yawmd: multiple medium support and performance improvements for wmediumd

Miguel José Meireles Moreira
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2019/2020

Orientador
Rui Pedro de Magalhães Claro Prior, Professor Auxiliar, FCUP

Coorientador
Eduardo Filipe Amaral Soares, Estudante de Doutoramento, FCUP

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /

Acknowledgements

I want to thank my advisors, Rui Prior and Eduardo Soares, for the suggestions and the help provided over the course of the year. I also want to thank my friends and family, for they support, friendship and belief, essential for this work to be brought to fruition.

Abstract

Wireless networks, comparatively to wired networks, suffer from higher error rates, faster signal attenuation and more interference. These effects can significantly limit the performance of these networks, and for some applications, they can not be ignored when simulating communication using wireless links, if there is the expectation of obtaining accurate results. Network simulators, network emulators and testbeds are tools primarily used for research, software development and to test deployments. These tools offer different levels of abstraction of real network components, which provide a cost reduction and a more practical comparison to real environments. The focus of this thesis is wmediumd, a wireless medium simulator used in the network emulator Mininet-WiFi. Wmediumd works in conjunction with mac80211_hwsim, a device driver for Linux that can simulate multiple IEEE 802.11 interfaces. The objectives of this thesis are to find areas where performance of wmediumd may be improved, find ways to increase its scalability, and implement the solutions found.

Keywords: yawmd, wmediumd, mac80211_hwsim, Mininet-WiFi, performance, network emulator, simulation

Resumo

Redes sem fios, comparativamente às redes com fio, sofrem the maiores taxas de erro, atenuação de sinal mais rápida e mais interferência. Estes efeitos podem limitar significativamente o desempenho destas redes, e, para algumas aplicações, eles não podem ser ignorados na simulação de comunicação através de ligações sem fios, se se espera obter resultados precisos. Simuladores de rede, emuladores de rede e bancos de ensaio são ferramentas usadas primariamente para pesquisa, desenvolvimento de programas e para testar implantações. Estas ferramentas oferecem diferentes níveis de abstração dos componentes de rede reais, o que proporciona uma redução de custos e uma comparação mais prática com ambientes reais. O foco desta dissertação é o wmediumd, um simulador de meio sem fios usado no emulador de rede Mininet-WiFi. O wmediumd trabalha em conjunto com o mac80211_hwsim, um controlador de dispositivo para Linux que pode simular múltiplas interfaces IEEE 802.11. Os objetivos desta dissertação são encontrar áreas onde o desempenho do wmediumd pode ser melhorado, encontrar formas de aumentar a escalabilidade, e implementar as soluções encontradas.

Palavras chave: yawmd, wmediumd, mac80211_hwsim, Mininet-WiFi, desempenho, emulador de rede, simulação

Contents

Acknowledgements **i**

Abstract **iii**

Resumo **v**

List of Tables **xi**

List of Figures **xv**

List of Listings **xvii**

Acronyms **xix**

1 Introduction **1**

2 State of the art **5**

2.1 Mininet-WiFi 5
2.2 mac80211_hwsim 6
2.3 wmediumd 7
2.3.1 Overview of the operation of mac80211_hwsim and wmediumd 8
2.3.2 Configuration options 8
2.3.3 Simulation 9

3 Performance analysis of wmediumd **17**

3.1 Test 1: Determination of throughput 17
3.1.1 Configuration 18
3.1.2 Results 20
3.1.3 Results analysis 22
3.2 Test 2: Behaviour with a fixed offered load 24
3.2.1 Configuration 24
3.2.2 Results 24
3.2.3 Results analysis 25
3.3 Profiling wmediumd 26

4 yawmd – Yet Another Wireless Medium Daemon **29**
4.1 yawmd v1: Improve the communication with mac80211_hwsim 29
4.2 Multiple simulated mediums: yawmd mediums and yawmd pthreads 30
4.2.1 yawmd mediums: Events for each medium 32
4.2.2 yawmd pthreads: One thread per medium 33

5 Performance analysis of yawmd **35**
5.1 Test 3: One medium 35
5.1.1 Configuration 36
5.1.2 Results 37
5.1.3 Results analysis 38
5.2 Test 4: Multiple mediums 40
5.2.1 Configuration 40
5.2.2 Results 40
5.2.3 Results analysis 42
5.3 Results summary and final considerations 43

6 Conclusions **45**
6.1 Future Work 46

Bibliography **48**

A Test 1 complementary plots **49**
A.1 Variation H 49
A.2 Variation W 51

B Test 2 complementary plots **53**
B.1 Variation H 53
B.2 Variation W 54

C Configuration files for wmediumd and yawmd **57**
C.1 wmediumd configuration file options 57
C.2 yawmd configuration file options 59

D Medium access simulation simultaneous transmission **63**

E Commands used **65**

F Test 3 and Test 4 complementary information **67**
F.1 Table summaries of the data used in chapter 5 67
F.1.1 Test 3: One medium 67
F.1.2 Test 4: Multiple mediums 71
F.2 Plots of medium usage by QoS queue 76

F.2.1 Test 3: One medium 76
F.2.2 Test 4: Multiple mediums 78
F.3 Plots of medium usage by QoS queue with overlapping 80
F.3.1 Test 3: One medium 80

List of Tables

3.1 Average cumulative throughput in test 1 with *variation H* (mac80211_hwsim). . . 20
3.2 Average cumulative throughput in test 1 with *variation W* (wmediumd). 21
3.3 Cumulative target bitrates for test 2. 25
3.4 Results of test 2. 25

D.1 Example of a timeline of frame arrival to show the overlapping transmissions problem in the medium access simulation. 64

F.1 Results of wmediumd by QoS queue in test 3. 67
F.2 Results of yawmd v1 by QoS queue in test 3. 68
F.3 Results of yawmd mediums by QoS queue in test 3. 68
F.4 Results of yawmd pthreads by QoS queue in test 3. 69
F.5 Results of wmediumd CPU and memory usage in test 3. 69
F.6 Results of yawmd v1 CPU and memory usage in test 3. 70
F.7 Results of yawmd mediums CPU and memory usage in test 3. 70
F.8 Results of yawmd pthreads CPU and memory usage in test 3. 71
F.9 Results of yawmd mediums by QoS queue in test 4, with $S = 10$. 71
F.10 Results of yawmd mediums by QoS queue in test 4, with $S = 20$. 72
F.11 Results of yawmd pthreads by QoS queue in test 4, with $S = 10$. 72
F.12 Results of yawmd pthreads by QoS queue in test 4, with $S = 20$. 73
F.13 Results of yawmd mediums CPU and memory usage in test 4, with $S = 10$. . . . 73
F.14 Results of yawmd mediums CPU and memory usage in test 4, with $S = 20$. . . . 74
F.15 Results of yawmd pthreads CPU and memory usage in test 4, with $S = 10$. . . . 74
F.16 Results of yawmd pthreads CPU and memory usage in test 4, with $S = 20$. . . . 75

List of Figures

2.1 High level view of the interaction between Mininet-WiFi, wmediumd and mac80211_hwsim. 6
2.2 Fork tree of wmediumd. All these versions maintained the name wmediumd. . . 8
2.3 Workflow of wmediumd. 10
2.4 Steps to handle the two types of events in the event loop of wmediumd. 10
2.5 Netlink messages exchanged between mac80211_hwsim and wmediumd to perform the simulation of delivery of one frame. 11
2.6 Steps of the delivery of a frame. 12

3.1 Example of the startup order of the iperf clients in test 1 and test 2. 19
3.2 Average cumulative throughput in test 1 for each instance N with *variation H*. . 21
3.3 Average cumulative throughput in test 1 for each instance N with *variation W*. . 21
3.4 Cumulative throughput in test 1 of the instance with 20 stations with *variation H*. 22
3.5 Cumulative throughput in test 1 of the instance with 20 stations with *variation W*. 22
3.6 Cumulative throughput in test 1 of the instance with 40 stations with *variation H*. 22
3.7 Cumulative throughput in test 1 of the instance with 40 stations with *variation W*. 22

4.1 Messages exchanged between yawmd and mac80211_hwsim to perform the delivery simulation of a frame. 31
4.2 Evolution of the versions of yawmd. 32
4.3 Events managed by the event loop in yawmd mediums. 33
4.4 Workflow of yawmd pthreads. 34
4.5 Events managed by the *main thread* and the *medium threads* event loops in yawmd pthreads. 34

5.1 Average of the *medium utilization* at each second during each instance of test 3. 38
5.2 Average of the CPU used at each second during each instance of test 3. 39
5.3 Average of the memory used at each second during each instance of test 3. . . 40
5.4 Example of the setup used in test 4. 41
5.5 Average of the mediums average *medium utilization* at each second during each instance of test 4. 41
5.6 Average of the CPU use at each second during each instance of test 4. 42

5.7 Average of the memory use at each second during each instance of test 4. . . . 43

A.1 Throughput of test 1 during *variation H* with 2 stations. 49
A.2 Cumulative throughput during test 1 in *variation H* with 4 stations. 49
A.3 Cumulative throughput during test 1 in *variation H* with 10 stations. 49
A.4 Cumulative throughput during test 1 in *variation H* with 20 stations. 49
A.5 Cumulative throughput during test 1 in *variation H* with 40 stations. 50
A.6 Cumulative throughput during test 1 in *variation H* with 60 stations. 50
A.7 Cumulative throughput during test 1 in *variation H* with 80 stations. 50
A.8 Throughput of the last pair of test 1 during *variation H* with 80 stations. 50
A.9 Throughput during test 1 in *variation W* with 2 stations. 51
A.10 Cumulative throughput during test 1 in *variation W* with 4 stations. 51
A.11 Cumulative throughput during test 1 in *variation W* with 10 stations. 51
A.12 Cumulative throughput during test 1 in *variation W* with 20 stations. 51
A.13 Cumulative throughput during test 1 in *variation W* with 40 stations. 52
A.14 Cumulative throughput during test 1 in *variation W* with 60 stations. 52

B.1 Packet loss ratio during test 2 in *variation H* with 2 stations. 53
B.2 Packet loss ratio during test 2 in *variation H* with 4 stations. 54
B.3 Packet loss ratio during test 2 in *variation W* with 20 stations. 54
B.4 Packet loss ratio during test 2 in *variation W* with 40 stations, with target data rate *mean 75*. 55
B.5 Packet loss ratio during test 2 in *variation W* with 40 stations, with target data rate *mean 90*. 55
B.6 Packet loss ratio during test 2 in *variation W* with 60 stations, with target data rate *mean 75*. 56
B.7 Packet loss ratio during test 2 in *variation W* with 60 stations, with target data rate *mean 90*. 56

D.1 Representation of the order of delivery of packages of the situation described in table D.1, which shows overlapping frame transmissions. 64

F.1 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3 with wmediumd. 76
F.2 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3 with yawmd v1. 76
F.3 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3 with yawmd mediums. 77
F.4 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3 with yawmd pthreads. 77

F.5 Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance of test 4 with yawmd mediums, with $S = 10$. 78
F.6 Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance of test 4 with yawmd mediums, with $S = 20$. 78
F.7 Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance of test 4 with yawmd pthreads, with $S = 10$. 79
F.8 Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance of test 4 with yawmd pthreads, with $S = 20$. 79
F.9 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3, using wmediumd without the patch to prevent overlapping transmissions. 80
F.10 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3, using yawmd v1 without the patch to prevent overlapping transmissions. 80
F.11 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3, using yawmd mediums without the patch to prevent overlapping transmissions. 81
F.12 Average of the *medium utilization* by QoS queue, at each second, during each instance of test 3, using yawmd pthreads without the patch to prevent overlapping transmissions. 81

List of Listings

2.1 Pseudocode of the frame transmission simulation in the procedure `queue_frame()` of wmediumd. 13
2.2 Pseudocode of the algorithm used for medium access simulation in the procedure `queue_frame()` of wmediumd. 15

3.1 Results of profiling wmediumd. 27

C.1 Configuration file options for wmediumd. 57
C.2 Configuration file used by yawmd mediums and yawmd pthreads. 59

Acronyms

ACK Acknowledgment

ACM Association for Computing Machinery

CCA Clear Channel Assessment

CPU Central Processing Unit

IBSS Independent Basic Service Set

IEEE Institute of Electrical and Electronics Engineers

MAC Medium Access Control

OS Operating System

PHY Physical layer

QoS Quality of Service

RSS Resident Set Size

SNR Signal to Noise Ratio

TCP Transmission Control Protocol

UDP User Datagram Protocol

Wi-Fi IEEE 802.11

yawmd Yet Another Wireless Medium Daemon

Chapter 1

Introduction

Wireless networks, comparatively to wired networks, suffer from higher error rates, faster signal attenuation, more interference, among others. These effects can significantly limit the performance of these networks, and for some applications, they can not be ignored when simulating communication using wireless links, if there is the expectation of obtaining accurate results.

Network simulators, network emulators and testbeds are tools primarily used for research, software development and to test deployments. These tools offer different levels of abstraction of real network components, which provide a cost reduction and more practical comparison to real environments, for different needs.

Mininet [7] is a network emulator that allows the creation of a network of virtual hosts, switches, controllers and links on a Linux machine. An extension of Mininet, Mininet-WiFi [2], adds support to wireless communications, by using virtualized Wi-Fi stations and access points. A fundamental component for Mininet-WiFi is the device driver mac80211_hwsim. This driver permits the simulation of multiple IEEE 802.11 radios in a single machine. These radios have the particularity of being exposed as real network interfaces to Linux, opening the possibly to use these simulated interfaces as if they were real ones, without need for special modifications to the programs in upper network layers. The transmission of information that, in real interfaces, would happen through electromagnetic waves, is also simulated in a simplistic way: all interfaces in the same frequency channel receive copies of the information transmitted by an interface. This way, relevant information to an interface is processed, and irrelevant information is discarded.

While for some applications mac80211_hwsim's simulated inter-interface information transmission is appropriate, others require a more realistic simulation of these mechanisms. For these, there are simulators such as wmediumd, which works together with mac80211_hwsim to provide a more realistic simulation of transmission of information between interfaces. This simulation takes into consideration radio signal propagation limitations, as well as the characteristics and limitations of the mechanisms implemented in real interfaces to work with the medium.

Another characteristic of the simulations in mac80211_hwsim and wmediumd is that they must be carried out in real-time. Because of this, overheads in processing can cause the throughput (traffic effectively received) to be lower than what the simulation restrictions determine, due to processing limitations. The primary objective of this work was to reduce overheads in order to improve the performance and scalability of wmediumd. There were no particular application requirements that were being sought with these improvements, but many possible uses of Mininet-WiFi with wmediumd, or just wmediumd can benefit from larger instances and smaller resource usage. For example [18], which uses wmediumd to evaluate the performance of routing protocols in wireless mesh networks in a scenario of a natural disaster to restore communications, is a good example of a situation where it is interesting to test with high numbers of interfaces.

To achieve the objective of improving performance and scalability, it is required an understanding of the simulation that wmediumd carries out, on what information relies, and the way it interacts with mac80211_hwsim and Mininet-WiFi. Results from [1, chapter 4] seem to indicate that wmediumd shows a slow throughput as well as scalability problems. To confirm and to analyse these results more in depth, more tests were made in the context of this thesis. The results, exposed and discussed in chapter 3, confirmed the previous work results, and pointed to the communication between wmediumd and mac80211_hwsim being a probable source. This directed the efforts to analyse this communication, were was found room for improvement.

A more complete and detailed list of the initial objectives is presented now:

1. Explore and test options for communication between Linux kernel-space and user-space:
    (a) Implement prototypes for the options adequate to the simulation necessities;
    (b) Implement in wmediumd and mac80211_hwsim the best determined communication option;
2. Create tests for multiple configurations of wmediumd and, in particular, for the communication between mac80211_hwsim and wmediumd;
3. Document:
    (a) wmediumd simulation requirements (of data and timing);
    (b) Changes made and their implications;
    (c) wmediumd and mac80211_hwsim limitations;
    (d) wmediumd and mac80211_hwsim communication and simulation architecture;
4. Reduce the load related to communication between wmediumd and mac80211_hwsim;
5. Improve the traffic delivery capacity of wmediumd;
6. Maintain inter-operability with Mininet-WiFi.

The initial objectives were mainly directed to improving the communication between mac80211_hwsim and wmediumd, due to an incorrect understanding that it was the main factor limiting the capacity of the medium simulated by wmediumd. This was later found not to be correct, so no communication options between mac80211_hwsim and wmediumd were explored beyond improving what was already available. The focus was then directed to providing multiple mediums, also identified in [1] as an area needing improvement, replacing the objective 1 (testing other communication options).

The contributions of this thesis are the following:

- a more efficient and scalable communication protocol with mac80211_hwsim;
- two approaches to simulate multiple mediums, one with a single thread, and another with a thread for the simulation of each medium;
- documentation of the operation and simulation of wmediumd and of its interaction with mac80211_hwsim.

The remaining of this thesis is structured as follows: chapter 2 provides an explanation of the operation of Mininet-WiFi, wmediumd and mac80211_hwsim, and details about the simulation performed in wmediumd; chapter 3 describes tests to wmediumd and mac80211_hwsim, and provides an analysis of the results used to guide the changes to wmediumd; in chapter 4 explains the changes proposed, named yawmd, to overcome the problems of performance and scalability of wmediumd; in chapter 5 a new performance analysis to the changes introduced by yawmd and a comparison with wmediumd is presented; and finally chapter 6 provides the conclusions.

Chapter 2

State of the art

Network simulators, network emulators and testbeds are tools mainly used to reduce costs and simplify testing or development of network protocols, algorithms and applications. They provide different levels of abstraction and functionality to better fit different needs.

A network simulator is a software that abstracts a network or parts of a network, such as a device or a protocol stack with models of their behaviour. The models may accept parameters to control characteristics of the model, such as delay and packet loss. These simulators provide repeatability and a cheap and controlled environment for experiments but are limited to the accuracy of the abstractions and are unable to interact with real implementations.

A network emulator builds upon the benefits of a network simulator by extending its ability with the capacity to operate with real workloads. The simulated components are realistic enough to interact with real implementations without adaptations. A network emulator can effectively replace part of a real network, and the real components interact seamlessly with the emulated components. Network emulators must run in real time, while network simulators can run faster or slower, according to the needs and demands of the simulation.

A testbed is made of real devices, with real operating systems and applications. They can provide the most realistic results, if the setup environment can provide the relevant characteristics. Testbeds tend to be the more expensive of these three tools, because of the cost of the devices, their software and physical configuration, the costs of the location, and the higher difficulty of access to different users.

In this chapter, the focus is the network emulator Mininet-WiFi, especially the programs it uses to perform the network traffic delivery simulation, wmediumd, and to simulate the IEEE 802.11 (Wi-Fi) radios, mac80211_hwsim.

2.1 Mininet-WiFi

Mininet-WiFi [2] is an extension to the network emulator Mininet [7], which provides support for IEEE 802.11 (Wi-Fi) networks. Mininet allows the creation of a network of virtual hosts, switches, controllers and links in a Linux machine. To do this, it uses network namespaces, a lightweight

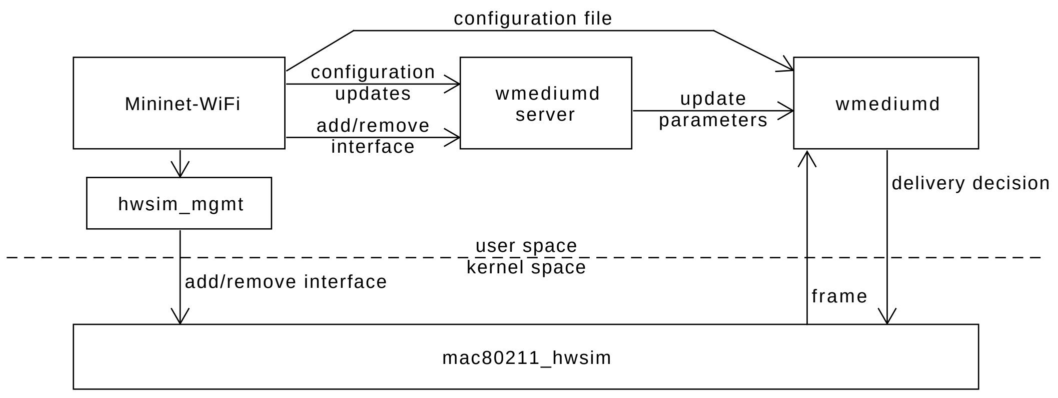

Figure 2.1: High level view of the interaction between Mininet-WiFi, wmediumd and mac80211_hwsim.

virtualization mechanism provided by the Linux kernel, which allows the existence of multiple network stacks. Processes in one network namespace can only see the network resources in that namespace, therefore by having all resources related to a device (such as a host) in one namespace, with several network namespaces, it is possible to have several devices.

Mininet-WiFi adds support to wireless communications, with virtualized Wi-Fi stations and access points, provided by mac80211_hwsim, a device driver capable of simulating an arbitrary number of IEEE 802.11 radios. While maintaining support to Mininet's functionalities, Mininet-WiFi adds support to simulation of movement and signal propagation effects, characteristic of mobile stations.

In order to simulate signal propagation, Mininet-WiFi can use Traffic Control to configure the kernel packet scheduler, or use wmediumd, to which mac80211_hwsim is capable of delegating the traffic delivery simulation, becoming wmediumd's responsibility to decide the details of the delivery of each frame. This work focuses in the operation of the wmediumd.

Mininet-WiFi and wmediumd can communicate through a server started by wmediumd. This server allows the dynamic addition and deletion of interfaces, as well as to update wmediumd's internal simulation parameters (see figure 2.1). This way, Mininet-WiFi can have its own internal simulations, such as movement and signal propagation, and only update wmediumd's simulation parameters, which will then affect the frame delivery decisions.

2.2 mac80211_hwsim

Mac80211_hwsim is a kernel module that simulates an arbitrary number of IEEE 802.11 radios for Linux. It was initially developed by Jouni Malinen to test mac80211 and is maintained and comes with the Linux kernel [15].

Mac80211_hwsim's simulated radios are seen by the operating system as real devices, which allows the use of standard network commands to configure the simulated interfaces. By default, all radios receive all the frames transmitted by any radio in the same frequency channel. No simulation of the wireless medium constraints (such as signal propagation delays or interference), nor hardware constraints (such as signal to noise ratio), nor signal modulation constraints (such as bitrate) exist.

Mac80211_hwsim can also be used together with wmediumd, in which mac80211_hwsim hands over the control of the frame by frame delivery decision, that is, wmediumd becomes responsible for informing mac80211_hwsim which interfaces, if any, are supposed to receive a frame.

2.3 wmediumd

Wmediumd is a user-space simulator for mac80211_hwsim that performs per frame delay and delivery decision. This is attained by simulating some CSMA/CA mechanisms, as well as transmission errors due to wireless medium conditions.

Wmediumd was originally developed by Cozybit[1], which later ceased operation and "decided to move on to other adventures"[2]. Since then multiple persons did modifications based on either the original wmediumd developed by Cozybit, or on previous modifications of the original (see figure 2.2). Cozybit's wmediumd has two major modifications. One by Alberto Illán that expanded the probability based simulation with path loss and mobility simulation [4], and another by Bob Copeland [3], with extensive changes to the code and simulation, which most notoriously added more types of simulation models, adding options beyond the original probability models for frame delivery decisions. The work of Bob Copeland was the basis for the work of Patrick Große [4] which added the possibility of external configuration during runtime. Mininet-WiFi's [5] wmediumd [16] is based in the work of Patrick Große, which, among others, added support to more path-loss models, made some updates to the communication with mac80211_hwsim, and made some adaptations to work with new Mininet-WiFi features.

A review of the literature with search keywords "wmediumd OR mac80211_hwsim" in IEEE Xplore[6] with full-text search and up to the end of 2019, returned 19 results. Of these, 15 correspond to works that used either mac80211_hwsim or both, and 4 are references to related or future work. Of the 15, 6 used Mininet-WiFi with wmediumd as the simulator, 5 only used mac80211_hwsim, and 4 used wmediumd. Some of these works made adaptations to fit their specific needs. Some works [6, 13, 14] correspond to emulators, but provide no relevant information to the type of changes intended to wmediumd. The same search was made in ACM digital library[7], which returned 2 results, both relative to works also published in IEEE. [4] is the only published work found whose topic was changes to wmediumd. As previously referred, this

[1]`https://github.com/cozybit/wmediumd`
[2]`http://cozybit.com/`
[3]`https://github.com/bcopeland/wmediumd/`
[4]`https://github.com/patgrosse/wmediumd/`
[5]`https://github.com/intrig-unicamp/mininet-wifi`
[6]`https://ieeexplore.ieee.org/Xplore/home.jsp`
[7]`https://dl.acm.org/`

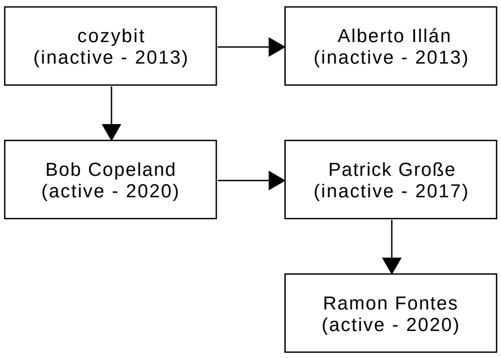

Figure 2.2: Fork tree of wmediumd. All these versions maintained the name wmediumd.

work based on cozybit's wmediumd, expanded the probability based simulation with path loss and mobility simulation [8].

2.3.1 Overview of the operation of mac80211_hwsim and wmediumd

After loading the module mac80211_hwsim, the interfaces can be configured and wmediumd can be started. Wmediumd begins by reading and loading the configuration, and then performs the registration protocol with mac80211_hwsim. This registration indicates to mac80211_hwsim that wmediumd is available and ready to handle the traffic delivery for all of its interfaces. From this point, wmediumd receives copies of the frames from all mac80211_hwsim's interfaces, controls which interfaces receive each frame, and imposes transmission restrictions. Then wmediumd sends back to mac80211_hwsim copies of the frame and to which interface it should be delivered. Only the receivers of the frame can receive a copy, and other interfaces in range don't get one.

Wmediumd replaces the default mac80211_hwsim behaviour, in which all interfaces with the same radio channel as the transmitter, receive a copy of any frame transmitted in that channel. Wmediumd's behaviour permits avoiding some frame copies because, generally, the frames that are not addressed for an interface (or broadcasted) are discarded [9].

2.3.2 Configuration options

Wmediumd can be configured to operate in three modes:

1. **standalone:** configuration file determines the simulation parameters. Configuration can not be altered during runtime.

[8]`https://github.com/amarti2038/wmediumd`

[9]An interface may be specifically configured to also capture frames not addressed to it, called promiscuous mode, but in this case, with wmediumd, this mode will make no difference, since no copies are being provided. Mac80211_hwsim would still work as expected.

2. **dynamic:** no initial configuration is provided. All configuration is made and updated through messages to a server started by wmediumd.

3. **hybrid:** configuration file determines initial simulation conditions and can receive new configuration during runtime.

The configuration file, used by standalone and hybrid mode, follows libconfig [8] format and is parsed by it. For details about the configuration file options refer to appendix C.1. Of the possible contents of the file, it is required the list of MAC addresses of the interfaces for which traffic delivery is intended to be simulated. The optional fields permit to enable interference, and select and tune the signal propagation model. The types of propagation model available are *SNR*, *probability* and *path-loss*.

The model type *SNR* can use a default value for all interfaces, or use information from the configuration file, using the configured Signal to Noise Ratio (SNR) values between pairs of interfaces.

The model type *probability* works in a similar way to model type SNR. Instead of SNR values, it receives probability of error values for each possible combination of interfaces. The error probability values can be asymmetric for the same pair.

The final model type, *path-loss*, requires the definition of initial positions and of signal transmission power. Optionally, the model permits the use of a simple movement model in a 2D environment, based on direction vectors. The movement happens at a fixed time interval. The models supported are free-space [3], log distance [12], log-normal shadowing [12], and ITU [5].

2.3.3 Simulation

As was previously explained, first mac80211_hwsim is started and then wmediumd can be started. After wmediumd loading the configuration and preparing its data structures, it is almost ready to handle the frame delivery simulation (see figure 2.3). For this to happen, wmediumd must perform the registration protocol with mac80211_hwsim, so that mac80211_hwsim knows wmediumd is available and ready. After this, wmediumd uses an event loop, managed by libevent1 [9] that handles two types of events, namely *new message from mac80211_hwsim*, and *frame ready for delivery*.

All communication between mac80211_hwsim and wmediumd happens using netlink sockets to perform a message based protocol. Wmediumd uses libnl3 [10] to handle communication on its end, while mac80211_hwsim uses the netlink's kernel API. The reception of a new netlink message, in wmediumd, triggers the event *new message from mac80211_hwsim*.

Simplifying the process a bit, for a *frame ready for delivery* event to happen, first the frame must be received, then stored, then the delivery simulation is performed, it must then wait until the timestamp determined by the simulation, which when reached will trigger the *frame ready for delivery* event, and finally the frame is sent back to mac80211_hwsim to be received by the destination(s) of the frame. More details regarding the delivery simulation will be provided later.

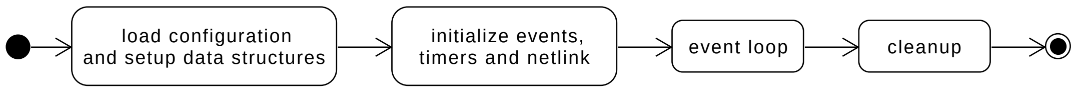
load configuration and setup data structures
initialize events, timers and netlink
event loop
cleanup

Figure 2.3: Workflow of wmediumd.

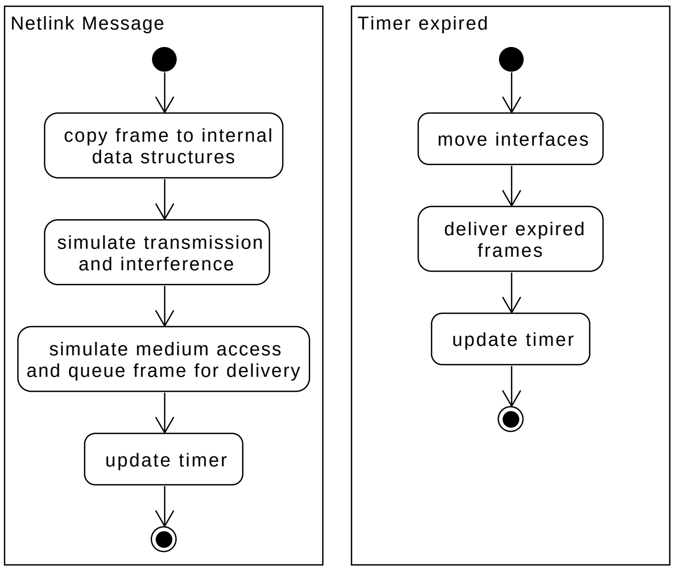
Netlink Message
copy frame to internal data structures
simulate transmission and interference
simulate medium access and queue frame for delivery
update timer
Timer expired
move interfaces
deliver expired frames
update timer

Figure 2.4: Steps to handle the two types of events in the event loop of wmediumd.

As previously explained, wmediumd's main loop consists in responding to two types of events, managed by libevent, the *frame ready for delivery* and *new message from mac80211_hwsim*. Libevent monitors the events and handles them sequentially.

The event *frame ready for delivery* is caused by a timer expiring. The timer used is timerfd, a system call provided by the Linux kernel. This timer delivers expiration notifications, through a file descriptor, that libevent is monitoring. The timer is set/updated every time a new frame is received, after the medium access simulation takes place, and notifies that a frame is ready to be delivered. This triggers the frame delivery procedure, that will search for the frame(s) ready to be delivered and send them to mac80211_hwsim. The right side of figure 2.4 depicts this.

The event *new message from mac80211_hwsim* is caused by the presence of new data in the netlink socket. This triggers the procedures to simulate frame transmission and medium access, that will calculate the timestamp for the frame be sent to mac80211_hwsim for delivery. The left side of figure 2.4 depicts this.

The delivery simulation performed by wmediumd can be split in three steps, which for the purposes of this description will be called medium access simulation, transmission simulation and interference simulation. The messages traded by wmediumd and mac80211_hwsim can be seen in figure 2.5, and figure 2.6 shows the entire delivery simulation process.

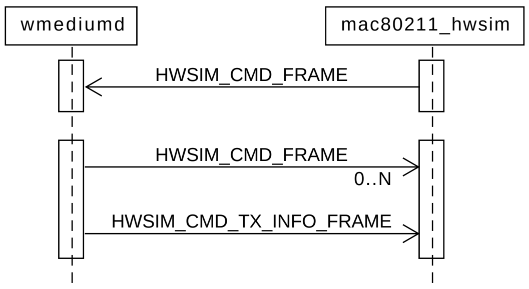

Figure 2.5: Netlink messages exchanged between mac80211_hwsim and wmediumd to perform the simulation of delivery of one frame. The message `HWSIM_CMD_FRAME` contains the frame, and is sent from mac80211_hwsim to wmediumd so that wmediumd can extract information for the simulation. When it is ready to be delivered, it is sent a copy to each of the receivers, that may be none (out of range, interference, …). The message `HWSIM_TX_INFO` reports transmission details to the transmitter interface.

Transmission simulation

The transmission simulation (see listing 2.1 for the algorithm pseudocode) performs attemps to transmit a frame using the SNR value, derived from conditions of the medium, to determine transmission error probability. The objective of this simulation is to determine how much time is used to transmit, or fail to transmit, this frame. The transmissions or retransmissions depend on the options provided by mac80211_hwsim for the signal modulation used in the attempt of transmission, and on the number of retransmissions performed with each signal modulation option. Each signal modulation option provides different bitrates and different transmission error probabilities.

The conditions of the medium influence the value of the SNR used to determine the probability of transmission error for each type of signal modulation. Different values for the configuration key `model.type` (SNR, prob and path_loss) of wmediumd, affect how the probability of transmission error is calculated. If the model type SNR is used, the SNR value for the pair transmitter/receiver provided in the configuration file is used to calculate the error. If the model type is path-loss, the distance between the transmitter and the receiver, the transmission power and the antenna gains affect the SNR value used to calculate the error. If the model type probability is used, the SNR value is not relevant, and the error probability is the value from the configuration file, which remains constant, independently of the signal modulation used.

Additionally, the model type path-loss supports interface mobility, which affects the distances between the interfaces and, consequently, the SNR values provided by the path-loss model. The position updates can only occur at fixed time intervals.

Medium access

The medium access simulation (see listing 2.2 for pseudocode) consists in finding the frame with the latest delivery timestamp among the frames with the same or higher delivery priority.

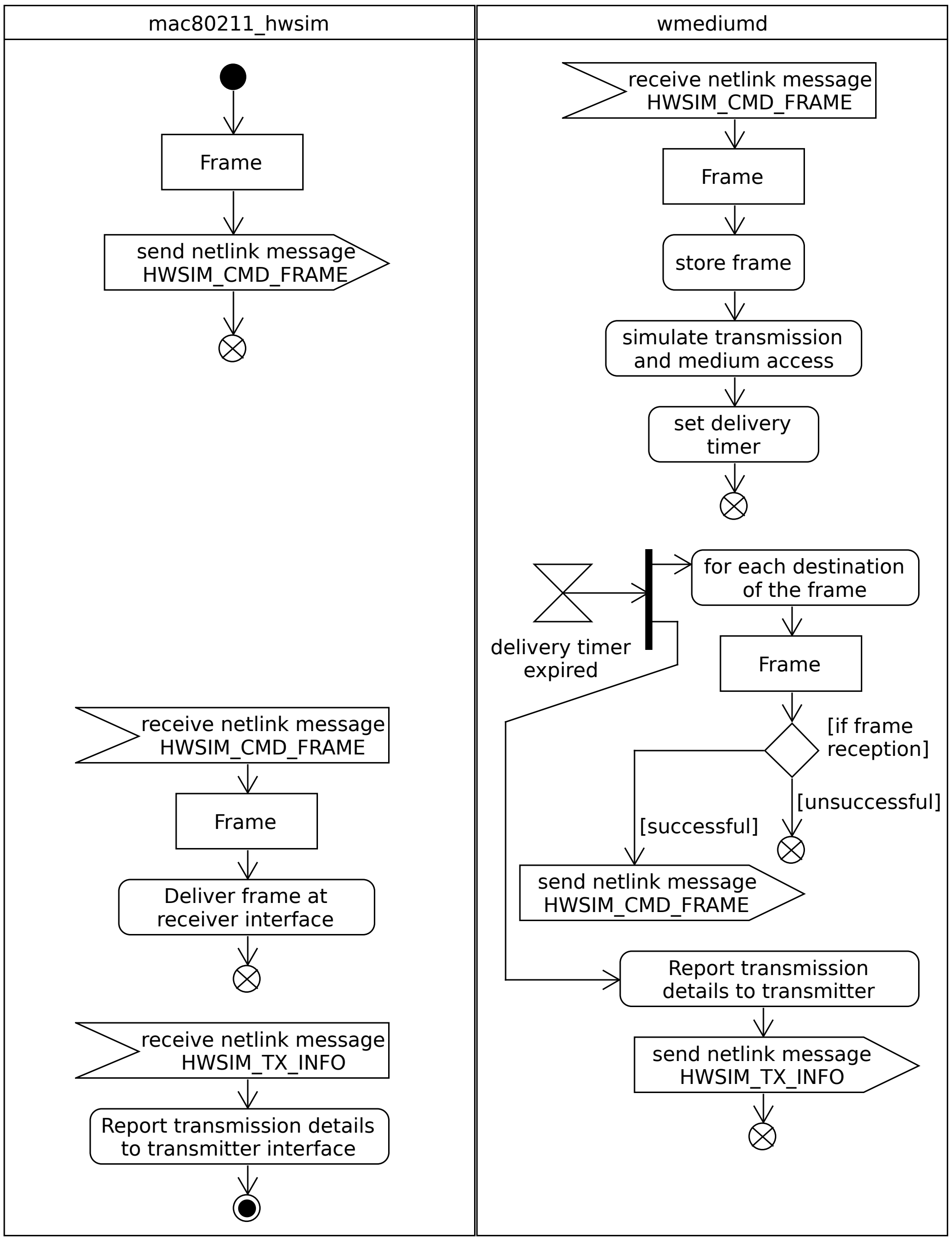
mac80211_hwsim
wmediumd
Frame
send netlink message HWSIM_CMD_FRAME
receive netlink message HWSIM_CMD_FRAME
Frame
Deliver frame at receiver interface
receive netlink message HWSIM_TX_INFO
Report transmission details to transmitter interface
receive netlink message HWSIM_CMD_FRAME
Frame
store frame
simulate transmission and medium access
set delivery timer
delivery timer expired
for each destination of the frame
Frame
[if frame reception]
[successful]
[unsuccessful]
send netlink message HWSIM_CMD_FRAME
Report transmission details to transmitter
send netlink message HWSIM_TX_INFO

Figure 2.6: Steps of the delivery of a frame, with the communication between wmediumd and mac80211_hwsim, and the simulation performed by wmediumd.

Listing 2.1: Pseudocode of the frame transmission simulation in the procedure `queue_frame()` of wmediumd (in `wmediumd/wmediumd.c` of [16]). This pseudocode keeps the bug, present in wmediumd, of not counting the ACK transmission time when the frame is successfully transmitted. To be correct, line 34 should be before line 30.

```
 1  void queue_frame(struct wmediumd env, struct frame frame) {
 2      int ack_time = transmission_time(ACK_LENGTH,
 3                                       frame.transmission_rates[0].bitrate);
 4      int send_time = 0;
 5      int snr = DEFAULT_SNR; // MT_PROB
 6      if (env.model_type == MT_PATH_LOSS)
 7          snr = power_at_receiver(env, frame.transmitter, frame.receiver)
 8                - NOISE_LEVEL;
 9      else if (env.model_type == MT_SNR)
10          snr = power_at_receiver(env, frame.transmitter, frame.receiver);
11      int i = 0;
12      while ((i < length(frame.transmission_rates)) and (not frame.acked)) {
13          // Probability of the frame being received with uncorrectable
14          // errors.
15          double error_prob;
16          if (env.model_type == MT_PROB) // From configuration file
17              error_prob = get_error_prob(env, frame.transmitter,
18                                          frame.receiver);
19          else // Calculated
20              error_prob = calc_error_prob(snr, frame.length,
21                                           frame.transmission_rates[i]);
22          int j = 0;
23          while (j < frame.transmission_rates[i].retransmission_attempts) {
24              send_time = send_time + transmission_time(frame.length,
25                                      frame.transmission_rates[i].bitrate);
26              if (no_ack(frame.flags)) {
27                  frame.acked = true;
28                  break;
29              }
30              if (normal_distrib_random() > error_prob) {
31                  frame.acked = true;
32                  break;
33              }
34              send_time = send_time + ack_time;
35              send_time = send_time + backoff_time(frame, i, j);
36              j = j + 1;
37          }
38          i = i + 1;
39      }
40      frame.duration = send_time;
41      // ...
42  }
```

Then the delivery timestamp of the frame is the timestamp found plus the frame transmission duration. This creates a “perfect” medium access control for the interfaces, since the interfaces can sense the presence of a transmission even when out of their range. An alternative view of this is: from the medium access simulation point of view, all interfaces are in the same place, independently of any other configuration.

There is a problem with the medium access algorithm, because it allows successful frame transmissions to overlap. This is presented in more detail at appendix D.

Interference simulation

Interference simulation is an option that may be enabled for the model type path-loss. As was previously presented, the medium access simulation guarantees a “perfect” busy medium sensing, even beyond range. As is implemented, interference reduces the SNR value used calculate the probability of transmission error of the frame.

A frame may have its SNR value affected by interference, if there have been previous transmissions on the medium in which the frame signal power was below CCA_THRESHOLD. The probability that a frame will be affected by interference depends on the sum of the duration of the frames transmitted previously with signal power below CCA_THRESHOLD, and the signal power of the interference will be the signal power of the last frame transmitted in these conditions.

For more details, see the procedures `queue_frame()`, `deliver_frame()`, `set_interference_duration()` and `get_signal_offset_by_interference()` of [16] at `wmediumd/wmediumd.c`.

Listing 2.2: Pseudocode of the algorithm used for medium access simulation in the procedure `queue_frame()` of wmediumd (in `wmediumd/wmediumd.c` of [16]).

```
void queue_frame(struct wmediumd env, struct frame frame) {
    // ...

    // Priority of the frame according to its QoS type (lower value means
    // higher priority: range 0 to 3). The frame may not have QoS info, and
    // in that case, it has the highest priority, 0.
    int frame_priority = get_priority(frame);
    struct time now = get_time();
    struct time timestamp = now;
    // We want to obtain the latest timestamp of the queued frames, but if
    // it is in the past or there are no frames queued we want "now".
    int priority_queue = 0;
    while (priority_queue <= frame_priority) {
        // queue_max_all_interfaces(): find the latest timestamp in the
        // queue of priority "priority_queue" from all interfaces. If they
        // are empty returns 0 (always < "now").
        timestamp = max(timestamp,
                        queue_max_all_interfaces(env, priority_queue));
        priority_queue = priority_queue + 1;
    }
    // Delivery timestamp marks when the frame finishes using the medium, and
    // the time after which it may be sent to mac80211_hwsim.
    frame.delivery_timestamp = timestamp + frame.duration;

    // find the frame with closest delivery timestamp and set timer
    // accordingly.
    queue_for_delivery(frame);
}
```

Chapter 3

Performance analysis of wmediumd

To demonstrate the scalability of Mininet-WiFi, Fontes performed several tests, namely memory usage, startup time, delay and throughput [1, section 4.3]. Of particular interest for this work is the throughput test results with wmediumd. It is not exactly clear what Fontes' throughput results represent, but it is assumed that the values reported correspond to the cumulative throughput of the stations. Comparing the results with wmediumd to those without, the results with wmediumd degrade faster and more abruptly, as the number of stations increases.

In order to better explore these results, two new tests were conducted, both with the two ways of interest to simulate the wireless medium, just mac80211_hwsim, and mac80211_hwsim with wmediumd. The objective of test 1 was to determine the capacity of the simulators, that is, the amount of traffic mac80211_hwsim and wmediumd can deliver. The objective of test 2 was to analyse the packet loss ratio when the traffic was set below the values determined in test 1.

Additionally, it was made an analysis to wmediumd with a profiler. The main objective of all these tests was to identify bottlenecks, overheads, and areas where the performance might be improved.

3.1 Test 1: Determination of throughput

This test measures the throughput of mac80211_hwsim's built-in method and of wmediumd's wireless medium simulation. Throughput is a measure of traffic effectively received at the receiver, which may be lower that the traffic offered to the network, by the transmitter.

The test permits to see how the throughput changes as the number of stations using the medium increases. (A station is an abstraction of a real device that has one of mac80211_hwsim's simulated interfaces). The stations are grouped in pairs, that will establish a wireless link. The number of pairs with traffic load progressively increases until they all are communicating, and then progressively decreases until they are all stopped.

This test allows not only to observe the effects of multiple stations communicating simultaneously, but also to observe the effects on the throughput of having more stations, even if idle.

The objective of using the throughput values was to measure the capacity of the medium (by adding the throughput of all the stations), and have an idea of how the capacity was affected with the increase of the number of stations, due to overheads. A lowering of the capacity meant that the overheads were increasing. This was later found not to be the correct way to see the effects of overheads in capacity of the medium. Instead, the data from this test is more appropriate to determine if mac80211_hwsim is a limiting factor in the performance, and at which number of stations performance problems show. A more effective methodology to see the effects of overheads was later developed, and its results can be seen at chapter 5.

3.1.1 Configuration

The test was run in a machine with the following characteristics and software: CPU AMD Ryzen 5 1600, (3.2 GHz, 3.6 GHz with Precision Boost, 6 cores, 12 threads), RAM 16 GiB DDR4 2667 MHz, OS Arch Linux with kernel version 5.4.6-arch3-1, iperf version 3.7 [1] and a copy of wmediumd from [16]. The CPU had the Precision Boost and Simultaneous Multithreading enabled. Since wmediumd has no release versions, the commit hash[2] from the version control system is most likely the best identifier of the code used in the test. The version of mac80211_hwsim used is the one provided by the kernel [15].

The mac80211_hwsim simulated interfaces were left with their default configuration, using IEEE 802.11g, which has bitrates up to 54 Mb/s.

The test had two *variations*, *variation H* and *variation W*, that determine how the wireless medium is simulated. This is the only change in the configuration between *variations*.

For *variation H*, it was used the mac80211_hwsim's simulation of the medium. The medium simulation of mac80211_hwsim consists in copying frames to all simulated interfaces in the same frequency channel as the transmitting interface. Mac80211_hwsim is configured by its `radio` parameter, whose range is limited from 2 to 100.

For *variation W*, it was used the simulation provided by wmediumd. To more closely match the conditions of mac80211_hwsim's perfect medium, and to avoid additional variables, wmediumd was configured with model type *probability*. This model does not simulate interference, and the transmission simulation calculates the probability of error in the transmission of the frame, using the values provided in the configuration file. This value was set to 0, so the transmission is always successful. The medium access simulation is not configurable [3]. For details about the simulation performed by wmediumd see section 2.3.3, and for details about the configuration options see section 2.3.2.

A *station* has one of mac80211_hwsim's simulated interfaces, and is isolated in a different Linux network namespace. A network namespace provides an independent IP protocol stack,

[1]`https://iperf.fr/`

[2]commit hash: d56b9ee55d1cb6975630ba832a482e72225d754f

[3]From the medium access simulation point of view, all interfaces are in the same place, independently of any other configuration.

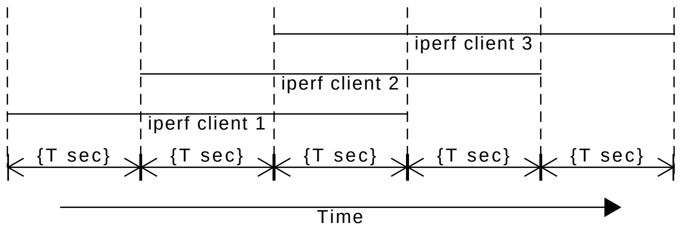

Figure 3.1: Example of the startup order of the iperf clients in test 1 and test 2. The lines indicate the iperf clients execution duration. The iperf client connects to the iperf server of the station pair that was previously started. There is only one interval T in which all iperf clients are active.

routing table, firewall rules and sockets for each namespace. Consequently, processes executing in a namespace, only see the resources of that namespace. Namespaces were used in order to prevent in-host routing of the packets at the network layer level (OSI layer 3), since the destination IP address would be in the same host.

Each station establishes an IEEE 802.11 ad-hoc link (IBSS) with one other station, forming a *station pair*. Each station pair uses a different IBSS network. All interfaces (and consequently their stations) use the same frequency channel. Ad-hoc networks were chosen because of the simple configuration.

In addition to different *variations*, a test also has *instances* with different even numbers of stations, N, where $N \in \{2, 4, 10, 20, 40, 60, 80\}$. Each *variation* is tested with all the *instances*, N. For each instance of the test, N, the number of pairs communicating progressively increases until all pairs are communicating simultaneously, and then the number of pairs progressively reduces until they are all stopped. More specifically, the pairs orderly start to communicate, each pair starting T seconds after the last one until they are all simultaneously communicating. They all maintain communication during at least T seconds and then, by startup order, they stop communicating, always at intervals of T seconds after the last pair stopped. So if the number of interfaces is n, there are $p = n/2$ pairs, each pair is communicating for $T \times p$ seconds and the test duration is $2(T \times p) - T$, (see figure 3.1 for an example). It was used $T = 20$ seconds.

To measure the throughput between two stations, it was used iperf, a tool designed for that purpose. One station of the pair acts as an iperf server and the other as an iperf client. The client was configured to transfer the data over TCP. The Transmission Control Protocol (TCP) was chosen because of its ability to automatically adapt the data rate and to use all the available capacity.

As per default configuration of iperf3, when the client starts, it connects to the server and sends to it packets, as fast as possible. The client and the server record the amount of traffic sent and received, respectively. The information report rate used in iperf was 1 second, therefore the data acquired is the average throughput per second.

3.1.2 Results

The values of throughput were obtained from the iperf server of each station pair. The cumulative throughput for each second of the test is measured by summing the throughput values from the iperf server of each pair of stations that is transmitting at that second.

The results obtained are summarized in tables 3.1 and 3.2, and plotted on figures 3.2 and 3.3. The plots of the cumulative throughput values over time for the instances with 20 and 40 stations of both *variations* are at figures 3.4 to 3.7. Additionally, the plots of the cumulative throughput over time of all the instances are available at appendix A.

Assuming that the duration of an instance of a test is d seconds, the 10 seconds to which tables 3.1 and 3.2 description refers to and are labelled as “all stations” in the tables, correspond to $[\lfloor d/2 \rfloor - 5, \lfloor d/2 \rfloor + 4]$.

No results for instance $N = 80$ with *variation W* are presented because, despite the three runs with that *variation*, the results indicated that the iperf server of at least one pair crashed during each run of the test.

The results presented correspond to statistics from a single run. Other runs were made and showed similar results.

After collecting this data, it was noticed that the simulation performed by wmediumd, was not correctly accounting for the transmission time of the Acknowledgment (ACK) frame. This means that the cumulative throughput results presented here are higher than they should have been, but this makes no significant difference in the analysis. This bug was corrected for the execution of the tests in chapter 5.

Table 3.1: Average cumulative throughput in test 1 with *variation H* (mac80211_hwsim), during the entire test and over a period of 10 seconds when all station pairs iperf servers and iperf clients were communicating. Values of mean, μ, and standard deviation, σ, presented.

	entire test		all stations	
N	μ (Mb/s)	σ (Mb/s)	μ (Mb/s)	σ (Mb/s)
2	3720,863	210,063	3674,158	251,02
4	2559,442	157,626	2729,833	76,145
10	1304,149	55,335	1350,202	4,048
20	614,987	46,01	542,965	4,311
40	274,113	30,633	247,265	16,252
60	165,115	37,82	159,863	38,961
80	116,425	35,804	113,685	15,918

Table 3.2: Average cumulative throughput in test 1 with *variation W* (wmediumd), during the entire test and over a period of 10 seconds when all station pairs iperf servers and iperf clients were communicating. Values of mean, μ, and standard deviation, σ, presented.

	entire test		all stations	
N	μ (Mb/s)	σ (Mb/s)	μ (Mb/s)	σ (Mb/s)
2	35,417	0,817	35,558	0,727
4	35,178	1,093	35,239	1,010
10	34,112	2,341	34,334	1,178
20	31,570	2,697	30,568	1,832
40	22,408	4,134	22,925	1,946
60	10,997	5,071	12,464	3,307
80				

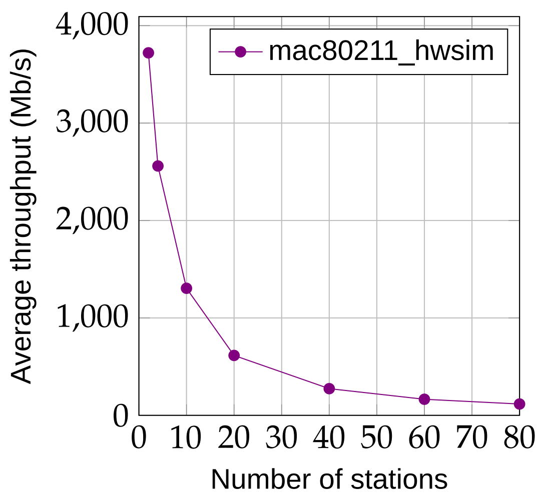

Figure 3.2: Average cumulative throughput in test 1 for each instance N with *variation H*. The values plotted correspond to "entire test" from table 3.1.

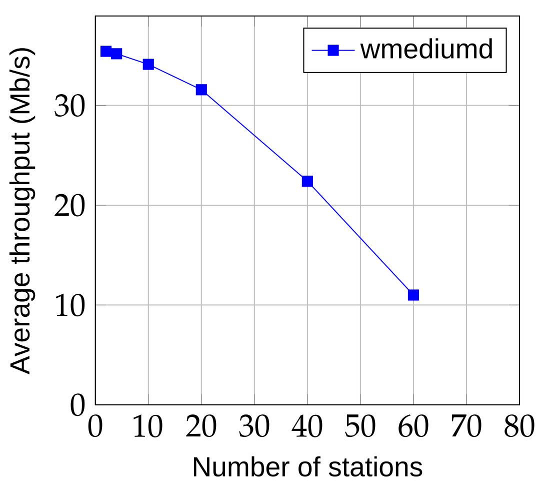

Figure 3.3: Average cumulative throughput in test 1 for each instance N with *variation W*. The values plotted correspond to "entire test" from table 3.2.

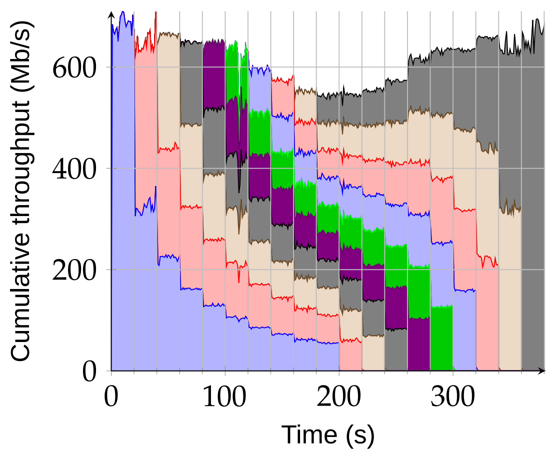

Figure 3.4: Cumulative throughput of the instance with 20 stations with *variation H*. The throughput values from each server are stacked and plotted with different colours, (which repeat).

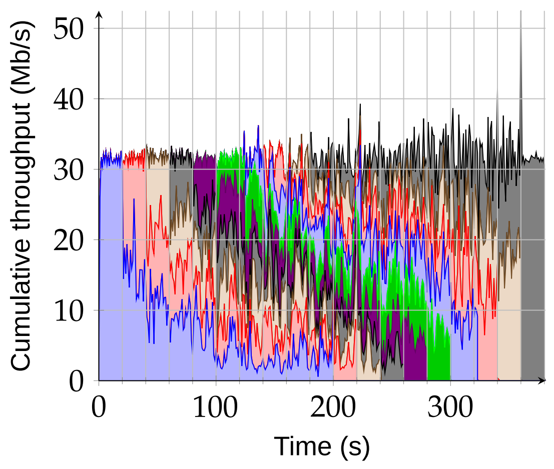

Figure 3.5: Cumulative throughput of the instance with 20 stations with *variation W*. The throughput values from each server are stacked and plotted with different colours, (which repeat).

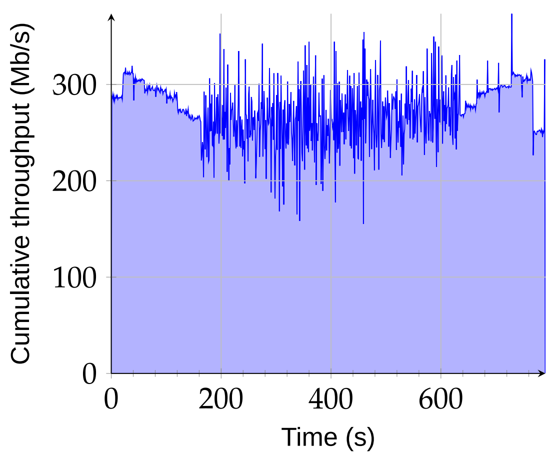

Figure 3.6: Cumulative throughput of the instance with 40 stations with *variation H*.

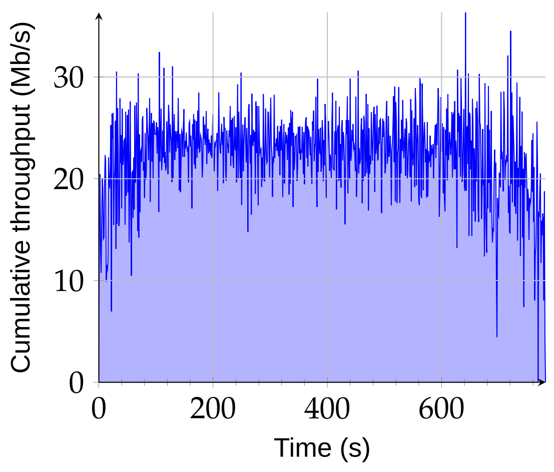

Figure 3.7: Cumulative throughput of the instance with 40 stations with *variation W*.

3.1.3 Results analysis

Analysing the cumulative throughput change with *variation H* between instances of different N, (see table 3.1 and figure 3.2), the results indicate that instances with more stations have a lower cumulative throughput, which is not surprising, given the mode of operation of mac80211_hwsim. Its traffic delivery method consists in copying the frames from one interface to all other interfaces that share the same frequency channel (which is how they were configured). The more interfaces there are, the more copies of each frame are made, which justifies the drop in cumulative throughput registered. Also, there is the effect of the link layer traffic inherent to the increase in the number of interfaces but, this was not measured here, so its impact is unknown.

Variation W results (see table 3.2 and figure 3.3) show a relatively close but decaying cumulative throughput with the rise of N for instances with $N \leq 10$, a slight drop in $N = 20$ and more pronounced drops in $N = 40$ and $N = 60$. The results for $N \geq 40$ also show other irregularities that will be analysed later. Contrarily to mac80211_hwsim's delivery method, wmediumd is performing the medium access simulation, (described in section 2.3.3), which limits the capacity of the medium and consequently the cumulative throughput. Also, wmediumd only sends copies of the frame to the destination(s).

With *variation W*, with the increase in the number of stations, N, the cumulative throughput values decrease, irrespective of the fact the stations are transmitting or not. As can be seen in figure 3.5, the values of throughput achieved with one station pair (first 20 seconds) are maintained (with a bit more irregularity) throughout the instance of the test. The reason behind the drop of the cumulative throughput with the increase in the number of stations present, is probably a mix between increased overhead of handling more stations and bigger share of the capacity used by the link layer traffic. Section 3.3 describes an attempt to find the sources of overhead in wmediumd.

Looking at the plots of cumulative throughput over the duration of an instance, for both *variations*, when $N \geq 40$, they show an unexpected high variability in the values. For example compare figure 3.4 with figure 3.6 and figure 3.5 with figure 3.7. The reason for the high variability became evident when the results of each pair were analysed. The iperf clients output indicated that, on occasion, multiple seconds went by were no traffic was sent, with the same behaviour for the iperf servers, as figure A.8 of the iperf output from one station illustrates.

A comparison of the values from tables 3.1 and 3.2 of "entire test" and "all stations", in both *variations*, shows no significant difference between the values throughout the test and when all stations were communicating, except for *variation H* with $N = 20$, which shows a cumulative throughput lower that the average when the number of simultaneous communications is higher. Analysing the plots of *variation H* (see figures A.4 to A.6), before the high variability zone begins, there was a trend in the throughput values lowering as the number of stations communicating increased, and later of a throughput rise as the number of stations communicating decreased. These results, along with the anomalies previously referred, might indicate that the system was struggling to handle so many demanding processes.

These anomalies may be related to two relevant variables not controlled, noticed after performing these tests (also applies to test 2), present because of the methodology used to acquire the data. The first one related to the data being collected at the application layer, which means that there are many intermediaries, in the protocol stack, that might be responsible for the effects seen, therefore they are not necessarily attributable to wmediumd or mac80211_hwsim. These effects may also be related to TCP retransmissions, because the values used correspond to new data transmitted. This first variable is eliminated in the tests of chapter 5, with the data collected directly at wmediumd. The second variable is related to the CPU Precision Boost being enabled, because for example in the case of wmediumd, if the simulation is sometimes executed with the Precision Boost and sometimes it isn't, during the periods when the Precision

Boost is enabled, there will be more capacity in the medium, because the overhead effect will be lowered, which can lead TCP to send more packets. But when the simulation is later executed without the Precision Boost, wmediumd will no longer be able to handle the same amount of frames, which will lead to some being retained in the protocol stack queues, inhibiting additional transmissions. This second variable is present in all tests of this thesis.

In conclusion, the results of cumulative throughput with mac80211_hwsim (*variation H*), show that it is an unlikely source of performance problems in wmediumd, given that it has much higher throughputs than the ones achieved with the simulation from wmediumd (*variation W*). With the higher numbers of stations, mac80211_hwsim showed some irregularity in the cumulative throughput that was also visible in wmediumd. Given that mac80211_hwsim was performing copies to all interfaces, with wmediumd's delivery method of providing copies only to the receivers, this irregular effect might disappear, if the performance of wmediumd is improved.

3.2 Test 2: Behaviour with a fixed offered load

This test is similar to test 1. The objective is to obtain the values of packet loss ratio in loads below the maximum throughputs measured in test 1, because of the high variability detected in the results with more interfaces. The target bitrates selected for this test are fractions of the average values from tables 3.1 and 3.2, concretely $\mu_N \times 0.9$, referred to as *mean 90* and $\mu_N \times 0.75$, referred to as *mean 75*. These fractional values of the average cumulative throughput were selected arbitrarily.

3.2.1 Configuration

The configuration was the same as in test 1 except for the use of UDP instead of TCP in iperf. The cumulative target bitrates for each instance were the values of table 3.3, which determines as targets for each station pair, for each N, $V/(N/2)$, where V is the value of *mean 90* or *mean 75*. Since the pairs start communicating at different times, the values of table 3.3 are only achieved and maintained after all station pairs have started, during $T = 20s$, moment after which the pairs gradually stop communicating.

3.2.2 Results

The results of packet loss ratio obtained from the iperf servers are summarized in table 3.4.

The values presented for *variation W* with values of $N \in \{40, 60\}$ were only obtained in second and third runs. The reason for this was the presence of pairs that showed "error - unable to read from stream socket: Resource temporarily unavailable" and iperf servers that crashed during the test, which lead to discarding the previous runs for those instances. The reasons that caused this are unknown.

Table 3.3: Cumulative target bitrates for test 2.

	Variation H		Variation W	
N	mean 90 (Mb/s)	mean 75 (Mb/s)	mean 90 (Mb/s)	mean 75 (Mb/s)
2	3348.776	2790.647	31.875	26.563
4	2303.498	1919.582	31.660	26.383
10	1173.734	978.112	30.701	25.584
20	553.488	461.240	28.413	23.677
40	246.702	205.585	20.168	16.806
60	148.603	123.836	9.897	8.247
80	104.782	87.319		

Table 3.4: Results of test 2.

	Variation H		Variation W	
	mean 90	mean 75	mean 90	mean 75
N	lost (%)	lost (%)	lost (%)	lost (%)
2	0.0102	0.0013	0.0000	0.0000
4	0.0109	0.0028	0.0000	0.0000
10	0.0001	0.0000	0.0000	0.0000
20	0.0000	0.0000	0.0249	0.0034
40	0.0000	0.0000	24.2699	21.9631
60	0.0000	0.0000	43.3497	42.5203
80	0.0000	0.0000		

The results presented correspond to statistics from a single run, with the already referred exceptions. Other runs showed similar results.

3.2.3 Results analysis

Variation H results (see table 3.4) show a low frame loss ratio for instances of $N \leq 10$. Results from other runs showed no losses in some instances while others had about the same loss ratios. Analysis of the moments these losses occurred (see figure B.1 and figure B.2) indicate that these occurred without pattern. This might be related to other process activity on the system that did not allow mac80211_hwsim to handle all the packets fast enough.

Results from multiple runs with *variation W* show significant losses for $N \geq 40$ (see table 3.4 and figures B.3 to B.7). The cumulative throughput results from test 1 for these instance sizes already showed a descending trend, and since the target bitrates for these tests are based on the values from test 1, it is an indication that these targets were still too high. These packets must have been dropped by arriving at a full queue of the protocol stack, including mac80211_hwsim, since wmediumd does not drop packets (as was configured). This is likely caused by a limit (in mac80211_hwsim) in the frames that are transmitted to wmediumd to perform simulation, but this process in not understood. Unlike the results from test 1, in this test there were no intervals with no packets received.

3.3 Profiling wmediumd

As the results from the previous tests indicate, mac80211_hwsim has a significantly higher throughput than wmediumd, which suggests that it is comfortable handling large amounts of frames, making it a highly unlikely responsible for loss of medium capacity in wmediumd.

In order to have a better idea of what are wmediumd's biggest cpu cycle spenders, a performance analysis with a profiler was conducted. The profiler used was `perf` [4] [11], and the data collected was relative to its metric `cpu-cycles`. The configuration of wmediumd was the same as test 2 (section 3.2), using 10 stations and the target bitrate of *mean 90*.

The results indicate that about 47% of `cpu-cycles` were spent sending and receiving frames (see listing 3.1). These results show that a considerable amount of time is spent handling communication with mac80211_hwsim. Based on this data, it was attempted to improve wmediumd's performance by improving the communication with mac80211_hwsim (described in section 4.1).

[4]perf version 5.4.g219d54332a09

Listing 3.1: Output from `perf report` of profiling wmediumd. Values of `__libc_sendmsg` and `__libc_recvmsg` indicate the load of functions associated with netlink communication.

```
Samples: 393K of event 'cycles', Event count (approx.): 41811239984
  Children      Self  Command   Shared Object
+   93.26%     0.00%  wmediumd  [unknown]
-   66.35%    65.29%  wmediumd  [kernel.kallsyms]
   - 47.87% 0
      + 28.10% __libc_sendmsg
      + 17.40% __libc_recvmsg
      + 2.16% nl_recvmsgs_report
   - 14.89% 0x100000000
      + 14.87% epoll_wait
   + 2.39% 0x55856c1478f0
   + 1.05% entry_SYSCALL_64_after_hwframe
+   48.69%     0.65%  wmediumd  libpthread-2.30.so
+   26.51%     7.93%  wmediumd  libc-2.30.so
+   19.75%     6.24%  wmediumd  wmediumd
+   17.62%     5.84%  wmediumd  libnl-3.so.200.26.0
+   12.83%     3.06%  wmediumd  libevent-2.1.so.7.0.0
+    9.02%     8.94%  wmediumd  libm-2.30.so
+    6.69%     0.96%  wmediumd  [mac80211_hwsim]
     0.60%     0.60%  wmediumd  [vdso]
```

Chapter 4

yawmd – Yet Another Wireless Medium Daemon

Mininet-WiFi communicates with wmediumd through a configuration file and network messages. Given that this communication is merely intended to change parameters of the simulation happening in wmediumd, and that all relevant simulation configurations can be achieved through the configuration file, it was decided to focus efforts on improving performance and scalability in wmediumd's operation while attempting to maintain the external interface of the wmediumd's server and the configuration file unchanged.

There are plenty of modifications of wmediumd, as was described in section 2.3 and figure 2.2 shows. For this reason, the modifications of wmediumd presented in this chapter, received a new program name, Yet Another Wireless Medium Daemon, yawmd for short.

Three modifications were made[1], *yawmd v1*, *yawmd mediums* and *yawmd pthreads*. In overview, yawmd v1 eliminates a waste of resources in the communication with mac80211_hwsim, yawmd mediums adds support to more than one medium, and yawmd pthreads adds multi-threading to the multi-medium simulation of yawmd mediums. The effect of these changes in the performance is analysed in chapter 5.

4.1 yawmd v1: Improve the communication with mac80211_hwsim

As was previously explained (in section 2.3.3), wmediumd receives a copy of the frame to simulate its delivery (in the message `HWSIM_CMD_FRAME`), and when the frame is ready to be delivered, it sends to mac80211_hwsim, for each receiver, a message with a copy of the frame (also with type `HWSIM_CMD_FRAME`). Along with this, a message (`HWSIM_CMD_TX_INFO_FRAME`) with the transmission details (such as successful transmission and number of retransmissions), is sent to mac80211_hwsim to be reported to the transmitter interface. See figure 2.5 for an illustration of this process.

The copies of the frame from wmediumd to mac80211_hwsim are not necessary, since the

[1]The source code will be made available at `https://github.com/mjmoreira/yawmd/`

frame is kept in the transmitter interface queue until the message with the transmission details of the frame is received from wmediumd [2]. This means that mac80211_hwsim can perform a copy of the frame to the receiver interfaces, by retrieving the frame from the transmitter. Then, it is only required to be able to identify the frame in mac80211_hwsim and make that information known to wmediumd, so that wmediumd can inform mac80211_hwsim in the message `HWSIM_CMD_FRAME` which frame must be received by which interface(s). This way, instead of sending a copy of the frame, wmediumd sends the identifier. This frame identifier is already available, provided in the message `HWSIM_CMD_FRAME`, so that the transmission details reported in the message `HWSIM_CMD_TX_INFO_FRAME` can be assigned to the right frame.

Furthermore, it is not necessary to provide wmediumd with a copy of the entire frame. Wmediumd only requires fields from the header of the frame, and so the size and amount of data sent from mac80211_hwsim to wmediumd can also be reduced, by only providing a copy of the header instead of a copy of the frame.

Yet another change introduced was merging the messages to mac80211_hwsim, `HWSIM_CMD_FRAME` and `HWSIM_CMD_TX_INFO_FRAME`, into just one (see the original flow in figure 2.5 and the updated in figure 4.1). The way wmediumd works does not require separate messages, and there is no challenge in mac80211_hwsim handling the delivery of the frames and transmission report at the same time. This change requires wmediumd to create a list of receivers, instead of sending a `HWSIM_CMD_FRAME` message for each one, which can lead to a significant reduction of messages, if a frame has many receivers.

All these changes in the information traded between mac80211_hwsim and wmediumd, intend to make a more efficient use of the resources by eliminating unnecessary copies of information and reducing the number of messages. The results of these changes are explored in chapter 5. The changes were implemented together, therefore the individual contribution of each is unknown. These changes resulted in yawmd version 1, also referred to as yawmd v1.

4.2 Multiple simulated mediums: yawmd mediums and yawmd pthreads

As wmediumd and yawmd v1 are implemented, all interfaces share the same medium, independently of the distance between them. The fact that a interface is out of range for the signal to be detected and allow frame delivery, does not stop an interface from detecting the other transmissions, for the purposes of medium access. (These details were presented in section 2.3.3). Given that not all deployments would be in conditions of medium access contention, it is an interesting feature to have support for multiple mediums.

Allowing yawmd to support multiple mediums (with the same implementation as in wmediumd), while not a perfect solution, gives the user some flexibility for deployments that would

[2]This change was inspired by the solution to a similar issue with cozybit's wmediumd, `https://github.com/cozybit/wmediumd/issues/2`.

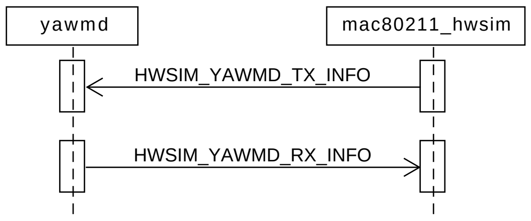

Figure 4.1: Messages exchanged between yawmd and mac80211_hwsim to perform the delivery simulation of a frame. The message `HWSIM_YAWMD_TX_INFO` contains the frame header, the identifier for the frame, along with other fields. The message `HWSIM_YAWMD_RX_INFO` contains the transmission details for the transmitter, the identifier of the frame, and the list of addresses of the receivers of the frame. To deliver a frame, mac80211_hwsim finds the frame in the transmitter's interface queue using the identifier, and then makes copies for each of the receivers from the list.

not "share" the medium with one another, such as distant groups of interfaces, or interfaces in non-interfering radio channels [3].

A test (not reported in this thesis) that bypassed the frame delivery simulation, showed that there was unused communication capacity between yawmd and mac80211_hwsim. This test removed the simulation from yawmd v1, that is, as soon as a frame was received and processed, the delivery information was immediately sent to mac80211_hwsim, instead of simulating transmission and queueing the frame for delivery at a later time. Without the limitations the simulation imposed, the throughput values rose to values beyond 200 Mb/s, meaning that the Netlink communication between yawmd v1 and mac80211_hwsim is not a bottleneck. As is, a medium normally achieves a throughput around 35 Mb/s, with IEEE 802.11g. This means that there is enough capacity for yawmd to be used with versions of IEEE 802.11 with higher throughput, or with more mediums simultaneously, without imposing reductions in the throughput, as a consequence of the capacity of the netlink communication between yawmd and mac80211_hwsim.

This available communication capacity in the link with mac80211_hwsim, means that there is no immediate need to explore other communication options between wmediumd/yawmd and mac80211_hwsim, as was initially planned. It was considered more interesting to provide the ability to use multiple mediums, than attempt to improve the communication capacity and performance.

In order to support multiple mediums, it is required to change the configuration file format. The changes made in the file options are quite extensive, therefore instead of adapting the old parsing code, new code was produced. This also allowed to fix some problems with the existing code and introduce some features. The objective was to make the configuration more flexible and more preventive of failings because of (accidental) incorrect configuration. The new features introduced include:

[3]Even if the radio channels were in conditions to cause interference with one another, the current implementation of interference does not take this into account. See section 2.3.3 for details of the implementation of the interference.

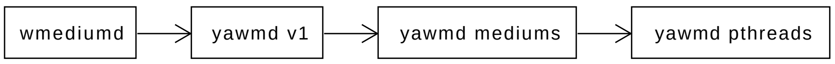

Figure 4.2: Evolution of the versions of yawmd. *yawmd v1* introduced overhead reduction in the communication with mac80211_hwsim. *yawmd mediums* introduced support for multiple isolated mediums of communication. *yawmd pthreads* executes the simulation of each medium in a separate thread.

- different model types and model configurations for each medium;
- some previously hardcoded parameters can be set for each medium (NOISE_LEVEL, MOVE_INTERVAL, ...);
- introduction of warnings for unused options – a sign that the configuration might not be doing what was intended. Previously, some of these were silently ignored;
- indication of file and line where an error or invalid value is found.

The old file format can be seen at appendix C.1 and the new at appendix C.2. The new file format is read by libconfig [8], (as was the old).

Two approaches were taken to support multiple mediums. The first approach, named *yawmd mediums*, adds the multiple mediums, but maintains a single thread to handle all the events. The second approach, named *yawmd pthreads*, improves the first approach by using one thread to perform the simulation of each medium. Both approaches are built on top of yawmd v1 (see figure 4.2). The mediums are implemented by running multiple (independent) instances of the simulation described in section 2.3.3. Each medium is composed by a non-intersecting subset of the interfaces simulated by mac80211_hwsim.

4.2.1 yawmd mediums: Events for each medium

In wmediumd and yawmd v1 there are two types of events: *new message* and *timer expired*. The *timer expired* is associated with frame delivery and interface position updates. For yawmd mediums, the old *timer expired* event was replaced by two separate timers, *movement timer* and *frame delivery timer*. Figure 4.3 shows the new events. For the events in wmediumd refer to figure 2.4.

In yawmd mediums, there are *timers* for *movement* and *frame delivery* for each medium. All the events triggered by these timers are managed by the same libevent event loop, and run on a single thread. Another change introduced is the update from libevent version 1 to libevent version 2 [9].

The primary objective of these changes was to improve the flow and understandability of the code, while also preparing the code to work with threads. There were also some functionalities lost, mainly due to time constraints, namely interference, dynamic changes to the configuration, and packet error rate probabilities from values in a file. All these functionalities can be reintegrated.

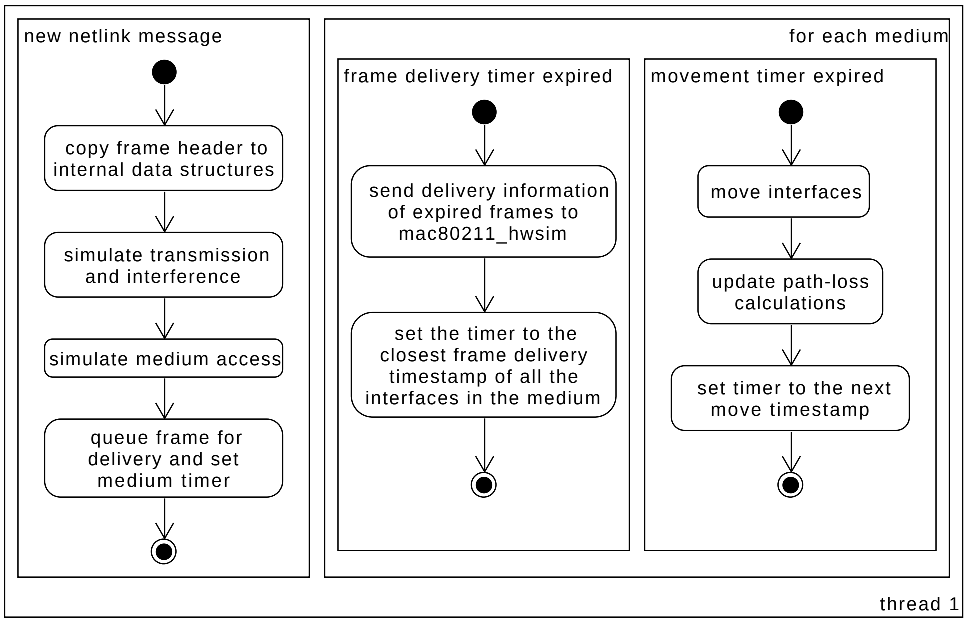
new netlink message
copy frame header to internal data structures
simulate transmission and interference
simulate medium access
queue frame for delivery and set medium timer
for each medium
frame delivery timer expired
send delivery information of expired frames to mac80211_hwsim
set the timer to the closest frame delivery timestamp of all the interfaces in the medium
movement timer expired
move interfaces
update path-loss calculations
set timer to the next move timestamp
thread 1

Figure 4.3: Events managed by the event loop in yawmd mediums. Each medium has frame delivery and movement events, (if enabled), for itself.

4.2.2 yawmd pthreads: One thread per medium

Yawmd pthreads makes use of parallelism in an attempt to minimize the decay of performance associated with the increase of concurrent mediums. Yawmd pthreads is a modification on top of yawmd mediums, that instead of handling all the mediums in one thread, uses one thread for each medium. These threads are not completely independent, because mac80211_hwsim does not know about the division of mediums. The concept of multiple mediums exists entirely in yawmd mediums/pthreads. This means that mac80211_hwsim can not send the message to perform the delivery simulation, `HWSIM_YAWMD_TX_INFO` (see figure 4.1), directly to the thread handling the simulation for the medium.

The approach chosen consists in having one thread, the *main thread*, to receive the messages from mac80211_hwsim, and process the messages to figure out which medium the transmitter interface belongs, so that it can inform the *medium thread* that a new frame is available, to perform the delivery simulation. Each medium has a thread to manage its events, with an event loop of its own (see figure 4.4). The medium events are signaled by three timers. Two of the timers are the ones already available for each medium in yawmd mediums, and a new one, *frame ready for simulation*, set by the *main thread* to trigger immediately when a new frame header is placed in the *frame queue* of the *medium thread* (see figure 4.5). The *frame queue* is a list of frames headers, queued by the *main thread*, to be processed by the *medium thread*, to perform all the steps of the delivery simulation.

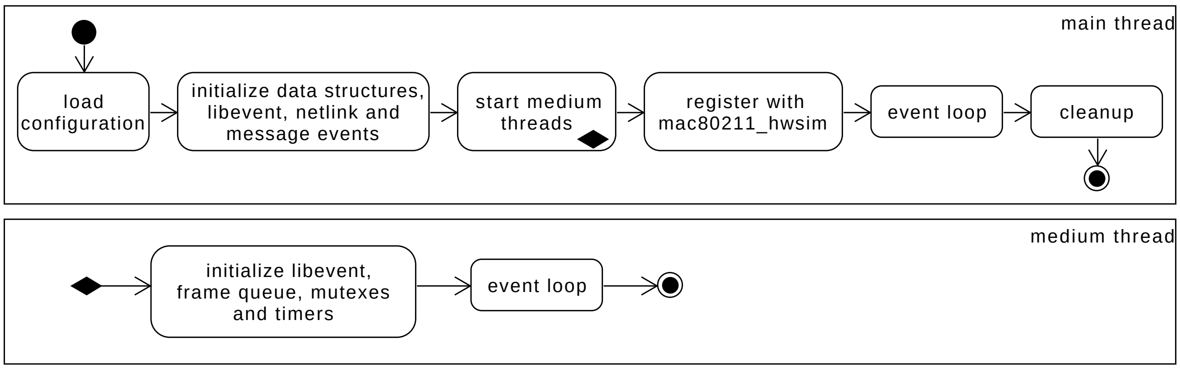
main thread
load configuration
initialize data structures, libevent, netlink and message events
start medium threads
register with mac80211_hwsim
event loop
cleanup
medium thread
initialize libevent, frame queue, mutexes and timers
event loop

Figure 4.4: Workflow of yawmd pthreads.

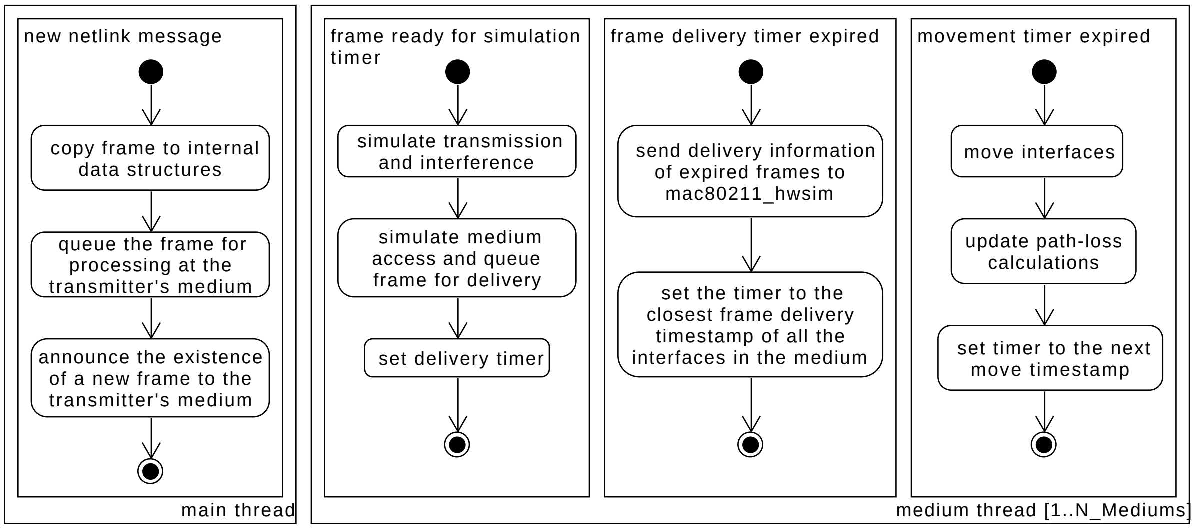
new netlink message
copy frame to internal data structures
queue the frame for processing at the transmitter's medium
announce the existence of a new frame to the transmitter's medium
main thread
frame ready for simulation timer
simulate transmission and interference
simulate medium access and queue frame for delivery
set delivery timer
frame delivery timer expired
send delivery information of expired frames to mac80211_hwsim
set the timer to the closest frame delivery timestamp of all the interfaces in the medium
movement timer expired
move interfaces
update path-loss calculations
set timer to the next move timestamp
medium thread [1..N_Mediums]

Figure 4.5: Events managed by the *main thread* and the *medium threads* event loops in yawmd pthreads. The *main thread* receives the messages from mac80211_hwsim and distributes the simulation of the frame to the respective *medium thread*. Each medium has its own events and event loop. The *medium thread* is responsible for the simulation and for sending the transmission report to mac80211_hwsim.

Chapter 5

Performance analysis of yawmd

Having implemented the changes to wmediumd as described in chapter 4, it is now time to test them and assess their impact in the performance. To do this, three new metrics were collected: medium utilization to assess the impact of overheads in the capacity of the medium; CPU and memory usage to assess the impact on the system.

Two new tests were made: test 3 was executed with wmediumd and all three versions of yawmd, to compare the performance with one medium; test 4 was executed with yawmd mediums and yawmd pthreads, to compare the implementations with multiple mediums. These tests collect different data and use a different methodology than the ones of chapter 3 to address several issues detected during the analysis of the data and later during development of yawmd.

5.1 Test 3: One medium

The objective of this test is to compare the performance of wmediumd, yawmd v1, yawmd mediums and yawmd pthreads, when one medium is being simulated. This comparison is based on three metrics: medium utilization, CPU usage and memory usage.

The *medium utilization* data is collected directly at wmediumd/yawmd, and provides information about the fraction of the transmission time available effectively used. More precisely, each frame has a duration, also referred as send time (see listing 2.1), that includes not only the frame's transmission time, but also wait times required for the transmission. For example, if T_0 is the timestamp of the end of transmission of the last scheduled frame on the queue, and two new frames arrive, where $frame_1.duration = L_1$ and $frame_2.duration = L_2$, $frame_1$ will finish being transmitted at $T_0 + L_1$, and $frame_2$ will finish being transmitted at $(T_0 + L_1) + L_2$. This means that the sum of the duration of all frames transmitted in a time interval, divided by the time interval, provides the fraction of the time the medium was used.

Medium utilization is a good way to get a detailed picture of the medium simulation, in particular of the effects due to overheads, because it is based on the value used to limit the amount of frames that are transmitted, because all sources of traffic are accounted for (not just the iperf traffic as in test 1), and because this value is not affected by the intermediaries of the protocol stack (from the view point of what is happening in the simulation).

The CPU and memory usage data allows seeing the difference of usage in these resources, and of particular interest, the effect of reducing the amount of data being transmitted, (by comparing yawmd v1 with wmediumd), and the effect of execution with threads, (by comparing yawmd pthreads with yawmd mediums).

In this test iperf is still being used, not to collect data, as in test 1, but to add load to the network.

5.1.1 Configuration

The configuration used for this test follows the same principles as section 3.1 (Test 1: Determination of throughput), but with the changes listed here.

The machine used was the same as in chapter 3, with updates to the software. The CPU Precision Boost and Simultaneous Multithreading were enabled. The operating system was Arch Linux, with kernel version 5.8.7. It was also used iperf version 3.7 and libevent version 2.1.12 [9]. For the test with wmediumd, the version of mac80211_hwsim used was the one provided with the kernel, and for the tests with yawmd, it was used a modified version to accommodate the changes from section 4.1.

The definitions of *station*, *station pair*, *instance* and *variation* used in the previous tests remain.

This test was run with four variations, *wmediumd*, *yawmd v1*, *yawmd mediums* and *yawmd pthreads*, which use the programs of their name. To facilitate the writing, the term variation will be skipped, using just the name of the variation.

The test was run with instances of N stations, which form $N/2$ station pairs, where $N \in \{2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

The simulation of wmediumd and yawmd was patched to correctly account the ACK frame. The transmission simulation (see section 2.3.3) includes in the transmission time associated with a frame, the time of the transmission of the ACK. The original algorithm had a bug that prevented the ACK frame from being counted when the transmission was successful. The patch corrects the code of listing 2.1, which should have line 34 before line 30. This means that now the frames which require an ACK, have a higher transmission time associated, which reduces the throughput.

The problem with the medium access part of the simulation, which allowed transmissions of frames at the same time, described in appendix D, was also patched. The patch forced all frames to be placed in only one of the Quality of Service (QoS) queues, which eliminates the conditions necessary for the overlapping transmissions to happen. The frames' QoS type is not modified, they are just placed in the “wrong” queue.

The configuration of wmediumd and yawmd was the same as *variation W* in test 1 (see section 3.1), where the simulation uses a probability of error in the transmission of a frame equal to 0.

All interfaces use the same frequency channel, and use the same IBSS network. In yawmd mediums and yawmd pthreads, all interfaces are in one medium.

The way the iperf servers and clients are started and the duration of execution was changed. Each iperf clients executes during 90 seconds and starts sequentially, 0.3 seconds after the previous client. The communication between the iperf client and iperf server is over TCP, without restrictions, once again to leverage TCP's capability to adapt the data rate to the capacity available.

To collect the medium utilization data used to assess the impact of the changes made, it was made an addition to wmediumd and yawmd, to report every second, the sum of the duration of all the frames transmitted by each QoS queue. This sum is reported by adding the field `frame.duration`, (see listing 2.1), which accounts the time of transmission and wait associated with a frame, of all the frames sent to mac80211_hwsim. To report the data, a new event was added to the main loop (the loop is explained in section 2.3.3), triggered by a timer similar to the other timers already used by wmediumd and yawmd.

To collect the CPU and memory usage during the execution of wmediumd and yawmd it was used `pidstat` from the `sysstat` package. It was used sysstat[17] version 12.4.0.

See appendix E for the commands and options used.

5.1.2 Results

The medium utilization values used in the analysis were derived from values of total transmission time by QoS queue reported by wmediumd/yawmd. The values of CPU and memory usage were reported by `pidstat`. The values presented for CPU usage correspond to the percentage of usage of the total CPU time available, and the memory to the Resident Set Size (RSS) value. All the values used were reported at intervals of one second.

The iperf clients of the station pairs run for 90 seconds. Of the 90 seconds of data, it is selected a 60 seconds interval, after the iperf clients of all the station pairs are active, for analysis of the data of medium utilization reported by wmediumd/yawmd. The iperf reports are not used in the analysis.

This test was run twice. The results of run 1 are plotted at figures 5.1 to 5.3. A lower level view of the data of medium utilization by QoS queue plotted at figure 5.1, is available at appendix F.2.1. Additionally, the results of medium, CPU and memory usage, for both runs, with information of mean and standard deviation, are available at appendix F.1.1.

This test was made with the CPU Precision Boost and Simultaneous Multithreading enabled, which adds some uncertainty to the results, because these settings can have an effect in the magnitude of the values reported, and in the difference between the *variations*.

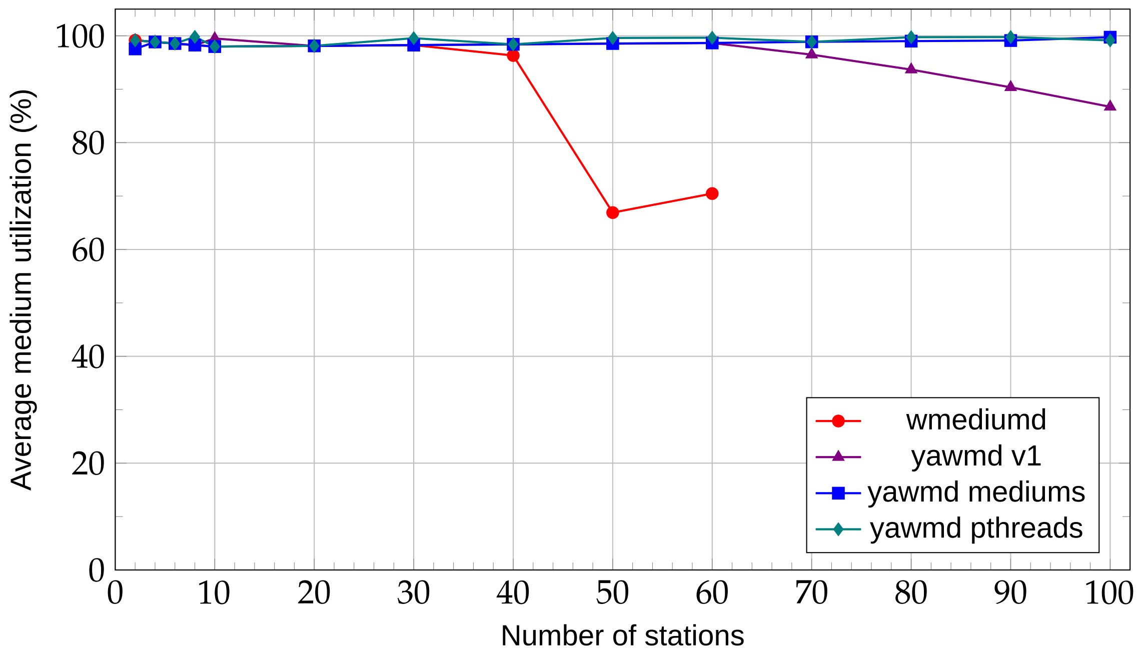

Figure 5.1: Average of the *medium utilization* at each second during each instance of test 3.

5.1.3 Results analysis

First, a note about the medium utilization metric: the simulation aims at always using the medium at 100% as long as there are enough frames available, (which should not be an issue, since iperf is running in TCP mode), therefore lower values mean that there is some component underperforming, and imposing a limitation, which will be called overhead [1]. The effect of overhead is particularly noticeable in data that was collected with the same configuration as this test, but without the patch to avoid transmission overlapping. This data will not be analysed, but its plots are available at appendix F.3. The overhead effect is seen in the linear drop of medium utilization after the inversion point around the peak.

Looking at the results of medium utilization, (see figure 5.1), the baseline, wmediumd, shows the worse results. For instances with more than 40 stations, the medium utilization drops rapidly, and for instances with more than 60 stations, some iperf clients report communication failures with the iperf server of their station pair, therefore these results were discarded. With its optimizations to the communication with mac80211_hwsim, yawmd v1, presents a significant improvement over wmediumd, not only because it maintained a medium utilization of 100% up to 60 stations, but also because it successfully finished the test with the instances up to 100 stations. However, it was unable to maintain the medium utilization at 100% for instances with 70 or more stations. Finally, yawmd mediums and yawmd pthreads managed to obtain better results than the version they are based upon, yawmd v1, with an 100% medium utilization for all instances. None of the changes introduced in yawmd mediums and pthreads was made

[1]The values reported can be lower than 100%, even with full medium utilization, because there are no fractional transmissions of a frame.

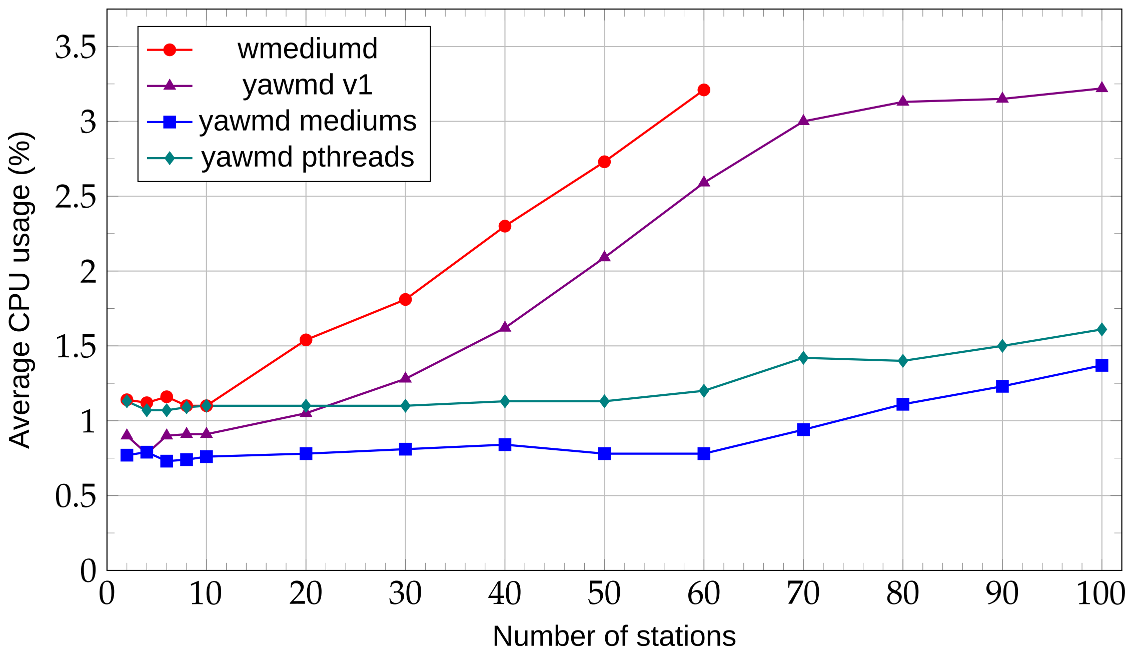

Figure 5.2: Average of the CPU used at each second during each instance of test 3.

with the intention of improving their performance, but the update of libevent[9] from version 1 to version 2 is a very likely candidate for these results [2].

Looking now at the results of CPU usage (see figure 5.2), they all increase with the number of interfaces, which is expected. Wmediumd and yawmd v1 show the fastest CPU use increase trends, but wmediumd starts with higher usage. As in medium utilization, the changes to the communication in yawmd v1 also show performance improvements. The best results are shown by yawmd mediums and yawmd pthreads, by using less CPU and having a slower growth as the instances get larger, which, again, is most likely related to the update of the version of libevent used. The utilization of yawmd pthreads is slightly higher than yawmd mediums, which is expected, given that it is already using two threads.

Focusing now in the results of memory usage (see figure 5.3), the changes observed in the values are quite steep between instances, but rarely change during the execution of an instance, and so they are most likely related to memory management. These results allow to conclude that it is required about 3 MiB to execute wmediumd and yawmd.

Taking into consideration the results of medium, CPU and memory utilization, yawmd mediums is the best option, because it combines an 100% medium utilization with the lowest CPU usage.

[2]The improvement would then be associated with a more efficient management of the events of reception of messages from mac80211_hwsim, and of the timer that signals the response (with the delivery information) to mac80211_hwsim, as described in section 2.3.3.

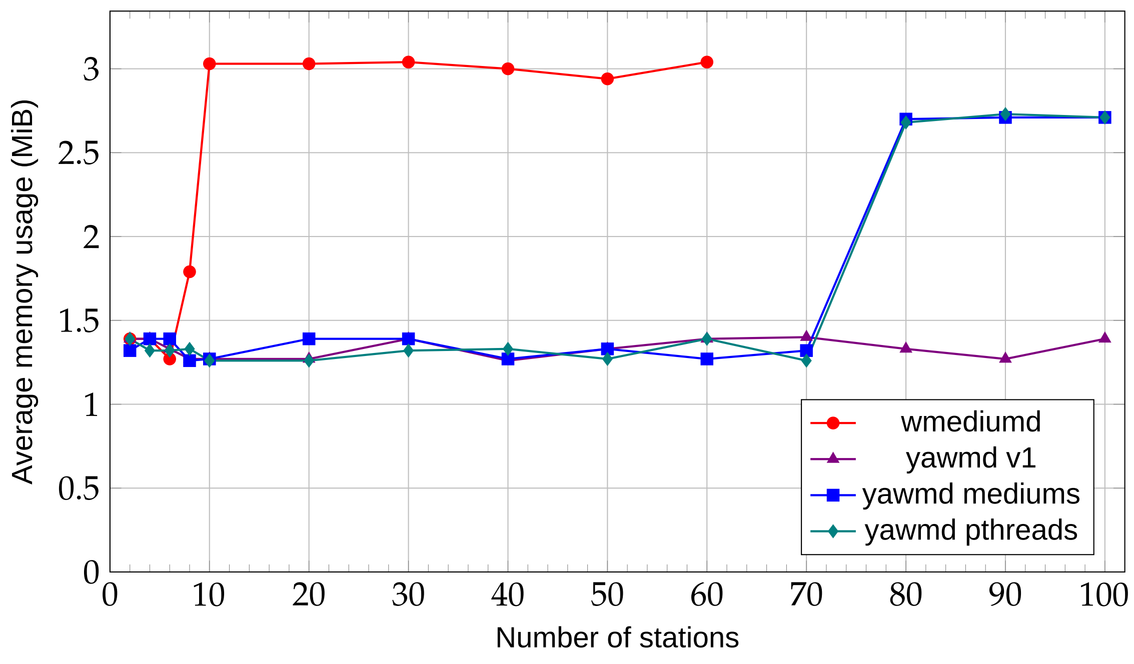

Figure 5.3: Average of the memory used at each second during each instance of test 3.

5.2 Test 4: Multiple mediums

The objective of this test is to compare the performance of yawmd mediums and yawmd pthreads when multiple mediums are being simulated. The methodology is the same used in section 5.1 (Test 3: One medium).

5.2.1 Configuration

The configuration was the same used in section 5.1 (Test 3: One medium) with the following differences.

The test was executed with two variations, *yawmd mediums* and *yawmd pthreads*, where each variation uses the program of their name. Once again, they will only be referred by their names.

The instances have two variables, M, the number of mediums, and S, the number of stations in each medium. $M \in \{1, 2, \dots, 10\}$ and $S \in \{10, 20\}$. Figure 5.4 shows an example of an instance with $S = 4$ and $M = 2$.

All stations are in the same frequency channel, and all stations that belong to a medium share the same IBSS network, but each medium has a unique IBSS network.

5.2.2 Results

The *medium utilization* values used in the analysis were derived from the values of total transmission time by QoS queue reported by yawmd. These values provide the average *medium utilization* of each medium. It was then made an average of this value, deriving the average of

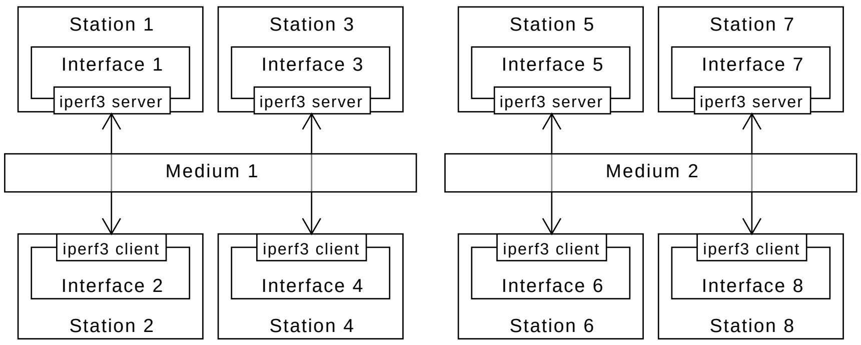

Figure 5.4: Example of the setup used in the test 4, where the number of stations per medium is $S = 4$, and the number of mediums is $M = 2$.

the mediums average *medium utilization*, so that the *medium utilization* can be seen as a value $\leq 100\%$, independently of the number of mediums of the instance.

This test was run twice. The results of run 1 are plotted at figures 5.5 to 5.7. A lower level view of the data of medium utilization by QoS queue plotted at figure 5.5, is available at appendix F.2.2. Additionally, the results of medium, CPU and memory usage, for both runs, with information of mean and standard deviation, are available at appendix F.1.2.

This test was made with the CPU Precision Boost and Simultaneous Multithreading enabled, which adds some uncertainty to the results, because these settings can have an effect in the magnitude of the values reported, and in the difference between the *variations*.

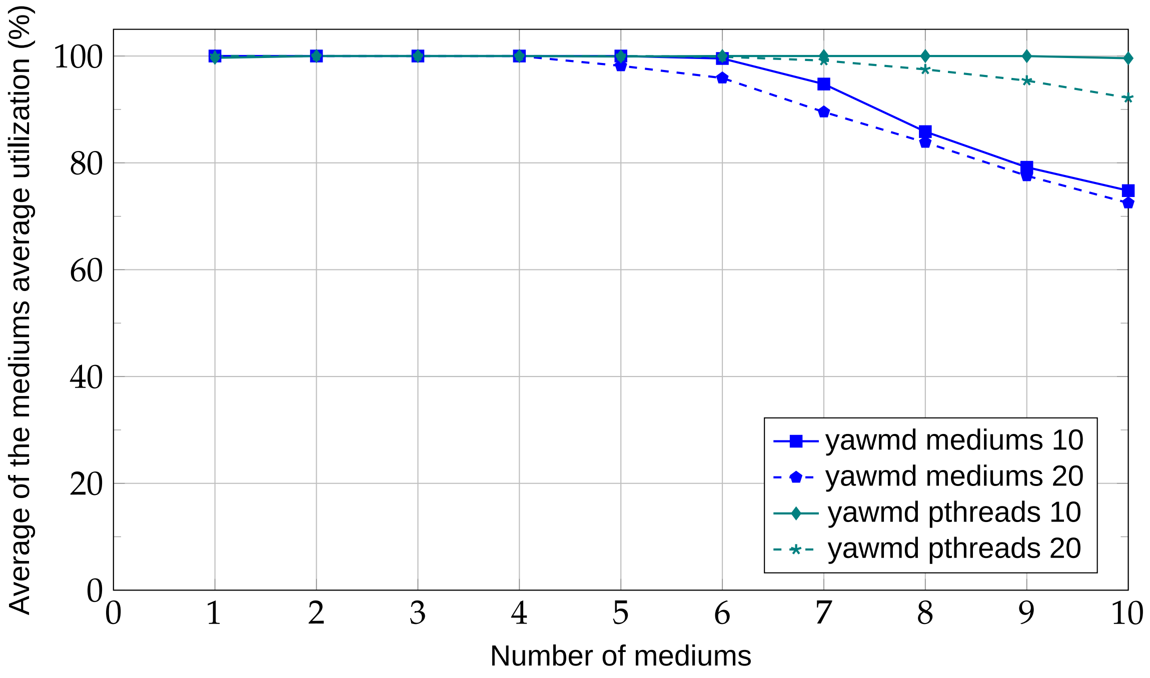

Figure 5.5: Average of the mediums average *medium utilization* at each second during each instance of test 4.

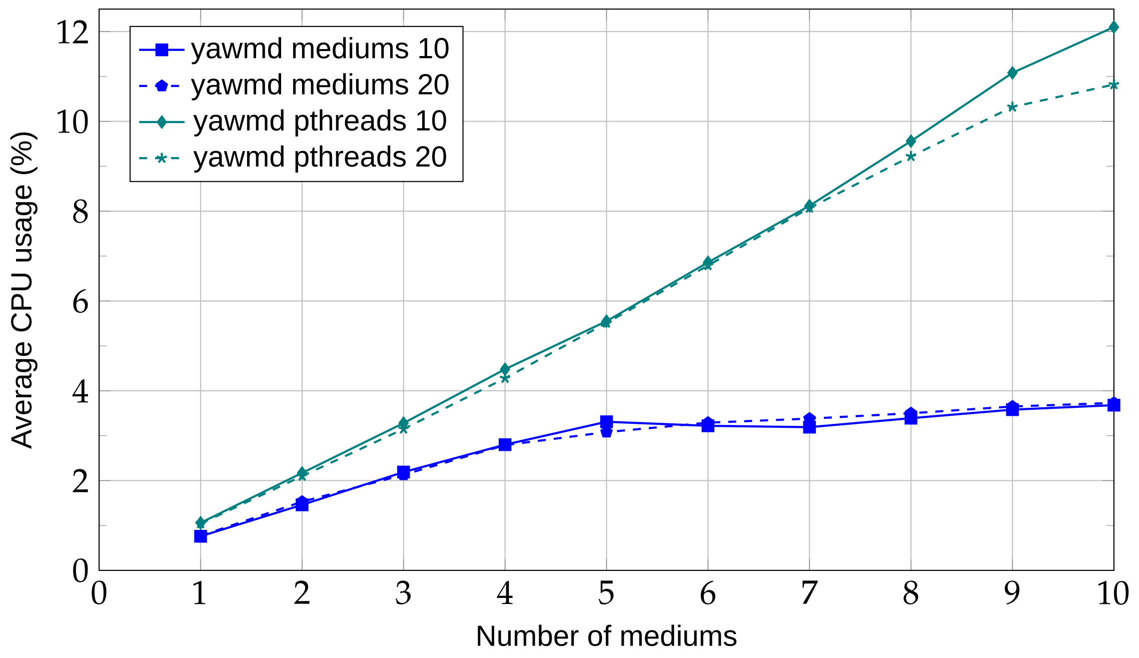

Figure 5.6: Average of the CPU use at each second during each instance of test 4.

5.2.3 Results analysis

Looking at the results of medium utilization, (see figure 5.5), yawmd pthreads performed better in both cases, with $S = 10$ and $S = 20$ interfaces per medium. With $S = 10$, yawmd pthreads was capable of maintaining a medium utilization of 100% up to $M = 10$. This means that with $S = 10$ and $M = 10$, $S \times M = 100$ stations, has the same result of test 3 with $N = 100$ (which would be equivalent to $M = 1$ and $S = 100$ in this test). This same result, with $S = 10$, is not achieved by yawmd mediums, as it shows results below 100% with $M >= 7$. With $S = 20$, neither was able to maintain 100% medium utilization with the larger values of M, but the medium utilization of yawmd pthreads shows a slower drop and starts with bigger values of M. It is clear with yawmd mediums that the drop is caused by overheads, because yawmd pthreads achieved better results. The source of the drop of yawmd pthreads with $S = 20$ is not clear. It is not associated with a bottleneck in the communication link with mac80211_hwsim, because in that case the addition of a new medium would cause a more pronounced drop in the values. It may be associated with performance issues of mac80211_hwsim or yawmd, but there other possible participants in the issue, such as the dozens of iperf clients and servers used.

Looking now at the results of CPU usage (see figure 5.6), they all increase with the number of mediums, M, which is expected. The differences between yawmd mediums and yawmd pthreads are more expressive in this test, with yawmd pthreads showing the most intensive use, exposing the tradeoff between medium utilization, scalability and CPU usage. The rate of increase of the CPU usage of yawmd mediums with $M \geq 5$ reduces, which matches with drop of medium utilization. The same happens with yawmd pthreads, comparing the results of $S = 20$ with $S = 10$, the CPU use of $S = 20$ deviates from the one of $S = 10$ at $M \geq 8$, where the

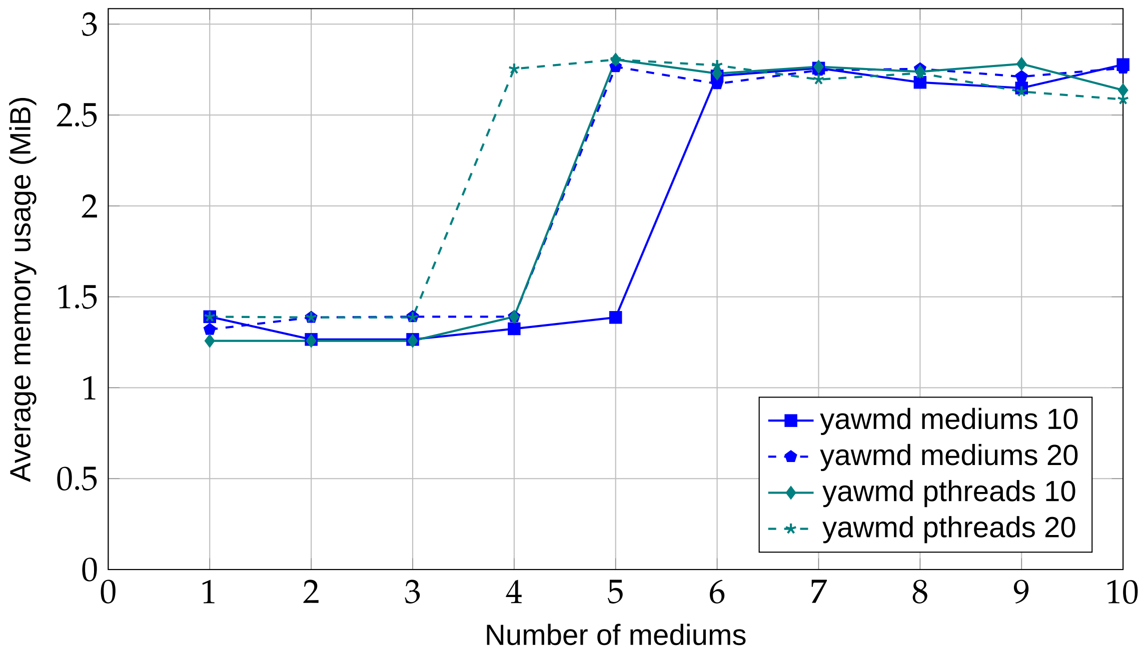

Figure 5.7: Average of the memory use at each second during each instance of test 4.

reduction of medium utilization becomes more visible. It is not clear what is the source of this behaviour.

Focusing now in the results of memory usage (see figure 5.7), the changes observed in the values are quite steep between instances, but rarely change during the execution of an instance, and so they are most likely related to memory management. These results allow to conclude that it is required about 3 MiB to execute yawmd mediums and yawmd pthreads, the same value determined in test 3 with one medium.

Putting together the results of medium, CPU and memory utilization, yawmd pthreads shows the best medium utilization and scalability, but at the expense of higher CPU usage. Yawmd pthreads has an edge with higher M, and is the best choice if the higher CPU usage is acceptable. When CPU usage is more contrained, yawmd might be the best choice.

5.3 Results summary and final considerations

Having looked at the results of test 3, with one medium, and of test 4, with multiple mediums, it is clear that yawmd v1 provides a good improvement over wmediumd, and that yawmd mediums provides a further improvement over yawmd v1, not only in terms of performance, but also with the added support for multiple mediums. While the results point to yawmd mediums being a great choice for configurations with one medium, the choice for configurations with multiple mediums, between yawmd mediums and yawmd pthreads, is more dependent on the execution environment and the configuration.

An important factor to consider when looking at the results of the tests, is that the data was collected in only one machine. Different machines have different architectures and difference performance, and that might cause one version of the program to perform better or to perform worse than what is shown here, which might change what version is more adequate for the situation. Another factor to consider is that the analysis is based on the results of only two runs of the tests. Nevertheless, there is confidence in the results presented, because not only there are no significant changes between runs, but also there are no sudden changes in the values between instances. In short, these results serve the purpose of showing improvements, not of their quantification.

Chapter 6

Conclusions

The primary objective of this thesis was to provide performance and scalability improvements for wmediumd. This was achieved with the reduction of data used in the delivery simulation protocol, in yawmd v1, and with the introduction of support to multiple mediums, with yawmd mediums and yawmd pthreads.

Taking now a closer look at the objectives as enumerated in the introduction, the objective to study other communication options, 1, was replaced by the introduction of multiple medium support, in yawmd mediums and yawmd pthreads. The objective of creating tests to the various functionalities provided, 2, was partially achieved with the tests in chapters 3 and 5. Objectives 3 (b) and (d) pertaining documentation were achieved in chapter 4 and section 2.3, respectively; and (a), related to the simulation requirements, was also achieved in chapter 4, otherwise the changes introduced with yawmd would not be possible. The objective of reducing the communication load, 4, was achieved with yawmd v1 (section 4.1). The improvement to the traffic delivery capacity, objective 5, was also achieved, as the data from chapter 5 shows, with the improvement of medium utilization with one medium, and with the support for multiple mediums. Finally, objective 6, maintaining interoperability with Mininet-WiFi, was only achieved with yawmd v1 (section 4.1).

The main contributions of this thesis are a more efficient and scalable communication protocol with mac80211_hwsim (yawmd v1), two approaches to multiple mediums, one with a single thread (yawmd mediums), and another with a thread for each medium (yawmd pthreads), and the documentation of the operation and simulation of wmediumd and its interaction with mac80211_hwsim.

While several limitations of the initial tests 1 and 2 of chapter 3 were overcome in the tests 3 and 4 of chapter 5, they were not completely eliminated, in particular, the CPU configuration in the machine used to run the tests, which adds unnecessary and removable variables, and can have some influence in the results. Another limitation is the amount of repetitions (previously called runs) used in the tests, which increases the chance that some values are distant from the real expected value. Nevertheless, there is confidence in the results, because the change in the values between instances, (which are repetitions, but with different amount of stations), does not deviate much from the trend.

6.1 Future Work

While dealing with wmediumd, several areas which might be targeted for improvement were detected:

- Both wmedium and yawmd make no distinction between frequency channels to simulate the medium access and the interference. The implementation of multiple mediums makes this easier to fix.
- The current interference simulation relies only on the data transmitted in the medium to determine the probability of interference. It could be interesting and more realistic to have external sources of interference, either random or deterministic.
- The effect of interference in a frame is applied before the transmission simulation, therefore it impacts all the attempts of retransmission of a frame, while in reality it may only affect on or some attempts.
- The possibility of overlapping successful transmissions in a perfect medium sensing algorithm, as described in appendix D, because it creates an unrealistic situation with higher throughputs.

Some areas in which the contributions of this thesis can be improved are:

- Make the tests, without the limitations found, with more repetitions, and in machines with different hardware.
- Make a more detailed and deeper performance analysis to yawmd, for example with a profiler.
- Reintroduce support for Mininet-WiFi in yawmd mediums and yawmd pthreads, and add support for configurations with multiple mediums in Mininet-WiFi.
- Explore the feasibility of other options of communication between mac80211_hwsim and wmedium/yawmd and their performance.

Bibliography

[1] Fontes, Ramon R. “Mininet-WiFi: emulation platform for software-defined wireless networks = Mininet-WiFi: plataforma de emulação para redes sem fio definidas por software”. PhD thesis. Campinas, São Paulo, Brazil: Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, 2018.

[2] Fontes, Ramon R. et al. “Mininet-WiFi: Emulating software-defined wireless networks”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. Nov. 2015, pp. 384–389. DOI: `10.1109/CNSM.2015.7367387`.

[3] Friis, Harald T. “A Note on a Simple Transmission Formula”. In: *Proceedings of the IRE* 34.5 (1946), pp. 254–256. DOI: `10.1109/JRPROC.1946.234568`.

[4] Illán, Alberto Martínez. “Medium and mobility behaviour insertion for 802.11 emulated networks”. MA thesis. Universitat Politècnica de Catalunya, 2013.

[5] ITU. *Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 900 MHz to 100 GHz*. Recomendation ITU-R P.1238-2. International Telecommunication Union, 2001.

[6] Kato, Arata, Mineo Takai, and Susumu Ishihara. “Design and implementation of a wireless network tap device for IEEE 802.11 wireless network emulation”. In: *2017 Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*. Oct. 2017, pp. 1–6. DOI: `10.23919/ICMU.2017.8330098`.

[7] Lantz, Bob, Brandon Heller, and Nick McKeown. “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks”. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: Association for Computing Machinery, 2010. DOI: `10.1145/1868447.1868466`.

[8] *Libconfig – library for processing configuration files*. URL: `https://hyperrealm.github.io/libconfig/`.

[9] *Libevent – an event notification library*. URL: `https://libevent.org/`.

[10] *Libnl – Netlink Library Suite*. URL: `https://github.com/thom311/libnl`.

[11] *perf – Performance analysis utility*. URL: `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/perf`.

[12] Rappaport, Teodore S. *Wireless Communications: Principles and Practices*. Prentice Hall PTR, 2002, pp. 138–141. ISBN: 9780130422323.

[13] Rethfeldt, Michael et al. "ViPMesh: A virtual prototyping framework for IEEE 802.11s wireless mesh networks". In: *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Oct. 2016, pp. 1–7. DOI: `10.1109/WiMOB.2016.7763263`.

[14] Silvano, Gilles et al. "A Hybrid Architecture for Experimentation in Wireless Sensor Networks". In: *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. Nov. 2016, pp. 116–121. DOI: `10.1109/SBESC.2016.025`.

[15] *Source code of mac80211_hwsim*. Linux Kernel Organization, Inc. URL: `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/net/wireless/mac80211_hwsim.c`.

[16] *Source code of wmediumd used in Mininet-WiFi*. URL: `https://github.com/ramonfontes/wmediumd`.

[17] *sysstat – System performance tools for the Linux operating system*. URL: `https://github.com/sysstat/sysstat`.

[18] Tchinda, A. Paguem et al. "Performance analysis of WMN routing protocols for disaster networks". In: *2017 IEEE Symposium on Communications and Vehicular Technology (SCVT)*. IEEE. 2017, pp. 1–6. DOI: `10.1109/SCVT.2017.8240309`.

Appendix A

Test 1 complementary plots

A.1 Variation H

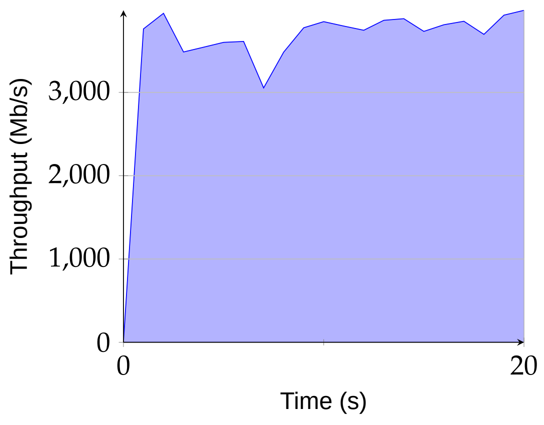

Figure A.1: Throughput during *variation H* with 2 stations.

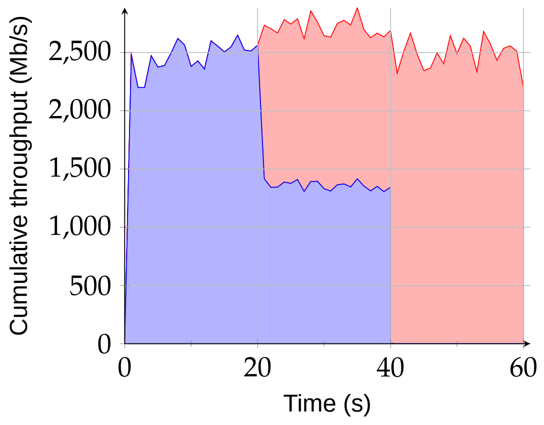

Figure A.2: Cumulative throughput during *variation H* with 4 stations.

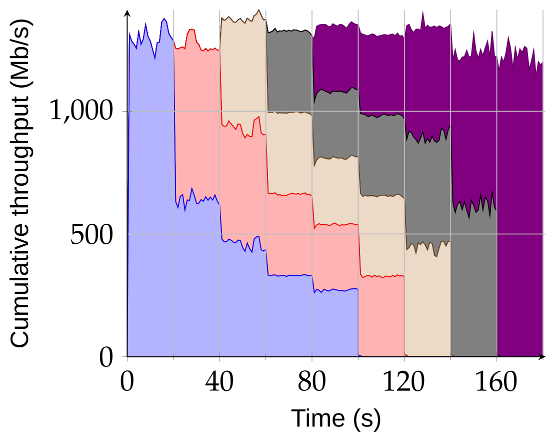

Figure A.3: Cumulative throughput during *variation H* with 10 stations.

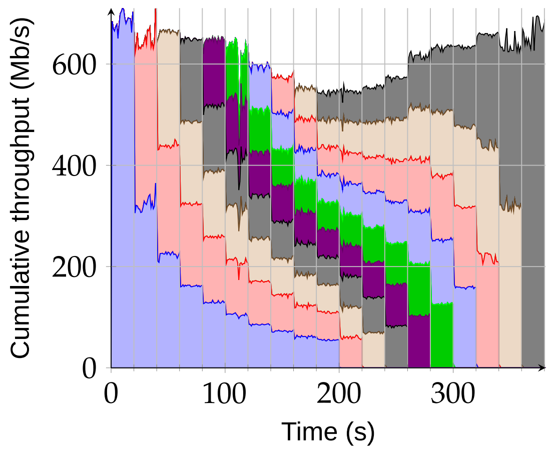

Figure A.4: Cumulative throughput during *variation H* with 20 stations.

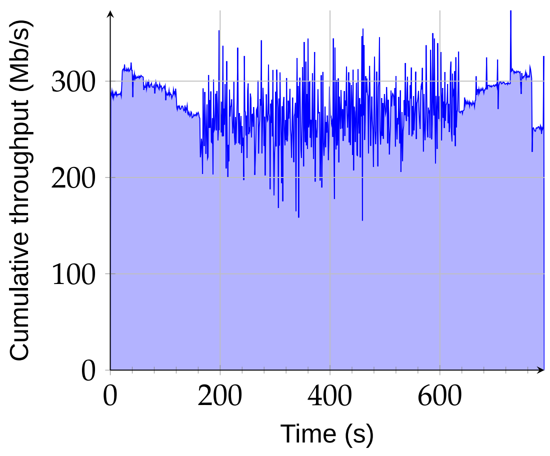

Figure A.5: Cumulative throughput during *variation H* with 40 stations.

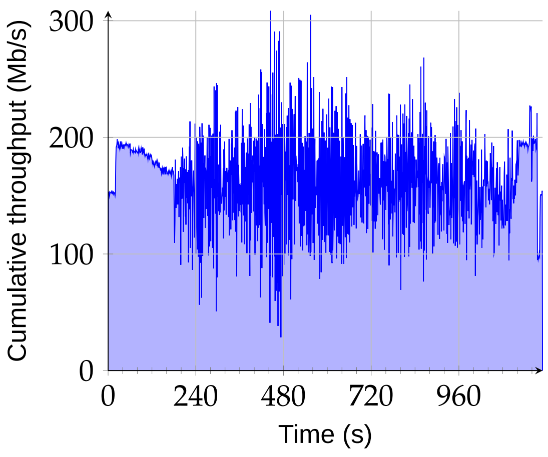

Figure A.6: Cumulative throughput during *variation H* with 60 stations.

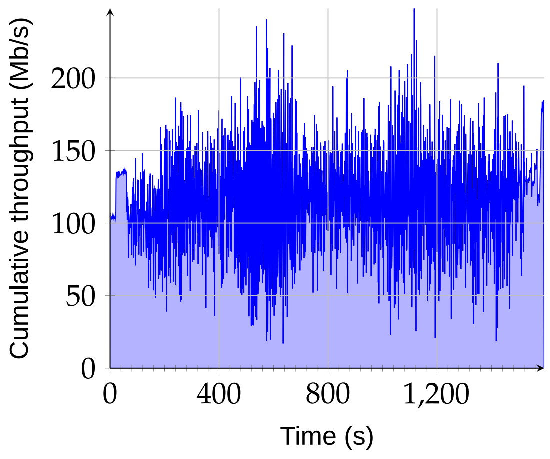

Figure A.7: Cumulative throughput during *variation H* with 80 stations.

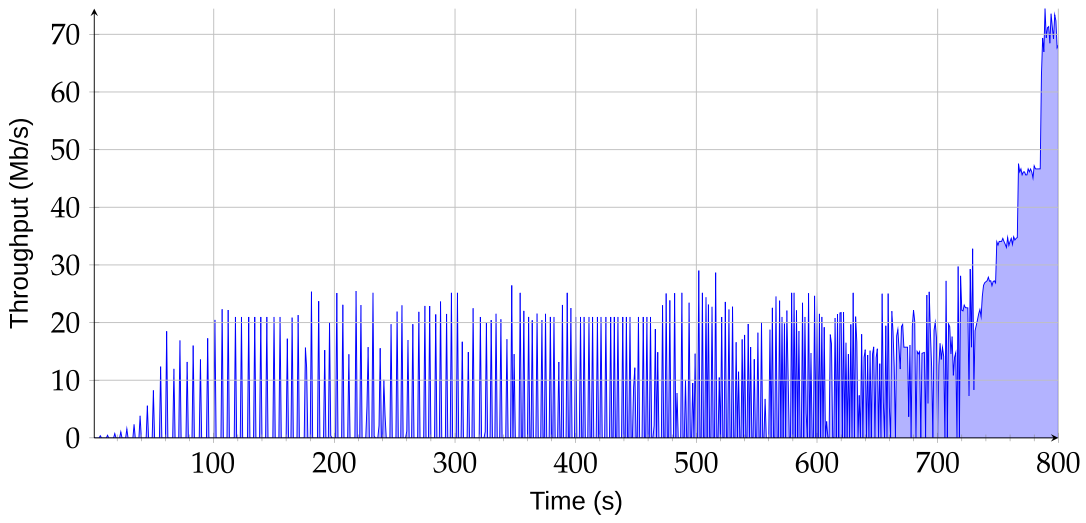

Figure A.8: Throughput values of the last pair to start communicating with 80 stations.

A.2 Variation W

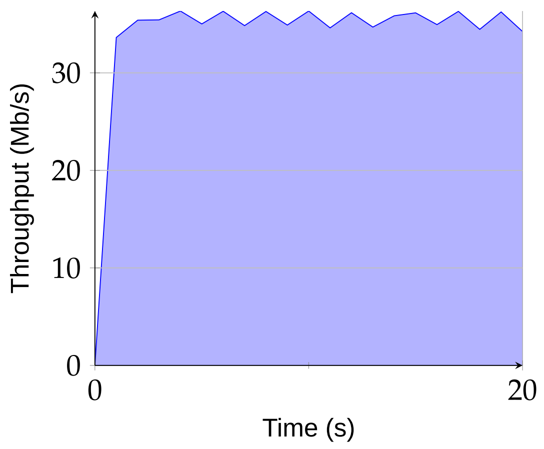

Figure A.9: Throughput during *variation W* with 2 stations.

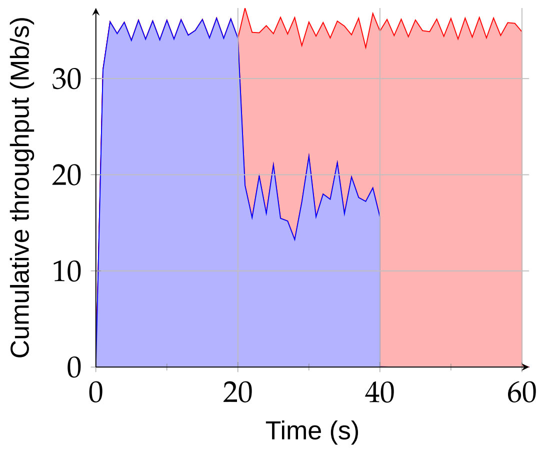

Figure A.10: Cumulative throughput during *variation W* with 4 stations.

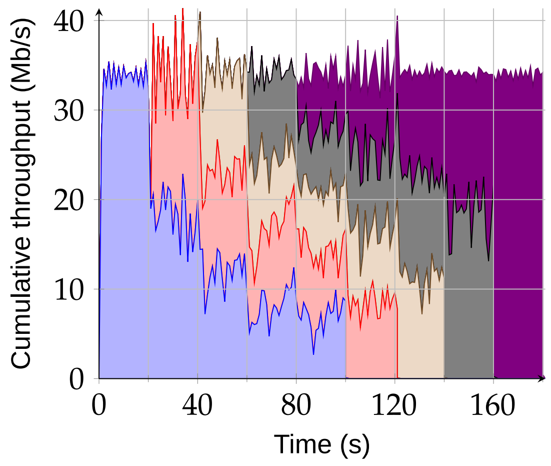

Figure A.11: Cumulative throughput during *variation W* with 10 stations.

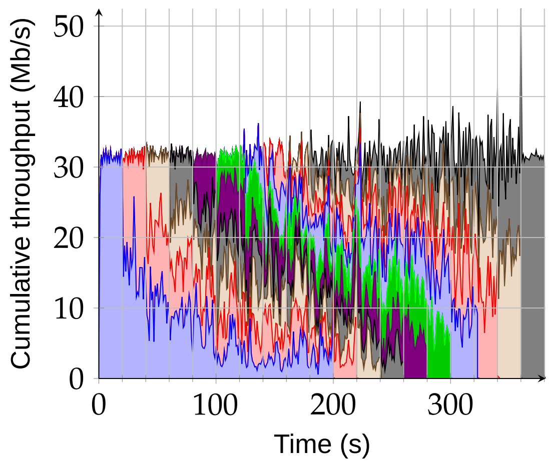

Figure A.12: Cumulative throughput during *variation W* with 20 stations.

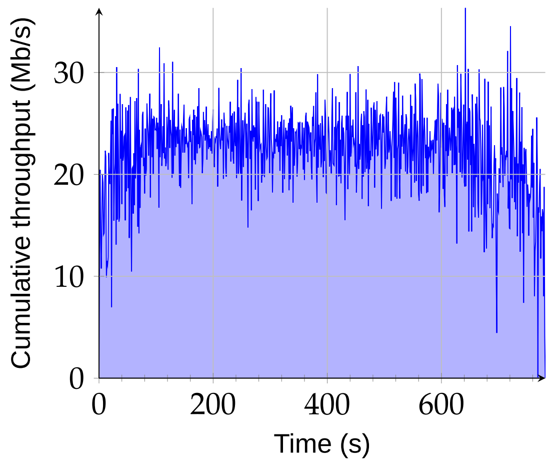

Figure A.13: Cumulative throughput during *variation W* with 40 stations.

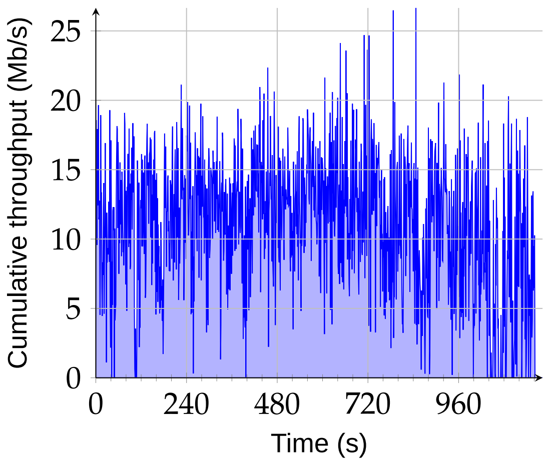

Figure A.14: Cumulative throughput during *variation W* with 60 stations.

Appendix B

Test 2 complementary plots

B.1 Variation H

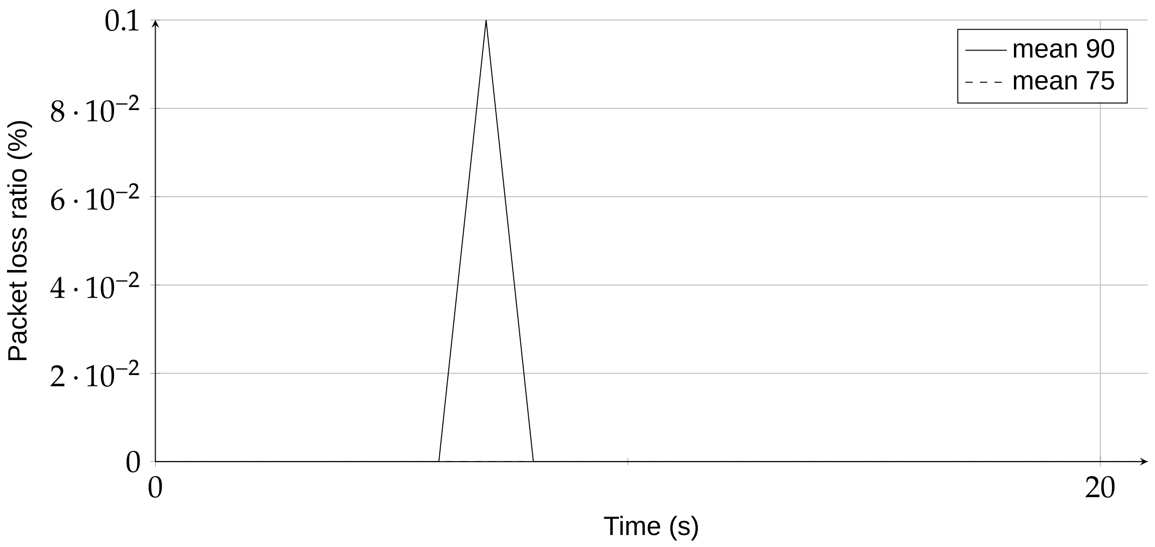

Figure B.1: Packet loss ratio during *variation H* with 2 stations.

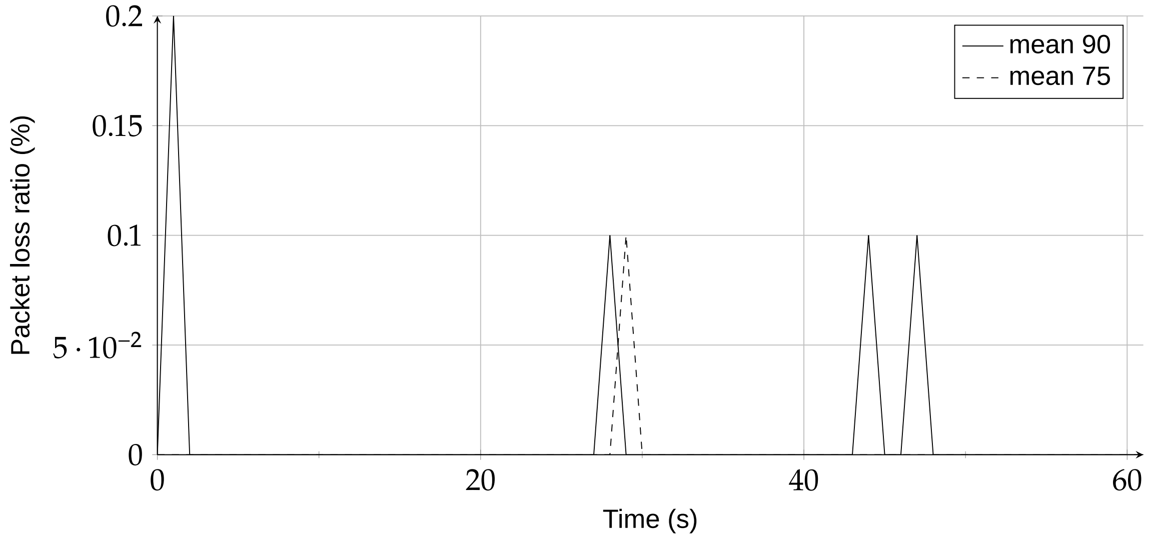

Figure B.2: Packet loss ratio during *variation H* with 4 stations.

B.2 Variation W

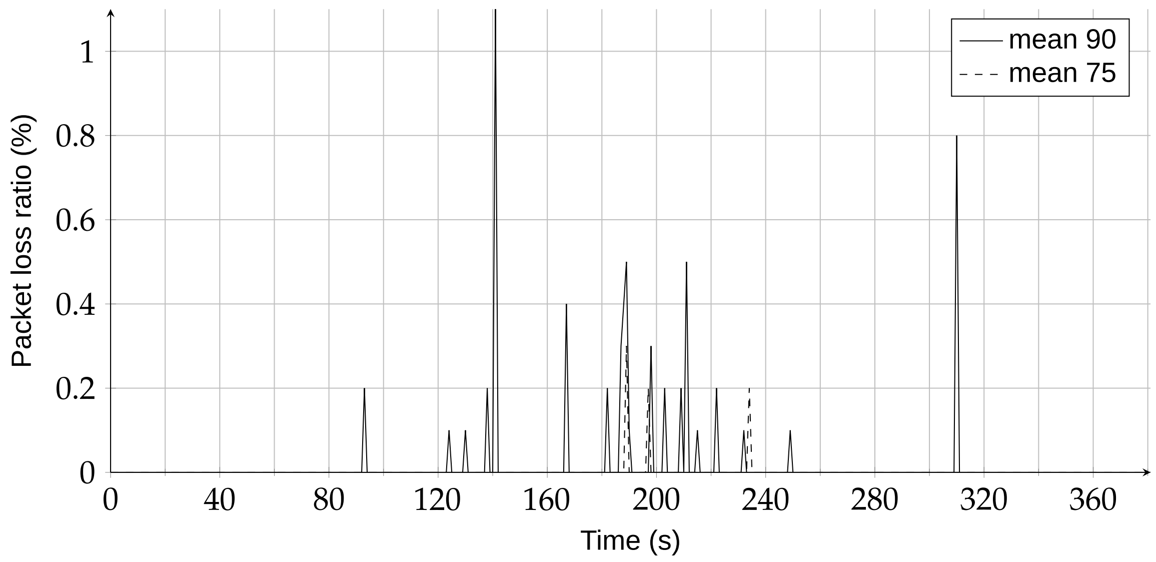

Figure B.3: Packet loss ratio during *variation W* with 20 stations.

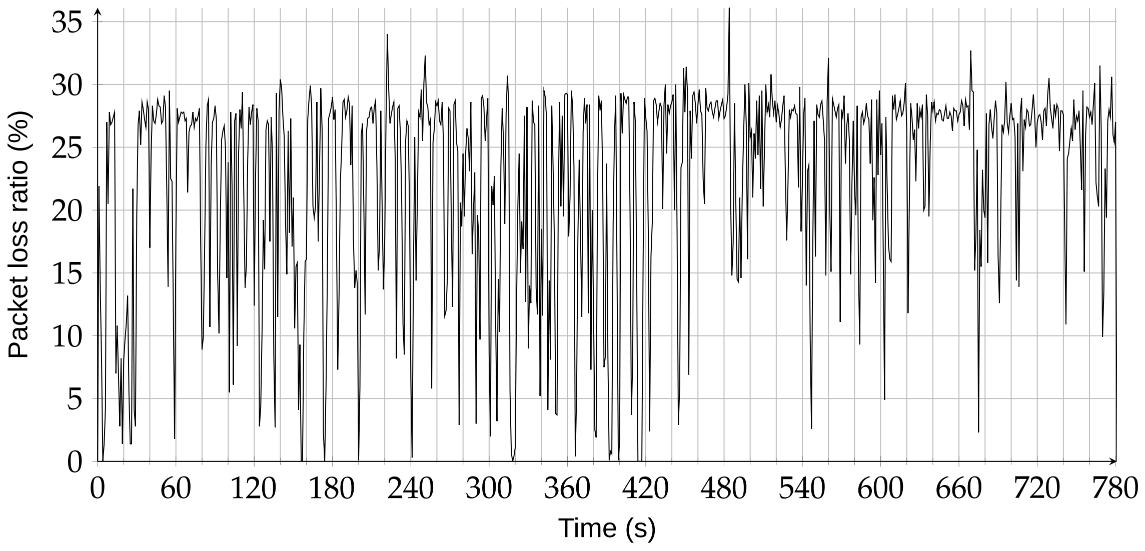

Figure B.4: Packet loss ratio during *variation W* with 40 stations, with target data rate *mean 75*.

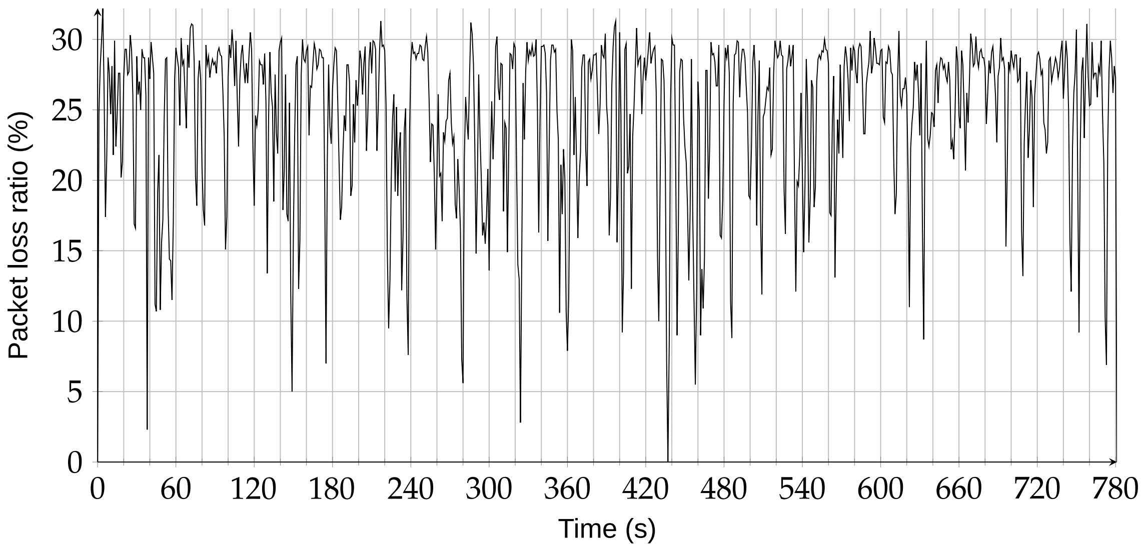

Figure B.5: Packet loss ratio during *variation W* with 40 stations, with target data rate *mean 90*.

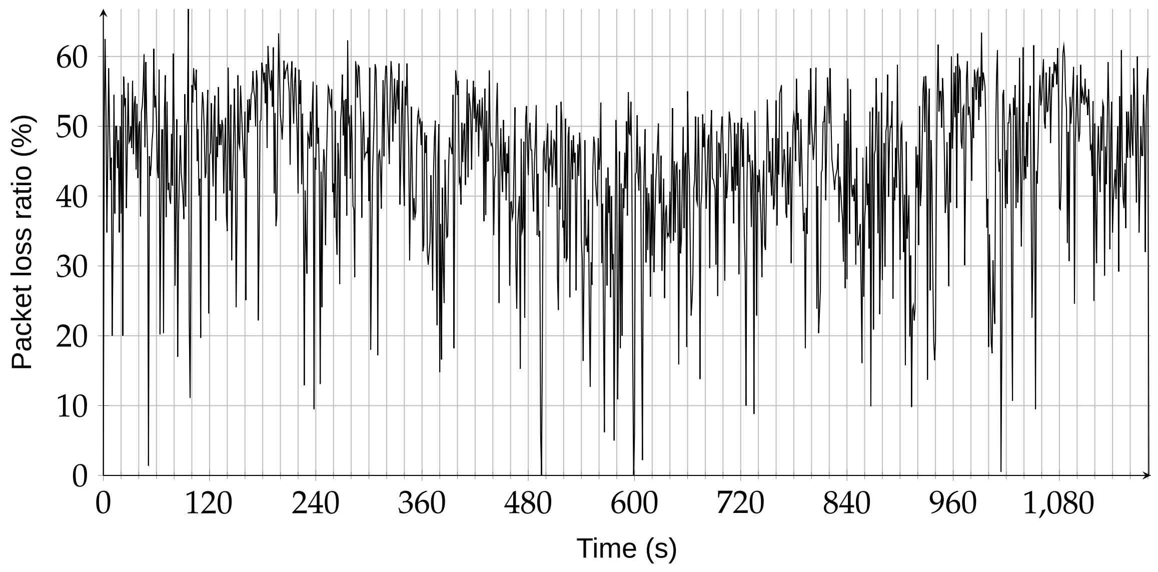

Figure B.6: Packet loss ratio during *variation W* with 60 stations, with target data rate *mean 75*.

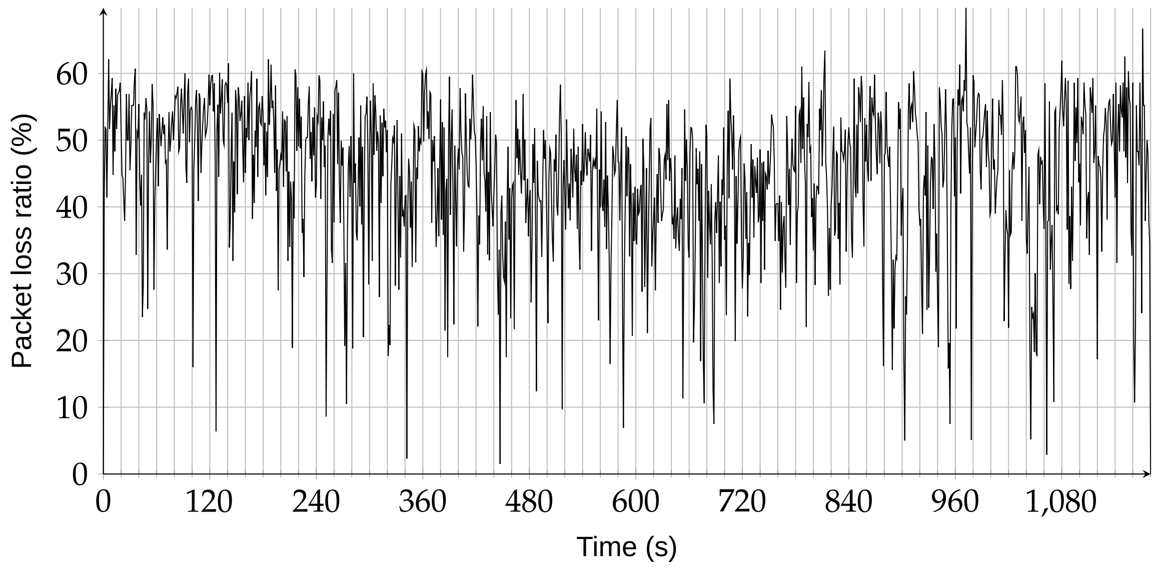

Figure B.7: Packet loss ratio during *variation W* with 60 stations, with target data rate *mean 90*.

Appendix C

Configuration files for wmediumd and yawmd

C.1 wmediumd configuration file options

Listing C.1: Configuration file options for wmediumd.

```
# ifaces.ids is required. The other settings are optional.
ifaces:
{
  ids = ["<mac address 1>", "<mac address 2>", "<mac address 3>"];
  enable_interference = <true | false>;
  # indices correspond to the ifaces.ids position
  links = ((<source index>, <destination index>, <snr>), ...);
};

# All the following are optional, but some are required once the
# setting commented above is present.
model:
{
  noise_threshold = <int>;
  fading_coefficient = <int>;
  type = < "snr" | "prob" | "path_loss" >;
  # .type = "path_loss"
  name = < "free_space" | "log_distance" | "log_normal_shadowing" |
           "two_ray_ground" | "itu" >;
  # Optional: used by .type = < "snr" | "prob" >
  links = ((<source>, <destination>, <snr value | prob value>), ...);
  # Optional: used by .type = "prob"
  default_prob = <float>;
  # Optional: used by .type = "snr"
  default_snr = <int>;
  # .type = "path_loss"
  positions = ((<float_x>, <float_y>, <float_z>), ...);
  # .type = "path_loss"
  tx_powers = [<float>, <float>, ...];
  # Optional: used by .type = "path_loss"
  directions = ((<float_x>, <float_y>, <float_z>), ...);
  # Optional: used by .type = "path_loss"
  isnodeaps = [<int>, <int>, ...];
  ## The following are required if type = "path_loss" and if name=x
  # .name = < "log_distance" | "log_normal_shadowing" >
  path_loss_exp = <float>;
  # .name = "log_distance"
  xg = <float>;
  # .name = <"free_space"| "log_normal_shadowing"| "two_ray_ground">
  sL = <int>;
  # .name = "itu"
  nFLOORS = <int>; lF = <int>; pL = <int>;
};
```

C.2 yawmd configuration file options

Listing C.2: Configuration file used by yawmd mediums and yawmd pthreads.

```
medium =
(
  {
    # required
    id = 2;
    # required
    interfaces = ["00:00:00:00:00:00",
                  "00:00:00:00:00:01",
                  "00:00:00:00:00:02",
                  "00:00:00:00:00:03"];
    # required
    model =
    {
      # required
      type = "prob";
      # optional - (default_probability = 1.0)
      # probability defines the ERROR probability
      default_probability = 0.5; # for the other interfaces
      # optional - all missing pairs have default_probability
      links =
      (#(transmitter, receiver, error probability)
        (0, 1, 0.1),
        (1, 0, 0.2),
        (2, 0, 0.005),
        (1, 3, 0.001)
      );
    }
  },

  {
    # required
    id = 1;
    # required
    interfaces = ["00:00:00:00:00:04",
                  "00:00:00:00:00:05",
                  "00:00:00:00:00:06"];
    # required
    model =
    {
      # required
      type = "snr";
      # optional - default_snr = -100
      default_snr = 110;
      # optional - all missing pairs have default_snr
      links =
      (#(transmitter, receiver, receiver signal)
        (0, 1, 110),
        (1, 0, 110),
        (1, 2, 50)
      );
    }
  },
  {
    # required
    id = 3;
    # required
    interfaces = ["00:00:00:00:00:07",
                  "00:00:00:00:00:08",
                  "00:00:00:00:00:09",
                  "00:00:00:00:00:0a"];
    # required
    model =
    {
      # required
      type = "path_loss";
      # optional - simulate_interference = false;
      simulate_interference = true;
      # optional - noise_level = -91
      noise_level = -91; # dBm
      # optional - fading_coefficient = 0
      fading_coefficient = 1;
      # required - (x,y,z). z for two_ray is antenna height
      # units: meters
      positions = ((2.0, 3.0, 8.0), (4.0, 5.0, 0.0),
                   (1.0, 1.0, 0.0), (2.3, 5.1, 1.3));
      # optional (seconds > 0) move_interval = 5.0
      move_interval = 1.0;
      # optional. every move_interval: position += direction
      directions = ((2.0, 3.0, 2.0), (5.0, 1.0, 1.0),
                    (0.0, 0.0, 0.0), (1.0, 0.0, 0.0));
      # required
      tx_powers = [15, 20, 10, 30]; # int
      # optional antenna_gain = 0
      antenna_gain = [3, 3, 5, 8]; # int - dBm
      # required: free_space | itu | log_distance |
      #           log_normal_shadowing | two_ray_ground
      # model_name = "free_space"; # string
      # model_name = "log_distance";
      # model_name = "log_normal_shadowing";
      model_name = "two_ray_ground";
      # model_name = "itu";
      # required - model_name parameters
      # model_params = # free_space
      # {
      #   system_loss = 1; # int
      # }
      # model_params = # log_distance
      # {
      #   path_loss_exponent = 0.1; # float
      #   xg = 1.0; # float
      # }
      # model_params = # log_normal_shadowing
      # {
      #   path_loss_exponent = 0.1; # float
      #   system_loss = 1; # int
      # }
      model_params = # two_ray_ground
      {
        system_loss = 1; # int
      }
      # model_params = # itu
      # {
      #   n_floors = 0; # int
      # # floor penetration factor
      # # floor_pen_factor * n_floors
      #   floor_pen_factor = 22; # int dBm
      # # distance power loss coefficient (N)
      #   power_loss_coefficient = 1; #int
      # }
    }
  }
);
```

Appendix D

Medium access simulation simultaneous transmission

The medium access simulation determines when an interface will transmit a frame. It is described in more detail at section 2.3.3. This algorithm suffers from a problem, because it allows two or more frames to be transmitted at the same time, thus violating the objective of only one frame be transmitted at one time. If a frame with higher priority arrives after a frame with lower priority is already queued and not delivered, when calculating the delivery timestamp, the domain of the search is only the frames with same or higher priority, thus the lower priority frame will be missed and both will be scheduled for delivery with overlapping time intervals. An example of this behaviour can be seen in the situation described in table D.1 and represented graphically in figure D.1.

Table D.1: This table represents a simplified hypothetical timeline of frame delivery that the algorithm used by wmediumd/yawmd to simulate medium access (see listing 2.2) will allow. This example shows that if frames with higher priority arrive after frames with lower priority that are still not delivered, simultaneous transmission may occur. For this example the medium is shared by two interfaces, and each interface has two QoS queues. The following abbreviations are used on the table. Ord: order of reception of the frame by wmediumd/yawmd; Rcv: timestamp of reception of the frame by wmediumd/yawmd; Str: timestamp of start of frame transmission; End: timestamp of end of frame transmission.

Interface 1								Interface 2							
High Priority				Low Priority				High Priority				Low Priority			
Ord	Rcv	Str	End	Ord	Rcv	Str	End	Ord	Rcv	Str	End	Ord	Rcv	Str	End
								9	35	41	50	8	31	51	60
6	15	31	40					4	12	13	20	3	5	21	30
5	12	21	30	7	17	41	50	1	0	1	10	2	3	11	20

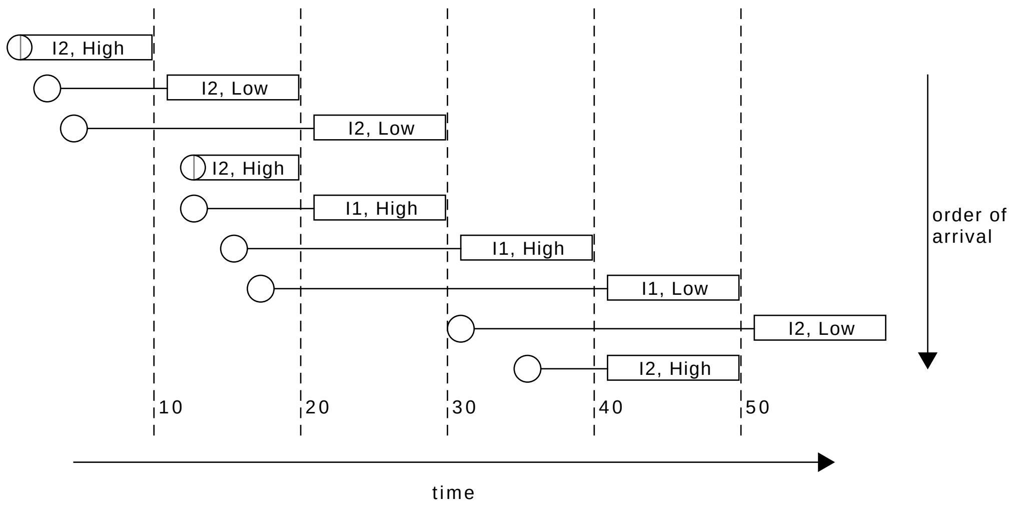

Figure D.1: Representation of the order of delivery of packages of the situation described in table D.1. The circles mark the time of reception of the frame by wmediumd/yawmd.

Appendix E

Commands used

```
## iperf3:
# server:
ip netns exec <net_namespace> iperf3 -s --json --logfile \
  <out_file> -1

# client:
ip netns exec <net_namespace> iperf3 -c <server_ip> -t <run_time> \
  --json --logfile <out_file>

## mac80211_hwsim
# kernel
modprobe mac80211_hwsim radios=<number>
modprobe -r mac80211_hwsim

# modified for yawmd
modprobe mac80211 # might not be required if it is already started
insmod <my_mac80211_hwsim.ko_path> radios=<number>
modprobe -r mac80211_hwsim
modprobe -r mac80211


## interface configuration
ip netns add <namespace>
iw phy <phy> set netns name <namespace>
ip netns exec <namespace> iw dev <wlan> set type ibss
ip netns exec <namespace> ip link set dev <wlan> up
ip netns exec <namespace> ip addr add <ip>/<mask> dev <wlan>
ip netns exec <namespace> ip route add default dev <wlan>
```

```
## create/join network
ip netns exec <namespace> iw dev <wlan> ibss join <ibss_name> \
   <channel_frequency>

## collect cpu and memory usage by a process
# pidstat is part of the sysstat package
pidstat -h -H -r -u -I -p <pid>
```

Appendix F

Test 3 and Test 4 complementary information

F.1 Table summaries of the data used in chapter 5

F.1.1 Test 3: One medium

Medium utilization

Table F.1: Results of wmediumd by QoS queue. Values of mean, μ, standard deviation, σ, and $\mathit{medium\ utilization} = VO_{\mu} + BE_{\mu}$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
2	1.39	0.062	97.77	6.575	99.16	1.39	0.059	96.15	14.146	97.54
4	2.77	0.119	96.08	9.014	98.85	2.76	0.126	94.49	15.231	97.25
6	4.16	0.193	94.40	11.443	98.55	4.16	0.170	94.39	11.293	98.56
8	5.54	0.237	92.72	12.174	98.26	5.55	0.229	92.71	12.157	98.26
10	6.93	0.308	91.05	12.453	97.98	6.93	0.302	91.06	12.447	97.98
20	13.85	0.550	84.26	11.551	98.11	13.86	0.490	84.26	11.586	98.11
30	20.81	0.763	77.44	10.746	98.25	20.78	0.774	77.47	10.653	98.24
40	27.71	0.898	68.61	13.329	96.31	27.75	0.858	70.74	7.048	98.49
50	34.65	0.992	32.25	16.050	66.90	34.57	0.995	25.33	12.617	59.90
60	41.38	1.534	29.08	11.746	70.47	41.37	1.332	20.43	13.306	61.80
70	-	-	-	-	-	-	-	-	-	-
80	-	-	-	-	-	-	-	-	-	-
90	-	-	-	-	-	-	-	-	-	-
100	-	-	-	-	-	-	-	-	-	-

Table F.2: Results of yawmd v1 by QoS queue. Values of mean, μ, standard deviation, σ, and $medium\ utilization = VO_\mu + BE_\mu$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
2	1.39	0.062	97.78	6.526	99.17	1.39	0.060	97.77	6.592	99.16
4	2.77	0.116	96.08	8.990	98.85	2.77	0.119	96.08	8.976	98.85
6	4.16	0.183	94.39	11.427	98.54	4.16	0.176	94.39	11.336	98.55
8	5.55	0.227	92.70	12.166	98.25	5.55	0.230	92.72	12.142	98.27
10	6.93	0.286	92.57	3.874	99.50	6.93	0.274	92.57	3.935	99.50
20	13.85	0.505	84.24	11.550	98.09	13.85	0.527	85.67	3.632	99.52
30	20.81	0.697	77.45	10.715	98.26	20.81	0.686	77.44	10.732	98.26
40	27.70	0.962	70.69	9.750	98.39	27.70	0.865	71.87	3.478	99.57
50	34.63	0.998	63.90	8.851	98.53	34.64	0.965	63.89	8.925	98.53
60	40.83	1.561	57.82	8.273	98.65	40.56	1.818	59.07	3.376	99.63
70	46.86	2.715	49.61	8.398	96.47	45.68	3.227	51.69	8.393	97.37
80	49.08	4.959	44.59	9.281	93.67	48.88	5.401	45.63	7.157	94.51
90	45.27	7.939	45.09	9.280	90.36	45.09	7.222	45.52	8.521	90.61
100	42.44	6.741	44.27	7.963	86.71	42.17	7.536	45.40	9.552	87.57

Table F.3: Results of yawmd mediums by QoS queue. Values of mean, μ, standard deviation, σ, and $medium\ utilization = VO_\mu + BE_\mu$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
2	1.39	0.060	96.16	14.121	97.55	1.39	0.060	97.77	6.555	99.16
4	2.77	0.121	96.08	8.984	98.85	2.77	0.121	96.08	9.012	98.85
6	4.16	0.174	94.39	11.291	98.56	4.16	0.180	94.39	11.358	98.55
8	5.55	0.225	92.71	12.155	98.26	5.54	0.234	92.72	12.151	98.26
10	6.94	0.266	91.04	12.479	97.98	6.93	0.297	91.05	12.449	97.98
20	13.87	0.508	84.24	11.593	98.11	13.85	0.504	85.68	3.760	99.53
30	20.78	0.800	77.48	10.653	98.26	20.80	0.699	78.75	3.629	99.55
40	27.75	0.805	70.66	9.820	98.41	27.73	0.797	70.68	9.775	98.41
50	34.68	0.902	63.87	8.883	98.56	34.63	1.096	64.96	3.030	99.59
60	41.55	1.463	57.11	7.986	98.66	41.56	1.089	57.14	7.921	98.69
70	48.44	1.413	50.42	7.080	98.87	48.46	1.336	51.26	2.740	99.72
80	55.19	1.791	43.82	6.312	99.01	55.18	1.734	44.57	2.739	99.75
90	60.96	2.111	38.15	5.770	99.11	60.90	2.180	38.87	2.905	99.77
100	61.36	2.911	38.38	3.517	99.74	61.43	3.174	37.61	6.151	99.05

Table F.4: Results of yawmd pthreads by QoS queue. Values of mean, μ, standard deviation, σ, and $medium\ utilization = VO_{\mu} + BE_{\mu}$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
2	1.39	0.060	96.16	14.121	97.55	1.39	0.060	97.77	6.555	99.16
4	2.77	0.121	96.08	8.984	98.85	2.77	0.121	96.08	9.012	98.85
6	4.16	0.174	94.39	11.291	98.56	4.16	0.180	94.39	11.358	98.55
8	5.55	0.225	92.71	12.155	98.26	5.54	0.234	92.72	12.151	98.26
10	6.94	0.266	91.04	12.479	97.98	6.93	0.297	91.05	12.449	97.98
20	13.87	0.508	84.24	11.593	98.11	13.85	0.504	85.68	3.760	99.53
30	20.78	0.800	77.48	10.653	98.26	20.80	0.699	78.75	3.629	99.55
40	27.75	0.805	70.66	9.820	98.41	27.73	0.797	70.68	9.775	98.41
50	34.68	0.902	63.87	8.883	98.56	34.63	1.096	64.96	3.030	99.59
60	41.55	1.463	57.11	7.986	98.66	41.56	1.089	57.14	7.921	98.69
70	48.44	1.413	50.42	7.080	98.87	48.46	1.336	51.26	2.740	99.72
80	55.19	1.791	43.82	6.312	99.01	55.18	1.734	44.57	2.739	99.75
90	60.96	2.111	38.15	5.770	99.11	60.90	2.180	38.87	2.905	99.77
100	61.36	2.911	38.38	3.517	99.74	61.43	3.174	37.61	6.151	99.05

CPU and memory usage

Table F.5: Results of wmediumd CPU and memory usage. Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
2	1.14	0.08	1.39	0.00	1.14	0.11	1.27	0.00
4	1.12	0.16	1.39	0.00	1.16	0.13	1.27	0.00
6	1.16	0.10	1.27	0.00	1.02	0.22	2.32	0.84
8	1.10	0.19	1.79	0.70	1.19	0.10	1.27	0.00
10	1.10	0.23	3.03	0.00	1.18	0.18	3.10	0.00
20	1.54	0.19	3.03	0.00	1.58	0.13	3.05	0.00
30	1.81	0.36	3.04	0.00	1.93	0.31	2.96	0.00
40	2.30	0.29	3.00	0.00	2.31	0.28	2.89	0.00
50	2.73	0.18	2.94	0.00	2.70	0.20	3.03	0.00
60	3.21	0.20	3.04	0.00	3.16	0.21	3.00	0.00
70	-	-	-	-	-	-	-	-
80	-	-	-	-	-	-	-	-
90	-	-	-	-	-	-	-	-
100	-	-	-	-	-	-	-	-

Table F.6: Results of yawmd v1 CPU and memory usage. Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
2	0.90	0.10	1.39	0.00	0.92	0.06	1.26	0.00
4	0.78	0.22	1.39	0.00	0.93	0.04	1.27	0.00
6	0.90	0.13	1.33	0.00	0.81	0.19	1.27	0.00
8	0.91	0.06	1.27	0.00	0.93	0.04	1.33	0.00
10	0.91	0.11	1.27	0.00	0.92	0.09	1.32	0.00
20	1.05	0.09	1.27	0.00	1.04	0.10	1.27	0.00
30	1.28	0.13	1.39	0.00	1.28	0.14	1.39	0.00
40	1.62	0.21	1.26	0.00	1.59	0.25	1.33	0.00
50	2.09	0.25	1.33	0.00	2.04	0.24	1.27	0.00
60	2.59	0.29	1.39	0.00	2.60	0.27	1.27	0.00
70	3.00	0.37	1.40	0.00	2.96	0.32	1.40	0.00
80	3.13	0.31	1.33	0.00	3.18	0.28	1.40	0.00
90	3.15	0.30	1.27	0.00	3.19	0.23	1.39	0.00
100	3.22	0.23	1.39	0.00	3.14	0.28	1.39	0.00

Table F.7: Results of yawmd mediums CPU and memory usage. Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
2	0.77	0.06	1.32	0.00	0.74	0.14	1.26	0.00
4	0.79	0.05	1.39	0.00	0.71	0.11	1.27	0.00
6	0.73	0.14	1.39	0.00	0.77	0.09	1.39	0.00
8	0.74	0.13	1.26	0.00	0.68	0.14	1.33	0.00
10	0.76	0.09	1.27	0.00	0.72	0.11	1.26	0.00
20	0.78	0.07	1.39	0.00	0.70	0.13	1.27	0.00
30	0.81	0.11	1.39	0.00	0.85	0.05	1.26	0.00
40	0.84	0.11	1.27	0.00	0.83	0.09	1.27	0.00
50	0.78	0.13	1.33	0.00	0.80	0.10	1.39	0.00
60	0.78	0.13	1.27	0.00	0.85	0.11	1.39	0.00
70	0.94	0.17	1.32	0.00	0.91	0.14	1.39	0.00
80	1.11	0.17	2.70	0.00	1.05	0.16	2.64	0.00
90	1.23	0.20	2.71	0.00	1.22	0.19	2.74	0.00
100	1.37	0.16	2.71	0.00	1.36	0.15	2.69	0.00

Table F.8: Results of yawmd pthreads CPU and memory usage. Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
2	1.13	0.10	1.39	0.00	1.01	0.15	1.39	0.00
4	1.07	0.10	1.32	0.00	1.08	0.10	1.26	0.00
6	1.07	0.13	1.32	0.00	1.11	0.08	1.27	0.00
8	1.09	0.10	1.33	0.00	1.15	0.10	1.27	0.00
10	1.10	0.12	1.26	0.00	1.11	0.08	1.39	0.00
20	1.10	0.10	1.26	0.00	1.10	0.11	1.39	0.00
30	1.10	0.13	1.32	0.00	1.07	0.11	1.39	0.00
40	1.13	0.13	1.33	0.00	1.18	0.11	1.39	0.00
50	1.13	0.08	1.27	0.00	1.22	0.10	1.26	0.00
60	1.20	0.09	1.39	0.00	1.30	0.09	1.32	0.00
70	1.42	0.13	1.26	0.00	1.37	0.10	1.39	0.00
80	1.40	0.13	2.68	0.00	1.50	0.09	2.79	0.00
90	1.50	0.17	2.73	0.00	1.57	0.17	2.67	0.00
100	1.61	0.18	2.71	0.00	1.65	0.14	2.77	0.00

F.1.2 Test 4: Multiple mediums

Medium utilization

Table F.9: Results of yawmd mediums by QoS queue, with $S = 10$ (10 stations per medium). Values of mean, μ, standard deviation, σ, and $medium\ utilization = VO_\mu + BE_\mu$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
1	6.93	0.287	93.07	0.371	100.00	6.93	0.280	92.73	3.179	99.66
2	6.93	0.273	93.07	0.295	100.00	6.93	0.270	93.07	0.285	100.00
3	6.94	0.250	93.06	0.279	100.00	6.93	0.276	92.94	1.249	99.87
4	6.93	0.260	93.07	0.281	100.00	6.93	0.261	93.07	0.281	100.00
5	6.89	0.289	93.11	0.290	100.00	6.87	0.289	93.13	0.295	100.00
6	6.63	0.477	92.91	0.833	99.54	6.69	0.354	93.14	0.594	99.83
7	6.04	0.680	88.71	2.230	94.75	6.07	0.755	87.69	2.596	93.76
8	5.66	0.647	80.18	3.695	85.84	5.72	0.698	79.85	3.578	85.57
9	5.02	0.634	74.16	3.867	79.17	5.16	0.713	74.01	4.223	79.16
10	4.56	0.725	70.24	4.605	74.80	4.63	0.637	70.09	3.936	74.72

Table F.10: Results of yawmd mediums by QoS queue, with $S = 20$ (20 stations per medium). Values of mean, μ, standard deviation, σ, and $\textit{medium utilization} = VO_\mu + BE_\mu$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
1	13.86	0.519	86.14	0.556	100.00	13.88	0.544	86.11	0.571	99.99
2	13.87	0.481	86.13	0.494	100.00	13.87	0.510	86.14	0.503	100.00
3	13.87	0.489	86.13	0.508	100.00	13.87	0.475	86.13	0.463	100.00
4	13.84	0.464	86.16	0.433	100.00	13.83	0.498	86.17	0.500	100.00
5	11.65	0.492	86.51	1.120	98.16	11.67	0.521	86.04	1.214	97.71
6	8.81	0.820	87.07	1.750	95.88	8.59	0.716	87.20	1.949	95.79
7	6.84	0.912	82.68	2.761	89.52	6.78	0.824	82.19	2.947	88.97
8	5.97	0.747	77.87	3.942	83.83	5.69	0.780	77.70	3.735	83.39
9	4.98	0.574	72.60	3.578	77.58	4.94	0.604	72.97	3.104	77.91
10	4.36	0.620	68.13	3.823	72.49	4.36	0.613	68.02	4.002	72.38

Table F.11: Results of yawmd pthreads by QoS queue, with $S = 10$ (10 stations per medium). Values of mean, μ, standard deviation, σ, and $\textit{medium utilization} = VO_\mu + BE_\mu$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
1	6.93	0.285	92.74	3.161	99.66	6.93	0.271	93.07	0.363	100.00
2	6.93	0.257	93.07	0.271	100.00	6.93	0.266	93.07	0.288	100.00
3	6.93	0.251	93.07	0.272	100.00	6.93	0.258	93.07	0.285	100.00
4	6.93	0.264	93.07	0.269	100.00	6.93	0.251	93.07	0.265	100.00
5	6.93	0.265	92.98	0.861	99.91	6.93	0.268	93.07	0.272	100.00
6	6.93	0.246	93.07	0.263	100.00	6.94	0.265	93.06	0.273	100.00
7	6.92	0.251	93.08	0.243	100.00	6.93	0.258	93.07	0.269	100.00
8	6.79	0.267	93.21	0.277	100.00	6.83	0.284	93.17	0.278	100.00
9	6.48	0.226	93.50	0.241	99.99	6.54	0.261	93.39	0.572	99.93
10	5.77	0.185	93.81	0.297	99.58	5.82	0.172	93.78	0.282	99.60

Table F.12: Results of yawmd pthreads by QoS queue, with $S = 20$ (20 stations per medium). Values of mean, μ, standard deviation, σ, and $medium\ utilization = VO_\mu + BE_\mu$, presented. The QoS queues are voice, VO, and best effort, BE.

N	Run 1					Run 2				
	VO QoS		BE QoS		Medium utilization (%)	VO QoS		BE QoS		Medium utilization (%)
	μ (%)	σ (%)	μ (%)	σ (%)		μ (%)	σ (%)	μ (%)	σ (%)	
1	13.86	0.555	86.14	0.592	100.00	13.88	0.556	86.12	0.593	100.00
2	13.87	0.494	86.13	0.504	100.00	13.88	0.448	86.12	0.478	100.00
3	13.86	0.499	86.14	0.507	100.00	13.87	0.495	86.13	0.483	100.00
4	13.87	0.477	86.13	0.482	100.00	13.86	0.503	86.14	0.501	100.00
5	11.95	0.449	88.05	0.451	100.00	11.90	0.428	88.10	0.472	100.00
6	9.92	0.368	89.94	0.395	99.86	9.88	0.370	89.93	0.413	99.81
7	8.47	0.331	90.67	0.564	99.14	8.45	0.330	90.71	0.579	99.16
8	7.38	0.268	90.12	0.777	97.50	7.38	0.263	90.33	0.842	97.71
9	6.56	0.265	88.83	0.944	95.39	6.56	0.193	88.67	0.880	95.22
10	5.91	0.153	86.27	1.305	92.18	5.91	0.160	85.67	1.376	91.59

CPU and memory usage

Table F.13: Results of yawmd mediums CPU and memory usage, with $S = 10$ (10 stations per medium). Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
1	0.76	0.10	1.39	0.00	0.67	0.17	1.39	0.00
2	1.46	0.13	1.27	0.00	1.42	0.15	1.39	0.00
3	2.19	0.12	1.27	0.00	2.19	0.15	1.32	0.00
4	2.80	0.24	1.32	0.00	2.85	0.24	1.26	0.00
5	3.31	0.37	1.39	0.00	3.43	0.26	1.27	0.00
6	3.22	0.41	2.71	0.00	3.01	0.39	2.71	0.15
7	3.19	0.30	2.76	0.00	3.25	0.31	2.71	0.00
8	3.39	0.25	2.68	0.00	3.45	0.23	2.75	0.00
9	3.58	0.16	2.65	0.00	3.59	0.19	2.70	0.00
10	3.68	0.14	2.78	0.00	3.66	0.15	2.76	0.00

Table F.14: Results of yawmd mediums CPU and memory usage, with $S = 20$ (20 stations per medium). Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
1	0.77	0.09	1.32	0.00	0.78	0.08	1.32	0.00
2	1.53	0.10	1.39	0.00	1.47	0.15	1.27	0.00
3	2.13	0.23	1.39	0.00	2.23	0.10	1.27	0.00
4	2.79	0.23	1.39	0.00	2.85	0.17	2.68	0.00
5	3.08	0.32	2.77	0.00	3.13	0.29	2.70	0.00
6	3.29	0.24	2.67	0.00	3.26	0.29	2.64	0.00
7	3.38	0.26	2.75	0.00	3.43	0.23	2.68	0.00
8	3.50	0.21	2.75	0.00	3.54	0.21	2.71	0.00
9	3.65	0.13	2.71	0.00	3.66	0.14	2.77	0.00
10	3.73	0.12	2.76	0.00	3.75	0.15	2.70	0.00

Table F.15: Results of yawmd pthreads CPU and memory usage, with $S = 10$ (10 stations per medium). Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
1	1.06	0.10	1.26	0.00	1.10	0.11	1.39	0.00
2	2.17	0.17	1.26	0.00	2.16	0.13	1.39	0.00
3	3.28	0.18	1.26	0.00	3.26	0.18	1.39	0.00
4	4.48	0.19	1.39	0.00	4.38	0.19	1.32	0.00
5	5.55	0.23	2.80	0.00	5.64	0.20	2.76	0.00
6	6.86	0.17	2.73	0.01	6.84	0.19	2.77	0.00
7	8.12	0.15	2.77	0.00	8.17	0.20	2.79	0.00
8	9.56	0.18	2.74	0.00	9.48	0.17	2.75	0.01
9	11.08	0.21	2.78	0.00	11.07	0.24	2.75	0.00
10	12.10	0.33	2.64	0.00	12.19	0.27	2.82	0.00

Table F.16: Results of yawmd pthreads CPU and memory usage, with $S = 20$ (20 stations per medium). Values of mean, μ, and standard deviation, σ, presented.

N	Run 1				Run 2			
	CPU usage		Memory usage		CPU usage		Memory usage	
	μ (%)	σ (%)	μ (MiB)	σ (MiB)	μ (%)	σ (%)	μ (MiB)	σ (MiB)
1	1.04	0.12	1.39	0.00	1.15	0.09	1.39	0.00
2	2.10	0.18	1.39	0.00	2.20	0.19	1.26	0.00
3	3.15	0.21	1.39	0.00	3.13	0.14	1.26	0.00
4	4.28	0.14	2.75	0.00	4.46	0.24	2.71	0.00
5	5.51	0.19	2.80	0.00	5.51	0.19	2.76	0.01
6	6.79	0.14	2.77	0.00	6.85	0.16	2.72	0.00
7	8.07	0.18	2.70	0.00	8.09	0.19	2.72	0.00
8	9.22	0.19	2.73	0.00	9.27	0.21	2.79	0.00
9	10.32	0.18	2.63	0.00	10.37	0.19	2.63	0.00
10	10.82	0.16	2.59	0.00	10.82	0.25	2.79	0.00

F.2 Plots of medium usage by QoS queue

F.2.1 Test 3: One medium

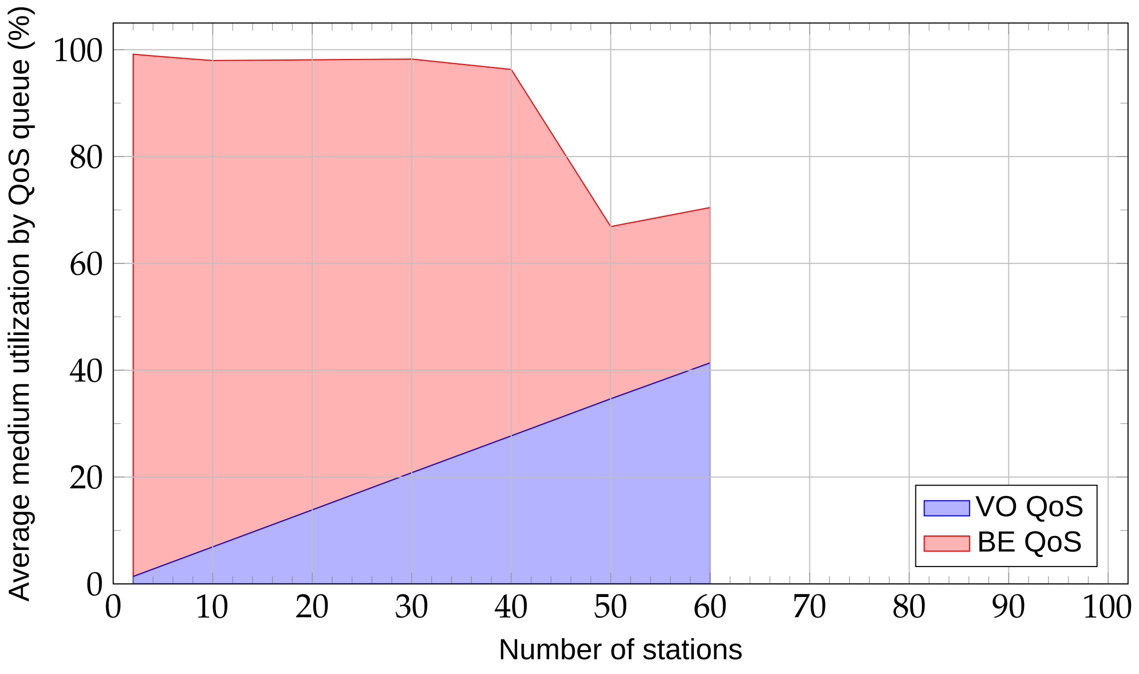

Figure F.1: Average of the *medium utilization* by QoS queue, at each second, during each instance, with wmediumd.

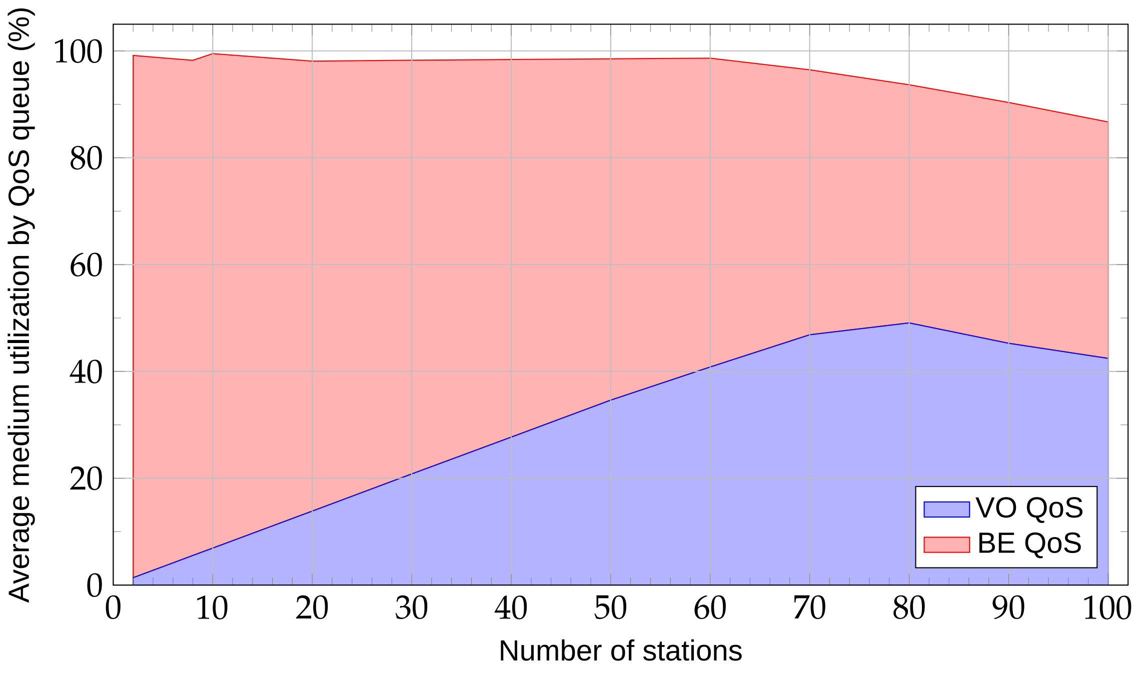

Figure F.2: Average of the *medium utilization* by QoS queue, at each second, during each instance, with yawmd v1.

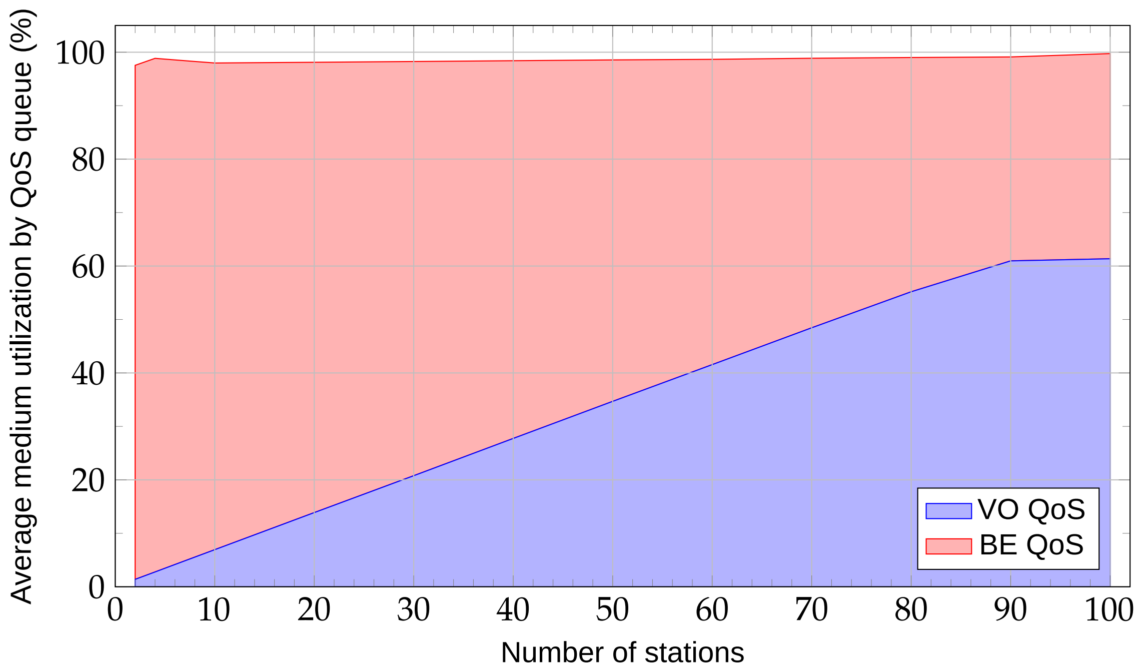

Figure F.3: Average of the *medium utilization* by QoS queue, at each second, during each instance, with yawmd mediums.

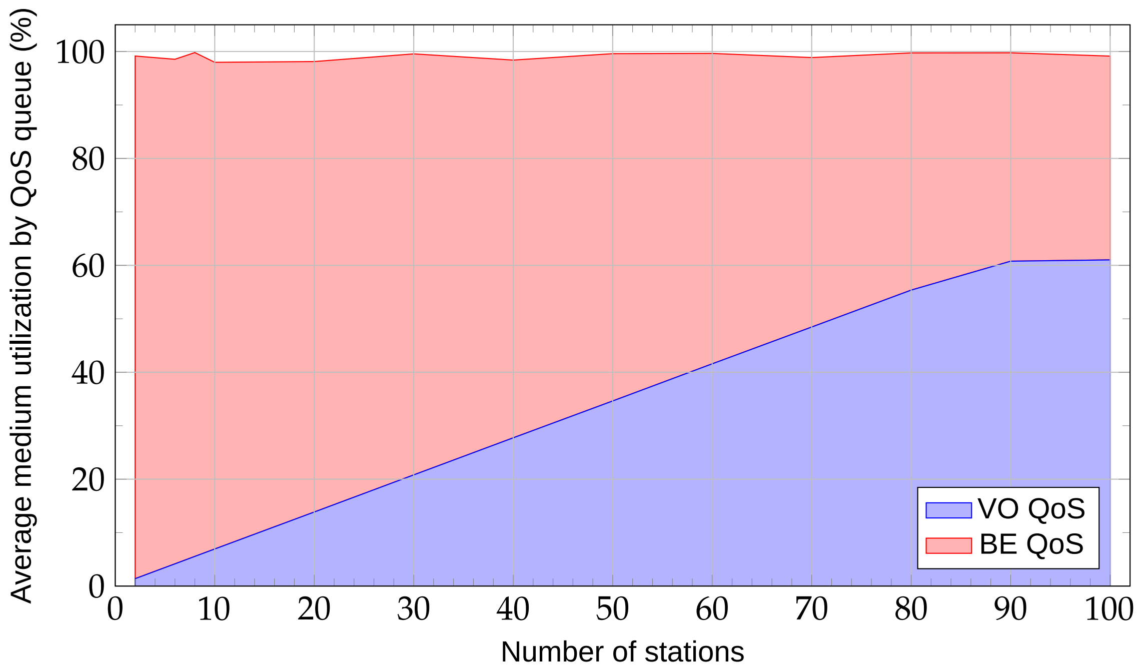

Figure F.4: Average of the *medium utilization* by QoS queue, at each second, during each instance, with yawmd pthreads.

F.2.2 Test 4: Multiple mediums

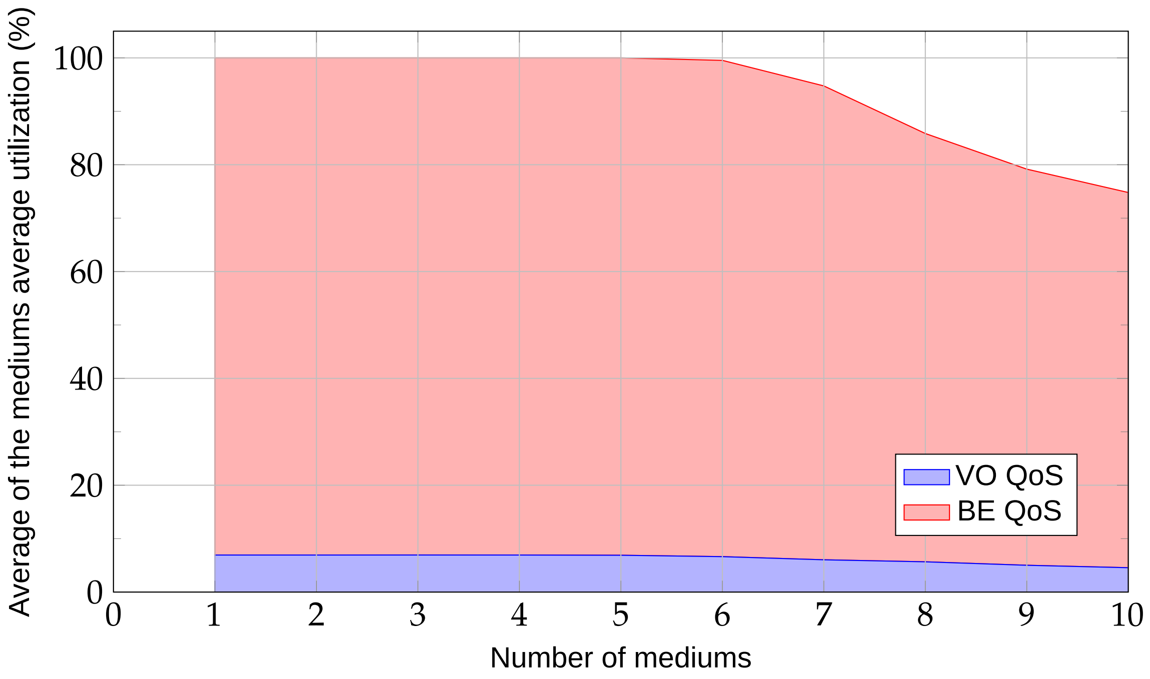

Figure F.5: Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance, with yawmd mediums, with $S = 10$ (10 stations per medium).

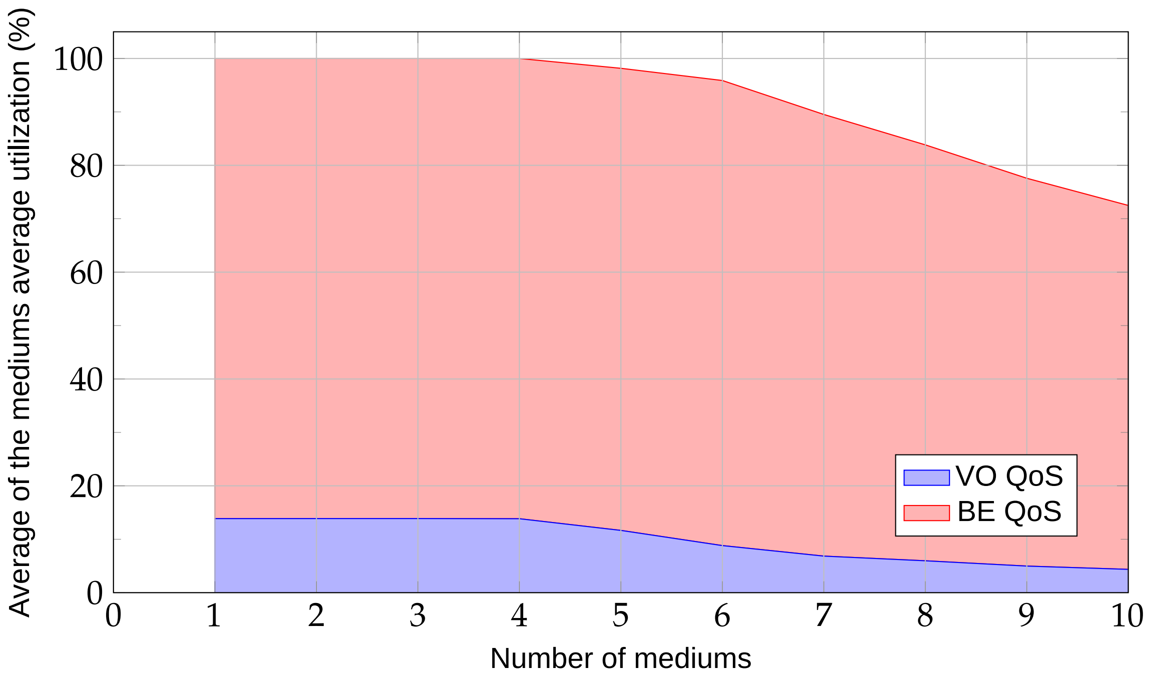

Figure F.6: Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance, with yawmd mediums, with $S = 20$ (20 stations per medium).

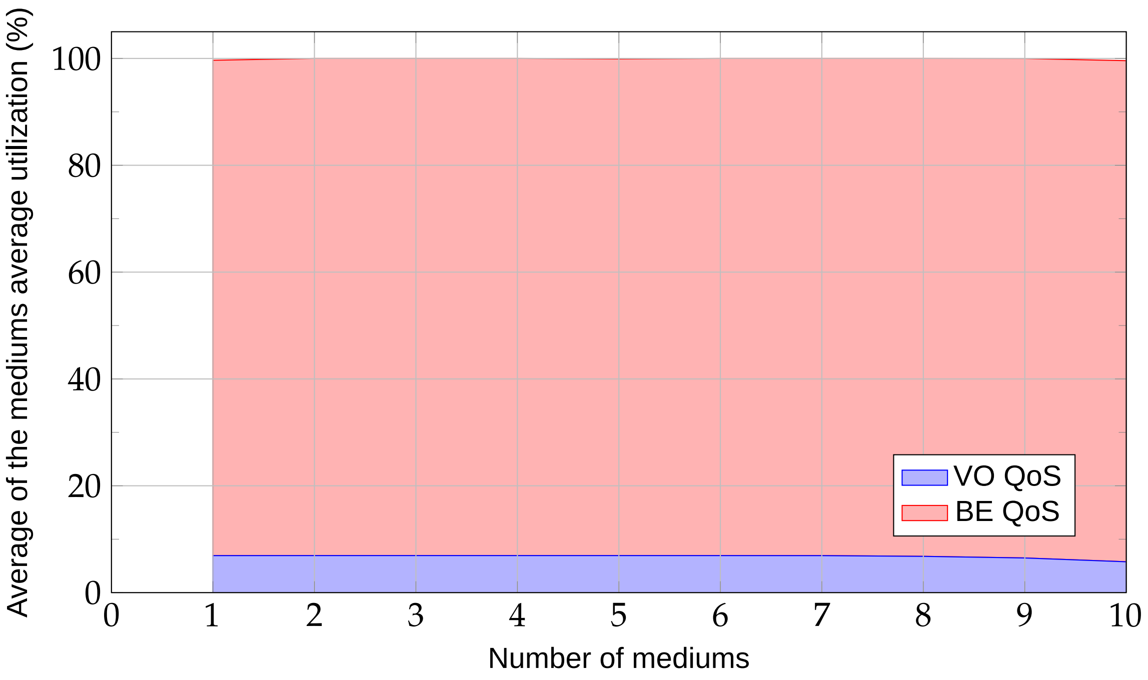

Figure F.7: Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance, with yawmd pthreads, with $S = 10$ (10 stations per medium).

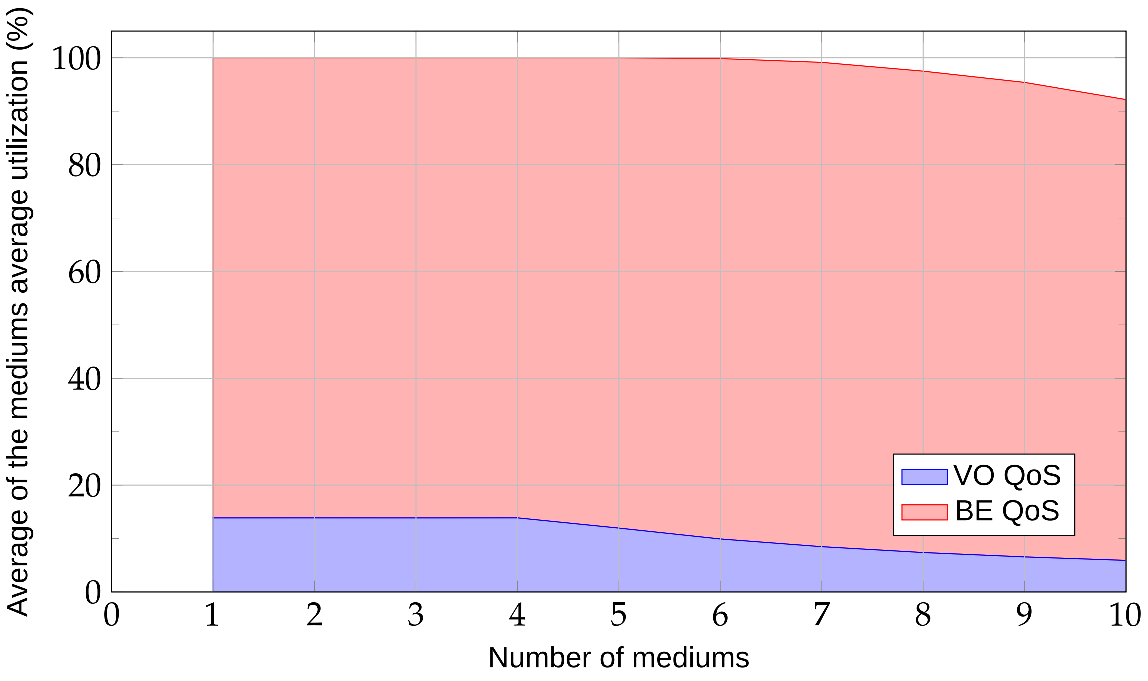

Figure F.8: Average of the mediums average *medium utilization* by QoS queue, at each second, during each instance, with yawmd pthreads, with $S = 20$ (20 stations per medium).

F.3 Plots of medium usage by QoS queue with overlapping

F.3.1 Test 3: One medium

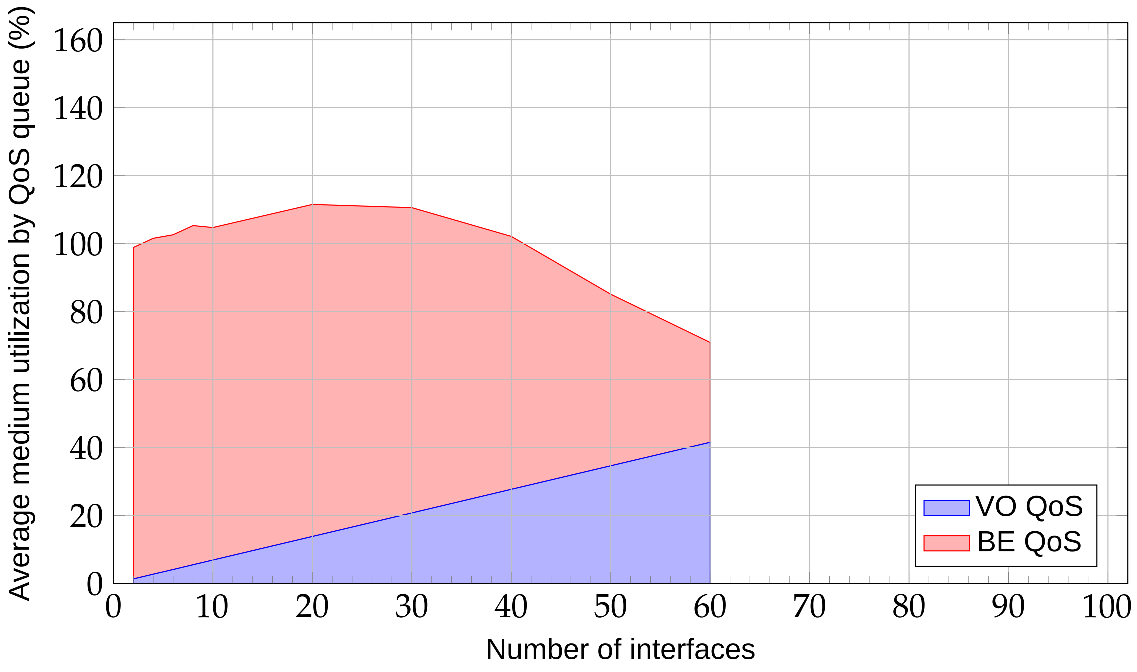

Figure F.9: Average of the *medium utilization* by QoS queue, at each second, during each instance, using wmediumd without the patch to prevent overlapping transmissions.

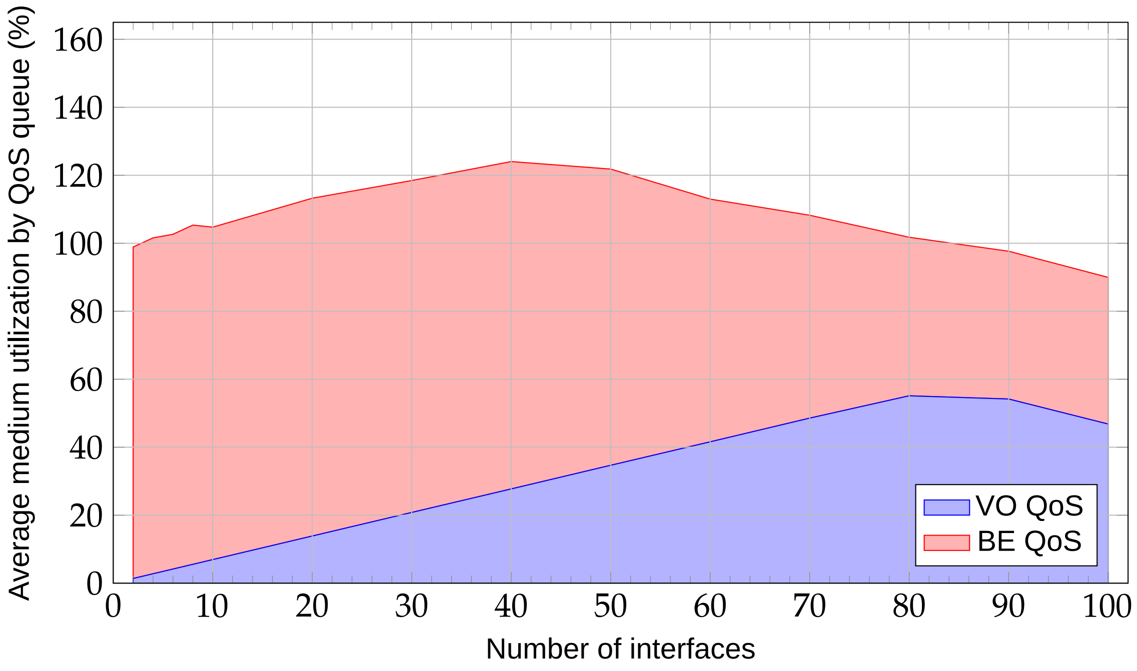

Figure F.10: Average of the *medium utilization* by QoS queue, at each second, during each instance, using yawmd v1 without the patch to prevent overlapping transmissions.

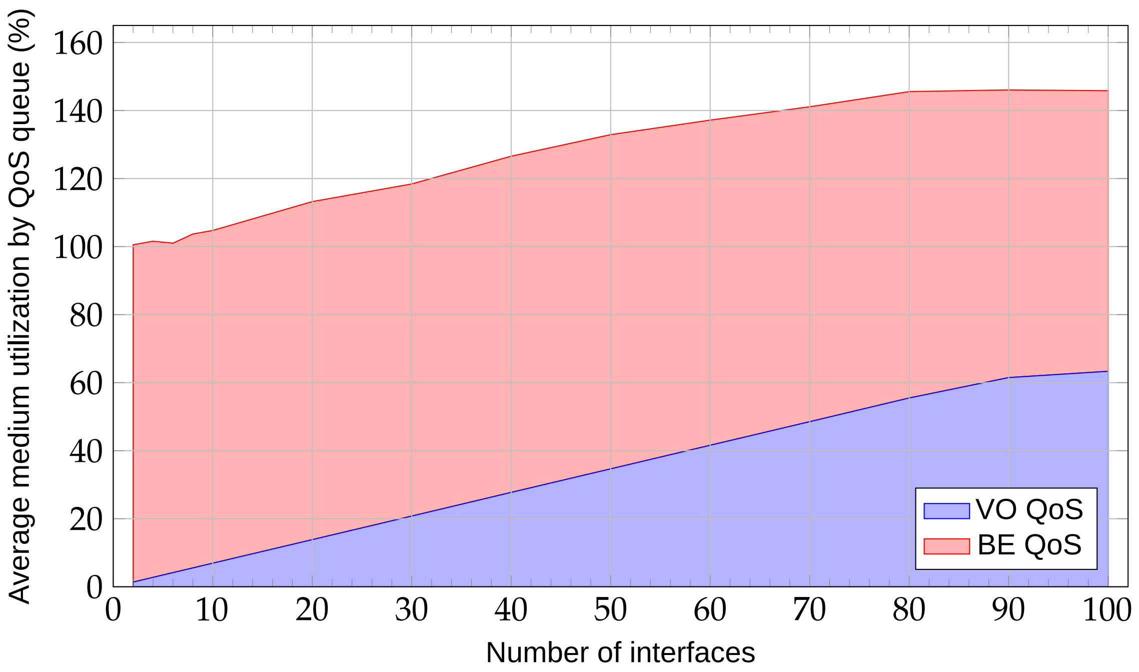

Figure F.11: Average of the *medium utilization* by QoS queue, at each second, during each instance, using yawmd mediums without the patch to prevent overlapping transmissions.

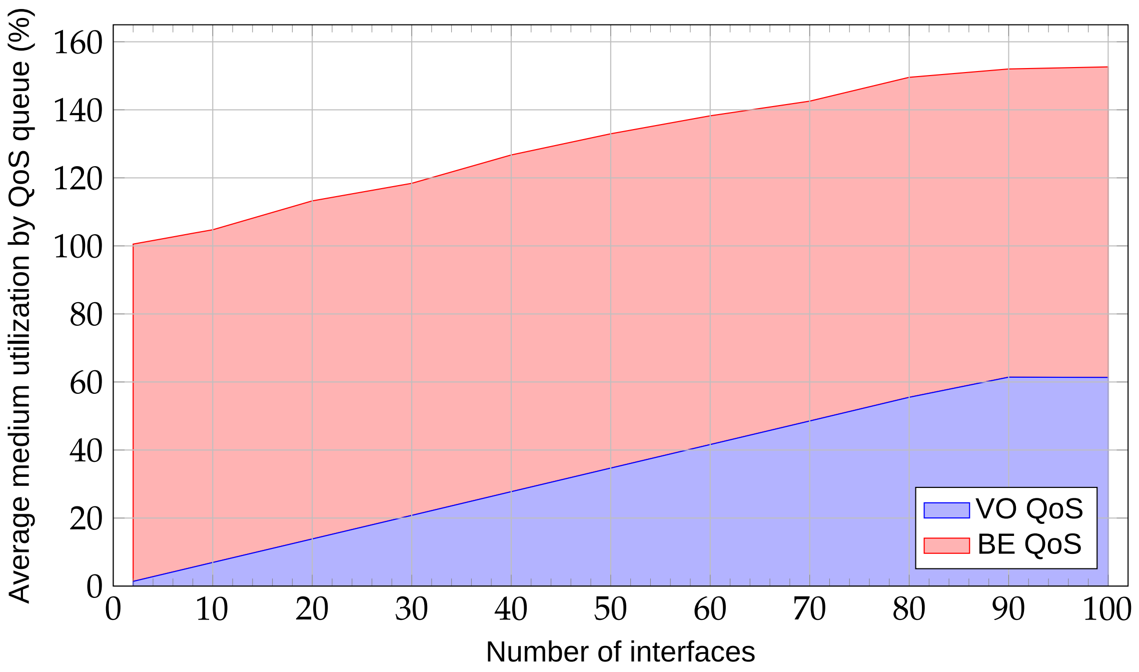

Figure F.12: Average of the *medium utilization* by QoS queue, at each second, during each instance, using yawmd pthreads without the patch to prevent overlapping transmissions.