DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS

STOCKHOLM, *SWEDEN 2020*

Trace-Driven WiFi Emulation

Accurate Record-and-Replay for WiFi

ABHISHEK KUMAR MISHRA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Accurate record-and-replay for WiFi

Trace-Driven WiFi Emulation

Author:
Abhishek Kumar MISHRA

Email:
akmishra@kth.se

Degree Project Report
KTH supervisor: Viktoria Fodor
Host company supervisor: Renata Teixeira
Examiner: Gunnar Karlsson
School of Electrical Engineering and Computer Science
Host Organisation: Inria, Paris

July 13, 2020

Abstract

Researchers and application designers need repeatable methods to evaluate applications/systems over WiFi. It is hard to reproduce evaluations over WiFi because of rapidly changing wireless quality over time. In this degree project, we present NemFi, a trace-driven emulator for accurately recording the WiFi traffic and later using it to emulate WiFi links in a repeatable fashion. First, we present the advantages of trace-driven emulation over simulation and experimentation. We capture the fluctuating WiFi link conditions in terms of capacity and losses over time and replay captured behavior for any application running in the emulator. Current record-and-replay techniques for web traffic and cellular networks do not work for WiFi because of their inability to distinguish between WiFi losses and losses due to self-induced congestion. They are also lacking other WiFi specific features. In the absence of a trace-driven emulator for WiFi, NemFi is also equipped to avoid self-induced packet losses. It is thus capable of isolating WiFi related losses which are then replayed by the NemFi's replay. NemFi's record also addresses the frame aggregation and the effect it has on the actual data transmission capability over the wireless link. NemFi can record frame aggregation, at all instants of the record phase and later accurately replays the aggregation.

Experimental results demonstrate that NemFi is not only accurate in recording the variable-rate WiFi link but also in capturing cross-traffic. NemFi also replays the recorded conditions with considerable accuracy.

Keywords

WiFi, trace-driven emulation, record & replay tools, networked system evaluation

Sammanfattning

Forskare och applikationsdesigners behöver repeterbara metoder för att utvärdera applikationer och system via WiFi. Det är svårt att reproducera utvärderingar genom WiFi eftersom den trådlösa kvalitén snabbt förändras över tid. I denna rapport presenterar vi NemFi, en spårstyrd emulator för att noggrant registrera WiFi-trafiken och senare använda den för att emulera WiFi-länkar påett repeterbart sätt. Först presenterar vi fördelarna med spårstyrd emulering jämfört med simulering och experiment. Vi fångar de varierande WiFi förhållanden med avseende påkapacitet och förluster över tid och spelar upp fångat beteende för alla applikationer som körs i emulatorn. Nuvarande inspelning och uppspelningstekniker för webbtrafik och mobilnät fungerar inte för WiFi pågrund av deras oförmåga att skilja mellan WiFi-förluster och förluster pågrund av självinducerad överbelastning. De saknar ocksåandra WiFi-specifika funktioner. I avsaknad av en spårdriven emulator för WiFi är NemFi ocksåutrustade för att undvika självinducerade paketförluster. Den kan alltsåisolera WiFi-relaterade förluster som sedan spelas upp igen av NemFi: s uppspelning. NemFi adresserar ocksåramaggregering och det är effekten påfaktiska dataöverföringsförmåga via den trådlösa länken. NemFi kan spela in ramsamling, vid alla ögonblick i inspelningsfasen och ersätter senare noggrant aggregeringen.

Experimentella resultat visar att NemFi inte bara är användbart när det gäller att registrera WiFi-länken med variabel hastighet, utan ocksåför att fånga tvärgående trafik. NemFi ersätter ocksåde inspelade förhållandena med betydande noggrannhet.

Nyckelord

WiFi, spårstyrd emulering, inspelnings- och återuppspelningsverktyg, nätverkssystemutvärdering

List of Figures

List of Tables

List of Notations

- LAN - Local Area Network
- WAN - Wide Area Network
- TCP - Transmission Control Protocol
- UDP - User Datagram Protocol
- LTE - Long-Term Evolution
- PHY - Physical Layer
- WLAN - Wireless Local Area Network
- MAC - Medium Access Control
- MSDU - MAC Service Data Unit
- MPDU - MAC Protocol Data Unit
- HTTP - Hypertext Transfer Protocol
- A-MSDU - Aggregate MSDU
- A-MPDU - Aggregate MPDU
- GUI - Graphical User Interface
- MP-TCP - Multipath Transmission Control Protocol
- MCS - Modulation and Coding Scheme
- SISO - Single Input Single Output
- MIMO - Multiple Input Multiple Output
- RTT - Round-Trip Time
- ACK - Acknowledgement packet
- NIC - Network Interface Card
- BA - Block Acknowledgement frame

- EDCA - Enhanced Distributed Channel Access
- FDR - Frame Delivery Ratio
- P - Physical Rate
- LC - Estimated Link Capacity

Contents

Chapter 1

1 Introduction

Mobile networks are becoming increasingly more popular due to the widespread use of mobile devices (e.g. smartphones, laptops, tablets, smartwatches, etc.) [15]. The quality of wireless connectivity varies drastically from place to place. There are several factors that affect the signal quality or create interference like poor network configuration, old equipment, fluctuating demands of users, interference from non-WiFi devices such as microwave, router position, congestion, and coverage. These factors cause the wireless quality to vary significantly over time. As many of today's applications and services will be running over wireless networks, it is very crucial to evaluate the performance of these applications in different wireless networks. The variability of WiFi makes it hard to predict how an application/service will work with just a few experiments. Testing in one/few settings tells little of how a service will behave when deployed at a large scale over a long period.

There are different options for evaluating networked applications and services: simulation, testbed experiments, and emulation (Chapter 2). Simulation is the easiest way to experiment with different wireless network conditions. Simulators are used to mimic the behavior of a certain network in a software-based environment. The advantages of the simulation are repeatability, control, configurability, and experiments of large scale networks. The main limitation of simulation tools, however, is that they require the user to tune different parameters e.g., level of interference, congestion, loss rate, etc. which may not reflect real wireless network conditions. Even with good parameter settings, a simulator cannot capture the complex inter-dependencies of real systems.

At the other end of the spectrum, there is testbed experimentation, where developers evaluate their applications over deployed wireless links either over testbeds or by relying on volunteer testers. The results of such experiments capture the impact of real wireless network conditions. The major disadvantage of experimentation is that it offers no repeat-ability and is difficult to scale. The variability of wireless networks makes it hard to reproduce the results. The results of experimentation are, therefore, hard to interpret and

one cannot distinguish the issues with application versus wireless issues.

Finally, trace-driven emulation involves recording traces in deployed wireless networks and later emulating the recorded behavior. The clear benefits of trace-driven emulations are its ability to capture real network conditions and the repeat-ability of the experiments. One can run the same network conditions several times, which eases application or system debugging, and enables comparative analysis of different applications or protocols over the same network conditions. While there exist trace-driven emulators for cellular and HTTP traffic, to the best of our knowledge, there exist no such tools for WiFi.

Most of the existing solutions follow record-and-replay method where we capture packet-delivery opportunities in the trace file and then pass those opportunities to the replay, which allocates the recorded delivery opportunities when the tested application has packets to send. *Delivery opportunities* are instances where an MTU sized packet could be released by the replay framework.

The goal of this degree project is to develop emulation methods and systems for WiFi. We first identify many limitations with the existing method designed for cellular links, which together prevent us from applying it directly for WiFi. Then, we design and implement a new set of methods and algorithms to adapt the record and replay method for WiFi.

The rest of this section highlights the challenges of trace-driven emulation over WiFi and then presents the contributions of this degree project.

1.1 Trace-driven emulation over WiFi

The state-of-the-art trace-driven emulator works only for cellular networks. This is because, in cellular networks, there are per device queues. If the bottleneck is the cellular network, then the congestion at the base-station is mostly self-induced and the effect of cross-traffic is muted. Moreover, in cellular networks, the up-link and down-link communications of users take place on different time slices and the two do not interfere with each other. Whereas the medium is shared in WiFi and hence, delivery opportunities

are shared between the upstream and the downstream flows as well as with competing traffic.

LTE base-stations hold much larger queues than WiFi and allow for more re-transmissions. In WiFi, the queues are smaller and packet loss is more common. This introduces the question of how to saturate the wireless link without inducing too many packet losses which may have a negative impact on the measurements. In WiFi, even in the case of static users, there could be a lot of fluctuation in the supported PHY rates due to variation in channel conditions.

1.2 Major contributions

We design, evaluate, and implement, NemFi, a trace-driven WiFi emulator based on the record and replay method. We highlight major contributions below:

- Identifying the limitations of the existing method for WiFi and developing a new rate control algorithm to record traces with delivery opportunities over WiFi. This algorithm accurately captures variations of WiFi physical rate as well as WiFi losses. NemFi's record module also logs information of frame aggregation.
- Designing the sharing of up-link and down-link queues for the distribution of packet delivery opportunities during the NemFi's replay. Also, introducing the frame aggregation in the replay phase and playing back WiFi losses.

1.3 Thesis outline

The thesis is organized as follows in upcoming sections:

- Chapter 2 presents existing work done in different methods of evaluating network conditions, protocols, and mobile applications.
- Chapter 3 discusses overall system design and challenges in using existing methods.

- Chapter 4 elaborates on the design and implementation of the NemFi's record.
- Chapter 5 evaluates the record phase compared to existing tools and justifies the answers to the research questions posed earlier.
- Chapter 6 comes up with the design and implementation of the NemFi's replay.
- Chapter 7 validates the whole record and replay pipeline, after the success-full implementation of the replay phase introduced in the previous chapter.
- Chapter 8 concludes the work with remarks and suggestions for future work.

Chapter 2

2 Background and Related Work

This chapter aims to introduce basic concepts for the understanding of the solutions discussed in this degree project. Also, we dive into the developments that have already been made regarding the problem addressed by this work.

2.1 Basics of WiFi

WiFi is a widely adopted wireless communication protocol, used in most home and office networks to allow laptops, printers, and smartphones to talk to each other and access the Internet without connecting wires. [24]

WiFi standards are part of the IEEE 802 set of Local Area Network protocols, and specify the set of medium access control and physical layer protocols for implementing wireless local area network communication in various frequencies, including but not limited to 2.4 GHz, 5 GHz, 6 GHz, and 60 GHz frequency bands. The 802.11 family consists of a series of half-duplex over-the-air modulation techniques that use the same basic protocol. WiFi employs carrier-sense multiple access with collision avoidance whereby equipment listens to a channel for other users (including non 802.11 users) before transmitting each packet. [24]

2.1.1 Frame aggregation

Nowadays, WiFi standards focus more and more on improving metrics like per-user throughput and latency. *Frame aggregation* is a key feature of the latest WiFi standards that increases the throughput by sending two or more data frames in a single transmission. In general, every frame transmitted by an 802.11 device has a significant amount of overhead, including radio level headers, medium access control frame fields, inter-frame spacing, and acknowledgment of transmitted frames. [26]

WiFi standard defines two major types of frame aggregation: MSDU aggregation and MPDU aggregation. The major difference between an MSDU

and an MPDU is that the former corresponds to the information that is imported to or exported from the upper part of the MAC sublayer from or to the higher layers, respectively, whereas the latter relates to the information that is exchanged from or to the PHY by the lower part of the MAC. [27] Multiple MPDUs are acknowledged with a single block ACK in response to a block acknowledgment request (BAR). This block ACK facilitates the exchange sequences to be aggregated.

A-MSDU: The principle of the A-MSDU (or MSDU aggregation) is to allow multiple MSDUs to be sent to the same receiver, put together in a single MPDU. When there are many small MSDUs, such as TCP acknowledgments, the efficiency of the MAC layer is improved. Upper MAC receives and buffers multiple packets (MSDUs) to form the A-MPDU. The A-MSDU is ready to be transmitted either when the size of the waiting packets reaches the maximal A-MSDU threshold or the maximal delay of the oldest packet reaches a pre-assigned value. A-MSDU is not beneficial to use in the case of poor quality channel conditions. As all MSDUs are joined into a single MPDU with a single sequence number, for any corrupted sub-frames, we must re-transmit the entire A-MSDU. [27]

A-MPDU: The concept of A-MPDU aggregation is to merge multiple MPDU sub-frames within a single leading PHY header. A-MPDU is different from A-MSDU aggregation as it functions after the MAC header encapsulation process. All the MPDUs within an A-MPDU must be addressed to the same receiver address. Moreover, there is no waiting time to form an A-MPDU. Therefore, the number of MPDUs to be aggregated completely depends on the number of packets already in the transmission queue. The maximum length of an A-MPDU is 65,535 bytes. [27]

As all of the management information needs to be sent only once per frame, the ratio of payload data to the total volume of data is higher, allowing higher throughput to be achieved.[26]

Frame aggregation, therefore, improves the link utilization by a significant margin. *Link utilization* refers to the average traffic that we can send on a link in the network compared to the PHY rate.

There has been a lot of work on measuring WiFi and estimating the

capacity of it. Hora [5] formulates methods for estimating the link capacity of commodity access points.

2.2 Existing simulation frameworks

Researchers rely on network simulators both for academia and industry to evaluate new protocols. NS-2 [7] and NS-3 [8] are open-source network simulators to reproduce network systems. NS-2 provides support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks [25]. While NS-3 simulation core allows research on both IP and non-IP based networks. With NS-3 users can easily perform simulation for wireless simulations that involve models for WiFi, WiMAX, or LTE for layers 1 and 2 and a variety of static or dynamic routing protocols such as OLSR and AODV for IP-based applications.

OMNET++ [9] is a simulation over wired and wireless communication networks, on-chip networks, and queuing networks. OPNET [10] is an open-source network simulation tool that offers various topologies and configurations. It could be used for cross-layer design, defining new technologies like ultra-wideband, software radio, and more localized algorithms. NETSIM [11] is a commercial network simulator that provides simulation for layer 1 and layer 2 capabilities of WLAN.

QualNet [16] is a commercial network simulator for scalable network technologies. It offers a GUI for users to complete the simulation without the need for coding. TraceReplay is an application layer simulator built in NS-3 for network traces [11].

Even though there are a lot of simulators available, simulation models fail to capture the diversity of deployed systems. Simulation models also require input parameters that are hard to set.

2.3 Existing testbeds

Testbeds refer to platforms that provide support for testing and development of new technologies, products, or scientific theories.

Wireless Hybrid Network (WHYNET) [23] is a hybrid testbed as it enables the use of simulation, emulation, and real hardware. It allows us to integrate all three components on both individual and combined levels. There is limited remote access to WHYNET testbed infrastructure. MONROE [19] is an open access hardware-based measurement platform for doing experiments on mobile broadband. It lacks the support for WiFi.

ORBIT [22] is an example radio grid testbed that provides the functionality of reproducing wireless experiments with a large number of nodes. It helps users to introduce fading and controlled interference. The advantages of testbeds include running experiments over real wireless links.

However, the testbeds come with major drawbacks like no repeat-ability, small-scale in nature and are location dependent.

2.4 Existing emulation solutions

In network emulation, we bring a device to a test network in a lab environment that alters the packet flow in such a way as to mimic the behavior of a network such as a LAN or WAN. This device may be either a general-purpose computer running software to perform the network emulation or a dedicated emulation device. This technique is used for testing the performance of real applications over a virtual network.

Emulation is different from network simulation where purely mathematical models of traffic, network models, channels, and protocols are applied. The aim is to assess performance, predict the impact of change, or optimize technological decision-making. [25] Network emulators take care of many network attributes such as latency, available bandwidth, packet loss, and jitter. [25]

Trace-driven emulators follow the record-and-replay paradigm, which refers to the act of recording the delivery opportunities, generally in the form of a trace file, and using the trace later to emulate the captured network conditions. Figure 1 illustrates the paradigm.

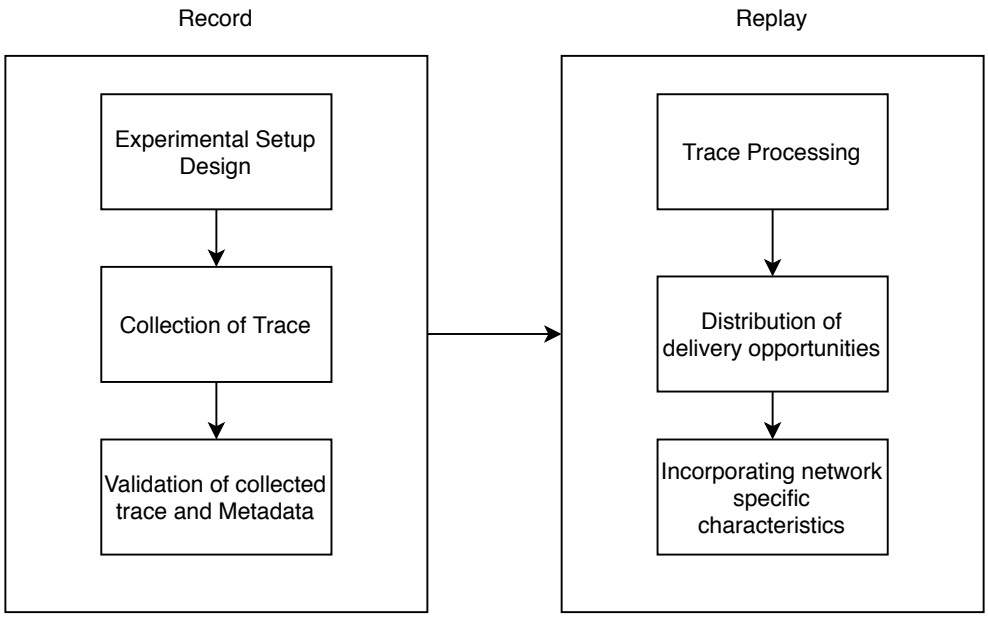


Figure 1: Record-and-Replay paradigm

The record starts with designing an experimental setup that is capable of capturing the network effectively and accurately. It follows with actual trace collection, where we generally send packets to record the state of the network. Collected traces could then be validated for the correctness in recording network parameters.

Replay processes the trace and metadata collected from the record and extracts the packet delivery opportunities. It needs to be distributed for both up-link and down-link flows. Finally, replay also incorporates in itself, network-specific characteristics like variable link rates and losses, frame aggregation, etc.

2.4.1 Trace-driven emulators for cellular

Several network emulators have been previously used to emulate network conditions for traffic over cellular. Mobile network tracing [20] observes traffic passively to generate traces and then uses the Packet Modulator (PaM) to corrupt, delay, or drop captured packets. However mobile network tracing does not address the question of different machines sharing the same band-

width. Trace-modulation [21] listens to a path passively multiple times to generate traces of real network behavior.

Sprout [1] introduces *Saturator*, a tool to collect traces over cellular links. Collected traces could later be used to emulate traffic using their simulator, CellSim. Figure 2 presents the architecture of the Saturator.

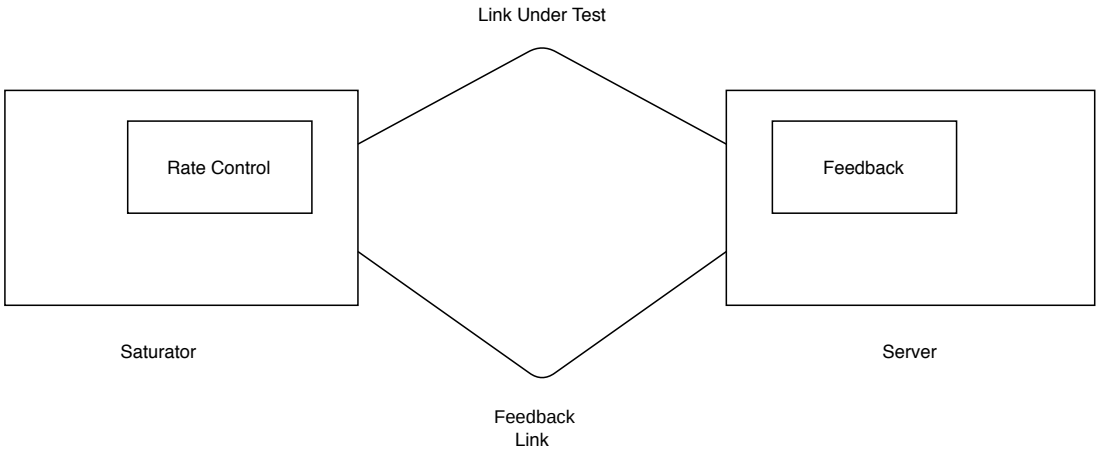


Figure 2: Architecture of Saturator [1]

We see in the Figure 2 that Saturator is connected to the server by two links. The first one is the link under test which is the wireless link. It could be cellular or WiFi. Saturator sends UDP packets with a very basic rate control to the server. It increases the number of packets in flight by one, after every successful reception of the ACK packet from the server. Saturator has a *Maximum Window Size* beyond which it stops increasing its sending rate. The saturator assumes that the maximum window size is reached when an RTT of 3 seconds is reached, or when the window size hits 1500 MTUs. There is a feedback link that reliably sends back the ACK packet to the client machine running the Saturator.

Saturator does not prove to be accurate for recording WiFi due to the major problem of induced losses. Section 3 and 4 describes this in detail.

2.4.2 Trace-driven emulators for web traffic

Mahimahi [2] is a framework for recording and replaying HTTP traffic under different network conditions. It includes three network emulation tools: DelayShell, LinkShell, and LossShell. Mahimahi uses DelayShell for emulating a fixed propagation delay and LinkShell for emulating fixed and variable capacity links. MpShell [14] extends the Mahimahi framework to record WiFi and LTE traces simultaneously. This work was mainly developed to evaluate the performance of MP-TCP in different network conditions and for various types of applications.

Figure 3 describes the architecture of LinkShell. It contains separate up-link and down-link queues for storing the packets from the up-link and the down-link application traffic. Queues release packets based on the trace given as an input. LinkShell uses separate private network traces to isolate traffic between multiple instances of LinkShell spawned in the client machine.

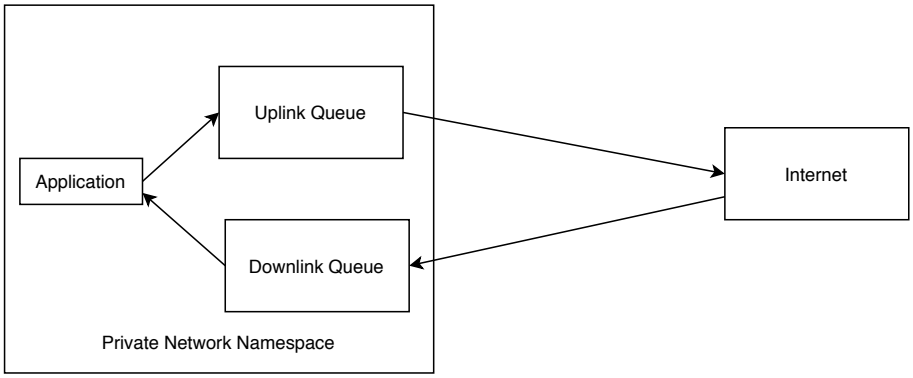


Figure 3: Architecture of LinkShell [2]

Both MpShell and LinkShell have major drawbacks in design as they do not share opportunities between the up-link and the down-link and also they have no support for replaying frame aggregation and WiFi only losses.

In this degree project, we opt for trace-driven emulation. A trace-driven emulation is a good option because 1) testing takes place on the real network and 2) traces help in repeat-ability, the results and testing environment can be reproduced later. Therefore trace-driven emulation is preferred over simulation, testbeds, and experimentation.

Chapter 3

3 System Design

In this chapter, we justify the design decisions of NemFi. NemFi is a trace-driven emulator for WiFi, which consists of two phases: the record and the replay phase. In essence, NemFi extends the Saturator which is the existing state-of-the-art trace-driven emulator for cellular. However given the distinctions between cellular and WiFi, NemFi adapts the saturator in several ways to make it suitable to record and replay WiFi network conditions. In the sequel, we explain why the Saturator in its current state fails to accurately capture WiFi network conditions, and how NemFi is designed to address these limitations.

3.1 Consideration & existing problems

In efforts of creating a trace-driven emulator for WiFi, a major challenge is the collection of traces which would help in getting a picture of the constantly changing channel capacities. This is because traces contain packet delivery opportunities in the form of timestamps for the client machine. Therefore traces must display the link capacity of the wireless connection to capture the network behavior.

In Figure 4, we see the problem just discussed. Here a client connected to an access point is static and is trying to send traffic in order to estimate channel capacity. To avoid induced losses, the sender ends up sending less than the supported rate and misjudges the packet delivery opportunities. Saturating the wireless link is important concerning NemFi or for that matter any record and replay tool as we would like to replay just the WiFi losses in the replay phase. Thus avoiding the induced losses due to buffer overflows at the client machine during the period of the trace collection is essential.

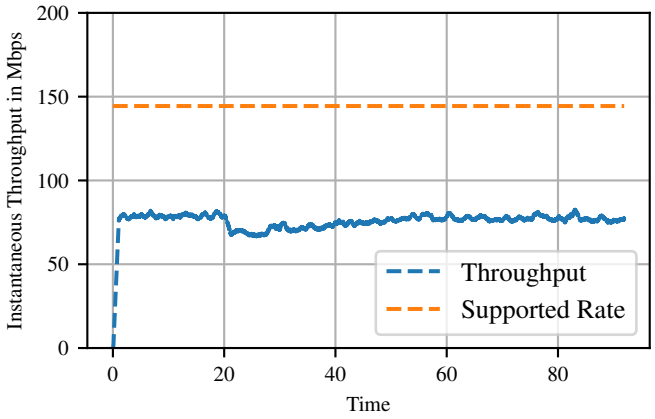


Figure 4: Underflow from ideal saturation rate for a static WiFi client (Throughput in Mbps vs Time in seconds)

There are other challenges in saturating the WiFi links namely frame aggregation and cross-traffic from other applications during the record phase. Due to frame aggregation, the client is capable of achieving a much higher link utilization by aggregating the acknowledgements for a group of received frames. This feature is at the heart of improvements in recent WiFi standards like IEEE 802.11 n, ac, and ax. Therefore the record and replay framework proposed must investigate and consider frame aggregation. Moreover, frame aggregation could be A-MSDU or A-MPDU with varying implementation parameters. Thus, a universal solution is required. Another factor affecting the recorded traces is cross-traffic generated by other applications during the process of getting the traces. This could affect the packet delivery opportunities to some extent.

In this degree project, we introduce *NemFi* to emulate different WiFi scenarios from recorded traces. It follows record and replay methodology. During the record phase, we are focusing on capturing the varying channel capacity of WiFi, and replay emulates the recorded conditions taking into account the losses as well as cross-traffic. We evaluate the NemFi's record with the Saturator, which was originally designed for cellular networks. Saturator for the comparison experiments sends on a very high sending rate to saturate the channel. This acts as a benchmark of achieved saturated throughput because we would be sending a lot more than WiFi link capacity allowing induced packet losses due to local buffer overflows.

Looking into the architecture, the system consists of two components. The overall emulator has a record component and a replay shell. While the NemFi's record captures the packet delivery opportunities over WiFi, record shell creates a new network namespace for itself before launching the shell; which is logically another copy of the network stack, with its routing tables, network devices and rules for the firewall.

While NemFi's record is built on top of state-of-the-art trace-driven emulator Saturator [1] which was made for LTE, replay shell is built on top of Mahimahi's [2] link emulator LinkSell, originally made for replaying recorded HTTP traffic. Mahimahi was chosen amongst other emulation frameworks introduced in Section 2.1.3, because of two main features possessed by it. First of all, it is very light and performance efficient. Then, it supports a very high degree of isolation. By using different network namespaces of Linux, creates the environment which allows multiple instances of MahiMahi shells to communicate using isolated individual traffics.

Design considerations and introduced features of the NemFi's record framework with respect to the reference, Saturator, are illustrated in the Figure 5

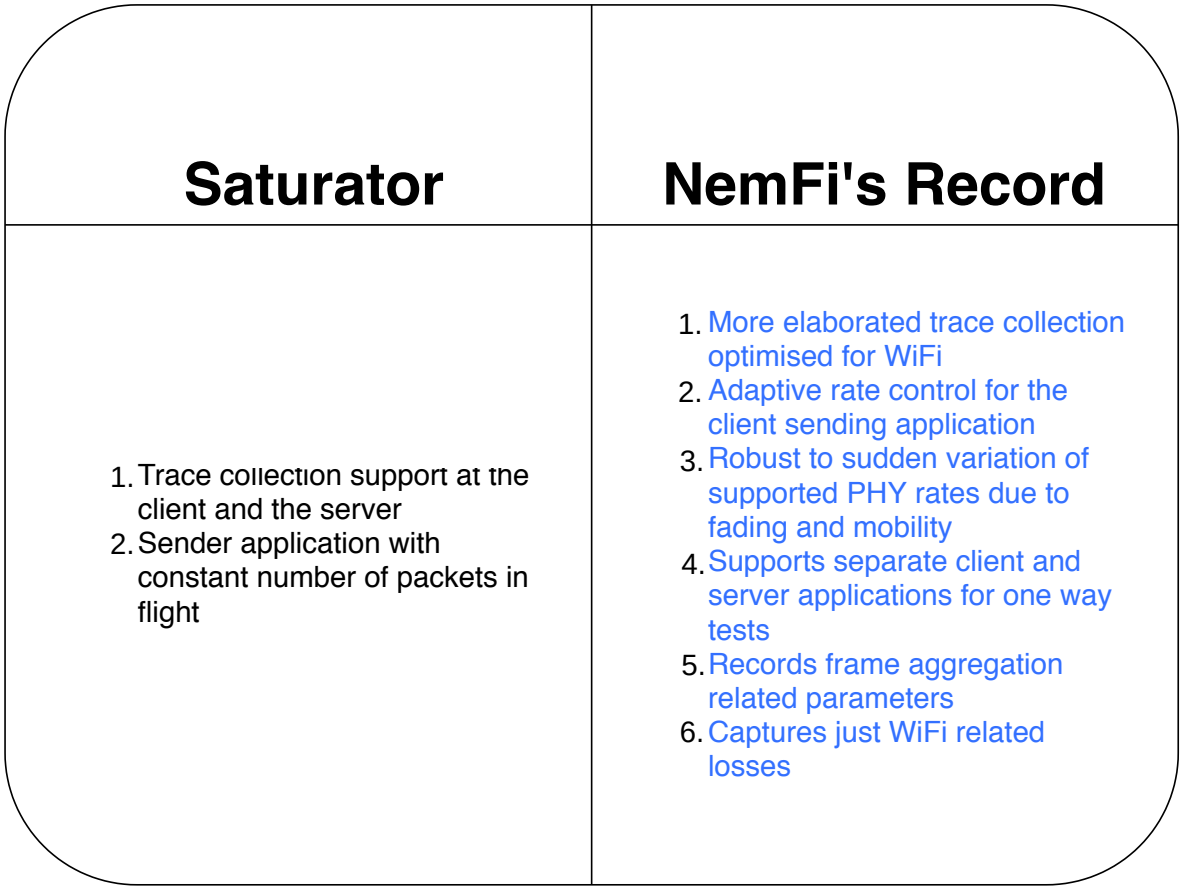


Figure 5: Design consideration of NemFi's record with respect to Saturator

Similarly, major considerations in the design of the NemFi's replay framework compared to the reference, MahiMahi's are shown in the Figure 6

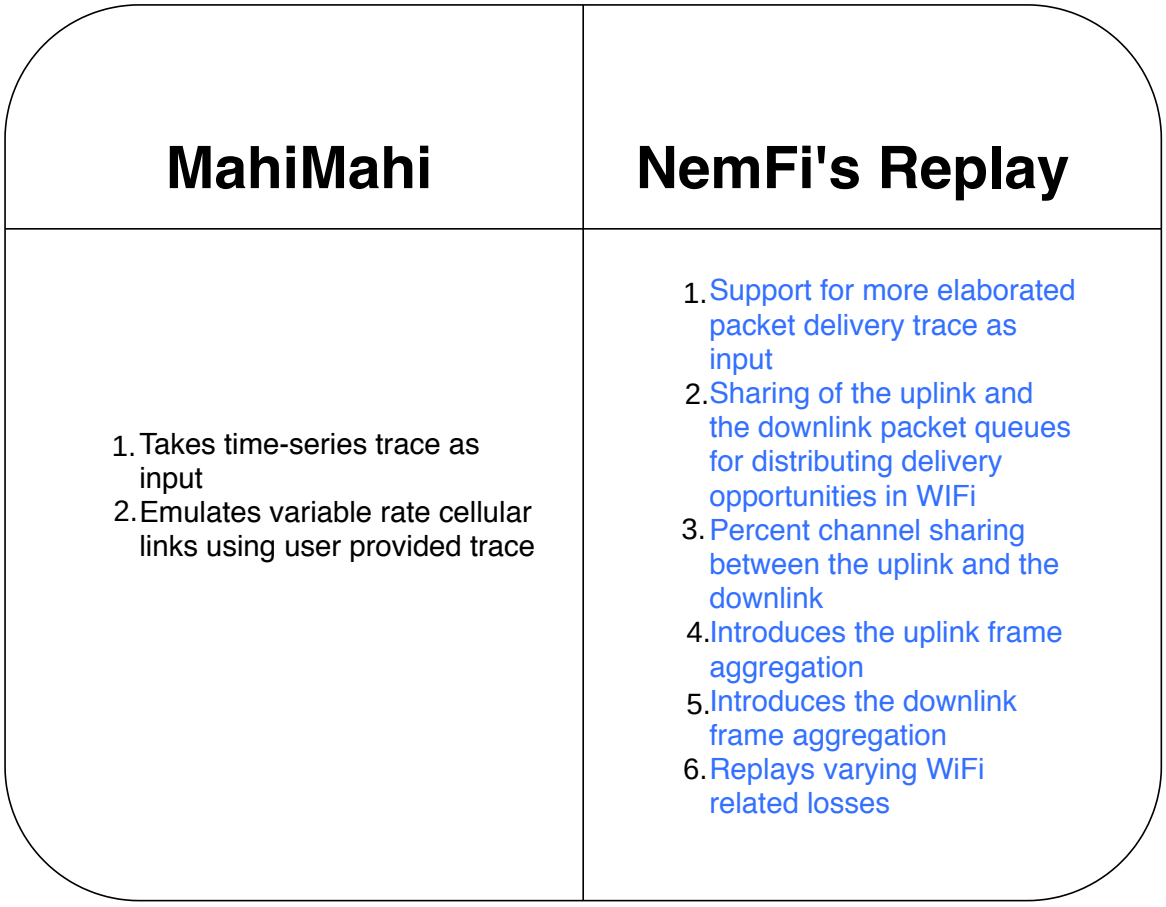


Figure 6: Design consideration of NemFi's replay with respect to MahiMahi(LinkShell)

The next sections describe respectively four major milestones in developing NemFi:

- ***Design and Implementation*** **of NemFi's record**
  This section describes designing an accurate packet delivery trace along with other necessary parameters that are necessary to estimate the wireless link condition at each instant in the record interval. After the design implementation is done which takes care of capturing every parameter proposed to be the part of the record trace.

- ***Performance Evaluation*** **of NemFi's record**
  This section does the preliminary evaluation of recorded traces in order

to be sure that all the research questions initially looked into could be tackled during the replay phase using the trace.

- ***Design and Implementation* of NemFi's replay**
  This section primarily focuses on designing methods for the usability of collected packet delivery traces and other information recorded to emulate all the WiFi features. Further, replay shell is implemented which creates a virtual platform for the user to send and receive traffic following the same behavior as WiFi itself.

- ***Performance Evaluation* of NemFi's replay**
  Here, after the implementation of the whole pipeline, we evaluate the overall emulator by various methods as well as to measure and justify the accuracy of the solution.

Chapter 4

4 NemFi's record: Design and Implementation

As discussed previously in section 2.3, the first step in the record-and-replay paradigm is the record phase. This phase concerns with designing a framework to record the WiFi network's characteristics in the form of a trace. The idea is to feed this collected trace to the developed emulator in order to replicate the same behavior as that found during the record phase.

4.1 Overview of the NemFi's record

NemFi's record consists of a client and a server connected over two links, the WiFi channel that we want to record and a reliable Ethernet link for the feedback. The client sends MTU sized packets over the WiFi link trying to saturate it at all PHY rates while the server replies with acknowledgements over the reliable link.

NemFi's record has two sender programs running at the client and the server respectively. Separate programs are needed to enable them to measure the up-link traffic, the down-link traffic, or both. The sending rate is primarily controlled by the window size, which represents a window of N packets in flight at an instant, which is maintained by each of the sender programs. Using the feedback packets, each sender adjusts the window size to ensure that the link is saturated without causing any self-induced packet loss. Both the client and the server store in their log the time each data or ACK packet is received, as well as its sequence number, and estimated RTT or 1-way delay. Using these logs, the up-link, and the down-link latency, throughput, and packet loss can be computed. The feedback consists of ACK packets sent to the sender for the packets received by the receiver. The sender can then keep sending consistently to saturate the link reliably. Therefore a separate interface is needed for feedback to ensure timely delivery of ACK packets to the sender, and to avoid any impact of feedback delay on the link saturation. If the interface that has to be saturated is also used for feedback, queuing

might cause enough delay for ACK packets to arrive on time, which causes a possibility that the link might not get properly saturated.

4.2 Need for elaborated packet delivery trace

Just the time-series of packet delivery opportunities in the trace is not enough for emulating WiFi. WiFi replay requires information like PHY rates information, losses, and transport-layer sequence numbers. Sequence numbers are useful in grouping delivery opportunities which do not contain lost packets between them. Up-link and down-link PHY rate at each instant of successful packet delivery is extracted from the client machine using the Linux *iw* command and appended to the output trace during the record phase. Optionally, more WiFi related parameters like *current bandwidth* and *guard interval size* are also recorded as metadata which could later be needed in improving the NemFi's replay.

NemFi's record thus stores the following elements in the recorded trace:

- *Timestamp* - Timestamps of successful packet delivery in the trace act as delivery opportunities for MTU packet delivery in the replay phase.

- *PHY rate* - PHY rate in WiFi represents the maximum bit rate that the wireless link could achieve. We use the PHY rate observed at different instances of the record phase as an indication to decide the number of packets to aggregate if the frame aggregation is present.

- *Sequence Number* - Sequence numbers of delivered packets in the trace helps NemFi's replay in tracking WiFi losses.

- *Instantaneous loss rate* - We use the observed instantaneous loss rate in the replay to emulate WiFi related losses. We calculate the instantaneous loss rate during the record from missing sequence numbers during the previous second.

4.3 Channel sharing between up-link and down-link

Up-link and down-link traffic for a given client compete for the WiFi channel. One of the major challenges for the design of replay was to verify if the up-link recorded trace is enough to characterize the WiFi channel. We observe that in case of bi-directional traffic with enough flows to saturate the wireless link, the channel is fairly/equally shared by the up-link and the down-link. In order to test this, we carry the following experiments with frame aggregation enabled and as well as disabled.

4.3.1 Up-link traffic only

Firstly, the AP was set to the bandwidth of 20 MHz, with IEEE 802.11n running on it. Despite being set to 20 MHz, channel bonding in 5 GHz which made it currently to work in 40 MHz(Channel 36) with MCS 15 during aggregation. MCS 15 has PHY rate of 300 Mbps. Running NemFi's record on just up-link only, showed that the client converges to the throughput of 251.5361 Mbps which was as expected.

Next, after disabling the frame aggregation in both up-link and down-link directions, client traffic switched to IEEE 802.11a which gives a max PHY rate of 54 Mbps. Running NemFi's record on just up-link only, resulted in client converging to the throughput of 43.334 Mbps which was as expected.

4.3.2 Bi-directional traffic

We first keep client and server machines with enabled frame aggregation. Running NemFi's record on both client and the server, resulted in both the client and the server converging to the throughput of 101.5 Mbps.

Again, we disabled the support for frame aggregation at the client machine. While executing NemFi's record on just up-link only, resulted in client converging to the throughput of 17 Mbps which was as expected.

Thus, we observe from the above experiments, that indeed there is a fair share of resource in the form of transmission opportunity in the wireless channel between the client and the server flows. This result allows us to record only in one direction i.e the up-link. Although, ideally channel is shared in

half by up-link and down-link flows; NemFi's replay gives users the flexibility to provide this as an input parameter when launching the replay shell.

4.4 Link capacity

NemFi's record is based on the principle of recording actual packet delivery opportunities, along with other necessary information for capturing the behavior of WiFi. In order to capture the packet delivery opportunities in a WiFi network, knowing the wireless link capacities is essential. At every instant of the record phase, we must send at a rate such that we hit the maximum achievable throughput. This will make sure that our collected traces have truly pictured the wireless channel between the client machine and the access point.

The first thing to investigate for the development of the record component for NemFi is, therefore, how to track WiFi link capacities. WiFi link capacity varies because of the rate adaptation being done by different nodes connected to the access point to avoid high losses. This continuous rate adaptation is done by the collaboration of the transmitter and receiver to deploy the best MCS for the given channel conditions. These different possible MCS values produce different supported up-link and down-link rates.

These MCS values not only contain the PHY rates but also the number of spatial streams and modulation information [3]. Rate Adaptation algorithm also controls, whether to transmit in legacy mode (802.11a or 802.11g) or in a non-legacy mode (802.11 n, ac or ax) in a SISO or MIMO, whether a Short Guard Interval (SGI) or a Long Guard Interval (LGI) is used and when to enable the frame aggregation. [4] Therefore, in NemFi's record the periodic updates of the MCS-based Up-link and Down-link supported PHY rates are used as the limit to aim for when collecting the saturated traces.

4.5 Frame aggregation

Next, in current WiFi standards, it is necessary to look into the effects of frame aggregation on the recording client and target server's sending capabilities in case of bi-directional traffic data generation. In order to understand

the effect of frame aggregation on rates achieved by the NemFi's record and the consequences of aggregation on the saturation channel capacities of WiFi, it is essential to find the answers to few research questions. One important question to investigate was whether perfect fairness is achieved between a single client and access point pair when both are competing for the channel access with enabled and disabled frame aggregation. The results showed that indeed with sufficient client and server sending rates, we achieve perfect fairness irrespective of the presence of different frame aggregation methods and their supported parameters (More on this discussed in section 6.2). This proves that we could divide the packet delivery opportunities into half while collecting the traces with only the client sending the data traffic for link saturation. This will be taken advantage of by NemFi's replay in the next section. Now, let us look into the overview of NemFi's record before describing the working and underlining algorithms in detail.

4.6 Method

4.6.1 Feedback routine

Routine 1 is present on the sender program at the client-side, which is called every time the sender receives an ACK packet from the server through the feedback interface. Apart from getting the updated feedback variables [Algorithm 1: line 2 and 3], the routine also keeps track of the last throughput observed on the link (*lastThroughput*) as well as the last PHY rate supported (*lastTheoreticalBound*) on the link [Algorithm 1: line 5 and 6]. Feedback variables are *currentRate* and *currentBitrateUplink*, which represent current throughput achieved and current up-link PHY rate supported at the link.

Algorithm 1 NemFi's Feedback Routine

1: **procedure** AckRoutine(*recvAck*) ▷ On receiving Ack from Server

▷ Update feedback variables

2: *currentRate* ← *calRate*()

3: *currentBitrateUplink* ← *calBitrateUplink*()

▷ Call the control algorithm

4: SatControl(currentBitrateUplink, currentRate)

▷ Keep track of last feedback received

5: *lastThroughput* ← *currentRate*

6: *lastTheoreticalBound* ← *currentBitrateUplink*

7: **end procedure**

These records are then passed on to the control algorithm [Algorithm 1: line 4]. The control algorithm 2 adjusts the behavior of the client's sender program to keep saturating the link without committing any induced losses. After the execution of Control Algorithm 2, the routine again waits for the next ACK from the server.

4.6.2 NemFi record's rate control algorithm

In rate control algorithm 2, as discussed in section 4.4.1, we get the periodic PHY rate values on the client-side. This current up-link PHY rate is passed on as an input variable to the control algorithm along with current throughput achieved. This acts as a theoretical bound [Algorithm 2: line 2] to the maximum PHY bit rate supported on the link. The actual usage of the link capacity depends on many parameters such as frame aggregation, type of aggregation i.e A-MSDU or A-MPDU, buffer sizes used as well as the hardware and processing delays. Typically the link utilization which denotes the fraction of supported bit rate lies between 80 to 90 percent with advanced aggregation schemes.

At any instant, the difference between achieved throughput and the PHY rate is the upper bound on the error of our estimated capacity[Algorithm 2:

line 3]. Next, the challenge is to figure out when the link is saturated. We define differential throughput gain as the difference between the last instance of observed throughput and the current rate [Algorithm 2: line 4]. Thus, differential throughput gain gives an idea of how much gain we achieved by window adaptation done by the last call to Control Algorithm 2.

Algorithm 2 NemFi's Rate Control algorithm

1: **procedure** SATCONTROL(*currentBitrateUplink*, *currentRate*) ▷ Input feedback variables

2: $theoreticalBound \leftarrow currentBitrateUplink$
3: $error \leftarrow theoreticalBound - currentRate$
4: $diffThroughptGain \leftarrow lastThroughput - currentRate$

▷ Check if we need window size adaptation
5: **if** $window < upperWindow$ & $satFlag \neq 0$ & $diffThroughptGain \neq 0$ **then**
6: $window \leftarrow window + alpha * error$
7: **else**
8: $satFlag \leftarrow 1$
9: **end if**

▷ Decrease window on MCS drop
10: **if** $lastTheoreticalBound > theoreticalBound$ **then**
11: $window \leftarrow 0.8 * window$
12: $satFlag \leftarrow 0$
13: **else if** $lastTheoreticalBound < theoreticalBound$ **then**
14: $satFlag \leftarrow 0$
15: **end if**
16: **return** ▷ Control Algorithm executed for this round
17: **end procedure**

The gain is practically large at the start of the application, but as we get closer to saturation link utilization throughput gain will decrease and decrease, until the arrival of a point when new packets are just queuing at the client-side. The approach is to detect this point and stop sending more,

to avoid buffer overflow and induced losses. This way the losses observed in the trace are actual WiFi losses and thus, they could be replayed.

In order to get to saturation link capacity at the client-side at a particular instant, a constant number of packets in flight, or constant window size is not sufficient. High constant window size would lead to overshooting ideal saturation sending rate and content increase in losses due to the buffer overflow. Whereas low constant window size would cause underutilization of available link capacities as seen in Figure 4. Therefore an adaptive solution is needed that somehow adjusts these windows during the time of recording.

In order to solve the above-stated problem, Algorithm 2 is introduced. The window size in Algorithm 2 is kept proportional to the error in theoretical bound and current throughput recorded earlier [Algorithm 2: line 6]. The proportionality constant again could either be a constant or a variable depending on the PHY rate supported at that instant. The constant alpha will control how aggressively this saturation state is being targetted. Initially, the saturation flag is false and as soon as we see that differential throughput gain has become zero, we assign saturation flag as true in order to avoid increasing the window size any further [Algorithm 2: line 8].

Moreover, on observing a PHY rate drop due to deteriorating channel conditions, the window size is decreased to eighty percent [Algorithm 2: line 10] and the saturation flag is turned false. This will cause the algorithm to get to a new saturation state by adapting the window size on the further reception of ACK packets. Drop in the window size could be as aggressive as fifty percent on major drops, but in mobile client scenarios such as WiFi, there are too many PHY rate fluctuations. Dropping the sending rate by a large percentage would result in more transient time for reaching the next saturation state. Eighty percent turned out to be reasonably quick in getting the saturation state even in extremely mobile scenarios which lead us to choose this value. Lastly, the saturation flag is also turned false on noticing a major PHY rate improvement to increase the window possibly further and adapt to a new saturation state [Algorithm 2: line 13].

4.7 Implementation details

To understand WiFi behavior and shortcomings in current network emulators for HTTP traffic and cellular networks, the first two months of this project were spent for extensive research and experiments. From findings like the effect of frame aggregation on achieved link utilization, hardware constraints in the form of WiFi standards support, induced losses due to over-sending to effects of cross-traffic on recording a wireless link were some of the milestones achieved during that phase.

Next, Saturator [1] was chosen as a reference for NemFi's record, and changes in record Framework as well as added features were done in C++. Several sub-modules in the Saturator needed to changed apart from the inclusion of Algorithm 1 and Algorithm 2 in the record framework. Most significantly two instances of NemFi's record were created to cater the special needs of the client and the server application while performing the record. Since most of the information in the record phase is retrieved from the packet-delivery trace, a lot of trace processing needs to be done. Python was used for this purpose(1000+ lines).

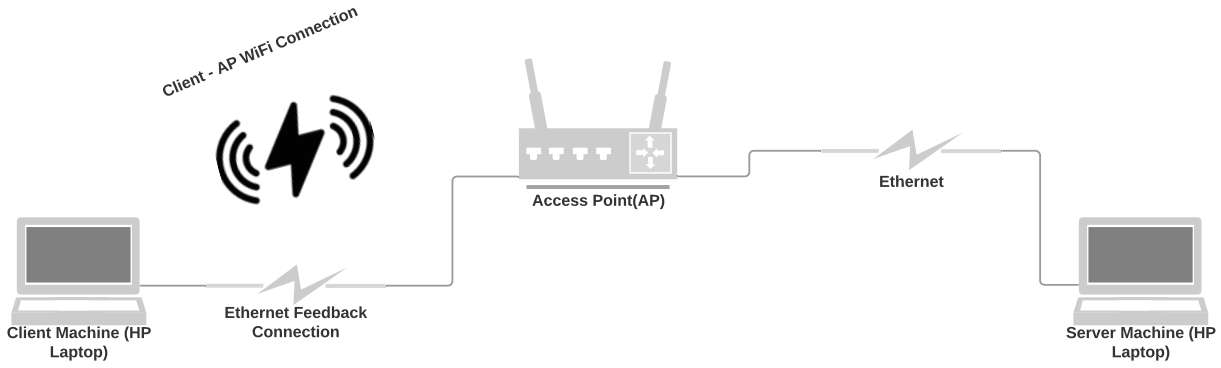
Chapter 5

5 NemFi's record: Performance Evaluation

This section discusses methods used to evaluate the NemFi's record as well as results and inferences from the experiments performed.

5.1 Experimental setup

The goal of experiments is capturing necessary packet delivery traces using instances of NemFi's record in a controlled WiFi network. Later in future work, public access points and outdoor scenarios will be tested. While we allow the sender to send UDP data packets on the wireless link, Ethernet feedback is proposed to avoid unnecessary WiFi losses for the server acknowledgements. Figure 7 describes the experimental setup.



(a) Static client

Figure 7: Experimental Setup of NemFi

1. **Client**: HP Elitebook 840 G2 laptop with both WiFi and Ethernet connection to access point
2. **Server**: HP Elitebook 840 G2 laptop connected to access point via Ethernet
3. **Access Point**: TP-Link AC1750 connected with an Ethernet feedback interface to the client as well as through WiFi.

The AP was a TP-Link AC1750, Archer C7 V5 which supports IEEE 802.11ac/n/a on 5GHz and IEEE 802.11b/g/n on 2.4GHz. We did our experiments with IEEE 802.11ac/n/a on 5GHz with a channel width of 20 MHz. It supports 5GHz Up to 1300Mbps. The two laptops are HP Elitebook 840 G2 with 5th Generation Intel Core i7-5600U 2.6 GHz (max turbo frequency 3.2-GHz), 4 MB L3 Cache, two active cores and four threads. Laptops have 8 GB RAM and 512 GB SSD hard drive.

The network interface card on the client-side is Intel Wireless 7265 with iwlwifi driver. Client and Server both have Intel I218LM Gigabit Network Connection (10/100/1000 NIC) cards for communications. Both the client and the server machines have freshly installed operating systems to avoid any unnecessary load on CPUs while running the setup. There are two NemFi record instances running on the client machine as well as the target server. The client instance constantly tries to send MTU sized packets on to the wireless link, in order to keep it saturated.

5.2 Experimental scenarios

There are two scenarios considered for validation. Both of them are taken indoors. In the *static* scenario, client and server machines are not moving with respect to the access point. Whereas in the *mobile* scenario, the server is stationary with respect to the access point but the client shows mobility in home/office environments.

5.3 Evaluation metrics

There are two evaluation metrics for the evaluation of NemFi's record. *Throughput* in Mbps and instantaneous *Packet Loss Rate* obtained during the record

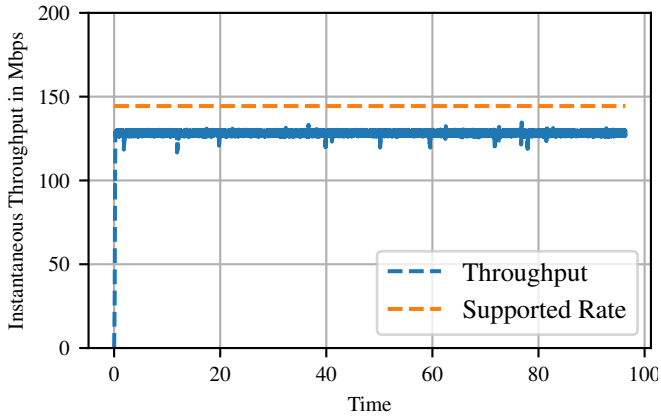
phase. In forthcoming sections, both the metrics will be investigated for the scenarios in which the client is static with respect to the access point as well as when it is mobile.

5.4 Validation

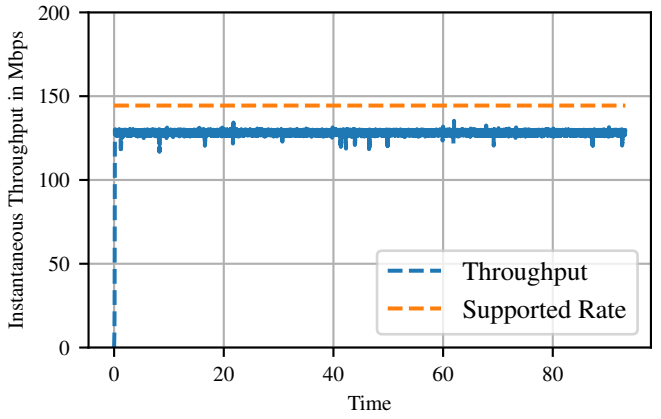
To validate NemFi's record next two sub-sections investigate whether throughput obtained from NemFi's traces reflect the capacity and whether the losses captured by the packet delivery trace are indeed WiFi related losses.

5.4.1 Throughput validation

In Figure 8, we test the throughput achieved by the client under two scenarios for the NemFi's record as well as state-of-the-art saturator [1] with a very high window size of 1500. The client is kept near the Access point in Figures 8a and 8b, whereas the client machine is mobile in Figures 9a and 9b. All the sub-figures show the actual throughput achieved vs supported up-link bit rate. The supported up-link bit rate is sampled every 100ms and average throughput over the same period. In ideal static scenarios, the PHY rate achieved in 20 MHz at the client side is pretty constant i.e 15 or equivalently PHY bit rate of 144.40 Mbits per second. Fairly stable throughput of 128.20 Mbits per second could be seen on reaching the saturation phase in both Figure 8a (NemFi's record) and 8b (Saturator).



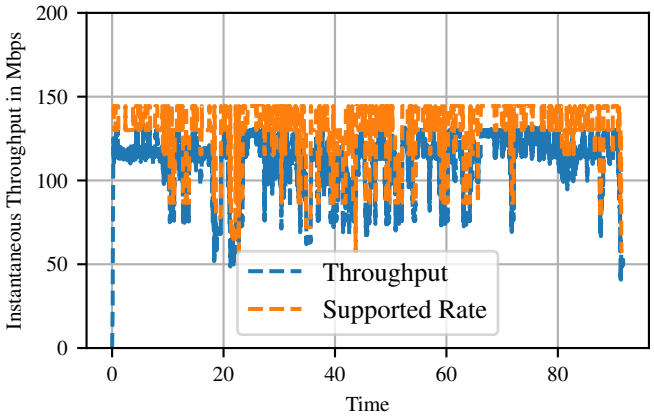
(a) NemFi's record



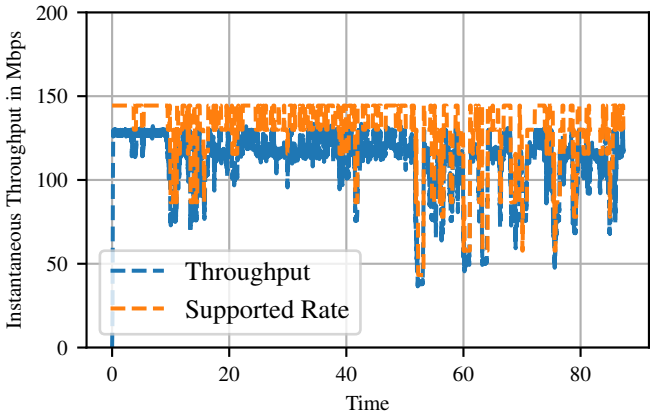
(b) Saturator

Figure 8: Time-Series(Seconds) of throughput observed in the static scenario

In the mobile scenarios too, we see in Figure 9a that the fluctuating supported rate is tracked very well by NemFi's record. Figure 9b is state-of-the-art saturator [1] with buffer overflows. We see that NemFi's record tracks the supported up-link rates with similar preciseness.



(a) NemFi's record



(b) Saturator

Figure 9: Time-Series(seconds) of throughput observed in the mobile scenario.

Link capacity estimation

We use the model of Hora [5] to get the estimated link capacity of the WiFi for different PHY rates. We use the results of this model to get the instantaneous link utilization of NemFi's record. A detailed explanation of the usage of the model [5] for getting theoretical Link Capacity in order to validate NemFi's record may be found in Appendix (A.1).

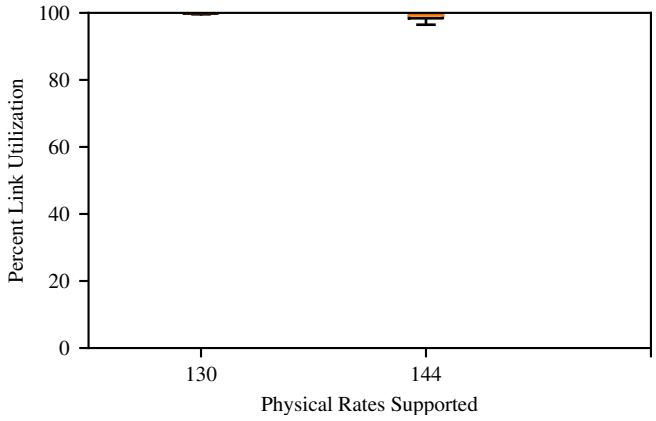
Physical Rate(P)	Estimated Link Capacity(LC)
144	116.69
130	108.03
117	97.33
115	95.83
104	83.42
86	64.03
78	61.43
72	59.73
65	54.50
57	45.36
52	42.31
43	34.20
28	24.22

Table 1: Estimated Link Capacities

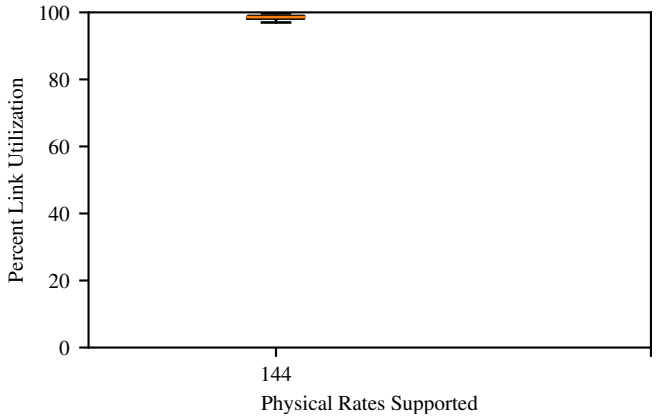
Comparison of estimated link utilization with that of NemFi's record

Now that estimated link capacities are obtained as of table 1, we could now validate the record process of NemFi's record too to see if around 100 percent of link utilization is achieved during the record phase. Results are analysed by looking at link utilization for all the observed physical rates by the client over multiple numbers of experimental runs.

Next in Figure 10, we look at the mean percentage link utilization achieved and deviation from the estimated ideal rate by the client for NemFi's record as well as saturator with a very high window size of 1500 in both static scenarios. In Figure 10a and Figure 10b, we see that link utilization near 100 percent is observed throughout the simulation duration for all the physical rates encountered by both Saturator as well as our record.



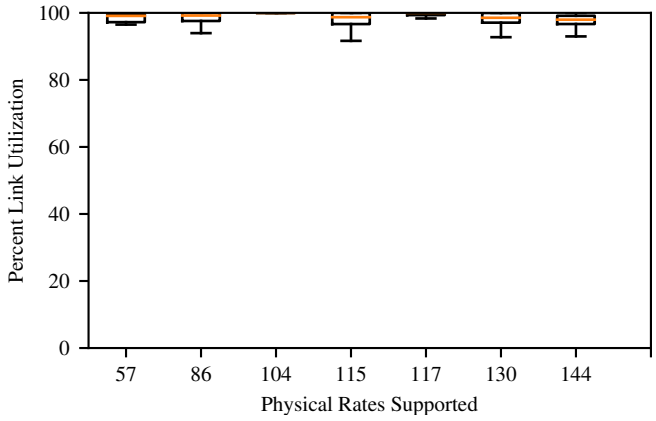
(a) NemFi's record



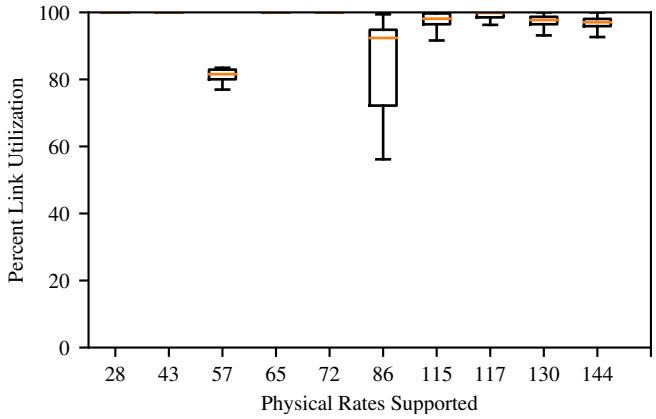
(b) Saturator

Figure 10: Percentage link utilization observed over ten experimental runs in the static scenario

On the other hand in Figure 11, in dynamic scenarios too, due to MCS fluctuations, there is always slight variability in mean link utilization. Looking at these utilization values over ten runs, we observe that NemFi's record achieves better link utilization than saturator itself for the encountered physical rates. This could be explained by the fact that NemFi's record due to the rate adaptation algorithm does not send more than capacities in severe sudden drops of channel quality. It prevents unnecessary processing at the client machine of packets that will eventually be dropped.



(a) NemFi's record



(b) Saturator

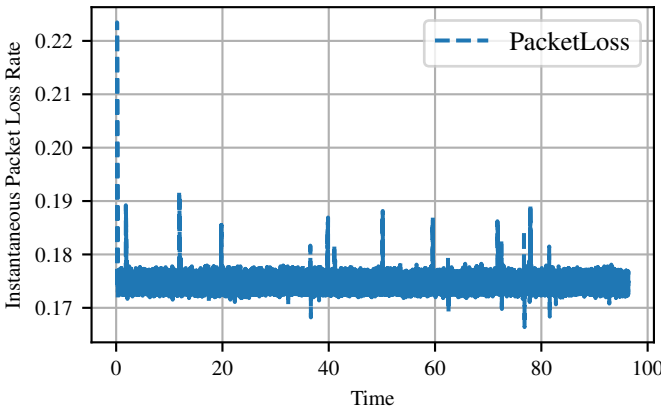
Figure 11: Percentage link utilization observed over ten experimental runs in the mobile scenario

5.4.2 Packet loss validation

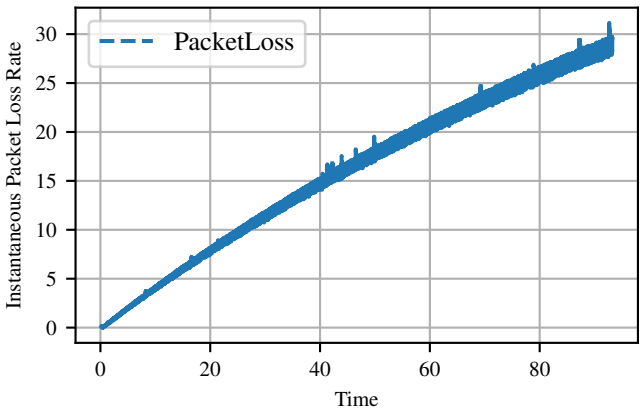
Packet losses are one of the most important quality metrics of wireless communication. NemFi's record aims at capturing these packet losses in order to use them for emulation. Recorded losses would only make sense when they are actual WiFi related losses. Thus, in this section, we would conduct some experiments to check the validity of losses shown by NemFi record's trace.

The packet loss incurred by the client is tested for NemFi's record as well as saturator with a very high window size of 1500 packets in flight in Figure

12. The client is kept in close proximity to the Access point in Figures 12a and 12b, whereas the client machine is mobile in Figures 13a and 13b. The packet loss in all of the sub-figures is the instantaneous percentage packet loss rate.



(a) NemFi's record



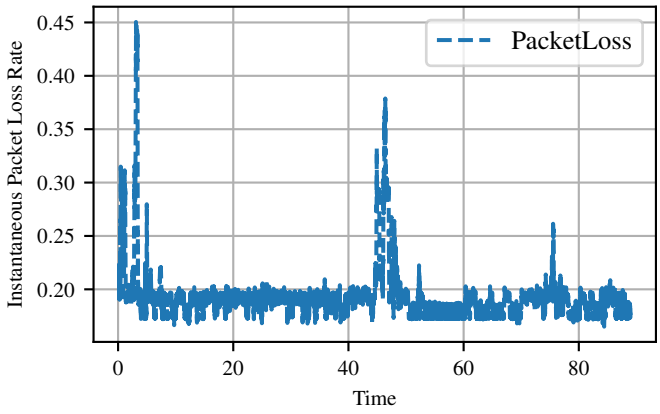
(b) Saturator

Figure 12: Time-Series(seconds) of packet Loss observed in the static scenario

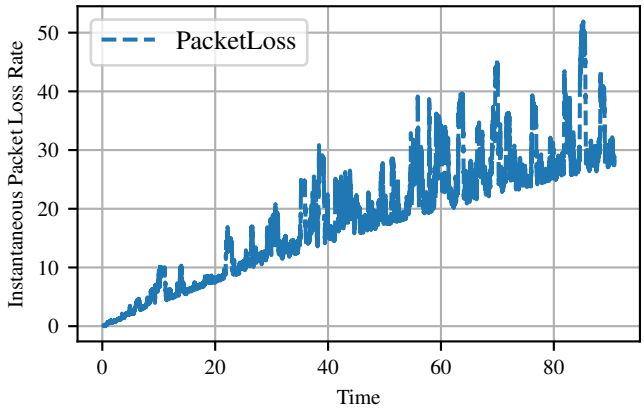
Both in the ideal static scenarios as well as mobile scenarios i.e Figures 12a and 13a, NemFi's record incurs negligible losses of around 0.20 percent throughout the recording duration which are WiFi transmission losses. But on the other hand in Figures 12b and 13b, saturator shows a constant increase in losses as high as 50 percent, due to continuous induced losses.

These losses are pretty high for the chosen window size of 1500 in saturator due to packet drops by the buffer overflows at the client machine.

Thus, NemFi's record is the framework that better captures the WiFi and not show any misleading losses to be replayed. This is one of the significant issues that this project addresses.



(a) NemFi's record

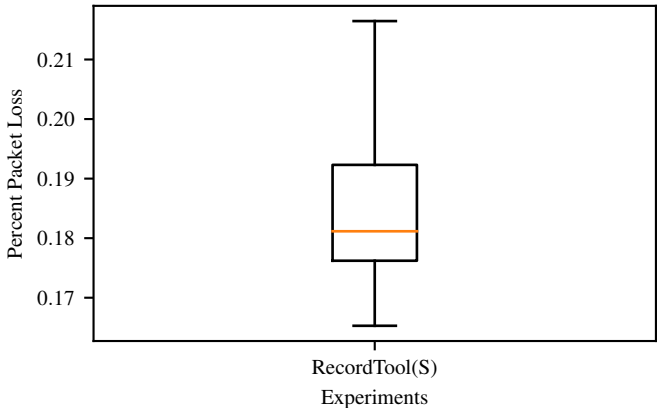


(b) Saturator

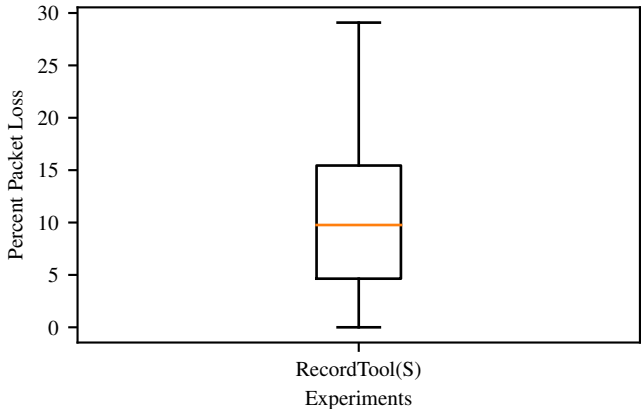
Figure 13: Time-Series(seconds) of packet loss observed in the dynamic scenario

Next, instead of losses in terms of time-series as shown previously, we look at the results for ten runs of the experiment. Now let us look into the losses

observed by the NemFi's record vs the Saturator in static client scenario. We observe that while NemFi's losses depict WiFi losses as confirmed from client driver's dump which comes to around 0.18 percent; Saturator showed high losses of around 10 percent in the same conditions due to induced buffer losses locally at the client machine.



(a) NemFi's record



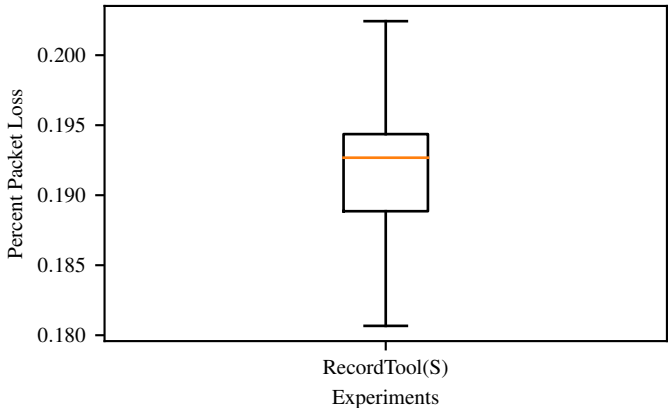
(b) Saturator

Figure 14: Packet Loss observed over ten experimental runs in the static scenario

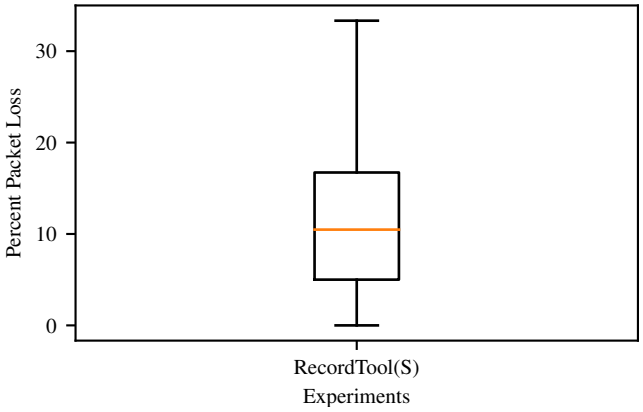
Similarly, on looking across ten experimental runs in mobile scenarios for the NemFi's record and Saturator, it is seen that the same trend is being followed as just above in static client use-case. While NemFi's record showed around 0.193 percent losses, induced losses caused around 11 percent loss in

the Saturator.

Therefore, concluding observations can be drawn in terms of packet losses observed by the NemFi's record. Using Algorithm 1 and rate control algorithm 2, true WiFi related losses are observed in the recorded traces. This is essential in order to replay WiFi losses later in the replay phase. Looking back into rapidly changing WiFi link capacities due to rate adaptation algorithms, the adaptive control method proposed by NemFi completely takes care of channel saturation as well as induced packet drops.



(a) NemFi's record



(b) Saturator

Figure 15: Packet Loss observed over ten experimental runs in the mobile scenario

Chapter 6

6 NemFi's replay: Design and Implementation

This chapter describes our replay method, which takes as input the packet-delivery traces collected during the record phase. As already discussed in Section 2.1.3, MahiMahi is used as a reference for the development of NemFi's replay. Though, MahiMahi being designed specifically for fixed-rate links or, cellular links does possess many challenges to be solved in order to make the framework suitable for WiFi.

6.1 Sharing the delivery opportunities between up-link and down-link queues

NemFi's replay places an incoming packet into either the up-link or down-link packet queue depending upon the direction of arrival. At each delivery opportunity, NemFi replay selects which queue to serve based on a scheduling algorithm.

NemFi's replay uses the algorithm 3 inspired by *weighted round-robin*, for the same purpose. In algorithm 3, firstly we add the action of polling the up-link and the down-link socket [Algorithm 3: line 2]. Then *waitTime* is calculated after which NeemFi's poller will check the added actions. This time is the difference between the timestamp of the current index in the packet delivery trace and the current timestamp of the system itself[Algorithm 3: line 4].

After getting a callback from any of the sockets, the packet is read into the *linkQueue* which contains the separate up-link and queues[Algorithm 3: line 5]. Then a random number (*randGen*) is generated. If the number is greater than *percentShare* and down-link queue is not empty, then queue direction for the usage of opportunity is chosen to be down-link. Else up-link queue is chosen for the consumption of current packet delivery opportunities[Algorithm 3: line 14]. Finally, actual usage of the delivery opportunity

in contention happens for the queue in the direction just decided before. Afterward, the index is increased by one for the next opportunity to be competed for[Algorithm 3: line 15].

Algorithm 3 NemFi's Queue sharing algorithm

```
1: procedure QueueShare(percentShare)
                                        ▷ Poll up-link and down-link sockets
2:     poller− > AddAction(pollUplinkSocket)
3:     poller− > AddAction(pollDownlinkSocket)

4:     waitTime ← getWaitTime()
                                    ▷ Address the callbacks after waitTime
5:     if callback(pollUplinkSocket) then
6:         linkQueue− > readPacket(uplinkSocket.read(), up)
7:     else if Callback(pollDownlinkSocket) then
8:         linkQueue− > readPacket(uplinkSocket.read(), down)
9:     end if

            ▷ Whenever replay shell is rationalized/emulated till present
10:    randGen ← rand()
11:    direction ← up

12:    if randGen > percentShare and size(packetQueueDown) > 0
   then
13:        direction ← down
14:    end if

                                    ▷ Actually use the delivery opportunity
15:    if direction == up then
16:        useDeliveryOpportunity(up)
17:    else if direction == down then
18:        useDeliveryOpportunity(down)
19:    end if

20: end procedure
```

6.2 Replaying frame aggregation

To replay the frame aggregation, NemFi uses the model of Hora [5] to estimate the total number of aggregated frames given the current PHY rate. These values are found in table 3. As we already have PHY rates observed at each entry in the packet delivery trace, an inference can be made for the number of packets to group together for sending it to the output queue.

In the given trace for the NemFi's replay, all packets which are part of the same aggregate(A-MPDU or A-MSDU) have practically the same delivery timestamps. Thus, whenever the NemFi calls the replay shell to emulate either the up-link or the down-link until the present time, *AGG(P)* (table 3) number of packets are sent to the output queue. The only case in which the whole set of opportunities will not be grouped is when there are missing sequence numbers in them. This is because of the losses in the aggregated frame during the record phase. In that case, as soon as the missing sequence number in the input trace is detected, no more packets are grouped in the same emulation opportunity. This prevents the false usage of the actual group of frame aggregated packets sent during the NemFi's record.

6.3 Replaying WiFi losses

After capturing the actual WiFi losses by eliminating the induced losses at the client machine, NemFi's task for emulating WiFi losses becomes easier. Every time a packet is being read from the socket into the corresponding packet queue, we take a look at the instantaneous loss rate at the current index in the packet delivery trace.

If the loss rate turns out to be positive then a new random number between zero and one is generated. If it is less than the current loss rate, then we drop the packet read from the socket and that packet is not appended to either of the packet queues.

6.4 Implementation details

A significant time was spent in choosing the replay framework to be used as a reference for NemFi's replay. After careful consideration and experiments,

MahiMahi and the emulator introduced by it in the form of LinkShell was chosen. One of the most challenging but essential tasks was to merge threads for the up-link and the down-link queues. This is because in the case of WiFi, we need to actively share packet delivery opportunities between up-link and down-link flows.

C++ was used as a language for this task of merging threads. Then, the poller also needed to be changed simultaneously for polling up-link and down-link sockets from a single thread. Afterward, new features of replaying frame aggregation and losses were added apart from adding the Algorithm 3 for realistically sharing the up-link and down-link queue.

Chapter 7

7 NemFi's replay: Performance Evaluation

Now that we have completed the design and implementation of both the record and the replay phases of NemFi, we move to evaluate the overall solution. Apart from the traffic generated by the NemFi's record, there is traffic being generated by devices in the vicinity too. These flows are referred to as cross-traffic. Before evaluating the replay itself, let us figure out how the input packet-delivery opportunities behave in the presence of cross-traffic that could be present during the record phase.

7.1 Consideration of cross-traffic

First, we want to understand if NemFi had any effect on the cross-traffic; i.e., whether it will take over the wireless medium by consuming all the available delivery opportunities. That would be problematic because it means that NemFi does not capture the real WiFi condition and available bandwidth.

We run the record in the same experimental setup as described in section 5.1 but now there are two separate machines also connected to the same access point of interest. On these two machines, we sequentially create single-thread Secure Copy Protocol(SCP), single-thread TCP, 5, and 10 threaded TCP as well as UDP traffic. We fix the bandwidth of UDP cross-traffic to 50 Megabits per second.

The result of the experiment in Figure 16 shows that NemFi's record is not saturating the channel all by itself as in previous experiments without cross-traffic such as Figure 8a. It shows that indeed NemFi captures available bandwidth. This is because the presence of cross-traffic leads to the consumption of packet-delivery opportunities by flows of the non-client machines. This also proves that NemFi's record does contain information cross-traffic in the recorded trace itself.

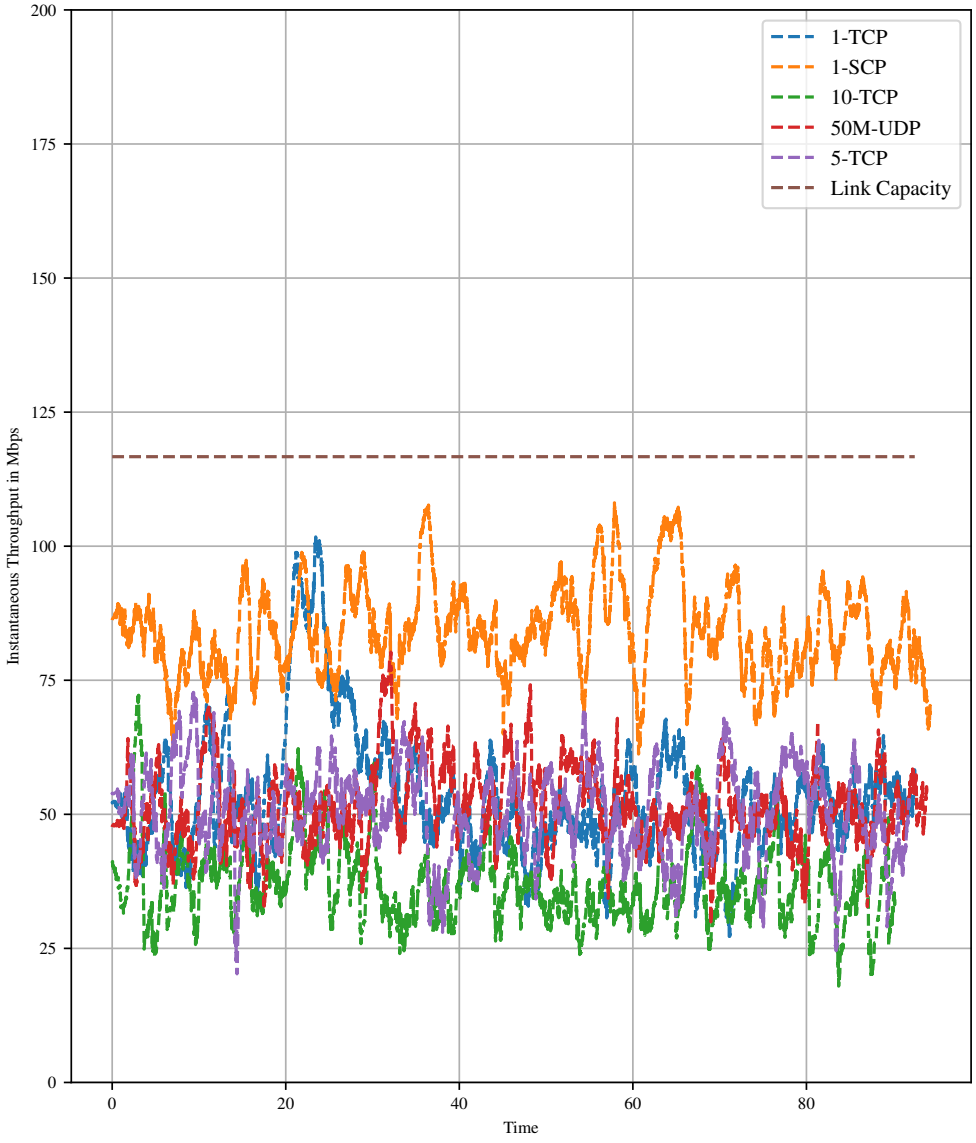


Figure 16: Instantaneous throughput(Mbps) vs Time(s) observed by our NemFi's record in the presence of cross-traffic

7.2 Experimental results

7.2.1 Scenario

Before describing the experimental conditions in detail, we have a look at an application that is used for validating NemFi's replay: *Iperf*. Iperf is a network performance measurement tool. It creates TCP or UDP data streams to measure the throughput between the client and the server. The replay will be evaluated with metrics of throughput achieved by iperf in actual WiFi connection versus that inside the replay shell of NemFi. Iperf with client and server instances is used as the application to evaluate the replay. Iperf is run with the traces collected by the record module of Nemfi and we compare the throughput of the application in the real network environment with that of emulated conditions.

We perform the experiments in the static indoor scenarios with the client being stationed a few meters away from the access point. All of the experiments have been repeated 10 times for 50 seconds. We run the experiments in the same experimental setup as described in section 5.1 except the fact that NemFi replays the application traffic over the earlier feedback Ethernet interface to avoid WiFi related losses twice.

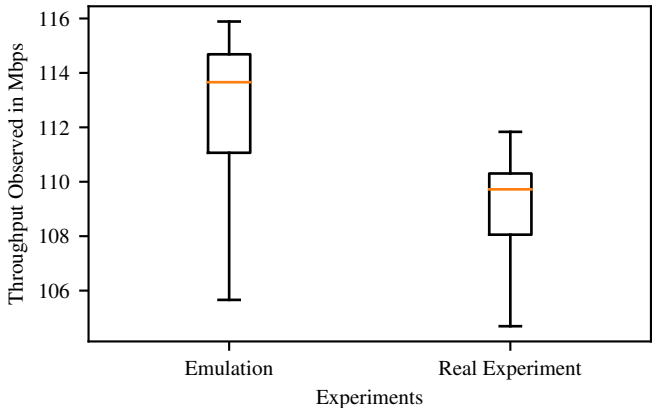
We run the NemFi's replay immediately after running the iperf on WiFi so that the wireless environment is similar. Moreover, we repeat the experiment multiple times in order to take into account the variability of the wireless channel over the runs of application in NemFi and the WiFi itself.

7.2.2 Replay with just WiFi losses

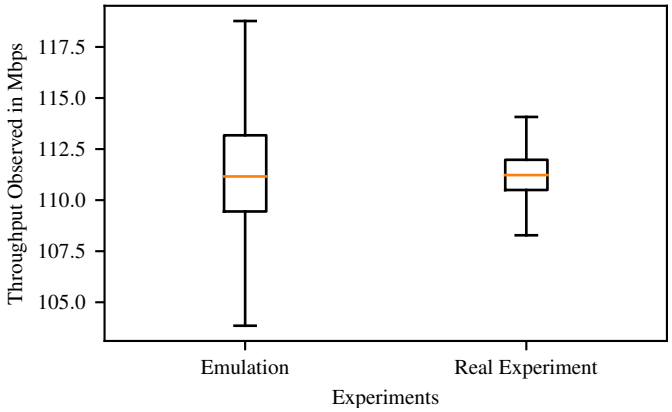
In this sub-section, we look into the throughput achieved by the client in up-link transmission to the server. We test for single-thread as well as a 5-threaded TCP connection. We see in the Figure 17 that in both kinds of TCP connections, the mean throughput observed in the NemFi's replay is within the bounds of five percent to that of iperf over real WiFi conditions.

Sightly more throughput is observed occasionally in replay, which is explained from the fact that PHY rate based frame aggregation needs the transmission opportunity *txop* duration. Txop duration used is currently fixed in NemFi's replay for the calculation of frame aggregation parameters

such as the number of frames aggregated in an opportunity. The number of frames aggregated is currently calculated as a function of the current PHY rate. This is a fair approximation as seen by the results in the Figure 17. But sometimes in the real hardware after incurring losses, the station is allowed to aggregate a lower number of frames even with a high PHY rate. In the future, we could fine-tune this device-specific behavior of frame aggregation parameters in NemFi's replay.



(a) Single Thread TCP

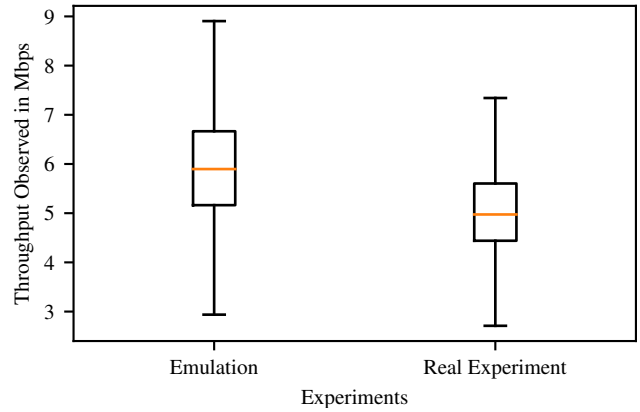


(b) 5 - Threaded TCP

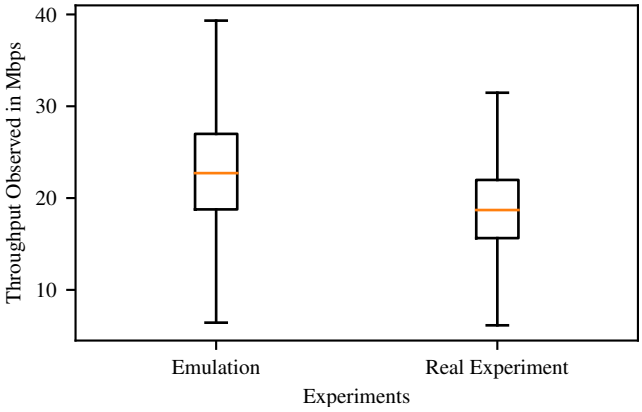
Figure 17: Throughput observed by iperf with real WiFi conditions versus NemFi

7.2.3 Replay with forced losses

We measure the performance of Iperf with losses induced on the client machine with the help of *TC command*. TC command helps the client to randomly drop a percentage of packets from the transmit queue in the kernel. We look into the throughput achieved by the client up-link transmission for a single-thread and 5-threaded TCP connection in the presence of 10 percent packet losses induced by TC on the top of WiFi losses. We again see in the Figure 18 that in both kinds of TCP connection, the mean throughput observed in the NemFi's replay is within a few percents to that of the iperf over real WiFi conditions. This proves the validity of the NemFi's replay in the case of severe losses.



(a) Single Thread TCP



(b) 5 - Threaded TCP

Figure 18: Throughput observed by iperf with real WiFi conditions versus NemFi with forced losses

8 Conclusion and Future Direction

8.1 Conclusion

In this degree project, we presented a survey of existing trace-driven emulation techniques for wireless networks. We showed how the state-of-the-art emulator for the wireless network is tailored for cellular networks and the reason it fails to faithfully emulate WiFi networks. The first problem we identified was the fact that existing trace-driven emulator for cellular causes severe packet losses when recording WiFi network conditions. To this extent, we designed and developed a new routine for feedback during the record phase which facilitated us to effectively implement the rate control algorithm. This adaptive rate control algorithm accurately recorded the rapid variations in WiFi PHY rates while keeping track of WiFi related losses.

Moreover, before the implementation of formulated methods, we also looked upon fundamental questions for the record phase. NemFi also investigated aspects like capturing cross-traffic during the record as well as whether one-directional record is enough to get a picture of a bi-directional wireless link. Information regarding the frame aggregation was also logged into the traces collected by the NemFi's record apart from the packet-delivery opportunities.

After validating the correctness of the record by comparing the link utilization achieved by NemFi and the theoretical link utilization for that PHY rate, we moved on to designing the sharing of up-link and down-link queues for the distribution of packet delivery opportunities during the NemFi's replay. A variation of the weighted round-robin was introduced to formulate NemFi's queue sharing.

Afterward, we introduced methods for replaying the captured frame aggregation and WiFi related losses in the packet-delivery trace. We tested the correctness of replay by evaluating the achieved throughput of iperf with real WiFi conditions verses the replay environment of NemFi for single as well as multiple threaded TCP. TC command was used to force losses which enabled us to verify the behavior of replay in presence of heavy losses.

Thus, we conclude that the major contribution of this degree project is to

introduce a new set of methods and algorithms in the form of NemFi, which successfully brings and adapts the record-and-replay paradigm for WiFi emulation.

8.2 Future works

This degree project achieved the objectives of introducing an accurate record-and-replay method for WiFi. But in doing so, it opens up many directions for further research and testing in the field of WiFi emulation.

In this degree project, however, the WiFi emulation solution, NemFi is kept confined to being tested for static scenarios with respect to the client as well as dynamic indoor office scenarios for validation of the implemented record-and-replay pipeline. Outdoor testing could be done in the future to adapt the NemFi if needed. Also, Access point in consideration for the test purposes is taken as local instead of public one for more controllable and infer-able results to validate the solution.

During the replay phase, we could re-investigate device-specific behavior of frame aggregation parameters that are needed for the replay to fine-tune the existing NemFi's replay. We could look into the effectiveness of replay in the presence of cross-traffic. Moreover, it would be interesting to test the behavior of the most popular form of internet traffic i.e video traffic over NemFi. Further research could be initiated to look into the evaluation of latency-critical applications on WiFi using our proposed solution.

8.3 Ethical and sustainable aspects

During this degree project, we never violate the ethics of data collection. Especially in the phase of collecting packet-delivery traces, none of the information collected contains personal attributes of any firm or an individual. Privacy has been respected throughout this work. A personal WiFi network was created and used for validation purposes in order to minimise any interference caused to other individuals during the testing phase.

Regarding sustainability, our software solution is kept as light as possible to not only be efficient in terms of performance but also be power efficient. NemFi does not require any extra hardware excluding regular client-server

machines and the access point. This helps us in aligning with our goals of conservation and sustainability.

9 Bibliography

References

[1] Winstein, K., Sivaraman, A. and Balakrishnan, H. "Stochastic forecasts achieve high throughput and low delay over cellular networks" Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 13. 2013.

[2] Netravali, R., Sivaraman, A., Das, S., Goyal, A., Winstein, K., Mickens, J. and Balakrishnan, H. "Mahimahi: Accurate Record-and-Replay for HTTP" In 2015 USENIX Annual Technical Conference, USENIX ATC 15, pp. 417-429. 2015.

[3] https://www.cwnp.com/link-adaptation/

[4] Grunblatt, R., Guerin-Lassous, I., Simonin, O. Study of the Intel WiFi Rate Adaptation Algorithm. CoRes 2019 - Rencontres Francophones sur la Conception de Protocoles, Evaluation de Performance et Experimentation des Reseaux de Communication, Jun 2019, Saint-Laurent-de-laCabrerisse, France. pp.1-4. ffhal-02126333f

[5] Hora, D., Doorselaer, K., Oost, K., Teixeira, R., Diot, C. Passive Wi-Fi Link Capacity Estimation on Commodity Access Points. Traffic Monitoring and Analysis Workshop (TMA) 2016, Apr 2016, Louvain-la-Neuve, Belgium. ffhal-01292633f

[6] Netsim simulator by tetcos. https://www.tetcos.com/wlan.html.

[7] Ns-2 network simulator. https://www.isi.edu/nsnam/ns/

[8] Ns-3 network simulator. https://www.nsnam.org/.

[9] Omnet++ network simulator. https://omnetpp.org/intro/.

[10] Opnet network simulator. http://opnetprojects.com/opnet-networksimulator/.

[11] Wikipage of ns-2 network simulator. http://nsnam.sourceforge.net/wiki/index.php/User_Information.

[12] Agrawal, P., and Vutukuru, M. Trace based application layer modeling in ns-3. In 2016 Twenty Second National Conference on Communication (NCC) (March 2016), pp. 1-6.

[13] Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H. Core: A real-time network emulator. In MILCOM 2008 - 2008 IEEE Military Communications Conference (Nov 2008), pp. 1-7.

[14] Deng, S., Netravali, R., Sivaraman, A., and Balakrishnan, H. Wifi, lte, or both?: Measuring multi-homed wireless internet performance. In Proceedings of the 2014 Conference on Internet Measurement Conference (New York, NY, USA, 2014), IMC'14, ACM, pp. 181-194.

[15] O'Dea, S. Number of smartphone users worldwide from 2014 to 2020 (in billions).https://www.statista.com/statistics/330695/number-of-smartphone-usersworldwide/, 2016.

[16] Dinesh, S., and Sonal, G. Qualnet simulator. International Journal of Information Computation Technology. ISSN 0974-2239 Volume 4, Number 13 (2014), pp. 1349-1354

[17] Ivanic, N., Rivera, B., and Adamson, B. Mobile ad hoc network emulation environment. In MILCOM 2009 - 2009 IEEE Military Communications Conference (Oct 2009), pp. 1-6.

[18] Karimi, P., Mukherjee, S., Kolodziejski, J., Seskar, I., and Raychaudhuri, D. Measurement based mobility emulation platform for next generation wireless networks. In IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (April 2018), pp. 330-335.

[19] Midoglu, C., Wimmer, L., Lutu, A., Alay, O., and Griwodz, C. Monroe-nettest: A configurable tool for dissecting speed measurements in mobile broadband networks. In IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (April 2018), pp. 342-347.

[20] Noble, B., Nguyen, G., Satyanarayanan, M., and Katz, R. Mobile network tracing, 1996.

[21] Noble, B. D., Satyanarayanan, M., Nguyen, G. T., and Katz, R. H. Trace-based mobile network emulation. SIGCOMM Comput. Commun. Rev. 27, 4 (Oct. 1997), 51-61.

[22] Raychaudhuri, D., Seskar, I., Ott, M., Ganu, S., Ramachandran, K., Kremo, H., Siracusa, R., Liu, H., and Singh, M. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In IEEE Wireless Communications and Networking Conference, 2005 (March 2005), vol. 3, pp. 1664-1669 Vol. 3.

[23] Zhou, J., Ji, Z., Varshney, M., Xu, Z., Yang, Y., Marina, M., and Bagrodia, R. Whynet: A hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. In Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation Characterization (New York, NY, USA, 2006), WiNTECH '06, ACM, pp. 111-112.

[24] WiFi Standards. https://en.wikipedia.org/wiki/IEEE_802.11

[25] Network emulation. https://en.wikipedia.org/wiki/Network_emulation

[26] Frame aggregation. https://en.wikipedia.org/wiki/Frame_aggregation

[27] Skordoulis, D., Ni, Q., Chen, H., Stephens, A., Liu, C., Jamalipour, A. (2008). IEEE 802.11N MAC frame aggregation mechanisms for next-generation high-throughput WLANs. Wireless Communications, IEEE. 15. 40 - 47. 10.1109/MWC.2008.4454703.

A Appendix

A.1 Calculation of estimated link capacity

To calculate the link capacity for a given PHY rate, P, we divide the UDP payload of the A–MPDU by its transmission time where B_0 is the fraction of time the AP is busy sending beacons.

$$LC(P) = \frac{AGG(P) \times UDPpayload}{A - MPDUTxDelay(P)} \times (1 - B_0) \tag{1}$$

In order to calculate the UDP payload of the A-MPDU frame, we multiply the UDP payload per IP packet by the number of MPDUs sent at P, AGG(P). AGG(P) is the number of MPDUs per medium access. Frame aggregation in IEEE 802.11n reduces the MAC overhead by allowing the delivery of multiple aggregated Mac Protocol Data Units (A-MPDUs) in a single medium access.

We calculate this AGG(P) values experimentally for our experimental setup by fixing the physical rate in the static scenario, sending at a much higher rate with a constant window size of 2500, and measuring the number of MPDUs transmitted in a transmission opportunity. These AGG(P) values are shown in table 3

We compute the A-MPDU transmission delay by using the 802.11 protocol parameters (table 2) to model an A-MPDU exchange of N packets of size S using PHY rate P.

$$\begin{aligned} TxDelay(N, S, P) = T_{AIFS} + T_{BO} + 3 \times T_{SIFS} + T_{RTS} \\ + T_{CTS} + T_{ACK} + DATA(N, S, P) \end{aligned} \tag{2}$$

With no losses, the back-off timer does not exponentially increase. The back-off timer is chosen using a uniform distribution between 0 and CWmin, giving the expected value of CWmin2. We estimate the delay to transfer the

data block as:

$$DATA(N, S, P) = \frac{22 + N \times (S_{MH} + S)}{P} + T_{PH} \tag{3}$$

Above, T_{PH} is the transmission delay of the PHY header, and S_{MH} is the MAC header size which is 38 bytes. We add 22 trailing bits (16 + 6) to form the OFDM symbols since we do not consider padding bits. To calculate the estimated link capacity(LC), we use S = 1444 bytes, UDP payload of 1400 bytes and N = AGG(P).

We consider frame exchanges using Best Effort Access Category and the use of implicit Block Ack Request. We consider control frames with PHY rate at 24 Mbps, and assume that the PHY rate to transmit a control frame is lower than that of a data frame. Finally, we calculate the estimated link capacities for each physical rate as shown in table 1

As the WiFi link capacity varies over time, our model estimates the link capacity for a given time interval. Even though the PHY rate changes over time, the AP uses only one PHY rate for each frame. Thus, we can obtain "instant" link capacity measurements by applying the model previously described. Let P(t) be the PHY rate used at t and FDR(t) be the frame delivery ratio at t. We estimate the link capacity for the time interval as follows:

$$LC(t_0, \tau) = \frac{1}{\tau} \int_{t_0}^{t_0+\tau} FDR(t) \times LC(P(t)) \, \mathrm{d}t \tag{4}$$

$$FDR(t) = 1 - frameLoss(t) \tag{5}$$

where frameLoss(t) is the instantaneous frame loss ratio. We periodically sample the PHY rate of the last data frame, with periodicity λ over an esti-

mation period τ

In our context, we use the following parameters,

Parameter	Value
T_{AIFS}	43 μs
T_{BO}	139.5 μs
T_{SIFS}	16 μs
T_{RTS}	28 μs
T_{CTS}	28 μs
T_{ACK}	32 μs
T_{PH}	32 μs

Table 2: Model Instance Parameters

Physical Rate(P)	AGG(P)
144	42
130	42
117	38
115	38
104	25
86	25
78	20
72	20
65	16
57	16
52	10
43	10
28	8

Table 3: Aggregate A-MPDU sizes in the experimental setup

TRITA-EECS-EX-2020:620
www.kth.se