ns-3 Training

ns-3 training, June 2018

Outline

- Experience from several recent personal use cases with ns-3
  - LAA/Wi-Fi Coexistence
  - Smart Grid networks
  - AQM performance
  - Public-safety based LTE networks

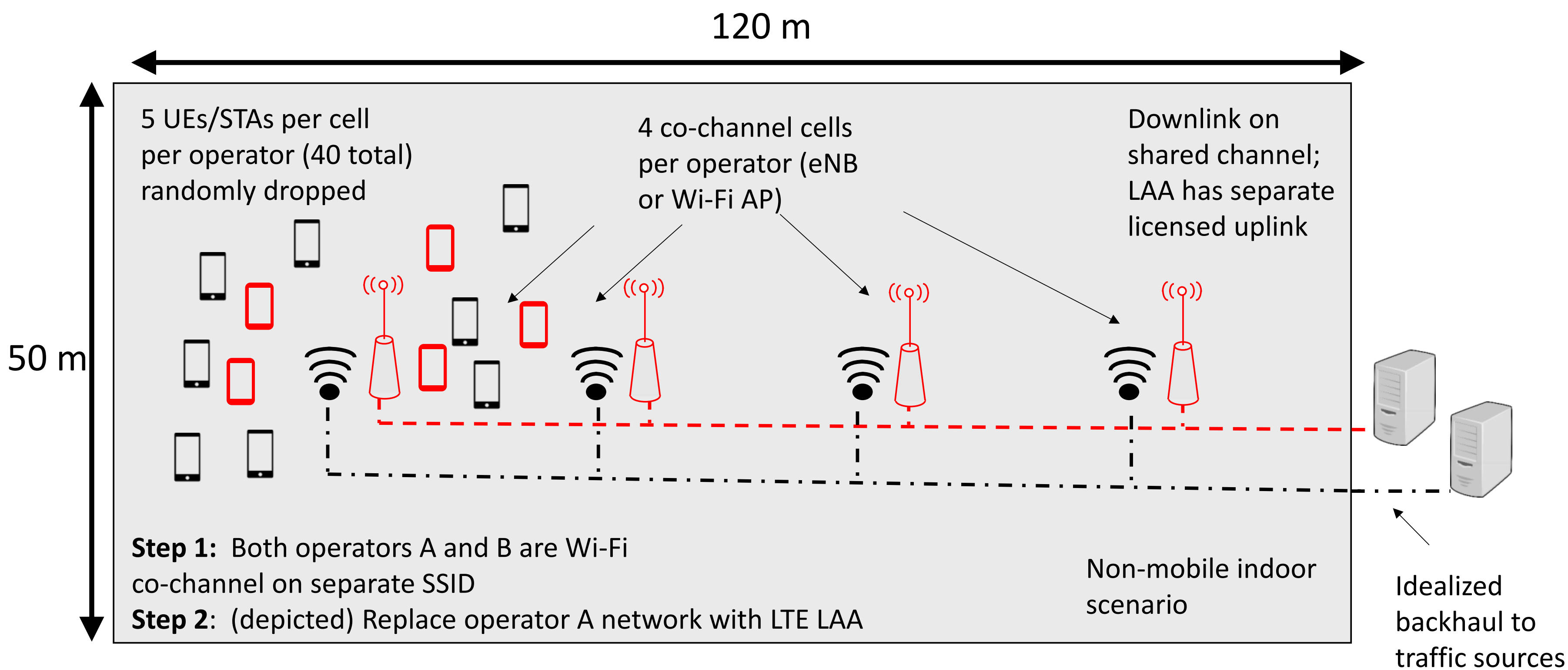
LTE/Wi-Fi Coexistence

case study

Use case: LAA Wi-Fi Coexistence

- ns-3 has been extended to support scenarios for LTE LAA/Wi-Fi Coexistence
- Methodology defined in 3GPP Technical Report TR36.889
- Enhancements needed:
  – Wireless models (LBT access manager, SpectrumWifiPhy, propagation/fading models)
  – Scenario support (traffic models)
  – Output data processing

Indoor 3GPP scenario

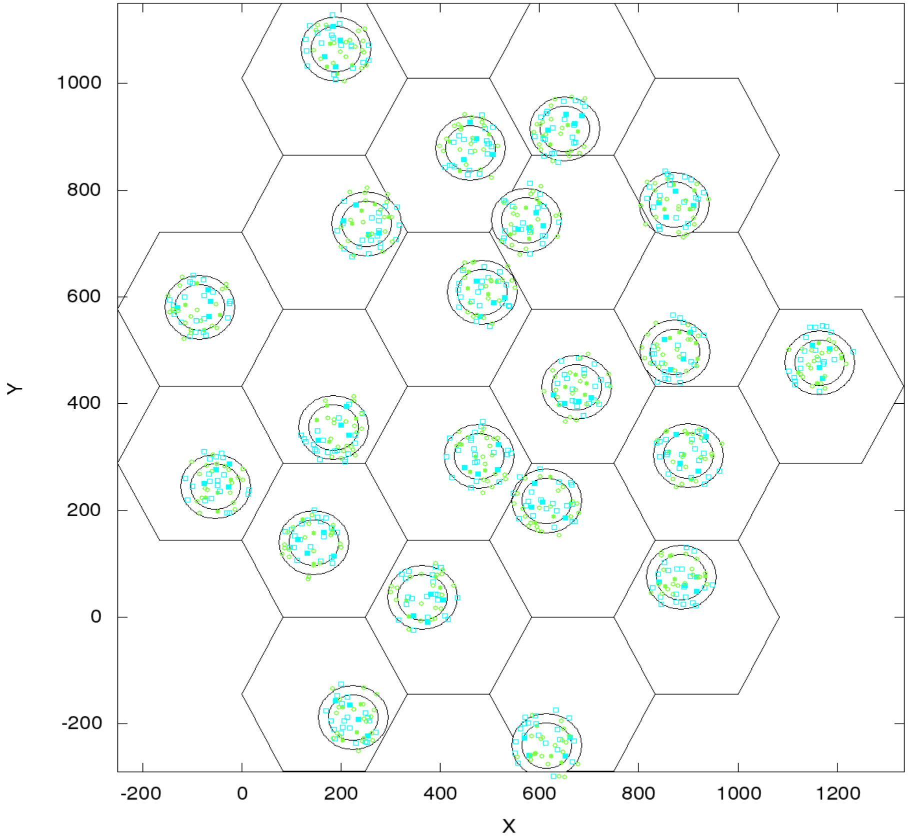


Indoor scenario details

Unlicensed channel model	3GPP TR 36.889	ns-3 implementation
Network Layout	Indoor scenario	Indoor scenario
System bandwidth	20 MHz	20 MHz
Carrier frequency	5 GHz	5 GHz (channel 36, tunable)
Number of carriers	1, 4 (to be shared between two operators) 1 for evaluations with DL+UL Wi-Fi coexisting with DL-only LAA	1 for evaluations with DL+UL Wi-Fi coexisting with DL-only LAA
Total Base Station (BS) transmission power	18/24 dBm	18/24 dBm Simulations herein consider 18 dBm
Total User equipment (UE) transmission power	18 dBm for unlicensed spectrum	18 dBm
Distance dependent path loss, shadowing and fading	ITU InH	802.11ax indoor model
Antenna pattern	2D Omni-directional	2D Omni-directional
Antenna height	6 m	6 m (LAA, not modelled for Wi-Fi)
UE antenna height	1.5 m	1.5 m (LAA, not modelled for Wi-Fi)
Antenna gain	5 dBi	5 dBi
UE antenna gain	0 dBi	0 dBi
Number of UEs	10 UEs per unlicensed band carrier per operator for DL-only 10 UEs per unlicensed band carrier per operator for DL-only for four unlicensed carriers. 20 UEs per unlicensed band carrier per operator for DL+UL for single unlicensed carrier. 20 UEs per unlicensed band carrier per operator for DL+UL Wi-Fi coexisting with DL-only LAA	Supports all the configurations in TR 36.889. Simulations herein consider the case of 20 UEs per unlicensed band carrier per operator for DL LAA coexistence evaluations for single unlicensed carrier.
UE Dropping	All UEs should be randomly dropped and be within coverage of the small cell in the unlicensed band.	Randomly dropped and within small cell coverage.
Traffic Model	FTP Model 1 and 3 based on TR 36.814 FTP model file size: 0.5 Mbytes. Optional: VoIP model based on TR36.889	FTP Model 1 as in TR36.814. FTP model file size: 0.5 Mbytes Voice model: DL only
UE noise figure	9 dB	9 dB
Cell selection	For LAA UEs, cell selection is based on RSRP (Reference Signal Received Power. For Wi-Fi stations (STAs), cell selection is based on RSS (Received signal power strength) of WiFi Access Points (APs). RSS threshold is -82 dBm.	RSRP for LAA UEs and RSS for Wi-Fi STAs
Network synchronization	For the same operator, the network can be synchronized. Small cells of different operators are not synchronized.	Small cells are synchronized, different operators are not synchronized.

Outdoor 3GPP scenario

Outdoor layout: hexagonal macrocell layout. 7 macro sites and 3 cells per site. 1 Cluster per cell. 4 small cells per operator per cluster, uniformly dropped. ITU UMi channel model.



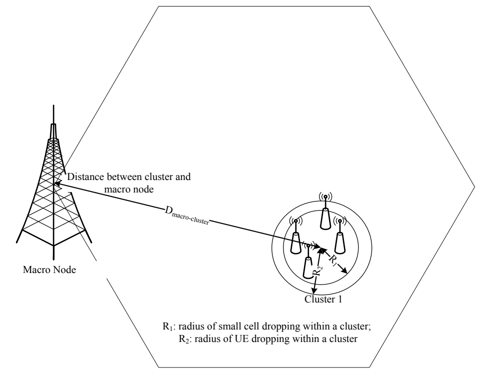
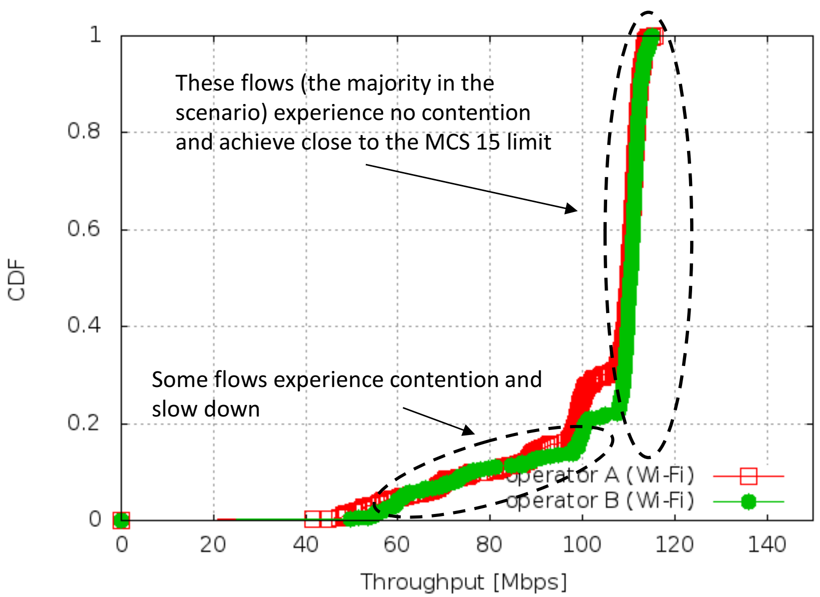


Figure source: 3GPP TR 36.889 V13.0.0 (2015-05)

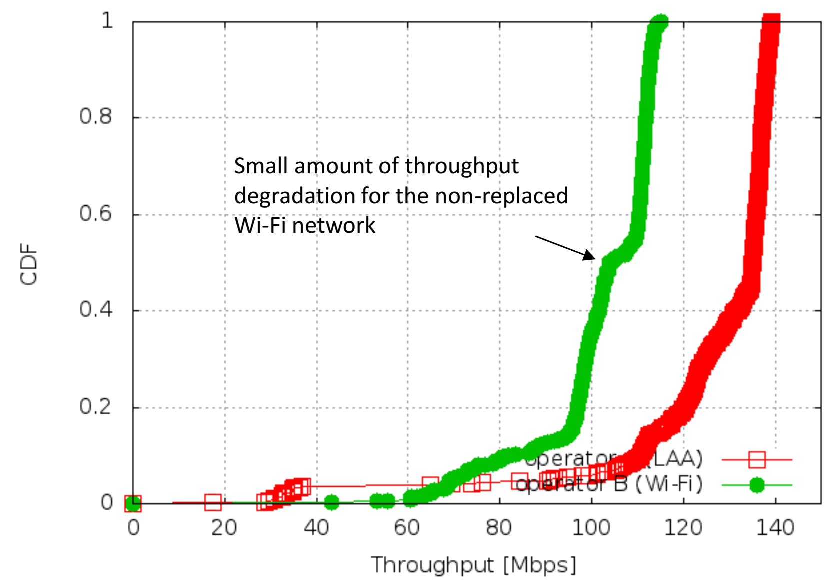
References

- ns-3 Wiki page:
  - https://www.nsnam.org/wiki/LAA-WiFi-Coexistence
    - module documentation
    - references to various publications
    - documentation on reproducing results
- Code:
  - git clone https://bitbucket.org/ns3lteu/ns-3-dev-lbt

Sample results



a) Step 1 (Wi-Fi)

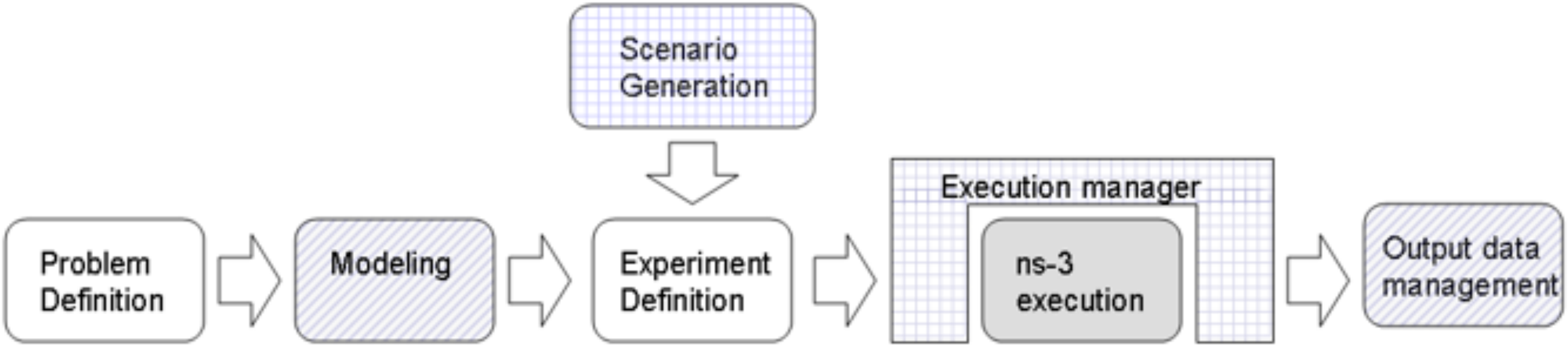


b) Step 2 (LAA)

ns-3 NETWORK SIMULATOR

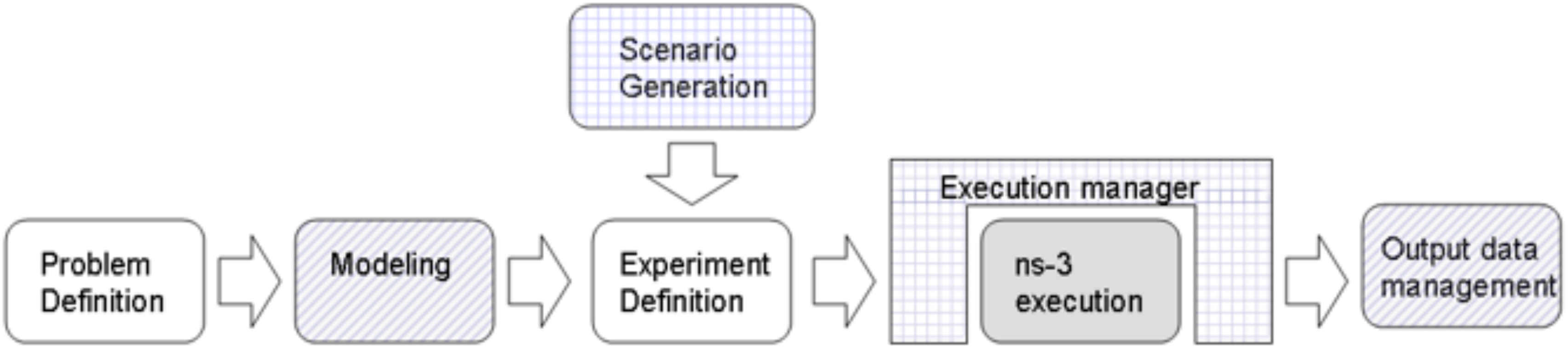
Lessons learned

- Think about the data and results you want, and work backwards from there
- As much, or more time, spent on scenario development than on model development
- Choose the right levels of abstraction in the scenario
- Several rewrites may be needed as project evolves



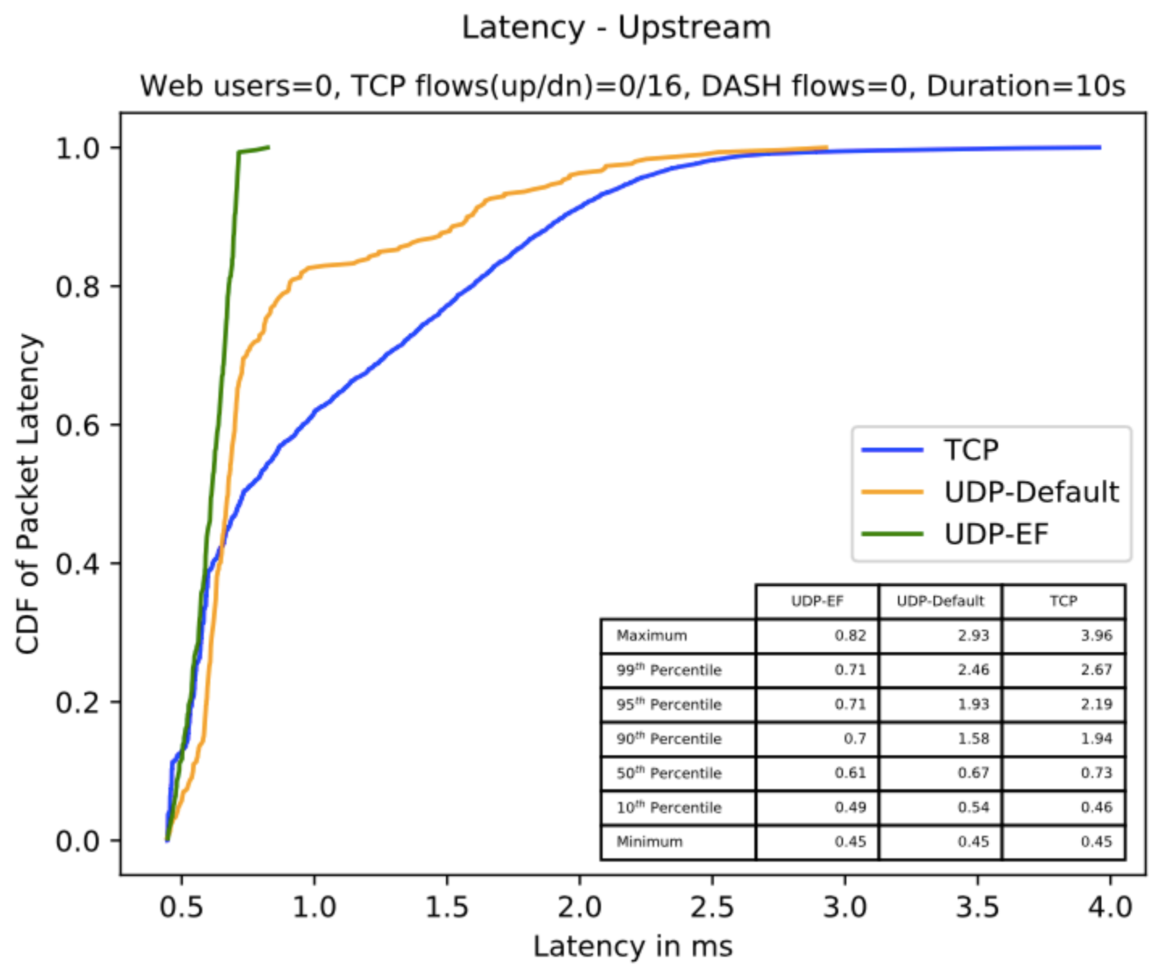
Key concepts

- Configuration and execution management
- Tracing (gathering output data)
- Design for future sharing of your code
  - Good APIs (attributes, trace sources), tests, and documentation



Example

- Pass UDP data (marked as EF and best effort) and TCP data through a prioritized queue
- Plot CDF of data including percentiles
- Repeat for different traffic configurations, different queue configurations



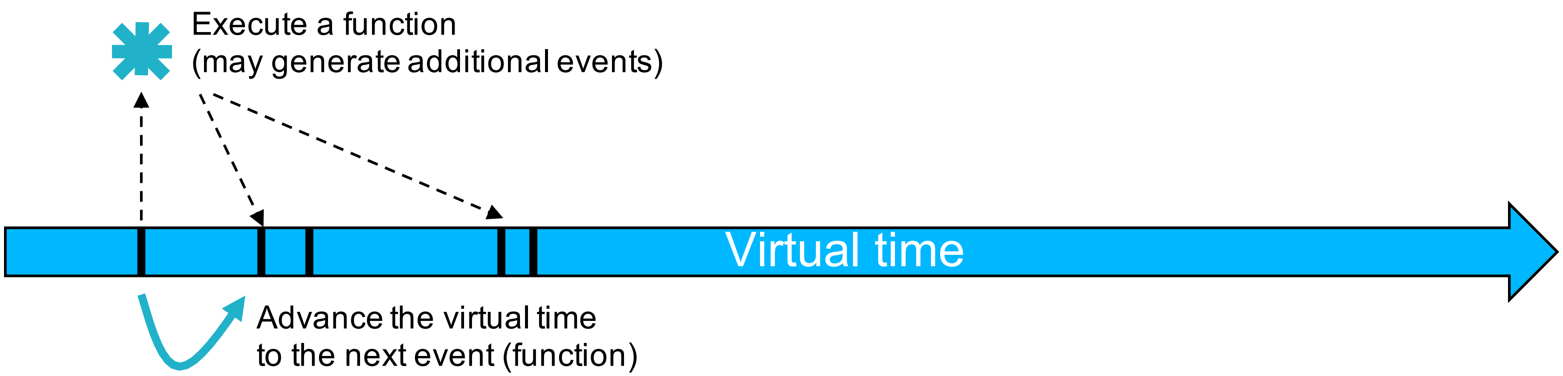
	UDP-EF	UDP-Default	TCP
Maximum	0.82	2.93	3.96
99th Percentile	0.71	2.46	2.67
95th Percentile	0.71	1.93	2.19
90th Percentile	0.7	1.58	1.94
50th Percentile	0.61	0.67	0.73
10th Percentile	0.49	0.54	0.46
Minimum	0.45	0.45	0.45

Design aspects

- What plotting framework am I using?
  - Matplotlib can meet requirements
- How to manage multiple replications
  - Bash script, expose configuration as command line arguments
- How to obtain the output data
  - Hook SojournTime trace source of the queue?
  - Or do I want to also capture device latency?
- Other design questions
  - What flavor of TCP?
  - How long should simulation trials run for?
  - etc.

Simulator core

- Simulation time
- Events
- Simulator and Scheduler
- Command line arguments
- Random variables



Simulator example

```
#include <iostream>
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include "ns3/command-line.h"
#include "ns3/double.h"
#include "ns3/random-variable-stream.h"

using namespace ns3;
```

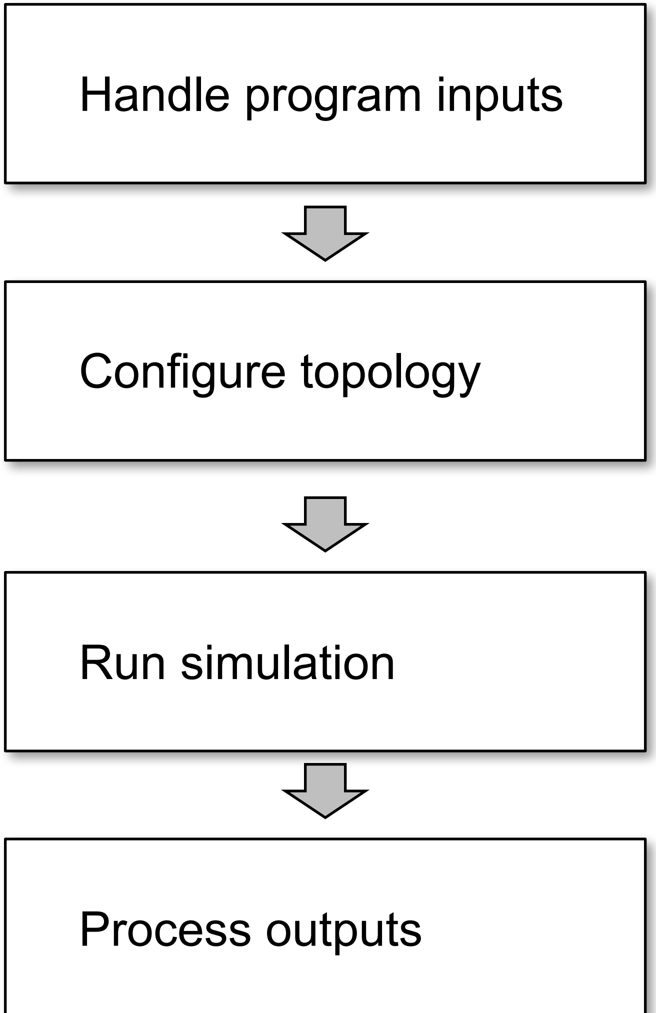
```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  MyModel model;
  Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
  v->SetAttribute ("Min", DoubleValue (10));
  v->SetAttribute ("Max", DoubleValue (20));

  Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);

  Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);

  EventId id = Simulator::Schedule (Seconds (30.0), &CancelledEvent);
  Simulator::Cancel (id);

  Simulator::Run ();

  Simulator::Destroy ();
}
```

Simulator example (in Python)

```
# Python version of sample-simulator.cc

import ns.core
```

```
def main(dummy_argv):

    model = MyModel()
    v = ns.core.UniformRandomVariable()
    v.SetAttribute("Min", ns.core.DoubleValue (10))
    v.SetAttribute("Max", ns.core.DoubleValue (20))

    ns.core.Simulator.Schedule(ns.core.Seconds(10.0), ExampleFunction, model)

    ns.core.Simulator.Schedule(ns.core.Seconds(v.GetValue()), RandomFunction, model)

    id = ns.core.Simulator.Schedule(ns.core.Seconds(30.0), CancelledEvent)
    ns.core.Simulator.Cancel(id)

    ns.core.Simulator.Run()

    ns.core.Simulator.Destroy()

if __name__ == '__main__':
    import sys
    main(sys.argv)
```

Simulation program flow



Command-line arguments

- Add CommandLine to your program if you want command-line argument parsing

```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);
```

- Passing --PrintHelp to programs will display command line options, if CommandLine is enabled

```
./waf --run "sample-simulator --PrintHelp"
```

```
--PrintHelp: Print this help message.
--PrintGroups: Print the list of groups.
--PrintTypeIds: Print all TypeIds.
--PrintGroup=[group]: Print all TypeIds of group.
--PrintAttributes=[typeid]: Print all attributes of typeid.
--PrintGlobals: Print the list of globals.
```

Time in ns-3

- Time is stored as a large integer in ns-3
  – Minimize floating point discrepancies across platforms
- Special Time classes are provided to manipulate time (such as standard operators)
- Default time resolution is nanoseconds, but can be set to other resolutions
  – Note: Changing resolution is not well used/tested
- Time objects can be set by floating-point values and can export floating-point values

```
double timeDouble = t.GetSeconds();
```

  – Best practice is to avoid floating point conversions where possible and use Time arithmetic operators

Events in ns-3

- Events are just function calls that execute at a simulated time
  - i.e. callbacks
  - this is another difference compared to other simulators, which often use special "event handlers" in each model
- Events have IDs to allow them to be cancelled or to test their status

Simulator and Schedulers

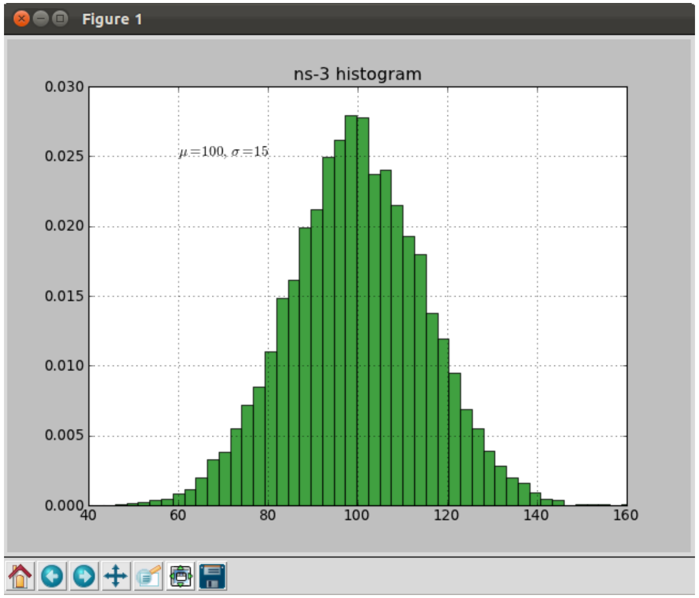
- The Simulator class holds a scheduler, and provides the API to schedule events, start, stop, and cleanup memory
- Several scheduler data structures (calendar, heap, list, map) are possible
- "RealTime" simulation implementation aligns the simulation time to wall-clock time
  - two policies (hard and soft limit) available when the simulation and real time diverge

Random Variables

from src/core/examples/sample-rng-plot.py

- Currently implemented distributions
  - Uniform: values uniformly distributed in an interval
  - Constant: value is always the same (not really random)
  - Sequential: return a sequential list of predefined values
  - Exponential: exponential distribution (poisson process)
  - Normal (gaussian), Log-Normal, Pareto, Weibull, triangular

```
# Demonstrate use of ns-3 as a random number generator integrated with
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial

import numpy as np
import matplotlib.pyplot as plt
import ns.core

# mu, var = 100, 225
rng = ns.core.NormalVariable(100.0, 225.0)
x = [rng.GetValue() for t in range(10000)]

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.title('ns-3 histogram')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



Random variables and independent replications

- Many simulation uses involve running a number of *independent replications* of the same scenario

- In ns-3, this is typically performed by incrementing the simulation *run number*
  - *not by changing seeds*

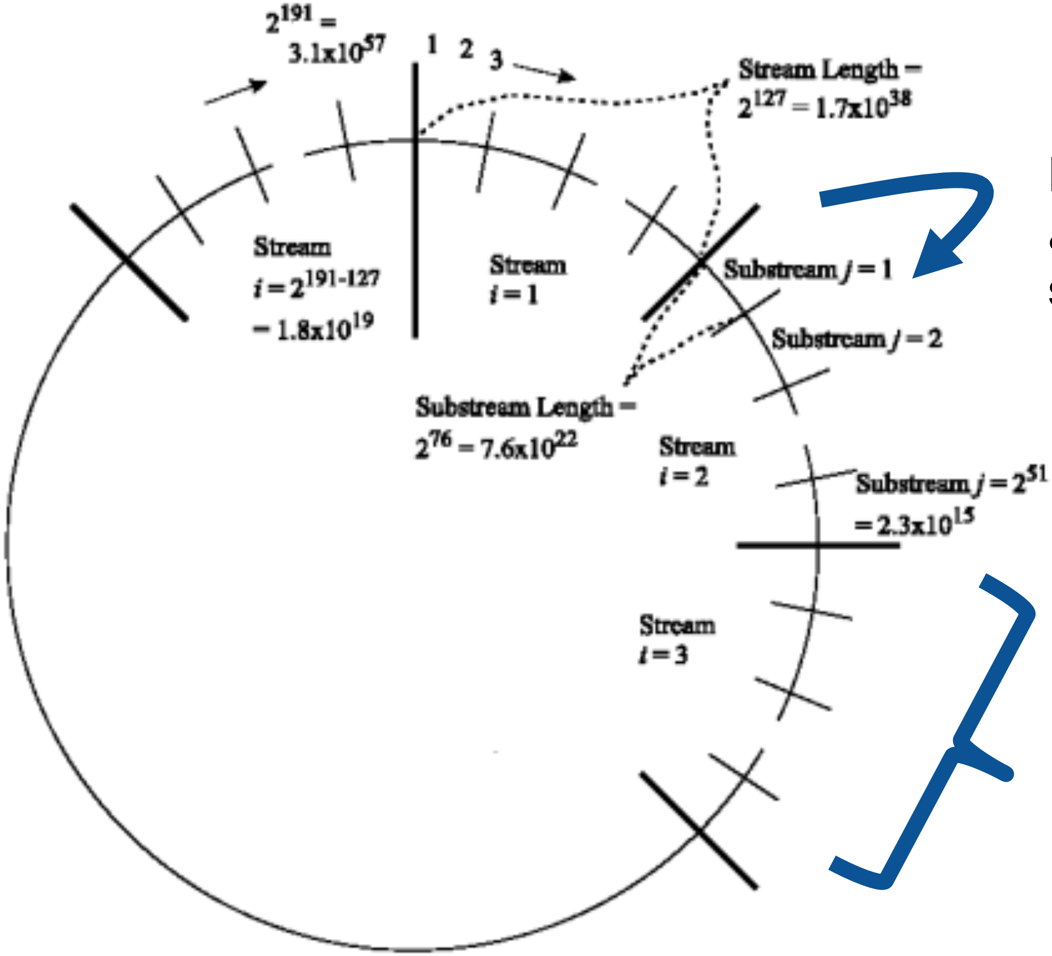
ns-3 random number generator

- Uses the MRG32k3a generator from Pierre L'Ecuyer
  - http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf
  - Period of PRNG is 3.1x10^57
- Partitions a pseudo-random number generator into uncorrelated *streams* and *substreams*
  - Each RandomVariableStream gets its own stream
  - This stream partitioned into substreams

Key Terminology

- **Seed:** A set of values that generates an entirely new PRNG sequence
- **Stream:** The PRNG sequence is divided into non-overlapping intervals called streams
- **Run Number (substream):** Each stream is further divided to substreams, indexed by a variable called the run number.

Streams and Substreams



Incrementing the Run Number will move ***all*** streams to a new substream

Each ns-3 RandomVariableStream object is assigned to a stream (by default, randomly)

Figure source: Pierre L'Ecuyer, Richard Simard, E. Jack Chen, and W. David Kelton. An object-oriented random number package with many long streams and substreams. Operations Research, 2001.

Run number vs. seed

- If you increment the seed of the PRNG, the streams of random variable objects across different runs are not guaranteed to be uncorrelated
- If you fix the seed, but increment the run number, you will get uncorrelated streams

Setting the stream number

- The ns-3 implementation provides access to 2^64 streams
- 2^63 are placed in a pool for automatic assignment, and 2^63 are reserved for fixed assignment

```
<----------------------------------------------------------------------------->
^                                    ^^                                      ^
|                                    ||                                      |
stream 0                 stream (2^63 - 1)  stream 2^63          stream (2^64 - 1)
<- automatically assigned ----------><- assigned by user ----------------->
```

- Users may optionally assign a stream number index to a random variable using the `SetStream ()` method.
  - This allows better control over selected random variables
  - Many helpers have AssignStreams () methods to do this across many such random variables

Putting it together

- Example of scheduled event

```
static void
RandomFunction (void)
{
  std::cout << "RandomFunction received event at "
            << Simulator::Now ().GetSeconds () << "s" << std::endl;
}
```

```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  MyModel model;
  Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
  v->SetAttribute ("Min", DoubleValue (10));
  v->SetAttribute ("Max", DoubleValue (20));

  Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);

  Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);
```

Demo real-time, command-line, random variables...

ns-3 Training: Packets

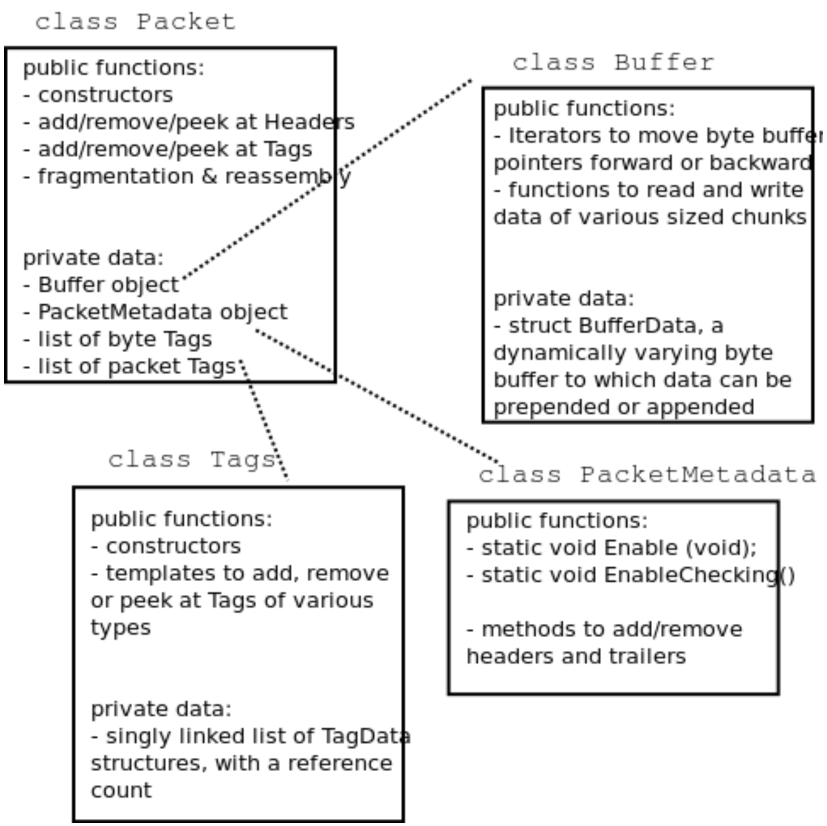
ns-3 Annual meeting June 2018

ns-3 Packet

- Packet is an advanced data structure with the following capabilities
  - Supports fragmentation and reassembly
  - Supports real or virtual application data
  - Extensible
  - Serializable (for emulation)
  - Supports pretty-printing
  - Efficient (copy-on-write semantics)

ns-3 Packet structure

- Analogous to an mbuf/skbuff



Copy-on-write

- Copy data bytes only as needed

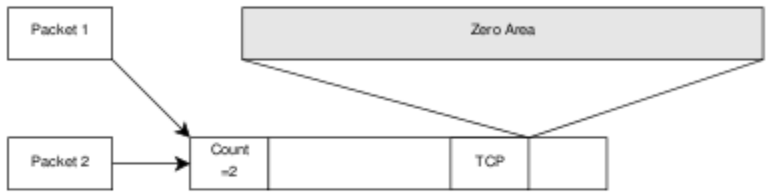


Figure 3.8: The TCP and the IP stacks hold references to a shared buffer.

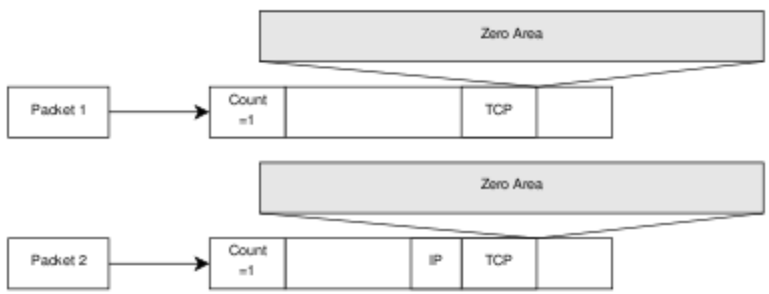


Figure 3.9: The IP stack inserts the IP header, triggers an un-share operation, completes the insertion.

Figure source: Mathieu Lacage's Ph.D. thesis

Headers and trailers

- Most operations on packet involve adding and removing an ns3::Header
- class ns3::Header must implement four methods:

```
Serialize()
Deserialize()
GetSerializedSize()
Print()
```

Headers and trailers (cont.)

- Headers are serialized into the packet byte buffer with Packet::AddHeader() and removed with Packet::RemoveHeader()
- Headers can also be 'Peeked' without removal

```
Ptr<Packet> pkt = Create<Packet> ();
UdpHeader hdr; // Note:  not heap allocated
pkt->AddHeader (hdr);
Ipv4Header iphdr;
pkt->AddHeader (iphdr);
```

Packet tags

- Packet tag objects allow packets to carry around simulator-specific metadata
  - Such as a "unique ID" for packets or cross-layer info
- Tags may associate with byte ranges of data, or with the whole packet
  - Distinction is important when packets are fragmented and reassembled
- Tags presently are not preserved across serialization boundaries (e.g. MPI)

PacketTag vs. ByteTag

- Two tag types are available: PacketTag and ByteTag
  - ByteTags run with bytes
  - PacketTags run with packets
- When Packet is fragmented, both copies of Packet get copies of PacketTags
- When two Packets are merged, only the PacketTags of the first are preserved
- PacketTags may be removed individually; ByteTags may be removed all at once

Tag example

- Here is a simple example illustrating the use of tags from the code in src/internet/model/udp-socket-impl.cc:

```
Ptr<Packet> p;  // pointer to a pre-existing packet
SocketIpTtlTag tag
tag.SetTtl (m_ipMulticastTtl); // Convey the TTL from
UDP layer to IP layer
p->AddPacketTag (tag);
```

- This tag is read at the IP layer, then stripped (src/internet/model/ipv4-l3-protocol.cc):

```
uint8_t ttl = m_defaultTtl;
SocketIpTtlTag tag;
bool found = packet->RemovePacketTag (tag);
if (found)
  {
    ttl = tag.GetTtl ();
  }
```

Packet metadata

- Packets may optionally carry metadata
  - record every operation on a packet's buffer
  - implementation of Packet::Print for pretty-printing of the packet
  - sanity check that when a Header is removed, the Header was actually present to begin with
- Not enabled by default, for performance reasons
- To enable, insert one or both statements:

```
Packet::EnablePrinting ();
Packet::EnableChecking ();
```

Ptr<Packet>

- Packets are reference counted objects that support the smart pointer class `Ptr`
- Use a templated "Create" method instead of CreateObject for ns3::Objects
- Typical creation:
  - `Ptr<Packet> pkt = Create<Packet> ();`
- In model code, Packet pointers may be const or non-const; often Packet::Copy() is used to obtain non-const from const
  - `Ptr<const Packet> cpkt = ...;`
  - `Ptr<Packet> p = cpkt->Copy ();`

ns-3 Training

Program Structure

Example program walkthrough

We will use the program at:

- `examples/traffic-control/queue-discs-benchmark.cc`

```
$ ./waf --run 'queue-discs-benchmark'
```

Structure of an ns-3 program

```
int main (int argc, char *argv[])
{

  // Set default attribute values

  // Parse command-line arguments

  // Configure the topology; nodes, channels, devices, mobility

  // Add (Internet) stack to nodes

  // Configure IP addressing and routing

  // Add and configure applications

  // Configure tracing

  // Run simulation

  // Handle any post-simulation data processing
}
```

Helper API

- The ns-3 “helper API” provides a set of classes and methods that make common operations easier than using the low-level API
- Consists of:
  - container objects
  - helper classes
- The helper API is implemented using the low-level API
- Users are encouraged to contribute or propose improvements to the ns-3 helper API

Containers

- Containers are part of the ns-3 "helper API"
- Containers group similar objects, for convenience
  - They are often implemented using C++ std containers
- Container objects also are intended to provide more basic (typical) API

The Helper API (vs. low-level API)

- Is not generic
- Does not try to allow code reuse
- Provides simple 'syntactical sugar' to make simulation scripts look nicer and easier to read for network researchers
- Each function applies a single operation on a "set of same objects”
- A typical operation is "Install()"

Helper Objects

- NodeContainer: vector of Ptr<Node>
- NetDeviceContainer: vector of Ptr<NetDevice>
- InternetStackHelper
- WifiHelper
- MobilityHelper
- OlsrHelper
- ... Each model provides a helper class

Installation onto containers

- Installing models into containers, and handling containers, is a key API theme

```
NodeContainer c;
c.Create (numNodes);
...
mobility.Install (c);
...
internet.Install (c);
...
```

Native IP models

- IPv4 stack with ARP, ICMP, UDP, and TCP
- IPv6 with ND, ICMPv6, IPv6 extension headers, TCP, UDP
- IPv4 routing: RIPv2, static, global, NixVector, OLSR, AODV, DSR, DSDV
- IPv6 routing: RIPng, static

IP address configuration

- An Ipv4 (or Ipv6) address helper can assign addresses to devices in a NetDevice container

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
csmaInterfaces = ipv4.Assign (csmaDevices);

...

ipv4.NewNetwork ();  // bumps network to 10.1.2.0
otherCsmaInterfaces = ipv4.Assign (otherCsmaDevices);
```

Applications and sockets

- In general, applications in ns-3 derive from the ns3::Application base class
  - A list of applications is stored in the ns3::Node
  - Applications are like processes
- Applications make use of a sockets-like API
  - Application::Start () may call ns3::Socket::SendMsg() at a lower layer

Sockets API

Plain C sockets

```
int sk;
sk = socket(PF_INET, SOCK_DGRAM, 0);
```

```
struct sockaddr_in src;
inet_pton(AF_INET,"0.0.0.0",&src.sin_ad
    dr);
src.sin_port = htons(80);
bind(sk, (struct sockaddr *) &src,
    sizeof(src));
```

```
struct sockaddr_in dest;
inet_pton(AF_INET,"10.0.0.1",&dest.sin_
    addr);
dest.sin_port = htons(80);
sendto(sk, "hello", 6, 0, (struct
    sockaddr *) &dest, sizeof(dest));
```

```
char buf[6];
recv(sk, buf, 6, 0);
}
```

ns-3 sockets

```
Ptr<Socket> sk =
udpFactory->CreateSocket ();
```

```
sk->Bind (InetSocketAddress (80));
```

```
sk->SendTo (InetSocketAddress (Ipv4Address
    ("10.0.0.1"), 80), Create<Packet>
    ("hello", 6));
```

```
sk->SetReceiveCallback (MakeCallback
    (MySocketReceive));
•   […] (Simulator::Run ())

void MySocketReceive (Ptr<Socket> sk,
    Ptr<Packet> packet)
{
...
}
```

NetDevice trace hooks

- Example: CsmaNetDevice