An Anomaly based Wireless Intrusion Detection System

Davide Papini

Kongens Lyngby 2008
IMM-MSc-2008-110

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Contents

Chapter 1

Introduction

In the last decade we have witnessed the birth and the growth of a technology that has very much changed the way we work and live: the IEEE 802.11 also known as Wi-Fi. There are many features that made this technology very popular, some of them are: immediate and seamless connectivity without wires, faster network deployment, scalability easier than wired networks, dynamic environment. Unfortunately deployment and fast growth did not always match the need for security thus leaving many issues unsolved or not addressed. Although the IEEE 802.11 standard has been modified through the years and extended to include stronger cryptographic mechanisms and security policies, many threats are still there, some of them very severe. These threats are very difficult to mitigate since they are enabled by protocol basics which for the moment can not be changed due to legacy devices support. Moreover dealing with radio waves is not the same as dealing with wires. Radio waves spread out to uncontrolled areas and are difficult to contain thus environment control is not feasible. Hence the demand for a technology to monitor malicious activities and detect attacks arises. Intrusion Detection Systems for wireless networks have been researched from different perspectives some of them focus on network topology monitoring, others on different and independent layers traffic analysis others assuming knowledge of network infrastructure.

The reader of this thesis is supposed to know about wireless networks, how they

work and which are their basic mechanisms. Appendix C [page 81] provides some informations.
The project focuses on *Infrastructure* networks because they are the most common however nothing precludes to use some of the described techniques with *Ad-Hoc* networks.

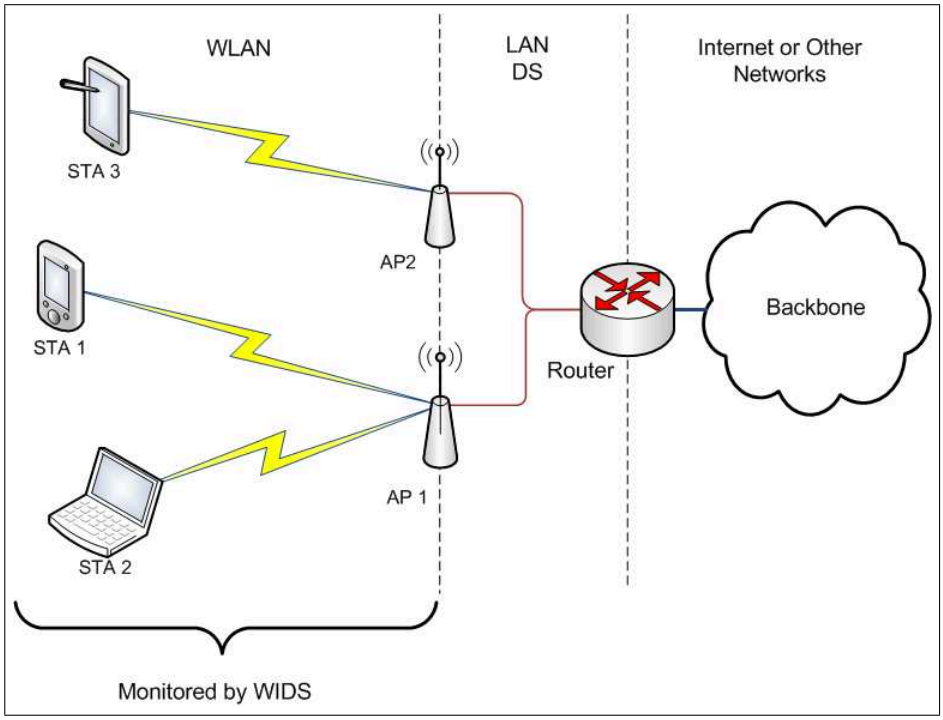

Figure 1.1: Infrastructure network

1.1 Common IEEE 802.11 threats

Before introducing and going through Wireless Intrusion Detection Systems description is convenient to list common vulnerabilities of IEEE 802.11 networks and describe related attacks. Such vulnerabilities fall into eight different areas:

1. Network Discovery;
2. Passive Eavesdropping and Traffic Analysis;

3. Message Injection and Active Eavesdropping;
4. Message Deletion and Interception;
5. Masquerading: Malicious AP and Session Hijacking;
6. Man-in-The-Middle;
7. Denial of Service (DoS) attacks;
8. IEEE 802.11i specific attacks.

Some of them (like no. 2) are merely passive while other can be both passive and active (like no. 1). Passive attacks cannot be detected since the attacker as a matter of fact is only an observer of the IEEE 802.11 medium, therefore he has no interaction with the system at all. Such attacks are used to gather information about the network such as topology, security policies enforced, stations connected and so on. In case data connections are weakly protected (i.e. with WEP) or no protection is present at all, attackers can easily access user data sent throughout the network.

Active attacks, on the other hand, can be detected (with more or less precision depending on the kind of attack). These attacks may be used for instance to gain access to network resources, deny access to the network to predetermined users/stations or force the network to use weaker security policies (this is the case where pre-RSN and full RSN protocols are both allowed by network security policies).

1.1.1 Network Discovery

Network Discovery can be cataloged both as an active and passive attack. This is not literally an attack since the network does not sustain any real damage. It is the very first attack that is usually performed. Is used to discover network information such as SSID, AP MAC addresses, radio channel and security protocols enforced. This attack is also known as *Wardriving*.

> ***Wardriving*** *is the act of searching for Wi-Fi wireless networks by a person in a moving vehicle using such items as a laptop or a PDA.*

There are many tools that can be used for Network Discovery and that can be linked to a GPS device in order to automatically mark the position of wireless

networks[1]. Most used tools are Netstumbler [4], Kismet or WiFiFoFum. Detection depends on the tools used by the attacker, some of them have specific signature patterns that allow their identification. Netstumbler for instance uses probe requests to discover networks. These probes have a particular payload that differs between Netstumbler versions [34] (i.e. for ver. 3.2.3 is "All your 802.11b are belong to us").

1.1.2 Passive Eavesdropping and Traffic Analysis

Passive Eavesdropping and Traffic Analysis are easy to perform due to the very nature of the wireless medium. Everyone in range of an Access Point can successfully sniff IEEE 802.11 frames. And since management and control frames are not encrypted[2] (even in last IEEE 802.11-2007 standard [8]) information gathering about AP and STA MAC addresses, along with hardware manufacturers and models[3], should always be accomplished. Moreover fingerprinting techniques for wireless devices use only information included in frame sections like duration, sequence number (see [14] for detailed information), always sent in cleartext. The attacker can also create statistical results concerning signal power, channels activity, devices interconnections, network throughput[4], or use the captured traffic to perform offline attacks on the encryption key used in the network. 802.11i implementation has provided strong encryption mechanisms that are difficult to break. Dictionary attacks are the most likely to succeed since, in WPA/WPA2-PSK networks, passwords are often chosen by users and therefore are usually common words[5].

1.1.3 Message Injection and Active Eavesdropping

This is the very first active attack. To carry it out successfully the attacker has to be able to craft an IEEE 802.11 frame bit by bit (in other words specify duration, sequence number etc.). Nowadays this can be achieved very easily from expert users since some cards chips, like Atheros, Prism or Cisco Aironet,

[1] There exist online databases like http://www.wigle.net/ where wardrivers or just normal people share information about wireless networks locations.

[2] Security protection for management frames is addressed in IEEE 802.11w by TGw group. Status July 2008: meeting set for September 2008 to issue P802.11w D7.0.

[3] A simple freeware program can be found at http://www.paglo.com/opensource/roguescanner.

[4] The most widely used software in this category is Omnipeek (www.wildpackets.com).

[5] There are websites such as http://www.renderlab.net/projects/WPA-tables/ and http://rainbowtables.shmoo.com/ that provide precomputed hash tables of passwords that can be used to speed up dictionary attacks. The biggest hash table at the moment is about 33Gbytes large! Tools and information are available at [35, 36].

allow a station provided with the right tools (airjack or the more recent Lorcon) to modify every bit of the frame. Injecting certain frames in the network can be the first step of a Man-in-The-Middle attack, or a beginning of a DoS attack carried by deauthentication frames flooding or insertion of anomalous values in specific frame fields [26]. However it can be simply used as a mean to get more information about the network itself (again security policies, SSID and so on).

1.1.4 Message Deletion and Interception

To carry out this kind of attacks the attacker must be able to remove a packet from the network before it reaches the receiver. This can be accomplished by causing CRC errors at the receiver so that the packet is automatically dropped. An attacker could achieve this by producing radio interference or exploiting similar techniques. The attack should be performed surgically: the attacker must receive the packet before the receiver actually does in order to decide whether or not interfere.

Message interception is a step further message deletion. With this attack an adversary is able to control a connection completely. The adversary can capture a packet before the receiver actually receives it and decide whether to delete or forward it to the receiver. Message interception may seem difficult in wireless LANs because the legitimate receiver might detect a message at the same time as the attacker does, however there are some ways to achieve message interception using hardware not at the hand of ordinary users. The attacker could use for example a directional antenna to delete a packet on the receiver side, while simultaneously using another antenna to receive the packet itself. Detecting this threat can be difficult since if the sensor is placed near the AP it would virtually receive the same frames received from the AP. Truth is that sensors usually do not discard CRC failed frames, nevertheless relying on these frames could be a mistake because of the noisy nature of wireless medium. In these cases it is much simpler to deploy the sensor far from the AP or better, use multiple sensors that would detect discrepancies between traffic sent from the STAs and traffic received from APs.

1.1.5 Masquerading: Malicious AP and Session Hijacking

This is another active and possibly very dangerous attack. Here attackers steal and imitate the characteristics of a valid user, or worst those of a legitimate AP. After learning about network topology the attacker could masquerade himself by changing his MAC address to that of a valid user or act as a fully legitimate

AP by using software tools like HostAP (http://hostap.epitest.fi). This attack is also known as Rogue AP aiming primarily at controlling the traffic inside the network, thus making eavesdropping easier for the aggressors. In the worst case scenario the attack enables the attacker to gain authentication credentials simply by waiting for a user to authenticate with the Rogue AP[6]. By the introduction of 802.11i and RSN networks this threat was mitigated since protocols use strong authentication and cryptographic techniques.

This attack is commonly known as MAC spoofing. Detection can be very tricky. Methods based on frame information, like sequence numbers [24, 21], can be fooled by most recent tools in combination with specific cards (for instance airjack and lorcon with atheros or prism based chip cards). Methods that exploit other unspoofable features like signal power and RTTs[20, 18] seem to be much more effective.

1.1.6 Man-in-The-Middle

Man-in-The-Middle attack is taken under consideration although IEEE 802.11i provides effective countermeasures through strong authentication, confidentiality and integrity algorithms. Still there are networks where IEEE 802.11i is not used for many reasons: presence of legacy devices, complexity of 802.11i deployment (CA infrastructure, certificate distribution, devices support and so on), presence of anonymous users[7].

A successful MITM attack will place the attacker into the data-path between a user and an AP or between two user devices in ad-hoc mode. As a result, the attacker can maliciously intercept, modify, add or even delete data, provided he has access to the encryption keys.

The attack could take place in this way (AP and STA are legitimate, ATK is the attacker) :

1. ATK sends de-authentication frames to STA so that it is forced to leave the network;
2. ATK uses HostAP to spoof AP credentials;

[6] This attack can be also used as part of MITM attack. In this context, the AirJack (http://sourceforge.net/projects/airjack) and MonkeyJack (http://www.wikipedia.org/monkeyjack) software tools are most commonly used to launch a masquerading attack.

[7] For instance "DTU wireless" Network has no encryption and is the same for many universities or public structures, in other words every network that is meant to host anonymous and unidentified stations.

3. STA sends probe requests looking for another AP in order to reauthenticate with the Network. STA founds the ATK (disguised as AP) to be the best suitable one;
4. ATK associates with a legitimate AP using legitimate STA credentials;
5. There exists a data-path now STA↔ATK↔AP.

It is now obvious why confidentiality and integrity performed with strong encryption render MITM attack infeasible. As a matter of fact, without decryption keys, the attacker cannot learn STA or AP credentials.

It can be easily inferred that MITM is a combination of masquerading, spoofing and DoS attacks, so detection is feasible as long as these attacks are detectable [30].

1.1.7 Denial of Service (DoS) attacks

The purpose of DoS attacks is to prevent legitimate users to access Network resources. Denial of Service attacks are the most dangerous for many reason:

- They are nearly impossible to stop;
- They can be successful even in 802.11i;
- They can disrupt any wireless communication in a determined area;
- They can target single STA or entire Networks;
- They can be performed both at PHY and MAC level.

A straightforward but raw DoS attack is jamming Wi-Fi frequencies. This type of DoS can be easily detected but is impossible to prevent. Another way to perform DoS is to inject deauthentication frames so that STA are disconnected from the network. This is also difficult to stop since deauthentication frames are not protected even in 802.11i therefore they can be easily forged. All these methods prevent anyone to access the network, the next one on the contrary privileges the attacker to access the network blocking all the other users. The attack consists in reducing the backoff time of the attacker to zero and set the duration time of RTS frames to the maximum value in order to force NAV of STAs to be always positive. In other words the medium appears to be always busy resulting in a very effective DoS attack.

There are also DoS attacks based on resources consumption. The attacker makes APs consume their resources by flooding the network with management or control frames. Almost every AP is vulnerable to this kind of attack[9] and results are very effective.

Moreover IEEE 802.11i is vulnerable to new DoS attacks that exploit EAPOL message used by EAP [33, section 2E] or reply attack in the 4-way handshake [22].

1.1.8 IEEE 802.11i specific attacks

802.11i is an amendment to 802.11-1999 fully implemented at present in 802.11-2007[8]. It defines and explains a new concept: RSN (Robust Security Network). Its goal is to strengthen IEEE 802.11 security policies by implementing new security algorithms (very unlikely to be broken) that render authorization, confidentiality and integrity secure. Although strong confidentiality has been achieved by certificates infrastructures and AES CCMP protocol employment in WPA/WPA2 networks (some weaknesses have been found in CCMP protocol[25]. CCMP is considered a long term solution for data confidentiality [23, 16] and therefore widely recommended), new attacks (mainly DoS) have been discovered.

RSNA establishment procedure [23, section 4.1] is divided into six stages:

1. Network and Security Capability Discovery;
2. 802.11 Authentication and Association;
3. EAP/802.1X/RADIUS Authentication;
4. 4-Way Handshake;
5. Group Key Handshake;
6. Secure Data Communication.

Attacks are feasible only in stages 1,2,3 and 4.

Attack in stage 1 consists in forcing both participants (supplicant and authenticator) to use pre-RSN protocols. Since nowadays there exist devices that are not RSN fully compliant, IEEE 802.11-2007[8] allows the coexistence of both pre-RSN and RSN, for migration purposes only and as a transitional measure.

If the attacker is successful he could force the use of insecure protocols such as WEP and therefore be able to decrypt everything that goes on the air. Stage 2 is the usual authentication procedure as it was in early 802.11 thereafter former DoS attacks are feasible. Stage 3 can be attacked by DoS that exploit unprotected EAP messages in 802.1X [23, section 5.1].

Stage 4 represents the key of RSN establishment procedure and it is the very new feature added by 802.11i. The 4-Way handshake is used to confirm the existence of the PMK, verify the selection of the cipher suite, and derive a fresh Pairwise Transient Key (PTK) for the following data session. Simultaneously, the authenticator might also distribute a Group Transient Key (GTK). After this stage, a fresh PTK (and maybe GTK) is shared between the authenticator and the supplicant. A flaw has been discovered in the 4-way handshake [22, 29] that permits, through a reply of message 2 of the handshake, to disrupt the communication between participant thus resulting in an effective DoS attack[8].

1.2 Intrusion Detection Systems: classification and description

> ***Intrusion Detection*** *is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices*[9].

Intrusion detection systems main purpose is to detect and identify possible incidents by traffic monitoring. IDS can be divided into four categories, depending on types of event they recognize:

- Network Based;
- Network Behavior Analysis (NBA);
- Host Based;
- Wireless.

[8] For a full and detailed analysis of 802.11i please refer to [23, 29, 22, 28, 11, 16].
[9] From [27, NIST 800-94].

The first ones are used to monitor network segments at levels typical of wired networks like TCP, UDP or IP, and are able to analyze protocols activity in order to identify strange behaviors. NBA are used to detect threats that generate unusual traffic flows and violate network policies. Host based IDS are usually deployed within single hosts thus they perform a very different analysis compared to the others. They monitor system activity, system logs, applications behavior and generally host related features. Wireless IDS on the other hand are used to monitor and analyze traffic features that are proper of IEEE 802.11. They are not meant to analyze higher level traffic (i.e. TCP/IP), but they can forward it either to Network Based or NBA IDS.

There exists then a second distinction used to identify the IDS detection methodologies:

- Signature or Misuse Based;
- Stateful Protocol Analysis;
- Anomaly Based.

The first uses patterns of well-known attacks or weak spots of the system to match and identify known intrusions. Such systems are virtually error free (very low false positives occurrences) since they detect intrusions comparing traffic patterns to malicious known ones. On the other hand they cannot detect new kinds of attacks and therefore their usefulness is in a way limited.

Stateful Protocol Analysis consists in comparing predetermined profiles of known protocols activity against the observed one. Each protocol state is traced so that any deviation from the usual state raises alarms. This analysis relies on vendor developed universal profiles that specify particular protocols behavior.

Anomaly Based is designed to detect unknown kind of attacks. This is true as long as the attack generates unusual traffic. This type of IDS observes and monitors the network over specific features. If traffic deviates significantly from normal values, an attack is likely to take place and alarms are raised. Setting a good anomaly based IDS needs a period of training during which surveys on controlled environment are used to identify normal traffic behaviors. If this is not done accordingly, anomaly detection will generate both false positives and true negatives occurrences[10].

[10]There have been studies about reducing the false positive and the true negative rates by thresholds tuning, use of neural networks or by increasing the observation window thus increasing traffic correlation [15, 18].

Thinking about wireless, training an anomaly based IDS for every kind of legitimate traffic pattern is a very difficult and complex task. For this reason is convenient to focus on a reasonable number of features (6-8) whose normal behavior is well-known, better if specified by standard. Note that there exist behaviors that standards does not specify at all (like duration values, SSID forbidden ones[11] or probing procedures). These are therefore vendor/hardware or OS specific and they could be used for other purposes like fingerprinting [14, 17].

Other techniques that can be employed in a wireless IDS are:

- State verification through tracking IEEE 802.11 state machines [32, 19];
- 802.1X Authorization and Authentication behavior checking [31].

However is not possible to detect all attacks. As shown in [10] if someone is patient and capable enough to exploit normal traffic and send malicious code within "low-profile" traffic, a worm could spread in hundreds of thousands stations within a year. This could be used to launch several kind of distributed attacks, like "Zero day" DoS, or to create bot networks.

1.3 Present solutions

Since wireless networks are employed both in private and corporate networks, and especially last ones do an extensive use of them, many proprietary solutions were developed. However there exists some open source ones that are interesting. Those are:

- Snort-Wireless[12];
- Kismet.

Snort-Wireless is an IDS that checks each 802.11 frame against a rule-set. If the rule set is violated an alarm is raised [6]. Kismet is a more complex IDS. Is a Signature-based distributed IDS that checks for known attacks.

Proprietary solutions are:

[11] Null SSID value can bring a DoS attack [26].
[12] Different from AirSnort that is a tool for WEP cracking.

- AirMagnet;
- AirDefense;
- Red-M.

All these solutions are far more complex then previous ones since they are developed to be a comprehensive IDS solution. Therefore they are not really classifiable in a specific category. They are designed to monitor specific networks, therefore they use informations about network topology, policies and infrastructure.

1.4 Feature selection for Anomaly based Wireless IDS

As already explained in 1.2, behavior of feature considered for IDS development should be well-known or at least reasonable. Moreover if features comes from different layers, cross referencing can ease anomalous traffic detection. Since the wireless medium possesses interesting and difficult to forge features, and since higher layers[13] should not be addressed by Wireless IDS, features should be selected from PHY and MAC layers. Some of them, like sequence numbers or power, result directly from PHY or MAC values, others, like frame rates or RTT, need processing. After reviewing many possible choices, selected features were:

1. Signal Power;
2. Round Trip Time (RTT);
3. Number of AP;
4. Number of SSID;
5. Rate of de-authentication frames;
6. Rate of Probe Requests and Probe Response;
7. Rate of Beacon Frames;
8. Rate of Clear-to-Send (CTS) Frames;
9. Frame Sequence numbers.

[13] TCP and IP.

1.4.1 Signal Power and Round Trip Time

This two features were chosen after a thorough reading of [18, 20]. Signal Power and RTT are features very easy to obtain at the receiver and, on the attacker side, impossible to decide, predetermine or generally to forge. They can both be inferred from the Prism or radiotap header[14] [37] by RSSI and MACtime fields.

The first feature - Signal Power - depends indeed on the path the signal covers, the obstacles it find, the attenuation (surely not strictly constant during the route), and finally, but probably most importantly, on the power emitted at the transmitter (that depends on the hardware, the antenna etc.). The second feature - Round Trip Time - is computed at the sensor by subtracting RTS frame timestamp from the first data frame timestamp. The computed RTT should then be:

$$RTT = T_{S\_AP} + T_{comp\_AP} + T_{AP\_STA} + T_{comp\_STA} + T_{STA\_S}$$

Where T_{S_AP} is the time between Sensor and AP RTS receipt that can be either negative or positive (depending whether the AP or the Sensor receive the RTS first), T_{comp_AP} is the time the AP needs to process and reply to the RTS, T_{AP_STA} is the time between AP CTS transmission and station CTS receipt, T_{comp_STA} is the time STA needs to process and reply and T_{STA_S} is the time between STA data transmission and Sensor data receipt.

In order to be able to forge these parameters the attacker should:

1. Know the STAs hardware;
2. Know the WIDS sensor location;
3. Be able to reproduce every power profile as needed;
4. Be exactly in the same place of the spoofed station in case of multiple sensors.

These are indeed almost impossible abilities to achieve and this is the reason why measurements from these features seems reliable.

Time resolution depends on the hardware and on the card manufacturer. In [7, documentation: timestamp section] can be found that Winpcap, and consequently libpcap too, automatically uses the full resolution that the hardware

[14] Description of these two header types can be found in 3.1.1 at page 23.

permits, up to nanoseconds. Tests made with wireshark shows that atheros based cards resolution is within microseconds. There has not been the chance to test different hardware, but extensive research has shown that only specialized hardware has nanosecond resolution. One microsecond at $2.5 * 10^8$ m/s corresponds to 250 m, therefore RTT is determined mostly by computation times that are within tens of microseconds. Computational time are still card specific hence experimental data has to be gathered in order to see if RTT can be useful feature.

1.4.2 Number of AP and SSID

Utilizing these two features could seem useless in some cases or absolutely necessary in others. As explained in 1.1, many tools like Rogue AP, HostAP or Airsnarf [1], add to a wireless environment either new APs or new SSIDs or both, thereby making possible different attack typologies like DoS, MITM or just credential stealing. Since usually network topology is known so that every AP MAC is registered and SSID and policies are known, it is easy to see if suspicious activity is taking place. Even if network topology is unknown a huge number of APs or SSIDs would likely be suspicious. If this feature is combined with the Signal Power feature, the possibility of malicious activity can become a certainty in the event that all new APs have the same power.

Moreover since APs and SSIDs detection is straightforward, training and defining thresholds should be an easy task.

1.4.3 Management and Control frames Rate

Monitoring management and control frames rate is crucial since up to now [8] they are still unprotected and unencrypted. Training can be very simple for periodic frames such as beacon frames, and difficult for user dependent frames like RTS, CTS, Probe Requests, and so on. However normal traffic should be characterized by usual frame rate so that training can be sufficiently precised after long observation (see [37, tables II and III] for examples). Anomalous readings could indicate attacks such those explained in 1.1.1, 1.1.3, 1.1.5, 1.1.6, 1.1.7. Combining this feature with those in 1.4.1 could help identify the type of attack with more precision. It could be also opportune to verify if this frames contain unusual field values to prevent attacks that exploit lack of definition of usual or allowed values such as firmware-based attacks [26].

1.4.4 Frame Sequence Numbers

Standard specification for sequence number (SN) field states:

- It is a 12 bit field;
- Each MSDU or MMPDU has a sequence number assigned;
- Sequence numbers are not assigned to control frames.

Sequence numbers behavior is never really specified and is used only for MSDU or MMPDU retransmissions. Therefore its behavior is undetermined for most of the frames. Usually each card increment his sequence number every frame they transmit, so that SN is periodic every 4096 frames. If something odd or unexpected happens SN should be reset. Research and surveys has shown that each card vendor implement its own SN behavior. That means that behavior of this feature is unpredictable. Moreover there exists tools and device drivers that permits to set SN. However this feature was chosen because monitoring its behavior could lead to interesting results and also because detecting SN is effortless.

1.5 Report structure

This report is divided into five chapters. The second chapter examines all requirements that a WIDS should meet. The third discusses first how preliminary issues concerning application developing were solved and then implementation stages and application design. The fourth chapter is the most important. It provides results along with data analysis and in the end requirements assessment. Finally the fifth chapter elicits conclusions and future work.

Chapter 2

Requirements

The chapter lists and describes requirements that should be met by the application. Requirements analysis permits to have a clear idea of what the system will look like and will do, coping with standard problems such as ambiguities/incompleteness in the specifications or difficulties in understanding how the system works. Requirements can be of two types: functional and non-functional requirements. The first ones define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be. Several ways can be used in order to express the system requirements (natural language, standard model or diagram) and all of them flow into a final requirement document.

A common procedure to follow in order to express the system requirements in a complete way is to apply the FURPS methodology, namely describing the requirements based on the following categories:

1. Functionality;
2. Usability;
3. Reliability;
4. Performance;

5. Supportability.

After a careful analysis of the problem and thorough examination of wireless IDS problematics, general requirements for a fully functioning application have been identified. They are introduced in following sections along with a brief description. Please note that the system under development is going to meet only some of the requirements since the aim of this project is to show that proposed system works and some attacks are actually detectable.

Natural language has been chosen to describe the requirements and the FURPS methodology introduced above has been applied for completeness with functional requirements mainly described under *functionality* whereas non-functional requirements under the remaining categories.

2.1 Functionality

WIDS should detect following kind of attacks:

- Session hijacking;
- MITM based on user identity theft;
- DoS that exploits management and control frames;
- Rogue AP;
- Generally every kind of attack that generates anomalous feature values.

Second attack detection results from first one, since MITM attacks may occur along with session hijacking. Third one should be easy to detect since DoS attacks are sometimes simple and often impossible to stop and rely on flooding of specific frame types.

2.2 Usability

The application should run independently and monitor traffic without system-user interaction. It should then provide results in a convenient format, so that they can be analyzed when needed.

2.3 Reliability

WIDS should detect attacks with a high accuracy in order to reduce false positive and true negative occurrences. Requirements that fulfill such purpose are:

- Low packet loss rate;
- Correct information gathering and parsing;
- Features accuracy;
- Precise time reference;
- Attacker traceability;
- WIDS should be immune from attacks, above all DoS.

2.4 Performance

Performance requirements are needed when a WIDS solution is going to be employed in a real environment scenario. In this case the machine has to be able to detect attacks as soon as they occur and signal them in order to let proper countermeasures to be taken in place. Such requirements are:

- Real-time packet processing;
- Fast attack detection;
- Fast alarm signaling;
- Generally any real-time associated requirement.

These requirements are not really addressed since the main purpose of this project is to develop an application that shows if attacks are actually detectable aside from real time detection. Real-time capabilities are assessed after all.

2.5 Supportability

Network topology knowledge is meant to be minimal or near to zero so that the application can be employed in any wireless network. Therefore there is

no special configuration that has to be done and WIDS must support different kinds of network configurations and policies. Moreover the application should be implemented using off the shelf hardware and common resources so that portability and scalability are seamless. In order to achieve this it should present some kind of software-hardware independence.

Therefore following requirements are desirable:

- Common 802.11 device support;
- Monitoring of features despite network topology and infrastructure;
- Device independent packet capture engine;
- Device independent parsing functionalities;
- Use of Device independent libraries.

The first and the last one could seem the same but they are slightly different. The first one pertains to general 802.11 device support (hardware side), meanwhile the last one to the use of a library that can be used with any kind of interface (even virtual interfaces). In the next chapter application developing is discussed and requirements that concern it are dealt with right along.

Chapter 3

Developing and Implementation

This chapter is divided into two main sections. The first one describes common issues in WIDS development such as packet loss, choice of suitable hardware, software - hardware interface and problems bounded by the very nature of the medium involved. The second section describes the actual implementation starting from Data Types definition, going through application modules description and finally getting to output data format specification.

3.1 Issues and solutions adopted during WIDS Design

When you want to implement a Wireless IDS or generally any kind of application that needs to interact with a radio medium and specifically with 802.11 devices, many issues arise. These are due to the way you interface the software with the 802.11 medium. Here every issue is analyzed by sectioning the interface in two parts, *A: medium* $\leftrightarrow$ *802.11 device* and *B: 802.11 Device* $\leftrightarrow$ *Software* so that *A* section concerns hardware and *B* software. By this division the analysis results clearer and solution adopted straightforward. Analyzed issues are:

- Hardware selection for 802.11 Monitoring;
- Choice of suitable programming language for real time application;
- Packet Loss;
- Packet Capturing;
- 802.11 Frame Parsing.

3.1.1 Hardware selection for 802.11 Monitoring

Since the proposed WIDS application relies on information concerning 802.11 medium like signal strength and frequency, a suitable hardware that provides such informations is needed. Moreover the hardware should be reasonably common and well supported in order to make software to hardware interaction seamless and simpler. There are also other useful informations that network cards can provide that make the surveys more reliable. One is the MAC time, that is the time in microseconds observed from the card at the reception of the first bit of the MPDU[1][5, TSFT field]. This information is more accurate and reliable than system time, therefore is used as main time reference. Moreover, since data frames dimension is not fixed, MAC time is the only feasible time reference and this limit the choice between card vendors.

From *A* part of the interface the device should meet these features:

- Good sensitivity[2];
- Fast chipset so that packets are not dropped by the card;
- Monitor mode capability;
- The device should be able to survey all the possible channels defined in [8, IEEE 802.11 Standard].

Sensitivity is crucial for the device capability to correctly receive and decode 802.11 frames. Without a good sensitivity a WIDS is ineffective since many frames would be lost and therefore information would be incomplete and inaccurate. For a similar reason a fast chipset is desirable as well. As a matter of fact a WIDS has to process much more data than a normal 802.11 device in

[1] That is the definition for the Radiotap header, that is the header chosen for implementation.

[2] Receiver's sensitivity is a measure of its ability to detect low-level signals.

Managed mode[3], therefore it should be provided with a fast chipset within the wireless device so that frames are not dropped from the card itself. Most of wireless cards usually drop frames that are addressed to other devices or whose CRC check failed. Such cards cannot be used as device for a WIDS for obvious reasons[4]. Monitor mode permits to receive and decode every frame, even with CRC check failed. Therefore having a card with this feature is mandatory. Finally a WIDS should be able to scan all frequencies, even if they are not allowed in a specific country, because obviously attackers do not follow rules.

On the *B* side we have:

- Chipset must provide the system with medium information;
- Such information should be attached to every frame;
- Information format must be well documented so that interpretation by the application is not ambiguous[5].

While capturing traffic from a device, the card, along with the 802.11 frame, passes a proprietary header that provides additional information about the frame. There exist two main header formats called Prism and Radiotap.

Prism Header

Prism Header was first designed for Prism Card and then adopted by most card vendors. Is older and limited compared with Radiotap Header. Prism Header is 144 byte fixed length and has all the main fields Radiotap has: channel, RSSI[6], MAC time and so on. On the other hand it lacks flexibility and field interpretation is sometime ambiguous (i.e. RSSI is different between different vendors). Moreover definition of fields is not as well documented as Radiotap. Finally it lacks of a field that can be exploited for parsing purposes, the *FCS present* field.

[3]This is the default mode. While in managed mode a station is supposed to be connected to a network, looking for it or generally behave like a normal device.

[4]A WIDS has to collect traffic that is not addressed to it, sometimes even if their CRC check failed.

[5]This should be provided seamlessly, unfortunately specific information from card vendors is not always provided.

[6]Received Signal Strength Indication: is an 8 bit measure of relative signal strength whose interpretation varies from card vendors i.e. Cisco 0-100 scale, atheros 0-60.

Radiotap Header

Radiotap Header was initially designed by David Young. Is more flexible than Prism Header. Radiotap is variable-length and extensible. First 64 bits contains the header version[7], the length of the header and a bitmap that signals the presence of specific fields. Fields are strictly ordered and length implicit, that means that each field must be specified by the vendor. An extended bit is provided. This bit signals the presence of an extended radiotap header. Is currently unused but it makes the header ready for future modifications.

Field that are used by the WIDS are:

- **TSFT**: Value in microseconds of the MAC's 64-bit 802.11 Time Synchronization Function Timer when the first bit of the MPDU arrived at the MAC. For received frames only.
- **Channel**: Tx/Rx frequency in MHz.
- **Antenna signal**: RF signal power at the antenna, decibel difference from an arbitrary, fixed reference.

MACtime values is taken directly from the wireless card. The card has an internal timer that is started when the card is first turned on. Since the timer is 64 bit length and has microsecond resolution the timer can work for $2^{64}\mu s = 584942.42 years$ therefore there shouldn't be overflow issues indeed. RF signal power is not an absolute measure therefore it must be managed accordingly.

After analysis of all previous requirements the wireless card chosen for the WIDS was an Atheros AR5005G card. Atheros are well known to be a very good solution both for *A* and *B* issues. Moreover the chipset is "software-defined". This means that the card's behavior can be tightly controlled by software. This additional feature makes the Atheros chipset probably the most desirable one for the purposes of this project.

3.1.2 Choice of suitable programming language

Since WIDS purpose is to monitor 802.11 networks to signal strange behaviors and raise alarms if suspicious activities are detected, the application must be as

[7] Currently only v0 exists

fast as possible and must be designed to match a real-time environment. After thorough evaluation of available solutions, both from an operating system and programming language perspective, Linux as operating system and C/C++ as programming language were chosen as best solution both for the wide support provided for Atheros cards and availability of capture and parsing libraries. Detailed choices are explained further on.

3.1.3 Packet Loss

Packet loss is one of the most critical issues. Obviously loosing too many packets could yield to undetected situations, especially for Hijack and DoS attacks. Therefore packet loss must be analyzed thoroughly. Research on this side is very large and well established in a wired environment. There exists libraries, like PF_RING[12], that can bring to virtually zero loss in a Gigabit scenario. Since packet capture is done at Kernel level, these solutions provide a patch for the kernel that optimizes and makes packet capture faster. In a wireless scenario, where throughput is lower, normal kernel can be used as long as the CPU is not scheduled to other processes except those dedicated to packet capture and traffic analysis. In addition packet dimension is directly related to packet loss: the more the packet is small the higher is the chance of loss. This is very problematic in wireless scenario since monitored packets are mainly management or control frames whose dimension is ranges from 12 to 200 bytes, anyway tests showed that those frames are seldom dropped. As a matter of fact modern kernels are improved and optimized for packet capture comparing the ones of 3-4 years ago (see [13]) therefore using normal libraries shouldn't yield to packet loss.

3.1.4 Packet Capturing

After having defined the environment the choice of the library for Packet Capture was obvious. Libpcap [3] is beyond doubts the most spread and well documented library, included in every Linux distribution and C/C++ native. So as it should be clear is the natural and straightforward choice. Moreover, libraries and patches like PF_RING already supports libpcap C/C++ programs. This means that every program already written with libpcap needs only to be recompiled to work with above mentioned patches.

3.1.5 802.11 Frame Parsing

Frame parsing is the the most critical and crucial issue. Every information taken from 802.11 frames is obtained by parsing raw frame data and such informations are the only data that WIDS process. Moreover the presence or the radiotap header makes frame parsing harder since it adds variable length bits at the beginning of raw data captured by libpcap. Therefore careful and accurate programming must take place and has to be checked at every step in order to assure that genuine data are sent to the WIDS core module for analysis. For this purpose Airware libraries [2] were found to be the most suitable choice. These libraries provide valuable tools for IEEE 802.11 frame parsing although they are not fully developed yet. Main features that make Airware a good choice are:

- Support of Radiotap header;
- Management, Control and Data frame fields awareness;
- Source code availability;
- C++ native.

The most noticeable feature is support of Radiotap Header indeed. Values are directly converted from radiotap format[8]. Since the library is just a tool for advanced programmers, documentation is not present, therefore a thorough analysis of the source code was required for library understanding and use. Airware is written in C++ and each type of frame/packet is initialized and managed as an object.

Specific data fields and formats parsed from the frame will be explained in next sections.

3.2 Application Design

Network monitoring applications, depending on the purpose of the application, consist usually of two or three sub applications that perform different and independent task. These can be summarized in:

1. Capture/sniffing;

[8]Each radiotap field has its own particular format, little endian and with bit padding variable length.

2. Data parsing/processing;
3. Data analysis.

In some cases above tasks are performed sequentially, for example when you want to do traffic analysis with Ethereal/wireshark [7], first you capture the traffic from the chosen interface, then you stop the capture and do data analysis. When it comes to network monitoring things are slightly different. Since monitoring implies a continuous process that in principle could go on indefinitely, capture, processing and analysis must take place at the same time.

The WIDS application is divided into 7 threads:

- Packet Capture and Parsing;
- Power monitor;
- RTT monitor;
- SSID monitor;
- Rates monitor;
- Sequence Number monitor;
- Memory cleaner.

The first one carries out capture and per frame parsing. Second to seventh handle feature monitoring through processing appropriate data types defined further on. Last one is not exactly a WIDS specific thread; it serves as memory cleaner. Capturing packets consumes a lot of memory in a high traffic network scenario, so if an attacker would like to bring down a WIDS system that doesn't free memory from already analyzed traffic, a simple DoS attack consists in generating a huge amount of traffic so that WIDS memory is quickly exhausted. Therefore a WIDS system must check and clean its memory regularly.

3.2.1 Data types description

Data types developed for WIDS are:

- 802.11 parsed elements;

- Power monitor data;
- Rate data;
- Sequence Numbers data;
- RTT data;
- SSID data.

3.2.1.1 802.11 parsed elements

Is the main data type used by all the threads. It consists of a structure declared as folows:

```
struct parsed_Header_elements {
    u_int8_t bssid[6];
    u_int8_t srcAddr[6];
    u_int8_t dstAddr[6];
    u_int16_t channel;
    u_int64_t MACtime;
    int8_t RxPwr;
    u_int8_t type;
    u_int8_t subtype;
    bool fromDS;
    bool toDS;
    string SSID;
    u_int16_t B_Interval;
    u_int16_t B_Capabilities;
    int seqNum;
};
```

It contains all the data necessary for chosen features monitoring. The data is frame type independent, so every thread checks field consistency by type/subtype validation. For instance, if a control frame such as *CTS* or *RTS* is received, fields like Interval and Capabilities, proper of Beacon frames, are ignored. Note that there exist types of integer defined by length that matches directly IEEE 802.11 fields length.

3.2.1.2 Power Monitor data

Power Monitor data type is used by the power monitor thread. The data structure is defined as follows:

```
struct powerMonitor_data{
        int8_t diffs[4000];
        u_int8_t MAC_ADDR[6];
        double mean;
        double mean_sq;
        double var;
        int count;
        int8_t prevPwr;
        int8_t pwr[4000];
        double mean_sp[4000];
        double var_sp[4000];
};
```

The data is associated to each monitored 802.11 device by *u_int8_t MAC_ADDR[6]* that identifies the station by the MAC address. Variables ***double*** *mean,* ***double*** *var* store the actual variance and mean of the signal power[9] of the station while *int8_t diffs [4000], int8_t pwr[4000],* ***double*** *mean_sp[4000],* ***double*** *var_sp[4000]* stores last 4000 samples of power differences[10], of the signal power, of the mean and the variance measured at that specific sample detection. Finally ***int*** *count* counts the number of frames processed from the thread.

3.2.1.3 Rate data

This data type follows a slightly different approach from the other ones. Is divided into two structures. The first one stores general data needed to perform rate computation. The second one is used to make a per station rate measure, and it makes use of the first structure for rate computation. Both the structures are frame type independent. That means that modify WIDS to compute new rates of different frames is very straightforward.

Data type definition follows:

```
struct rate_structure {
        unsigned int rate_samples[50];
        u_int64_t time_diffs[50];
        int sample_dimension;
        unsigned int occurencies;
        double mean;
        double instant;
        double mean_sq;
```

[9] Taken directly from Radiotap Header [see 3.1.1 at page 24].
[10] Difference relative to the previous frame from the same station.

```
        double var;
10      unsigned int array_index;
        int counter;
12      double mean_samples[50];
        double var_samples[50];
14      double instant_samples[50];
};
16
struct rateBYmac {
18
        string MAC;
20      rate_structure rate;
        int interval;
22      double expected_rate;

24 };
```

For the first structure their meaning is:

1. *rate_samples[50]*: each of 50 samples contains the number of occurrences since the beginning of the measure;
2. *time_diffs[50]*: each of 50 samples contains time difference between start time and time when sampling takes place;
3. *sample_dimension*: shows every how many occurrences the sampling has to take place;
4. *occurencies*: number of occurrences since the beginning of the capture;
5. *mean*: stores the actual mean;
6. *instant*: stores the instant rate;
7. *mean_sq*: auxiliary variable for variance calculus;
8. *var*: stores the actual variance;
9. *array_index*: used to point out the next sample array index to be written;
10. *counter*: stores the actual number of measured samples[11];
11. *mean_samples[50]*: stores last 50 mean samples;
12. *var_samples[50]*: stores last 50 variance samples;

[11] Is different from occurencies. Is equal to $|\frac{occurencies}{sample\_dimension}|$

13. *instant_samples[50]*: stores last 50 instant rate samples.

For the second:

1. *MAC*: Identifies the station the rate refers to;
2. *rate*: is the above structure;
3. *interval*: for *beacon* frames only is the time interval stated from the AP;
4. *expected_rate*: for *beacon* frames only is the expected rate;

3.2.1.4 Sequence Numbers data

This data type stores information about sequence numbers for each device. Structure follows:

```
struct seqnum_data {
        string MAC;
        int seqs[12288];
        int counter;
};
```

MAC identifies the associated station, *counter* counts number of samples taken since the beginning and *seqs[12288]* stores last 12288 sequence numbers. Since sequence number is a 12 bit field it can assume $2^{12} = 4096$ values. Therefore the size was chosen to be $4096 \times 3 = 12288$ in order to sample three sequence number cycles[12].

3.2.1.5 Round Trip Time data

Since RTT detection needs to keep track of the state of 802.11 devices, RTT data consist of two types. The first enumerates the states the device can assume, the second stores RTT measures.

Types are defined as follows:

[12] As a matter of fact each 802.11 device resets sequence numbers depending on the vendor. Therefore this feature can be used for device fingerprinting.

```
enum RTT_state {None, RTS_Received, CTS_Received, DATA_Received};

struct RTT_monitor_data {
        string MAC;
        u_int64_t RTS_time;
        u_int64_t CTS_time;
        u_int64_t DATA_time;
        RTT_state status;
        u_int64_t CTS_samples[4000];
        u_int64_t samples[4000];
        double mean;
        double mean_sq;
        double var;
        double CTSmean;
        double CTSmean_sq;
        double CTSvar;
        int errors_CTS;
        int errors_RTS;
        int errors_DATA;
        int counter;
        int RTScounter;
        int CTScounter;
};
```

As already explained RTT sampling needs to track the state of a device, therefore data structure has many auxiliary variables that are needed to carry out the computation. Moreover, to add information, there are variables that keep track of errors occurred. All these variables are:

1. *status*: keeps track of actual state;
2. *errors_CTS*: counts errors occurred at unexpected CTS receipt;
3. *errors_RTS*: counts errors occurred at unexpected RTS receipt;
4. *errors_DATA*: counts errors at unexpected DATA receipt;
5. *RTScounter*: counts RTS occurencies;
6. *CTScounter*: counts CTS occurencies.

MAC, RTS_time, CTS_time, DATA_time are used to identify the 802.11 device and to compute the RTT value.

All the remaining variables serve to compute statistical data as in previous data types.

3.2.1.6 SSID data

SSID data type keep tracks of all discovered SSIDs and their relative AP. Is composed by two entities. Definition follows:

```
typedef list<string> MAC_ptr_list;

struct SSID_monitor_data{
        string SSID;
        int numAP;
        MAC_ptr_list AP_BSSID;
};
```

SSID contains the SSID value, *numAP* the number of discovered associated access points and *AP_BSSID* the list of all AP MAC addresses.

3.2.2 Thread description

In this section the application is thoroughly described. The complete source code can be found in appendix A at page 69.

3.2.2.1 Packet Capture and Parsing

The application as already explained is divided into seven threads. The main thread is *Packet capture and parsing*. It starts in the *main* function where all variables are initialized and every thread started. The program accepts the following parameters:

1. interface;
2. number of packets to capture and process.

The first is the interface the WIDS should monitor and second is number of packets. If this is set to 0 the capture is continuous. The function ***int*** *pcap_loop(pcap_t * p,* ***int*** *cnt, p*

does the capturing of the traffic and passes each packet to the callback function that is the data parsing function.

The parser then takes place. It starts by initializing the object *Packet_80211* defined in the libairware library. This object is a generic 802.11 frame that provides basic functionalities. At this point radiotap parsing takes place. Field processed are: TSFT, channel and signal power. Then a more specific object is initialized depending on the frame type. Possible objects are *Packet_80211_mgmt, Packet_80211_ctrl* and *Packet_80211_data*. This distinction is required since address fields assume different meanings depending on the type/subtype of the frame. At this point addresses are parsed into *BSSID, dstAddr* and *srcAddr*. If they are not existent (i.e. CTS frames does not have a TA address) they are set equal to 0. After this step all the other informations are parsed and a new variable ***struct*** *parsed_Header_elements* is created and filled out with parsed parameters. The structure is then pushed at the end of a list of *parsed_Header_elements* types. This list is a global variable and is shared among all the threads. The only thread that modify the list are the memory cleaner and the parser.

Finally the parser signals through semaphores that a new packet has been processed. Each thread has it's own semaphore so that it can work completely independently without use of any race condition control thus enabling real time processing.

3.2.2.2 Power monitor

This thread is responsible for 802.11 devices power data processing and monitoring. It creates a list of *powerMonitor_data* elements where each element is associated to a device by its MAC address. When the presence of a new *parsed_Header_elements* is signaled by the parser, it starts by searching the list for a matching source MAC address. If such element is found is then updated. On the other hand if such element does not exists, the thread simply create a new element and add it to the list.

The update process perform these steps:

1. Compute difference between actual and previous Rx Power;
2. Update *arraypointer* value so that it points to the position of arrays where data will be stored;
3. Store new *diffs* and *pwr* samples;
4. Compute and update new statistical information (mean and variance);

5. Store actual mean and variance;
6. Update auxiliary variables.

All gathered data are stored in a file for data analysis.

3.2.2.3 RTT monitor

To measure and monitor RTTs is a very complex and sensitive task. RTT is the time elapsed between the RTS receipt and first DATA frame after a successful RTS-CTS handshake[13]. In order to do this the thread creates a list of RTT data type. Each element represent a 802.11 device and keeps track of it's state. The monitor perform the following steps:

- For each received RTS frame:
  1. Finds the device in the list by the TA field[14];
  2. Checks the state of the device, and if is wrong updates error counters
  3. Reset the state to *RTS_Received*;
  4. Stores *MACtime* in *RTS_time*;
  5. Updates *RTScounter*.
- For each received CTS frame:
  1. Finds the device in the list by the RA field[15];
  2. Checks the state of the device, if wrong reset it to *None*, updates error counters and stops processing;
  3. Stores *MACtime* in *CTS_time*;
  4. Set state to *CTS_Received*.
- For each received DATA frame:
  1. Finds the device in the list by the SA field[16];
  2. Checks the state of the device, if wrong reset it to *None*, updates error counters and stops processing;

[13] For a detailed description see [8, section 9.2.5].
[14] for RTS frames TA field is stored in *srcAddr* of *parsed_Header_elements* data type.
[15] for CTS frames RA field is stored in *dstAddr* of *parsed_Header_elements* data type.
[16] for DATA frames SA field is stored in *dstAddr* of *parsed_Header_elements* data type. For frames coming from DS this correspond to BSSID.

3. Stores *MACtime* in *DATA_time*;
4. Compute the RTT and CTS_time[17];
5. Computes statistical informations (mean and variance);
6. Update all counters;
7. Set state to *DATA_Received*.

Tracing RTS-CTS state across a 802.11 Network with many devices is not easy and present many issues due to the standard definition such as:

- CTS frames does not have TA field, therefore in a scenario where the AP is initiating a RTS-CTS handshake is not possible to determine which station sent the CTS frame. In principle should be the station the RTS was addressed to, but attacker usually ignore that;
- There exist the possibility that for data transmission RTS-CTS is not used;
- RTS and CTS frames are very small, that reduces the chance to acquire them;
- Contention problems that could lead to state mismatch.

Some of them, like RTS-CTS detect capability, can be solved by placing the WIDS near the AP thus chances that WIDS detect all frames detected from AP are increased.

3.2.2.4 SSID monitor

SSID process data in order to keep trace of present SSIDs and their BSSIDs[18]. The thread gather the required informations from *beacon* frames. It creates a list of SSID elements and each processed frame informations are updated.

Since BSSIDs can be retrieved also from many other frame types, this thread keeps trace of every BSSID ignoring the SSID it belong to. That is done by maintenance of a separate list that keeps trace of BSSIDs. Double checking BSSIDs could seem useless since every AP should usually send beacon frames. The point is that a malicious activity that would tamper with frames could lead to BSSIDs mismatch thus resulting in seamless and immediate detection.

[17] Time elapsed from RTS and CTS.
[18] BSSID is the MAC address of an Access Point

All gathered data are then stored in a file for data analysis.

3.2.2.5 Rates monitor

This is probably one of the most time consuming threads, since it makes many computations and processes several kinds of frame. Frames monitored from this thread are:

1. Beacon frames;
2. Deauthentication frames;
3. Probe request frames;
4. Probe response frames;
5. ClearToSend frames.

All are monitored ignoring source or destination addresses except Beacon frames. In fact the thread monitors also the Beacon frame rate for each AP, and stores information about expected rate[19] and measured one. First received packet *MACtime* is used as time reference.

Action performed for each monitored frame are:

1. Time difference computing;
2. Instant rate computing by division: $\frac{\#frames}{\Delta t}$;
3. Statistical information update (mean and variance);
4. Sample storing.

The rate computation is performed every *sample_dimension* frames so that the number of frames is sufficient to compute the rate. Each frame type has is own *sample_dimension* since some are frequent and other rare.

Beacon frame rate per single AP are computed in the same way, and stored in a list of *rateBYmac* elements. All gathered data are then stored in a file for data analysis.

[19] Expected rate is computed by the interval field in the beacon frame.

3.2.2.6 Sequence Number monitor

Monitoring sequence numbers is a very easy and simple task. For every processed frame the thread just record the *seqNum* in a sample array associated with a MAC address and store it in a list.

Normally sequence number value is managed directly by the hardware of the device thus is not user defined [24]. Unfortunately there exist cards where malicious user can set the value by software but card behavior should remain the same. Therefore sample dimension was chosen to be 12288 so that data are significant (see 3.2.1.4 at page 31).

3.2.2.7 Memory Cleaner

This thread is used only for memory maintenance purposes. As stated before, all threads process elements from the main *parsed_Header_elements* list in order. When all the monitoring threads have processed a specific element, they signal to the Memeory Cleaner that the element is no longer needed and it can be safety deleted . The element is then eliminated and memory freed.

Chapter 4

Results

In this chapter results are presented as gathered from the application, then they are analyzed and screened to see whether or not they are relevant and consistent. It is divided into three parts, the first one describes scenarios set to perform surveys, the second shows gathered data and their analysis, the third one outlines which requirements were met and which not.

4.1 Case studies

All the different surveys were performed with a general setup. Three machines under direct control and two public wireless networks within DTU control. Specifically:

- Controlled machines:
  - A laptop running WIDS application;
  - A laptop posing as STA or ATK;
  - A PDA posing as STA.
- Wireless networks:

  - *"DTU Wireless"* network: unprotected network with captive portal authentication;
  - *"eduroam"* network: WPA-TKIP protected network with TTLS PAP authentication.

Location was the *Department of Informatics and Mathematical Modeling - DTU Informatics*. Some surveys were performed during daytime, therefore uncontrolled traffic is sometimes present, other during nighttime, so that the only uncontrolled traffic was the one produced from AP only (Beacon frames for the most part). However analysis are performed only on controlled traffic.

Considered scenarios are:

- Station data session:
  - STA uploading and downloading data far from WIDS;
  - STA uploading and downloading data near the WIDS.
- Beacon frame only, no STA present;
- Hijack Attack:
  - Two STA transmitting simultaneously with the same MAC address;
  - ATK hijacking STA: two MAC transmitting at the same time for few frames only;
  - ATK hijacking STA: no simultaneous transmission.
- FakeAP: creates a huge number of fake access points and SSIDs.

Data session

Surveys were performed in two different situations:

- Station near the WIDS in order to receive strong signal thus decoding all its frames.
- Station far from WIDS (about 5 meters) to simulate a real situation where stations are far from the WIDS.

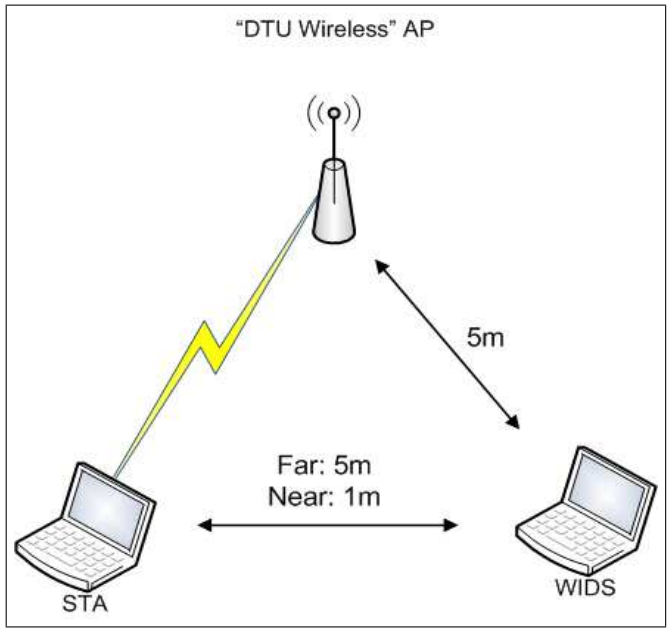

Figure 4.1: Data Session setup

AP and WIDS relative location and distance was the same in both cases (about 5 meters).

Data session was generated by SFTP session in upload and by FTP one in download. Session lasted for about 1 hour in order to have consistent data to analyze. It was performed during daytime.

Beacon frames

In order to receive only Beacon frames, these surveys were performed during nighttime. Two AP were present, one for *DTU Wireless* and one for *eduroam* networks. Purpose is to see whether or not AP comply with beacon interval stated in the frames.

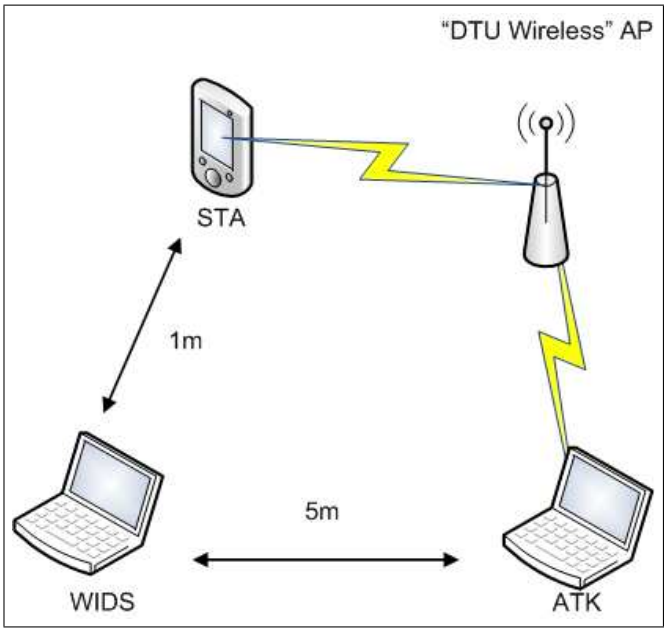

Figure 4.2: Hijack Attack setup

Hijack attack

Three different scenarios were set in order to evaluate in which way and to what extent traffic is anomalous or not.

In the first there are two stations set with the same MAC address in order to see if there are features that signal this case. The second is a kind of not well mastered hijack attack: ATK starts to use legitimate STA MAC before the STA is actually torn down. The third one is a true hijack attack. ATK connects with STA MAC after STA is torn down.

FakeAP

These surveys were made in order to test SSID and AP detection features. FakeAP is a tool that creates and publishes a random number of AP and SSID by sending fake beacons. It creates new MAC and SSID from a predetermined list created by the user. To detect this kind of traffic, Beacon rate should be a good feature too.

In the next section detailed results are shown and analyzed.

4.2 Result analysis

The application stores results in text files for analysis. Each measure is stored along with MAC addressed it was taken from. Power, RTT, SN and rates are shown in graphs, others just by logging singular values.

In all surveys packet loss ratio reported by pcap library was roughly 1 on $100000 = 10^{-5}$ so 0.001%. There is no way to know if and how many packets were dropped by the card, but since data gathered are consistent it should be by all means very low (possibly lower than pcap one).

4.2.1 Data sessions analysis

Since Access Points were always at the same distance from WIDS, features relating them are first examined, then those about far and near data transmission will be presented.

Figures 4.3 and 4.4 show power measures. For each one, first graph shows time differences between subsequent frames, second the samples of signal power measured for each frame, third one mean values computed at each frame receipt[1] and last one the variance of the power computed alike the mean.

From graph is possible to see that mean and variance values are steady and do not present any sudden step. This behavior was very much the same in every survey.

Sequence number are shown in the same graph for comparison in figure 4.5. Complete SN cycles, from 0 to 4096, are done within few frames. The most plausible explanation for this behavior is that SN are incremented also for control frames, which do not have SN field.

Regarding stations, figure 4.6 shows power measures related to far station. As we can see measures are even more stable than AP one. Reason for this could be numerous:

[1] At each step mean is computed relating to previous frames and stored.

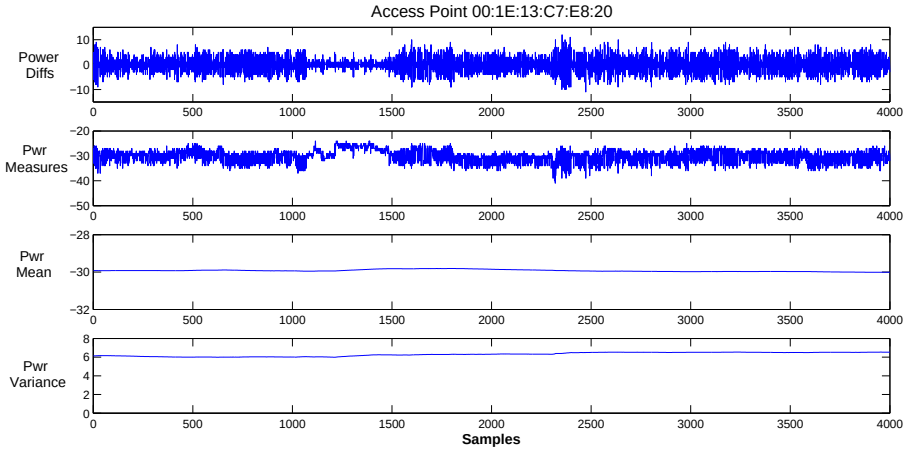

Figure 4.3: First AP measures

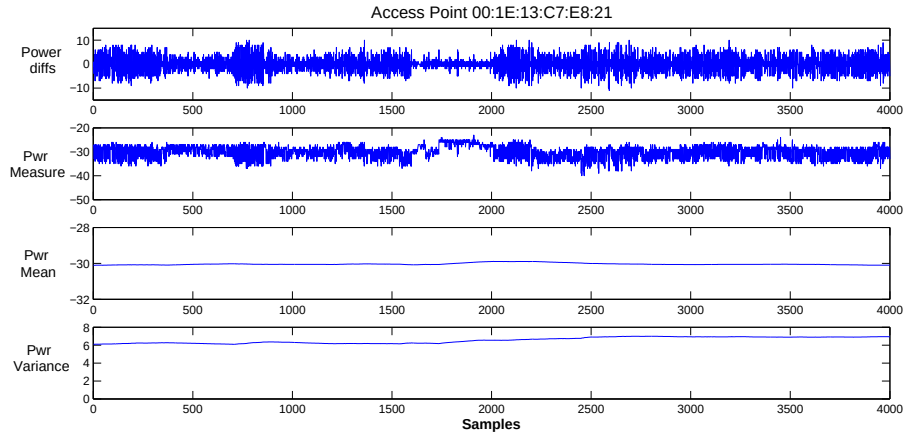

Figure 4.4: Second AP measures

- AP could have a power management profile more complex than station;
- The fact that STA and WIDS were in the same room could explain that;
- Multipath propagation of AP signal.

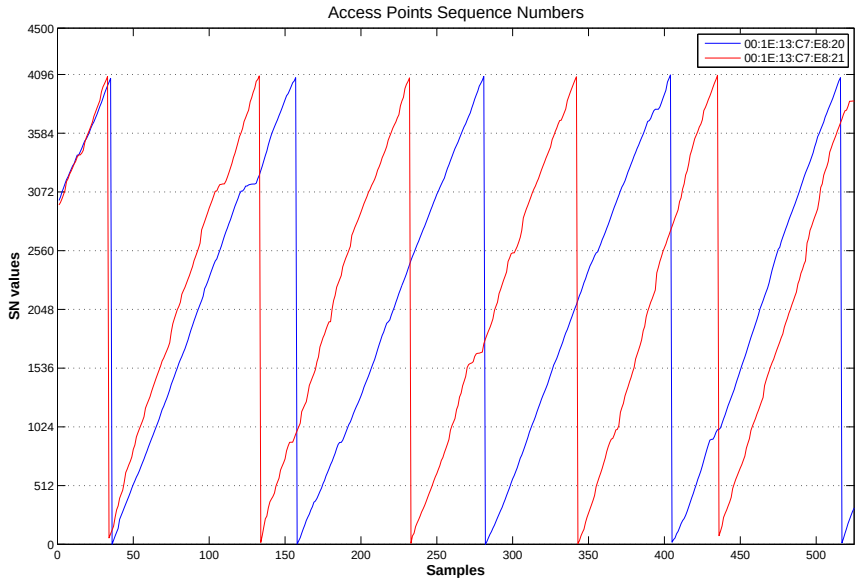

Figure 4.5: APs Sequence Number behavior comparison

Sequence numbers of STA are drawn in figure 4.7. As shown, in contrast to AP case, SN cycles are around 4000 samples long, as should be. Looking more in detail, cycles are longer than 4096, that is because sometimes sequence numbers are repeated through consecutive frames, that is probably because for retransmitted frames the sequence number remain unchanged[2].

Measurements in near station case were very similar. As shown in figure 4.8 mean and variance values are still steady although power measures present some spikes. These are probably due to vicinity of the STA. However such spikes does not alter mean and variance values.

Looking at SN values, they present very much the same behavior. In addition samples proposed in figure 4.9 shows a new behavior: just within first hundred samples there is a SN reset. This fall within unknown card behavior[3].

For completeness Beacon frame rate is reported for each access point and the

[2]See [8] part 7.1.3.4.1, Sequence Number Field.
[3]It is reported that wireless cards sometimes reset SN for unknown card specific reasons.

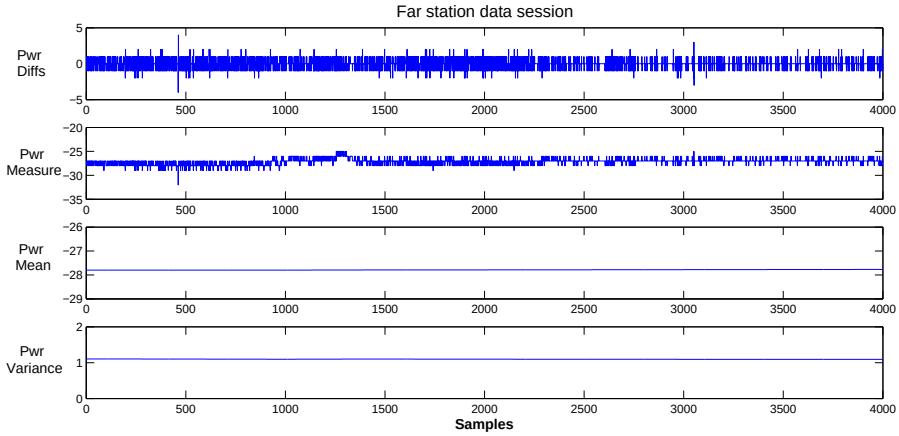

Figure 4.6: Far station measures

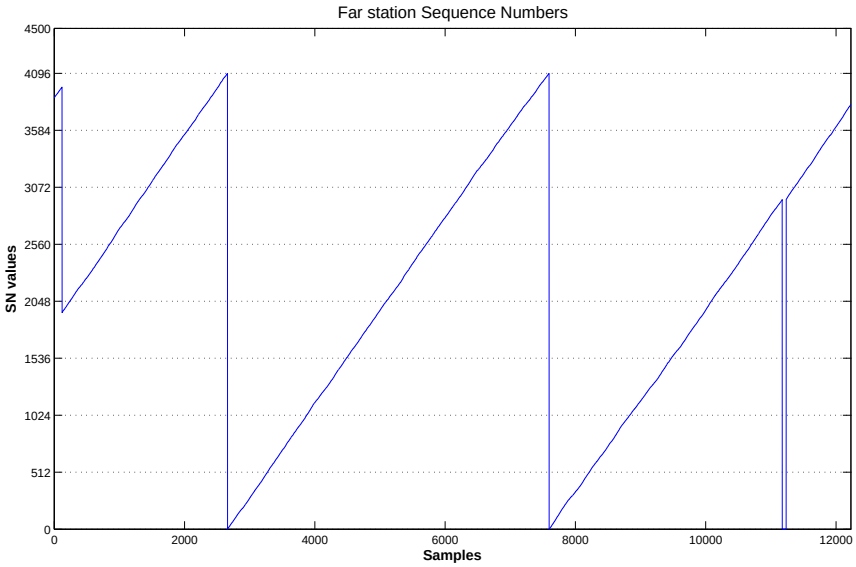

Figure 4.7: Far station SN values

cumulative one. Expected rate for each AP is 9.765625 since the interval value

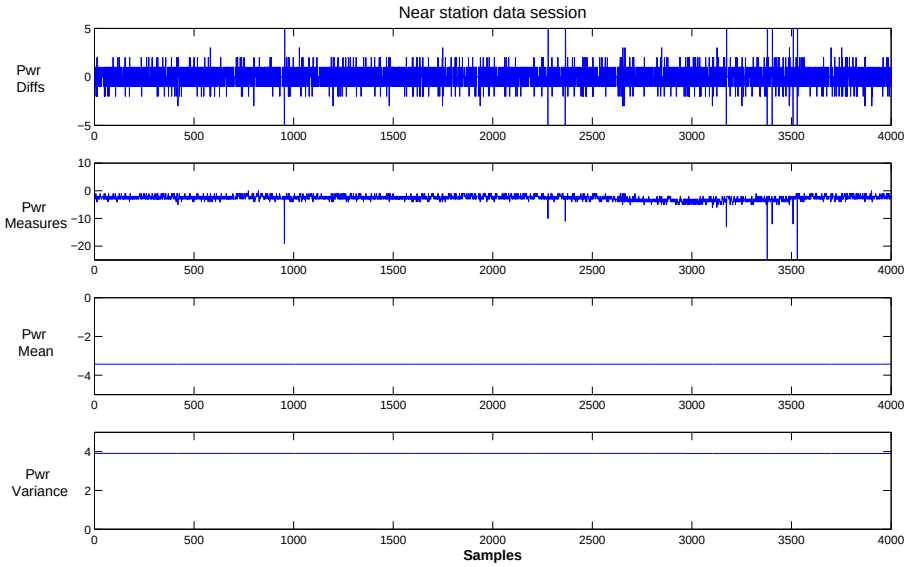

Figure 4.8: Near station measures

declared was 100 therefore:

$$\frac{1s}{100 \times 1024\mu s} = 9.765625 frame/sec$$

Rate figures are 4.10, 4.11 and 4.12.

Other rates are not reported because number of samples were insufficient for general analysis. Moreover, by traffic analysis, results that *RTS-CTS* procedure is never used for data transfers since data are fragmented in frames never exceeding *RTS threshold*.

During data surveys, RTT feasibility as good feature was assessed. Since RTT relies on *RTS-CTS* handshake mechanisms, STA not using this procedure are not traceable. All *RTS-CTS* handshakes were found to be initiated by the AP to ensure TCP ACKs delivering. Given this a different mechanism to compute RTT using AP initiated handshakes could be developed. Unfortunately *CTS* frames contain only RA address[4], thus making the problem more complex and very

[4]This is a huge security hole. Making a DoS attack that exploits TA absence is very easy and impossible to stop, since behavior in these cases is fully specified by standard.

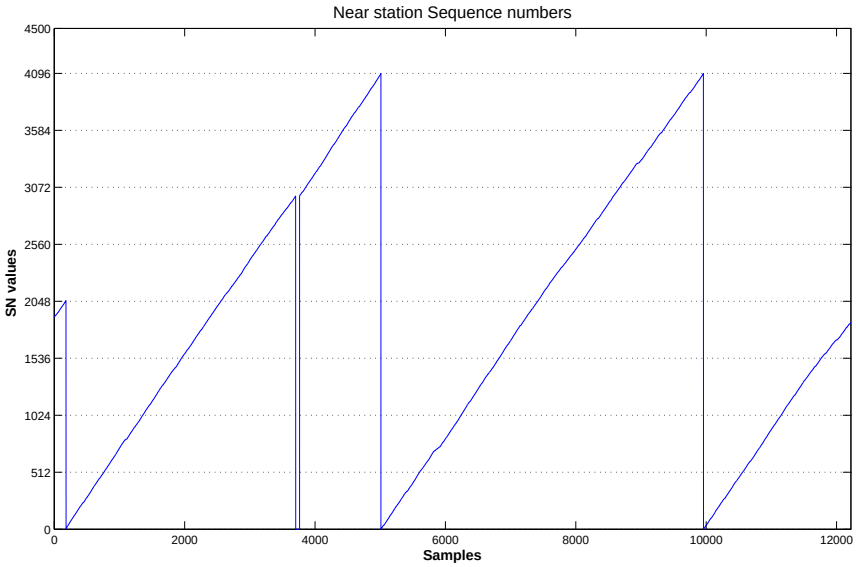

Figure 4.9: Near station SN values

difficult to solve, especially in a true environment where many STAs transmit together. Results are shown in figure 4.13. *CTS time* is the time passed between RTS and CTS receipt while *RTT* is the whole time between RTS and first DATA frame receipt. RTT measured belong to AP1.

4.2.2 Beacon frame session analysis

WIDS detected two AP present. Features for both AP were found to be nearly identical. That was strange nad after analysis was found that both AP were actually one AP, managing two networks with two different MAC addresses owned by the same device. SN analysis was decisive in this as each beacon frame from one MAC were found to be all odd values, and from the other even ones. That is because in absence of STAs, AP sends only beacon frames, and since the beacon interval has the same value, subsequents beacons belong to different networks, thus having subsequent SN. Results are shown in figures 4.14 4.15 and 4.16.

For better understanding follows a sample of the two sequence number values:

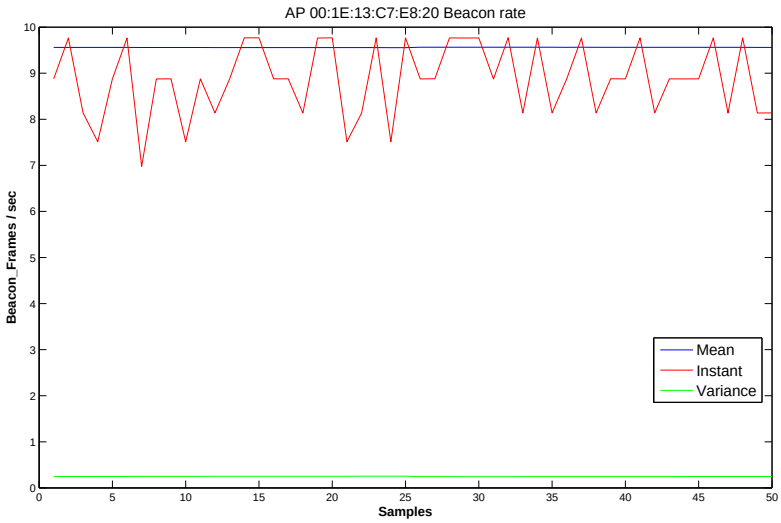

Figure 4.10: Beacon Frame rate for AP1

$MAC1 : [3537, 3539, 3541, 3543, 3545, 3547, 3549, 3551, 3553, 3555, 3557, 3559, 3561, 3563...]$
$MAC2 : [3536, 3538, 3540, 3542, 3544, 3546, 3548, 3550, 3552, 3554, 3556, 3558, 3560, 3562...]$

Beacon rate samples are shown in figures 4.17, 4.18 and 4.19. Measures are more stable than ones taken during data session. Sometimes instant beacon rate drops for few samples. That is because in the environment were present also 3 uncontrolled STAs. However during the survey they transmitted only five to fifty frames while beacon frames detected were thirty thousands therefore analysis can be considered accurate enough. This explains also why within SN samples there is a sample that suddenly drop to zero. That must be a control frame that by standard has no SN field.

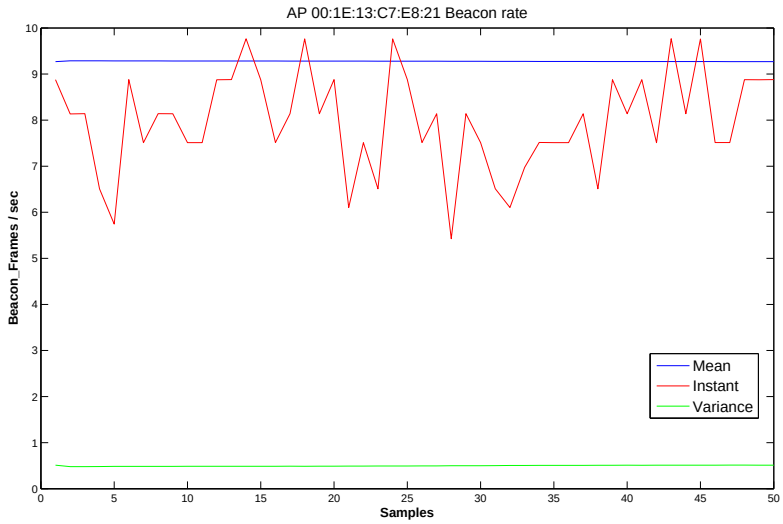

Figure 4.11: Beacon Frame rate for AP2

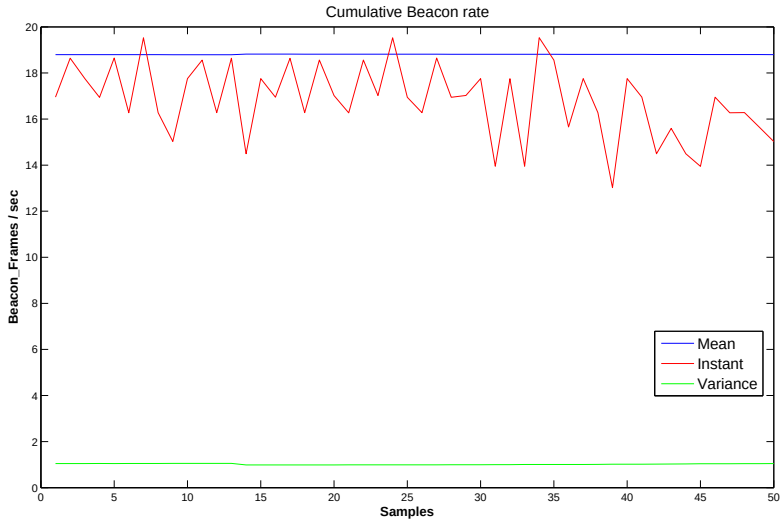

Figure 4.12: Cumulative Beacon Frame rate

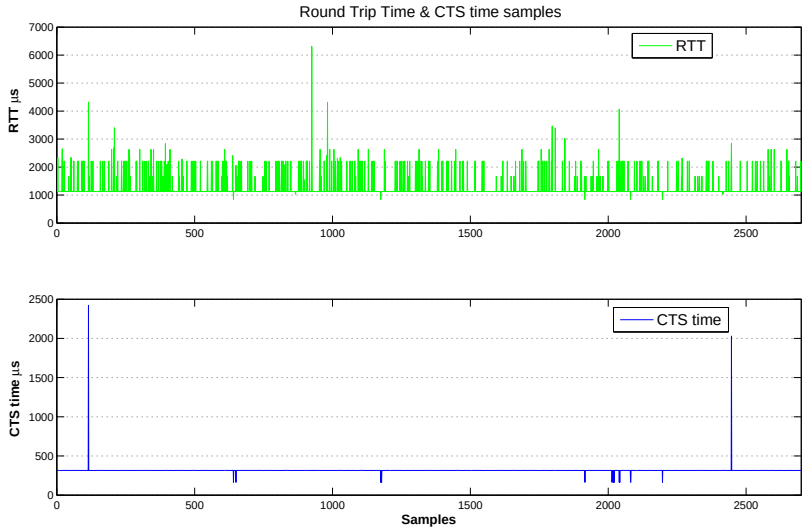

Figure 4.13: RTT AP1 Data

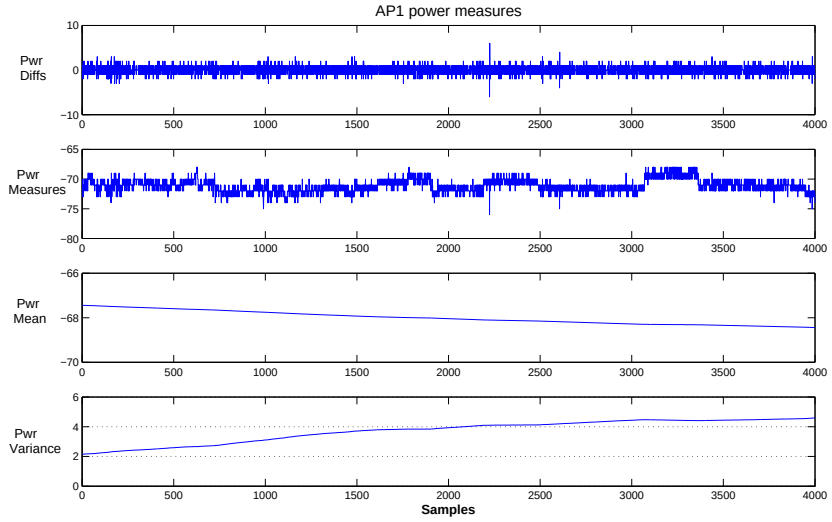

Figure 4.14: MAC1 power data

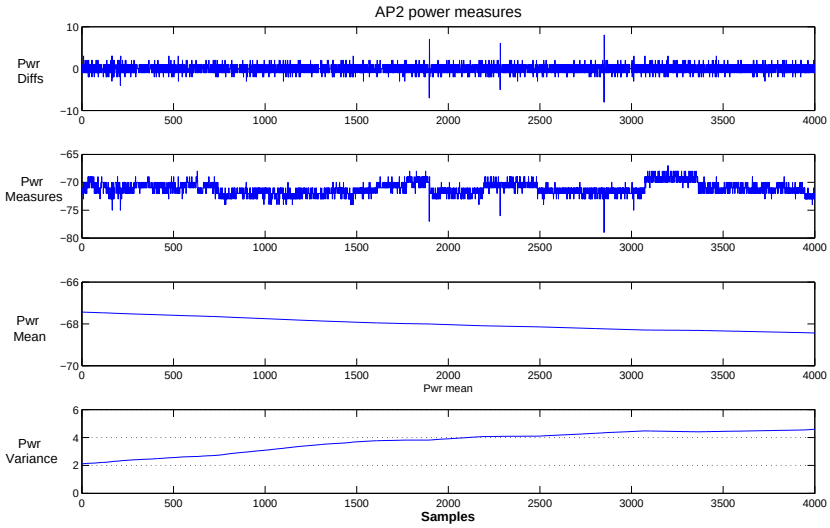

Figure 4.15: MAC 2 power data

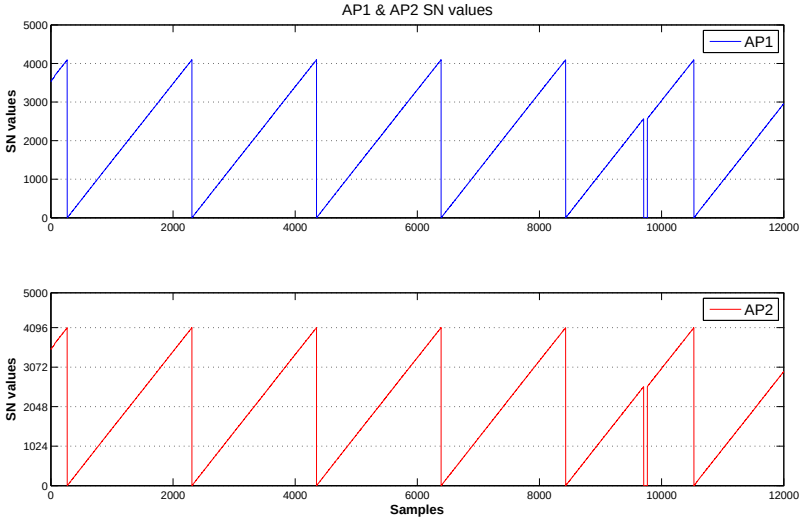

Figure 4.16: Sequence number comparison

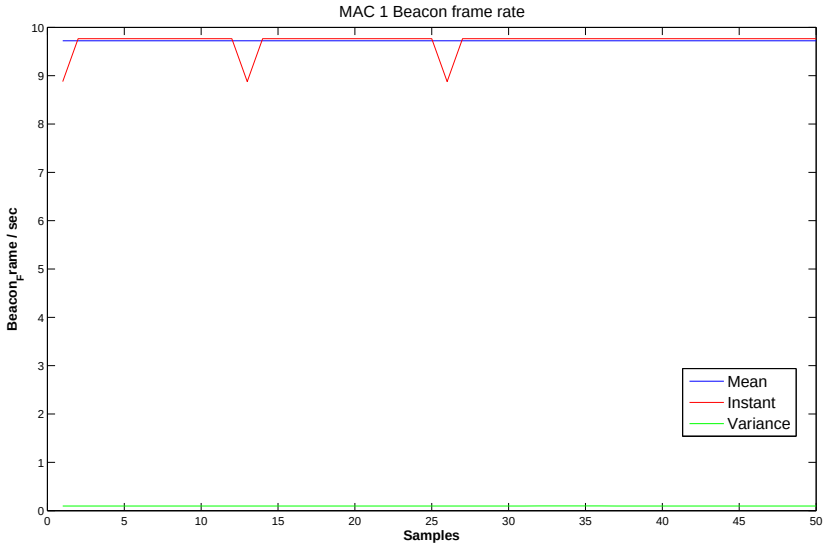

Figure 4.17: MAC1 Beacon Rate

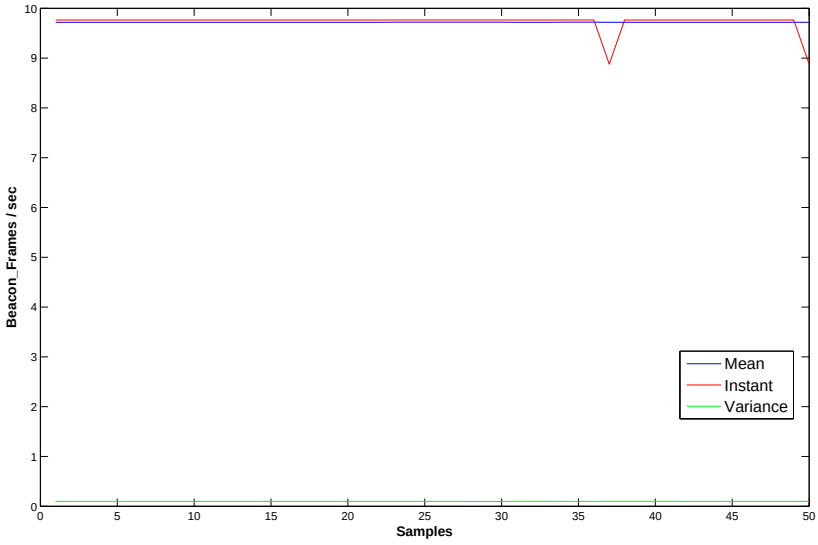

Figure 4.18: MAC2 Beacon rate

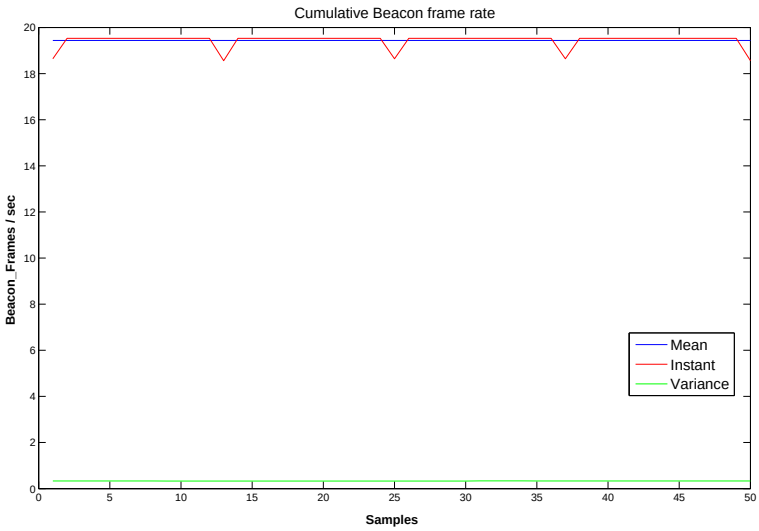

Figure 4.19: Cumulative Beacon rate

4.2.3 Hijack attack analysis

Results will be presented for all three tested scenarios.

Two STA transmitting with same MAC address at the same time

Results from two different surveys are presented in figures 4.20, 4.21, 4.22 and 4.23.

As shown, variance measures are quite different from one gathered in normal data sessions. Values are higher and less steady, although after roughly 1000 samples variance become steadier but still with anomalous high values.

Sequence Numbers analysis is as much interesting as power one. As seen both the surveys present numerous subsequent steps. That is because each wireless card increment its own SN. Looking carefully at sequences is possible to see that there are two different series. This phenomenon is more visible in figure 4.23. This is without doubt an anomaly and detecting it seems very accurate and easy since values differences are remarkable.

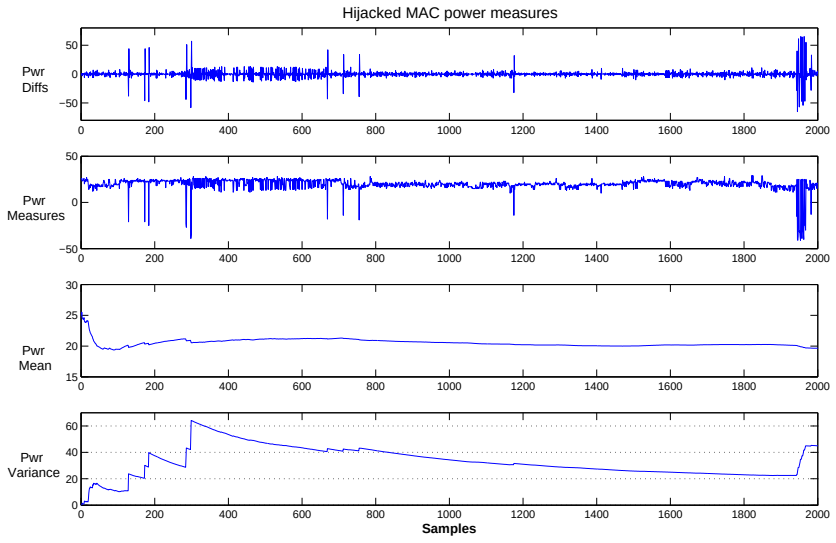

Figure 4.20: Hijacked MAC power measures - 1st survey

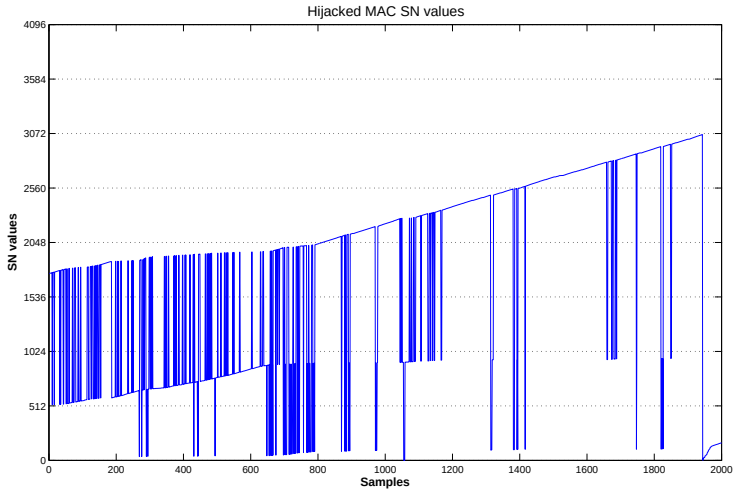

Figure 4.21: Hijacked MAC SN - 1st survey

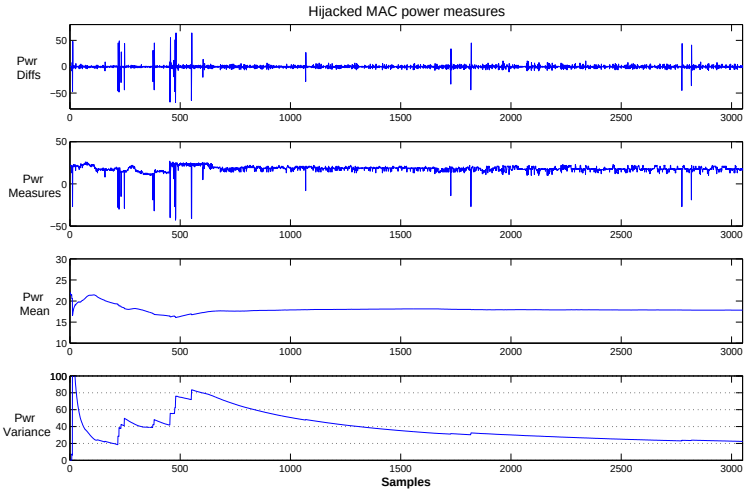

Figure 4.22: Hijacked MAC power measures - 2nd survey

Two STA transmitting with same MAC address at the same time for few frames only

Results of two different surveys are presented in figure 4.24, 4.25, 4.26 and 4.27. In this case we still have high values of variance in power samples. However

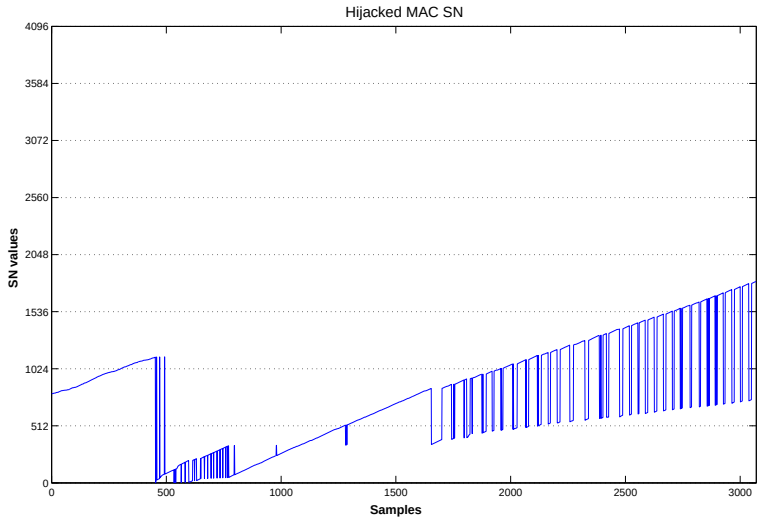

Figure 4.23: Hijacked MAC SN - 2nd survey

here values are more erratic then previous case. Single steps are present and they arise after relatively steady samples, see figure 4.24 around sample 2600. Sequence Numbers shows a slightly different behavior too. Subsequent steps now are present only in few frames compared to the previous case. Is interesting that variance step and SN steps appears around the same samples. Therefore these two features can be cross referenced to increase the accuracy of attack detection.

Hijack attack: stations transmitting one at a time

Results from two surveys are shown in figure 4.28, 4.29, 4.30 and 4.31. Both surveys present the same behavior. Power measures show a sudden growth of the variance right after the attack while SN values just reset to zero. Comparing these measurements with above ones, variance growth is fast and continuous, it start from normal variance values (around 5) and goes up to 50 with a constant slope. Is true that also some surveys reported before shows high values of variance, however those are erratic and they not present such regular slopes. Sequence number analysis shows that at the time the attack is carried out, the sequence number just resets from its value to zero. These is not a profile that could be used to detect the attack since good attackers can easily match attacked station SN. However power measures seem to be a very effective feature

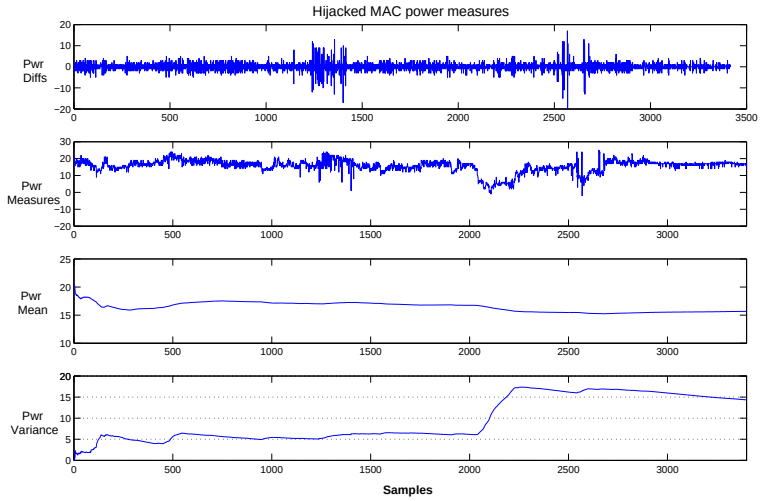

Figure 4.24: Hijacked MAC power measures - 3rd survey

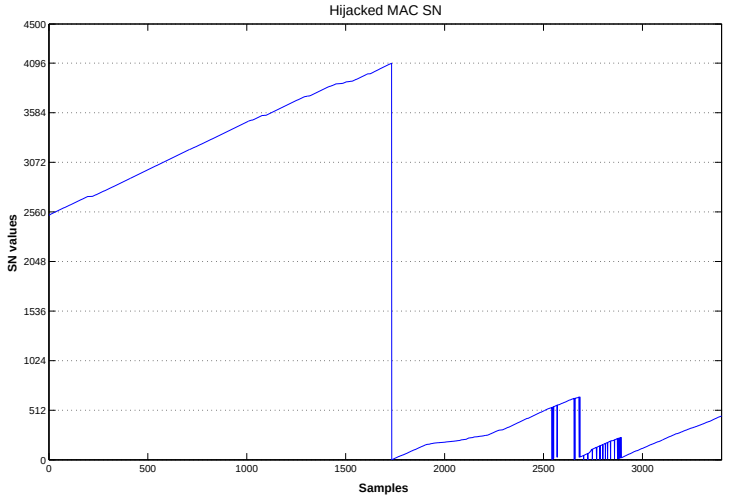

Figure 4.25: Hijacked MAC SN - 3rd survey

since they present very specific behavior, moreover as already explained is very complex and difficult to adjust power in order to fool the WIDS.

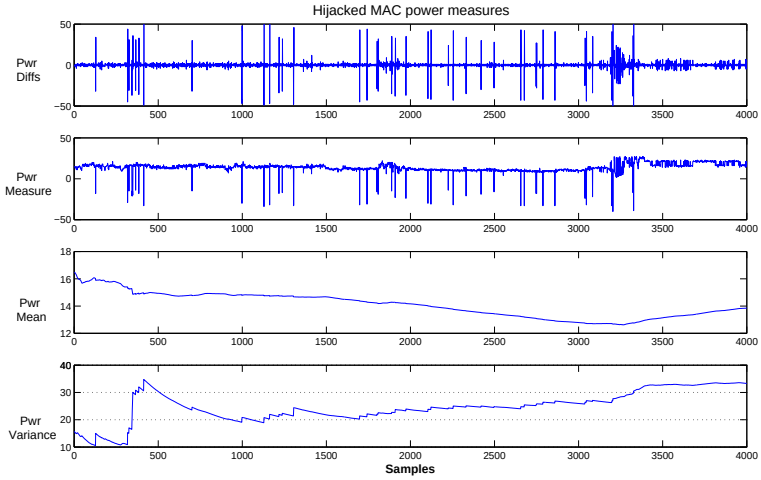

Figure 4.26: Hijacked MAC power measures - 4th survey

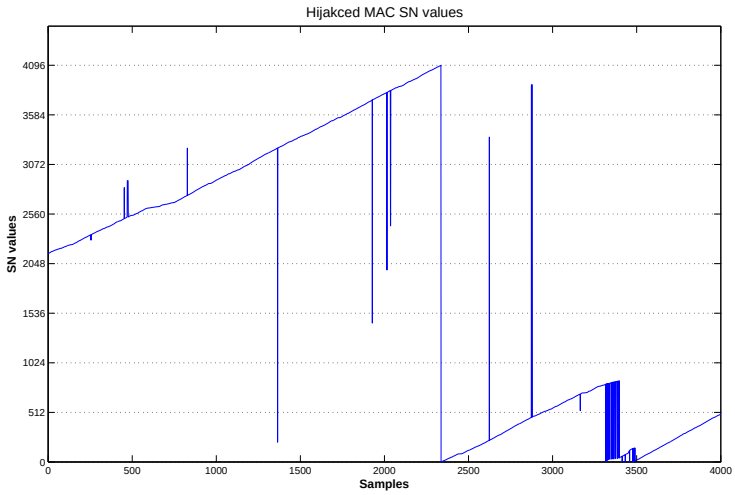

Figure 4.27: Hijacked MAC SN - 4th survey

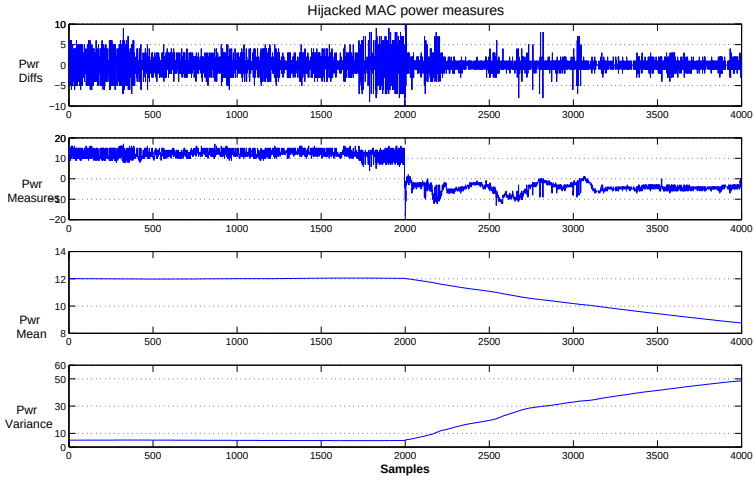

Figure 4.28: Hijacked MAC power measures - 5th survey

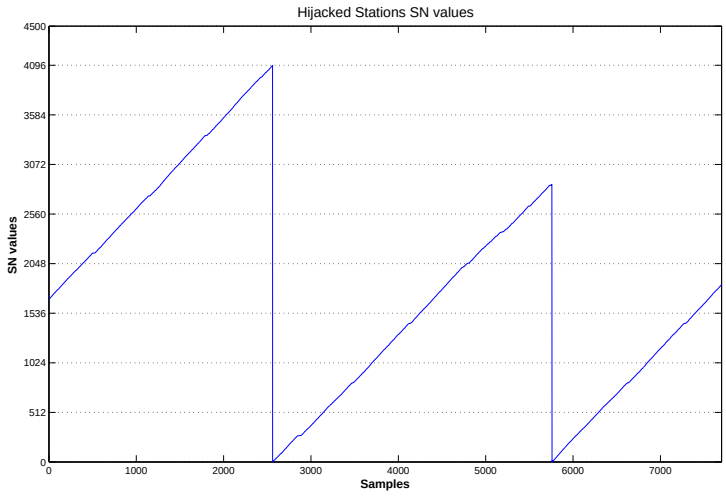

Figure 4.29: Hijacked MAC SN values - 5th survey

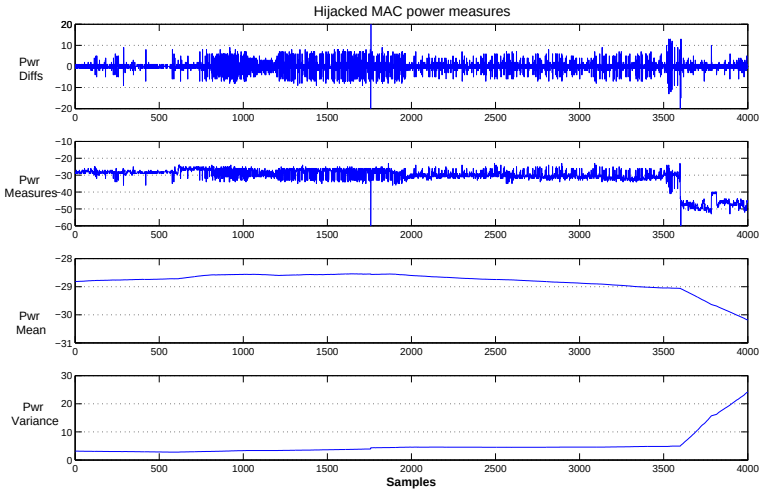

Figure 4.30: Hijacked MAC power measures - 6th survey

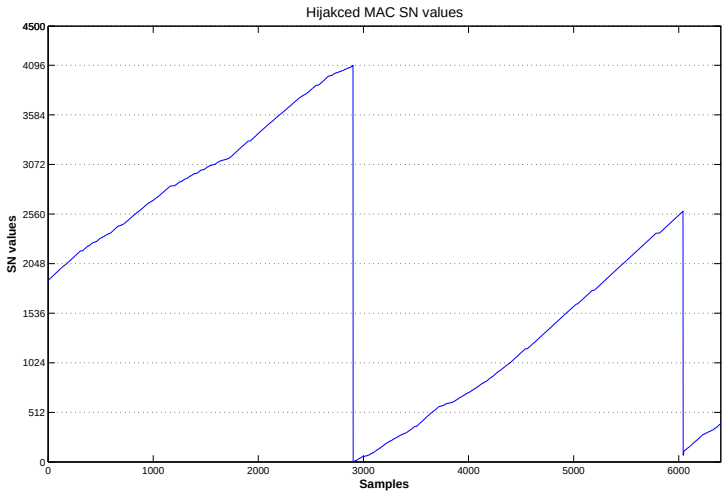

Figure 4.31: Hijacked MAC SN values - 6th survey

4.2.4 FakeAP analysis

This survey use different features from those employed until now. It analyze SSID, BSSID and Rate features. Two surveys were performed:

1. With a limited number of SSID and AP MAC;
2. With unlimited number of SSID and AP MAC.

Data gathered from first one shows the presence of four more SSIDs, with 40 AP each, that is an impossible number, since interference would be great. Cross referencing it with beacon rates, data shows that AP never comply with Beacon interval stated in Beacon frames.

Second simulation generated hundreds of SSID along with different BSSID. A different MAC address was generated for each SSID. WIDS successfully detected 1146 different AP MAC addresses, along with their SSID. Although cross referencing these data with Beacon frame rate made clear that those AP are fake since beacon rate resulted to be zero for each anomalous AP.

Detailed results are reported in appendix A

4.3 Requirements assessment

In this section requirement fulfillment is checked against requirements listed in chapter 2.

4.3.1 Satisfied requirements

Here met and partially met requirements are listed along with explanation of why some of them were not completely met. Notice that requirements shown in chapter 2 are for a general WIDS application.

Functionality

After testing successfully attack that are very likely to be detected are:

- Session Hijack;
- MITM that exploit Identity Theft;
- Rogue AP;
- DoS attacks that exploits control and management flood;
- Attacks that exploits specific frame types flood.

The first one is met completely along with the second one since that kind of attack begin with hijacking the station. The third one is feasible as long as the network infrastructure is known. SSID thread data can be used to monitor legitimate AP identities. Last two ones are only partially satisfied since no survey were made in order to check detection capabilities. However since beacon frame rate is relatively steady, attacks that increase the beacon rate should be easily detectable.

Usability

This requirement was partially satisfied. The application is not completely independent. It just gather information and provides them in separate files in order to be analyzed. It runs independently although it just monitor a fixed number of frames, decided by the user at runtime. Anyway, after training and programming a thread to report anomalies, modifications to apply in order to make a proper WIDS are few and not complex.

Reliability

Most important and critical reliability requirements were completely satisfied:

- **Low packet loss**: around 0.001%[5];
- **Correct info gathering and parsing**: during program stage all field values were checked against known ones[6];
- **Precise time reference**: by *MACtime* field use.
- **Feature accuracy**: both rate and power measures result accurate enough for anomaly detection. RTT were accurate but inconclusive.

[5] therefore virtually zero.
[6] These were gathered at the same time using Wireshark [7] or Tcpdump [3].

On the other hand requirements partially satisfied are:

- **Attacker traceability**: MAC is used to identify attackers. This is only a logic association therefore very weak for real traceability. However it permits to keep track of attacker activities as far as they MAC layer activities are logged.
- **WIDS immunity from DoS attack**: the only DoS attack the WIDS could be vulnerable to is by traffic overload. Test made with available hardware showed that the WIDS cannot analyze traffic in real time. However this should be easily solved by using more powerful hardware[7].

Performance

The only requirement partially satisfied is **Real time**. Simulations showed that as long as frame rate was low (i.e. web surfing) the threaded application managed to process all the information in real-time. However by increasing the traffic (i.e. ftp download) the application couldn't stand the rate. To see whether or not this is a design problem, another application with a single thread for data processing was developed. Results were a little bit better but still definitely not real time. Still hardware improvements should fix the problem.

Supportability

Satisfied requirements:

- **Monitoring of features despite network topology and infrastructure**: chosen features are common to every type of *infrastructure* network, despite its access points or its security policies;
- **Use of Device independent libraries**: this requirement was already addressed in chapter 3. Pcap library can interface with every type of 802.11 device therefore its use makes the application device independent.
- **Device independent packet capture engine**: since the packet capture engine uses Pcap this requirement is implicitly satisfied.

Partially satisfied requirements:

[7] For hardware description see appendix B at page 79.

- **Common 802.11 device support**: this is satisfied as long as the device supports radiotap header.
- **Device independent parsing functionalities**: This is satisfied as long as the device supports radiotap header with required fields[8].

4.3.2 Unsatisfied requirements

As said in chapter 2 some requirements were not addressed since not in the purpose of the project. Such requirements are:

- Fast attack detection;
- Fast alarm signaling.

Since no monitor thread has been developed it has not been possible to assess such requirements.

[8] Each device support a different set of fields. There are some that does not provide *TFST* field, used as time reference by the application. Required fields are listed at page 24.

Chapter 5

Conclusions and Future work

Wireless networks have shown several security holes and many weaknesses since the standard has been released. Such weaknesses come directly from design and security algorithms flaws and from undefined behavior due to unusual frame values. Moreover wireless promotes malicious activity since network is no more confined into the wires but is spread over the air out in the open, beyond houses or companies walls.

Anomaly based Intrusion Detection Systems have proved to be very effective since they detect suspicious traffic just by comparison with the normal one. The purpose of this project is to show that employing accurate and precise features of different communication layers, and cross referencing their data, could be very effective in detecting attacks that usually are very hard to detect. As a matter of fact such attacks usually exploit single layer vulnerabilities, therefore anomaly detection with single layer features can be often useless.

A monitor application has been developed and it has been used to gather data from different prepared scenarios. Afterward analysis has shown that anomalies in gathered traffic were remarkable, consistent and rather easy to track. Along with this performance has been evaluated in terms of real time application and packet loss. Moreover requirements of a fully functional anomaly based intrusion detection system were assessed. Despite hardware availability for developing and testing was limited, results are promising and they show that an effective

Wireless IDS can be developed.

Future work

There are many aspects that have not been taken under consideration during application design and development, and others that could be improved. Listing and describing all of them would be long and, since most of them are not directly related to this project, useless.

Most interesting and intriguing ones are:

- Support of other type of headers, i.e. Prism;
- Development and implementation of a modified version of *PF_RING* libraries with a view to newer standards as *IEEE 802.11n* that defines higher bandwidths up to 600 Mbit/s;
- Use more than one wireless card to monitor also other channels, in order to have more consistent data, thus using channel information that is already parsed but not actually used;
- Development of a distributed WIDS in order to cross reference traffic measures from different areas and design of features that arise from this architecture;
- Find and test different ways than RTS-CTS to compute RTT and see if it can be a decisive feature as reported in [18, 20].

Appendix A

Detailed results

This appendix show detailed data for FakeAP, Beacon sessions. A survey with RTT data is also shown. Hijack attack and Data session are not reported since results are already shown in chapter 4 figures.

A.1 FakeAP

Detailed results for a limited number of SSID are shown.

4 SSID session

SSIDs list

- SSID:tsunami - AP number:42 AP_MACs: 00:00:0C:6C:7F:F0 00:00:CE:9E:A1:74 00:00:CE:47:9F:F4 00:00:CE:E9:98:0E ...
- SSID:airport - NumeroAP:32 AP_MACs: 00:00:CE:8A:8E:A2 00:00:CE:9B:3A:5F 00:00:CE:83:15:59 00:00:CE:93:57:A4 ..

- SSID:host - NumeroAP:36 AP_MACs: 00:00:0C:73:25:8D 00:00:CE:93:F7:4F 00:00:CE:18:34:20 00:00:0C:84:92:E1 ...
- SSID:Access Point NumeroAP:36 AP_MACs: 00:00:CE:DD:C2:D4 00:00:0C:12:33:93 00:00:0C:FC:7E:E1 00:00:0C:13:57:0C ...

BSSID section

```
Number of BSSID detected:148
List of BSSID:

00:1C:0E:42:75:91
00:1C:0E:42:75:90
00:00:0C:6C:7F:F0
00:00:CE:8A:8E:A2
00:00:CE:9E:A1:74
00:00:0C:73:25:8D
00:00:CE:47:9F:F4
00:00:CE:9B:3A:5F
00:00:CE:93:F7:4F
00:00:CE:83:15:59
00:00:CE:18:34:20
00:00:CE:93:57:A4
00:00:0C:84:92:E1
00:00:CE:E9:98:0E
00:00:CE:59:3B:61
00:00:0C:10:75:2F
00:00:CE:8B:BB:47
00:00:CE:30:92:56
...
```

Rate section

```
AP:"00:00:CE:9E:A1:74" Interval:100 Expected rate:9.765625

Total Frames:10 Mean:3.056685 Instant:3.056685 Var:0.000000 Num samples:1


Mean: 3.056685

Instant: 3.056685
```

```
Var: 0.000000

AP:"00:00:0C:73:25:8D" Interval:100 Expected rate:9.765625

Total Frames:10 Mean:2.059904 Instant:2.059904 Var:0.000000 Num samples:1


Mean: 2.059904

Instant: 2.059904

Var: 0.000000

AP:"00:00:CE:47:9F:F4" Interval:100 Expected rate:9.765625

Total Frames:10 Mean:1.542822 Instant:1.542822 Var:0.000000 Num samples:1


Mean: 1.542822

Instant: 1.542822

...
```

1146 SSID session

SSIDs list

- SSID:mistymountain NumeroAP:1 AP_MACs: 00:40:33:E7:F9:4D
- SSID:functions NumeroAP:1 AP_MACs: 00:02:B3:6F:83:EF
- SSID:demo2088 NumeroAP:1 AP_MACs: 00:02:A5:18:86:D5
- SSID:libp NumeroAP:1 AP_MACs: 00:40:AE:B5:A7:BC
- SSID:bathshua NumeroAP:1 AP_MACs: 00:40:96:8D:B4:8A
- SSID:campbllbks2-mil-tac NumeroAP:1 AP_MACs: 00:80:C6:1D:DC:3E
- SSID:deserve NumeroAP:1 AP_MACs: 00:40:96:13:47:47
- ...

BSSID section

```
Number of BSSID detected:1146
List of BSSID:

00:1C:0E:42:75:90
00:1C:0E:42:75:91
00:40:33:E7:F9:4D
00:02:B3:6F:83:EF
00:02:A5:18:86:D5
00:40:AE:B5:A7:BC
00:40:96:8D:B4:8A
00:80:C6:1D:DC:3E
00:40:96:13:47:47
00:80:C8:29:FE:8C
00:40:96:D6:BA:C0
00:05:86:EC:76:A5
00:02:A5:C4:86:7C
00:04:5A:BD:02:DA
00:04:25:EB:85:33
00:02:B3:5C:AD:66
00:40:AE:7B:61:CC
00:90:96:41:A6:2B
00:40:AE:1E:68:0D
...
```

Rate section

```
AP:"00:40:33:E7:F9:4D" Interval:100 Expected rate:9.765625

Total Frames:2 Mean:0.000000 Instant:0.000000 Var:0.000000 Num samples:0

Sample arrays:

Mean:

Instant:

Var:
```

```
AP:"00:02:B3:6F:83:EF" Interval:100 Expected rate:9.765625

Total Frames:2 Mean:0.000000 Instant:0.000000 Var:0.000000 Num samples:0

Sample arrays:

Mean:

Instant:

Var:

AP:"00:02:A5:18:86:D5" Interval:100 Expected rate:9.765625

Total Frames:2 Mean:0.000000 Instant:0.000000 Var:0.000000 Num samples:0

Sample arrays:

Mean:

Instant:

Var:

...
```

A.2 Beacon Session

SSIDs list

- SSID:DTU Wireless NumeroAP:1 AP_MACs: 00:1E:13:C7:E8:20
- SSID:eduroam NumeroAP:1 AP_MACs: 00:1E:13:C7:E8:21

BSSID section

```
Number of BSSID detected:2
List of BSSID:
```

00:1E:13:C7:E8:20
00:1E:13:C7:E8:21

Rate section

Beacon Section

Total Frames:29617 Mean:19.438297 Instant:19.531097 Var:0.326309 Num samples:1480

Sample arrays:

Mean: 19.437725 19.437789 19.437853 19.437917 19.437981 19.438045 19.438109
19.438173 19.438236 19.438300 19.438364 19.438427 19.437826 19.437890
19.437954 19.438017 19.438081 19.438144 19.438207 19.438270 19.438333
19.438396 19.438459 19.438522 19.437983 19.438046 19.438109 19.438172
19.438235 19.438297 19.438340 19.438405 19.438470 19.438534 19.438599
19.438663 19.438109 19.438174 19.438238 19.438303 19.438367 19.438432
19.438496 19.438560 19.438624 19.438688 19.438752 19.438816 19.438879
19.438273

Instant: 18.642787 19.531097 19.531040 19.531136 19.531097 19.531097
19.531078 19.531097 19.531078 19.531097 19.531040 19.531136 18.559470
19.531097 19.531097 19.531078 19.531097 19.531097 19.531078 19.531097
19.531097 19.531078 19.531097 19.531078 18.642804 19.531078 19.531097
19.531097 19.531078 19.531097 19.531078 19.531097 19.531097 19.531078
19.531097 19.531097 18.642769 19.531097 19.531097 19.531021 19.531155
19.531097 19.531078 19.531097 19.531097 19.531078 19.531097 19.531097
19.531078 18.559470

Var: 0.331701 0.331479 0.331257 0.331035 0.330813 0.330592 0.330371
0.330151 0.329930 0.329710 0.329490 0.329271 0.329574 0.329354 0.329135
0.328917 0.328699 0.328481 0.328263 0.328045 0.327828 0.327611 0.327395
0.327178 0.327386 0.327170 0.326954 0.326739 0.326524 0.326309 0.334811
0.334583 0.334356 0.334129 0.333902 0.333675 0.333884 0.333657 0.333432
0.333206 0.332981 0.332756 0.332531 0.332307 0.332083 0.331859 0.331636
0.331412 0.331190 0.331494

End Beacon section

```
**Beacon Rate by single AP section**


AP:"00:1E:13:C7:E8:20" Interval:100 Expected rate:9.765625

Total Frames:14810 Mean:9.721788 Instant:9.765539 Var:0.095470 Num samples:1481

Sample arrays:

Mean: 9.722107 9.722137 9.722167 9.722197 9.722227 9.722256 9.722286
9.722316 9.722345 9.722375 9.722405 9.722434 9.721857 9.721887 9.721916
9.721946 9.721976 9.722005 9.722035 9.722065 9.722094 9.722124 9.722153
9.722183 9.722212 9.721640 9.721670 9.721699 9.721729 9.721759 9.721788
9.722151 9.722181 9.722211 9.722242 9.722272 9.722302 9.722332 9.722362
9.722392 9.722422 9.722452 9.722482 9.722511 9.722541 9.722571 9.722601
9.722630 9.722660 9.722690

Instant: 8.877778 9.765549 9.765539 9.765549 9.765511 9.765577 9.765549
9.765539 9.765549 9.765511 9.765577 9.765549 8.877770 9.765539 9.765549
9.765520 9.765568 9.765549 9.765539 9.765549 9.765549 9.765539 9.765549
9.765520 9.765568 8.877770 9.765549 9.765539 9.765549 9.765549 9.765539
9.765568 9.765549 9.765539 9.765549 9.765539 9.765549 9.765549 9.765520
9.765568 9.765549 9.765539 9.765549 9.765549 9.765539 9.765549 9.765520
9.765568 9.765549 9.765530

Var: 0.096425 0.096360 0.096295 0.096230 0.096165 0.096100 0.096036
0.095971 0.095907 0.095842 0.095778 0.095714 0.096135 0.096071 0.096007
0.095943 0.095879 0.095815 0.095751 0.095687 0.095623 0.095559 0.095496
0.095432 0.095369 0.095787 0.095723 0.095660 0.095597 0.095533 0.095470
0.097183 0.097116 0.097050 0.096984 0.096917 0.096851 0.096785 0.096719
0.096653 0.096588 0.096522 0.096456 0.096391 0.096325 0.096260 0.096195
0.096130 0.096064 0.095999


AP:"00:1E:13:C7:E8:21" Interval:100 Expected rate:9.765625

Total Frames:14807 Mean:9.720059 Instant:9.765549 Var:0.097704 Num samples:1480

Sample arrays:
```

```
Mean: 9.719150 9.719182 9.719214 9.719246 9.719278 9.719309 9.719341
9.719373 9.719404 9.719436 9.719468 9.719499 9.719531 9.719562 9.719593
9.719625 9.719656 9.719687 9.719719 9.719750 9.719781 9.719812 9.719843
9.719874 9.719905 9.719936 9.719967 9.719998 9.720028 9.720059 9.719742
9.719774 9.719806 9.719838 9.719870 9.719902 9.719316 9.719348 9.719380
9.719412 9.719444 9.719476 9.719508 9.719540 9.719572 9.719604 9.719635
9.719667 9.719699 9.719118

Instant: 9.765539 9.765549 9.765549 9.765539 9.765549 9.765539 9.765549
9.765549 9.765539 9.765549 9.765549 9.765530 9.765558 9.765539 9.765549
9.765549 9.765539 9.765549 9.765549 9.765539 9.765549 9.765539 9.765549
9.765549 9.765539 9.765549 9.765549 9.765539 9.765549 9.765549 9.765577
9.765539 9.765549 9.765549 9.765539 9.765549 8.877770 9.765530 9.765558
9.765549 9.765539 9.765549 9.765539 9.765549 9.765549 9.765520 9.765568
9.765539 9.765549 8.877770

Var: 0.099615 0.099548 0.099480 0.099414 0.099347 0.099280 0.099213
0.099147 0.099080 0.099014 0.098947 0.098881 0.098815 0.098749 0.098683
0.098617 0.098551 0.098486 0.098420 0.098355 0.098289 0.098224 0.098159
0.098093 0.098028 0.097963 0.097898 0.097833 0.097769 0.097704 0.099990
0.099922 0.099853 0.099785 0.099717 0.099649 0.100073 0.100005 0.099937
0.099869 0.099801 0.099733 0.099666 0.099598 0.099531 0.099463 0.099396
0.099329 0.099262 0.099682



**END Beacon Rate by single AP section**
```

Sequence Numbers

```
MAC:00:1E:13:C7:E8:20 Counter:14824
Sequence Numbers:
3536 3538 3540 3542 3544 3546 3548 3550 3552 3554 3556 3558 3560 3562
3564 3566 3568 3570 3572 3574 3576 3578 3580 3582 3584 3586 3588 3590
3592 3594 3596 3598 3600 3602 3604 3606 3608 3610 3612 3614 3616 3618
3620 3622 3624 3626 3628 3630 3632 3634 3636 3638 3640 3642 3644 3646
3648 3650 3652 3654 3695 3697 3699 3701 3703 3705 3707 3709 3711 3713
3715 3717 3719 3721  3723 3725 3727 3729 3731 3733 3735 3737 3739 3741
3743 3745 3747 3749 3751 3753 3755 3757 3759 3761 3763 3764 3766 3768
3770 3772 3774 3776 3778 3780 3782 3784 3786 3788 3790 3792 3794 3796
```

```
3798 3800 3802 3804 3806 3808 3810 3812 3814 3816 3818 3820 3822 3824
3826 3828 3830 3832 3834 3836 3838 3840 3842 3844 3846 3848 3850 3852
3854 3856 3858 3860 3862 3864 3866 3868 3870 3872 3874 3876 3878 3880
3882 3884 3886 3888 3890 3892 3894 3896 ...

MAC:00:1E:13:C7:E8:21 Counter:14810
Sequence Numbers:
3537 3539 3541 3543 3545 3547 3549 3551 3553 3555 3557 3559 3561 3563
3565 3567 3569 3571 3573 3575 3577 3579 3581 3583 3585 3587 3589 3591
3593 3595 3597 3599 3601 3603 3605 3607 3609 3611 3613 3615 3617 3619
3621 3623 3625 3627 3629 3631 3633 3635 3637 3639 3641 3643 3645 3647
3649 3651 3653 3655 3680 3682 3684 3686 3688 3690 3692 3694 3696 3698
3700 3702 3704 3706  3708 3710 3712 3714 3716 3718 3720 3722 3724 3726
3728 3730 3732 3734 3736 3738 3740 3742 3744 3746 3748 3750 3752 3754
3756 3758 3760 3762 3765 3767 3769 3771 3773 3775 3777 3779 3781 3783
3785 3787 3789 3791 3793 3795 3797 3799 3801 3803 3805 3807 3809 3811
3813 3815 3817 3819 3821 3823 3825 3827 3829 3831 3833 3835 3837 3839
3841 3843 3845 3847 3849 3851 3853 3855 3857 3859 3861 3863 3865 3867
3869 3871 3873 3875 3877 3879 3881 3883 ...
```

A.3 RTT data

```
Number of devices detected:4

MAC  Counter RTS counter CTS counter LastStatus mean var CTS mean CTS var
errors RTS CTS DATA

00:1E:13:C7:E8:20 2779 3770 2785 RTS_Received 1250.322418 139507.372837
316.107953 2771.168627 988 6 22854


CTS samples: 315 316 316 315 316 316 316 316 315 316 315 315 315 315
314 315 315 315 315 315 316 316 316 316 316 316 316 316 316 316 315
315 316 316 316 316 315 315 316 316 316 316 316 315 315 315 316 315
315 315 315 315 315 315 316 316 316 316 316 315 315 316 316 316 316
315 315 315 316 316 316 315 315 316 315 315 315 316 316 315 316 315
316 316 316 315 315 315 316 315 315 315 316 315 315 315 316 316 315
315 315 315 315 315 316 316 316 316 316 316 316 316 316 315 2423 316
316 315 316 316 315 315 315 315 315 315 316 316 316 315 315 315 315
315 315 315 315 315 316 315 315 316 315 316 316 316 316 316 316 315
316 316 316 316 316 316 316 316 316 315 316 316 316 316 315 316 315
```

315 316 316 316 315 315 315 315 316 316 316 315 315 315 315 315 316
315 315 315 316 316 315 316 316 316 316 316 316 315 315 315 315 316
316 316 316 316 316 315 315 315 316 316 316

RTT samples: 1126 1127 1127 2334 1126 1126 1126 1126 1125 1126 1125
1125 1126 1126 1125 1125 1126 1126 2214 2653 1127 2214 1126 1127 1127
1127 1127 2215 1127 1127 1126 1126 1126 1126 1127 1127 1126 1125 1126
1126 1126 1126 1670 1125 1125 1126 1126 1125 1126 2213 1125 2344 1126
1126 1126 1126 1127 1128 1128 1126 1126 2214 1126 1126 1126 1125 1125
1125 1126 1126 1126 1125 1125 1126 2213 1125 1126 2214 1126 1126 1126
1126 1127 1127 1126 1126 1125 1126 1126 1125 2213 1126 1126 1125 2213
1125 2214 1126 2213 1125 1126 1126 1126 1125 1126 1127 1127 1127 1126
1126 1127 1127 1127 1126 4322 1126 1126 1670 1126 1126 1125 1125 1125
1126 2214 1126 1126 1126 1126 1125 2213 1125 1126 1125 1126 1126 1126
1126 1126 1125 1125 1126

00:13:CE:DA:08:7B 0 0 0 RTS_Received 0.000000 0.000000 0.000000 0.000000
0 0 45389

00:1E:13:C7:E8:21 0 0 0 RTS_Received 0.000000 0.000000 0.000000 0.000000
0 0 7

00:1F:3B:AA:A5:2B 0 0 0 RTS_Received 0.000000 0.000000 0.000000 0.000000
0 0 8

Appendix B

Hardware & OS used for surveys

This appendix shows data regarding devices used for surveys.

WIDS device

Table B.1 shows hardware and software specifications of the WIDS. To develop and test the application was used a *Toshiba Satellite A100*. The hardware is sufficient to gather all the data, however is not enough for real-time applications.

Station devices

Hardware and OS used by two station is reported in tables B.3 and B.2. OS used on dell device was both Windows and Linux.

Model: **Toshiba Satellite A100-508**	
Type	**Specs**
CPU	Intel Celeron M370, 1.5GHz
RAM	1GB DDR2 533MHz
HD	5400 RPM, SATA
Wireless Card	Atheros AR5005G
Front Side Bus	400 MHz 3.2GB/s
Operating System	Ubuntu Linux 8.04
Kernel version	2.6.24-19
Wireless Module	Madwifi-ng

Table B.1: WIDS Hardware and configuration

Model: **Dell Latitude D410**	
Type	**Specs**
CPU	Intel Pentium M 740, 1.73GHz
RAM	512MB DDR2 533MHz
Wireless Card	Intel Pro Wireless 2200BG
Operating System (1)	Windows XP SP2
Operating System (2)	Backtrack Linux 3
Kernel version	2.6.21.5
Wireless Module	ip2200

Table B.2: Dell station Hardware and configuration

Model: **HP IPAQ 214 Enterprise**	
Type	**Specs**
CPU	Marvell PXA310, 624 MHz
RAM	128 MB SDRAM
Wireless Card	Marvell wireless SDIO8686
Operating System	Microsoft Windows Mobile 6 Classic

Table B.3: Dell station Hardware and configuration

Appendix C

IEEE 802.11 summary

This appendix provides a brief description of IEEE 802.11 mechanisms and procedures that are directly related to this project. It is divided into two parts. The first provides basic knowledge while the second frame formats and interested fields values.

C.1 Basic operations and procedures

802.11 networks can operate in two modes: *Infrastructure* and *Ad-hoc*. The first one is the most common: one or more nodes (STAs) are connected to one or more Access Points (APs) that maybe connected with a backbone network. The second one is also known as *computer-to-computer* connection, and basically provides connection between two wireless devices.

The 802.11 standard employs Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA): in order to transmit each node first listen to the traffic for a predefined period of time to check whether or not the channel is idle, then it proceeds with transmission otherwise it waits for the channel to be idle. The *Coordination Function* (CF) determine how long a device has to listen to the idle channel before transmitting. The most common CF is the *Distributed Coordination Function*. It provides an algorithm to compute a random backoff time

that gives to all nodes the same transmission priority. The time is composed by a standard *Inter Frame* time plus a random time. IEEE 802.11 defines several *Inter Frame Spaces*, DCF uses only two:

1. **SIFS**: Short InterFrame Space;
2. **DIFS**: DCF InterFrame Space.

SIFS is the basic and shortest interframe space and is used prior to transmission of an ACK frame, CTS frame or DATA frames after CTS.

DIFS is used by stations operating under the DCF to transmit data frames and management frames. Figure C.1 shows a RTS-CTS handshake along with SIFS and DIFS.

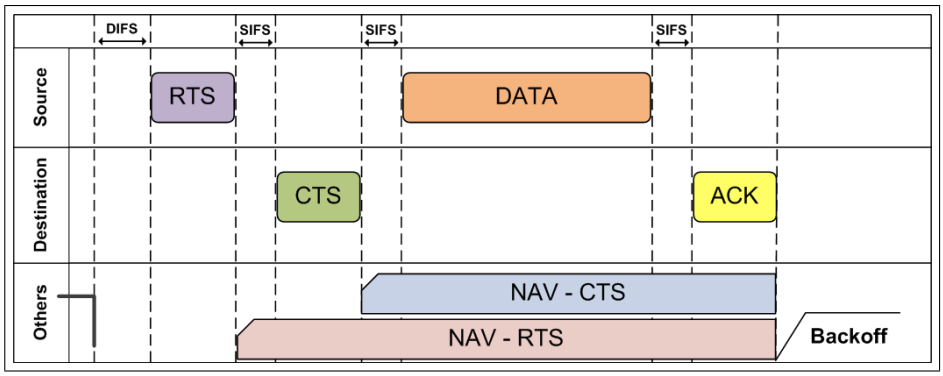

Figure C.1: RTS-CTS handshake with IFS

RTS-CTS handshake

This mechanisms is used by transmitting stations to ensure that data transmission is successful. It was introduced to solve the hidden-terminal problem. A STA wishing to transmit a data frame that exceeds a predetermined threshold (called *RTSThreshold*) shall use RTS-CTS handshake. The mechanisms follows this steps:

1. After DIFS a STA transmits an RTS frame wich contains TA, RA and duration of RTS-CTS handshake plus DATA transmission. With this frame it asks the receiver the permission to transmit data. All devices receiving the RTS set a time vector called *Network Allocation Vector*[1] according to duration specified in RTS.

2. After SIFS the STA corresponding to RA answer with a CTS frame wich contains the same duration minus RTS time. Again all devices receiving CTS set a corresponding NAV[2].

3. After SIFS transmitting STA start data transmission.

4. After SIFS receiving STA send an ACK frame.

If DATA dimension is lower than *RTSThreshold*, DATA frames are sent directly. If a collision occurs at some point, all the procedure needs to be carried out again after a random period. This period is computed multiplying a random integer from 0 to Congestion Window (CW) by the Slot Time. Each time that a collision occurs CW is increased sequentially by a power of 2 minus 1 until reaching CW_{MAX}.

Each device is identified by a MAC address, 6 bytes: first three ones identify the vendor last three the card.

C.2 Frame Types

Each frame start with a Frame Control field that contains informations about the frame like type, subtype, protocol and so on.

Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgt	More Data	Protected Frame	Order
Protocol Version	Control	Subtype	0	0	0	0	Pwr Mgt	0	0	0
Bits : 2	2	4	1	1	1	1	1	1	1	1

Figure C.2: General frame format

[1] NAV is used by STAs to know when and for how long the channel is busy.
[2] STAs that receive RTS may not be the same as CTS receving ones.

There exist 3 types of frames:

- Management frames: type 0x00;
- Control frames: type 0x01;
- Data frames: type 0x02.

Management frames

Management frames format is shown in figure C.3. Management frames handle primary operations like association, authentication, information distribution.

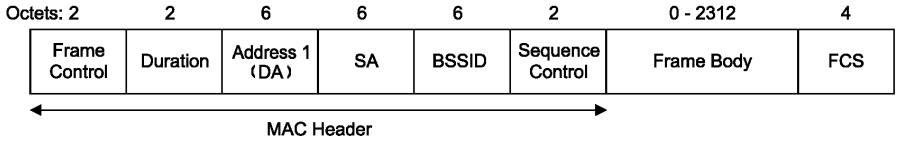

Figure C.3: Management Frame format

Fields of interest are Source Address, Destination Address and BSSID. First two ones has the same meaning as in ethernet protocol, BSSID is the identifier of the AP that manage the frame. The body is a variable length field that contains Information Elements like SSID, Capabilities etc.

Figure C.4 shows management frames subtypes.

Control frames

Control frames provides reliability functions for the MAC layer and administrative access to the wireless medium. Subtype considered by this project are RTS, CTS and ACK frames. Their format is shown in figures C.5, C.6 and C.7.

Figure C.8 shows control frames subtypes.

Subtype value b7b6b5b4	Subtype description
0000	Association request
0001	Association response
0010	Reassociation request
0011	Reassociation response
0100	Probe request
0101	Probe response
0110–0111	Reserved
1000	Beacon
1001	ATIM
1010	Disassociation
1011	Authentication
1100	Deauthentication
1101	Action
1110–1111	Reserved

Figure C.4: Management frames subtypes

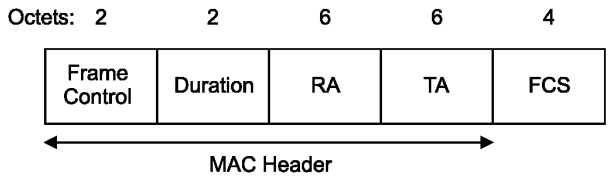

Figure C.5: RTS frame

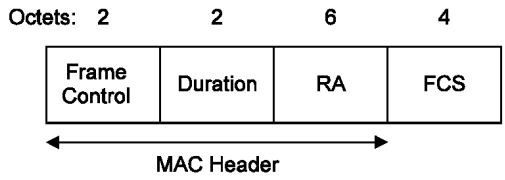

Figure C.6: CTS frame

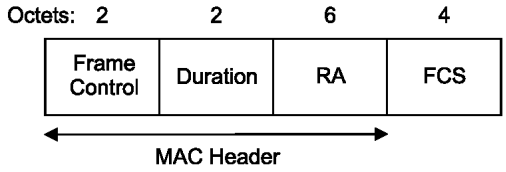

Figure C.7: ACK frame

Subtype value b7 b6 b5 b4	Subtype description
0000–0111	Reserved
1000	Block Ack Request (BlockAckReq)
1001	Block Ack (BlockAck)
1010	PS-Poll
1011	RTS
1100	CTS
1101	ACK
1110	CF-End
1111	CF-End + CF-Ack

Figure C.8: Control frames subtypes

Data frames

Data frames are used to transmit data over the wireless network. General format is shown in figure C.9. It contains 4 addresses and their meaning change according to the *Frame Control* field. Figure C.10 shows data address fields contents while figure C.11 shows data frames subtypes.

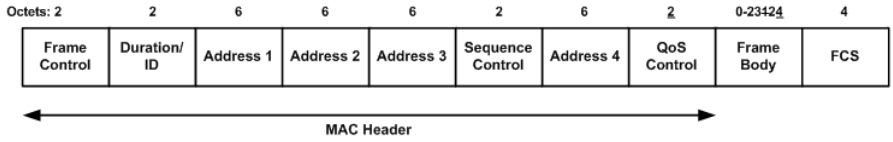

Figure C.9: General data frame format

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	RA = DA	TA = SA	BSSID	N/A
0	1	RA = DA	TA = BSSID	SA	N/A
1	0	RA = BSSID	TA = SA	DA	N/A
1	1	RA	TA	DA	SA

Figure C.10: Data address field contents

Subtype value b7 b6 b5 b4	Subtype description
0000	Data
0001	Data + CF-Ack
0010	Data + CF-Poll
0011	Data + CF-Ack + CF-Poll
0100	Null (no data)
0101	CF-Ack (no data)
0110	CF-Poll (no data)
0111	CF-Ack + CF-Poll (no data)
1000	QoS Data
1001	QoS Data + CF-Ack
1010	QoS Data + CF-Poll
1011	QoS Data + CF-Ack + CF-Poll
1100	QoS Null (no data)
1101	Reserved
1110	QoS CF-Poll (no data)
1111	QoS CF-Ack + CF-Poll (no data)

Figure C.11: Data frames subtypes

Bibliography

[1] Airsnarf website. http://airsnarf.shmoo.com/.

[2] Lib airware website. http://www.802.11mercenary.net/libairware/.

[3] Lib pcap & tcpdump website. http://www.tcpdump.org.

[4] Netstumbler website. http://www.netstumbler.com/.

[5] Radiotap header website. http://www.radiotap.org/.

[6] Snort-wireless website. http://snort-wireless.org.

[7] Wireshark. http://www.wireshark.org.

[8] IEEE 802.11-2007. 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. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1–1184, June 12 2007.

[9] M. Bernaschi, F. Ferreri, and L. Valcamonici. Access points vulnerabilities to DoS attacks in 802.11 networks. *Wirel. Netw.*, 14(2):159–169, 2008.

[10] R.A. Beyah, C.L. Corbett, and J.A. Copeland. The case for collaborative distributed wireless intrusion detection systems. *Granular Computing, 2006 IEEE International Conference on*, pages 782–787, 10-12 May 2006.

[11] Halil Ibrahim Bulbul, Ihsan Batmaz, and Mesut Ozel. Wireless network security: comparison of WEP (Wired Equivalent Privacy) mechanism, WPA

(Wi-Fi Protected Access) and RSN (Robust Security Network) security protocols. In *e-Forensics '08: Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[12] Luca Deri. *PF_RING User Guide, Linux High Speed Packet Capture.* ntop.org, version 1.1 edition, January 2008.

[13] Luca Deri, Netikos S. P. A, Via Del Brennero Km, and Loc La Figuretta. Improving passive packet capture:beyond device polling. In *In Proceedings of SANE 2004*, 2004.

[14] Jonathan P. Ellch. Fingerprinting 802.11 devices. Master's thesis, Naval Postgraduate School, 2006. Thesis Advisor: Dennis Volpano.

[15] Samer Fayssal, Salim Hariri, and Youssif Al-Nashif. Anomaly-based behavior analysis of wireless network security. *Mobile and Ubiquitous Systems: Networking & Services, 2007. MobiQuitous 2007. Fourth Annual International Conference on*, pages 1–8, 6-10 Aug. 2007.

[16] S. Frankel, B. Eydt, L. Owens, and K.Scarfone. *Establishing Wireless Robust Security Networks: A Guide to IEEE 802.11i.* National Institute of Standards and Technology, SP 800 − 97, February 2007.

[17] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *Proceedings of 15th USENIX Security Symposium*, pages 167–178, 2006.

[18] Rupinder Gill, Jason Smith, and Andrew Clark. Experiences in passively detecting session hijacking attacks in ieee 802.11 networks. In *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, pages 221–230, Darlinghurst, Australia, 2006. Australian Computer Society, Inc.

[19] Rupinder Gill, Jason Smith, and Andrew Clark. Specification-based intrusion detection in wlans. *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, pages 141–152, Dec. 2006.

[20] Rupinder Gill, Jason Smith, Mark Looi, and Andrew Clark. Passive techniques fo detecting session hijacking attacks in IEEE 802.11 wireless networks. In *Proceedings of AusCERT Asia Pacific Information Technology Security Conference AusCERT2005, Referred R&D Stream, Clark A., Kerr, K., and Mohay, G. (Eds), University of Queensland*, pages 26–38, 2005.

[21] Fanglu Guo and Tzi-Cker Chiueh. Sequence number-based mac address spoof detection. *Lecture Notes in Computer Science : Recent Advances in Intrusion Detection*, pages 309–329, 2006.

[22] Changhua HE and John C Mitchell. Analysis of the 802.11i 4-way handshake. In *Electrical Engineering and Computer Science Departments Standford University, CA 94305*, 2004.

[23] Changhua He and John C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *NDSS Symposium 2005*. Stanford University, 2005.

[24] Joshua Wright GCIH CCNA. Detecting Wireless LAN MAC Address Spoofing. http://www.willhackforsushi.com/papers/wlan-mac-spoof.pdf, 2003.

[25] M. Junaid, Dr. Muid Mufti, and M. Umar Ilyas. Vulnerabilities of IEEE 802.11i Wireless LAN CCMP Protocol. In *Prooceedings of World Academy of Science, Engineering and Technology*, volume 11, February 2006.

[26] Mike Kershaw and Joshua Wright. 802.11b firmware-level attacks. - http://www.willhackforsushi.com/papers/firmware_attack.pdf, 2006.

[27] K.Scarfone and P. Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. National Institute of Standards and Technology, SP $800-94$, February 2007.

[28] Arunesh Mishra, Nick L. Petroni Jr., William A. Arbaugh, and Timothy Fraser. Security issues in IEEE 802.11 wireless local area networks: a survey. *Wireless Communications and Mobile Computing*, 4(8):821–833, 2004.

[29] Floriano De Rango, Dionigi Cristian Lentini, and Salvatore Marano. Static and dynamic 4-way handshake solutions to avoid denial of service attack in Wi-Fi protected access and IEEE 802.11i. *EURASIP Journal on Wireless Communications and Networking*, pages 1–19, 2006. Article ID 47453.

[30] T.R. Schmoyer, Yu Xi Lim, and H.L. Owen. Wireless intrusion detection and response: a classic study using main-in-the-middle attack. *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, 2:883–888 Vol.2, 21-25 March 2004.

[31] Elankayer Sithirasenan and Vallipuram Muthukkumarasamy. An early warning system for 802.11i wireless networks. In *Auswireless 2006 Conference Papers*, 2006.

[32] Elankayer Sithirasenan, Vallipuram Muthukkumarasamy, and Danny Powell. IEEE 802.11i wlan security protocol - a software engineer's model. In *4th AusCERT Asia Pacific Information Technology Security Conference*, pages 39–50, 2005.

[33] A. Tsakountakis, G. Kambourakis, and S. Gritzalis. Towards effective wireless intrusion detection in ieee 802.11i. *Security, Privacy and Trust in Pervasive and Ubiquitous Computing, 2007. SECPerU 2007. Third International Workshop on*, pages 37–42, 19-19 July 2007.

[34] Alexandros Tsakountakis, Georgios Kambourakis, and Stefanos Gritzalis. On RSN-oriented wireless intrusion detection. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (2), On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II*, volume 4804 of *Lecture Notes in Computer Science*, pages 1601–1615. Springer, 2007.

[35] Joshua Wright web site. http://www.willhackforsushi.com.

[36] Raul Siles web site. http://www.raulsiles.com.

[37] J. Yeo, S. Banerjee, and A. Agrawala. Measuring traffic on the wireless medium: Experience and pitfalls, 2002.