IEEE 802.11s Mesh Networking NS-3 Model

Kirill Andreev
Institute for Information Transmission Problems
Bolshoy Karetny per. 19,
Moscow, 127994, Russia
andreev@iitp.ru

Pavel Boyko
Institute for Information Transmission Problems
Bolshoy Karetny per. 19,
Moscow, 127994, Russia
boyko@iitp.ru

ABSTRACT
We present an IEEE 802.11s wireless mesh networking model implemented in the open source Network Simulator 3 environment. Design solutions, supported and unsupported features of draft standard, optional protocols extensions and heuristics are summarized. Simulation of two typical mesh networking scenarios illustrates model behavior and simulation methodology. We argue that our NS-3 802.11s model provides a general framework for WiFi mesh networking simulation and future research.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

Keywords
IEEE 802.11s, mesh networking, NS-3

1. INTRODUCTION
Wireless mesh networking is a relatively new technology originating out of ad-hoc networking research of the last decade. There are many research solutions in mesh networking, none of them being accepted as universal. This situation is supposed to be changed by an adoption of an IEEE 802.1s draft standard [1], which is one of the most actively developed solutions in the field.

It is needless to say, that wireless networking research is fundamentally depends on simulations and mesh networking is not an exception. In this paper we present an IEEE 802.11s wireless mesh networking model implemented in the open source Network Simulator 3 environment [2]. The contribution of this paper is threefold: first, the freely available 802.11s model is developed, we are the first in doing so. Second, we present a common protocol-independent multi-radio mesh station architecture which can be used to develop a model of any other link layer meshing solution. Third, the behaviour of base 802.11s mechanisms is illustrated in two simple but significant mesh usage scenarios.

The rest of the paper is organized as follows. In the section 2 we overview IEEE 802.11s draft standard version 3.0. All core protocols and rules are explained as well as a number of advanced features. Section 3 presents the design and implementation of the NS-3 common mesh architecture and implementation details and features of IEEE 802.11s model based on this architecture. This is the central section of this paper. Section 4 show model behaviour in two simple mesh networking use cases. In the section 5 we overview the directions of future work and the paper ends with the conclusion at section 6.

2. BACKGROUND

2.1 IEEE 802.11 standard
IEEE 802.11 Standard [3] specifies the physical, MAC and link layer operations for wireless LANs. At the MAC layer IEEE 802.11 uses both carrier sensing and virtual carrier sensing and random backoff procedure prior to sending data to avoid collisions. Virtual carrier sensing is accomplished through use of Ready-to-Send (RTS) and Clear-to-Send (CTS) frame exchange, NAV (network allocation vectors). The NAV is used to perform virtual channel sensing by indicating that the channel is busy. This is known as CSMA/CA medium access mechanism. After a destinations properly receives a data frame, it sends an acknowledgment (ACK) to the source. This signifies that the packet was successfully received. If ACK is not received by a source, frame transmission shall be retried until short (or long retry counter when no answer for RTS was received) retry counter will exceed a threshold. All broadcast frames are sent without CTS/RTS and ACK procedures.

IEEE 802.11 standard defines two types of WLAN deployment. The first one (BSS mode) is Access Point (AP) and a number of associated stations (STA). All traffic goes from AP to station and back. The second form of network (IBSS mode) is an ad-hoc mode, when stations communicate directly with each other without infrastructure. Put in other words, IEEE 802.11 Standard provides wireless networks with one- or two-hop topologies.

2.2 802.11s: mesh networking
IEEE 802.11s Draft standard [1] is an extension of 802.11 enabling transparent frame forwarding in arbitrary multi-hop

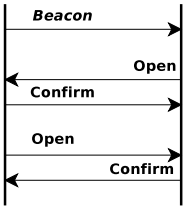

Figure 1: Peering link open handshake.

topologies. Multi-hop forwarding requires corresponding address scheme extension, detection of duplicates, etc. and is discussed in more detail in section 2.2.1. Each *mesh station* participating in the 802.11s wireless mesh network operates as link layer router and is responsible for cooperation with all the other mesh stations in frame delivery. This is done using several dedicated protocols, two of them being most important. Peering management protocol is responsible for neighbor mesh station (peer) discovery and management, it is described in section 2.2.2. Routing protocol is responsible for multi-hop path discovery and maintenance, see section 2.2.3 for more details.

Apart of basic multi-hop path discovery and forwarding functions, 802.11s standard includes a number of advanced features such as inter-networking, security, power safe, channel assignment, etc., they are briefly presented in section 2.2.4.

2.2.1 Addressing and forwarding

802.11s mesh stations are addressed using unique 48-bit MAC addresses. If mesh station has multiple physical interfaces then *both* station and every interface are addressed, it is assumed that station itself has an address of its first interface. The minimal information needed for successful multi-hop frame delivery includes four addresses: two of them being the frame original source and final destination and another two are current transmitter and receiver. This is known as 4-address addressing scheme. To forward frames, originating or ending (or both) at a point outside of mesh network, one or two additional addresses are required. This is known as 6-address scheme. Additional, comparing to standard 802.11 header, addresses are placed in the dedicated 6 to 24 octets long *Mesh Control* header which is added to all multi-hop frames right before 802.11 header.

Apart of address extension Mesh Control header contains an 8-bit flags field, one octet time to live (TTL) field to limit the effect of path loops and 4-octet mesh sequence number to suppress duplicates. Mesh sequence numbers are used for broadcast frames only and must be unique in the context of originator address. Every mesh station remember maximal known sequence number for every known originator and discards all frames with (cyclically) smaller sequence numbers from that originator as duplicates.

2.2.2 Peering management

The Peering management protocol is used to open, maintain and close links (or peerings) with neighbour mesh stations, choosing whether open a new link and closing links when detecting their failures. Mesh stations are not allowed to transmit frames, other than the ones used for peering management, to a neighboring mesh stations until a corresponding peer link has been established.

To be visible to its neighbors, every mesh station periodically sends small one-hop management frames known as beacons. As soon as beacons are sent strictly periodically (because it is needed for power save), if two beacons from two neighbor station have collided, they may collide forever. So, there is a mechanism that allows to avoid collisions among beacons. Peer link open handshake starts when mesh stations receives beacon from the previously unknown mesh station and decides to open link with it. A *Peering Open* one-hop management frame containing mesh parameters is sent to potential peer station. The mesh station processes the received parameters. If it agrees with the parameters, it sends a *Peering Confirm* management frame in response of the Peering Open frame. The peer link is established only when *both* stations have sent Peering Open requests and received Peering Confirm replies, this requirement guarantees that all established links are bidirectional. An example of successful peer link open handshake is shown on Fig. 1.

Standard intentionally does not specify any events causing peer link close. Link failure detection heuristics are important for routing protocol operation and will be discussed in section 3.2.1.

2.2.3 Routing

The routing (or path selection) protocol is used to discover and maintain multi-hop paths in the mesh network. As usual, routing protocol uses path selection *metric* to choose the optimal one of different possible paths.

802.11s standard allows vendor implement *any* routing protocol and/or path selection metric to meet special application needs. At the same time, standard includes a default mandatory routing protocol (HWMP) and default mandatory path selection metric (Airtime Link Metric) for all implementations, to ensure minimum interoperability between devices from different vendors. Only one routing protocol and only one path selection metric shall be used by a mesh station at a time.

The Hybrid Wireless Mesh Protocol (HWMP) is a mesh routing protocol inspired by AODV and tree-routing. It is believed that the combination of reactive (on-demand) and proactive elements of HWMP enables efficient path selection in a wide variety of mesh networks. HWMP uses a common set of protocol primitives, generation and processing rules adapted from Ad Hoc On Demand Distance Vector Protocol [4] for MAC addressing and link metric awareness. HWMP supports two modes of operation depending of configuration. Note that these modes are not exclusive and can be used concurrently.

In the *on-demand* mode the path discovery starts when a source mesh station has has a data to transmit to unknown destination. It broadcasts a Path Request (PREQ) management frame for that destination. Each station, receiving PREQ creates a route to a source, updates metric and forwards it. If the station has a valid routing information to a destination or the station is the destination itself, it generates a Path Reply (PREP) management frame. Whether an

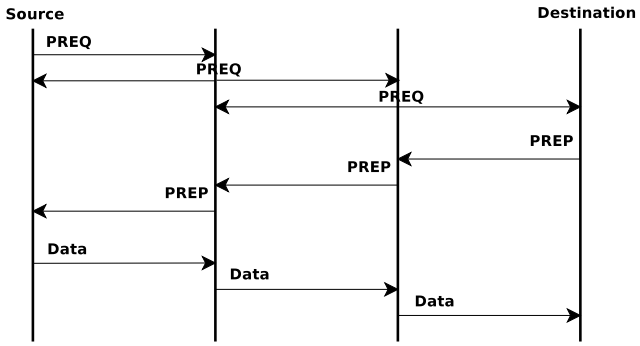

Figure 2: HWMP on-demand route discovery.

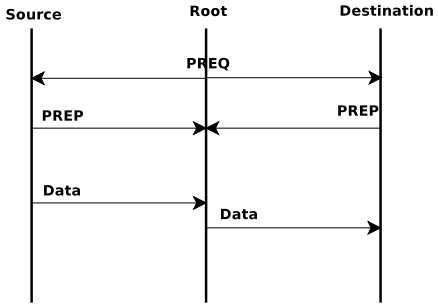

Figure 3: HWMP proactive route discovery.

intermediate station sends PREP depends on PREQ flags. PREP is forwarded hop-by-hop using created route to the source. The best route among multiple PREPs is chosen at source by path metric. Path selection uses a sequence number mechanism to assure that every station can distinguish current path information from stale one at all times in order to maintain loop-free connectivity. On-demand route discovery cycle is illustrated on Fig. 2. When peering management protocol detects that a link is broken, a Path Error (PERR) management frame is sent to all stations, which are known to use broken link in their routes. To remember potential receivers of PERR routing table keeps list of precursors. Precursor is the previous station in mesh path, since HWMP path discovery discovers a path both to source and a destination at each intermediate station, each intermediate station knows both next and previous stations of mesh path. When actively used route is discarded by PERR station starts new route discovery process as described above.

In the *proactive* mode of operation, a single mesh station is configured to be a path tree root and it broadcasts PREQs periodically. Each station receiving a proactive PREQ updates it path to root and answers to it by PREP always or when it has data to send (depends on the flags of the PREQ). Due to this, every station knows route to tree root and root knows a route to every mesh station. If direct route to destination is unknown, data will be forwarded to root which is responsible to forward it to the final destination, this is illustrated on Fig. 3.

Airtime multi-hop path selection metric is defined as the sum of independent *airtime link metrics*, it is assumed that better links have smaller metric. Airtime link metric reflects the amount of channel resources consumed by transmitting the frame over a particular link. This measure is known to be approximate and designed for ease of implementation and interoperability. Namely, the airtime estimates total medium access time needed for frame transmission. It takes into account an average number of retries, current data rate and some medium access overhead, such as frame headers, training sequences, acknowledgements etc.

2.2.4 Other protocols and features

Apart of these core functions, IEEE 802.11s draft includes a number of advanced features. Though they are not implemented in our NS-3 model, this sections briefly presents them to make the 802.11s picture complete. The first and one of the most important things is Mesh Coordination Function, which is an optional access method that allows mesh stations to access the wireless medium at selected times with lower contention that would otherwise be possible. Also our implementation of Peering Management protocol does not support mesh link security (i.e transmissions of all management frames needed to establish a secure link between a pair of mesh stations are not implemented). A portal announcement and proxy protocols, needed for inter networking are also out of scope of our model. These protocols are needed to transmit a data between mesh station and station outside a mesh network through portals and mesh access points. Power save is not implemented too in our model.

2.3 Network Simulator 3

Network Simulator 3 [2] is an actively developed discrete-event network simulator targeted primarily for research and educational use. NS-3 is free software, licensed under the GNU GPLv2 license [5], and is publicly available for research, development, and use. NS-3 was designed and implemented from scratch to be easily extensible, maintainable and understandable.

Ease of use and contributing, available high fidelity IEEE 802.11 MAC and PHY models together with real world design philosophy and concepts made NS-3 our platform of choice for 802.11s model development.

3. MODEL DESIGN & IMPLEMENTATION

In this section we overview the implementation of 802.11s model. Reader is supposed to be familiar with general NS-3 concepts and object model, though in-depth understanding of 802.11 model is not necessary.

The two main requirements to model design were identified:

1) Support multiple interfaces (wireless devices) at one node handled by single instance of mesh protocols “stack”, e.g. IEEE 802.11s.

2) Support different mesh networking protocol “stacks” via single protocol-independent mesh station architecture. The development of mesh networking protocols is considered as one of the hot research topics and no universally accepted solution is known for now. This is reflected in IEEE 802.11s draft which allows vendor implement any routing as well as auxiliary protocols.

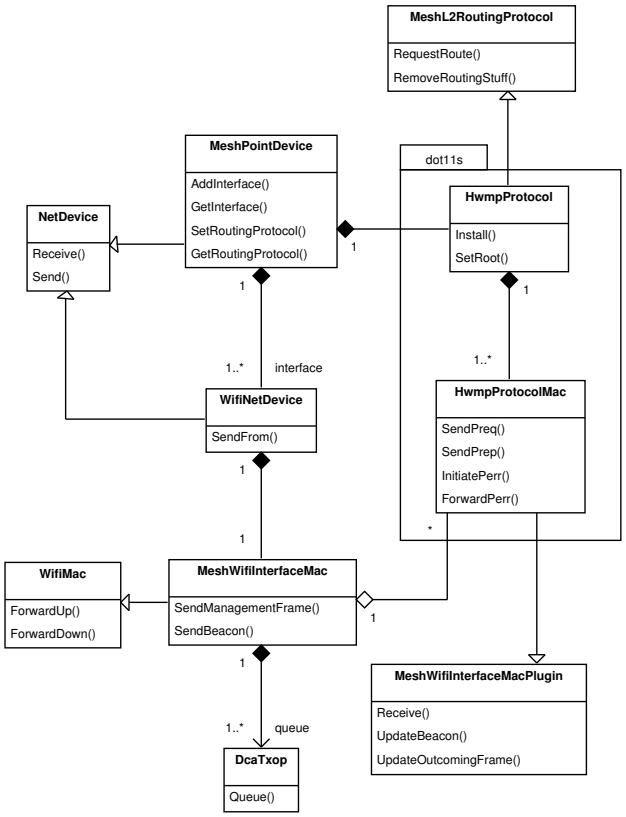

Figure 4: UML diagram of mesh core classes.

To meet these requirements we have designed and implemented runtime configurable multi-interface and multi-protocol mesh station architecture described in the next section. Section 3.2 presents particular IEEE 802.11s protocols implemented using this architecture.

3.1 Mesh module

The protocol-independent part of mesh networking model is located in `src/devices/mesh` module of NS-3 distribution[1]. Section 3.1.1 defines mesh object model and describes its core classes. Section 3.1.2 overviews transmission and reception packet flow.

3.1.1 Structure

The UML diagram of mesh module core class is shown on Fig. 4. This is a two tier architecture: the station tier includes mesh point device and mesh protocols, while the interface tier includes mesh interfaces and protocols *plugins* – a concept to be defined below.

The mesh station is modeled with special kind of network device, class *MeshPointDevice*, with routing functionality and control over underlying real network devices (interfaces) hidden from the upper-layer protocols, class *WifiNetDevice*. This network device operates by the following scheme: it receives a packet, makes all route discovery procedures, chooses an output interface and sends a packet. When receiving a packet, it delivers a packet to upper-layer protocols or forwards it further to a proper interface using routing protocol. Note, that any other network devices can coexist with MeshPointDevice on the same node.

Every MeshPointDevice has its own MAC address, known by network layer protocols. It is asserted, that single interface MeshPointDevice has an address of its interface. The concept of mesh station address is similar to the *main address* of OLSR protocol [6]. MAC addresses of the individual mesh interfaces are hidden from upper layers.

Apart of being the coordinator of its interfaces, mesh point device serves as the fixing point of all mesh *protocols*. The examples of mesh protocols are Peering Management Protocol and HWMP, discussed above. Any other mesh protocol can be implemented in the same way. All enabled mesh protocols are supposed to be tied with (aggregated into) mesh point device. Every mesh protocol (let it be *Foo* protocol) is implemented by two classes:

1. Class *FooProtocol* implements all station-level protocol logic and data base. An instance of FooProtocol is tied to mesh point device.

2. An instance of class *FooProtocolMac* is tied to each mesh interface MAC and extends its functions to support corresponding protocol. The extension mechanism is based on the plugin concept and is discussed below.

The interface between FooProtocol and FooProtocolMac is not restricted, but it is recommended for MAC level plugin to parse/make specific management frames and run interface specific state machine, while station level protocol shouldn't known frame details but implements protocol logic, global state, MLME and interacts with other protocols running on the same mesh station.

Single routing protocol (subclass of *MeshL2RoutingProtocol*) must be explicitly selected from all enabled protocols to allow route discovery. All frames will first pass through the routing protocol to resolve route information before forwarding down to interface(s). Class MeshL2RoutingProtocol is an abstract interface for all unicast mesh routing protocols. Its RequestRoute method allow asynchronous route discovery. RequestRoute is supposed to annotate route information as some protocol-specific tag(s) on the packet to send. Then, corresponding MAC plugin is responsible to read this tag(s) and correctly create frame header. Complementary RemoveRoutingStuff method is used to remove route annotation from packet. Explicit sequence of RequestRoute and RemoveRoutingStaff calls is shown in the next section.

Every mesh interface has its own MAC (class *MeshWifiInterfaceMac*[2]) and PHY. Instead of implementing all known mesh protocols, MeshWifiInterfaceMac implements only basic functions and runtime extensions mechanism – plugins (class *MeshWifiInterfaceMacPlugin*). Basic functions include Ad-Hoc MAC, as defined in [3], and beacon generation. Note

[1]We use ns-3.6 version to prepare this paper.

[2]Strictly speaking this is "high" MAC in parlance of NS-3 WiFi model.

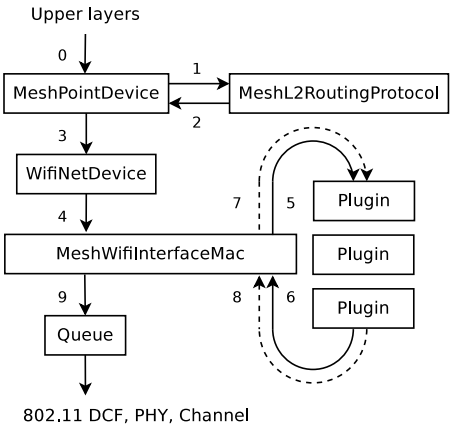

Figure 5: Sending a packet. Solid: data packet flow, dashed: beacon flow.

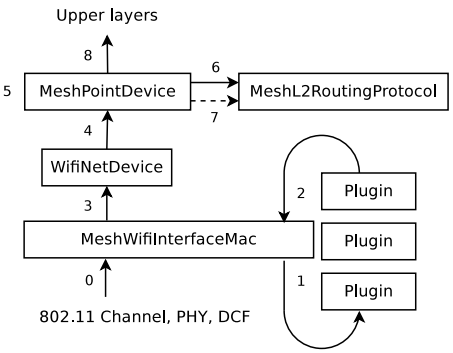

Figure 6: Receiving a packet. Solid: locally delivered packet, dashed: forwarded packet.

that ad-hoc MAC functions are inherited from the base class and beacons are periodically sent, beacon send time can be adjusted to avoid beacon collisions. MeshWifiInterfaceMac maintains a list of installed plugins. Each MAC plugin must define three pure virtual methods inherited from MeshInterfaceMacPlugin: Receive, UpdateOutcomingFrame and UpdateBeacon used to modify MAC behaviour for particular protocol needs. The detailed way, MeshWifiInterfaceMac uses its plugins is shown in the next section.

3.1.2 Packet flow

Figure 5 illustrates send packet flow in the operating mesh station, which is commented step-by-step below.

0) This step starts packet transmission. In this case we see packet, originated from upper (from the link layer point of view) layers of node protocols stack. Packet body, Ethernet header and QoS priority class (known as traffic ID, TID) are supplied.

1) Mesh point device calls its routing protocol RequestRoute method for every packet. Both packet and Ethernet header are passed. It is assumed, that RequestRoute works asynchronously. This means that RequestRoute immediately returns while packet is buffered inside routing protocol while path discovery procedure will be finished. To return packet on track when route is found, a *callback* of type RouteReplyCallback is passed as argument to RequestRoute. This paradigm allows to implement on-demand routing protocols as well as proactive ones.

2) This step starts when routing protocol has routing information for this packet. This information is annotated on packet using some protocol-specific packet tag(s). Then route reply callback is called by routing protocol to signalize that route is found and packet can be forwarded. The only protocol-independent information, mesh point device will know is an outcoming interface number (or *all* if packet should be sent from all interfaces).

3) Mesh point device dispatches packet to one (or all) of its interfaces. Note that all data packets are send in the name of mesh point, using WifiNetDevice::SendFrom method. This results in mesh station address in the From field of 802.11 header instead of interface address.

4) At this step mesh interface device passes packet to its MAC, calling MeshWifiInterfaceMac::ForwardDown method. Mesh interface MAC prepares 802.11 header to be added to packet before transmission.

5) Here mesh interface MAC asks all installed plugins to update packet and/or its 802.11 header calling UpdateOutcomingFrame method of every plugin. It is important, that one of these plugins corresponds to routing protocol used before. It is responsible to remove routing information annotated as packet tag(s) and update 802.11 header and add some protocol-specific headers if needed.

6) Every plugin can discard packet returning false from UpdateOutcomingFrame. Processing will stop and packet will be silently dropped in this case. If all plugins return true from UpdateOutcomingFrame, processing continues.

7) This is the first step for beacon frames. Beacons are periodically generated as an empty container of information elements (class *WifiInformationElementVector*), see [3] for the information element definition. Every plugin is asked to add protocol-specific information to beacon if needed. Beacon frame will be generated from the list of information elements added by individual plugins.

8) In contrast with data packets, plugins are not allowed to discard beacons and this step is always passed.

9) 802.11 header is added to the packet and it is enqueued to one of the 4 available TX queues depending on access category mapped from TID. Beacons are queued to the dedicated beacon queue.

Figure 6 illustrates packet receiving flow in the operating mesh station, which is commented step-by-step below.

0) This step starts packet reception for the mesh point interface. Packet body and 802.11 header are known here. Packet can be data as well as management frame.

1) Here mesh interface MAC asks all installed plugins to read and probably update the packet calling Receive method of every plugin. Both packet and its 802.11 header are passed to Receive. Protocol plugins are responsible to detect and parse own management frames or information elements and notify corresponding protocol instances about received information. All protocol-specific headers, such as Mesh Control header discussed in section 2.2.1 are removed here. Information from these headers, which is used later in the packet flow (e.g. TTL value from Mesh Control header), is annotated as protocol-specific packet tag(s).

2) Every plugin can discard packet returning false from Receive. It is recommended for protocol plugins to drop own management frames after processing. Packet processing continues only if all plugins return true from Receive.

3) Mesh interface MAC passes packet to mesh interface device. Packet, source and destination addresses are passed.

4) At this step LLC header is removed. Incoming interface number, packet, source and destination addresses and protocol number as said in LLC header are passed further to mesh point device.

5) Here mesh point device decides whether packet should be locally delivered or forwarded or both. Packet is locally delivered if its destination address is broadcast or multicast or equals to mesh point address. Packet is forwarded if destination address is broadcast or multicast or not equal to the mesh point address.

6) This step is for locally delivered packets only. Before passing packet to upper layers, mesh point devices removes all mesh specific annotated information calling RemoveRoutingStuff by routing protocol. The action of RemoveRoutingStuff is protocol specific, but it is asserted that no extra tags will be present after it.

7) This step is for forwarded packets only. Mesh point device calls its routing protocol RequestRoute method to initiate packet forwarding. This step corresponds to the Step 1 of Fig. 5. From this step packet flow is defined by "Sending packet" diagram.

8) This step is for locally delivered packets only. Packet is passed to upper layers using preset callback of type NetDevice::ReceiveCallback. From this step packet flow is the same for all devices.

3.2 802.11s protocols

In this section we present an overview of 802.11s specific part of mesh networking model. This set of models located in `src/devices/mesh/dot11s` module of NS-3 distribution. All relevant classes are placed inside mesh::dot11s namespace. At present, two core protocols of 802.11s draft standard were implemented: Peering management protocol and HWMP. This two parts are crucial for the IEEE 802.11s network to operate. In next two section we overview the corresponding implementation details.

3.2.1 Peering management

Peering management protocol (PMP) is responsible for maintaining connections with neighbours and for beacon collision avoidance, see section 2.2.2 for details. PMP is implemented by *PeerManagementProtocol* and *PeerManagementProtocolMac* classes as described in section 3.1.1.

According to standard, links are opened with all neighbours, if their beacons contain mesh ID equal to mesh ID of a given station.

IEEE 802.11s standard does not define the way of closing peer link, we have implemented two ways of doing this. The first one is to close link due to beacon loss. Every beacon contains beacon interval, and beacons are sent strictly periodically by each station. So, at every moment for every link we exactly know, how many beacons were lost since last successfully received one. If beacon loss counter has exceeded a configurable threshold value MaxBeaconLoss link shall be closed. We use MaxBeaconLoss = 5 in our simulations. Note, that at least 2.5 s is needed to detect link failure this way using default beacon interval of 0.5 s. The second heuristic for link failure detection applies when link is used for data transmission. If the number of successive frames were failed to transmit to a given neighbour (transmission failure is an event, when MAC refuses to transmit frame due to retry threshold), exceeds a threshold value MaxPacketFailure, link shall be closed. It is worth to mention, that a single transmission failure may occur because station has many neighbours and frame was failed to transmit due to collisions. We use MaxPacketFailure = 5 in our simulations. This heuristics works much faster than the first one, but requires active data flow.

It worth to mention that values of MaxBeaconLoss and MaxPacketFailure setup a trade-off between fast link failure detection and link stability. Setting lower thresholds will generally cause faster failure detection and less stable links. Also it should be noted, that since link close causes PERR propagation in HWMP, the efficiency of PMP to detect link failures is important for routing protocol performance.

As required by 802.11s draft Peer Link Close message is sent when PMP detects link failure and closes link. Comparing our model with Linux kernel implementation [7] (formerly known as open80211s [8]) we have found that it doesn't send peer link close messages when closing links. This seems to be reasonable, because when link is detected as failed by several successive transmission failures, sending peer link close in addition may not be useful. In our model we send peer link close in any case to follow standard.

Implementation of beacon collision avoidance conforms to 802.11s draft.

3.2.2 HWMP

As any other mesh protocol, HWMP is implemented by two classes: *HwmpProtocol* and *HwmpProtocolMac*.

Apart of routing, our implementation of HWMP is responsible for adding/parsing Mesh Control header and filtering broadcast data frames by sequence number and all data frames by TTL, this is done by HwmpProtocolMac.

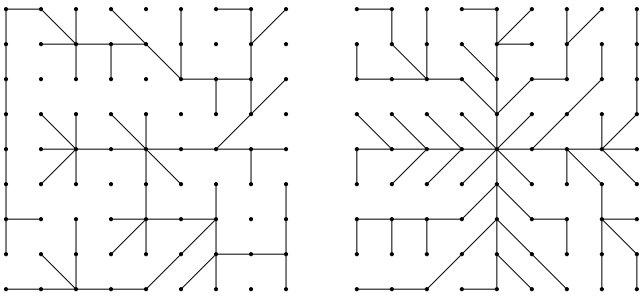

Figure 7: Examples of HWMP proactive path tree in 9×9 grid topology. Left: after first PREQ, right: steady state.

HwmpProtocol class is responsible for route discovery procedure, routing table management and route request queue management. Known routing information is the next hop address in this case and is annotated in dedicated packet tag. Also packet TTL and sequence number are annotated by HwmpProtocol and are managed as defined by standard. HwmpProtocolMac is responsible to remove this tags and fill local source and destination addresses (receiver and transmitter) of 802.11 header, and adds a Mesh Control header using sequence number and TTL. Since internetworking is not supported we don't use 5- and 6-address extensions of Mesh Control header.

HwmpProtocol keeps station level routing table. In addition to 802.11s standard defined information our routing table keeps also identifiers of outgoing interfaces in addition to next hop address and precursor address. This allows multiple interfaces HWMP operation.

We have implemented an optional feature of 802.11s to send PERR as unicast frame to each precursor or as broadcast frame. The same option exists with broadcast data frames: one may send it as unicast to all neighbours or to send as broadcast. In our model this features were implemented as follows. The configuration of our model has three configurable thresholds: UnicastDataThreshold, UnicastPreqThreshold and UnicastPerrThreshold. These thresholds define a maximal number of neighbors when unicast data, PREQ and PERR delivery for all neighbors is used instead of single broadcast delivery. We use UnicastDataThreshold = 1, UnicastPreqThreshold = 1 and UnicastPerrThreshold = 32 in our simulations.

Current implementation of 802.11s standard in Linux kernel does not support adding multiple destinations to a single PREQ or PERR. Since standard requires, that PREQ generation and PERR generation ratio shall be limited (10 frames per second by default), this drawback may seriously descend quality of the whole network. We have implemented this feature in the HWMP model.

4. SIMULATION RESULTS

To show our model "in work" we present simulation results for two simple scenarios. Both are static grids of 81 (9×9) mesh stations, with grid step being 0.6 RX range, so each station can communicate with maximum eight neighbors by the grid cell sides and diagonals. The first use case (section 4.1) shows an operation of the mesh network in proactive mode, while the second one (section 4.2) demonstrates the operating of the on-demand mode. Much more detailed simulation study is presented in [9].

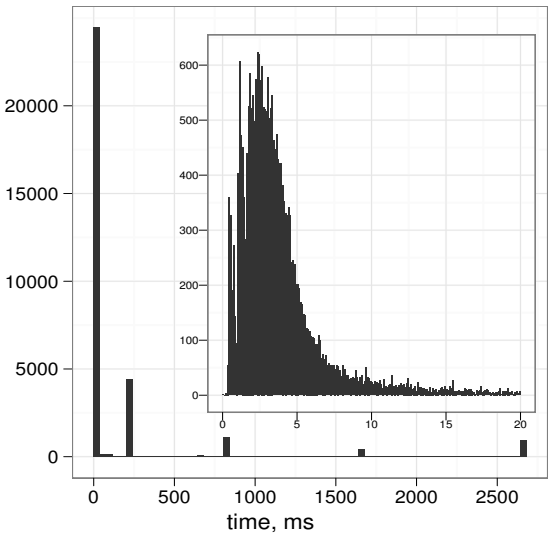

Figure 8: Histograms of route discovery time. Y axis: number of attempt per bin of total 32263 attempts. Main plot: all attempts, bin width 40 ms. Subplot: first attempt only, bin width 0.1 ms.

4.1 Proactive mode

This use case simulates a static mesh network used to create an Internet access infrastructure. The grid center station is assumed to have an Internet connection, all the other mesh stations are clients. This is natural use case for HWMP to operate in proactive (tree-building) mode. We have selected central mesh station as a HWMP tree root and initiated 10 TCP streams with a root station as the source of the stream and 10 stations taken randomly among other stations as the sink of the stream. An example of built proactive trees in this scenarios are presented at figure 7. On can see that in spite of significant PREQ loss probability, the tree is refined with time and is almost perfect after 100 seconds of protocol operation (50 proactive PREQ intervals). After 20 independent runs with duration of 100 seconds each we can estimate that an average total "download" speed to be about 200 KBit/s. This is much less than channel speed (which was equal to 6 Mbit/s, because the distance between stations and TX-power did not allow to use higher channel speeds) because of multiple retransmissions and big overhead due to proactive paths maintenance.

4.2 On-demand mode

This scenario simulates mesh network used to enable peer-to-peer communications between mesh stations. This is a natural use case for HWMP to operate in the reactive (on-demand) mode. The same grid topology was used and we initiated 10 CBR streams with random endpoints to simulate VoIP traffic. Every stream has an intensity of 50 packets per second with 20-bytes payload, which is similar to mod-

ern vocoder [10]. Note, that UDP, IP and 802.11 headers increase the real length of the packet.

To estimate the quality of the voice, we have used E-model described in [11]. It is believed that if the R-factor is more than 50 [12], the voice service is available. We define the voice *availability* as the total (for all stations and runs) part of time, when the voice was available. After 100 independent runs with duration of 150 seconds each we assume that voice service is available for 85.4% of time. Also we have fixed a duration of every route discovery procedure and figure 8 shows the empirical distribution of route discovery time. The final peak at main plot is responsible for undiscovered routes (the total number of successive PREQ for single destination was limited to 4). Large number of runs was needed to demonstrate route discovery time histogram.

5. FUTURE WORK

We identify three distinct directions of future work with IEEE 802.11s NS-3 model.

First, model should be refined to include all draft features as described in section 2.2.4. This development should concur with IEEE 802.11s draft standard development.

Second, model should be extensively verified and validated. For now we have verified the model using a large number of unit tests for each individual component (such as all information elements and headers, routing table, etc.) as well as a number of trace based system tests (such as two mesh stations, four mesh stations in the chain, etc.). These tests are included into NS-3 test system. It is needless to say, that the number of non-trivial system tests should grow. The fact, that 802.11s draft is implemented in latest Linux kernels allows our model to be experimentally validated. Indeed, comparing simulation routing behaviour with measured one using some form of wireless testbed (e.g. ORBIT [13] or CMU Wireless Emulator [14]) one will develop in potential users the confidence they require to use a model.

Third, mesh station architecture created allows for fast and easy implementation of other layer 2 meshing solutions, e.g. Batman [15]. This will allow users to do realistic mesh networking simulations and compare different meshing solutions using the same usage scenarios, applications and PHY models. Working along this line, we have developed a model of FLAME mesh protocol [16] (to be discussed elsewhere).

6. CONCLUSION

We have implemented the IEEE 802.11s mesh networking model in the open-source Network Simulator 3 environment. The model includes all core functions of draft standard. It is shown to be correct and useful for realistic simulation study of wireless mesh networking. In the course of writing the model, some changes and heuristics were added to IEEE 802.11s draft standard to enable its protocols operate correctly. These changes are highlighted in this paper.

Although described in this paper, we have not implemented many advanced function of the 802.11s including internetworking, security, power save, mesh coordination function, etc. This is a subject of future work.

As the by-product of 802.11s model development, we have created protocol-independent multi-interfaces mesh station architecture, suitable for fast modeling of any link layer meshing solution. It is expected that models of the other mesh protocols will follow.

7. ACKNOWLEDGMENTS

We are grateful to Alexey Kovalenko and Andrey Mazo who have contributed to the 802.11s model development on its early stages, as well as to Faker Moatamri and Mathieu Lacage who have spend a lot of time reviewing our code before it official merge into NS-3. Artem Krasilov has made an important contribution finding a number of implementation bugs.

8. REFERENCES

[1] *IEEE P802.11s/D3.0. Draft STANDARD for Information Technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment: Mesh Networking [Electronic resource]*, 2009.

[2] The ns-3 network simulator. `http://www.nsnam.org/`.

[3] *IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2007.

[4] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[5] GNU General Public License, version 2. `http://www.gnu.org/licenses/gpl-2.0.html`.

[6] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[7] Linux Wireless. `http://linuxwireless.org/`.

[8] open80211s Consortium. `http://open80211s.com/`.

[9] Kirill Andreev and Pavel Boyko. Simulation study of 802.11s mesh networking. In preparation.

[10] ITU-T Recommendation G.729: Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP). Technical report, 2007.

[11] J. A. Bergstra and C. A. Middelburg. ITU-T Recommendation G.107 : The E-Model, a computational model for use in transmission planning. Technical report, ITU, 2003.

[12] ITU-T Recommendation G.109: Definition of categories of speech transmission quality. Technical report, 1999.

[13] Orbit. `http://www.orbit-lab.org/`.

[14] Carnegie mellon university wireless emulator. `http://www.cs.cmu.edu/~emulator/`.

[15] B.a.t.m.a.n. `http://www.open-mesh.org/`.

[16] Herman Elfrink. Forwarding Layer for Meshing. Revision 2.0. Technical report, Twente Institute for Wireless and mobile Communications, 2006.