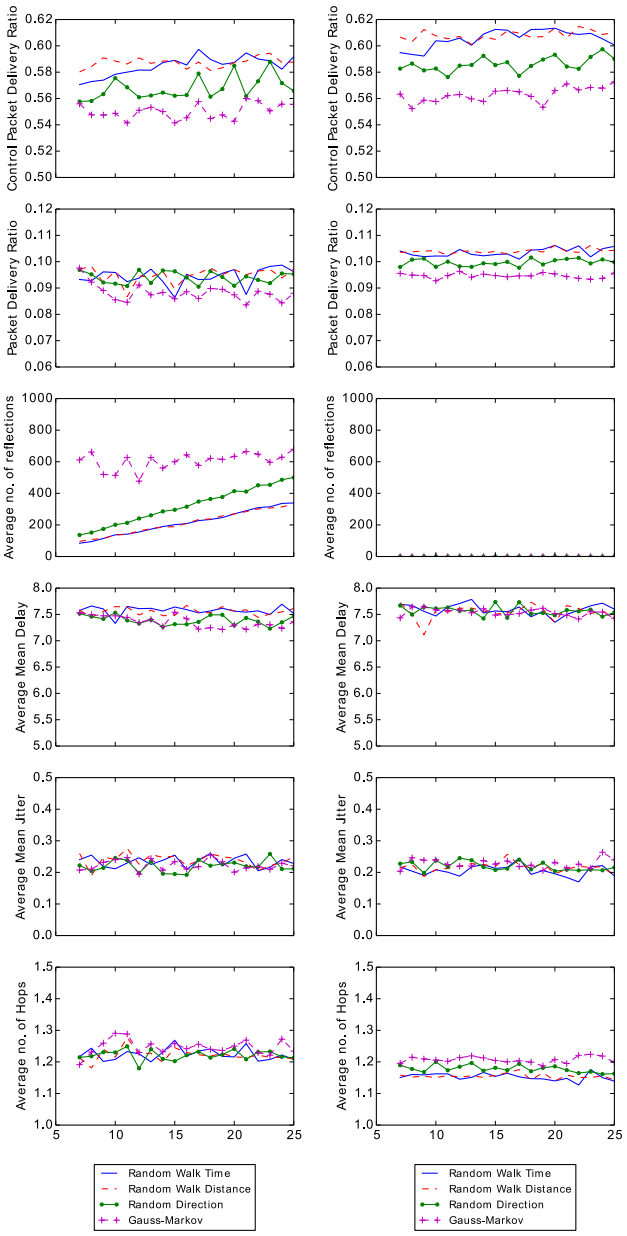
Figure 5: Trajectory with one obstacle.



(a) With obstacles

(b) Without obstacles

Figure 6: Control packets delivery ratio vs number of nodes.



(a) With obstacles **(b)** Without obstacles

Figure 7: Network performance parameters.

9. REFERENCES

[1] Federal Aviation Administration approved UAS test sites. https://www.faa.gov/uas/legislative_programs/test_sites.

[2] ns-3. https://www.nsnam.org.

[3] OMNeT++. https://omnetpp.org.

[4] Open Street Map (OSM). https://www.openstreetmap.org.

[5] Opnet. http://www.riverbed.com.

[6] Sumo. http://sumo.dlr.de.

[7] I. Bekmezci, O. K. Sahingoz, and Ş. Temel. Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254–1270, 2013.

[8] O. Bouachir, A. Abrassart, F. Garcia, and N. Larrieu. A mobility model for uav ad hoc network. *2014 International Conference on Unmanned Aircraft Systems (ICUAS 2014)*, pages 383–388. IEEE, 2014.

[9] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[10] S. E. Carpenter and M. L. Sichitiu. An obstacle model implementation for evaluating radio shadowing with ns-3. In *Proceedings of the 2015 Workshop on ns-3*, WNS3 '15, pages 17–24, New York, NY, USA, 2015. ACM.

[11] D. S. Gaikwad and M. Zaveri. A novel mobility model for realistic behavior in vehicular ad hoc network. *IEEE 11th International Conference on Computer and Information Technology (CIT 2011)*, pages 597–602. IEEE, 2011.

[12] A. E. Kamal and J. N. Al-Karaki. A new realistic mobility model for mobile ad hoc networks. *IEEE International Conference on Communications (ICC 2007)*, pages 3370–3375. IEEE, 2007.

[13] E. Kuiper and S. Nadjm-Tehrani. Mobility models for uav group reconnaissance applications. *International Conference on Wireless and Mobile Communications (ICWMC 2006)*, pages 33–33, July 2006.

[14] B. Liang and Z. Haas. Predictive distance-based mobility management for pcs networks. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1377–1384, March 1999.

[15] D. B. J. D. A. Maltz and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Computer Science Department Carnegie Mellon University Pittsburgh, PA*, pages 15213–3891, 2001.

[16] E. Royer, P. Melliar-Smith, and L. Moser. An analysis of the optimum node density for ad hoc mobile networks. *IEEE International Conference on Communications (ICC 2001)*, volume 3, pages 857–861, 2001.

[17] E. SMG. Universal mobile telecommunications system (umts); selection procedures for the choice of radio transmission technologies of the umts. *ETSI Document TR*, 101:112, 1997.

Topology Simulation for Aeronautical Communication Protocols with ns-3 and DCE

Andreas M. Lehmann
Chair of Information Transmission
Friedrich-Alexander-Universität Erlangen-Nürnberg
andreas.lehmann@fau.de

Matthias Kreuzer
Chair of Information Transmission
Friedrich-Alexander-Universität Erlangen-Nürnberg
matthias.kreuzer@fau.de

Jörg Deutschmann
iAd GmbH
Unterschlauersbacher Hauptstr. 10
D-90613 Großhabersdorf, Germany
software@iad-de.com

Ulrich Berold
iAd GmbH
Unterschlauersbacher Hauptstr. 10
D-90613 Großhabersdorf, Germany
hardware@iad-de.com

Johannes B. Huber
Chair of Information Transmission
Friedrich-Alexander-Universität Erlangen-Nürnberg
johannes.huber@fau.de

ABSTRACT

The currently used aeronautical communication systems have huge limitations and need to be replaced in order to meet the requirements for air-ground communications due to a constant growth in air traffic. In order to develop, test and optimize protocols it is desired to have proper tools to simulate large scale (up to global) networks of ground and mobile stations. We propose an ns-3 topology module that is tailor fit for aeronautical communication protocols, compatible with typical location information and provides propagation parameters for communications between nodes.

CCS Concepts

•**Networks** → **Network protocols;** •**Applied computing** → **Avionics;** •**Information systems** → *Information systems applications*.

Keywords

Avionics, communication, navigation, LDACS, ns-3, DCE, protocol design, physical layer, network simulation, protocol stack development

1. INTRODUCTION

Today, flying is one of the safest ways to travel and the demand for air transportation is continuously growing. This of course increases the demand for air traffic management as the traffic itself increases. Efficient and effective management depends on reliable and efficient communications. The currently used system for air-ground communications nevertheless is based on Double-Sideband Amplitude Modulation (DS-AM), which has been introduced in the 1940's. Furthermore, the current system is designed for voice transmission only. With an ever growing demand for data exchange, e.g. for navigation and surveillance, it became obvious in the recent years that new protocols for aeronautical communication need to be developed. Two projects have been initiated, which have the goal to modernize air traffic management: Single European Sky Air Traffic Monitoring Research (SESAR) [6] in Europe and Next Generation National Airspace System (NextGen) [5] in the United States of America. Both of which are coordinated by the International Civil Aviation Organization (ICAO) [2]. Those projects both state the importance and requirement of modern communication systems. There are already two different candidates for protocols with the recommendation of ICAO for further development, namely L-band Digital Aeronautical Communications System 1 (LDACS1) [3] and LDACS2 [4], respectively. LDACS1 relies on Orthogonal Frequency-Division Multiplexing (OFDM) together with adaptive coding and modulation, whereas LDACS2 relies on a single-carrier system that is based on the well-known Global System for Mobile Communications (GSM). Both protocols have in common that, in contrast to DS-AM, both rely on a cellular network structure with ground base stations. From a protocol development point of view it is highly desirable to have network simulations available for development and testing of the different protocol layers. Especially as the safety and security in air traffic control are of major importance. In order to do so it is vital to have simulation scenarios that are close to reality but also efficient enough to allow for simulations of large networks. Large in this context means geographically but also in terms of number of communication nodes.

There have been approaches for modeling airborne networks in ns-3. Broyles et al. suggested a Gauß-Markov

WNS3, June 15 - 16, 2016, Seattle, WA, USA

ACM 978-1-4503-4216-2/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2915371.2915375

Model for the movement of the aircraft in [8] very similar to the BonnMotion mobility generation tool [7]. Newton et al. [11] introduced a model with real flight data and also implemented antenna directions, but for the purpose of optimized link state routing in airborne networks. Further simulations of airborne networks with focus on higher layer protocols are described in [14] and [9].

We decided that the existing wireless topology modules were not perfectly suitable and not sophisticated enough for the problem at hand. Furthermore we required the ability to process spherical coordinates from a special file format with interpolation in between waypoints as well as support for antenna characteristics and 3-dimensional antenna-pointing functionality. Therefore a new module has been developed that is custom tailored to aerospace topology and the communication needs accordingly.

The remainder of the paper is organized as follows. Section 2 will present the details and constraints of the topology situation. In Section 3 our framework and its integration with ns-3 and DCE will be discussed. First exemplary results will be presented in Section 4. The paper provides an overview about future work in Section 5 and is concluded with Section 6.

2. MODELING OF AIR TRAFFIC

As the movement of aircraft around the globe follows many rules and regulations it is obviously preferable to use actual flight data instead of mobility models that rely on probabilistic mechanisms. This point was also made in [11]. We currently use a file format in Extensible Markup Language (XML) to feed the topology information into our topology module. The description is designed to be compatible with the well known "All Purpose STructured Eurocontrol SuRveillance Information EXchange Category 62" (Asterix CAT 62) format [1]. It provides one data set every few seconds for every aircraft containing coordinates (waypoint), velocity and call sign together with a time stamp. In the XML description this data is completed by information about ground station locations and optionally antenna pointing azimuth and elevation for each communication node. Instead of XML description our framework is also able to accept real-time data in the Asterix CAT 62 format.

2.1 Position and Interpolation

The coordinates are provided in spherical format, i.e. latitude, longitude and altitude. The altitude is measured in meters above sea level. We assume the earth to be spherical with a radius of 6371 km. The latitude is measured between -90 degrees and $+90$ degrees, where the former denotes the south pole and the latter the north pole, respectively. The equator is at latitude 0 degrees. Longitude is measured between -180 degrees and $+180$ degrees, with 0 degrees being the Greenwich meridian. The spherical coordinates are transformed into Cartesian coordinates as that allows the reuse of ns-3 functions and simplifies some calculations within the topology module. The origin of the coordinate system is located at the center of the earth. The position in between waypoints of the aircraft can be acquired by different methods. The simplest variation is to use the last waypoint until a newer one is provided without interpolation in between. Alternatively linear interpolation can be carried out between the last and the future waypoint taking into account the current time and the time stamps of the waypoints. Furthermore it is possible to interpolate the position by assuming the airplane travels along a half versed sine (Haversine), i.e. a great circle between the two waypoints. This requires the altitude to be averaged between the waypoints in order to have both waypoints on a common great circle. Especially for a low time resolution of provided waypoints this enhances accuracy. All of the various modi of interpolation are provided by our module. As mentioned in the beginning of this Section it is also possible to feed the topology module with real-time data telegrams. Of course no interpolation is possible in that case and the static update of waypoints is thus employed.

2.2 Direction of Antenna

As mentioned in the beginning of this Section the Asterix data also optionally includes information about the antenna direction of a node (aircraft or base station) in the format of azimuth and elevation. Similar to the coordinates azimuth is measured between -180 degrees and $+180$ degrees, whereas elevation is measured between -90 and $+90$ degrees. The reference direction for mobile nodes is their direction of movement and the tangent to the longitude meridian in northern direction for fixed nodes. The reference direction of course corresponds to 0 degree azimuth and elevation. Initially all nodes have the reference direction of north until they move. The information about antenna direction together with the information about the directional radio pattern of the respective antennas enables us to obtain the specific gains of transmit and receive antenna via the transmit and receive directions, respectively. This information is then incorporated into the path loss.

3. FRAMEWORK DESCRIPTION

Our simulation framework is based on ns-3 together with DCE. Some components are similar to the ns-3 ones but differ slightly. Most different from an out-of-the-box ns-3 DCE is the integration of real-world Layer 2 code into DCE as shown in Figure 1. For a power line communication protocol, the advantages of this approach have already been described in [10]. We use a customized interface between pure ns-3 physical layer simulation models and Layer 2+ protocol stacks running in ns-3 DCE as shown in Figure 2. Because we want to perform detailed physical layer simulations and also handle different protocols, the physical layer models are split into a universal shared medium handler (USMH) and a specific physical layer handler (SPLH).

The USMH is comparable to the channel and models common influences, i.e. attenuation, delay, interference, etc. It is able to track all packets and obtain interference information in an efficient way. In order to obtain the physical information the USMH relies on the Topology module, c.f. Figure 2.

The communication nodes are connected to the USMH via the SPLH which also provides the interface to ns-3 DCE, i.e. the SPLH is responsible for the transmission (DCE $\rightarrow$ SPLH $\rightarrow$ USMH) and reception (UMSH $\rightarrow$ SPLH $\rightarrow$ DCE) of packets. Unlike a ns-3 NetDevice an SPLH does not model Layer 2 functionalities, which are in our case running within DCE. However, an SPLH is somehow similar to an ns-3 NetDevice in a way that it models protocol-specific properties of a communication protocol and enables us to have multiple SPLHs per communication node. As different protocols have different parameters for sending a packet and different per-

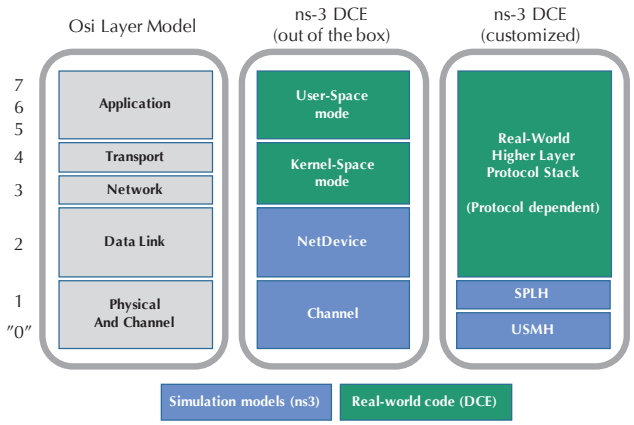


Figure 1: Layer overview of ns-3 DCE framework.

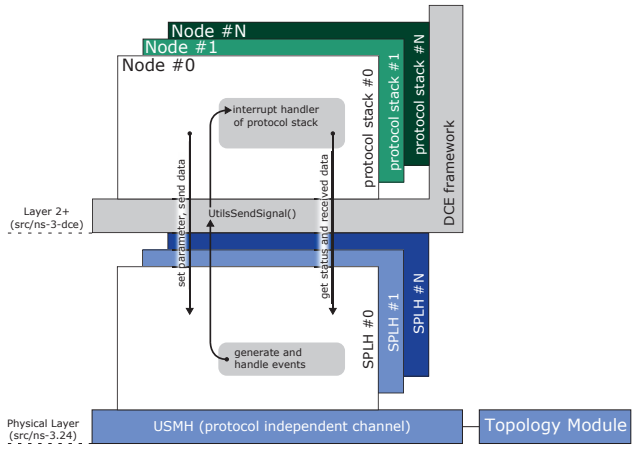


Figure 2: Structure of the (protocol agnostic) network simulator.

formance in the same physical conditions they each need an own SPLH implementation.

3.1 SPLH Packet Transmission and Reception

If an SPLH is instructed by its higher layers it calls the USMH interface and provides it with a packet struct that includes among data and protocol specific information all necessary parameters. For example carrier frequency, bandwidth and transmit power, which is modeled as a vector of values. The length of that vector decides how many subbands the transmit bandwidth is divided into and which transmit power is used. This creates a flexible way of including power spectral density of transmit packets. The USMH keeps track of all packets to determine interference between them and it obtains propagation parameters from the topology module. The interface is simple and contains only the current time, sending node and the packet struct. Based on this information the topology can provide the following propagation parameters

- channel model
- path loss
- delay
- relative speed

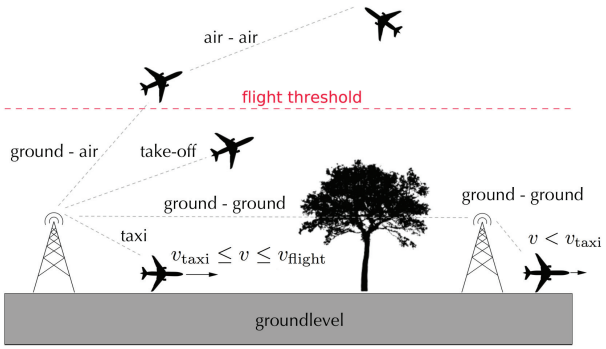


Figure 3: Examples for the various channel models.

- direction of reception

All those parameters are of course obtained for all SPLHs connected to the USMH and a receive event is scheduled for each SPLH after the appropriate time, i.e. when the packet duration and the respective propagation delay is over. Upon reception the SPLH can decide on the probability of correct reception of the packet based on the physical parameters that were originally obtained from the SPLH, interference packets that have been registered by the USMH and individual parameters like receiver noise etc. As the specific protocols are out of the scope of the paper, it is explained in the following how the topology module obtains the aforementioned parameters.

3.2 Channel Model

The packet error ratio is highly dependent on the channel between transmitter and receiver. For complexity reasons usually no physical layer simulation is desired in large scale network simulations as shown in [13]. Usually packet error curves are employed that have been obtained by simulation beforehand. In order to account for the channel conditions in various geometrical settings of transmitter and receiver we distinguish between five different channel models that are depicted in Figure 3. The channel model needs to be obtained first as it influences how other parameters are obtained, e.g. if line of sight (LOS) can be assumed for the calculation of delay and attenuation. The algorithm that decides for the channel model does not only take into account if both communication nodes are on the ground (ground-ground), airborne (air-air) or mixed (air ground), but also two special situations where one of the communication nodes is a ground station while the aircraft is on the taxi way (taxi) or in a take-off or landing phase (take-off). These channel models are among others standardized in the LDACS1 standard as they significantly influence the performance of the protocol. The SPLH design should take this into account and provide different packet error behavior for each channel model. Table 1 summarizes the parameters that lead to the different channel models. The flight threshold as well as v_{flight} and v_{taxi} can be configured by the user.

3.3 Path Loss and Delay

Depending on the channel model the path loss is calculated either as free space propagation (LOS) or free space propagation with a shadowing penalty (non-LOS) that can

Table 1: Channel Model Properties.

Channel Model	LOS	Speed	Altitude (threshold)
ground-air	yes	$v > v_{\text{flight}}$	above
air-air	yes	$v > v_{\text{flight}}$	above
ground-ground	no	$v < v_{\text{taxi}}$	below
taxi	yes	$v_{\text{taxi}} < v < v_{\text{flight}}$	below
take-off	yes	$v > v_{\text{flight}}$	below

be configured. As the attenuation is frequency dependent it is obtained separately for each subband of the power spectral density. The delay is obtained in a similar fashion. Speed of light is assumed as propagation velocity for the line of sight case and an additional penalty is assumed for the non line of sight case.

As the areal dimension of aircraft networks, and with that distances between aircraft, can be very large, one has to think about limitations of the free space propagation and its reach. In order to limit the propagation range we keep track of the radio horizon of the communication nodes. No ionospheric wave propagation can be assumed in the relevant communication carrier frequencies. Therefore, the radio horizon is calculated for every link and additional attenuation is added for propagation after the horizon in form of a larger propagation exponent. This means that the attenuation increases with the quadratic distance up to the radio horizon and with a free to choose exponent for the part of the distance that exceeds the horizon.

3.4 Relative Speed and Direction of Reception

As the Doppler effect can influence the performance of the communication the relative speed between two communication nodes is calculated and thus provided to the SPLH. The relative direction between sender and recipient of a transmission is calculated in order to obtain the transmit and receive antenna gains, respectively. Different antenna characteristics can be stored (in a table format), namely azimuth versus elevation with arbitrary resolution and thus provide gain values to look up for different directions of reception and transmission, respectively. If necessary, interpolation is carried out between the gain values. At the receiver the direction of reception is also stored for the receiver SPLH additionally to the reception gain. Thus, protocols that implement direction estimation can be modeled in an SPLH. Furthermore this is an interesting tracing information e.g. for cell handover procedures or optimization and verification.

3.5 Visualization

Especially with dynamic topologies it is not only helpful for tracing simulations but also for designing simulations that the topology can be visualized. Our model can be visualized by the standard NetAnim functionality that is provided by ns-3, but can also output Keyhole Markup Language (KML) files that can be read by e.g. Google Earth and visualized in 3D. Figure 4 shows a NetAnim XML plot with 30 aircraft that are depicted by blue dots and two ground stations that are depicted in red. At the time of the snapshot three nodes are sending a packet, which is indicated by the larger green bubbles. Although the plots are only in bird eye view in 2D the plots are very helpful for debugging and understanding the topology itself as well as interaction

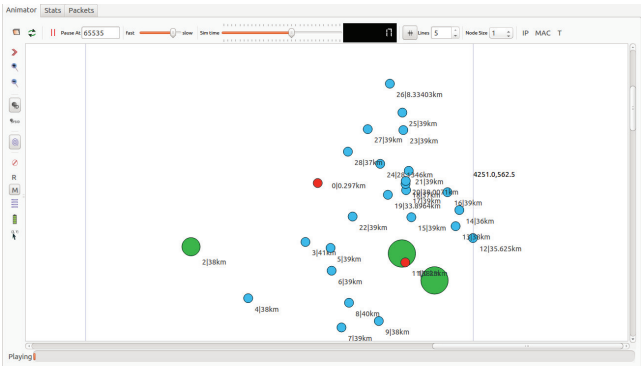


Figure 4: Snapshot of a topology scenario visualized in NetAnim.

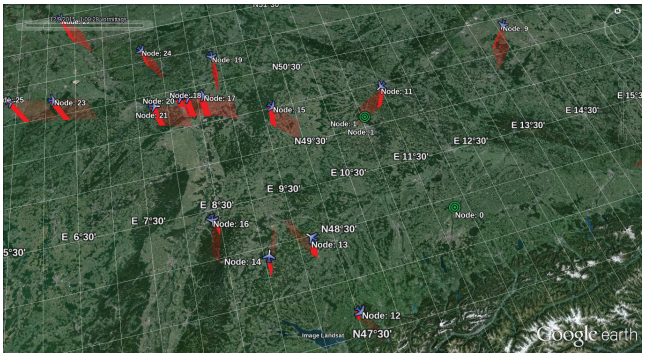


Figure 5: Snapshot of a topology scenario visualized in Google Earth.

between nodes. Also desired information can be depicted next to the node, e.g. node ID and altitude as in the snapshot in Figure 4. If 3D is required, one can switch to the KML files in e.g. Google earth. The same scenario is depicted in Figure 5. The additional geo information as well as the 3D plots provide contextual information on the geometric scenario. Figure 6 for example shows a flight profile of a single aircraft zoomed in. This view is very helpful to track the location history of an airplane. All introduced topology examples in this section have been produced by iAd GmbH with realistic aircraft data in the format according to Eurocontrol's Asterix standard described in Section 2 with additional information.

3.6 Implementation

Figure 7 depicts a block diagram of the topology module. As described in the beginning of this Section we are not using the NetDevice and Channel functionality of ns-3, but a setup that is tailored to our needs. Like the existing ns-3 mobility models, our flight mobility module can be installed on any ns-3 node. The `FlightMobilityModel` is the elemental class of the topology model and contains all attributes that concern a base station or aircraft. Location and antenna information retrieved by `XMLInput` or via the `RealtimeSocket` is processed here. The class `SphericVector` (VectorS) provides all conversion functionality from and to spherical and Cartesian coordinates as well as various geometric calculations. The waypoint updates are handled by the `FlightTrace` class and the `TopoParameters` class con-

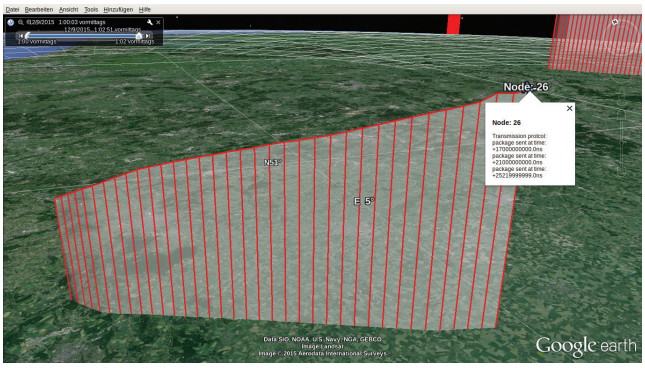


Figure 6: Zoomed snapshot of Figure 5 that shows the flight profile of one aircraft.

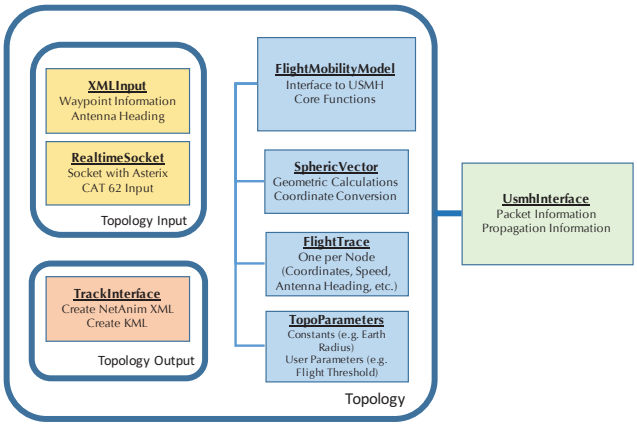


Figure 7: Block diagram of the topology module.

tains the variables for e.g. flight threshold, taxi speed, etc. as well as constant parameters like speed of light, earth radius etc. The class `TrackInterface` provides all the optional output functionality for visualization and the `UsmhInterface` symbolizes the interaction with the simulator, that requests propagation information from the topology and handles the USMH and all higher layers.

4. RESULTS

The visualization options described in Section 3.5 are already a valuable result of the flight mobility model. In the following, we describe two artifical scenarios which demonstrate the relevance of physical layer parameters. For both scenarios, the following settings are chosen:

- Air-air channel model
- Free space path loss
- 0 dBm transmit power for all ground stations and aircraft
- Carrier frequency 1 GHz
- Haversine distances and interpolation
- Ground station height level: 0 m above sea level
- Airplane height level: 10 km above sea level
- Airplane is always connected to only one airport/ground station, i.e. no multi-channel receivers

Both scenarios describe a flight from the south of Germany (Munich, IATA-Code MUC) to the north of Germany (Hamburg, IATA-Code HAM), which is a distance of 600 km.

The first scenario depicted in Figure 8 shows an inter-station handover. Besides the ground stations at the departure and destination airport, two additional base stations exist along the way, whereas ground station 1 (GS1) is located eastwards enroute and ground station 2 (GS2) is located westwards enroute. All airports and ground stations are equipped with an omnidirectional antenna in this scenario. Figure 9 shows the received signal power and the direction to the ground station that leads to the highest receive power relative to the plane movement. During the first 100 km the airport of departure, MUC, is the ground station with the highest receive power. The heading towards this airpot is 180 degree, i.e. straight behind the airplane. After a flight distance of 100 km, the receive power drops to −80 dBm and a handover to GS1 is performed. With the handover, the heading changes from 180 degree to approximately 75 degree as GS1 is north-east from the pilot's view. The heading increases monotonically until the next handover. Obviously, when the heading is +90 degree (at distance 125 km) the airplane is closest to BS1 and has maximum receive power. The next handover takes place at a distance of 275 km between BS1 and BS2, with a receive power of −83 dBm. The heading changes from a positive (north-east) to a negative (nort-west) value and decreases until the radio coverage area of the destination airport is reached. With the last handover at 450 km, the airplane heads straight towards the destination airport and thus the receive power is significantly rising again.

The second scenario depicted in Figure 10 describes an inter-cell handover. Different from the first scenario there is only one additional ground station (GS) enroute but with a three-sector antenna. We employed the 3 element Yagi-Uda antenna proposed in [15] simulated with the matlab toolbox for antenna simulations by Orfanidis [12]. Its directivity pattern is depicted in Figure 11. The antenna elements are oriented vertically and ϕ denotes the azimuth angle around the antenna, where $\phi = 0$ degree is the maximum gain direction. The maximum gain is 8.18 dBi and all gain values read from the pattern diagram are relative to that factor. Please note that this is just an example case to demonstrate the functionality of the simulator and that ground station as well as aircraft antennas employ other designs. The GS is placed west from the flight route, with an equal distance to both airports. The main radiation direction of the first sector is south-east (150 degree relative to north direction), the main radiation direction of the second sector is north-east (30 degree relative to north direction), and the third sector is of no interest for this scenario. As for the previous secenario the receive power as well as the heading towards the ground station are provided, c.f. Figure 12. The airplane flies again from Munich to Hamburg with the first handover from Munich airport to GS after 90 km (slightly earlier compared to the first scenario due to the chosen antenna gains) which is again a inter-station handover. The second handover at a flight distance of 290 km is then a inter-cell handover between GS's sector 1 and sector 2. Of course, the direction to the GS does not change during the inter-cell handover. The second scenario demonstrates that considering antenna

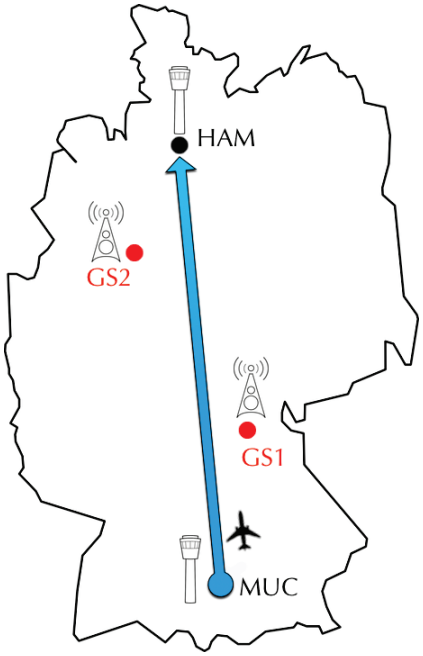


Figure 8: Examplary flight over Germany, with 1 aircraft 2 airports and 2 base stations.

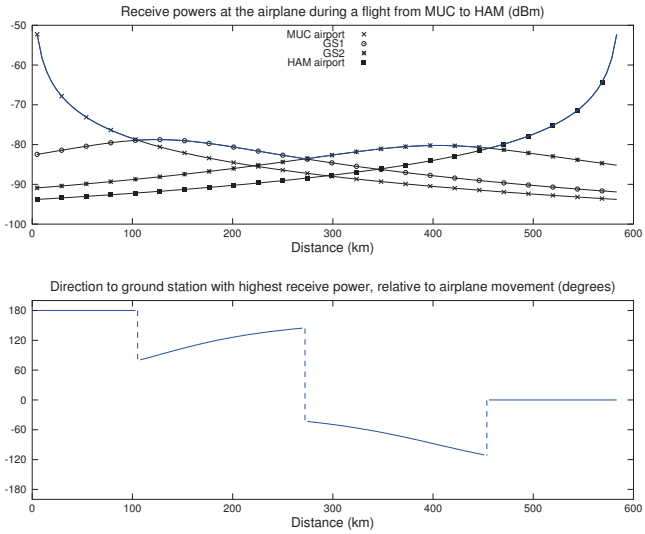


Figure 9: Received power at the aircraft from the different ground stations vs. travel distance (upper) and heading to best GS over distance (lower).

gains in a flight mobility model is very important.

In summary we are able to perform very large, yet very detailed simulation scenarios for airborne networks.

5. FUTURE WORK

For even more realistic topology simulation it would be interesting to have not only a free space propagation model to determine the attenuation, but also other propagation models. Especially for the non line of sight case attenuation models that consider shadowing effects would be desirable.

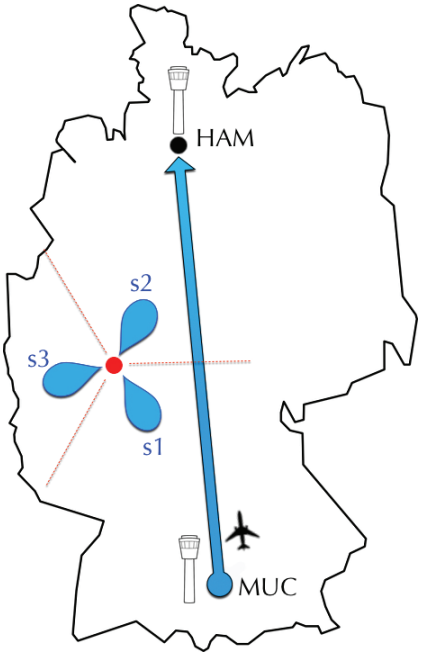


Figure 10: Examplary flight over Germany, with 1 aircraft 2 airports and a base station with sectorial antenna.

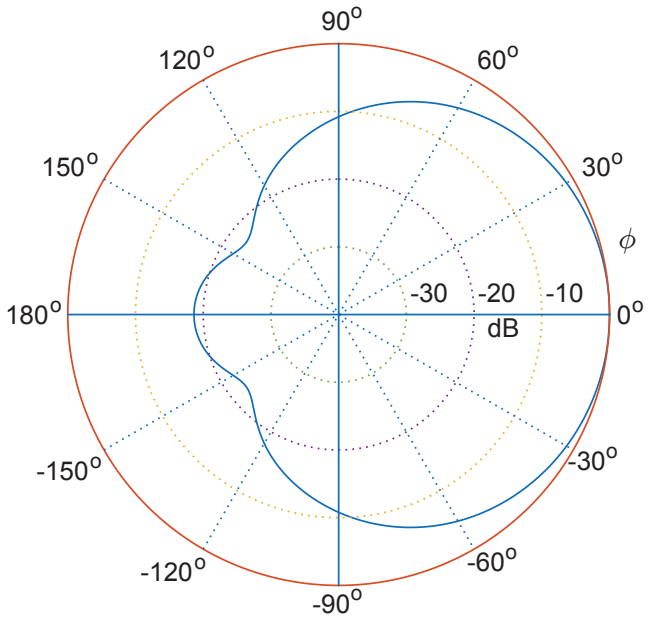


Figure 11: Antenna pattern of a 8,18 dB gain 3 element Yagi-Uda antenna.

Furthermore the influence of weather effects could be taken into account.

A large improvement in terms of reality would be modeling the surface (mountains, trees, etc.) and not only consider the earth to be a sphere. On the other hand this of course would also significantly increase the complexity. It is highly dependent on the application if the reality gain outweighs the drawback of added complexity and worse run-

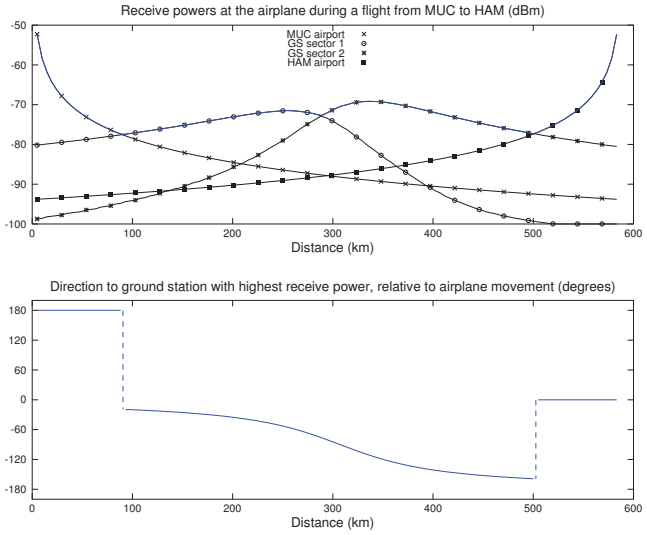


Figure 12: Received power at the aircraft from the ground stations an the sectors vs. travel distance (upper) and heading to best GS over distance (lower).

time performance. It might for example be less relevant for communication scenarios but highly beneficial for position estimation through multilateration.

6. CONCLUSIONS

We have presented a topology module for ns-3 and DCE that is customized for iAd GmbH's simulation framework, that is described in Section 3 and in [10], but that also applies to and can easily be fit to the standard framework as well. It accepts ICAO standard position information formats and is able to simulate realistic civil air traffic with it. The topology module together with the USMH is completely protocol indepent and therefore very versatile as only the SPLH and the higher layers have to be modified in order to fit to the respective simulation objective. Furthermore, realistic antenna patterns and directivity can be simulated and taken into account for the link budget. Direction estimation functionality as well as protocol behavior can be developed and tested. With the assistance of the DCE framework, the actual machine code of the higher layers of the protocol stack can be used for testing and development. We presented a 2D and a 3D visualization of the topology module and showed examples of how simulation traces could be obtained and also visualized. We believe that the framework is a significant help for testing and developing modern aircraft protocols for communications, navigation and surveillance.

7. REFERENCES

[1] Eurocontrol asterix format, http://www.eurocontrol.int/asterix.

[2] International civil aviation organization (icao), http://www.icao.int.

[3] Ldacs1 system definition proposal, eurocontrol, http://www.ldacs.com/wp-content/uploads/2014/02/ldacs1-specification-proposal-d2-deliverable.pdf.

[4] Ldacs2 system definition proposal, eurocontrol, https://www.eurocontrol.int/sites/default/files/article/content/documents/communications/11052009-ldcas2-d2-deliverable-v1.0.pdf.

[5] Next generation air transportation system (nextgen), http://www.faa.gov/nextgen/.

[6] Single european sky atm research (sesar), http://www.sesarju.eu.

[7] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn. Bonnmotion: a mobility scenario generation and analysis tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, page 51. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[8] D. Broyles, A. Jabbar, and J. P. Sterbenz. Design and analysis of a 3-d gauss-markov mobility model for highly dynamic airborne networks. In *Proceedings of the international telemetering conference (ITC), San Diego, CA*, 2010.

[9] E. K. Çetinkaya, J. P. Rohrer, A. Jabbar, M. J. Alenazi, D. Zhang, D. S. Broyles, K. S. Pathapati, H. Narra, K. Peters, S. A. Gogi, et al. Protocols for highly-dynamic airborne networks. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 411–414. ACM, 2012.

[10] J. Deutschmann, A. Lehmann, J. Hampel, and J. Huber. Network simulation for powerline protocols with direct code execution applied to dlc-3000 sfn. In *IEEE International Conference on Smart Grid Communications*, pages 464–468, Nov 2014.

[11] B. Newton, J. Aikat, and K. Jeffay. Simulating large-scale airborne networks with ns-3. In *Proceedings of the 2015 Workshop on ns-3*, pages 32–39. ACM, 2015.

[12] S. J. Orfanidis. *Electromagnetic waves and antennas*. Rutgers University New Brunswick, NJ, 2002.

[13] S. Papanastasiou, J. Mittag, E. G. Ström, and H. Hartenstein. Bridging the gap between physical layer emulation and network simulation. In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1–6. IEEE, 2010.

[14] K. S. Pathapati, T. A. N. Nguyen, J. P. Rohrer, and J. Sterbenz. Performance analysis of the aerotp transport protocol for highly-dynamic airborne telemetry networks. In *Proceedings of the International Telemetering Conference (ITC),(Las Vegas, NV)*, 2011.

[15] P. P. Viezbicke. Yagi antenna design. *Final Report National Bureau of Standards, Boulder, CO. Time and Frequency Div.*, 1, 1976.

Index of Authors

Index of Authors

Workshop on ns-3

WNS3 2016

www.nsnam.org/overview/wns3/wns3-2016

Seattle, Washington, USA
June 15-16, 2016

ISBN: 978-1-4503-4216-2

http://www.nsnam.org/overview/wns3