inf | Informatik
Computer Science

Jamming Detection and Performance of Rate Switching under Jamming

Dietmar Schediwie and Nicolas Häfelin

Masterthesis

October 2009 - April 2010

Department of Computer Science

Advisors: Dr. V. Lenders and M. Strasser
Professors: Prof. Dr. S. Capkun and Prof. Dr. B. Plattner

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

Denial of service attacks are an important topic in the whole area of networking. In this work we develop and evaluate methods to detect and reduce the effects of a particular kind of attack on wireless networks called jamming. Based on commercial hardware and drivers, we concentrate on building an easily deployable system for jamming detection, using the RSSI, physical rate and PDR. We also subject the current rate switching algorithm Minstrel, to an in-depth study, aiming at improving the achieved throughput during an attack.
The results of our indoor tests show a high probability to detect a constant jamming adversary in almost all settings. This also includes mobile scenarios and such with legitimate background traffic on the same channel, while producing no overhead for the detection itself. In the cases, where the implementation has only a reduced success, suggestions are made, how these situations could be included in a future version.
Our improvements on Minstrel however, have only a small influence on the transmission rate, demonstrating that its adaptivity is already very high. Still, some enhancements are made and others theoretically discussed.

Im Bereich von Computernetzwerken sind Angriffe auf die Verfügbarkeit ein wichtiges Thema. In dieser Arbeit entwickeln und bewerten wir Methoden um eine spezielle Art von Angriff auf drahtlose Netzwerke (WLAN) zu Erkennen und ihre Auswirkungen zu reduzieren. Wir fokussieren unsere Anstrengungen darauf mit handelsüblichen Geräten ein leicht einsetzbares System zur Erkennung zu entwickeln bei dem wir die RSSI, physikalische Rate und PDR als Kriterien verwenden. Desweiteren unterziehen wir den aktuellen Algorithmus zur Auswahl der physikalischen Rate - Minstrel - einer detaillierten Studie, die das Ziel verfolgt, den Datendurchsatz während eines Angriffs zu verbessern.
Die Ergebnisse unserer Tests, welche wir im Gebäudeinneren durchgeführten - zeigen eine hohe Wahrscheinlichkeit, einen konstanten Angreifer in fast allen Situationen zu erkennen. Dies beinhaltet mobile Szenarien und solche mit sich korrekt verhaltendem Verkehr anderer Stationen auf dem gleichen Kanal, ohne dass dies zu zusätzlichem Datentransfer für die Erkennung führt. In den Fällen, in denen unsere Implementation weniger erfolgreich ist, machen wir Vorschläge, wie diese Situationen in einer zukünftigen Version abgedeckt werden können.
Unsere Verbesserungen für Minstrel haben jedoch nur einen geringen Einfluss auf die Datenrate, was ein Beweis für die bereits vorhandene Anpassungsfähigkeit des Algorithmus ist. Trotz allem führen einige Ideen zu einer Steigerung des Durchsatzes, während weitere auf theoretischer Basis diskutiert werden.

Contents

List of Figures

List of Tables

1 Introduction

In this chapter, we talk about our motivations for the thesis and present detailed assumptions and prerequisites regarding the considered system and attacker model. All notations defined here remain constant throughout this document.

1.1 Motivation

For radio transmissions, denial of service is a prominent issue, because a lot of important applications rely on wireless communication. The most common ones are sensor networks, mobility and broadcast in general.
One type of denial of service is called jamming, in which the attacker interferes with the physical transmission of the participants. The counter strategies can be divided into two major parts: Jamming avoidance and jamming detection.
Jamming avoidance is strongly influenced by the military sector and their long history with radio transmission. This led mainly to large spreading codes and different kinds of frequency hopping[1]. Yet due to hardware restrictions and public standards, none of these are applicable for jamming avoidance in common IEEE 802.11g networks.
Jamming detection on the other hand, relies mainly on observed characteristics and relates them to each other to make a decision which then may or may not lead to countermeasures by the users. Schafroth presented in his work [1] an idea that considered a broad set of jammers and had a strong focus on mobility. The

[1]The focus lies in hiding the signal in a large frequency range. Spreading usually reduces the signal strength for each frequency to a level that is lower than the noise. This leads to a situation where an attacker is unable to determine the presence of a signal. Frequency hopping uses a secret-depending algorithm that selects a very small part of the whole frequency range on which the participants communicate with each other. The key for success is, that the frequency changes faster than the amount of time, an attacker needs, to determine the one currently used.

problem is that his implementation is impractical for real time detection and needs additional tools to produce the measurements.
In addition to the jamming detection, he also showed, that the rate selection algorithms he used failed to select a good rate under jamming and therefore reduced the achievable throughput significantly.
Our goal is to adapt his approach for real-time use without the need for uncommon equipment and to develop a rate selection algorithm that takes advantage of the jamming detection results to provide a better performance during an attack.

1.2 Related Work

We divide our examination of related work into two parts. We start with jamming detection, followed by rate switching.

1.2.1 Jamming Detection

Several studies were made and published regarding jamming detection for radio transmissions in general and WLAN in particular.
Özcerit and Çakıroğlu present in [2] an anomaly based approach for wireless sensor networks using the packet delivery ratio[2] (PDR), the bad packet ratio[3] and the energy consumption over time. They use an initialization phase to determine the conditions without the influence of jamming and compare the later measurements to the initial distribution. The works of Schafroth [1] and of Xu, Trappe, Zhang and Wood [3] use the signal strength, PDR, received signal strength indication (RSSI) and carrier sense timing[4]. While [3] uses PDR/'signal strength' and distance/PDR relations as consistency checks, [1] reduced the set of jammers while adding movement and a threshold for the strength of the noise on the channel.
In case of a reactive jammer, Hamieh and Ben-Othman published an approach in [4], that correlated bit errors of a message to their relative position in the message, saying that the jammer reveals itself in the time he needs to react to a message, thus influencing the distribution of errors in a message to detect the presence of an

[2]Defined as: 'delivered packets'/'sent packets'
[3]Defined as: 'erroneous packets'/'received packets'
[4]The average time, that a sender needs to wait for a channel to be idle.

attacker, Strasser brought the relationship between the signal strength and error probability up in [5].

1.2.2 Rate Switching

There are only few papers that suggest anti-jamming measurements that can be easily applied to current systems and don't affect the MAC or PHY layer.
In [6], the authors tested the rate adaptation algorithms AMRR, Onoe and Sample and showed that their performance could quite effectively by decrease with a random jammer[5]. They propose an Anti-jamming Reinforcement System (ARES) that basically just switches between rate adaptation and using a fixed rate. This can help against such a jammer, as using a fixed rate outperforms the three tested algorithms. The main reason for this is, that those three algorithms do not recover fast enough while the jammer sleeps. However, fixed rates can only be used when the environment does not change, which excludes mobility.
In [7], the ACK-Guided Immediate Link rate Estimation algorithm (AGILE) is proposed. It uses an offline profile of Signal-to-Noise Ratio (SNR)[6] associated with the physical rate that performs best under these conditions. Any received packet (including ACKs) will provide a SNR and with a look up in the profile, the next rate is determined. The paper however does not account for jamming and we assume that the profile created under normal conditions would not be optimal under jamming. Moreover for different jammers also different profiles would be needed in order to be able to detect not only jamming but the jammer type as well.
In [8], the author compares different bitrate selection algorithms in an indoor environment without any jammer interference. He then presents the rate control algorithm SampleRate that selects the best rate by maximizing the theoretical throughput given the measured packet loss. The algorithm applies a sampling technique to update the Packet Delivery Ratio (PDR) on the different physical rates. This algorithm is the base for Minstrel[7], the rate control algorithm used in the Linux kernel 2.6.31.4. We discuss Minstrel in chapter 4, because it is the

[5]Is described in section 2.3
[6]Defined as $SNR = P_{signal}/P_{noise}$
[7]http://wireless.kernel.org/en/developers/Documentation/mac80211/RateControl/minstrel

algorithm used for most of our tests.

1.3 Contribution

We have two goals in our thesis: The first one is the development and assessment of a jamming detection algorithm that can be easily deployed on current systems. The second is to determine, how different rate switching algorithms perform under jamming and how we can improve Minstrel with the intend to make it more jamming resistant. For this purpose, we select an IEEE 802.11g ad-hoc network and consider constant jammers that reduce, but not completely disrupt, the throughput.
Making use of the WLAN metrics reported by the operation system, we build an algorithm that uses a no-jammer model as reference to evaluate current measurements. We compare a set of different variations of our algorithm to study the influence of internal factors on the detection success and demonstrate through the results of these experiments the possibilities and problems with the reduced informations for jamming detection.
We test the rate switching algorithms AMRR, Onoe, SampleRate and Minstrel under different jammers and show how Minstrel performs under bit, noise and frame jamming. We elaborate several ideas to improve the performance of Minstrel under these three jammers and test the following ideas:

- Using a smaller packet size.
- Changing interval duration and $\text{EWMA}_{\text{Level}}$ of Minstrel.
- Sampling less often.
- Dynamically adapt the sampling limits for each physical rate.

1.4 Outline

In this section, we present the structure of our report. In chapter 2, we will first give an introduction to the underlying theory, upon which our work is based, followed by a detailed description of the settings and attackers we considered and tested.

In chapter 3, we will discusses the problem of jamming detection. It starts with some specific theory and then presents and evaluates two approaches to recognize a jammed situation. The end are a comparison with [1] and multiple ideas for possible follow-ups on the detection. In chapter 4, we will talk about our findings, regarding the physical rate switching on the basis of the Minstrel algorithm. We will start again with a small theory part, followed by some hypotheses to improve the physical rate selection and the results of our tests. Chapter 5 recapitulates our achievements and findings before we present some ideas for a combination of jamming detection and rate switching in chapter 6. Along with some additional informations, the appendix finally contains a copy of our assignment and time schedules.

2 System and Attacker Model

We first give a general introduction to WLAN, explaining some of the underlying principles. Following that, we specify the settings and jammers we considered.

2.1 Technical Introduction

The purpose of this section is to give a quick introduction to the necessary knowledge to help the reader understand the rest of the document. Detailed information that is only relevant for either jamming detection or rate switching will be presented in their own chapter. If the reader is already knowledgeable about these topics, he may skip this section. We will limit ourself to a broad level of understanding and assume, that the reader is familiar with the OSI model and has a basic knowledge in data transmission and networking.

IEEE standard 802.11, WLAN, is one of the popular methods to transmit data between different devices without the need of a physical connection. Other possibilities are Bluetooth and infrared (IR), each having its own advantages and disadvantages. The standard defines 13 overlapping **channels**, each with a bandwidth of 20 MHz (+1 MHz on each side as a buffer), covering the frequencies from 2.412 GHz up to 2.472 GHz. Japan added a 14th channel at 2.484 GHz, while a lot of American countries only allow the use of channels 1 to 11. As shown in figure 2.1, only three non-overlapping (four, counting the 14th) sections exist.

WLAN supports two operation modes: **Infrastructure** and **ad-hoc**. In infrastructure mode, a dedicated station, called access point (AP), acts as an interface for the whole network. Every client registers with the AP to form a star shaped

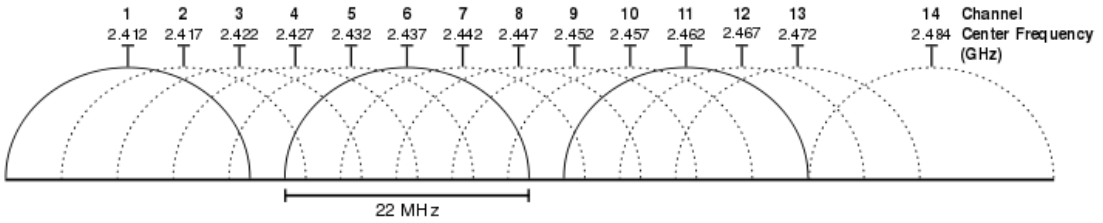

Figure 2.1: Graphical representation the WLAN channel distribution. Source: Wikipedia.

topology. The access point functions as a bridge, controlling the data flow in and out of the network as well as the communication among clients. Ad-hoc mode on the other hand forms a bi-directional graph with usually more edges than the infrastructure mode. Ideally everyone is in transmission range of everyone else and the topology becomes a complete graph, where everyone can directly send messages to every other participant and where no hidden node problem exists. A hidden node situation occurs, if there are three stations A, B and C of which A and B see each other as well as B and C. A and C don't see each other and thus may interfere with each other at B.

On the physical layer, a **carrier sense multiple access with collision avoidance** (CSMA/CA) system is used to reduce collisions of data frames. If a station wants to send, it first listens to the channel for a predefined time. If no activity is found, the channel is assumed to be idle and the station is allowed to transmit. Otherwise, if the channel is busy, the station waits a certain back-off time and tries again. The sender first reserves the channel for a short time, directly followed by the rest of the MAC header[1]. It then transmits its payload and error correction code and waits for an acknowledgement (ACK) from the target station. If no ACK is received, the station tries to send the message again, starting the process all over. A listening station checks each message for the intended receiver. If it is the target, it will examine the correctness of the frame, using the redundancy checks included in the message header. If the information was received correctly, the station will immediately send an acknowledgement to the sender.

[1]MAC frames have the following structure: < Frame control and duration (4 Byte) | Address and sequence control (26 Byte) | Payload (0-2312 Byte) | Redundancy check (4 Byte) >.

According to IEEE 802.11g, which is the standard used in our experiments, a frame can be modulated in two different ways with multiple subtypes, resulting in 12 possibilities to encode a frame at the physical layer[2]. Each of these has a different redundancy, enabling distinct **physical transmission rates**. The two major modes are called direct-sequence spread spectrum (**DSSS**) and orthogonal frequency-division multiplexing (**OFDM**).
DSSS uses a pseudo random sequence to distribute the energy over the whole frequency band of a channel. The resulting signal seems like noise but can be de-spreaded back into the original form, using the same random sequence. Other noise and signals are most likely to be cancelled out during the de-spreading. The drawback is, that for it to work, the stations have to be synchronized during the send and receive phases. An advantage is the possibility of code division, multiple access (CDMA), which enables multiple transmissions over the same frequency at the same time, if they use different spreading codes. DSSS has four different redundancy ratios, resulting in the physical transmission rates of 1 Mb/s, 2 Mb/s, 5.5 Mb/s and 11 Mb/s.
OFDM uses the frequency range differently. While DSSS transmits one bit per symbol, using the whole band, OFDM takes longer for each symbol but sends multiple bits in it. OFDM splits the signal band into 52 orthogonal carrier frequencies, out of which four are used as pilot frequencies to help with the signal synchronization and to monitor the channel condition. The remaining carriers make it possible to transmit 48 bits per symbol, but since each one takes longer and requires an additional intermission phase, the possible throughput is not 48 times as high as with DSSS. Different carrier modulation schemes and error correction codes result in the physical rates of 6 Mb/s, 9 Mb/s, 12 Mb/s, 18 Mb/s, 24 Mb/s, 36 Mb/s, 48 Mb/s and 54 Mb/s.

[2]Other 802.11 protocols use different physical data rates.

2.2 System Model

For our work, we consider a scenario where two parties want to communicate with each other via WLAN. They may or may not have a good connection and they may or may not move around. If they move, they do so with walking speed and without leaving the transmission range. Because the participants want to achieve the best possible throughput, they are interested in detecting external sources disturbing their communication so that they can take appropriate actions against them. It is important to note, that disturbing means lowering the throughput significantly without disrupting it[3].
We assume, that every transmission takes place using UDP over the network layer, which uses the IP protocol.
All testing done by us happened inside a building with each participant being represented by a Dell Latitude E6400 ATG laptop, equipped with a Netgear WG511T wireless card, using an Atheros chip. We selected a basic Ubuntu 9.04 installation as operating system with an ath5k v0.6.0 driver, using Minstrel as the default rate switching algorithm.

Figure 2.2: Netgear WG511T wireless card.

The notation of the settings in the thesis obeys the following arrangements:

- Setting 1, as shown in figure 2.3, represents a good connection[4] in which the jammer only influences one of the nodes A and B[5].

[3] $(0 \text{ Mb/s} < throughput_{disturbed} < 0.7 * throughput_{undisturbed})$
[4] We define a good connection as one, where Minstrel chooses the rate 36 Mb/s or higher for at least 90% of the packets, while in the absence of a jammer.
[5] In the case of noise jamming, we define *influence* as a nonnegligible change in the RSSI of a node. The signal strength of the bit jammers is set to a 5 dBm lower output strength while the frame jammer reduced it by 25 dBm.

- Setting 2, as shown in figure 2.4, represents a good connection in which the jammer influences both nodes A and B.

- Setting 3, as shown in figure 2.5, represents a relatively bad connection[6] in which the jammer only influences one of the nodes.

- Setting 4, as shown in figure 2.6, represents a good connection in which the jammer influences both nodes A and B but one more than the other.

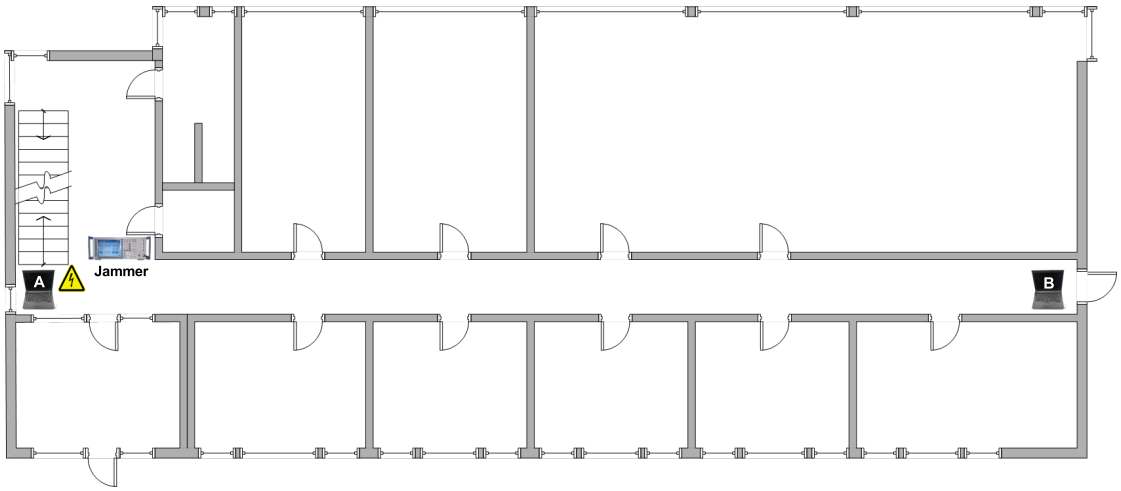

Figure 2.3: Setting 1. The two nodes are about 35 meters apart on the same floor and are in line of sight. The jammer is positioned around the corner.

We expect, that the results achieved with our installations can be generalised to the broader setting description, because comparable tests in other locations as well as experiments with movement gave us similar results.

2.3 Attacker model

This section describes possible attackers and which of these we used for our work. Then we will explain how and with what hardware we implemented these different jammers.

[6] We define a bad connection as one, where Minstrel chooses the rate 12 Mb/s or lower for at least 90% of the packets, while in the absence of a jammer.

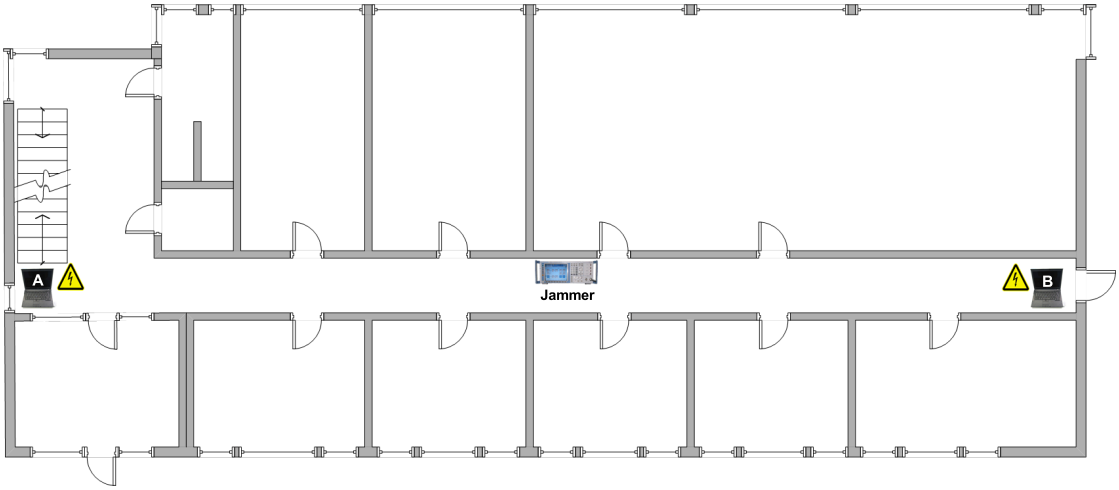

Figure 2.4: Setting 2. The two nodes are about 35 meters apart on the same floor and are in line of sight. The jammer is positioned right between them.

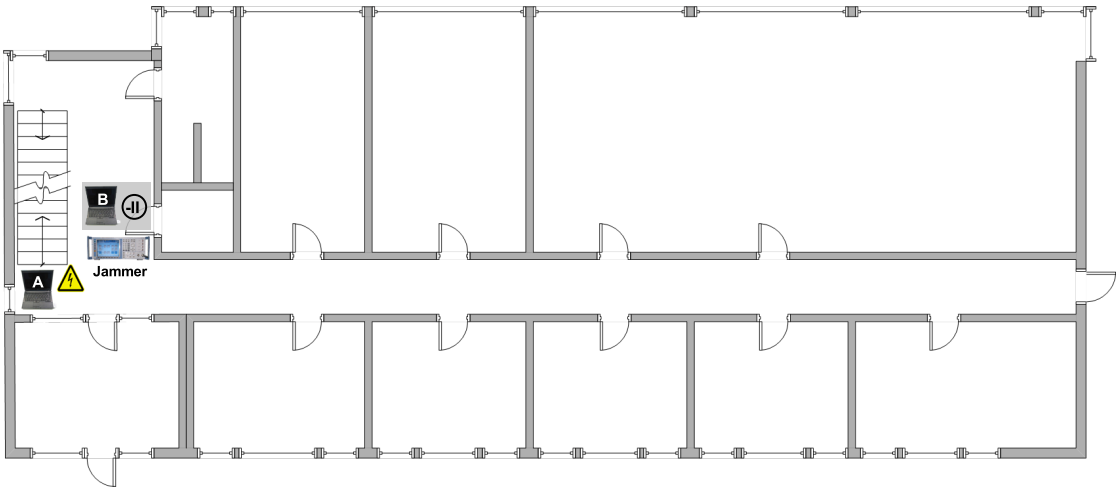

Figure 2.5: Setting 3. Node B is placed two floors below A with a small offset. The jammer is positioned as in setting 1.

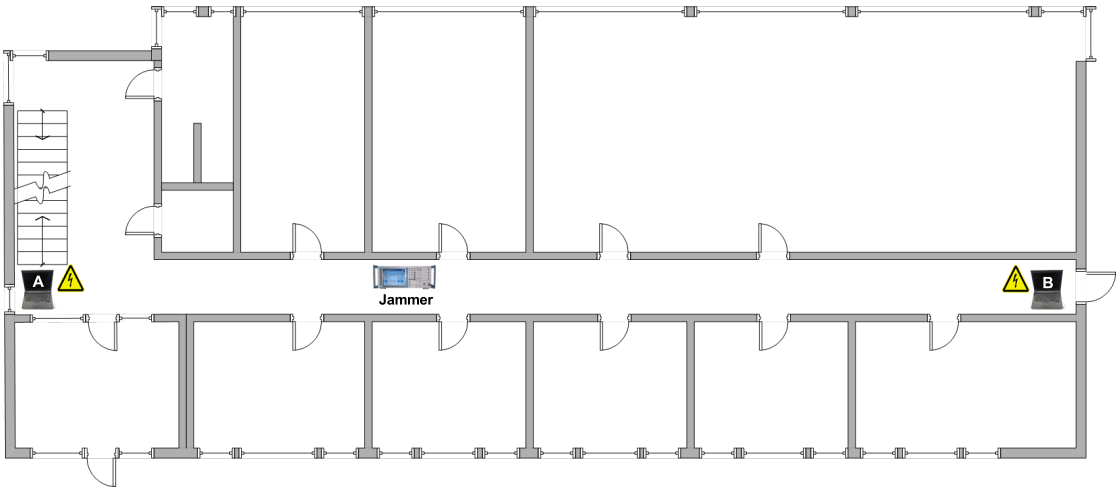

Figure 2.6: Setting 4. The two nodes are about 35 meters apart on the same floor and are in line of sight. The jammer is positioned at about a third of the way between them.

2.3.1 Jamming Strategy

A jammer targets the communication between two or more parties and tries to partially or completely interrupt it. To achieve this there are several strategies and techniques to use. Some are simple and others more complex requiring advanced and expensive hardware to be implemented. We now present our classification of the different jamming strategies shown in figure 2.7.

First we divide the strategies by whether the jammer is reactive or not. Being reactive means, that the jammer listens to the communication and then decides what to do. The most basic form is to just listen if there is any communication and on what frequency it takes place. Then the attacker starts jamming this frequency for a short period of time and starts the next scan. It is called a follow jammer.

A reactive jammer is called smart, if it uses sophisticated methods to jam the target. First we have those which analyse the received signal and try to send a specially modulated signal to make decoding the resulting signal as hard as possible for the intended receiver. But as this requires expensive hardware and is still only possible in some scenarios, it is not used very often. An other smart strategy is to decode the signal and then jam specific parts of transmitted packets. This could for example be crucial parts of the packet header. Such a jammer we call a packet jammer. The last smart jammer we considered jams single packets to attack the

communication protocol itself. The attacked protocol has a weakness which can be exploited to reduce the throughput[7]. The advantage of smart jammers is, that they are harder to detect and consume less energy achieving the same throughput degradation.
The non-reactive jammers are rather easy and cost-efficient to implement. We divide this category into the constant, random and pulse jammers. Constant jammers are the easiest to implement as they send at the same power all the time. A slight variation would be the so called chirp jammer that does not use a constant transmission power but linearly increases it until a certain level is reached and then starts at the initial value again. The idea is to save energy by not transmitting at high power all the time. The next category are random jammers, which are constant jammers with a random active and sleep period. The idea again is to save power, but still degrade the connection as much as possible. This can be very effective against some rate switching algorithms and communication protocols as they do not recover fast enough during the jammer's sleep period. The last group of attackers, the so called pulse jammers only jam a short period of time at very high power and sleep for most of the time. The sleep period can either be fixed or random.

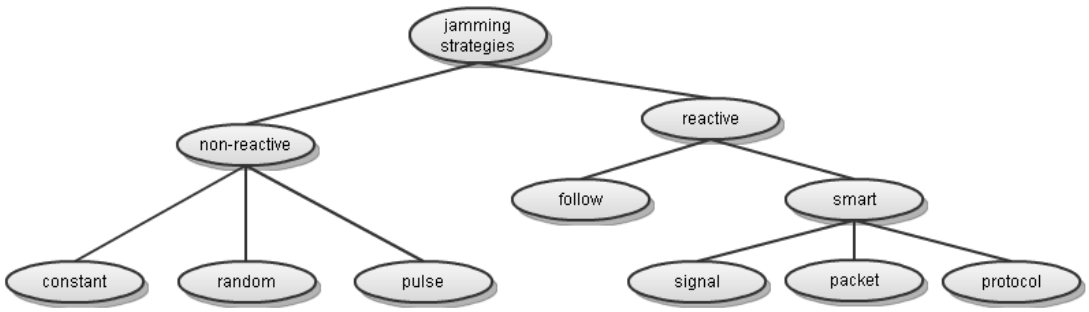

Figure 2.7: Our classification of the different jamming strategies. Source: author's own

[7]E.g. jamming the ACKs on the MAC layer or disrupting TCP traffic to abuse its congestion control.

2.3.2 Jamming Technique

Each of the jamming strategies requires a jamming technique in order to be implemented. They are shown in figure 2.8. These techniques are categorized in two main groups. Noise jammers simply generate a random signal over a given frequency range and we distinguish broadband noise and tone jammers. Tone jammers only send on a single frequency, the smallest frequency range there is, any other frequency range we classify as broadband noise. In [9] the author differentiates between partial- and broadband, where partial-band jammers only jam a subset of the available channels and broadband jam all. We decided to denote both as broadband.
Jammers that use a modulated signal are split into two groups, bit and frame jammers. A bit jammer generates a random bit sequence that is correctly modulated. In the case of WLAN, this means that one of the physical rates is used and the signal is modulated accordingly. Frame jammers don't just send a random sequence of bits but send complete and correct frames. If the jamming signal is strong enough, these frames will be decoded at a receiver and in the case of WLAN hinder it from sending packets during this transmission. As the required signal strength for decoding a frame is quite low, frame jammers are very energy efficient.

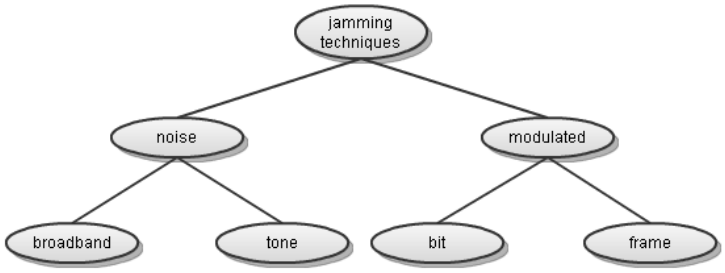

Figure 2.8: The categories of jamming techniques we distinguish. Source: author's own

2.3.3 Selected Jammers

We chose to use only non-reactive jammers, as we would have had to implement a reactive jammer ourself, which is outside the scope of this work.

We used a constant broadband noise, bit and frame jammer as done in the work of Schafroth [1]. This allows a good comparison between the offline version elaborated in his work and our online version of the jamming detection algorithm.
To implement these three jammers we used a R&S SMU200A Vector Signal Generator (cf. [10]) as depicted in figure 2.9. We used this device to generate white noise with a range of 20 MHz around the centre frequency of the used WLAN channel, which we adjusted for each experiment. For the bit and frame jammer we used the capabilities of the signal generator to modulate IEEE 802.11g signals and set the rate to 11 Mb/s using Complementary Code Keying (CCK) as physical layer mode. For the frame jammer, we additionally set a pause of 0.1 ms after each data frame[8].

Figure 2.9: The Rhode & Schwarz SMU 200A Vector Signal Generator.

[8] A data frame has 1024 bytes of random data.

3 Jamming Detection

In this part of the report, we first provide a short section, introducing parts of WLAN theory that are only relevant for the jamming detection. After that, we present our first approach that focuses on the channel capacity and data transmission rate to detect jamming. We discuss the reasons, why this approach isn't good enough to reach our goals and why we changed our focus to an approach using the PDR, RSSI and physical rate as in Schafroth's work [1]. After introducing the concept, we present the implementation and an in-depth evaluation, followed by a review of our findings.

3.1 Technical Introduction for Jamming

We present two topics in this section: The first is about the signal strength and RSSI, showing their relation and dependencies with the noise strength. The second deals with the capacity of a wireless channel which is important to understand the first approach.

3.1.1 Signal Strength and RSSI

The signal strength and received signal strength indication (RSSI) are two important concepts which may sound similar but have large differences. The signal strength refers to the magnitude of the electric field, while the RSSI is a vendor specific value that is loosely defined as a measurement of power present in a received radio signal.
Atheros and other manufacturers publish a table or formula to calculate RSSI, signal and noise strength based on each other. However, our measurements didn't comply with these calculations.
In figure 3.1, we present a measurement of the RSSI in the presence of a noise

jammer whose signal strength is gradually increased. We placed two nodes next to each other in close proximity to the jamming device. The graph shows how the RSSI goes down as the noise becomes stronger. Another important point is that with increased noise strength, the frequency of RSSI jumps grows. These jumps in the RSSI were always around 10 dBm large and even though they occurred in both directions, heightening and reducing the RSSI, they were more likely to lower the value. The third observation was, that the moment the jammer was powered down, the RSSI jumped almost immediately to a very high value. In the measurement shown in figure 3.1 the RSSI started around -25 dBm and went down to -50 dBm. As the jammer was switched off, the RSSI jumped to around -5 dBm and took a few seconds before settling back to the starting value[1].

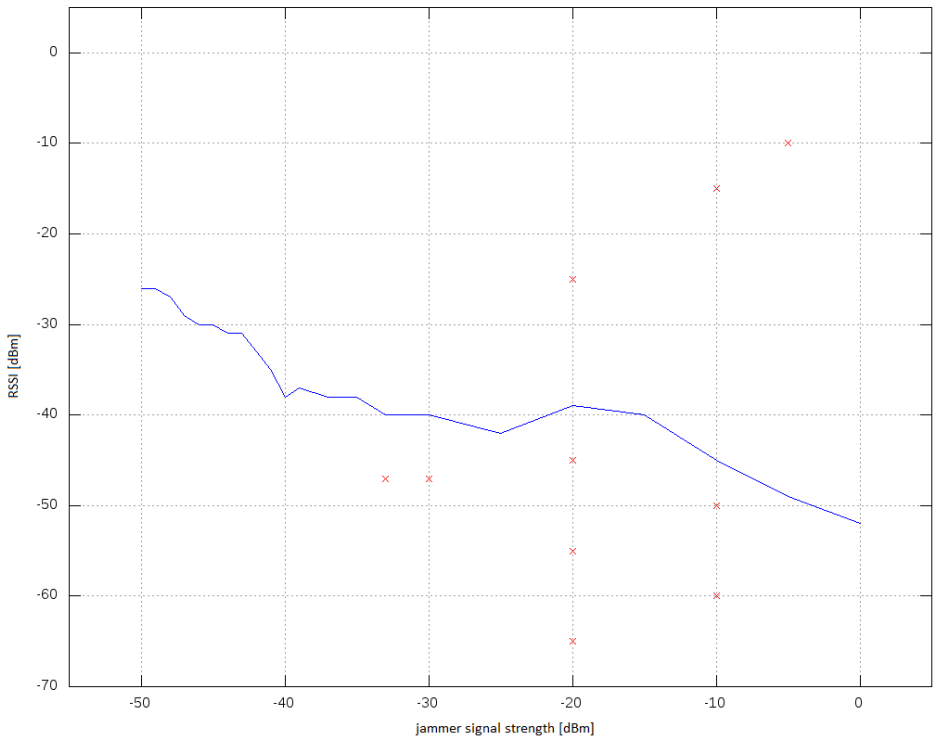

Figure 3.1: The blue line shows the change in the RSSI of a node while being close to a gradually increased noise jammer. The x indicate some of the observed jumps.

[1]See table A.1, to see how this can influence the measurements during a jamming period.

Following these measurements we concluded, that Atheros bases the RSSI not only on the signal and noise strength, but that the noise floor[2] is adaptive too. This means that the determination of the noise floor by the card can be influenced by an attacker. Let's take figure 3.1 as an example: In an undisturbed case, the noise floor lies at -98 dBm, the noise level may be around -94 dBm and the RSSI at -35 dBm. Now the noise jammer increases its output until it reaches -40 dBm. Until here, the RSSI acts like the signal to noise ratio (SNR), as it decreases to a value around -35 dBm. Now the jammer increases its output another 10 dBm, but instead of the RSSI following this development, it stays around the old value but starts to show jumps of 10 dBm. We think, that the wireless card adjusted the noise floor to a new value, 10 dBm higher than the old one (that means around -88 dBm). This shifts the value of the noise strength, which is determined by comparing it to the noise floor. The same goes for the signal strength, resulting in the RSSI being seemingly unchanged. But the calibration of the noise floor is not stable, changing its value and thus triggering the observed RSSI jumps.

3.1.2 Channel Capacity

In an AWGN channel, the capacity is defined by the Shannon-Hartley theorem as:

$$C = B * log_2(1 + S/N)$$

C = channel capacity [bit/second]; B = bandwidth of the channel [hertz]; S = total received signal power [watt] or [(volt)^2]; N = total received noise power [watt] or [(volt)^2].
This equation enables the calculation of the maximum transmission rate for any environment. It is however not possible to achieve it using the 802.11 protocol, due to overheads in the protocol[3], back off time and multi-path effects.

3.2 Throughput Based Algorithm

Our first approach focuses directly on the data transmission rate and the effective use of the wireless medium to detect a jamming adversary.

[2]Usually defined as the imprecision through the noise of the card itself.
[3]This includes header data, redundancy and signal spreading.

3.2.1 Concept

We designed this algorithm so that it doesn't need a predefined threshold and considers the data transmission rate directly, instead of looking at secondary characteristics like the PDR.
The goal of all our jammers was to reduce the throughput of the communicating nodes. This means, that every attacker either reduces the channel capacity (noise jammer through an increased noise power) or achieves his goal by targeting the communication directly (the frame jammer through the busy channel back-off and the bit jammer through the bit-error rate.).
If one has exact values for the signal and noise strength, the maximum data transmission rate can by calculated using the Shannon-Hartley theorem. Using the same data we assume it to be possible to calculate the optimal encoding variant and therefore the perfect physical rate. The overhead could be calculated based on the 802.11 characteristics and then compared to the measured throughput.
As the nature of wireless communication is broadcast, every participant can know how much of the capacity is actually used, which makes the conclusion fairly simple:

> If the channel capacity worsens unexpectedly[4], or if the observed data transmission rate is significantly smaller than the theoretically possible while at the same time there is a need to make use of the rest, then there is a jammer present.

The last point requires either a detailed knowledge of the other nodes, or has to be restricted to the times when the station itself wants to send something. It is also necessary to take further steps in classifying jamming for this approach. For example the presence of other wireless nodes on the same channel may be considered jamming in an environment where no such things are expected (a sensor network on a volcano, measuring the movement of the ground to predict an

[4] In the case of a sensor network it may very well be, that the weather changes, increasing the noise strength and therefore reducing the capacity. Because of this, a change over time threshold would need to be defined.

eruption) while perfectly normal in an urban environment. Following this thought process the setting has to be divided into the following cases:

1. No wireless communication other than from predefined participants is expected.

2. Correct behaviour (= following all specifications, especially the back off time and frame length) is tolerated, even if it comes from a station that does not belong to our network.

The first case can be ensured by proper sender and message authentication on a software level, by hardware characteristics on the physical layer [11] or a few known positions [12].
If the behaviour of stations is monitored, it is also possible to detect a station that disregards an occupied channel (by cutting in while another station is talking), endless sending (disregarding a maximum or allocated time slot), etc. which solves the second case.
The last remaining complication is a hidden node situation, where the network topology leads to a situation where two stations, that don't see each other, produce interferences at a third station. This problem would have to be handled on a higher level between all involved participants or by accepting the overhead of RTS/CTS[5] through additional 'request to send' and 'clear to send' messages.
This combination of existing technology and concepts, combined with an approach that targets the throughput directly made this idea promising.

3.2.2 Evaluation

As a preliminary test, we arranged two nodes as in setting 1. To account for the knowledge of the transmission rate, a fixed traffic rate was used and instead of calculating the channel capacity and overhead, we set it to the maximal measured throughput without interferences.
All this made the detection itself a triviality, but had important implications for the further development, especially for the characteristic based algorithm.
The first problem we encountered was the inability to get the exact values for the

[5] http://en.wikipedia.org/wiki/IEEE_802.11_RTS/CTS

noise and signal strength, making a real implementation in our system an impossibility. The second problem that revealed itself were the effects of mobility. As soon as we moved one or both stations in walking-speed[6], the throughput was reduced. The point is, that the signal and noise strength aren't influenced by the movements and therefore the theoretical channel capacity remains unchanged.
We also tested the idea to simply filter the time of movement and only decide while standing still. This would either require a movement detection like [12] or external modules like GPS. Another possibility we thought of was that the standard deviation of the RSSI, throughput, PDR or physical rate would be different while moving. The experiments however showed, that this idea doesn't work.
This would lead to a lot of false positives and we would have needed to extend the system model to account for all the extra hard- and software. The algorithm would also create a transmission overhead through the exchange of channel situations at different positions. We therefore judged the approach to be insufficient and changed our focus to the characteristic based algorithm.

3.3 Characteristic Based Algorithm

Our second approach focuses on transmission characteristics like the RSSI, physical rate and PDR to detect a jamming adversary.

3.3.1 Concept

The basis for this approach lies in the observation made in [1]:

> Depending on the manufacturer and model of a wireless card, there exists a direct relationship for each physical rate, between the PDR and signal strength pairs in the absence of a jammer. This domain is disjoint from the PDR and signal strength pairs while suffering disruptions by the environment.

[6]The effect on the distance was negligible and the line of sight was also not broken.

[1] further uses the noise level as a different input which is compared to an environment dependent threshold. The two evaluations are then combined to result in a $\{jamming, no\text{-}jamming\}$ decision.
To collect the informations, he used an offline approach, relying on wireless monitoring tools. The signal strength value could be calculated from the reported RSSI and the noise value, while the PDR came from comparing data frames with acknowledgements from the receiver.
We decided to follow his conclusion and tried to adapt the algorithm, so that it can be used in real-time without any special soft- or hardware requirements. Because of that, we had to modify the approach to be compatible with the input available through the operation system and debug mode of the rate switching algorithm. The limitations of our online mode can be summarised in the following way:

- The noise strength is not available in ad-hoc mode.
- The RSSI reported is unstable and highly dependent on the noise level.
- Because of the retry chain Minstrel uses and a sometimes small number of packets per physical rate, the PDR estimations are vague.

On the other hand, the shift in the RSSI enabled us to relate the three values and produce charts like figure 3.2 that show a clear difference between jamming and no-jamming measurements.

The core observation that is shown in figure 3.2 is, that the physical rate declines faster during jamming as during the absence of an attacker. The reason is simply that the bit and frame jammer reduce the PDR (and therefore the selected physical rate) without lowering the RSSI enough to make it seem like an undisturbed connection with a low signal strength. The noise jammer is similar, because the physical rate and RSSI change, due to an increased noise, is different from the one, originating from a reduced signal strength.
Based on these measurements, we implemented our idea with different variations, using the reduced data as input. We will show in the evaluation that it is - even with less informations - possible to get good results in deciding between jamming and no-jamming.

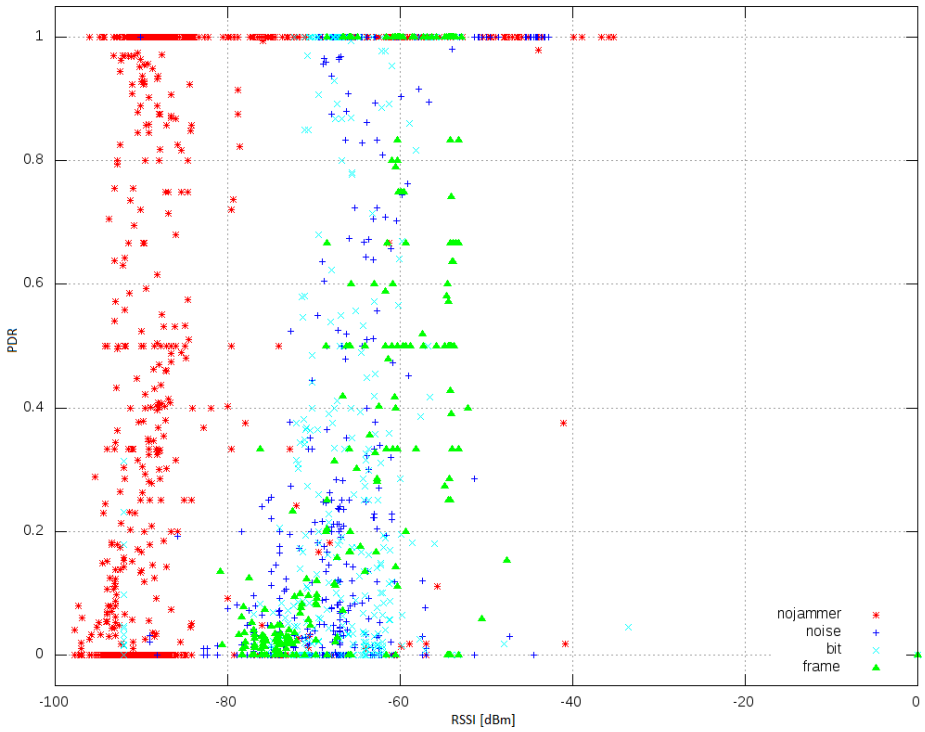

Figure 3.2: Comparison of no-jammer points * to the jammer points +, x and ▲ on a PDR to RSSI metric for the physical rates 1 Mb/s up to 11 Mb/s.

3.3.2 Implementation

We built the algorithm in a way that it takes an input, consisting of a batch comprising a MAC-address, a signal value and an array of integer pairs $([sig, (s_1, t_1), (s_2, t_2), ..., (s_{54}, t_{54})]_{mac})^*$ and returns a decision $result_{mac}$ consisting of a decision $\{jamming, no\text{-}jamming, no\text{-}decision\}$ and a certainty $c = [0, 1]$. The MAC-address serves as identifier for each connection, while the signal sig is the arithmetic average over repeatedly measured RSSI values. The certainty acts as an additional information that shows roughly what percentage of the packets during the last batch are enforcing the decision. Zero stands for a total uncertainty that happens if the input provided no usable data, while a one means, that the evaluation of all physical rates resulted in the returned decision. No-decision and a confidence of zero are equivalent, enforcing the confidence as a way to show the advantage of this decision over a random choice between jamming and no-jamming. Each integer-pair (s_i, t_i) for $i = \{1|2|5.5|6|9|11|12|18|24|36|48|54\}$ represents the delta of transmitted packets on each of the physical rates separated into a success s_i and try t_i value with $s_i <= t_i$.
The detection function itself has a threshold and a short-term memory. The threshold T_i for $i = \{1|2|5.5|6|9|11|12|18|24|36|48|54\}$ marks the area of PDR and RSSI values which are observed in the absence of an adversary. See figure 3.3 for an example, where the red line indicates the border of the threshold-domain[7].

This threshold based approach makes sense, since the triple of physical rate, RSSI and PDR lies very often outside these borders if a jammer is present, as we show in figure 3.4. One can see, that the PDR has a very high dispersion, almost producing a rectangular shape and thus strongly limiting the impact of this value for the decision. Nonetheless, there are also physical rates in which the no-jamming points form a different structure[8], showing that the PDR is still important.

To determine the presence of a jammer, a series of measurements is compared against the threshold. We do this by comparing each pair of $[sig, (s_i, t_i)]$ to the

[7]The reason for combining the rates of 2, 5.5 and 11 Mb/s lies in an almost perfect overlap. We assume, that one of the reasons for this lies in the fact that they are all DSSS encoded.
[8]See figures A.2 and A.3.

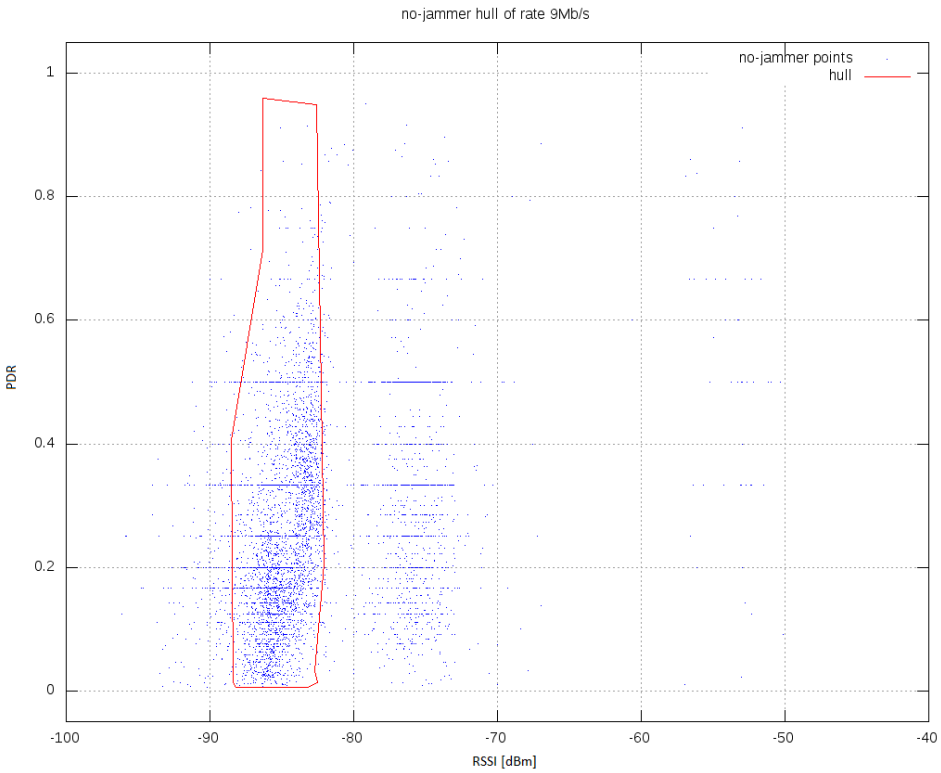

Figure 3.3: No-jammer hull of rate 9 Mb/s. It may be difficult to see, but the number of points outside of the red line is less then 5% of the total.

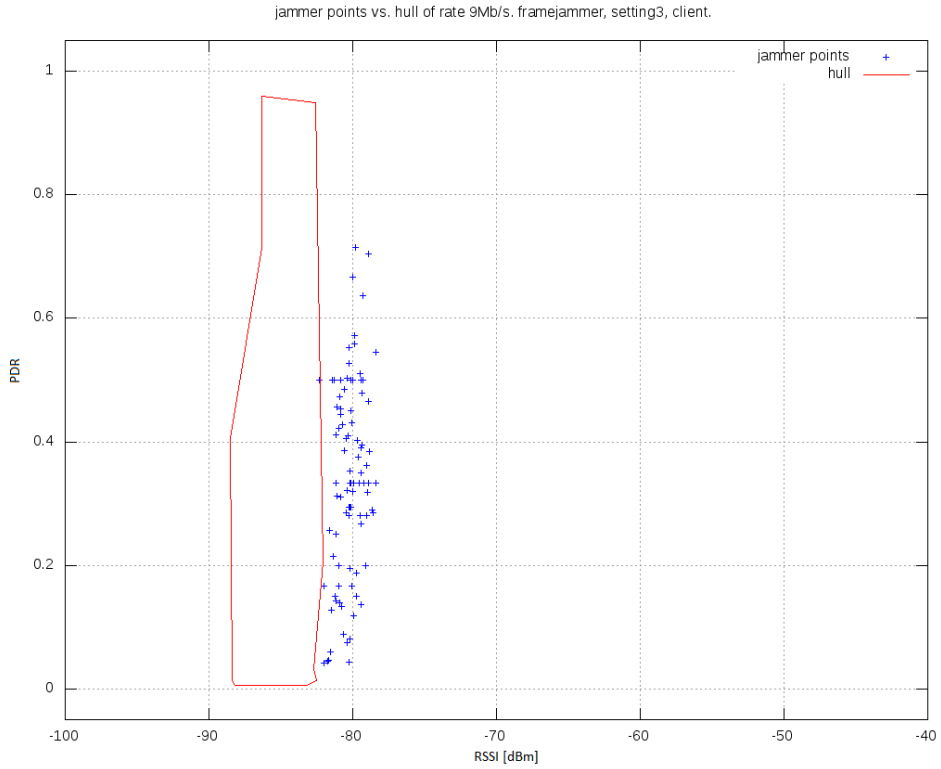

Figure 3.4: Jammer points vs. hull of rate 9 Mb/s.

threshold T_i. We then decide on no-jammer if the set is inside the threshold. If it is outside, it is filtered for a PDR equal to 1 or 0. Figure 3.2 showed already the impossibility to clearly differentiate between jamming and no-jamming at these edges. If the pair has a PDR of]0,1[and is outside the threshold, it is considered to be jammed.
Each decision d_i for $i = \{1|2|5.5|6|9|11|12|18|24|36|48|54\}$ with $d_i = \{jamming, no\text{-}jamming, no\text{-}decision\}$ is weighted by their factor $f_i = t_i/i$ to account for the relative time spent on the different rates. This way, we form the batch-decision $[d]_{mac}$ out of the different d_i as a pair of the decision $\{jamming, no\text{-}jamming, no\text{-}decision\}$ and a value denoting the certainty $c = [0, 1]$ of this decision. This certainty is determined by the relation of the sum of f_i that support the judgement to the total of all f_i.

```
Pseudo code for the process from input to current decision:
for(i={1|2|5.5|6|9|11|12|18|24|36|48|54})
  d := decide(sig,succ,try)
  if (d = jamming)
     jam_c += try/i
  elseif (d = no-jamming)
     noj_c += try/i
  else
     nodec_c += try/i
  endif
endfor
if (jam_c = noj_c = 0)
  return {no-decision, 0}
elseif (jam_c >= noj_c)
  return {jamming, jam_c/(jam_c + noj_c + nodec_c)}
else
  return {no-jamming, noj_c/(jam_c + noj_c + nodec_c)}
endif
```

The new decision d_{mac} is inserted into a buffer of fixed size, which exists separately for each connection. The purpose of this is to function as a filter against outliers and can have a different weight assigned for each of its n entries with the

conditions that each weight is positive and the sum equals one.

```
Pseudo code for the filter of a connection:
buffer.insert(d)
for (each item in the buffer)
   if (buffer.item.decision = jamming)
     jam_c += buffer.item.certainty * weight[n]
   elseif (buffer.item.decision = no-jamming)
     noj_c += buffer.item.certainty * weight[n]
   endif
   n += 1
endfor
if (jam_c = noj_c = 0)
   return {no-decision, 0}
elseif (jam_c >= noj_c)
   return {jamming, jam_c}
else
   return {no-jamming, noj_c}
endif
```

Following the descriptions above, the flow diagram in figure 3.5 gives another way to illustrate the data flow.

After presenting how the algorithm itself is implemented, we now show, where and how we collect the information for the input.
The RSSI value for each connection is built by repeated queries to the iw environment via the

```
$ iw dev wlan0 station dump
```

command. We also tried

```
$ iwconfig wlan0
```

and

```
$ ifconfig wlan0
```

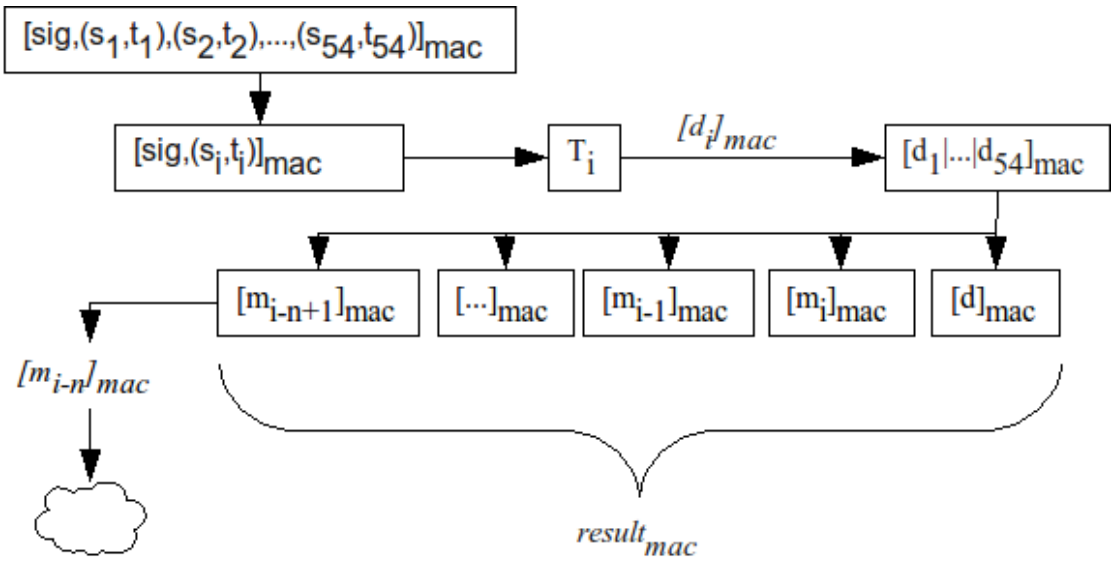

Figure 3.5: Sketch of the characteristic based algorithm for an evaluation circle of a connection.

but because they can't provide informations for each individual connection and also don't report the strength of the noise in ad-hoc mode, they gave us no additional input.

We collected the PDR defining pairs (s_i, t_i) by reading the debug output from the Minstrel algorithm. Since it isn't enabled per default, we have to activate the debug mode through

```
$ sudo mount -t debugfs debugfs /sys/kernel/debug
```

After this, Minstrel will generate and update the rc_stats file in the folder[9]:

```
/sys/kernel/debug/ieee80211/phy*/stations/*/
```

The text file[10] contains a lot of information about each connection, but the only points of interest to us are the delta of the success and attempts column between two points in time, showing the new sent packages for each rate and their success: $(s_i, t_i) = (success_{i\_new} - success_{i\_old}, attempts_{i\_new} - attempts_{i\_old})$.

[9] The first * represents an integer. We observed a range from 0 to 2. The second * represents a MAC-address in the format of .. : .. : .. : .. : .. : .. with each . standing for a hexadecimal value from 0 to f.

[10] This example is taken from http://linuxwireless.org/en/developers/Documentation/mac80211/RateControl/minstrel

```
 rate   throughput ewma prob this prob  this succ/attempt success  attempts
  P1      0.9        99.9       100.0          0(  0)         105       111
   2      0.4        25.0       100.0          0(  0)           1         1
   5.5    1.2        25.0       100.0          0(  0)           1         1
  11      1.1        12.5        50.0          0(  0)           1         2
   6      0.0         0.0         0.0          0(  0)           0         0
   9      0.0         0.0         0.0          0(  0)           0         0
  12      0.0         0.0         0.0          0(  0)           0         0
  18      0.0         0.0         0.0          0(  0)           0         0
  24      0.0         0.0         0.0          0(  0)           0         0
  36      0.0         0.0         0.0          0(  0)           0         0
 t48     16.0        40.9        88.8          0(  0)           9        10
T 54     16.2        91.1        91.2        115(126)       96429    109032

Total packet count::    ideal 5756      lookaround 641
```

3.3.3 Evaluation

In this section, we discuss the results for the implementation of the characteristic based algorithm. As an introduction, we describe the different variants of the algorithm, along with the parameters which were constant for all of them.
Following this, we will briefly repeat the different settings and our assumptions before presenting the results.

3.3.3.1 Evaluation Variants

All tests were done by us with two nodes, called *client* and *server*. If not specified otherwise, these two used iperf to send a UDP-stream to each other and the only differences between the client and server are the amount of data they try to send and their proximity to the jammer. While the client wants to transmit as much as he can, the server tries to achieve only 1 Mb/s.
Following the implementation described above, the following parameters were adjusted to influence the outcome.

Threshold

The threshold plays the most significant role, since ti defines the area in which points are recognized as not-jammed.

We recorded no-jammer data that covers the whole RSSI range as well as all physical rates, but since the 48 Mb/s and 54 Mb/s rates don't work[11], only measurements up to the 36 Mb/s rate are meaningful. This lead us to an additional problem: Not only the no-jammer area of the 1 Mb/s rate, but especially the one of 36 Mb/s grows large. While the other physical rates have a relatively narrow RSSI range in which they are commonly selected, the 1 Mb/s rate covers the area from the minimal value (little over -98 dBm) up to the lower limit of the 2 Mb/s rate. This is even worse for the 36 Mb/s rate because due to the rates 48 Mb/s and 54 Mb/s being unusable, the 36 Mb/s rate has to cover their conditions as well. This is further worsened by the fact, that the borders between two physical rates aren't fix and there is a smooth transition between them, increasing the no-jammer areas even further.

The problem with the 36 Mb/s rate is therefore, that it is simply too big, which means, that not only no-jammer points are inside the area, but also a lot during jamming. From this observation, we decided on a variant that ignores the 36 Mb/s rate the same way as the 48 Mb/s and 54 Mb/s[12].

The *-base* variation uses the 36 Mb/s rate, while *-threshold* considers only the rates up to 24 Mb/s.

Weighting

During the discussion about the use of the 36 Mb/s rate we also thought of another possibility to counter the problem. In some prior tests, it was very noticeable that the false negative rate was much higher than the false positive one[13].

Therefore we assigned different weights to the intermediate results d_i, according to their decisions.

The *-base* case is defined as using the same weights for jamming and no-jamming decisions, while the variant *-weight* favours jamming decisions with a factor of

[11] This is an effect only observed in ad-hoc mode. We don't know why that is but the biggest part of the recordings there are probes from Minstrel to update the internal channel statistics.

[12] That means they always return no-decision.

[13] That means, that points outside the threshold were much more likely to be jammed, than an inside point was to be not-jammed.

four.[14]

Filtering

Another way to handle outliers is the discussed buffer for each connection resulting in a variable buffer size and their respective weights for the final decision. We tried different values but concluded, that the order of the weights had only a negligible influence. This made our first idea, to let the newest data have the biggest influence while being kept in check by the older results, futile. Following this finding, we assigned the same weight for every position in the buffer.
The *-base* case is defined as having a buffer with size one, while the variant *-filter* shows the results for a buffer of size four with 0.25 as the weight for each item.

Global Settings

The last variable that we could have changed is the time interval between two evaluations. If it is too short, we assume that the fluctuations in the PDR become too big, but if it is too long, the information may come too late for the intended purpose. We set the interval time on one second which is pretty long for computers, while still acceptable to humans and changes in the environment. It also improves the stability of the RSSI and gives Minstrel enough time to create more accurate PDR pairs that are neither 1 nor 0.

Summary

For reference, we name the algorithm *ocba* for online characteristic based algorithm. With the intend to study the influence of the variations, we split ocba into four variants as shown in table 3.1.

The different settings are discussed more closely in section 2.2, but we will repeat the notions and global settings of the algorithm in the following list that is valid if not specified differently:
UDP throughput the client wants to achieve: **100 Mb/s** (meaning 'as much as possible').

[14] We decided on the number four because it should demonstrate a trend without overpowering everything else.

name	buffer size	physical rates	weight *jamming*/*no-jamming*
ocba-base	1	1 Mb/s - 36 Mb/s	1
ocba-filter	**4**	1 Mb/s - 36 Mb/s	1
ocba-threshold	1	1 Mb/s - **24** Mb/s	1
ocba-weight	1	1 Mb/s - 36 Mb/s	**4**

Table 3.1: Summary of the four variants.

UDP throughput the server wants to achieve: **1 Mb/s**.
Time between two evaluations: **1 second**.
A **good connection**: Minstrel chooses the rate 36 Mb/s or higher for at least 90% of the packets, while in the absence of a jammer.
A **bad connection**: Minstrel chooses the rate 12 Mb/s or lower for at least 90% of the packets, while in the absence of a jammer.
Jammer influence on a node: Nonnegligible change in the RSSI during noise jamming. The bit jammer uses a 5 dBm and the frame jammer a 25 dBm smaller signal strength.

To clarify the meaning of each row in the evaluation tables, we describe them here:
no dec: No decision. Number of times where the algorithm didn't have enough informations to decide and returned $d_{mac} = \textit{no-decision}$.
dec: Decisions. Number of times where the algorithm reached a decision $d_{mac} = \textit{no-jamming}$ or $d_{mac} = \textit{jamming}$.
correct: Number of times where the algorithm recognised the situation (jamming vs. no-jamming) correctly, divided by the total number of decisions.

3.3.3.2 Setting 1

In the first setting, we used two static nodes that had a good connection with each other and the jammer was only capable of interfering with the server. We predicted that the client would give a more accurate result, since his packets are the ones that are disturbed, lowering the PDR and physical rate despite leaving the RSSI high. The server on the other hand should see a small reduction in the rate through the destruction of client-ACKs but record a noticeable reduction of

RSSI while staying on a high physical rate. In case of the frame jammer, the packets of the client are damaged while the server should suffer mainly through the carrier sensing.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	2	363	99%	97%	336	50	ocba-base
ocba-filter	1	365	100%	97%	351	35	ocba-filter
ocba-threshold	117	248	97%	25%	12	374	ocba-threshold
ocba-weight	2	363	99%	97%	336	50	ocba-weight

Table 3.2: Results for setting 1 without a jammer.

Table 3.2 shows no surprises for the absence of a jammer. The explanation for the different results in the ocba-threshold algorithm for the client and server is the rate usage. The client spent a similar amount of time on rate 36 Mb/s and 24 Mb/s, leaving the threshold variant with enough data to evaluate. The server on the other hand used almost exclusively rate 36 Mb/s and changed this only through outliers.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	15	339	98%	92%	364	12	ocba-base
ocba-filter	0	354	98%	92%	376	0	ocba-filter
ocba-threshold	15	339	99%	100%	345	31	ocba-threshold
ocba-weight	15	339	99%	93%	364	12	ocba-weight

Table 3.3: Results for setting 1 with noise jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	10	365	60%	27%	340	61	ocba-base
ocba-filter	0	375	62%	30%	368	33	ocba-filter
ocba-threshold	84	291	96%	94%	117	284	ocba-threshold
ocba-weight	10	365	69%	30%	340	61	ocba-weight

Table 3.4: Results for setting 1 with bit jammer.

In the case of jamming, the client ratios are as high as assumed, following the predicted schema of a reduced physical rate through a lowered PDR while keeping

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	185	283	91%	8%	153	336	ocba-base
ocba-filter	154	314	94%	7%	304	185	ocba-filter
ocba-threshold	185	283	99%	100%	28	461	ocba-threshold
ocba-weight	185	283	97%	12%	153	336	ocba-weight

Table 3.5: Results for setting 1 with frame jammer.

a high RSSI. It has to be noted, that the results of the bit jammer in table 3.4 strengthen the threshold approach, since it is the only variant that is not confused through the use of the 36 Mb/s rate. The price for this is a large number of no-decision results.
The server on the other hand shows a diverse picture. The case of the bit jammer is similar to the one of the client and lies somewhat between the noise and the frame jammer. As expected of the bit jammer, the RSSI is no different from the no-jammer scenario, but the PDR is reduced. This is not detected in the 36 Mb/s area due to it's size presented in the implementation, but when Minstrel switches to a lower rate, the other variants beside ocba-threshold pick it up and decide on jamming too. The frame jammer results of table 3.4 show the expected difficulty. The frame jammer did not lower the RSSI but exploited the carrier sensing, forcing the server to wait. This lead to a large number of intervals in which no data was transmitted[15], producing a high number of no-decision results.
The part where our prediction was off, was in the case of the noise jammer for the server. The correctness shows a good result, but because of a different reason: The RSSI was influenced and also the expected jumps are visible, but the effect was not as strong as we thought it would be. The noise jammer was strong enough to interfere with the ACK packets from the client, reducing the PDR and physical rate without being on a level, it would need to be for a constant RSSI shift.

3.3.3.3 Setting 2

In the second setting, we used the same two static nodes as before but this time, the jammer interfered with both participants. We predicted that both, the server and client, show no significant difference from each other and return success ratios

[15] A case not distinguishable from the time the server actually doesn't have anything to send.

similar to the server from setting 1.
Our prediction was correct as can be seen in the tables 3.6, 3.7 and 3.8. The correctness is higher, because the data packets and the ACK messages both got disrupted, lowering the physical rate even further. This is most obvious in the bit jammer scenario where the 36 Mb/s rate was used only an almost negligible amount of time. The frame jammer worked mainly through the carrier sensing, leaving the physical rate at 36 Mb/s with a fairly high PDR and unchanged RSSI. The reason for this lies in the jammer configuration: The frame jammer was strong enough to cause back-offs, but not as strong, as he would need to be to produce the same amount of bit errors as the other attackers.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	1	344	99%	100%	363	3	ocba-base
ocba-filter	0	345	100%	100%	366	0	ocba-filter
ocba-threshold	1	344	100%	100%	363	3	ocba-threshold
ocba-weight	1	344	100%	100%	363	3	ocba-weight

Table 3.6: Results for setting 2 with noise jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	4	349	88%	100%	356	14	ocba-base
ocba-filter	0	353	92%	100%	370	0	ocba-filter
ocba-threshold	20	333	93%	100%	356	14	ocba-threshold
ocba-weight	4	349	93%	100%	356	14	ocba-weight

Table 3.7: Results for setting 2 with bit jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	168	96	2%	2%	147	109	ocba-base
ocba-filter	101	163	3%	2%	150	106	ocba-filter
ocba-threshold	263	3	100%	100%	3	253	ocba-threshold
ocba-weight	168	96	2%	2%	147	109	ocba-weight

Table 3.8: Results for setting 2 with frame jammer.

3.3.3.4 Setting 3

The third setting is similar to the first one in regards of the jammer position, but due to a loss of the line of sight, the participants start with a bad connection. We predicted, that the condition would leave the threshold variant with the same results as the base. We further assumed, that the other versions would approach the outcome of ocba-threshold in setting 1.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	0	304	98%	94%	322	1	ocba-base
ocba-filter	0	304	100%	96%	323	0	ocba-filter
ocba-threshold	19	304	98%	94%	322	1	ocba-threshold
ocba-weight	0	304	97%	93%	322	1	ocba-weight

Table 3.9: Results for setting 3 without a jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	21	322	99%	94%	324	36	ocba-base
ocba-filter	0	343	99%	98%	360	0	ocba-filter
ocba-threshold	21	322	99%	94%	324	36	ocba-threshold
ocba-weight	21	322	99%	99%	324	36	ocba-weight

Table 3.10: Results for setting 3 with noise jammer.

The no-jammer table 3.9 shows the usual good results for each kind of the implementation. The same goes for the noise jammer, presented in 3.10.
For the bit jammer, the outcome is both good and bad: The ocba-base, ocba-filter and ocba-weight case doubled their success ratio at the server, but the ocba-threshold lost a third. The prediction that the different variants grow close was confirmed and we also found a good situation to show the use of the weight variation: Compared to the others, ocba-weight had a 10% larger correctness. The cause for this lies in the rate distribution. The measurement points were outside of the area for each case except rate 18 Mb/s. There, the points lie perfectly inside, resulting in a mixed view on the situation, leaving each batch with jammer and no-jammer decisions which are then weighted against each other. The client didn't use the 18 Mb/s rate and thus achieved an almost perfect result.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	6	274	100%	62%	273	21	ocba-base
ocba-filter	0	280	100%	64%	294	0	ocba-filter
ocba-threshold	6	274	100%	62%	273	21	ocba-threshold
ocba-weight	6	274	100%	73%	273	21	ocba-weight

Table 3.11: Results for setting 3 with bit jammer.

For the frame jamming we get a similar outcome as for the bit jamming. The difference is, that this time the client also uses the 18 Mb/s rate from time to time. This leads to a similar conclusion as before, demonstrating the use of the weighting. On the other hand it might also lead to one of two different conclusions: The first is, that the highest physical rate of a batch cannot be trusted the same as the others, while the second possible explanation says, that it is just a subset of physical rates that has special conditions making their results less trustworthy than others. However, against the idea of a special subset of physical rates speak the results of other settings where the 18 Mb/s rate was perfectly fine for jamming detection. The reason for the worse server results lies in the distribution of the rates. Client and server used pretty much the same physical rates, but the server used the 18 Mb/s one significantly more often, thus giving the wrong decisions a higher impact.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	8	254	72%	34%	229	45	ocba-base
ocba-filter	2	260	79%	24%	273	1	ocba-filter
ocba-threshold	8	254	72%	34%	229	45	ocba-threshold
ocba-weight	8	254	93%	48%	229	45	ocba-weight

Table 3.12: Results for setting 3 with frame jammer.

3.3.3.5 Moving

For the mobility tests, we positioned the nodes and jammer according to setting 1 and then carried the client up and down the hallway at walking speed. The

distance range was around 5 meters and we tried to keep the rhythm and way of carriage constant for all measurements.
Through prior tests, we knew, that the movement reduces the throughput itself and therefore we had to lower the energy output of the jammer, compared to the statical tests. Otherwise the jammer would have been too strong and the traffic reduced to zero, resulting in 100% no-decision results. We expected the no-jammer to pose no problem but guessed that the jamming detection might give a lower success ratio than before because of the reduced, external interferences.
In table 3.13 we show that even in mobile scenarios, the ocba provides correct results. The physical rate is less stable and in average lower than in setting 1, which is especially visible in the higher correctness and lower no-decision number of ocba-threshold. The static server also suffers under the movements of the client, but the results are still within acceptable bounds.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	10	328	98%	90%	316	41	ocba-base
ocba-filter	0	338	99%	92%	335	22	ocba-filter
ocba-threshold	252	86	79%	83%	172	185	ocba-threshold
ocba-weight	10	328	96%	87%	316	41	ocba-weight

Table 3.13: Results for moving without a jammer.

The results during jamming follow the same pattern: The ocba-threshold variant gives the best correctness for the simple reason that the points of the 36 Mb/s rate lie once again in the no-jammer area. With a large difference but still significant second is the ocba-weight implementation that tries to compensate for the misjudgments of the 36 Mb/s rate through the jamming decisions of the other physical rates. The client has the best success ratio during frame jamming, where the physical rate is very diversely chosen and thus reducing the influence of the 36 Mb/s rate a lot. The server on the other hand is a little bit better than in setting 1, but still, only ocba-threshold delivers a good percentage of correct decisions.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	7	266	42%	41%	281	10	ocba-base
ocba-filter	0	273	40%	40%	290	0	ocba-filter
ocba-threshold	28	245	82%	100%	160	131	ocba-threshold
ocba-weight	7	266	55%	51%	281	10	ocba-weight

Table 3.14: Results for moving with noise jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	2	231	38%	26%	220	26	ocba-base
ocba-filter	0	233	35%	26%	226	20	ocba-filter
ocba-threshold	27	206	87%	81%	105	141	ocba-threshold
ocba-weight	2	231	53%	33%	220	26	ocba-weight

Table 3.15: Results for moving with bit jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	64	174	68%	20%	113	134	ocba-base
ocba-filter	50	188	76%	23%	164	83	ocba-filter
ocba-threshold	65	173	98%	97%	35	212	ocba-threshold
ocba-weight	64	174	83%	26%	113	134	ocba-weight

Table 3.16: Results for moving with frame jammer.

3.3.3.6 TCP and Background Traffic

TCP

We assumed, that our implementation fits TCP as well as UDP and that the results show only a negligible difference. To test this, we used setting 1 and let both nodes try to transmit data as fast as they could.
Our prediction was correct in case of the no-, bit and frame jamming for the client and server, although the server correctness for the bit jammer was better than in setting 1, because the 36 Mb/s rate was used less.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	51	835	100%	99%	761	130	ocba-base
ocba-filter	16	870	100%	100%	809	82	ocba-filter
ocba-threshold	880	6	0%	0%	6	885	ocba-threshold
ocba-weight	51	835	99%	99%	761	130	ocba-weight

Table 3.17: Results for TCP without a jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	14	283	100%	62%	280	24	ocba-base
ocba-filter	0	297	100%	62%	294	10	ocba-filter
ocba-threshold	14	283	100%	98%	203	101	ocba-threshold
ocba-weight	14	283	100%	68%	280	24	ocba-weight

Table 3.18: Results for TCP with bit jammer.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	393	185	99%	8%	13	556	ocba-base
ocba-filter	304	274	100%	13%	556	4	ocba-filter
ocba-threshold	394	184	100%	100%	1	568	ocba-threshold
ocba-weight	393	185	99%	8%	13	556	ocba-weight

Table 3.19: Results for TCP with frame jammer.

The noise on the other hand was different. While client and server detected the noise jammer very well in setting 1, this time the server had large difficulties, doing so. Our analysis showed, that the server used the physical rates 18 Mb/s up to

36 Mb/s and that all the measurement points were well inside the threshold. The reason for this lies in a RSSI shift. In setting 1, the server reported a RSSI of -60 dBm, while this time, the RSSI lied around -75 dBm. It seems obvious to us, that this time, the RSSI shift worked against us, pushing the PDR and physical rate measurements in the no-jammer areas.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	2	269	100%	9%	255	20	ocba-base
ocba-filter	1	270	100%	7%	275	0	ocba-filter
ocba-threshold	2	269	100%	11%	221	54	ocba-threshold
ocba-weight	2	269	100%	11%	255	20	ocba-weight

Table 3.20: Results for TCP with noise jammer.

Background Traffic

The last test we conducted had the aim to prove, that our approaches are capable to distinguish valid traffic on the same frequency from frame jamming. To do so, we placed the transmitting nodes at the positions of setting 1, giving them a good connection. Then we placed two additional nodes, each one right next to one of the others. These additional stations were set to transmit with the maximum transmission rate possible, using UDP. The test stations were adjusted to try achieving a throughput of 1 Mb/s at the server and maximum at the client. Due to the good connection and the carrier sensing we assumed that client and server would keep the physical rate high, while the PDR would also stay in the no-jammer ranges, since the additional traffic respects the protocol and doesn't produce a hidden station situation.

The correctness was as high as we hoped for, while the grown number of no-decision results come from the times where the channel was kept busy by the additional nodes. The ocba-threshold shows once again the problem of leaving a usable physical rate out of calculation, but that was to be expected and therefore not alarming.

client	no dec	dec	correct	correct	dec	no dec	server
ocba-base	506	209	99%	99%	298	496	ocba-base
ocba-filter	490	225	99%	99%	308	486	ocba-filter
ocba-threshold	713	2	0%	40%	10	784	ocba-threshold
ocba-weight	506	209	99%	98%	298	496	ocba-weight

Table 3.21: Results for setting 1 with background traffic instead of a jammer.

3.3.4 Review

In this section, we make a review of the different variants and various influences. This will lead directly to new hypotheses and possible follow-ups, which we present in section 3.4.

Base Results

Overall, the results show that our approach and implementation succeeded in detecting jamming, while staying correct in the absence of an adversary. In the no-jammer cases, the success ratio was between 98% and 100% during good connectivity[16]. In the cases of asymmetrical settings, the client had better results, while no significant difference could be found in setting 2. This also means, that the throughput doesn't influence the result as long as the transmission rate is bigger than zero. Even in the cases where the ocba-base variant didn't work, at least one of the others showed promising results as we will discuss in the following paragraphs.

Influence of Filtering

Looking at all test results, we found a clear trend when compared to the ocba-base variant:

$$(correct_{ocba-base} < 50\%) \rightarrow (correct_{ocba-filter} < correct_{ocba-base})$$
$$(correct_{ocba-base} \geq 50\%) \rightarrow (correct_{ocba-filter} \geq correct_{ocba-base})$$

Since the variant was designed to reduce the effect of outliers, this is exactly what we expected. The rule we stated above has only one exception, and that are measurements with a lot of no-decision results. All the other variants immediately

[16]The server ratio in setting 3 and the moving scenario was a bit lower, but these results were discussed in the specific settings.

accepted this, while the filter provided a few more seconds in which the correct decision lingered on.

Influence of the Threshold Decision

Out of all variations, the decision to ignore the 36 Mb/s had the biggest influence on the correctness. For every kind of jamming, concentrating on the rates only up to 24 Mb/s gave astonishing results. The problem is, that this variation is not practical in the way we used it here. The high number of no-decisions during jamming and no-jamming alike shows this, because the maximal physical rate (36 Mb/s) is also picked during jamming and even more without external disturbances. Additionally, setting 3 questioned the fixation on the 36 Mb/s rate, compared to the highest physical rate of each individual batch.
Combining these arguments, we believe that this point needs a closer evaluation for future projects. It has to be tested if the assumption of the maximal batch rate holds, and if, how the algorithm should use this information. Simply ignoring the highest rate each time (as long as there is data about other physical rates) could lead to a lot of wrong decisions due to outliers, meaning that some sort of additional weight would have to be implemented and determined.

Influence of the Weighting

As we expected, in case of jamming, the success ratio and especially the confidence were superior, when compared to the ocba-base variant. The surprising point is, that the results during no-jamming were not significantly worse, with a maximal difference of 3%. Judging from these results we assume, that the factor four was too low, and that the use of a higher value would increase the jamming detection greatly with little influence in the absence of an attacker. To determine the optimal factor, one will have to rank the jamming and no-jamming cases according to their importance and then compare the false positive and false negative decisions for each variation. Yet we predict, that the outcome will be highly influenced by other possible additions. This means, that the results for the same factor will probably be very different if filtering or a threshold modification is used at the same time, especially since the threshold variant needs to have its own factor determined.

Comparison with Schafroth [1]

Since our work was largely influenced by [1], we directly compare the parts where this is possible and discuss the differences in the other cases. To do this, we will consider the results of his transmitter and receiver based algorithm during frame jamming with our performance. This includes setting 1 and 2, mobility, the use of TCP instead of UDP and the experiment with background traffic.

In his evaluation, Schafroth didn't use a parallel traffic as we did and therefore differentiated the jammer position in separate 'near the sender' and 'near the receiver' cases. His 'in between' on the other hand can be compared directly to our setting 2 so we start with that.
In the case of the frame jammer being placed in the middle of the two nodes, both the transmitter and receiver based algorithm achieved a 98.04% correctness to detect jamming. As discussed in the specific paragraph, we achieved a 100% correctness with the ocba-threshold variant, but because of the enormous use of the 36 Mb/s rate and the diversity in our PDR numbers, our other variations didn't detect it (around 2% success). For the other two jammer positions, Schafroth's correctness ranged from 97.06% to 98.15%. As explained before, the result for the server is slightly better than in setting 1, but with 7% to 12% (not counting ocba-threshold) still nowhere as good. The detection done by the far away node (client in our case) on the other hand is comparable to [1] with a range of 91% in case of ocba-base up to 99% for ocba-threshold.
The effects of mobility are hard to compare, because we used a different walking path. The results however show that our algorithms give a broader correctness range than his, meaning that our worst, ocba-base, only achieved 68%, while Schafroth's transmitter-based approach returned 84.42%. On the other hand, his receiver-based variation only got up to 93.06%, while our ocba-threshold reached 98% correctness (we used the client results on our site, since the server problems with the frame jammer were already explained).

The other settings and studies can be summarised as follow:
The question if the correctness differs for TCP and UDP was answered equally, with the result that the upper layer protocol didn't matter for the jamming detec-

tion. His perfect score in case of the noise and bit jammer were explained by the use of a noise threshold while we only had the physical rate / RSSI correlation. Nonetheless, we still achieved a correctness of over 90% most of the time. The results on background traffic can't be compared because Schafroth used it to build a hidden node situation while we actively avoided this case.

3.4 Possible Follow-ups

In this section, we present some ideas that may be pursued on the basis of our results to gain a higher correctness for jamming detection.

Setting

We tested our approach only for constant jammers, but in theory, most jammers reduce the PDR one way or another and thus influence the physical rate. We couldn't test it, but we see no reason why it shouldn't work with any kind of jammer that targets the data transmission instead of the protocol. Also we only used nodes in ad-hoc mode but expect it to produce comparable results in infrastructure mode. The infrastructure mode would also make it possible to study the physical rates of 48 Mb/s and 54 Mb/s and their no-jammer thresholds. Further studies could also be made to inspect different hardware and their interdependencies[17]. Lastly, the rate selection algorithm we used was Minstrel, but the question if the whole idea only works with this specific algorithm or if it has a larger validity, is still open.

Algorithm

To improve the correctness, we already suggested some variations in the review parts. As an additional measurement, the RSSI itself could be used in the sense, that a rapid change over time would reveal the presence of a jammer. With such an additional module, the noise jammer during the TCP tests would have been detected at the server due to the RSSI shift from -60 dBm to -75 dBm. A fixed threshold is impossible, but if we assume a non-disturbed setup phase and no

[17] For example, is it only important what type of card the sending node uses or does the receivers card matter too?

movement, it is easy to implement. An additional suggestion that may reduce the PDR diversity and therefore strengthen this measurement could be the use of a fixed threshold for a minimal value of t_i that is needed for a physical rate to be considered for the detection. The reason for this is our observation that the PDR that Minstrel reports closes in asymptotically to the real PDR as more packets are transmitted on a specific physical rate. Our last idea is not to use discrete physical rates as input but average values like in [1]. Further than that, we don't think, that our algorithm can be improved substantially without adding additional measurements like the noise strength or the bad packet ratio. Still, a combination of all tested variations, together with more precisely tuned threshold areas and factors might prove to be enough for every common situation.
In an environment where all possible measurements are available, a combination with the approach to use the channel capacity might further improve the correctness and certainty of the decisions, adding the possibility to detect jamming even without any throughput at all.

3.5 Recommended Implementation

In the case, that someone wants to implement our approach directly, we present the following recommendations as a starting point.

name	buffer size	physical rates	weight *jamming*/*no-jamming*
ocba-suggestion	5	1 Mb/s - 36 Mb/s	6

Table 3.22: Suggested configuration.

In case of a planned use in infrastructure mode, the physical rates considered should include all from 1 Mb/s up to 54 Mb/s. For the interval between two evaluations we suggest to choose more than one second, but this is probably fixed through external requirements. As a compromise with the ocba-threshold variant we recommend to add an additional weight in the process from the different d_i to the batch-decision $[d]_{mac}$. This weight should ensure that no physical rate - regardless of its t_i value - has more than 49% influence in the final decision.
We also recommend the implementation of a module to detect large changes in

the RSSI, since especially the jumps discussed in section 3.1.1 are a strong hint for the presence of an attacker.

4 Rate Switching

In this chapter we evaluate the performance of different existing rate switching algorithms under jamming and elaborate ideas for improvements. First we will give an overview on the technical background of rate switching and introduce the algorithms AMRR, Onoe, SampleRate and Minstrel. After a short part on comparing these algorithms we will examine the performance of Minstrel in different jamming scenarios and then explain and test some of our improvement ideas.

4.1 Technical Background

With the support of different physical rates the problem arises to chose the best physical rate for transmitting. With not only different encoding schemes but as well different modulations, the problem became more difficult to be solved effectively with a minimal overhead.
While the first rate control algorithms were fairly simple and only increased or decreased the physical rate by one depending on a simple rule, newer algorithms are more sophisticated and use different measures to determine the current channel quality and then try to choose the best rate. Some algorithms use the SNR to look up the best rate, others try to evaluate the packet losses or look at the achievable throughput. The main problem all face is that they should adapt fast to changed channel and environment conditions and have as few overhead as possible. While in a static environment the theoretical best can be determined, changing conditions as mobility, changes in the environment or a jammer can affect this optimum. Therefore, an algorithm has to be able to adapt to these changes. The faster the adaptation is, the less time is spent on a non-optimal rate.
In [7] the authors developed an algorithm that tries to determine the best physical rate for each individual packet. Therefore they created an offline profile of the

SNR and the according rate to use. Now all that is needed is an incoming packet, e.g. an ACK, to measure the current SNR. So basically if the first packet is sent at the lowest physical rate, all following packets will have an up to date SNR from the previous ACK.
But what they did not consider in their work are jamming scenarios. As a jammer affects the SNR at the receiver in another way than at the sender, these profiles become futile, unless the sender knows the SNR at the receiver[1].
Another approach is to use the PDR to determine what rate is best to use. This however always has some overhead as the PDR depends on the used physical rate and can not be estimated good enough for other rates. While older algorithms as Onoe and AMRR used the PDR just to switch to either the next higher or next lower rate, SampleRate and it's successor Minstrel use the PDR to do online profiling and estimate the currently possible throughput for each rate. To do so, they use sampling packets sent at random rates to check the conditions. This allows the algorithm to find the best rate in a short time. We now give a short introduction to the algorithms AMRR, Onoe, SampleRate and Minstrel, those familiar with them may as well skip these parts.

AMRR

This algorithm uses the given Multi Rate Retry (MRR) functionality of the Hardware Abstraction Layer (HAL) of Atheros WLAN chipsets. It sets up the pairs of rates and retry counts as

$$[(r_0, 1), (r_0 - 1, 1), (r_0 - 2, 1), (0, 1)]$$

where the first value denotes the number of the physical rate used and the second value in each bracket is the retry count, r_0 is determined by the performance of the algorithm in the previous interval using the last physical rate r_0. If there were more failures than a given threshold, the rate will be reduced by one and if there were more than a certain amount of successful transmissions, then the rate will be increased. For the exact implementation details we refer to [13].

[1] The SNR could be included in the header of each packet, but then it would no longer conform to the IEEE 802.11 standard.

Onoe

This algorithm tries to find the highest physical rate with less than 50% packet loss. The idea comes from the fact that 802.11b networks only use the rates 1, 2, 5.5 and 11 Mb/s where the next lower rate is always outperformed when there is less than 50% packet loss. For the 802.11g rates the algorithm is less optimal as the rates are closer together than by a factor of two. We could not find any references on this algorithm as all the referenced URLs are offline. Perhaps the only remaining documentation are the comments in the source code found in MadWifi.

SampleRate

SampleRate starts sending at the highest available physical rate. After four successive transmission failures the algorithm will lower the bitrate. In addition SampleRate sends packets on random rates to do sampling. A lower rate is only sampled if its lossless transmission time is lower then the average transmission time of the current rate in the last interval of 10 seconds. No rate that had four successive transmission failures in this interval is used. In [8] the implementation is described in more detail.

Minstrel

Minstrel[2] keeps for each physical rate a record of the number of packets it attempted to send on this rate and the number of those that were successful. In addition it stores for each rate the number of successful and total transmissions in the current interval of 100ms, the resulting current PDR and the EWMA [3] of the PDR. The EWMA is calculated as follows:

$$EWMA_{new} = \alpha * EWMA_{old} + (100 - \alpha) * PDR_{current}.$$

In Minstrel α is called the $\text{EWMA}_{\text{Level}}$ and is by default 75, meaning the old EWMA accounts for 75% and the current PDR for 25% to the new EWMA.
To determine the next physical rate a sample will be sent on, the algorithm uses

[2] http://linuxwireless.org/en/developers/Documentation/mac80211/RateControl/minstrel
[3] exponentially weighted moving average

a sample table where each row contains all possible rates randomly permuted. As there is no benefit in sending packets at lower than the current rate while the current rate still has a high PDR, the randomly chosen rate to send the sample on will only be used if it could provide more throughput, assuming it would have no transmission failures. If this is not the case, the sample is deferred and the current rate is used instead. To avoid burst sampling, two deferred samples count as one actually sent sample. The default setting is to send 10% of the packets on a sample rate if MRR is enabled or 5% otherwise.
How to set up the Multi Rate Retry Chain (MRRC) is another important factor for the performance of a rate switching algorithm. Minstrel uses three different MRRCs, one for all normal packets, one for sampling packets at a higher than the current rate and one for sampling packets at a lower rate. The rates used are similar but are arranged differently in each chain. There are the rates that have the best throughput, the second best throughput and the highest probability. Additionally a random rate[4] and as well the lowest rate are used. The three chains are shown in table 4.1 and the rates are used from top (first rate) down to the bottom (4th rate) with a variable number of retries on each rate. The left column is for samples with a lower physical rate than the current one, the middle column is for sampling at higher rates and when no sample is sent the chain to the right is used. This setup allows Minstrel to sample at lower rates as soon as

Try	Sample Rate		Normal Rate
	random < best	random > best	
1	Best throughput	Random rate	Best throughput
2	Random rate	Best throughput	Next best throughput
3	Best probability	Best probability	Best probability
4	Lowest baserate	Lowest baserate	Lowest baserate

Table 4.1: Multi Rate Retry Chains of the Minstrel rate control algorithm. Source: http://linuxwireless.org/

the current rate gets worse and transmission failures occur. While in practice this strategy seems to yield good results, in theory there are some flaws to this. A rate should only be sampled when its maximum throughput is higher than the throughput on the current rate. For higher rates this always holds, but they usu-

[4]Sometimes also called sample rate.

ally have more transmission failures decreasing the throughput and sampling often will waste throughput.
In normal situations it is more likely that the rates close to the current one will become the best and therefore should be sampled more often than the others. It is useless to sample on lower rates when the link is good and when the link gets worse not all lower rates should be sampled with the same frequency. The lower the rate is, the less it should be sampled. The same holds for higher rates, less sampling the higher the rate is.
In the case of a jammer, where the link quality can change fast, this strategy could decrease the throughput, but if a jamming detection algorithm is available the strategy can be adjusted. An idea is to use two tables[5] and switch between these when the jammer is turned on or off. This allows the rate control algorithm to swiftly switch back to higher rates. The faster the jamming detection works the faster the transmission is back at full speed. However, when the decision takes too long the rate control algorithm will have altered the table already rendering it useless to switch to the other table.
In this case the sampling could be adapted to send additional sampling packets on the previously best rate as soon as the jammer is detected. This could help to switch back faster when the jammer is turned off.

Measurement Duration

The WLAN channel suffers a lot from short term variations independent of the presence of an attacker. But we assume, that the long-term performance does not change during each of our experiments. To get reliable results we have to adjust the duration of our experiments accordingly.
We run a long experiment to evaluate what duration each test should have for reliable results. In the figures 4.1 - 4.3 we show the confidence intervals for different test durations using the physical rate 36 Mb/s under a frame jammer. While the 60 seconds tests still vary by up to 30%, with a duration of 5 minutes the variation is down to less then 10%. The tests support our assumption and allow us to compare the performance of the rate switching algorithm to the best physical rate.

[5]The table contains throughput, PDR, EWMA, success and attempts

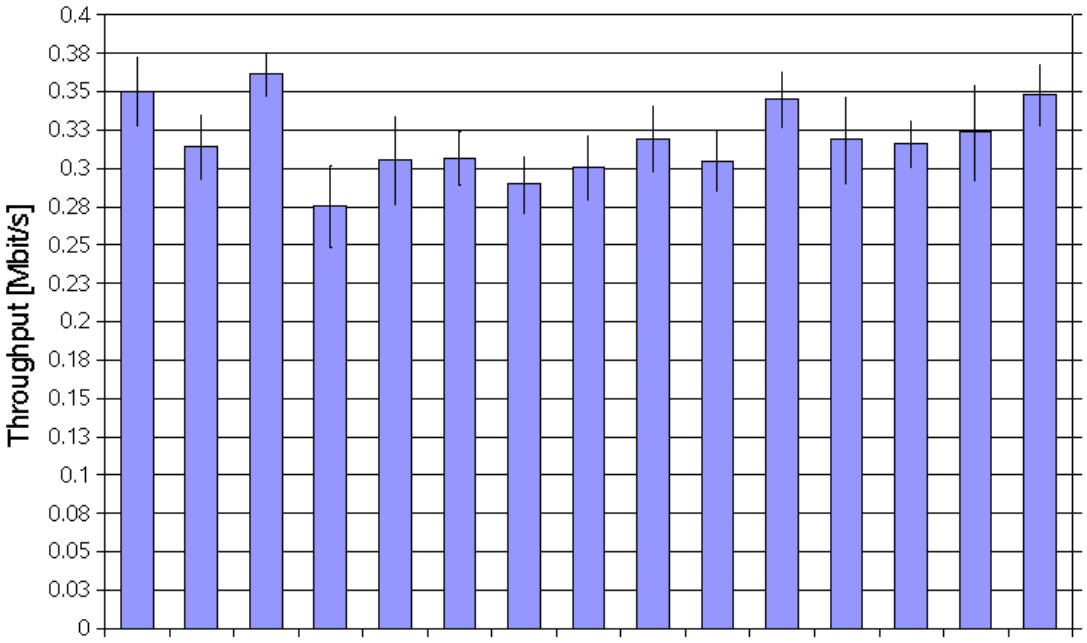

Figure 4.1: The throughput reached in 15 one minute tests using 36 Mb/s under a frame jammer. The error bars indicate the 5% confidence interval.

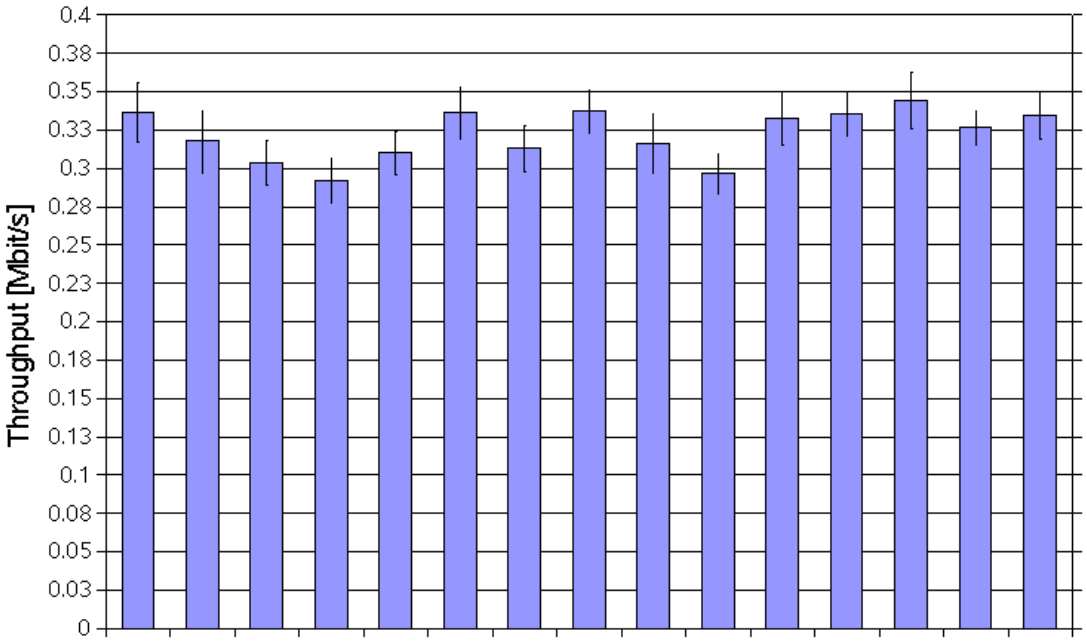

Figure 4.2: The throughput reached in 15 two minute tests using 36 Mb/s under a frame jammer. The error bars indicate the 5% confidence interval.

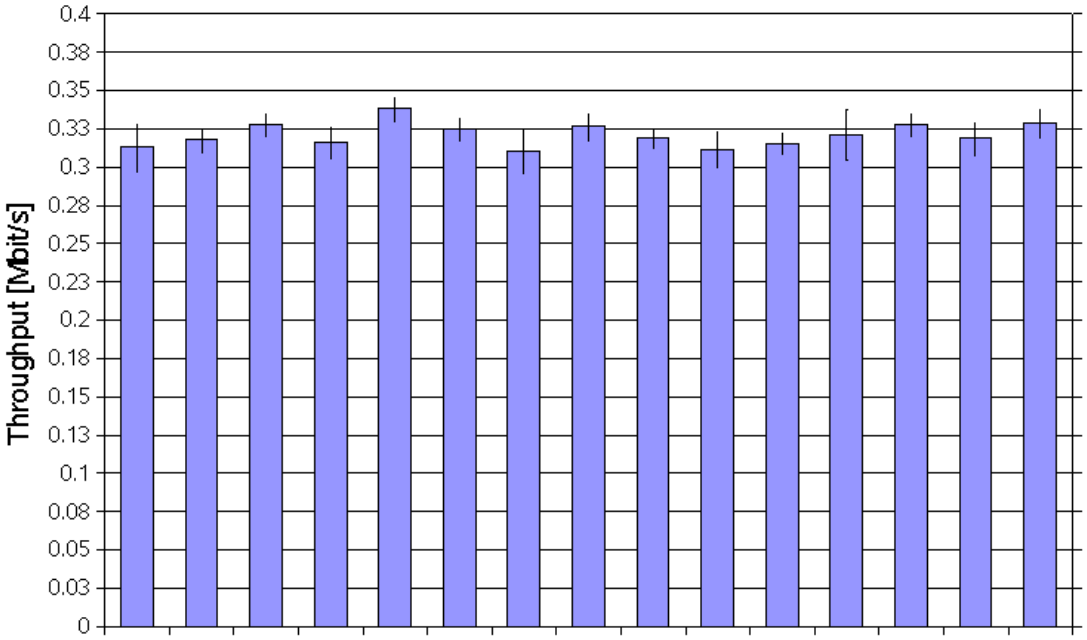

Figure 4.3: The throughput reached in 15 five minute tests using 36 Mb/s under a frame jammer. The error bars indicate the 5% confidence interval.

4.2 Comparing Existing Algorithms

We want to compare Minstrel, AMRR, Onoe and SampleRate by their performance under different jamming scenarios. For this we installed the MadWifi[6] driver on the sender laptop so we could switch between the different algorithms.
When we tried to test the algorithms, we found that in ad-hoc mode AMRR, Onoe and SampleRate did not work. So after several failed attempts we decided to use an access point for this comparison. The basic setup was still the same, except that now the receiver was linked over an access point with a 100 Mb/s Ethernet connection. A side-effect was that for these experiments all physical rates worked.
After a few experiments we found that comparing the different algorithms was not as easy as we thought. For the experiments we ran one minute tests with the four rate control algorithms and each fixed rate. The fixed rates were tested using ath5k and Minstrel. We used the same jammer settings for all physical rates and algorithms. As shown in figure 4.4 two of the algorithms seem to have outperformed the best fixed rate, what is rather unlikely in this static setting. Therefore we investigated this issue further and found the problem to be the reloading of the wireless module that is necessary to switch between the different algorithms. The effect the jammer has on the performance of the WLAN channel changed when reloading the module.
We ran an additional series of experiments where we used Minstrel and the physical rates 5.5 Mb/s and 24 Mb/s to show that reloading the wireless module under jamming has an effect on the throughput. For this we used a frame jammer at -25 dBm and the setting 4. Figure 4.5 - 4.7 show the average throughput for Minstrel, 5.5 Mb/s and 24 Mb/s and the corresponding 5% confidence interval. Each average is taken from 30 ten seconds intervals to calculate the confidence interval. While the 5.5 Mb/s rate is quite stable the performance of the 24 Mb/s rate as well as of Minstrel varies a lot between the different tests. As the confidence intervals are mostly small we can conclude that not only restarting the jammer but as well reloading the wireless module or restarting the sender effects the throughput over WLAN.
As there is no possibility to change the rate control algorithm without reloading the wireless module we tried to find another way to compare the algorithms. The

[6] http://madwifi-project.org/

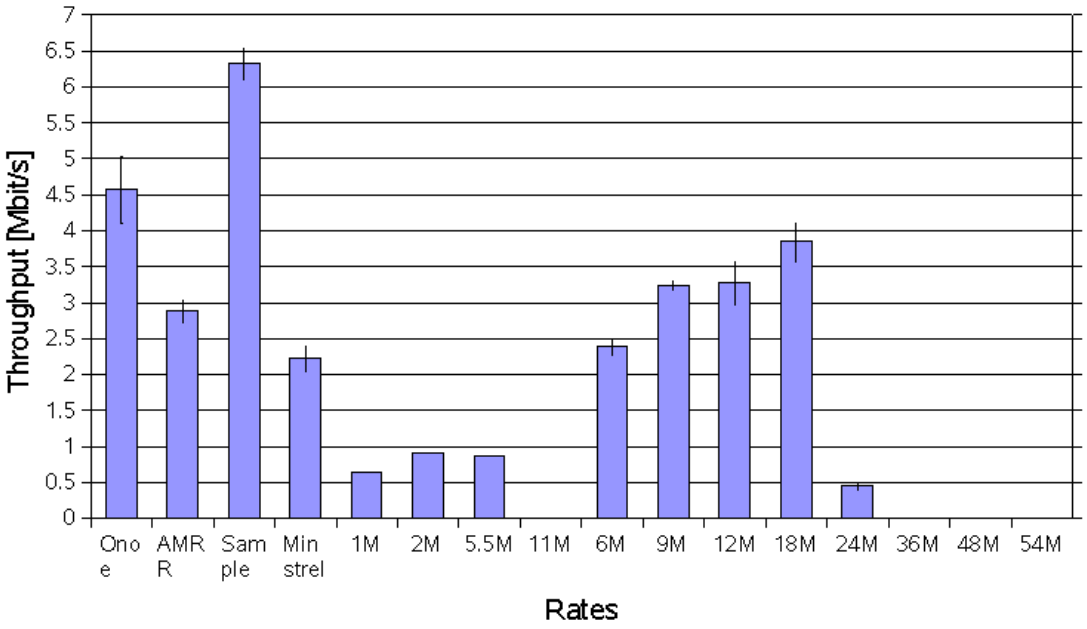

Figure 4.4: Performance of Minstrel, AMRR, Onoe and SampleRate under a noise jammer in a two minute test. The error bars indicate the 5% confidence interval.

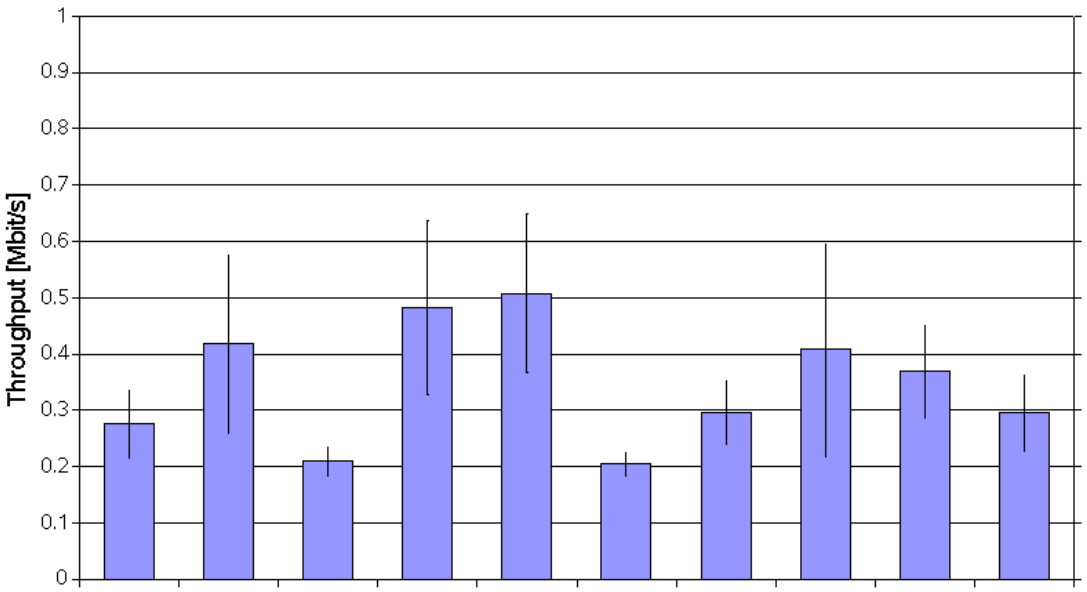

Figure 4.5: The throughput in Mbit/s that was reached in ten separate five minute tests under frame jamming using Minstrel and reloading the wireless module after each test. The error bars indicate the 5% confidence interval.

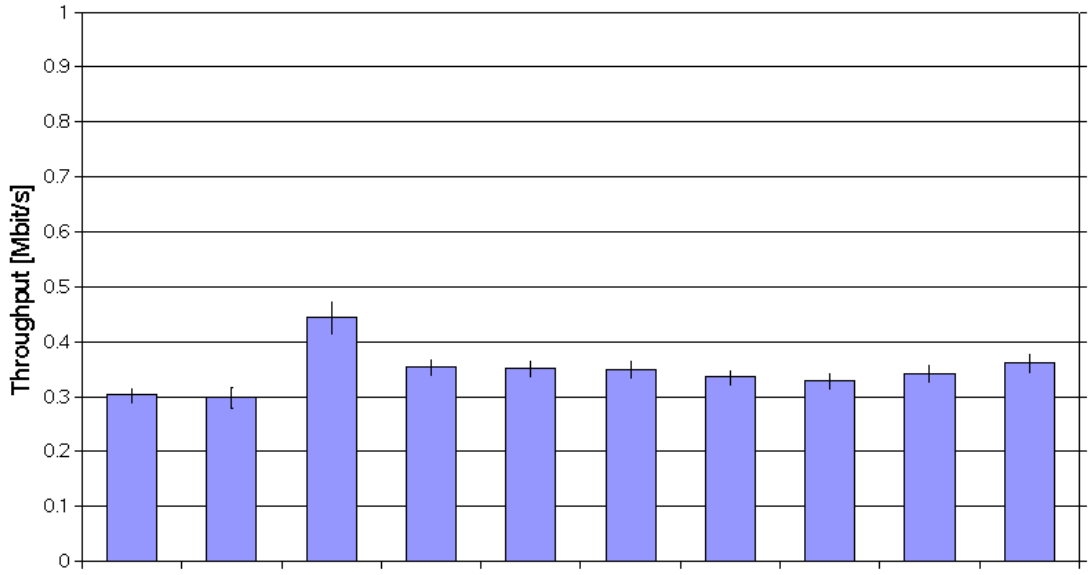

Figure 4.6: The throughput in Mbit/s that was reached in ten separate five minute tests under frame jamming using the fixed rate 5.5 Mb/s and reloading the wireless module after each test. The error bars indicate the 5% confidence interval.

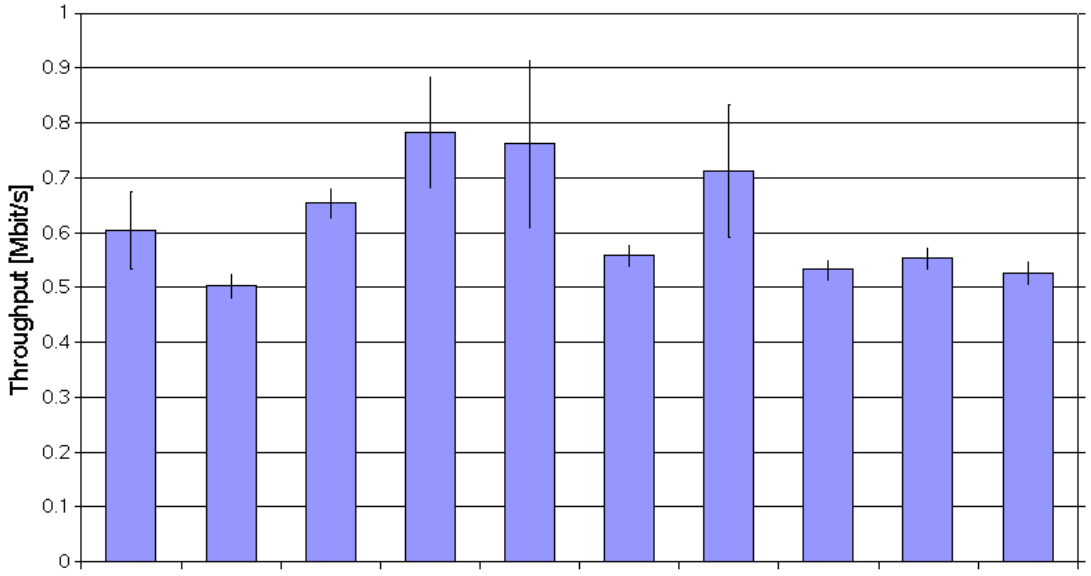

Figure 4.7: The throughput in Mbit/s that was reached in ten separate five minute tests under frame jamming using the fixed rate 24 Mb/s and reloading the wireless module after each test. The error bars indicate the 5% confidence interval.

only real comparison probably would be to run hundreds of tests and reload the wireless module before each test. This would allow averaging out the variations. However, as this would take more time than we had left to achieve reliable values, we had to drop this method and took a different approach. Instead of doing a great number of test we wanted to compare the relative performance of the different algorithms. We compared the throughput the rate control algorithm reached with the one of the best fixed rate. The performance of the fixed rates should be independent of the currently loaded rate control algorithm and give us a good estimation how well each algorithm performed.
But the next series of experiments showed that we were wrong, as illustrated in figure 4.8. In the Linux kernel[7] the rate control algorithm has to handle fixed rates itself and therefore would have to check if a fixed rate was set. But as AMRR and Onoe fail to do so, the fixed rate is simply ignored and the rate control is used all the time. This forced us to drop this method as well and leave the comparison of these algorithms unfinished.

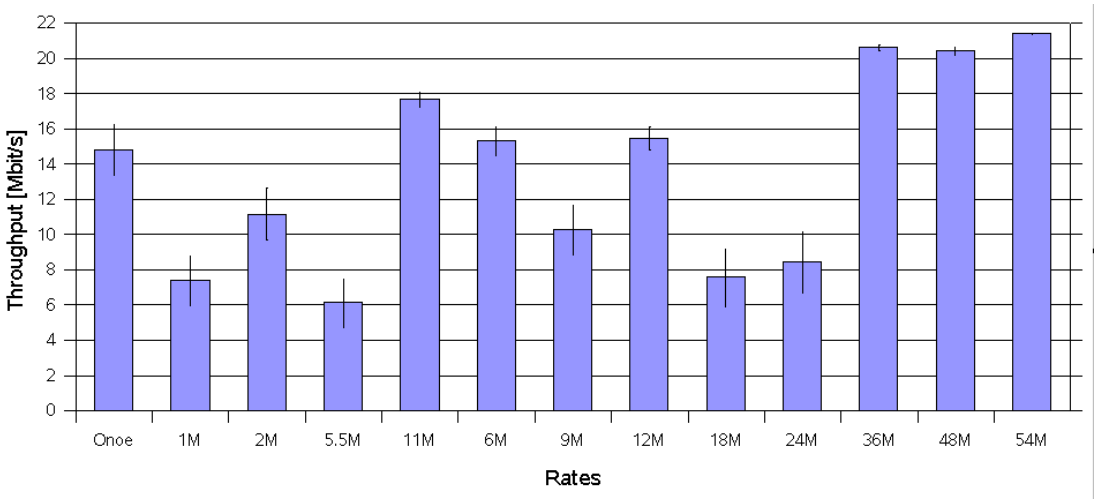

Figure 4.8: The rates on the x-axis denote the fixed physical rate that was set for the test, but the throughput is too high for the rates and shows that the set rates are not actually used when Onoe is loaded.

[7]Version 2.6.31.4

4.3 Minstrel Performance Evaluation

In this section we evaluate the performance of Minstrel under different jammers in an ad-hoc network. The setup for the following tests is,if not mentioned otherwise, described in section 2.2 as setting 2. In all tests node B (client) tries to send 100 Mbit/s UDP traffic using the maximum packet size.

4.3.1 No Jammer

To have some reference values for the actual jammer tests we first ran the experiment without any interference from the jammer. This allows us to see how much throughput actually can be achieved with this hard- and software setting.

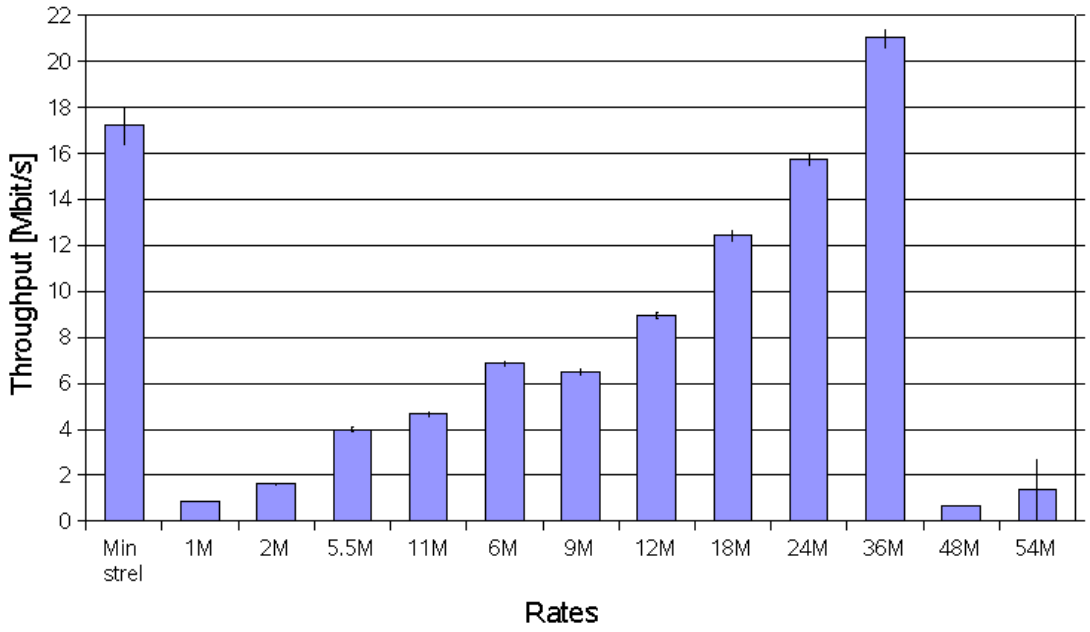

Figure 4.9: Throughput reached in a one minute test without any jammer in Mbit/s. The error bars indicate the 5% confidence interval.

Rather unexpected we found that even when no jammer was present the physical rates 48 Mb/s and 54 Mb/s performed poorly and reached in this test only 0.68 Mbit/s and 1.39 Mbit/s respectively as shown in figure 4.9. We assumed the outcome would be closer to the tests in infrastructure mode listed in table 4.2.
As we could not find an explanation for this, we investigated this issue further and

R_f	1	2	5.5	11	6	9	12	18	24	36	48	54
R_m	0.9	1.8	4.5	8.0	5.1	7.3	9.5	13.3	16.8	22.4	26.9	29.0

Table 4.2: R_f stands for the physical rate and R_m denotes the measured throughput in Mbit/s.

found, that even when the two machines are on top of each other and hence the cards just a few centimetres apart, the throughput is really low on the two highest rates. However, as the rates work fine in infrastructure mode we assume the problem is caused by the implementation of the ad-hoc mode and is software related. Investigating this issue further would have been out of scope and we therefore just excluded these two rates from our tests.
It still needs to be discussed how this issue interfered with our tests. While for the other fixed rates there is no influence at all, the performance of the rate switching algorithm does suffer. As the rates do work sometimes for a short time period and then again fail completely, the sample packets Minstrel sends will be sent successfully now and then and Minstrel will switch to this rate and be able to send a burst of packets on this rate which increases it's EWMA. But after a short period of time the rate will stop working again and no packets will be delivered, reducing the throughput to zero for several hundred milliseconds. This decreases the overall performance of Minstrel.
While this is a quite severe issue when no jammer is present, it has less effect when a noise or bit jammer is active. As long as the jammer is not too weak the rates 48 Mb/s and 54 Mb/s will stop working as their forward error correction only allows few errors. For a frame jammer it depends on the setting. If the frame jammer targets the sender the two highest rates will still work from time to time and reduce the performance of the algorithm. If the target is the receiver and the sender can't hear the frame jammer the situation is similar to a bit jammer and the algorithm should work as if all rates would function properly.

4.3.2 Noise Jammer

Now that we've seen the case where no jammer is present we can move on and take a look at a scenario with a noise jammer[8]. For this test we set the noise jammer to -10 dBm transmission power. In figure 4.10 one can see the throughput diagram.

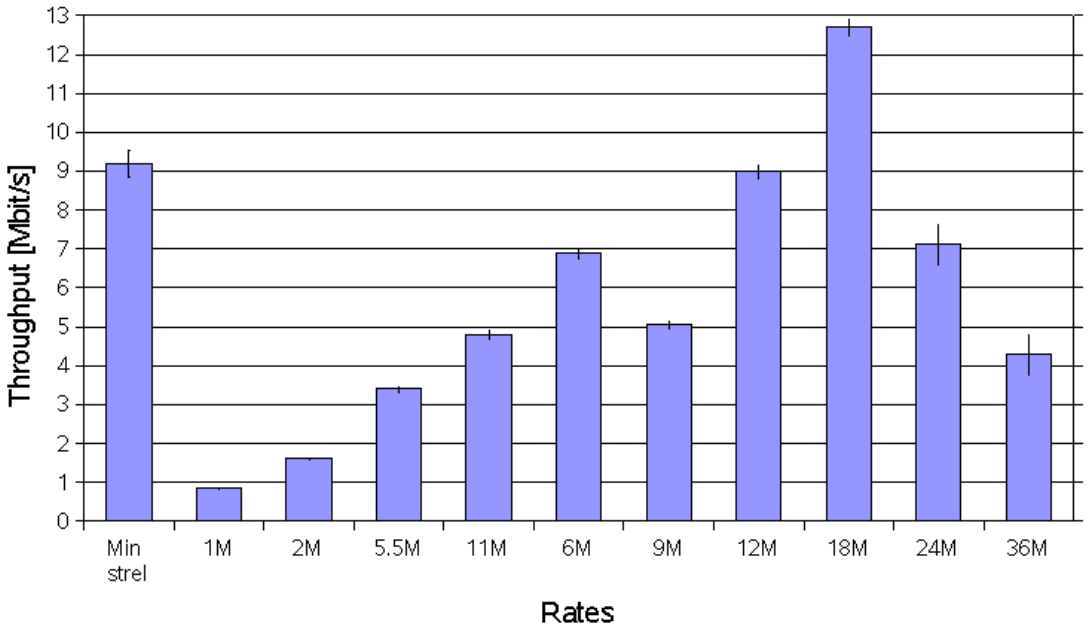

Figure 4.10: Throughput reached in a one minute tests under noise jamming in Mbit/s.

While the physical rates with stronger forward error correction (FEC) are unaffected by the jammer, the higher rates that have less correction bits reach a lot less throughput. As well a difference makes the modulation scheme used. While the OFDM rates 24 Mb/s and 36 Mb/s lose 55% and 80% of their throughput, the DSSS rate 11M that has no error correction bits has a loss of only 24%.
Minstrel lies with 9.6 Mbit/s between the best and second best fixed rate that were 18 Mb/s with a throughput of 13.2 Mbit/s and the 12 Mb/s rate that reached 9.3 Mbit/s. This means that Minstrel reached 73% of the best fixed rate. While this value first seems to be quite low, one has to consider that 10% of the packets are sent at another than the best rate to get sample data. But still these 73% allow for improvement.

[8]How we realized the noise jammer with our hardware is described in chapter 2.3

4.3.3 Bit Jammer

Next we ran our experiments with a bit jammer where Minstrel only reached 0.91 Mbit/s while the fixed rates were nearly unaffected by the jammer. Only the 36 Mb/s rate had it's throughput reduced by 18%. First we thought this was the normal behaviour of Minstrel under a bit jammer.
We thought that the problem could be either the throughput calculation or the initialization of the PDR to zero. By changing the initial PDR value to 100% and using, instead of the theoretical throughput, values actually reached in tests, we suddenly had a great improvement in performance. However, as we investigated this issue further we found that Minstrel did not initialize correctly and could not use any but the lowest physical rate.
With this knowledge we could actually check before the tests whether Minstrel can use all rates correctly and avoid this issue affecting the test results. The rerun of the bit jammer test was with a slightly different setup, the jammer was now placed at a distance of about 22m from the sender and about 13m from the receiver as it can be seen in figure 2.6 and the transmission power was set to -7 dBm.
The best physical rate in this scenario was 12 Mb/s with a throughput of 3.81 Mbit/s while Minstrel only reached 58% of this value as one can see in figure 4.11. However, it is still a gain of 200% over using the lowest physical rate.
In contrast to the results found in [1], where the DSSS modulation seemed robust against bit jamming and the 1 Mb/s and 2 Mb/s rates still reached about 90% of their maximal throughput when the OFDM rates 6 Mb/s and 9 Mb/s were down at about 10%, in this test the 1 Mb/s and 2 Mb/s rates only reach about 50% while the OFDM rates 6 Mb/s and 9 Mb/s as well have about 50% and therefore outperform the DSSS rates.

4.3.4 Frame Jammer

The frame jammer[9] is different to the others, as it does not try to hinder the stations from receiving but from sending packets. To do so the frame jammer needs to target the sender.
For this test we used a low transmission power of just -33 dBm in setting 4 to

[9]The frame jammer is explained in more detail in chapter 2.2

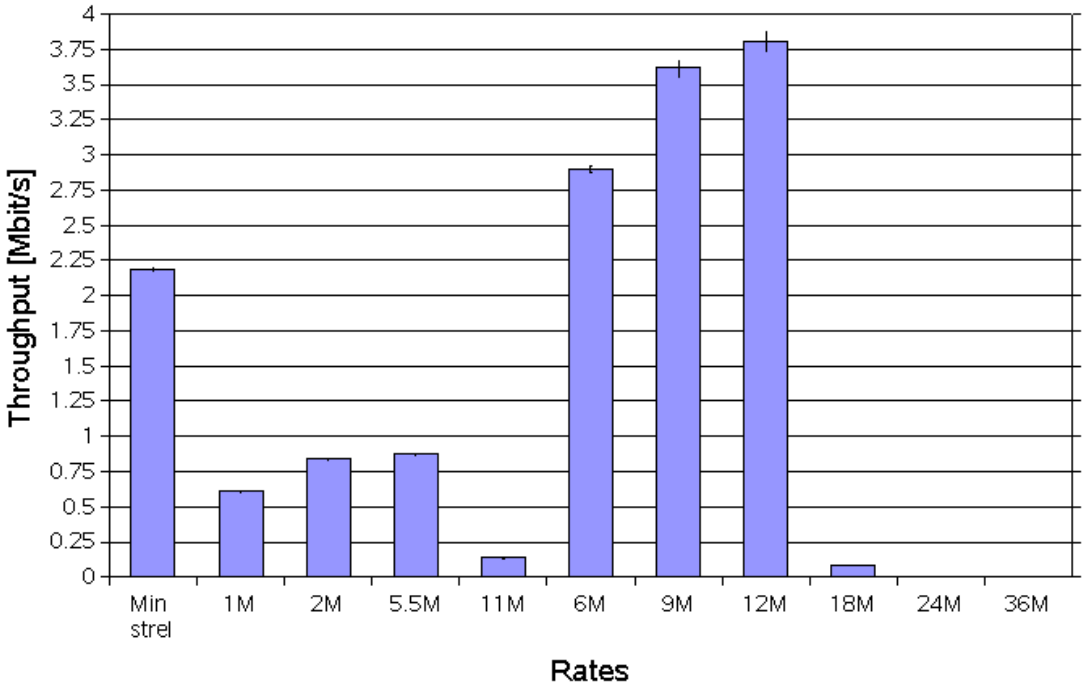

Figure 4.11: Performance under bit jamming in a five minute experiment using different fixed rates and the rate control algorithm Minstrel.

ensure that the jammer's interference with normal packets is not too strong. But the client will send less packets as the channel is sensed to be busy. This means that even when the throughput is low the PDR is still rather high and therefore all rates have a lower throughput as shown in figure 4.12.
While the best physical rate in this test was 18 Mb/s with a throughput of 6.78 Mbit/s, Minstrel only used 24 Mb/s and higher rates as it can be seen in figure 4.14, that shows the total number of packets Minstrel sent on each rate and the number of successful transmissions. One possible explanation for this behaviour is, that because the frame jammer sends at 11 Mb/s the channel is busy for a long time and the legitimate sender sends most packets separately and has to wait after it finishes sending a packet. This reduces the channel capacity more for the higher rates than it does for the lower rates an thus the throughput calculation done by Minstrel is no longer correct.
If the frame jammer only jams the receiver it is actually quite similar to a bit jammer. To show this we used setting 1 and set the transmission power to -25 dBm. As expected, in this test only the higher rates suffered from a throughput loss as shown in figure 4.13

4.3.5 Review on Minstrel's Performance

In setting 1, where the jammer is not in line of sight of the sender and much further away from it then from the receiver, frame and bit jammer have a similar effect and require about the same transmission power to achieve the same throughput degradation. The noise jammer requires more power for the same effect. Minstrel reaches between 50% and 80% of the throughput the best physical rate achieves.
In setting 4 the frame jammer is in line of sight of both stations and has a different influence on the throughput. It requires less transmission power to degrade the overall throughput to the same level as in setting 1 and all physical rates are affected. Minstrel performs worse in this scenario and only reaches about 20% of the best physical rate. For the same effect on the throughput the bit and noise jammer need a higher transmission power than in setting 1, but Minstrel still reaches about 50% to 80% of the throughput of the best physical rate.

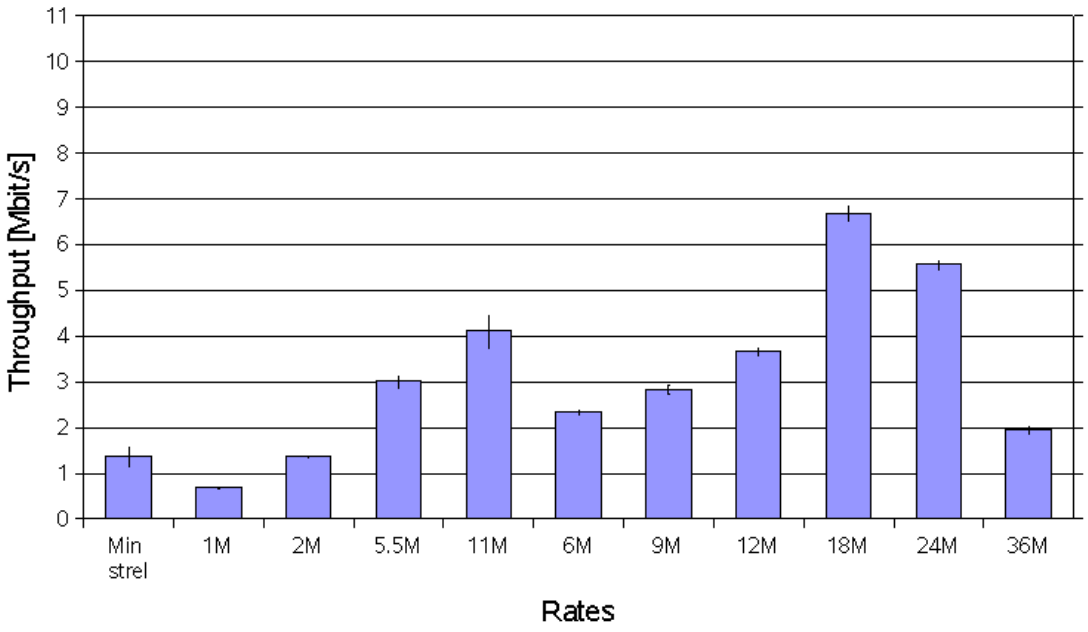

Figure 4.12: The throughput in Mbit/s during one minute tests using different fixed rates and the rate control algorithm Minstrel. Both stations are under weak frame jamming.

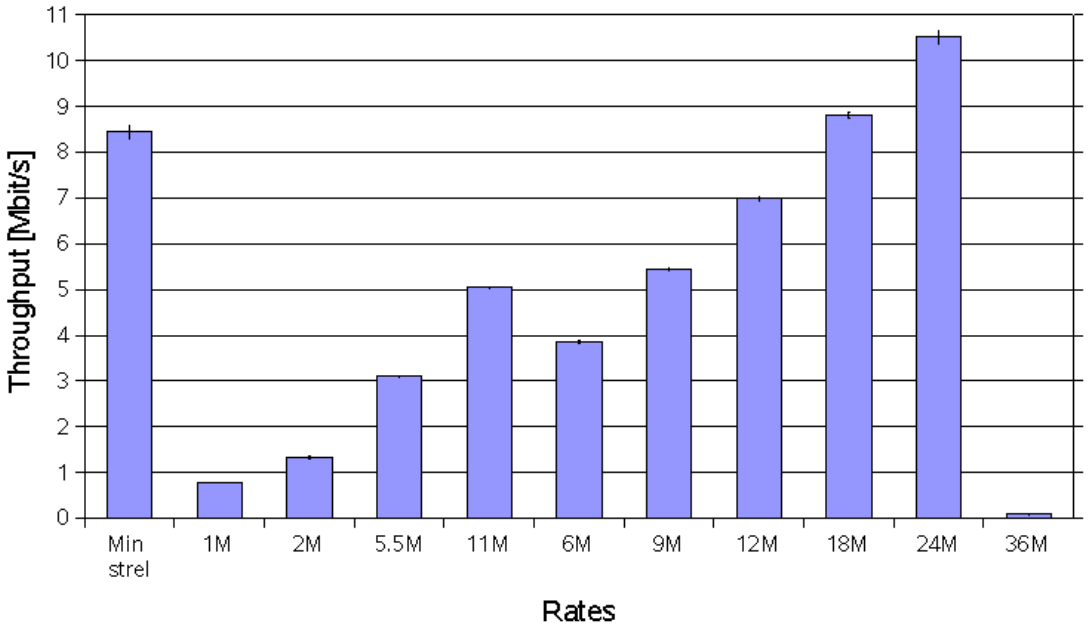

Figure 4.13: The throughput in Mbit/s during one minute tests using different fixed rates and the rate control algorithm Minstrel. Only the receiver is under frame jamming.

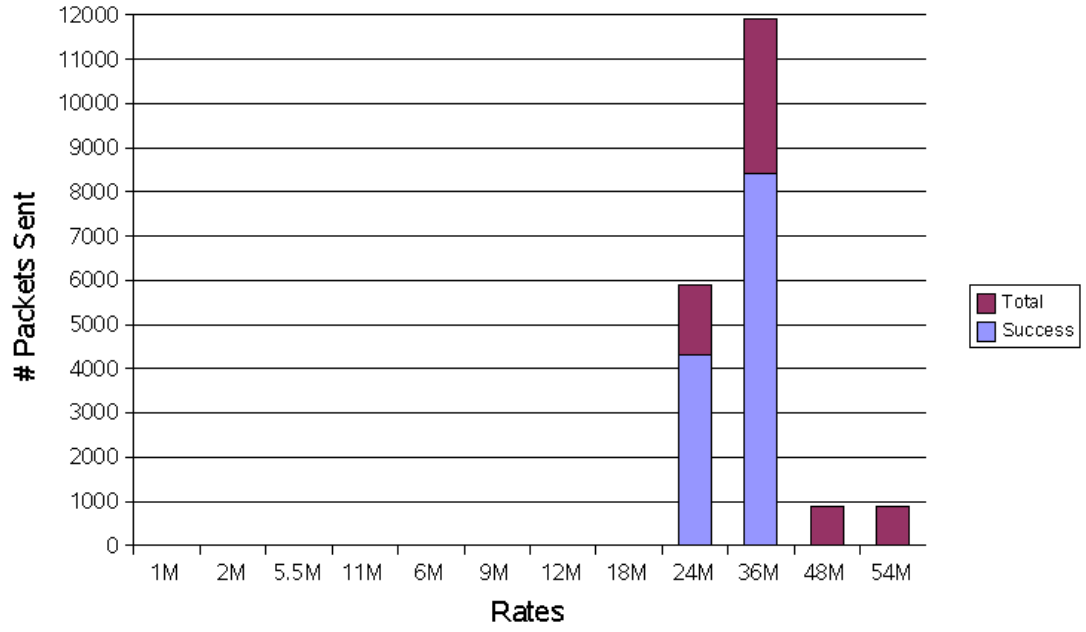

Figure 4.14: The number of successful and failed transmissions during a one minute time interval under a frame jammer using Minstrel.

4.4 Improvement Ideas

We showed so far, that Minstrel does not perform as well as the best fixed rate in a static setting under jamming. In this chapter we now discuss and test some ideas to improve the performance of Minstrel under jamming without decreasing its flexibility too much.

To implement the different algorithms on the same machine we used different kernel versions[10] and compiled each with a different adapted version of Minstrel. This simplified the experiments as we did not have to change the machine for each test and allowed us to keep the positions exactly the same during the experiments.

Switching between two algorithms required rebooting the system, what as well reloads the wireless module and as discussed in section 4.1 this affects the impact the jammer has on the communication. This means that we can't directly compare the different algorithms by the achieved throughput. We came up with two different measurements on how optimal the algorithms are. First we have the already

[10]The kernel versions used are 2.6.31.4-7 and these minor changes to the kernel should not have had any influence on the outcome of the experiments.

addressed ratio between the measured throughput for the algorithm and the best fixed rate. We will refer to it as the Max-Throughput-Ratio (MTR). We assume, that this ratio is a good indication which algorithm performs better as long as the throughput of the best fixed rate is about the same. The more these values differ in the different tests the less we can say about its significance.
We looked at another ratio to evaluate the tests but it could not provide a reliable measure for the algorithms performance, because the recorded data did not match the measured results[11]. We assume that this could be due to incorrectly reported data by the hardware, but could not verify this. While the data reported by Minstrel does differ from the packets we monitored with an additional laptop, we could not find any pattern that would support our assumption.

4.4.1 Packet Size

As found in [14], under some jamming scenarios a smaller packet size can increase the throughput. The basic idea is, that a smaller packet has a higher probability of being received correctly than a larger packet which may increase the throughput. To test this with our jammer settings we did a series of experiments where we tested 4 different packet sizes with 4 different jammer settings[12]. The tests had a duration of one minute for each packet size and rate. In addition to the fixed rates we used the Minstrel rate control algorithm. The result was, that for all three different jammer settings a smaller packet size decreases the throughput. The best performance was reached in all scenarios with 1470 bytes[13] of data per packet. In Figure 4.15 the throughput reached with Minstrel and at different fixed rates under a noise jammer is shown.

4.4.2 Less Samples

As discussed above in section 4.1, Minstrel does not yet sample in an optimal way. To improve the performance, the amount of samples has to be reduced.
The first algorithm we implemented takes an approach that only works in our

[11] While we measured a PDR of 60% for the physical rate 11 Mb/s when testing the algorithm, the physical rate 11 Mb/s actually performed poorly and only reached 0.5 Mbit/s.
[12] No jammer, noise jammer, bit jammer and frame jammer with the setting 2.
[13] Not including the headers which are an additional 30 bytes.

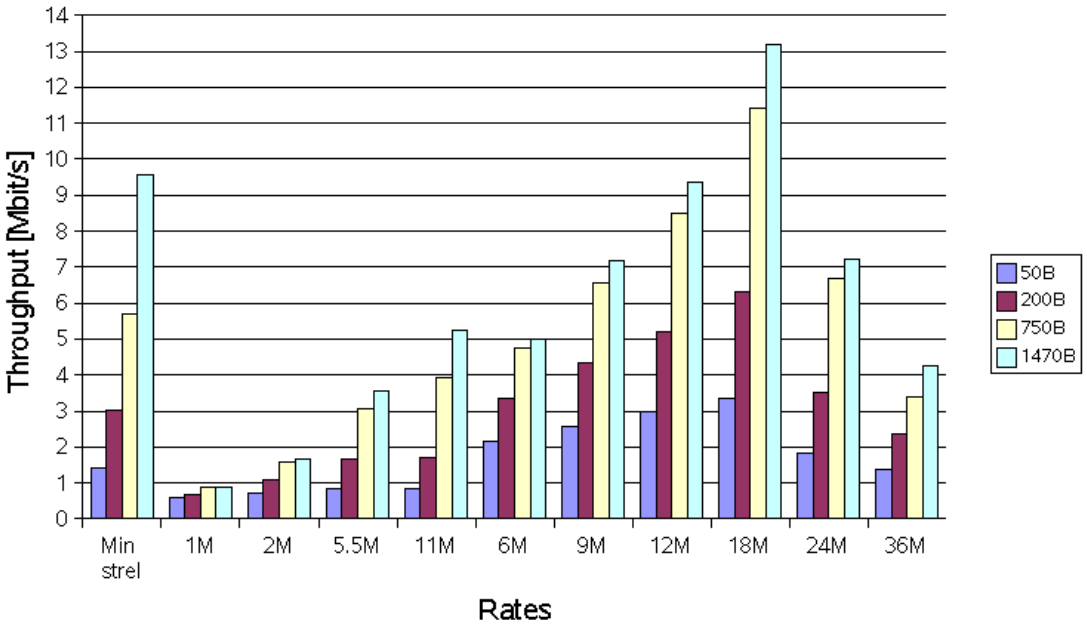

Figure 4.15: The throughput in Mbit/s that was reached in one minute tests under noise jamming using different packet sizes and rates.

setting. It ignores the physical rates 48 Mb/s and 54 Mb/s as these don't function correctly. Sampling and transmitting at these rates only decreases the over all throughput and should therefore be avoided. However, the version we tested prevented the algorithm from sending normal packets at 48 Mb/s and 54 Mb/s but still sampled at these rates. We will refer to it as $\text{Minstrel}_{\text{no48no54}}$.
In the tests, $\text{Minstrel}_{\text{no48no54}}$ did not perform well, most likely it did not make any difference at all as the jammer was strong enough to kill all sample packets sent on the two highest rates. If the sampling on these rates would be disabled as well, then there could be an improvement, as sampling these two rates is useless when they don't work properly. We did not rerun the tests with a corrected version as we assume that in general these rates will work and the algorithm would only be useful, when these rates do not work. The idea to implement it correctly is to just select a new sample rate until it isn't one of the broken rates.
Another idea is to just reduce the total amount of sample packets that are sent. This would increase the throughput when the connection is stable but as well increases the time it takes to switch to another rate. And in the current implementation it may even cause some unwanted rate changes as a few successfully

transmitted sample packets could be enough to change to a higher rate while this rate still suffers significant packet loss. To avoid this, a new throughput estimation should only be calculated after enough sample packets were sent. How high this threshold should be would have to be determined in additional experiments. This version of Minstrel will be referred to as $\text{Minstrel}_{\text{less}}$. To implement this version of Minstrel we reduced the number of samples sent from the default 10% to 1%.
All tests showed that $\text{Minstrel}_{\text{less}}$ does improve the throughput in a static environment. The MTR increased by up to 25%. In figure 4.16 $\text{Minstrel}_{\text{less}}$ is compared to the original version by how many transmission failures occurred on what rate. It is clearly visible that there are a lot less failed transmissions on higher rates where the sampling is done, and this increases the throughput.

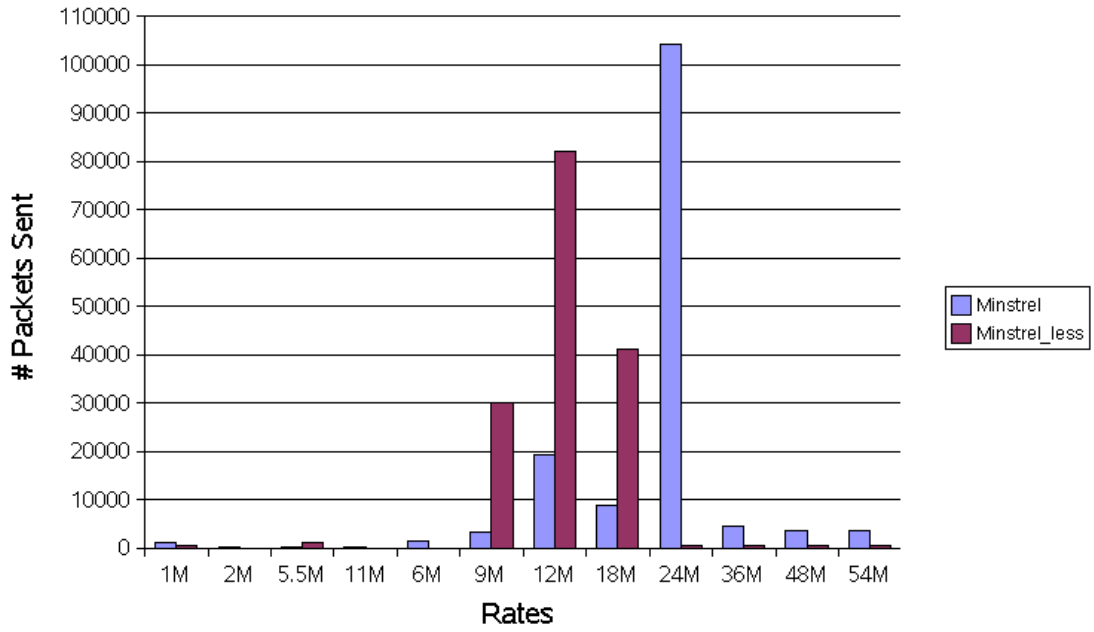

Figure 4.16: Number of packets sent at the different rates using Minstrel and $\text{Minstrel}_{\text{less}}$. That actually less samples are sent can be seen by the low numbers of sent packets at high rates.

A more sophisticated approach, to which we will refer as $\text{Minstrel}_{\text{dyn}}$, would be to set dynamic sample limits on the different rates to decrease the overall samples sent. To avoid that the samples are just sent on a rate with a higher limit one could adjust the sampling rate or increase the sample count even when the limit is reached and no sample was sent. The latter would allow for dynamic adaptation

of the limits without worrying about the overall sample count. How to set these limits optimal would require more tests.
$\text{Minstrel}_{\text{dyn}}$ should allow to send less samples in total but still react fast on changed channel conditions. We therefore set the sample limits for each rate depending on its distance to the current rate[14]. The further away from the current rate the less samples are sent in each interval.

Physical Rate	1	2	5.5	11	6	9	12	18	24	36	48	54
Number	1	2	3	4	5	6	7	8	9	10	11	12

Table 4.3: The ordering of the rates used in Minstrel and for calculating the distance in $\text{Minstrel}_{\text{dyn}}$.

The test using $\text{Minstrel}_{\text{dyn}}$ did not work well. Mainly because the sample limits we used are not working as intended and only on the highest rat there are significant less samples sent. This can be seen in figure 4.17 for the bit jammer test. Instead of the intended 4:2:1:0 ratio the three lower rates still seem to be used nearly the same amount of the time. This and the worse conditions may explain the lower MTR that can be seen in table 4.4.

jammer	Minstrel	$\text{Minstrel}_{\text{dyn}}$
noise	75%	75%
bit	79%	65%
frame	80%	76%

Table 4.4: For Minstrel and $\text{Minstrel}_{\text{dyn}}$, the MTR measured under noise, bit and frame jamming.

4.4.3 Packet Delivery Ratio

Another point that should be discussed is the estimation of the PDR that currently is implemented using an EWMA where the last value accounts for 75%[15]. The PDR of the last interval is calculated using the number of successfully transmitted packets and the number of transmission attempts. The PDR of the new interval

[14]The distance is the difference between the number of the physical rates, the numbers can be found in table 4.3
[15]This value will be denoted as $\text{EWMA}_{\text{Level}}$

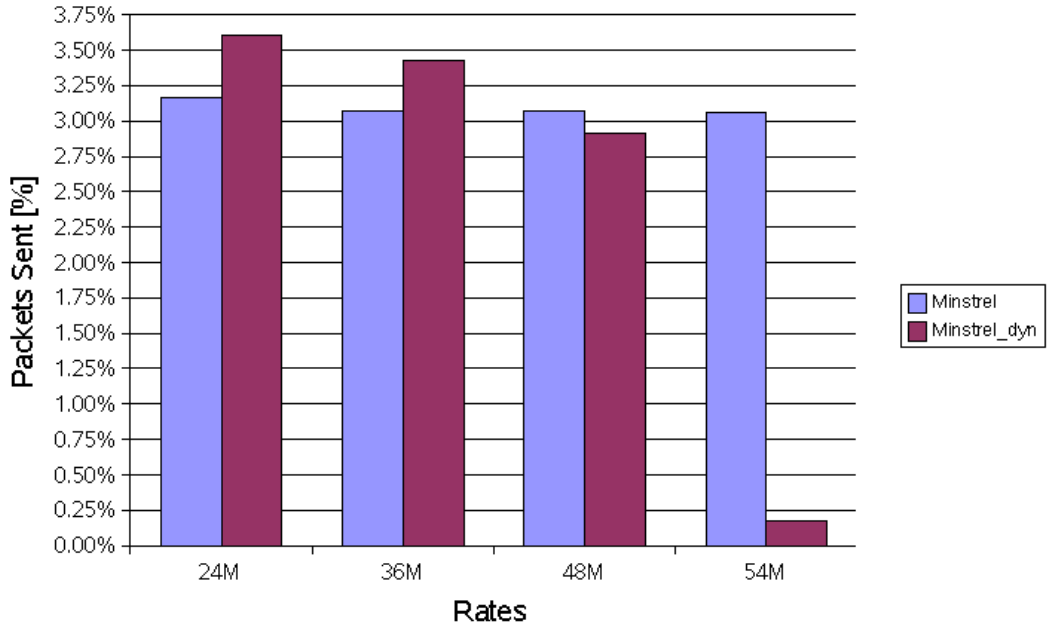

Figure 4.17: The percentage of packets sent at each of the higher rates, most as sample packets, using Minstrel and $\text{Minstrel}_{\text{dyn}}$ under bit jamming.

always accounts for 25% to the new PDR value, no matter if only one or hundreds of packets were sent. Especially with sampling this can lead to bad decisions. As often only few samples are sent in each interval, these samples are weighted to much compared to the regularly sent packets. This leads to a too high estimation for the PDR and the algorithm is likely to switch to this higher rate and then will have to spend some intervals on this wrongly chosen rate until the PDR again dropped to its real value.

The solution that probably comes up first is to collect the samples until a minimal number of tries is reached avoiding that single packets have a too strong influence on the PDR. Another possibility would be to adjust the $\text{EWMA}_{\text{Level}}$ to the number of packets that were transmitted. This would ensure that every packet has an influence on the PDR in the interval it was sent but ensure that this influence is not too strong.

We also considered that in a static environment the PDR changes slower than in a mobile setting. This leads to the conclusion that if mobility could be detected it might be of use to adapt the $\text{EWMA}_{\text{Level}}$ accordingly, lowering it when mobility is detected allowing for faster changes and raising it to ignore short-term variations

when the conditions are stable.
For this we tried to implement a simple mobility detection that monitors the physical rate changes and assumed that frequent rate changes indicate mobility. While this worked well for a non-jammed environment, it completely failed as soon as a jammer was turned on. And as implementing a good mobility detector was not in the scope of this thesis, we dropped this approach and just assumed that we would have such a detector[16] and separated the two scenarios of mobile and static environment for our experiments. This allowed us to simply use an increased $\text{EWMA}_{\text{Level}}$ for the test in the static setting. We call this version $\text{Minstrel}_{\text{high}}$ and used an $\text{EWMA}_{\text{Level}}$ of 90 instead of the default 75.
Another factor that has influence on the reaction time and the accuracy of the estimated PDR is the interval duration. Increasing the duration of an interval will reduce the influence of short-term variations but also decrease the algorithms flexibility. With a mobility detector the interval period could be increased when the machines are stationary and allow for a better estimation of the actual PDR.
The last variation of Minstrel we tested is referred to as $\text{Minstrel}_{\text{pdr}}$. It uses an $\text{EWMA}_{\text{Level}}$ that is dependent on the PDR and has a different value for each rate. It gives new measurements less weight if the PDR is low. We tested $\text{Minstrel}_{\text{high}}$, $\text{Minstrel}_{\text{long}}$, $\text{Minstrel}_{\text{pdr}}$ and the original version of Minstrel under a noise-, bit- and frame-jammer scenario.
Figures 4.18(a) - 4.18(d) show the throughput reached by each of the four version of minstrel under noise jamming. While in this test the higher $\text{EWMA}_{\text{Level}}$ had the best MTR, under the other jammers it did not perform that well. The dynamic PDR seems to be worse than the original version as it has a much lower average MTR. The best algorithm is the one using an interval of 1 second instead the default 100ms. The MTRs for all jammers can be found in table 4.5

4.4.4 Throughput Estimation

Minstrel estimates the throughput on the basis of the estimated PDR and the perfect transmission time. As discussed earlier in this chapter and as shown in table 4.2 the achievable throughput is for the higher rates lower than expected. Taking this into account could help to maximize the throughput in some scenarios.

[16]It could be implemented for example with GPS or any other localization technique.

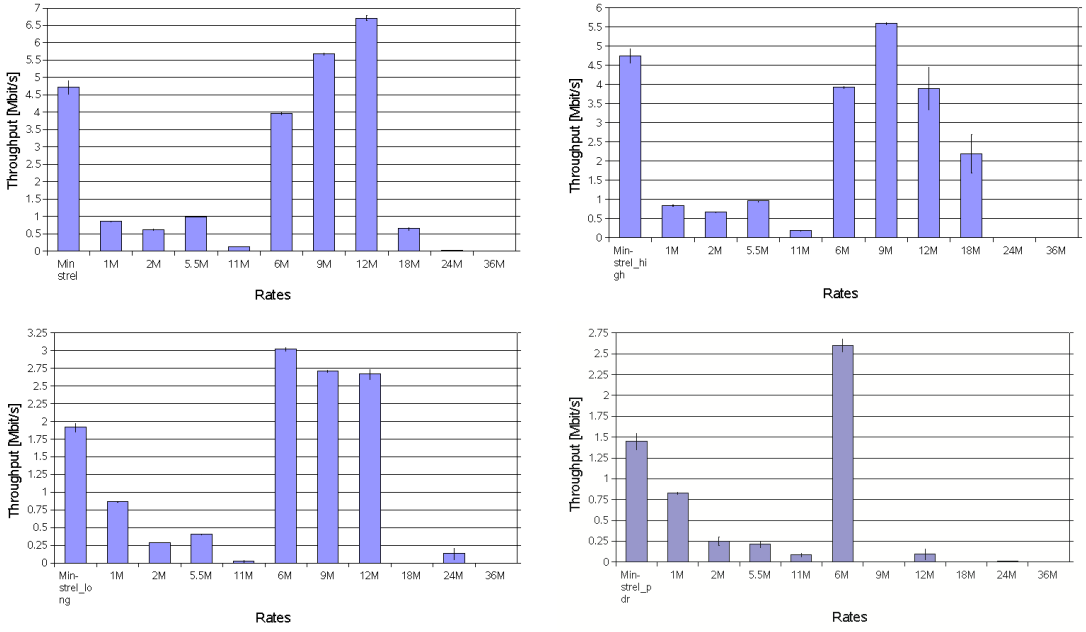

Figure 4.18: Performance of the different rate switching versions of Minstrel under noise jamming. The error bars indicate the 5% confidence interval.

jammer	Minstrel	$\text{Minstrel}_{\text{high}}$	$\text{Minstrel}_{\text{long}}$	$\text{Minstrel}_{\text{dynamic}}$
noise	71%	85%	64%	57%
bit	84%	76%	87%	61%
frame	57%	65%	68%	60%

Table 4.5: The MTR reached under different jammers for the original Minstrel and three variations we implemented. 'high' stands for the higher $\text{EWMA}_{\text{Level}}$, 'long' for the longer interval and 'dynamic' for the dynamically adapted $\text{EWMA}_{\text{Level}}$.

In our setup the two highest physical rates reached an even lower throughput as shown in table 4.6. For this measurement we put the sender and receiver as close to each other as possible to achieve the best possible connection.

R_f	1	2	5.5	11	6	9	12	18	24	36	48	54
R_m	0.9	1.8	4.3	7.0	5.0	7.4	9.5	13.2	16.7	22.3	9.0	6.4

Table 4.6: The measured throughput matrix in Mbit/s for a 802.11g ad-hoc network, where R_f stands for the physical rate and R_m denotes the measured throughput.

4.4.5 History Management

Minstrel adjusts the PDR only when a packet is sent on a given rate and as Minstrel in practice does not send samples on lower rates, their data is often inaccurate. This means that when the conditions get worse Minstrel may drop directly to the lowest rate and it then basically has to sample its way back to the optimal rate as if there would be no data at all.
A simple but inefficient solution would be to just sample lower rates as well, but as sampling lower rates is only useful when the conditions get worse, it would decrease the throughput for most common cases and should not be used. Instead, we could do an estimation of the PDR of lower rates depending on the PDR of the current rate. A simple estimation is to just assume that lower rates will have at least the same PDR as the current rate[17]. This should work quite well within the same modulation, but estimating the PDR of a rate with another modulation might be more problematic. Especially with different jammers the situation can get complex.

4.5 Review on Rate Switching

While previous rate switching approaches would drop to a too low rate under jamming and regenerate slowly during a sleep period of the jammer as shown in [6], we showed that Minstrel does perform well under bit and noise jamming as

[17]This assumption is based on the fact that lower rates have better error correction codes and therefore should be less prone to bit errors.

it often chooses one of the best rates and reached in all our tests at least 50% of the throughput of the best fixed rate. Thanks to its high sampling rate Minstrel recovers within a short amount of time[18]. When we tested Minstrel under a frame jammer it only reached a MTR between 20% and 60%. We found that Minstrel still could be improved under all jammers, while the tests without a jammer showed that it already is close to the optimum in such situations.
We discussed the trade-off between overhead and fast adaptation to changes in the environment regarding our tests. We worked out several ideas to improve the performance under jamming, implemented some of them and evaluated their performance with a series of experiments.
We showed that it is possible to increase the MTR by up to 25% when reducing the number of samples sent. In some scenarios the throughput could be increased as well by extending the interval period or raising the $\mathrm{EWMA}_{\mathrm{Level}}$. If the samples are not distributed equally but adapted to the probability that these rates will become better in the next interval this could again improve the performance while reducing the flexibility just a little. The best distribution of samples is yet to be determined and probably differs whether or not a jammer present.

[18]The EWMA used ensures that it only takes a few intervals of 100ms to switch back to the best rate

4.6 Possible Follow-Ups

In this section we give an outlook on what could be done to further improve the performance of rate switching under jamming.

Sampling Strategies

We showed that sampling less often does improve the throughput in a static setting. Now the next step would be to evaluate different sampling strategies under different jammers and in mobile scenarios. Mobility detection could also give a big performance improvement.

Weight of Samples

We believe that single packets are weighted too much in the current implementation of Minstrel. This may lead to unfortunate decisions and decrease the throughput. It could help to collect samples until a certain number of samples is available to get a better estimation of the PDR. Another approach would be to use a higher $\text{EWMA}_{\text{Level}}$ if there is only little sample data.

Throughput Estimation

Currently Minstrel uses the lossless transmission time and the estimated PDR to calculate the achievable throughput. But for higher rates this throughput calculation is incorrect. It should be looked into this issue to find better ways of calculating the throughput, especially in the case of a frame jammer.

History Management

Another problem Minstrel has is, that the PDR of lower rates often is not updated for a long time and therefore often inaccurate. We believe that the PDR of lower rates can be roughly estimated at least for the same modulation scheme to improve their precision.

5 Conclusion

We've shown that it is possible to detect jamming in real-time and without transmission overhead using only the limited informations that the operation system provides. The real difficulty is the detection of a frame jammer as it attacks not the PDR but the protocol itself. We also discussed different positive points, disadvantages and possible improvements of Minstrel and compared their performance to fixed physical rates.
In addition to that, we were able to show the different requirements in sense of timing for the two topics of rate switching and jamming detection. Rate switching has to be fast and highly adaptive to account for mobile scenarios, while the jamming detection relies on large intervals to calculate the channel characteristics with the highest possible precision.

6 Outlook

We suggest to implement a jamming-aware rate switching algorithm that adjusts its rate selection to the jamming scenario it faces. This would additionally require a differentiation of the attackers. For the sample strategy this could mean that the algorithm keeps track of Minstrel's PDR table and switches between two versions whether there is jamming or not. To make this possible, Minstrel would need to backup its table long enough for the jamming detection to raise the alarm. This would prevent a constant jammer from corruption of the physical rate statistics. Another way that Minstrel could profit from jamming detection is an adaptive $\text{EMWA}_{\text{Level}}$ as it was shown, that a higher value is less responsive to outliers which reduce the throughput due to a suboptimal choice of the physical rate.
Another point of interest would be a new reporting mechanism. Whatever rate switching algorithm is used, it naturally has a lot of informations about the channel situation and characteristics. A small improvement in the precision and only a few more measurements (like the bad packet ratio) would highly increase the possibilities of jamming detection.
We can also imagine to include the jamming detection directly into the rate switching algorithm as all the data is already there and calculation overheads are small compared to the time a wireless card needs to transmit a package. This would greatly increase the performance of the jamming identification as the rate switching algorithm could select exactly those physical rates, the detection part needs more informations about. As an idea to reduce a possible slow-down of the transmission rate due to the calculations, we would suggest to try and store a few multiple retry chains in advance, instead of generating each in real-time.

Bibliography

[1] Markus Schafroth. Jamming detection in wireless ad hoc networks. Master's thesis, ETHZ, March 2009.

[2] Murat Çakiroğlu and Ahmet Turan Özcerit. Jamming detection mechanisms for wireless sensor networks. In *InfoScale '08: Proceedings of the 3rd international conference on Scalable information systems*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[3] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 46–57, New York, NY, USA, 2005. ACM.

[4] Jalel Ben-Othman Ali Hamieh. Detection of jamming attacks in wireless ad hoc networks using error distribution. In *ICC*, pages 1–6, 2009.

[5] M. Strasser. *Novel Techniques for Thwarting Communication Jamming in Wireless Networks*. PhD thesis, ETHZ, 2009.

[6] Konstantinos Pelechrinis, Ioannis Broustis, Srikanth V. Krishnamurthy, and Christos Gkantsidis. A measurement driven, 802.11 anti-jamming system. *CoRR*, abs/0906.3038, 2009.

[7] Lochan Verma, Seongkwan Kim, Sunghyun Choi, and Sung-Ju Lee. Agile rate control for ieee 802.11 networks. In *FGIT '09: Proceedings of the 1st International Conference on Future Generation Information Technology*, pages 237–245, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] Robert T. Morris, John C. Bicket, and John C. Bicket. Bit-rate selection in wireless networks. Technical report, Master's thesis, MIT, 2005.

[9] Poisel Richard. *Modern communications jamming principles and techniques*. Artech House, 2004.

[10] Rohde & Schwarz GmbH & Co. KG. R&s®smu200a vector signal generator operating manual. February 2010.

[11] Kasper Bonne Rasmussen and Srdjan Capkun. Implications of radio fingerprinting on the security of sensor networks. In *PROCEEDINGS OF IEEE SECURECOMM*, 2007.

[12] Srdjan Čapkun, Levente Buttyán, and Jean-Pierre Hubaux. Sector: secure tracking of node encounters in multi-hop wireless networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 21–32, New York, NY, USA, 2003. ACM.

[13] Mathieu Lacage, Mohammad Hossein Manshaei, and Thierry Turletti. Ieee 802.11 rate adaptation: a practical approach. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 126–134, New York, NY, USA, 2004. ACM.

[14] Emrah Bayraktaroglu, Christopher King, Xin Liu, Guevara Noubir, Rajmohan Rajaraman, and Bishal Thapa. On the performance of ieee 802.11 under jamming. In *INFOCOM*, pages 1265–1273. IEEE, 2008.

Appendices

A Additional Measurements

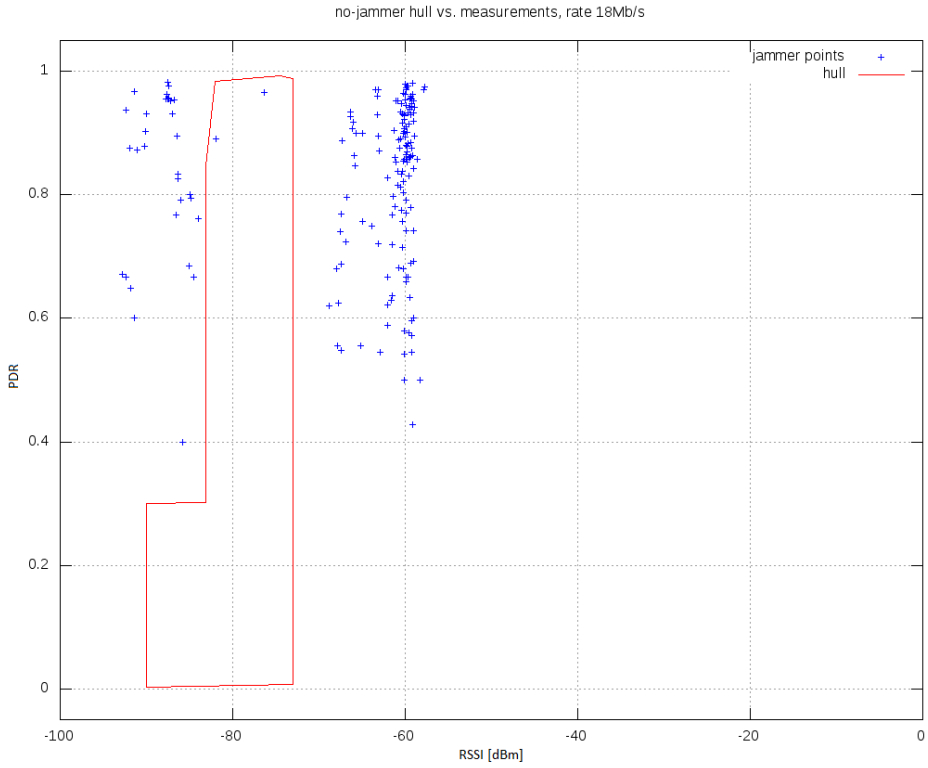

Figure A.1: Results of RSSI jumps because of jamming in comparison to the threshold.

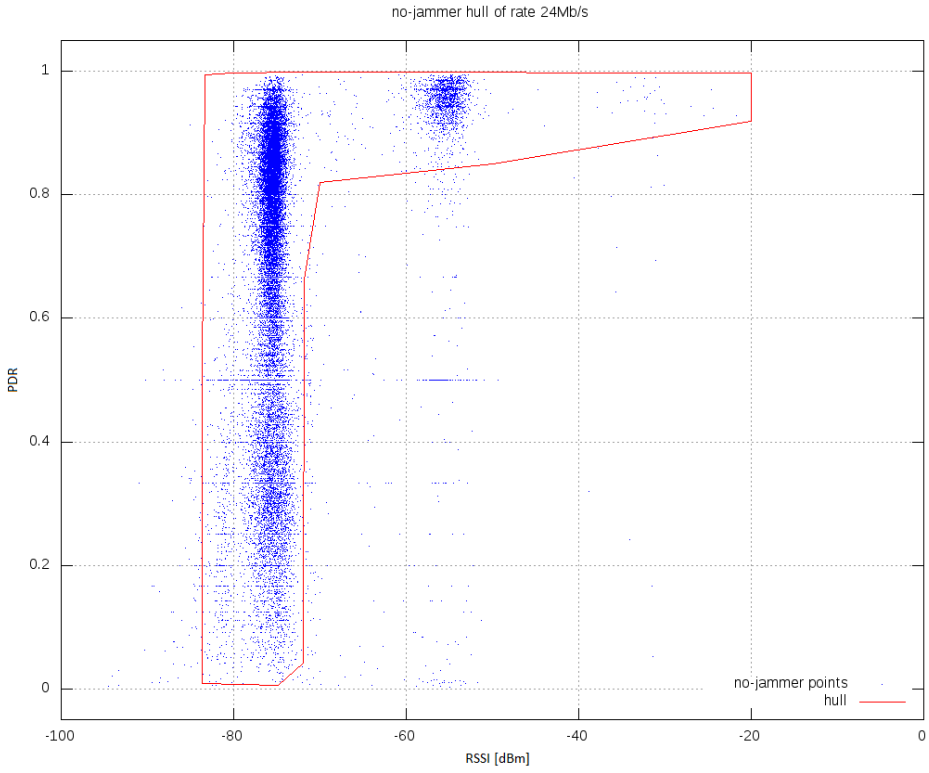

Figure A.2: No-jammer hull of rate 24Mb/s.

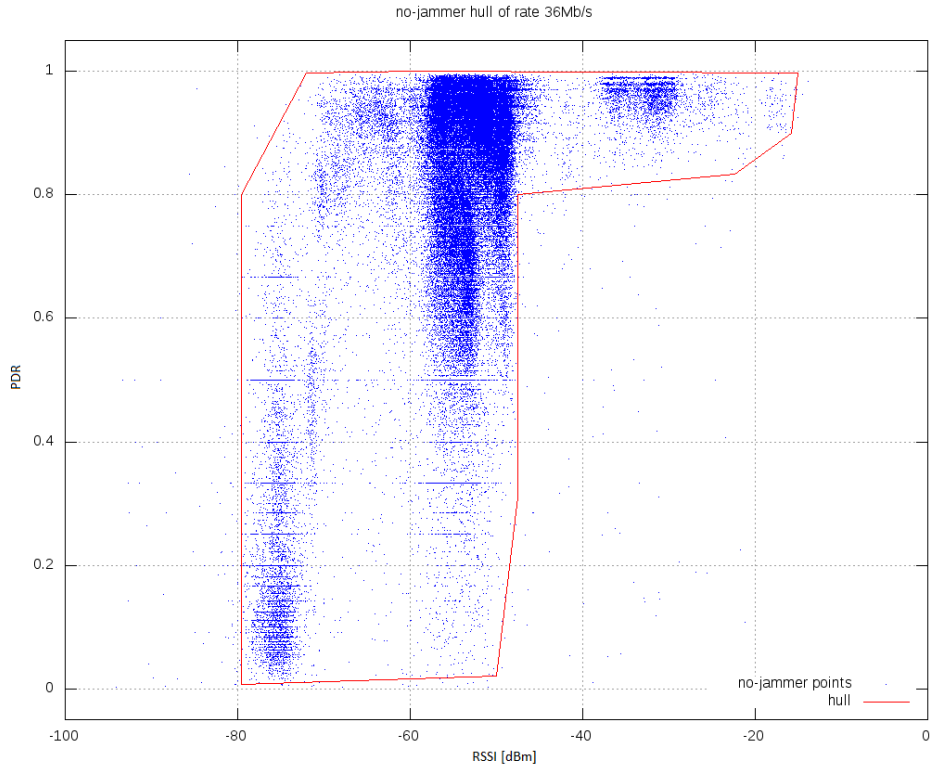

Figure A.3: No-jammer hull of rate 36Mb/s.

B Assignment

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Eidgenössisches Departement für Verteidigung,
Bevölkerungsschutz und Sport VBS

armasuisse
Wissenschaft und Technologie W+T

Master of Science Thesis Assignment
for
Dietmar Schediwie and Nicolas Häfelin (D-INFK, ETH Zürich)

Main advisor:	Dr. Vincent Lenders, armasuisse
Advisor ETH:	Mario Strasser, Prof. Dr. Bernhard Plattner, ETH Zürich
Supervisor:	Prof. Dr. Srdjan Capkun, ETH Zürich

Start date:	October 5th 2009
End date:	April 9th 2010

Jamming-aware Wireless Rate Switching

Introduction

Wireless technologies like IEEE 802.11, Bluetooth, Zigbee, Wimax have gained tremendous popularity in the recent past. A fundamental threat with these technologies is however denial-of-service attacks to the medium access (so called jamming attacks). In contrast to their wired counterparts, wireless networks are highly vulnerable to such attacks as they operate over a shared medium and attackers can simply deny access to the medium by sending malicious radio frequency (RF) signals that interfere with the regular communication.

Jamming attacks affect particularly the performance of wireless networks that adopt a dynamic physical rate switching scheme, as for example IEEE 802.11. Dynamic rate switching algorithms are designed to decrease the physical rate when frame losses occur in order to increase the robustness of the transmissions. This approach is beneficial when the cause of frame losses is due to elevated signal attenuation between the sender and receiver as it increases the overall probability of successful packet reception. However, this performs poorly in the presence of a jammer. Prior studies indicate that the highest throughput is achieved when using higher physical rates.

Goal

The goal of this thesis is to design and implement a dynamic wireless rate switching algorithm for mobile networks that is aware of jamming and remains robust in its presence. The problem can be decomposed in two work packages:

1. Detecting the presence of jamming (Dietmar Schediwie)
2. Selecting the best channel and optimal physical rate that offers the best throughput (Nicolas Häfelin)

A requirement for the grade 5.0 is to implement the detection (D. Schediwie) and selection (N. Häfelin) algorithms on Linux and to show empirically their effectiveness over traditional packet loss or signal strength based algorithms under various environmental conditions and jammer models.

Tasks

The tasks of the thesis are:

1. Review the literature on jamming detection, jamming prevention, and adaptive wireless rate switching algorithms for wireless LANs. (D. Schediwie and N. Häfelin)
2. Analyze the IEEE 802.11 channel selection and wireless rate switching algorithms in Linux (which do not account for jamming) and study in the lab how they behave under different jamming models. (D. Schediwie and N. Häfelin)
3. Develop a jamming detection algorithm that accounts for different jamming models. (D. Schediwie)
4. Develop a jamming-aware wireless rate switching algorithm that remains robust when jamming occurs. (N. Häfelin)
5. Implement your jamming detection algorithm on Linux for IEEE 802.11 radios. (D. Schediwie)
6. Implement your channel selection and wireless rate switching algorithm on Linux for IEEE 802.11 radios. (N. Häfelin)
7. Evaluate the jamming detection and selection algorithms with realistic measurements including mobility and different jamming models and compare the obtained throughput with jamming-agnostic IEEE 802.11 radio stacks (D. Schediwie and N. Häfelin)

Deliverables

- At the end of the second week, a detailed time schedule of the thesis must be given and discussed with the main advisor.
- At half time of the master thesis, a short discussion of 15 minutes with the supervisor and the advisors will take place. The students have to talk about the major aspects of the ongoing work. At this point, the students should already have a preliminary version of the written report, including a table of contents. This preliminary version should be brought along to the short discussion.
- At the end of the thesis, a presentation of 20 minutes must be given at armasuisse and at ETH Zürich. The presentations should give an overview as well as the most important contributions of the work. If possible, a demonstrator should be presented at this time.
- The final report should be written in English but may be written in German. It must contain a summary written in both English and German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Four written copies of the final report must be delivered to the main advisor.

References

[1] Markus Schafroth: Jamming Detection in Wireless Ad Hoc Networks, Master Thesis MA-2008-21, March 2009.

armasuisse
Science and Technology
Command and Control Lab

Dr. Vincent Lenders
Berne, October 16th 2009

C Time Table

Kalenderwoche	Mo	Tätigkeit
41	05.10.09	Literaturüberblick
42	12.10.09	**Literaturüberblick beenden**
43	19.10.09	Testsetup aufbauen
44	26.10.09	**Messung 1. Experimentreihe fertig**
45	02.11.09	Auswerten, Theorie und Alorithmus entwerfen
46	09.11.09	Auswerten, Theorie und Alorithmus entwerfen
47	16.11.09	Implementation
48	23.11.09	Testreihe
49	30.11.09	Ergebnisse Analysieren und Algorithmus verbessern
50	07.12.09	Ergebnisse Analysieren und Algorithmus verbessern
51	14.12.09	Implementierung anpassen
52	21.12.09	Testreihe
53	28.12.09	Bisherige Ergebnisse Auswerten, Gerüst für Arbeit
1	04.01.10	**Mid-Präsentation**
2	11.01.10	Ergebnisse Analysieren und Algorithmus verbessern
3	18.01.10	Implementation
4	25.01.10	Testreihe
5	01.02.10	Ergebnisse Analysieren
6	08.02.10	**Beide Teile Zusammenfügen**
7	15.02.10	Testen, Debuggen
8	22.02.10	Debuggen, Testen, Messen
9	01.03.10	Messung der Endgültigen Ergebnisse, Auswertung
10	08.03.10	**Messungen abschliessen, schreiben anfangen**
11	15.03.10	Schreiben der Arbeit
12	22.03.10	**Schreiben abschliessen**
13	29.03.10	Puffer
14	05.04.10	**Abgabe Report**

Table C.1: Zeitplan Schediwie

Kalenderwoche	Mo	
41	05.10.09	Literaturüberblick
42	12.10.09	**Literaturüberblick beenden**
43	19.10.09	Testsetup aufbauen
44	26.10.09	**Messung 1. Experimentreihe fertig**
45	02.11.09	Auswertung der Ergebnisse
46	09.11.09	Entwickeln eines Algorithmus für Rate Switching under Jamming
47	16.11.09	Implementierung des Algorithmus auf Linux
48	23.11.09	Implementierung des Algorithmus auf Linux
49	30.11.09	Testen der ersten Version unter Jamming
50	07.12.09	Ergebnisse Analysieren und Algorithmus verbessern
51	14.12.09	Implementierung anpassen
52	21.12.09	Neue Version Testen
53	28.12.09	Puffer
1	04.01.10	**Mid-Präsentation**
2	11.01.10	Algorithmus feintunen
3	18.01.10	Neue Version Testen
4	25.01.10	Analyse der Ergebnisse
5	01.02.10	Puffer
6	08.02.10	**Beide Teile Zusammenfügen**
7	15.02.10	Testen der kombinierten Version
8	22.02.10	Analyse der Ergebnisse & feintunen
9	01.03.10	Letzte Testreihen
10	08.03.10	**Messungen abschliessen, schreiben anfangen**
11	15.03.10	Schreiben der Arbeit
12	22.03.10	**Schreiben abschliessen**
13	29.03.10	Puffer
14	05.04.10	**End-Präsentationen, späteste Abgabe Report**

Table C.2: Zeitplan Häfelin