Ey-Wifi: Active Signaling for the ns-3 802.11 Model

Hana Baccouch, Cédric Adjih, Paul Mühlethaler

Ey-Wifi : Active Signaling for the ns-3 802.11 Model

Hana Baccouch, Cedric Adjih, Paul Muhlethaler

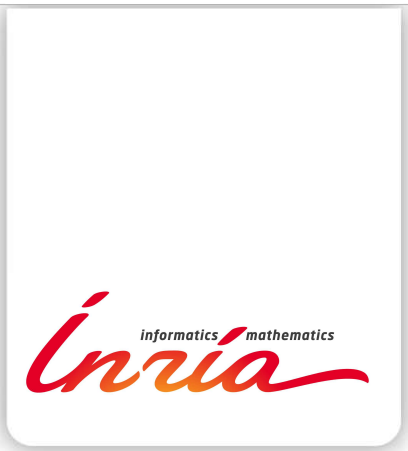
RESEARCH REPORT

N° 8418

November 2013

Project-Team Hipercom2

ISSN 0249-6399 ISRN INRIA/RR--8418--FR+ENG



Ey-Wifi : Active Signaling for the ns-3 802.11 Model

Hana Baccouch, Cedric Adjih, Paul Muhlethaler

Project-Team Hipercom2

Research Report n° 8418 — November 2013 — 26 pages

Abstract: In wireless networks, wireless channels are often shared between multiple nodes, which may interfere when they transmit simultaneously. The media access control (MAC) scheme coordinates channel access between the different nodes. In this article, we focus one specific method of channel access, Elimination-Yield Non-Preemptive Priority Multiple Access, EY-NPMA, which is part of the European standard wireless HIPERLAN [1]. EY-NPMA is a wireless MAC method that partly relies on "active signaling": i.e. transmitting (activity) to simply signal a node's presence (with pseudo-noise), rather than transmitting data (bits). In this article, we focus on the implementation of EY-NPMA channel access scheme as a module in the ns-3 network simulator. We are interested particularly to study the adaptation of the ns-3 Wifi module to support EY-NPMA features. Hence, our main focus in this work is the implementation of a such an extension, yielding Ey-Wifi, in a module of the network simulator, ns-3, with the emphasis on the software architecture. We are interested further to present some simulation results, analyzing the performance of both Wifi and Ey-Wifi.

Key-words: ns-3, MAC, 802.11, EY-NPMA, Hiperlan type 1

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Ey-Wifi : Active Signaling for the ns-3 802.11 Mode

Résumé : Dans les réseaux sans fil, le canal est partagé par plusieurs noeuds, qui peuvent interférer s'ils transmettent simultanément. Un mécanisme d'accès au canal (MAC) cherche à bien gérer l'accès au canal entre les différents noeuds. Dans ce rapport, nous nous intéressons à une méthode spécifique d'accès au canal; EY-NPMA (Elimination-Yield Non-Preemptive Priority Multiple Access) qui correspond à la méthode d'accès du standard européen Hiperlan 1. EY-NPMA est une méthode d'accès basé sur le signalement actif. En effet, le noeud signale sa présence en envoyant un burst sur le canal. Celui qui a le plus long burst accède au canal et envoit ses données. Nous nous intéressons particulièrement à l'adaptation du module Wifi du simultaeur ns3 pour supporter les caractéristiques de EY-NPMA. Nous nous intéressons aussi à l'évaluation de cette méthode d'accès EY-NPMA. Nous présentons ainsi les résultats de simulations iluustrant les performances des deux méthodes d'accès.

Mots-clés : ns-3, MAC, 802.11, EY-NPMA, Hiperlan type 1

Contents

1 Introduction

In the radio spectrum, available frequencies for radio communication systems are a scare resource, and therefore efficiency is a prime concern for modern wireless communications. Part of the efficiency depends on the media access control (MAC) of a wireless network: the objective of the MAC is to coordinate wireless channel access between devices.

Unlike, for instance, mobile wireless telephony networks, in license free networks such as 802.11 networks [2] (Wifi, VANET-802.11p), HIPERLAN [1] or 802.15.4, a single entity is not controlling all the potential co-interferers in many situations. This makes efficient channel access a greater challenge.

For these reasons, most of these wireless technologies include at least one uncoordinated medium access control scheme: it is based on carrier sensing (in order to check whether another transmission is currently occurring), on acknowledgments (in order to attest that a transmission has been successful), and a general protocol which includes a number of additional mechanisms. The ideal functioning is as follows: a node with data to exchange transmits after it senses that the wireless channel is free; and repeats the transmission until it is acknowledged by the destination (or until a maximum retry is reached).

In reality, a practical medium access scheme has to consider the case where multiple transmitters that wish to transmit exist at the same time, and where some potential transmitters might not hear each other. Hence, an actual wireless access scheme is more complex. For instance, most methods in the 802.11 family use a "contention window" based scheme, where essentially a node has to check that the channel is unoccupied for a randomly selected duration before attempting transmission; upon failure, the duration (maximum) is increased (see Distributed Coordinated Function, DCF in [2]).

An additional access mechanism is present in HIPERLAN [1], Elimination-Yield Non-Preemptive Priority Multiple Access, EY-NPMA. Historically, Hiperlan is a European standard from ETSI, which was developed at the same time as 802.11 (an initial version was defined in the mid-90s), with the same objective of providing wireless local area networks, hence with the same constraint of efficient access. In HIPERLAN, the objective was to provide access with less variation in the medium access delay, with two components: active signaling (see for instance [4, 5]), and a priority scheme (similar to the one which is also present in 802.11, with the extension 802.11e [3]).

The ever-growing number of wireless devices, and the continued success of Wifi networks, are motivations for designing improvements of the 802.11 media access method: a natural choice is to use the EY-NPMA mechanism in the 802.11 methods.

In this article, our main goal is to present our implementation of such an extension in ns-3, that we called Ey-Wifi. To implement Ey-Wifi module in ns-3, we extended the existing implementation of Wifi in ns-3. Because 802.11 and HIPERLAN access methods have many common elements, so that integration is relatively seamless. We describe in greater detail the implementation itself of such a method in the ns-3 simulator: this is a first step towards evaluating the performance of the mechanism, in a modern wireless environment.

This paper is organized as follows: in section 2, we start with a description of EY-NPMA; in section 3, we give a brief overview of Ey-Wifi; in section 4 we describe the ns-3 simulator, and the architecture, design, implementation and internals of the 802.11 module. Section 5 is devoted to the implementation of Ey-Wifi in the ns-3 simulator and the addition of an EY-NPMA module. Section 6 illustrates the simulation results analyzing the performance of both acces protocols (using Ey-Wifi and Wifi module). Finally section 7 concludes. In appendices, we describe how to use the Ey-Wifi module : downloading, compilation running examples, tracing, plotting and also how to write a scenario using Ey-Wifi.

2 EY-NPMA Medium Access Protocol

EY-NPMA stands for Elimination-Yield Non-Pre-emptive Priority Multiple Access. EY-NPMA is a contention based protocol that has been used as the medium access scheme in HIPERLAN type 1. EY-NPMA is based on active signaling. A node requests access to the medium by transmitting a burst signal. It employs prioritized access to handle different classes of traffic regarding quality of services and demonstrates very low collision rates. The channel access cycle comprises three phases : the priority phase, elimination phase and yield phase. Below, each phase is presented briefly, according to the ETSI specifications [1].

2.1 Priority phase

The purpose of the priority phase is to guarantee that only nodes whose transmission has the highest priority will survive the first phase. At the end of a transmission, when a node detects that the channel is free, it waits for a period called the channel synchronization interval. If no transmissions were detected during this interval, all the nodes take a waiting time according to their priorities. There are five priorities (from 0 to 4) in total such that the shorter the lifetime of the packet, the higher the priority allocated to it is. During this period, a node keeps listening to the channel. If the channel remains idle, the node sends an assertion slot. The transmission of an "assertion slot" indicates that the node has the highest priority. The other low priority nodes detect that the channel is busy, and will quit the contention.

2.2 Elimination phase

In the second phase, only nodes with the same priority are contending for the channel. The contending nodes send an elimination burst of a random length, which is from 0 to 12 elimination slot intervals long. After transmitting the burst, a node senses the channel for the elimination survival verification interval. If the channel remains idle, it continues to the yield phase. Otherwise, it exits the contention. In this phase, the nodes which send the longest elimination burst will survive and proceed to the next phase. Other nodes, which have shorter burst, stop their burst transmission earlier, and sense that the channel is busy. Then, those nodes will quit the contention. In this way, a great number of the contending nodes is eliminated, and only few of them proceed to the yield phase.

2.3 Yield Phase

This phase is the last phase before the transmission of packets. The purpose of the yield phase is to further reduce the number of contending nodes. Each surviving node, listens to the channel during the yield period, which may be from 0 to 9 yield slot intervals long. The node having the smallest yield period is the final winner and it starts sending its packet right after the yield period. All the contending nodes detect its transmission, and quit the contention.

3 Overview of Ey-Wifi

The goal of Ey-Wifi is to integrate the features of the EY-NPMA in 802.11. As described in more detail below, in section 4.1.1, the 802.11 MAC is similar to some parts of EY-NPMA. This is specially true for the variants of enhanced distributed channel access (ECDA) of 802.11e [3] which include a similar mechanism to the "priority phase" of EY-NPMA.

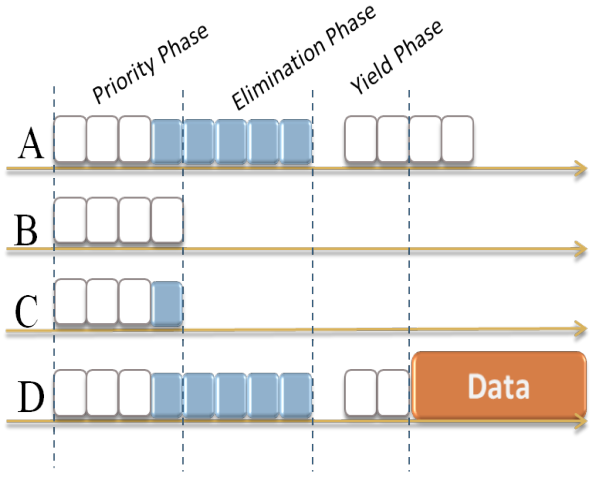


Figure 1: Overview of the EY-NPMA protocol

Hence, from the conceptual point of view, the main addition of Ey-Wifi, is the addition of the elimination phase, with the transmission of a burst. The start of the slots, and their duration, are selected to be identical to the 802.11 variant considered (SIFS is respected etc.). The transmission of the burst is done, as mentioned in Hiperlan standard, with a modulation similar to the preamble in 802.11.

In addition, the yield phase can be adjusted, because of the lower probability of collision after the elimination phase: it is not necessary to apply the update of the contention windows over several periods of inactivity of the wireless channel. Instead, in the yield phase, a new (small) number of slots is selected every time, leading to a shorter average access time.

Our main contribution is the implementation of th EY-NPMA scheme in Wifi networks in the ns-3 simulator, while maintaining the interoperability of the module with Wifi nodes. We aim to implement a configurable module which may enable simulation at once Wifi nodes and Ey-Wifi nodes.

4 WLAN simulation in ns-3

The ns-3 network simulator is a recently released next generation simulator intended to replace the popular ns-2 simulator. It introduces many features over its predecessor and aims to become the leading network simulator. The existing library of modules such as Wifi and WiMax, routing protocols and mobility models, makes ns-3 well suited for simulating wireless networks. The 802.11 based wireless module implements most of 802.11 standards (802.11a, 802.11b, 802.11g, 802.11e ..). QoS or non-QoS modes are both supported by the currently Wifi module. Moreover, the physical layer associated to the Wifi module, implements several physical layer models. Another major feature of ns-3 is its ability to be used as a real-time emulator. ns-3 has been designed to allow the use of the same protocol implementations in both simulation and testbed environments.

Key objects in the simulator are Nodes, Packets and Channels (fig. 2). A node represents a network element, to which Applications, Protocol stack, and NetDevices can be added. Each NetDevice is attached to a channel, over which it sends and receives packets. A Node may be

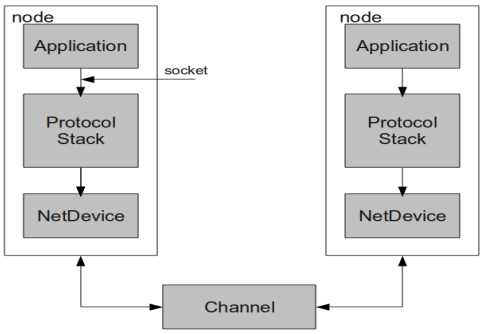


Figure 2: ns-3 node architecture

connected to more than one Channel through multiple NetDevices. A node may use multiple protocol components, which handle packets received by the NetDevice. Each node can also handle a list of Applications.

The binding interfaces between the components of the node are designed to be similar to those in real systems. The interface between Application and Protocols is based on a class called Socket. This class is used by the application to send and receive traffic to the protocol stacks that are attached to a node. A Channel can represent a wired network cable or a wireless transmission medium. Each Channel object may be connected to a list of NetDevice.

To design Ey-Wifi module, we focus on the implementation of the WifiNetDevice on ns-3. More precisely, we are interested to go through the MAC layer implementation of the Wifi module which is the topic of next section.

4.1 Architecture of IEEE 802.11 module

The implementation of IEEE 802.11 was ported to ns-3 from yans [7]. yans (Yet Another Network Simulator) was a prototype project by Mathieu Lacage and Tom Henderson. It implements a MAC and a physical layer that conforms to the 802.11a specification. The focus of the MAC design is to model all the complexities in the IEEE 802.11 CSMA/CA mechanism which is known as DCF (Distributed Coordination Function). The 802.11 DCF is used to decide when to grant access to the medium. Its mechanism is presented briefly below.

4.1.1 802.11 DCF access mechanism

The DCF access method is based on a CSMA/CA MAC protocol. Every node must sense the medium, in order to check the state of the channel (idle or busy). If a node has data to send, but it sees that the channel is busy, then it waits for the end of transmission.
At the end of transmission, it must wait for a DIFS (DCF Interframe Space). To prevent all nodes sensing the channel from beginning their transmission at the same time, each node chooses a random waiting time called Backoff before starting transmission. The range from which the Backoff value is chosen is called the Contention Window (CW). If the channel becomes busy during Backoff, Backoff counter decrementing is stopped until the channel becomes idle again.

DCF uses ACK packets to acknowledge the reception of unicast data transmission.

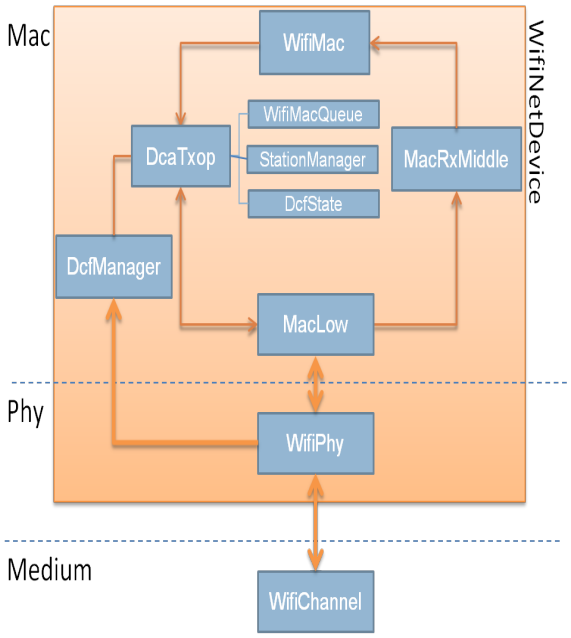


Figure 3: Wifi module architecture

The DCF mechanism is also based on the use of the optional RTS/CTS (Request- To-Send and Clear-To-Send) scheme. If a node has a packet to send, it firstly transmits an RTS packet to request the channel. If the receiver is ready to receive data, it replies with a CTS packet. After the sender receives the CTS packet successfully, it transmits the actual data packet.

4.1.2 Wifi Module overview

Figure 3, shows an overview of 802.11 module architecture in the ns-3 simulator. The WifiNetDevice class holds together WifiChannel, WifiPhy, WifiMac, and WifiRemoteStationManager.

If transmission is initiated by an application, the WifiNetDevice interface sends the packet to WifiMac class which handles high MAC level functions.

Each WifiNetDevice is aggregated to a StationManager which stores all transmission parameters (e.g. payload data rate, RTS/CTS, fragmentation, etc.) The main classes implementing the IEEE 802.11 DCF scheme are DcfManager and DcaTxop classes.

DcaTxop can handle only one packet. If other transmissions have been initiated, WifiMac pushes the packets to WifiMacQueue.

DcaTxop is used to handle the request access to the channel from DcfManager. When access is granted, DcaTxop pushes the packet to MacLow for transmission. MacLow initiates data transmission.

WifiPhy class is designed to support all physical layer related issues, such as channel sensing, SINR computation, interferences, etc. Its design is sufficiently generic that it is able to support the implementation of different MAC designs.

YansWifiPhy is the implementation of the IEEE 802.11 physical layer. It models an additive Gaussian Noise Channel (AWGN) with cumulative noise handled by InterferenceHelper.

The abstract class WifiChannel is designed to model the radio signal transmission. YansWifiChannel is the only implementation modeling an IEEE 802.11 channel. WifiChannel calcu-

lates the delay of received packets according to PropagationDelayModel. PropagationLossModel is used to calculate the reception power.

4.1.3 Implementation of DCF access mechanism

In this section, we describe in detail the process of computing DCF Backoff and channel access granting. Each packet in WifiMacQueue is associated to a DcfState. DcfState is designed to keep track (such as Backoff, CW, ..) of the state needed for a single DCF function. DcfManager can handle a set of DcfStates. Its major function is to handle Backoff counting and schedule events, such as access granting, Backoff end, etc. For each packet in the queue, DcaTxop calls DcfManager::RequestAccess to request access to the medium. The channel can be in one of two possible states:

- The channel is busy. In this case, a collision is notified, and the Backoff procedure is restarted.
- Otherwise, the channel is idle. The function GetBackoffEndFor() is called to calculate the expected end of Backoff. If the Backoff has already expired, medium access is granted by calling DoNotifyAccessGranted(). Then, packet is pushed to MacLow for transmission.

If two local DcfState instances are expected to get access to the channel at the same time, the first registered DcfState gains access to the channel. DcfManager notifies other DcfStates of an internal collision via DoNotifyInternalCollision().

When medium access is granted, transmission parameters are requested from StationManager. They are signaled to the MacLow via MacLowTransmissionParameters. If fragmentation is enabled. It is handled by DcaTxop, which sends only the packet fragment to the MacLow. In the IEEE 802.11 standard, using acknowledgments is mandatory for unicast transmission. Received or lost packets are signaled to DcaTxop which handles retransmissions. In [10], Bingmann presents a detailed description of the implementation of Wifi in the ns-3 simulator.

5 The EY-NPMA Module: Ey-Wifi for ns-3

In this section the modifications added to ns-3 to support EY-NPMA access rules are described. The purpose is to extend the existing code of the Wifi MAC protocol to implement the EY-NPMA channel access mechanism, yielding Ey-Wifi.

The Dcf scheme and EY-NPMA access rules have common phases. To implement Ey-Wifi module as an extension of the ns-3 Wifi module, several changes are integrated. Figure 4 shows the main changes in the architecture of Wifi implementation.

- **Modified classes** : The base classes modeling and handling Dcf scheme have been modified to support burst transmission. We have mainly changed DcaTxop, DcfManager, StationManager and MacLow classes to add EY-NPMA features.
- **Added classes** : EyState class, whose principal task is to handle burst transmission.
- **Reused classes** : The classes modeling the physical layer are used without any changes.

Major changes are described in detail in section 5.1. In Dcf, a node may transmit if and only if the medium is sensed to be idle during the Backoff interval. The main idea of the EY-NPMA implementation is to repeat this process twice.
In the first phase, the node senses the medium. If the channel is idle, it sends a burst signal. At

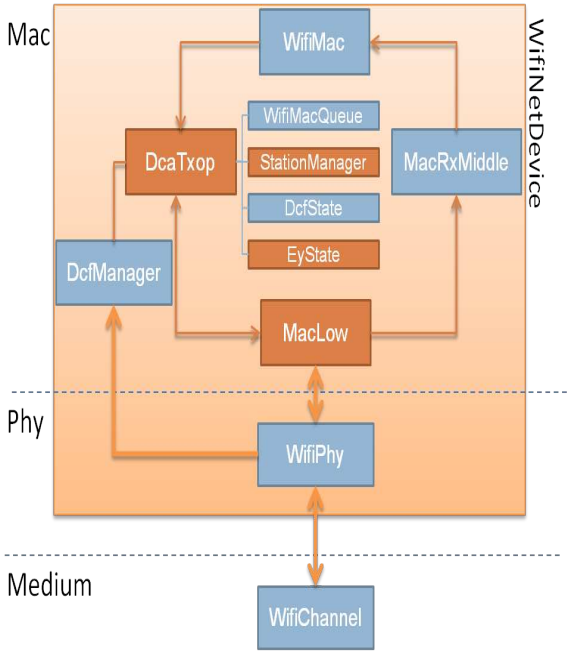


Figure 4: Ey-Wifi module architecture

the end of the burst transmission, the node must sense the medium again and sends data packets if the channel is idle. The yield phase and data transmission in EY-NPMA can be considered as equivalent to Backoff countdown and grant access processing in DCF. The extra feature of EY-NPMA is the elimination phase.
As explained above, DcaTxop and DcfManager already handle many of the required functionalities.

We have chosen to create a new "State" which keeps track of the burst for a single elimination phase. EyState (for Elimination-Yield State) has been designed to handle the first two phases of EY-NPMA. Several modifications have been performed mainly in DcaTxop , DcfManager and MacLow classes. To be able to run both modules Wifi and Ey-Wifi on the same ns-3 repository, we have renamed all classes of our module (even Wifi non modified classes) to "Ey<ClassName>" (for example : DcfManager becomes EyDcfManager).

5.1 Operation sequence

In this section, we describe the sequence of actions that take place during a simulation of the EY-NPMA protocol. In figure 4, the UML diagram shows an overview of EY-NPMA components. As seen in the figure, the module is composed of several C++ classes, each representing a component of the Ey-Wifi module and handling a particular functionality. This diagram only shows important attributes and functions.

EyState class inherits all methods and attributes from EyDcfState class. In addition to functionalities and tracks stored by EyDcfState, EyState handles the burst slots processing.
For each packet in WifiQueue, two EyDcfStates are aggregated; an EyState to handle the burst transmission, and a EyDcfState to handle Data transmission. When a node has a packet to send, EyDcaTxop requests access by calling EyDcfManager::RequestAccess(EyDcfState). If a

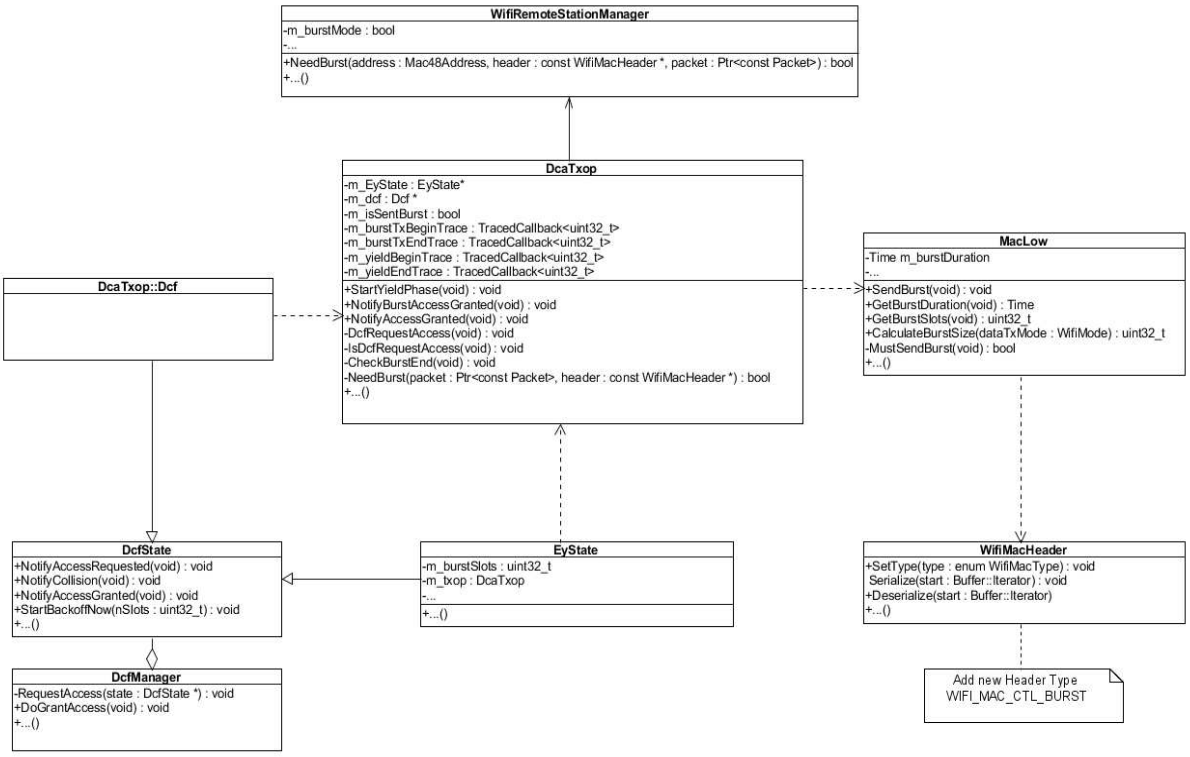


Figure 5: UML diagram of Ey-Wifi related classes

burst has already been sent, RequestAccess takes a "normal" EyDcfState as parameter. Otherwise, RequestAccess(EyState) is called.
RequestAccess(EyState) checks if Backoff is fully elapsed (the Backoff here is equivalent to priority waiting time in priority phase) The same function GetBackoffEndFor() is called to calculate the expected end of Backoff. If access is requested, Backoff is fully elapsed and channel is idle, the EyDcaTxop::NotifyAccessGranted is called by EyDcfManager. This function has been modified, to support the transmission of the burst signal.

In NotifyAccessGranted(), the coordinator verifies whether the burst has not been already sent. If not, NotifyBurstAcessGranted() is called to inform EyMacLow that access is granted for burst transmission.

EyMacLow handles the creation and transmission of the burst signal. The burst signal is modeled by a ns-3 packet object. The size of this burst packet depends on the random length burst chosen by the node. CalculateBurstSize() is called to calculate burst size. Once the access is granted and burst size is calculated, SendBurst() initiates the transmission of the burst. At the end of transmission, StartYieldPhase() is scheduled. RequestAccess(EyDcfState) is called to request access for data. If the channel is idle, the node takes a random waiting time before transmitting its data. StartBackoffNow() is called to start a Backoff by initializing the Backoff

counter to the number of yield slots specified. At the end of yield phase, EyDcaTxop checks again medium state. If channel is idle and Backoff is entirely elapsed, medium access is granted. Then, NotifyAccessGranted is called and data transmission is initiated.

On collision events, NotifyCollision and NotifyInternalCollision are invoked. Nodes, who quit the contention, must wait for the end of the current transmission. Requests for access are rescheduled at the next predicted end of current transmission automatically.

Note that the burst signal is internally transmitted in ns-3 as a data packet. A new EyWifiMacType has been defined to distinguish a burst from other packets. EyWifiMacHeader class implements the IEEE 802.11 MAC header. The new header type, yielding `WIFI_MAC_CTL_BURST`, has been added to the predefined mac type.

5.2 Illustration

To illustrate the functioning of the MAC layer, a scenario containing 10 nodes was used, and is described in this section. The distance between the nodes is fixed to 5m. Nodes 0 and 1 periodically transmit UDP packets to node 9 at a constant rate. Unicast packets containing 1096 bytes are sent every 1s. If the packet is received, node 9 transmits an ACK to acknowledge the reception. This experiment is based on a physical layer operating at a bit rate of 6 Mbps. The simulation run executes for 10s and 20 packets are sent in total. To implement and test this scenario, we used 802.11a default configuration.

Figure 6 represents the channel access of the three main nodes (0, 1, and 9). The plots show the duration of transmission for each node (a value of 0 represents channel listening/receiving, a value of 1 represents transmission) depending on time. Node 0 and node 1 request access to the medium. Both of them send an elimination burst. Node 0 has the shortest burst. After transmitting its burst, it senses that the channel is busy. Therefore, it quits the contention. The surviving node listens to the channel during a yield period. During this interval, the channel remains idle. Node 1 wins the contention and begins its data transmission. Node 9 sends an ACK packet in unicast to acknowledge the reception of data from node 1.

At the end of the transmission, node 0 tries to gain access to the channel again. It sends a burst signal, and then it waits during a yield interval. In this scenario, there are no other contending nodes, so it sends its data to node 9. When the data is received correctly by node 9, it transmits an ACK packet.

5.3 Tracing and Logging

In addition to the features detailed above, tracing and logging functionalities have been implemented.

- Tracing : The main goal of a simulation is to generate traces for study. To analyse the performance of our module and check state of the contending nodes, a tracing system has been implemented. The added tracing sources may enable generating information about the beginning and the end of the different phases of EY-NPMA scheme. It also enables tracing the number of idle and busy slots chosen by each node. Given that DcaTxop class handles the main functionalities and features of EY-NPMA access scheme, the main addition of tracing functions has been done at this level.

- Logging functionality : the Wifi module, as most of the ns-3 modules, uses the ns-3 logging system to provide a multi-level approach to message logging. Benefiting of this feature, we have implemented logging functions to output warning messages and to enable programs debugging. The implementation of logging system follows the general design of ns-3 logging

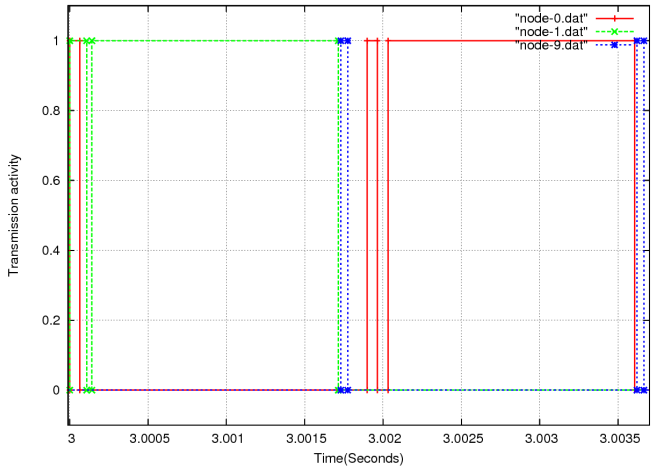


Figure 6: Transmission activity of each node w.r.t time

system. Different levels of logging have been used. Each log level can be requested using logging function call, or by setting the global shell environment variable `NS_LOG`.

6 Simulation

To evaluate the performances of our module, we created different simulation scripts. For each scenario tested, we compare the results of our Ey-Wifi module to Wifi module. We evaluate these aspects using specially designed scenarios.

6.1 Scenario 1 : Unicast

In this section, we focus on point to point transmissions in a one hop network. The scenario is composed of 100 nodes arranged in a grid. Each node sends a packet to a random neighbor. Packet size is fixed at 1000 bytes. The maximum number of packets sent by each node for 10 seconds of simulation is 20 packets. We vary the density of the network and we compute two metrics: the number of packets correctly received, and the number of lost packets. The simulation results are presented in figures 7 and 8.

The results show that for Ey-Wifi based scenarios the rate of packet loss is very close to zero. All sent packets are correctly received by the recipients. This is unlike Wifi, for which loss rate is about 8%. This shows a first important property of Ey-Wifi: the EY-NPMA method reduces collisions among packets with similar priority.

6.2 Scenario 2 : Broadcast

The broadcast scenario consists of 30 nodes which are arranged in a grid. The distance between the nodes is set to 1m. This simulation uses a physical layer to 6Mbps. Nodes periodically transmit packets of 1000 bytes. The total simulation time is 10 seconds. In this simulation, all the nodes are in range of each other. Packet rate is increased until saturation (which takes place when the interval is set around 40 ms) is varied. The metric shown in figure 9 is the total number of packets received per unit of time. The results show a second important property

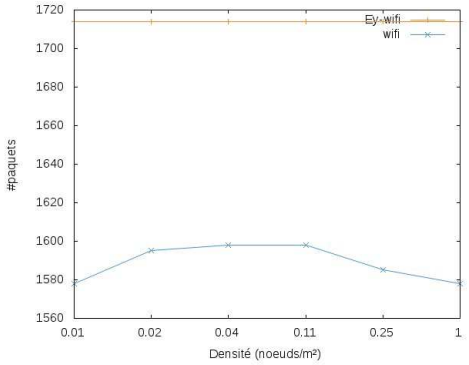


Figure 7: Packet rate

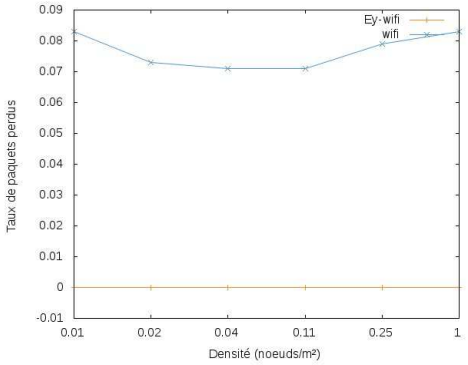


Figure 8: Packet loss

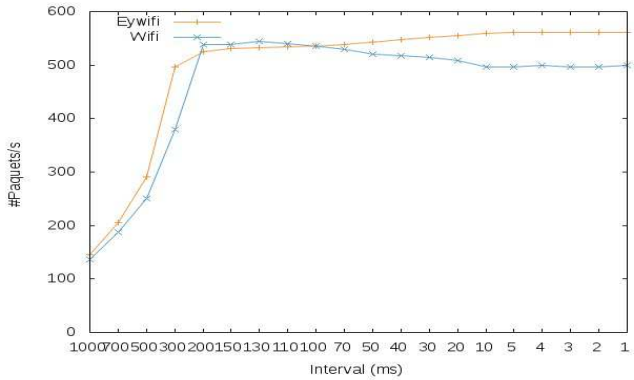


Figure 9: Broadcast : Packet rate

of Ey-Wifi: at saturation, the throughput of Ey-Wifi is higher than Wifi, even if EY-NPMA scheme has more overhead. This illustrates the fact that EY-NPMA scheme can compensate its overhead, and prove more efficiency in the context of saturated networks.

6.3 Scenario 3 : Flooding

The goal of the third scenario is to evaluate the EY-NPMA method in flooding context and to compare it to Wifi based networks. The network consists of 2000 nodes which are arranged in a grid (4×500). A node in the middle of the network sends a packet of 1000 bytes every second. Each node receiving this packet, retransmits it once to its neighbors.

We vary the density of the network and compute two metrics: reception rate (fig. 10) and the number of received packets (fig. 11). The reception rate here represents the percentage of nodes that have received correctly the packet sent by the source. The results show that in EY-NPMA based networks, all nodes in the network receive the packets regardless of the network density. However, the good reception of the packet is not guaranteed for all the nodes in a wireless network. The second metric is the number of packets received. The results show that the total number of packets received during the simulation is much higher than Wifi. This once again proves the effectiveness of the EY-NPMA method in reducing collisions.

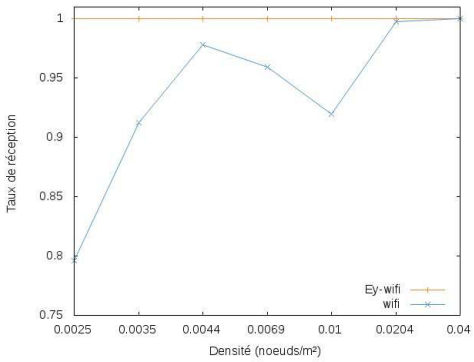


Figure 10: Reception rate

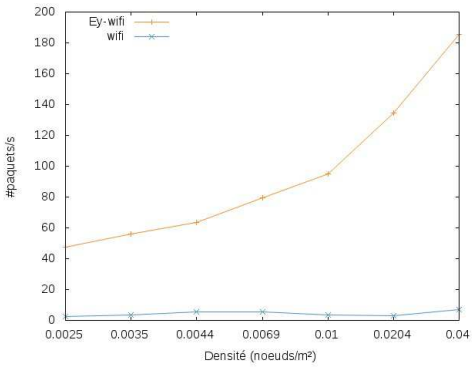


Figure 11: Packet rate

7 Conclusion

This paper detailed the design and implementation of Ey-Wifi, an EY-NPMA module for the ns-3 simulator. There are few suitable simulation environments for EY-NPMA protocol. The purpose of Ey-Wifi module is to facilitate the evaluation and the design of scenarios for EY-NPMA based wireless networks. Based on existing Wifi implementation, this module implements key components of EY-NPMA channel access protocol. Benefiting from the features of ns-3, this module may enable emulation, and implementation of new scenarios.

We have presented also our simulation results on the two standard acces methodes EY-NPMA and 802.11(DCF). We have been focusing on the performance for different densities and under different load conditions. According to the results, both protocols ensure satisfying performance in general conditions. However, compared to WiFi, Ey-Wifi provides lower collision rates.

The Ey-Wifi module has been implemented and tested using 802.11a parameters. The present module is sufficiently generic to be used with any 802.11 protocol implemented in ns-3. The proposed module Ey-Wifi implements the different phases of EY-NPMA channel access protocol, but it still have some limitations. As explained before, the burst is designed as a data packet. We consider implementing the burst signal as pseudo noise rather than transmitting data bits. Work to implement burst signal on the physical layer of ns-3 is envisaged.

8 Appendix 1 : Download and Build

You can download the Ey-Wifi module and build it using bake. "bake" is a tool for distributed integration and building, developed for the ns-3 project. One usual practice is to create a directory called "repos" in one's home directory under which one can keep local Mercurial repositories. You can get the bake code by typing the following instruction:

```
mkdir repos
cd repos
hg clone http://code.nsnam.org/bake
cd bake
```

We need to configure bake to download and build the Ey-Wifi module, that's why you should download the appropriate configuration file :

```
wget http://hipercom.inria.fr/Ey-Wifi/source/bakeconfEyWifi.xml
```

To configure bake type the following instruction :

```
./bake.py configure -e ey-wifi -c bakeconfEyWifi.xml
```

Next, try to download the module:

```
./bake.py download -vvv
```

should yield something like:

```
>> Searching for system dependency python-dev -
>> Search python-dev - OK
>> Searching for system dependency pygraphviz -
>> Search pygraphviz - OK \\
>> Searching for system dependency pygoocanvas -
>> Search pygoocanvas - OK
>> Download pybindgen-0.16.0.825 - OK
>> Searching for system dependency g++ -
>> Search g++ - OK
>> Searching for system dependency qt4 -
>> Search qt4 - OK
>> Downloading netanim-3.103 -
... >> Download netanim-3.103 - OK
...
>> Download ns-3.16 - OK
..
>> Downloading ey-wifi (target directory:ns-3.16) -
...
>> Download ey-wifi - OK
```

Now we are ready to build our distribution. The source code is under source/ns-3.16/ directory :

```
cd source/ns-3.16/
```

Now type ls on src directory, you should see "ey-wifi" module. The examples are not enabled by default. So, in order to run Ey-Wifi examples, you have to enable examples as follow:

```
./waf -d debug --enable-examples configure
```

Otherwise, copy the example into the scratch folder and use waf to build it just like building any ns-3 script.

```
cp src/wifi/examples/ey-wifi-example.cc scratch/ey-wifi-example.cc
```

We use waf to build ns-3.

```
./waf
```

9 Appendix 2 : Running the first example

We typically run scripts under the control of Waf. To run a program, simply use the `--run` option in `waf`. The first example is located in src/wifi/examples. Let's run it by typing the following, you should see the expected output:

```
./waf --run src/ey-wifi/examples/ey-wifi-example
Waf: Entering directory ‘/home/local/baccouch/INRIA/demo/bake/source/ns-3.16/build’
Waf: Leaving directory ‘/home/local/baccouch/INRIA/demo/bake/source/ns-3.16/build’
’build’ finished successfully (1.929s)
50000000 ns Node 10.0.0.1 broadcasts packet.
50000000 ns Node 10.0.0.2 broadcasts packet.
50000000 ns Node 10.0.0.3 broadcasts packet.
51623033 ns Node 10.0.0.1 received packet from node 10.0.0.2
51623033 ns Node 10.0.0.3 received packet from node 10.0.0.2
53468066 ns Node 10.0.0.2 received packet from node 10.0.0.3
```

The goal of the first example is to simulate three nodes. Each node broadcasts a packet of 1000 bytes periodically.
The actual code begins by module include files :

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ey-wifi-module.h"
#include "ns3/netanim-module.h"
```

The first step is to create 3 nodes that will connect via Wifi link. The NodeContainer is used to do this.

```
NodeContainer nodes;
nodes.Create(size);
```

Now we are going to set the mobility model. We use a MobilityHelper to set nodes stationnary on a grid model. First, we instanciate a MobilityHelper object. Then, we set the attributes handling the position of nodes. The distance between nodes is fixed to “step“ meters (default value of step is 10 meters).

```
MobilityHelper mobility;
mobility.SetPositionAllocator("ns3::GridPositionAllocator", "MinX",
       DoubleValue(0.0), "MinY", DoubleValue(0.0), "DeltaX",
       DoubleValue(step), "DeltaY", DoubleValue(step), "GridWidth",
       UintegerValue(5), "LayoutType", StringValue("RowFirst"));
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(nodes);
```

We create a NetDeviceContainer to keep track of the EyWifiNetDevices and we install devices on the eyWifi nodes.

```
NetDeviceContainer devices;
```

Then, we configure physical and channel helpers. We create a Ey-Wifi channel using EyYansWifiChannelHelper. Default configuration is used in this example by calling EyYansWifiChannelHelper::Default() method. The physical layer is created by calling EyYansWifiPhyHelper. As an error rate model, we used the EyYansErrorRateModel(equivalent to YansErrorRateModel), already defined and documented in the API doxygen documentation.

```
EyYansWifiChannelHelper eyWifiChannel = EyYansWifiChannelHelper::Default();
EyYansWifiPhyHelper eyWifiPhy;
eyWifiPhy.SetErrorRateModel("ns3::EyYansErrorRateModel");
```

Once these objects are created, we create the channel object and aggregate it the PHY layer, already created by the helper.

```
eyWifiPhy.SetChannel(eyWifiChannel.Create());
```

Now we focus on the MAC layer. Here we choose to work with a non QoS MAC (because QoS features from Ey-Wifi are used), so we use NqosWifiMacHelper ato set MAC parameters.

```
EyNqosWifiMacHelper eyWifiMac = EyNqosWifiMacHelper::Default();
eyWifiMac.SetType("ns3::EyAdhocWifiMac");
```

We use the WifiHelper to create and connect Wifi devices and channels.

```
EyWifiHelper eyWifi = EyWifiHelper::Default();
```

A new RemoteStationManager mode has been defined to support burst transmission. To enable active signaling in this scenario, we have to set the BurstMode on true as follow :

```
eyWifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "BurstMode", BooleanValue(true)
```

Once the physical and the mac layers are defined, we install the configuration on the NodeContainer.

```
devices = eyWifi.Install(eyWifiPhy, eyWifiMac, nodes);
```

Now, our nodes, devices and channels are created but no protocol stacks are installed. We use InternetStackHelper to install these stacks. We only need to install these stacks on the nodes container

```
InternetStackHelper stack;
stack.Install(nodes);
```

To assign IP addresses to our interfaces, we use the Ipv4AddressHelper

```
Ipv4AddressHelper address;
address.SetBase("10.0.0.0", "255.0.0.0");
interfaces = address.Assign(devices);
```

The topology is built, so we need to install applications on nodes. Now, we will create two applications for transmitting and receiving data packets. Both applications will be installed on each node of the topology already built.

```
ReceivePacket(Ptr socket);
GenerateTraffic(Ptr socket, uint32_t pktSize,uint32_t pktCount, Time pktInterval);
```

To generate traffic, we create a socket on each node. We provide the required attribute to the socket in the Send method (in this case the packetsize). Then, we schedule the transmission every second. GenerateTraffic method is defined as follow :

```
if (pktCount > 0) {
socket->Send(Create(pktSize));
Simulator::Schedule(pktInterval, &GenerateTraffic, socket, pktSize,pktCount - 1, pktInterval);
Ptr ipv4 = socket->GetNode()->GetObject();
Ipv4InterfaceAddress iaddr = ipv4->GetAddress(1, 0);
Ipv4Address addri = iaddr.GetLocal();
} else {
socket->Close();
}
```

Another socket has been created to receive data. We provide to the RecvFrom method the source Address of data. The ReceivePacket method is defined as follow :

```
Address sourceAddress;
Ptr p = socket->RecvFrom(sourceAddress);
InetSocketAddress inetSourceAddr = InetSocketAddress::ConvertFrom(sourceAddress);
Ipv4Address sender = inetSourceAddr.GetIpv4();
Ptr ipv4 = socket->GetNode()->GetObject();
Ipv4InterfaceAddress iaddr = ipv4->GetAddress(1, 0);
Ipv4Address addri = iaddr.GetLocal();
```

The last part of the code runs and cleans up the simulation, just like any ns-3 scenario.

```
Simulator::Stop(Seconds(totalTime));
Simulator::Run();
Simulator::Destroy();
```

10 Appendix 3 : Tracing and plotting graphs

The main goal of a simulation is to generate traces for study. To analyse the performance of our module and check state of the contending nodes, a tracing system has been implemented. The added tracing sources may enable generating informations about the begging and the end of the different phases of EY-NPMA scheme. It enables also tracing the number of idle and busy slots chosen by each node.

The Trace Sinks

We have provided an example showing how to use tracing in Ey-Wifi module. You can find the code in the examples folder as ey-wifi-tracing-example.cc. The main goal of this simulation is to get trace callbacks from the DcaTxop class indicating the beginning and the end of each phase (elimination and yield phases) for the same scenario, as well as the number of slots of each phase. Most of the code is explained in the section above. The next functions have been added to implement the corresponding trace sink :

```
void
TracingExample::EyWifiBurstBeginTrace(std::string context, uint32_t slots)
{
  NS_LOG_UNCOND ("Burst Begin at "<< Simulator::Now().GetSeconds() << " slots " << slots);
}
void
TracingExample::EyWifiBurstEndTrace(std::string context, uint32_t slots)
{
 NS_LOG_UNCOND ("Burst End at "<< Simulator::Now().GetSeconds() << " slots " << slots);
}
```

Once the trace sink function is defined, we have to connect it to the trace source. The trace sinks which handle the elimination phase will be connected to the "BurstTxBegin" and "BurstTxEnd" trace sources of the WifiNetDevice. These trace sources fires respectively when a burst is transmitted by the WifiNetDevice and at the end of the burst transmission.

```
  pathBurstBegin<<"NodeList/*/DeviceList/*/$ns3::EyWifiNetDevice/Phy/BurstTxBegin";
  Config::Connect(
                 pathBurstBegin.str(),
                 MakeCallback(&TracingExample::EyWifiBurstBeginTrace, this));
```

If you now build and run the example, you will see the output from the trace sink functions each time the trace source is executed:

```
./waf --run src/ey-wifi/examples/mac-tracing-example
Waf: Entering directory ‘/home/local/baccouch/INRIA/demo/bake/source/ns-3.16/build’
Waf: Leaving directory ‘/home/local/baccouch/INRIA/demo/bake/source/ns-3.16/build’
’build’ finished successfully (1.953s)
Burst Begin at 0.027 slots 5
Burst End at 0.027055 slots 5
PhyTxBegin 0.027082
PhyTxEnd 0.028526
Burst Begin at 0.042 slots 14
Burst End at 0.042135 slots 14
PhyTxBegin 0.042207
PhyTxEnd 0.043651
Burst Begin at 0.116 slots 16
Burst End at 0.116151 slots 16
```

Plotting graphs

First example

Under scratch directory, you can find some scripts to plot graphs. You can plot the activity of the channel during the simulation of our first example. You can run this script simply by typing:

```
./scratch/script-plot-tx.sh
```

You can open script-plot-tx.sh with your favorite editor, you should see the following script:

```
for i in  $(seq 0 2)
do
  ./waf --run "scratch/ey-wifi-plot-tx --nodeId=$i" > "log-node-$i.dat"
done

gnuplot scratch/plot-tx.gp
xdg-open plot-tx.jpeg
xdg-open plot-tx-zoom.jpeg
```

Now, if you run the script-plot-tx.sh program, you should see two graphs.

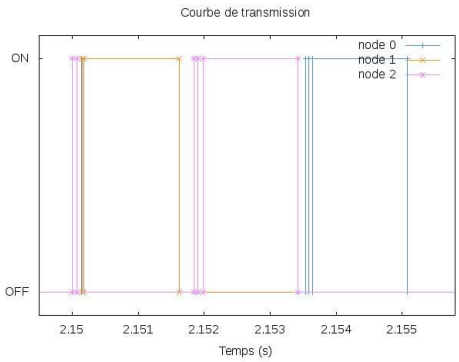


Figure 12: Transmission activity

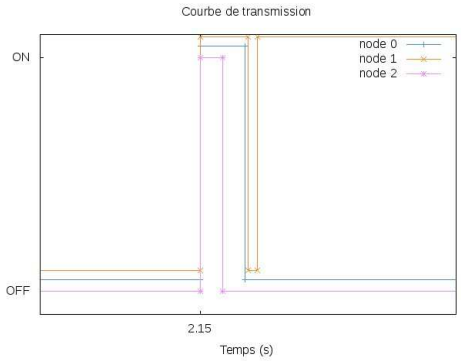


Figure 13: Transmission activity (Zoom)

The first graph shows the transmission activity of three nodes (ON represents transmission, OFF represents channel listening/receiving), during a defined interval ([[2.14,2.15]]s) The second graph is just a focus on burst transmission of three nodes.

Plot channel activity

For more complicated scenarios, we have written a python script which parses logs to plot transmission activity of many nodes. The "parse-channel.sh" script, executes a simple broadcast example (ey-wifi-broadcast-example) and parse logs to plot the activity of the nodes on channel. You can run the example, and you see immediately prints of your traces.

```
./waf --run scratch/ey-wifi-broadcast-example
Waf: Entering directory ‘/home/local/baccouch/INRIA/demo/bake/source/ns-3.16/build’
Waf: Leaving directory ‘/home/local/baccouch/INRIA/demo/bake/source/ns-3.16/build’
’build’ finished successfully (7.549s)
```

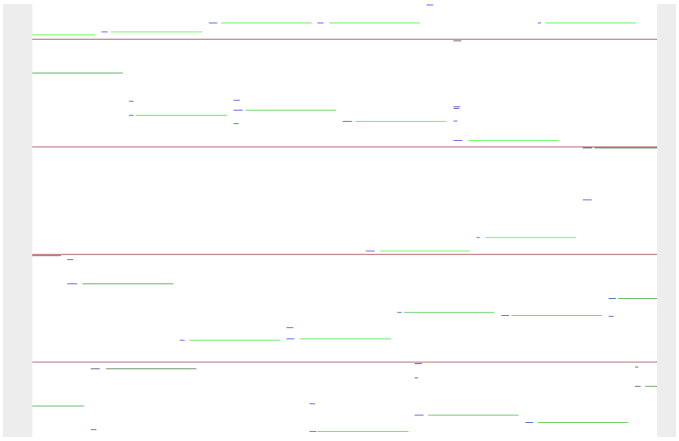


Figure 14: Channel activity

```
0 0.013 BurstBegin 4
0 0.013043 BurstEnd 4
0 0.013133 PhyTxBegin 0
0 0.014577 PhyTxEnd 0
167 0.019 BurstBegin 7
167 0.019071 BurstEnd 7
167 0.019152 PhyTxBegin 2
167 0.020596 PhyTxEnd 2
131 0.025 BurstBegin 13
162 0.025 BurstBegin 5
162 0.025055 BurstEnd 5
...
```

Let's redirect this output to a file named "channel-log.dat". The "EyWifiChannel.py" script will parse this log file and plot the channel status during the simulation. Since we have provided the "parse-channel.sh" script, you can edit the file. It looks like :

```
./waf --run scratch/ey-wifi-broadcast-example 2> "channel-log.dat"
#Parse logs and plot channel
python scratch/EyWifiChannel.py
okular ey-wifi-channel.png
```

You have just to run it, and the result should be as 14.
If you intend to parse your log files using our parser, note that your logs should looks like the "channel-log.dat".

Broadcast Example

The goal of the first example is to evaluate the performances of Wifi and Ey-Wifi in broadcast context. The "script-ey-wifi-broadcast.sh" program executes the same broadcast scenario using Ey-Wifi and Wifi. The scenario consists of 25 nodes arranged in a grid (5×5). The distance between the nodes is fixed to 5m. Nodes transmit periodically packets containing 1000 bytes. The simulation run executes for 10s.

You can run the ey-wifi-broacast.cc and see immediately the traces printed in your terminal. You can redirect that output to a specific file for further use. That is exactly what we do in our script "script-ey-wifi-broadcast.sh". You can now open the script in your favorite editor:

```
INTERVAL="1000 500 300 200 150 100 40 20 10 5 4 1"
for inter in  $INTERVAL
do
    eywifi=$(./waf --run "scratch/ey-wifi-broadcast --interval=$inter")
    plotEyWifi="$plotEyWifi$eywifi\n"
    eywifi=""

    wifi=$(./waf --run "scratch/wifi-broadcast --interval=$inter")
    plotWifi="$plotWifi$wifi\n"
    wifi=""
done
echo -e $plotEyWifi > eywifi-results.tr
echo -e $plotWifi > wifi-results.tr
```

As you can see, the results of both simulation for Ey-Wifi and Wifi based networks are respectively redirected to "eywifi-results.tr" and "wifi-results.tr" Now you can run gnuplot and tell it to generate graphs.

```
gnuplot "scratch/plot-broadcast.gp"
```

Gnuplot script is already written in plot-broadcast.gp and it is as follow :

```
set terminal jpeg medium 8
set ylabel "#packets"
set xlabel "Interval"
set output "broadcast-packets-received.jpeg"
plot 'eywifi-results.tr'  using 4:xticlabels(1) t "Ey-wifi" w lp linecolor rgb "#f68e2f",
'wifi-results.tr' using 4:xticlabels(1) t "wifi" w lp linecolor rgb "#4F94CD"
```

Now, you can simply run the following script to simulate the broadcast scenario and generate the graph:

```
./scratch/script-ey-wifi-broadcast.sh
```

It should looks like figure 15.

Flooding Example

The second example intend to show the performance of the flooding example for both protocols Ey-Wifi and Wifi. The used scripts are under scratch directory : ey-wifi-flooding-example.cc and wifi-flooding-example.cc. To run both programs and plot the results, you should simply execute the "script-ey-wifi-flooding.sh" program.

```
./scratch/script-ey-wifi-flooding.sh
```

This script executes the ey-wifi-flooding-example and wifi-flooding-example for different values of inter packets interval firstly, then it executes them by varying the density of nodes. Figures 11 and 10 show the graphs that you should obtain when you execute the above script.

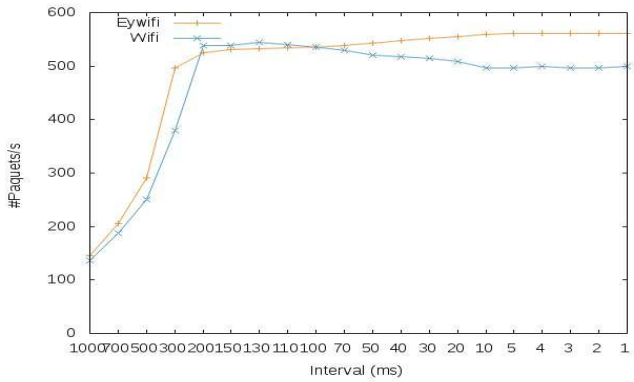


Figure 15: Broadcast : Packet rate

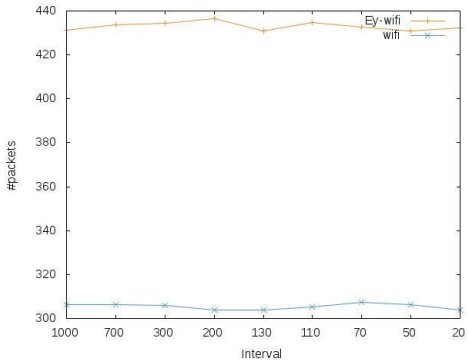


Figure 16: Flooding : Packet rate varying inter packet interval

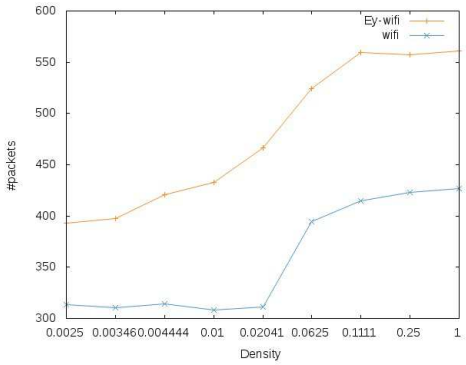


Figure 17: Flooding : Packet rate varying density

Unicast Example

The goal of the third example is to compare the performance of both Ey-Wifi and Wifi based networks in unicast context. The scenario is composed of 100 nodes are arranged in grid. The step is fixed to 5 meters. The simulation run for 10s. The used script is "script-ey-wifi-unicast.sh". This script executes ey-wifi-unicast-example and wifi-unicast-example. As flooding example, we compute the number of received packets by varying firstly the inter-packet interval. Then we increase the density and compute the same metric. You can run the script, it will take some time to plot two graphs. It should look like figures 18 and 19.

Acknowledgments

We would like to thank Yasser Toor and Anis Laouiti for contributing with their EY-NPMA work. We would also like to thank Mathieu Lacage, Daniel Camara, Walid Dabbous and Thierry Turletti for their help and suggestions during the development of this work.

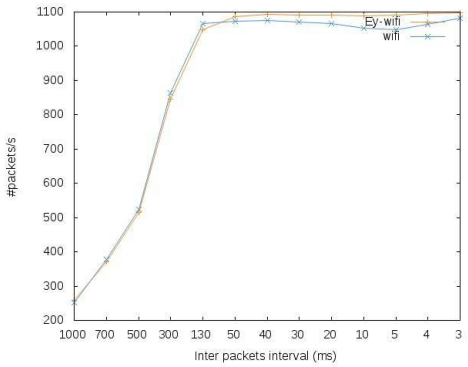


Figure 18: Unicast : Packet rate varying inter packet interval

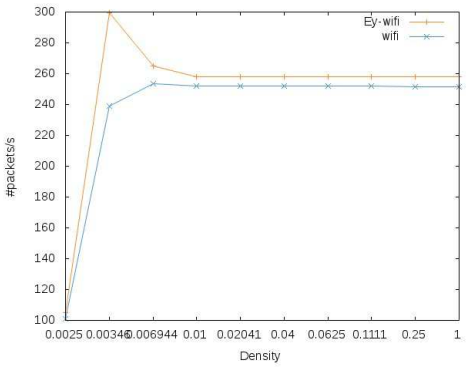


Figure 19: Unicast : Packet rate varying density

References

[1] EN 300 652 V1.2.1 (1998-07) *Broadband Radio Access Networks (BRAN) ; High PErformance Radio Local Area Network (HIPERLAN) Type 1* ; Functional specification, European Standard (Telecommunications series), ETSI. July 1998.

[2] IEEE 802.11 WG, "*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,*" IEEE Std. 802.11-2007 (Revision of IEEE Std. 802.11-1999), 2007.

[3] IEEE Std. 802.11 WG, "*Amendment to Standard for Information Technology. LAN/MAN Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS),*" Nov. 2005.

[4] Philippe Jacquet, Pascale Minet, Paul Muhlethaler, and Nicolas Rivierre, "*Priority and Collision Detection with active Signaling: The Channel Access Mechanism of HIPERLAN,*", Wireless Personal Communications Vol 4, No 1, pp. 11-25, 1997.

[5] P. Muhlethaler, Y. Toor and A. Laouiti, "*Suitability of HIPERLAN's EY-NPMA for traffic jam scenarios in VANETs,*", ITST '10, Kyoto, Japan.

[6] A. Qayyum. *Analysis and evaluation of channel access schemes and routing protocols in wireless LANs.* PhD thesis, University of Paris-Sud, 2000.

[7] Lacage, M. and Henderson, T.R. *Yet another network simulator*, ACM Proceeding from the 2006 workshop on ns-2: the IP network simulator.

[8] Khosroshahy, M. and Turletti, T. and Obraczka, K. and others *Snapshot of MAC, PHY and Propagation Models for IEEE 802.11 in Open-Source Network Simulators*, 2007

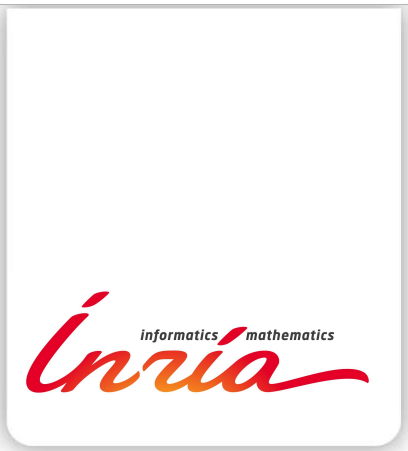
[9] M. Lacage *Experimentation Tools for Networking Research* PhD thesis, University of Nice-Sophia Antipolis, November 2010.

[10] T. Bingmann. *Accuracy Enhancements of the 802.11 Model and EDCA QoS Extensions in ns-3.* PhD thesis, University of Karlsruhe, 2009.

[11] D. Dhoutaut. *Etude du standard IEEE 802.11 dans le cadre des reseaux ad hoc : de la simulation a l'experimentation.* PhD thesis, CITI Laboratory, INSA - Lyon, 2003

[12] The ns-3 network simulator available at , *http://www.nsnam.org/* , 2012

[13] The Ey-Wifi module at , *http://hipercom.inria.fr/Ey-Wifi* , 2013



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399