NTNU

Norwegian University of
Science and Technology

Detecting Wireless Identity Spoofs in Urban Settings, Based on Received Signal Strength Measurements

Øystein Aas Pedersen

Master of Science in Communication Technology
Submission date: June 2010
Supervisor: Stig Frode Mjølsnes, ITEM
Co-supervisor: Martin Eian, ITEM
Thomas Jelle, Trådløse Trondheim

Norwegian University of Science and Technology
Department of Telematics

Problem Description

There are several proposed methods to discover attacks in wireless networks. This thesis focus on measuring received signal strength (RSS) values to detect attacks. Current research culminates in the paper written by Chandrasekaran et. al. where the authors solves a newly discovered vulnerability regarding the 802.11e QoS protocol and also proposes a new way of using RSS measurements for localizing wireless nodes. Using these two methods together, they claim to achieve a false positive rate of 0.5%, even when nodes are mobile.

Could the methods proposed be used to decide if an attack is ongoing in an urban environment? What about type 1 and 2 errors in this setting? How dependable are the method(s) in regards to circumvention and active attacks?

This thesis will cover existing RSS detection methods, and how these can be used in open wireless LANs like Trådløse Trondheim. Work will also look into how to implement and qualitatively test these methods in a live setting, by using the lab setup and the SIDS application from previous work.

Assignment given: 15. January 2010
Supervisor: Stig Frode Mjølsnes, ITEM

Abstract

The most common gateway for executing attacks in 802.11 networks are the MAC spoofing attack. Current Todays Wireless IDS implements different methods to detect MAC spoofing, but are particularly interested in using methods that are based on characteristics that are considered unspoofable. One such characteristic is the received signal strength (RSS). Current research are often tested in office environments only, and this work aims to test how the methods work in Wireless Trondheim's urban environment. To research the effects, a wireless sensor network was made. A framework for treating captured data from the sensor network was developed that can be augmented with various detection methods for 802.11 based networks. A RSS detection method has been developed and tested with real test data from an urban environment. A RSS based detection method was tested, and the results depicts the challenges of using such methods in an urban environment. Results also show that existing statistically based RSS methods would work poorly in such environments.

Preface

This master thesis was finalized during the fall semester in 2010. The thesis marks an end to a 5 year study for the master's degree in Communication Technology with specialization in Information Security. The work has been accomplished at the Department of Telematics at Norwegian University of Science and Technology (NTNU).

The background for this thesis was a pre-project done in the fall semester of 2009 by Øystein Aas Pedersen and Eirik Holgernes. Wireless Trondheim was the assignment owner and provided equipment and offices during the making of the thesis.

I would like to thank the staff at Wireless Trondheim for equipment and help in setting up and configure the sensor network.

I would also like to thank the supervisor Martin Eian for invaluable contributions during times of mental drought that helped me continue my work.

This thesis is dedicated to my fiancé Agnete, who has always been there for me.

Trondheim, June 2010

Contents

Abstract **I**

Preface **III**

List of Figures **IX**

List of Tables **XIII**

Abbreviations and acronyms **XV**

1 Introduction **1**
1.1 Motivation 2
1.2 Research Scope 3
1.3 Research Goals and Questions 4
1.4 Contributions 5
1.5 Methodology 6

2 Background **7**
2.1 Wireless LANs 7
2.2 802.11 Radio 8
2.3 Wireless based Intrusion Detection Systems 10
2.3.1 MAC Frame based detection methods 12
2.3.2 802.11 Radio based IDS Detection Methods 13
2.4 802.11 Protocol 13

2.4.1 The 802.11 MAC Frame 14
2.4.2 802.11 Sequence Numbers 17
2.4.3 Physical Layer Metrics 19

3 Related Work **23**
3.1 Fingerprint Detection Methods 24
3.1.1 RSS Based Fingerprinting 25
3.1.2 RF based fingerprinting 27
3.2 Combination of Detection Methods 28

4 Distributed Wireless Capture System **31**
4.1 The Sensor 33
4.2 Kismet-Newcore 34
4.2.1 Kismet-drone 35
4.2.2 Kismet-Server 36
4.2.3 Kismet-client 37
4.3 Architecture Overview 38
4.4 Code Framework 39
4.4.1 Software Specifics 41

5 Experimentation **43**
5.1 Design of Experiments 44
5.2 Execution of Experiments 45
5.3 Data Analysis 47

6 Results **49**
6.1 Control Experiments 50
6.1.1 Control Experiment 1 50
6.1.2 Control Experiment 2 52
6.1.3 Control Experiment 3 (Northug only) 54
6.1.4 Analysis 54
6.2 Attack Experiments 56
6.2.1 Attack Experiment 1 56

6.2.2 Attack Experiment 2 . . . 58
6.2.3 Attack Experiment 3 . . . 60
6.2.4 Attack Experiment 4 . . . 62
6.2.5 Attack Experiment 5 (Northug only) . . . 64
6.2.6 Attack Experiment 6 (Northug only) . . . 65
6.2.7 Analysis . . . 67
6.3 IDS method testing . . . 67
6.3.1 Detection Method on Control Data . . . 68
6.3.2 Detection Method on Attack Experiments . . . 69

7 Discussion **75**
7.1 Measurement Variability in Experiments . . . 75
7.1.1 Distance . . . 76
7.1.2 Environmental Factors . . . 76
7.1.3 IDS Infrastructure . . . 78
7.2 Detection Method Results . . . 78

8 Conclusion **81**
8.1 Future Work . . . 82

Bibliography **83**

A Appendix **89**
A.1 Theory addendum . . . 89
A.1.1 802.11 . . . 89
A.1.2 Per-Packet information (PPI) . . . 91
A.2 Infrastructure . . . 96
A.2.1 Sheevaplug Details . . . 96
A.2.2 Kismet-Newcore Config Files . . . 96
A.2.3 Kismet Server Configuration . . . 96
A.3 Experiments . . . 105
A.3.1 Setup of stations . . . 105
A.3.2 Station setup . . . 105

A.3.3 Attacker setup 106
A.4 Code 106
A.4.1 Framework Code 106
A.4.2 R Code 117

List of Figures

2.1 A simple illustration of a 802.11 network 8

2.2 The MAC frame format [1] 14

2.3 The Frame Control field. 16

2.4 Sequence control field [1] 18

2.5 PPI field as viewed in Wireshark 21

3.1 From [2]. WA is aggregating measurements of the two clients, creating a RSS profile per frame. 25

3.2 Proposed algorithm in [3] to detect identity spoofs in 802.11e-enabled networks. 30

4.1 Placement of the two sensors, where each are placed 20 cm away from the monitored APs 32

4.2 Sheevaplug as advertised on the www.plugcomputer.org, June 2010. 34

4.3 Screenshot of a capture session with Kismet-client. Left side shows the output of the two sensors, while the right side shows the Kismet-client. 38

4.4 Logical architecture of the distributed capture system. 40

6.1 Control experiment 1, sensor Bjorg 51
6.2 Control experiment 1, sensor Northug 51
6.3 Control experiment 2, sensor Bjorg 53
6.4 Control experiment 2, sensor Northug. 53
6.5 Placements of attacker and victim 55
6.6 Control experiment 3 (Northug only) 55
6.7 Attack experiment 1, sensor Bjorg. 57
6.8 Attack experiment 1, sensor Northug. 57
6.9 Attack experiment 2, sensor Bjorg. 59
6.10 Attack experiment 2, sensor Northug. 59
6.11 Attack experiment 3, sensor Bjorg. 61
6.12 Attack experiment 3, sensor Northug. 61
6.13 Attack experiment 4, sensor Bjorg. 63
6.14 Attack experiment 4, sensor Northug. 63
6.15 Attack experiment 5 (Northug only). 64
6.16 Attack experiment 6 (Northug only) 66
6.17 Detection method on control experiment 1 70
6.18 Detection method on control experiment 2 70
6.19 Detection method on control experiment 3 (Northug only) 70
6.20 Detection method on attack experiment 1 71
6.21 Detection method on attack experiment 2 71
6.22 Detection method on attack experiment 3 72
6.23 Detection method on attack experiment 4 72

6.24 Detection method on attack experiment 5 (Northug only) 73
6.25 Detection method on attack experiment 6 (Northug only) 73

A.1 802.11 type and subtype combinations from IEEE 802.11 [1]. . . 90

List of Tables

2.1 To DS / From DS table 17

6.1 Summary table for control experiment 1 and 2. 52

6.2 Summary table for the control experiments 56

6.3 Summary table for attack 1 and 2. 60

6.4 Summary table for attack 3 and 4. 62

6.5 Summary table for attack 5 and 6 with a comparison of the data from the single sensor control experiment. 65

6.6 Experiment summary table 67

Abbreviations and Acronyms

ATK attacking station

AP access point

CA collision avoidance

CRC cyclic redundancy check

CSMA carrier sense multiple access

CTS clear-to-send

CUWN Cisco Unified Wireless Network

DS distribution system

FCS Frame Check Sequence

GMM gaussian mixture model

IDS intrusion detection system

IEEE Institute of Electrical and Electronics Engineers

iid independent and identical distributed

IOS Internetwork Operating System

IPS intrusion prevention system

LAP lightweight access point

LWAPP light weight access point protocol

MAC media access control

MiM Man-in-the-Middle

MSB most significant bit

NAV network allocation vector

NIC network interface

OS operating system

OSI Open System Interconnection Reference Model

PARADIS Passive RAdiometric Device Identification System

Pcap packet capture

PDU protocol data unit

PPI Per-Packet Information

QoS Quality of Service

QoE Quality of Experience

RF radio frequency

RSS received signal strength

RSSI received signal strength indication

RSN Robust Security Network

RTS request-to-send

RTT round-trip-time

SN sequence-number

SIDS simple IDS

STFT short-time Fourier transform

SSFA signal strength Fourier analysis

SSID service set identifier

SSL secure socket layer

STA station

VPN virtual private network

wIDS Wireless Intrusion Detection System

WLAN wireless local area network

WLC Wireless LAN Controller

WMM Wi-Fi mulitmedia

WPA Wi-Fi Protected Access

WUN Wireless Unified Network

WT Wireless Trondheim

1 Introduction

Wireless Trondheim (WT) in Trondheim City is a company owned by NTNU, the local authorities and several other local companies to create an arena for research and development for new wireless and mobile services.

Today WT has installed wireless local area network (WLAN) coverage in the main areas of Trondheim City, free of use for students and employees of Trondheim kommune. Other people can connect as well, by paying a fee, e.g. 10 KR for 3 hours, 30 Kr for 24 hours etc. After payment is verified, the station that was associated with the payment will be given access to use the network wherever WT has coverage. The station can be any WLAN enabled device. What actually happens is that the stations media access control (MAC) address is

added to a whitelist on a captive portal. Any traffic going to the Internet must go through this portal, and stations not in the whitelist will be denied access.

In an open wireless network like WT it is quite easy to circumvent the installed captive portal by pretending to be a legitimate user. An attacker could do this in several ways, but one of the easier methods is to set its own MAC address as an already registered one, thereby *masquerading* as a valid user.

There are several proposed methods to discover such attacks in 802.11 wireless networks. This thesis will focus on detection methods that measure ***received signal strength (RSS)*** values.

1.1 Motivation

Previous work in the pre-project assignment in the fall semester of 2009 by Pedersen and Holgernes [4], together with work by Chandrashekaran et. al. [3] and Gill [5] is the primary motivation for this thesis.

The main task in the pre-project was to research possible solutions to the problem of detecting session hijacking in open 802.11 networks *after the event had happened*. Primarily a system that analyzes capture logs after an incident has happened is preferred in contrast to block users from the network **during use**. As mentioned, WTs network is completely open, and ways of determining if a customer's session has been hijacked for malicious use is important in legal matters. This is especially true in the wake of the EU data retention directive and its soon-to-be decided fate here in Norway. There is much research going into surveillance of computer systems, both wired and wireless, and the EU directive will impose a law on all ISPs, making them register logs of traffic for up to 2 years. Making a system that can trigger alarms by analyzing log files is in this context important both to satisfy the EU directive, but also to avoid blaming an innocent user.

The project was set in a lab environment where the authors had access to a

replica of WTs city wide WLAN, based on the Cisco Unified Wireless Network (CUWN). Interesting results found within the area of MAC sequence number analysis and RSS measurements showed that there are valid methods that can be implemented in solving the initial problem of spoof detection.

Future work was identified to be:

- Remedy the 802.11e sequence number vulnerability (solved in [3]).
- Usage of RSS measurements for attack detection in Wireless Trondheim.
- Centralization of an IDS system within Wireless Trondheim.
- Augment and further develop the simple IDS (SIDS) application detection methods.

This thesis will focus on RSS as a detection method in urban environments, creating a centralized system for capturing live data and analyze afterwards for spoof detection.

My colleague from the pre-project in [4] will address the other two identified future works in his thesis in [6].

1.2 Research Scope

All work will be based on IEEE 802.11 infrastructure networks. The research is especially focused on the usage of RSS in intrusion detection systems with only minor focus on other detection techniques.

In respect to what was done during the project in [4], it was now time to change the laboratory to a real life setting in Trondheim. This meant widening the scope to an urban setting, with all the constraints a wireless operator like WT would face.

I chose to concentrate on one physical area in Trondheim and do my experiments there. All experiments was done outdoors and in the open, i.e. no testing indoors. As a note, Wireless Trondheim are not prioritizing indoor coverage, only at select places in Trondheim [7].

1.3 Research Goals and Questions

The main goal was to use existing RSS based detection methods in academia to test if spoofing of the MAC address could be detected in an urban environment. The goals can be broken down into three sub-problems:

1. Gathering of reliable test data for analysis from an urban environment.
2. Create software that processes the test data for use with IDS detection methods.
3. Analyze test data for evaluation of a RSS detection method.

Gathering test data By utilizing open source software together with the equipment and support given to me by Wireless Trondheim, I created the infrastructure needed to gather experimental data. This infrastructure was placed in an environment that resembles an **urban environment**.

The process of building a sensor network and process captured data is most often done in an office setting with an ubiquitous amount of sensors giving the authors nearly perfect results. To actually implement this in a live setting is another matter, and my goal therefore was to contribute on the validity of the methods proposed and also to some of the practical difficulties of building such a system.

By using an experimental approach, I generated and gathered the test data based on experiences from the pre-project [4] and the work from others.

Create software for test data processing To be able to analyze the test data, I needed software that organized the captured data in such a way that detection methods can be applied to them.

By creating a framework that parses the test data and then logically **process** the capture data for the detection methods was an important part of the work.

Analyze After the test data had been generated they were used with the developed software for analysis. In the analysis my goal was to identify some of the key aspects of using RSS detection methods in an urban environment and to shed some light on the usability of such methods.

1.4 Contributions

By using open source software and Sheevaplugs, a distributed sensor system was set up in WTs production environment in Trondheim. The system was used to generate test data from an urban environment with the purpose of being fed to a RSS based detection method. Testing of this method with measured effects is considered a contribution.

Creation of a framework that can process the resulting capture log files from the sensor system, that can be augmented with other detection methods in the future, not just based on RSS measurements. This can be used in future work for building a more complete IDS program based on even more detection methods.

Previous work on RSS detection methods have not considered how RSS measurements behave in urban environments. Measurements and analysis of how RSS values actually behave in the urban setting are considered a contribution.

1.5 Methodology

My methodological approach was to test and research how earlier work done in an office or cubical setting will work in the urban setting of Wireless Trondheims infrastructure.

Below is the described workflow during this thesis.

1. **Reviewing existing** literature.
2. **Design and analyze experiments** by setting up a basis for testing in Trondheim City
3. **Implement and test** resulting data for IDS purposes.
4. **Analyze IDS results** from IDS testing
5. **Conclude** what is usable as testing metrics regarding received signal strength (RSS)

First I set up the testing environment with sensors and a central server for processing. Then I used the network to generate test data for my IDS detection program based on RSS.

All changes to the detection program was based on experiences from the tests, and the program was changed in an iterative manner. Typically the points 2 through 4 is based on an iterative approach. The iterative approach is preferable because of the many uncertainties when working in an uncontrolled experiment environment.

Conclusions were then based on analysis of the resulting data.

2

Background

2.1 Wireless LANs

Wireless LANs got introduced in the 802.11 standard in 1997 and has ever since been a popular method for wireless communication.

A basic WLAN consist of several mobile stations (STAs) communicating with an access point (AP). The AP handles all communication between the stations, it also handles all communication the station has with the Internet. The system that forwards traffic from the AP and to the network is called a distribution system (DS). See fig.2.1 for an illustration.

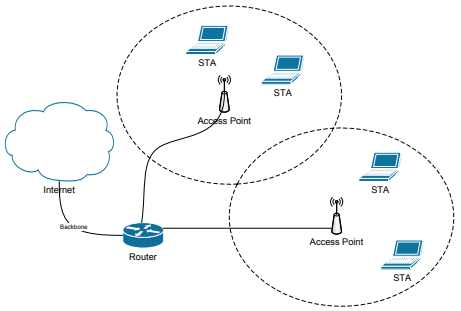


Figure 2.1: A simple illustration of a 802.11 network

Since STAs must use a predefined frequency for communication in a shared medium, only one STA can use the airwaves at the same time. Without proper handling of when the stations are transmitting, collisions would occur and no data can be sent.

To manage this, the 802.11 standard defines a physical layer sensing mechanism called carrier sense multiple access with collision avoidance (CSMA/CA). A MAC-based virtual sensing mechanism called RTS-CTS methods is also used, but are optional and not always turned on in WLANs. Further studies of RTS-CTS can be found in the pre-report [4], as it is not considered further in this thesis.

2.2 802.11 Radio

All 802.11 radio receivers have an ability to deduce the power level of a received signal. This measurement of the power signal is called *received signal strength indication (RSSI)*.

IEEE 802.11 [1, p. 489] defines RSSI to be:

> 14.2.3.2 RXVECTOR RSSI

> The RSSI is an optional parameter that has a value of 0 through RSSI Max. This parameter is a measure by the PHY of the energy observed at the antenna used to receive the current PPDU. RSSI shall be measured between the beginning of the SFD and the end of the PLCP HEC. RSSI is intended to be used in a relative manner. Absolute accuracy of the RSSI reading is not specified.

The RSSI is an integer from 0 - 255, or 0 - RSSI_Max. Each manufacturer of network interface (NIC)s implements their own granularity of this integer, hence the accuracy (as stated in the last sentence) is not specified. Therefore, when using 802.11 wireless sensors, it is important to use the same type of sensors or in some other way correct the differences caused by differing implementation between manufacturers.

The parameter is normally used in station roaming and to set a lowest limit for what the NIC can handle regarding reception of frames. It is measured in decibel per milliwatts or the said RSSI between 0 and 255.

To avoid confusion, in this work all RSS readings are in dBm, and I will use the acronym RSS throughout.

Perceived power at a station is the product of factors like the stations transmission power, distance between station-AP and the environment that causes multi-path distortion and absorption effects.

Multi-path distortion When a RF signal is propagating through the air, it will encounter objects that reflect, refract[1] or diffract the signal, creating multiple wavefronts that reach the receiver at different points in time. This creates among other things increased (or decreased) signal amplitudes [8] and be considered noise from the receivers point of view.

[1] Refraction is the change in direction of a wave due to a change in its speed.

Antenna Diversity Is a technique used to remedy multi-path distortion issues. Antenna diversity helps overcome this problem by using two antennas spaced at multiples of one wavelength apart (12.5 cm for 2.4GHz). This gives the receiver better signaling conditions when receiving, but can also make it **send** frames on either of the two antennas. The most notable element that has an effect on transmission power is the distance. The power fades inversely as the square of the distance [9].

$P = \frac{1}{m^2}$

A short explanation of how the 802.11 standard organizes its radio spectrum.

802.11 defines three versions of the standard; 802.11a, 802.11b and 802.11g.

- 802.11a and 802.11b was an amendment in 1999, introducing physical layer changes to the standard.
- 802.11a operates in the 5GHz area with a maximum bitrate of 54Mbps.
- 802.11b operates in the 2.4GHz area with a maximum bitrate of 11Mbps.
- 802.11g operates in the 2.4GHz area with a maximum bitrate of 54Mbps.

Each channel in 802.11b/g is 22MHz wide, making room for 14 overlapping channels. Only one channel in the standard can be used at any one time, or else traffic would be impeded by frames being distorted by the overlapping frequencies. Only channels 1, 6 and 11 are non-overlapping, making them ideal for use in the same coverage area.

2.3 Wireless based Intrusion Detection Systems

Intrusion detection systems are used to monitor networks and detect activity that are considered illegal or hurtful for the network. Hurtful activities can be be detected by different *detection methods*. Each method is associated to a

specific layer in the OSI model, and can be combined with other methods for better accuracy of detection.

Compared to wired-based IDSs that concentrate on the OSI layers of 3 and up, the 802.11 Wireless Intrusion Detection System (wIDS)s are focusing on PHY layer and MAC layers. By exploiting the information available on these layers, several detection methods are available. In this thesis, I will consider 802.11 based intrusion detection system (IDS) only, i.e. only layers 1 and 2. Specifically I will focus on layer 1 detection using RSS values.

A Wireless Intrusion Detection System (wIDS) uses wireless sensors that are placed togheter with the wireless network it is supposed to monitor. The placement of these sensors must be strategical, to cover the WLAN but also such that it can overlap in certain areas with other sensors. Important for a wIDS is that it's sensors must be sensitive enough to be in range to cover the wanted area for capture. It is also important to capture as many frames as possible to for the detection method.

As described in section 2.2, 802.11 uses different channels to communicate. A wIDS must either be set to monitor a specific channel or use a hopping algorithm, enabling it to monitor as efficiently as possible to avoid missing to many frames. The work of U. Desphande [10] proposes such an algorithm.

Metrics for measuring how an IDS operates in terms of reliability and robustness are measured in ***false positives and false negatives***.

- An IDS that triggers a false positive is detecting an attack when there exists none.
- An IDS that triggers a false negative fails to detect an attack when there are an attack present.

Current IDSs can be divided into to camps; Misuse-based and anomaly based [5].

Misuse based IDSs uses signatures or patterns to detect an attack. Each signature explicitly defines the attack, such that if it lacks a signature for a (potentially) new attack, it will generate false negatives when failing to detect it. On the other hand, it will generate few false positives.

Anomaly-based IDSs do not require signatures, but monitors behavior and raises alarm if it registers any deviation from the normal. Expected behavior is the metric of detection, where deviance from the normal raises an alarm.

Gill [5] defines anomaly-based IDSs in two ways; a statistical model or a specification model.

- A statistical model uses variables and characteristics measured over time to develop the normal behavior of a system. A threshold is found by a training phase of the system, which if exceeded, raises an alarm.
- Specification-based IDSs (also by smith [11]) *specify* the correct behavior of a system, and any deviance from the expected correct behavior raises an alarm.

A more comprehensive discussion of wIDS can be found in [5].

2.3.1 MAC Frame based detection methods

Detection based on analyzing the MAC frame utilizes the different fields within the frame to be able to detect attacks.

Regular MAC sequence-number (SN) analysis comprises of methods that utilizes the MAC sequence-number in data and management frames. This number is presumed to be monotonic increasing, adding a number for each new outgoing frame. This expected linear behavior of the numbers can be used to detect when another attacking station sends frames that have SN that deviates from the expected numbers. If the numbers deviate too much, a spoofing alert can be raised.

As later work has identified [3, 4], 802.11e enabled networks uses a different MAC SN per QoS channel. This creates situations where regular detection of MAC sequence numbers creates more false positives than an non-802.11e enabled network. Any detection method based on sequence numbers must take the QoS enabled APs into account.

2.3.2 802.11 Radio based IDS Detection Methods

Using RSS values in a detection context is useful because of their relative unspoofability. It is also directly dependent on the senders *location*, since any change in location will alter the environment, and also affect the signal power.

Using RSS values also has its drawbacks. The radio environment the signal has to propagate through will affect how the receiver perceives the RSS.

Any radio signal at the frequency band 2.4Ghz is easily absorbed by water or bounced off of concrete. In an urban setting with a lot of people, moving vehicles and buildings, the radio environment is intuitively changing all the time, causing a receiving station to see more fluctuations of the signal power than normal.

Analyzing RSS values therefore comes with a preconceived notion of not being a very reliable detection metric when the station sending is on the move, or is stationary but in a constant changing environment. It will generate fluctuations, shaped by the state of the environment at the time. Therefore many techniques rely on statistical methods or increase the number of sensors to improve accuracy.

2.4 802.11 Protocol

802.11 WLANs is a member of the 802 family of standards defined by the IEEE. All 802 standards focus on the two lowest layers of the OSI model, namely the Physical PHY layer and the Datalink layers. The PHY layer defines details of

transmission and reception of signals, and the datalink layer how the medium is accessed and how the data is sent.

2.4.1 The 802.11 MAC Frame

This section gives an overview of the MAC frame structure as defined in the IEEE 802.11 standard [1].

The 802.11 MAC frame consist of three parts; **Header**, **Frame Body** and **Frame Check Sequence**. The header contains information about e.g. what kind of frame it is, and which address the frame should be sent to. The Frame Body contains the user payload, plus any encryption specific data. The Frame Check Sequence (FCS) is a cyclic redundancy check (CRC) error field, used for the correct delivery of the frame.

WLANs are governed by three different message types that are used for communication:

- **Control frames** - are used for control of the communication between stations,
- **Management frames** - these are frames used to handle relationship between a station and a AP.
- **Data frames** - Payload data from the upper layers. Can be QoS data or non-QoS data.

The MAC frame is shown in fig.2.2.

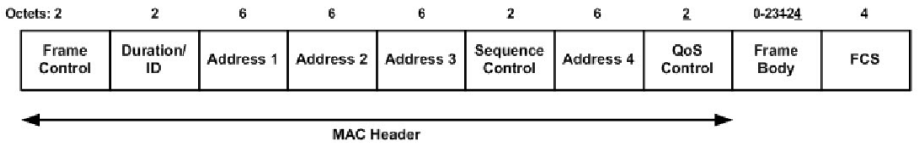


Figure 2.2: The MAC frame format [1]

The MAC frame fields:

- **Frame control** - Contains frame type information and other control information.
- **Duration/ID** - Can be a connection identifier in some Control frames, and a time duration field in microseconds.
- **Addresses 1-4** - Context dependent addresses that consist of the hardware MAC addresses that reside in every link node in the wireless network. Every MAC address is 6 bytes long, and is described by 12 hexadecimal characters, separated with a colon [1].
- **Sequence control** - 12-bits counter and a 4-bit fragment number. Is used to number the frames transmitted between a given transmitter and receiver. Also contains a 4-bit fragmentation field. Only present in Management and Data frames,
- **QoS control** - 16-bits. Identifies various QoS related information. Identifies if it is a QoS frame or a non-QoS frame
- **Frame body** - Frame body is dependent on what type of frame it is (Control, Management or Data).
- **Frame Check Sequence** - CRC check for the frame

In size, the *Header* is between 14 and 36 bytes, and the *Frame Body* between 0 and 2324 bytes. The total size of the 802.11 MAC frame ranges between minimum 14 bytes and maximum 2360 bytes[2]. To be able to interpret the values in the fields correctly, IEEE [1, section 7.1.1] defines reading the bits in the fields in increasing order of significance, from lowest bit to highest numbered bit, i.e. little-endian.

[2] 32 bytes *Header* + 2324 bytes *Frame Body* + 4 bytes *FCS* = 2360 bytes.

The Frame Control field which is 2 bytes resides in the MAC header shown in fig.2.3.

B0 B1	B2 B3	B4 B7	B8	B9	B10	B11	B12	B13	B14	B15
Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgt	More Data	Protected Frame	Order
Bits : 2	2	4	1	1	1	1	1	1	1	1

Figure 2.3: The Frame Control field.

The Frame control field is shown below. Fields B8 - B15 are 1 bit each.

- **Protocol version** - 2-bits. Currently always 00.
- **Type** - 2-bits. Frame type (Control, Management or Data). See the appendix A.1.1 for table over valid types and subtypes for each frame.
- **Subtype** - 4-bits. Various types of the given frame, dependent of frame.
- **To DS / From DS** - 2 bit fields that describes the direction of the frame. This field dictates the address fields and what is in them. See table 2.1 for possible variations.
- **More Frag** - Indicates if the frame is a fragment frame.
- **Retry** - set to 1 if the frame is a retransmit frame and 0 otherwise.
- **Pwr Mgt** - Indicates power management mode of sending station - 1 is station will be in power save mode, 0 in active mode. When station are in power save mode, the AP will buffer any frames destined to it.
- **More Data** - used with power save modes, where a 1 indicates that the AP still has buffered data to the station.

To DS	From DS	Interpretation	SA Address in
0	0	Management and Control frames always set to 00.	Address 2
1	0	To DS	Address 2
0	1	From DS	Address 3
1	1	Used in AP to AP communication.	Address 4

Table 2.1: Combinations of To/From DS flags in MAC control frame [1]. The last column s which address field the Source address is found, for each combinations of the fields.

- **Protected Frame** - 1 if Frame Body has been processed by a cryptographic algorithm. Only for Data frames, and Management frames of subtype Authentication.

Addressing is based on what the To DS / From DS fields in the frame control is set to. Four different address types are defined:

- TA - Transmitter or transmitting address
- SA - Source address
- RA - Receiving address
- DA - Destination address

2.4.2 802.11 Sequence Numbers

The **Sequence Control** field is a mechanism for dependable delivery of MAC frames between two link points. All frames need to be acknowledged by the recipient to avoid losing frames. The sender keeps track of which frame with which sequence number that has been successfully acknowledged by the recipient. If there is an offset in the sequence number regarding acknowledged frames,

i.e. a frame with a higher number than +1 from the previous, the client will retransmit the lost frame.

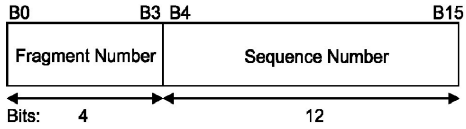


Figure 2.4: Sequence control field [1]

The sequence number field as shown in fig.2.2 comprises of two bytes. As fig.2.4 shows, the field starts with a 4 bit Fragment Number, followed by 12 bits of Sequence Number. The sequence number itself, ranges from 0 to 4095. This means that after the counter reaches 4095, it will wrap around and start again at 0, i.e. the counter is modulo 4096.

802.11e Sequence numbers

The 802.11e was approved in the the 802.11-2007 standard. The protocol defines a set of rules to be used to provide Quality of Service (QoS) on the link layer, similar to the regular sequence number. There can be potentially 8 different 802.11e classes, or TID (Type ID), as defined in [1, chapter 7.1.3.5 "'QoS Control field"'].

In that respect, 802.11e introduced a separate sequence number counter for each QoS class.

For example a 802.11e enabled station registered with an 802.11e compliant AP can potentially keep track of 8 different QoS sequence numbers, that also the AP must keep track of. The QoS data frames uses the same sequence control field to indicate sequence numbers, as the non-QoS frames.

2.4.3 Physical Layer Metrics

Current 802.11 standard do not support storing information pertaining to the physical layer, such as RSS values, channel frequency, bit rate etc. per frame. Information like these are available for the wireless interface, but only per frame received and only for the driver. This means that if one where to capture this information one must in some way add the information to the frame in-transit, and per-frame.

Current technology for achieving this exists, and the most known is the Radiotap [12] standard. Radiotap support a variable header length that is both more effective and economical (in terms of overhead in the frame) but also makes it more robust for adding or removing support of what kind of information one wants.

Current implementation of Radiotap exists on most operative systems, and is a widely adopted standard for frame reception and injection [3].

However, in this thesis I will use another similar standard called Per-Packet Information (PPI).

Per-Packet Information Header

Largely because the packet sniffing program Kismet supports the PPI header it was necessary to use this instead of the Radiotap header (I will come back to Kismet in later sections).

PPI is developed by CACE Technologies, which claims to be "sponsors and the innovative force behind Wireshark and WinPcap" [13]. Wireshark is perhaps the most known dissection tool for network professionals and academics world wide, and WinPcap is a port of the mentioned libpcap for *NIX to Windows.

The PPI header is a meta-information header or pseudoheader that is prefixed

[3]The famous Aircrack-ng suite uses this format.

to each protocol data unit (PDU) captured by a wireless interface. The purpose is to add information available to the wireless interface at the time of frame capture, such as 802.11 radio information. As mentioned, this information is only available at live-capture time and hence must be added per-frame.

Note that what is actually available in the PPI header depends of the driver of the wireless interface. Also note that PPI supports the newly standardized 802.11n protocol as well, making it ready for future use.

Important Fields in PPI

To get the data I need for this thesis, I must capture the RSS measurements for every packet. As mentioned PPI supports this by adding an extra header prefixed in the beginning of the frame. Using the PPI specification [14] the RSS measurement was found.

In the beginning of each PPI header, there is an additional header describing the contents of the header. This is **version**, **indicator flags**, **PPI header length** (between 4 and 65532) and **data link type**.

The version is currently always 0. The Indicator flag is a 8 bit mask that defines the behavior of the header. The header length including packet header and fields. The data link type defines a valid libpcap data type.

After the header comes a PPI Field Structure that defines the first of 6 field types. Field types in PPI can be viewed in appendix A.1.2. The one I am interested in is the **802.11-Common field** that contains common (pre-n and .11n) radio information.

In the 802.11-Common fields of the PPI specification (loosely based on Radiotap header format) there are many fields that could be interesting in a IDS context, but the one I am focusing on is the `dBm-Antsignal` field. The 802.11-common field is 20 bytes long, which gives the bytes on byte placement 31 as the one I need. A verification using Wireshark shows that this is correct, see fig.2.5.

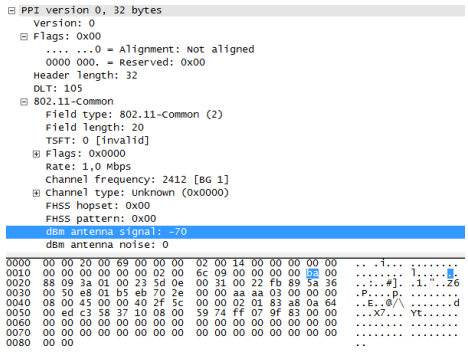


Figure 2.5: PPI field as viewed in Wireshark

3

Related Work

In the area of wireless intrusion detection systems in 802.11 networks, there exists several detection methods, each based on a specific layer in the standard. There are the ones based of the PHY layer, and the ones of the MAC datalink layer.

PHY layer techniques uses characteristics of the radio waves. Some techniques uses specialized monitoring equipment to measure different aspects of the signal. The more known received signal strength metric is popular in current academia because of not needing specialized hardware to measure signal amplitude. Almost all wireless 802.11 interfaces has the ability to measure RSS.

On the MAC layer the values inside MAC frames are utilized in different ways

to detect anomalies and trigger alarms. Typically the MAC sequence detection analysis (also for QoS enabled networks) is mostly used, but also behavioral or specification based methods like analyzing and counting certain types of frames are also promising.

The most promising and up to-date work of a complete intrusion detection system is (in my opinion) of Gill and Smith [5, 11]. By combining several detection methods from both PHY layer and MAC layer, and use these methods in a state based detection mechanism, Gill creates a robust IDS.

Work from Faria [2], Chen [15] and Sheng [16] uses RSS based methods only, but get good results in their experiments. However they only do their tests in office environments.

Chandrashekaran et.al. [3] utilizes MAC sequence number based analysis on both non-QoS frames and QoS frames, together with RSS measurements for detecting mobile attacks. I have not seen any other related work that uses RSS values for detecting attacks while stations are mobile.

As in Gill [5], the methods I am focusing on is the ones that mitigate or help mitigate the session hijacking and/or identity spoofing attacks. This is not only because of the Wireless Trondheim assignment, but also because most of the detection methods involve detecting MAC spoofing. This means primarily methods that detect if one or more stations are using the same MAC address i.e. MAC spoofing attacks.

3.1 Fingerprint Detection Methods

The concept of a fingerprint is intuitively something that is unique. In a 802.11 radio-based context, this means finding a fingerprint for a wireless station for the purpose of uniquely identifying it. These methods uses the characteristics of the radio frequency (RF) signals, and in 802.11, the measured RSSI values.

3.1.1 RSS Based Fingerprinting

Faria et.al. [2] proposed a method where they use the RSS readings from the same MAC at different sensors to create a fingerprint of the station, see fig.3.1

The idea is that any significant deviation from this fingerprint can be used as an indication of attack. The server aggregates all captured frames from the sensors, and creates signalprints as viewed in the **table** in fig.3.1.

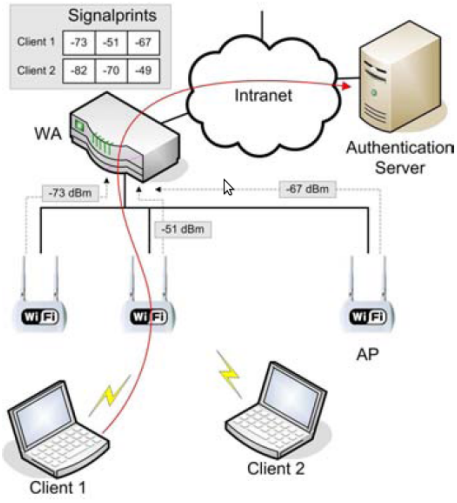


Figure 3.1: From [2]. WA is aggregating measurements of the two clients, creating a RSS profile per frame.

The work showed that more measurements of a MAC address from several sensors will strengthen the signalprint.

They use a trick to remedy the effect of transmission power fluctuation at one station (which is benign) by calculating *differential values* from the sensors. Differential value of a MAC address is found by finding the median RSS value at one sensor, and compute the absolute value between the maximum median sensed by all sensors.

They claim that a client rarely vary it's RSS value by more than 10 or 15 dB [2, p. 46]. If the median RSS values measured by at least one sensor differ by 10dB or more, the system sees two stations, thereby prompting an alarm.

Chen et.al. [15] created a system that both detects spoofing attacks and localizes the attacker. They assume that the registered RSS values are **Gaussian distributed** from a partially synthetic data set. As with Faria, Chen aggregates sensory information to a central server which do the calculations. If there are N sensors, the measurements from these sensors make up a N-dimensional vector of frame captures (not unlike Farias work). Using a K-means algorithm to cluster the readings, ideally each physical transmitter should represent a cluster. This means that the clusters can represent physical locations, where the locations of two nodes will be distinct in physical space in the terms of which clusters they belong to. The detection method then becomes the task of calculate the distance between these clusters. This can be done with an Euclidean distance algorithm. A distance of 5 dB from the centroid was proposed as the limit for issuing an alarm.

What the research do not answer is how it behaves in terms of mobile nodes, how this will work with a dataset from a real setting that is not synthetic, and what likely exploitations/attacks the system is vulnerable to.

Sheng et.al. [16] discovered that the single-Gaussian assumption taken by Chen was wrong. They found that a single station could produce two distinct values of the RSS mean, causing it to fluctuate more than 5dB in 20% of the cases, or 10 dB in 4% of cases, i.e. producing two different Gaussian distributions. This would introduce false positives, as the other methods did not take this into consideration. The reason for the dual Gaussian distribution was attributed to the *antenna diversity* feature that newer 802.11b/g/n NICS use. Please see page 9 for an explanation of the antenna diversity.

The detection technique from Sheng et.al. is based on first making a profile of each station/sensor pair. This profile is then used to detect attacks. They further extend this to be used with several sensors, aggregating the RSS mea-

surements for more robust detection.

Sheng compared the algorithms from Faria and Chen with their own. The gaussian mixture model (GMM)-based global method detected 98% of attacks, Faria's method detected 70% and Chen less than 65%. The results show that there is much to gain when considering multiple Gaussian distributions, however, how often the antenna diversity was used during the experiment, and if it is representable for normal use could be discussed.

The crux of their algorithm is that it depends heavily on stable RSS readings that do not violate the distribution of the measured RSS signals. This also becomes clear from the fact that they have not considered mobile stations, as this would create large fluctuations in time, violating the assumption as well.

The work from Franklin et.al. [17] is based on the unique signature a wireless device emits when it is sending 802.11 probe frames. By capturing the probes and creating a signature through a supervised Bayesian classifier, the technique creates a fingerprint of the device. The number of probes required are about 11, which they empirically got around one minute

The motivation behind their work is that attackers can launch a driver-specific exploit by identifying the driver used by the victim. However, this method is also usable for spoof detection where one can create a profile list of authenticated stations, and see if the profile changes from the original during use.

Unfortunately this method is easily bypassed where an attacker configures the probe sending intensity, or executes a MAC masquerading attack.

3.1.2 RF based fingerprinting

The work from Brik et.al. [18] suggests a fingerprint method is based on the fact that every manufactured wireless network interface (NIC) of the same model is inherently different, even when made from the same manufacturer and with the same components. This is because all such devices are build according to the

802.11 standard, which allows variances within the components, as long as it is within a certain threshold [18]. The variances in these components differ from NIC to NIC, creating the difference that is revealed in the proposed method called Passive RAdiometric Device Identification System (PARADIS)

By sampling on a per-frame basis, a series of metrics is extracted. Frequency, magnitude and phase errors, in-phase and quadrature (I/Q) errors and SYNC correlation are used to create the fingerprint [18]. Matching this value against a prerecorded fingerprint of the NIC, one can tell if the NIC is who he says he is.

According to the authors,

> *... the PARADIS is capable of distinguish between more than 130 identically manufactured 802.11 NICs.*

This method do not use standard 802.11 receivers but specialized variants of 802.11 sensors, thus making it harder to test this method in a live setting. Because of this, it will not be discussed further.

3.2 Combination of Detection Methods

The authors of [3] have published a method to detect MAC spoofing in a reliable and robust manner with a false positive rate of 0.5%. They are also one of the few that have published results concerning mobile spoofing. All previous work up until now have only considered static stations.

By keeping track of the sequence numbers for both non-QoS and QoS frames, they proposed an algorithm where, if sequence number detection proved to be inconclusive, they invoked a costlier method of determining the physical location of an attacker and a victim. This costlier method uses RSS values from each captured frame, feeds it to a localization infrastructure that can localize where

the frame originated from, (i.e. a GPS location of the sending station) and that way detect attacks.

The interesting part (from an RSS point of view) lies in the way they decide via RSS that a MAC has been spoofed. Basically using the same method as Faria, aggregating several RSS measurements from the same MAC address into a unique fingerprint - they also utilize a localization system with the same observed RSS values to find the physical location for where the packet originated. Using an euclidean distance metering algorithm, and observing the change rate of the positions, they can with a high degree of certainty decide if an attack has happened or not, even when the station is mobile. The threshold for speed (i.e. change rate) between measurement points are then used to detect if a station is mobile or not.

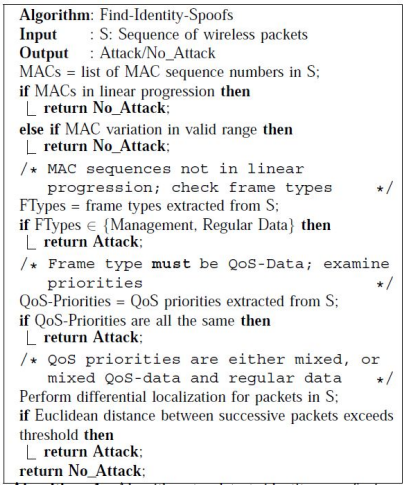
The algorithm was published with the paper and is shown in fig.3.2. In the last part of the algorithm, the system must use the localization feature to decide if it is an attack.

Differential in this context means that the rate of change in the distance between packets are measured. Any abrupt change in this distance will indicate that the station is either on the move (for a specific speed), or the physical location between the sending packets are so big that it must indicate an attack.

In a normal situation without any attacks, the differential Euclidean distance between successive packets from a single source, static or on the move, should be within a small threshold. This threshold should be based on a upper limit, based upon the speed of the device and the time interval between measurements.

Additionally, the work from Gill [5] also deserves recognition. He combines several detection methods (two based on RSS, a received signal strength type and a RTS-CTS type) into a distributed specification based IDS. Additionally he also counters Robust Security Network (RSN) based attacks, which will not be covered here.

The RSS based method uses a dynamic profile for each registered MAC;



Algorithm: Find-Identity-Spoofs
Input : S: Sequence of wireless packets
Output : Attack/No_Attack
MACs = list of MAC sequence numbers in S;
if MACs in linear progression **then**
  return No_Attack;
else if MAC variation in valid range **then**
  return No_Attack;
/* MAC sequences not in linear progression; check frame types */
FTypes = frame types extracted from S;
if FTypes ∈ {Management, Regular Data} **then**
  return Attack;
/* Frame type **must** be QoS-Data; examine priorities */
QoS-Priorities = QoS priorities extracted from S;
if QoS-Priorities are all the same **then**
  return Attack;
/* QoS priorities are either mixed, or mixed QoS-data and regular data */
Perform differential localization for packets in S;
if Euclidean distance between successive packets exceeds threshold **then**
  return Attack;
return No_Attack;

Figure 3.2: Proposed algorithm in [3] to detect identity spoofs in 802.11e-enabled networks.

> Every new RSS value observed for a node is to be compared against the last observed value and the absolute difference is measured (RSS-Ddiff). If the RSSdiff is abnormally high (greater than a pre-determined threshold), an alarm is raised for that node

This RSSdiff detection method is the one I am going to test in later chapters.

His results showed that by combining several detection methods into a specification-based model , and use distributed sensors, a reliable and accurate IDS can be made.

4

Distributed Wireless Capture System

The setting I am working in is defined, and limited by Wireless Trondheim's available infrastructure. In cooperation with WT, I got two physical locations, placed approximately 150m apart alongside Elgsetergate. Please see fig.4.1. For convenience I labeled them "Northug" and "Bjorg" both in the figure and in the code.

The Reasons to place the sensors on this location are several; both locations have power and a wired network infrastructure. Also, I can place the sensors as close to the APs under surveillance as possible. It is not too far away, making trips to setup, experiment and troubleshoot easier. It also reflects an urban environment in the sense that it is placed in the vicinity of a heavily

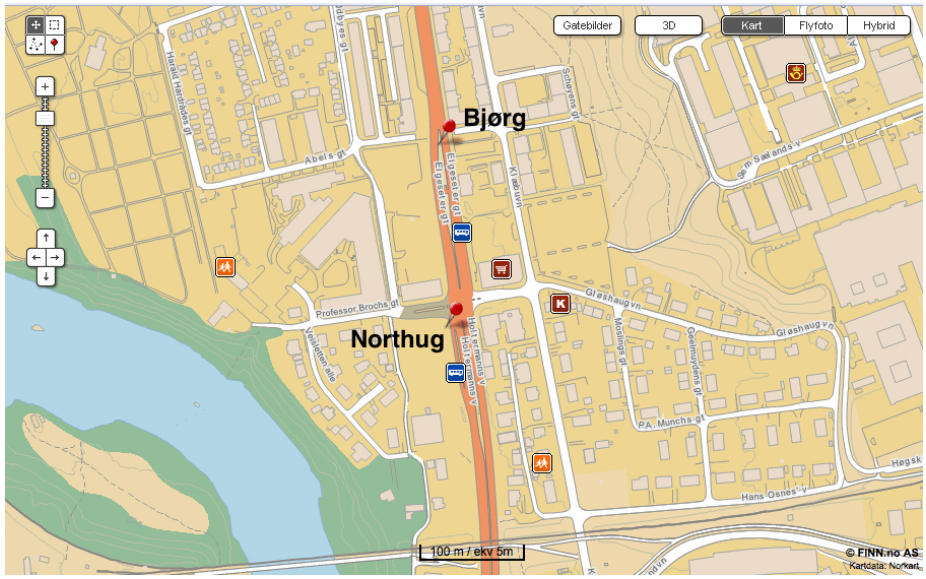


Figure 4.1: Placement of the two sensors, where each are placed 20 cm away from the monitored APs

trafficked road with traffic light-regulated intersections, and concrete buildings. The amount of traffic from both transport and people will affect the sensors, so will interfering wireless WLANs installed in the many homes and office buildings surrounding the locations.

This is also a typical outdoor setting for Wireless Trondheim WLAN coverage in Trondheim. They place two APs that transmits on different channels to create a larger coverage area.

The sensors will forward captured frames back to the central server for processing via the wired network.

The choice of hardware and software used for sensors and central server was largely decided of what available equipment Wireless Trondheim could offer. This will be further elaborated on in the next sections.

4.1 The Sensor

The sensor itself is a small form plug-pc called a **Sheevaplug**[1]. This is a device with limited hardware resources in respect to processing power and memory. Nevertheless, it is capable of running an Ubuntu Linux distribution, which in itself opens up for many possibilities regarding available software.

Besides its 100 $Mbit/s$ Ethernet interface, it has one USB interface for the WIFI dongle. Detailed specification of the plug can be viewed in appendix A.2.1, along with how to make it run with Ubuntu Linux.

The Sheevaplugs were installed inside steel cabinets on the given locations alongside the road in fig.4.1. The USB based sensor antenna was a low-cost WIFI plug, extended with a 3m USB cable mounted approximately 2.5m off the ground in a shielded plastic container.

See appendix A.2.1 for how it was setup.

[1] The Sheevaplug is referred to as a "wall wart device". A wall wart device is best described as a bulky AC adapter, often supplied with many commodity electronic devices.

Figure 4.2: Sheevaplug as advertised on the www.plugcomputer.org, June 2010.

4.2 Kismet-Newcore

Because the Sheevaplug was capable of running Linux programs, I chose the wireless sniffer program Kismet-ng [19] as a platform for retrieving captured 802.11 frames. I used the Kismet-ng release `Kismet-2010-01-R1` which can be downloaded from http://www.kismetwireless.net/. Details of compiling and configuring Kismet is found in the appendix A.2.2.

The main reason for using Kismet in my thesis is because of its distributed sensory features, and its logging features. It logs any raw 802.11 captured data from sensors in .pcapdump format, with it a log of *where the frame was captured from* (i.e. which sensor captured the frame) in an additional .gpsxml log file.

Apart from its distributed sensory capability it also has some basic IDS features that will not be covered in this work.

Kismet is a suite of three programs; For the server part it has the Kismet-server, for GUI clients it has Kismet-client, and lastly for the drones the Kismet-drone application. Kismet has the ability to aggregate captured data from sensors (the drones) to a central server running the Kismet-server application. The drones themselves runs kismet-drone for capturing frames via a wireless interface. The Kismet-client application is used to connect to the Kismet-server, to view the live data capture from sensors.

Important in respect to received signal strength (RSS) values, Kismet can be

configured to prefix Per-Packet Information (PPI) headers (see page 19 for an explanation of PPI) to captured frames, containing information from the physical layer, like RSS values.

To be able to run Kismet-ng, it must first be compiled (built) for use on the desired platform. This process is slightly different for the Sheevaplug and a regular x86 based computer.

There are two options for compiling on a Sheevaplug (or on ARM architectures generally); one is to download source codes directly onto the ARM device, and build the application with the device's limited resources. The other is to cross-compile on a regular x86 architecture and copy the finished application to the ARM architecture.

The main reason for not building on the original architecture is that it is very slow compared to building on a faster platform with more resources. I tried both, and quickly ran into problems when cross-compiling. A compilation done at the Sheevaplug worked best, and did not take more than 10 minutes. Compilation of Kismet is straightforward and is described in Kismet-ng documentation on the website.

Each of the components and their setup will be discussed in the next sections.

4.2.1 Kismet-drone

Each of the drones Northug and Bjorg was setup as shown in the configuration files in appendix A.2.3. Note that the only difference between the files are the name of the local capture device, and which port the application would be forwarding traffic to.

In this work it is important to capture as many frames as possible, and the default behavior of channel hopping in Kismet is not a wanted situation. This is remedied by locking the WIFI plug in monitor mode, to a predefined channel.

Additionally a dummy `gpsd` process must be running to feed the Kismet-drone

the GPS coordinates for the captures, specifically the GPS position of the sensor itself. The coordinates helps to distinguish which frame was captured at which sensor, and is added to every captured frame sent to the Kismet-server. In this work it is important to note that the GPS coordinates used during experiments are only used for identifying which sensor captured a corresponding frame.

Kismet-drone is started by executing `kismet_drone`. The program will then listen on the configured port for any Kismet-server trying to acquire it. When the Kismet-drone has acquired it, it will start forwarding captured packets.

Below is an example used to start monitoring on the Northug sensor. The `gpsfake` command is running in the background for GPS data, the `iwconfig` command locks the interface on desired channel (channel 1) and the `kismet_drone &` starts the drone.

```
/usr/bin/gpsfake -o "-n -N" -r "r" /etc/gpsnorthug1 > /dev/null 2>&1 &
iwconfig wlan0 mode monitor freq 2.412G
kismet_drone &
```

4.2.2 Kismet-Server

The Kismet-server was run on the same Dell PC used during the project in [4] with a Ubuntu 2.6.28-18-generic kernel. The configuration file used for the Kismet-server can be viewed in appendix A.2.3.

Note the configuration for the drones, and their respective addresses:

```
ncsource=drone:name=Northug,host=127.0.0.1,port=2502
ncsource=drone:name=Bjorg,host=127.0.0.1,port=2503
```

It was necessary to forward drone capture traffic via a SSH tunnel because of intermediate firewalls between the drones and the server (an overview of the architecture can be viewed on page 40). The tunnel was created via a second server called `samson.item.ntnu.no`, and then forwarded to the Kismet-server.

While this was less than optimal, it was a quick fix in contrast to update the access lists on the firewalls.

Please note that in a setting where the server can reach the drones directly, this tunnel is not necessary. The author does not recommend this approach if it can be avoided.

The command to create the tunnel (on the Kismet-server):

```
ssh -f -L 2502:129.241.95.221:2502 oysteipe@samson.item.ntnu.no -N
ssh -f -L 2503:129.241.95.222:2503 oysteipe@samson.item.ntnu.no -N
```

After the configuration and the setup of the tunnels, the Kismet-server is started by typing `kismet_server` on the central server. It will then create the logfiles as specified in the configuration file, and fetch captures from the drones.

If the Kismet-server was stopped, it would finish the current capture and write it to file. Restarting the Kismet-server will cause it to start a new capture with new files. In this way it is possible to periodically restart the application, generating log files for later analysis. The trigger for restarting can be put in `crontab` for time based restarts, or in a script triggering a restart when the files exceeds a certain size.

4.2.3 Kismet-client

The Kismet-client is a ncurses[2] based GUI for the Kismet-server. It connects to the defined port that the server is running on, and displays the current capture on the server. It was not needed for the capture process to work, but was a handy tool for verification during experiments.

A screenshot from a capture session is shown in fig.4.3. This is a GNU screen-session[3] where the terminal window from both sensors and the Kismet-client

[2] A library that enables text-based GUI.

[3] From Wikipedia - GNU Screen is a free virtual terminal multiplexer that allows a user to access multiple separate terminal sessions inside a single terminal window or remote terminal

are in the same window. In this way it is possible to keep an eye on the sensors and keep track of the capture process on the Kismet-server.

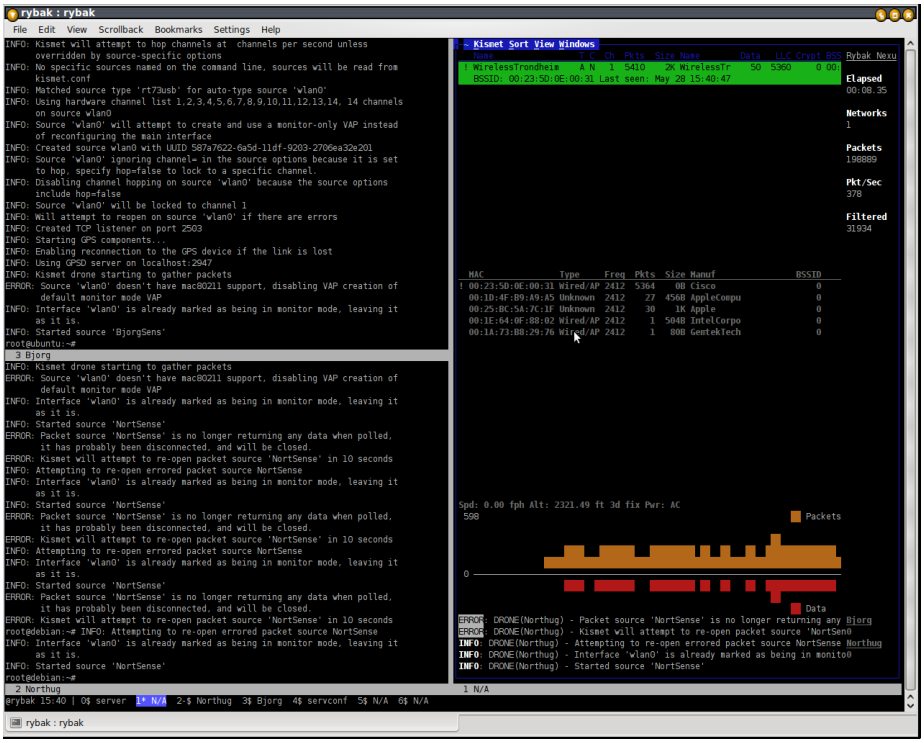


Figure 4.3: Screenshot of a capture session with Kismet-client. Left side shows the output of the two sensors, while the right side shows the Kismet-client.

4.3 Architecture Overview

Overview of the system architecture is shown in fig.4.4. The top tier show the wireless infrastructure with both WTs access points and the Kismet-drones

session. It is useful for dealing with multiple programs from the command line, and for separating programs from the shell that started the program.

installed [4] beside them. Stations communicate with WTs wireless network, while the drone sensors monitor the traffic.

In tier two, all stations route its Internet traffic via the NOMADIX (in blue), while all captured frames from the drones is routed through the SSH tunnel to the Kismet-server (in red).

In the last tier, captured data is processed by the Kismet-server and written to disk.

For the parser to get access to the logfiles, the Kismet-server has to be restarted, thus releasing the files by writing any last capture data to them.

Now that all captured data is in the log files, the data can be parsed and processed by the framework.

4.4 Code Framework

As mentioned in the introduction, the IDS program developed in [4] was never designed for RSS analysis or treating packets from multiple sensors. Consequently, a rework of the code had to be done.

Features that needed to be implemented in rss.pl were identified to be the following:

1. Correctly recognize 802.11 frames and parse needed information from PPI headers and regular frames from .pcap file
2. Parsing of Kismet-ng .gpsxml files.
3. Linking discovered packet to which sensor who captured it(via gpsxml file).

[4]The sensor antenna is mounted 20cm from the WT AP antenna to get approximately the same signaling conditions as the AP. This however is also reliant of the sensors sensitivity, as will be apparent during tests.

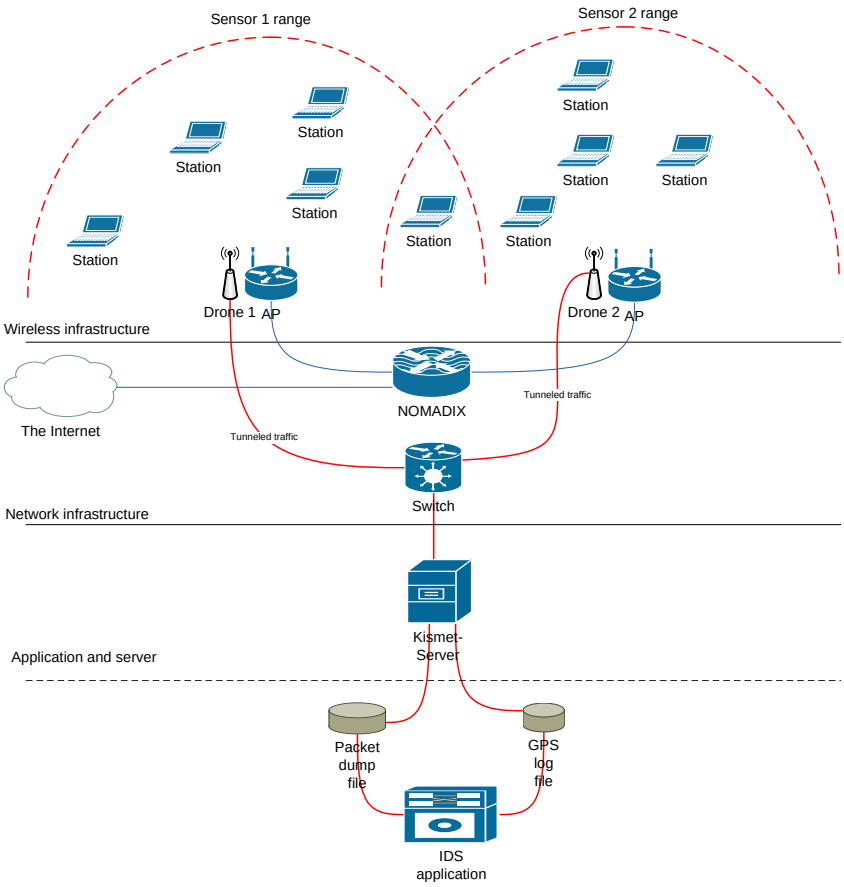


Figure 4.4: Logical architecture of the distributed capture system.

4. Methods for data extraction and calculation for detection purposes.

Source code is found in appendix A.4.1.

The program is started by giving two files as arguments; first a .pcapdump file, the second a .gpsxml file. A typical execution looks like this:

```
rss.pl log-20100530-20-04-34-1.pcapdump log-20100530-20-04-34-1.gpsxml
```

In essence, the program compares the two files, and by per-packet basis matches the captured frames in the .pcap log file with the lines in .gpsxml. The result is a dataset which can be used as basis for further analysis.

4.4.1 Software Specifics

To be able work with MAC frames an IDS program must be able to communicate with the wireless interface. The library Pcap (short for packet capture and referenced as libpcap in *NIX systems) is open source and widely supported in much of todays work within networking. It is an API enabling software to capture packets onto the link layer. It can also write packet dumps to file, and read files in pcap format for processing.

Using Perl and it's wast support of libraries, including a wrapper to libpcap, makes experimenting with network capture easy, and is my choice of a software platform for experimentation.

For example, the ability to read files in .
pcap format is especially important for testing an IDS, as the dataset is non-changing.

5

Experimentation

Wireless Trondheim's infrastructure is made up of a Cisco Unified Wireless Network and a captive portal. The stations involved in the experiments uses WTs logon based solution, which adds authenticated stations to its whitelist on the NOMADIX (see further discussion of WTs infrastructure in [4]).

Attacker and victim uses standard off-the shelf DELL laptops with wireless transmitters, with or without QoS provisioning (se chapter 2.4.2). The antennas on the laptops are omni-directional.

The wireless distributed system is setup as explained in chapter 4.

5.1 Design of Experiments

The experiments are divided in two parts. The first part consists of control experiments with no attacks. The second part consists of attack experiments with freeloading attacks (see [4]). Both sensors are operative and monitoring in control experiments 1-2 and in attack experiments 1-5.

Note that a special set of experiments was done for just one sensor in control experiment 3 and in attack experiments 5 and 6. This was decided largely because of suspicion of low sensor sensitivity with loss of frames as a result. This will be further elaborated in later sections.

Control experiments was executed with a single laptop sending packets continuously. Attack experiments was executed with the victim on the same spot as in the control experiments, while the attacker was attacking from different positions. All attacks was based on the freeloader attack, where the attacker spoofs the victims MAC address.

During attack experiments, the attacker is free to change MAC address, victim is not. Both stations generate traffic by primarily generating ICMP packets to an external site unless otherwise stated. The duration of each experiment is given in seconds in the resulting plots.

During each experiment a new Kismet-server instance is started at the central server. It will then connect to the two drones and start capturing frames to file. After the experiment is finished the Kismet-server program will be terminated.

The sensors will be in monitor mode and set to a specific channel. All captured data will be forwarded to the central Kismet-server where it will be aggregated into log files as described in section 4 about Kismet. In this way, there will be reproducible attack data for later testing of the IDS detection methods.

After these are executed and gathered as logfiles at the centralized server, the script will be used to see if any MAC spoofing attack is performed. The description of each case is explained below.

Because of the low number of sensors, and to be able to capture as many frames as possible, I must concentrate on monitoring one channel at each scenario. Both sensors will be set to monitor channel one, or frequency band 2.412 GHz. To further ease the testing, the station used in the test will be forced to use channel 1 as well.

In a more natural scenario, the station would roam between the two towers when the sensitivity threshold is low enough, causing it to change frequency. If one were to capture frames in such a scenario, the sensor should change its listening frequency based on a channel hopping algorithm [10], or simply just install another sensor operating at the desired channel. Because of some indication of lost packets on both sensors detected throughout the experiments, the sensors is restarted and setup anew between sessions. Each gap of lost frames in the results will show to be about 10 seconds, correlating to error messages on the drones themselves. Possible causes can be infrastructure related, e.g. with the SSH tunnel used, or software based issues e.g. with the drone installation on the Sheevaplugs or the Kismet-server. The effects could be negligible from my point of view, but should be considered if new experiments with the same setup is to be done.

5.2 Execution of Experiments

A short explanation of the key elements in each experiment is shown. All experiments was executed in the area as shown in fig.4.1 on page 32. The setup of the stations is described in appendix A.3.2. The setup and commands used to execute attacks are in appendix A.3.3.

Regarding the traffic mix from both attacker and victim stations, experimentation with different programs ensued before settling on a more stable traffic mix.

To generate enough frames for the experiments, the clients must transfer data.

Experimentation with different programs like `ping` , `scp` and regular browsing was tested in the first experiments. Unless stated, both attacker and victim stations generates ICMP packets by using the `mtr`[1] program for linux. Usage of the flood option in `ping` was considered, but was deemed too aggressive for these experiments. The attacker was always contacting `www.db.no` and the victim/station was always contacting `www.vg.no`. This was done to differentiate between frames in Wireshark analysis, and was not used in the detection methods.

Control Experiments The station was stationary at a bus stop in the middle of the two sensors. In control experiment 1, the traffic mix consisted of pinging `www.vg.no` and copying a large file via scp to an external server, while the control experiment 2 used the `mtr` program.

In control Experiment 3 the station was positioned close to the sensor Northug during the session, about 3-4 meters. The sensor Bjorg was disabled during this session via commenting-out the `nsources` option in the Kismet-server config file.

Attack experiments The victim station was placed at the same position as in the control experiments. The attacker was positioned approximately 50 meter away from the victim.

In attack experiments 1 and 2 the attacker and victim had a traffic mix of both web browsing, pinging and file copy with scp. This mix was abandoned in later experiments

Attack experiment 4 lasted 25 minutes, where the first 10 minutes consisted of an attack at the same spot as before, and the last 10 minutes the position was changed. The attacker was not transmitting during the move to the new position, so all attacks was when the attacker was not moving.

[1] MyTraceRoute - a different version of traceroute. Keeps sending ICMP packets for deciding latencies between routes to the destination.

5.3 Data Analysis

After the experiments was done, I needed a way to analyze the data. By using the framework from section 4.4, the logfiles from each experiment was processed. By using `R` to generate plots and statistics from the log files, I was able to visualize the results, easing the analysis of the data. Code for generating R graphs and statistics are included in appendix A.4.2.

After each experiment a pdf file with plots of the following was created:

1. Differential RSS values, RSS samples and Sequence number plot for Bjorg sensor
2. Differential RSS values, RSS samples and Sequence number plot for Northug sensor
3. IDS plots
4. Density distributions of RSS values plots
5. Normal Q-Q plots

These files was used as basis for analysis, together with numerical generated data from `R`.

6

Results

This chapter will illustrate the results from the experiments conducted on the sensor network. All experiments will be illustrated with plots from each sensor that will give a foundation for further improvement of the detection methods using RSS measurements.

An explanation of the plots follows.

I illustrate the *power differences between frames* in the first plot, the *sampled RSS from each frame* in the second plot. Lastly, plots of *sequence numbers per frame* is shown in the last plot. Higher values in dBm in plots indicate a stronger signal. A line is added to the RSS sample plots, resembling mean values at the corresponding times.

Note that when the system misses packets (for some reason) it will show as empty gaps in the plots at corresponding times as explained in chapter 5.

In the attack experiments the sequence number plots will show if a MAC address is sending with deviating sequence numbers. Note that the sequence number plot was*not* considered in analysis of MAC address spoofing.

All frames recorded are from the same station used in the experiments with MAC 00:22:fb:89:5a:36. An attacker will also use this address during attack.

6.1 Control Experiments

All control experiments are used as a basis for comparison later when analyzing data from the attack experiments. It also serves as an indication of how well the sensors work in capturing frames.

As an initial note,it seems that for some reason the sensors (both of them) are reporting RSS values in even numbers. This is very unfortunate, as it will decrease the resolution of the RSS measurements.

6.1.1 Control Experiment 1

The results from the experiment are shown in fig.6.1, fig.6.2 and table 6.1

Notable observations Differential RSS readings from both sensors seem to be clustered in the area of -10 to +10 dBm. This is a much higher variance compared to the results from Papini [20], where the variance were in the region of +/- 5dBm. Although the sensors only report even numbers, it does not explain the large difference.

Power samples at both sensors are fairly constant in local areas, shown with steady means . Power samples also show a correlation where Bjorg has a drop

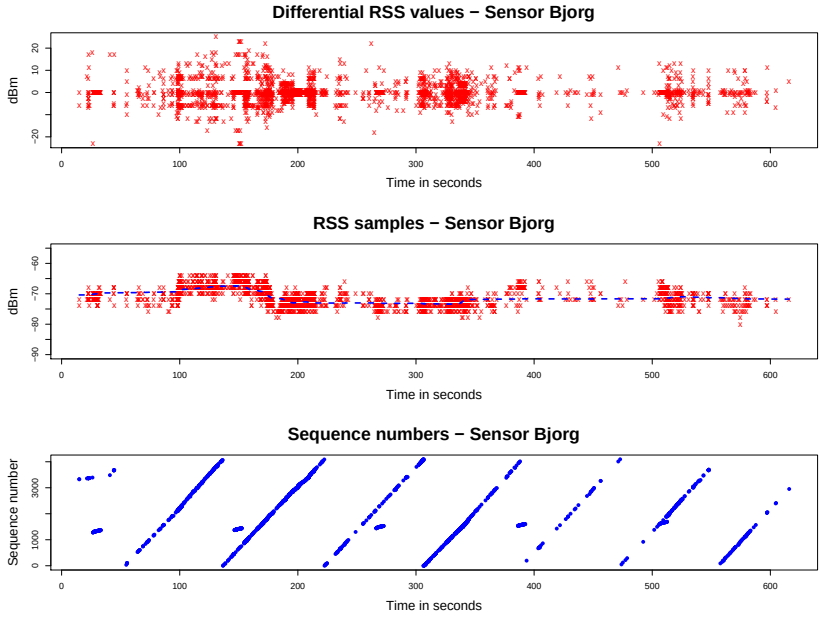


Figure 6.1: Control experiment 1, sensor Bjorg

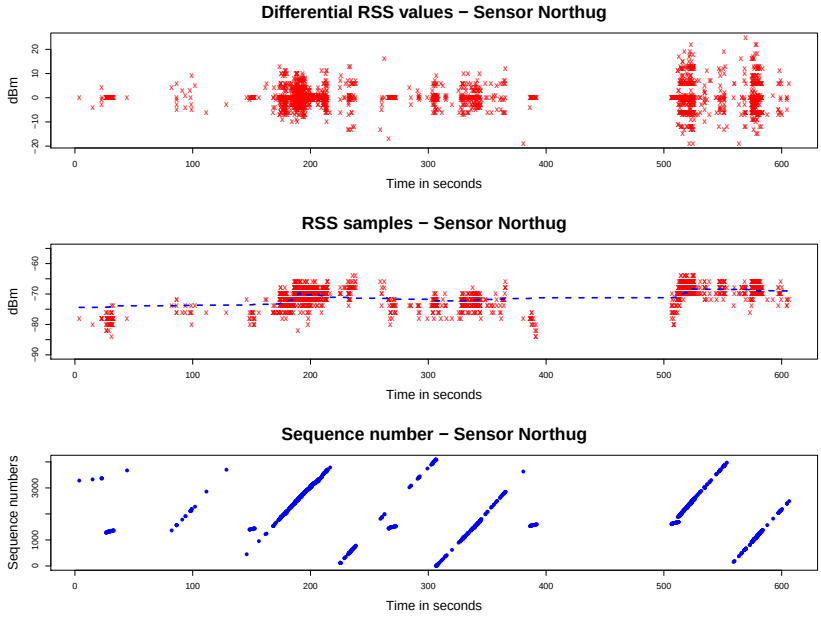


Figure 6.2: Control experiment 1, sensor Northug

Experiment	Sensor	Frames	Std.dev
Control Experiment 1	Bjorg	2570	4.8886
	Northug	2457	4.7849
Control Experiment 2	Bjorg	860	4.8847
	Northug	5910	4.6352

Table 6.1: Summary table for control experiment 1 and 2.

in power measured, at the same time as Northug power measures *increases*. This indicates that the environment of the experiment changed significantly at that time.

Samples that actually are registered shows a normal sequence of numbers, increasing as it should without additional multiples of sequences. However, there are repeated clusters of numbers appearing regularly on both sensors. After closer scrutiny of the logfiles together with viewing of the .pcapdump file with Wireshark, there seems to be repeated 802.11e QoS Null frames that is being sent from the station. The 802.11 standard defines sequence numbers in QoS(+)Null frames to be set to any value, [1, p.66], so this explains that anomaly. All subsequent graphs show the same pattern.

6.1.2 Control Experiment 2

The results from the experiment are shown in fig.6.3, fig.6.4 and table 6.1

Notable observations The sparse graph at sensor Bjorg compared to Northug is explained by the number of sampled frames at the sensors. Frame count for Bjorg was 858 and Northug 5904.

The high sample rate on Northug helps in giving confidence that the signal did

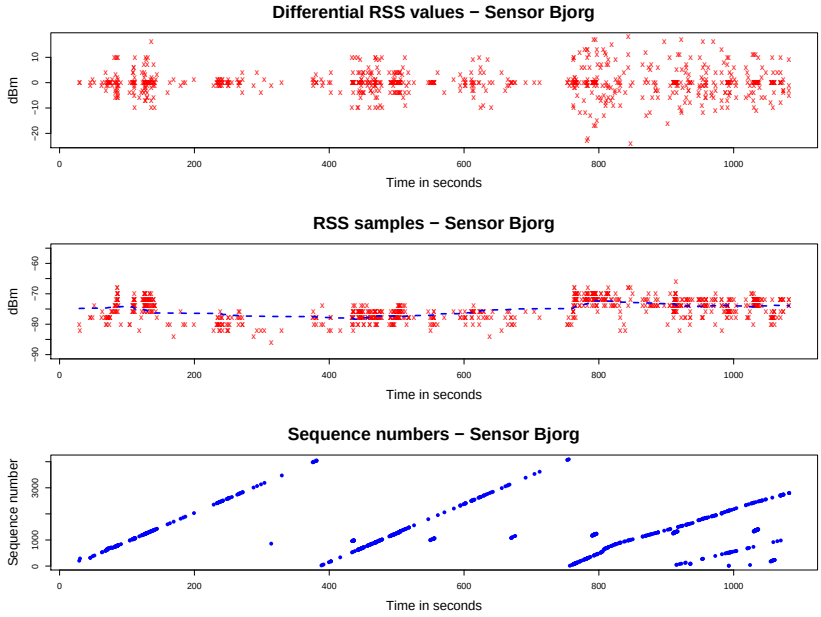


Figure 6.3: Control experiment 2, sensor Bjorg

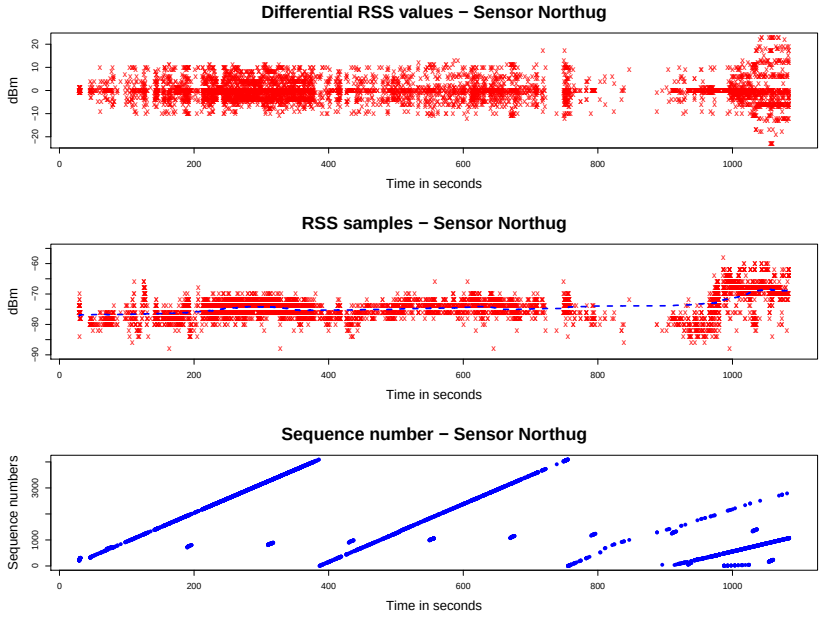


Figure 6.4: Control experiment 2, sensor Northug.

not differ more than +/- 10dBm, with an exception in the end of the experiment. Bjorg sensor showed some indication of the same variance, although it had few captured frames.

Power sampled at both sensors show a slightly changing mean. At around 780 to 860 seconds Bjorg mean increases slightly, while Northug loses captures altogether. Around 900 to 1100 seconds the RSS mean at Northug suddenly gets a rising mean, with some larger differential RSS values, while Bjorg mean seems constant.

Sequence graphs show indication of one station sending with the MAC address.

6.1.3 Control Experiment 3 (Northug only)

Since the resolution from experiments when the two of the sensors were capturing were inconclusive, it makes sense to test if using a single sensor only will give any conclusive answers. This time both attacker and victim was in the vicinity of the sensor. The victim was approx. 2m away from the sensor. The attacker was approx. 30m away the sensor. See fig.6.5 for illustration of placement.

The results from the experiment are shown in fig.6.6 and in table 6.2

Notable observations The graphs show a clear change in regards to variance in the differential RSS values. The frame count is also much higher than the other control experiments (10857 frames).

These plots resembles the results from Papini [20] with little or no deviation.

6.1.4 Analysis

Comparing the values from the experiments I get the following:

By studying the table 6.2 it is clear that the sensors capture frames more readily when the source is much closer, and also get less variable readings.

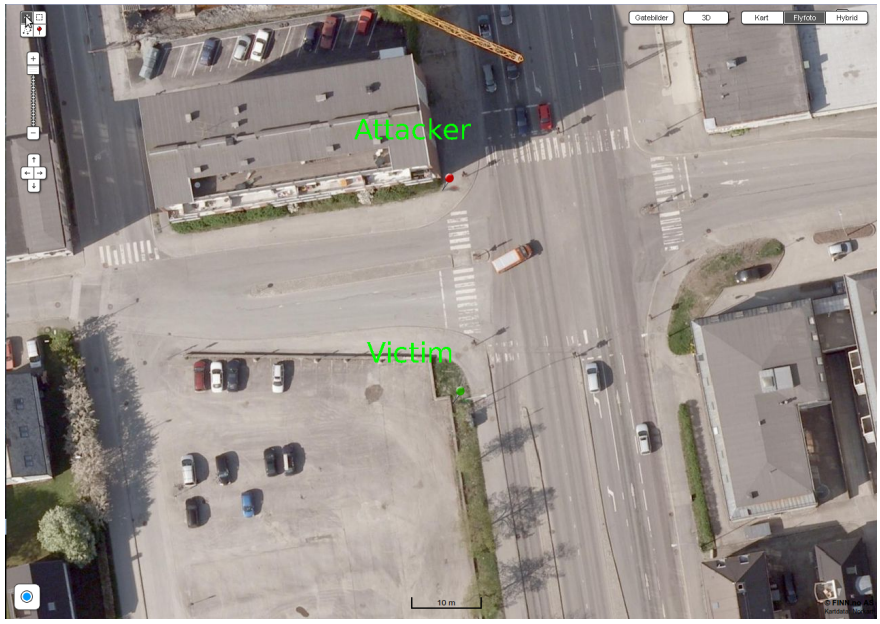


Figure 6.5: Placements of attacker and victim

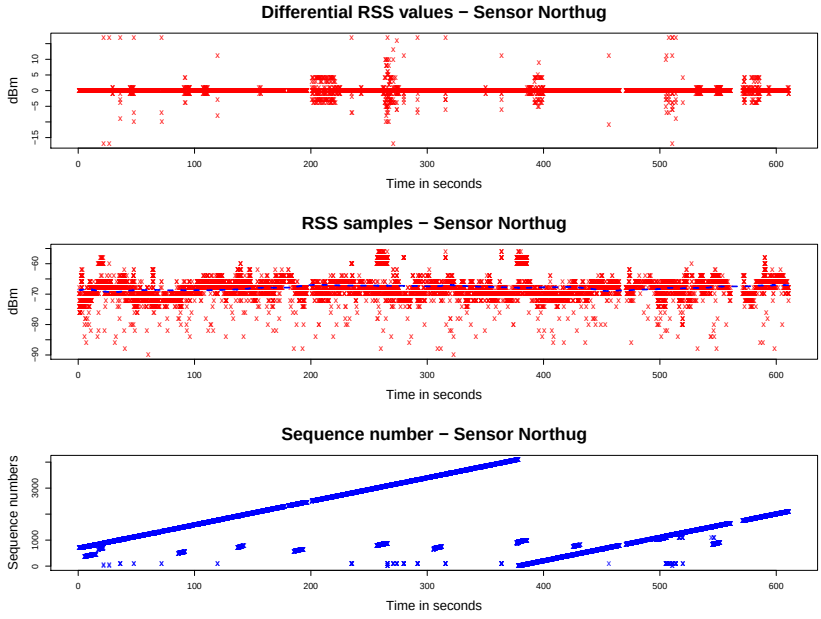


Figure 6.6: Control experiment 3 (Northug only)

Experiment	Sensor	Frames	Std.dev
Control Experiment 1	Bjorg	2570	4.8886
	Northug	2457	4.7849
Control Experiment 2	Bjorg	860	4.8847
	Northug	5910	4.6352
Control Experiment 3	Northug	10857	1.0850

Table 6.2: Summary table for the control experiments

Nevertheless, this is the base measurements that is to be measured against the results from the attack experiments.

6.2 Attack Experiments

6.2.1 Attack Experiment 1

Both attacker and victim were non-moving while the experiment lasted. The victim station was placed at the same spot as in the non-attacking experiments, in the bus stop. The attacker station was placed on the curb some 50 meters from the victim. Attacker was closest to Northug sensor, while the victim was closest to the Bjorg sensor.

The first minutes in the beginning of the experiment was used to prepare and walk over to the spots before actually executing the attack.

Both attacker and victim were pinging `db.no` and `vg.no` respectively to generate generate ICMP messages, and browsing the web.

The results from the experiment are shown in fig.6.7, 6.8 and table 6.3.

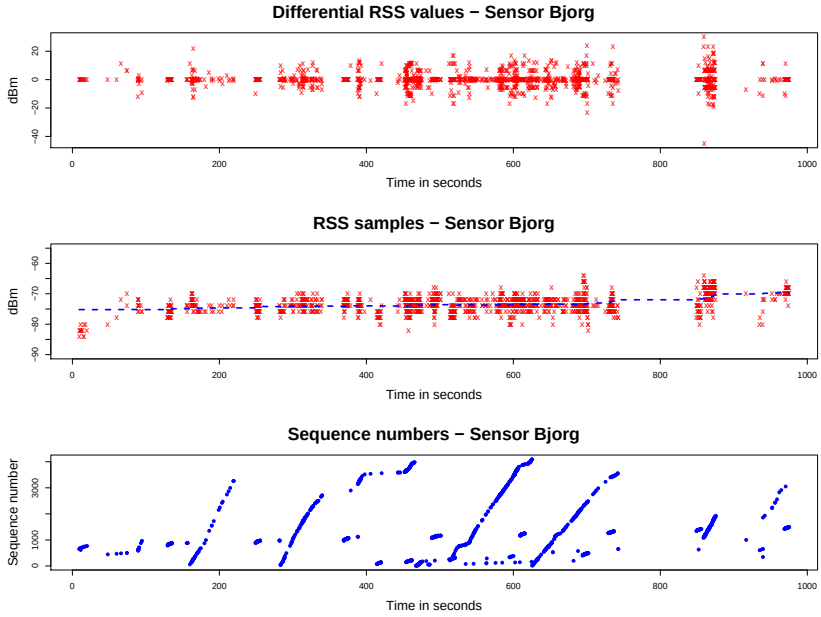


Figure 6.7: Attack experiment 1, sensor Bjorg.

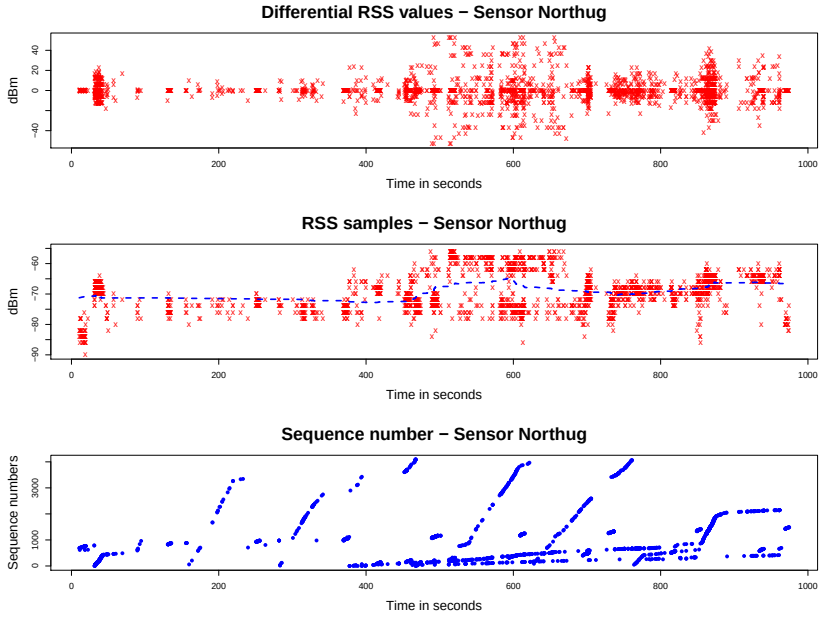


Figure 6.8: Attack experiment 1, sensor Northug.

Notable observations Studying the table 6.3, the standard deviation shows twice the readings compared to the control experiment on Northug. One may come to the conclusion that Northug is receiving the signals from the attacker more strongly than Bjorg, causing the relatively low packet count. This is further verified by the sequence-number plots, were Bjorg seems to only capture most packets from one station, causing the rapidly increasing sequences, while Northug clearly (from approximately half-way in the plot) receives frames with two different sequence numbers.

The two sensors' delta plots show that between 0 and 380 seconds there are little traffic. Because the victim station was a Linux client, it did not have any active firewall in place, and would send a TCP-Reject message, promptly disconnecting TCP sessions initiated by the attacker when trying to browse or initiate other TCP connections. By executing the same IP table command as in appendix A.3.3, this was solved. This happened halfway in the experiment, and correlates with what the plots are showing.

Comparing the two sensor's power delta plots after the IP table command, we see that Northug's captures vary more strongly than Bjorg's, causing the outliers to vary between -45 to +30 for Bjorg compared to -53 to +53 for Northug. Northug also has more captured frames than Bjorg, 2529 and 1555 respectively.

6.2.2 Attack Experiment 2

This experiment was conducted right after attack experiment 1, and the traffic mix was the same.

The results from the experiment are shown in fig.6.9, fig.6.10 and table 6.3.

Notable observations From table 6.3 we can see that Northug captured most of the traffic in this experiment. Because Northug sensor has so many more frames captured than Bjorg, it is natural to concentrate more on Northugs results.

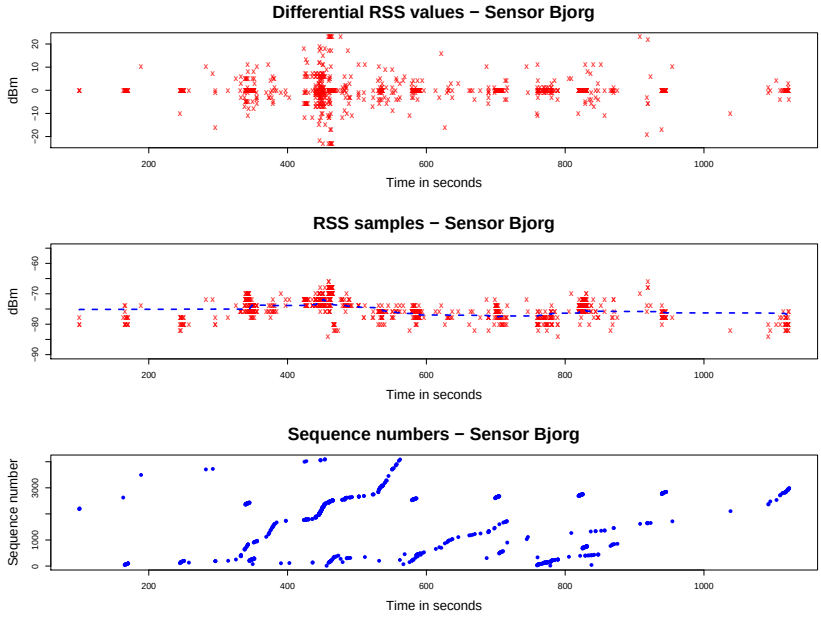


Figure 6.9: Attack experiment 2, sensor Bjorg.

Figure 6.10: Attack experiment 2, sensor Northug.

Experiment	Sensor	Frames	RSS diff. Std.dev
Attack experiment 1	Bjorg	1555	5.3500
	Northug	2562	11.9106
Attack experiment 2	Bjorg	984	5.1575
	Northug	8047	12.1045

Table 6.3: Summary table for attack 1 and 2.

If one were to analyze Northugs power deltas, one can see that there are huge differences from the control experiments. The standard deviation also suggests this.

This observation only holds for parts of the experiment though, as one can see that there are periods where no traffic are registered at all. It seems that both sensors lose packet captures in the last segment of the experiment. I suspect that this has something to do with the infrastructure delivering packets to the central server.

Clearly, the attacker (nearest Northug) had the best signaling conditions for Northug to capture, as one sees from the strong signal level registered in the power samples plot.

It is also apparent that now Northug captures more frames than the last experiment. The attack itself could explain this, as both attacker and victim generated more traffic.

6.2.3 Attack Experiment 3

Both attacker and victim were in the same positions as the former two attack experiments.

This time the author also had a Macbook to control the experiment. Viewing registered traffic at the central server via Kismet-client confirmed that the sen-

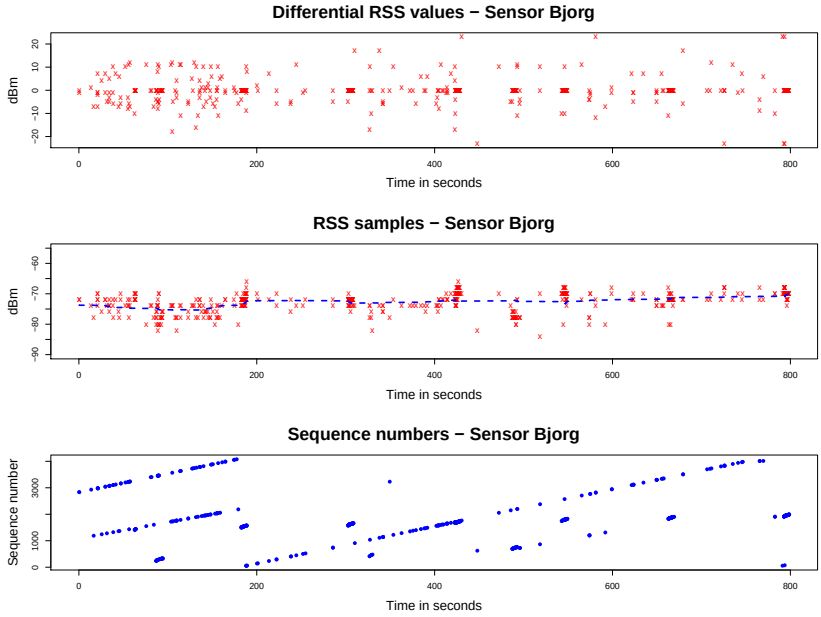


Figure 6.11: Attack experiment 3, sensor Bjorg.

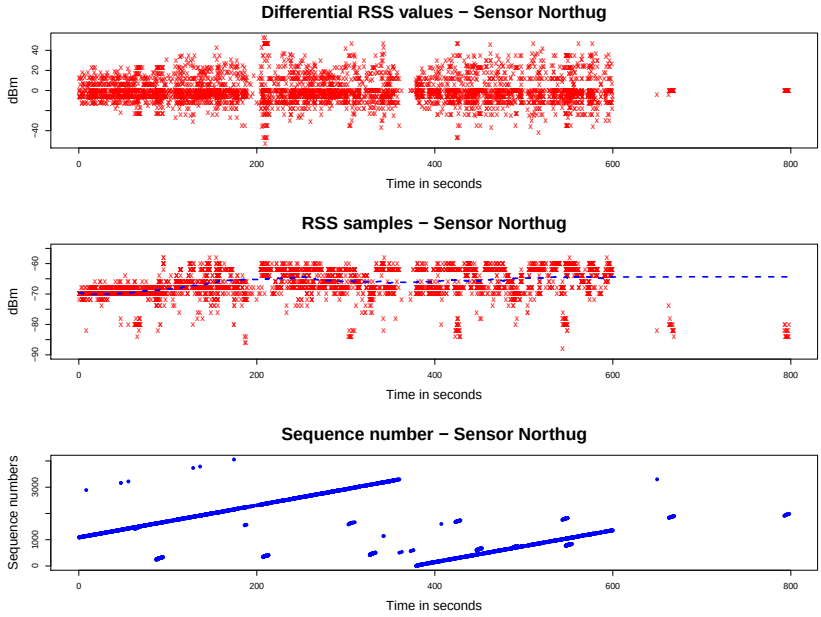


Figure 6.12: Attack experiment 3, sensor Northug.

Experiment	Sensor	Frames	Std.dev
Attack experiment 3	Bjorg	484	5.3562
	Northug	4453	12.7301
Attack experiment 4	Bjorg	3297	7.1216
	Northug	2014	4.1790

Table 6.4: Summary table for attack 3 and 4.

sors captured frames at all (see fig.4.3). As a note, Macbook wireless access operated on the eduroam network, which was on a different channel than the experiment.

The results from the experiment are shown in fig.6.11, fig.6.12 and table 6.4.

Notable observations Again Bjorg sensor did not capture as many frames as Northug. The variance at Northugs power deltas however do give a strong indication that an attack is ongoing.

6.2.4 Attack Experiment 4

This experiment is made up of two attacks. First part consists of a regular 10 minutes attack as the previous attacks, the second part the attacker logs of the network, and walks to a position closer to Bjorg sensor, across the road of the victim. Then another 10 min attack is executed, followed by 5 minutes of no attack where only the victim is online.

The results from the experiment are shown in fig.6.13, 6.14 and table 6.4.

Notable observations Because of the longer timespan than the other experiments the standard variance is lower than other attack experiments. The higher

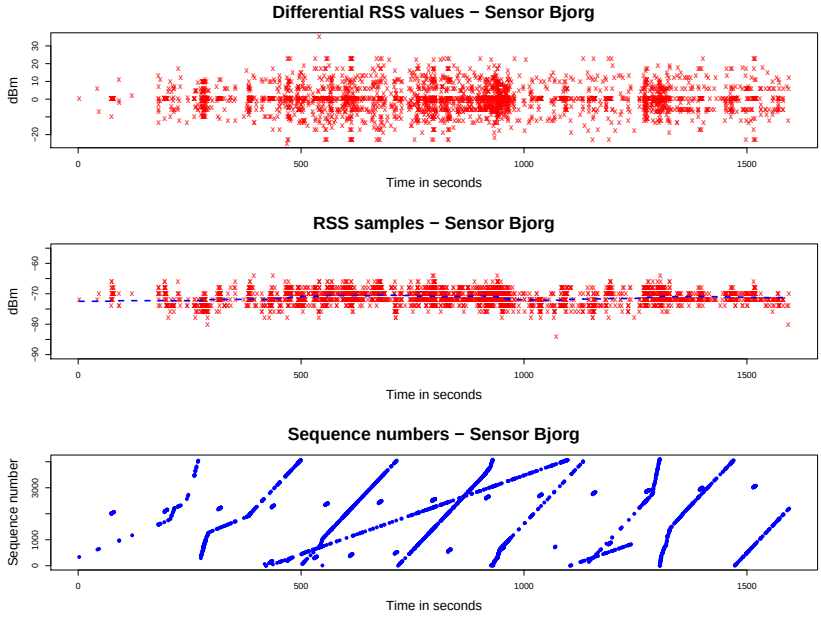


Figure 6.13: Attack experiment 4, sensor Bjorg.

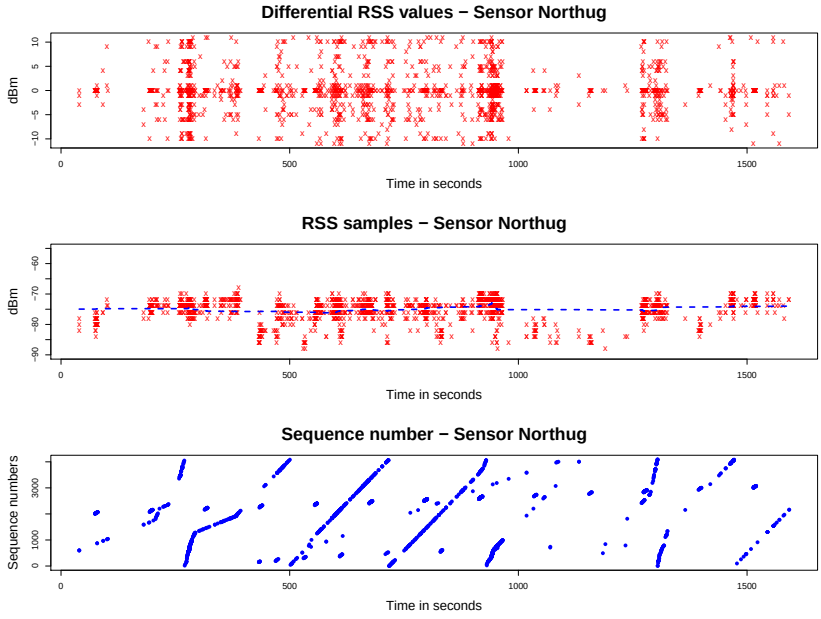


Figure 6.14: Attack experiment 4, sensor Northug.

frame count at Bjorg is expected, as the attacker moved closer to it during the experiment.

6.2.5 Attack Experiment 5 (Northug only)

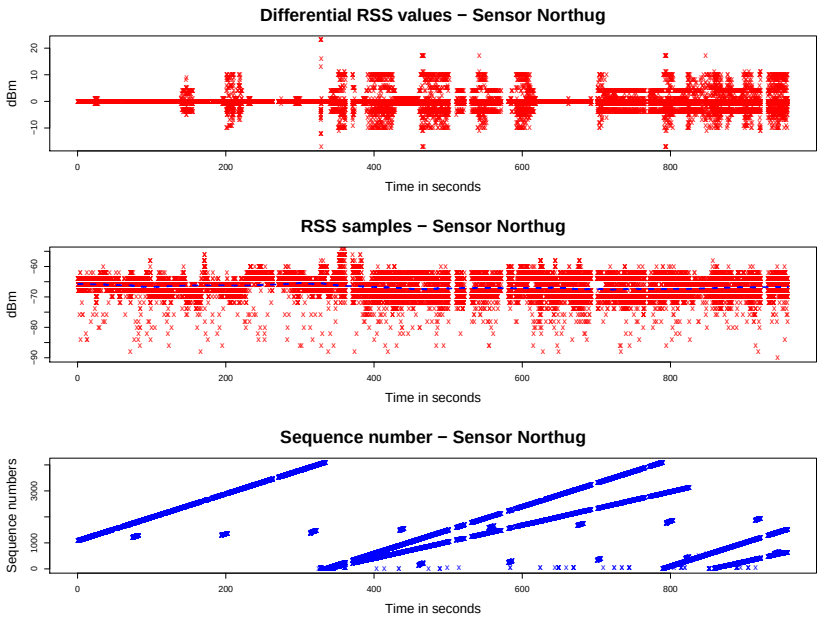


Figure 6.15: Attack experiment 5 (Northug only).

This experiment must be seen together with control experiment 3, as it was executed with the victim on the same place. The attacker was some 30 meters away across a road that was mildly trafficked, see fig.6.5 for illustration of placement.

In the first 6 minutes of this capture, the victim was alone in transferring packets. After 6 minutes the attacker commences its `mtr db.no` and continue to do that for 10 minutes.

Experiment	Sensor	Frames	Std.dev
Control Experiment 3	Northug	10857	1.0850
Attack experiment 6	Northug	16218	3.6689
Attack experiment 5	Northug	26497	2.40167

Table 6.5: Summary table for attack 5 and 6 with a comparison of the data from the single sensor control experiment.

The results from the experiment are shown in fig.6.15 and in table 6.5.

Notable observations The number of frames is more than doubled compared to the control experiment as shown in table 6.5, so has the standard deviation.

Also the graphs show clear signs of larger differential RSS values, clearly a sign of an attack.

The sequence number plot also support that there are two stations on the same MAC address.

6.2.6 Attack Experiment 6 (Northug only)

This experiment was conducted immediatly after the last one without interrupting the attack. Measurements lasted precicely for 10 minutes.

The results from the experiment are shown in fig.6.16 and in table 6.5.

Notable observations The frame count is 10k less this time, with a standard deviation of 3.6 vs 2.4 from previous experiment. Differential RSS values show more or less constant differences of +/- 10dBm, 5dBm more than control experiment 3. Again a clear trail of two linear sequences numbers is present.

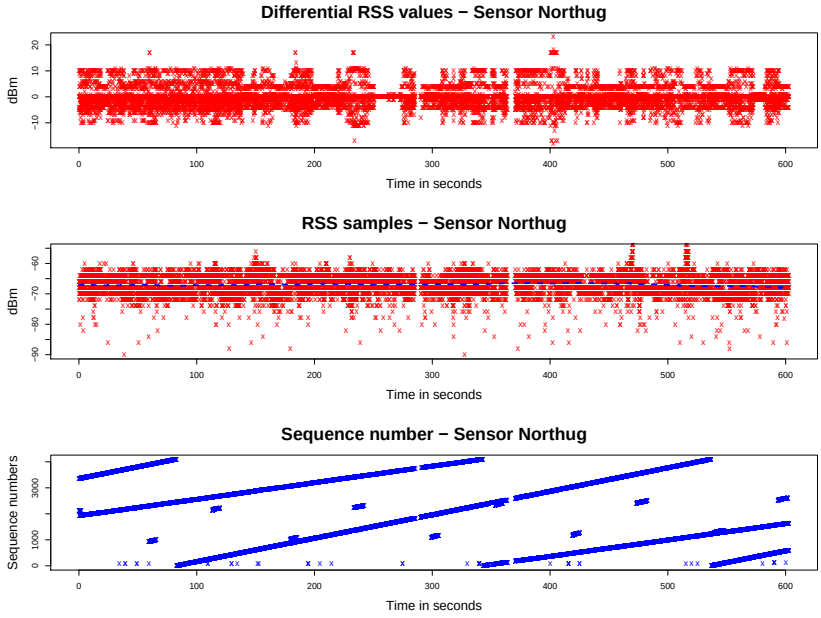


Figure 6.16: Attack experiment 6 (Northug only)

Experiment	Sensor	Frames	Std.dev
Control Experiment 1	Bjorg	2570	4.8886
	Northug	2457	4.7849
Control Experiment 2	Bjorg	860	4.8847
	Northug	5910	4.6352
Control Experiment 3	Northug	10857	1.0850
Attack experiment 1	Bjorg	1555	5.3500
	Northug	2562	11.9106
Attack experiment 2	Bjorg	984	5.1575
	Northug	8047	12.1045
Attack experiment 3	Bjorg	484	5.3562
	Northug	4453	12.7301
Attack experiment 4	Bjorg	3297	7.1216
	Northug	2014	4.1790
Attack experiment 5	Northug	26497	2.4017
Attack experiment 6	Northug	16218	3.6689

Table 6.6: Summary table for all the experiments.

6.2.7 Analysis

As table 6.6 show, it is evident that there are significant differences between measurements when a station is far away from the sensors.

6.3 IDS method testing

From the control-, and attack experiments (section 6.1 and section 6.2) it is clear that the most reliable metric for spoof detection in RSS context *could* be the rate of how *differential RSS (RSSdiff)* changes in a certain period of time.

This is also supported by previous work done by Gill [5].

Using the test data, I implemented a simple RSSdiff detection method based on differential RSS values and evaluated how good a detection method it was. The `rss.pl` script was modified with a subroutine to calculate how many outliers (deviants) were outside a specific threshold, over a certain period of time.

The idea was to see if this way of counting deviating RSSdiff values could be a reasonable detection metric.

For such a method to not generate to many false positives or false negatives, the differential RSS threshold must not be too small or too big. The time window that the method operates in should not be to big either. During this time window the algorithm simply counts if the number of times subsequent frames have a RSS value differing with more than a certain threshold. A time period of 10 seconds was decided after some initial testing for both cases.

Studying the results from the experiments, a threshold value of 15 dBm was considered abnormal for the experiments with two sensors, and 6 dBm for the single sensor experiments. The threshold for the single sensor experiments was decided from the results from Papini, where he seldom got readings over 5dBm in non-attack settings. 6dBm and not 5dBm because of the even-numbering issue with the sensors, to get an even more conservative threshold. This means that subsequent frames with RSS values deviating with 8 dBm or more will be flagged as an attack.

Note that the the x-axis has no relation to the x-axis in former plots, i.e theay are not time-comparable with them.

6.3.1 Detection Method on Control Data

All deviation plots show counts under 20 for the control experiments.

In control experiment 1 fig.6.17 the highest count for Bjorg sensor was 12 during only a few occurrences at the end for sensor Northug, no bigger than 8.

For control experiment 2 fig.6.18, the most notable graph is the sensor Northug, indicating almost no deviations until the end of the experiment where it touches the 20 count mark.

The single sensor experiment in fig.6.19 shows some deviances, but still under 20.

6.3.2 Detection Method on Attack Experiments

From the first attack experiment fig.6.17, Bjorg sensor show no counts over 20, while Northug sensor clearly has increased both the frequency and the counts of deviations. While only a couple of counts are over 20, there seem to be more frequent counts altogether between 30 - 70.

Attack experiment 2 fig.6.21 show Bjorg sensor with only one instance of a count over 20. Northug sensor has both frequent counts of deviation but also high counts altogether in each bin.

In attack experiment 3 fig.6.22 the differential RSS values registered at Northus continues to be frequent and with high counts. Bjorg sensor seems to have 10 fewer bins than Northug, probably caused by missing capture frames.

In attack experiment 4 (see fig.6.23) there are none registered deviances at Northug, and only a few at Bjorg.

In the final attack experiments 5 and 6 (see page 73), the differences are both frequent and many in each bin.

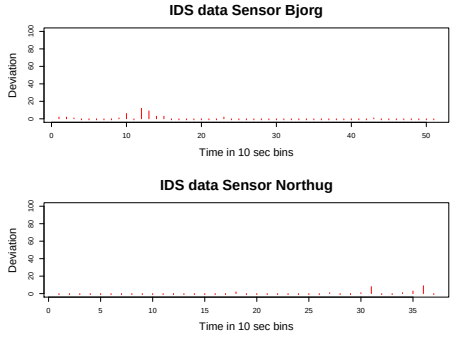


Figure 6.17: Detection method on control experiment 1

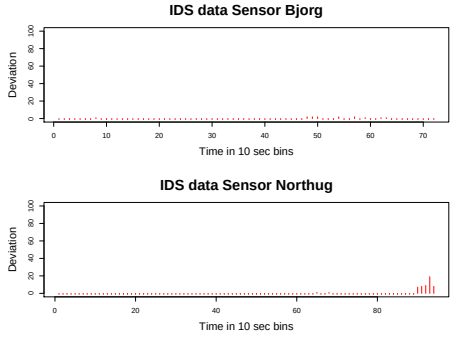


Figure 6.18: Detection method on control experiment 2

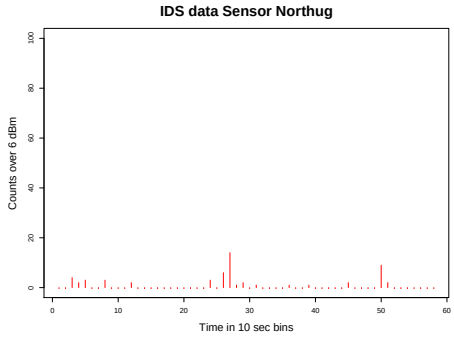


Figure 6.19: Detection method on control experiment 3 (Northug only)

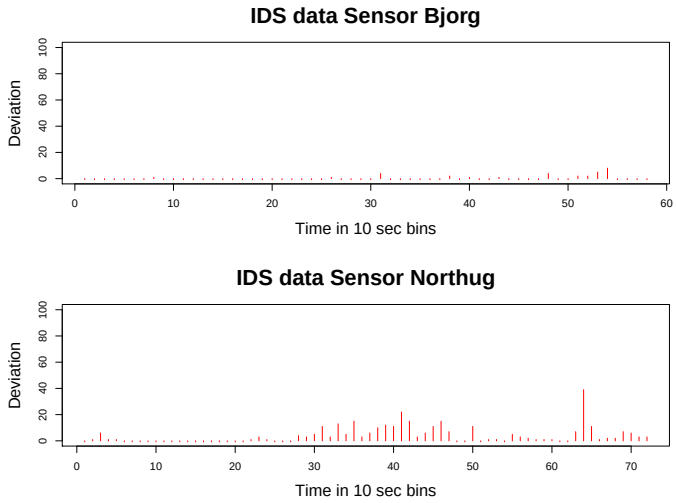


Figure 6.20: Detection method on attack experiment 1

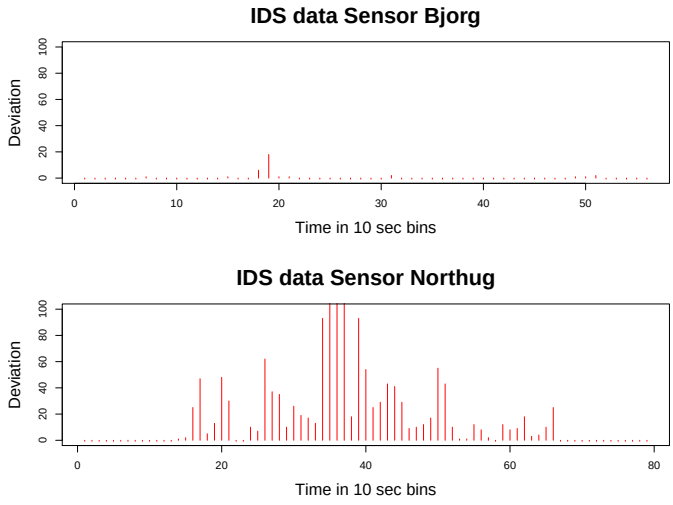


Figure 6.21: Detection method on attack experiment 2

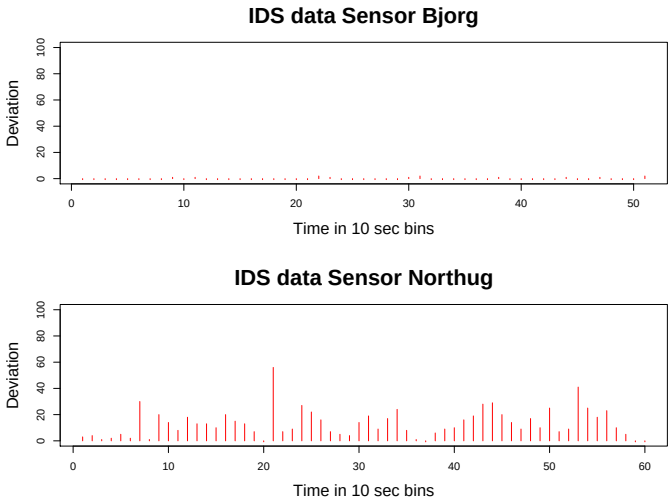


Figure 6.22: Detection method on attack experiment 3

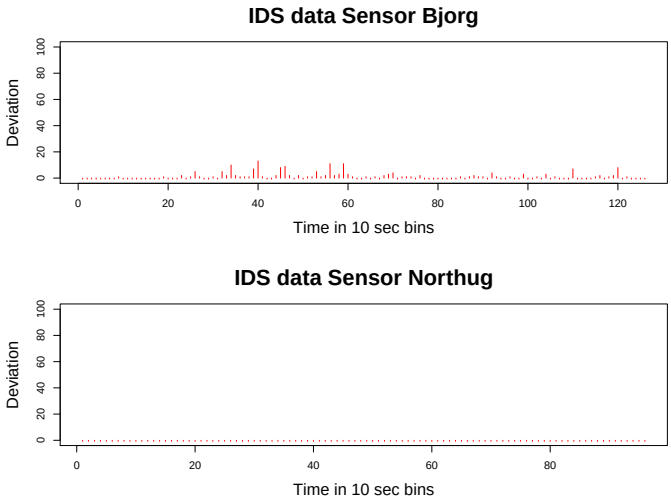


Figure 6.23: Detection method on attack experiment 4

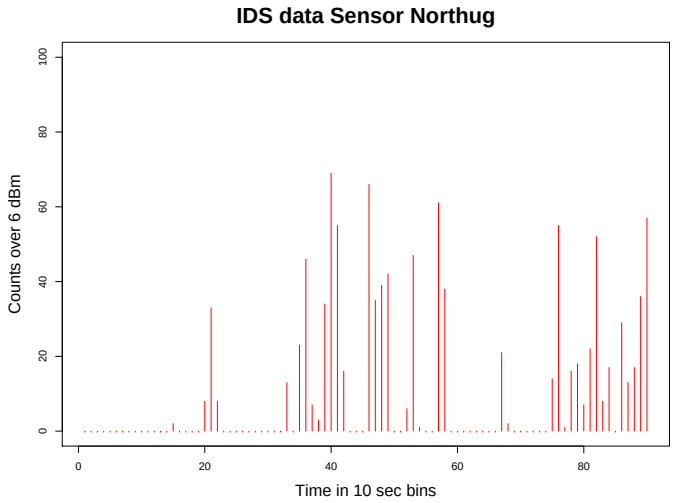


Figure 6.24: Detection method on attack experiment 5 (Northug only)

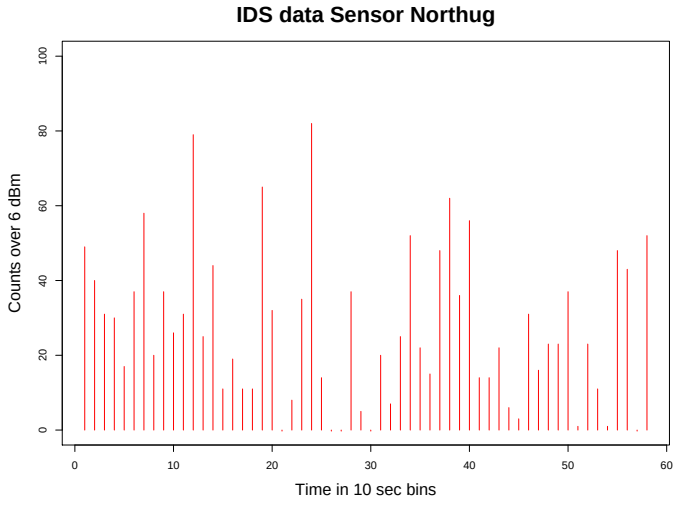


Figure 6.25: Detection method on attack experiment 6 (Northug only)

7

Discussion

Through the control-, and attack experiments in chapter 6, the behavior of RSS values in an urban environment were recorded. Based on the results, a simple RSS based detection method was implemented. The results from these tests will be discussed here.

7.1 Measurement Variability in Experiments

As we will see, variability in measurements directly impacts the performance of the detection method if not carefully considered.

7.1.1 Distance

First a recap of where the attacker and victim stations were placed in contrast to the sensors.

The station used during control experiments and attack experiments was approximately 94m from Northug and 90m from Bjorg (figure on page 32.

The attacker was placed on the curb by Rema 60 meters from Northug, and 133 m from Bjorg. In the last part of experiment 4, the attacker was 152m from Northug and 40m from Bjorg.

As mentioned, the main factor of signal degradation is the distance from the sender to the sensor. This is mainly shown in the number of capture frames, where Bjorg sensor captured the fewest in most of the scenarios.

Also, as the experiments show, with this distance comes large variances in the registered RSS values. In two-sensor experiments with clients far away, control experiments show a standard deviation of the RSS differentials of 4.8, while in the single-sensor experiment the standard deviation was closer to 1.

It is safe to say that distance plays an important role when it comes to how much subsequent frames differ between each frame captured at a sensor, as well as how many frames actually being captured by the sensor.

7.1.2 Environmental Factors

In an urban environment, it is natural to assume that the physical environment is a major factor for signal noise and fading, that will affect the mean RSS value. As proposed by Sheng, antenna diversity is also a contributor affecting the mean, where he reported that it could change 5dBm in 20% of the cases or 10dBm in 5% of the cases. How do this play out in the experiments?

It is evident that the radio environment is changing during tests that considerably affects the RSS mean in certain local time periods. By viewing the graphs

from the control experiments with two sensors, the sampled RSS mean is changing with means fluctuating a little under 10dBm, considered over the whole test period of 20 minutes.

As mentioned in 2.2, antenna diversity is used to mitigate multipath distortion issues. The author thinks that the changes in the graphs are a combination of antenna diversity and the changing environment, at least not diversity alone. If antenna diversity was the only factor for the fluctuations, it is highly unlikely that it was the main contributor in situations where one sensor looses the signal altogether, wile the other is unchanged. For that to happen the signal must have dropped almost 15 - 20 dBm during long time periods. Since the whole point of antenna diversity is to improve signaling conditions, it does not make sense that it would decrease the signal level that much for that amount of time.

Examples of this can be seen in control experiments 1 and 2. In control experiment 1 on page 51, between 400 and 500 seconds Northug loses captures altogether, while Bjorg have a huge decrease in frames captured but still indicates to be on the same RSS mean. In control experiment 2 on page 53, at around 780 to 860 seconds Bjorg's mean increases slightly, while Northug *loses* captures altogether. Around 900 to 1100 seconds the RSS mean at Northug suddenly gets a rising mean, with some larger differential RSS values, while Bjorg mean is unchanged.

There was however an example of the RSS mean changing 4-6dBm abruptly (and kept having a steady mean), indicating antenna diversity behavior. This can be seen in control experiment 1. There was a correlation between the two sensors, where one sensor loses signal strength while the other gains signal strength between 140 to 200 seconds.

Interference from other APs that transmits on the same channel is also a source of degradation of the signal. During a test (not shown here) for each of the sensors, the number of WLANs were counted to be more than 84 on both sensors. This was done by setting the sensors to monitor channel 1 for 20 minutes. The interference is time dependent, as it is based on how much the

AP is used.

The effects viewed in the results are mostly caused by environmental effects like people passing by or by larger vehicles like buses and trucks, that is seen frequently in Elgsetergate. One should also take note to the antenna diversity effect as mentioned in the work of Sheng et.al. [16].

7.1.3 IDS Infrastructure

The infrastructure used to capture frames was working as it should most of the time, although some problems were encountered, causing the system to loose frames. This is shown as gaps in the plots, especially noticable in the sequence number plots.

Reasons for these gaps were not found explicitly, but some suspicion is directed at the use of the SSH tunnel between the central logging server and the drones.

The gaps in the experiments are best illustrated in the single sensor experiments, as the gaps are more visible when there are more captured frames. As the experiments progressed, I noticed that the sensors were announcing error messages where it re-started their wireless interfaces. This could be caused by the SSH tunnel or by the drones themselves. If this setup is to be used again, it is important to take this into consideration.

7.2 Detection Method Results

Using the log parser program with the detection method produced results usable for detection. The method produced low counts of deviances (e.g. attack events) during control experiments, and frequent counts during attacks.

The metric in the method was to be counting the number of bins and the number of deviants inside each bin during a certain time period. In the experiments, the time period was as long as the experiment lasted.

Comparing the results from figures 6.20 through fig.6.23 we see the following:

- Attack 1 and 4 show some increase in frequency of the bins and a low count in each of them.
- Attack 2 and 3 show frequent bins with very high counts in attack 2 and some spikes in attack 3.

This suggests that even though an attack is present and essentially is the same attack at the same points, it produces different results for each of them. Viewing the RSS values for the experiments this is not that surprising. But it illustrates an important point; Even though the attacks are in the same area, this kind of detection method produces different results. Stability and reliability in a detection method is important to minimize false positives and negatives.

Even though the results were different from each experiment, comparing the results from attacks 5 and 6 on page 73 show that if the sensors get stable captures, it can work as a detection method. The challenge is to choose thresholds that work even though an attacker is far away, or when the interference is even worse than in these experiments.

8 Conclusion

In this work, RSS based detection methods has been researched, specifically in an urban environment. Previous work within the area has been conducted in office environments with promising results regarding RSS detection methods based on statistics.

A framework for processing captured 802.11 frames from a distributed sensor network has been made, and unique results from how RSS behaves in an urban environment has been gathered. Results has shown that because of the constantly changing radio environment caused by vehicles, buildings and competing APs around the sensors, it breaks the core assumption that the RSS distributions are Gaussian distributed, thereby making proposed statistical de-

tection methods unreliable.

The tested method that do not use a statistical model for attack detection was tested with the captured data. It must however be be finely tuned to work properly in specific situations, and can be cumbersome to use in highly fluctuating radio environments.

It can be argued that the problem can be overcome by introducing more sensors with higher sensitivity, placed at strategic locations. This is the prevailing method as shown in previous work done in office environments. However, the number of sensors in an urban environment is highly motivated by whether the operator has got proper locations with power and reporting infrastructure to house such equipment. The troubles of even being allowed to setup an antenna on a building is something mobile operators must cope with when setting up equipment, further limiting sensor numbers and placement.

The conclusion is that RSS based detection methods are difficult to use in unstable radio environments, and that they should be supplemented by other detection methods in an Wireless Intrusion Detection System.

8.1 Future Work

The wireless distributed sensory with Kismet has shown to be a success. By using more sensitive sensors, which also account for antenna diversity effects, it can serve as a prominent infrastructure for further IDS testing. Additional detection methods can be implemented using the made framework for the Kismet-Newcore program.

By acknowledging that usage of RSS based detection methods are difficult in urban environments, other forms of detection should be considered in addition. A combination of different methods in a specification based detection system is proposed in [5], and should be tested in an urban environment.

Bibliography

[1] IEEE, "IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," tech. rep., IEEE, 2007.

[2] D. B. Faria and D. R. Cheriton, "Detecting identity-based attacks in wireless networks using signalprints," in *WiSe '06: Proceedings of the 5th ACM workshop on Wireless security*, (New York, NY, USA), pp. 43–52, ACM, 2006.

[3] G. Chandrashekaran, J. A. Francisco, V. Ganapathy, M. Gruteser, and W. Trappe, "Detecting identity spoofs in 802.11e wireless networks," in *GC'09: Proceedings of the IEEE Globecom 2009 Communications and Information Security Symposium*, (Honolulu, Hawaii), IEEE Computer Society Press, Los Alamitos, California, USA, November/December 2009.

[4] Pedersen and Holgernes, "How to detect session hijacking in 802.11 open networks," tech. rep., NTNU, 2009.

[5] R. S. Gill, *Intrusion Detection Techniques in Wireless Local Area Networks*. PhD thesis, Queensland University of Technology, 2009.

[6] E. Holgernes, "Detecting Identity Thefts In Open 802.11e enabled Wireless Networks," Master's thesis, NTNU, 2010.

[7] "Web page trÃědlÃÿse trondheim." http://www.tradlosetrondheim.no/, 2010.

[8] "Multipath and Diversity." http://www.cisco.com/application/pdf/paws/27147/multipath.pdf, 2008.

[9] J. Bardwell, "Converting Signal Strength Percentage to dBm Values." http://www.wildpackets.com/elements/whitepapers/Converting_Signal_Strength.pdf, 2002.

[10] U. Deshpande, *A Dynamically Refocusable Sampling Infrastructure for 802.11 Networks*. PhD thesis, Dartmouth College, 2008.

[11] J. Smith, *Denial of Service: Prevention, Modelling and Detection*. PhD thesis, Queensland University of Technology, 2007.

[12] *http://netbsd.gw.com/cgi-bin/man-cgi?ieee80211_radiotap+9+NetBSD-current*.

[13] "Web page tcace technologies." http://www.cacetech.com/company/index.html, 2010.

[14] CACE Technologies, "Per-Packet Information Header Specification," tech. rep., CACE Technologies, 2009.

[15] Y. Chen, W. Trappe, and R. Martin, "Detecting and localizing wireless spoofing attacks," in *Proceedings of the Fourth Annual IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks*, pp. 193–202, Citeseer, 2007.

[16] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell, "Detecting 802.11 MAC layer spoofing using received signal strength," in *Proc. IEEE INFOCOM*, pp. 1768–76, 2008.

[17] D. McCoy, J. Franklin, P. Tabriz, J. Van Randwyk, and D. Sicker, "Passive Data-Link Layer 802.11 Wireless Device Driver Fingerprinting," in *Proceedings of the 15th conference on USENIX Security Symposium*, 2006.

[18] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, (New York, NY, USA), pp. 116–127, ACM, 2008.

[19] "Kismet-Newcore web page." `http://www.kismetwireless.net/`, 2010.

[20] D. Papini, "An Anomaly based Wireless Intrusion Detection System," Master's thesis, Kongens Lyngby, 2008.

[21] S. Haines and Thornton, *Kismet Hacking*. Syngress Publishing, Inc., 2008. Covers the old Kismet, not Kismet-Newcore.

[22] M. Saxena, P. Gupta, and B. N. Jain, "Experimental analysis of rssi-based location estimation in wireless sensor networks," 2008.

[23] E. Elnahrawy, X. Li, and R. Martin, "The limits of localization using signal strength: A comparative study," in *IEEE SECON*, vol. 2004, pp. 406–414, Citeseer, 2004.

[24] F. Guo and T. Chiueh, "Sequence number-based MAC address spoof detection," *Lecture Notes in Computer Science*, vol. 3858, p. 309, 2006.

[25] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *Computing*, vol. 2, p. 2, 2005.

[26] Q. Li and W. Trappe, "Relationship-based detection of spoofing-related anomalous traffic in ad hoc networks," *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, vol. 1, 2006.

[27] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," *Lecture Notes in Computer Science*, pp. 85–103, 2001.

[28] D. Madory, *New methods of spoof detection in 802.11 b wireless networking*. PhD thesis, Dartmouth College, 2006.

[29] *802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions*, 2003.

[30] "Wiki setup page for the sheevaplug." `http://plugcomputer.org/plugwiki/`, 2010.

[31] J. Edney and W. Arbaugh, *Real 802.11 security: Wi-Fi protected access and 802.11i*. Addison Wesley Publishing Company, 2004.

[32] K. Tao, J. Li, and S. Sampalli, "Detection of Spoofed MAC Addresses in 802.11 Wireless Networks," in *E-business and Telecommunications, Communications in Computer and Information Science, Volume 23. ISBN 978-3-540-88652-5.*, p. 201, Springer Berlin Heidelberg, 2009.

[33] R. Gill, J. Smith, M. Looi, and A. Clark, "Passive Techniques for Detecting Session Hijacking Attacks in IEEE 802.11 Wireless Networks," in *AusCERT Asia Pacific Information Technology Security Conference Refereed R&D Stream*, p. 26, 2002.

[34] R. Gill, J. Smith, and A. Clark, "Experiences in passively detecting session hijacking attacks in IEEE 802.11 networks," in *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*, pp. 221–230, Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 2006.

[35] "Deploying Cisco 440X Series Wireless LAN Controllers." `http://www.cisco.com/en/US/docs/wireless/technology/controller/deployment/guide/dep.html`, 2006.

[36] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols," tech. rep., 2005.

A

Appendix

A.1 Theory addendum

A.1.1 802.11

The bits in figure A.1 are shown in binary and the most significant bit (MSB) is **b7**.

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	0110–0111	Reserved
00	Management	1000	Beacon
00	Management	1001	ATIM
00	Management	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101	Action
00	Management	1110–1111	Reserved
01	Control	0000–0111	Reserved
01	Control	1000	Block Ack Request (BlockAckReq)
01	Control	1001	Block Ack (BlockAck)
01	Control	1010	PS-Poll
01	Control	1011	RTS
01	Control	1100	CTS
01	Control	1101	ACK
01	Control	1110	CF-End
01	Control	1111	CF-End + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF-Poll (no data)
10	Data	1000	QoS Data
10	Data	1001	QoS Data + CF-Ack
10	Data	1010	QoS Data + CF-Poll
10	Data	1011	QoS Data + CF-Ack + CF-Poll
10	Data	1100	QoS Null (no data)
10	Data	1101	Reserved
10	Data	1110	QoS CF-Poll (no data)
10	Data	1111	QoS CF-Ack + CF-Poll (no data)
11	Reserved	0000–1111	Reserved

Figure A.1: 802.11 type and subtype combinations from IEEE 802.11 [1].

A.1.2 Per-Packet information (PPI)

What is PPI? This is an explanation from the Kismet-Newcore documentation:

> The Per-Packet Information (PPI) Header is a general and extensible meta-information header format originally developed to provide 802.11n radio information, but can handle other information as well.
>
> By default Kismet will log the pcap file, gps log, alerts, and network log in XML and plaintext. By default, Kismet will try to log to pcapfiles using the PPI per-packet header. The PPI header is a well-documented header supported by Wireshark and other tools, which can contain spectrum data, radio data such as signal and noise levels, and GPS data. PPI is only available with recent libpcap versions. When it is not available, Kismet will fall back to standard 802.11 format with no extra headers.
>
> PPI is a header field that is added to every packet processed by the wireless interface. Based on the well-known Radiotap header in Linux, the header gives information that do not usually be described in a regular 802.11 frame. Added support for 802.11n radio networks as well enables it to be used in the future as well.

Why do I need PPI, or why do I use PPI?

While some drivers in linux support the Radiotap header, the use of PPI in Kismet is eqvivalent, plus the added benefit of handling 802.11n frames as well. Therefore, the perl script that analyzes the log files must take this format into account. How do i use it?

The only standard I seem to be able to dig up stems from the CACE Technologies websites. CACE Technologies has strong ties to the wireshark community and WinPCAP, as seen from their website [1].

[1] http://www.cacetech.com

RSSI	receive signal strength indicator
RX	receive or receiver
SGI	short guard interval
TSF	timing synchronization function
TSFT	timing synchronization function timer
UID	unique identifier
UTF	Unicode transformation format

1.4 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.5 References

Radiotap manual page: http://netbsd.gw.com/cgi-bin/man-cgi?ieee80211_radiotap+9+NetBSD-current

NTAR documentation: http://www.winpcap.org/ntar/

RFC 2119: http://www.ietf.org/rfc/rfc2119.txt

1.6 Overview

Section 2 provides a description of the PPI header, along with an explanation of its necessity. Section 3 defines the structure of the header, complete with C and C++-compatible data structures. Section 4 defines each data type. Data structures are assumed to be packed.

2. Overall Description

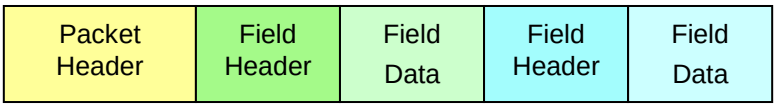
Existing header formats are typically made up of static data structures filled in by the capture mechanism and passed to user space. They suffer from the following problems:

- Limited scope. They are restricted to specific elements within a single domain.
- Rigidity. It is either impossible or very difficult to add new elements.
- Fixed DLTs. Each format only supports one encapsulated data link type.

PPI attempts to address each of these issues in a clean, consistent manner. Data elements are formatted as type-length-value (TLV) triplets, which allows for future expansion of the header while providing backward compatibility. Per-packet DLTs can be implemented by using an “empty” PPI header.

3. PPI Header Format

Each PPI packet header is made up of a packet header followed by zero or more fields. Each field is a type-length-value triplet.



PPI headers may contain only a packet header with no field header or field data elements. This removes any assumption or requirement that supplemental data exists for every packet captured. It also makes it possible to save packets with multiple data link types in a single capture file.

Multi-byte integers in the packet header and field headers MUST be stored as little-endian. The endian-ness of field data may be either big- or little-endian, and MUST be noted in the field description. The total length of the packet header plus all field headers and field data MUST be padded to a 32-bit boundary.

3.1 PPI Packet Header Structure

The PPI packet header provides a version, indicator flags, and the header length:

```
typedef struct ppi_packetheader {
    u_int8_t pph_version;     /* Version.  Currently 0 */
    u_int8_t pph_flags;       /* Flags.    */
    u_int16_t pph_len;        /* Length of entire message,
                               * including this header and TLV
                               * payload. */
    u_int32_t pph_dlt;        /* Data Link Type of the captured
                               * packet data. */
} ppi_packetheader_t;
```

3.1.1 pph_version

The version of the PPI header. MUST be set to zero (0).

3.1.2 pph_flags

An 8-bit mask that defines the behavior of the header. The following values are defined:

Bits (Bit 0 = LSB)	**Values**
0	Alignment. 32-bit aligned = 1, non-aligned = 0 Explained further in section 3.3
1-7	Reserved. MUST be 0.

3.1.3 pph_len

The length of the entire PPI header, including the packet header and fields. It MUST be between 8 and 65,532 inclusive.

3.1.4 pph_dlt

This MUST contain a valid data link type as defined in pcap-bpf.h from the libpcap distribution. If an official DLT registry is ever created by the libpcap development team, then it will supersede this list.

A capture facility can implement per-packet DLTs by setting pph_version to 0, pph_flags to 0, pph_len to 8, and pph_dlt to the DLT of the encapsulated packet.

3.2 PPI Field Structure

Each PPI field includes a type and length:

```
typedef struct ppi_fieldheader {
    u_int16_t pfh_type;       /* Type */
    u_int16_t pfh_datalen;    /* Length of data */
} ppi_fieldheader_t;
```

3.2.1 pfh_type

The type of data following the field header MUST be a valid type value as defined below:

Range	**Possible Values**
0-29,999	General-purpose field. Defined in section 4.
30,000-65,535	Vendor-specific fields. Defined in section 5.

If an unknown field value is encountered, it MUST be skipped according to the length rule in section 3.3. Implementations MAY mark it as “unknown” as appropriate.

3.2.2 pfh_datalen

The length of the data, in bytes, that follows MUST be between 0 and 65,520 inclusive. The end of the data MUST NOT exceed the total header length.

3.3 Field Processing

The first field header immediately follows the packet header (that is, if the packet header starts at byte 0, the first field header starts at byte 8). The starting point of each subsequent field header is defined by the "alignment" bit in pph_flags:

- If the "alignment" bit in pph_flags is set to 1 AND pfh_datalen in field *n* is not a multiple of 4, then field *n+1* will start at the next multiple of 4. For example, if pph_datalen in field 3 is 9, then the next three bytes MUST be considered padding, and field 4 will begin at byte 12.
- If the "alignment" bit in pph_flags is 0, then field *n+1* will start at the next byte offset following the data in field *n*.

The "alignment" bit in pph_flags also applies to field data. That is, if the "alignment" bit is set and the field type is 7 bytes long, then there will be one byte of padding between the field header and field data.

All padding bytes MUST be set to 0 in order to keep from exposing kernel memory to user space.

4. General-Purpose Field Types

The following general-purpose fields are currently defined. Further general-purpose fields will be defined in later revisions of this document. Vendor-specific fields may be defined externally.

Type	Length (Bytes)	Description
0-1		RESERVED
2	20	802.11-Common. Common (pre-n and .11n) radio information.
3	12	802.11n MAC Extensions. Extended (.11n) radio information.
4	48	802.11n MAC+PHY Extensions. Extended (.11n) radio information.
5	22-65,520	Spectrum-Map. Radio frequency spectrum information.
6	19-65,520	Process-Info. Process information, e.g. UID and GID
7	???	Capture-Info. Capture information, e.g. interface, drop counts, etc.
8	4	Aggregation Extension. Interface information for packets coming from aggregating interfaces.
9	8	802.3 Extension. Information regarding 802.3 (Ethernet) packets.
10 – 29,999	-	RESERVED

4.1 Field Descriptions

4.1.2 802.11-Common

Zero or one 802.11-Common fields may be present in a single header. All fields are little-endian.

The 802.11-Common field is loosely based on the existing Radiotap header format. It contains data common to both pre-n and 802.11n. Total length is 20 bytes.

Field Name	Semantics	Value Type	Length

TSF-Timer	7.3.1.10 and 11.1 of IEEE 802.11-1999 Invalid value = 0	Unsigned integer	8 bytes
Flags	Packet flags LSB = bit 0. Bits: Bit 0 = If set, FCS present Bit 1 = If set to 1, the TSF-timer is in ms, if set to 0 the TSF-timer is in us Bit 2 = If set, the FCS is not valid Bit 3 = If set, there was a PHY error receiving the packet. If this bit is set, Bit 2 is not relevant	Unsigned integer	2 bytes
Rate	Data rate in multiples of 500 Kbps Invalid value = 0x0000	Unsigned integer	2 bytes
Channel-Freq	Radiotap-formatted channel frequency, in MHz Invalid value = 0x0000	Unsigned integer	2 bytes
Channel-Flags	Radiotap-formatted channel flags: Bit 0-3 = Reserved Bit 4 = Turbo Bit 5 = Complementary Code Keying (CCK) Bit 6 = Orthogonal Frequency-Division Multiplexing (OFDM) Bit 7 = 2 GHz spectrum Bit 8 = 5 GHz spectrum Bit 9 = Only passive scan allowed Bit 10 = Dynamic CCK-OFDM Bit 11 = Gaussian Frequency Shift Keying (GFSK) (FHSS PHY) Bit 12-15 = Reserved	Unsigned integer	2 bytes
FHSS-Hopset	Radiotap-formatted Frequency-hopping spread spectrum (FHSS) hopset	Unsigned integer	1 byte
FHSS-Pattern	Radiotap-formatted Frequency-hopping spread spectrum (FHSS) pattern	Unsigned integer	1 byte
dBm-Antsignal	RF signal power at antenna Invalid value = -128	Signed integer	1 byte
dBm-Antnoise	RF noise at antenna Invalid value = -128	Signed integer	1 byte

Unlike Radiotap, these fields are packed without any padding or alignment.

A.2 Infrastructure

A.2.1 Sheevaplug Details

Follow the Howto from the openplug.org website, and build the newest kernel with the drivers we need.

As far as the author knows, there exists no previous work using a Sheevaplug with kismet-newcore drones.

The main beauty of using the Sheevaplug, is that all the tools available in Linux is also available to the plug. Apart from running anything that has been ported to the ARM branch of ubuntu linux, it can be customized in limitless ways by either compiling on the plug itself or cross-compile on a external computer for speed.

A.2.2 Kismet-Newcore Config Files

A.2.3 Kismet Server Configuration

```
# Kismet config file
# Most of the "static" configs have been moved to here -- the command line
# config was getting way too crowded and cryptic.  We want functionality,
# not continually reading --help!

# Version of Kismet config
version=2009-newcore

# Name of server (Purely for organizational purposes)
servername=Rybak_Nexus

# Prefix of where we log (as used in the logtemplate later)
logprefix=/home/rybak/kislog/

# Do we allow plugins to be used?  This will load plugins from the system
```

```
# and user plugin directiories when set to true (See the README for the default
# plugin locations).
allowplugins=false

# External Sheevadrones
ncsource=drone:name=Northug,host=127.0.0.1,port=2502
ncsource=drone:name=Bjorg,host=127.0.0.1,port=2503

# Comma-separated list of sources to enable.  This is only needed if you
    defined
# multiple sources and only want to enable some of them.  By default, all
    defined
# sources are enabled.
# For example, if sources with name=prismsource and name=ciscosource are
    defined,
# and you only want to enable those two:
# enablesources=prismsource,ciscosource

# Control which channels we like to spend more time on.  By default, the list
# of channels is pulled from the driver automatically.  By setting preferred
    channels,
# if they are present in the channel list, they'll be set with a timing delay
    so that
# more time is spent on them.  Since 1, 6, 11 are the common default channels,
    it makes
# sense to spend more time monitoring them.
# For finer control, see further down in the config for the channellist=
    directives.
#preferredchannels=1,6,11

# How many channels per second do we hop?  (1-10)
#channelvelocity=10

# By setting the dwell time for channel hopping we override the channelvelocity
# setting above and dwell on each channel for the given number of seconds.
#channeldwell=10

# Channels are defined as:
#channellist=Cropped:1,6
```

```
# or
# channellist=name:range-start-end-width-offset,ch,range,ch,...
#
# Channels may be a numeric channel or a frequency
#
# Channels may specify an additional wait period.  For common default channels,
# an additional wait period can be useful.  Wait periods delay for that number
# of times per second - so a configuration hopping 10 times per second with a
# channel of 6:3 would delay 3/10ths of a second on channel 6.
#
# Channel lists may have up to 256 channels and ranges (combined).  For power
# users scanning more than 256 channels with a single card, ranges must be used
    .
#
# Ranges are meant for "power users" who wish to define a very large number of
# channels.  A range may specify channels or frequencies, and will
    automatically
# sort themselves to cover channels in a non-overlapping fashion.  An example
# range for the normal 802.11b/g spectrum would be:
#
# range-1-11-3-1
#
# which indicates starting at 1, ending at 11, a channel width of 3 channels,
# incrementing by one.  A frequency based definition would be:
#
# range-2412-2462-22-5
#
# since 11g channels are 22 mhz wide and 5 mhz apart.
#
# Ranges have the flaw that they cannot be shared between sources in a non-
    overlapping
# way, so multiple sources using the same range may hop in lockstep with each
    other
# and duplicate the coverage.
#
# channellist=demo:1:3,6:3,11:3,range-5000-6000-20-10

# Default channel lists
# These channel lists MUST BE PRESENT for Kismet to work properly.  While it is
```

```
# possible to change these, it is not recommended.  These are used when the
    supported
# channel list can not be found for the source; to force using these instead of
# the detected supported channels, override with channellist= in the source
    defintion
#
# IN GENERAL, if you think you want to modify these, what you REALLY want to do
     is
# copy them and use channellist= in the packet source.
channellist=IEEE80211b:1:3,6:3,11:3,2,7,3,8,4,9,5,10
channellist=IEEE80211a:36,40,44,48,52,56,60,64,149,153,157,161,165
channellist=IEEE80211ab
    :1:3,6:3,11:3,2,7,3,8,4,9,5,10,36,40,44,48,52,56,60,64,149,153,157,161,165

# Client/server listen config
listen=tcp://0.0.0.0:2501
# People allowed to connect, comma seperated IP addresses or network/mask
# blocks.  Netmasks can be expressed as dotted quad (/255.255.255.0) or as
# numbers (/24)
allowedhosts=127.0.0.1,129.241.104.0/24,80.202.211.71/24
# Maximum number of concurrent GUI's
maxclients=5
# Maximum backlog before we start throwing out or killing clients.  The
# bigger this number, the more memory and the more power it will use.
maxbacklog=5000

# Server + Drone config options.  To have a Kismet server export live packets
# as if it were a drone, uncomment these.
# dronelisten=tcp://127.0.0.1:3501
# droneallowedhosts=127.0.0.1
# dronemaxclients=5
# droneringlen=65535

# OUI file, expected format 00:11:22<tab>manufname
# IEEE OUI file used to look up manufacturer info.  We default to the
# wireshark one since most people have that.
ouifile=/etc/manuf
ouifile=/usr/share/wireshark/wireshark/manuf
ouifile=/usr/share/wireshark/manuf
```

```
# Do we have a GPS?
gps=true
# Do we use a locally serial attached GPS, or use a gpsd server?
# (Pick only one)
gpstype=gpsd
# gpstype=serial
# What serial device do we look for the GPS on?
gpsdevice=/dev/rfcomm0
# Host:port that GPSD is running on.  This can be localhost OR remote!
gpshost=localhost:2947
# Do we lock the mode?  This overrides coordinates of lock "0", which will
# generate some bad information until you get a GPS lock, but it will
# fix problems with GPS units with broken NMEA that report lock 0
gpsmodelock=false
# Do we try to reconnect if we lose our link to the GPS, or do we just
# let it die and be disabled?
gpsreconnect=true

# Do we export packets over tun/tap virtual interfaces?
tuntap_export=false
# What virtual interface do we use
tuntap_device=kistap0

# Packet filtering options:
# filter_tracker - Packets filtered from the tracker are not processed or
#                  recorded in any way.
# filter_export  - Controls what packets influence the exported CSV, network,
#                  xml, gps, etc files.
# All filtering options take arguments containing the type of address and
# addresses to be filtered.  Valid address types are 'ANY', 'BSSID',
# 'SOURCE', and 'DEST'.  Filtering can be inverted by the use of '!' before
# the address.  For example,
# filter_tracker=ANY(!"00:00:DE:AD:BE:EF")
# has the same effect as the previous mac_filter config file option.


#filter_export=BSSID(00:23:5D:00:00:00/FF:FF:FF:00:00:00)
#filter_export=ANY(00:23:5D:00:00:00/FF:FF:FF:00:00:00)
```

```
#filter_tracker=BSSID(00:23:5D:00:00:00/FF:FF:FF:00:00:00)
#ilter_export=BSSID(00:23:5D:00:00:00/FF:FF:FF:00:00:00)
#filter_dump=BSSID(00:23:5D:00:00:00/00:23:5D:FF:00:00)
#
# Kun 0022fb895a36
#filter_dump=BSSID(00:23:5D:0E:00:31)
#filter_export=BSSID(00:23:5D:0E:00:31)
filter_tracker=BSSID(00:23:5D:0E:00:31)



#filter_dump=SOURCE(!00:19:A9:06:1C:00)
#filter_export=BSSID(00:23:5D:00:00:00/FF:FF:FF:00:00:00)
#filter_tracker=BSSID(AA:BB:CC:00:00:00/FF:FF:FF:00:00:00)


# filter_dump=...
# filter_export=...
#filter_netclient=SOURCE(!00:19:A9:06:1C:00)

# Alerts to be reported and the throttling rates.
# alert=name,throttle/unit,burst
# The throttle/unit describes the number of alerts of this type that are
# sent per time unit.  Valid time units are second, minute, hour, and day.
# Burst describes the number of alerts sent before throttling takes place.
# For example:
# alert=FOO,10/min,5
# Would allow 5 alerts through before throttling is enabled, and will then
# limit the number of alerts to 10 per minute.
# A throttle rate of 0 disables throttling of the alert.
# See the README for a list of alert types.
alert=ADHOCCONFLICT,5/min,1/sec
alert=AIRJACKSSID,5/min,1/sec
alert=APSPOOF,10/min,1/sec
alert=BCASTDISCON,5/min,2/sec
alert=BSSTIMESTAMP,5/min,1/sec
alert=CHANCHANGE,5/min,1/sec
alert=CRYPTODROP,5/min,1/sec
alert=DISASSOCTRAFFIC,10/min,1/sec
```

```
alert=DEAUTHFLOOD,5/min,2/sec
alert=DEAUTHCODEINVALID,5/min,1/sec
alert=DISCONCODEINVALID,5/min,1/sec
alert=DHCPNAMECHANGE,5/min,1/sec
alert=DHCPOSCHANGE,5/min,1/sec
alert=DHCPCLIENTID,5/min,1/sec
alert=DHCPCONFLICT,10/min,1/sec
alert=NETSTUMBLER,5/min,1/sec
alert=LUCENTTEST,5/min,1/sec
alert=LONGSSID,5/min,1/sec
alert=MSFBCOMSSID,5/min,1/sec
alert=MSFDLINKRATE,5/min,1/sec
alert=MSFNETGEARBEACON,5/min,1/sec
alert=NULLPROBERESP,5/min,1/sec
#alert=PROBENOJOIN,5/min,1/sec

# Controls behavior of the APSPOOF alert.  SSID may be a literal match (ssid=)
    or
# a regex (ssidregex=) if PCRE was available when kismet was built.  The
    allowed
# MAC list must be comma-separated and enclosed in quotes if there are multiple
# MAC addresses allowed.  MAC address masks are allowed.
apspoof=Foo1:ssidregex="(?i:foobar)",validmacs=00:11:22:33:44:55
apspoof=Foo2:ssid="Foobar",validmacs="00:11:22:33:44:55,aa:bb:cc:dd:ee:ff"

# Known WEP keys to decrypt, bssid,hexkey.  This is only for networks where
# the keys are already known, and it may impact throughput on slower hardware.
# Multiple wepkey lines may be used for multiple BSSIDs.
# wepkey=00:DE:AD:C0:DE:00,FEEDFACEDEADBEEF01020304050607080900

# Is transmission of the keys to the client allowed?  This may be a security
# risk for some.  If you disable this, you will not be able to query keys from
# a client.
allowkeytransmit=true

# How often (in seconds) do we write all our data files (0 to disable)
writeinterval=300

# Do we use sound?
```

```
# Not to be confused with GUI sound parameter, this controls wether or not the
# server itself will play sound.  Primarily for headless or automated systems.
enablesound=false
# Path to sound player
soundbin=play

sound=newnet,true
sound=newcryptnet,true
sound=packet,true
sound=gpslock,true
sound=gpslost,true
sound=alert,true

# Does the server have speech? (Again, not to be confused with the GUI's speech
    )
enablespeech=false
# Binary used for speech (if not in path, full path must be specified)
speechbin=flite
# Specify raw or festival; Flite (and anything else that doesn't need
    formatting
# around the string to speak) is 'raw', festival requires the string be wrapped
     in
# SayText("...")
speechtype=raw

# How do we speak?  Valid options:
# speech    Normal speech
# nato      NATO spellings (alpha, bravo, charlie)
# spell     Spell the letters out (aye, bee, sea)
speechencoding=nato

speech=new,"New network detected s.s.i.d. %1 channel %2"
speech=alert,"Alert %1"
speech=gpslost,"G.P.S. signal lost"
speech=gpslock,"G.P.S. signal O.K."

# How many alerts do we backlog for new clients?  Only change this if you have
# a -very- low memory system and need those extra bytes, or if you have a high
# memory system and a huge number of alert conditions.
```

```
alertbacklog=50

# File types to log, comma seperated.  Built-in log file types:
# alert    Text file of alerts
# gpsxml   XML per-packet GPS log
# nettxt   Networks in text format
# netxml   Networks in XML format
# pcapdump   tcpdump/wireshark compatible pcap log file
# string   All strings seen (increases CPU load)
#logtypes=pcapdump,gpsxml
logtypes=pcapdump,gpsxml,netxml,nettxt,alert

# Format of the pcap dump (PPI or 80211)
pcapdumpformat=ppi
# pcapdumpformat=80211

# Default log title
logdefault=Rybak_Nexus

# logtemplate - Filename logging template.
# This is, at first glance, really nasty and ugly, but you'll hardly ever
# have to touch it so don't complain too much.
#
# %p is replaced by the logging prefix + '/'
# %n is replaced by the logging instance name
# %d is replaced by the starting date as Mon-DD-YYYY
# %D is replaced by the current date as YYYYMMDD
# %t is replaced by the starting time as HH-MM-SS
# %i is replaced by the increment log in the case of multiple logs
# %l is replaced by the log type (pcapdump, strings, etc)
# %h is replaced by the home directory

logtemplate=%p%n-%D-%t-%i.%l

# Where state info, etc, is stored.  You shouldnt ever need to change this.
# This is a directory.
configdir=%h/.kismet/
```

Kismet Drone Configuration

```
# Common Kismet drone config file for sensors
# Northug and Bjorg.

version=newcore.1
servername=[Northug1 | Bjorg1]
dronelisten=tcp://127.0.0.1:[2502 | 2503]
droneallowedhosts=127.0.0.1,129.241.104.0/24,129.241.0.0/16
dronemaxclients=10
droneringlen=65535
gps=true
gpstype=gpsd
gpshost=localhost:2947
gpsmodelock=false
gpsreconnect=true
ncsource=wlan0:name=NortSense,hop=false,channel=1
channellist=IEEE80211b:1:3,6:3,11:3,2,7,3,8,4,9,5,10
channellist=IEEE80211a:36,40,44,48,52,56,60,64,149,153,157,161,165
channellist=IEEE80211ab
     :1:3,6:3,11:3,2,7,3,8,4,9,5,10,36,40,44,48,52,56,60,64,149,153,157,161,165
```

A.3 Experiments

A.3.1 Setup of stations

A.3.2 Station setup

Forcing client STA to ONE BSSID and one frequency.

Victim laptop was a Dell L2100 netbook with MAC 00:22:fb:89:5a:36

```
iwconfig wlan0 mode managed essid "WirelessTrondheim" ap 00:23:5D:0E:00:31 freq
     2.412G key off
```

A.3.3 Attacker setup

Desc: Forcing ATK to the same channel as sensors and STA

Attacking laptop was a Dell D630 with MAC 00:1F:3C:2E:6A:E3

```
# 1. Find STA MAC address
Find it by sniffing with tshark or the like, I used the MAC address of the DELL
      L2100 above.

# 2. Setup and clone STAS address.
ifconfig wlan0 down
macchanger --mac=00:22:fb:89:5a:36 wlan0
ifconfig wlan0 up
iwconfig wlan0 mode managed ap 00:23:5D:0E:00:31 key off essid "
     WirelessTrondheim"

# Set IP address as the same as STA:
ifconfig wlan0 10.100.0.180 netmask 255.255.254.0 up

# Set correct routing:
route add default gw 10.100.0.1

# 3. Fix firewall with IP tables.
iptables -A INPUT -i wlan0 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i wlan0 -p tcp -j DROP
```

A.4 Code

A.4.1 Framework Code

```
#!/usr/bin/perl -w
# Parser program for Kismet-ng .pcap files and .gpsxml files
# Input to program : rss.pl <kismetlogfile.pcap> <kismetlogfile.gpsxml>
use Net::Pcap;
use strict;
```

```
use warnings;
use DateTime;
use File::Basename;
use Cwd;
use warnings FATAL => qw(uninitialized);
my ( @gpsdata, $CAPFILE );

# Init counters - all set to 0
my (
  $count_hashed,      $counter_manag, $counter_gps_match,
  $counter_gps_total, $counter_total, $counter_manag_response,
  $counter_data,      $count_control, $counter_seek
) = ('0') x 9;
my $lower         = time();
my $upper         = time();
my %hashmap       = ();         # Key = sourceMACs
my $edist_counter = 0;          # Number of euclidian computations
my $SNAPLEN       = 56;         # Number of bytes to capture from each packet -
     PPI = 32 + 24
my $PROMISC       = 1;          # Sets interface in promiscous mode -
my $TIMEOUT       = 2000;       # specifies a read time-out for packet
    capturing in msec.
my $file          = $ARGV[0];
my ( $capfilename, $WORKDIR ) = fileparse($file);
print "Working directory:  $WORKDIR \n";

unless ( $CAPFILE = $ARGV[0] ) {
  die "Capture pcap file error: " . $!;
}
unless ( open FILE, $ARGV[1] ) {
  die "gpsxml file opening error." . $!;
}

# Calls gps log file parser.
unless ( gps_parse_xml() ) {
  die "GPS generation failed:" . $!;
}

#my $level = -1;    # Level of indentation
```

```
sub main () {
  loopit_offline();

  # Results
  hash_print_R();

  #ids('0022fb895a36');
  #hash_print_contents();
  #status();
}
main();

# Per MAC and Sensor IDS check.
sub ids {
  my ( $mac, $sensor, $thres_sec, $thres_dBm ) = @_;
  my ( $count_time, $start_time, $stop_time, $rss_delta, $frmcount, @diffarray,
       @deviation );

  # for my $sensor ( keys %{ $hashmap{$mac} } ) {
  my $framecnt = $#{ $hashmap{$mac}{$sensor}[0] };
  $start_time = ${ $hashmap{$mac}{$sensor} }[1]->[0];
  $stop_time  = $start_time + $thres_sec;

  #print "$start_time  $stop_time ".$start_time - $stop_time . "\n";
  for my $i ( 0 .. $framecnt ) {
    $rss_delta = &get_rss_delta( $mac, $sensor, $i );
    push @diffarray, $rss_delta;
    $count_time = ${ $hashmap{$mac}{$sensor} }[1]->[$i];
    if ( $count_time > $stop_time ) {
      push @deviation, ( &detect( \@diffarray, $thres_dBm ) );
      $start_time = ${ $hashmap{$mac}{$sensor} }[1]->[$i];
      $stop_time  = $start_time + $thres_sec;
      @diffarray  = ();
    } else {
    }
  }
  return \@deviation;

  # }
```

```
}

#calculates number of deviations from given threshold. Returns number of
    detected.
sub detect {
  my ( $values, $thres_dBm ) = @_;
  my $detect = 0;
  foreach my $rss ( @{$values} ) {
    if ( $rss > $thres_dBm or $rss < -$thres_dBm ) {
      $detect = $detect + 1;
    }
  }

  #print "Returning nubmer of detected: $detect\n";
  return $detect;
}

# Extracts given MAC and executes R for graphing.
sub hash_print_R () {
  my $thres_sec = 10;    # Time in seconds for detection interval
  my $thres_dBm = 6;    # RSS delta threshold in dBm
  my ( $frmcount, $deviation, $key, $DATADIR );
  $DATADIR = $WORKDIR . "data";
  mkdir($DATADIR)  or                                          #;
    chdir $DATADIR or die "Can't cd to working directory: $!\n";
  print "Creating data for R from file " . "$ARGV[1]... creating files under $
      DATADIR...\n";
  $key = '0022fb895a36';    # Remove for handling all mac adresses in capture.

  # for my $key ( keys %hashmap ) {
  for my $sensor ( keys %{ $hashmap{$key} } ) {
    $frmcount = $#{ $hashmap{$key}{$sensor}[0] };
    if ( $frmcount < 300 )  {# Skip MACs with fewer frames than 300.
    } else {                # creates log files under ./data
      chdir $DATADIR or     #;
        print cwd() . " Working on file " . $DATADIR . "/" . $key . "_" . $
            sensor . '.data' . "\n";
      open( FILE, ( '>' . $key . "_" . $sensor . '.data' ) )
        or die $!;
```

```
      open( IDS, ( '>' . $key . "_" . $sensor . '.ids' ) ) or die $!;
      $deviation = ids( $key, $sensor, $thres_sec, $thres_dBm );
      print IDS "deviation\n";
      foreach my $deviated ( @{$deviation} ) {
        print IDS "$deviated\n";
      }
      close(IDS);
      my ($rss_delta) = 0;
      print FILE "type time seq rss rate freq retry rss_delta \n";
      for my $i ( 0 .. $frmcount ) {
        $rss_delta = &get_rss_delta( $key, $sensor, $i );
        my $time = ${ $hashmap{$key}{$sensor} }[1]->[$i] - $lower;    # Counts
             seconds from the start.
        print FILE ${ $hashmap{$key}{$sensor} }[0]->[$i] . " ";
        print FILE $time . " ";
        print FILE ${ $hashmap{$key}{$sensor} }[2]->[$i] . " ";
        print FILE ${ $hashmap{$key}{$sensor} }[3]->[$i] . " ";
        print FILE ${ $hashmap{$key}{$sensor} }[4]->[$i] . " ";
        print FILE ${ $hashmap{$key}{$sensor} }[5]->[$i] . " ";
        print FILE ${ $hashmap{$key}{$sensor} }[6]->[$i] . " ";
        print FILE $rss_delta . "\n";
      }
      close(FILE);
    }
  }

  #}
  print "Executing R for plots...\n";
  system('R --slave --no-save < /home/peders/widscode/rplot_northug.r');
}

# For external capture files
sub get_offline_capture {
  my ($capturefile) = @_;
  my $offline_err;
  my $pcap = Net::Pcap::open_offline( $capturefile, \$offline_err )
    || die " Can't read the file '$capturefile' : $offline_err \n ";
  unless ( defined $pcap ) {
    die 'could not instanciate offline capture on file';
```

```
  }
  return $pcap;
}

# Offline Monitor
sub loopit_offline {
  my $capture  = get_offline_capture($CAPFILE);
  my $LINKTYPE = Net::Pcap::datalink_name_to_val('DLT_IEEE802_11_RADIO');
  my $datatype = Net::Pcap::datalink($capture);
  my ( $key, $value, $num_keys, );

  # Associates correct linktype to the capture device.
  Net::Pcap::set_datalink( $capture, $LINKTYPE );
  Net::Pcap::loop( $capture, -1, \&packetgrinder, '' );

  # Cleaning up
  Net::Pcap::close($capture);
}

# Prints status after script is finished
sub status {
  print "\n";
  print "Duration: " . ( ( $upper - $lower ) / 60 ) . " min \n";

  #print "Lower time limit: $lower | Upper time limit: $upper. \n";
  print "Sum Manag+Data+Contr pkts: " . ( $counter_manag + $counter_data + $
       count_control ) . "\n";

  # print "Missed GPS entries: "
  #   . gps_print_array()
  #   . " (after popping or deleting gps array) \n";
  print "Management pkts: "
    . $counter_manag / $counter_total . " ("
    . $counter_manag . "/"
    . $counter_total . ")\n";

  #print "B. response ratio: " . $counter_manag_response . "/" . $counter_total
        . "\n";
  print "Data pkts: " . $counter_data / $counter_total . " (" . $counter_data .
```

```
        "/" . $counter_total . ")\n";
  print "Control (ACKs) pkts: "
    . $count_control / $counter_total . " ("
    . $count_control . "/"
    . $counter_total
    . " - will be discarded) \n";
  print "GPS matches: " . $counter_gps_match . "/" . $counter_total . "\n";
  print "Hashtable filled: " . $count_hashed . "/" . $counter_total . "\n";
  print "\n######\n ";
  gps_print_array();
}

# Parses Kismet-ng log.gpsxml into array. Strips zero-sources.
sub gps_parse_xml {
  print "Parsing GPS file: " . $ARGV[1] . "\n";

  # Removes unwanted lines in file:
  # Removing unknown sources: ( source="00:00:00:00:00:00"  )
  my @result =
    grep { /<gps-point/ && !/GP:SD:TR:AC:KL:OG/ && !/source="00:00:00:00:00:00"
        / } <FILE>;
  foreach my $line (@result) {
    my (
      $f1,  $f2,  $f3,  $f4,  $f5,  $bssid, $source, $tsec, $tusec,
      $lat, $lon, $f12, $f13, $f14, $f15,   $rssi,   $f17
    ) = split( / /, $line );

    # This split can be done better with REGEXP.
    my ( $b1,  $b2 )  = split( /"/, $bssid );
    my ( $s1,  $s2 )  = split( /"/, $source );
    my ( $t1,  $t2 )  = split( /"/, $tsec );
    my ( $tu1, $tu2 ) = split( /"/, $tusec );
    my ( $l1,  $l2 )  = split( /"/, $lat );
    my ( $lo1, $lo2 ) = split( /"/, $lon );
    my ( $r1,  $r2 )  = split( /"/, $rssi );

    # Removes colon and makes lower-case:
    $b2 =~ s/://g;
    $b2 =~ tr/A-Z/a-z/;
```

```
    $s2 =~ s/://g;
    $s2 =~ tr/A-Z/a-z/;

    # Sets lower and upper time interval.
    if ( $t2 <= $lower ) {
      $lower = $t2;
    } else {
      $upper = $t2;
    }

    # Pushes line to array
    push( @gpsdata, ( [ $b2, $s2, $t2, $tu2, $l2, $lo2, $r2 ] ) );
  }
  undef @result;
  close FILE;
}

# Prints contents in finished array
sub gps_print_array {
  my $counter = 0;
  foreach my $id (@gpsdata) {
    for my $i ( 0 .. 6 ) {
      $counter++;
      print $id->[$i] . "\n";
    }
  }
  if ( $counter != 0 ) {
    return print "Left: " . ( $counter / 7 ) . "\n";
  }
  print "GPS array is empty. \n";
}

# Returns GPS position if found in parsed GPSxml file.
sub gps_get_pos {
  $counter_gps_total++;
  my ( $macaddr, $tstamp_sec, $tstamp_usec, $rss_dbm ) = @_;
  $counter_seek = 0;
  my $frmcount = $#gpsdata;    # antall linjer som sjekkes
  for my $i ( 0 .. $frmcount ) {
```

```
    $counter_seek++;
    my $bssid     = $gpsdata[$i]->[0];
    my $source    = $gpsdata[$i]->[1];
    my $time_sec  = $gpsdata[$i]->[2];
    my $time_usec = $gpsdata[$i]->[3];
    my $lat       = $gpsdata[$i]->[4];
    my $lon       = $gpsdata[$i]->[5];
    my $rssi      = $gpsdata[$i]->[6];

    if ( $time_sec == $tstamp_sec
      && $time_usec == $tstamp_usec
      && $macaddr eq $source
      && $rssi eq $rss_dbm )
    {
      $counter_gps_match++;
      if ( $i < 1 ) {
        shift(@gpsdata);
        return ( "$lat", "$lon" );
      }
      splice( @gpsdata, $i, 1 );
      return ( "$lat", "$lon" );
    } elsif ( $counter_seek > 200 ) {
      return ( "0", "0" );    # Did not find any match;
    }
  }
  return ( "0", "0" );        # Did not find any match
}

sub packetgrinder {
  $counter_total++;
  my ( $user_data, $header, $packet ) = @_;
  my (
    $tstamp_sec, $tstamp_usec, $type,  $subtype, $rss_dbm,   $sequence, $
        macaddr,
    $lat,        $lon,         $flags, $tofroDS, $data_rate, $channel,  $retry
  );
  $tstamp_sec  = ${$header}{tv_sec};
  $tstamp_usec = ${$header}{tv_usec};
  $lat         = 1;
```

```
  $lon         = 1;
  $data_rate   = hex( unpack( 'H2', substr( $packet, 22 ) ) ) * 0.5;
  $channel     = hex( ( unpack( 'H2', substr( $packet, 25 ) ) ) . ( unpack( 'H2
      ', substr( $packet, 24 ) ) ) );
  $rss_dbm     = hex( unpack( 'H*', substr( $packet, 30, 1 ) ) ) - 256;
  ( $subtype, $type ) = split( //, unpack( 'H*', substr( $packet, 32, 1 ) ) );
  $flags = unpack( 'B*', substr( $packet, 33, 1 ) );
  $retry = substr( $flags, 4, 1 );
  $tofroDS = unpack( 'b2', substr( $packet, 33, 1 ) );

  if ( $type eq '4' ) {    # Control frame - Will be discarded
                           # Getting GPS posisjon of sensor - DEPRECATED
                           # All zero sources are removed from gps log file.
    $count_control++;
    $macaddr = '000000000000';

    # Setting sequence number for gps search
    $sequence = 'ffff';
    return;
  } elsif ( $type eq '0' ) {    # MANAG frame
    $counter_manag++;

# according to the standard, all management frames have the same structure,
     source MAC is always on the same spot.
# Source MAC is always in adrfield 2
    $macaddr = unpack( 'H*', substr( $packet, 42, 6 ) );

    #my @pos = gps_get_pos( $macaddr, $tstamp_sec, $tstamp_usec, $rss_dbm );
    #return;
  } elsif ( $type eq '8' ) {    # DATA frame
    $counter_data++;

    #print $tofroDS."\n";
    if ( ( $tofroDS eq '00' ) or ( $tofroDS eq '10' ) ) {

      #from client to DS, Adrfield 2
      $macaddr = unpack( 'H*', substr( $packet, 42, 6 ) );
    } elsif ( $tofroDS eq '01' ) {
```

```
      #from DS to client, Adrfield 3
      $macaddr = unpack( 'H*', substr( $packet, 48, 6 ) );
    } else {

      #Between DS's, adrfield 4
      # By some reason, cannot check correct timing in these packets with gps
          log. Do they exist at all?
      #$macaddr = unpack( 'H*', substr( $packet, 56, 6 ) );
      return;
    }
  } else {
    return;
  }    # End of Type check
  $sequence = unpack( 'H*', substr( $packet, 54, 2 ) );
  my $seq = hex( ( substr( $sequence, 2, 2 ) . substr( $sequence, 0, 1 ) ) );

  # Getting GPS posisjon of sensor's packet
  my @pos = gps_get_pos( $macaddr, $tstamp_sec, $tstamp_usec, $rss_dbm );
  $lat = $pos[0];
  $lon = $pos[1];
  if ( $lat == "0" && $lon == "0" ) {
    return;
  }
  addit(
    $macaddr, ( $type . $subtype ),
    $tstamp_sec, $tstamp_usec, $seq, $rss_dbm, ( $lat . "+" . $lon ),
    $data_rate, $channel, $retry
  );
}

# Pushes values recored for given MAC address to array within hash table.
sub addit {
  $count_hashed++;
  my ( $key, $type, $tstamp_sec, $tstamp_usec, $sequence, $rss_dbm, $sensor, $
      data_rate, $channel, $retry ) =
    @_;

  #my $dt = DateTime->from_epoch( epoch => $tstamp_sec . "." . $tstamp_usec );
  my $dt = ( $tstamp_sec . "." . $tstamp_usec );
```

```
  push( @{ $hashmap{$key}{$sensor}[0] }, $type );
  push( @{ $hashmap{$key}{$sensor}[1] }, $dt );
  push( @{ $hashmap{$key}{$sensor}[2] }, $sequence );
  push( @{ $hashmap{$key}{$sensor}[3] }, $rss_dbm );
  push( @{ $hashmap{$key}{$sensor}[4] }, $data_rate );
  push( @{ $hashmap{$key}{$sensor}[5] }, $channel );
  push( @{ $hashmap{$key}{$sensor}[6] }, $retry );

  #print_last_added($key, $sensor);
}

sub print_last_added {
  my ( $key, $sensor ) = @_;
  print
"Added! Source: $key @ sensor $sensor |Type: ${ $hashmap{$key}{$sensor}
    }[0]->[-1] | Timestamp: ${ $hashmap{$key}{$sensor} }[1]->[-1].${ $hashmap{
    $key}{$sensor} }[2]->[-1] | Sequence: ${ $hashmap{$key}{$sensor}
    }[3]->[-1] | RSS: ${ $hashmap{$key}{$sensor} }[4]->[-1] \n";
}

sub get_rss_delta {
  my ( $mac, $sensor, $index ) = @_;
  if ( $index == 0 ) {
    return my $rss_delta = 0;
  } else {
    my $rss_delta =
      ${ $hashmap{$mac}{$sensor} }[4]->[$index] - ${ $hashmap{$mac}{$sensor}
          }[4]->[ $index - 1 ];
    return $rss_delta;
  }
}
```

A.4.2 R Code

```
#!/usr/bin/R --no-save

require("gplots")
#library(psych)
library(pastecs)
```

```
#library(Hmisc)

## Read capture data into variable
bjorg <- read.table("./0022fb895a36_11.185184+222.368713.data", header=TRUE)
northug <- read.table("./0022fb895a36_48.117298+11.516666.data", header=TRUE)


#bjorg.l <- bjorg[bjorg$time < foo]
#bjorg.l <- bjorg.l[bjorg.l$time > bar]

## Read IDS data into variable
bjorgids <- read.table("./0022fb895a36_11.185184+222.368713.ids", header=TRUE)
northugids <- read.table("./0022fb895a36_48.117298+11.516666.ids", header=TRUE)

# Writing descr. stats to file us ing Pastecs (Package for Analysis of Space-
    Time Ecological Series)
bjorgrss_delta_desc <- stat.desc(bjorg$rss_delta, basic=TRUE, desc=TRUE)
bjorgrssdesc <- stat.desc(bjorg$rss, basic=TRUE, desc=TRUE)
nordrss_delta_desc <- stat.desc(northug$rss_delta,  basic=TRUE, desc=TRUE)
nordrssdesc <- stat.desc(northug$rss,  basic=TRUE, desc=TRUE)

## Write statistics to file
write.table(c(bjorgrssdesc), "./bjorgrssdesc.sum",  sep="\t")
write.table(c(bjorgrss_delta_desc), "./bjorgrss_delta_desc.sum",  sep="\t")
write.table(c(nordrssdesc), "./nordrssdesc.sum", sep="\t")
write.table(c(nordrss_delta_desc), "./nordrss_delta_desc.sum", sep="\t")


### Generates PDF with the following content:
# 1 - RSS_delta, RSS and seqn plot sensor 1
# 2 - RSS_delta, RSS and seqn plot sensor 2
# 3 - IDS graphs
# 4 - Density distributions of RSS values
# 5 - Normal Q-Q plot

postscript("",command=" ps2pdf - result.pdf ")

## Summary Bjorg sensor
par(mfcol=c(3,1), cex.main=2, cex.lab=1.5)
```

```
# Power deltas
plot(bjorg$time,bjorg$rss_delta, main="Differential RSS values - Sensor Bjorg",
      xlab="Time in seconds", ylab="dBm", type="p", pch="x", col="red")
#bandplot (bjorg$time, bjorg$rss_delta, add=TRUE, sd = c(), sd.col = c("blue"))
# Power samples
plot(bjorg$time,bjorg$rss, main="RSS samples - Sensor Bjorg", xlab="Time in
     seconds", ylab="dBm", type="p", pch="x",  ylim=c(-90,-55), col="red")
# Mean graph of power samples
bandplot (bjorg$time,bjorg$rss, add=TRUE, sd = c(0), sd.col = c("blue"),method
     = "range", width, n=15)
# Sequence numbers
plot(bjorg$time,bjorg$seq, main="Sequence numbers - Sensor Bjorg", ylab="
     Sequence number", xlab="Time in seconds",  type="p", pch=16, col="blue")

## Summary Northug sensor
par(mfcol=c(3,1))
# Power deltas
plot(northug$time,northug$rss_delta, main="Differential RSS values - Sensor
     Northug", xlab="Time in seconds", ylab="dBm", type="p", pch="x", col="red"
     )
#bandplot (northug$time,northug$rss, add=TRUE, sd = c(1), sd.col = c("blue"))
#Power samples
plot(northug$time,northug$rss, main="RSS samples - Sensor Northug", xlab="Time
     in seconds", ylab="dBm", type="p", pch="x",  ylim=c(-90,-55), col="red")
# Mean graph of power samples
bandplot (northug$time,northug$rss, add=TRUE, sd = c(0), sd.col = c("blue"),
     method = "range", width, n=15)
# Sequence numbers
plot(northug$time,northug$seq,  main="Sequence number - Sensor Northug", ylab="
     Sequence numbers", xlab="Time in seconds",  type="p", pch=16, col="blue")

## IDS graphs for both sensors
par(mfcol=c(2,1))
# Power Deviation
plot(bjorgids$deviation, main="IDS data Sensor Bjorg", xlab="Time in 10 sec
     bins", ylab="Counts over 15 dBm", type="h", pch="x", col="red", ylim=c
     (0,100))
plot(northugids$deviation, main="IDS data Sensor Northug", xlab="Time in 10 sec
      bins", ylab="Counts over 15 dBm", type="h", pch="x", col="red", ylim=c
```

```
     (0,100))

## Density plot
par(mfcol=c(2,1))
hist(bjorg$rss ,main="Density plot Sensor Bjorg", xlab="dBm",col="red")
abline(bjorg$rss, bjorg$time)
qqnorm(bjorg$rss);qqline(bjorg$rss)
par(mfcol=c(2,1))
hist(northug$rss,main="Density plot Sensor Northug", xlab="dBm",col="red")
qqnorm(northug$rss);qqline(northug$rss)

dev.off()
```