Maynooth University
National University of Ireland Maynooth

Practical Realisation and Validation of a Wi-Fi Policing Algorithm

A dissertation submitted for the degree of
Doctor of Philosophy

By:

Hessan Feghhi

Under the supervision of:

Dr. David Malone

Hamilton Institute
National University of Ireland Maynooth
Ollscoil na hÉireann, Má Nuad

To my beloved Audrey,
and my caring mother.

Contents

List of Figures

List of Tables

Declaration

I hereby declare that I have produced this manuscript without the prohibited assistance of any third parties and without making use of aids other than those specified.

The thesis work was conducted from February 2010 to October 2015 under the supervision of Dr. David Malone in Hamilton Institute, National University of Ireland Maynooth. This work was supported by Science Foundation Ireland (SFI) grant *08/SRC/I1403*.

Maynooth, Ireland,

22/10/2015

Acknowledgement

Having clear goals and finding one's path in life is a luxury most people look for years to achieve. Even when you find exactly what you desire to be, life presents its challenges and obstacles. To overcome these obstacles, there is nothing more valuable than the support and assistance of those who decide to help you, out of love, compassion, kindness, or even a sense of responsibility. At the beginning of my Ph.D. dissertation, I would like to acknowledge those individuals who helped me during this stage of my life.

First and foremost, my sincere gratitude goes to Dr. David Malone. With his great knowledge and experience, he showed and walked me through the bridge from technology to science. For a student like me who began his Ph.D. with an engineering mindset, his help was a truly valuable asset. Added to that, he is one of the nicest people that I know; his kindness and his eagerness to help his students makes him more than just a supervisor, and more of a good friend who is always there when you need them. I will always be indebted to him for everything he has done for me.

Secondly, I'd like to thank Prof. Douglas Leith, head of Hamilton Institute, for his great help and valuable collaboration to my work, and Dr. Paul Patras, a former colleague in Hamilton Institute and a collaborator in the second and third years of my Ph.D. study. I also thank Dr. Francesco Gringoli from the University of Brescia, Italy, for his great work, which gave me the means to most of the experimental research I did, and also for his support and friendly help towards understanding and applying his work.

I have some very good memories from my time in Hamilton Institute; a brilliant research environment with many first-class researchers. I thank all my current and former colleagues in Hamilton Institute who provided such a superb research environment. I like to particularly thank Fernando López-

Caamal, my first office-mate and the first good friend I made in Hamilton Institute, Alessandro Checco, with whom I share many good memories, Esteban Abelardo Hernandez-Vargas, Vahid Samadi Bokharaie, Sonja Stüdli, Andrés Peters, Jane Breen, Karl O'Dwyer, Mahsa Faiz Rahnemoon, Bahar Partov, Mohammad Karzand, and Cristina Cano, for their sincere friendships and all the good times we had together. I would also like to thank Andrés García Saavedra who has been a good friend and a sincerely helpful colleague during the very short time we've been acquainted.

As a student in Hamilton Institute, there are two people I cannot thank enough; Ms. Rosemary Hunt and Ms. Kate Moriarty, the administrators of Hamilton Institute for always being nice and helpful, and doing what they possibly could to help make our Ph.D. life as paperwork-free as possible.

My special thanks go to Dr. Rouzbeh Razavi, who has been there for me since the first day I came to Ireland, and who has been like a brother to me. With his unconditional support and his friendly and helpful advices, I have been able to make good decisions that I wouldn't have made otherwise.

I would also like to sincerely thank Audrey, my fiancé, who always looked up to me since the very beginning of our relationship, and had faith in me even at times when I didn't have faith in myself. Her love and companionship got me through many of the hardships I faced during my Ph.D. life.

And finally, being a dutiful son, I thank my parents, for believing in me and always helping me in my path. I would especially like to thank my mother, who raised me as a single mom, and always did all she could so her children could get the best education, and dedicated her life solely to the success of her children. I also thank my brother, Saman, for always looking up to me, and believing in everything I did.

Abstract

Node misbehavior has attracted much research interest. However, much of the focus of prior research is on detection of node misbehavior, and little work has been done on counteracting that misbehavior. Those who do address this issue lack robustness, flexibility, or feasibility for implementation on real hardware.

As the IEEE 802.11 standard's medium access mechanism is decentralized, stations can potentially refuse to abide by the standard and gain performance benefit at the cost of the performance of compliant stations. In this thesis, we study, extend, and implement a policing algorithm that has been previously introduced for IEEE 802.11 b/g networks, and make it practically feasible for implementation on real hardware. We provide proof that the extended algorithm is robust and can effectively address the problem of policing IEEE 802.11 node misbehavior by eliminating the performance advantage for non-compliant stations. We outline the domain of scenarios this algorithm can be adapted to, and those it is not designed for.

To prove the effectiveness of the scheme in a real network, we implement it on real wireless adapters using the OpenFWWF firmware, conduct a wide range of experiments for different network scenarios. We provide results that confirm the extended algorithm's correct functionality for cases it is designed to support. We also consider the results in light of newer features of IEEE 802.11 standard. We conclude that the extended policing algorithm can in fact force stations to comply with the IEEE 802.11 standard (although some new IEEE 802.11 features may impair its functionality) and its application does not result in network degradation, and it is effective even under undesirable network conditions.

List of Abbreviations

Abbreviation	Meaning
AC	Access Category (for QoS)
ACK	Acknowledgement Frame or Packet
AIFS	Arbitration Inter-frame Space
AP	Access Point
BA	Block Acknowledgement
BAR	Block Acknowledgement Request
BEB	Binary Exponential Backoff
BSS	Basic Service Set
CBR	Constant BitRate
CCA	Clear channel assessment
CCMP	Counter Mode Cipher Block Chaining Message Authentication Code Protocol
CFP	Contention Free Period
CP	Contention Period
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CTS	Clear to Send
CW	Contention Window
DCF	Distributed Coordination Function
DIFS	Distributed Inter-frame Space
DLS	Direct Link Setup
DMA	Direct Memory Access
DS	Distribution System
EDCA	Enhanced Distribution Channel Access

EDCF	Enhanced Distribution Coordination Function
EIFS	Extended Inter-frame Space
FCS	Frame Check Sequence
FOSS	Free and Open-Source Software
FTP	File Transfer Protocol
HAL	Hardware Abstraction Layer
HCCA	HCF Controlled Channel Access
HCF	Hybrid coordination function
HT	High-Throughput
IBSS	Independent Basic Service Set
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISM	Industrial, Scientific, and Medical (radio bands)
LLC	Logical-Link Control
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
MBSS	Mesh Basic Service Set
MMIO	Memory-Mapped Input and Output system
MMPDU	MAC Management Protocol Data Unit
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
MTU	Maximum Transmission Unit
MWO	MicroWave Oven
NAV	Network Allocation Vector
OFDM	Orthogonal Frequency-Division Multiplexing
PC	Program Counter/Personal Computer
PCF	Point Coordination Function
PHY	Physical layer (802.11)
PLCP	Physical Layer Convergence Protocol/Procedure
QBSS	QoS Enhanced Basic Service Set
QoS	Quality of service
RCMTA	ReCeive Match Transmitter Address
RF	Radio Frequency
RNG	Random Number Generator
RTS	Request to Send

RTT	Round-Trip Time
RX	Receive/Reception
SHM	Shared memory
SIFS	Short Inter-frame Space
SDR	Software-Defined Radio
SPR	Special-Purpose Register
STA	Station
TBTT	Target Beacon Transmission Time
TCMA	Tiered Contention Multiple Access
TCP	Transmission Control Protocol
TIM	Traffic Indication Map
TKIP	Temporal Key Integrity Protocol
TS	Traffic Stream
TSF	Timing Synchronization Function
TSPEC	Traffic SPECification
TX	Transmit/Transmission
TXOP	Transmission Opportunity
UDP	User Datagram Protocol
VoWLAN	Voice over Wireless LAN
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Networks
WNIC	Wireless Network Interface Card
WPA	Wi-Fi Protected Access

CHAPTER 1

Introduction

In this chapter, we discuss the motivations behind the work of this thesis and provide an overview of the material presented in the following chapters.

1.1 Motivation

WLANs have attracted much research interest in recent years. This ranges from models to analyze the performance of existing network protocols (e.g. [1, 2]), to the experimentally evaluated modifications of standards (e.g. [3, 4]). The focus of this thesis is on Wi-Fi networks, not only from a theoretic perspective, but also from a practical point of view. The term Wi-Fi is a trademark used to denote any wireless local area network (WLAN) that is based on the IEEE 802.11 standard [5].

One of the interesting subjects in IEEE 802.11 research is the study of node misbehavior. The protocol used in the initial version of this standard is designed to provide fair medium access to all users. [1] However, as we will see later in Chapter 2, the decentralized nature of the channel access mechanism used in the protocol clears the way for some greedy users to gain benefit by cheating on the standard. With the wide availability of open-source wire-

[1] Though we will see in Section 3.2.1 that fairness itself is a complicated concept. Besides, new features such as service differentiation prioritize some traffic, which means that the medium is not shared among users in a fair manner.

less drivers and firmware, this has become a real issue in IEEE 802.11-based networks.

Different methods have been proposed to detect and mitigate the effects of such misbehavior in the wireless network. One of these methods is the policing algorithm, introduced in Chapter 5. Proposed by Dangerfield et al. [6] for IEEE 802.11 b/g networks, this algorithm has a few benefits over other methods, namely (i) it is agnostic to the type of misbehavior: it only takes into account the throughput, (ii) it provides a countermeasure against non-compliance without dissociating the greedy stations, and (iii) it involves no modification on the (possibly misbehaving) stations, or any message-passing among stations and/or with the access point. The algorithm's implementation scope includes only the access point.

There are, however, shortcomings to the policing algorithm. The first problem is when it comes to the measurement of the compliant throughput. In the original work, this measure is assumed to be known. The second issue involves the amount of penalty applied to non-compliant stations, which is only enough to equalize the throughput of all stations, but not enough to mitigate the channel degradation caused by the penalty. Thirdly, a station that is aware of the policing algorithm might be able to play smart and gain benefit regardless. There is yet another shortcoming to said algorithm, which is that it fails to consider the more recent advancements in the IEEE 802.11 standard.

The main contribution of this thesis is to amend the policing algorithm and address the said issues. We verify that we have addressed the challenges above by showing that our estimator provides a satisfactory estimate of compliant throughput, and that compliant stations achieve close to their expected throughput when policing is applied.

We supplement analysis with experiments to show that non-compliant stations with either constant or bursty traffic do not see a gain under the policing scheme. We provide a series of experiments that test the performance of the extended policing algorithm under different traffic types and conditions. We use commercial off-the-shelf devices in our testbeds. These are the same wireless adapters that come in your laptop or your desktop PC. The ability of some of these devices to be reprogrammed enables us to use them as test devices. Thus we can implement the policing algorithm (and possibly others)

and put it to practice in a cost-effective setting.

1.2 Overview

We begin by providing a background of the IEEE 802.11 protocol in Chapter 2. Details provided in this chapter will equip the reader with enough information about IEEE 802.11 required for this work. The complete IEEE 802.11 standard and its amendments are far beyond the scope of this thesis. Further, Chapter 3 provides a review of the literature around the subject, and helps put the work in this thesis into perspective.

Chapter 4 covers the programmability of Broadcom BCM43xx-based wireless cards, which are used extensively in this work. We describe how we reprogram the cards and their Linux drivers to implement our algorithms. Online documentation on this highly flexible hardware is not sufficient for a beginner to start implementing their ideas on it. Our main goal in Chapter 4 is to provide this introduction and help the keen developer understand the architecture and thus be able to reprogram the cards.

Further into the thesis, we tackle the problem of node misbehavior that can occur in an IEEE 802.11 network as we mentioned in the previous section. We introduce Dangerfield's policing algorithm, and amend it to address its shortcomings in Chapter 5. In Chapter 6 we implement the policing algorithm on real hardware, and provide experimental results from our testbed where the scheme is implemented at the access point.

Finally, in Chapter 7 we summarize what we talk about throughout the thesis and discuss possible extensions to the present work.

1.3 Publications

The following journal publication was prepared and published in the course of this doctorate:

> P. Patras, H. Feghhi, D. Malone, and D. Leith, "Policing 802.11 MAC Misbehaviours," *Mobile Computing, IEEE Transactions on*, accepted and to appear, pp. 1–15, 2015 [7]

The following conference publications were also prepared and presented in the course of this doctorate:

> H. Feghhi, P. Patras, and D. Malone, "Practical node policing in 802.11 WLANs," World of Wireless, *Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a,* pp.1,3, 4-7 June 2013 [8]

> H. Feghhi, and D. Malone, "WiFo: A diagnostic tool for IEEE 802.11 MAC," *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a,* pp.1,10, 14-17 June 2015 [9]

1.4 Resources

There are some resources available online for extra work generated during the course of this doctorate, which the reader may find useful to refer to. The following three videos are taken from a demo presented during WoWMoM 2013 for policing[8]. The description of each video describes the exact scenario for the particular test case.

- Scenario 1: https://www.youtube.com/watch?v=UPPhnG4mNQc
- Scenario 2: https://www.youtube.com/watch?v=cyGrJdT3ERM
- Scenario 3: https://www.youtube.com/watch?v=eVnoYfafWZc

Below are the GitHub repositories for the diagnostic tool described in Appendix A:

- WiFo front-end: https://github.com/hessan/wifo
- WiFo back-end: https://github.com/hessan/wifoserver

CHAPTER 2

IEEE 802.11 Background

This chapter provides a description of the IEEE 802.11 protocol and its aspects that are relevant to the purpose of this thesis.

2.1 Introduction

Wireless Local Area Networks (WLANs) have become popular during the past decade. The flexibility and the convenience they provide has made them the dominant type of network for home users and nowadays you can find them in many public places such as department stores, hotels, restaurants and even public transport vehicles. For home users, getting stuck near a network socket in order to surf the Internet has nearly become a myth. They can now move around and surf with speeds comparable to wired internet. The ease of installation adds to the value and popularity of these networks.

Today the dominant protocol for WLANs is IEEE 802.11 [5] which is maintained by the IEEE LAN/MAN Standards Committee (IEEE 802). Since its original release in 1997, IEEE 802.11 has been developed and many amendments have been introduced to the standard, such as IEEE 802.11b [10], IEEE 802.11g [11], IEEE 802.11e [12], and IEEE 802.11n [13]. We will provide details on some of these amendments later in this chapter. The need for higher connection speeds, efficiency, and increasingly larger numbers of users has been the reason behind all the effort. In this chapter we will describe the IEEE

802.11 protocol as background information for this thesis. We will cover the basics, but the focus will be on what is relevant to our work.

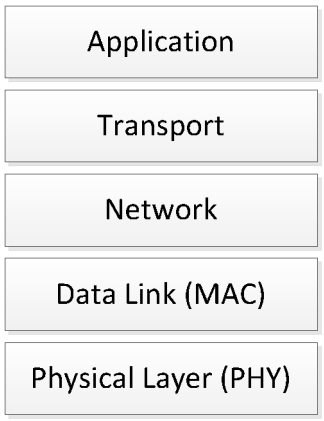


Figure 2.1: Network protocol stack

Figure 2.1 demonstrates a simplified version of the network model described in [14]. The top three layers are responsible for taking information to its rightful destination for consumption, once it is transmitted successfully. The bottom two layers are responsible for transmission and are involved in the IEEE 802.11 standard. In this thesis we are particularly interested in the MAC layer which is a sublayer of the data link layer.

We use the term "frame" to refer to a transmission unit in the data link layer. A frame is a small piece of data along with MAC-level headers. The structure of a frame depends on the protocol used in the data-link layer. An IEEE 802.11 frame can be of three main types: *data*, *management*, or *control*.

A data frame contains the actual information that is to be carried. Control frames assist in the delivery of data frames; they administer access to the wireless medium and provide MAC-layer reliability functions. Management frames are used to provide services that are simple on a wired network; for example, establishing the identity of a network station (STA)[1] is easy on a wired network because network connections require dragging wires from a central location to the new workstation, but this requires an "association"

[1]The terms "node" and "station" are used to describe each user in a network, such as a laptop or a mobile device.

procedure in IEEE 802.11 and is carried out using the corresponding management frames. We will describe the detailed structure of an IEEE 802.11 frame later in Section 2.7.2.

2.2 Modes of Operation

IEEE 802.11 WLAN architecture provides a number of operational modes, corresponding to different network topologies. This section provides a brief description of the types of networks IEEE 802.11 supports.

2.2.1 Basic Service Set (BSS)

A basic service set consists of a set of wireless STAs connected to a central node called the access point (AP), which could be connected to a wired network. This mode of operation is commonly known as the infrastructure mode. Coordination between the AP and the stations is managed through a special kind of frame called beacon[2]. APs send beacon frames at regular intervals (usually every 0.1 second). Timing synchronization is one of the applications of beacon frames and is described in Section 2.7.1.[3] A basic service set is identified by a service set identifier (SSID), which is selected and advertised by the access point.

2.2.2 Independent Basic Service Set (IBSS)

If there is no central node, and stations in a WLAN connect to each other in a peer-to-peer fashion, the network is called an independent basic service set (IBSS). The IBSS is the most basic type of IEEE 802.11 LAN. A minimum IEEE 802.11 LAN may consist of only two STAs. This mode of operation is possible when IEEE 802.11 STAs are able to communicate directly. Because this type of IEEE 802.11 LAN is often formed without pre-planning, for only as long as the LAN is needed, this type of operation is often referred to as an ad hoc network. The SSID for an IBSS is determined by the node that starts the network.

[2]Beacon is an example of management frames.
[3]They are also used for network identification and to broadcast network capabilities.

2.2.3 Mesh Basic Service Set (MBSS)

A mesh BSS is an IEEE 802.11 LAN consisting of autonomous STAs. Inside the mesh BSS, all STAs establish wireless links with neighbor STAs to mutually exchange messages. Further, using the multi-hop capability, messages can be transferred between STAs that are not in direct communication with each other over a single instance of the wireless medium. From the data delivery point of view, it appears as if all STAs in a mesh BSS are directly connected at the MAC layer even if the STAs are not within range of each other. The multi-hop capability enhances the range of the STAs and benefits wireless LAN deployments.

2.2.4 Extended Service Set (ESS)

An extended service set is a set of two or more interconnected BSSs that share the same SSID. The ESS network appears the same to the link layer as an IBSS network. STAs within an ESS may communicate and mobile STAs may move from one BSS to another (within the same ESS) transparently. The difference between an ESS and an MBSS is that, unlike an MBSS, an ESS does not have access to a distribution system (DS), so different BSSs cannot be located in disjointed areas.

2.3 Distributed Coordination Function

The MAC layer consists of a series of rules that determine how nodes should access the medium and send information. How transmissions are physically performed is the responsibility of the PHY layer. In this section we will describe the scheme used in the MAC layer in IEEE 802.11 networks. Before we begin, two points are worth mentioning. First, when we talk about medium or “carrier”, we simply mean the air (or any space that conducts electromagnetic waves), since we are transmitting over the air via antennas and the air is responsible for carrying our signals as wires would in a wired LAN. We may also refer to the air as “channel” since the medium is split into different frequency ranges and we usually use only one per network and this makes that particular channel our medium. The second thing worth noting is that the

radio transmission is half-duplex. This means that devices cannot send and receive simultaneously: at any given time only one of these can be done.[4]

2.3.1 Carrier Sensing

The scheme used in the MAC layer is called Distributed Coordination Function (DCF), which employs a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) system. As the name suggests, the method depends on carrier sensing. Carrier sensing is the act of observing the medium for incoming signals. In CSMA/CA, whenever we have a frame to transmit, we first sense the medium to make sure it is idle (i.e. nobody else is already transmitting). One of the two situations will occur:

1. The channel is sensed idle and has been idle for more than a time interval called DCF Inter-frame Space (DIFS)[5]: In this case the transmission begins immediately.

2. The channel is sensed busy: In this case the station waits until the channel is idle again for a DIFS period, and prepares for the exponential backoff procedure.

In addition to physical carrier sensing, a complementary mechanism called the Network Allocation Vector (NAV) is used in IEEE 802.11. NAV is a "virtual" carrier sensing mechanism or, in other words, a logical abstraction which limits the need for physical carrier sensing at the air interface to facilitate power saving. The MAC layer frame headers contain a *duration* field that specifies the transmission time required for the frame, during which time the medium will be busy. The stations listening on the wireless medium read the duration field and update their NAV, which is an indicator for a station on how long it must defer from accessing the medium.

[4]If a radio transceiver can send and receive at the same time, it is called full-duplex. Full-duplex radio has attracted attention like many areas of wireless networking, and there has been great progress in this area. For instance, in [15] they propose a technique for full-duplex radio using self-interference cancellation, and in [16] they design and prototype a full-duplex Wi-Fi. However, the IEEE 802.11 standard is still based on half-duplex radio.

[5]This is in the original IEEE 802.11 MAC standard. Newer versions of the standard introduce arbitration inter-frame space (AIFS) instead, which will be described in Section 2.4.

Standard	Slot Time (μs)	SIFS (μs)	DIFS (μs)
IEEE 802.11-1997 (FHSS)	50	28	128
IEEE 802.11-1997 (DSSS)	20	10	50
IEEE 802.11b	20	10	50
IEEE 802.11a	9	16	34
IEEE 802.11g	9 or 20	10	28 or 50
IEEE 802.11n (2.4 GHz)	9 or 20	10	28 or 50
IEEE 802.11n (5 GHz)	9	16	34

Table 2.1: DCF parameters for different IEEE 802.11 standards.

The NAV may be thought of as a counter that counts down to zero at a uniform rate. When the counter is zero, it is an indication that the medium is idle; when nonzero, the medium busy. The medium shall be determined to be busy when the STA is transmitting. In IEEE 802.11, the NAV represents the number of microseconds the transmitting STA intends to hold the medium busy (maximum of 32,767 microseconds). Wireless stations are often battery powered, so in order to conserve power the stations may enter a power-saving mode. A station decrements its NAV counter until it reaches zero, at which time it is awakened to sense the medium again.

2.3.2 Binary Exponential Backoff

The backoff procedure is the collision-avoidance mechanism of DCF. If nodes in a network transmit at the same time, the signals overlap and the resulting signal will be undecodable by the receiver or receivers. We call this situation a "collision". Collisions cause data to be lost, and lost information needs to be retransmitted. Imagine a situation where a station is transmitting on the channel. Let's assume the scheme used by the stations to send as soon as they see the channel idle. Using this scheme, all stations begin to transmit once the currently-transmitting station has finished its transmission. This will lead to a collision if there is more than one station with information to send. For this reason, according to the standard, each station should wait for a random period before sending, once the channel is sensed idle for a DIFS after a busy period. DIFS here is a fixed amount of time (see Table 2.1 for more details). This procedure is illustrated in Figure 2.2.

The time after each busy period is divided into discrete time-slots. The slot time is also a fixed duration, although it can be different for different protocol

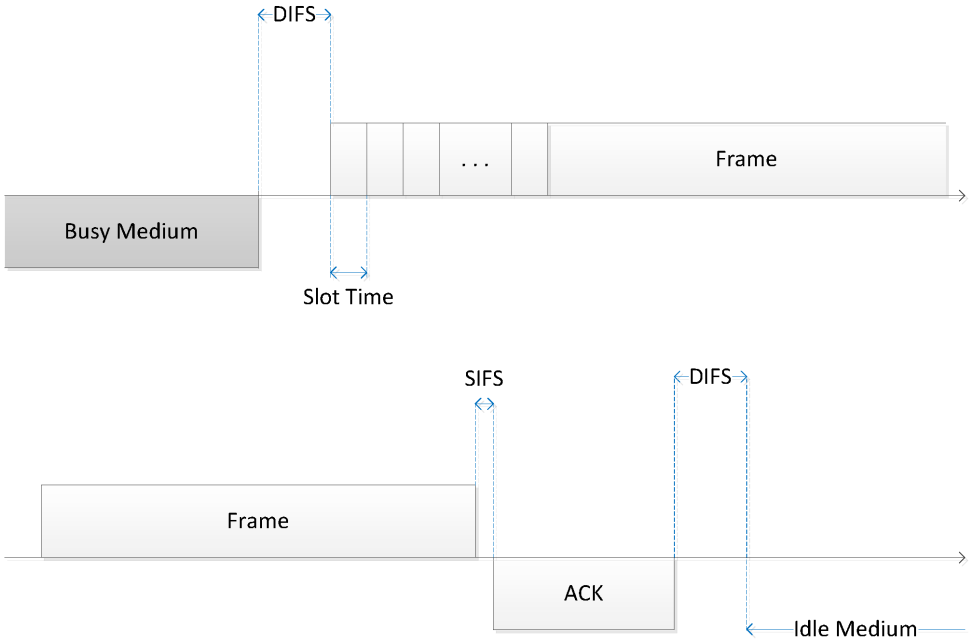


Figure 2.2: Distributed coordination function: random backoff (top) and acknowledgements (bottom) are illustrated in this figure.

amendments (see Table 2.1). The initial backoff procedure can be described as follows:

1. Choose a random integer w in the range $[0, CW_{min})$ which is the number of time-slots the node will wait.
2. Start a count-down decrementing w for each time-slot.
3. If the channel becomes busy, freeze the count-down until it is idle again for at least a DIFS period.
4. When w reaches zero, transmit the frame.

Contention Window (CW) is the size of the window from which we choose the random number. It is originally set to CW_{min}. Despite random backoff being performed by all stations, the chances of collision are not eliminated. A collision is an indication of a crowded channel. In such case, stations repeat the backoff procedure, but this time in a window of $[0, 2CW_{min})$.

This trend continues and each time a collision occurs, CW is doubled until a maximum of CW_{max}. There is also a limit on the number of retries, after which the frame will be considered lost and the failure reported back to the host machine. We will describe this in more detail in Section 2.3.4. On a successful attempt, the value of CW is reset to its original value CW_{min}.

The scheme described here reduces the likelihood of a collision. We call this scheme "binary exponential backoff" (BEB). The default values for CW_{min} and CW_{max} are different for different IEEE 802.11 protocols. For IEEE 802.11b for example, CW_{min} is 31 and CW_{max} is 1023. For IEEE 802.11a and IEEE 802.11g, CW_{min} is 15 and CW_{max} is 1023.

2.3.3 Acknowledgements (ACKs)

Now that we know how stations coordinate their transmissions to avoid collisions, one important question remains unanswered. Since stations have a half-duplex radio system (as we mentioned earlier), then how can they know whether their transmissions have been successful if they cannot listen to the channel for colliding signals? The answer is also part of the DCF. For this very reason, each data frame from a sender should be followed by another frame called an acknowledge frame (ACK) from the receiver.[6] After a station sends a frame, it starts listening to the channel almost immediately. The receiver should normally send an ACK, after a period of time called SIFS following the completion of the received frame. SIFS is smaller than DIFS so other stations will not view the channel as free before the ACK is sent. If the sending station does not receive an ACK in a timely manner, it takes this as an indication that the receiver has not received the frame due to a collision or channel noise, and it starts the exponential backoff.

2.3.4 RTS/CTS Mechanism

RTS/CTS is an additional method to provide virtual carrier sensing in CSMA/CA to overcome the problem described later in Section 2.6.2. A node wishing to send data initiates the process by sending a Request to Send (RTS) frame. The destination node replies with a Clear To Send (CTS) frame. Any other node receiving the RTS or CTS frame should refrain from sending data for a given time (solving the hidden node problem). The amount of time the

[6] Not to be confused with ACKs in the transport layer.

node should wait before trying to get access to the medium is included in both the RTS and the CTS frame. Since RTS/CTS are actual frames and introduce overhead to the medium, they are typically not used unless the packet size exceeds a certain threshold. If the packet size that the node wants to transmit is larger than the threshold, the RTS/CTS handshake is triggered; otherwise, the data frame is sent immediately.

This mechanism reduces the chances of collision when stations cannot all hear one another (see Section 2.6.2), and the need for retransmission is diminished. We mentioned in Section 2.3.2 that there is a limit on the number of retries for a single frame. This number is different when the RTS/CTS mechanism is used. The retry limit for transmissions that do not use this mechanism is called the "short retry limit", and its default value is 7. The limit for when the RTS/CTS mechanism is used is called the "long retry limit", the default value of which is 4.

The effectiveness of the RTS/CTS mechanism is debated in the literature. In [17] they investigate the effectiveness of this mechanism in ad-hoc networks and show that in some situations it cannot function well. [18, 19] show that the RTS/CTS mechanism is not as effective as expected in different network scenarios and in some cases it is even worse than CSMA.

2.4 Enhanced Distribution Channel Access

IEEE 802.11e [12] introduces a new coordination function: the Hybrid Coordination Function (HCF). Figure 2.3, taken from IEEE 802.11 standard specification [20], depicts where HCF resides in the IEEE 802.11 architecture. Within the HCF, there are two methods of channel access, similar to those defined in the legacy IEEE 802.11 MAC: HCF Controlled Channel Access (HCCA) and Enhanced Distributed Channel Access (EDCA). Both EDCA and HCCA define Traffic Categories (TCs). For example, emails could be assigned to a low-priority class (category), and Voice over Wireless LAN (VoWLAN) could be assigned to a high priority class.

With EDCA, high-priority traffic has a higher chance of being sent than low-priority traffic, which is achieved through the choice of contention parameters. The protocol used is called TCMA, which is a variation of CSMA/CA using a

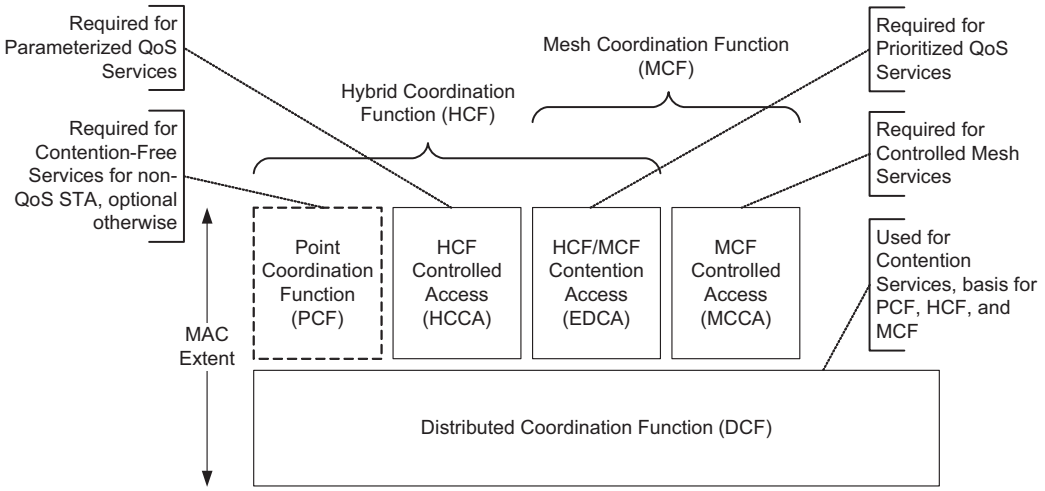


Figure 2.3: IEEE 802.11 MAC architecture; taken from [20]

shorter arbitration inter-frame space (AIFS)[7] for higher priority frames. The exact values depend on the physical layer that is used to transmit the data. It is defined by the formula $AIFSN[AC] \times \sigma + SIFS$, where the AIFSN depends on the AC (see Table 2.3) and σ denotes slot time.

EDCA also provides contention-free access to the channel through Transmit Opportunity (TXOP). A TXOP is a period of time when a station has the right to initiate frame exchange sequences onto the wireless medium without contention. A TXOP is defined by a starting time and a maximum duration. The TXOP is either obtained by the STA by successfully contending for the channel or assigned by the hybrid coordinator. The use of TXOPs reduces the problem of low rate stations gaining an inordinate amount of channel time in the legacy IEEE 802.11 DCF MAC [21]. A TXOP time interval of 0 means it is limited to a single MAC service data unit (MSDU) or MAC management protocol data unit (MMPDU).

Priority levels in EDCA are called Access Categories (ACs). The CW_{min} and CW_{max} can be set according to the traffic expected in each AC, as shown in Table 2.2. Their effective values are calculated from the aCWmin and aCWmax values that are defined for each physical layer supported by IEEE 802.11e. For a typical of aCWmin=15 and aCWmax=1023, the resulting values are as as shown in Table 2.3.

[7] AIFS replaces DIFS in EDCF. Its value varies for each access category (AC).

AC	CW_{min}	CW_{max}
Background (AC_BK)	$aCWmin$	$aCWmax$
Best Effort (AC_BE)	$aCWmin$	$aCWmax$
Video (AC_VI)	$\frac{aCWmin+1}{2} - 1$	$aCWmin$
Voice (AC_VO)	$\frac{aCWmin+1}{4} - 1$	$\frac{aCWmin+1}{2} - 1$

Table 2.2: Calculation of Contention Window boundaries

AC	CW_{min}	CW_{max}	**AIFSN**	**Max TXOP**
Background (AC_BK)	15	1023	7	0
Best Effort (AC_BE)	15	1023	3	0
Video (AC_VI)	7	15	2	3.008ms
Voice (AC_VO)	3	7	2	1.504ms

Table 2.3: Default EDCA Parameters for each AC in IEEE 802.11e

EDCA access parameters (AIFS, CWmin/CWmax, and TXOP) are set in each STA in advance. They are also broadcast by the AP through beacon frames. Upon reception of a beacon, the STA updates its EDCA parameters and uses them to transmit frames. Thus the AP can control the EDCA parameters of STAs that it is handling. There are also scenarios in which the data needs to be protected from other data of the same class. Admission Control in EDCA addresses these type of problems. The AP publishes the available bandwidth in beacons. Clients can check the available bandwidth before adding more traffic.

2.5 Additional Features

Each amendment to the IEEE 802.11 standard introduces new methods to improve its performance and usability. In this section we introduce some of the additional features in IEEE 802.11g [11], IEEE 802.11e [12], and IEEE 802.11n[13].

2.5.1 Block Acknowledgements (BA)

PHY level data rate improvements do not increase user level throughput beyond a point because of IEEE 802.11 protocol overheads, such as inter-frame

spacing, PHY level headers, and ACKs. Frame aggregation is a process of packing multiple MSDUs or MPDUs together to reduce the overheads and average them over multiple frames, thereby increasing the user level data rate. Two types of aggregation are defined: (i) Aggregation of MAC service data units (MSDUs) at the top of the MAC (referred to as MSDU aggregation or A-MSDU), and (ii) Aggregation of MAC protocol data units (MPDUs) at the bottom of the MAC (referred to as MPDU aggregation or A-MPDU).

A-MPDU aggregation requires the use of block acknowledgements, or Block ACKs. This feature was first introduced in IEEE 802.11e [12] as an optional feature, and later was enhanced and made mandatory in IEEE 802.11n[13]. Using this feature, instead of sending an ACK for every single MPDU, a single ACK is sent for a group of MPDUs. A block acknowledgement (BA) can support up to 1024 data units (fragments). There are two types of Block Acks: immediate, and delayed. With Immediate Block ACK, the BA is required after the receipt of Block ACK Request (BAR) whereas with Delayed Block ACK, the BAR itself is acknowledged (by the recipient) with a simple ACK frame and the BA is sent later on separately which is also acknowledged (by the originator).

2.5.2 No ACK

One of the QoS features of IEEE 802.11e is the QoSNoAck service class. In QoS mode, service class for frames to send can have two values: QosAck and QosNoAck. Frames with QosNoAck are not acknowledged. This avoids retransmission of highly time-critical data. But note that a station cannot be sure if a QosNoAck frame was successfully transmitted.

2.5.3 Direct Link Setup

Direct link setup (DLS) allows direct station-to-station frame transfer within a BSS. This is designed for consumer use, where station-to-station transfer is more commonly used. DLS provides an avenue for devices to perform consumer station-to-station functions while simultaneously connected to an enterprise WLAN.

2.6 Challenges for the MAC

In this section we describe the challenges that the MAC layer in a WLAN faces. The IEEE 802.11 protocol overcomes these challenges to some extent, but some can still be kept in mind when implementing new algorithms on top of IEEE 802.11.

2.6.1 RF Link Quality

On a wired Ethernet, it is reasonable to transmit a frame and assume that the destination receives it correctly if there is no collision. Moreover, collisions can be detected explicitly. Radio links are different, especially when frequencies used are unlicensed ISM bands. The devices must assume that noise and interference will exist and work around these problems. The designers of IEEE 802.11 considered ways to work around the radiation from microwave ovens and other RF sources. In addition to the noise, multipath fading may also lead to situations in which frames cannot be transmitted because the node moves into a dead spot.

To overcome this problem, the IEEE 802.11 DCF uses acknowledgements and if any part of the transfer fails, the frame is considered lost. Acknowledgements were explained in Section 2.3.3. This transaction is necessary since the link is not reliable and the frame might be lost during transmission.

Radio link quality also influences the speed at which a network can operate. Good quality signals can carry data at a higher rate. Available data rates for different IEEE 802.11 protocols are listed in Table 2.4. Signal quality degrades with range, which means that the data transmission speed of an IEEE 802.11 station depends on its location relative to the AP and the noise relative to signal strength. Stations must implement a method for determining when to change the data rate in response to changing conditions. Furthermore, the complete collection of stations in a network must manage transmissions at multiple speeds.

There are many rate adaptation algorithms introduced in the literature. An old example is ARF [23], which decreases the rate on two consecutive transmission failures, and increases it after receiving 10 ACKs without failure. It also has a probing packet after each increase, the failure of which causes an instant step-back. This is meant to prevent false-positives. Another example

802.11 protocol	Freq. (GHz)	Bandwidth (MHz)	Data rate per stream (Mb/s)	Modulation
—	2.4	20	1, 2	DSSS, FHSS
a	5	20	6, 9, 12, 18, 24, 36, 48, 54	OFDM
	3.7			
b	2.4	20	1, 2, 5.5, 11	DSSS
g	2.4	20	6, 9, 12, 18, 24, 36, 48, 54	OFDM, DSSS
n	2.5/5	20	7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2	OFDM
		40	15, 30, 45, 60, 90, 120, 135, 150	

Table 2.4: Available rates and modulation schemes for different IEEE 802.11 protocols. For detailed information regarding modulation, see [22].

is SampleRate [24], which chooses the rate with the smallest average packet transmission time, including loss recovery time. A good feature of this algorithm is that it chooses a random rate every 10th packet to measure its performance, therefore it never gets stuck in a low rate. A variation of SampleRate is called Minstrel [25], which is used in Linux drivers. It gathers statistics from transmitted frames and calculates a probability of success for each frame. It also spends a particular percentage of frames "looking around" (i.e. randomly trying other rates) to gather statistics. This percentage is usually 10%. The distribution of lookaround frames is also randomized somewhat to avoid any potential "strobing" of lookaround between similar nodes.

2.6.2 The Hidden Node Problem

In Ethernet networks, stations depend on the reception of transmissions to perform the carrier sensing functions. Wires in the physical medium distribute signals to network nodes. Wireless networks have fuzzier boundaries and nodes might not be all interconnected as they would be in a wired network. Take a look at Figure 2.4. In this figure, node 2 can communicate with both nodes 1 and 3, but something prevents nodes 1 and 3 from communicating directly. This could be due to an obstacle or simply because they are far away from each other. We call node 3 a "hidden" node from node 1's perspective. As they do not hear each other on the channel, CSMA/CA may fail to hear transmissions and therefore it is highly probable that their transmissions will overlap each

other, and node 2 cannot make sense of any of the transmissions.

Collisions resulting from hidden nodes may be hard to detect in wireless networks because, as mentioned before, wireless transceivers are generally half-duplex. To prevent collisions, IEEE 802.11 allows stations to use RTS/CTS signals to clear out an area. This mechanism was explained in Section 2.3.4.

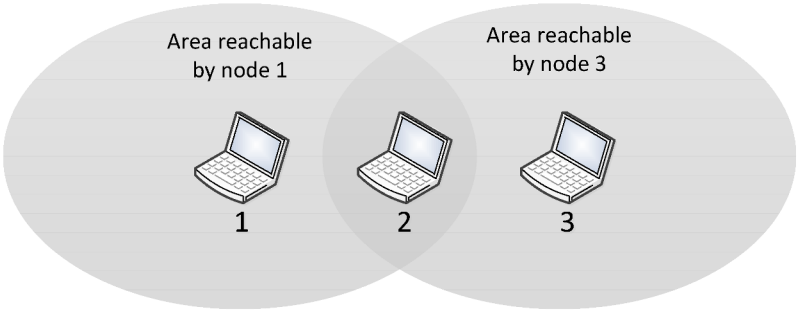


Figure 2.4: The hidden node problem: Node 1 cannot directly communicate with node 3, so it will not be able contend properly with it.

2.6.3 The Exposed Node Problem

From the same characteristic of a wireless network that is described in Section 2.6.2, another problem might arise, and that is the situation depicted in Figure 2.5. In this figure, nodes 2 and 3 are in each other's range, but they are not in the same network, or at least they do not intend to send to each other. Node 2 is transmitting to node 1, while node 3 intends to transmit to node 4. However, once node 2 begins transmission, node 3 should wait because it hears something on the channel. But the truth is that if node 3 also starts transmitting, there will be no collision since it is not in the range of node 1, and similarly node 2 is not in the range of node 4. Therefore both transmissions could be received correctly by their corresponding destination nodes even if sent at the same time, but this scenario would not arise because nodes 2 and 3 are "exposed" to each other's signals.

There has been interest in mitigating this effect in the literature. In [26] they study both the hidden node and the exposed node problems and provide schemes to remove or mitigate the effects. The exposed node problem is specially critical in ad-hoc networks and there are works like [27] that specifically focus on this kind of network.

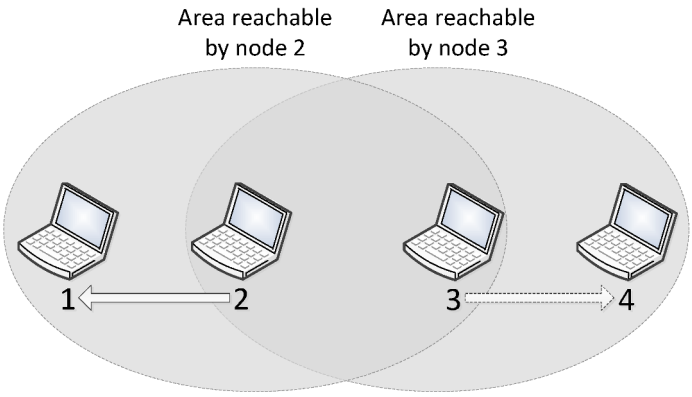


Figure 2.5: The exposed node problem: node 2 is transmitting to node 1 and node 3 wants to transmit to node 4.

The IEEE 802.11 RTS/CTS mechanism could help solve the exposed node problem as well, only if the nodes are synchronized and packet sizes and data rates are the same for both transmitting nodes. When a node hears an RTS from a neighboring node, but not the corresponding CTS, that node can deduce that it is an exposed node and is permitted to transmit to other neighbors. If the nodes are not synchronized (or if their packet sizes or the data rates are different) the problem may occur that the sender will not hear the CTS or the ACK during the transmission of data of the second sender.

2.7 Under the Hood

In this section we will describe in detail a few things related to the standard. Knowing these details will help the reader have a better understanding of the technicality behind some implementations discussed in later chapters.

2.7.1 Timing in IEEE 802.11 MAC

Since we are using the timing mechanism in IEEE 802.11 broadly in our implementations, we will briefly describe it here. As already discussed, time between transmissions is slotted in the IEEE 802.11 standard. In order for stations to operate correctly, the time slots for all stations should be aligned. This can be tricky since each station uses different hardware and has a different distance from other stations. So we need a way of synchronizing the time stations see.

Timing Synchronization Function (TSF) is specified in the IEEE 802.11 wireless local area network (WLAN) standard to fulfill timing synchronization among users. The TSF keeps the timers for all stations in the same Basic Service Set (BSS) synchronized. All stations maintain a local TSF timer. Each station maintains a TSF timer with modulus 2^{64} counting in increments of microseconds.

On a commercial level, industry vendors assume the IEEE 802.11 TSF's synchronization to be within 25 microseconds. Timing synchronization is achieved by stations periodically exchanging timing information through beacon frames. Each station in the network adopts the received timestamp if it is later than the station's own TSF timer. We introduced beacon frames in Section 2.1.

All stations in the IBSS adopt a common value, normally noted as *aBeaconPeriod*, that defines the length of beacon intervals or periods. This value, established by the station that initiates the IBSS, defines a series of Target Beacon Transmission Times (TBTTs) exactly aBeaconPeriod time units apart. Time zero is defined to be a TBTT.

2.7.2 Frame Format and Types

Throughout this thesis we will be discussing the implementation of different tweaks and algorithms on IEEE 802.11 hardware and sometimes we will need to identify frames and/or modify their information. For this reason we will discuss IEEE 802.11 frame formats here briefly. A MAC frame contains all information needed to successfully land a piece of information on its destination within the WLAN. Figure 2.6 shows the structure of an IEEE 802.11 frame. The second diagram shows the three main sections of a frame.

The first diagram in Figure 2.6 corresponds to the PHY header. While we are more interested in the MAC layer, there are some useful points regarding the PHY level headers which we will discuss briefly here. The first component of the PHY header is called the preamble. To put it simply, preamble is a signal used to synchronize transmission timing. In other words it is used as a series of transmission criteria to be understood as "somebody is about to transmit data". There are two different kinds of preamble used in IEEE 802.11, the details of which are not in the scope of this thesis. Legacy IEEE 802.11 uses a 128-bit SYNC in the preamble; this is called a *long preamble*. IEEE 802.11b

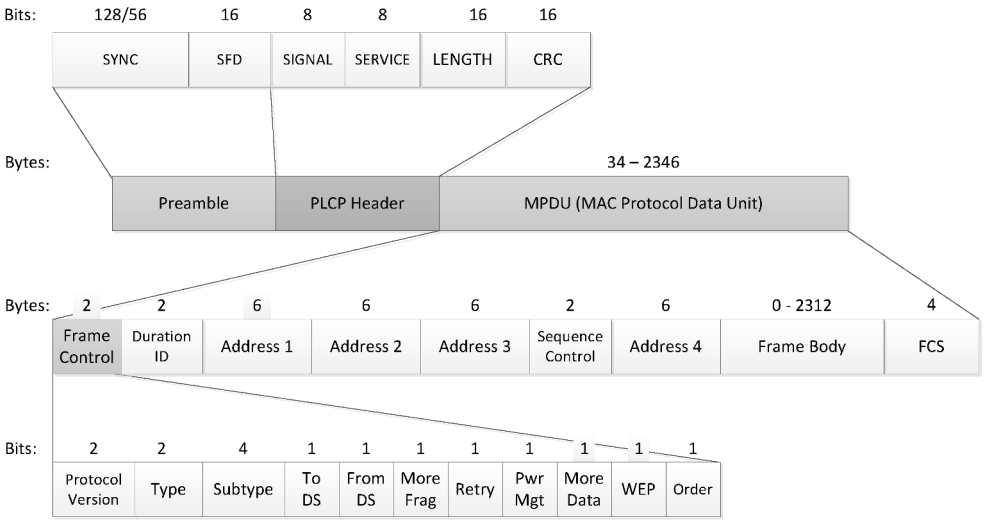


Figure 2.6: IEEE 802.11 frame format. The length of SYNC is different for short and long preambles.

and IEEE 802.11g also use an optional *short preamble* which has a 56-bit SYNC.[8]

The PLCP header has some extra information about the frame that is about to be transmitted. For our applications, the most important field in the PLCP header is the LENGTH field, which determines the size of the frame. The value is the number of microseconds required to transmit the remainder of the frame, excluding the PHY header. For more detailed information regarding PLCP header and the preamble, see [22] (section 18.2.3).

The part of the frame that is covered in the MAC layer is the MAC Protocol Data Unit (MPDU). The structure of MPDU is shown in the middle diagram in Figure 2.6. The first part of the MPDU is Frame Control. It contains information about the type and some characteristics of the frame. The information contained in this field is shown in the bottom diagram. The most important fields in Frame Control are the type and subtype fields. These together determine the nature of the frame. For example, beacon frames have a management type, and a beacon subtype.

The next two bytes are reserved for the Duration ID field. This field can take one of three forms: Duration, Contention-Free Period (CFP), and Association

[8]For example, IEEE 802.11g is required to support both short and long preambles.

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	Destination	Source	BSSID	N/A
0	1	Destination	BSSID	Source	N/A
1	0	BSSID	Source	Destination	N/A
1	1	Receiver	Transmitter	Destination	Source

Table 2.5: Meanings of different IEEE 802.11 address fields based on DS flags

ID (AID).

An IEEE 802.11 frame can have up to four address fields. Each field can carry a MAC address and what they refer to depends on whether the frame is being forwarded through the DS. Table 2.5 shows the address each field holds in different scenarios.

The Sequence Control field is a two-byte section used for identifying message order as well as eliminating duplicate frames. The first 4 bits are used for the fragmentation number and the last 12 bits are the sequence number.

An optional two-byte Quality of Service control field was added with 802.11e.

The Frame Body field is the main information that is delivered. Its size varies from 0 to 2304 bytes plus any overhead from security encapsulation and contains information from higher layers.

The Frame Check Sequence (FCS) is the last 4 bytes in the standard IEEE 802.11 frame. Often referred to as the Cyclic Redundancy Check (CRC), it allows for an integrity check of retrieved frames. As frames are about to be sent the FCS is calculated and appended. When a station receives a frame it can calculate the FCS of the frame and compare it to the one received. If they match, it is assumed that the frame was not distorted during transmission. For detailed information regarding the FCS field and the CRC algorithm used, see [22] (section 7.1.3.7).

2.8 Summary

IEEE 802.11 is the dominant protocol for WLANs. It allows stations to communicate either in infrastructure mode, where all stations communicate through a central node called the AP, or ad-hoc mode, where stations transmit information to their neighbors as well as forwarding messages for other

stations.

For the MAC layer, the IEEE 802.11 standard uses a method called DCF which is based on CSMA/CA. The DCF ensures that different nodes in a network can contend for channel access and all have fair chances of accessing the channel and transmitting data. The collision avoidance mechanism in DCF employs a binary exponential backoff mechanism. Stations originally back off a random amount of time before sending, in order to avoid collisions. They double their backoff windows each time their transmissions fail.

IEEE 802.11e enhances the mechanism by introducing HCF. With HCF, stations can contend in different ways according to the priority of their traffic, adding Quality of Service (QoS) capabilities to the protocol.

Wireless MAC protocols face several challenges, the most basic one of which is link quality. It is not as straightforward in a wireless medium to transmit frames correctly as it is in a wired network. The second problem is called the "Hidden Node Problem", caused when two stations communicate with a third party but they are not in each other's range. In this case, they would not pause their backoff and transmission and collisions would occur. The third problem is called the "Exposed Node Problem", which happens when a station can practically transmit a frame to a second station, but its transmissions keep being delayed because of received signals from a third party, even though simultaneous transmission would not cause any harm (because receivers are not hearing the other sender's signals).

CHAPTER 3

Related Work

In this chapter we will discuss state of the art in the area we focus on in this thesis.

3.1 Introduction

The main focus of this thesis is to practically implement a form of traffic shaping that is used to prevent selfish behavior in an IEEE 802.11 network. This has different aspects, which include misbehavior detection and prevention, protocol manipulation, and traffic shaping. In this chapter we discuss what has been done before in each of these areas, and how the work in this thesis relates to them. The focus will mainly be on IEEE 802.11, but other technologies will also be looked at briefly where relevant.

3.2 Fairness and Compliance

Fairness is an important subject when it comes to resource allocation. In wireless networks and IEEE 802.11 in particular, fairness has received special attention, an old example of which is the proposed framework by Nandagopal et al. [28] for modeling fairness models. As mentioned in said paper, achieving fairness in the wireless medium is inherently more complicated than the wired version because of its location-dependent contention, inaccurate state estimates, and decentralized control. Besides, there is always a trade-off between channel utilization and fairness [29], although some work aims to alleviate this

by designing new variations of the DCF [30]. Other examples of works that focus on resource scheduling in wireless networks include the paper by Jamshaid et al. [31] where they use a rate-based centralized scheduling techniques to enforce max-min fairness (see Section 3.2.1). In this section we first discuss different notions of fairness that have been introduced in the literature, and how they are used. Then we move on to defining how fairness compares to standard compliance.

3.2.1 Notions of Fairness

Fairness can have different, sometimes conflicting, definitions depending on how it is measured and where it is applied. The DCF is designed to give equal chances to all stations to transmit frames, and it is proven to provide this degree of fairness in the long term [32]. When we hear fairness in computer networks, the first thing that comes to mind is *throughput fairness*, which means stations being able to send the same amount of data in a given time. This is a notion of fairness used in works such as that of Lee et al. [33] where they adjust contention window parameters to achieve fair bandwidth allocation. But this notion is not always desirable. For instance, with multi-rate networks, stations employing lower bitrates take more time on the medium than stations with higher bitrates.

If we take time as the resource that needs to be shared between stations, we come to the notion of airtime fairness [34][35]. The reason airtime is commonly considered is that it is proven that in a multi-rate DCF network, the network saturation throughput corresponds to the saturation throughput of the station with the lowest bitrate [21]. In other words, such a station will drag the performance of other stations down. That is why airtime is an important measure. Joshi et al. [36] propose a protocol called time-fair CSMA (TFCSMA) in which the contention window of each station is adjusted in a decentralized manner to ensure airtime fairness. Tinnirello et al. take a different approach, and use TXOP [37] to achieve temporal fairness. These methods rely on the fact that airtime fairness does not imply throughput fairness: stations with higher bitrates can send more data in the same period of time.

A popular measure for fairness is Jain's fairness index [38]. If x_i is the amount

of the desired resource allocated to i, then Jain's index is defined as

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

which equals to 1 when all stations get the same amount of the desired resource, and less than 1 if they don't. $\frac{1}{n}$ is the worst value this index can get (e.g. when one station gets all resources). It is worth noting that Jain's fairness index does not take the absolute level of achieved throughput into account. For instance, all stations getting 0.1 Mb/s is "better" than one getting 10 Mb/s and the rest getting 1 Mb/s as far as this index is concerned. In other words, it does not indicate efficient resource utilization.

Another notion of fairness that has applications far beyond IEEE 802.11 networks is *max-min fairness* [39]. A resource allocation is max-min fair if and only if it is feasible (i.e. possible in the network) and an attempt to increase the allocation of any participant necessarily results in the decrease in the allocation of some other participant with an equal or smaller allocation. The resource in question is usually airtime or bitrate in wireless networks. Examples of works on max-min fairness are [40] and [41] propose methods to achieve max-min fairness in a decentralized way in terrestrial wireless ad-hoc networks and underwater sensor networks respectively. This fairness measure does not have the limitation of Jain's fairness index, and ensures optimal utilization of resources.

As far as multi-rate networks are concerned, there is another notion of fairness, introduced by Kelly [42], and studied in more recent works (e.g. [43][44]). In this notion, a utility function $u_i(x_i)$ is defined for each station, where x_i is the rate at which the station i sends. The goal is then to maximize the sum of all utilities, given network constraints, for instance:

$$\max_{\sum x_i = c} \sum u_i(x_i)$$

A special case of utility fairness is called proportional fairness, which is studied for IEEE 802.11 [45, 46] as well as earlier technologies such as Aloha [47]. The (marginal) utility here is proportional to the bitrate the station already has. The notion is based on the idea that the satisfaction of a station that is sending at 11 Mb/s with additional 1 Mb/s will be less than that of a station sending

at 1 Mb/s, because the bitrate of the latter would be doubled. In practice, $u(x_i) = \log(x_i)$ is often used.

Proportional fairness can also be described in terms of cost: the utility function is inversely proportional to the energy cost of sending at the given bitrate. A newer example application of proportional fairness is described by Haiyang et al. [48], where they use a higher layer scheduler in conjunction with IEEE 802.11's rate control mechanisms to provide proportional fairness for DASH (Dynamic Adaptive Streaming over HTTP).

Not all mentions of fairness correspond to sophisticated utility fairness notions. What works, such as the paper by Jiang et al. [26] and the original policing algorithm [6], often consider fairness is throughput fairness, as described in the first paragraph of this section.

3.2.2 Short-term Fairness

Although much of the work in fairness focuses on long-term fairness, short-term fairness is also of great importance, specially when it comes to real-time applications; and this importance has been long recognized in the literature [49]. IEEE 802.11 medium access mechanism is designed to provide fair access to the channel in the long term. However, short-term fairness is not guaranteed by the DCF [50] [51] (though Vlachou et al. [52] show that IEEE 802.11 provides more short-term fairness than the similar access protocol IEEE 1901 when the number of stations is less than 15).

Many methods and schemes have been introduced to achieve short-term fairness in IEEE 802.11 networks. An example is iBEB by Almotairi et al. [53], which employs an inverse exponential backoff mechanism, meaning that it starts from the maximum backoff window and reduces the contention window by half on each collision. They prove that their method improves short-term fairness. An obvious downside of this method is low channel utilization, and that is due to the large initial contention window of 511.

Kim et a. [54] introduce another method called renewal access protocol (RAP), which is a simplified version of the DCF with fixed contention window and an a priori backoff distribution. They achieve high short-term fairness and optimal channel utilization through the choice of this distribution.

Another method for achieving fairness especially in the short term is opportunistic scheduling [55], which is used mainly for cellular networks. It was first introduced by Knopp and Humblet [56]. In a pure opportunistic approach, the scheduler always chooses the user in the best channel condition to use the resources. Opportunistic scheduling depends on the multiuser diversity due to random wireless channel impairments such as fading and multipath.

3.2.3 Fairness and Service Differentiation

Different stations in a wireless network have different needs. Take the difference between a voice call, and a file download as an example. Delay is less tolerable in a voice conversation than it is for a file download, while frame loss is better tolerated [57]. Since IEEE 802.11e, service differentiation was introduced to the protocol, making it possible for different types of traffic to be treated according to their specific requirements. Fairness in this case can be more complicated to define and achieve, as it is no longer the question of throughput or airtime fairness among different nodes, but also among different traffic classes [58][59].

Bottigliengo et al. propose a method [60] to guarantee fairness in the presence of service differentiation. Their method combines contention window adaptation and transmission opportunity (TXOP) to guarantee quality of service for flows of the same class. Qiang et al. use distributed scheduling at both the AP and stations to achieve quality of service for each traffic class while maintaining fairness among different classes [61]. AS-MAC [58] is another method that simultaneously provides the absolute priority and *successful transmission time fairness*

We will further discuss the impact of service differentiation on fairness in Chapter 5 as we see the implications it has for the policing algorithm.

3.2.4 Standard Compliance

The original IEEE 802.11 standard is designed to give equal chances to all stations to access the channel. However, the protocol is a decentralized one, and stations can choose not to comply. Giri et al. describe some examples of non-compliance and their effects [62], and Guang et al. introduce a model [63] to compute the saturation throughput under both normal case and the

case with the existence of selfish nodes. The fairness that is achieved by the standard is only possible if all stations implement it correctly, so standard compliance is closely tied with airtime and throughput fairness: if one station maliciously takes up more channel time, it takes the opportunity away from compliant stations (see Section 3.3 for experimental work on this subject).

In this thesis the focus is on standard compliance rather than fairness. The idea is to enforce a compliant behavior while trusting the standard to provide fair channel access. This can sometimes come at the price of lower channel utilization but better throughput for compliant stations, so linking indirectly to the trade-offs in max-min and utility fairness.

3.3 Node Misbehavior

The main contribution of this work, as described in Chapter 5, is the implementation and experimental analysis of a misbehavior detection system, and a method for determining the correct behavior. What we mean here by misbehavior is non-compliance with standard, as described in the previous section. Node misbehavior has received much attention in the literature, mainly because of the decentralized nature of the IEEE 802.11 protocol, which exposes it to selfish behavior. It can happen when users selfishly manipulate their channel access parameters to gain a performance advantage. This can severely degrade the performance of the users that abide by the standard [64, 65]. Availability of open-source drivers such as MadWifi [66], and even open-source firmware for some Wi-Fi adapters [67] makes this a real, rather than theoretical, issue.

Research has proven that said issue is not limited to malicious users. Giordano et al. [68] evaluate maximum achievable goodput using theory, simulations, and experimental analysis. They find differences between different IEEE 802.11b wireless adapters, and deviations from expected goodputs. In [69], they take a step further and measure access parameters of different wireless card using a probe cards, and find the source of performance differences between them to be in the MAC layer. Bianchi et al. [70, 71, 72] run experiments with different wireless adapters and measure contention parameters used by each of them and show that off-the-shelf adapters have deferring behavior, which sometimes deviates from standard, and that the unfairness in

a heterogeneous wireless network is due to the protocol implementation on different cards rather than environmental factors.

3.3.1 Types of Misbehavior

Although the focus of this work is on cheating via contention parameters, there are various other kinds of attacks on IEEE 802.11 that are worthy of consideration. In this section we go through some of these attacks and the literature around them. Some of these attacks aim to degrade the performance of the whole network without any gain for the misbehaving station, while others seek benefit at the cost of the performance of others. The focus of this thesis is on the latter, while also ignoring security and privacy attacks, which are themselves subjects of broad research, which is not directly relevant to this work.

3.3.1.1 Cheating on Contention Parameters

These series of attacks are very popular among researchers. The main reason is that they are easiest to implement, and often most effective. In these attacks, the associated station refuses to use the EDCA parameter settings broadcast through the beacon frame (or the standard contention parameters for older versions of the IEEE 802.11). Hoang et al. propose a model to estimate the impact of this kind of misbehavior [73]. Many misbehavior detection algorithms focus on these attacks (e.g. [64] [6][74]). We will discuss them in further detail later. Possible attacks of this nature include: choosing a smaller CW_{min}, choosing $CW_{max} = CW_{min}$, using a large TXOP time, and using a small AIFS value, which can be equal to the previous attack if chosen small enough. Example attacks presented in Chapter 5 are of this type.

3.3.1.2 Carrier Sensing Attack

Carrier sensing is a foundation of IEEE 802.11. Stations need to sense a clear channel before they can begin to transmit. In reality, there is always something on the channel, including noise, and clear channel is a relative definition. Stations in an IEEE 802.11 network use a clear channel assessment (CCA) threshold for received signal power below which they consider the channel idle. A misbehaving station can increase its CCA threshold and thus

get an advantage over other stations [75]. Pelechrinis et al. employ frequency hopping to counteract this type of misbehavior [76].

3.3.1.3 Jamming Attacks

A malicious node can intentionally cause collisions either to degrade the network's performance or to gain advantage over other stations using so called "selective jamming" [77] where the attacker targets specific frame types. Both motives have been studied in the literature [78], and possible methods have been devised to counteract jamming. For example, DeBruhl et al. [79] and Konorski [80] use game theoretical methods to find optimal strategies of both the jammer, and the victim (sender), for cases where the motive of the attackers is to gain benefit. The focus of this work, is on the same motive.

An IEEE 802.11 sender needs an ACK to confirm successful reception of its frame. An attacker can simply send signals to jam the receiver's ACK messages, and thus degrade the performance of the sender by causing it to double its contention window [81] [64]. A similar approach would be to jam CTS frames after hearing RTS [64]. We can even extend this to jamming association requests, which prevents any station from connecting to the AP, this leaving the attacker alone in the channel.

3.3.1.4 Upper Layer Attacks

Some attacks in wireless networks target a layer higher than the one they are performed. Jamming attacks, for instance, are performed in the physical layer, but they use information from the MAC layer to jam the right signals. This idea can be taken further, and to higher layers. For example, an attacker can jam TCP ACKs and greatly affect the TCP's congestion control and thus the sender's performance [64].[1] An attack on the application layer is also suggested in the aforementioned work by Raya et al.. Video streaming software uses adaptive compression/encoding to conform to different bandwidths. If an attacker causes a compression fallback, it can benefit from the extra bandwidth.

[1] For this attack to be effective, all MAC-level retries of a TCP ACK must be jammed, which could be expensive for the attacker in terms of power consumption.

3.3.1.5 Packet Forging Attacks

Previously we described the jamming of TCP ACKs. An attacker can be smart, and rather than jamming, it can forge legitimate-looking packets and gain benefit. A similar attack would be to forge TCP ACKs when the packet has not been received correctly [82]. These false ACKs have a negative impact on network's operation, and are hard to detect.

This kind of attack can also target the MAC layer. For example, flooding RTS/CTS handshakes on the channel causes other stations to wait longer than they should, which gives the attacker more chances to access the channel [82]. In the same work, Rachedi et al. also introduce another variation of these attacks, which is sending forged beacons. The idea is to send beacons with the wrong TSF time in an Ad-Hoc network to desynchronize stations.

3.3.1.6 Denial of Service (DoS)

Denial of service attacks are a common threat in computer networking, and go way beyond IEEE 802.11. The victim of DoS attacks are usually servers with multiple clients, and the goal of the attacker is to reduce service quality by keeping the server busy. Different kinds of DoS attacks include *bandwidth depletion attacks* [83][84][85] which flood the server with control, data packets, or requests, and *resource depletion attacks* which misuse protocol-related packets to confuse communication parties [86]. Different methods have been proposed to counteract DoS attacks, including swarm networks [87], TCP probing [88], and probabilistic approach [89], which are all described and evaluated by Robinson et al. [90].

A form of DoS attacks in IEEE 802.11 is achieved by modifying the duration field (see 2.7.2) of a sent packet to a large value. This causes other stations to wait for the whole period, which gives the attacker more time to transmit [91]. Partial deafness [92] is another form of DoS attacks, which is designed to degrade the network performance in Ad-Hoc networks. In this attack, the attacker refuses to send ACKs for received packets for enough times to cause a rate fallback. Then, combined with the performance anomaly of IEEE 802.11b networks [21], the network throughput is degraded. This of course depends on the rate control algorithm used.

3.3.2 Protection Provided by the Standard

IEEE 802.11 provides an encryption mechanism called Wi-Fi Protected Access version 2 (WPA2) which, beside protection against eavesdropping, can disable some of the attacker types we mentioned before. Packet forging and jamming attacks, especially those targeting higher layers, depend on the ability to read and analyze packet contents. With such encryption in place, those attackers will not be able to operate, or their efficacy will be impaired. However, attacks on the MAC and physical layers, such as contention parameter tweaking and carrier sensing attacks will still be effective, and that is one reason they are the most popular among research community.

3.3.3 Misbehavior Detection

In order to prevent or take action against node misbehavior, we first need to be able to identify misbeaving nodes. Although protocols are flexible, they have set rules for how the medium can be accessed (e.g. [12]). So, theoretically we can have an estimate of how each station should perform given specific set of QoS-related and channel parameters. The literature is no stranger to misbehavior detection. Many of the papers described in the previous section propose methods to detect the specific kinds of misbehavior they cover. In this section the focus will be on the most prominent and easy-to-implement form of misbehavior, which is the modification of contention parameters and backoff behavior.

Many of the proposed schemes [93, 94, 95, 96] use statistical methods to identify misbehaving stations, and mainly focus on back-off distribution as the only random component of CSMA/CA. Szott et al. for instance [96] use a chi-square test on backoff values to detect non-compliant behavior. In [97] the receiver dictates backoff value to the sender, and then counts backoff values and compares them with the expected value with a threshold. The reason for the threshold is that channel conditions have an effect on the performance of each station, and a fixed value cannot be assumed. They also propose some modifications to the IEEE 802.11 protocol which allows a form of message passing between stations to facilitate detection. This could be a downside, because in a real network, modifying all stations is not usually desirable or possible.

There are two main issues with the latter approach; (i) it requires modifications to the standard, namely changing from a decentralized binary exponential backoff (BEB) to a dictated backoff mechanism. (ii) there are edge-cases, such as hidden terminals, which cause misdiagnosis. They propose to address the latter using RTS/CTS: the receiver only considers a slot to be busy when it has overheard an RTS/CTS frame. While this eliminates the original problem, it adds another, which is the possibility for senders to pose as hidden terminals and gain advantage.

DOMINO [64] and the method proposed by Serrano et al. [98] on the other hand do not involve a distributed implementation. They reside in or near the access point and run statistical tests to measure the amount of deviation of stations' behavior from the standard. DOMINO's target statistics include AIFS, backoff time, NAV, and retransmissions. The advantage of this algorithm is the great insight it has over the stations' behavior. However, DOMINO does have some blind spots. An example of such a blind spot is again the existence of hidden terminals, which causes false positives for misbehavior. Although they propose a method to alleviate this problem, some extent of the effect is inevitable given the statistics they use.

Radosavac et al. [99] use a *sequential probability ratio test (SPRT)* [100] to statistically detect cheating on backoff behavior. They recognize their weakness in the case when the sender and the receiver collude to gain benefit. Another example of methods of this sort is the one proposed by Toledo et al. [101], which uses statistical analysis of idle slots to identify non-compliant station. All said methods (including DOMINO) use an "observer" station to measure packet interarrivals. For this reason, they suffer from the effects of interference and poor channel conditions, as they need to gather as much information as they can, and this gets harder as channel conditions degrade. The method introduced by Yanxia et al. [102] can be implemented so as to avoid this problem for AP-based networks.

Dangerfield et al.'s policing algorithm [6] also uses statistics for misbehavior detection, but the only metric it uses is throughput. While throughput equality is one type of fairness, it is not well suited for detecting misbehavior, as different stations may have different throughput values depending on packet size, traffic access categories, etc. In this work we extend this algorithm to use

attempt rate, which is more closely related to the station's behavior. Regardless of the statistic used, the simplistic approach of the algorithm makes it agnostic to the type of misbehavior, and robust to the effect of hidden nodes, and channel conditions. The reason is that it only focuses on the final and concrete information available on the access point rather than possibly lossy information gathered through observing the channel. However, the same simplistic approach opens the algorithm to cases such as the existence of QoS, where detection can be impacted by the indeterministic mix of contention parameters.

Cardenas et al. introduce a method [103] to ensure backoff randomness in ad hoc networks, and detect those deviating from it. The benefit here is that channel conditions don't affect the detection mechanism. However, like some of the methods mentioned above, it needs message passing between stations. Another shortcoming of their method compared to methods such as DOMINO is that it requires at least one honest party in every transmission pair (either the sender or the receiver) while DOMINO does not need that, as the observer does not participate in the network. Another method is proposed by Djahel et al. [104] for MANETs, which solves this problem of colluding stations by having each station monitor all transmissions in its vicinity in case of a degradation in its own throughput.

3.3.4 Traffic Shaping

Traffic shaping is the practice of regulating network data transfer to assure a certain level of performance [105] or quality of service (QoS) [106]. This can be through delaying packets or even dropping them as described in Chapter 5. Works on this subject often attack the problem of increased residential user-to-user traffic [107] [108], and they commonly use traffic shaping against bulk flows such as peer-to-peer file-sharing networks [106].

Traffic shaping has also received attention in regard to IEEE 802.11. Vollero et al. [109] use frame dropping to manage traffic to ensure quality of service for multimedia communications. The same principle in different works [110, 111] to mitigate the effect of the performance anomaly of IEEE 802.11b networks [21]. This technique is also one of the building blocks of the policing algorithm [6] introduced, where it is used as a countermeasure against stations that do

not comply with the IEEE 802.11 standard. Frames from greedy nodes are dropped just enough to reduce its throughput to that of a compliant station.

3.4 Policing

While much work has been done on misbehavior detection in IEEE 802.11, only a limited number of proposals address counteracting greedy actions, and these suffer from significant practical drawbacks. For instance, [103] requires a reputation management system to prevent MAC layer misbehavior. Shi et al. introduce a modified version of the DCF where the back-off value is dictated by the receiver [112], but it does not go any further than this "prevention" scheme that in fact cannot be enforced in practice. Kyasanur et al.'s method goes a step further [97] with a penalty system, but a cross-layer interaction is assumed in it to enable higher layers to restrict the traffic that non-compliant clients generate. The penalty system involves assigning larger backoff values to the receiver. Obviously this does not necessarily lead to compliance, but can serve as a warning to the non-compliant station, and if it continues its non-compliant behavior, the receiver may stop responding to its RTS frames.

We previously mentioned Djahel et al.'s method [104] as a detection algorithm in MANETs. They also propose a countermeasure. Once a non-compliant station is detected in a network, a special warning message is broadcast by the receiver which, beside warning the non-compliant station, notifies all neighbors to monitor its behavior. If the misbehavior persists beyond that, all neighbors begin punishing the station by not responding to its RTS messages, and refusing to relay any control message sent by the misbehaving node.

Both of the above methods employ a penalty system in which non-compliant stations are removed from the network completely. This form of black-and-white punishment suffers from a major drawback: false alarms can have a severe effect on network performance. In the latter method they try to alleviate this by introducing an accuracy factor, which is an small arbitrary value removed from the interarrival time.

Giri et al. use a collective approach to disincentivize non-compliance [62]. In their scheme, all compliant stations estimate the level of misbehavior of the selfish node, and try to replicate that misbehavior as a reaction response. This method effectively disarms misbehaving nodes. However, aggressive non-

compliance could choke the network should all stations employ it. PAS [74] takes a similar approach. However, they prove their algorithm is stable (by conducting a Lyapunov stability analysis) and does not suffer the instability caused by constantly increased punishments.

The policing algorithm [6] uses a different method against non-compliance. In their method, it is assumed that we know the throughput of a compliant station at all times. Using that value, they calculate the level of misbehavior of all nodes, and employ a penalty system in which the punishment is proportional to the severity of misbehavior. The penalty is inflicted by not acknowledging received frames with a probability proportional to the aggressiveness of the selfish node. An obvious shortcoming in their method is that we usually don't know the compliant throughput, as it depends highly on channel conditions. The other drawback is that it depends on ACKs, and with new IEEE 802.11 features such as No ACK, this method could be less effective.

The work in this thesis refines the scheme from [6], looking at implementation challenges, adapting the scheme to be resistant to a wider range of misbehaviors, and showing how to estimate compliant throughput.

3.5 Experimental Evaluation

Experimental evaluation of wireless networks has become an essential part of wireless research. The main reason is that, while mathematical models that are based on standards [1, 113, 114] provide excellent tools to predict the behavior of a network, they do not fully represent real networks. The reason is that there are many factors that alter the expected behavior of a network, many of which are often ignored or are just too complex to incorporate in a model. The more obvious examples are issues like neighboring wireless networks that interfere with each other, and reflection of electromagnetic waves from nearby walls. Although some models such as [2] try to incorporate some characteristics of real networks, but many factors are inevitably missing even from these models.

The concept of implementing ideas on radio hardware is not new. Software-defined radio (SDR) has been used to implement many theoretical ideas and processes in recent years (e.g. [16]). SDR is a radio communication system where components that have been typically implemented in hardware are instead implemented by means of software, which means total customizability

of the radio system. The problem with SDR devices, however, is that those with operating frequency ranges covering Wi-Fi frequencies are often very expensive. The use of commercial off-the-shelf wireless adapters in experimental research is relatively newer, but it is getting more and more popular. This is partly due to widely-available open-source device software [115]. Examples of existing work include the work by Cardenas et al. [93], where they evaluate previously proposed schemes including DOMINO [64] and SPRT-based schemes [103] by using experimental results from off-the-shelf devices to confirm the correctness of their theoretical analysis. There is a wide range of other experimental work, from rate adaptation schemes [3, 4], to the paper by Raman et al. [116] where they introduce and experimentally evaluate a new MAC protocol suitable for long-distance multi-hop links. Another example is the method proposed by Palletta et al. [117] to measure the saturation throughput of commercial IEEE 802.11 access points. They then use it on common devices such as the Ericsson A11d and Cisco Air 1200.

The flexibility of programmable Wi-Fi hardware has furthered their applications to those beyond IEEE 802.11. There are works that have taken advantage of the carrier sensing in those devices for applications such as cognitive radio [118] or detecting non-Wi-Fi sources of interference [119].

Although real implementation of algorithms is rewarding in terms of proof of concept, it is not completely straightforward. It is widely understood in the research community that debugging an implementation can take many hours. Most of the available tools and approaches e.g. [120, 121] focus on protocol level monitoring, and less on other interactions. At the end of this thesis we show a tool for testing and debugging IEEE 802.11 hardware (Appendix A) developed and used during this research, which shows an ongoing divergence between the IEEE 802.11 standard and operational hardware. Read said appendix for further details.

3.6 Discussion and Conclusion

This work is built upon the policing algorithm introduced by Dangerfield et al. [6], which is a misbehavior detection and penalty scheme. Many detection algorithms have been introduced in the literature. There are three reasons for this choice to build on in the present thesis. First, this method does not

modify the protocol and does not need any modification for stations, or any message-passing. Second, it focuses on the detection of the performance of the non-compliant station, rather than detecting the type of misbehavior, which makes it flexible towards different types of misbehavior. Third, it introduces a penalty system which is proportional the the greediness of misbehaving stations.

The policing algorithm assumes we know the compliant throughput at all times, which is not the case. In this thesis we introduce a method to estimate the compliant throughput in any given channel condition. We also refine the algorithm and explains its possible problems and shortcomings.

The other contribution of this work is experimental evaluation of the policing algorithm. The implementation of methods and algorithms on real hardware has recently received much attention. By implementing the policing algorithm on commercial hardware we can show that it works in practice, and we can observe how real channel conditions affect its performance.

CHAPTER 4

Broadcom Chipset Programming

In this chapter we discuss how the Broadcom BCM43xx wireless adapters work and how we implement and evaluate algorithms on them. We also give a brief description of the architecture.

4.1 Introduction

The use of commodity hardware to implement custom behaviors on IEEE 802.11 has recently attracted much interest. Work in this area falls into three main categories.

The first category of work focuses on the study of the behavior of the wireless adapters themselves through experimentation. These works do not modify the behavior of wireless cards and minor modifications are made only if needed to extract results. Examples include [72], [122] and [123]. For instance in [72] they study the back-off behavior of common off-the-shelf wireless cards and investigate the differences in their compliance with the standard.

Works in the second category evaluate their proposed models and algorithms by implementing them on wireless adapters. Examples are [93], [4], [3], and

[116]. In [3] and [4] they introduce new rate adaptation[1] algorithms and back their proposals up with experimental results; and in [116] they introduce and experimentally evaluate a new MAC protocol suitable for long-distance multi-hop links.

The third category of work implements new mechanisms and alternative functionality on wireless adapters. Examples are [118] where they implement a spectrum "sensor" on Atheros cards, and works from the European project FLAVIA [124], [125, 126] where they propose a visually reprogrammable wireless adapter design and implement it on Broadcom BCM43xx chipsets.

Experimental validation is important because many models and algorithms are designed to improve the current IEEE 802.11 networks, and experimental validation would be a strong indication that they are useful. Different wireless adapters have been used for this purpose in the literature. Examples are Broadcom BCM43xx adapters, used in [125] and [126], Atheros chipsets, used in [4], [3] and [118], and Digital RoamAbout adapters, used in [123].

Software-defined radio (SDR) has also become popular recently (see also Section 3.5). SDR is a radio communication system where components that have been typically implemented in hardware (e.g. filters, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system. Works like [127] use SDR as their experimental tool. However, using off-the-shelf devices is still relatively cheaper and the results reflect an everyday network rather than perfectly crafted wireless components.

The availability of open-source wireless drivers is a great asset for our work. Hardware manufacturers have always been reluctant to distribute the source-code for the drivers of their products. This was mainly due to intellectual property protection, and to protect their hardware from misuse. However, with the growth of open-source software and new attention for drivers, some manufacturers began to release open-source drivers. As far as wireless adapters are concerned, Qualcomm Atheros chipsets and Broadcom wireless adapters are common examples.

[1] Rate adaptation involves changing a station's physical transmission rate adaptively in varying conditions to achieve high throughput.

The original open-source Atheros driver, namely MadWifi [66], consists of an open-source driver on top of a proprietary Hardware Abstraction Layer (HAL). The latter is responsible for direct communication with the hardware. Using the MadWifi driver you can modify queuing parameters of IEEE 802.11e with simple shell commands on-the-fly, and you can change other aspects of the IEEE 802.11 behavior by modifying the driver itself. MadWifi was succeeded by ath5k which combined MadWifi and HAL to make a Free and Open-Source Software (FOSS) driver for Linux. The release of the source code for HAL in 2008 was a driver for the development of ath5k. Qualcomm Atheros later released ath9k, a fully open-source driver for their new IEEE 802.11n chips.[2]

Despite all the flexibility open-source drivers provide, these drivers lack the low-level control that we need on the protocol. While MadWifi and ath9k provide a reasonable control on the protocol stack, when it comes to the hardware part, there is still little control. For example, the binary exponential backoff procedure is hardwired and although some parameters can be modified, the procedure itself cannot be altered. We use Broadcom BCM4318 wireless adapters for our experiments. The choice of Broadcom over Atheros cards is due to the higher flexibility and control. Similar to Atheros cards, there is an open-source driver, called the `b43` driver, available for Broadcom cards. However, unlike Atheros, there is also an open-source firmware available. A programmable firmware means control at the lowest level of the protocol. This enables us to access features like carrier sensing, ACK generation, backoff behavior, etc. The firmware we are using is a reverse-engineered firmware called OpenFWWF [67] which is developed as a part of FLAVIA [124], and used in their works.

In this chapter we will discuss the wireless card programming for Broadcom BCM43xx chipsets. We explain how different software and hardware components work together, and give an idea of how implementations are done by altering the behavior of these components. The descriptions in this chapter are based on the 4.xx series of the BCM43xx chipset by Broadcom.

[2]There is also a more recent driver named ath10k for Qualcomm Atheros QCA988x family of chips, which support IEEE 802.11ac.

4.2 Broadcom BCM4318 Chipset Basics

4.2.1 Processing

At the very bottom of the BCM43xx architecture is the "Backplane". The role of the Backplane is to interconnect between different cores available on the chipset. It functions as a switch that can choose the active "core" to enable or disable functionality. This is known as "switching to the core". Each core can be viewed as a light-weight CPU that handles a very specific task. When a core is switched to, it can access its parameters (registers) as well as the common registers on the Backplane.

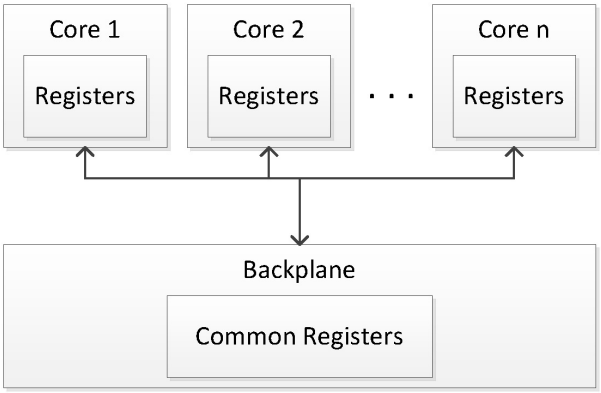


Figure 4.1: The structure of the Backplane

On top of the Backplane, there are several cores, depending on the specific card. This can be schematically drawn as Figure 4.1. As depicted in the figure, each core has its own set of specific registers, depending on its functionality. The number and nature of the cores on the chipset depends on the core revision and the type of hardware. Two important examples of cores are "802.11" and "NAND Flash". In this chapter we are particularly interested in the former.

4.2.2 IEEE 802.11 Functionality

The IEEE 802.11 functionality of the device consists of the following components.

802.11 Core is the main core on the BCM43xx chipset and is at the heart of IEEE 802.11 functionality: it is where the firmware microcode is actually executed. We can break the features of this core into the following

components: 802.11 MAC, 802.11 PHY, and 802.11 Radio. These components consist of registers specific to their respective tasks. These registers make up the register set of the 802.11 core, along with additional control registers. The purpose of the control registers is mainly to avoid concurrent access to shared resources that may cause malfunction.

In modern processing units, calling external functionalities such as radio transmission is done through interrupts. In 802.11 core, these calls are handled through registers. For instance, suspending the MAC [3] is performed by writing a specific value in the "MAC Enabled" register. To track the status of a previously accessed external function, there are additional registers called "Interrupt Status" registers. There are two such registers; one is called the "MAC Interrupt Status" and holds information about MAC function calls; the other one is called "DMA/PIO Interrupt Status". Reading and writing packet information is performed using DMA to the memory of the host machine, and this can and should be done in parallel with the normal MAC behavior because of its time-consuming nature. The DMA/PIO status register provides the firmware with the means to know whether the required information is ready or whether the received packets are sent to the host.

Template RAM or FIFO RAM is 32KB in size and it can be accessed by the 802.11 core from the firmware code. For each transmission, the frame is first written to Template RAM, and then the chip is instructed to transmit the required portion of the data through control registers. As stated in [128], Template RAM space is structured as shown in Table 4.1. Address ranges are shown in hexadecimal format.

Writing to Template RAM is another example of the poll-based interrupt handling in 802.11 core. To write to Template RAM, we first choose the target offset using the "TX Template Pointer" register. Then we specify the start and end of the source data (i.e. data that is going to be copied to the target location) through the two designated "TX Template Data" registers, which initiates the copy process. At this stage we keep checking for the bit 1 of the pointer register to be cleared, which indicates that the data is copied.

[3] Suspending MAC is temporarily disabling MAC functionalities of the card with the intent of removing the device or for power saving purposes.

Range	Usage
0000 - 05FF	Template for ACK (0000), beacons and probe response (0068, 0268, 0468)
0600 - 0EFF	TX FIFO number 0, size is 0x0900
0F00 - 1BFF	TX FIFO number 1, size is 0x0D00
1C00 - 25FF	TX FIFO number 2, size is 0x0A00
2600 - 2DFF	TX FIFO number 3, size is 0x0800
2E00 - 3AFF	TX FIFO number 4, size is 0x0D00
3B00 - 3BFF	TX FIFO number 5, size is 0x0100
3C00 - 43FF	unknown but modifiable
4400 - 7FFF	unknown but fixed, not modifiable

Table 4.1: The memory structure of Template RAM

Object Memory is a virtual address space used to access different kinds of information on the chip. It is accessed by selecting an object type, and a 16-bit address space. Object types include Microcode Memory, Microcode registers, Internal Hardware Registers and Shared Memory. The latter is what we use for bookkeeping and communication with the host when implementing algorithms. It is a piece of memory that is mapped to a portion of the host RAM that is assigned to the firmware when we load the driver. It is 4 KB in size and our observations show that a large portion of it is unused by the normal MAC behavior. We will go back to shared memory later on in this chapter.

Firmware is the microcode run on the core, and what we use to implement our algorithms. We introduce it as a main component of the MAC functionality because it defines what the adapter does and how it behaves. It can be easily written onto the NAND Flash when we load the driver.

4.3 Related Work

The Broadcom platform has previously been used by [125] [126] as part of FLAVIA [124]. In these publications, a new mechanism is introduced for wireless cards using which one can visually change the MAC behavior and modify the state machine to run their designed algorithms. In [125] they have implemented schemes such as Piggybacked ACKs, Pseudo TDMA, and Multi-channel access using their framework.

4.4 Programming for Broadcom IEEE 802.11 Adapters

Broadcom BCM43xx programming has two major components: the driver, which is executed by the host CPU. It is characterized by fast processing, large amount of available memory, and communication capabilities with the card; and the firmware, which is executed on the 802.11 core in the chipset. It is characterized by low latency, low-level control, and high processing time. A typical 802.11 core on the Broadcom bcm4318 has a 8MHz clock. This is definitely slower than the host machine's CPU, which is typically faster than 1GHz. Most of the applications need modification on both the firmware and the driver as we will discuss later.

4.4.1 Firmware Programming

The firmware is a piece of code written on the card's NAND Flash and executed by the 802.11 core. This code is responsible for the MAC/PHY/Radio functionality of the card and is what defines how the card works. It can be overwritten by software on the host machine. Firmware modification is necessary whenever we need low-level control over the MAC behavior, and whenever changing functionality requires communication with the card through the shared memory, which can be slow. More detail about this will be provided when when we describe implementations in later chapters.

To program for Broadcom chipsets we need a set of tools called `b43-tools`, which are available from the project's git-hub[4]. A list of these tools and their functions is shown in Table 4.2. Our main focus is on `b43-asm` to assemble the firmware and `b43-fwdump` for debugging.

In order to modify the firmware, we should extract the microcode, disassemble it, find the piece of code responsible for our specific task, and then modify as we want it to work. This can be a difficult task. Extracted firmware is a series of instructions, and finding relations and figuring out the information flow in the code is time-consuming. Fortunately this has been done before. As mentioned before, we use the open-source firmware OpenFWWF as a basis for our firmware modifications. It is a firmware assembly code where information flow has been reverse engineered and the code is "beautified" (see Table 4.2

[4] https://github.com/mbuesch/b43-tools

Tool	Usage
b43-fwdump	Dumping shared memory, register values, and firmware microcode (see Section 4.4.1.1)
b43-beautifier	Replacing constant expressions in raw disassembled firmware code with human-readable names
b43-asm	Creating binary firmware from human-readable assembly code
b43-dasm	Disassemble extracted firmware microcode into human-readable assembly code
b43-fwcutter	Extracting firmware from binary Broadcom 43xx driver files
ssb_sprom	Convenient modification of the Broadcom Sonics Silicon Backplane SPROM

Table 4.2: Tools for b43 firmware development and debugging

for some details) and documented in the project home page [67]. It is worth noting that OpenFWWF is NOT the original firmware included with the b43 driver and it has fewer capabilities. However, the version we use has all the functionalities required for IEEE 802.11b.

The assembly of the firmware microcode is also documented on the sip-solutions website [128] and thus all building blocks required are available to program for Broadcom cards. Different revisions of the 802.11 core have slightly different instruction sets. Table 4.3 is a list of instructions available in core revision 5 which is the core revision of the cards we use. The instruction set is quite simple and lacks operations such as floating point operators and even multiplication (the latter is available in core revision 11). However, it provides the essentials for implementing IEEE 802.11 on the card.

As an example of the assembly code, what follows is a piece of code used in device initialization to erase the shared memory. The code is extracted from OpenFWWF and some descriptive comments are added. The first line puts the address of the last word of the shared memory in an offset (base) register[5]. It is then followed by an inverse loop that clears the words one by one. This code snippet is just to have an idea of how we program for Broadcom chipsets.

```
  mov  SHM_LAST_WORD, SPR_BASE5          // index = MEMORY_END
erase_shm:                               // Loop entry
  orx  7, 8, 0x000, 0x000, [0x00,off5]   // Set the word to zero
  sub  SPR_BASE5, 0x001, SPR_BASE5       // index = index - 1
  jges SPR_BASE5, 0x000, erase_shm       // While index > 0
```

[5]Offset registers are used for relative addressing of the shared memory, and as an alternative to using constant hard-coded addresses.

Mnemonic	Operands	Description	Operation
		Arithmetic and logic instructions	
add	A,B,rD	Add	$rD = A + B$
add.	A,B,rD	Add, set Carry	$rD = A + B$ Save Carry
addc	A,B,rD	Add with Carry	$rD = A + B + Carry$
addc.	A,B,rD	Add with Carry, set Carry	$rD = A + B + C$ Save Carry
sub	A,B,rD	Subtract	$rD = A - B$
sub.	A,B,rD	Sub, set Carry	$rD = A - B$ Save Carry
subc	A,B,rD	Sub with Carry	$rD = A - B - Carry$
subc.	A,B,rD	Sub with Carry, set Carry	$rD = A - B - C$ Save Carry
		Branch instructions	
jand	A,B,l	Jump if binary AND	$if(A\&B)PC = l$
jnand	A,B,l	Jump if not binary AND	$if(!(A\&B))PC = l$
js	A,B,l	Jump if all bits set	$if((A\&B) = A)PC = l$
jns	A,B,l	Jump if not all bits set	$if((A\&B) \neq A)PC = l$
je	A,B,l	Jump if equal	$if(A = B)PC = l$
jne	A,B,l	Jump if not equal	$if(A \neq B)PC = l$
jls	A,B,l	Jump if less (signed)	$if(A < B)PC = l$
jges	A,B,l	Jump if greater or equal (sign.)	$if(A \geq B)PC = l$
jgs	A,B,l	Jump if greater (signed)	$if(A > B)PC = l$
jles	A,B,l	Jump if less or equal (signed)	$if(A \leq B)PC = l$
jl	A,B,l	Jump if less	$if(A < B)PC = l$
jge	A,B,l	Jump if greater or equal	$if(A \geq B)PC = l$
jg	A,B,l	Jump if greater	$if(A > B)PC = l$
jle	A,B,l	Jump if less or equal	$if(A \leq B)PC = l$
jdn	A,B,l	Jump if diff is < 0, no carry	$if(nc(A - B) < 0)PC = l$
jdpz	A,B,l	Jump if diff is ≥ 0, no carry	$if(nc(A - B) \geq 0)PC = l$
jdp	A,B,l	Jump if diff is > 0, no carry	$if(nc(A - B) > 0)PC = l$
jdnz	A,B,l	Jump if diff is ≤ 0, no carry	$if(nc(A - B) \leq 0)PC = l$
call	lrX,l	Store PC, call function	$lrX = PC; PC = l$
calls	l	Store PC, call function	$PC- > stack; PC = l$
ret	lrX,lrY	Store PC, ret from func	$lrX = PC; PC = lrY$
rets		ret from function	$stack- > PC$
jzx	M,S,A,B,l	Jump if zero after shift + mask	
jnzx	M,S,A,B,l	Jump if nonzero after shift+msk	
jext	E,A,B,l	Jump if External Condition true	$if(E)PC = l$
jnext	E,A,B,l	Jump if External Condition false	$if(!E)PC = l$
		Bitwise instructions	
sra	A,B,rD	Arithmetic rightshift	$rD = A >> B$ fillup sign
or	A,B,rD	Bitwise OR	$rD = A\|B$
and	A,B,rD	Bitwise AND	$rD = A\&B$
xor	A,B,rD	Bitwise XOR	$rD = A^B$
sr	A,B,rD	Rightshift	$rD = A >> B$
sl	A,B,rD	Leftshift	$rD = A << B$
srx	M,S,A,B,rD	Shift right over two registers	
rl	A,B,rD	Rotate left	$rD = lrot(A, Bbits)$
rr	A,B,rD	Rotate right	$rD = rrot(A, Bbits)$
nand	A,B,rD	Clear bits (notmask+and)	$rD = A\&(\ B)$
orx	M,S,A,B,rD	OR with shift and select	
		Other instructions	
nap	none	Sleep until event	

Table 4.3: The instruction set of 802.11 core revision 5

Something worth noting in the previous code snippet is the use of a complicated instruction like `orx` [6] for setting a memory word to zero. One can argue that this can also be carried out using a simple `mov` instruction, but sometimes instructions like this are used in the firmware not only to perform the desired task, but also to control the flow of the core (clearing the pipeline, etc.). In cases like this, using the trivial approach might lead to unexpected behavior that has roots in the architecture of the device.

When modifications are made, we take the steps below to install the new firmware. If you are using OpenFWWF, you can simply execute `make && make install` in the source directory instead of steps 2 and 3, as these procedures are incorporated in the Makefile that is included with the OpenFWWF source.

1. Disable the driver by executing: `modprobe -r b43`.
2. "Compile" the firmware assembly using b43-asm.
3. Copy the assembled firmware files to the firmware directory in Linux (typically /lib/firmware).
4. Enable the driver again, leaving QoS disabled (as it is not supported by OpenFWWF) by executing: `modprobe b43 qos=0`.

4.4.1.1 Debugging Firmware Code

Testing and debugging are essential parts of software development in general, and they are even more crucial when it comes to firmware programming. Modifications we make to the firmware code might cause malfunction or even kernel panics if we affect the integrity of the communicated content between the firmware and the driver. However, debugging firmware code is more complicated than debugging a piece of software on a PC.

Using modern debuggers, one can halt a running program at any point in code, and observe the values of different symbols and variables. This helps programmers find bugs in their code. They can also choose to execute their

[6] The `orx` instruction takes four arguments. The first two arguments define two adjacent bit ranges on the word and the bits in each of the two ranges are then ANDed with the corresponding bits of the following two arguments respectively.

code line by line to see where an exception occurs or an unexpected behavior initiates.

Unfortunately, these sophisticated features cannot be implemented for code running on a chipset like BCM4318. The most important reasons for this are: (i) firmware code runs on a single-threaded machine with memory and processing power limitations; (ii) there is no external control on the flow of the program, and even if this was somehow implemented into the firmware itself, the resulting firmware code would be too large for the allocated space; and (iii) there is no hardware support and capacity for running instructions remotely. If there was such possibility, all firmware code could be handled by a debugging program running on the host machine, and instructions could be handed in to the 802.11 core one by one.

Although we cannot implement a sophisticated debugger for BCM43xx-based chipsets, debugging them is not completely impossible. There is a very useful tool in `b43-tools`, namely `b43-fwdump`, that facilitates debugging to some extent. The tool can be used to dump shared memory, register values, and even firmware assembly. In order to use this tool, a kernel module called DebugFS must be installed, and the corresponding virtual file system must be mounted. What exactly this module does is out of the scope of this chapter. We will discuss DebugFS in more detail in Chapter A.

If `b43-fwdump` is executed without any arguments, it simply lists the values of all general-purpose registers, offset registers, and the program counter (PC). You still cannot pause the code at an arbitrary point, but you can practically have the values of your variables at any time instant, as they are all stored in general-purpose registers. Adding `-s` to the tool's command line also prints out the shared memory. Listing 4.1 is a partial example of `b43-fwdump` output.

Tracing registers and the shared memory can help us understand the firmware operation. As we will discuss in coming chapters, a large portion of the 4 KB shared memory is unused, which makes it convenient for storing traces, counters, or any other sort of information from within the firmware, which can help us debug the code.

Listing 4.1: Example `b43-fwdump` output. This is just the first page of the trace. The complete output is lengthy as it prints out the whole shared memory. Each word in the architecture is 2 bytes long, which is why hexadecimal representations of the data are grouped into 16-bit (4-digit hexadecimal) numbers in this output.

```
--- B43 microcode state dump ---
PC: 030  PSM-COND: 0000
Link registers:
lr0: 039F  lr1: 03EC  lr2: 0059  lr3: 041D
Offset registers:
off0: 041E  off1: 0504  off2: 0370  off3: 0391
off4: 010E  off5: 011E  off6: F15F
General purpose registers:
r00: 000E  r01: 0000  r02: 0000  r03: 000F
r04: 03FF  r05: 001F  r06: 0007  r07: 0004
r08: FFFF  r09: B382  r10: 0000  r11: 0001
r12: 0000  r13: 0000  r14: 0035  r15: 0000
r16: 001F  r17: 4021  r18: 0035  r19: 0035
r20: 0105  r21: 0000  r22: 0000  r23: 0000
r24: 0000  r25: 0000  r26: 0000  r27: 0000
r28: 0000  r29: 00A3  r30: 4789  r31: EB2C
r32: 0000  r33: 75A0  r34: 0802  r35: 013F
r36: 9C6A  r37: 0000  r38: 0008  r39: 0000
r40: 0000  r41: 0000  r42: 0000  r43: 0001
r44: 0001  r45: 0000  r46: 0000  r47: 0000
r48: 0000  r49: 0000  r50: 0000  r51: 0000
r52: 0000  r53: 0000  r54: 0000  r55: 0000
r56: 0000  r57: 0000  r58: 0000  r59: 0000
r60: 0000  r61: 0000  r62: 001F  r63: 03FF
Code:
<No binary supplied. See --binary option>
Shared memory:
0x0000:  9A01 7008 1A75 0A7C 0000 0000 C000 0A00   ..p..u.|........
0x0010:  1400 0000 8000 0900 4700 4700 8301 6400   ........G.G...d.
0x0020:  3009 C0FC 0000 0000 0000 0000 0000 0103   0...............
0x0030:  0001 0000 0200 0200 0100 0400 1E00 0000   ................
0x0040:  0200 0000 0300 0200 0E00 4700 0028 0000   ..........G..(..
0x0050:  0700 0200 C0FC 7E05 167F 7F7F 0A00 8300   ......~.........
0x0060:  0001 0000 0300 0000 0000 0000 0000 1200   ................
0x0070:  7F7F 7F7F 0100 0000 0000 0000 2C15 2258   ............,."X
0x0080:  0600 1027 0100 7391 1212 1010 5401 0702   ...'..s.....T...
0x0090:  0000 0000 6009 FA00 090D 0A08 0D01 0000   ....`...........
0x00A0:  0E00 0000 0000 3F01 FFFF 0000 0000 0000   ......?.........
0x00B0:  0000 0000 0000 0000 0000 0000 B403 B403   ................
0x00C0:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x00D0:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x00E0:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x00F0:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x0100:  0000 0000 26F9 0000 0000 0000 9072 0000   ....&........r..
0x0110:  7E08 0000 0000 0000 0000 0000 92B0 33F9   ~.............3.
0x0120:  0000 0000 0000 9B05 5300 0000 0000 0000   ........S.......
0x0130:  0000 0000 2CEF 0000 0000 0000 0000 0000   ....,...........
0x0140:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x0150:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x0160:  4252 434D 5F54 4553 545F 5353 4944 0000   BRCM_TEST_SSID..
```

4.4.2 Driver Programming

The driver is a piece of software that is installed as a module on the host machine. The role of a driver is to communicate with the wireless card and bring information from the card to the host machine and vice versa. In a Wi-Fi card, this information is basically sent and received frames. The communication takes place through the *shared memory*. The term is generically used for any piece of memory that provides communication between applications. When used in the context of device drivers, it is also called Memory-mapped I/O (MMIO)[7]. Figure 4.2 illustrates shared memory mechanism. As depicted in the figure, a piece of host memory is mapped to an internal memory on the chipset designated for this purpose. MMIO is a channel for the host and the device to communicate. Each time either of the two counterparts of the shared memory is modified, the modification is also applied on the other side, although this does NOT happen instantly. In fact, this communication can often be lengthy and sometimes unreliable if communication time exceeds the validity period of the communicated information. Hence we cannot solely rely on the driver for our implementations, especially if we have actions that need to be performed within microseconds, such as ACK generation.

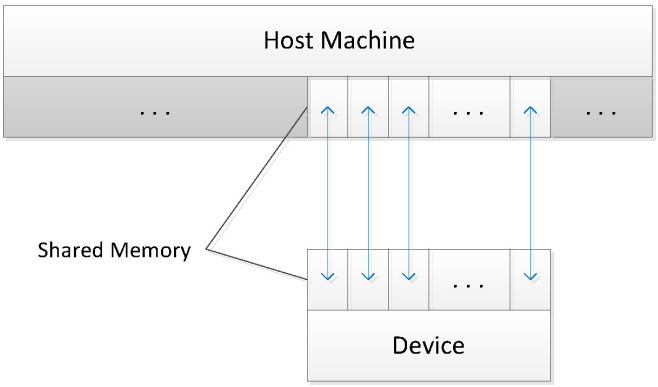


Figure 4.2: An illustration of shared memory

We use a package called `compat-wireless` [129] for the driver part. It is a collection of open-source driver modules that build the wireless sub-system for Linux, and it includes the `b43` driver. In order to modify a driver, it is worth-

[7]This term is also used for memory-mapped file I/O, which should not be confused with what we are discussing here.

Stub	Broadcom Implementation	Function
tx	b43_op_tx	Transmit a frame
conf_tx	b43_op_conf_tx	Set transmission parameters
add_interface	b43_op_add_interface	Add an interface
remove_interface	b43_op_remove_interface	Remove an interface
config	b43_op_config	Change bands, channels, power, etc
bss_info_changed	b43_op_bss_info_changed	Act upon BSS info. change
configure_filter	b43_op_configure_filter	Configure filtering capabilities
set_key	b43_op_set_key	Set encryption key
update_tkip_key	b43_op_update_tkip_key	Update TKIP key
get_stats	b43_op_get_stats	Retrieve operation statistics
get_tx_stats	b43_op_get_tx_stats	Retrieve TX queue statistics
get_tsf	b43_op_get_tsf	Retrieve TSF timer
set_tsf	b43_op_set_tsf	Update TSF timer
start	b43_op_start	Initiate the driver
stop	b43_op_stop	Stop the driver
set_tim	b43_op_beacon_set_tim	Set TIM for beacon frames
sta_notify	b43_op_sta_notify	Notify the AP of station updates
sw_scan_start	b43_op_sw_scan_start_notifier	Start notifier for scanning
sw_scan_complete	b43_op_sw_scan_complete_notifier	Complete notifier for scanning
rfkill_poll	b43_rfkill_poll	Poll `rfkill` hardware state

Table 4.4: Broadcom implementations of mac80211 sub-system functions.

while first having an idea how drivers work under Linux. Each IEEE 802.11 driver in Linux consists of an implementation of a set of operations. These operations are defined in the `mac80211` module which is a part of the wireless sub-system and is also included in `compat-wireless`. Table 4.4 shows a list of these functions. The `b43` driver has additional internal functions apart from those required by the `mac80211` sub-system. One example of such functions is `do_periodic_work`, which is triggered periodically. The calling interval can be modified in the driver source code. As you will see in later chapters, we use this function to export periodic reports and to perform periodic calculations related to our algorithms. The driver also provides the means to read from and write to the shared memory, which is very useful for us.

4.5 Implementing and Testing Algorithms

In this section we will describe the process by which an algorithm is implemented for use on a testbed using Broadcom adapters. Since implementing algorithms is difficult and time consuming, we make sure algorithms have been validated through analysis and often simulations before moving to this stage.

The first step is to think which parts of the algorithm can be implemented in the driver, and which parts in the firmware. The speed/capacity trade-off should always be kept in mind when making this decision. A task involv-

ing heavy calculations on large amounts of data is an example of what is best assigned to the driver. The firmware runs on an 8 MHz core and some instructions can take up to $4\mu s$ to execute. This makes heavy workloads unsuitable for the firmware. Other tasks that should be assigned to the driver are report generation and information dump. The reason for this is simply because the host machine has access to all the resources and output devices required to print out information or consume reports.

Tasks such as carrier-sense-based decision-making, ACK-dropping, and device fingerprinting are implemented in the firmware. The reason is that these tasks require access to the lowest level of the architecture, or save communication overhead if done in the firmware.

The next step is to design a communication mechanism. We mentioned earlier that any communication should be through MMIO, and we also mentioned that this communication can be slow, and unreliable in the sense that changes may not be mapped immediately. Therefore, we should minimize this communication and also plan it so that delays in MMIO communication do not cause misbehavior. Thus, deciding how much information should be shared between the driver and the firmware and how this information will be shared and consumed by either side is an important design decision.

After choosing how to split the algorithm and designing a communication framework, we move on to the implementation. We usually finalize the driver first, since it takes much longer to compile and deploy. Adding some runtime signals and parameters[8] to the driver can make things much easier and eliminate the need to recompile when small changes are necessary. The firmware side should then be developed in line with the implementation of the driver part. In our experience, the driver might also require some changes at this stage, but a careful design helps us avoid this as much as possible. Programming the firmware is more difficult, but deploying changes usually takes a matter of seconds. Thus, the firmware can be developed and tested continually with little overhead.

The final step is testing and reporting. In order to see whether what we see

[8]Signals are anything that can be monitored by the driver, such as creation of a new file. Linux drivers also have a capability of handling start-up parameters. The `modprobe` command can send parameters to drivers as they start up.

from the wireless card is what we should see, we need some reports. Report generation is usually programmed into the driver and is aided by some shell scripts. Deciding what reports are required depends both on the purpose of the implementation, and on implementation details. We need reports to infer results as well as reports to check the sanity of the implementation. The reason we implement reporting at the end is that implementing reporting alongside algorithm implementation brings more complication to both sides, and may cause implementation mistakes.

The execution of tests and the collection of experimental results is accomplished using scripts. These scripts vary between simple shell commands that connect wireless nodes and send traffic between them, and more complex scripts that analyze and digest the information reported by the driver.

4.6 Conclusions

Experimental validation of wireless algorithms and protocols is important because we can see how they work in real world scenarios. Open-source drivers for IEEE 802.11 adapters are now widely available and enable us to manipulate the behavior of Wi-Fi cards. Some researchers have been using them alongside theoretical validation and simulations. Broadcom BCM43xx wireless adapters have the advantage of facilitating low-level manipulation of the protocol stack. There is open-source firmware available for these chipsets and this makes them an asset in experimental validation. We briefly reviewed the architecture of the Broadcom chipset and the development process around it.

The implementation of a behavior on a card consists of making two main decisions. One is how to break the implementation into tasks handled by the firmware and the host machine (driver); the other is deciding how these two parts communicate and how much memory should be allotted to this communication. For evaluation of algorithms, we also need a reporting mechanism implemented in the driver.

CHAPTER 5

Policing Algorithm

With the increasing availability of flexible wireless IEEE 802.11 devices, the potential exists for users to selfishly manipulate their channel access parameters and gain a performance advantage. Such practices can have a severe negative impact on compliant stations. A policing algorithm has previously been introduced that enables the access point to counteract these attacks in wireless networks, and drives misbehaving users into compliant operation without requiring any cooperation from clients. In this chapter we discuss this algorithm and propose amendments to it by the present work, which are aimed to overcome its shortcomings.

5.1 Introduction

Computers equipped with Wi-Fi devices that follow the popular IEEE 802.11 specification [5] employ a decentralized Medium Access Control (MAC) protocol to coordinate their transmissions on the channel. By design, this mechanism ensures compliant users connecting to a wireless network receive equal opportunities to access the medium and in this sense share resources in a fair manner.[1] Each client station, however, operates independently and thus anyone could act more aggressively in order to gain performance benefits if changes can be made to the protocol behavior. This already occurs in practice

[1] This means equal channel access in the original version of the IEEE 802.11 protocol. For later versions that include QoS, however, this equality is limited to traffic belonging to the same access category.

when network interface cards are not designed correctly as reported in [72], where they find that the behavior of the card may even change depending on the number of contenders. More critically, it can happen when users selfishly manipulate their channel access parameters to gain a performance advantage (see e.g. [97]). This can cause significant unfairness, with the performance of the users that abide by the standard severely degraded [64, 65]. Such MAC misbehavior attacks are of increasing concern as open-source device drivers (e.g. MadWifi [66], compat-wireless [129], etc.) are prevalent and allow users to modify the protocol rules either from the command line or with basic programming knowledge. Looking ahead, the trend is towards introducing even further flexibility, e.g. versatile architectures that allow changing the MAC operation of commodity hardware by reprogramming the protocol state machine with the help of simple visual tools [126].

For said reasons, misbehavior detection has received much attention from the research community (see e.g. [103, 97, 64, 101, 98, 96, 95]). Existing work, however, largely focuses on how undesired behavior can be achieved with current cards and on engineering solutions that assist the AP in identifying disobedient users, as well as the nature of their misbehavior. As discussed previously in Chapter 3, only a limited number of proposals address counteracting greedy actions, and these suffer from significant practical drawbacks.

Dangerfield et al. propose an effective policing scheme [6] for IEEE 802.11 WLANs that overcomes the above limitations, since it does not require any modifications to the protocol stack of client stations. It is implemented completely in the access point. By design, a key benefit of their policing algorithm is that it does not require any information about the number of nodes or the nature of their misbehavior. Thus it is effective against a broad class of misbehaviors. With their policing scheme, the AP controls the throughput of misbehaving stations by acknowledging their frames with a probability that depends on the deviation of the stations' throughput from the compliant value. Decreasing the probability of acknowledgement causes a client station to backoff its contention window, thereby reducing their throughput and restoring fairness provided by IEEE 802.11 protocol.

An important feature of this approach is that it only requires measuring the throughput of each client station, which is straightforward as all traffic passes

through the AP in the infrastructure operational mode, and does not require identification of the specific type of attack being used (e.g. shorter backoff, frame bursting, etc.).

There are, however, shortcomings to their approach. An important unsolved problem in the original work is the computation of the compliant throughput. This throughput highly depends on network characteristics (e.g. number of stations and volume of transmissions) and channel conditions. This value is assumed to be known in the original work, and in their simulations, they have precalculated it for each point. One of the main contributions of this thesis is the introduction of a method to calculate this value on-the-fly, given any network condition. The main importance of this addition is that it makes the algorithm practical, and paves the way for its implementation on real hardware. Another shortcoming of the policing algorithm that is addressed in this thesis is that the punishment inflicted on non-compliant stations degrades the network throughput as a whole, as those stations keep retrying their transmission. This issue is solved by equalizing transmission attempt rates instead of throughputs.

The last problem in the policing algorithm is that it can be gamed by a smart station that is aware of the algorithm. In collaboration with P. Patras and D. J. Leith, we have modified the policing algorithm so it is immune to gaming. This will also be discussed in this chapter, along with the rest of the aforementioned amendments.

The solution proposed in this chapter leverages the algorithm designed in [6], but differs in that *(i)* it aims to control the attempt rate instead of throughput, thus seeking to equalize stations' channel access opportunities by driving the channel access probabilities of all clients to the same value, regardless of the contention parameters they employ, we effectively preserve short-term fairness; *(ii)* it carries forward penalties, thus also achieving long-term fairness provided by the standard; and *(iii)* it guarantees that the mechanism cannot be gamed by attackers that detect its operation. In addition to all these changes, we then introduce the Virtual MAC, a novel technique for estimating the transmission attempt rate of a compliant station, and provide sufficient analysis to demonstrate its efficacy. This concept has been previously used in other contexts, such as service differentiation [130]. But in this work we

extend the idea to compliant attempt rate estimation.

5.2 Policing Algorithm

In this section we explain the class of attacks that the policing algorithm tackles and detail the operation of the policing algorithm. We also point out the amendments this work proposes to the algorithm. We consider WLANs with a single-AP (or, alternatively a group of co-operating APs) operating in infrastructure mode, i.e. all packets are transmitted through the AP, as this is the default and most widespread operational mode of today's Wi-Fi deployments.

5.2.1 Class of Attacks

In Section 3.3.1 we discussed different types of attacks on IEEE 802.11. Our focus here is on IEEE 802.11 MAC layer attacks. We do not consider lower layer PHY attacks such as ACK jamming, or higher layer attacks, such as TCP ACK manipulation or station association attacks. We also confine consideration to attacks that seek to obtain performance benefits, rather than simply to disrupt the network operation through e.g. signal jamming [131], or exploiting security vulnerabilities [132].

Our interest in this class of greedy MAC attacks arises from the observation that they can be especially easily realized with currently available open-source drivers, which allow manipulation of the MAC layer parameters (CW_{min}, CW_{max}, AIFS and TXOP [5]), sometimes simply by issuing a single command on the system console (see e.g. `iwpriv` for Atheros-based cards). Each of the attacks considered in this chapter correspond to the modification of one of the aforementioned MAC layer parameters. Thus, they constitute an exhaustive collection of possible attacks in this class.

Note that, despite the possibility of broadcasting set EDCA configurations by means of beacon frames from the AP, non-compliant clients are free to ignore any of the contention parameter values assigned through this (advisory) mechanism and the prevalence of such open drivers provides them sufficient incentives to do so.[2] We assume WLANs implement an authentication mech-

[2]Consequently, earlier TXOP-based airtime allocation approaches (e.g. [35, 37] do not provide effective policing when stations are misbehaving.

anism such as Wi-Fi Protected Access (WPA2) [133], that prevents short and repeated aggressive sessions facilitated by MAC address spoofing techniques. Note also that the IEEE 802.11i standard ensures replay protection through several mechanisms, of which the use of CCMP (Counter Mode Cipher Block Chaining Message Authentication Code Protocol) or TKIP (Temporal Key Integrity Protocol) procedures are particularly relevant to our scheme. Thus, a selfish user will be unable to impersonate compliant clients and jeopardize their reputation.

This can be adapted to open-access networks by augmenting it with a signal-strength based MAC layer spoofing detector [134] or a passive device fingerprinting tool [135]. The resilience of the policing algorithm to more sophisticated security attacks can be further strengthened if used in combination with fine-grained PHY layer information [136].

5.2.2 Controller Operation

To tackle this class of attacks, the policing algorithm's AP exploits the fundamental nature of the ACKs within the ARQ mechanism of IEEE 802.11. Specifically, it uses the fact that stations will usually increase their contention window and re-attempt to deliver a frame that was not acknowledged, before sending the next packet. By appropriately suppressing ACK generation for non-compliant users, the AP can reduce their transmission rate and drive them into compliant operation. We consider, for example, WLANs that operate in a commercial setting where the service provider seeks to monetize connectivity.

A naive solution that simply disassociates users with marginal, possibly accidental, misbehavior (see e.g. [72]) would be operationally unacceptable. Instead, the goal is to effectively correct such behaviors. It is possible, though, that a misbehaving station does not increase its contention window despite not receiving ACKs. For such blatantly and deliberately misbehaving stations, it is not possible to use ACK suppression to drive the station to compliant operation and instead the policing algorithm adapts to drop all ACKs and associated data packets, reducing the goodput of such misbehaving stations to zero.

The key to the performance of this algorithm is the manner in which the

rate of ACK suppression P^i_{ACK} is adjusted for user i. Algorithm 1 details the proposed approach.

Algorithm 1 Determining the rate of ACK suppression.

1: Initialize $t = 0$, $p^i_t = 0$, $P^i_{ACK,0} = 0$ for client station $i, \forall i$.
2: **loop**
3: Estimate the maximum compliant transmission attempt rate $\bar{x}_t$, given the current network conditions;
4: **for** each associated client station i **do**
5: Measure transmission attempt rate x^i_t of the station; Update the penalty:

$$p^i_{t+1} = \max\left(0, p^i_t + \alpha\left(\frac{x^i_t}{\bar{x}_t} - 1\right)\right), \tag{5.1}$$

where $0 < \alpha < 1$ is a parameter that drives the speed of reaction to deviations from the compliant behavior;
6: $P^i_{ACK,t+1} = \min\{p^i_{t+1}, 1\}$;
7: $t \leftarrow t + 1$;
8: **end for**
9: **end loop**

For each station, the algorithm works as follows. At each step t, it compares the measured station attempt rate x^i_t against the compliant value $\bar{x}_t$. The meaning of attempt rate will be described in further detail in the next subsection. When the value of this metric is above the compliant value, the rate of ACK suppression is increased, and vice-versa when the attempt rate is below the compliant value. Thus at a fixed point we have $\frac{x^i_t}{\bar{x}_t} - 1 = 0$, i.e. $\frac{x^i_t}{\bar{x}_t} = 1$ and the station's attempt rate is driven to the compliant value.[3]

Figure 5.1 shows an example of the policing algorithm in operation. In this example we consider an IEEE 802.11g WLAN with three stations: two stations use standard IEEE 802.11g parameters and the third uses a smaller value of $\mathrm{CW}_{\min}$. The time evolution of the stations' throughputs are illustrated as the policing system operates, with the throughputs modeled using a two-class Bianchi-like model described in [137]. Observe that while the more aggressive station initially claims more throughput due to the increased transmission attempt rate, the policing algorithm quickly adjusts the ACK drop probability, so that the aggressive station receives lower performance.

[3]Note that to streamline notation, we will often drop the i superscript from now on, provided there is no scope for confusion.

5.2.3 Throughput vs. Attempt Rate

One might notice a difference between the original policing controller and the one presented here. Dangerfield's policing algorithm looked like the following (notations have slightly changed to better match those described here):

$$P^i_{ACK,t+1} = \left[P^i_{ACK,t} + \alpha \left(\frac{S^t_{c,i}}{S^t_f} - (1 - \gamma P^i_{ACK,t}) \right) \right]_{0,1-\epsilon} \tag{5.2}$$

where $S^t_{c,i}$ is the throughput of client node i and S^t_f is the maximum throughput of a well-behaved node. This controller ensures that the throughput of non-compliant stations is reduced until it converges to S^t_f. However, the non-compliant station still sends packets, whether or not they are acknowledged, and this takes channel time, and consequently degrades channel utilization. The design parameter γ in the original algorithm aims to mitigate this issue by adjusting the size of the penalty.

In the present work we take a different approach, and use transmission attempt rate instead of throughput. This is the proportion of MAC layer slots (with Bianchi's definition) that the station attempts transmission in.[4] The rationale behind using this metric rather than the original *throughput* is that, by design, the policing algorithm ensures standard compliance rather than throughput fairness (see Chapter 3 for more information regarding fairness and compliance). And transmission attempts and their timings are what determine the standard compliance of a station. Hence the new metric better fits the operation and the goal of the algorithm. Moreover, with this metric in place, the γ parameter, which was originally added to address this channel utilization issue, becomes superfluous and hence is omitted.

The algorithm requires an estimate of the maximum compliant transmission attempt rate. That is, the TX rate that would be achieved by a client station employing the standard recommended IEEE 802.11 MAC configuration. We discuss in detail how to estimate this quantity in Section 5.3.

[4]Note that this attempt rate is different from the PHY modulation and coding rate used by individual packets.

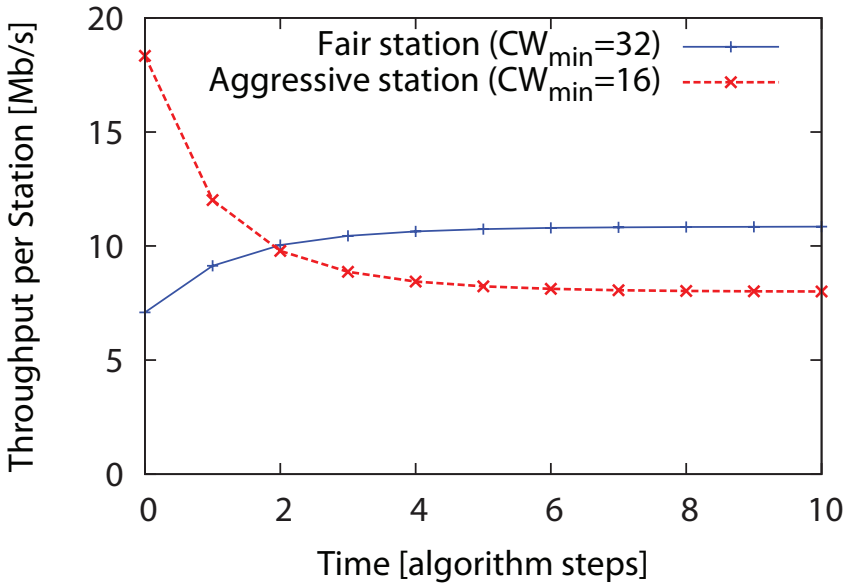


Figure 5.1: Performance in a network with three stations, two using standard IEEE 802.11g parameters, and one with $\mathrm{CW}_{\mathrm{min}} = 16$. The policing algorithm is applied with $\alpha = 0.1$, packet size is 1500 bytes and the stations are saturated.

5.2.4 Penalty Carry Forward

Since $P^i_{ACK,t}$ is a probability value, it can only take values in $[0, 1]$, but for aggressive attacks $P^i_{ACK,t}$ reaches 1 quickly. However, as we do not impose an upper bound on the update of p^i_t, we also consider a version of the algorithm that allows to carry forward and accumulate the penalty when $p^i_t - P^i_{ACK,t} > 0$, until the greedy station reverts to compliant operation or is otherwise disassociated. Thus we prevent gaining long-term advantage over compliant stations. The effect of this change is discussed detail in Section 5.2.5.2 when the robustness of the modified policing algorithm is proven.

5.2.5 Mathematical Analysis

In this section, we first present an analysis for the convergence properties of the policing algorithm. We prove convergence for Algorithm 1, but similar results hold for the version that carries forward the penalty. Then, we study the robustness of the proposed solution under attacks that seek to game its operation with the goal of achieving long-term performance benefits.

5.2.5.1 Convergence

We begin by establishing general conditions under which Algorithm 1 converges to a fixed point. For well-behaved stations we have the following important result.

Theorem 5.2.1 (Well-behaved stations)**.** *For stations satisfying* $x_t \leq \bar{x}_t(1 - cP_{ACK,t})$ *for some* $c > 0$*, Algorithm 1 ensures* $\lim_{t\to\infty} p_t = 0$*. That is, for well-behaved stations the policing algorithm does not drop any ACKs.*

Proof: First note $p_t \geq 0$ and if $p_t = 0$ then subsequent terms are zero. If the sequence does not become constant at zero, then the max with zero is not active in Algorithm 1, and we consider two cases:

1. if $0 < p_t \leq 1$, then
$$p_{t+1} = p_t + \alpha\left(\frac{x_t}{\bar{x}_t} - 1\right) \leq p_t - \alpha c p_t;$$
2. if $p_t > 1$, then
$$p_{t+1} \leq p_t - \alpha c.$$

So, at each step, p_t decreases by at least $\alpha c \min(p_t, 1)$. Thus p_t is non-increasing and bounded below, and so convergent. As $p_t - p_{t+1} \to 0$ we see $\alpha c \min(p_t, 1) \to 0$, and thus $p_t \to 0$. □

Using a model such as [1], we can see that a station following the DCF standard meets the conditions for a well-behaved station in Theorem 5.2.1. The attempt rate will be proportional to the transmission probability (Bianchi's τ) which we can calculate as a function of P_{ACK}, the collision probability for the station and other (fixed) MAC parameters. Figure 5.2 shows that for a range of collision probabilities, these can be bounded with $c \leq 0.4$. Note that with this choice, $\alpha c < 1$ and thus p_{t+1} cannot become negative in case 2) above.

We now show that in many reasonable situations with misbehaving stations Algorithm 1 also converges. Firstly, for misbehaving stations whose transmit attempt rates remain sensitive to ACK suppression we have the following.

Theorem 5.2.2 (Moderately misbehaving stations)**.** *Suppose the transmit rate of a station satisfies the following conditions:*

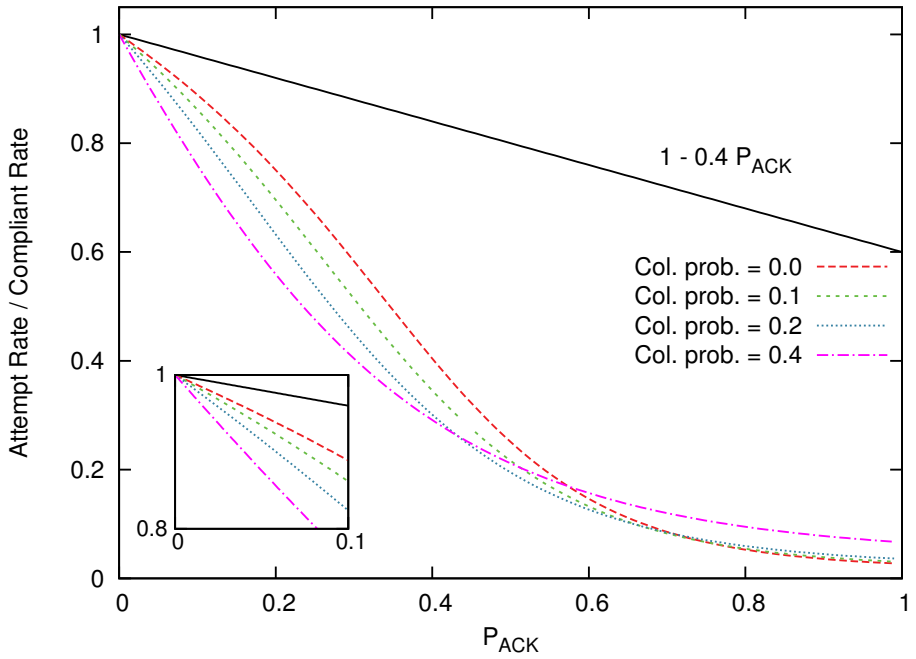


Figure 5.2: Normalized attempt rate, $x_t/\bar{x}_t$, for a standard compliant station under a range of network conditions.

i) $x_t/\bar{x}_t > 1$ *when* $P_{ACK,t} = 0$,

ii) $x_t/\bar{x}_t < 1$ *when* $P_{ACK,t} = 1$ *and*

iii) $x_t/\bar{x}_t$ *is strictly decreasing with* $P_{ACK,t}$ *and Lipschitz with a constant smaller than* $2/\alpha$.

Then Algorithm 1 converges to a point where $x_t = \bar{x}_t$.

Proof: Since $x_t/\bar{x}$ is strictly decreasing, there exists a unique value of $P_{ACK,t}$ where $x_t/\bar{x}_t = 1$. We call this value P. Let $V_t = (p_t - P)^2$. Note that V_t is positive definite and radially unbounded in p_t and

$$V_{t+1} = (p_{t+1} - P)^2 \leq \left(p_t - P + \alpha\left(\frac{x_t}{\bar{x}_t} - 1\right)\right)^2.$$

Expanding, we find

$$V_{t+1} \leq V_t + \alpha\left(\frac{x_t}{\bar{x}_t} - 1\right)(p_t - P)\left(2 - \alpha\frac{\left(\frac{x_t}{\bar{x}_t} - 1\right)}{p_t - P}\right).$$

Note that $\alpha > 0$ and $(x_t/\bar{x}_t - 1)(p_t - P)$ is strictly negative except when $p_t = P$, so if

$$2 > \alpha \frac{\left(\frac{x_t}{\bar{x}_t} - 1\right)}{p_t - P},$$

then we can ensure that V_t converges asymptotically to zero as $t \to \infty$. However, this condition is ensured by requiring $x_t/\bar{x}_t$ be Lipschitz in $P_{ACK,t}$ (and consequently p_t) with a constant smaller that $2/\alpha$. Thus, as $V_t \to 0$ we have $p_t \to P$. □

In the case of highly-aggressive stations for which the transmit attempt rate cannot be made fair using ACK suppression alone (e.g. when backoff of the MAC contention window has been disabled), we have the following.

Theorem 5.2.3. *For stations where* $\exists c > 0$ *such that* $x_t \geq \bar{x}_t(1 + c)$ *for all* $P_{ACK} \in [0, 1]$*, Algorithm 1 ensures* $P_{ACK,t} \to 1$.

Proof: By assumption, $x_t/\bar{x} > 1$. Hence, $p_{t+1} \geq p_t + \alpha c$. It follows that p_t increases to a value greater than 1 and so $P_{ACK,t} \to 1$. □

Of course, some non-compliant stations may not meet the smoothness conditions for convergence of P_{ACK}. Indeed, the station might randomly choose an attempt rate at any time. However, in the next section we show that in this case the station cannot gain from any such strategy, even if does not converge.

5.2.5.2 Robustness

Next we consider a scenario where an attacker becomes aware of the policing algorithm running at the AP and attempts to game its operation with the goal of achieving a long-term benefit in terms of throughput. We demonstrate that our scheme is robust to such sophisticated attacks by showing that, by design, the algorithm will penalize any strategy that deviates from the compliant behavior.

Suppose that the attacker seeks to maximize its goodput and we run the algorithm that carries forward the penalty. The mean goodput over the interval $[0, T]$ is given by

$$S(T) := \frac{1}{T} \sum_{t=1}^{T} x_t(1 - p_t) = \frac{\bar{x}}{T} \sum_{t=1}^{T} (1 + y_t)(1 - p_t) \tag{5.3}$$

where $y_t = x_t/\bar{x} - 1$. Note, our policing update becomes

$$p_{t+1} = \max\left(0, p_t + \alpha y_t\right), \tag{5.4}$$

and if we iterate this backwards to the previous time t^* where p_t was zero, we see

$$p_{t+1} = \max\left(0, \alpha \sum_{k=t^*}^{t-1} y_k\right).$$

Suppose there is a time $T^* > 0$ with $p_{T^*} = 0$ but $p_t > 0$ for $1 \leq t < T^*$. Then, we see $\sum_{k=0}^{T^*-1} y_k \leq 0$, so the average attempt rate of the station up to time T^* is less than that of a compliant station. As $p_{T^*} = 0$, we may remove this interval from our consideration and consider just the times from T^* onwards. By repeating this argument, we see that we only need to consider the potential non-compliant behavior of stations where $p_0 = 0$ and $p_t = \alpha \sum_{k=0}^{t-1} y_k > 0$ for $1 \leq t < T$. We have the following result.

Theorem 5.2.4. *For policing Algorithm 1, suppose $\alpha \sum_{k=0}^{t-1} y \geq 0$ for $1 \leq t < T$. Let Y be an upper bound for y_i and let $\Delta > 1/\alpha + Y$ be a positive integer. Then, if $T > \Delta$ and we consider the values of $S(T)$ as we vary $y_1, \ldots, y_{T-\Delta}$ and hold the other y_i fixed, we find $S(T)$ is maximized by choosing $y_1 = \ldots = y_{T-\Delta} = 0$.*

Proof: *With policing update (5.4) we have*

$$p_{t+1} = \alpha \sum_{k=1}^{t} y_t,$$

and we consider terms in $S(T)$ as follows.

$$S(T) = \bar{x} + \underbrace{\frac{\bar{x}}{T} \sum_{t=1}^{T} y_t}_{\text{goodput gain}} - \underbrace{\frac{\bar{x}}{T} \sum_{t=1}^{T} (1 + y_t) p_t}_{\text{goodput cost}} \tag{5.5}$$

Now,

$$\begin{aligned} \sum_{t=1}^{T} (1 + y_t) p_t &= \sum_{t=1}^{T} (1 + y_t) \alpha \sum_{k=1}^{t-1} y_t \\ &= \sum_{t=1}^{T} y_t \alpha \sum_{k=t+1}^{T} (1 + y_k). \end{aligned}$$

So, the net relative gain is bounded by

$$\sum_{t=1}^{T} y_t - \sum_{t=1}^{T} y_t \alpha \sum_{k=t+1}^{T} (1 + y_k)$$
$$= \sum_{t=1}^{T} y_t (1 - \alpha(T - t)) - \alpha \sum_{t=1}^{T} \sum_{k=t+1}^{T} y_t y_k.$$

Taking the derivative with respect to y_i we get

$$(1 - \alpha(T - i)) - \alpha \sum_{t \neq i} y_i = \alpha \left(\frac{1}{\alpha} - T + i - \sum_{t=i}^{T-1} y_t + y_i \right)$$

which is negative when $i \leq T - \Delta < T - 1/\alpha - Y$, as the sum is non-negative and $y_i \leq Y$. Thus, to maximize the gain, we choose the smallest possible values of y_i subject to the constraint on the partial sums being non-negative. Thus $y_1 = \ldots = y_{T-\Delta} = 0$. □

This results confirms that no benefit can be obtained by deviating from the compliant behavior over $T - \Delta$ steps. Note however that an attacker could potentially attempt to use a more aggressive transmit rate over the last Δ iterations before leaving the network, seeking to gain a small throughput benefit. But the fact that we allow for the penalty to carry forward to future times and consider networks that employ authentication prevents the occurrence of such situations.

5.3 Compliant Attempt Rate Estimation

The main analytical contribution of this thesis is an estimation method for compliant transmission attempt rate. As mentioned before, to decide whether to police an associated station, our algorithm measures their performance and compares this to the maximum transmission attempt rate a compliant client would attain under current network conditions. The fact that network setup and channel conditions affect this value, makes it impossible to be precomputed and hard-coded into the AP, and we need another way to find it at each step. If we did have a compliant station in the network that was sending saturated traffic, we could use its attempt rate as our measure. However, this is almost never the case in a real network: real network traffic is often bursty and sporadic. Deliberately adding such a compliant saturated station

to the network is also undesirable because it wastes airtime with unnecessary transmissions, and this has a severe negative impact on channel utilization. The existence of service differentiation makes said scheme even less practical, since we would need to have a saturated node for each traffic category.

5.3.1 Description of the Virtual MAC

According to what we have discussed so far, we need a mechanism for achieving compliant attempt rate estimation non-intrusively, i.e. without injecting traffic into the network or requiring message-passing between the AP and other stations. To this end we run a *virtual MAC* instance at the AP that reproduces the operation of a compliant station, but does not release packets on the channel. Instead, we monitor channel slots and check the outcome of "virtual" transmissions, i.e. whether virtual attempts would have resulted in successes or collisions. Based on these observations, the mechanism estimates the failure probability f experienced by a compliant station, which can be then used to derive the attainable transmission attempt rate. Virtual MAC was first introduced in [130] for service differentiation, as a tool for measuring packet delays. We extended the Virtual MAC to additionally provide transmission attempt rate estimation, to be combined with the policing algorithm.

We can run the estimator just like a normal DCF-based station, and replace the transmission procedure by the procedure of observing the channel for the slot the sending should occur. Thus a "virtual collision" is identified by the channel becoming busy in that slot, and a virtual success is identified by the channel remaining idle. However, we take an approach which is more suitable for implementation on wireless adapters. We count idle and busy slots and use them to calculate the transmission attempt rate using Bianchi's model [1]. According to this model, the probability of transmission τ of a compliant station is defined

$$\tau = 2\frac{(1-2p)}{(1-2p)(W+1)+pW(1-(2p)^m)} \tag{5.6}$$

where p is the probability of failure, W is the size of the minimum contention window, and m is the maximum backoff stage (e.g. 5). Looking at this equation, τ depends on the variable p, which in turn is dependent in the number of contenders, channel conditions, etc.. In order to compute p, we use the information collected by our Virtual MAC. Given the number of idle and

busy "Bianchi slots", p can be calculated as

$$p = \frac{n_b}{n_b + n_i}$$

where n_b is the number of busy slots, and n_i is the number of idle slots. The logic behind this calculation is that transmissions fail if they collide with other stations' transmissions. So, if the virtual station were to transmit in a slot already containing a transmission from another station, it would be a collision. So it only considers that slot as a failure candidate, contributing to p. Plugging the result in (5.6), we can calculate τ, and then the attempt rate will be calculated as follows:

$$x = \frac{\tau(1-p)(n_b + n_i)}{d} \tag{5.7}$$

that is, the total number of slots that contain a successful transmission attempt from the station per unit time (d is the duration of observation in the above formula).[5] This value can then be used as an estimate for $\bar{x}_t$ in Algorithm 1. In Section 5.3.2 we discuss in further detail whether and how we can use this estimate.

Note, our Virtual MAC assumes all non-colliding transmissions are successes, i.e. for chosen PHY rate there are no channel errors. This is the best case for a real station, and so in the worst case we overestimate the attempt rate for a compliant saturated station with channel errors. This will not result in the punishment of a compliant station.

5.3.2 Mathematical Analysis

In this section we give a formal analysis of this approach and investigate its accuracy. Suppose we have a network of n stations transmitting with probabilities $\tau_1, \ldots, \tau_n$. Further, suppose that a station is saturated, for instance station 1. Assume for now that this station is compliant. We can write the failure probability due to collisions for this station as

$$f_1 = 1 - (1 - \tau_2) \ldots (1 - \tau_n). \tag{5.8}$$

[5] A meticulous reader might note that (5.7) produces successful transmission attempts rather than total number of attempts (i.e. including collisions). It is worth mentioning that this is exactly what we are after. What we include as a failed attempt in our calculation for policing is not a collision, but rather a successful attempt by a station that is deliberately not acknowledged by the AP. With this definition, the Virtual MAC has no failed attempts.

As the station is compliant,

$$\tau_1 = g(f_1),$$

where g is a function mapping the failure probability to the transmission probability and is given by [138]:

$$g(f) = \frac{2(1-2f)(1-f^{R+1})}{W(1-(2f)^{m+1})(1-f)+(1-2f)(1-f^{R+1}) \\ +W2^m f^{m+1}(1-2f)(1-f^{R-m})}. \tag{5.9}$$

In the above, $W = \mathrm{CW}_{\min}$, m is the maximum backoff stage and R denotes the retry limit.

Consider now that the AP runs a saturated Virtual MAC instance. We can similarly express the failure probability f_v the Virtual MAC observes, as follows:

$$\begin{aligned} f_v &= 1-(1-\tau_1)(1-\tau_2)\dots(1-\tau_n) \\ &= (1-\tau_1)(1-f_1) = 1-(1-g(f_1))(1-f_1), \end{aligned} \tag{5.10}$$

where the second line is derived from (5.8). g here is the compliant backoff function given by (5.9). Note that if we know f_v, we can solve the above for f_1. In Figure 5.3, we plot the relationship between the virtual and actual failure probability of a saturated station. To add perspective, we also plot f_v with a dotted line. We observe that the difference between the two is relatively small and reduces as the contention rate increases.

Since there is a one-to-one mapping from f_v to f_1, we could invert this[6] to obtain an exact value for the failure probability of a compliant saturated station and apply (5.9) to compute the maximum achievable rate $\bar{x}$ of a compliant station. Another approach is to compute the virtual attempt rate, $g(f_v)$, and scale this up by 14%, as numerical calculations of both the virtual and actual maximum achievable attempt rate show this is a good estimate of their gap, over a broad range of network conditions. To make this clearer, we plot τ_v against τ_1. As we just mentioned $\tau_1 = g(f_1)$ can be calculated using (5.9), and τ_v is also a function of only f_1, given by (5.10), so it can be calculated similarly. Figure 5.4 shows this plot over its possible range. It can be seen in

[6] We can do it using a standard root finding algorithm, such as bisection.

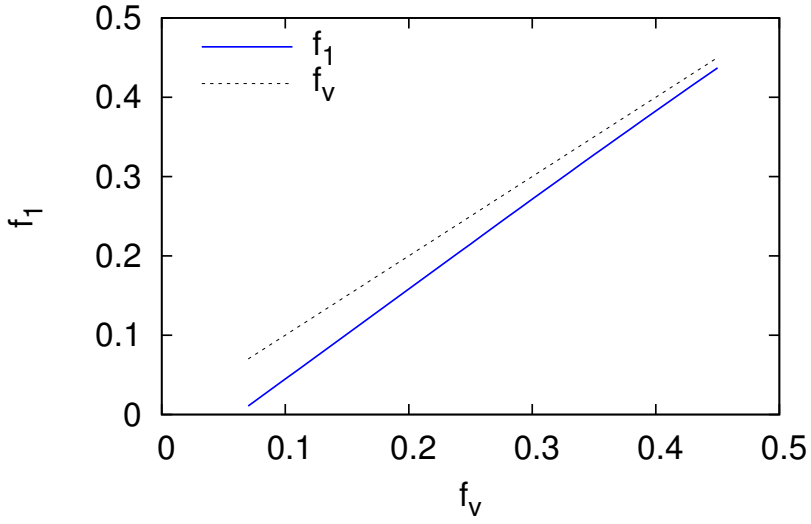


Figure 5.3: Relationship between failure probability of a virtual station and that of a real compliant client.

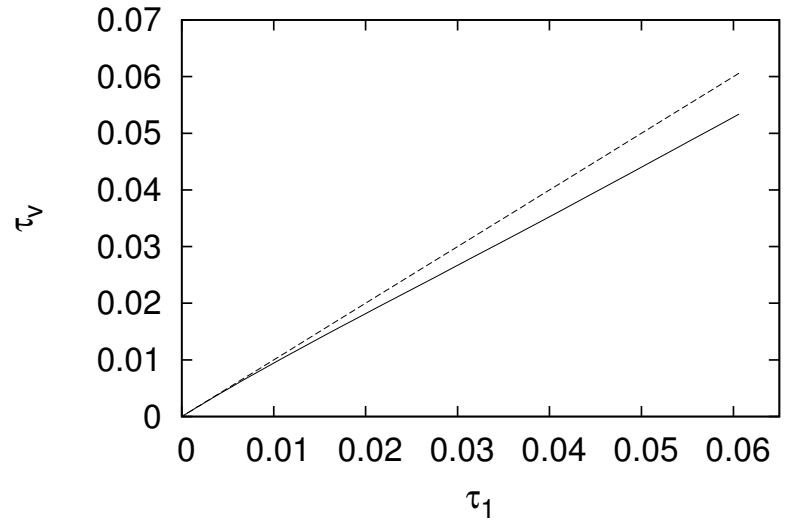


Figure 5.4: Relationship between failure probability of a virtual station and that of a real compliant client.

the plot and the corresponding numbers that the difference between the two lines never exceeds 13.5%.

On a more practical note, we can also see the effect of increasing network

size on the transmission attempt rate of both the virtual STA and a saturated compliant station, as network size is what actually affects failure probabilities, and transmission probabilities as a result. We know that the transmission attempt rate of a compliant station can be expressed as $x_1 = \tau_1(1-f_1)$ frames per slot. The error of the attempt rate of the virtual station S_v according to S_1, can be written as:

$$e_v = 1 - \frac{x_v}{x_1} = 1 - \frac{\tau_v(1-f_v)}{\tau_1(1-f_1)}.$$

As this is a function of one unknown f_1 and a function of just the behavior of f of a compliant node, we can plot its possible range. Figure 5.5 shows the evolution of attempt rates and the error, with increasing network size. As we can see, the number of attempts decreases as expected while the Virtual MAC estimation stays close to this measure. Even relative error in Figure 5.5b stays small, less than the 14% we use.

The remaining question is how long should the channel observation period be to ensure an accurate estimation of f_v. To answer this, we regard the virtual transmission attempt as a Bernoulli trial, whereby a failure is observed with probability $\hat{f}_v$ and a success with probability $1 - \hat{f}_v$. By the central limit theorem, if the number of observations N is large, the distribution of $\hat{f}_v$ is approximately normal with mean f_v and variance $\sigma^2 = f_v(1-f_v)/N$.

Say we want to compute the number of samples N that gives us 95% confidence that the estimated mean has precision ϵ, i.e. $P(|f_v - \hat{f}_v| > \epsilon) < 0.05$. The confidence interval is $\hat{f}_v \pm z\sigma$, where $z = 1.96$ is the z-score required for 95% confidence. Since σ is unknown and $\hat{f}_v(1-\hat{f}_v) \leq 0.5$, using this conservative upper bound [139], N must satisfy

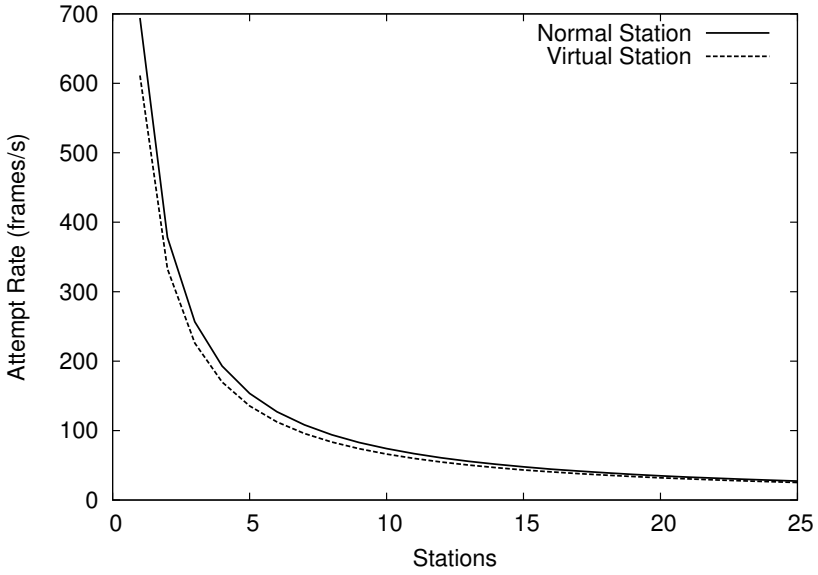
$$\frac{z}{2\sqrt{N}} = \epsilon.$$

Thus,

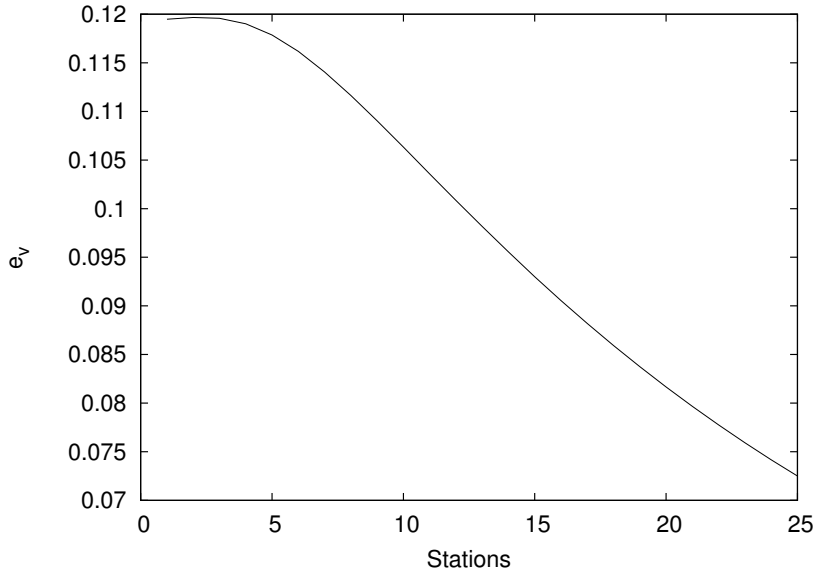
$$N = \left(\frac{z}{2\epsilon}\right)^2.$$

To translate this into an observation period required for a good estimation of compliant performance before an update of the P_{ACK} probabilities, consider the average slot duration in a network with saturated stations

$$E[T_{slot}] = P_e\sigma + P_sT_s + P_cT_c,$$



(a) Saturation Attempt Analysis



(b) Virtual MAC Estimation Error

Figure 5.5: Accuracy of the Virtual MAC estimation

where P_e, P_s and P_c are the probabilities that a slot is empty, contains a success, or contains a collision respectively, and σ, T_s and T_c are the corresponding slot durations (see [1] for detailed calculations).[7] Thus we compute the observation interval that gives an accurate estimation of the mean as

$$T_{update} = N \cdot E[T_{slot}].$$

To indicate the values T_{update} would take in practice for $\epsilon = 0.01$, in Figure 5.6 we plot the necessary channel observation time for obtaining an estimate according to the above requirements for different network conditions in terms of number of saturated stations and assuming nodes send packets with 1000 byte payload at 11 Mb/s (IEEE 802.11 HR/DSSS). We conclude, that an observation interval above 5 seconds will ensure a good estimation of the compliant performance in many scenarios. In our experiments we conservatively use a $T_{update} = 10$s for all tests. In what follows, we evaluate the performance of our prototype in a real testbed and demonstrate its effectiveness under different types of misbehavior.

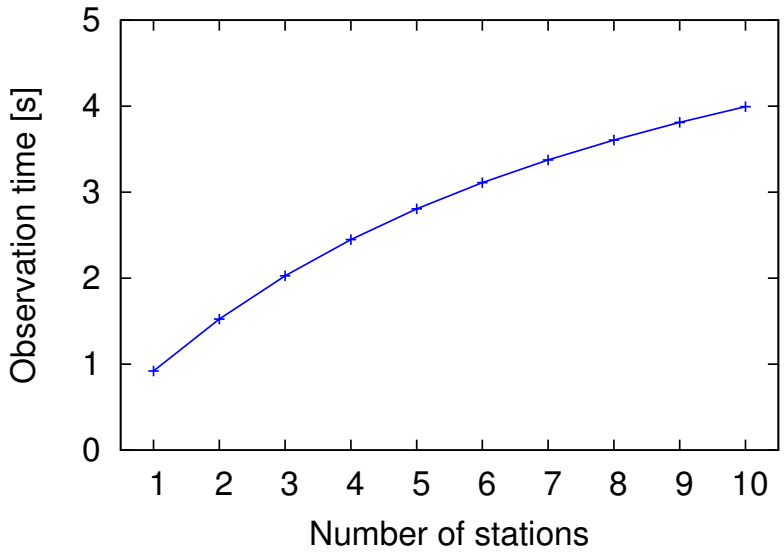


Figure 5.6: Observation time required to estimate f_v vs. network size.

[7] Note that $T[slot]$ is upper bounded by the length of a successful transmission T_s, which is readily obtainable in practice from the "duration" field of correctly received frames.

5.3.3 Adapting the Estimator to EDCA

We introduced Enhanced Distributed Channel Access (EDCA) in Section 2.4. This DCF amendment introduces traffic categories, each of which can have different contention parameters. With EDCA, high-priority traffic has a higher chance of being sent than low-priority traffic: a station with high-priority traffic waits a little less before it sends its packet, on average, than a station with low-priority traffic. It might seem that the policing algorithm conflicts with this scheme at first. We will explain here how this algorithm can be adapted to work under EDCA without interfering with service differentiation.

We know that the goal of the policing algorithm is to ensure standard compliance, and traffic with different priorities that still abide by EDCA standard should not be penalized. To solve this problem, we recognize the fact that each EDCA traffic category can have a different AIFS, and idle/busy slot counts can be different. Different contention windows also cause variability in g calculation for different categories. A basic step in accommodating this variability would be to introduce more than one instance of the Virtual MAC in the AP: one for each traffic category. Thus, we can determine the attempt rate a station can achieve in each AC. The next step would be dividing transmission attempts of each station into different traffic classes, through TSPECs. So, a station that sends traffic in two categories will be treated as two separate stations by the policing algorithm, each of which judged by the corresponding Virtual MAC estimate.

As the estimator uses Bianchi's model [1], and said model is for a homogeneous network, it might seem challenging to extend the estimator to multiple classes. However, it is more straightforward than it first appears. Bianchi's model of a Wi-Fi network has two components. One component relates the probability of a station transmitting τ in an available slot to the probability of collision, p, through a function, g that models the MAC. This is then combined with a model of the network, which says the probability of no collision is the probability no other station transmits.

Bianchi combines these in a homogeneous network to give $\tau = g(p)$ and $1-p = (1-\tau)^{n-1}$. The Virtual MAC only makes use of the function g to determine the transmission probability τ from its estimate of the collision probability p, and so does not depend on the homogeneous network assumption used elsewhere

in Bianchi's paper.

As access categories are determined by the MAC parameters CW_{min}, CW_{max}, AIFS, and TXOP, we must show how to accommodate these in the virtual MAC or estimator. The MAC parameters CW_{min}, CW_{max} and the maximum number of retries are implicit parameters of the function g, and so can be accounted for by selecting an appropriate g. The AIFS/DIFS MAC parameter determines which slots should be considered when counting busy/idle slots, and so can be accounted for by adjusting which slots are considered by the Virtual MAC. Finally, TXOP changes the amount of time that can be used for transmission, and this is accounted for by having the throughput estimator count the extra time if a station exceeds its allocation.

To run the virtual MAC for multiple access categories, the Virtual MAC must know the AIFS value for each access category i, and count the busy/idle slots accordingly before calculating a per-access category collision probability p_i. Each access category will have its own function g_i which accounts for the the backoff parameters, so $\tau_i = g_i(p_i)$, allowing the calculation of the expected throughput for that category. One barrier to this method, however, could be implementation cost. As we will see in the next chapter, the driver runs the algorithm, so it would be capable of running multiple instances of the Virtual MAC. Nevertheless, the information needs to be stored on the shared memory, which is relatively small. With EDCA in place, the memory required to store station information will be multiplied by the number of ACs, which leads to a reduced number of stations our algorithm can accommodate.

5.4 Limitations and Workarounds

In this section we discuss limitations of the policing algorithm. We discuss how new IEEE 802.11 features affect the operation of the algorithm, and what can be done to mitigate the effects. We also discuss the attacks that are immune to the policing scheme.

5.4.1 New IEEE 802.11 Features

The policing algorithm was designed and analyzed for IEEE 802.11 b/g networks. However, these rather dated standards no longer rule the IEEE 802.11-based WLANs. New amendments have introduced features that may make

the policing algorithm less effective. In Chapter 2 we introduced some of these features. Here we discuss how these features can affect the operation of the algorithm, and how the algorithm can be amended to mitigate the effects.

5.4.1.1 Block Acknowledgements (BA)

As we described before, using this feature, instead of sending an ACK for every single MPDU, a single ACK can be sent for a group of MPDUs. A block acknowledgement (BA) can support up to 1024 data units (fragments). Even with the modest $11Mb/s$ speed of IEEE 802.11b this many frames with 1024 bytes of payload would take less than a second to transmit, while the policing iteration is usually much longer (see Section 5.3.2). Besides, data rates in IEEE 802.11n are typically much higher than in IEEE 802.11b. So the gap between ACKs does not create a problem for the algorithm, and we can still achieve node policing by exploiting BlockACKs. A BA contains a bit-map, each bit of which indicates the successful reception of a single MPDU fragment. By distributing ACK-dropping samples over this bitmap, we can implement the policing scheme. This is assuming that immediate Block ACKs are used. Delayed Block ACKs are more complicated, because there is no link between the frames and the Block ACK, which means that the misbehaving station may have a long head start. However, this requires further investigation, because of the choices left open to the implementor.

5.4.1.2 No ACK

This feature is intended for traffic that is time-critical. It prevents the retransmission of such data, and corresponding frames are neither acknowledged by the receiver, nor expected to be acknowledged by the sender. This feature can be troublesome for the policing algorithm as it relies on the ACK mechanism. With No ACK, the policing algorithm will not be able to operate normally for that traffic class. A quick solution to this problem comes from the way QoS works in IEEE 802.11. In order for a station to send a traffic stream (TS) with specific QoS requirements, it should first send traffic specification (TSPEC) for that TS. The AP can then choose to accept or reject the TSPEC [140]. One solution would be to reject all TSPECs that include the QoSNoAck policy, although it disables this feature altogether.

Even if we don't take this extreme measure, it does not mean that a non-compliant flow will still gain benefit. Without the signaling provided by ACKs, the station cannot be sure if the transmission was successful. Consequently, it will be unaware of the punishment, and will not increase its contention window. This leads to a persistent non-compliance which causes the policing algorithm increase the ACK-dropping (or frame dropping in this case) probability to 1 and beyond, and eventually it can disassociate the station with the non-compliant TS. Also note that frames will still be dropped, even though there are no ACKs to be skipped.

5.4.1.3 Direct Link Setup

This feature can also disarm the policing algorithm at the AP, as there will be no more central control when two stations in a BSS communicate directly. The only real solution to this problem is again to disable this feature. In order to setup a direct link, a station needs to send a DLS action frame to the AP, and the AP needs to approve it. The policing-equipped AP can refuse all DLS requests to always be in full control, or reject them from stations that have exhibited significant misbehavior.

Another less elegant solution would be to implement the policing algorithm in compliant stations. The AP can then use device fingerprinting [135] to recognize trusted stations that are equipped with the policing algorithm, and only allow DLS when at least one side of the pair is in the circle of trust! However, this is a decentralized approach which is not in the spirit what the policing algorithm is intended for.

5.4.2 Attacks That Are Immune

We described different types of misbehavior in IEEE 802.11 in in Section 3.3.1 of Chapter 3. In Section 5.2 we described classes of attacks the policing algorithm covers and the reason behind that choice. In this section we put it in perspective by describing attacks that can still be effective with the policing algorithm.

Jamming attacks (see 3.3.1) are very good examples of attacks that are immune to the policing algorithm. By jamming control frames, a station will simply buy more time on the channel by continuously causing other stations to

back off. The AP will not notice the misbehavior if the non-compliant station uses correct EDCA parameters, and it will only assume that other stations are not active. Packet forging attacks also have a similar results. Although a smart AP can be programmed to detect packet forging attacks, the policing algorithm alone cannot detect these attacks.

Furthermore, the policing algorithm neither targets nor is effective against attacks that aim only to degrade network performance without an intended gain for the attacker. Examples of these attacks include DoS attacks, and those jamming attacks that don't rely on higher layer information and frame types.

While these attacks are hard or impossible to treat using the policing algorithm, they are also hard to execute, and only a more highly skilled user can implement them. The group of attacks we discussed in this chapter are those that are both easy to implement and give the non-compliant station throughput advantage.

5.4.3 Rate Control

The policing algorithm can have a negative impact on rate control algorithms that rely on retries, as it forces retries. This can be a problem due to the performance anomaly of IEEE 802.11 [21]. It is, however, compatible with other rate control algorithms such as Minstrel [25]. The next chapter will provide experimental results showing the impact of this algorithm on rate adaptation algorithms.

5.5 TCP Traffic

As both the original and our extended version of the policing algorithm rely on ACK-dropping for their operation, one might understandably ask the question of whether this could have a negative impact on TCP's congestion control. The answer to this question is that TCP ACKs are on a higher layer than the IEEE 802.11 ACKs. For a TCP packet to be discarded, all MAC-level attempts to send the corresponding frame must fail. For example if the IEEE 802.11 retry limit is $r = 7$, then 7 consecutive frames must be dropped by the AP in order for the TCP packet transmission to fail.

So, the probability of a TCP packet being dropped will be $P = P_{ACK}^{r} = P_{ACK}^{7}$, where P_{ACK} is the current ACK-dropping probability for the station. This value is only 0.0078125 for $P_{ACK} = 0.5$, which is a relatively high ACK-dropping probability. However, this also means that a station will lose about 1.5 packets on average if it sends 250 frames per policing iteration. For less aggressive stations, this value will be even smaller, and marginally misbehaving stations will see a healthy TCP link with reduced throughput due to lower layer retries. This is desirable, as these limited losses will have little impact on TCP's congestion control.

For stations involved in significant misbehavior, the extra degradation caused by TCP congestion control is not a big issue as it will reduce throughput and our goal is to incentivize standard compliance. But we do not want to cause a TCP backoff for compliant stations. As far as P_{ACK} is concerned, we will see in the next chapter that it never goes above 5%, and is usually considerably less in practice. For this ACK-dropping probability, the probability of a TCP frame to be dropped is only 10^{-9}, which we consider negligible. In Chapter 6 we run experiments to show the effect of policing on misbehaving as well as compliant stations using TCP.

5.6 Conclusions

In this chapter we introduced a policing scheme that penalizes MAC misbehavior and preserves the fairness provided by the DCF in IEEE 802.11 wireless networks. We chose this scheme because is executed at the AP and does not require any modifications to compliant devices. We demonstrated the convergence of our algorithm, and presented the proof for robustness to sophisticated attacks that seek to game its operation.

We amended said policing scheme by adding a compliant attempt rate estimator using a Virtual MAC mechanism, and showed that the estimation error is limited and decreased when network size increases. We also calculated the measuring interval required to achieve our desired accuracy.

The policing algorithm is designed to work under IEEE 802.11b/g. However, it can also work with new IEEE 802.11e/n features such as block acknowledgements, No ACK, and direct links, although in some cases solutions are too complicated, or need decentralized control. Furthermore, while the algo-

rithm treats greedy stations that do not comply with contention parameters, it is not effective on jamming attacks and those attacks that aim only to reduce network performance.

CHAPTER 6

Experimental Evaluation

In the previous chapter we introduced an effective and robust policing algorithm to counteract misbehaving IEEE 802.11 stations. This scheme is amenable to practical implementation on existing commodity hardware. This chapter describes implementation details of the policing algorithm on such hardware, and provides a wide range of experimental results which put the algorithm into test in different scenarios.

6.1 Introduction

The previous chapter introduced the policing algorithm. As an important part of the contribution of this thesis, we added a compliant transmission attempt rate estimator to this algorithm, using a method called Virtual MAC (see Section 5.3). With this mechanism in place, the policing algorithm is now feasible for implementation on real hardware. To establish this feasibility, we present a prototype implementation of the policing algorithm that uses off-the-shelf hardware. We explain this prototype in great detail through flowcharts and pseudocode. We validate the performance of our implementation by conducting extensive experiments over a wide range of misbehavior scenarios.

Experiments in this chapter are chosen to evaluate the performance of both the policing algorithm and the Virtual MAC. We are interested in determining whether the policing algorithm effectively penalizes attackers irrespective of the network size, number of attackers and the parameters manipulated. We

also want to show that our amended algorithm does not mistakenly penalize compliant stations, even in complex situations where compliant stations generate different volumes of traffic and so some clients consume the air time underutilized by others. Further, we study the impact of the policing algorithm on state-of-the-art PHY rate control algorithms.

As we discussed previously, the underlying principle behind the policing algorithm approach is to control the throughput of attackers by censoring the generation of MAC layer ACKs. Although this technique is used even in works prior to the policing algorithm[1], but to the best of our knowledge, before present work, it has not been implemented with real devices, as this fundamental operation is handled at the firmware level.

6.2 Implementation

To demonstrate that deploying the amended policing algorithm is feasible with off-the-shelf hardware, in this section we present a Linux-based prototype implementation that we developed and discuss details of the implementation of the policing algorithm and the Virtual MAC technique for transmission attempt rate estimation. You can see Appendix B for more information regarding the actual code.

6.2.1 Architecture

In order to implement the algorithm, we need to implement the suppression of MAC ACKs with existing devices. This is a challenging task, since generation of ACK frames is a basic operation that is handled at a low level within the wireless stack, below the device driver. To tackle this challenge, we based our implementation on an AP equipped with a Broadcom BCM4318 wireless adapter that employs the OpenFWWF firmware [67]. The key advantage of using this open-source firmware is that it allows modifying the MAC protocol state machine running on the device, as already reported in [141, 125]. However, as we mentioned in Chapter 4, the firmware runs on a modest 8 MHz processing unit on the network interface card. So the more computationally demanding operations of the algorithm need to be managed by the driver which resides on the host machine.

[1] ACK skipping has been suggested as an effective means to allocate bandwidth for traffic prioritization in a network of well-behaved nodes [109, 110, 111]

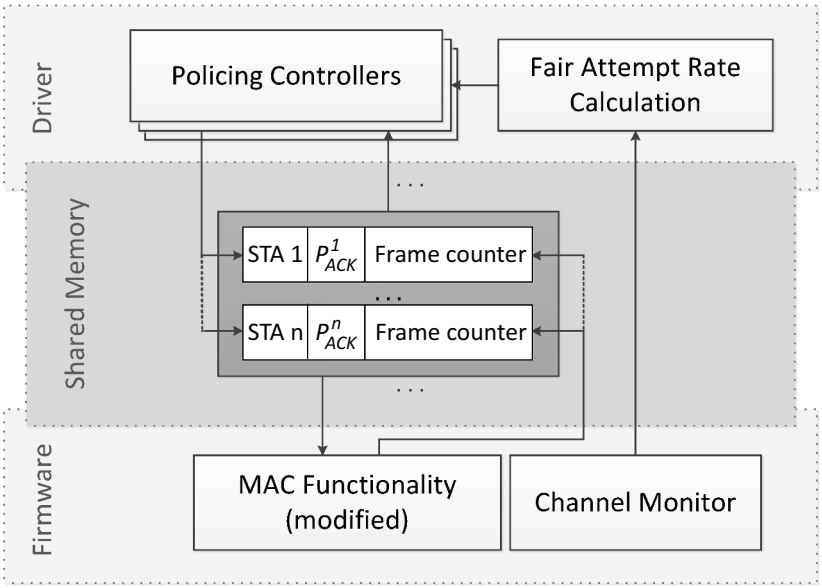


Figure 6.1: Schematic view of the policing algorithm implementation

Figure 6.1 illustrates the essential building blocks of our prototype. As shown in the figure, the implementation is split between the firmware and the driver. The former handles bookkeeping of per-station frame count, channel monitoring and ACK generation, while the latter manages the TX rate computation and updating the ACK suppression rate for each associated client based on the policing algorithm. Note that the two parts can communicate using the 4 KB *shared memory* shared memory (see Chapter 4). Since a large portion of this remains unused during normal operation of the card we use it to store the information pertaining to each station and required by our algorithm. To maximize the efficiency of this small memory and inherently time-consuming communication, we design a data structure in the shared memory that holds required information with minimum space requirement.

Figure 6.2 shows the structure of the memory allocated for policing. The allocated memory starts with a hash map. This part of the memory block is 512 bytes long, and contains 256 words, each of which can contain a single memory address. The addresses stored in this hash map point to memory locations of the following record table, and are used for fast lookup in this table.

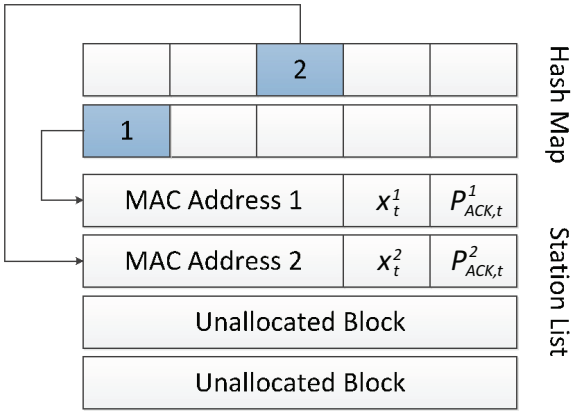


Figure 6.2: Memory structure used to store policing data. The hash map items point to per-station information elements.

As you can see in Figure 6.2, each record in the table contains three pieces of information. The longest piece of information is the station's MAC address, which is 3 bytes long. This is to confirm the address resolved by the hash function. In case of a hash collision, a different slot is chosen using open addressing. The second part of a record is the current number of frames received from the station, and the third part is the probability of ACK-dropping. This field is not a floating point number, but rather a 16-bit integer. A value of 0 for this field corresponds to $P_{ACK} = 0$, and a value of 65535 corresponds to $P_{ACK} = 1$.

To summarize the operation of the implementation, when a frame arrives, the AP quickly looks up in the hash map, and creates a record in the table if necessary. Then it uses the ACK-dropping probability field to determine whether it should drop the ACK. It then increments the frame counter. Once every iteration, the P_{ACK}^t needs to be updated for each record. The AP goes through all valid records, and uses (5.1) to update the probability. Subsequently, it also clears the attempts field (sets it to zero) to mark the beginning of the next iteration. Next, this operation will be explained in further detail.

6.2.2 Firmware Implementation

We implement ACK handling in the firmware, as this is a highly time-sensitive operation. Specifically, the decision whether or not to acknowledge a correctly received frame must be made within SIFS time and thus must not be inter-

rupted or delayed by other tasks. For each frame received with a correct frame check sequence (FCS), the lookup routine hashes the source MAC address to retrieve the pointer storing the information for the corresponding station. If such record does not exist, this routine creates one, and returns the new record instead. Algorithm 2 shows the pseudocode for the lookup routine.

Algorithm 2 MAC address lookup routine.

```
1: function POLICING-LOOKUP(addr)
2:     hash ← h(addr);
3:     if map[hash] not set then
4:         list.size ← list.size + 1;
5:         map[hash] ← list.size;
6:         return list[list.size];
7:     else
8:         while list[map[hash]].address ≠ addr do
9:             hash ← hash + 1;
10:        end while
11:        return list[map[hash]];
12:    end if
13: end function
```

After finding the data record for the station, the firmware increments the frame counter for the sending station. We then fetch the corresponding P_{ACK} value, and use it together with the internal random number generator (RNG) of the BCM4318 chipset to decide whether to generate or suppress the acknowledgement. If the frame is not acknowledged, the memory allocated for the packet is released and the state machine returns to idle state. Conversely, if we decided to send an ACK, we jump to ACK generation from here. This procedure is depicted in Algorithm 3. The *Random* function in this algorithm returns the value of the internal 16-bit integral RNG register.

Virtual MAC is more complicated, as it must capture and interpret carrier-sensing information. In b43 chipset, a special-purpose register called IFS_STATUS always holds the current channel state. We can use this register to determine when the channel goes busy and idle.[2] Algorithm 4 shows a single iteration of the VMAC operation within the firmware. This procedure is placed in the firmware's idle process, and is continuously called whenever the device is

[2]This register holds a bitmap of flags, and channel busy/idle status is only one of the flags it provides.

Algorithm 3 Policing firmware implementation.

```
1: procedure POLICE-FRAME(frame)
2:     sta ← POLICING-LOOKUP(frame.addr);
3:     if frame.type ≠ DATA then return ;
4:     end if
5:     sta.attempts ← sta.attempts + 1;
6:     if RANDOM < sta.P then
7:         DISCARD-FRAME(frame);
8:     end if
9: end procedure
```

idle. It is normally called by the firmware every $1\mu s$ (for more information regarding the firmware architecture, see Chapter 4).

Algorithm 4 Virtual MAC firmware implementation.

```
1: procedure VIRTUAL-MAC-ITERATION(time, state)
2:     if first call then
3:         store.start_time ← time;
4:         store.cur_state ← INVALID;
5:         store.prev_state ← BUSY;
6:     end if
7:     if mem[cur_state] = state then
8:         return ;
9:     end if
10:    backup_state ← store.cur_state;
11:    store.cur_state ← state;
12:    diff = time − store.start_time;
13:    if diff ≥ 50μs & store.prev_state ≠ backup_state then
14:        store.prev_state ← backup_state;
15:        if backup_state = BUSY then
16:            store.busy_slots ← store.busy_slots + 1;
17:        else
18:            store.idle_time ← store.idle_time + diff;
19:        end if
20:        store.prev_state ← backup_state;
21:    end if
22:    store.start_time ← time;
23: end procedure
```

Note the object called *store* in said algorithm. You can assume that members of this object are preserved between calls, and can also be shared with the driver. This imaginary object notation is solely for ease of understanding. In

the real implementation, each field of this object is in fact a word in the shared memory, and is accessed by its address within the firmware. The main output values of this Algorithm 4 are *store.busy_slots*, and *store.idle_time* which are later retrieved and used by the driver.

One thing you may notice in Algorithm 4 is that we ignore spikes that are shorter than $50\mu s$. This is due to the fact there are so-called "training sequences" in which the device tries to determine noise level. In a training sequence, the signal is amplified and the hardware may show the channel as busy, even though it is free. However, these spikes are always small, and shorter than $50\mu s$.

6.2.3 Driver Implementation

As we discussed in 4.5, the driver code runs on the CPU of the host and can perform calculations more quickly. So, the policing update which controls the penalty associated to each client is implemented in the driver. We modify the `b43` driver of the open-source `compat-wireless` [129] package to manage the more computationally demanding operations of our algorithm.

The computation of the transmit rates and updates of the penalties according to (5.1) are executed at configurable discrete time intervals[3], when the driver reads the information stored in the shared memory for each associated station and performs the following operations: *(i)* computes the transmission attempt rate of each station based on the frame count, *(ii)* estimates the compliant attempt rate, and *(iii)* updates the ACK-dropping probabilities P^i_{ACK} and writes their values back into the corresponding blocks, and *(iv)* resets frame counters.

So far we know how ACK-dropping probabilities are used in the firmware. Algorithm 5 shows how they are assigned in the driver. It uses the principles described in 5.3). Arguments *store* and *list* represent shared memory data structures, and use the same notation as previous algorithms. You may have noticed the use of a method `Selective-Copy` in this algorithm. This method copies only the MAC address, and the attempt counter from a record, and

[3]To do this, we leverage a function called `do_periodic_work` in the b43 driver, which is normally used to perform periodic driver-specific tasks.

leaves the destination P untouched. The P_{ACK} is fully controlled in the driver, and is never modified by the firmware.

Algorithm 5 Policing driver implementation.

```
1: procedure POLICING-ITERATION(list, store)
2:     for i from 1 to list.size do
3:         driver_list[i] ← SELECTIVE-COPY(list[i]);
4:         list[i].attempts ← 0;
5:     end for
6:     x̄ ← VIRTUAL-MAC-ESTIMATE(store)
7:     for i from 1 to driver_list.size do
8:         sta ← driver_list[i];
9:         x ← sta.attempts;
10:        if x > 0 then
11:            p ← sta.P;
12:            p ← max(0, p + α(x/x̄ − 1));
13:            sta.P = p;
14:            list[i].P ← min(1, p);
15:        end if
16:    end for
17: end procedure
```

You may also notice that we go through the station record list twice. On the first pass, we copy the whole list locally and reset the source counters, and on the second pass, we do the actual work. The reason we don't do everything in one pass lies in the fast-paced nature of the firmware task. As new frames can arrive every millisecond, we want to mark the beginning of a new iteration (with zero attempt counters) as quickly as possible, before moving on to more time-consuming tasks.

While above reason may justify local bookkeeping and its inherent redundancy, there is another reason for this, which is as important. Remember from Chapter 5 that the ACK-dropping probability can exceed 1, and our driver implementation precisely handles that. However, the simple firmware code cannot handle probabilities greater than 1. So, indeed we need two copies of each probability: one that is handled by the driver as is unbounded from above, and one that is reported to the firmware, and is bounded by 1. That is exactly why `Selective-Copy` does not copy P from the shared memory, as it could have less information than the local copy.

The final piece of the puzzle in Algorithm 5 is the `Virtual-MAC-Estimate`

method, which is the driver part of the Virtual MAC. Algorithm 6 shows the pseudocode for this method. In this algorithm, n_{busy}, t_{idle}, n_{total}, and m represent the busy slots, idle time (in μs), total number of slots, and the scaling multiplier (e.g. 1.14) respectively. Also, d_{AIFS} and d_{SLOT} represent durations of AIFS and a time slot respectively.

Algorithm 6 Virtual MAC driver implementation.

procedure VIRTUAL-MAC-ESTIMATE(*store*)
  $n_{busy} \leftarrow store.busy\_slots$;
  $t_{idle} \leftarrow store.idle\_time$;
  Reset *store* values
  if $n_{busy} > 0$ **then** ▷ VMAC is in operation
    $n_{total} \leftarrow n_{busy} + (n_{idle} - n_{busy} * d_{AIFS})/d_{SLOT}$;
    $f_v \leftarrow n_{busy}/n_{total}$;
    Using Bianchi's model [1]:

$$\tau_v \leftarrow 2\frac{1-2f_v}{(1-2f_v)(W+1)+f_vW(1-2f_v^m)}$$

      where W is the minimum contention window;
    return $m\tau_v(1-f_v)n_{total}$
  else
    return ALTERNATE-ESTIMATION-METHOD
  end if
end procedure

6.2.4 Verification

Modifying the firmware and the driver may alter the normal behavior of the wireless adapter in unwanted ways. In order to verify the correct behavior of the updated software, we used the tool described in Appendix A. We used a compliant device in the initial tests to only focus on whether the changes have an impact on anything beyond their scope. For all measurements in this section we used a compliant station sending saturated traffic with payload size of 500 bytes.

First we measure the duration of the frame with the tool. The resulting average over 180 seconds was 603.1, with the expected value being 604.9 (see section A.3.1 for information regarding the computation of the expected value). We also measure the average inter-frame space, which was 15.7 over the three minutes. Although this value differs from the expected value of 16, it was the

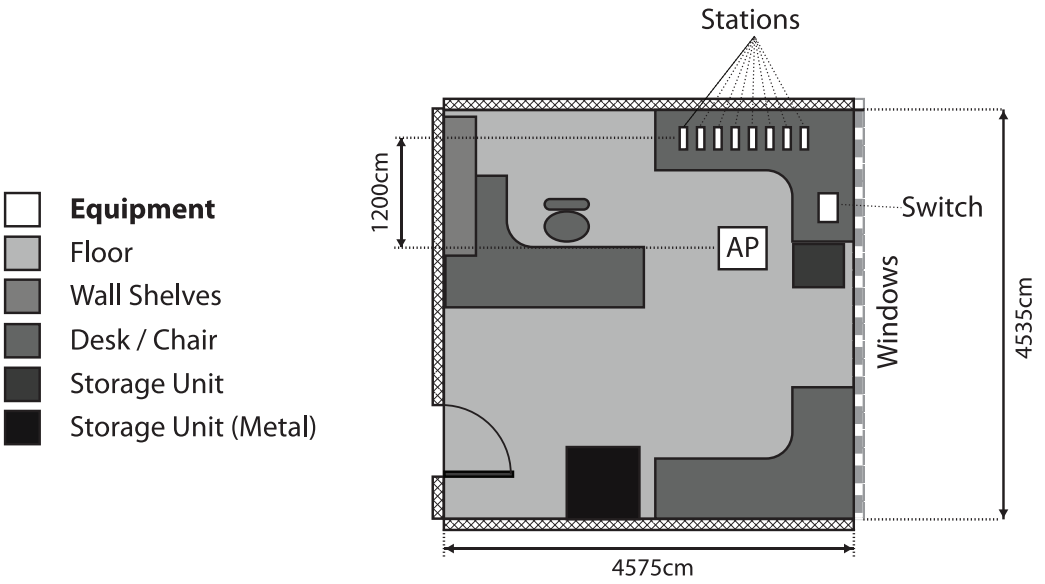


Figure 6.3: Plan of the room where our testbed was situated

same with and without the modifications. Finally, we check the throughput, which is 830.35 packets per second for the original firmware, and 830.33 for our modified firmware. This compares to the expected value of 844.206. According to measurements such as the ones described above, we deduce that the implementation has not had any effect on the normal behavior of the card, and we move on to testing the algorithm itself.

6.3 Experimental Setup

Heaving designed and implemented the policing algorithm, we can now evaluate its performance in a real testbed. In this section we describe the testbed and the environment we used for our experiments. The testbed was set up in an empty office at the end of a corridor on the edge of Hamilton Institute in the Rye Hall building in Maynooth University[4]. Figure 6.3 shows the plan of the testbed room. Although there were some old computer equipment on the desk at the bottom of the picture, the room was empty and dedicated to the tests.

Our testbed consisted of 8 stations, and an AP. For the stations, we use Soekris Engineering Inc.'s net4801 embedded PCs. Each of the stations is equipped with a Atheros AR5001X+ chipset wireless adapter. The operating system

[4]Hamilton Institute has since moved to a new building.

installed on the stations is Debian with kernel version 2.6.32.16, and they use the *ath5k* driver. The *ath5k* driver is modified so the contention parameters it uses can be specified on `modprobe`.

The AP is a Dell Dimension 3100C desktop computer with an Intel(R) Celeron(R) CPU (2.80GHz), and 1 GB of main memory. The AP, too, runs Debian, and it is equipped with two wireless adapters. A Broadcom BCM4318 adapter is used for the policing algorithm, and an Atheros AR5004-based adapter is used for sniffing packets over the channel. The latter is for the sole purpose of debugging, and was not active during tests. For the Broadcom adapter we use the *b43* driver and *OpenFWWF*[67], with modifications presented in the previous section.

The clients use Atheros AR5212 chipset adapters and the `ath5k` driver, which are modified to allow manipulating the MAC parameters by simple commands from the system console. All nodes employ the IEEE 802.11 HR/DSSS physical layer (IEEE 802.11b) and, unless otherwise stated, do not perform rate adaptation. The reason we use IEEE 802.11b is mainly due to the fact that the open-source firmware only works under this protocol. In Chapter 5 we described how the algorithm can be adapted to newer versions of the IEEE 802.11 standard.

Unless otherwise stated, we consider all nodes to be backlogged and to send unidirectional UDP traffic to the AP. In all cases, we measure the performance of the stations when the network is operating with a standard AP and an AP running the policing algorithm configured with the following settings: $\alpha = 0.2$ and $T_{update} = 10$s. Table 6.1 shows an overview of all experiments conducted in this chapter.

6.4 Controller Validation

First we study the impact of four types of attacks that can be easily implemented with current hardware, whereby aggressive MAC settings are used. Specifically, we investigate the scenarios where an attacker seeks to obtain performance benefits by employing more aggressive configurations as follows: *(i)* contending with a CW_{min} parameter half the default value ("CW_{min} Halved"), *(ii)* disabling the Binary Exponential Backoff (BEB) mechanisms while keep-

Parameter	Figures	Comments
MAC protocol		
IEEE 802.11b	All	Fully supported by OpenFWWF
Packet Size		
Fixed (∼MTU)	6.5–6.19, 6.27–6.31	MTU sized packets common, frequently used for evaluation
TCP generated	6.20–6.26	TCP is most common transport protocol
Application generated	6.26	Based on typical current applications
Number of Active Stations		
2–8	all	Small to medium network sizes
Traffic		
Saturated/CBR	6.5–6.15,6.27–6.31	Basic type representing busy station
Specific on/off	6.16–6.17	To show reactiveness of system
Periodic on/off	6.18–6.19	Basic strategy to game policing system
TCP	6.20–6.26	Both long file uploads and shorter transfers
Application	6.26	Specific to selected applications
Station Behaviour		
Compliant	all	Baseline behaviour, also used to verify baseline is not penalised.
Incorrect CW_{min}/ AIFS/CW_{max} /TXOP	6.5–6.25	Adjustable MAC parameters in many drivers.
Rate Adaptation		
Fixed (11Mbps)	6.5–6.26,6.31	Typical rate used for evaluation of 802.11b for evaluation without dynamics
Minstrel	6.27–6.30	Default rate controller for Linux mac80211 layer
PID	6.29–6.30	Other implemented controller for Linux mac802.11 layer

Table 6.1: Summary of Experiments

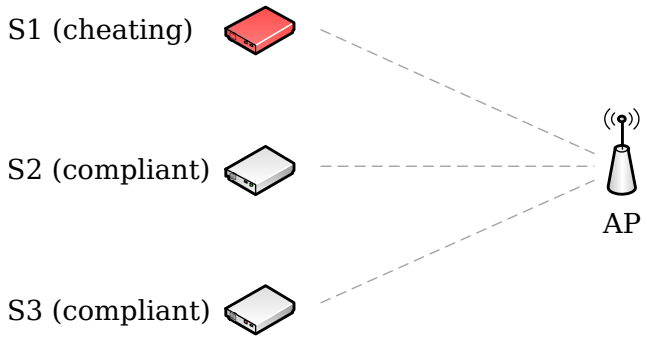


Figure 6.4: Network topology

Figure 6.5: Performance under different types of attacks. Throughput shown above, attempt rate below. Experimental results.

ing a smaller CW_{min} setting ("CW_{min}=CW_{max}"),[5] *(iii)* using a shorter inter-frame space post-backoff ("AIFS = SIFS"),[6] and *(iv)* retaining the access to the medium for 6.413ms by violating the $TXOP_{limit}$ parameter ("Large

[5]Note that compliant devices employ $CW_{max} > CW_{min}$ settings to reduce failure probability upon subsequent attempts, thus being less aggressive.

[6]AIFS $\geq 2\sigma +$ SIFS is the amount of time a station is required to sense the channel idle before entering the backoff procedure. SIFS=10μs is the short inter-frame space. σ is the duration of an idle slot.

TXOP"), thus being able to transmit multiple frames upon a single attempt.

In these scenarios we consider a simple network topology with one attacker sharing the medium with two compliant stations that contend for the channel using the default MAC parameters specified by the IEEE 802.11 standard (i.e. $CW_{min} = 32$, $CW_{max} = 1024$, AIFS = DIFS = 50μs, TXOP = 0). The network setup used for these experiments is depicted in Figure 6.4. Each client is saturated and transmits 1000-byte UDP packets to the access point for a total duration of 3 minutes. We measure the throughput and attempt rate performance of each station under each scenario, with and without the policing algorithm running at the AP, and repeating each test 13 times to compute average and 95% confidence intervals with good statistical significance.

Figure 6.5 shows the throughput and attempt rate attained by each client in each of the scenarios considered, both with and without our policing algorithm running at the AP. To add perspective, we also plot with a dotted line the performance of one station when when all clients behave correctly ("All Compliant"). Observe that an attacker using a smaller CW_{min} attains nearly twice the throughput of complaint stations if not policed, while reducing the throughput and attempt rate of the compliant stations ("CW_{min} Halved", light bars). When we activate the policing algorithm (dark bars), this behavior is effectively counteracted, as our solution equalizes the attempt rates, while the attacker sees its throughput performance reduced. If this attack becomes more aggressive ("$CW_{max} = CW_{min}$", light bars), e.g. the non-compliant station uses a fixed contention window and thus does not backoff upon failures, the policing algorithm rapidly increases the ACK-dropping probability corresponding to that client to 1, thereby disassociating the attacker from the AP. This is reflected in both the attempt rate and throughput performance, which are effectively zero when policing is applied (dark bars).

A more subtle attack could employ a short post-backoff inter-frame space, e.g. the greedy station only waits SIFS before a new attempt, which is the minimum time separating two consecutive frames. Although less significant (since the attacker can sometimes randomly select a large backoff counter and wait more than other stations that wait DIFS plus a short backoff value), the attacker still achieves performance gains to the detriment of the other stations in the network ("AIFS=SIFS", light bars). Once again, if we execute

the policing algorithm at the AP, the transmission attempt rates are equalized.

Lastly, if the attacker transmits several frames upon a single channel access ("Large TXOP"), their throughput performance is significantly higher than that of the compliant stations if no action is taken. In contrast, with our policing algorithm, attempt rates stay equal and the attacker sees their throughput throttled down below the value corresponding to compliant operation.

Let us now take a closer look at the behavior of the controller implemented by our scheme. Specifically, we are interested in validating the convergence of the algorithm under different attacks. For this purpose, we examine the time evolution of the network performance for all scenarios. We begin by the case where all stations are compliant. In our tests, not a single frame was dropped by the AP, although p was slightly increased at some points. Figure 6.6 shows this change. The error bars are calculated over thirteen experiments. Their large size shows that there are only random spikes of slightly positive p. The reason for these spikes is that, although we overestimate the compliant attempt rate, the virtual station is still contending with stations that use random backoff. Hence there are isolated instances that those stations get a slightly higher share of the channel than the virtual MAC.

Further, Figures 6.7, 6.8, 6.9, and 6.10 show throughput and attempt rate for the attacker and a compliant station, as well as the penalty applied by the algorithm. Most of these cases converge in 5 to 6 iterations. We can reduce the convergence time by either using a shorter iteration time (10s is relatively high, and is used only to reduce randomness in the graphs), or a larger α value, which determines the rate of adaptation.

In most cases, observe that the policing algorithm successfully brings the attempt rate of the attacker down to that of a compliant client (middle graph), while their throughput is reduced (top graph). An exception to this is the CW_{min}=CW_{max} case (Figure 6.8), where the penalty is increased much further than 1, and the station can no longer get any frames through. This is a good example of the case where we have an option to disassociate the station. However, we don't do that in our tests and. The result is that the station still sends some traffic bursts every once in a while, which fail due to the high p value.

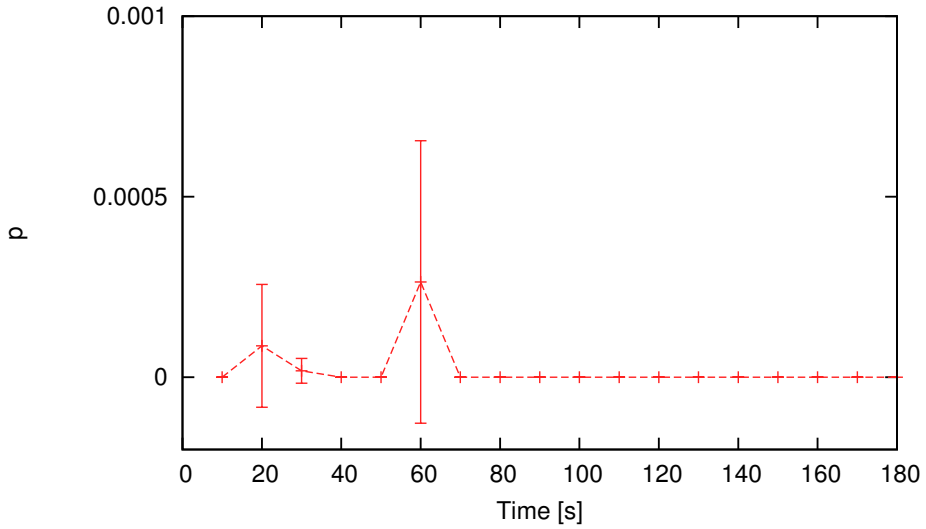


Figure 6.6: Time evolution of penalty when all stations are compliant.

What is also important to remark is that the algorithm is close to convergence after a few steps. Based on the results, within 5 iterations, p is within 10% of its long-term value. Convergence time is shorter for more aggressive attacks (i.e. with manipulated TXOP), and it can be further reduced by choosing a larger α parameter.

6.4.1 False Alarms

Further, we verify that our algorithm does not unnecessarily penalize compliant stations, i.e. does not trigger false alarms, due to the channel access randomness inherent in 802.11 DCF. Results we discussed previously show little or no penalty for compliant stations.[7] To put this in perspective, we examine the time evolution of a station's attempt rate, the maximum achievable attempt rate estimated by our algorithm, and the penalty applied to each client. We investigate these with the same network settings (three backlogged stations) in two scenarios, namely all stations compliant and respectively one of them misbehaving with a CW_{min} half the default value. As we show in Figure 6.12, our estimate closely follows the actual performance attainable by a compliant client, and consequently the penalty applied to these exhibits only small variations above zero. To put things in perspective, we plot a 0.02

[7] In all experiments in this chapter, the controller adds 14% to the Virtual MAC estimation. See Section 5.3.2 for the explanation of this amount.

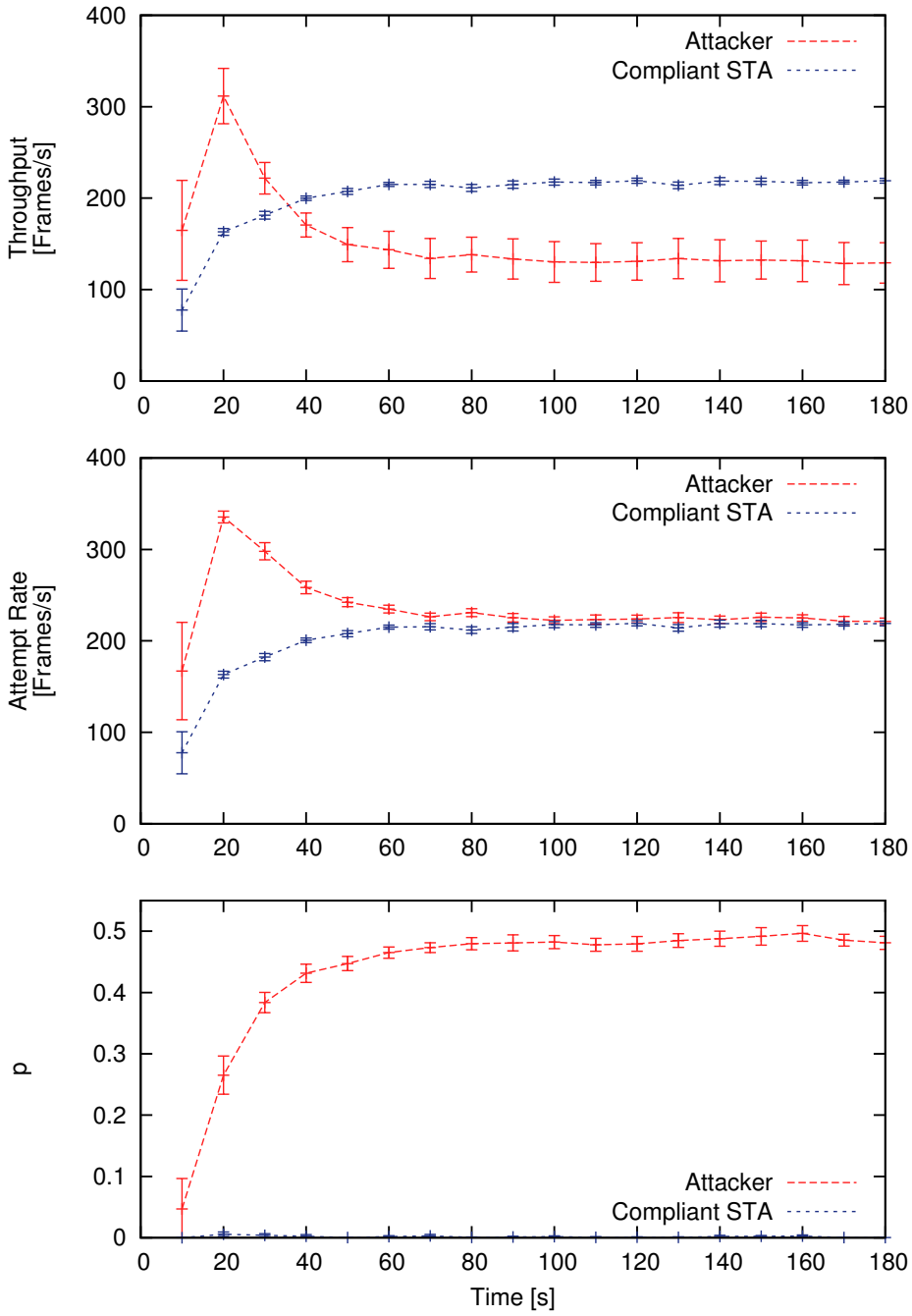


Figure 6.7: Time evolution of throughput, attempt rate, and penalty for a compliant and a non-compliant station (CW_{min} halved).

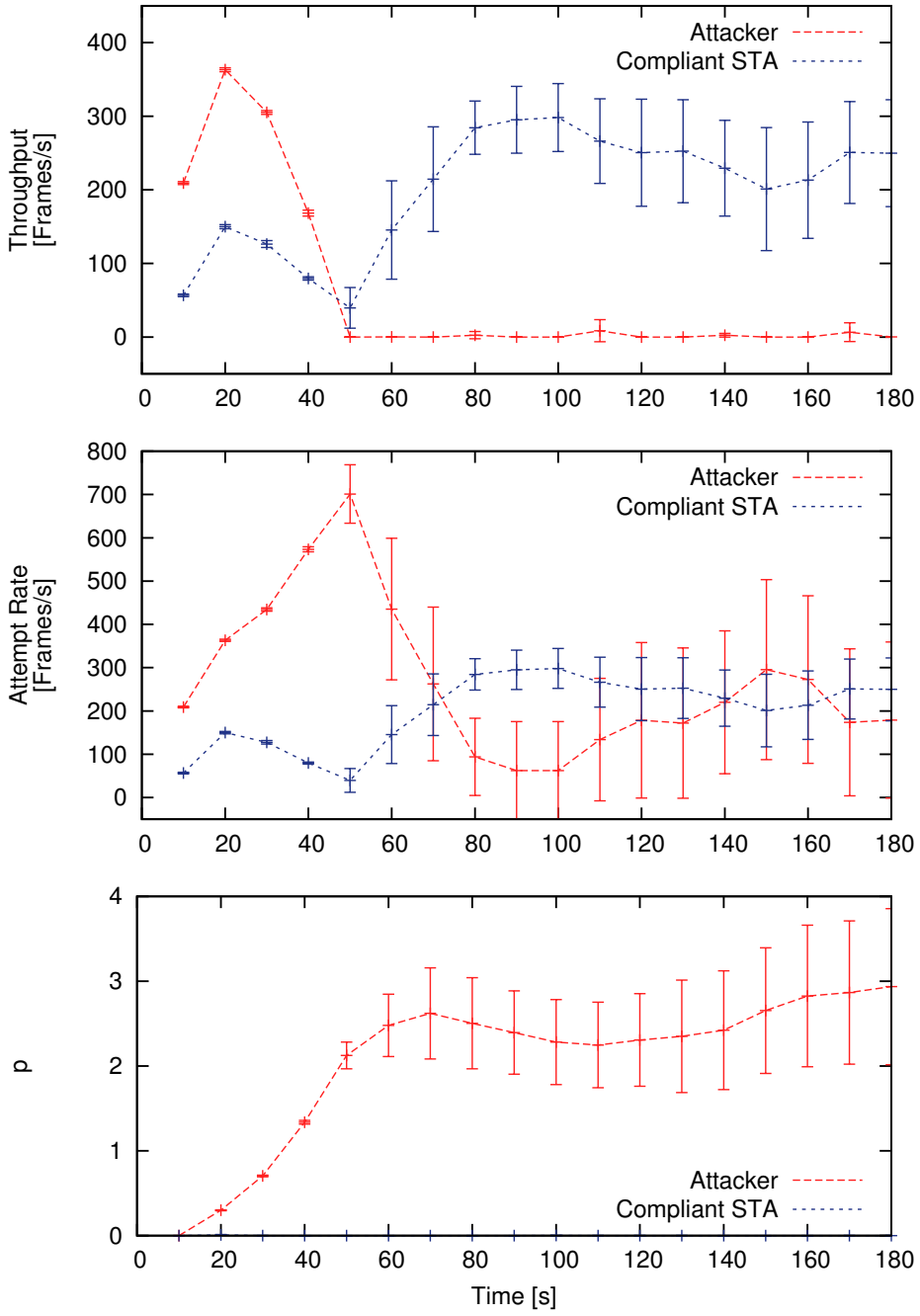


Figure 6.8: Time evolution of throughput, attempt rate, and penalty for a compliant and a non-compliant station ($\mathrm{CW}_{\mathrm{min}}$ halved and $CW_{\mathrm{max}} = CW_{\mathrm{min}}$).

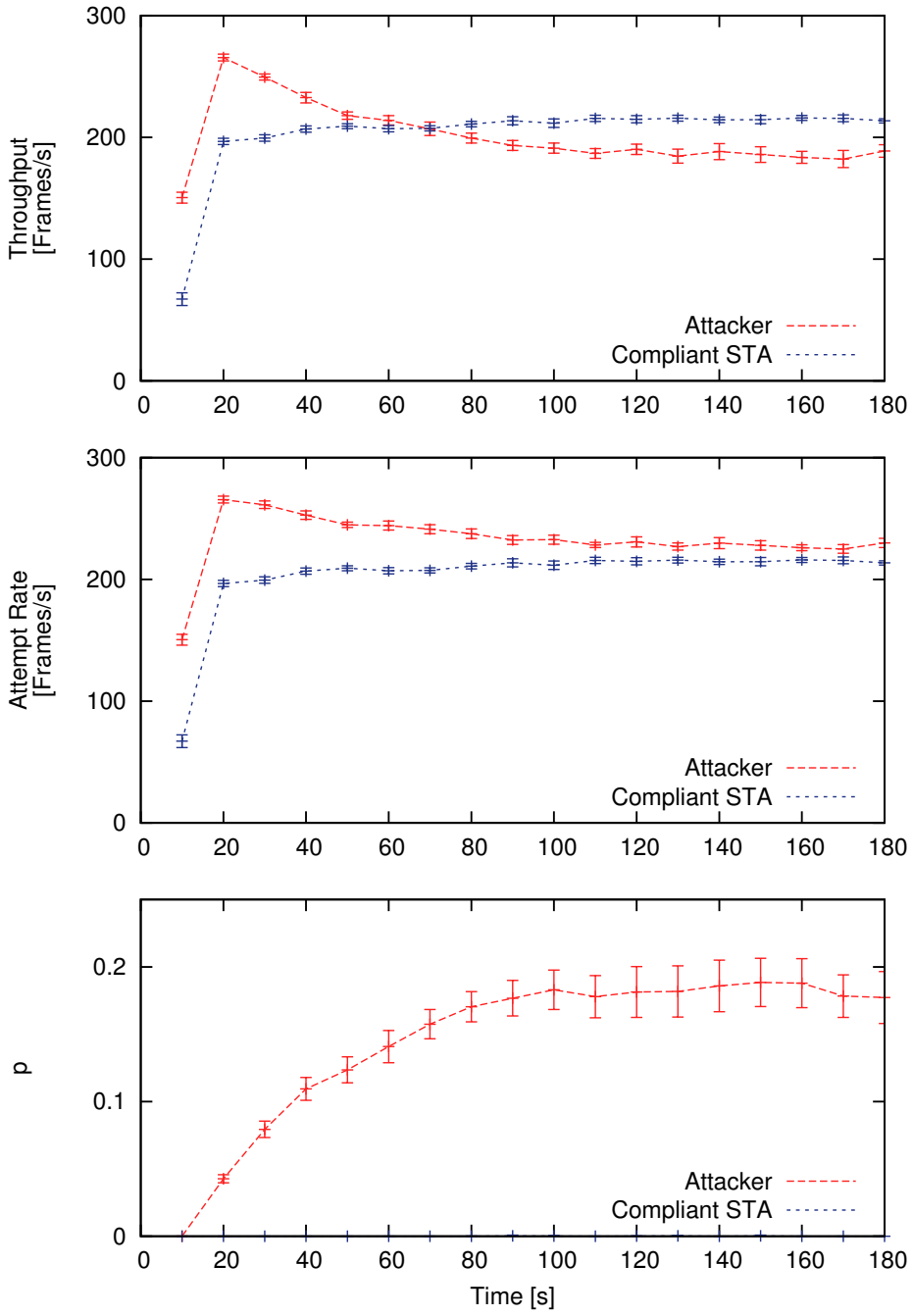


Figure 6.9: Time evolution of throughput, attempt rate, and penalty for a compliant and a non-compliant station (AIFS = SIFS).

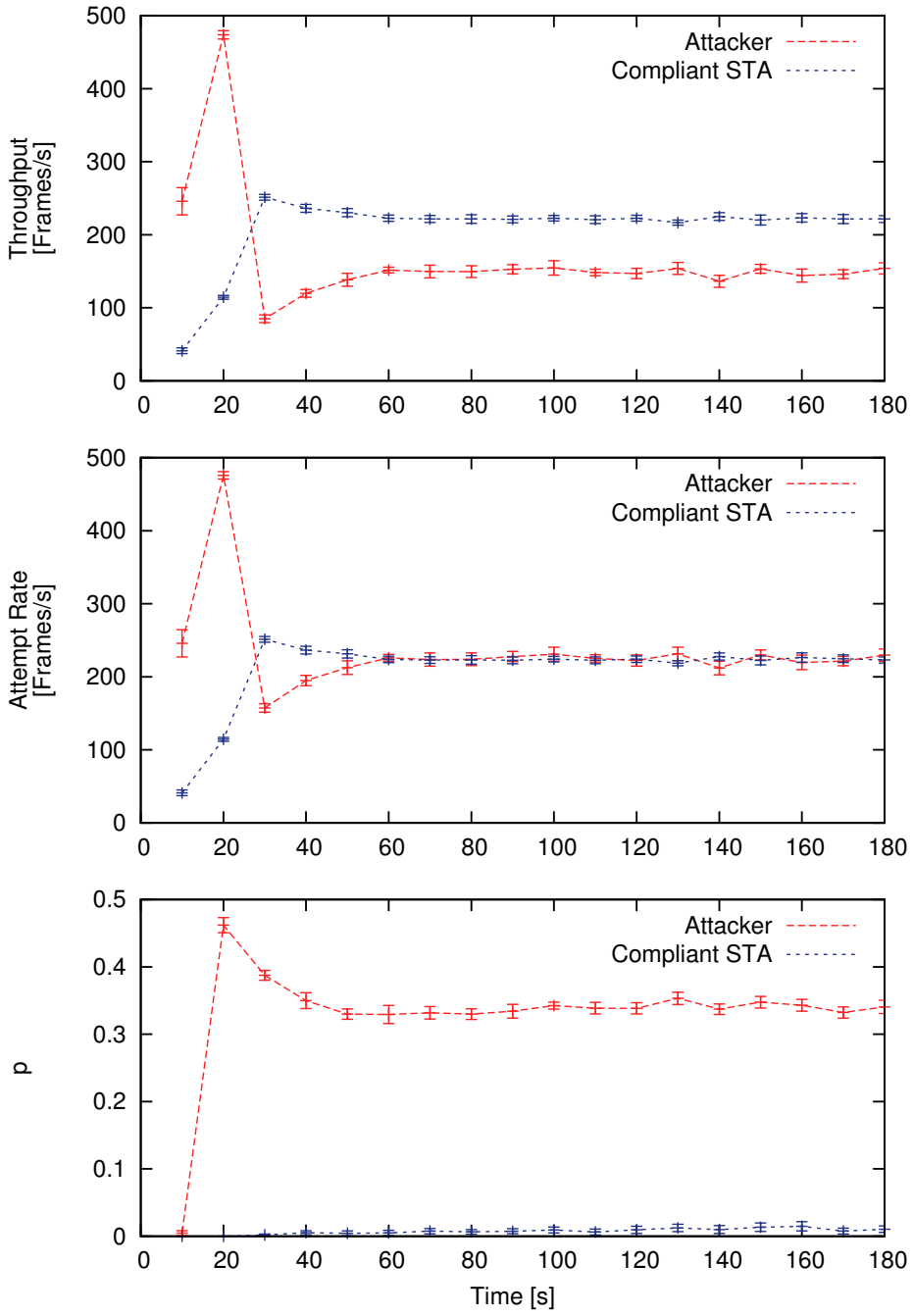


Figure 6.10: Time evolution of throughput, attempt rate, and penalty for a compliant and a non-compliant station (TXOP = 6.413 ms).

penalty threshold and confirm that the percentage of times the penalty applied to compliant clients exceeds this value is zero in all scenarios.

6.4.2 Impact on Network Throughput

While the policing algorithm is designed to enforce standard compliance and ensure the fair channel access provided by the standard, it does have a negative effect on overall airtime utilization. In this section we study this impact and its implications. We use the experiments presented in Figure 6.7, when a non-compliant station uses half the standard minimum contention window.

Figure 6.11 shows the effect of policing on network utilization. It compares two scenarios, (i) the original scenario where there is one misbehaving and 2 compliant stations, and (ii) where all three stations are compliant. As shown in the top graph, the network throughput decreases when we begin policing the misbehaving station, and this decrease brings it to a lower value than when all stations behave normally. The reason is that the AP drops ACKs for more frames as p increases, but those frames still take up channel time.

However, the middle graphs shows that compliant stations still return to what they would get in the all-compliant case after a few iterations of the algorithm (within 5 iterations, the error bars overlap), so this network degradation does not impact compliant stations at all. To understand where the extra throughput goes, note the throughput evolution of the non-compliant station in the same graph. This station gets a penalty that reduces its throughput to a much lower value than if it was compliant. The high penalty value helps put the expense of lost channel time solely on the non-compliant station. Therefore, while policing does impact network throughput, this only affects non-compliant stations. This is why we use attempt rates instead of throughput or successful frame count, as we discussed previously in Section 5.2.3.

6.5 Impact of Other Stations

Increased network size affects the estimate provided by the Virtual MAC as well as the impact of non-compliant nodes on compliant ones. In the previous chapter, we provided proof that in fact the estimate is improved as the number of stations in the network increases. In this section we are interested in the

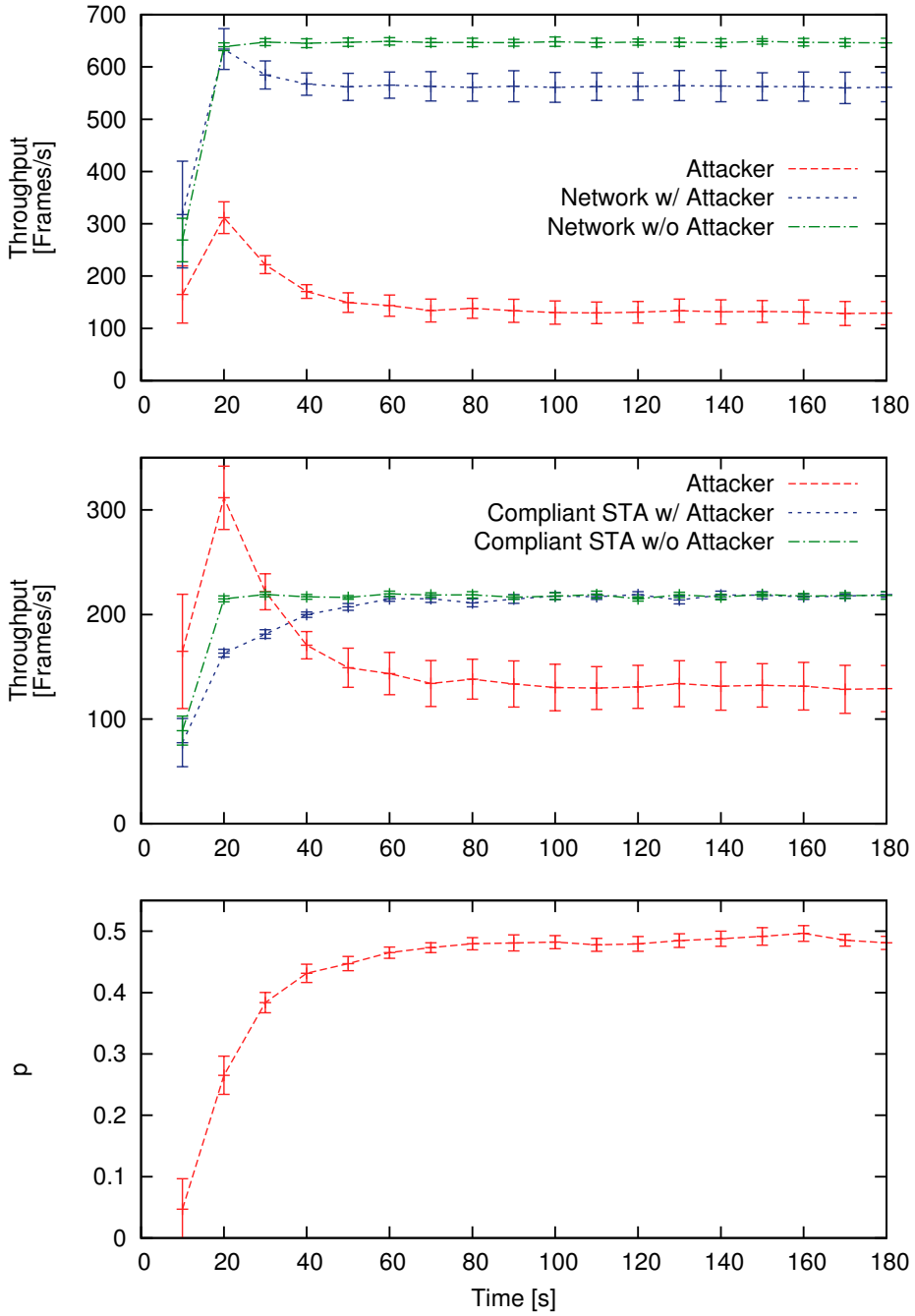


Figure 6.11: The impact of policing algorithm on network throughput in practice, comparing network throughputs when non-compliant stations are present and absent (top), throughput of a compliant client in the presence and absence of a non-complaint one (middle), and the evolution graph for p when in the presence of a non-compliant station (bottom). The non-compliance is question is choosing half the standard $\mathrm{CW}_{\mathrm{min}}$ value.

impact of network size on the performance of the policing algorithm as a whole, when the network is a mix of attackers and compliant stations.

6.5.1 Hiding in the Crowd

We now investigate whether an attacker could "hide in the crowd" as the number of network users increases. For this purpose, we consider a network with one non-compliant station employing a CW_{min} based attack and we vary the number of compliant stations while we examine the performance of both. In each case, all clients are backlogged and send 1,000-byte packets for a total duration of 3 minutes. We repeat each experiment 13 times and compute again average and 95% confidence intervals for the attempt rate and throughput obtained by each station.

In Figure 6.13 we show the throughput and attempt rate of the attacker and a compliant client, with a standard AP as well as with an AP executing our algorithm. Observe that the performance of the attacker decreases as the network size increases from 2 to 8 STAs (to be exact, from the average of 464 frames per second for the smallest network to 165 for the largest), but is always significantly above that of a compliant client if no action is taken to counteract the greedy behavior. In contrast, when the AP runs the policing algorithm, the attempt rate of the attacker never exceeds that of a compliant client (observe the overlapping lines in the top sub-figure), while their throughput performance falls below that of compliant clients in all circumstances.

We can conclude that the network size does not impact the performance of the policing algorithm, which effectively penalizes attackers even in denser topologies.

6.5.2 Multiple Attackers

In what follows, we study the performance of the proposed policing algorithm when multiple attackers are present in the WLAN. Here, we aim to understand whether the presence of a large number of attackers could influence the penalty update of our algorithm. We demonstrate that, despite its prevalence, such behavior will not be regarded as compliant by the proposed policing scheme. We use the same methodology as in the previous subsection, running 3-minute tests for each experiment and getting 10 measurements for each case in order

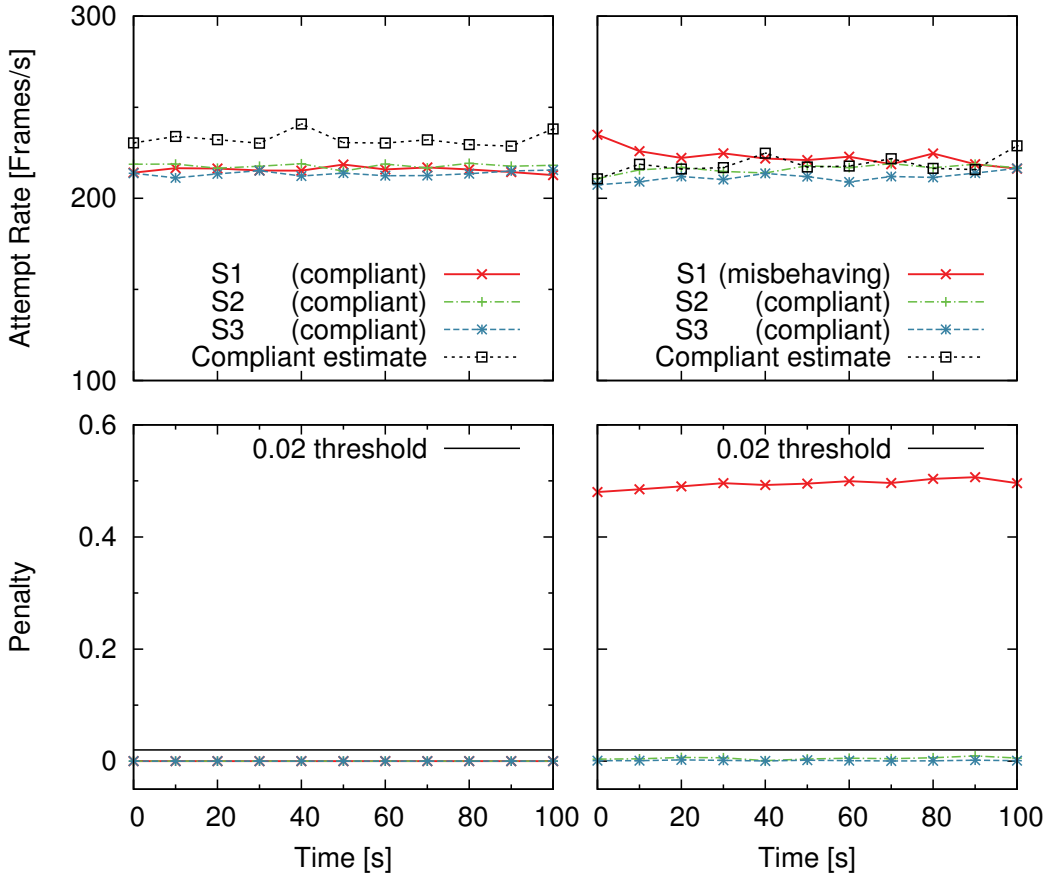


Figure 6.12: WLAN consisting of three saturated stations. The AP runs the proposed policing scheme ($\alpha = 0.2$). Time evolution of the attempt rate and compliant rate estimate (top), and penalty applied (bottom) when all clients are compliant (left), respectively one employs a $\mathrm{CW}_{\mathrm{min}}$ of half the default value.

to measure the performance of both compliant stations and attackers in terms of attempt rate and throughput.

First let us consider the case where only one station is compliant and increase the number of attackers present in the network. The results of these experiments are depicted in Figure 6.14, where we plot the attempt rate and throughput of the compliant station and one attacker, with and without the policing algorithm running at the AP. We observe that also in this setting, the policing algorithm equalizes the attempt rate of all stations while the throughput performance attained by attackers is reduced.

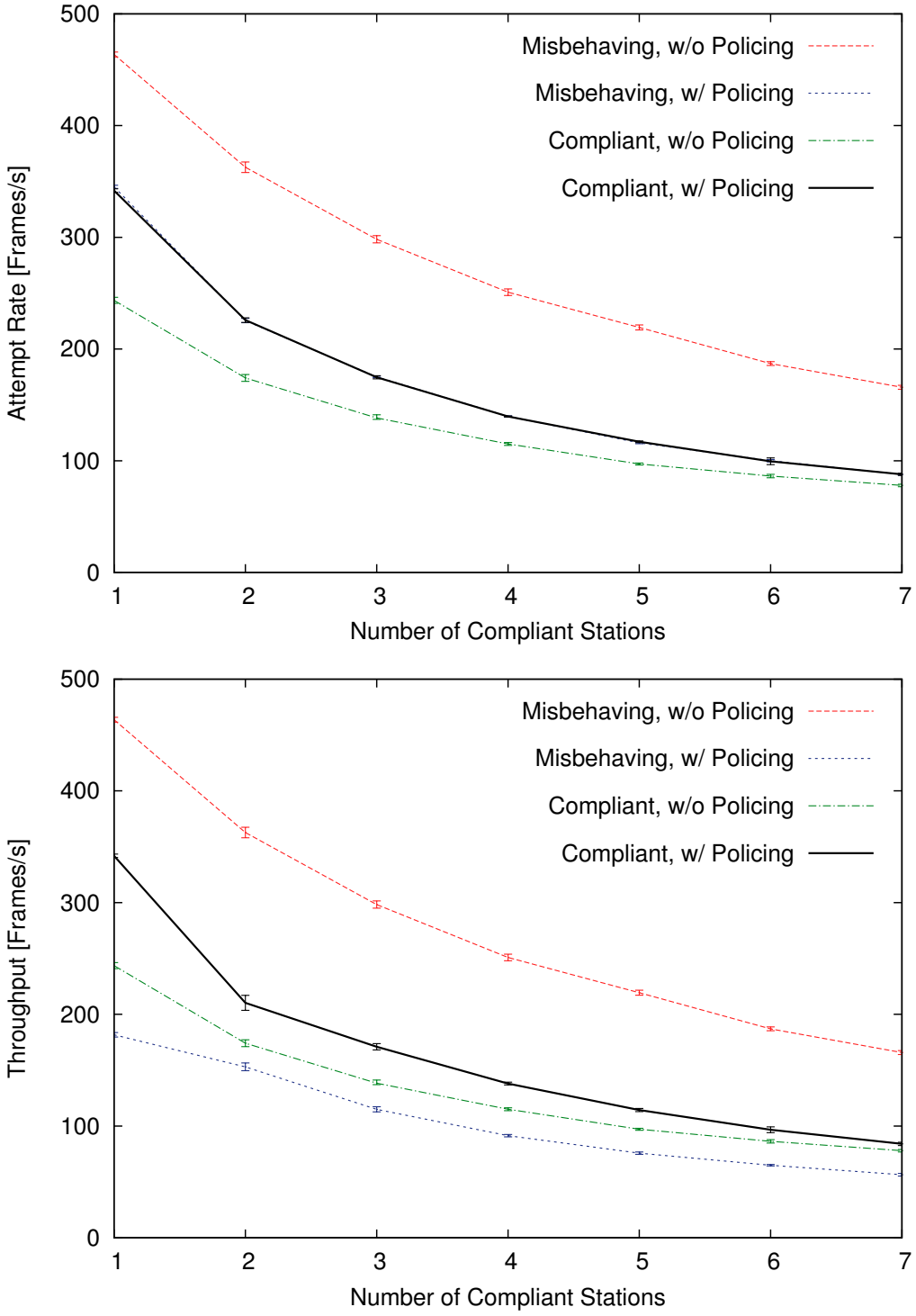


Figure 6.13: Attempt rate and throughput as the network size changes. The non-compliant station employs a CW_{min} of half the default value. Experimental results.

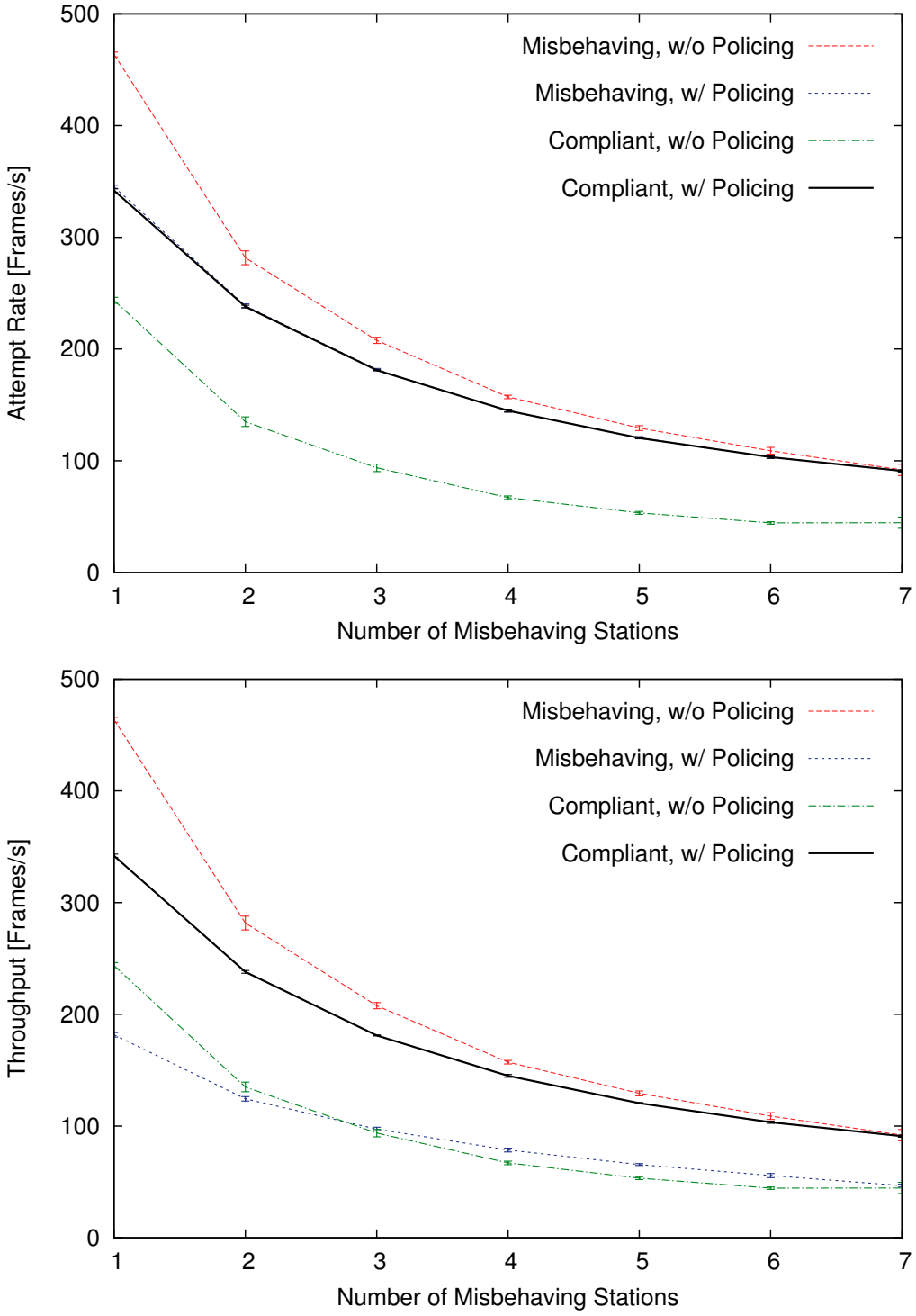


Figure 6.14: Attempt rate and throughput vs. number of attackers. The non-compliant station employs a CW_{min} of half the default value. Experimental results.

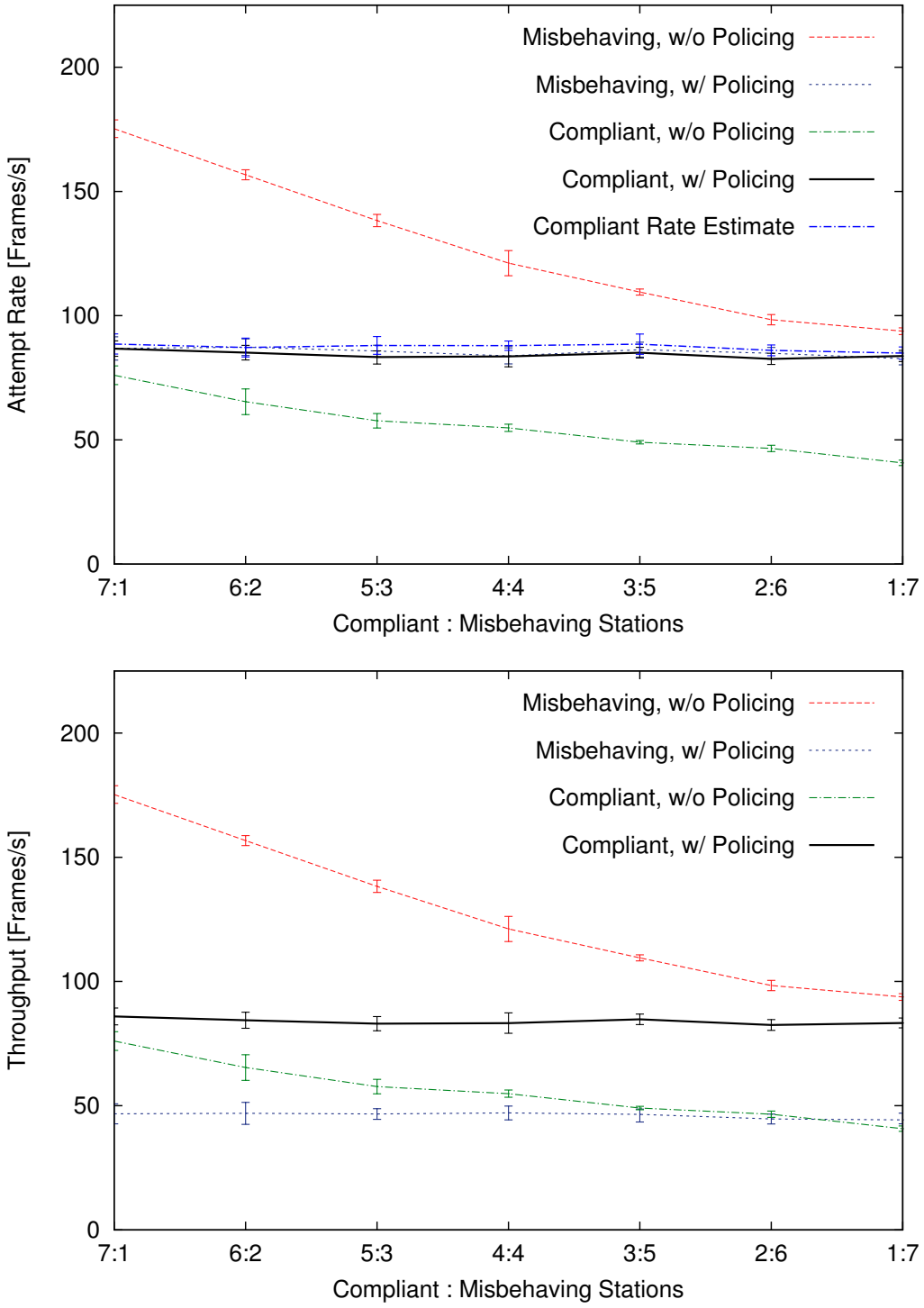


Figure 6.15: Attempt rate and throughput for different compliant/attacker station ratios. Experimental results.

6.5.3 Fixed Network Size

So far we have examined the effect of increased network size, when additional nodes are either non-compliant or compliant stations. We now take a step closer to fully isolating the type of stations added from the network size. To this end, we examine a network with a fixed number of clients ($n = 8$) and vary the proportion of compliant and misbehaving stations. The performance of one client within each category is shown in Figure 6.15, which further confirms the effectiveness of our proposal in the presence of several attackers. Notice the equalized attempt rate, and the difference (of 40 frames per second on average) between throughputs of a compliant station and that of an attacker when policing is applied.

6.6 Dynamic Network Conditions

We next consider two scenarios to demonstrate the effect of network dynamics on the behavior of the policing algorithm. Our goal here is twofold: *(i)* verify that our proposal adapts quickly to changes in the network topology, and *(ii)* demonstrate the algorithm carries forward the penalty of selfish users when they leave the network. The access-point runs the policing controller, for which measurement and parameter adjustment iterations are 5 seconds apart.

In the first experiment, two compliant stations connect to the WLAN and start transmitting to the AP at $t = 0$s. After 100s, a misbehaving station (S3) joins the network, contending with a $\mathrm{CW}_{\mathrm{min}}$ parameter half the standard value. At $t = 200s$ another standard-compliant station (S4) connects to the WLAN. Finally, S3 leaves the network after transmitting for 200s and S4 disassociates 100s later.

The result of this experiment is depicted in Figure 6.16 where we plot the time evolution of the attempt rate, throughput and penalty corresponding to each client. We can see clearly that our algorithm quickly detects and starts penalizing the misbehaving station, equalizing the attempt rates in 5 iterations. As the fourth client joins, our solution re-estimates the maximum achievable attempt rate and continues penalizing the selfish user, without affecting the performance of the new station. Lastly, as the non-compliant station leaves the network, the penalty is preserved and carried forward to be applied when this client reconnects. Thus we confirm that the performance of our algorithm

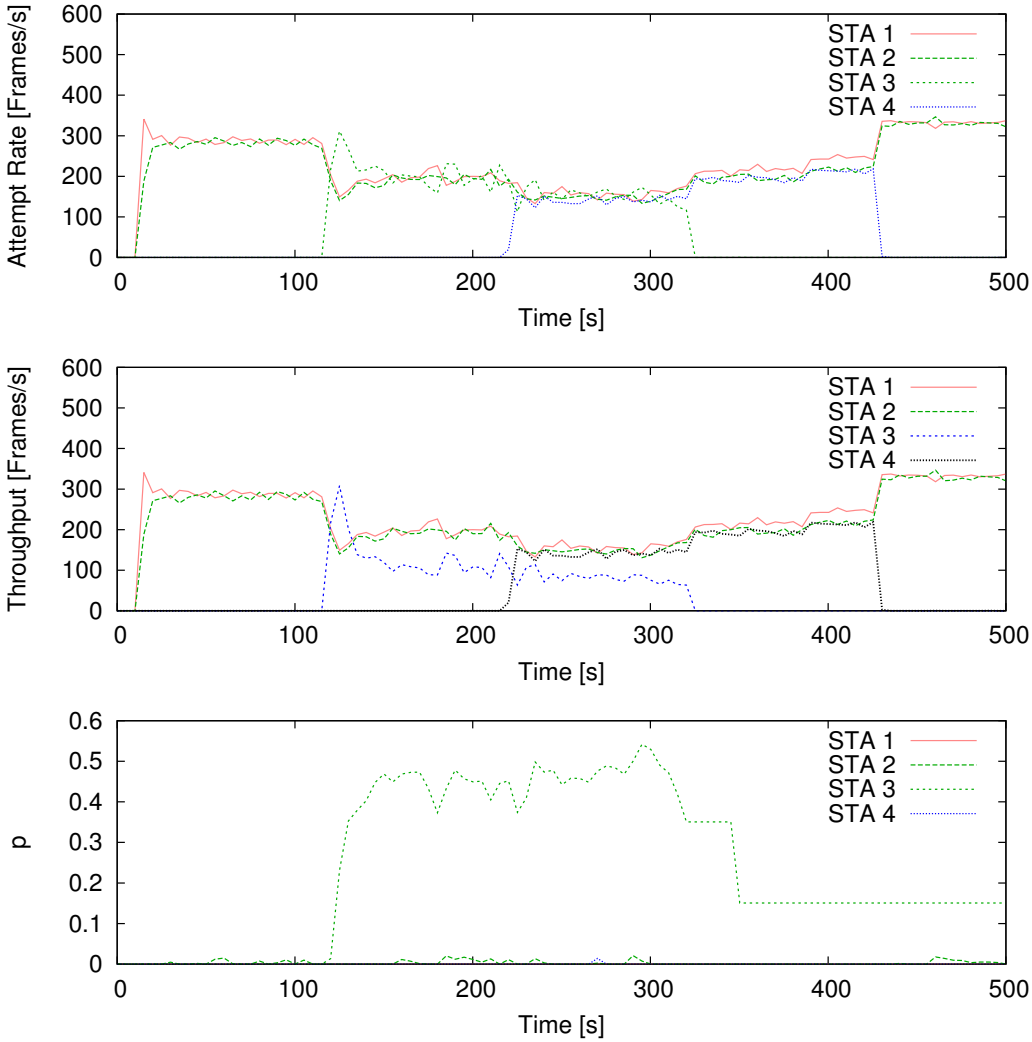


Figure 6.16: Scenario 1: two compliant stations are joined by a misbehaving one (CW_{min} half the default value) and subsequently by a third compliant client. Stations S3 and S4 transmit for 200s each and then leave the network. The AP runs the proposed policing scheme. Time evolution of the attempt rate (top), throughput (middle) and penalty applied by the proposed policing algorithm (bottom) for each client. Experimental data.

is not affected by network dynamics and penalties are successfully carried forward.

In the second experiment, we have 3 standard-compliant stations transmitting saturated traffic at 11 Mb/s. At time 100, station 4 associates to the network and starts transmitting saturated traffic with half the standard contention window. The policing algorithm adjusts the penalty accordingly to equalize

S4's attempt rate with the compliant value. The penalty for behaving stations is always around zero, with little variation caused as a result of real-time estimation. Even so, the probability is always below 0.05.

After 100 seconds of transmitting, station 4 disassociates and re-associates with correct parameters. The penalty is maintained while this is happening (taking about 10 seconds) and the station is still penalized even after it joins with good behavior. But soon as the compliant behavior is observed, the penalty is reduced to zero in a few iterations (5 to be exact). Following 100 seconds of standard behavior, the station is reconfigured again not to comply and re-associates. Fig 6.17 shows the response of the policing algorithm to these changes.

6.6.1 Non-compliant station with Bursty Traffic

In the experiments presented so far, all the contenders, whether compliant or non-compliant, transmitted saturated traffic. Indeed misbehavior becomes problematic under heavy network loads, since the performance of compliant users suffers as a result of the gains achieved by the non-compliant clients. However, it is also useful to verify that our algorithm can detect misbehaving clients that transmit on/off (bursty) traffic, since intuitively the average attempt rate of these might fall below the expected maximum compliant value.

We note that the robustness analysis described in Section 5.2.5.2 guarantees that no transmission strategy can game the operation of the policing algorithm, though verifying this in practice with such bursty traffic is a useful demonstration. To this end we conducted additional experiments where a misbehaving client alternates periodically between silent and active periods, while sharing the network with two complaint stations. Figure 6.18 shows results when these periods are 10s long, and Figure 6.19 shows them when they are 20s long.

These results demonstrate that the algorithm reacts quickly to such bursty traffic, noticing misbehavior within one iteration, and recognizing its bursty nature with a gradually increasing p over time. Further, the selfish user does not gain any long-term throughput advantage from employing this strategy (i.e. the mean throughput for the misbehaving stations in these experiments is less than that of compliant stations, as we illustrate in the middle sub-plots

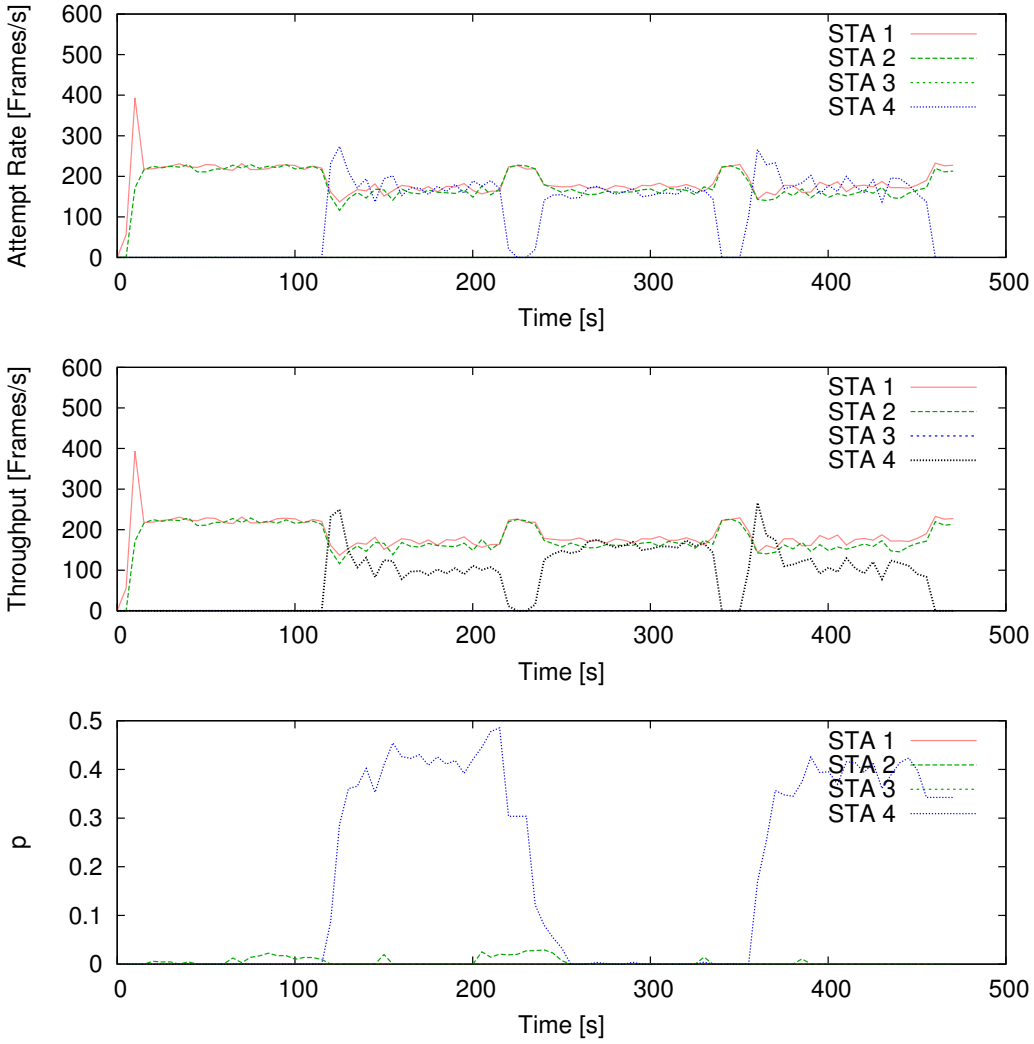


Figure 6.17: Scenario 2: the effect of changes in behavior on policing. Experimental data.

with dashed lines). The results confirm that the extended policing scheme is robust to selfish users generating bursty traffic, as the algorithm detects rapidly their deviation from compliant behavior and penalizes them accordingly.

6.7 Real Traffic

So far we have only tested the policing algorithm mostly for constant bitrate (CBR) traffic. In this section we demonstrate the performance of the policing algorithm in more realistic scenarios. First we consider a scenario where

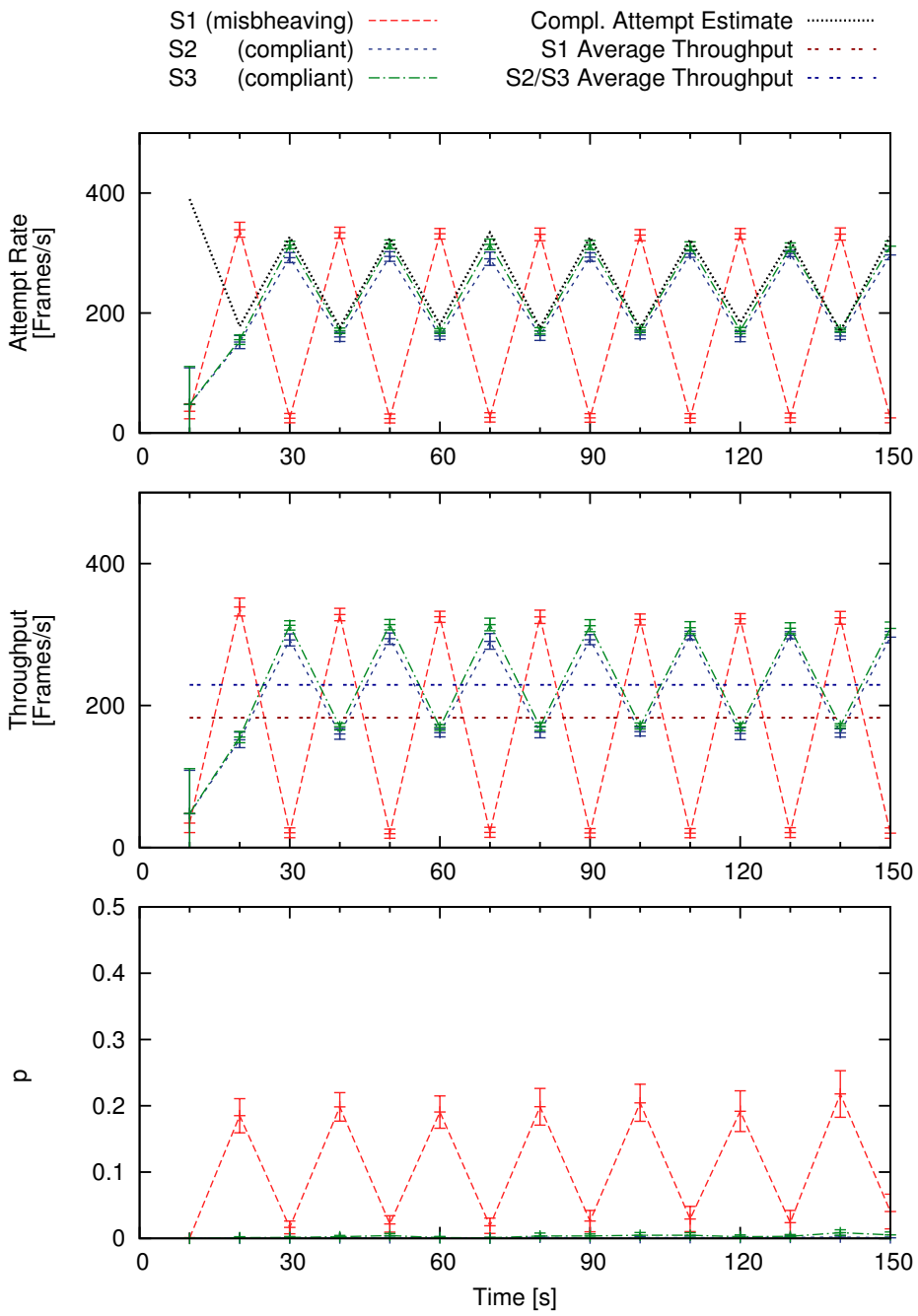


Figure 6.18: WLAN with three client stations. S1 transmits on/off traffic, alternating between silent and active periods of 10 seconds with CW_{min} halved. S2 and S3 always have packets to transmit. The AP runs the proposed policing scheme ($\alpha = 0.02$). Stations' attempt rates and the maximum achievable compliant attempt rate as estimated by the algorithm (top), instantaneous and average throughputs (middle), and penalties applied to each client (bottom).

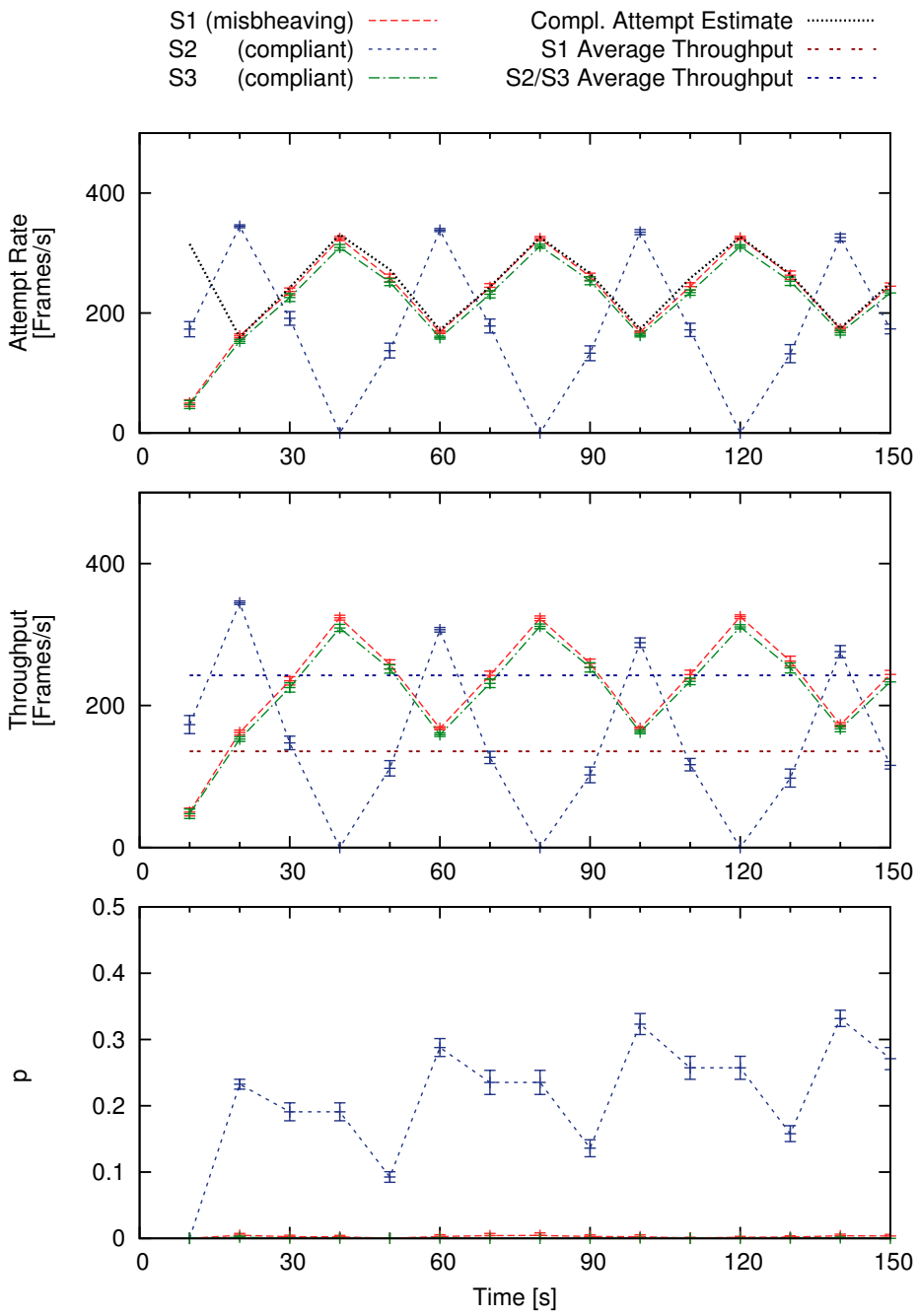


Figure 6.19: WLAN with three client stations. S1 transmits on/off traffic, alternating between silent and active periods of 20 seconds with CW_{min} halved. S2 and S3 always have packets to transmit. The AP runs the proposed policing scheme ($\alpha = 0.02$). Stations' attempt rates and the maximum achievable compliant attempt rate as estimated by the algorithm (top), instantaneous and average throughputs (middle), and penalties applied to each client (bottom).

stations upload files to a remote server, and then we consider one with mixed traffic.

6.7.1 FTP File Upload

File upload is a good example of real uplink traffic on the Internet. It is also a valuable test for the policing algorithm as it is a non-CBR traffic over TCP. So, we can study the effect of policing on TCP's congestion control. In the following tests we have two compliant stations and one non-compliant station which is the same network as that of the previous tests. All these stations send a very large file to a remote server[8] through FTP. We run each test 13 times and plot the results with error bars (95% confidence). Before the tests, we pinged the server 1000 times with each station, and the average round-trip time (RTT) was $87.3 \pm 0.1ms$ for all stations, without any loss.

First we examine a mild attack by the non-compliant station, namely AIFS = SIFS, which we know is contained at around $p = 0.18$ through results presented in Figure 6.9. Figure 6.20 shows the time-evolution of attempt rate, throughput, and ACK-dropping probability for this test. Notice that there is no noticeable change in the performance of any of the stations, and Figures 6.9 and 6.20 are very similar. The throughputs (and attempt rates) are less in Figure 6.20 due to the different nature of the test, which involves sending TCP packets to a remote server, rather than CBR UDP datagrams to a nearby host. The unchanged behavior means that the probability is low enough not to affect TCP's congestion control.

Next, we try a more aggressive attack, namely CW_{min} halved. This leads to a higher p for the non-compliant station, and as you can see in Figure 6.21 it makes the time evolution graph quite variable. You can identify the effect of the policing algorithm on TCP congestion control for the non-compliant station: when the probability gets high enough, the chances of losing TCP packets because of reaching the IEEE 802.11 retry limit increases, and once a loss happens, we see a large drop in the throughput of the non-compliant station. As a result, p also decreases in the next step, reducing chances of TCP packet loss again. That is why we see an oscillatory pattern in the plots. This pattern is even more visible in Figure 6.22. Both plot stacks contain

[8] annahid.com

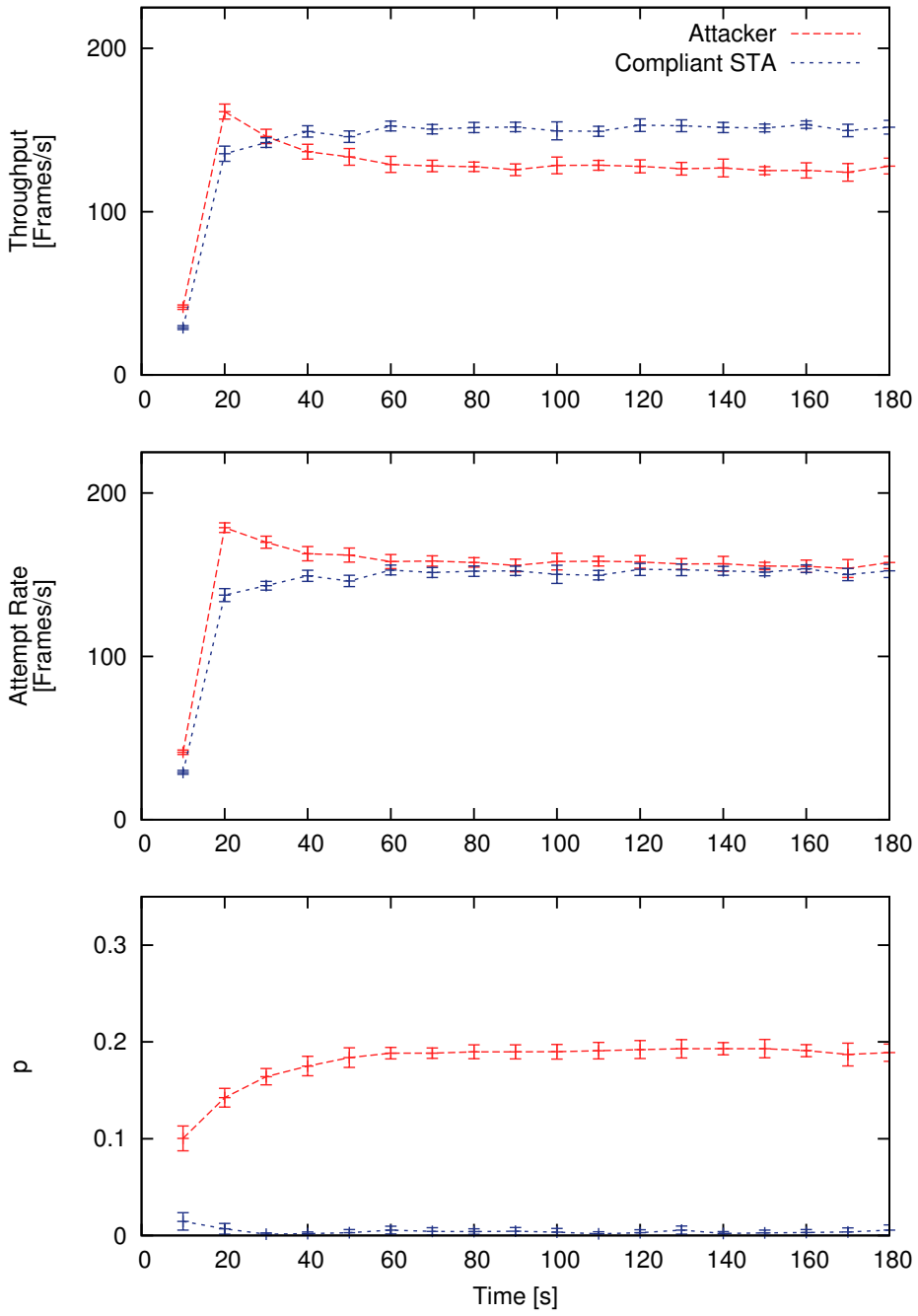


Figure 6.20: Time evolution of throughput, attempt rate, and penalty for a compliant and a non-compliant station (AIFS = SIFS), all sending FTP traffic (over TCP). Experimental results.

averages over all runs, and although the pattern stays the same, exactly when the packet loss occurs may differ from one test to another, and that is why we see large error bars. To get a better view of this effect, Figure 6.23 shows a single run that contributes to Figure 6.22. Note the large drop in throughput (and attempt rate) whenever p reaches approximately 0.4.

To compare the performance of the network in scenarios with non-compliance, we have also run tests in the same network where every STA complies with the standard. Figure 6.24 shows the results. We have plotted the station that cheats in the other tests with a different line to demonstrate that it is neither punished, nor gets a higher throughput if it is complying with the standard. Comparing attempt rate and throughput values in this figure with those previously discussed, we see little to no difference between that of the compliant station in the non-compliance scenarios and that of these stations. To aid comprehension Figure 6.25 shows the throughput and the attempt rate of a compliant station in all scenarios we just described. Error bars are omitted for the compliant case so as not to make the plot too crowded. Notice that most error bars overlap with the compliant line itself (with momentary exceptions such as $t = 30$). This means that the combination of the policing algorithm and TCP's congestion control mechanism does not have a negative impact on the compliant station, even with the presence of an aggressively non-compliant station.

6.7.2 Mixed Traffic

Next, we demonstrate the performance of the policing algorithm in another realistic scenario with heterogeneous traffic. More specifically, we consider a network with $n = 4$ clients, the first one uploading a large file, the second generating web traffic, the third streaming a video file and the last performing a system update. All stations are standard-compliant. Our goal here is to verify that the policing algorithm will not unnecessarily penalize compliant clients that have higher demands and attain higher transmission rates simply due to the reduced activity of the other contenders.

To emulate the file upload, we generate saturated traffic using `iperf` on the first client. The second station establishes finite size TCP connections, alternating between periods of activity, during which a 2 Mbyte file is transferred, and silent periods exponentially distributed with mean $\lambda^{-1} = 60$s [142]. The

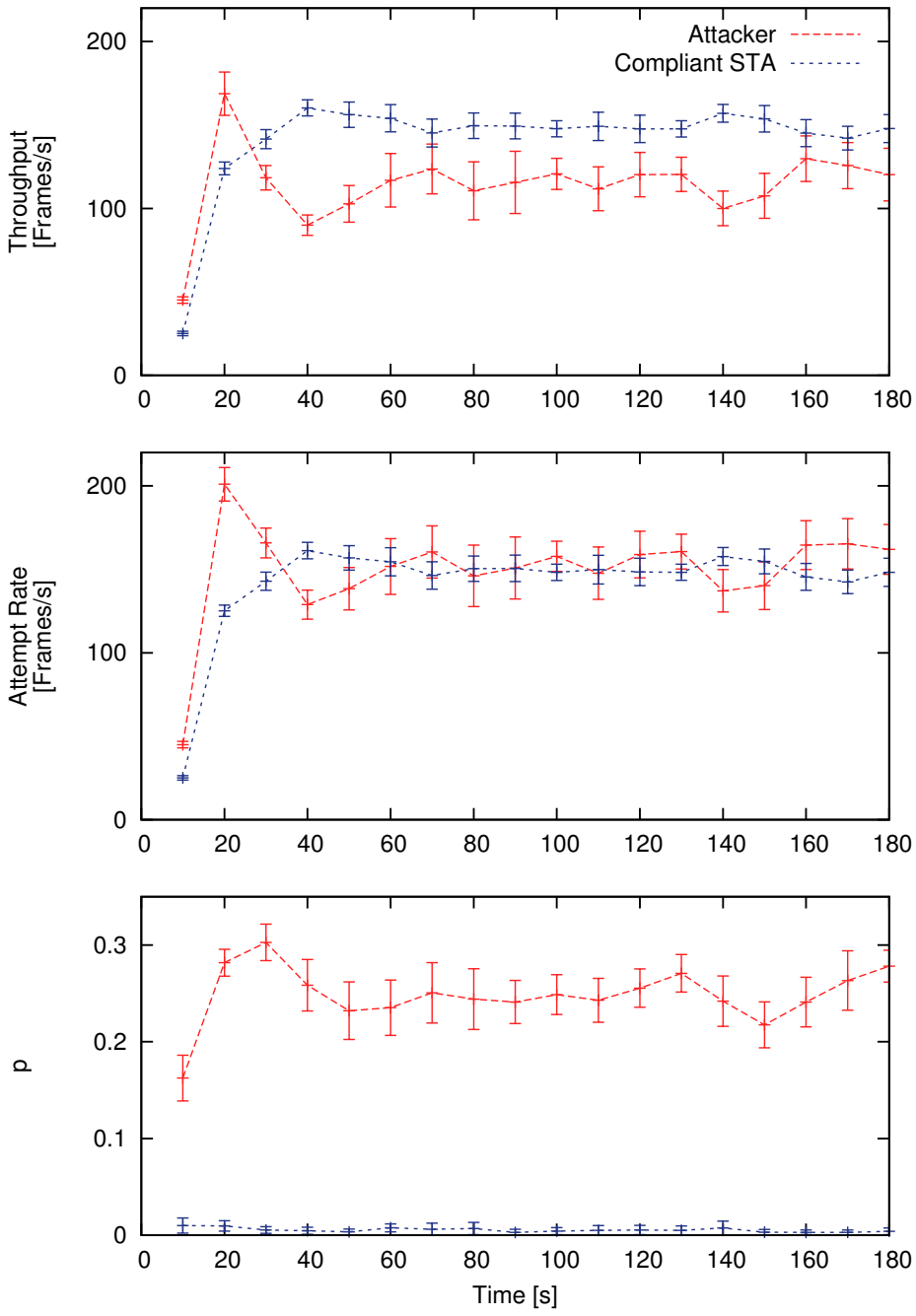


Figure 6.21: Time evolution of throughput, attempt rate, and penalty for a compliant and a non-compliant station (CW_{min} halved), all sending FTP traffic (over TCP). Experimental results.

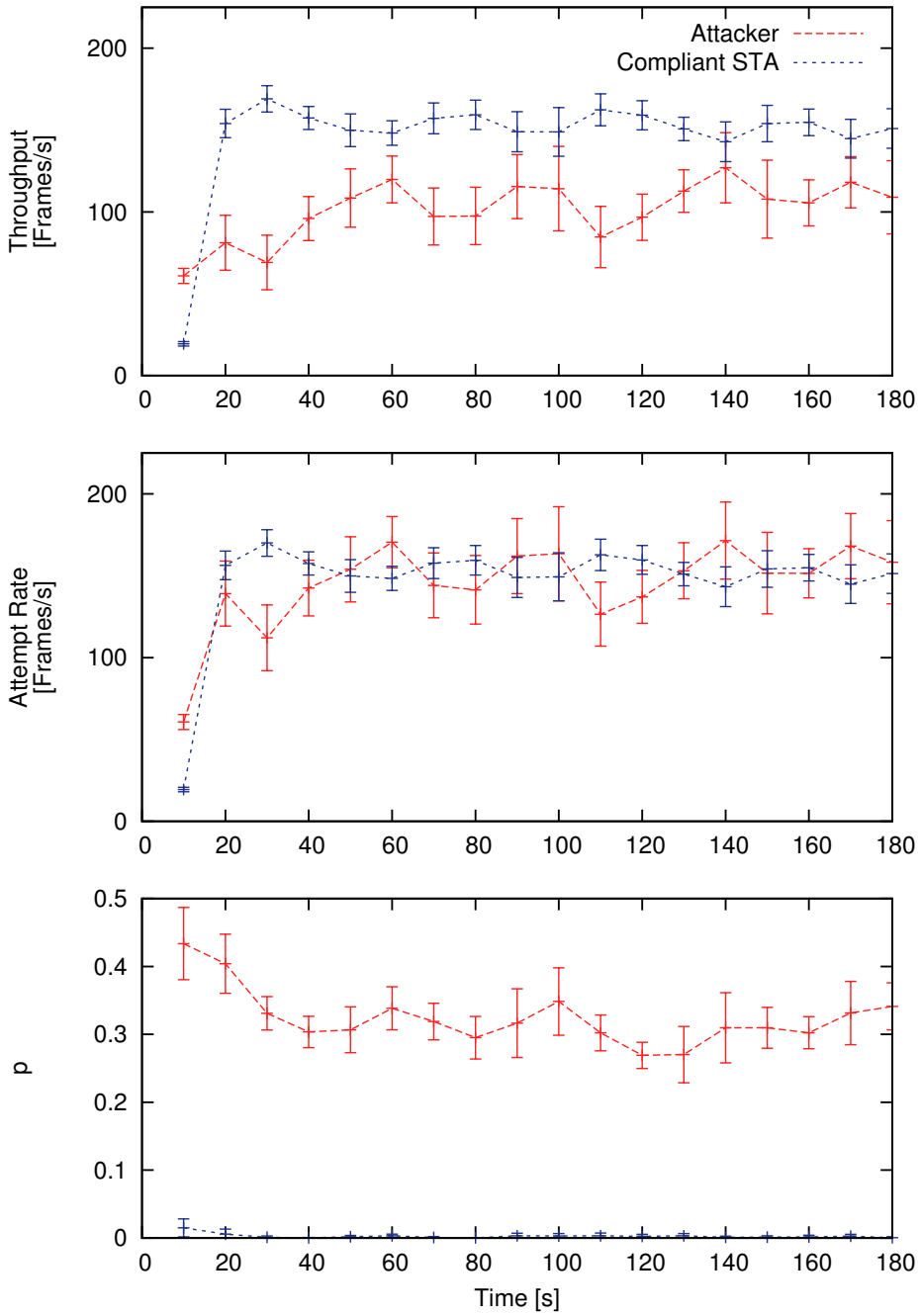


Figure 6.22: Time evolution of throughput, attempt rate, and penalty for a compliant and an aggressively non-compliant station (which sets CW_{min} to a quarter of the standard value), all sending FTP traffic (over TCP). Experimental data.

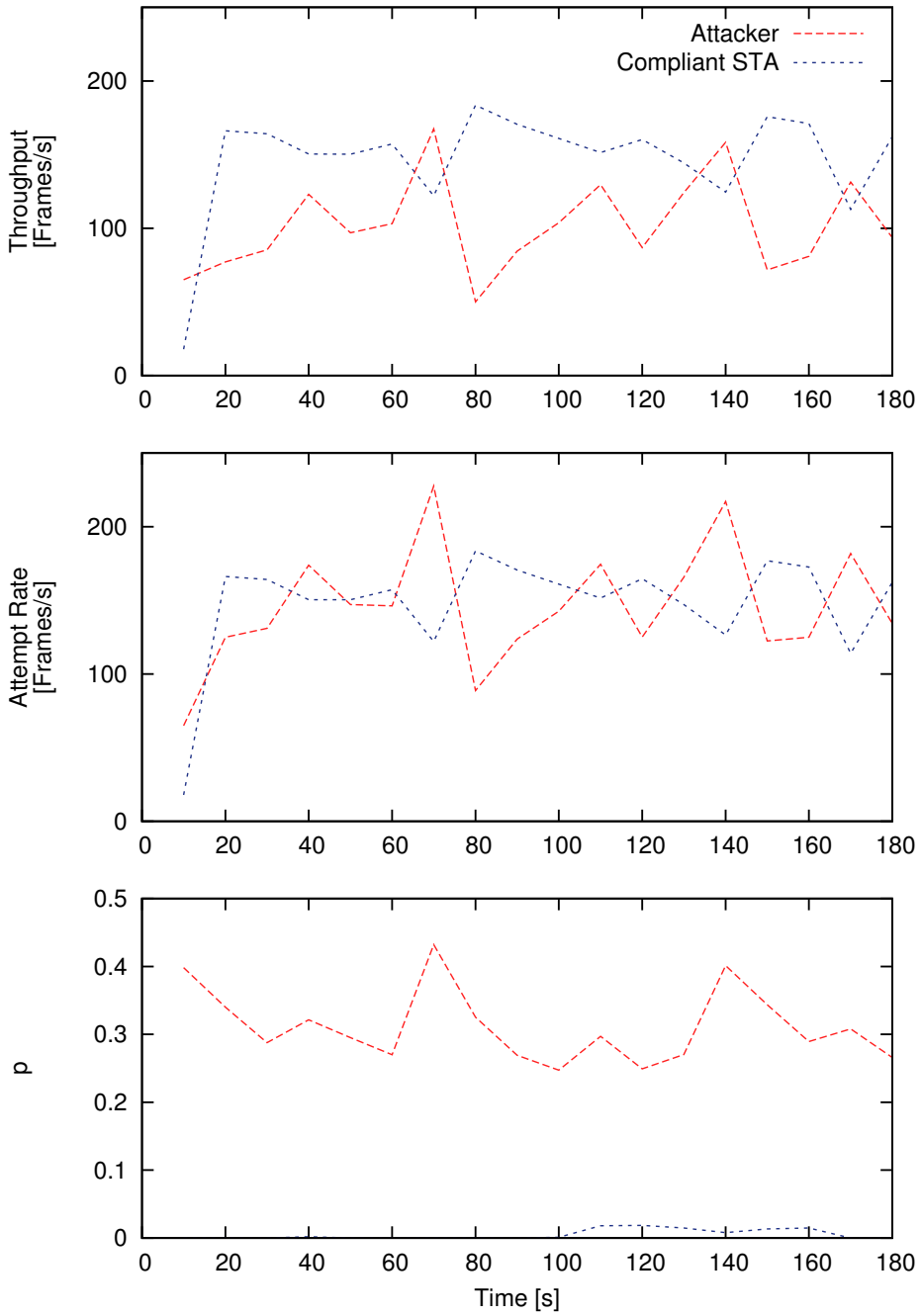


Figure 6.23: Single run time evolution of throughput, attempt rate, and penalty for a compliant and an aggressively non-compliant station (which sets CW_{min} to a quarter of the standard value), all sending FTP traffic (over TCP). Experimental data.

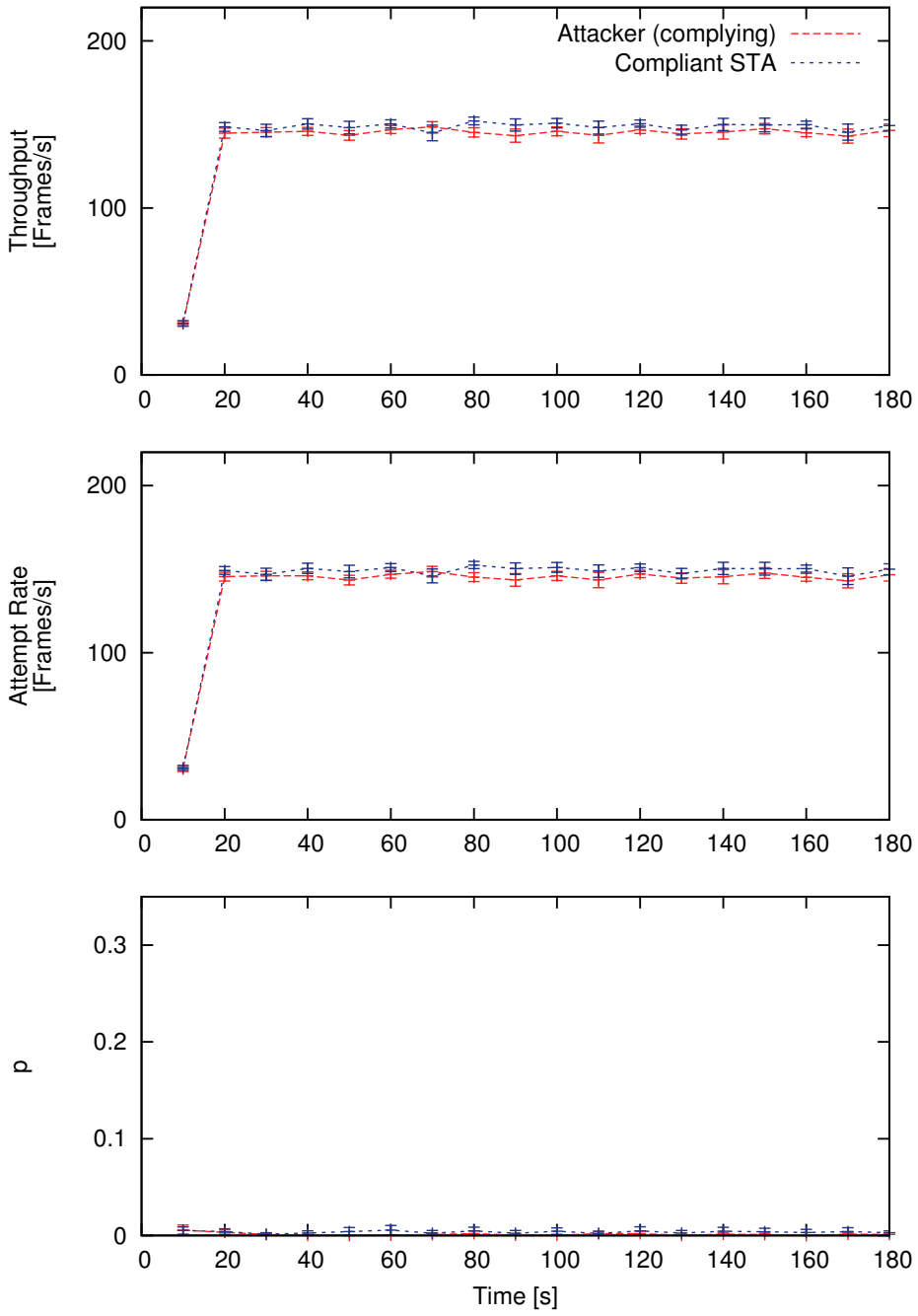


Figure 6.24: Time evolution of attempt rate, throughput, and penalty when all stations are compliant and sending FTP traffic (over TCP). Experimental results.

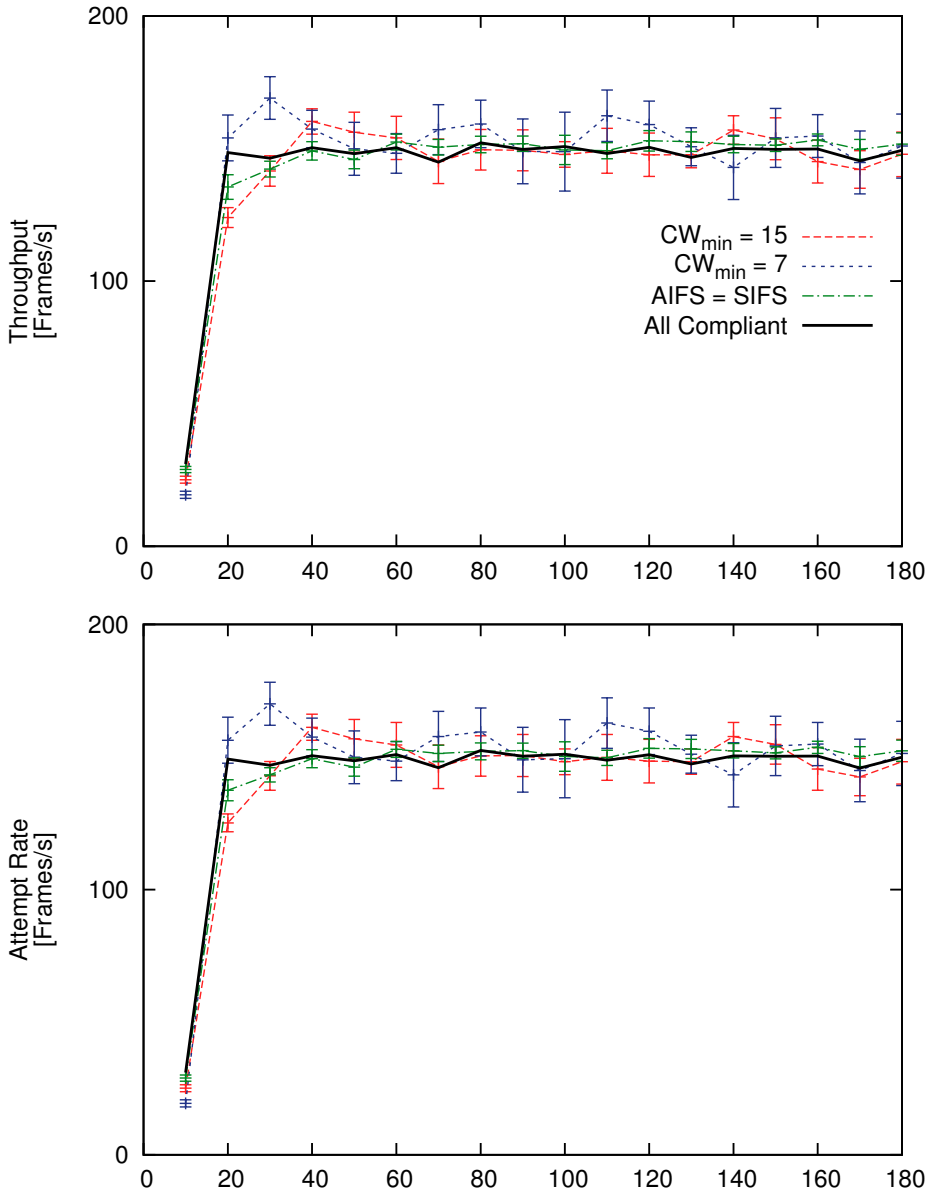


Figure 6.25: Time evolution of attempt rate and throughput for the compliant station in Figures 6.20, 6.21, 6.22, and 6.24. Error bars are omitted for the all-compliant case and the corresponding line is plotted with a thicker stroke for better comparison with others. Experimental data.

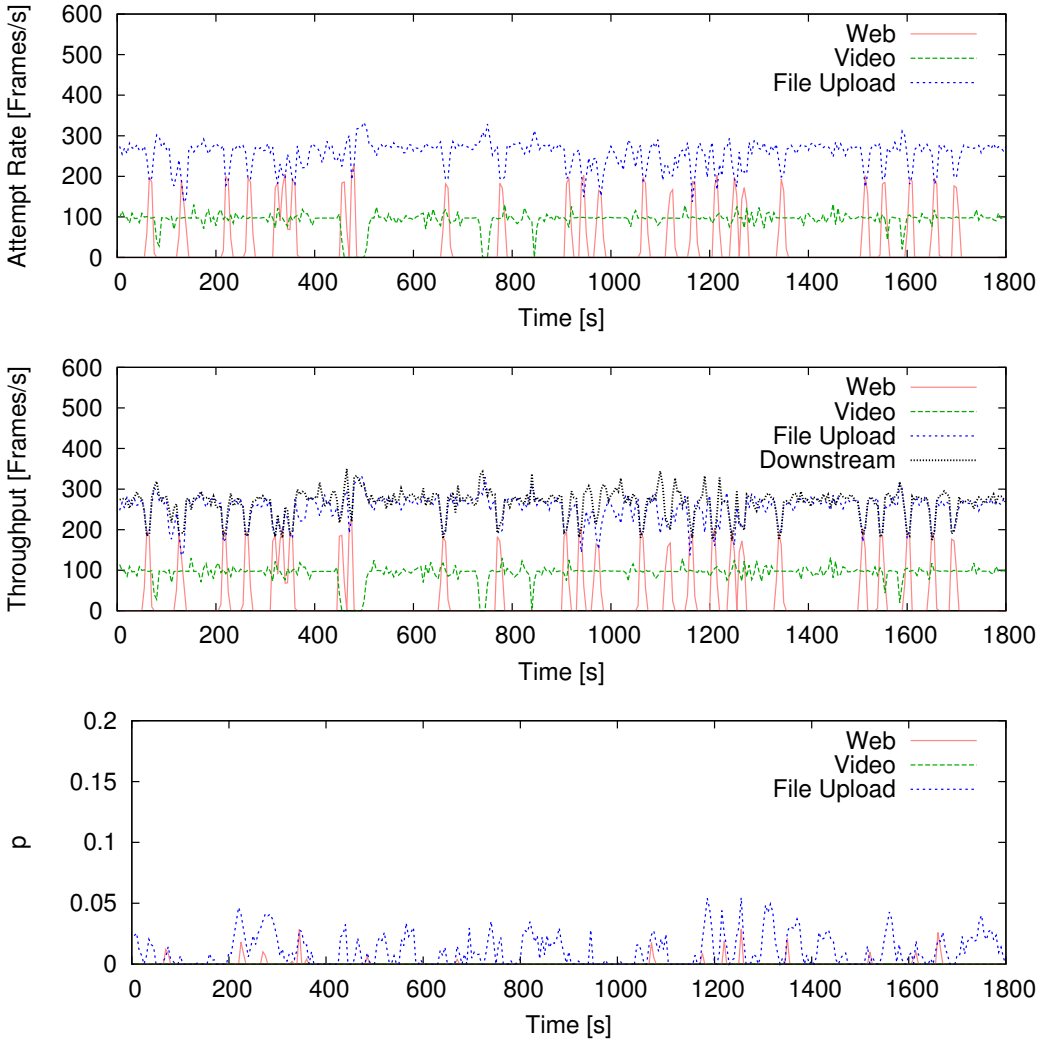


Figure 6.26: Performance under mixed traffic. Experimental data.

third station streams a MPEG-4 encoded version of "Resident Evil: Apocalypse" at 1 Mb/s using the VLC media player[9]. To emulate the activity of the fourth station, we use a backlogged `iperf` downstream session from the AP to the client. We run this experiment for a total duration of 1 hour. In this scenario, as the AP is always standard-compliant, we use the downstream flow to estimate the compliant throughput.

In Figure 6.26 we plot a 30-minute snapshot of the network operation in this experiment, showing the time evolution of the attempt rate of each client, the throughput attained by each flow, as well as the penalty applied by our

[9] http://www.videolan.org/

policing algorithm to each station. First, we observe that the penalty stays at zero about 85%[10] of the time, only with infrequent and small variations (below 0.05) above zero. Second, the medium-quality video flow sees its bandwidth demand satisfied most of the time. Third, the bandwidth demanding upload and download flows equally share the remaining available air time. Lastly, the spurious web traffic experiences similar performance to that of the other data flows whenever they are competing.

We conclude that the proposed policing algorithm does not penalize nodes that generate more traffic than their competitors as long as they comply with the MAC configuration defined by the IEEE 802.11 standard.

6.8 Non-ideal Channel Effects

Next we investigate the performance of our implementation under several challenging situations that occur frequently in practice. Specifically, we assess the impact of our algorithm on rate-switching decisions taken by state-of-the-art rate control algorithms and investigate the potential of our scheme to alleviate unfairness issues that arise due to the PHY/MAC interactions occurring in the presence of the capture effect.

6.8.1 Rate Adaptation

We study the behavior of a rate control algorithm executed at a greedy client that manipulates their MAC configuration and is being penalized by our policing algorithm to counteract their misbehavior. Our goal here is to verify that rate control (RC) algorithms will not wrongly interpret suppressed ACKs as losses caused by poor channel conditions and thus will not trigger downgrades of the PHY rate. This is particularly important, since unnecessarily selecting a lower modulation scheme can be wasteful of channel time and have a significant impact on overall network utility [21].

Though there are many rate adaptation algorithms in the literature, `mac80211` driver on Linux systems only implements Minstrel[25] and PID, with Minstrel

[10]This number is an average for all stations in this test. Values for individual stations differ, with the highest belonging to the saturated station which is 36%, and the lowest belonging to the video traffic for which the penalty is actually always 0. However, it does not have a noticeable impact even for the station paying the highest toll, as the effect is small and transient.

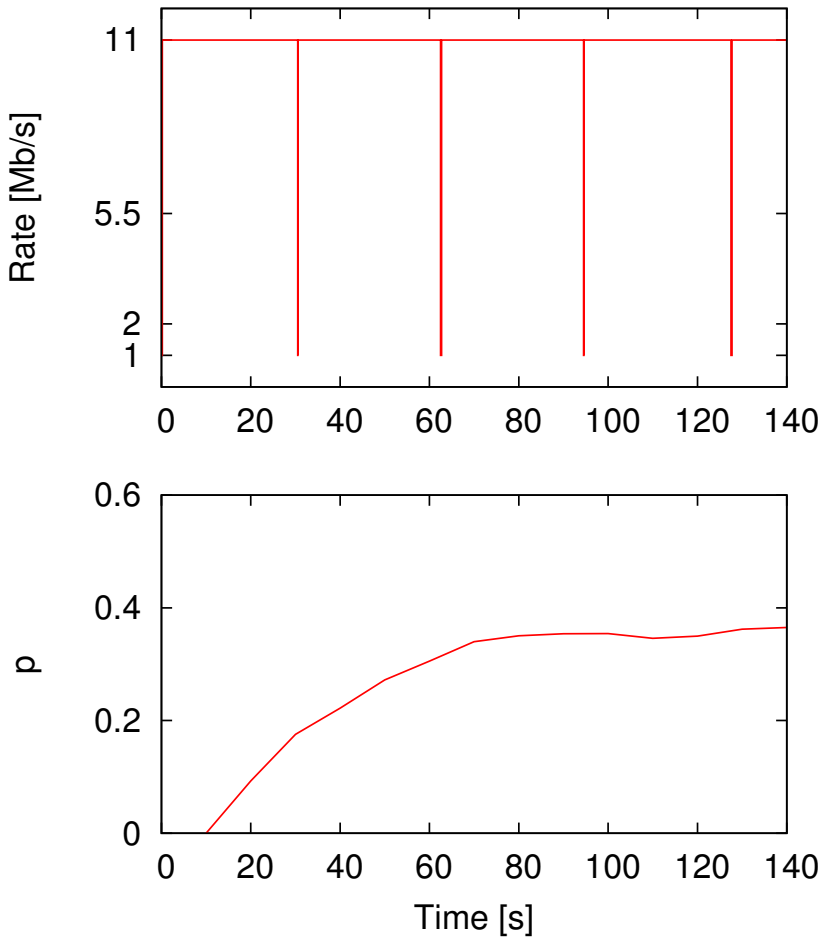


Figure 6.27: Rate selection when policing is applied. Experimental results.

currently being the default. In our first test we consider a non-compliant station that uses Minstrel. We examine the time evolution of the penalty applied by the policing algorithm to the attacker, as well as the rate selected by Minstrel during the operation of our scheme. We consider again a simple scenario with two compliant clients and one attacker using a smaller CW_{min} parameter (see Figure 6.4). As shown in Figure 6.27, increasing the penalty does not influence the rate selection decisions taken by the rate control algorithm, since packets are transmitted almost always at the maximum rate (11 Mb/s) and lower rates are only periodically sampled (approx. every 30s), with only a couple of frames.

Further, to illustrate that the network utility is not affected when policing is

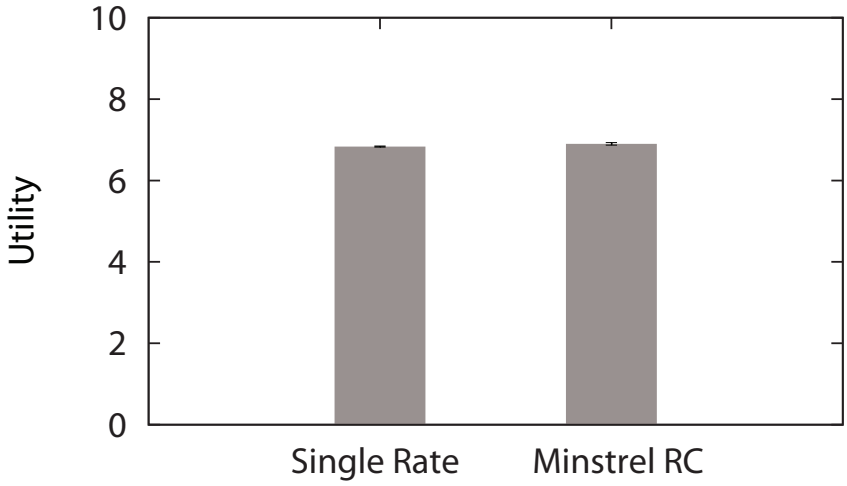


Figure 6.28: Utility of a policed network, with and without rate control

applied, in Figure 6.28 we plot this metric for the same network configuration when the attacker does not perform rate adaptation and respectively executes Minstrel. Note that we compute the network utility as in [42], i.e. the sum of the log of the individual throughputs, which is considered a good measure of proportional fairness [46]. From the results in Figure 6.28 we conclude that, indeed, our policing algorithm does not have a negative impact on the network utility when clients run current rate control mechanisms.

Next we observe the impact of policing on stations' throughputs when they employ rate adaptation. In what follows we test the performance of stations when they use Minstrel and PID rate adaptation algorithms. As we mentioned earlier, these are two prominent rate adaptation algorithms in Linux drivers, and they are the only options available in the ath5k driver. We use the same scenario described above, and run the tests for 180 seconds. For each test, all stations use the same rate adaptation algorithm.

Figure 6.29 summarizes the performance of the non-compliant station for each rate selection strategy. You can see in the figure that when rate adaptation is employed, lower ACK-dropping probability is enough to bring back the station's attempt rate to that of a compliant station. Also, the converged throughput is larger as a result, which means that less channel time is wasted. Figure 6.30 shows the performance of a behaving station in the same set of tests. As shown in the figure, there is little to no difference in the performance

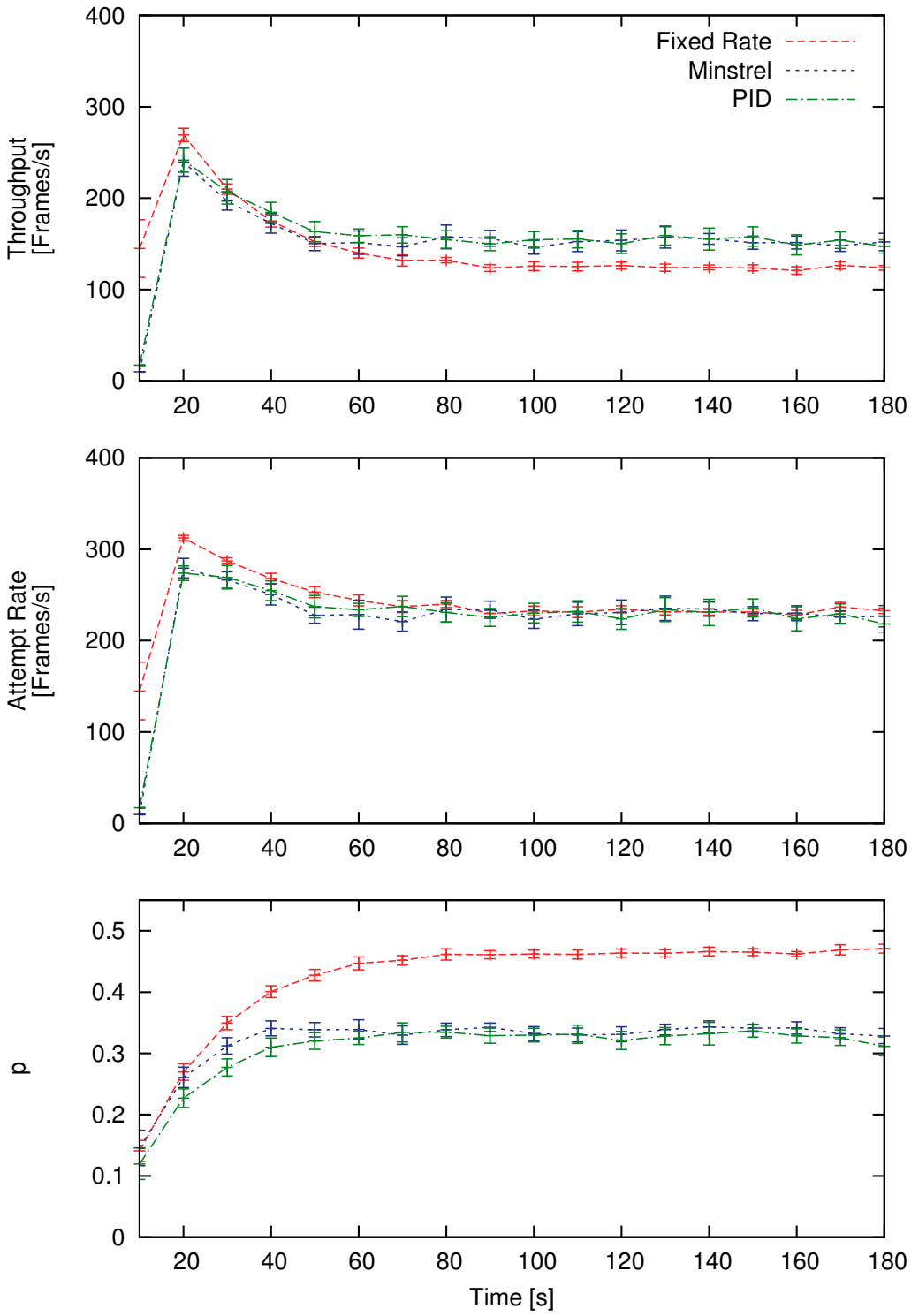


Figure 6.29: Performance impact of the policing algorithm on the non-compliant station when all stations employ rate adaptation algorithms. There is one non-compliant station (CW_{min} halved) and two well-behaved stations in this network. Experimental results.

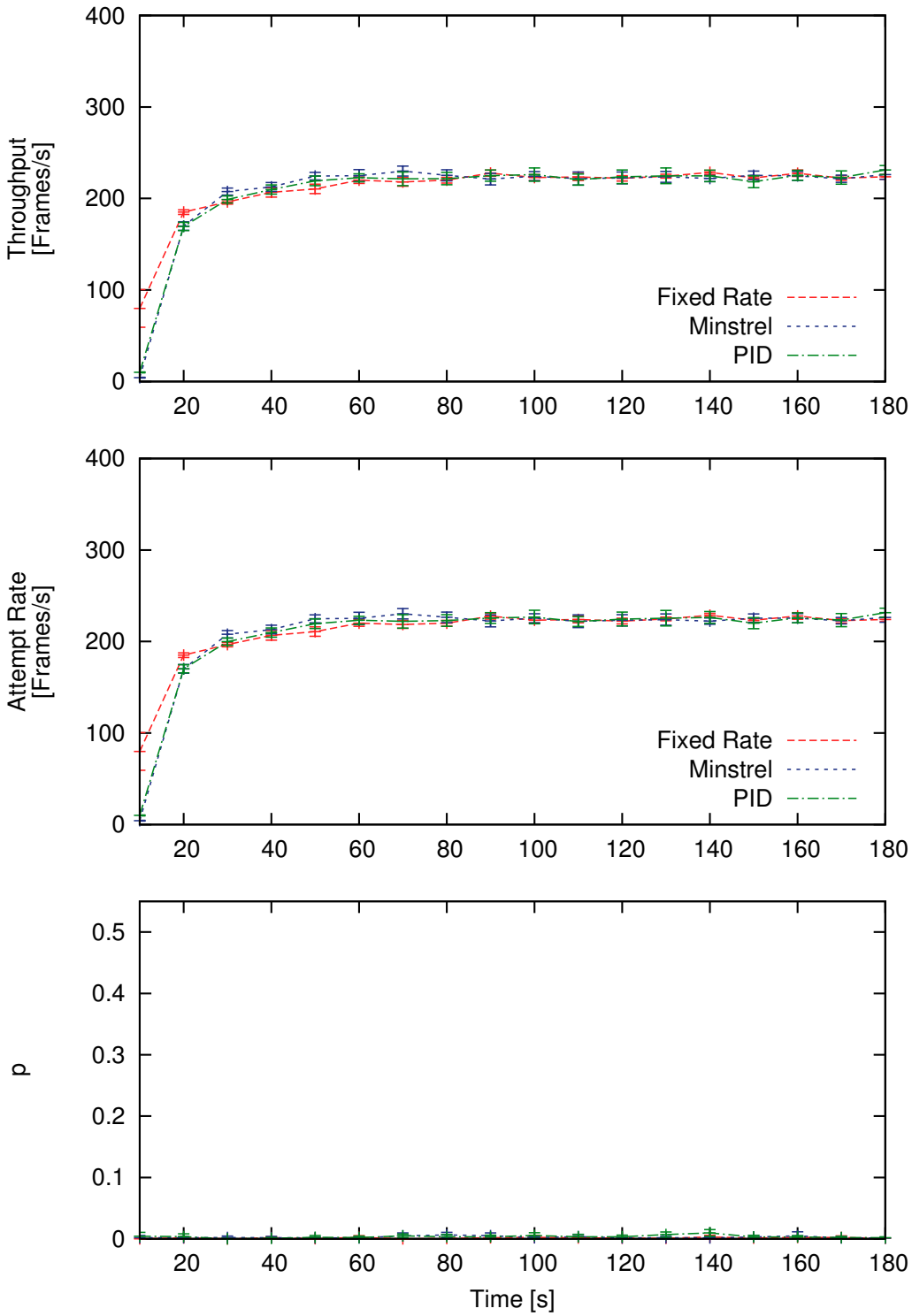


Figure 6.30: Performance impact of the policing algorithm on a behaving station when all stations employ rate adaptation algorithms. There is one non-compliant station (CW_{min} halved) and two well-behaved stations in this network. Experimental results.

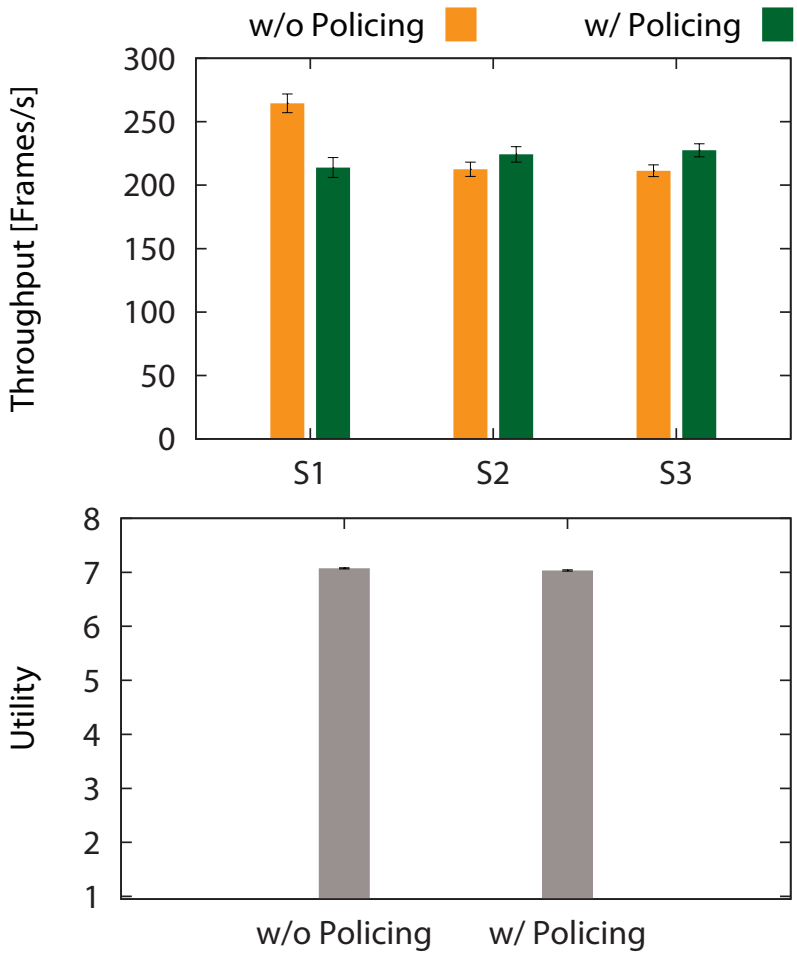


Figure 6.31: Performance under capture effect. Experimental results.

of the compliant station with any rate selection strategy.

6.8.2 Capture Effect

Next we investigate a scenario where all stations obey the standard specification, but experience different performance due to their placement relative to the AP. Specifically, we are interested in checking whether our policing scheme can improve fairness when a client that is located closer to the AP captures the channel while transmitting simultaneously with stations that reside farther away. This effect is frequently encountered in practical deployments and may cause significant unfairness, as already documented in e.g. [143, 144].

For this purpose, we examine the performance of a network with three compliant stations again, but this time with one station (S1) located next to the

AP. As shown in the top plot of Figure 6.31, in this scenario an AP that does not perform policing will take no action to correct the distribution of the throughput among contenders. Thus S1 achieves significantly better performance than the other two clients. On the other hand, when the AP executes our policing algorithm, the attempt rate of the node positioned near the AP is reduced and consequently all stations attain nearly identical throughputs. Note that this correction comes at no network utility cost, as we show in the lower plot of Figure 6.31.

We conclude that, though not designed to do so, the proposed extended policing algorithm not only combats MAC misbehavior, but can also be used to mitigate unfairness that arises in real deployments due to the PHY/MAC interactions.

6.9 Conclusions

In this chapter we implemented the policing algorithm and the Virtual MAC estimator on off-the-shelf hardware and demonstrated the effectiveness of policing by conducting experiments in a real network over a wide range of scenarios. The results of these experiments show that our policing algorithm drives non-compliant users into compliant operation, regardless of the type of attacks employed (among those considered in this study), and does not penalize compliant users that consume more airtime than lightly-loaded stations. Additionally, the results showed that our proposal has no negative impact on common rate control algorithms, and can also alleviate unfairness incurred by the capture effect.

CHAPTER 7

Conclusions

In this chapter, we review and summarize the work presented in this thesis, draw conclusions out of our analysis, and give recommendations for possible extensions to this work.

7.1 Summary and Conclusions

As we discussed in Chapter 1, the wireless medium is not always as predictable and easily examined as in simulations or analytical models. This makes experimental analysis difficult and time-consuming. In Chapter 4 we introduced tools and methods that can facilitate this process. We provided an overview of the Broadcom BCM43xx chipset architecture. We introduced the firmware as a special-purpose piece of software that runs on the wireless adapter. We introduced the assembly language used in the firmware of Broadcom devices and concepts such as Template RAM and shared memory. Then we talked about the driver and how it works and how it interacts with the device. Finally, we described how we split the workload between the driver and the firmware.

The most important theoretical contribution of this thesis is the analysis and amendment of the policing algorithm introduced in [6]. The IEEE 802.11 standard leaves room for stations to abuse the back-off mechanism to gain more channel time and this causes compliant stations to have less. In Chapter 5 we introduced said policing algorithm. Although the original algorithm detects

such misbehaving stations and penalizes them proportionately, it does have some shortcomings as mentioned before:

1. It does not provide a mechanism to estimate the throughput of a compliant station, which is required in the policing controller.
2. The amount of penalty applied to a non-compliant station is enough to equalize throughputs, but not to compensate for the network degradation induced by the non-compliance and the policing.
3. It is prone to gaming, and a station that is aware of the algorithm can still choose a winning strategy using bursty traffic.

In this work we amend the algorithm to be more robust and effective, and create an estimator for the compliant throughput. The new algorithm is robust to non-compliant stations trying to game it, and the estimator provides sensible values in various scenarios that we have tested, regardless of the network topology and configuration. These are proven both theoretically, and through experiments on real hardware for various scenarios, including with realistic network traffic (video transmission, TCP file download, etc). These experiments and their results were presented in Chapter 6.

While the amended algorithm works for all scenarios conducted in IEEE 802.11b/g/a, this is not always the case. There are attacks that are immune to this scheme, and there are later IEEE 802.11 amendments that mitigate its effectiveness. Jamming attacks are examples of attacks that are immune. The AP will not notice the misbehavior if the station uses correct EDCA parameter, but jams control frames to buy time. Another example is packet forging attacks, which is overlooked by the policing algorithm. Furthermore, the policing algorithm neither targets nor is effective against attacks that aim only to degrade network performance without an intended gain for the attacker.

As for IEEE 802.11 amendments, service differentiation is one example that can invalidate the policing algorithm as it stands in this thesis. However, in Section 5.3.3 we described how it can be adapted to EDCA. Adapting it to newer features can be more complicated. Examples are No ACK, Block ACKs, and Direct Link Setup. Trying to make the policing algorithm work

alongside these features will often mean canceling the performance benefit of the features, or using complicated hardware-level measurements or unnecessary message-passing between nodes. So, while the algorithm can be amended for a marginal benefit in modern IEEE 802.11 networks, it is not applicable to those networks as it currently stands.

Rate adaptation is another potential source of problems for the policing algorithm. Although we showed through experiments that the most common rate adaptation algorithms will not get a negative impact by the policing algorithm, we know that older algorithms such as AMRR that rely solely on retries will perform poorly under policing.

During the course of this work, we learned that although experiments can be very quick to perform in an ideal environment and they are helpful in validating algorithms and ideas, they can be very time-consuming when it comes to debugging. A great portion of our time was always spent on debugging our testbeds and figuring out causes of problems and unexpected behavior of the network. The diagnostic tool was an attempt to help reduce this time and bring focus to what's more important, which is the validation process.

We also discovered that while current off-the-shelf wireless hardware can be reprogrammed and ideas can be implemented on them, there has not been enough interest in highlighting these capabilities. There are valuable works like [125] and [126] that try to develop this idea, but it is still unknown for many in the areas of research and development. This is partly because of insufficient publicly available documentation. We hope Chapter 4 helps to redress this lack of documentation.

7.2 Future Works

The present work opens the path to wider experimentation with Wi-Fi hardware. As for the policing algorithm itself, we mentioned how it could possibly be adapted to work alongside some of the new IEEE 802.11 features. Implementing those changes could also be interesting, and keep the policing algorithm relevant in the continuously evolving world of wireless communication.

With the power to modify the behavior of IEEE 802.11 wireless cards, not only we are able to make changes to the state machine, as we did in Chapter 6, but

we can also implement and test other protocols. An example of protocols that can be conformed to wireless adapters is the power-line communication (PLC) using the IEEE 1901 protocol[145]. PLC devices are used to carry network traffic over the conductors used for electric power distribution. The benefit of using these devices is that the existing electric backbone of a residence can be used for setting up a network, rather than setting up new cables and switches. The MAC protocol in IEEE 1901 is in fact very similar to that of IEEE 802.11, with the exceptional of a counter called the "deferral counter" (see [146]). This counter works as an extra collision avoidance measure, as collisions are more costly in PLC. We have implemented the PLC MAC protocol on Broadcom wireless devices. However, to see the real effect of the extra measures, we need a relatively large testbed, which carries its own challenges on the inherently noisy wireless medium.

Another interesting concept that can be implemented on a real testbed is collision-free medium access. The idea of a distributed, collision-free MAC protocol have been studied in [147], [148] (Learning BEB), and [149] (Learning MAC). Learning BEB is a decentralized algorithm that stations in a wireless network employ to pick time slots. Once a station finds a free time slot it always transmits in that slot. This scheme replaces the random backoff used in DCF. Learning MAC is an algorithm built on the same idea, with enhancements on how it handles collisions. In [149] they prove that their method converges if the number of stations is not too large, and achieves better network throughput. We have a current implementation of the Learning MAC, which, however, falls short on throughput expectations. With background noise and internal delays of the wireless adapter, the desired goal is not achieved on our testbed. A complete implementation could prove the concept practical as well as efficient.

There are also other works whose implementation should be possible using techniques we discussed in this manuscript, and we would like to evaluate their behavior on a real testbed. Examples of such works include: [111], where they use ACK-dropping to provide throughput guarantees to EDCA stations in a network where legacy DCF stations also contend; [27], where they attack the exposed node problem (see Section 2.6.3 using an algorithm that allows concurrent transmissions when possible; and [138], where they propose a scheme named DCF+ to enhance the performance of TCP over

WLAN.

Apart from evaluation of algorithms, the flexibility provided by open-source driver and firmware can be further cultivated for next-generation wireless hardware that is easily and perhaps visually programmable without much knowledge of assembly or driver programming. [126] is a significant step towards this end. With such frameworks, researchers can easily create new MAC protocol ideas and put them to practice and even competition with existing or other MAC protocols.

APPENDIX A

Designing a Diagnostic Tool for IEEE 802.11 MAC

Experimental assessment has been an important part of IEEE 802.11 research, however measuring the detailed behavior of the medium and hardware has been challenging. A diagnostic tool for IEEE 802.11-based WLANs is designed in this appendix, which helps developers and researchers monitor and analyze the wireless signals and details such as backoff distribution in a user-friendly environment. This tool is much cheaper and easier to use than existing tools, and provides more flexibility by allowing users to add functionality. We then use WiFo to study several aspects of some off-the-shelf hardware and their corresponding software drivers, and show some interesting results regarding how they apply standard specifications.

A.1 Introduction

With ever increasing interest in WLANs, researchers have been trying to improve current protocols (such as IEEE 802.11) in terms of performance [149, 148, 150], security [151, 144], and scalability [152]. Mathematical analysis and simulations are common ways of evaluating new methods [95, 27]. The final step in evaluating a new method is putting it in practice on a real network, which is what we have been discussing in this thesis.

In order to implement and experiment with something on a wireless medium, we need to understand the medium itself. Although protocol descriptions are available in detail [5, 10, 11, 133, 12, 22, 13] and we can know what "should" happen on the wireless medium per protocol, in most cases that is not exactly what happens. Sometimes this is due to the implementational flexibilities provided by the standard, and other times it is beyond those flexibilities and we have deviations from standard. Even if we assume all wireless hardware behaves exactly as the standards suggest, we still have other factors that interfere with our expected results in an experiment. We discussed some of these in Section 2.6. Besides, the truth is that our first assumption is also wrong and many devices do not completely follow the standard [72].

In order to have controlled experiments, we need to know the hardware we use and also channel conditions beforehand. For example, if a wireless device has an unexpectedly high saturation throughput, we can infer that it is not following the standard.[1] However, without better diagnostics we cannot know for sure which part of the standard is not being followed. It could be an abuse of TXOP, or using a small contention window. This gets worse if it is having an unexpectedly low throughput, as it can either be a hardware failure, a protocol adoption error, channel interference, or something else. In order to find out which one is the case, we need more information than just the throughput. In order to identify the underlying problem, we need more information.

One device that is designed for detailed analysis of the wireless medium is a spectrum analyzer. A spectrum analyzer measures the magnitude of an input signal versus frequency within its frequency range. By analyzing the spectra of electrical signals, dominant frequency, power, distortion, harmonics, bandwidth, and other spectral components of a signal can be observed that are not easily detectable in time domain waveforms. Spectrum analyzers are particularly useful for understanding the physical (PHY) layer of a transmitter or receiver, such as power levels, distortion and interference. Some spectrum analyzers even come with add-ons that characterize PHY symbols or packets. Figure A.1 shows a spectrum analyzer.

We can use a spectrum analyzer to observe the IEEE 802.11 medium by simply

[1] This was discussed in greater detail in Chapter 2.

Figure A.1: Spectrum analyzer

adjusting the frequency range to that of the specific channel we would like to study. In Figure A.2 we have done this for Wi-Fi channel 14 while one station transmits saturated traffic to an access point (AP) in this channel. The first plot shows the maximum, minimum and current power observed, and the Wi-Fi channel is clearly visible within the bump in the blue line that shows peak power per frequency. The spectrum analyzer can also give us the changes in power over a period of time using "zero span" mode. With zero span, we can actually understand temporal aspects of the channel. Figure A.3a and A.3b are snapshots from a spectrum analyzer screen on zero span mode while Wi-Fi traffic is ongoing on channel 14.

While we can observe traffic using a spectrum analyzer, and it provides a number of ways to process the observed data, there are some downsides, including financial cost. A spectrum analyzer is a versatile but relatively expensive device, and thus the expense may not be justified for a group developing Wi-Fi drivers or analyzing some performance anomaly. In addition, because it is a general-purpose device, many 802.11 properties are not recognized by a typical spectrum analyzer, or are only understood by specialist add-on packages. When we study IEEE 802.11, we are often interested in things like backoff period, throughput and transmission time. We often need to have numerous samples in order to understand the stochastic behavior of the Wi-Fi MAC and PHY. Although there are ways to export spectrum data using a spectrum

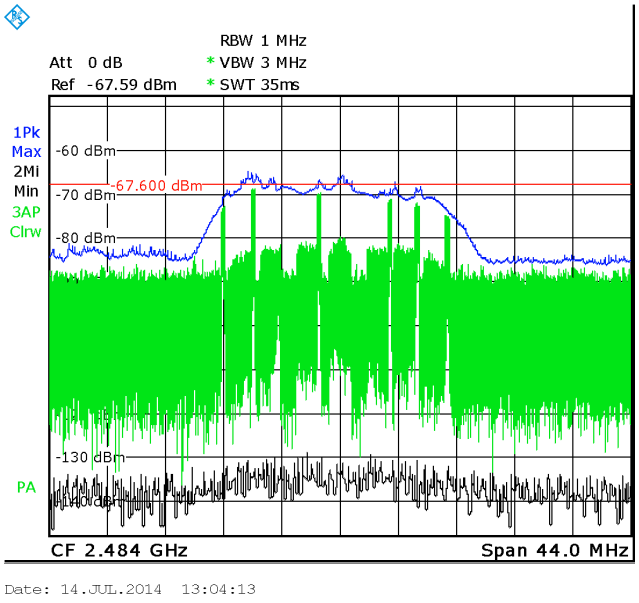
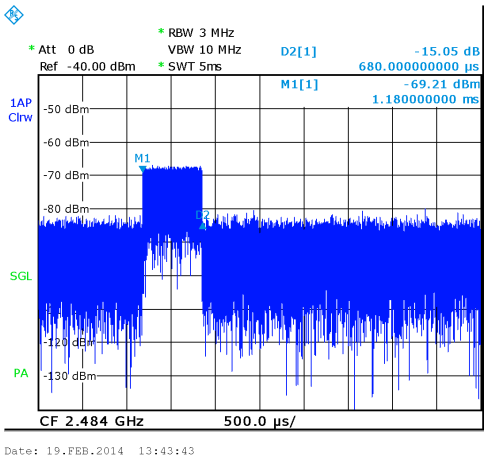
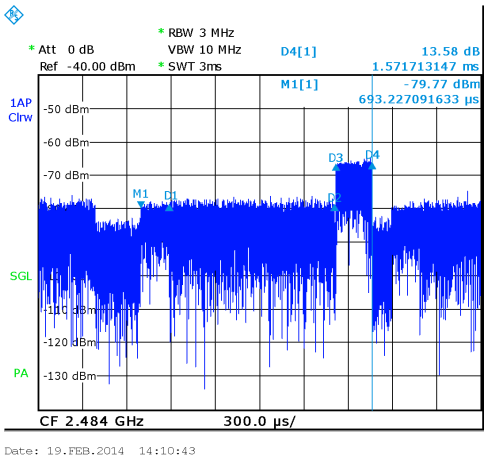


Figure A.2: One station transmitting saturated traffic on channel 14 (center frequency 2.484 GHz). Blue indicates peak power, black corresponds to the minimum received power, and green is current status of the spectrum.



(a) A single beacon



(b) Successful frame transmission and ACK

Figure A.3: Spectrum analyzer: 44 MHz frequency span for (a), and zero span for (b) and (c).

analyzer and process it later on a computer, the whole process is often not easily automated or tailored to those studying Wi-Fi, and so experiments can be difficult without human interaction.

In this appendix we provide a solution that addresses both these problems which are inherent in using a spectrum analyzer. While a spectrum analyzer receives, digests and displays electronic signals, a cheap off-the-shelf wireless adapter also receives all those signals, but it then uses them to provide data transmission and reception over the medium. We leverage the fact that all wireless adapters have the means of sensing the medium to make a test tool. This feature of the hardware has been used before, for example to implement spectrum sensing for cognitive radio [118] or to detect non-Wi-Fi sources of interference [119].

We use carrier sensing of off-the-shelf wireless hardware in a different way; rather than using data collected from the medium for one particular purpose, we aim to export it to the application layer, where it can be analyzed using high level tools. Works such as [72], where the study of existing wireless hardware or software is intended or required, emphasize the need for the tool we present.

We discussed OpenFWWF [67] in detail in Chapter 4. With the flexibility provided by this open-source firmware, we use a Broadcom wireless adapter to create an inexpensive tool for researchers and developers to study IEEE 802.11. What we want to develop is a tool that can monitor Wi-Fi traffic and provide a visual representation of the received data, as well as statistical studies on transmitted frames. The tool will sit just above the PHY layer and focus on the interaction of PHY and MAC layers. Additional functionality can also be introduced to the system through a plugin system.

A.2 Design and Architecture

This section covers implementation details of our wireless diagnostic tool. As we mentioned before, we use a commercial off-the-shelf wireless adapter to monitor the medium. To allow greater flexibility in the processing and visualization, we transfer the data from the monitoring host using a small TCP-based server to a front-end host. This separation of monitoring and front-end hosts allows us to install the monitor on a small device, such as a Soekris net4801 [153], while using a higher-powered device for storage and visualization. Our front-end visualizes the data and allows the user to analyze and study the data. Figure A.4 depicts the building blocks of our system. We

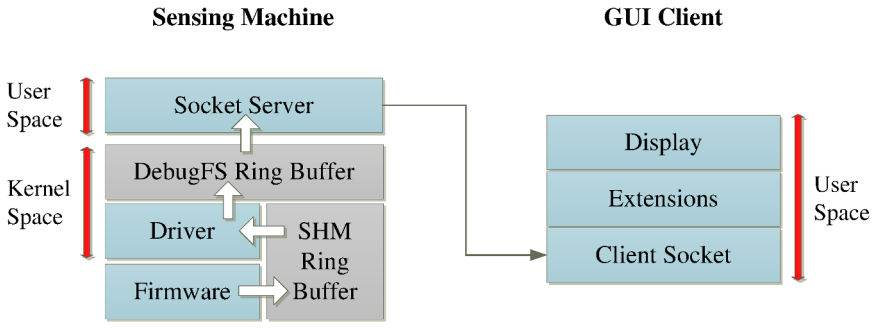


Figure A.4: Simple architecture diagram of the diagnostic tool

Bit	Meaning
15	Flip to 1 when time reserved for receiving PLCP has passed
11	Flip to 1 when RX'ing or TX'ing (same time receiver flips on, may 1μs after transmitter flips on)
10	Flip to 1 when RX'ing or TX'ing (same time receiver or transmitter flips on)
9	Flip to 1 when receiver has started decoding
8	Flip to 1 when transmitter is working
7	Flip to 1 when backoff is zero
4	Flip to 1 when time reserved for receiving MPDU has passed
3	Flip to 1 when channel is sensed free (phy+nav) for more than two slots
2	Flip to 1 when channel is sensed free (phy+nav) for more than one slot
1	Flip to 1 when channel is sensed free (physically)
0	Flip to 1 when channel is sensed free (virtually through NAV)

Table A.1: Important bits of the "IFS Status" register and their meanings

will discuss the different parts of this diagram in this section.

A.2.1 Firmware and Driver Modifications

The firmware is the software running directly on the chipset and the first layer above the hardware. Signals from the medium are translated into a digital representation and digested by different chips, the results of which are then fed to the firmware. Unfortunately we do not have full access to the raw data as we do using a spectrum analyzer. However, what we do receive from the chipset is enough to observe what is happening on the channel. What we use here is a set of flags stored in a register named the "IFS Status" register [128], which holds current status information about the channel. Some of the most important flags are listed in Table A.1. Using these indicators we can pretty much know what is happening on the channel. As you can see we can know when the card begins sending or receiving, and we can capture the occurrence of several important timeouts.

Given the completeness of the information on the IFS Status register, all we have to do is to record its value over time and base our analysis on flips of the flags. That is the fundamental idea of our implementation. As we mentioned in previous chapters, we have about $4KB$ of shared memory, most of which is unused by the normal workflow of the firmware. We use the available memory in the form of a ring buffer, and whenever at least one bit of IFS Status flips, we save its value along with a timestamp. Note that we only save a new record whenever there is a change as this saves space compared to periodic recording.

One might ask where this procedure is actually inserted into the firmware. Figure A.5 (a) shows the basic state machine of the OpenFWWF. Event handling in this architecture is not interrupt-based and the firmware continuously checks for events and handles them accordingly. Whenever there is no event, the firmware calls a *nap* instruction to sleep for a short while as a power saving measure, and then continues. This is the best place to insert any repeated code. Figure A.5 (b) shows how we place our code. We simply replace the nap instruction with our code. This is exactly what we did for the policing algorithm in Chapter 5 as well.

There is a barrier to our approach, and that is memory limitation. The timestamp we use is 32 bits in size, filled with the least significant bits of the card's current TSF timer. With the register itself being 16 bits in size, 48 bits (6 bytes) are used per record. Even if we could use the whole shared memory, we could only store 667 records, which is a very small number. A single frame can trigger multiple bits of the register during its timespan. Some bits such as bit 8 and bit 11 can be flipped $1\mu s$ apart from each other, which increases the possible number of events per unit time. Moreover, the shared memory is not completely free and putting aside the memory used for the firmware's normal workflow, we are left with space sufficient to accommodate only about 250 records. One solution is to mask out redundant flags and thus reduce the number of triggered events. We couple this solution with the greater resources available to the driver to get the most out of the information available to the firmware.

The driver was discussed before in Chapter 4. The amount of memory available to the driver is usually much larger than the firmware as it is running on the host machine which has more resources. Another useful feature of the

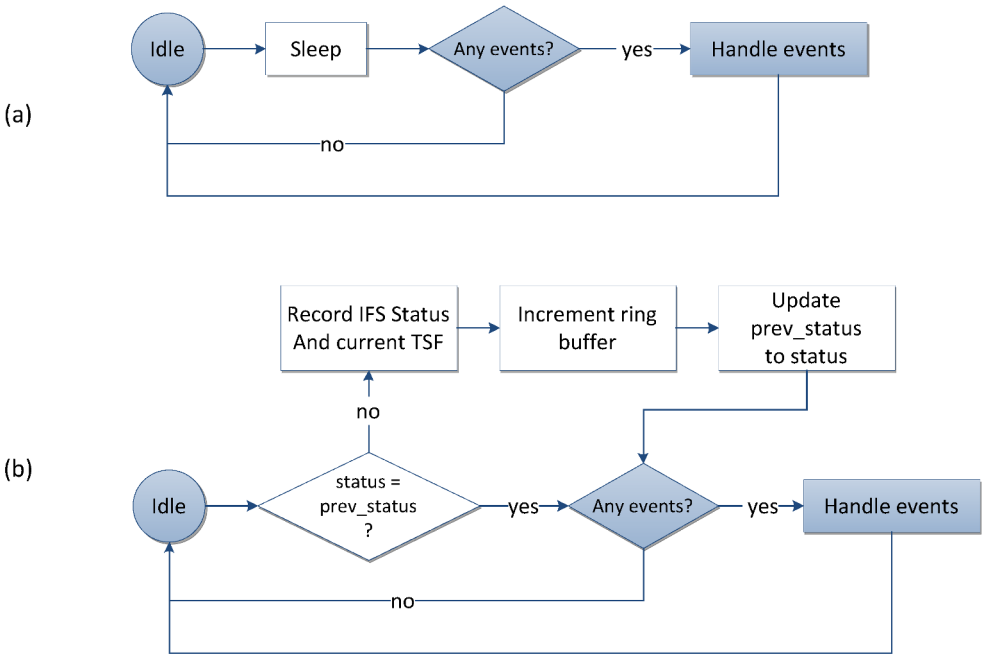


Figure A.5: Simple flowchart showing (a) the main loop of the OpenFWWF state machine, and (b) our modification for the diagnostic tool.

b43 driver is that it can do periodic work at relatively small intervals. We use these in our advantage. In our implementation, we allocate a large ring buffer in the driver, which can hold 10 times as many records as the firmware can hold. We then read the shared memory periodically and add new records to the ring buffer on the driver's side. The period is chosen so every record on the shared memory ring buffer can be read before it is overwritten. In our experiments with saturated traffic, this proves to be 25ms; anything less than this period will frequently read redundant information, and longer periods might lead to loss of information.

A.2.2 DebugFS and Socket Server

The next building block of our diagnostic tool is a socket server. While we are collecting state information from the wireless adapter, we want to let external clients access this information. The main problem here is accessing the information saved by the driver, which is located in the kernel memory, from the user space. Fortunately Linux kernel provides us with a tool named *debugfs*. It is a special file system that facilitates access to the kernel space,

Function	Application
debugfs_create_dir	Create a directory or subdirectory in DebugFS
debugfs_create_u8	Create a file representing an 8-bit unsigned integer
debugfs_create_u16	Create a file representing an 16-bit unsigned integer
debugfs_create_u32	Create a file representing an 32-bit unsigned integer
debugfs_create_u64	Create a file representing an 64-bit unsigned integer
debugfs_create_bool	Create a file representing a boolean variable
debugfs_create_blob	Create a file representing an arbitrary-sized block
debugfs_rename	Rename a file within the file system
debugfs_remove	Remove a previously created DebugFS file
debugfs_remove_recursive	Remove a DebugFS file and its subfiles

Table A.2: DebugFS API functions

and it is included in Linux kernel version 2.6.10-rc3 and higher. The main purpose of DebugFS is debugging Linux modules, however, it is suitable for our software solution.

Before we get to our specific application of DebugFS, we briefly discuss how it works. This file system can be set up using a simple Linux command[2]. The mounted file system is a form of RAM drive. Files in this RAM drive point to pieces of kernel memory, and can be read or written into based on their permissions. Drivers and modules that need debugging create directories and files using the DebugFS programming interface, and provide pointer and permission information for them. Table A.2 shows the most important functions in the API. Beside name and destination information, all *debugfs_create_* functions take permission information and those that create files take also a pointer to the data that will be accessed through the file. This can simply be a pointer to a global integer variable for a *u32* file, or to a structure or array for a *blob* file.[3]

General memory management rules should be taken into consideration, as DebugFS files will not provide access to anything other than what is explicitly assigned to them. For example, a multi-dimensional array could cause a problem, as each of the arrays corresponding to its second dimension are separate pieces of memory, not addressed by the file. A more obvious mistake is the use of local variables for DebugFS files. Pointers to these variables are stack pointers and will become invalid once their scope is removed from the stack.

[2] mount -t debugfs none /sys/kernel/debug

[3] Blob files can have arbitrary sizes and their size should be declared when creating the file. This is done by passing a *debugfs_blob_wrapper* structure to the *debugfs_create_blob* function, which contains data size and a pointer to the actual data block.

Once files are created with persistent data pointers and sufficient access permissions, they can be treated as ordinary files from the user space. A *u32* file will simply be a file with 4 bytes in size, containing a 32-bit integer. The contents of this file constantly changes as the variable it points to changes value in the kernel space.

We use a DebugFS blob to collect state information from the driver. We mentioned earlier that we collectively store firmware state information in a large ring buffer in the driver. This array is large enough to keep up with the overwriting speed of the firmware ring buffer. However it is not, and need not be, large enough to keep a full history. We choose to bring more work to the user space, so we do not keep and export the entire history from the driver. Instead, we design our server application so it dumps the exported information periodically and leaves room for more information on the ring buffer.

Exporting the information to user space is our next step. The only change we make on the driver side to do this is to create a DebugFS blob file that points to the ring buffer array. On the application side, we read the whole array from this file at an interval slightly less than it takes the driver to fill the ring buffer and begin overwriting. For instance, we use 200ms for the driver we described before, as it normally fills the DebugFS blob in 10 iterations (i.e. 250ms). This is to avoid missing data due to processing delays. Old items in the array can be identified via each record's TSF timestamp, allowing us to remove items that are read twice.

The server application does not keep newly read items. It listens on a TCP socket for incoming connections. Once a client application connects, it flushes the data to the client over the network on each DebugFS read. The system we described up to here can join the wireless adapter on a Linux box to a sophisticated client on a PC or any other processing agent.

A.2.3 Front-end

A.2.3.1 Main Graphical Interface

In the previous subsection we discussed how a small server solution relays a wireless card's internal state information to an external client. The client we use to receive this data is a graphical client named WiFo, which is developed using the .NET framework, and typically runs on a Windows PC. It connects

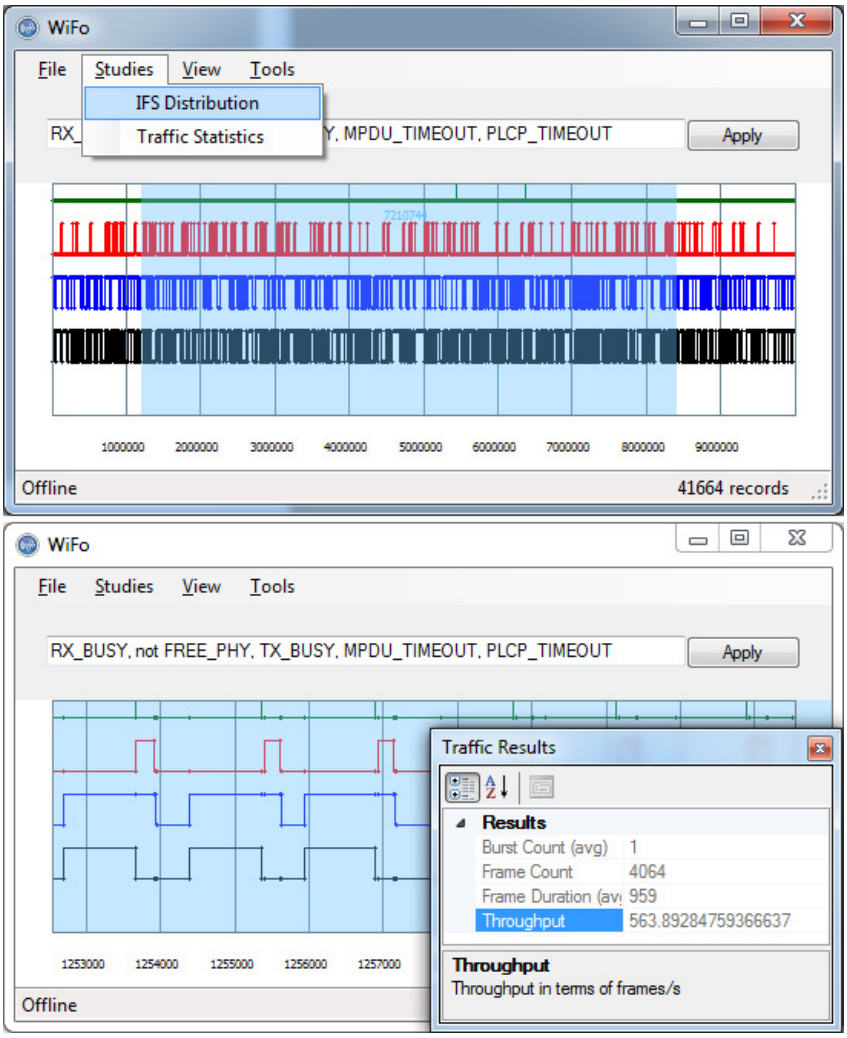


Figure A.6: Screenshots from the diagnostic tool's front-end (WiFo).

to the server and displays the data it receives from it. Figure A.6 shows a few screenshots from this application.

At its most basic state, you can use boolean expressions to monitor different status bits as the wireless card works. These are displayed on a live chart called the *timeline*. By choosing a suitable combination of bits to be displayed on the timeline you can recognize frames, ACKs, and other activities that take place on the channel. Figure A.7 shows the timeline. In this example we have chosen bit 9 (RX engine busy status), the complement of bit 1 (which corresponds to PHY busy status), and bit 8 (TX engine busy status). The black line here represents bit 9. Therefore bumps on the black line represent

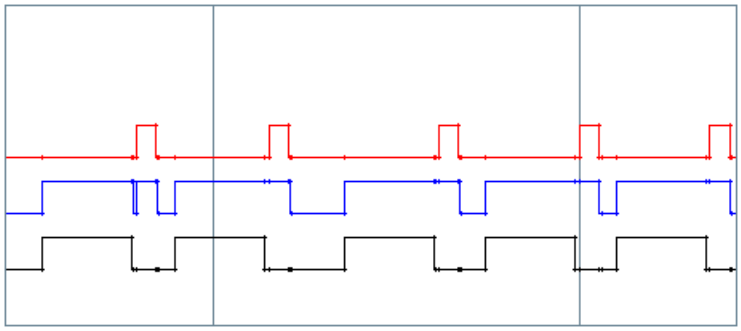


Figure A.7: Transmission of a few packets captured on WiFo. The bottom, middle, and top curves show busy status for the RX engine, the medium (PHY), and the TX engine respectively.

frame transmissions. The diagnostic tool in this figure runs on the AP. This suggests that bit 8 would be flipped on ACK or beacon transmissions. The red line in the same image shows bit 8's changes. As the figure shows, after each black period we have a short period where the red line is up. The duration of this flip is exactly the duration of an ACK, which confirms an ACK transmission.

An interesting fact is visible in Figure A.7, and that is a limitation of the system. One would expect to see SIFS periods on the blue line as it captures PHY busy status, and PHY is idle during SIFS. Curiously we only see this period only once in this figure, and it is for the first frame. The reason for this lies in firmware operation. As we discussed in Section A.2.1 and as depicted in Figure A.5, the firmware on the wireless chipset is constantly performing its normal IEEE 802.11 operation and the monitoring work is an additional task. The normal operation of the card requires it to sometimes halt until a certain even occurs. Scanning the firmware code, we often find code snippets whose only purpose is to delay the state machine. Listing A.1 is an example of such code. These delays, along with delays associated with the additional code, may sometimes cause a delayed capture of specific events. In some cases these events might be so quick that we might just miss them. We believe this is the reason why the SIFS period is missing from the timeline.

Listing A.1: A delay loop in OpenFWWF

```
                add    SPR_TSF_WORD0, 16, GP_REG5
        tx_frame_wait_16us:
```

```
jext  COND_TX_DONE, state_machine_idle
jne   SPR_TSF_WORD0, GP_REG5, tx_frame_wait_16us
```

While these delays do not impair our debugging capabilities, but their effect on the monitoring work can be mitigated by checking the *IFS Status* update within each of these loops, depending on how precise we want our event capturing to be. For normal studies such as those we do in this chapter, it is not necessary to complicate the firmware code by running our code in any other place than the idle state.

The diagnostic tool's front-end can also measure time distance between two events (top image in Figure A.6) to help you to further determine the nature of those events, or mark a larger time frame to perform automated studies. The application holds a full history of the data from the time it connects to the server. This combined with zooming and panning functionalities help you get various statistical data over large periods of time. The downside is that this will require sufficient memory on the host machine. But since the client is a separate piece of software, we can run it on a powerful machine. The backup solution is to purge relatively old historical data periodically, which the software does if it is short on memory.

A.2.3.2 Additional Functionality

Additional functionality can be added to WiFo through a plugin system with APIs for both .NET Framework and Python. Placing a .NET class library or a Python script in WiFo's extensions directory will automatically activate it on start-up. Extension developers do not need to worry about collection of data, as it is passed in an accessible data structure to an extension; this is a special-purpose enumerable list in .NET, and a list in Python. Each extension can have its own settings and output formats, which can be integrated into the user interface through the API.

There are currently two types of extensions. *Studies* are extensions that take a subset of the data identified by a time range and produce results. These results can be of any type and the API provides a flexible format to display results. The bottom image in Figure A.6 shows an example of results produced by a study. Results in the form of a plot can easily be displayed through the API.

Listings A.2 and A.3 show a very basic study extension written using the two available APIs. It simply asks for an index from the user and shows the details of the status record at that index. Figure A.8 shows the resulting GUI.

Listing A.2: C# for the 'SimpleStudy' extension

```
        public class SimpleStudy : IStudy
        {
                [Browsable(false)]
                public string DisplayName
                {
                        get { return "Simple Study"; }
                }

                [Browsable(false)]
                public string Author
                {
                        get { return "Hessan Feghhi"; }
                }

                public void Perform(RecordList records, IWiFoContext wifo)
                {
                        int? i = wifo.AskInt("Enter a record number", 0);

                        if (i != null)
                                wifo.ShowResults("Record Information",
                                    records[(int)i]);
                }
        }
```

Listing A.3: Python code for the 'SimpleStudy' extension

```
def displayname():
  return 'Simple Study'

def author():
  return 'Hessan Feghhi'

def perform():
  i = wifo.askint("Enter a record number")

  if not i:
    return

  results = {}
  results['Time'] = wifo.data[i].time
  results['State'] = wifo.data[i].state
  wifo.dictbox(results)
```

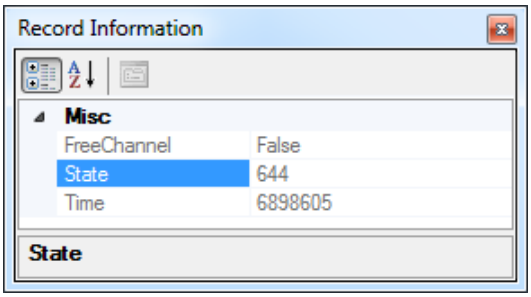


Figure A.8: Example of a result displayed by the 'SimpleStudy' plugin

Timeline view extensions are replacements for the original timeline view. They have access to the graphics canvas of the timeline and the state information, and they can offer a different representation of the existing data, or combine it with external data sources to give more insight. For performance reasons, the API for timeline views is currently only available for .NET Framework.

WiFo provides some default extensions, including packet recognition and an inter-frame space (IFS) distribution calculator. The former simply uses the state information to count packets in a given time frame and provides statistics (e.g., bottom image in Figure A.6). The latter generates a bar plot for the distribution of the inter-frame space (e.g. Figure A.13). We will discuss these plots in detail in Section A.4. For example, recognizing successfully-received packets is performed by scanning through all records and looking for the following pattern:

1. RX engine becomes busy for longer than $192\mu s$
2. RX engine becomes idle
3. TX engine becomes busy after $10\mu s$
4. TX engine becomes idle (after an ACK duration)

This pattern represents the transmission and acknowledgement of a single frame if WiFo's back-end runs on the receiving access point. For other receive cases, the final two steps examine the RX bit. In practice, WiFo checks both cases. Note also that these two steps will also be absent if the frame is not acknowledged. The inter-frame space can be calculated as the time difference

from one match of this pattern to the next. Starting from the first two, the time differences between the above steps correspond to frame duration, SIFS and ACK duration respectively.

A.2.4 Linking to PCAP Data

When studying network information, tools such as *tcpdump* are invaluable. These tools monitor the medium through a network adapter and capture frames as they are observed. The type and important parts of every frame are recorded and can later be filtered and studied.

There are differences between traffic capturing software and our diagnostic tool. One of the differences is that tools like *tcpdump* only capture frames that are transmitted successfully or at least key parts of which can be decoded. Our system on the other hand captures every activity on the channel, whether or not it is a successful frame. Even the noise from a microwave oven (MWO) can be seen using our tool as it triggers the decoder on the wireless adapter (see Section A.3.3).

Another important difference between our diagnostic tool and capturing tools, which is a disadvantage of ours, is that our tool only sees state changes. While this information is enough for determining where packets are situated on the timeline, it cannot tell us who each packet belongs to and who it is destined to. This makes our system less useful if the network has multiple stations. On the contrary, capturing software does give us this extra information along with everything else we need to know about the packets. We can even set them to record full-sized packets so we can read application-level data as well. If we use radiotap headers, we can access further information such as receiver power levels and modulation.

Each of these systems have their own advantages as we discussed. With the positive aspects of both put together, we can have a complete debugging tool that can observe every aspect of the medium. That is what we do in our final system. Tools like *tcpdump* and *tshark* save captured data in a common format called PCAP. There are APIs available in almost every programming language to read from and write to this format. We use the API for .NET Framework[4] to develop a PCAP extension for the system. This extension displays the

[4]SharpPCAP/PacketDotNet

PCAP information on the timeline, by aligning TSF and timestamp values. [5] This helps verify frame transmissions and have extra information to perform further analysis.

PCAP data used in our tool can come from any source and we usually use a second wireless adapter on the same machine as our main wireless receiver just to capture. The reason why we do not use the same card for both tasks is that our open-source firmware does not provide *promiscuous mode*, which is a requirement for capturing network packets. This also avoids burdening the firmware with additional work, which might result in missing more events.

A.3 Validation

Before we begin debugging wireless adapters, we need to make sure our system is robust and captures IEEE 802.11 signals correctly. We can examine many aspects of the system. In this section we present the most important validation tests we performed on our system.

A.3.1 TX Duration

One of the fundamental aspects of the system is the timing of flag changes, the correctness of which is crucial to any application of the diagnostic tool. To this end, we verify the effect of varying frame sizes on the observed duration of their transmission. We run a series of tests with different payload sizes, from 100 bytes to 1400 bytes (with granularity of 100 bytes). The diagnostic tool measures the duration of frames by using the pattern described in Section A.2.3 and measuring the time the RX engine remains busy for that frame.

We use saturated UDP traffic at $11Mb/s$ for all tests, and for each payload size we average transmission duration over 2500 frames. We use long preamble in these tests ($192\mu s$ PLCP). We also calculate the expected duration $D(l)$ for each payload size l as

$$D(l) = d_{PLCP} + d_f(l). \tag{A.1}$$

[5] BCM43xx chipset internally uses a TSF timer that is never synchronized with the network. Instead, a register keeps the difference between the internal and network TSF, the value of which is updated each time a beacon is received. We use this value to align signal records, which hold the internal TSF, with TSF values from PCAP data.

where $d_{PLCP} = 192\mu s$ is the duration of the preamble and the PLCP header and $d_f(l)$ is the time required to transmit the payload and protocol headers associated with different layers, which is calculated as

$$d_f(l) = \frac{8(l + l_{H,LLC} + l_{H,IP} + l_{H,UDP} + l_{H,802.11} + l_{FCS})}{r}.$$

Here $l_{H,x}$ is the length of the header associated with layer x, l_{FCS} is the length of the FCS, all in bytes, and r is the data rate used to transmit the frame. For example, for a payload size of 1000 and with our experiment settings, $d_f(1000) = \frac{8(1000+4+20+8+30+4)}{11\times10^6} \approx 7.75 \times 10^{-4}s$. Further, this translates to the following for $D(l)$ in microseconds:

$$D(l) = 240 + 0.727l. \tag{A.2}$$

Figure A.9 shows average durations as observed by the AP as well as the calculated expected duration for each payload size. In fact, the measured packet lengths are tightly clustered around the mean, with variations of only a single microsecond. As shown in the figure, the expected and observed values closely match. Fitting a straight line through measured data gives us the following for $D(l)$:

$$238.055 + 0.725l.$$

which is very close to (A.2). Slight changes are expected given firmware delays. This verifies the system's pattern matching capabilities and the timeliness of flag changes.

A.3.2 Throughput

Throughput is an important metric for IEEE 802.11 networks, as it can be an indicator of different network aspects such as performance and fairness. For this reason, it is important for our diagnostic tool to identify all transmitted frames and to measure network throughput correctly.

To validate the diagnostic tool's throughput calculation, we run a fresh test on the same network as previously described, but this time we do not use saturated traffic. Instead, we run a single UDP flow with PHY data rate of $11Mb/s$ with payload size of 1400 bytes for each frame for 30 seconds, and have the tool calculate the throughput. We use a arrival rate of 100 packets/s

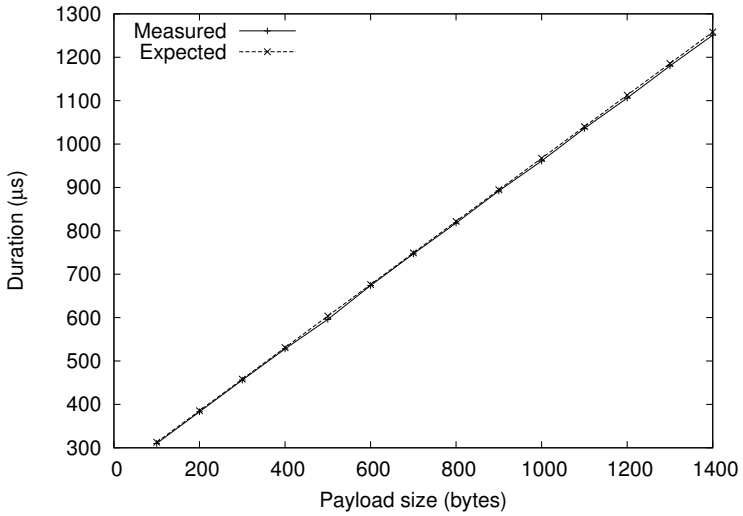


Figure A.9: Duration with increasing payload size.

for the first test, and for each subsequent test we increase the arrival rate, up to 800 packets/s.

The expected throughput is calculated simply by multiplying the transmission rate by the payload size. However, rate should not exceed the saturation throughput. We calculate saturation throughput S as

$$S = \frac{1}{D(l) + D_{SIFS} + D_{ACK} + D_{DIFS} + D_{bo}} = \frac{1}{D_t}, \tag{A.3}$$

where $D(l)$ comes from (A.1), D_{ACK} is the duration of the ACK, D_{bo} is the average backoff duration, and D_{DIFS} is the duration of DIFS ($50\mu s$). D_t is used to denote the total duration of a successful frame transmission, including its ACK and average backoff. We will be using this notation later. Also, note that in our experiments we use a fixed packet size l and hence $D(l)$ is a constant.

Figure A.10 depicts the observed and expected throughput values. As you can see, the two values match closely; and the small difference after saturation may be explained by the backoff behavior of the chipset as we will discuss in Section A.4. The reason this difference does not exist before saturation is that packet inter-arrival times are usually larger than the maximum backoff, eliminating the effect of the backoff mechanism.

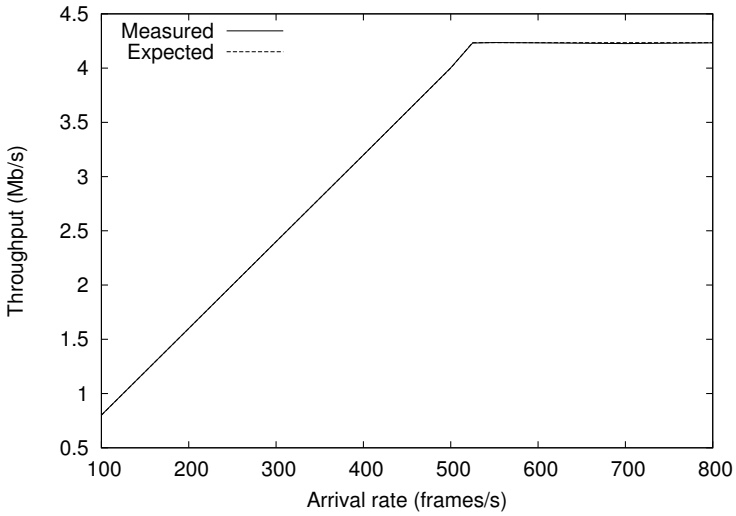


Figure A.10: Throughput with increasing packet-arrival rate.

A.3.3 Microwave Interference

In this section we show that we can see microwave interference using our tool. To demonstrate this, we test two scenarios on a channel where only the AP transmits beacons and there are no other stations or any interference from other Wi-Fi networks. In our first test we use a microwave oven to introduce interference, and for the second test we turn it off. Note, the interference generated by the oven is bursty. The top image in Figure A.11 shows a single microwave oven burst captured by a spectrum analyzer. The duration of this pattern is approximately $8ms$ and it is repeated at intervals of about $20ms$ (see bottom Figure A.11), which is related to the mains frequency of 50Hz.[6]

Figure A.12 shows what we observe using WiFo for both tests. The bottom plot is taken on a free channel with only beacons, which show as periodic spikes. Note that as the monitoring runs on AP the TX engine remains busy during the transmission of a beacon. The top plot is taken with running microwave oven. There are periodic spikes on the RX engine's activity. The duration of these spikes is around $140\mu s$, which is the time required by the decoder to distinguish noise from Wi-Fi signal. The RX spikes are separated by two slightly different distances which alternate. The shortest distance between the spikes is $8ms$, which is the burst size. This suggests that the

[6]These observations are made with kitchen microwave ovens. Commercial ones that are used in restaurants, as stated in [154], make pulses twice as frequent as the ones we use. Moreover, the resulting interference has different characteristics.

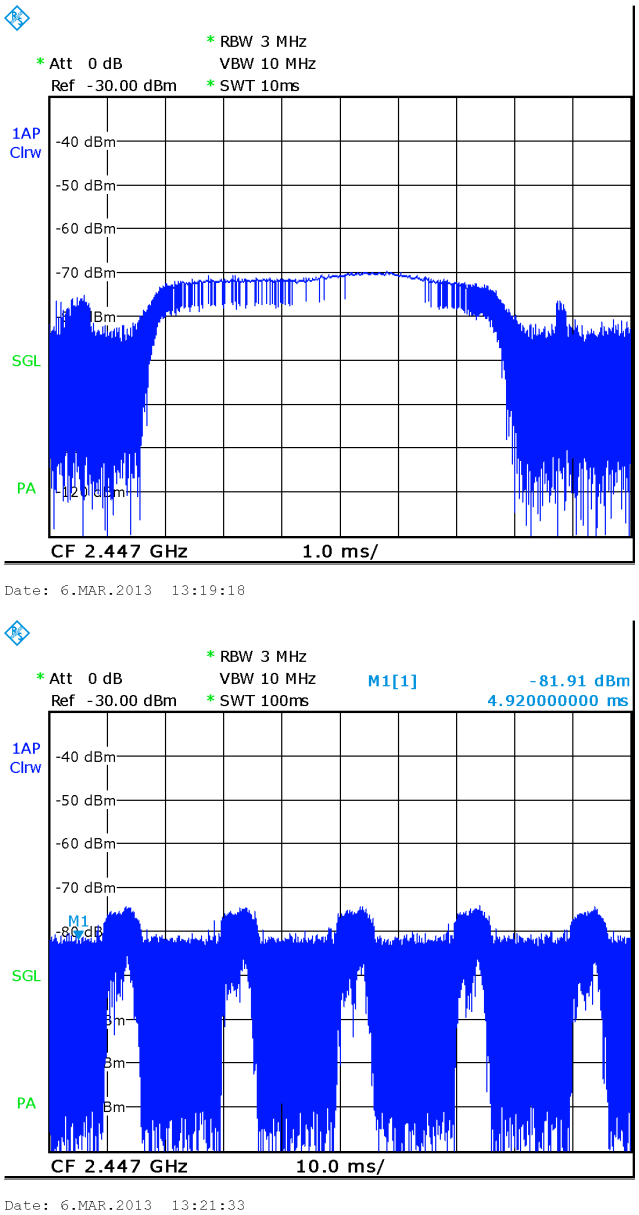


Figure A.11: Waveform of a single microwave oven burst (top), and a sequence of bursts (bottom).

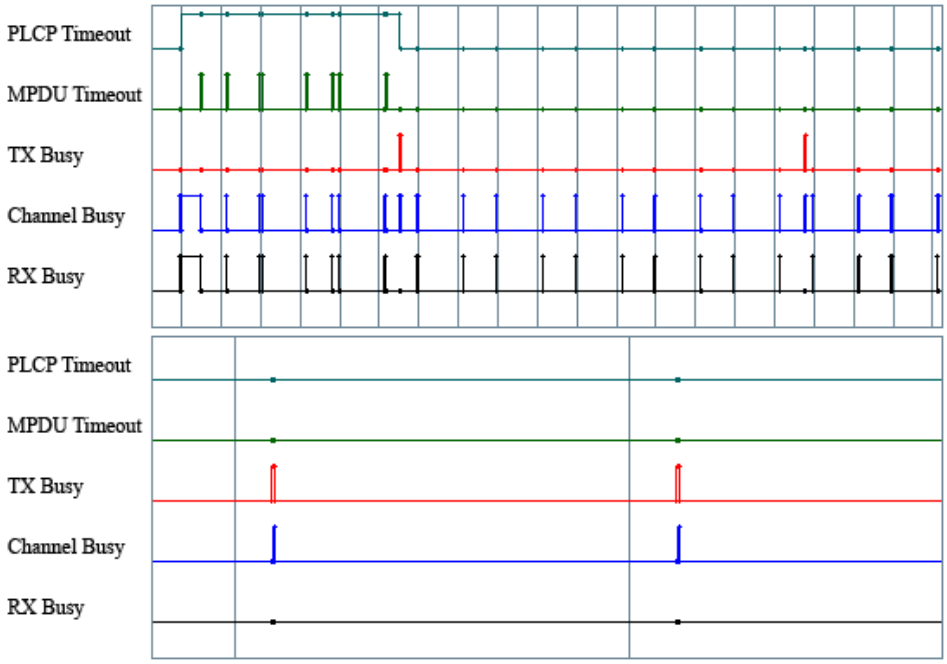


Figure A.12: Channel activity observed by AP running WiFo's backend when a microwave oven is working (top) and when it is turned off (bottom). Vertical lines are $10ms$ apart in both images.

beginning and the end of each burst triggers the chipset's decoder, which soon identifies it as noise and the decoder is deactivated. More generally, any interference on the channel triggers the decoder temporarily.

A.4 Debugging Examples

In this section we introduce a few examples of how our diagnostic tool can help debug wireless hardware. The main purpose of this tool is to help examine wireless networks closely and figure out deviations from standard and reasons behind unexpected behaviors. This is especially useful for driver and firmware developers, as it can help them verify the performance of the wireless cards and debug their implementations. It can also be a good practical research tool as it can visualize information that is otherwise hard to notice in a wireless network.

A.4.1 Contention Window

One of the most important parts of the standards which ensures equal opportunities for all the stations to use the channel is the contention window. The

contention window is sometimes misconfigured by wireless card vendors [72] and it is also the easiest to tweak as we discussed in Chapter 5. This makes the ability to detect possible misconfiguration or misbehavior very important. In this section we manually adjust the contention windows of the wireless adapters to standard and non-standard values and then use our system to observe the differences.

The diagnostic tool can scan through the received signals and measure the inter-frame space. Through this value it calculates the backoff slots the station chooses to wait and generates a distribution graph. We use this feature to see whether we can detect misconfigured stations.

In our next experiments, we use the same network setup that we used in Section A.3 and we send saturated UDP traffic using packets with 1000 bytes of payload, and we alter the contention window for each experiment. The duration of each experiment is again 30 seconds. Figure A.13 shows the resulting plots for three different values of the minimum contention window, namely 8, 16 and 32. We can see the number of times each backoff value is selected and the range of values in use.

Note that as we use IEEE 802.11 channel 14 for our experiments and we have little to no interference, the station almost never moves on to the second backoff stage, and what we have here is only the first backoff stage. It is also worth noting that observed values for the same backoff value are often $1\mu s$ apart as a result of firmware delays. We bin the results into $20\mu s$ bins to get a cleaner image, but even if we did not do that, they would appear as isolated spike groups rather than spread all over the time frame.

Using these plots we can easily distinguish where the wrong CW_{min} is selected. They can also help us see how evenly the backoff is chosen. As this value should be chosen completely at random, we expect a flat distribution graph. Although what we see in the plots are well distributed, they are not completely even. This could be due to the way the RNG works on the device, and it could also be the result of the small delays we mentioned previously. Table A.3 shows the chi-squared test values for these results, comparing them to uniform distribution (the null hypothesis is that the results are not uniform). As the table shows, p-values are too large, so it seems unlikely that the backoffs are truly uniform.

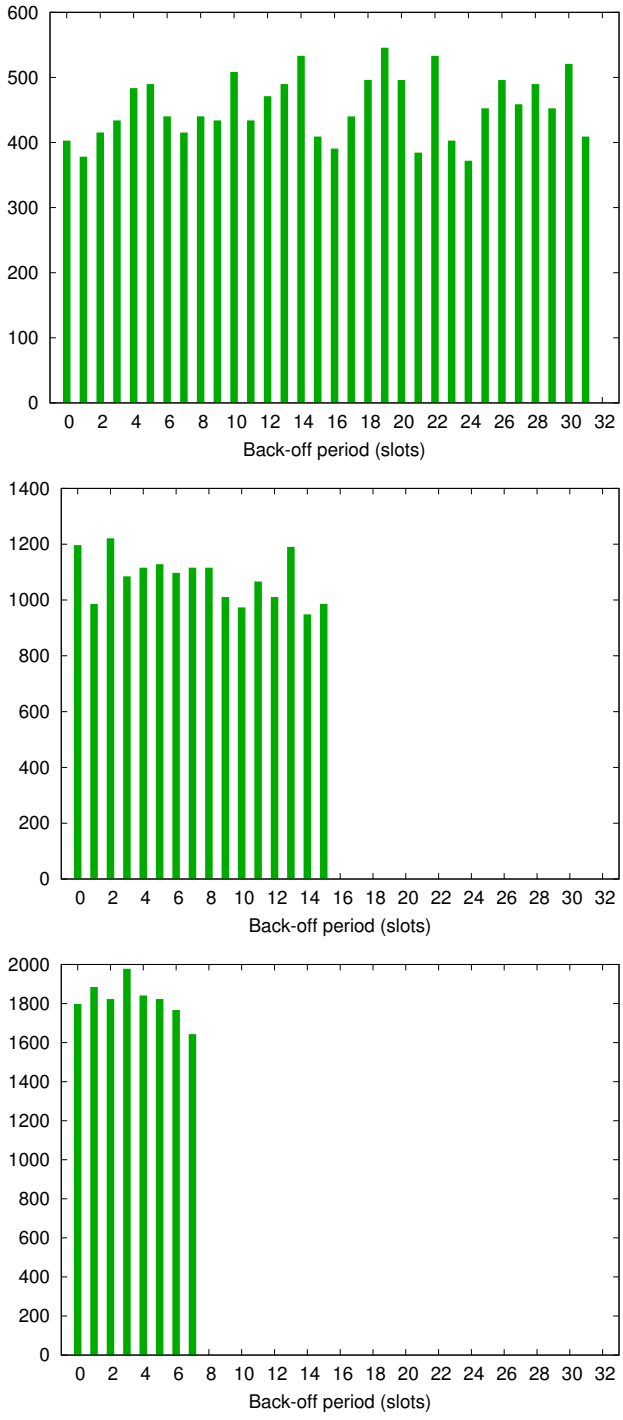


Figure A.13: Backoff distribution for $CW_{min} = 32$ (top), $CW_{min} = 16$ (middle), and $CW_{min} = 8$ (bottom), as recorded by our diagnostic tool. STA uses Broadcom BCM4318 chipset with b43 driver and OpenFWWF.

CW_{min}	χ^2	p-value
32	26.3006	0.70680
16	16.2286	0.36702
8	5.6899	0.57639

Table A.3: Chi-squared test for results in Figure A.13. We use $CW_{min} - 1$ degrees of freedom to obtain the p-value.

Standard-compliance is not solely the role of hardware and firmware, and differences could exist in the driver level. In the next study we use an Atheros mini-PCI wireless adapter for our station and compare the backoff behavior when using different drivers. One of the drivers we use is ath5k[7], which is a reliable driver for Atheros cards, and the other one is MadWifi[8].

Both drivers provide a similar average throughput, from which one might guess they both present a similar backoff distribution. However, our observations prove otherwise. Figure A.14 shows the results obtained using the diagnostic tool. As you can see in the figure, the ath5k driver hops between the two ends of the contention window rather than a uniform distribution over the whole window. According to [155], this does not give the station any advantage in the long run, as the average backoff is unchanged. Nevertheless, it is an obvious deviation from the standard and it may affect certain experiments by changing the collision probability, especially when more than one station behaves this way.

A.4.2 TXOP Burst

We introduced TXOP in Section 2.4. It is as a bounded time interval during which a station can send as many frames as possible. Although TXOP is advertised by the access point and stations can requested an RTS frame, there is no central control on how stations use it. Once a station wins the contention, it can practically send frames indefinitely, resulting in poor performance of other contending stations. We actually discussed this as a form of attack in Chapter 5. TXOP should normally not be used unless allowed by the AP or within a contention-free period.

[7]Our ath5k driver is slightly modified to enable us change contention parameters. However, we do not use this feature for these tests and the rest of the driver code is unchanged.

[8]The MadWifi version we use is 0.9.4-r4173, and the only modification made to the driver is disabling QoS.

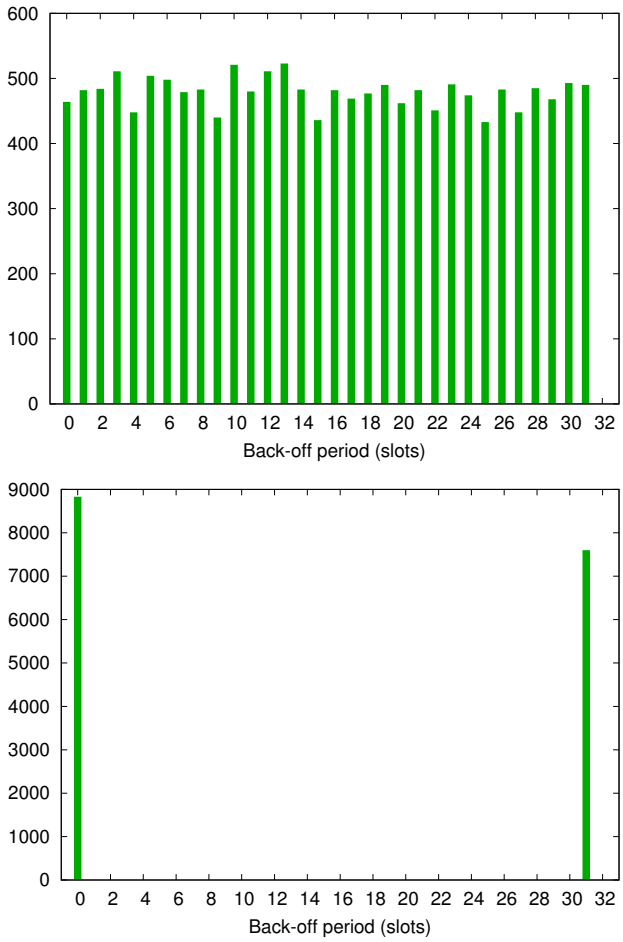


Figure A.14: Backoff distribution for Atheros chipset provided by MadWifi (top) and ath5k (bottom)

In our next experiment we demonstrate our system's ability to detect TXOP bursts. For this test we increase the TXOP period, and use the diagnostic tool to count the number of packets that come in a single burst. Our network has only one station connected to an AP equipped with the diagnostic tool's monitoring code. The station is equipped with an Atheros card with the MadWifi driver, and it transmits saturated traffic using frames with payload size of 500 bytes for 5 seconds in each test. Figure A.15 depicts the results.

As the figure shows, as long as the TXOP period is smaller than the time required to transmit one packet, the burst contains only one frame, i.e. there is no burst. Each time a new frame can fit in the given period, the burst size increases. In other words, we can calculate burst size using the following

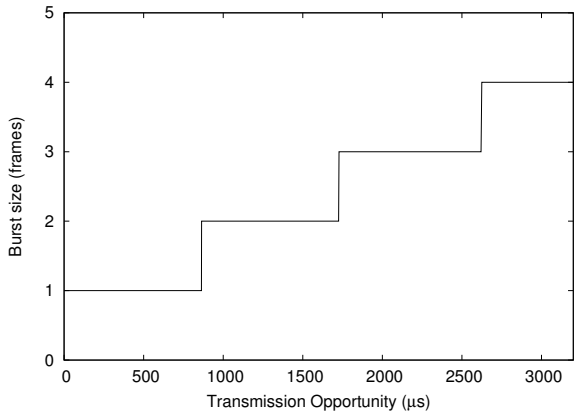


Figure A.15: The number of frames in a burst with increasing TXOP. STA uses Atheros chipset with ath5k driver.

formula:

$$n = \lceil \frac{t_{TXOP}}{D_t} \rceil$$

Where t_{TXOP} is the TXOP time, and D_t comes from (A.3).

We can also measure the duration of a single frame transmission as the distance between two bumps of n, which is measured $863\mu s$ based on the results used to plot Figure A.15.

A.4.3 ACK Skipping

Acknowledgements are normally used as a success signal for the transmitter. However, deliberately skipping ACKs can sometimes be desired, e.g. [109, 110] or what we did in Chapter 5. In this section we implement a simple scheme at the AP: we skip every other ACK for received frames, forcing stations to always make two attempts for each frame. We use WiFo to sanity check our implementation. For the experiment, we use one station connected to the AP, and send saturated traffic at 11Mb/s for 30 seconds using MGEN. Both the station and the AP use Broadcom BCM4318 wireless adapters with OpenFWWF. The station uses minimum CW of 32. By dropping the first ACK, we force it to double this value, and use 64. Figure A.16 shows the resulting backoff distribution graph calculated by WiFo. For values in the range $[0, 32)$, the numbers are almost twice as much as $[32, 64)$, which is exactly

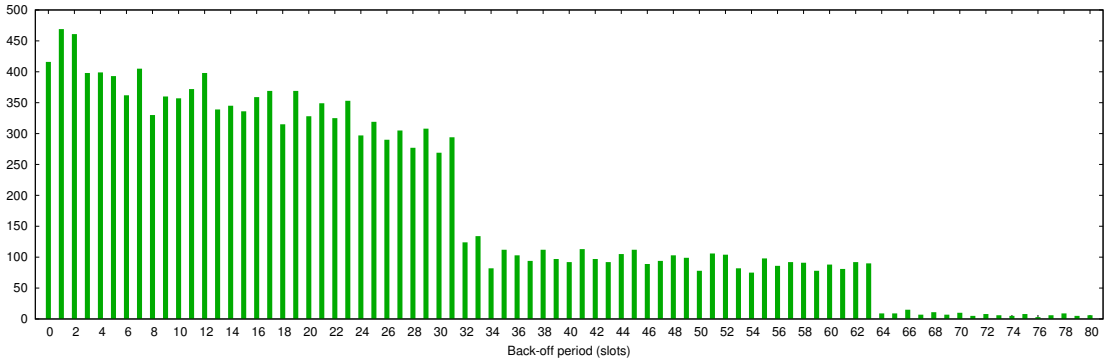


Figure A.16: Backoff distribution when the AP drops every other ACK.

what we expect; remember that, on the second backoff stage, the station uniformly selects a backoff time within the range of $[0, 64)$ slots. This not only proves that the implementation works, but also demonstrates another aspect of WiFo's usefulness.

A.5 Conclusion

In this appendix we designed an extensible diagnostic tool for IEEE 802.11 wireless cards which can be used to test various aspects of the protocol and detect standard compliance. The purpose of this tool is to give programmers and researchers enough flexibility to test and debug wireless cards and drivers. The diagnostic tool is made using only commercial off-the-shelf devices and it can be more practical than its more expensive alternatives. With an API to add new features to the application, the diagnostic tool can be virtually programmed to do sophisticated analyses on the data. We presented PCAP integration as an example of additional features that can be plugged in to the system. We presented results to validate the sanity and reliability of our system, and presented some experimental results that highlighted some of possible use-cases of this tool.

APPENDIX B

Firmware and Driver Code

This appendix provides samples of the code used to implement the policing algorithm, and the Virtual MAC, with brief explanation of design decisions.

B.1 Introduction

Implementing algorithms on hardware and testing them can be a time-consuming task. Many of the implementations described in this thesis are done by modifying the C code for the Linux drivers of the wireless adapters, or the assembly code for their firmware. In order to facilitate the reproduction of the present work, this appendix provides important pieces of the code used for each part of the policing code. As we mentioned in Chapter 6, each algorithm is broken into two components, one running in the driver, and one on the wireless adapter's firmware. Here we describe how each component of the system is implemented.

B.2 Driver Implementation

The policing controller and the Virtual MAC both involve recurring events that happen periodically. They only need to run on the driver once per iteration. Fortunately, the b43 driver provides a mechanism to schedule periodic works. This is performed using the `do_periodic_work` function in the main.c file of the driver. We modified this function, and inserted our own schedules into it. The original version of the function looks like this:

```
static void do_periodic_work(struct b43_wldev *dev)
{
	unsigned int state;

	state = dev->periodic_state;
	if (state % 4 == 0)
		b43_periodic_every60sec(dev);
	if (state % 2 == 0)
		b43_periodic_every30sec(dev);
	b43_periodic_every15sec(dev);
}
```

As one can see, the granularity of the period is not enough to run a task more frequently than every 15 seconds. To increase this granularity, we modify the `b43_periodic_work_handler` function, and insert the following code when it calculates the delay:

```
delay = round_jiffies_relative(HZ * 1); // The default driver uses 15
```

We would then need to adapt the `do_periodic_work`, and schedule the policing algorithm controller:

```
static void do_periodic_work(struct b43_wldev *dev)
{
	unsigned int state;

	state = dev->periodic_state;
	if (state % 60 == 0)
		b43_periodic_every60sec(dev);
	if (state % 30 == 0)
		b43_periodic_every30sec(dev);
	if (state % 15 == 0)
		b43_periodic_every15sec(dev);
	policing_iteration(dev);
}
```

Then we can define the policing iteration function, and implement the controller, and the Virtual MAC.

B.2.1 Importing Information

Unlike the driver, the firmware is always working, logging information to be used by the driver. So, in each iteration, the first thing we do is to fetch this

information from the shared memory, and store them locally. This code goes into the `policing_iteration` method.

```
u16 u;
unsigned int i, slist_end, ix = 0;
struct policing_per_sta* temp_sta_info;

if (min_samples == 0)
	return;

// Read the position of the end of the station list
slist_end = b43_shm_read16(dev, B43_SHM_SHARED, POLICE_SLIST_END);

/*
 * Read in accumulate special-purpose counters from the shared memory.
 * Each counter represents a measure (such as busy slots, dropped packet count, etc)
 */
for (i = 0; i < POLICE_CTR_COUNT; i++) {
	u16 temp_addr = POLICE_CTR1_HI + i * 4;
	long police_ctr;
	police_ctr = (unsigned int)b43_shm_read16(dev, B43_SHM_SHARED, temp_addr);
	u = b43_shm_read16(dev, B43_SHM_SHARED, temp_addr + 2);
	b43_shm_write32(dev, B43_SHM_SHARED, temp_addr, (u16)0);
	police_ctr = (police_ctr << 16) + (unsigned int)u;
	saved_police_ctr[i] += police_ctr;
}

curSamples = (curSamples + 1) \% min_samples;

// If the iteration interval (in seconds) has passed
if (curSamples == 0) {
	// TODO: Run policing and the Virtual MAC!
}
```

The constant `POLICE_SLIST_END` above is the shared memory location of the word where the location of the last station item is stored. Next we begin filling in the TODO part.

B.2.2 Policing

The first thing we need to do in order to execute the policing controller is to fetch station data from the shared memory.

```
// Go through the list (starting from the constant location POLICE_SLIST_BEGIN).
while (i < slist_end) {
	ix = (i - POLICE_SLIST_BEGIN) >> 3;
	temp_sta_info = sta_info + ix;
	temp_sta_info->pk_ptr = i;
	temp_sta_info->retries = b43_shm_read16(dev, B43_SHM_SHARED, i + 2);
```

```
        temp_sta_info->attempts = b43_shm_read16(dev, B43_SHM_SHARED, i + 4);
        temp_sta_info->packets = b43_shm_read16(dev, B43_SHM_SHARED, i + 6);
        all_packets += temp_sta_info->packets;
        b43_shm_write16(dev, B43_SHM_SHARED, i + 2, 0);
        b43_shm_write32(dev, B43_SHM_SHARED, i + 4, 0);
        i += 8;
}

ix++; // So it contains the count

// TODO: Use Virtual MAC to estimate the compliant attempt rate.
```

We get to the virtual MAC in the next subsection. Here we assume that the estimate is ready, and stored in the variable `p_Sf`. The controller has now all the required information to perform.

```
for (i = 0; i < ix; i++) {
        sf->data[i] = sta_info[i].attempts;

        if (p_Sf > 0 && sta_info[i].attempts > 0) {
                // Now calculate the new PK
                int pk = sta_info[i].pk;
                pk = pk + ALPHA_NOM * ((long)hflt_div(sta_info[i].attempts - HFLOAT_ONE) /
                     ALPHA_DENOM;

                /*
                 * Limit the calculated p value to the range [0, 1], and write it to the
                 * shared memory for firmware use
                 */
                if (pk < 0) pk = 0;
                sta_info[i].pk = pk;
                if (pk > 0xFFFE) pk = 0xFFFE;
                b43_shm_write16(dev, B43_SHM_SHARED, sta_info[i].pk_ptr, (u16)pk);
        }
}
```

The two constants `ALPHA_NUM` and `ALPHA_DENOM` together represent the α value for the algorithm. Note that floating point numbers are not allowed in Linux drivers. Also, the `pk` member of the station information structure represents the P_{ACK} value for the station. That is why we have also implemented a special float handling that corresponds the values between 0 and 65535 to the floating point range [0, 1]. The code for these functions will be listed at the end of this appendix.

B.2.3 Virtual MAC

In Chapter 6 we discussed how the Virtual MAC estimates the compliant attempt rate. Here we provide the actual code. The driver component in this implementation is responsible for using the information collected in the firmware to compute the estimate. The code below replaces he TODO line in the policing code presented earlier.

```
#define BUSY_SLOTS saved_police_ctr[0]
#define IDLE_TIME saved_police_ctr[1]
#define BEACONS saved_police_ctr[2]
#define DROPPED_PACKETS saved_police_ctr[3]

unsigned long total_slots, _tau, _p;

total_slots = BUSY_SLOTS + (IDLE_TIME) / 20 - BEACONS;

_p = hflt_div(BUSY_SLOTS, total_slots);

/*
 * The tau formula from Bianchi's paper
 * (we have two versions as we cannot have negatives)
 */
if (2 * _p > HFLOAT_ONE)
       _tau = 2 * hflt_div(2 * _p - HFLOAT_ONE,
              (2 * _p - HFLOAT_ONE) * 33 + 32 * hflt_mul(_p, (hflt_pow(2 * _p, 5) -
                  HFLOAT_ONE)));
else _tau = 2 * hflt_div(HFLOAT_ONE - 2 * _p,
              (HFLOAT_ONE - 2 * _p) * 33 + 32 * hflt_mul(_p, (HFLOAT_ONE - hflt_pow(2 * _p,
                  5))));

p_Sf = FIX_MULTIPLIER_NOM * (hflt_mul(hflt_mul(_tau, HFLOAT_ONE - _p), total_slots)) / 100;

// Reset the counters for the next iteration
for (i = 0; i < POLICE_CTR_COUNT; i++) {
       saved_police_ctr[i] = 0;
}
```

B.3 Firmware Implementation

Firmware implementation is more difficult because it is in assembly, and there are far less hardware capabilities available to use. However, the firmware is the last point from where frames are transmitted, and the first point of frame reception. This is why things such as ACK generation and frame counting are best implemented on the firmware. In this section we present the code for the firmware part of the policing algorithm and the Virtual MAC.

B.3.1 Policing

The policing algorithm implementation on the firmware involves mainly ACK transmission decision, and that is where it resides. However, there are helper functions it uses in its operation. One of these functions is a lookup function used to fetch a record from the hash table described in Chapter 6. Below is this function. The code is in the assembly of the BCM B43xx chipset (revision 5).

```
// HANDLER: policing_lookup
// PURPOSE: Looks up a MAC address in the STA list, or adds it if non-existent

policing_lookup:
       srx 7, 0, POLICE_M3, 0x000, POLICE_TEMP1
       srx 7, 8, POLICE_M3, 0x000, POLICE_M3
       xor POLICE_M3, POLICE_TEMP1, POLICE_TEMP1

       mov POLICE_SLIST_INDEX, POLICE_TEMP2

       add POLICE_TEMP2, POLICE_TEMP1, POLICE_OFFSET
       je [0x0, POLICE_OFFSET], 0, policing_create_item
       mov [0x0, POLICE_OFFSET], POLICE_OFFSET
       jext COND_TRUE, policing_lookup_complete

policing_create_item:;
       mov POLICE_SLIST_POS, [0x0, POLICE_OFFSET]
       mov POLICE_SLIST_POS, POLICE_OFFSET
       add POLICE_SLIST_POS, 4, POLICE_SLIST_POS
       sl POLICE_SLIST_POS, 1, [POLICE_SLIST_END]
       mov 0x000, [0x0, POLICE_OFFSET]
       jext COND_TRUE, policing_lookup_complete
```

The other helper function is used to discard a frame. It is used when we need to refuse to send an ACK.

```
// HANDLER: policing_discard
// PURPOSE: Discards a received frame (without sending an ACK)

policing_discard:
       /*
        * The following two lines increment the discarded frame counter
        * as a 2-word (32-bit) value.
        */
       add. [POLICE_CTR4_LO], 1, [POLICE_CTR4_LO]
       addc [POLICE_CTR4_HI], 0, [POLICE_CTR4_HI]
       jext COND_TRUE, rx_discard_frame
```

With these helper functions, the policing algorithm firmware has enough information to operate. The code we are going to present runs when a new frame is received, and before an acknowledgement is created. This would be directly after the following two lines in the OpenFWWF[67] code.

```
send_response:
       jext COND_RX_ERROR, rx_complete
```

And the code itself looks like the following. Descriptive comments are added to the code to aid comprehension.

```
       // Save the MAC address of the sender in our registers
       or [RX_FRAME_ADDR2_1,off1], 0x000, POLICE_M1
       or [RX_FRAME_ADDR2_2,off1], 0x000, POLICE_M2
       or [RX_FRAME_ADDR2_3,off1], 0x000, POLICE_M3

       // Backup offset register (a shared use global pointer)
       mov POLICE_OFFSET, POLICE_TEMP0

       // Call the policing lookup/create helper function
       jext COND_TRUE, policing_lookup
policing_lookup_complete:;
       // Don't discard if not a data frame
       jzx 0, 2, POLICE_FLAGS, 0x000, policing_dont_discard
       nand POLICE_FLAGS, 0x4, POLICE_FLAGS
       add [0x2, POLICE_OFFSET], 1, [0x2, POLICE_OFFSET]

       // Save PK in temp1
       mov [0x0, POLICE_OFFSET], POLICE_TEMP1

       // If PK >= RND then continue to call the discard frame helper function
       jle POLICE_TEMP1, SPR_TSF_Random, policing_dont_discard

       // Restore offset register
       mov POLICE_TEMP0, POLICE_OFFSET
       jext COND_TRUE, policing_discard
policing_dont_discard:;
       // Add 1 to STA's sent packet count
       add [0x3, POLICE_OFFSET], 1, [0x3, POLICE_OFFSET]

       // Restore offset register
       mov POLICE_TEMP0, POLICE_OFFSET
```

B.3.2 Virtual MAC

The firmware part of the Virtual MAC is responsible for counting the number of idle and busy slots. This is handled in the firmware's idle state logic.

Normally, the firmware sleeps for a while when it gets to this state, before polling for new events:

```
       orx 7, 8, 0x0FF, 0x0FF, SPR_MAC_MAX_NAP
       nap

state_machine_start:
```

We remove the first two lines to buy time for the Virtual MAC. Then we call our Virtual MAC estimator function under state_machine_start. Below is the body of this function. Descriptive comments are added to the code for better readability.

```
// HANDLER: hessan_vmac
// PURPOSE: Run the Virtual MAC code and return

hessan_vmac:
       // Are we initialized?
       jne r62, 0, vmac_inited;
       // Set up our start time variable
       mov SPR_TSF_WORD0, EST_START_TIME0;
       mov SPR_TSF_WORD1, EST_START_TIME1;
       // CurrentState = INVALID
       mov 0xFF, EST_CURRENT_STATE;
       // Set initialized
       or r62, 1, r62;
vmac_inited:
       // Temp1 = Channel idle flag
       and SPR_IFS_STAT, 0x1, POLICE_TEMP1

       // if(CurrentState == Temp1) return
       je POLICE_TEMP1, EST_CURRENT_STATE, tracker_end;

       // Backup = CurrentState
       mov EST_CURRENT_STATE, POLICE_TEMP3;

       // CurrentState = Temp
       mov POLICE_TEMP1, EST_CURRENT_STATE;

       // Calculate time difference
       sub. SPR_TSF_WORD0, EST_START_TIME0, POLICE_TEMP0
       subc SPR_TSF_WORD1, EST_START_TIME1, POLICE_TEMP1

       // If difference > 65535 (the second word is non-zero), it is long enough!
       jg POLICE_TEMP1, 0, vmac_not_too_short;
       // If difference > 49 it is long enough (idle)
       jg POLICE_TEMP0, 50, vmac_not_too_short;

vmac_too_short:
       // This part is to ignore short spikes
```

```
        je POLICE_TEMP3, 0, vmac_too_short_busy;
        jext COND_TRUE, final_step;
vmac_too_short_busy:
        // Jump to final step
        jext COND_TRUE, final_step;
vmac_not_too_short:
        // If LastState is the same as Backup, it is a continuation.
        je EST_LAST_STATE, POLICE_TEMP3, vmac_too_short;

        // If we just ended a busy slot (Backup == BUSY)
        je POLICE_TEMP3, 0, count_busy_slot;

        // Otherwise IdleTime += difference
        add. [POLICE_CTR2_LO], POLICE_TEMP0, [POLICE_CTR2_LO];
        addc [POLICE_CTR2_HI], POLICE_TEMP1, [POLICE_CTR2_HI];

        // We have processed the slot
        jext COND_TRUE, slot_counted;
count_busy_slot:
        // Add one to the busy counter
        add [POLICE_CTR1_LO], 1, [POLICE_CTR1_LO];
slot_counted:
        // It is safe to put Backup in LastState
        mov POLICE_TEMP3, EST_LAST_STATE;
final_step:
        mov SPR_TSF_WORD0, EST_START_TIME0
        mov SPR_TSF_WORD1, EST_START_TIME1

        // Jump back to the caller location (the label is defined just below the call)
        jext COND_TRUE, tracker_end;
```

In the above code, `POLICE_CTR1` is the first counter (counter index 0 in the driver code), which holds the number of busy slots. `POLICE_CTR2` (counter index 1 in the driver code) holds idle time in microseconds. Both these counters consist of two words (LO and HI words), because they hold 32-bit numbers while the firmware words are 16 bits long. The code ignores short spikes as they often correspond to antenna training and not real idle/busy switches.

B.4 Floating Point Helpers

As mentioned previously in this chapter, floating point operations are not allowed in driver code. For this reason, we implement a form of fixed-point arithmetic in the driver. We use the integer range [0, 65535] as a representation of the real range [0, 1] to have enough granularity in that range, and because a word in the chipset is 2 bytes, and it would be easier to store numbers this way. All operations are the defined in accordance with this representation.

What follows is the code used to handle floating points in throughout the implementation:

```
#define HFLOAT_DENOM 65536 /* Scale denominator used for calculations */
#define HFLOAT_ONE (HFLOAT_DENOM - 1) /* The value representing probability 1 */

unsigned long hflt_mul(unsigned long a, unsigned long b) {
	return (a * b) / HFLOAT_DENOM;
}
unsigned long hflt_div(unsigned long a, unsigned long b) {
	return (a * HFLOAT_DENOM) / b;
}
unsigned long hflt_pow(unsigned long a, unsigned int pow) {
	unsigned int i;
	long result = a;
	for (i = 0; i < pow - 1; ++i)
		result = hflt_mul(result, a);
	return result;
}
```

References

[1] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 3, pp. 535–547, 2000. 1, 38, 65, 70, 76, 77, 92

[2] Q. Ni, T. Li, T. Turletti, and Y. Xiao, "Saturation throughput analysis of error-prone 802.11 wireless networks: Research articles," *Wireless Communications and Mobile Computing*, vol. 5, pp. 945–956, Dec 2005. 1, 38

[3] K. D. Huang, K. R. Duffy, and D. Malone, "H-RCA: 802.11 collision-aware rate control," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, pp. 1–1, 2012. 1, 39, 41, 42

[4] M. Lacage, M. H. Manshaei, and T. Turletti, "IEEE 802.11 rate adaptation: a practical approach," in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, (New York, NY, USA), pp. 126–134, 2004. 1, 39, 41, 42

[5] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999. 1, 5, 57, 60, 139

[6] I. Dangerfield, D. Malone, and D. Leith, "Incentivising fairness and policing nodes in WiFi," *Communications Letters, IEEE*, vol. 15, no. 5, pp. 500–502, 2011. 2, 28, 31, 35, 36, 38, 39, 58, 59, 133

[7] P. Patras, H. Feghhi, D. Malone, and D. Leith, "Policing 802.11 MAC misbehaviours," *Mobile Computing, IEEE Transactions on*, vol. PP, no. 99, 2015. 3

[8] H. Feghhi, P. Patras, and D. Malone, "Practical node policing in 802.11 WLANs," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pp. 1–3, June 2013. 4

[9] H. Feghhi and D. Malone, "WiFo: A diagnostic tool for IEEE 802.11 MAC," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*, pp. 1–10, June 2015. 4

[10] *IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirement. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 2: Higher-Speed Physical Layer (PHY) Extension in the 2.4 GHz Band - Corrigendum 1*, 2001. 5, 139

[11] *IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2003. 5, 15, 139

[12] *IEEE Standard for Information technology–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*, 2005. 5, 13, 15, 16, 34, 139

[13] *IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, 2009. 5, 15, 16, 139

[14] H. Zimmermann, "OSI reference model–the ISO model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425–432, 1980. 6

[15] J. I. Choi, M. Jain, K. Srinivasan, P. Levis, and S. Katti, "Achieving single channel, full duplex wireless communication," in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, (New York, USA), pp. 1–12, ACM, 2010. 9

[16] D. Bharadia, E. McMilin, and S. Katti, "Full duplex radios," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pp. 375–386, 2013. 9, 38

[17] K. Xu, M. Gerla, and S. Bae, "How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 1, pp. 72–76 vol.1, 2002. 13

[18] J. Sobrinho, R. de Haan, and J. Brazio, "Why RTS-CTS is not your ideal wireless LAN multiple access protocol," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 1, pp. 81–87 Vol. 1, 2005. 13

[19] P. Chatzimisios, A. Boucouvalas, and V. Vitsas, "Effectiveness of RTS/CTS handshake in ieee 802.11a Wireless LANs," *Electronics Letters*, vol. 40, no. 14, pp. 915–916, 2004. 13

[20] *IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Redline*, March 2012. 13, 14

[21] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2, pp. 836–843 vol.2, 2003. 14, 26, 33, 36, 81, 126

[22] *IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2007. 18, 22, 23, 139

[23] A. Kamerman and L. Monteban, "Wavelan®-ii: a high-performance wireless LAN for the unlicensed band," *Bell Labs technical journal*, vol. 2, no. 3, pp. 118–133, 1997. 17

[24] J. He, W. Guan, L. Bai, and K. Chen, "Theoretic analysis of IEEE 802.11 rate adaptation algorithm samplerate," *Communications Letters, IEEE*, vol. 15, pp. 524–526, May 2011. 18

[25] "Minstrel rate adaptation algorithm documentation." http://linuxwireless.org/en/developers/Documentation/mac80211/RateControl/minstrel/. Accessed: 2015-10-23. 18, 81, 126

[26] L. B. Jiang and S.-C. Liew, "Improving throughput and fairness by reducing exposed and hidden nodes in 802.11 networks," *Mobile Computing, IEEE Transactions on*, vol. 7, no. 1, pp. 34–49, 2008. 19, 28

[27] D. Shukla, L. Chandran-Wadia, and S. Iyer, "Mitigating the exposed node problem in IEEE 802.11 ad hoc networks," in *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, pp. 157–162, 2003. 19, 136, 138

[28] T. Nandagopal, T.-E. Kim, X. Gao, and V. Bharghavan, "Achieving MAC layer fairness in wireless packet networks," in *Proceedings of the 6th ACM Annual International Conference on Mobile Computing and Networking*, MobiCom '00, (New York, USA), pp. 87–98, 2000. 25

[29] D. Pong and T. Moors, "Fairness and capacity trade-off in IEEE 802.11 WLANs," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pp. 310–317, Nov 2004. 25

[30] D. Qiao and K. Shin, "Achieving efficient channel utilization and weighted fairness for data communications in IEEE 802.11 WLAN under the DCF," in *Quality of Service, 2002. Tenth IEEE International Workshop on*, pp. 227–236, 2002. 26

[31] K. Jamshaid, P. Ward, M. Karsten, and B. Shihada, "The efficacy of centralized flow rate control in 802.11-based wireless mesh networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, 2013. 26

[32] M. Bredel and M. Fidler, "Understanding fairness and its impact on quality of service in IEEE 802.11," in *INFOCOM 2009, IEEE*, pp. 1098–1106, April 2009. 26

[33] J. Lee, H. Yoon, and I. Yeom, "Distributed fair scheduling for wireless mesh networks using IEEE 802.11," *Vehicular Technology, IEEE Transactions on*, vol. 59, pp. 4467–4475, Nov 2010. 26

[34] G. R. Cantieni, Q. Ni, C. Barakat, and T. Turletti, "Performance analysis under finite load and improvements for multirate 802.11," *Computer Communications*, vol. 28, no. 10, pp. 1095 – 1109, 2005. 26

[35] G. Tan and J. Guttag, "Time-based fairness improves performance in multi-rate WLANs," in *Proc. USENIX*, (Boston, Massachusetts, USA), June 2004. 26, 60

[36] T. Joshi, A. Mukherjee, Y. Yoo, and D. Agrawal, "Airtime fairness for IEEE 802.11 multirate networks," *Mobile Computing, IEEE Transactions on*, vol. 7, pp. 513–527, April 2008. 26

[37] I. Tinnirello and S. Choi, "Temporal fairness provisioning in multi-rate contention-based 802.11e WLANs," in *Proc. IEEE WoWMoM*, June 2005. 26, 60

[38] R. Jain, D.-M. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer system," *Digital Equipment, Tech. Rep. DEC-TR-301*, vol. 38, September 1984. 26

[39] J.-Y. Boudec, "Rate adaptation, congestion control and fairness: A tutorial," 2000. 27

[40] X. L. Huang and B. Bensaou, "On max-min fairness and scheduling in wireless ad-hoc networks: Analytical framework and implementation," in *Proceedings of the 2Nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*, MobiHoc '01, (New York, NY, USA), pp. 221–231, 2001. 27

[41] C.-C. Hsu, K.-J. Lin, Y.-R. Lai, and C.-F. Chou, "On exploiting spatial-temporal uncertainty in max-min fairness in underwater sensor net-

works," *Communications Letters, IEEE*, vol. 14, pp. 1098–1100, December 2010. 27

[42] F. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997. 27, 128

[43] M. Dianati, X. Shen, and S. Naik, "A new fairness index for radio resource allocation in wireless networks," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 2, pp. 712–717 Vol. 2, March 2005. 27

[44] M. Mehrjoo, M. Awad, M. Dianati, and X. Shen, "Maintaining utility fairness using weighting factors in wireless networks," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pp. 1–6, Nov 2009. 27

[45] L. B. Jiang and S. C. Liew, "Proportional fairness in wireless LANs and ad hoc networks," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 3, pp. 1551–1556 Vol. 3, March 2005. 27

[46] A. Checco and D. Leith, "Proportional fairness in 802.11 wireless LANs," *Communications Letters, IEEE*, vol. 15, no. 8, pp. 807–809, 2011. 27, 128

[47] K. Kar, S. Sarkar, and L. Tassiulas, "Achieving proportional fairness using local information in aloha networks," *Automatic Control, IEEE Transactions on*, vol. 49, pp. 1858–1863, Oct 2004. 27

[48] H. Ma, J. Hao, and R. Zimmermann, "Access point centric scheduling for dash streaming in multirate 802.11 wireless network," in *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, pp. 1–6, July 2014. 28

[49] C. E. Koksal, H. Kassab, and H. Balakrishnan, "An analysis of short-term fairness in wireless media access protocols (poster session)," *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, vol. 28, pp. 118–119, Jun 2000. 28

[50] G. Berger-Sabbatel, A. Duda, O. Gaudoin, M. Heusse, and F. Rousseau, "Fairness and its impact on delay in 802.11 networks," in *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 5, pp. 2967–2973 Vol.5, Nov 2004. 28

[51] A. Duda, "Understanding the performance of 802.11 networks," in *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on*, pp. 1–6, Sept 2008. 28

[52] C. Vlachou, J. Herzen, and P. Thiran, "Fairness of MAC protocols: IEEE 1901 vs. 802.11," in *Power Line Communications and Its Applications (ISPLC), 2013 17th IEEE International Symposium on*, pp. 58–63, March 2013. 28

[53] K. H. Almotairi, "Inverse binary exponential backoff: Enhancing short-term fairness for IEEE 802.11 networks," in *Wireless Communication Systems (ISWCS 2013), Proceedings of the Tenth International Symposium on*, pp. 1–5, Aug 2013. 28

[54] Y. Kim and G. Hwang, "Design and analysis of medium access protocol: Throughput and short-term fairness perspective," *Networking, IEEE/ACM Transactions on*, vol. 23, pp. 959–972, June 2015. 28

[55] A. Asadi and V. Mancuso, "A survey on opportunistic scheduling in wireless communications," *Communications Surveys Tutorials, IEEE*, vol. 15, pp. 1671–1688, Fourth 2013. 29

[56] R. Knopp and P. Humblet, "Information capacity and power control in single-cell multiuser communications," in *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on*, vol. 1, pp. 331–335 vol.1, Jun 1995. 29

[57] R. Reynolds and A. Rix, "Quality VoIP - an engineering challenge," *BT Technology Journal*, vol. 19, no. 2, pp. 23–32, 2001. 29

[58] Y. Lee, H. Lee, and C.-H. Choi, "Providing absolute priority and airtime fairness in WLANs," in *GLOBECOM Workshops (GC Wkshps), 2011 IEEE*, pp. 733–738, Dec 2011. 29

[59] A. Khalaj, N. Yazdani, and M. Rahgozar, "Service differentiation and fairness in IEEE 802.11 DCF," in *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication., 2005 13th IEEE International Conference on*, vol. 1, pp. 5 pp.–, Nov 2005. 29

[60] M. Bottigliengo, C. Casetti, C.-F. Chiasserini, and M. Meo, "Smart traffic scheduling in 802.11 WLANs with access point," in *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, vol. 4, pp. 2227–2231 Vol.4, Oct 2003. 29

[61] Q. Qiang, L. Jacob, R. Radhakrishna Pillai, and B. Prabhakaran, "MAC protocol enhancements for QoS guarantee and fairness over the IEEE 802.11 wireless LANs," in *Computer Communications and Networks, 2002. Proceedings. Eleventh International Conference on*, pp. 628–633, Oct 2002. 29

[62] V. Giri and N. Jaggi, "MAC layer misbehavior effectiveness and collective aggressive reaction approach," in *Sarnoff Symposium, 2010 IEEE*, pp. 1–5, April 2010. 29, 37

[63] L. Guang, C. Assi, and A. Benslimane, "Modeling and analysis of predictable random backoff in selfish environments," in *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '06, (New York, USA), pp. 86–90, 2006. 29

[64] M. Raya, I. Aad, J.-P. Hubaux, and A. El Fawal, "DOMINO: Detecting MAC layer greedy behavior in IEEE 802.11 hotspots," *Mobile Computing, IEEE Transactions on*, vol. 5, no. 12, pp. 1691–1705, 2006. 30, 31, 32, 35, 39, 58

[65] C. Liu, Y. Shu, W. Yang, and O. W. W. Yang, "Throughput modeling and analysis of IEEE 802.11 DCF with selfish node," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pp. 1–5, 2008. 30, 58

[66] "MadWifi project." http://www.madwifi-project.org/. Accessed: 2015-10-23. 30, 43, 58

[67] F. Gringoli and L. Nava, "Open firmware for WiFi networks." http://www.ing.unibs.it/~openfwwf/. Accessed: 2015-10-23. 30, 43, 48, 85, 94, 142, 172

[68] S. Giordano and S. Lucetti, "IEEE 802.11b performance evaluation: convergence of theoretical, simulation and experimental results," in *Telecommunications Network Strategy and Planning Symposium. NETWORKS 2004, 11th International*, pp. 405–410, June 2004. 30

[69] A. Di Stefano, A. Scaglione, G. Terrazzino, I. Tinnirello, V. Ammirata, L. Scalia, G. Bianchi, and C. Giaconia, "On the fidelity of IEEE 802.11 commercial cards," in *Wireless Internet, 2005. Proceedings. First International Conference on*, pp. 10–17, July 2005. 30

[70] A. Di Stefano, G. Terrazzino, L. Scalia, I. Tinnirello, G. Bianchi, and C. Giaconia, "An experimental testbed and methodology for characterizing IEEE 802.11 network cards," in *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a*, pp. 6 pp.–518, June 2006. 30

[71] A. Di Stefano, G. Terrazzino, L. Scalia, I. Tinnirello, G. Bianchi, and C. Giaconia, "On the anomalous behavior of IEEE 802.11 commercial cards," in *Proceedings of the 5th Annual Mediterranean Ad Hoc Networking Workshop*, June 2006. 30

[72] G. Bianchi, A. D. Stefano, C. Giaconia, L. Scalia, G. Terrazzino, and I. Tinnirello, "Experimental assessment of the backoff behavior of commercial IEEE 802.11b network cards.," in *INFOCOM, 2007 Proceedings IEEE*, pp. 1181–1189, 2007. 30, 41, 58, 61, 139, 142, 160

[73] T.-M. Hoang, T.-L. Tu, and T.-T. Nguyen, "An analytical model for estimating the impact of MAC layer misbehavior in IEEE 802.11 networks," in *Wireless Technology and Applications (ISWTA), 2014 IEEE Symposium on*, pp. 64–69, Sept 2014. 31

[74] A. Banchs, J. Ortin, A. Garcia-Saavedra, D. Leith, and P. Serrano, "Thwarting selfish behavior in 802.11 WLANs," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, pp. 1–1, 2014. 31, 38

[75] K. Pelechrinis, G. Yan, S. Eidenbenz, and S. Krishnamurthy, "Detection of selfish manipulation of carrier sensing in 802.11 networks," *Mobile Computing, IEEE Transactions on*, vol. 11, pp. 1086–1101, July 2012. 32

[76] K. Pelechrinis, C. Koufogiannakis, and S. Krishnamurthy, "On the efficacy of frequency hopping in coping with jamming attacks in 802.11 networks," *Wireless Communications, IEEE Transactions on*, vol. 9, pp. 3258–3271, October 2010. 32

[77] M. Vanhoef and F. Piessens, "Advanced wi-fi attacks using commodity hardware," in *Proceedings of the 30th Annual Computer Security Applications Conference*, ACM ACSAC '14, (New York, USA), pp. 256–265, 2014. 32

[78] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa, "On the performance of IEEE 802.11 under jamming," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pp. 13–18, April 2008. 32

[79] B. DeBruhl, C. Kroer, A. Datta, T. Sandholm, and P. Tague, "Power napping with loud neighbors: Optimal energy-constrained jamming and anti-jamming," in *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec'14, (New York, NY, USA), pp. 117–128, 2014. 32

[80] J. Konorski, "Protection of fairness for multimedia traffic streams in a non-cooperative wireless LAN setting," in *Protocols for Multimedia Systems* (M. van Sinderen and L. Nieuwenhuis, eds.), vol. 2213 of *Lecture Notes in Computer Science*, pp. 116–129, Springer Berlin Heidelberg, 2001. 32

[81] Z. Zhang, J. Wu, J. Deng, and M. Qiu, "Jamming ACK attack to wireless networks and a mitigation approach," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pp. 1–5, Nov 2008. 32

[82] A. Rachedi and A. Benslimane, "Smart attacks based on control packets vulnerabilities with IEEE 802.11 MAC," in *Wireless Communications*

and Mobile Computing Conference, 2008. IWCMC '08. International, pp. 588–593, Aug 2008. 33

[83] S. Kandula, D. Katabi, M. Jacob, and A. W. Berger, "Botz-4-Sale: Surviving organized DDoS attacks that mimic flash crowds," in *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005. 33

[84] S. M. Specht, "Distributed denial of service: taxonomies of attacks, tools and countermeasures," in *Proceedings of the International Workshop on Security in Parallel and Distributed Systems, 2004*, pp. 543–550, 2004. 33

[85] M. Agarwal, D. Pasumarthi, S. Biswas, and S. Nandi, "Machine learning approach for detection of flooding dos attacks in 802.11 networks and attacker localization," *International Journal of Machine Learning and Cybernetics*, pp. 1–17, 2014. 33

[86] N. Hachem, Y. Ben Mustapha, G. Granadillo, and H. Debar, "Botnets: Lifecycle and taxonomy," in *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pp. 1–8, May 2011. 33

[87] R. Lua and K. C. Yow, "Mitigating DDoS attacks with transparent and intelligent fast-flux swarm network," *Network, IEEE*, vol. 25, pp. 28–33, July 2011. 33

[88] L. Kavisankar and C. Chellappan, "A mitigation model for TCP SYN flooding with ip spoofing," in *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pp. 251–256, June 2011. 33

[89] B. Swain and B. Sahoo, "Mitigating DDoS attack and saving computational time using a probabilistic approach and HCF method," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pp. 1170–1172, March 2009. 33

[90] R. Rejimol Robinson and C. Thomas, "Evaluation of mitigation methods for distributed denial of service attacks," in *Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on*, pp. 713–718, July 2012. 33

[91] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM '03, (Berkeley, California, USA), pp. 2–2, 2003. 33

[92] J. Choi, J. Chiang, D. Kim, and Y.-C. Hu, "Partial deafness: A novel denial-of-service attack in 802.11 networks," in *Security and Privacy in Communication Networks* (S. Jajodia and J. Zhou, eds.), vol. 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 235–252, Springer Berlin Heidelberg, 2010. 33

[93] A. A. Cárdenas, S. Radosavac, and J. S. Baras, "Evaluation of detection algorithms for MAC layer misbehavior: Theory and experiments," 2008. 34, 39, 41

[94] J. Choi, A. Min, and K. Shin, "A lightweight passive online detection method for pinpointing misbehavior in WLANs," *Mobile Computing, IEEE Transactions on*, vol. 10, pp. 1681–1693, Dec 2011. 34

[95] J. Tang, Y. Cheng, and W. Zhuang, "An analytical approach to real-time misbehavior detection in IEEE 802.11 based wireless networks," in *INFOCOM, 2011 Proceedings IEEE*, pp. 1638–1646, 2011. 34, 58, 138

[96] S. Szott, M. Natkaniec, and R. Canonico, "Detecting backoff misbehaviour in IEEE 802.11 EDCA," *European Transactions on Telecommunications*, vol. 22, no. 1, pp. 31–34, 2011. 34, 58

[97] P. Kyasanur and N. H. Vaidya, "Selfish MAC layer misbehavior in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 4, pp. 502–516, 2005. 34, 37, 58

[98] P. Serrano, A. Banchs, V. Targon, and J. Kukielka, "Detecting selfish configurations in 802.11 WLANs," *Communications Letters, IEEE*, vol. 14, no. 2, pp. 142–144, 2010. 35, 58

[99] S. Radosavac, J. S. Baras, and I. Koutsopoulos, "A framework for MAC protocol misbehavior detection in wireless networks," in *Proceedings of the 4th ACM Workshop on Wireless Security*, WiSe '05, (New York, NY, USA), pp. 33–42, 2005. 35

[100] A. Wald, "Sequential tests of statistical hypotheses," *Ann. Math. Statist.*, vol. 16, pp. 117–186, June 1945. 35

[101] A. Toledo and X. Wang, "Robust detection of selfish misbehavior in wireless networks," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 6, pp. 1124–1134, 2007. 35, 58

[102] Y. Rong, S.-K. Lee, and H.-A. Choi, "Detecting stations cheating on backoff rules in 802.11 networks using sequential analysis," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–13, April 2006. 35

[103] A. A. Cárdenas, S. Radosavac, and J. S. Baras, "Detection and prevention of MAC layer misbehavior in ad hoc networks," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, SASN '04, (New York, NY, USA), pp. 17–22, 2004. 36, 37, 39, 58

[104] S. Djahel, Z. Zhang, F. Nait-Abdesselam, and J. Murphy, "Fast and efficient countermeasure for MAC layer misbehavior in MANETs," *Wireless Communications Letters, IEEE*, vol. 1, pp. 540–543, October 2012. 36, 37

[105] M. Portoles, Z. Zhong, and S. Choi, "IEEE 802.11 downlink traffic shaping scheme for multi-user service enhancement," in *Personal, Indoor and Mobile Radio Communications, 2003. PIMRC 2003. 14th IEEE Proceedings on*, vol. 2, pp. 1712–1716 vol.2, Sept 2003. 36

[106] M. Marcon, M. Dischinger, K. Gummadi, and A. Vahdat, "The local and global effects of traffic shaping in the internet," in *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pp. 1–10, Jan 2011. 36

[107] K. Cho, K. Fukuda, H. Esaki, and A. Kato, "The impact and implications of the growth in residential user-to-user traffic," in *Proceedings of the 2006 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, (New York, NY, USA), pp. 207–218, 2006. 36

[108] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is p2p dying or just hiding? [p2p traffic measurement]," in *Global*

Telecommunications Conference, 2004. GLOBECOM '04. IEEE, vol. 3, pp. 1532–1538 Vol.3, Nov 2004. 36

[109] L. Vollero and G. Iannello, "Frame dropping: A QoS mechanism for multimedia communications in WiFi hot spots," in *Parallel Processing Workshops, 2004. ICPP 2004 Workshops. Proceedings. 2004 International Conference on*, pp. 54–59, 2004. 36, 85, 164

[110] L. Vollero, A. Banchs, and G. Iannello, "ACKS: a technique to reduce the impact of legacy stations in 802.11e EDCA WLANs," *Communications Letters, IEEE*, vol. 9, no. 4, pp. 346–348, 2005. 36, 85, 164

[111] A. Banchs, P. Serrano, and L. Vollero, "Providing service guarantees in 802.11e EDCA WLANs with legacy stations," *Mobile Computing, IEEE Transactions on*, vol. 9, no. 8, pp. 1057–1071, 2010. 36, 85, 136

[112] F. Shi, J. Baek, J. Song, and W. Liu, "A novel scheme to prevent MAC layer misbehavior in IEEE 802.11 ad hoc networks," *Telecommunication Systems*, vol. 52, no. 4, pp. 2397–2406, 2013. 37

[113] D.-J. Deng, B. Li, L. Huang, C.-H. Ke, and Y.-M. Huang, "Saturation throughput analysis of multi-rate IEEE 802.11 wireless networks," *Wireless Communications and Mobile Computing*, vol. 9, no. 8, pp. 1102–1112, 2009. 38

[114] G. Martorell, G. Femenias, and F. Riera-Palou, "A refined 3D Markov model for non-saturated IEEE 802.11 DCF netwoks," in *Wireless Days (WD), 2013 IFIP*, pp. 1–8, Nov 2013. 38

[115] P. Serrano, P. Salvador, V. Mancuso, and Y. Grunenberger, "Experimenting with commodity 802.11 hardware: Overview and future directions," *Communications Surveys Tutorials, IEEE*, vol. 17, pp. 671–699, Second 2015. 39

[116] B. Raman and K. Chebrolu, "Design and evaluation of a new MAC protocol for long-distance 802.11 mesh networks," in *Proceedings of the 11th ACM Annual International Conference on Mobile Computing and Networking*, MobiCom '05, (New York, NY, USA), pp. 156–169, 2005. 39, 42

[117] E. Pelletta and H. Velayos, "Performance measurements of the saturation throughput in IEEE 802.11 access points," in *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, 2005. WIOPT 2005. Third International Symposium on*, pp. 129–138, April 2005. 39

[118] H. Kim, C. Cordeiro, K. Challapali, and K. G. Shin, "An experimental approach to spectrum sensing in cognitive radio networks with off-the-shelf IEEE 802.11 devices," in *IEEE 802.11 Devices, in IEEE Workshop on Cognitive Radio Networks, in conjunction with IEEE CCNC*, 2007. 39, 42, 142

[119] S. Rayanchu, A. Patro, and S. Banerjee, "Airshark: detecting non-WiFi RF devices using commodity WiFi hardware," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pp. 137–154, 2011. 39, 142

[120] J. Yeo, M. Youssef, and A. Agrawala, "A framework for wireless LAN monitoring and its applications," in *Proceedings of the 3rd ACM Workshop on Wireless Security*, WiSe '04, (New York, NY, USA), pp. 70–79, 2004. 39

[121] J.-Y. Yoo, T. Heuhn, and J. Kim, "Active capture of wireless traces: Overcome the lack in protocol analysis," in *Proceedings of the Third ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, WiNTECH '08, (New York, NY, USA), pp. 41–48, 2008. 39

[122] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi, "An experimental study on the capture effect in 802.11a networks," in *Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, WinTECH '07, (New York, NY, USA), pp. 19–26, 2007. 41

[123] G. Xylomenos and G. Polyzos, "TCP and UDP performance over a wireless LAN," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, vol. 2, pp. 439–446 vol.2, 1999. 41, 42

[124] "FLAVIA project." http://www.ict-flavia.eu/. Accessed: 2015-10-23. 42, 43, 46

[125] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity hardware.," in *INFOCOM, 2012 Proceedings IEEE* (A. G. Greenberg and K. Sohraby, eds.), pp. 1269–1277, 2012. 42, 46, 85, 135

[126] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello, "MAClets: active MAC protocols over hard-coded devices," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, ACM CoNEXT '12, (New York, NY, USA), pp. 229–240, 2012. 42, 46, 58, 135, 137

[127] A. Loch, R. Klose, M. Hollick, A. Kuehne, and A. Klein, "Practical OFDMA in wireless networks with multiple transmitter-receiver pairs," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pp. 1–3, June 2013. 42

[128] "Broadcom BCM43xx specification." http://bcm-v4.sipsolutions.net/Specification. Accessed: 2015-10-23. 45, 48, 143

[129] "Existing linux wireless drivers." http://wireless.kernel.org/en/users/Drivers. Accessed: 2015-10-23. 53, 58, 90

[130] A. Veres, A. Campbell, M. Barry, and L.-H. Sun, "Supporting service differentiation in wireless packet networks using distributed control," *Selected Areas in Communications, IEEE Journal on*, vol. 19, pp. 2081–2093, Oct 2001. 59, 70

[131] D. Thuente, B. Newlin, and M. Acharya, "Jamming vulnerabilities of IEEE 802.11e," in *Military Communications Conference, 2007*, IEEE MILCOM '07, pp. 1–7, 2007. 60

[132] Edney and W. A. Arbaugh, *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. 60

[133] *IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 11: Wireless LAN Medium Access*

Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Medium Access Control (MAC) Security Enhancements, 2004. 61, 139

[134] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell, "Detecting 802.11 MAC layer spoofing using received signal strength," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pp. 2441–2449, 2008. 61

[135] C. Neumann, O. Heen, and S. Onno, "An empirical study of passive 802.11 device fingerprinting," in *Proc. Distributed Computing Systems Workshops (ICDCSW)*, pp. 593–602, June 2012. 61, 80

[136] J. Xiong and K. Jamieson, "SecureArray: Improving Wifi security with fine-grained physical-layer information," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, pp. 441–452, 2013. 61

[137] K. Duffy, D. Malone, and D. Leith, "Modeling the 802.11 distributed coordination function in non-saturated conditions," *Communications Letters, IEEE*, vol. 9, no. 8, pp. 715–717, 2005. 62

[138] H. Wu, Y. Peng, K. Long, S. Cheng, and J. Ma, "Performance of reliable transport protocol over IEEE 802.11 wireless LAN: analysis and enhancement," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 599–607 vol.2, 2002. 72, 136

[139] D. S. Shafer and Z. Zhang, *Introductory Statistics*. Flat World Knowledge, 2012. 74

[140] C. McDonald, *Converged Networking: Data and Real-time Communications over IP*. IFIP Advances in Information and Communication Technology, Springer US, 2013. 79

[141] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller, "Maranello: Practical partial packet recovery for 802.11," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI '10, (Berkeley, California, USA), pp. 14–14, 2010. 85

[142] P. Barford and M. Crovella, "Generating representative Web workloads for network and server performance evaluation," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '98/PERFORMANCE '98, (New York, USA), pp. 151–160, 1998. 119

[143] P. Patras, H. Qi, and D. Malone, "Exploiting the capture effect to improve WLAN throughput," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pp. 1–9, 2012. 131

[144] P. Patras, H. Qi, and D. Malone, "Mitigating collisions through power-hopping to improve 802.11 performance," *Pervasive and Mobile Computing*, 2013. 131, 138

[145] *IEEE Standard for Broadband over Power Line Networks: Medium Access Control and Physical Layer Specifications*, Dec 2010. 136

[146] C. Cano and D. Malone, "Evaluation of the backoff procedure of Homeplug MAC vs. DCF," in *Personal Indoor and Mobile Radio Communications (PIMRC), IEEE 24th International Symposium on*, pp. 1846–1850, September 2013. 136

[147] J. Lee and J. C. Walrand, "Design and analysis of an asynchronous zero collision MAC protocol," *CoRR*, vol. abs/0806.3542, 2008. 136

[148] J. Barceló, B. Bellalta, C. Cano, and M. Oliver, "Learning-BEB: Avoiding collisions in WLAN," p. 23, 2009. 136, 138

[149] M. Fang, D. Malone, K. R. Duffy, and D. J. Leith, "Decentralised learning MACs for collision-free access in WLANs," *CoRR*, vol. abs/1009.4386, 2010. 136, 138

[150] L. Sanabria-Russo, J. Barceló, and B. Bellalta, "Prototyping distributed collision-free MAC protocols for WLANs in real hardware," in *MACOM* (M. Jonsson, A. V. Vinel, B. Bellalta, N. Marina, D. Dimitrova, and D. Fiems, eds.), vol. 8310 of *Lecture Notes in Computer Science*, pp. 82–87, Springer, 2013. 138

[151] Y. Erten, "A layered security architecture for corporate 802.11 wireless networks," in *Wireless Telecommunications Symposium, 2004*, pp. 123–128, May 2004. 138

[152] Z. Dong and T. Lai, "A scalable and adaptive clock synchronization protocol for IEEE 802.11-based multihop ad hoc networks," in *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pp. 8 pp.–558, Nov 2005. 138

[153] Soekris Engineering. http://www.soekris.com/. Accessed: 2015-10-23. 142

[154] A. Kamerman and N. Erkocevic, "Microwave oven interference on wireless LANs operating in the 2.4 GHz ISM band," in *Personal, Indoor and Mobile Radio Communications, 1997. Waves of the Year 2000. PIMRC '97., The 8th IEEE International Symposium on*, vol. 3, pp. 1221–1227 vol.3, Sep 1997. 157

[155] A. Kumar, E. Altman, D. Miorandi, and M. Goyal, "New insights from a fixed-point analysis of single cell IEEE 802.11 WLANs," *Networking, IEEE/ACM Transactions on*, vol. 15, pp. 588–601, June 2007. 162