MASTER THESIS

TITLE: Context Aware Self-Configuring WI-FI Network

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Carlos Ariel Quiel Rodriguez

DIRECTOR: Roc Messeguer Pallarès

DATE: July 4th 2011

Title: Context Aware Self-Configuring WI-FI Network
Author: Carlos Ariel Quiel Rodriguez
Director: Roc Messeguer Pallarès
Date: July, 4th 2011

Overview

Wireless networks are increasingly popular among end users primarily to provide Internet access services. For this reason it is becoming easier to find highly congested zones with multiple wireless networks. Many of these networks use default settings which causes, in some cases, that some of communication channels in the 2.4GHz band are overused.

This document presents the work done to design a context aware self-configuring system which is intended to improve the performance of independent WI-FI networks by avoiding the interference generated by neighboring wireless networks.

For the realization of this system were carried out many experiments which show that wireless networks working on the same channels suffer interference among them that reduces the capacity of each network. In addition to these experiments multiple metrics were proposed in an attempt to predict the capacity of a channel. The number of interfering radios and the number of rogue data packets were the two proposed metric that better fit the needs for capacity forecast.

From the obtained results a prototype that implements context aware self-configuring system was developed. The proposed algorithm uses a quality value which is calculated from the weighted sum of the two metrics explained before.

This prototype was tested against two algorithms (random and static). The results show that the proposed system improves the mean Throughput of the wireless network under realistic conditions.

Título: Context Aware Self-Configuring WI-FI Network
Autor: Carlos Ariel Quiel Rodriguez
Director: Roc Messeguer Pallarès
Fecha: 4 de Julio de 2011

Resumen

Las redes inalámbricas son cada vez más populares entre los usuarios finales principalmente para proveer de servicios de acceso a Internet. Por esta razón cada vez es más fácil encontrarse con zonas altamente congestionadas por múltiples redes inalámbricas. Muchas de estas redes utilizan configuraciones por defecto lo que provoca que, en algunos casos, algunos de los canales de comunicación de la banda de 2.4GHz estén sobre utilizados.

Este documento presenta el trabajo realizado para crear un sistema automático de adaptación en base al contexto para mejorar el rendimiento de las redes WI-FI independientes. El sistema tiene como objetivo principal maximizar el Throughput de la red reduciendo las interferencias generadas por las redes inalámbricas cercanas.

Para la realización de este sistema se llevaron a cabo múltiples experimentos los cuales demuestran que realmente redes que trabajan en los mismos canales sufren interferencias que reducen la capacidad de la red. Además de estos experimentos se han propuesto múltiples métricas que intentan predecir la capacidad de un canal. De estas métricas el número de radios interferentes y el número de paquetes de datos ajenos a la red son las dos métricas que mejor se ajustan para predecir la capacidad máxima de un canal.

Con los resultados de los experimentos se desarrolló un prototipo que implementa el sistema automático de adaptación en base al contexto. El algoritmo propuesto utiliza un valor de calidad que se calcula a partir de la suma ponderada de las dos mejores métricas obtenidas durante los experimentos.

Este prototipo se probó contra otros dos algoritmos (aleatorio y estático). De los resultados de esta prueba se ha demostrado que el sistema propuesto mejora el Throughput medio de la red inalámbrica en condiciones reales.

“I dedicate this thesis to my parents, siblings, friends, classmates and María who always encouraged me to achieve my goals and that without their support I could not have completed this work.”

Thank you all.

ÍNDEX

INDEX FIGURES

INDEX TABLES

INTRODUCTION

Since the emergence of the suite of wireless technologies defined under the standard IEEE 802.11(a, b, g and n) wireless networks have been very successful in providing solutions on many environments such as businesses, households, airports, government entities, among others. The global acceptance of these technologies can be ascribed to its ease of implementation, inexpensiveness of the equipment and the support among almost all the gadgets available on the market like laptops, cellphones, mp3 players, PDAs, tablets, etc.

The wireless networks based on the IEEE 802.11 standard are the most commonly used wireless solution around the world. These types of networks are commercially known as WI-FI that stands for wireless fidelity and is a trademark of the WI-FI Alliance. The term WI-FI was originally created as a simpler name for the "IEEE 802.11" standard.

The popularity of WI-FI networks has caused large deployments on many countries to provide different services. WI-FI networks are often used to provide Internet connection to a few numbers of clients within small coverage area or to offer other services through wireless mesh networking within a location. Some measurements done in [1] has shown that the number of interfering wireless networks on urban cities can increase very quickly and get to levels of up to 85 interfering radios for a single access point. In such environments the overall capacity of each wireless network is severely reduced.

The main problem is due to the fact that mainly all the WI-FI deployments are done in an uncoordinated way among neighbors and using defaults configurations. This generates that a lot of access points are working on the same communication channel. The main consequence is that large amount of wireless networks must share the scarce 2.4 GHz ISM band generating serious problems of interference and coexistence between the networks nearby.

According to [1] wireless network deployments have two important properties, they are: unplanned and unmanaged. Most of the access points are deployed by users in a spontaneous way with the consequence of variable AP densities. For most of the owners of an AP consider very complicated the management tasks and just rely on defaults configurations of the equipment.

In many cases the interference problems are overlooked because although there is degradation of the network users can use it for light and simple services like navigation and query websites. However taking into account that urban usage patterns of wireless networks are very high [2] and that nowadays real-time voice and video services are very frequent, the wireless degradation due to the interfering radios will be more appreciated by end users.

The chaotic deployment of WI-FI networks and the inexistence of a standard process which coordinates the configuration among independent wireless networks create an opportunity to develop framework through which a WI-FI network can be aware of its variable context and depending of the measured

context change its configuration, in other words, that the access point can detect high interference from nearby wireless networks and adapt itself changing its configuration to minimize the effects of interferences for its clients. This process can increase the satisfaction of nowadays clients which requires high throughput rates and good wireless quality.

The main objective of this master thesis is to develop a prototype which can provide of the previous capabilities of adaptation to the context and self-configuration to an access point. For the development of this tool is necessary to evaluate different monitoring methods to infer the quality of the wireless network and which provide us with sufficient data for decision-making in the network configuration. The decision is focus on the dynamic selection of transmission channels for the wireless access point.

The rest of the document is organized as follows: CHAPTER 1 provide a brief summary of actual state of research and solutions within this field, the CHAPTER 2 give an explanation of the different technologies that were used to develop the thesis, the CHAPTER 3 presents the experiments and results that were carried out to measure the relevance of certain metrics, CHAPTER 4 explains in detail the prototype implementation and a comparison between our model and others, and finally CHAPTER 5 presents global conclusions about the overall work.

CHAPTER 1. RELATED WORK

Nowadays a standard method to reduce the interference among congested zones of WI-FI networks does not exist. Some manufacturers and research groups have designed implementations that mitigate the problem of interference between WI-FI networks.

Large implementations of context self-configurations are done in infrastructures wireless networks under the same management company. These implementations use centralized schemes obtaining information from the clients and changing the wireless configuration depending on the feedback. Large manufacturers have developed proprietary solutions to solve this problem.

This section will provide a global perspective of some related work done in the area of context aware for 802.11 based wireless networks. This section divides the related work in two areas: the research solutions and the proprietary solutions.

1.1 Research solutions

Research solutions are adaptations schemes proposed by research groups. Most of these works presents solutions based on algorithms that through some measures determine the best operative channel to maximize the overall performance of the network.

The work on [3] describe two distributed algorithm, one of those allow that multiple interfering 802.11 access points (AP) to select their operative frequency in order to minimize the interference among them. The second algorithm is used on the clients and evaluates the best available access point to establish a connection in order to get fair share of the whole network bandwidth. Both algorithms are based on a simulated technique called Gibbs sampler.

In [4] the authors propose a decentralized mechanism to select an operative channel. This mechanism was denominated Hminmax and is based on the construction of an interference graph among wireless networks. This graph is constructed with the information that the clients reports to the AP. The edges of the graph represent the percentage of clients that receive certain level of interference from rogue wireless network. An example of this kind of graph is observed on Fig. 1.1. With this information the algorithm tries to minimize the maximum numbers of clients that suffer interference from the same BSS. An important fact in this approach is that the decision is made based on the perspective of the clients attached to the network.

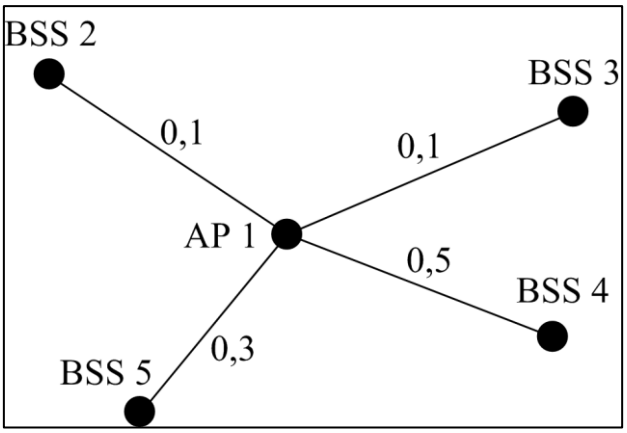

Fig. 1.1. Example of a localized interference graph from Hminmax mechanism

In [5] the authors propose a solution called MAXchop that improve the fairness in channel allocation between independent basic services set (BSS). This mechanism builds a localized interference graph in order to define channel hopping sequence for the BSS that make the fairness allocation sharing of available channels. The most relevant contribution of this work is the distribution of BSS among the 2.4GHz band in a fair way.

In [6] da Silva et al. design a mechanism to select the best available cannel based on two metric: the occupation level and the noise levels. The occupation level is the percentage of the time that the channel is considered busy by the carrier detection of the measuring node due to traffic from others BSS. Both metrics are measured on the clients and are reported to the access point through 802.11k messages. 802.11k is under standardization by a working group of the IEEE. With the information gathered from all the wireless clients the AP creates two vectors one with the mean occupation level of each channel and the other one with the noise levels of each channel. With these two vectors the AP selects n less occupied channels and among them selects the one with lowest noise as the operative channel. This algorithm is performed each T time if the occupation level on the working channel surpasses a tolerance threshold. This threshold is an adjustable parameter that defines the aggressiveness of the algorithm.

Others works were done on different mechanisms to select best operative channels on wireless mesh deployment in order to reduce the interference among AP of the same wireless mesh [7].

The main problem with almost all the previous explained schemes is that the results are based on simulations and not in real environments.

1.2 Proprietary solutions

The proprietary solutions describe tools that companies have developed in order to provide Self-Configuring capabilities to the networks based on its technologies. This section presents the solutions of three companies: Meraki [8], Meru Networks [9] and Aruba Networks [10]. The three companies provide wireless hardware solutions with access points. They also have different

software's which are capable of monitoring and manage the wireless equipment of the whole network.

Meraki develop an automatic system called Auto RF by which every access point on the network continuously and automatically monitors its surroundings for any source of interference that could affect Wi-Fi performance. The Meraki access points are equipped with a spectrum analyzer that collects information about metric used to evaluate the interference. The spectrum analyzer is capable of measure interference from non-802.11 sources like Bluetooth devices, microwave ovens, cordless phones, etc. Meraki access point monitors 802.11 channel utilization, on both, 2.4 and 5GHz, detecting interference from neighboring APs or rogue APs.

On Meraki deployments the auto RF tool can be interconnected with a Cloud Controller which is another Meraki tool used to manage the entire network from a centralized place. The cloud controller obtains information from all the access points in the network. The Cloud Controller continually assesses the health of the entire network, dynamically tuning the wireless channel, transmit power, and client connection settings to automatically adapt to changing interference conditions

The utilization of these tools increase the reliability under challenging RF conditions gives better coverage with fewer access points and allow higher client throughput.

On Wireless mesh deployments Meraki has designed a proprietary routing protocol to provide a scalable routing algorithm to mesh networks. The routing protocol is integrated with a dynamic RF planning technology owned by Meraki. This tool takes advantage of multiple wireless channels and radios whenever possible. In networks with multi-radio devices, the routing protocol will attempt to find full-duplex wireless links wherever possible to eliminate per-hop routing penalties found in single-radio networks. Furthermore, each wireless repeater will evaluate the available wireless channels to maximize overall system capacity and client throughput.

Another company that has developed some tool to monitor the interference that affect the working wireless network is Meru Networks. This company has an appliance with the proprietary software RF Network Manager. This software provides an intelligent and comprehensive management system for Meru 802.11 networks. The Network Manager provides centralized management through RF visualization and wireless performance dashboards. In the same way as Meraki does each Meru access point collects monitor its environment and report this information to the Network Manager. This software does not provides the capabilities of auto configuration but alerts the administrators of high interference level which can compromise the overall capacity of the WLAN.

Other important company who has developed very interesting solution in the context-aware area is Aruba Networks. This company has a proprietary spectrum analyzer detects 802.11 and non-802.11 sources of RF interference. Aruba networks AP scan the spectral composition of 2.4-GHz and 5-GHz radio bands and identifies RF interference, classifies its sources and provides real-time analysis. This tool can classify interference in up to 13 categories including

microwaves, Bluetooth devices, cordless phones, cordless base stations and fixed-frequency video and audio devices.

The data collected by the Aruba spectrum analyzer is used to isolate packet transmission problems, over-the-air quality of service (QoS) and traffic congestion caused by RF contention with other devices operating in the same band or channel. Appropriate remediation measures can be executed to optimize network performance.

The Aruba spectrum analyzer can be used in conjunction with other Aruba solution called Adaptive Radio Management (ARM). ARM employs infrastructure-based controls to optimize Wi-Fi client behavior and automatically ensures that APs stay clear of interference. The ARM uses the information gathered by the Aruba spectrum Analyzer for channel optimization.

CHAPTER 2. TECHNOLOGIES

This chapter describes the different technologies in which the project is based. First of all it describes the wireless network technologies, then several tools used during the project.

The objective of this chapter is to give a brief description of the context where the project was developed.

The section 2.1 explains the main characteristics of the wireless communication network which were used during the different experiments explained on CHAPTER 3. Section 2.2 describes the sniffer software Kismet. This software is used to motorize the environment and generates useful information used in the proposed algorithm to select the best operative channel. Section 2.3 explain the measurement tool Iperf which is used to generates traffic and measure important quality values like throughput, delay and packet loss. The section 2.4 gives a global view of the MadWifi driver which is used in this project to simulate an access point in an Ubuntu laptop in which is implemented the context aware self-configurable system.

2.1 Wireless Networks

Wireless networks are data transmission system designed to provide connectivity between equipment by using radio waves to transport the information.

The wireless networks are very different from the wired networks at the physical level. The main differences between both medium can be summarized in the following way:

- Wireless networks use a medium that has no observable boundaries and in which a station with a physical compliant transceiver would be able to detect network frames.
- The network is unprotected from other signals that share the medium.
- The wireless medium is less reliable that common wired medium.
- Wireless network can have dynamic topologies because of the mobility capabilities of stations.
- The medium have time varying and asymmetric propagation properties
- The wireless network may experience interference from other wireless networks working on overlapping areas.

There exist several types of wireless networks but those based on IEEE 802.11 standards are nowadays the most popular technologies used to provide broadband radio access to IP networks. This kind of network is normally called WI-FI networks.

WI-FI networks are usually implemented as the final link between the existing wired network and a group of client computers, giving these users wireless access to the full resources and services of the corporate network across a building or campus setting. Also this kind of network is very popular in home deployments to provide wireless access to broadband internet connection.

WI-FI networks are based on the 802.11 specification [11] which is a standard for wireless LANs and was ratified by the Institute of Electrical and Electronics Engineers (IEEE) in the year 1997. The 802.11 standard provides different sub protocols that describe specific functionalities and capabilities. The most popular implementations of 802.11 are: 802.11 b, 802.11g, 802.11a and recently 802.11n.

802.11b operates in the ISM band 2.4 GHz and provides modulation methods that accomplish rates up to 11 Mbps. 802.11b use DSSS (Direct-sequence spread spectrum) and CCK (Complementary Code Keying) as modulation techniques.

802.11g also works at the 2.4 GHz band and provides data rates up to 54 Mbps using OFDM (Orthogonal Frequency Division Multiplexing) as modulation scheme for high rates.

802.11a uses the same data link layer protocol and frame format as the original standard, but uses OFDM as modulation technique. It operates in the 5 GHz band with a maximum data rate of 54 Mbps. The utilization of the 5 GHz band gives 802.11a a significant advantage in terms of interference because this band is less used than the 2.4 GHz band. However, the utilization of this frequency also leads to certain disadvantages in coverage because 802.11a signals are absorbed more easily by walls and by other solid objects than signals from 802.11b/g.

802.11n is the most recent amendment to the 802.11 standard. It was approved on September 2009 and provides several improvements to its predecessors in order to provide theoretical data rates up to 600 Mbps. The mayor improvements in 802.11n are MIMO (multiple inputs, multiple outputs) technology, improved OFDM, 40 MHz of channel bandwidth and a MAC layer overhead reduction.

For the elaboration of this project were used 802.11b/g compliant wireless network adapters.

2.1.1 Architecture of 802.11 networks

The architecture of the 802.11 networks is very flexible and can be deployed on three types of topologies. These topologies are: independent basic service set (IBSS), basic service set (BSS) and extended service set (ESS) [12].

A service set is a group of equipment that starts to communicate between them. An 802.11 node emits the information that wants to transmit in an RF (Radio Frequency) carrier. Any node can be within the coverage of different transmitters so this node is able to receive the transmitted information. In order

to distinguish between the transitions of different service sets the information is tagged with a service set identifier (SSID) that is used by the receiving nodes to filter the signals that want to process. The SSID is the name of the wireless network which is listed by any wireless card before selecting desired wireless network.

An IBSS is a group of 802.11 nodes which have established effective communication between each other and also stands alone with no other infrastructure equipment attached to it. The Fig. 2.1 shows a diagram that represents an IBSS.

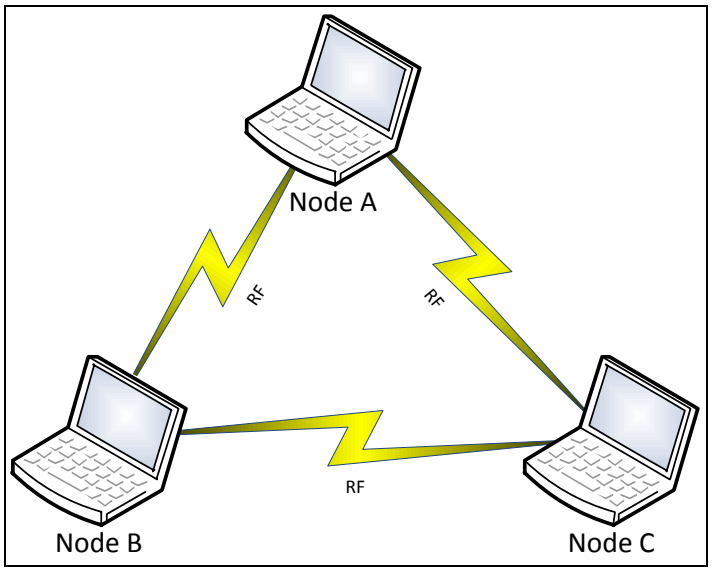

Fig. 2.1. IBSS

An IBSS is also referred to as an ad-hoc network because it is essentially a simple peer to peer WLAN. An ad-hoc network is a decentralized type of wireless network in which the nodes that are part of the network does not rely on any infrastructure device like Access Points. These kinds of networks are usually small and only last long enough for the communication of the information that need to be shared among devices.

A BSS is a group of devices that are able to communicate among each other with the utilization of a specialized station known as an Access Point (AP). The AP is the central point of communication for the whole network. Nodes cannot transmit frames directly to other nodes. In spite of this transmitter node must communicate with the AP which will forward the frames to the destination stations. A BSS can have an uplink connection with the wired infrastructure to provide the network of certain services. The Fig. 2.2 shows a diagram of a BSS.

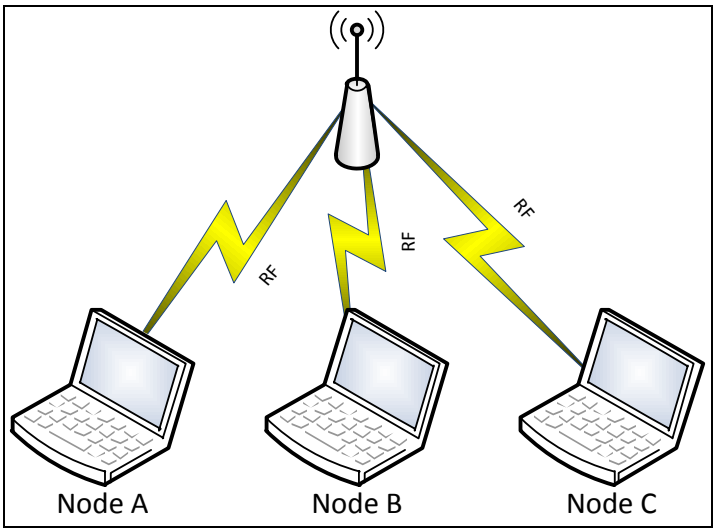

Fig. 2.2. BSS

When two or more BSS are interconnected by their uplink interfaces forms an ESS. The interconnection between BSS is made via a distribution system (DS). A DS increases network coverage because groups BSS within a common administration. Each BSS becomes a component of an extended, larger network. Entry to the DS is accomplished with the use of Access Points (AP). The Fig. 2.3 shows a diagram of an ESS.

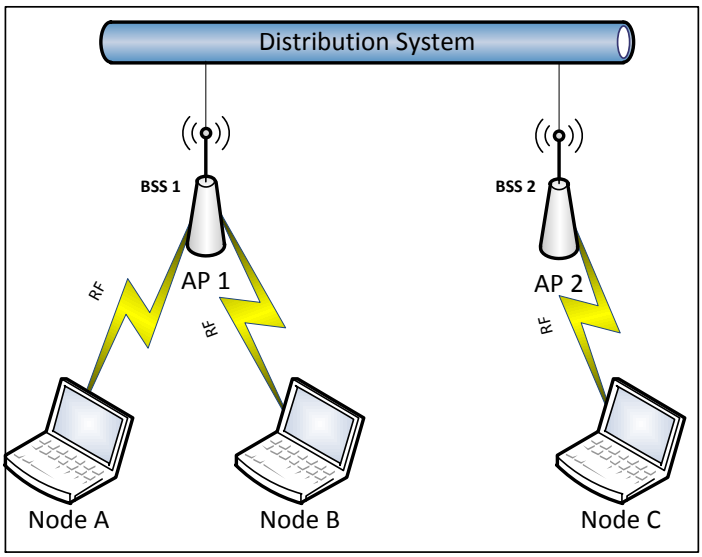

Fig. 2.3. ESS

The most relevant characteristic of an ESS is that the entire network looks like an independent basic service set to the Logical Link Control layer (LLC). This means that stations within the ESS can communicate or even move between BSS's transparently to the LLC.

Normal deployments of wireless network are made in the shape of BSS and ESS.

The Table 2.1present a summary of the 802.11 standard with the different protocols defined on it and the main characteristics of each one.

Table 2.1. 802.11 protocols summary

Protocol	Release	Frequency (GHz)	Channel Bandwidth (MHz)	Data rate per stream (Mbit/s)	MIMO streams	Modulation
802.11 a	Sep 1999	5	20	6, 9, 12, 18, 24, 36, 48, 54	1	OFDM
802.11 b	Sep 1999	2.4	20	5.5, 11	1	DSSS
802.11 g	Jun 2003	2.4	20	6, 9, 12, 18, 24, 36, 48, 54	1	OFDM, DSSS
802.11 n	Oct 2009	2.4/5	20	7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2[	4	OFDM
			40	15, 30, 45, 60, 90, 120, 135, 150		

2.1.2 Wireless operational modes

The wireless network interface cards compliant with 802.11 standards can work in different modes. The mode in which a wireless network interface works defines its utilization on the network architecture and the capabilities that can provide.

The operational modes are:

- Managed: This is the normal or default operation mode for a wireless network card. In this mode the network card behaves as a WLAN client, in such a way that it needs to associate with an access point in order to send and receive traffic. The Manage operation mode is also known as station mode (sta) or infrastructure.

- Master: In this mode the wireless network card acts as an access point providing connectivity services to many clients. As an access point the device becomes the central node of the network and is in charge of forwarding traffic between members attached to the SSID and also has to provide an uplink connection with a wired infrastructure.

- Ad-hoc: In this mode the network card can join or create an ad-hoc network. This mode allows a device to communicate directly with the nodes that are part of the ad-hoc network.

- Monitor: In monitor mode the wireless interface is used to capture the traffic travelling through the RF domain for different purposes. This mode

is used to implement sniffers, intrusion detection system or wireless network detectors. In this mode all the packets that the wireless interface is able to hear is passed to the application level to be analyzed. Much software uses this mode to obtain traffic statistics of the networks or to detect rogue connection to a wireless LAN. Normally this mode is configured to obtain just headers of the packets in order to respect the privacy of the clients that are using the wireless infrastructure.

- WDS: In WDS mode a device can be used to create large wireless networks by linking several Access Points together. In WDS mode the device is placed above the access point hierarchy and by this is able to manage the whole wireless network as one.

2.2 Kismet software

Kismet is an 802.11 wireless network detector, sniffer, and intrusion detection system [13]. Kismet works with any wireless card which supports monitoring mode, and can sniff 802.11b, 802.11a, 802.11g, and 802.11n traffic.

Kismet identifies WLANs by passively collecting packets and detecting networks, which allows it to detect hidden networks and the presence of non-beaconing networks via data traffic. The software creates a list of the networks detected and for each of its display several measures like: working channel, packet received, signal strength, noise level and client activity.

This software can be deployed on a client/server architecture basis in which some devices acts as clients (drones) capturing the traffic on the air and sending this information to a centralized equipment. This centralized device collects the information from several drones and presents a whole picture of the wireless environment.

Kismet has a text-based graphical interface that helps the user to display different windows that shows diverse type of information like channels, networks, clients, etc.

This software returns 3 types of output files that contain the collected information from the wireless environment. The outputs files are in XML format, text format and PCAP format. This last output file type can be opened with the software Whireshark[1]. The Fig. 2.4 shows the main screen of the Kismet text-based graphical interface

[1] Wireshark is a network protocol analyzer for UNIX and Windows. http://www.wireshark.org/

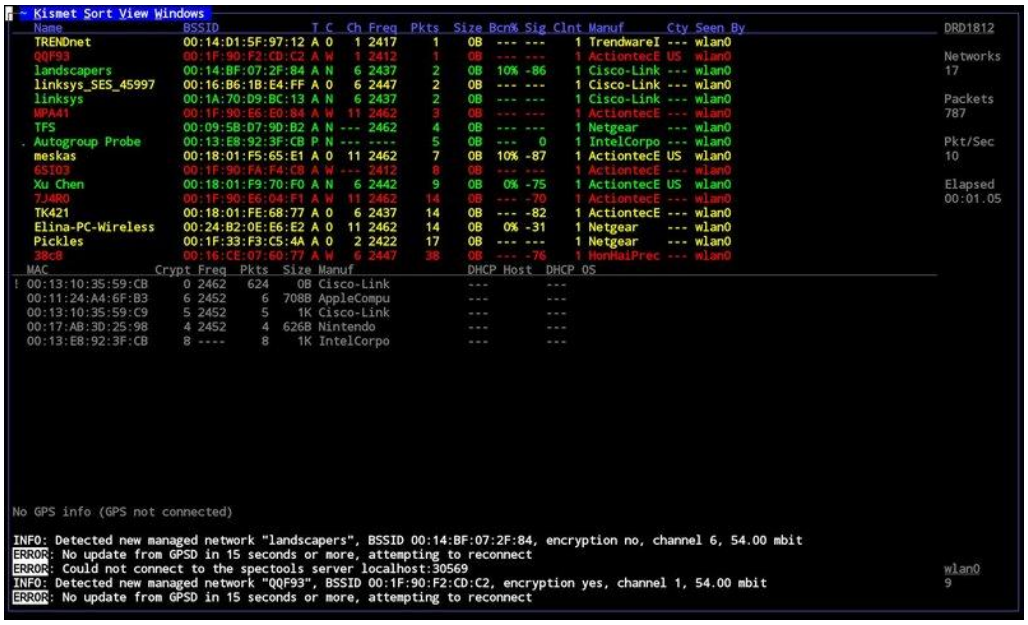

Fig. 2.4. Kismet main view

The Kismet software is used on this project as the source of the metrics used by the algorithm to calculate the best available channel to work. The software is implemented on the simulated AP that is intended to self-configure. From the output files for project purposes the XML file is used.

2.3 Iperf tool

Iperf is a network tool by which can be possible the measure of the throughput that a network is capable to transport [14]. Iperf is written on the programing language C++.

Iperf can generate TCP or UDP traffic streams to measure the throughput in a network. Also Iperf is capable to measure other network performance parameters like: packet loss and delay. The tool allows the user to change some parameters to test a network. Some of the modifiable parameters are: datagram size, testing time, buffer size, transmit bandwidth, report interval, etc.

Iperf is a client/server application, this means that, in order to work one device has to acts as the server and receive the data streams and other device is the client that sends the data streams.

Iperf has a text-based graphical interface and works by commands but a java GUI exist for uses Iperf in a more friendly way. This application is called Jperf and need a previously successful Iperf installation.

In this project the Iperf is used to measure the throughput, delay and packet loss in the metrics evaluation experiments. Also is used Jperf in order to have a more friendly way to uses the Iperf software.

2.4 MadWifi tool

MadWifi stand for Multiband Atheros Driver for Wireless Fidelity and is a Linux kernel device driver for Atheros-based Wireless LAN card [15]. MadWifi is one of the most advanced drivers for WLAN devices on for Linux today. This driver allows the user to configure in a very flexible way how the wireless card has to behave.

The MadWifi driver is very often used on the scientific research field to make wireless networks test or new implementation. This driver is open source and supports almost all the available Atheros chipsets.

The two main functionalities that this driver can accomplish are:

- Operational modes: The MadWifi driver allows an administrator to configure the wireless card in different modes. These modes make the network car to works as different wireless elements. The modes supported by the MadWifi driver are: sta, ap, adhoc, ahdemo, monitor and wds.

- Multi-BSSID: The MadWifi driver also allows the configuration of virtual interfaces that works above only one physical card. This feature is called VAP that stand for virtual AP. VAP sit on top of a base device usually called wifi0. The virtual interfaces are named usually “ath0” but the name is configurable through a command. The VAP lets create on a single device several interfaces with different operational modes. In order to manipulate VAPs, MadWifi comes with a command tool called “wlanconfig” that can be used to create and destroy VAPs.

Actually MadWifi driver is being replaced by new versions of it called ath5k and ath9k. Even if these new drivers support almost all Atheros chipset some features are still not working properly. But they are the future of the MadWifi project.

Within this project the MadWifi driver is used in the Atheros based wireless card used as the sniffer. The ath5k driver was first used but then was replaced by the MadWifi because the ath5k present some problems on reporting noise level to the Kismet software.

A MadWifi installation tutorial can be found in section 7.1.

CHAPTER 3. EXPERIMENTS

This chapter describes the different experiments that were done in order to evaluate different metrics that the Kismet software is able to capture from the air and are intended to describe the wireless network performance.

The experiments were done in two stages. The first stage is composed by three experiment to evaluate the metrics proposed that can describe the state of the wireless link. The second stage is an experiment to evaluate the correlation levels between the metrics and the expected capacity on each wireless channel.

The objectives of these experiments are to obtain information about how much the metrics affect the overall performance of a network. The results of the experiment will allow us to design an algorithm that will avoid the degradation of the network caused by the increase of a metric or a group of metrics.

Is important to mention that the measure in which these metrics benefits or worsens the wireless network is evaluated in terms of throughput, delay and loss packets. This data is provided by the Iperf software.

During the experiment are identified 2 types of wireless networks. The first one is the “working wireless network” and is nothing more than the network that tries to adapt to the context in which it is. The other one is the “rogue wireless network” that is a network or a group of networks that are not under the administration of the working network and therefore generate interference.

3.1 Metrics

Kismet software package allows us to monitor several different metrics that are obtained from the sniffer card, but not all the metrics are important to model the RF state. That is why only the most meaningful metrics were chosen to study and these metrics are:

- Interfering radios: This metric model the number of wireless networks that during the test were sharing the same communication channel of the 2.4GHz band. Some types of wireless networks do not report its working channel. In order to assign one channel to these kinds of networks some calculations are needed. Basically the Kismet software provides information about in which center frequency the packets are received. By using this information is calculated the center frequency in which the majority of the packets of certain network were received and this center frequency is used to assign the operational channel.

- Rogue Packets: this metric model the total amount of packets captured from other networks. In this metric does is not important the type of packets which can be, using the categories provided by Kismet, LLC (Link Layer Control), data or crypt (cryptography).

- Rogue Data Packets: This metric model the amount of data packets captured from rogue wireless networks.
- Rogue Bytes: This metric model the total amount of bytes transferred on the medium by rogue wireless networks.
- Best SNR: This metric shows the best signal to noise relation of the wireless network detected. It is obtained from the values of maximum signal received and minimal noise detected that the Kismet software provides.
- Maximum noise level: This metric model the maximum noise detected on certain channel, and is obtained from the noise levels reported by the wireless networks working on the channels. From the reported noise level the worst reported is going to be used as the maximum noise level.

3.2 Equipment

The Table 3.1 shows the specifications of the two types of equipment used to develop all the experiments. Nodes 1 and one of the nodes type 2 have two wireless cards installed. Node 1 uses both wireless cards to establish separated connections to two different BSS during the experiments. Node 2 instead uses one of the wireless cards to provided ad-hoc connectivity and the other one is used in monitor mode to act as a sniffer (PCMCIA card). This card is the one using the MadWifi driver. The rest of type 2 nodes are equipped with one wireless card that is configured as managed mode or ad-hoc depending of the test.

Table 3.1. Equipment

Node Type	Quantity	Brand	Model	Operating System	Wireless card	Specs
1	1	Dell	Inspiron 1545	Ubuntu 10.10	Eth1:Broadcom BCM4312 driver: wl0	Ram: 4Gb CPU: Intel Core 2 Duo T6400 2GHz
2	4	HP	Compaq nx6110	Ubuntu 10.10	Eth1: PRO/Wireless 2200GB driver: IPW2200	RAM: 500Mb CPU: Intel Pentium M 1.6GHz

The Table 3.2 shows detail information about the additional wireless cards that were needed to perform the several tests. Is important to note that although that both cards are Atheros based, the MadWifi driver can be used only with the PCMCIA, because the driver does not support USB interfaces

Table 3.2. Additional wireless cards

Interface	Manufacturer	Model	Chipset	Driver
USB	OvisLink	EVO-W301USB	Atheros AR9271	USB
PCMCIA	Proxim	ORiNOCO 11a/b/g Combo Card	Atheros AR5001x+	MadWifi (ath_pci)

3.3 Channel capacity

This experiment has two main objectives:

- Measure the wireless network performance without any interference radio. These measures will be used as a central point of comparison with other tests.
- Measure how the working wireless network is affected with just one interfering radio that introduces different traffic levels.

In order to model a free channel the experiment was carried out around the campus Baix Llobregat in a place where there is not sign of any wireless network. The Fig. 3.1 shows the place where the experiment was done.

Fig. 3.1. Location of the experiments

The experiment is divided on two scenarios. The first one can be appreciated on the Fig. 3.2.

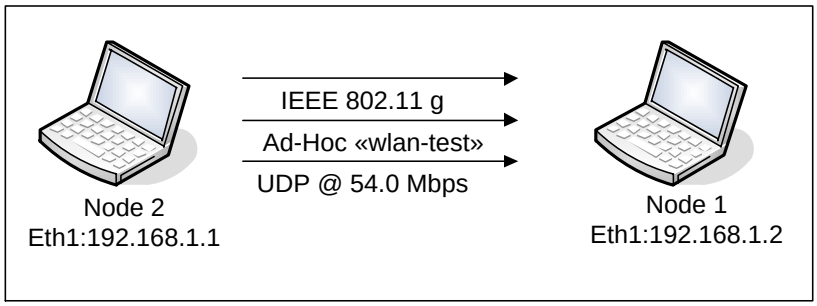

Fig. 3.2. Channel capacity first scenario

As can be noticed, the first scenario just has the working wireless network transmitting UDP packets at 54Mbps. The idea is to flood the channel with packets and measure the total capacity of the wireless communication channel.

The packets generation and reception was done with the Iperf software running the following commands:

- At node 1 the following command was executed "*iperf -s -u -P 0 -i 1 -p 5001 -w 1470.0K -f m*". This command place the node 1 in server mode listen UDP packets on port 5001. Also the buffer size is increased to 1470Kbytes.

- At node 2 the following command was executed "*iperf -c 192.168.1.2 -u -P 1 -i 1 -p 5001 -w 1470.0K -f m -b 54.0M -t 120 -T 1*". This command place the node 2 as the Iperf client generating UDP traffic to the destination IP address 192.168.1.2 and destination port 5001. Also this command specifies that the UDP buffer size is 1470Kbytes and the data rate to generate the packets is 54Mbps. With the parameters –i and –t the command specifies the report interval (1s) and the time of the test (120s).

A diagram of the second scenario is shown in the Fig. 3.3.

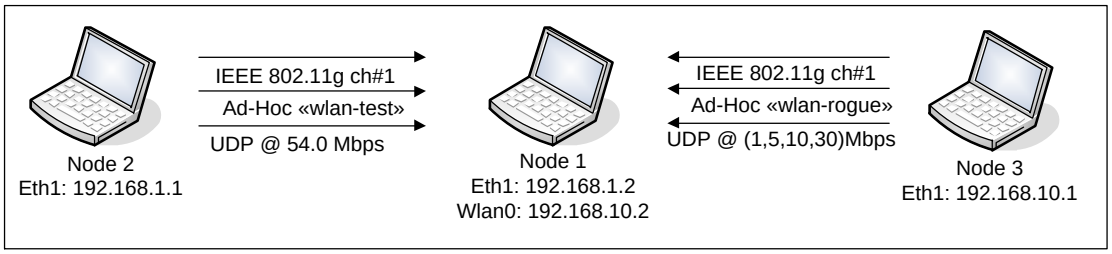

Fig. 3.3. Channel capacity second scenario

In this scenario a second wireless network is introduced on the same communication channel. This network is going to affect the performance of the main WLAN. In order to evaluate how the second network affects the performance of the working wireless network different levels of traffic are introduced by the rogue WLAN. With this scenario 4 test were done in which the rogue wireless network transmit data at 1,5,10 and 30Mbps.

The Node 1 has 2 wireless cards each one connected to different wireless LAN. The card Eth1 is connected to the network "wlan-test" which is the SSID of the

working WLAN. The card wlan0 is connected to the network “wlan-rogue” which is the SSID of the rogue wireless network.

The traffic generation and reception was also done with the Iperf software running the command explained before. The unique change is on the client side is in the parameter –b, that refers to the data rate to transmit the packets, that must be modified for each case.

On both scenarios the Kismet software was capturing information about the wireless environment. This software is running always in the Node 2 with the wireless card Ath1 on monitor mode.

3.3.1 Channel capacity results

The results of the experiment are going to be presented by scenario and a comparison of the metrics explained in 3.1 that were collected by the Kismet software during the experiments.

3.3.1.1 First Scenario (Free Channel)

The mean throughput achieved by the wireless network in this experiment is about 22.87±0.03Mbps (90% of confidence interval) with a standard deviation of 0.17Mbps. The mean delay obtained is of 0.5±0.08ms with a standard deviation of 0.5ms.

The values presented before are considered the best for the wireless communication channel for comparison purposes in this type of scenario. The wireless capacity may vary depending heavily in the wireless hardware and operational mode.

The delay histogram of the test is exposed on the Fig. 3.4. The delay values vary during the whole experiment but the mayor parts of the values are below 1ms of delay that can be considered as a good delay in the communication.

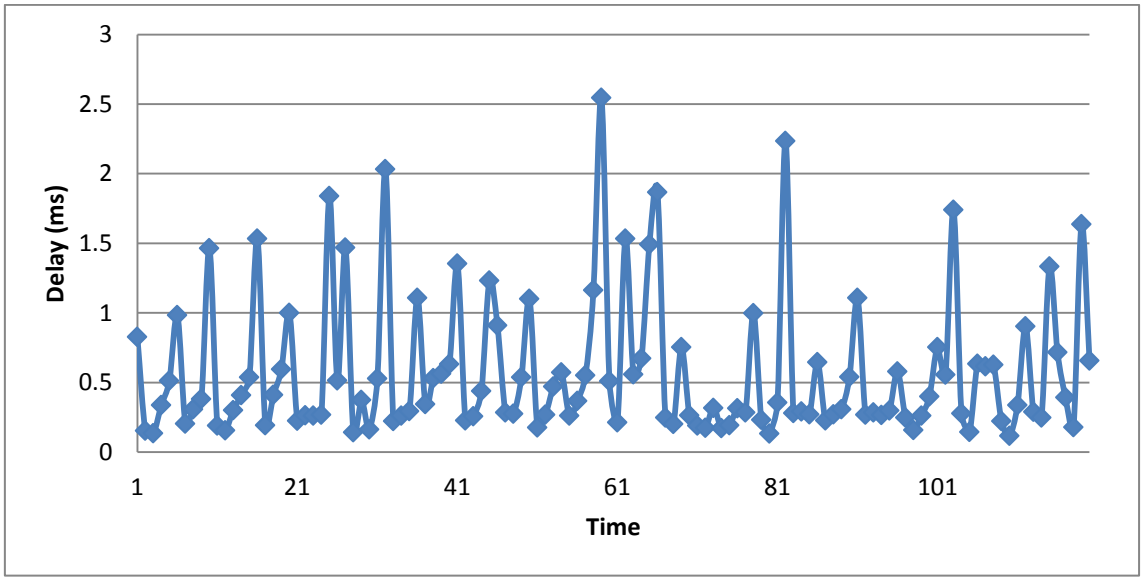

Fig. 3.4. Delay free channel

3.3.1.2 Second scenario (Variable rate interfering radios)

During the various tests of the experiment it was noted that the introduction of a new wireless network running on the same channel results in a decrease in the working network performance. Mainly the reduction in network performance occurs because the electromagnetic spectrum is a medium that must be shared by all wireless networks in a fair manner. In fact, the CSMA-CA[2] of the 802.11 networks is responsible for providing a mechanism by which different networks can access the medium fairly and minimizing the probabilities of packet collisions in the air.

The performance reduction in terms of throughout is shown on the Fig. 3.5. This graph compares free channel results and tests with the interfering radio transmitting at 1, 5, 10 and 30Mbps. As can be seen as interfering network increases its data rate requires more resources from the channel by which the working wireless network has to reduce its rate for both networks to coexist on the same medium.

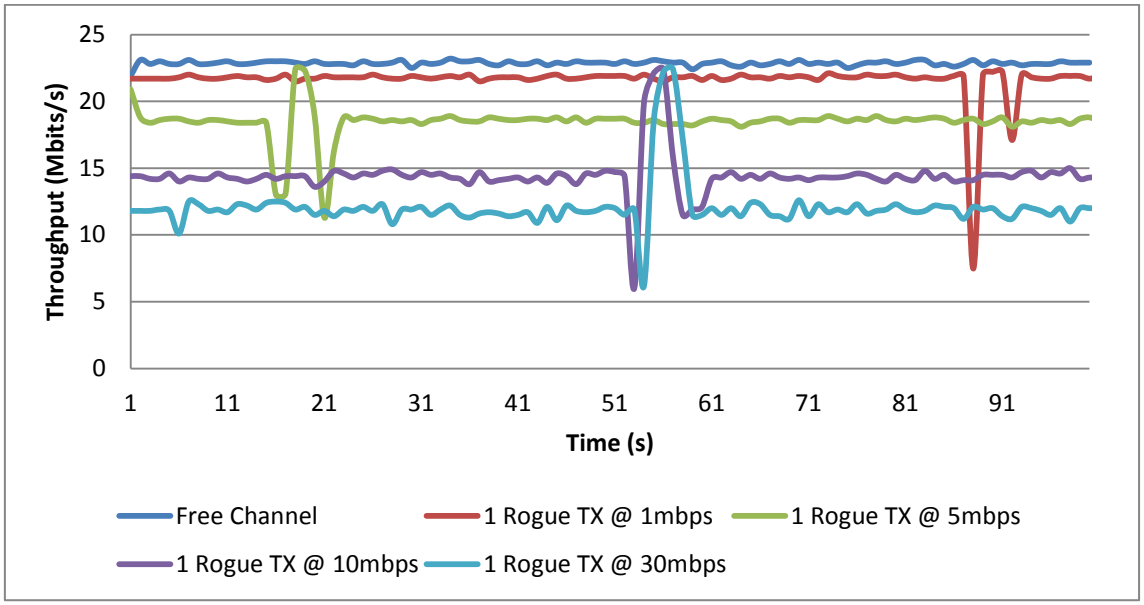

Fig. 3.5. Throughput comparison with interfering radios

The previous graph shows that the interfering radio affect the main wireless network, but is also important to evaluate if the existence of a working wireless network that try to obtain the maximum resources from the communication channel affect the other networks. The Fig. 3.6 shows the throughput histogram of the rogue wireless network on each test. An important fact that must be highlighted is that up to 10mbps the rogue network was not affected by the operation of our working wireless network. However when both wireless networks try to transmit at maximum speed the channel capacity is more or less divided by the half. From these results we can say that if the total traffic that a wireless network need to introduce in a wireless communication channel will not be affected as long as this traffic does not exceed the portion of the channel

[2] CSMA-CA stand for carrier sense multiple access with collision avoidance and is the medium access mechanism for 802.11 based wireless networks.

that correspond to the network. This capacity is the total capacity divided by the number of operational nodes.

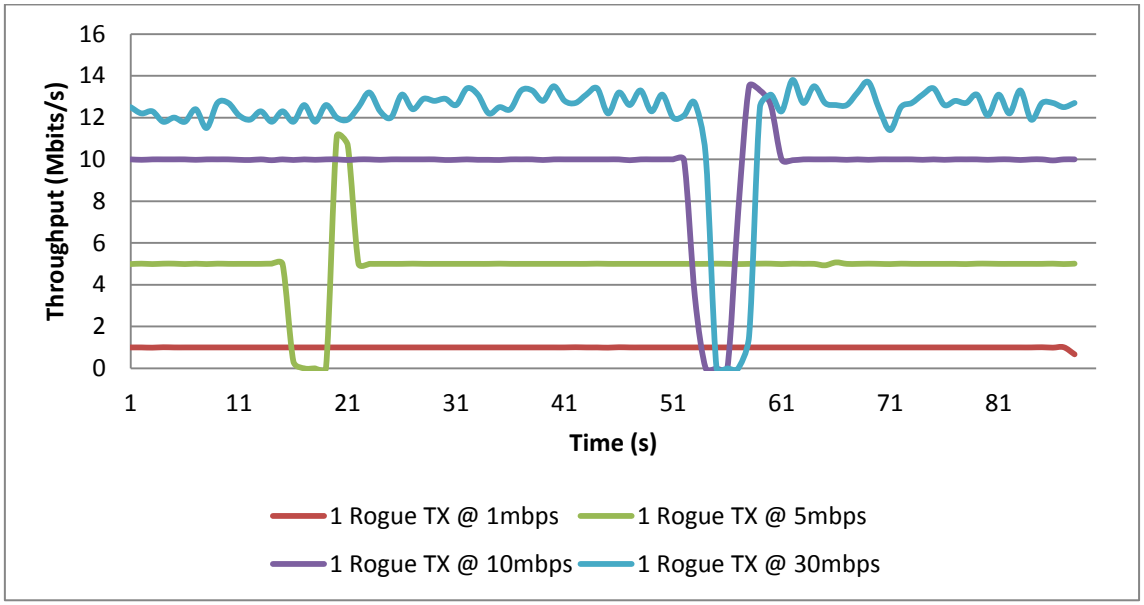

Fig. 3.6. Interfering radio throughput histogram

To illustrate the fair coexistence of the wireless networks within a communication channel the Fig. 3.5 shows the working and rogue throughput histograms. The figure shows that both networks are correlated in a way that if one increases its throughput on one instance the other one decrease its capacity and vice versa. This effect is also appreciable on delay values because when the delay of one network increases the delay of the correlated network decreases.

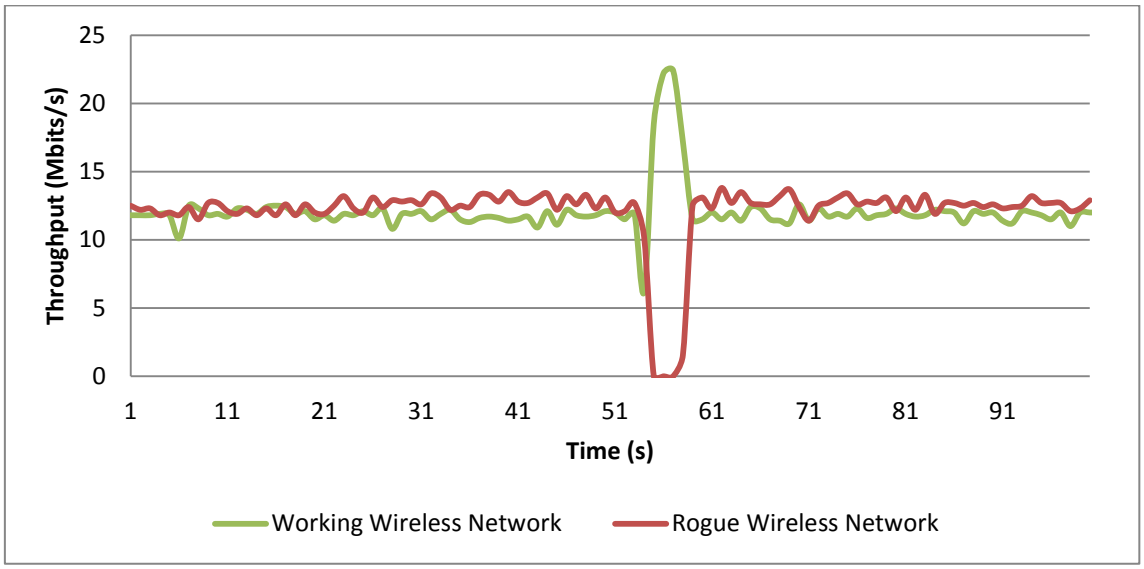

Fig. 3.7.Throughput comparison between working and rogue WLANs TX@30mbps

Up to now the experiments show that the total channel capacity is distributed among the networks that are working at any given time. The Fig. 3.8 shows that the aggregated throughput of the wireless networks is the same as the total capacity of the channel.

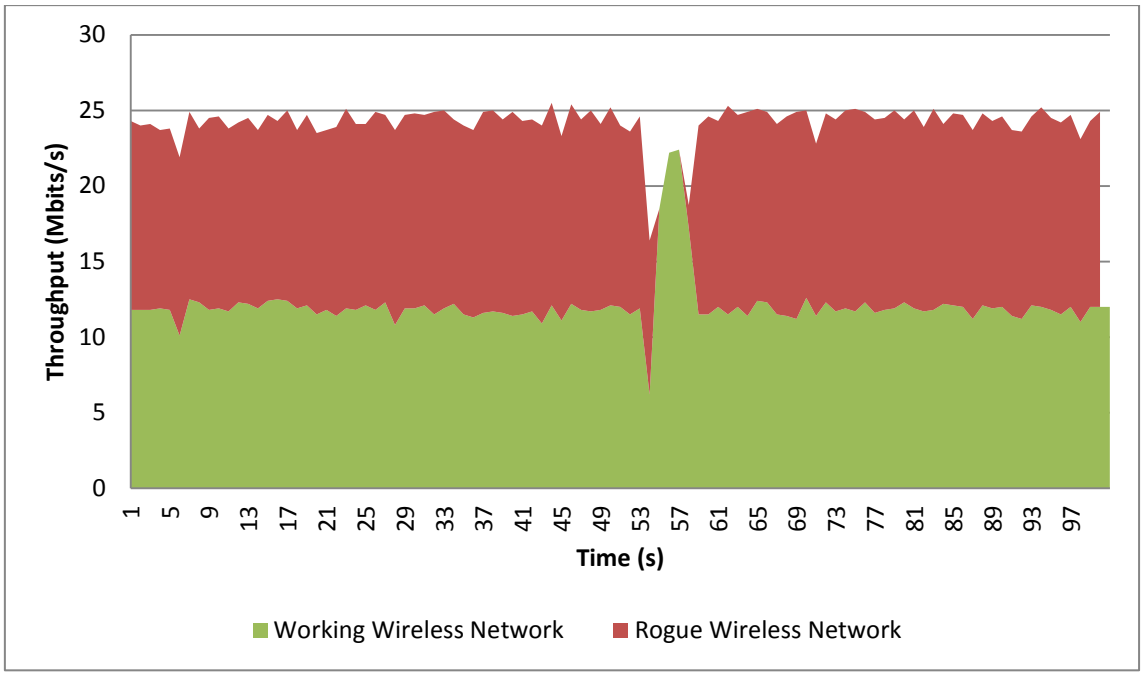

Fig. 3.8. Aggregated throughput of working and rogue WLANs TX@30mbps

The delay values experienced a behavior very similar to the throughput explained before. Delay values increase its value for the working network as interfering radio increase its data rate. This behavior can be noticed on the Fig. 3.9.

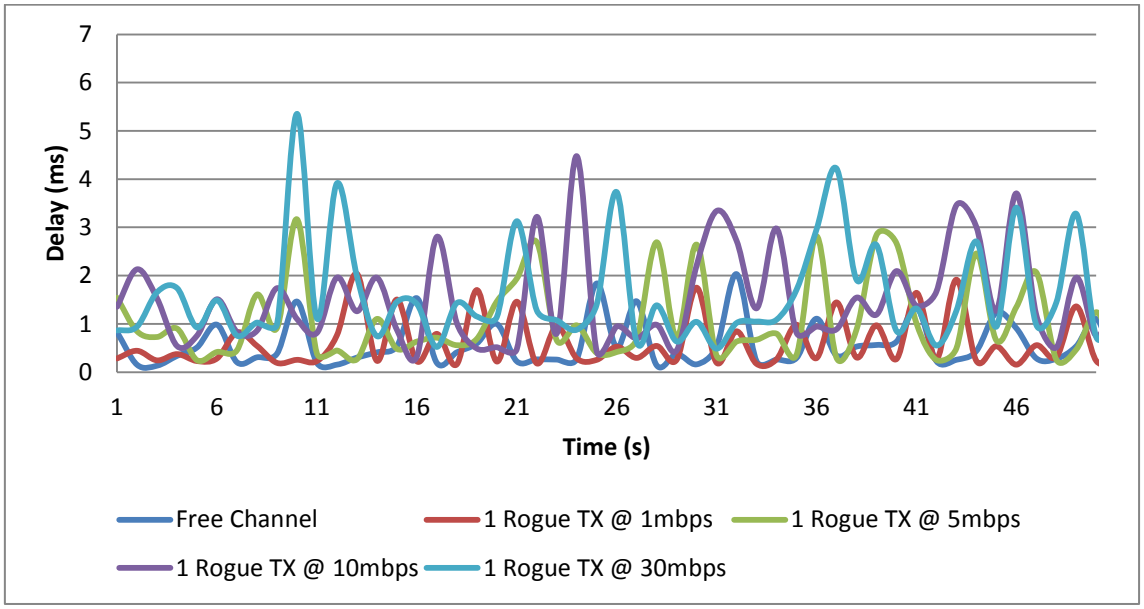

Fig. 3.9. Delay comparison with interfering radios

The results demonstrates that the rogue network affect the working network. The mean values of the performances parameters of the working wireless network are summarized on Table 3.3. The values on the table shows that the delay increases as the rogue network introduce more traffic. The percent of loss packets also increase by the same reason but are very low. Instead the mean throughput is decreased from 22.8Mbps on free channel to 12.72Mbps with a rogue network transmitting at 30Mbps.

Table 3.3. Working wireless network performance metrics comparison between scenarios

Metric\ Scenario	Free Channel	1 Rogue TX @ 1mbps	1 Rogue TX @ 5mbps	1 Rogue TX @ 10mbps	1 Rogue TX @ 30mbps
Throughput (Mbps)	22.865 ± 0.03	21.709 ± 0.21	18.746 ± 0.23	14.962 ± 0.38	12.724 ± 0.45
Delay (ms)	0.572 ± 0.08	0.581 ±0.08	0.963 ± 0.11	1.364 ± 0.15	1.617 ± 0.17
% Losses	0.00000%	0.00406%	0.00209%	0.00459%	0.00462%

3.3.1.3 Metrics comparison

So far only the behavior of performance values was shown. In this section is presented the values of the Kismet metrics that will be used to model the channel state. These values are summarized on the Table 3.4.

Table 3.4. Metrics evaluations between scenarios

Metric\Scenario	Free Channel	1 Rogue TX @ 1mbps	1 Rogue TX @ 5mbps	1 Rogue TX @ 10mbps	1 Rogue TX @ 30mbps
Interfering Radios	0	1	1	1	1
Rogue Packets	0	1134	3032	6299	7919
Rogue Data Packets	0	927	2470	6044	7532
Rogue Bytes	0	1418310	3779100	9247320	11518080
Best SNR	36.481	39.521	39.521	37.349	38.218
Maximum Noise Level	-85	-78	-77	-81	-76

The values of rogue packets, rogue data packets and rogue bytes increases on each test which is obvious because the rogue network is increasing its data rate. But the interesting fact is that even in the test with 1 rogue network transmitting at 30mbps the best SNR values detected for the working wireless network does not vary too much. The fact that the SNR values do not change even if the capacity of our network has been halved tells us that although we have a good signal reception and low noise value that does not mean that our network will operate at full capacity.

Other important thing is that the noise level was increased by the introduction of the rogue network. The variation is too small to say that the change was

because the rogue network. This will be discussed in more detail in the experiment of radio interference in section 3.5.

3.4 Variable transmit power

The objective of this experiment is to evaluate how the variation on the transmit power of an interfering radio affect the working wireless network.

This experiment was conducted in the same location as the previous experiment in order to avoid possible interference from other wireless networks. The location can be appreciated on the Fig. 3.1.

The Fig. 3.10 shows the diagram of how the experiment was carried out.

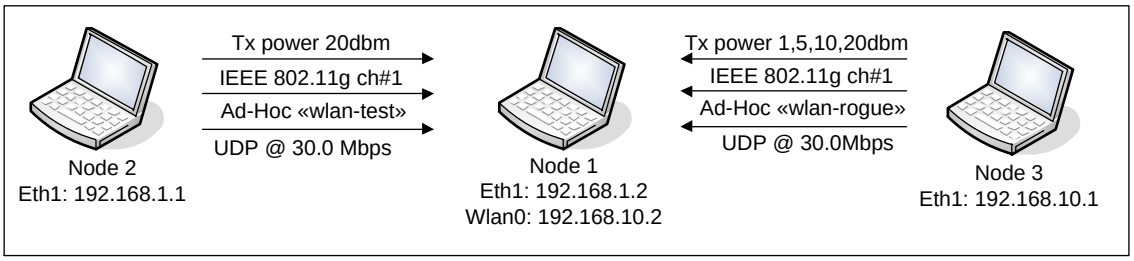

Fig. 3.10. Variable transmit power scenario 1

For this experiment the Node 2 transmit UDP packets at a rate of 30Mbps to flood the WLAN wlan-test. The Node 3 does the same but in the wlan-rogue network. Node 1 receives both traffics on independent wireless cards. The traffic generation and reception was done with the same commands explained on 3.3.

The experiment has 4 test on which the transmit power of Node 3 varies. Node 3 acts as the access point of the rogue wireless network. On each test the transmit power of Node 3 is changed on values 1, 2, 10 and 20dbm. To adjust the transmit power is used the command "*iwconfig eth1 txpower x*" where x is the transmit power in dbm. Each test has duration of 120 seconds.

In the same way that the previous experiment the Kismet software is capturing information about the wireless environment and is running in the Node 2 with the interface Ath1 in monitor mode.

3.4.1 Variable transmit power results

The results of this experiment are evaluated through 3 measures throughput, delay and best SNR detected. On each measure is important to notice if the variability of the transmit power cause a negative effect on the working wireless network.

The throughput comparison can be appreciated on the Fig. 3.11 and shows that the mean throughput values of the working network decrease in a small factor in the same way as the rogue network increases its transmit power. The working wireless network decreases its data rate from 12.8mbps on test 1 to 12.54mbps

on test 4. This is a very small variation so the transmit power does not affect in a significant way the performance of the working WLAN.

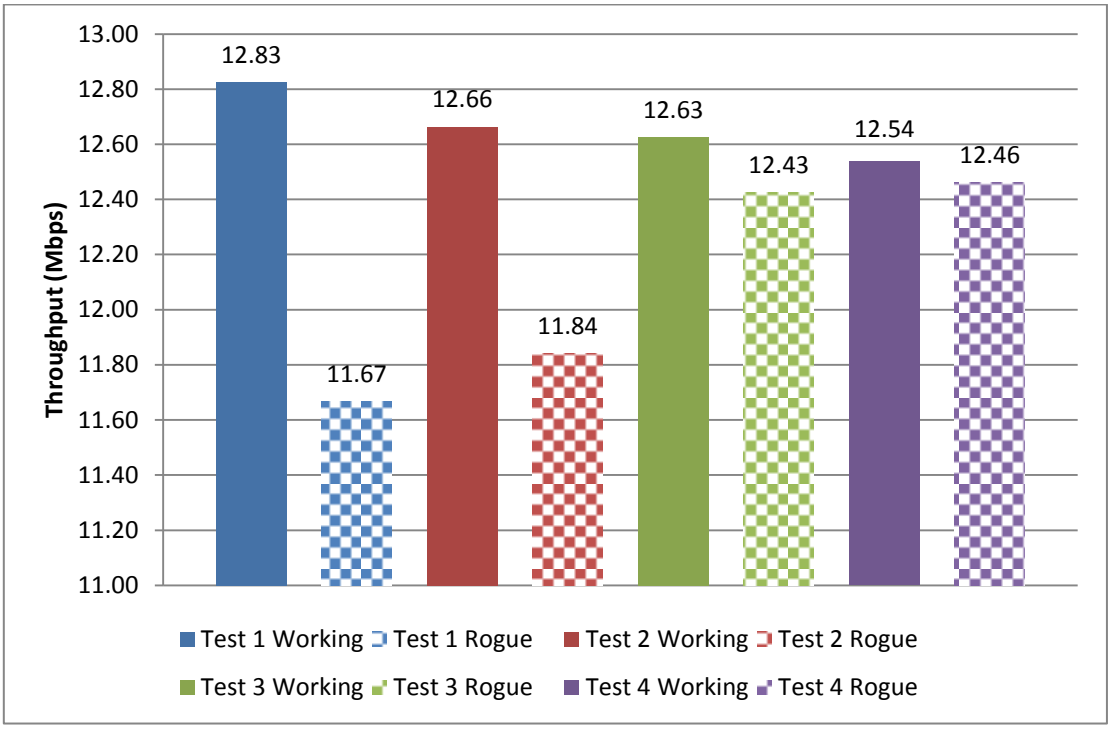

Fig. 3.11. Variable transmit power throughput comparison

The delay values observed on the experiment do not change in a significant quantity. On the previous experiment when 2 networks were sharing the channel the measure of delay was about 1.6ms which are the same mean delay values reported by the wireless network in this experiment.

The Table 3.5 shows the comparison of best SNR values detected by the sniffing software between the working and the rogue WLANs among the four tests. The comparison shows that as the transmit power increases the signal to noise ratio improves for the rogue network, but the best SNR values for the working network are not affected.

Table 3.5. Best SNR values comparison.

	Best SNR (db)			
	Working WLAN		Rogue WLAN	
	Mean	Sta. Dev.	Mean	Sta. Dev.
Test 1 Rogue WLAN tx@1dbm	36.11	0.61	24.52	1.00
Test 2 Rogue WLAN tx@5dbm	35.12	0.16	26.74	1.26
Test 3 Rogue WLAN tx@10dbm	35.12	0.74	29.48	1.53
Test 4 Rogue WLAN tx@20dbm	36.33	0.58	30.65	0.41

The results of this experiment shows that the variation on the transmit power of an interference radio does not have a major effect on the working network. The only detectable effect is that a wireless network with less transmit power cannot achieve the throughput that is able to reach on normal conditions. In the test 1 the rogue network reports a mean throughput of 11.67mbps and in normal condition reach to 12.7mbps. Even if the working network can take advantage of this reduction in traffic it does not represent a significant factor.

The problem of vary the transmit power is mainly the coverage that our network is able to achieve. If the transmit power is reduced the coverage area is also reduced so the wireless network won't be able to deliver services to far nodes.

3.5 Interfering radios

The objective of this experiment is to measure how in normal conditions interfering radios can affect the performance of the working wireless network. To do so, this experiment was carried out at the laboratory 016 of the ETTAC building. In this laboratory several networks are working on different channels so it is perfect to simulate a real environment in which the RF medium is crowded.

The Fig. 3.12 shows the scenario of this experiment.

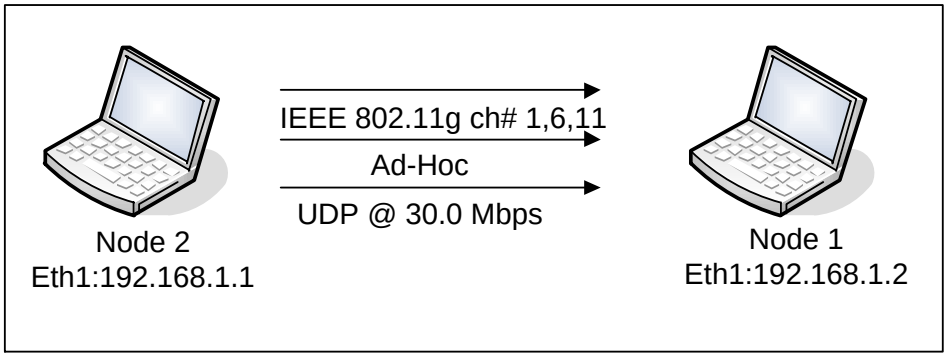

Fig. 3.12. Interfering radios scenario 1

This test is divided on 3 tests in which the communication channel of the wlan-test network is changed. The channels used are the non-overlapping channels of the 2.4GHz band channel 1, 6 and 11.

On each test the Node 2 flood the wireless channel with UDP traffic generated at 30Mbps. The traffic generation and reception was done with the same commands of the Iperf software that were explained on 3.3.

In this experiment the Kismet software is running on Node 2.

3.5.1 Interfering radios results

The Fig. 3.13 shows the comparison between the throughputs achieved by the working wireless network on the three channels in which the experiment was carried out. As can be appreciated the throughput values are variable in time, but the most stable behavior is observed on the channel 6. The channel 1 presents the worst values on the experiment with very low throughput.

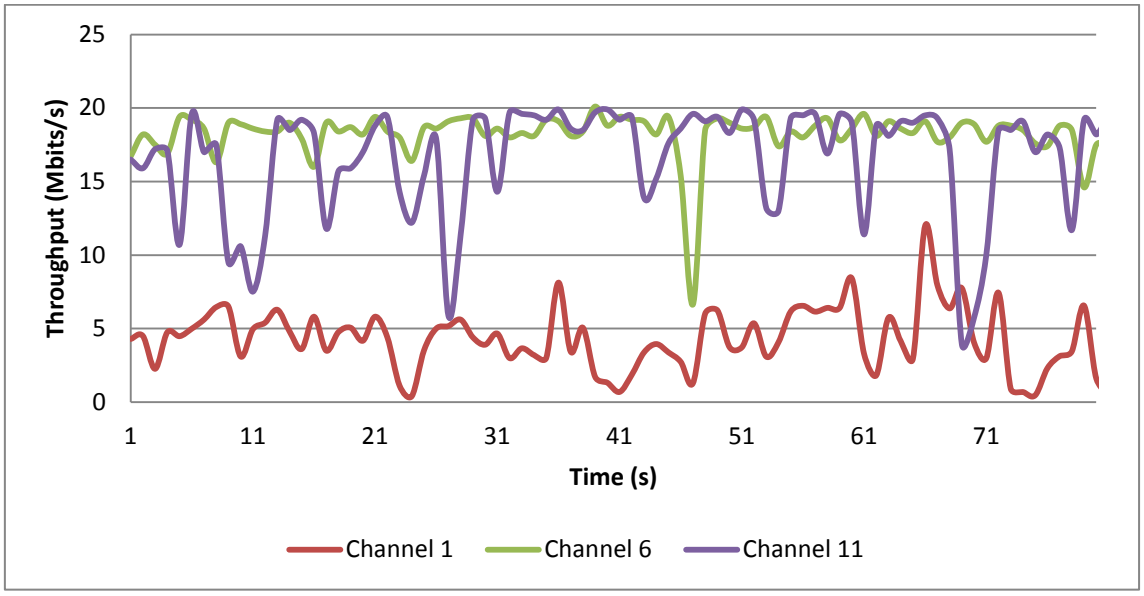

Fig. 3.13. Interfering radios throughput comparison

The delays values are represented on the Fig. 3.14 and in the same way that the worst throughput values are presented on channel 1 the delay observed on this channel are the most significant. Channel 11 and 6 present a similar behavior but the channel 11 have more delay than the channel 6.

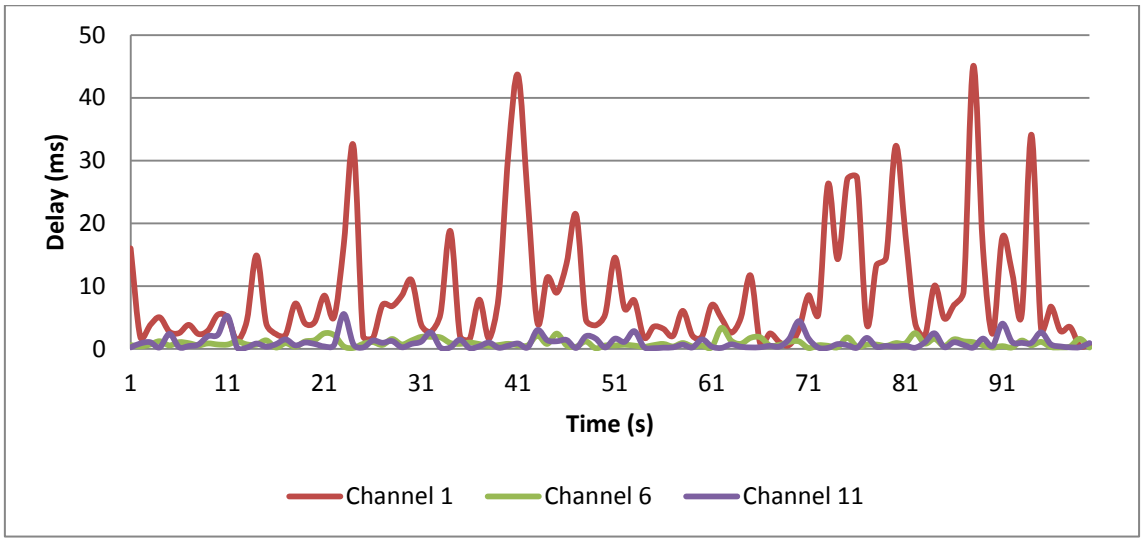

Fig. 3.14. Interfering radios delay comparison

The mean values, the confidence interval and the standard deviation of the performance metrics are presented on the Table 3.6. The best results are observed on channel 6 with 17.93Mbps of throughput, delay values below 1ms and negligible packet loss.

Table 3.6. Performance metrics results

Metric\Scenario	Channel 1	Channel 6	Channel 11
Throughput (Mbps)	6.06 ±0.77	17.93 ±0.33	16.03 ±0.60
Delay (ms)	7.79 ±1.36	0.96 ±0.10	1.10 ±0.19
% Losses	0.0114	0.0006	0.0049

Up to now the previous results demonstrate that the performance of the working wireless network on channel 6 is better that on the other wireless channels. The Table 3.7 shows the values of the metrics that are intended to model the channel state. In order to evaluate the measure in which these metrics model the wireless channel health some graphs will be presented next.

Table 3.7. Channel metrics results

Metric\Scenario	Channel 1	Channel 6	Channel 11
Interfering Radios	15	11	7
Rogue Packets	3868	236	202
Rogue Data Packets	3175	30	92
Rogue Bytes	199091	3202	15936
Best SNR	40.824	38.652	37.784
Maximum Noise Level	-81	-86	-85

The first metric is interfering radios. The Fig. 3.15 represents the comparison between the number of interfering radios detected on each channel and the achieved mean throughput. As can be noticed channel 1 is the most crowded with 15 wireless networks detected and also present the worst performance. Channel 6 have 11 rogue WLANs and achieve the best throughput. Channel 7 shows the smaller amount of rogue networks but the performance on this channel is below the achieved on channel 6.

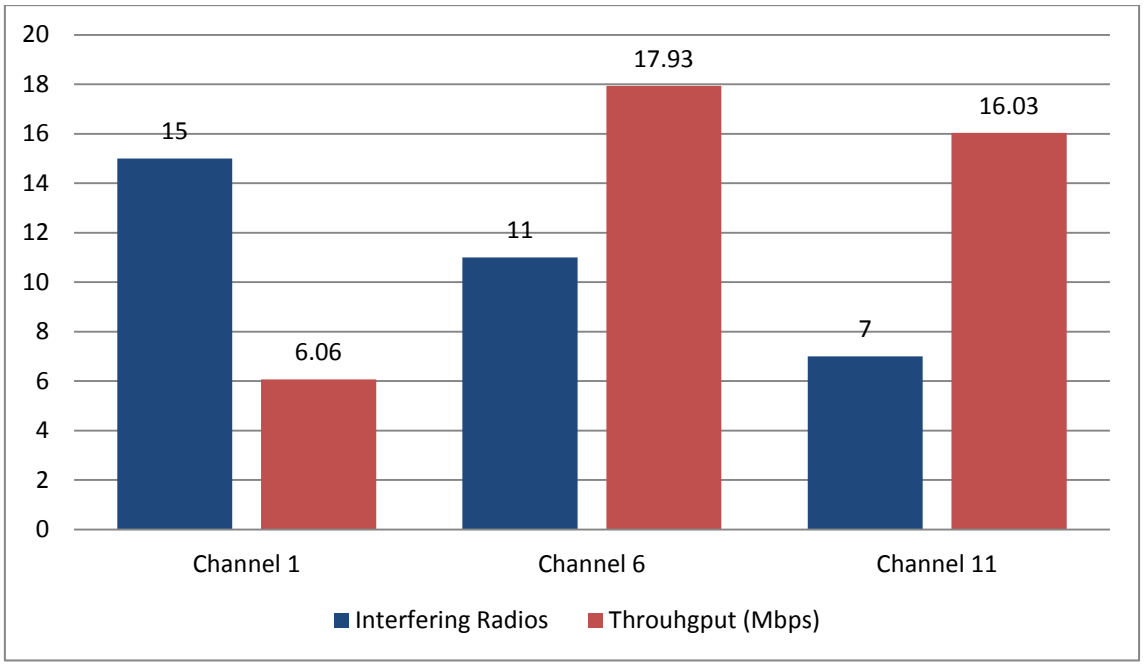

Fig. 3.15. Interfering WLANs VS achieved throughput

The number of interfering radios can be used to forecast that a channel will be crowded if all the WLANs transmit data at the same time. Taking the results of the first experiment we can expect that the 22.9Mbps of the channel capacity will be split fairly among WLAN, so in the channel 1 with 15 interfering radios the working WLAN will not get more than 1/16 of the channel that is 1.4Mbps. The problem with the utilization the number of interfering radios to model the channel is that not all the time the wireless network are occupying the RF medium with traffic. This fact is appreciable on channel 6 that with 11 rogue WLANs we can expect that in the worst case the working WLAN only will be able to get 1/12 of the total channel capacity (1.9Mbps), but instead of this is able to achieve up to 17.9Mbps.

The Fig. 3.16 shows the traffic metrics measured each channel during the experiment. And important fact is the differentiation between rogue packets and data rogue packets. This separation is done because the previous results demonstrate that the performance of a wireless network in a specific channel is more sensitive to traffic patterns than to the amount of WLANs working on that channel. Rogue packets metric quantify the total amount of packets from other networks including control frames (LLC). The data rogue packets values only take into account data frames.

In the values of channel 6 can be noticed that the total number of rogue packets detected is larger than in channel 11 but the number of data packets detected on channel 6 is below the one detected on channel 11. This shows that the data rogue packet metric model in a more accurate way the amount of traffic that is introduced by the rogue WLANs. The total rogue Kbytes is also small on channel 6 with just 3.2KB detected.

The result of the experiment demonstrate that a wireless network will obtain better performance working on a channel with less data packets and as a result of that less kilobytes detected.

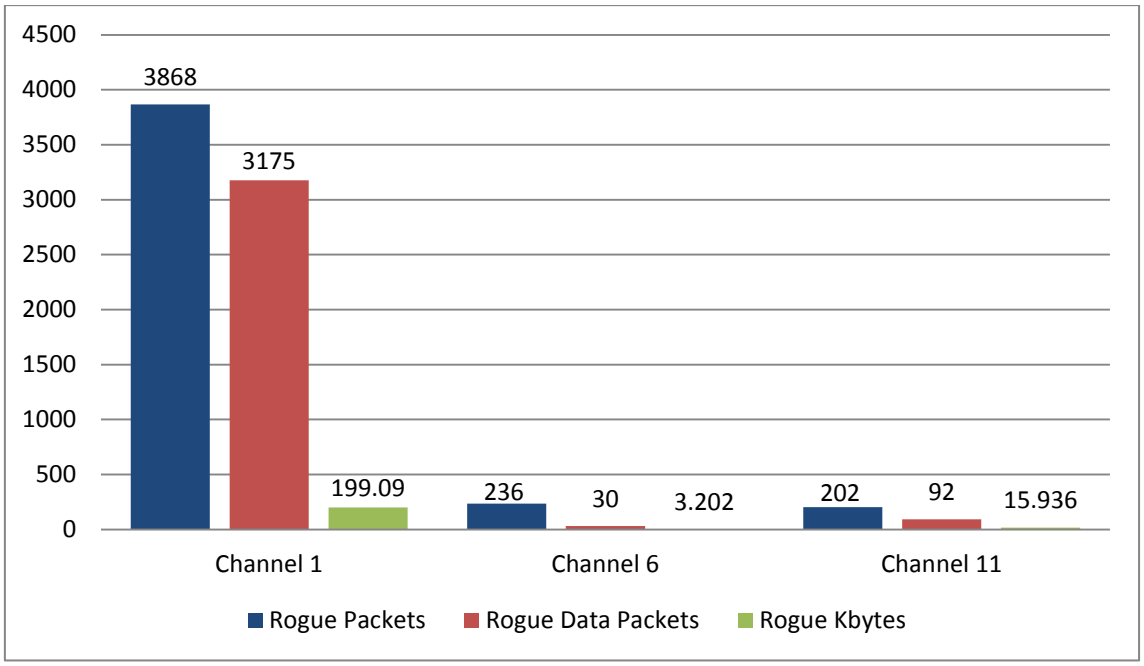

Fig. 3.16. Rogue packets, data packets and Kbytes detected comparison

The last comparison is shown on the Fig. 3.17 in which the detected values of SNR and noise are represented.

The best SNR value detected for the working wireless network on each channel does not vary in a large factor. This value does not show any different among the wireless communication channel.

The noise values show that the channel less noisy is the same channel that reflects the best performance. The values of noise represent a real indicator of the quality on the communication channel because a noisy channel can generate huge degradation on the quality of the communication for any WLAN.

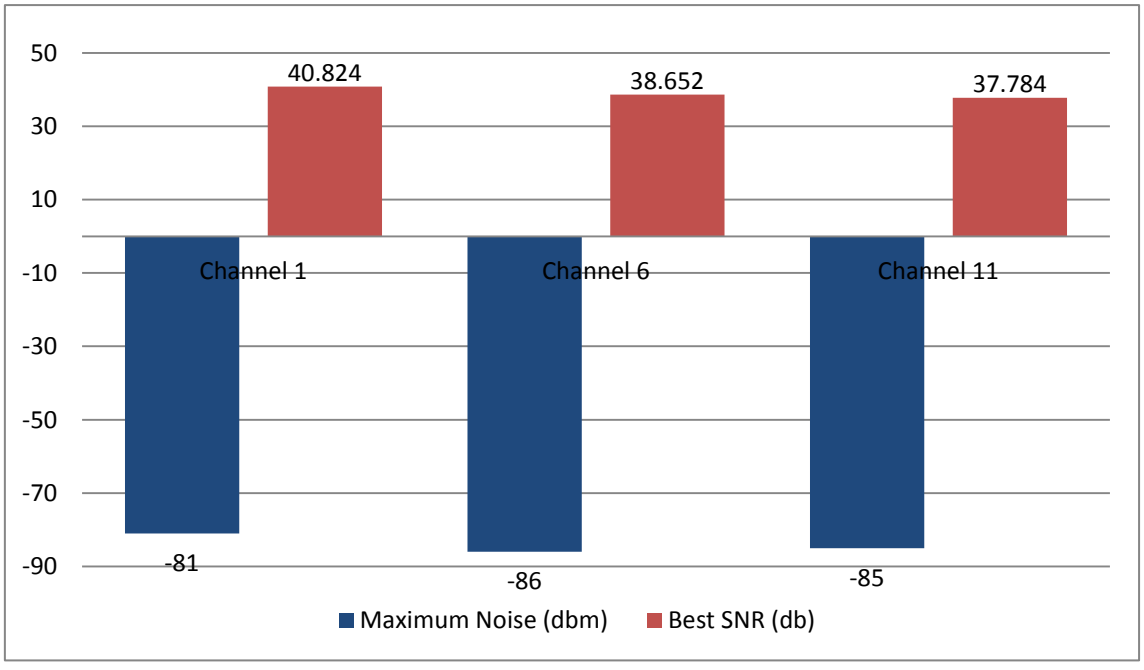

Fig. 3.17. Maximum noise and best SNR comparison

3.6 Correlation

Previous experiment has shown that some of the proposed metrics have impact on the performance of the network. The main objective of this experiment is to evaluate which of the metrics (Rogue data packets, Rogue bytes, Interfering radios and Maximum noise) has more relevancies in order to forecast the maximum capacity that the network will receive on a specific state.

In order to determine this relevancy the experiment explained on section 3.5 was repeated several times in order to obtain enough samples to model the relation between each metric and the maximum achieved throughput.

The correlation coefficient and the R-square values will be calculated from the obtained data for each pair of variables. The correlation coefficient determine whether two data sets are related, and if so, how strongly. The correlation coefficient ranges from +1, indicating a perfect positive linear relationship, to -1, indicating a perfectly negative linear relationship. The R-square is the square value of the correlation coefficient and has a range from 0 to 1 which determines the dependence between each pair of values.

Some changes were done to the parse software to obtain samples during the experiment. The parse software analyzes the data each 30 seconds and the values are compared with the mean throughput achieved within 30 seconds of observation interval.

The experiment uses UDP traffic to flood the working channel. One node type 2 will be running the Kismet software capturing the environmental variables with the Atheros based wireless card. The packet generation and reception is done with the Iperf software.

3.6.1 Results

From the experiment was obtained 117 samples of measures. The samples show results on working on different frequencies, different time of the day and with variable load on each test.
The Table 3.8 shows the correlation coefficient and the R-square values calculated between each of the proposed metrics and the obtained throughput values. The correlation and the R-square values show that the most relevant metrics are the rogue data packets and the interfering radios. Rogue data packets have a correlation value of -0.57 and the interfering radios have a correlation value of -0.83. The negative values of these values just detonates that the variables are inversely proportional.

Table 3.8. Correlation analysis per metric

Metric	Correlation Coefficient	R-Square
Rogue Data Packets	-0.57617	0.3319
Rogue Bytes	-0.43666	0.1906
Interfering Radios	-0.83866	0.7033
Maximum Noise	-0.48716	0.2373
Quality Value	-0.72148	0.5205

The Fig. 3.18 shows a dispersion graph that models the throughput depending of the number of interfering radios detected. This metric is the one that best describe the behavior of the maximum throughput achieved during the tests, however on a high congested channel the number of interfering radio will not give an indication of the quality of the network. This effect can be appreciated on section 3.3.1.2 in which with just one rogue network the overall capacity of the working network is decreased by half.

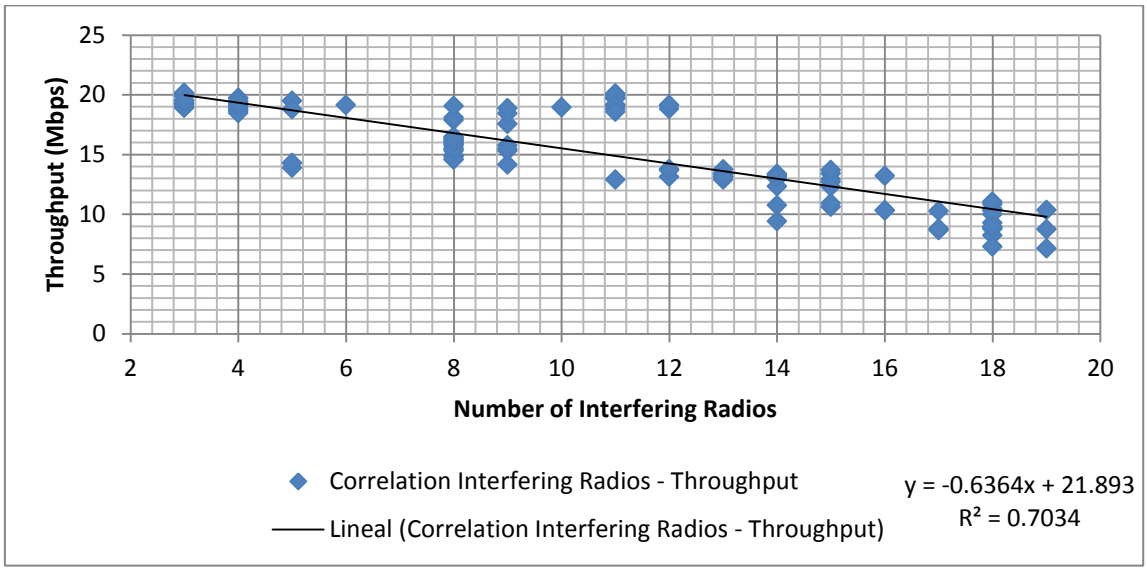

Fig. 3.18. Dispersion graph interfering radios VS throughput and linear regression

The dispersion graph of rogue data packets VS throughput can be appreciated on Fig. 3.19. As can be noticed as the number of rogue data packets increase the capacity is decreased on term of throughput. The correlation between rogue data packets and maximum achieved throughput is not as good as with the number of interfering radios, but on previous experiment was noticed that the traffic patterns on the wireless medium is an important fact due to the fact that all the available WI-FI networks must share the medium.

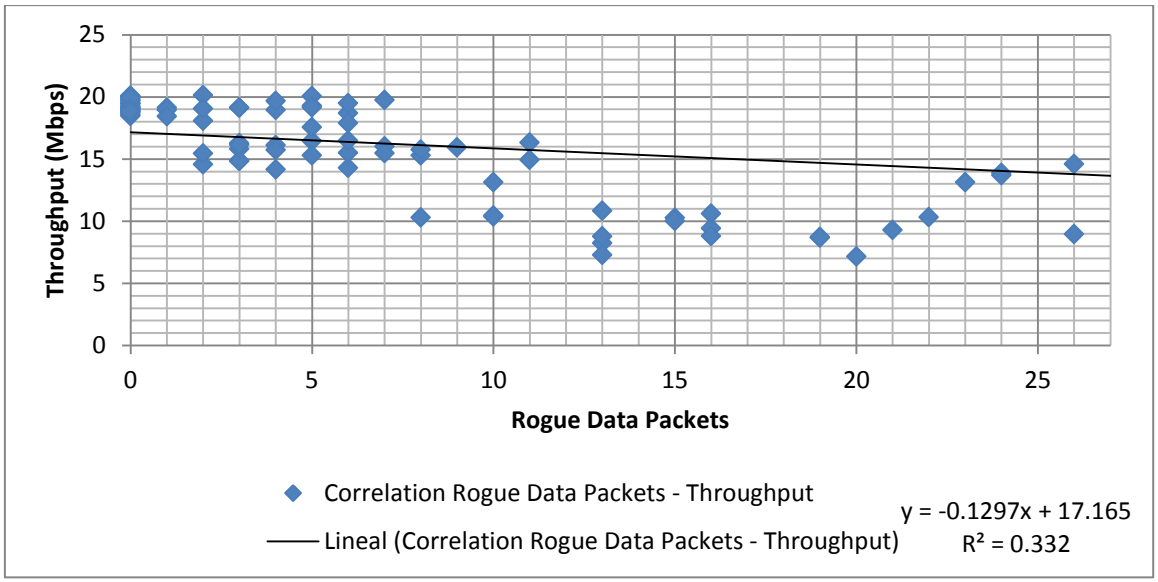

Fig. 3.19. Dispersion graph rogue data packets VS throughput and linear regression

The two metrics explained before are very important to evaluate the overall capacity of a wireless network but each one of them have a problem on some scenarios. The rogue data packets won't be a good estimation of quality if there is no traffic but a very crowded channel in which all the clients executes the CSMA-CA algorithm to access the medium and sending control frames that are not market as data traffic. The interfering radios will not model the best capacity of the network on congestion situation. For these reasons a new metric was created called Quality value. The correlation coefficient and the R-square values are shown on Table 3.8.

This metric is a weighted sum by the correlation coefficients of the metrics rogue data packets and interfering radios. The value is obtained from the expression equation 3-1 in which the IR means the number of interfering radios detected on the monitoring window, the IRCC is the correlation coefficient between the interfering radios and the throughput, the RDP is the number of rogue data packets detected within the monitoring windows and the RDPCC is the correlation coefficient between the rogue data packets and the throughput.

$$QV = \frac{((IR*IRCC)+(RDP*RDPCC))}{IRCC+RDPCC} \quad (3\text{-}1)$$

This new metric will forecast in a better way the capacity of the network because take into account the most two important metric. The interfering radios have better relevance in the calculation of the quality value because its correlation coefficient is greater than the one of the rogue data packets. Although the correlation coefficient and the R-square values of the quality value is worse than the interfering radios and better that the rogue data packets, is expected that this new metric have a better approach of the maximum available capacity on situations of crowded and congested channels.

The Fig. 3.20 show the dispersion graph of the quality value VS the throughput. The figure also shows the linear regression line and the respective equation which will be used to estimate the channel capacity in the purposed algorithm which will be explained later in the document.

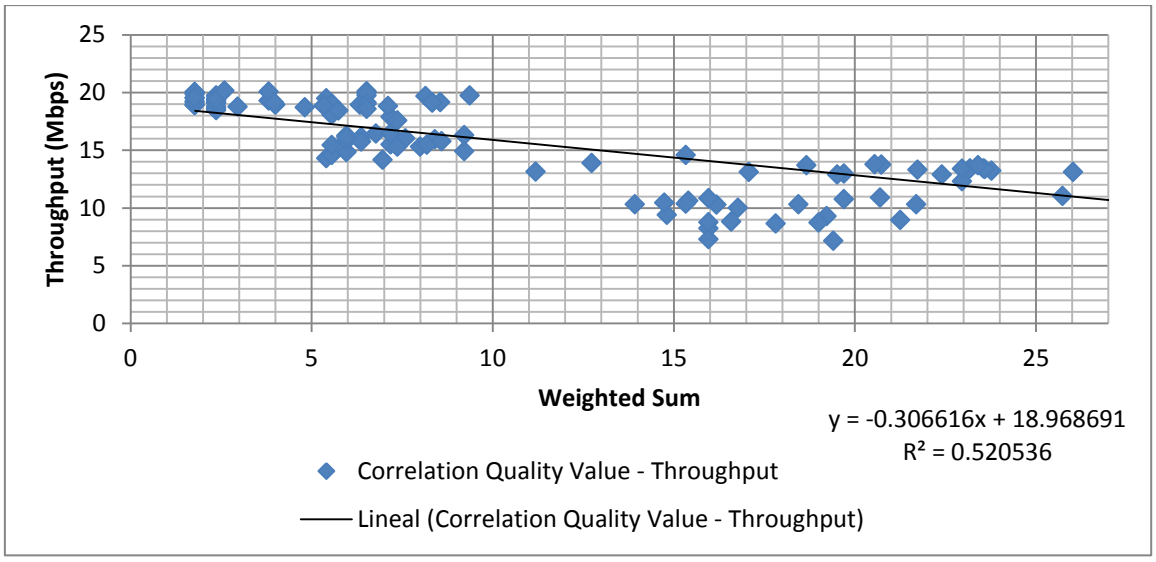

Fig. 3.20. Dispersion graph quality value VS throughput and linear regression

CHAPTER 4. PROTOTYPE IMPLEMENTATION

This section explains in detail the development of the prototype that performs the context aware self-configuring function in a standard WI-FI network. The context adaptation is based on the dynamic channel selection among the 3 non-overlapping channels available on the 2.4GHz ISM band.

The objective of the prototype is to prove that a system like this works on the test bet and that it can be implemented, in the future, on the most popular open source operative systems for wireless routers such as OpenWrt[3] and DD-WRT[4].

The implementation of the prototype was done on one of the nodes type 2(see 3.2). This node was provisioned with two wireless cards. The internal card is used as the sniffing source interface for the Kismet software. For this reason this interface works in monitor mode. The second wireless interface is the Atheros PCMCIA card which is used as an access point. The Atheros card supports the operative mode “master” that simulate the normal behavior of a wireless access point.

In addition to the above described the node two must executes the prototype logic application. This application contains the XML parse function and the context aware decision algorithm. The application is text-based and was developed on the Java language programming. The election of this programming language was done due to simplicity and previous knowledge on the language.

From the explained up to know it is possible to infer several important blocks that interact between each other to perform the adaptive function of the proposed prototype. The Fig. 4.1 is a block diagram that shows the interaction between the main components of the implemented prototype.

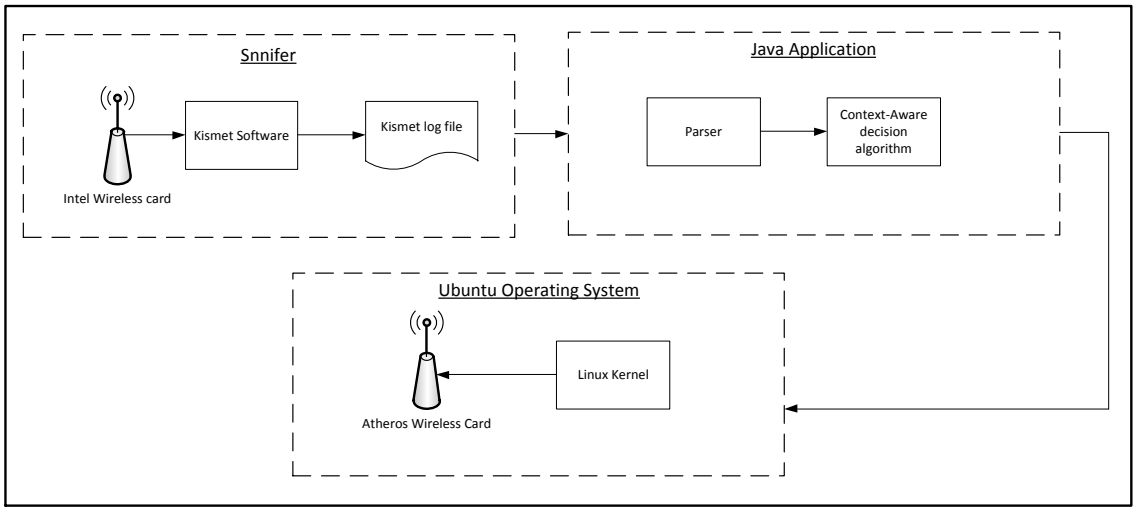

Fig. 4.1 Prototype block components interaction

[3] https://openwrt.org/
[4] http://www.dd-wrt.com/site/index

The sniffer block is in charge of obtains the environment data and save it in an XML file. This block is composed by two entities the Intel wireless card which works with the IPW2200 Linux driver and the Kismet software.

The Intel wireless card works on monitor mode and collect frames form the air. These frames are passed to the Kismet server who processes them and generates an XML file with all the detected wireless networks and information about each one of them. An example of the XML structure is shown in the Fig. 4.2.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE detection-run SYSTEM "http://kismetwireless.net/kismet-3.1.0.dtd">

<detection-run kismet-version="2011.03.R2" start-time="Thu May  5 11:17:25 2011">

<card-source uuid="7da88620-76f8-11e0-aa5f-c3041820e201">
 <card-source>wlan0:type=madwifi,channellist=IEEE80211b</card-source>
 <card-name>wlan0</card-name>
 <card-interface>wlan0</card-interface>
 <card-type>madwifi</card-type>
 <card-packets>43364</card-packets>
 <card-hop>true</card-hop>
 <card-channels>1,6,11,2,7,3,8,4,9,5,10,12,13,14</card-channels>
</card-source>
  <wireless-network number="1" type="ad-hoc" first-time="Thu May  5 11:20:22 2011" last-time="Thu May  5 11:20:22 2011">
    <BSSID>00:00:00:00:00:00</BSSID>
    <manuf>Unknown</manuf>
    <channel>0</channel>
    <freqmhz>2412 1</freqmhz>
    <maxseenrate>11000</maxseenrate>
    <carrier>IEEE 802.11b+</carrier>
    <encoding>PBCC</encoding>
     <packets>
       <LLC>0</LLC>
       <data>1</data>
       <crypt>0</crypt>
       <total>1</total>
       <fragments>0</fragments>
       <retries>0</retries>
     </packets>
```

Fig. 4.2. Portion of the Kismet XML file

The Java application block is in charge of process the output file generated by the Kismet application and with the obtained metrics executes the context-aware decision algorithm to select the best available channel. This block is composed basically by two elements the parser and the context-aware algorithm.

The parser processes the XML file and assigns the information to the classes defined in the java application. The java application contains two main classes WLAN and Channel. The WLAN class is a data structure that contains relevant information about the detected wireless networks. The most relevant fields are: network id, network type, SSID, operative channel, BSSID, number of packets, number of data packets and maximum noise. The Channel class is a data structure with information about the channels. The fields of the structure are: channel id, number of interfering WLAN, rogue packets, rogue data packets, rogue bytes and maximum noise level.

The context-aware decision algorithm is in charge of the decision of which is the current best channel to work by processing the data stored on the Channel class. The algorithm is explained with detail in section 4.1.

The last block is the Ubuntu operating system. This block is in charge of the change the operative when is needed. This block contains two elements the Linux Kernel and the Atheros wireless card.

The Linux Kernel uses the MadWifi driver to manage the wireless card as a standard access point. When the algorithm decides to change the actual wireless channel it sends a Linux terminal command to change the actual configuration. This command through the Linux Kernel is applied to the actual MadWifi configuration. In order to send commands to the Linux Kernel from the Java application the Process class is used.

An important point in this scheme is the fact that when the channel is changed the driver itself informs the connected clients of this events and automatically the clients change the configuration in order to match with the access point request.

4.1 Context-Aware decision algorithm

The decision algorithm is based on two main processes: the calculation of the quality values and the comparison between the expected capacity on the best channel and the current capacity.

First of all the application executes the parse function in order to fill a matrix called $WNETs_{ij}$ with the information of each wireless network detected within the access point coverage area.

After filling completely the network matrix the program starts with the calculation of the quality values associated to each channel. This quality value models the expected maximum capacity on each channel. The range of possible values is from 0 to infinite where lower values represent better quality.

The quality value is obtained from the weighted sum of the detected number of interfering networks and the number of detected data packets scaled by the correlations coefficient obtained in 3.6.1. The expression used to calculate the quality value was presented on equation (3-1). The system creates a vector QV_i with the calculated result for each channel.

After the vector QV_i is fully generated, the program selects the lower quality to obtain the best available channel. The best channel number is stored on a variable called best_channel.

In the next stage the algorithm verifies if the actual iteration is the first one. This is done because after the first iteration the system must be working on the best available channel. If the actual iteration is the first iteration the system evaluates if the best channel is the actual operative channel, if it is not the best channel a Linux command is called to change the operative channel to the actual best channel.

If the iteration is not the first one the system calculates the capacity per channel. This capacity is estimated from the linear regression equation obtained from the correlation between the quality value and the throughput. The equation (4-1) shows the way by which the capacity value is calculated for each channel. The values calculated are stored on a vector called CV_i.

$$CV_i = (-0.306616 * QV_i) + 18.968691 \quad (4\text{-}1)$$

Once CV_i is calculated the system verifies if the best channel capacity represent an improvement of at least 10% regarding the actual operative channel capacity. If this constraint is fulfilled then the system changes the operative channel to the best channel. If the constraint is not satisfied the channel does not change. The objective of this constraint is to avoid loops between channels due to small variations on the measured metrics. Moreover this percentage is a variable indicator that defines the aggressiveness of the system against the changes on the wireless environment.

After all the previous processes the system remains in an idle state during the measurement interval. In this case the idle time has been set to 30 seconds.

The Fig. 4.3 shows the flux diagram of the prototype logic.

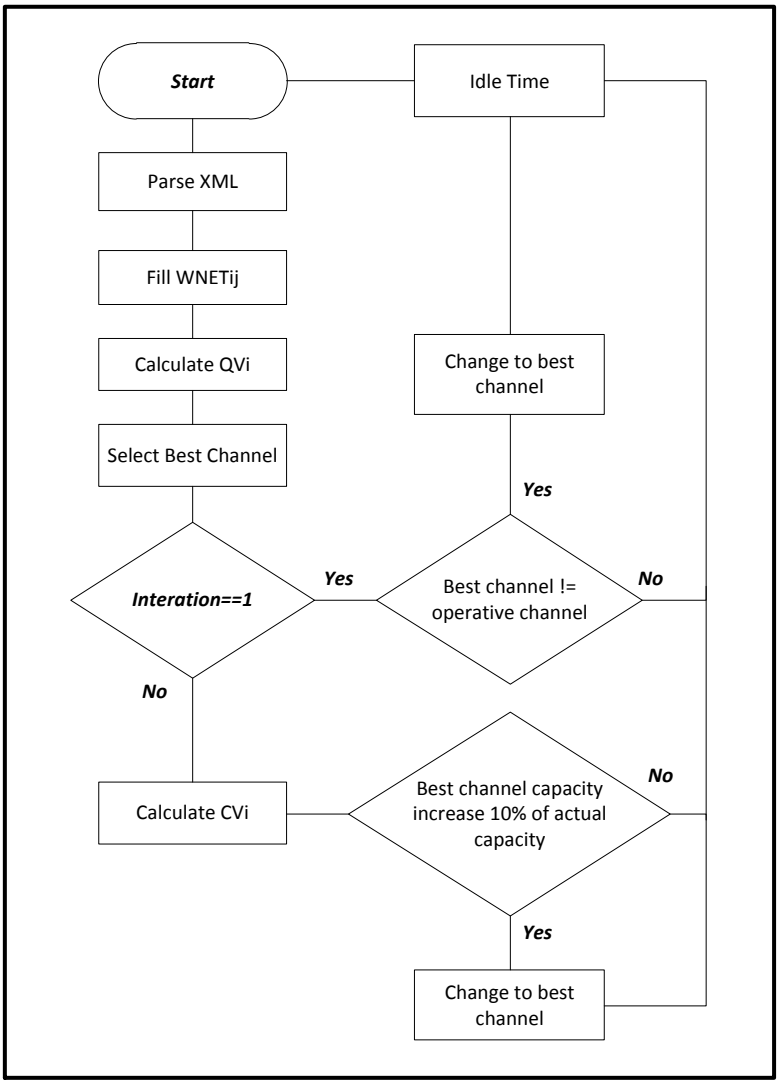

Fig. 4.3. Flux diagram of the prototype logic

4.2 Prototype testing

This section explains the experiment carried out in order to demonstrate the effectiveness of the proposed context-aware decision algorithm. The proposed algorithm is compared with two approaches: a random algorithm and a static algorithm.

4.2.1 Experiment

The experiment scenario can be appreciated on Fig. 4.4. In this scenario the node 1 is a node type 2 with the prototype implementation. The node 2 and node 3 are clients of the SSID “wlan-test”. This wireless networks was configured with the security mechanism WEP and a pre-shared key of 128 bits. In this scenario node 3 sends UDP packets to node 2 through the access point. The traffic generation and reception is done in the same way like previous experiments.

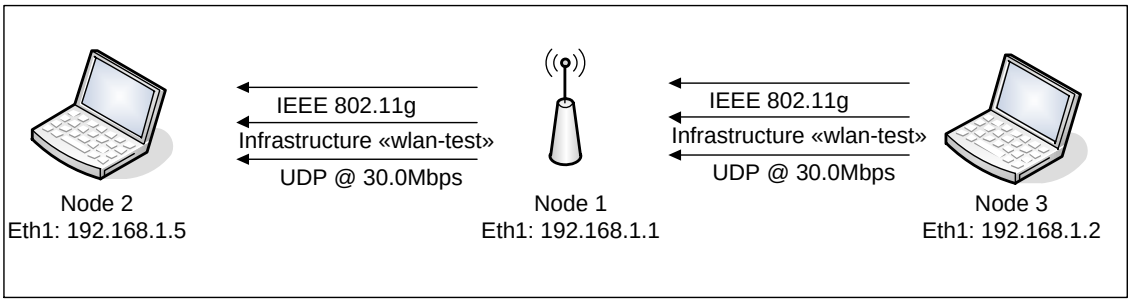

Fig. 4.4. Prototype testing scenario

This experiment consists in three separated test. On each test the start channel is the 11. The duration of each test is about 600 seconds. At the second 300 a rogue wireless network is introduced on the operative channel. This rogue network transmits UDP traffic at 30Mbps. The idea behind the introduction of the rogue network is to evaluate how the proposed algorithm changes the channel in order to maximize the throughput. At the second 450 the rogue network stops transmitting packets.

The entire experiment was carried out in the lab 016 of the EETAC campus. This laboratory has a very crowded wireless environment with several WI-FI networks working on the non-overlapping channels, so this environments is a good example of real uncoordinated AP deployment.

The first test executes the proposed context-aware decision algorithm. The second test executes a random algorithm and the last test does not change its configuration.

The random algorithm is a modified version of the java application developed for the prototype. The application generates a random number between 1 and 3. If the elected number is 1 the best channel is 1; however if the elected number is 2 the selected best channel 6, and is the random number is 3 the best channel is channel 11. The application verifies on each time interval if the actual working channel is the generated best channel and if not a Linux command is called to change the actual channel to the new best channel.

4.2.2 Results

The Table 4.1 shows the averages results of the quality metrics for each test. As can be observed the proposed algorithm gives the best average throughput of 12.55Mbps compared with the random algorithm 9.57 Mbps and the static algorithm with 10.99Mbps. The delays values are also better in our implementation.

Table 4.1. Prototype testing result comparison

Metric	Proposed Algorithm	Random Algorithm	Static Algorithm
Throughput (Mbps)	12.55 ±0.40	9.57 ±0.77	10.99 ±0.34
Delay (ms)	1.78 ±0.26	11.18 ±1.23	2.4 ±0.24

The overall result shows that the capacity of the wireless network is increased by the adaptation of the proposed system. Know is important to analyze the individual behavior of the working WLAN on each test.

The Fig. 4.5 shows the throughput histogram of the random algorithm. This algorithm is very variable due to its random nature. Even with the fact that the number of possible channels for change is reduced so there is a high probability that the chosen channel were the real best available channel the algorithm provides a very low quality.

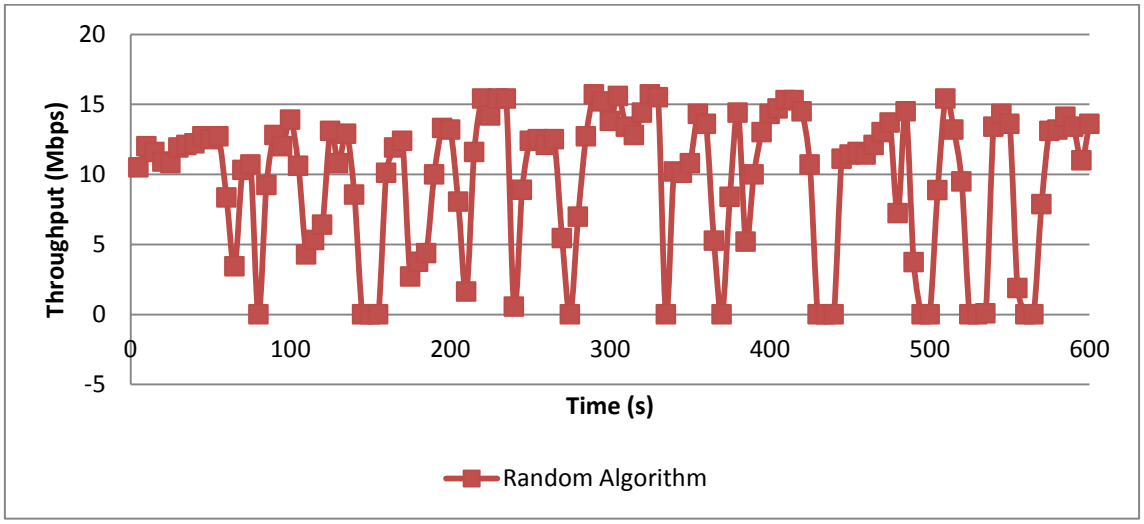

Fig. 4.5. Random algorithm throughput histogram

The Fig. 4.6 shows the throughput histogram of the static algorithm. An important point to highlight in the result of this test is the fact that within the time were the rogue network transmits data (300s-450s) the capacity of our WI-FI is seriously affected. This is caused because both networks have to share the same medium. In the rest of the experiment the channel 11 works well because

it has more or less the same quality value as the best channel. This fact will be appreciated on the analysis of the proposed algorithm.

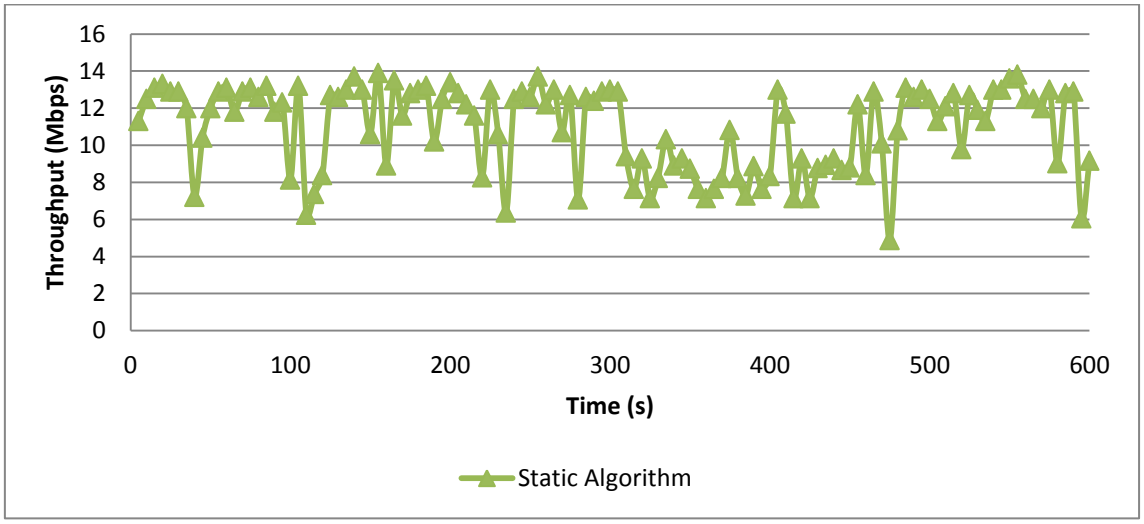

Fig. 4.6. Static algorithm throughput histogram

The Fig. 4.7 shows the throughput histogram of our proposed algorithm. As can be seen the decision algorithm, select the best channel in order to increase the maximum capacity of the network. The most relevant point on the graph is at second 300 where the rogue network starts to transmit. After 1 time interval the algorithm detects de congestion on the operative channel and changes its configuration. This is reflected on the increase on throughput from the second 340 of the test.

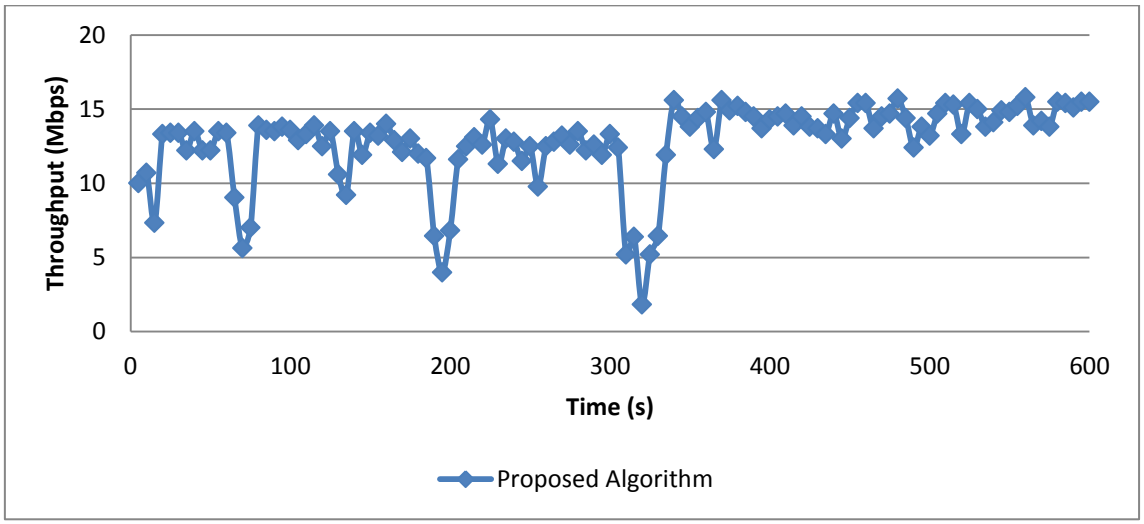

Fig. 4.7 Proposed algorithm throughput histogram

The Table 4.2 shows the more important output data from the Java application. On iteration 2 the algorithm selects the channel 11 as the best channel with a quality value of 6.928 and an associated capacity of 16.845. The decision of the algorithm is to stay on channel 11 because it was the operative channel at that moment.

In iteration 3 the system detects that the new best channel is the channel 1, but compared with the actual expected capacity it does not provides more than the 10% of improvement so the algorithm decides to stay on the channel 11.

In the iteration 11 the prototype detects the presence of congestion from a rogue WLAN. This can be appreciated on the quality values 22.181 of the channel 11. The algorithm decides to set the operative channel to 1. Is important to denote that in this case channel 1 provides an expected capacity if 16.706Mbps which if compared with the actual capacity on channel 11 of 12.168 represents an improvement of 27%.

The last iteration decides to stay on channel 1 because the channel 1 does not satisfy the 10% of improvement.

Table 4.2. Context-aware decision algorithm behavior

Iteration	Time on experiment(s)	Channel	QV_i	CV_i	Operative Channel	Decision
2	70	1	7.186	16.765	11	Best channel 11 Stay on channel 11
		6	15.036	14.358		
		11	6.928	16.845		
3	100	1	6.149	17.083	11	Best channel 1 Stay on channel 11
		6	7.706	16.606		
		11	7335	16.72		
11	340	1	7.371	16.709	1	Best channel 1 Change to channel 1
		6	9.113	16.174		
		11	22.181	12.168		
19	580	1	7.778	16.584	1	Best channel 11 Stay on channel 1
		6	8.891	16.242		
		11	7.52	16.663		

With this analysis is demonstrated that the proposed prototype works and take the decision of set a net channel only of the best channel provides a significant improvement. Is important to highlight the fact that the proposed algorithm provides a stable behavior in normal conditions with just 1 real channel variation during the test.

CHAPTER 5. CONCLUSIONS

The actual uncoordinated deployments of wireless networks based on the 802.11 standard creates a very competitive environment in which is necessary to share the limited spectrum available within the congested 2.4GHz ISM band.

A context-aware system is very useful in order to reduce the interference of neighbor's wireless networks through the intelligent election of communication channel to improve the throughput. A scheme like this permits a better utilization of the non-overlapping channel distributing the neighbors WLAN among them.

Through several tests was demonstrated that wireless networks working on the same channel can harm the performance of the working WLAN in terms of throughput and delay.

During the deployment of this thesis some metrics were proposed in order to model the capacity on each wireless channel. From those metrics the number of interfering radios and the numbers of data packets were the most relevant and the ones that model in an accurate way the real capacity of a communication channel. From these two metric a new variable was created as a weighted sum scaled by the correlation coefficient of each metric. This new proposed variable called quality value takes into account both metrics and is used to select the best available channel on the proposed prototype.

Even that our metrics gives us an approximated value for the capacity of the channel. More experiment can be done on different locations in order to obtain more samples with the relation between the metrics and the measured throughput. This can be done to improve the correlation coefficients of the metrics.

The prototype experiment demonstrates the viability of an implementation of the context-aware self-configuring system. On normal conditions the proposed algorithm select the actual best channel and decides of it is necessary to set a new channel only if this new channel improves the throughput in a significant amount.

Futures context-aware systems can make others configurations changes in order to increase even more the overall capacity of the wireless network especially on deployments of wireless mesh in which the algorithm can allocate the channels among the different link in order to reduce the interference between them.

A future trend of this work can be the implementation of the algorithm inside a real wireless access point. The wireless access point can use the OpenWrt or the DD-WRT operating systems. Both systems have working distributions of the kismet software, so the sniffing scheme presented on this document can fit perfectly.

CHAPTER 6. BIBLIOGRAPHY

[1]. Akella, A., G. Judd, S. Seshan, and P. Steenkiste. 2007. *Self-management in chaotic wireless deployments*. Wireless Networks 13 (6): 737-55.

[2]. Afanasyev, M., Tsuwei Chen, G. M. Voelker, and A. C. Snoeren. 2010. *Usage patterns in an urban WiFi network*. Networking, IEEE/ACM Transactions on 18 (5): 1359-72.

[3]. Kauffmann, B., F. Baccelli, A. Chaintreau, V. Mhatre, K. Papagiannaki, and C. Diot. 2007. *Measurement-based self-organization of interfering 802.11 wireless access networks*. Paper presented at INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE.

[4]. Mishra, A., S. Banerjee, and W. Arbaugh. 2005. *Weighted coloring based channel assignment for WLANs*. ACM SIGMOBILE Mobile Computing and Communications Review 9 (3): 19-31.

[5]. Mishra, A., V. Shrivastava, D. Agrawal, S. Banerjee, and S. Ganguly. 2006. *Distributed channel management in uncoordinated wireless environments*. Paper presented at Proceedings of the 12th annual international conference on Mobile computing and networking.

[6]. da Silva, M. W. R., and J. F. de Rezende. 2009. *TDCS: A new mechanism for automatic channel assignment for independent IEEE 802.11 networks*. Paper presented at Ad Hoc Networking Workshop, 2009. Med-Hoc-Net 2009. 8th IFIP Annual Mediterranean.

[7]. Ramachandran, K. N., E. M. Belding, K. C. Almeroth, and M. M. Buddhikot. 2006*. Interference-aware channel assignment in multi-radio wireless mesh networks*. Paper presented at INFOCOM 2006. 25th IEEE International Conference on Computer Communications.

[8]. Meraki [Online]. Available: http://www.meraki.com

[9]. Meru Networks [Online]. Available: http://www.merunetworks.com

[10]. Aruba Networks [Online]. Available: http://www.arubanetworks.com

[11]. IEEE standard for information technology--telecommunications and information exchange between systems--local and metropolitan area networks--specific requirements part 11: *Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 8: IEEE 802.11 wireless network management2011. IEEE std 802.11v-2011 (amendment to IEEE std 802.11-2007 as amended by IEEE std 802.11k-2008, IEEE std 802.11r-2008, IEEE std 802.11y-2008, IEEE std 802.11w-2009, IEEE std 802.11n-2009, IEEE std 802.11p-2010, and IEEE std 802.11z-2010)*.

[12]. Pejman Roshan, Jonathan. 2003. *802.11 wireless LAN fundamentals*. United States of America: Cisco Press.

[13]. Kismet tool [Online]. Available: http://www.kismetwireless.net/

[14]. Iperf tool [Online]. Available: http://sourceforge.net/projects/iperf/

[15]. MadWifi driver [Online]. Available: http://madwifi-project.org/

[16]. Dujovne, D., T. Turletti, and F. Filali. *2010. A taxonomy of IEEE 802.11 wireless parameters and open source measurement tools*. Communications Surveys & Tutorials, IEEE 12 (2): 249-62.

[17]. Wu, D., P. Djukic, and P. Mohapatra. 2008. *Determining 802.11 link quality with passive measurements*. Paper presented at Wireless Communication Systems. 2008. ISWCS '08. IEEE International Symposium.

[18]. Gupta, D., P. Mohapatra, and Chen-Nee Chuah. 2008. Efficient monitoring in wireless mesh networks: *Overheads and accuracy trade-offs. Paper presented at Mobile Ad Hoc and Sensor Systems*, 2008. MASS 2008. 5th IEEE International Conference.

CHAPTER 7. ANNEXES

7.1 MadWifi installation tutorial

The objective of this document is to give a brief description of the procedures needed to get, install and use the MadWifi driver. The equipment that used for the creation of this tutorial is a laptop Compaq nx6110 running Ubuntu 10.10. The wireless card that will use MadWifi is a PCMCIA Atheros AR5001x. The complete description of the wireless card is shown in the Fig. 7.1.

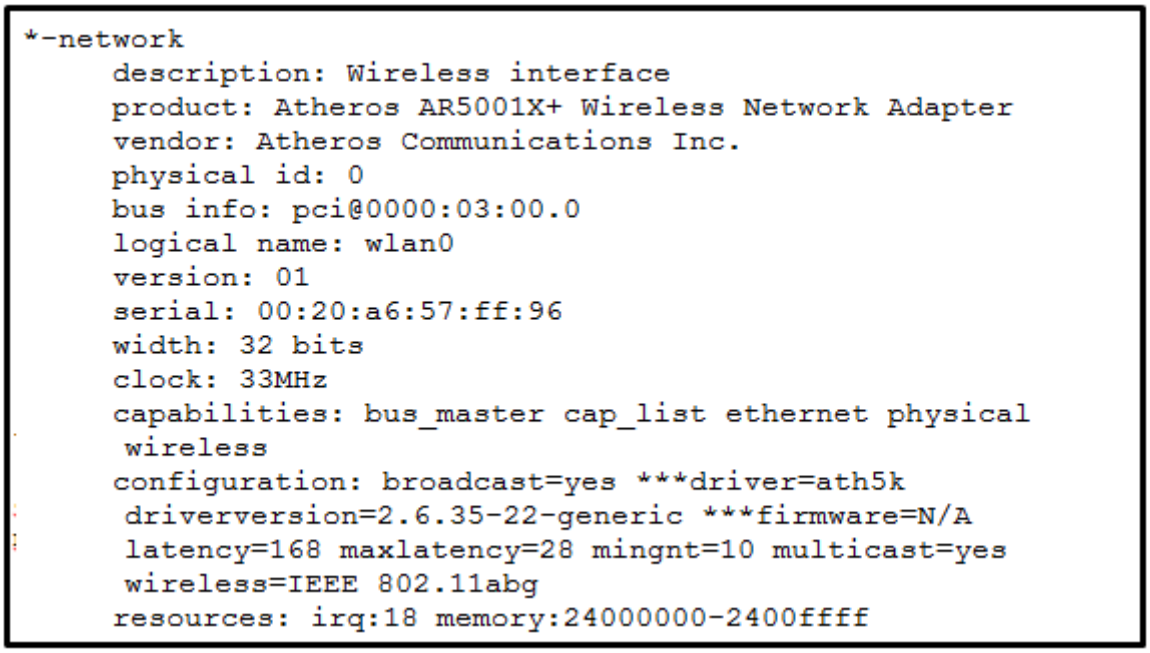

Fig. 7.1 Partial view of the command “lshw”

As can be noticed in the above figure the default driver that Ubuntu install for this card is the “ath5k”. This driver is fully working but presents some problems reporting noise level on its monitor mode. In order to know what driver is used on a computer the command “*lshw*” can be used, also the command “*lsmod*” shows the modules that are actually running on the system.

An important fact previous to the installation is to verify that your wireless card is supported by the MadWifi driver. This can be confirmed at http://madwifi-project.org/wiki/Compatibility.

7.1.1 Getting the driver

The latest MadWifi driver can be found at http://madwifi-project.org/. In this case is used the driver version 0.9.2.1 and can be downloaded from http://sourceforge.net/projects/madwifi/files/madwifi/0.9.2.1/madwifi-0.9.2.1.tar.gz/download.

Once the driver is downloaded is necessary to extract its content with the command:

tar -xvf madwifi-0.9.2.1.tar.gz

Then move to the MadWifi folder created with the previous step:

cd madwifi-0.9.2.1/

7.1.2 Installing the driver

Before the installation process is important to install some Linux features. First update the Ubuntu system through the update manager of the system.

The Linux headers of the current kernel also are requirements for MadWifi. To know the kernel version run the command *"uname -r".* In this case the version is 2.6.35-22-generic. To install the headers run the command below:

sudo apt-get install linux-headers-2.6.35-22-generic

Also is necessary to install some features of perl with the following command:

sudo apt-get install build-essential perl

With the headers installed the installation process can start, first building the MadWifi driver and then installing it. To achieve this run the commands below:

sudo make

sudo make install

7.1.3 Loading the MadWifi module

To load the module is used the command "*modprobe*". The commands below do two things remove the previous ath5k module and load the ath_pci module that is the MadWifi.

sudo modprobe -r ath5k

sudo modprobe ath_pci

Until now two interfaces were created one wlan0 and the ath0. The wlan0 has no functionality but represent the physical interface. The ath interface are the ones that can be used for different purposes.

To make prevalent the current configuration some changes are necessaries in the module file that is located at /etc/ in the Linux file system. At the end of this file add "ath_pci" and delete the previous driver (ath5k in the case of the tutorial). Also in the folder /etc/modprobe.d/ a black list can exist that will black list the MadWifi driver. If in this folder exist a file with the name "blacklist-ath_pci.conf" then it must be modified commenting the lines in it.

7.1.4 Creating VAP

The MadWifi driver works with VAP (Virtual AP). Using this feature can be possible to create from a physical interface several interfaces for different purposes. The tool to create these interfaces is "wlanconfig" and its utilization is explained below:

- To create an access point interface run the command: "*wlanconfig ath0 create wlandev wifi0 wlanmode ap".*

- To create an estation interface use the command: “*wlanconfig ath1 create wlandev wifi0 wlanmode sta nosbeacon*”.
- To create a monitor VAP use the command: “*wlanconfig ath2 create wlandev wifi0 wlanmode monitor*”.
- To destroy a VAP issue the command: “wlanconfig ath0 destroy”.

Notice that on each commands the ath’x’ is the logical name of the interface. This interface can be shown with the command “*iwconfig*”. In order to use the created interfaces the command “*ifconfig ath1 up*” is needed.

7.2 Prototype source code

```
package cascwn;
import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;
/**
 * The following software was developed as part of the Master Thesis "Context
aware self-configuring WIFI network""
 * @author Carlos A. Quiel Rodriguez
 */
class channel {
   int num_networks, num_packets,
data_size,data_packets,data_packets_history,packets_history;
   double max_noise,quality_value,actual_capacity;
channel(){
     num_networks=0;
     num_packets=0;
     data_size=0;
     data_packets=0;
     max_noise=-200;
     quality_value=0;
     actual_capacity=0;
     data_packets_history=0;
     packets_history=0;
   }

}

class wlan {
   String type,essid, bssid;
   int channel,num_clients,num_packets,data_size,data;
```

```
    double
last_snr,best_snr,worst_snr,last_signal_dbm,last_noise_dbm,min_noise_dbm,
max_signal_dbm,min_signal_dbm,max_noise_dbm;

wlan() {
    type=null;
    essid=null;
    bssid=null;
    best_snr=0;
    last_snr=0;
    channel=0;
    last_signal_dbm=0;
    last_noise_dbm=0;
    worst_snr=0;
    min_noise_dbm=0;
    max_signal_dbm=0;
    min_signal_dbm=0;
    max_noise_dbm=0;
    num_clients=0;
    num_packets=0;
    data_size=0;
    data=0;

}
}

public class CASCWN {
    public static int last_channel;

    public static double max_capacity;

    public static void delay(int time) { //delay in seconds
    try {
    time = time*1000;
    Thread.sleep(time);
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }

    }
    public static void change_channel (int channel) throws IOException{
       if (CASCWN.last_channel==channel) {
           System.out.printf("Is not necessary a channel change. Working on
channel: %d\n",channel);
       }
       else {
       CASCWN.last_channel=channel;
       System.out.printf("Changing the operative channel to: %d\n",channel);
```

```
      Process process = new ProcessBuilder("ip","link","set","ath0",
"down").start();
      InputStream is = process.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      BufferedReader br = new BufferedReader(isr);
      String line;
      while ((line = br.readLine()) != null) {
        System.out.println(line);
      }
      delay(1);
      String ch = String.valueOf(channel);
      Process process0 = new
ProcessBuilder("iwconfig","ath0","channel",ch).start();
      InputStream is0 = process0.getInputStream();
      InputStreamReader isr0 = new InputStreamReader(is0);
      BufferedReader br0 = new BufferedReader(isr0);
      String line0;
       while ((line0 = br0.readLine()) != null) {
        System.out.println(line0);
      }
      delay(1);
      Process process1 = new ProcessBuilder("ip","link","set","ath0", "up").start();
      InputStream is1 = process1.getInputStream();
      InputStreamReader isr1 = new InputStreamReader(is1);
      BufferedReader br1 = new BufferedReader(isr1);
      String line1;
      while ((line1 = br1.readLine()) != null) {
        System.out.println(line1);
      }

      }

   }



   public static void main(String[] args) throws IOException{

   int best_channel=0,loop_count=0,my_wlan_pos=-1;
   String my_wlan = "wlan-test";

   // Creation of the channel array
   channel[] Channels = new channel[14];
   for (int i=0; i<Channels.length; i++ ) {
   Channels[i] = new channel();
   }
   try {
       System.out.printf ("Sofware para el analisis del XML de salida del
Kismet\nAutor:Carlos Quiel\n\n");
       while (loop_count<30){
```

```
        System.out.println("\nIteracion " + (loop_count+1));

        DocumentBuilderFactory docBuilderFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder =
docBuilderFactory.newDocumentBuilder();
        docBuilder.setEntityResolver(new EntityResolver() {
        @Override
        public InputSource resolveEntity(String publicId, String systemId)
              throws SAXException, IOException {
           //System.out.println("Ignoring " + publicId + ", " + systemId);
           return new InputSource(new StringReader(""));
        }
        });

        Document doc = docBuilder.parse (new File("Kismet.netxml"));
        doc.getDocumentElement ().normalize ();
        NodeList listOfNetworks = doc.getElementsByTagName("wireless-
network");
        int totalNetworks = listOfNetworks.getLength();

        //Creation of the wlan array
        wlan[] Wlans = new wlan[totalNetworks];
        for (int i=0; i<Wlans.length; i++ ) {
        Wlans[i] = new wlan();
        }

        int mainch,mainpacket;
        //Begin the parse of the XML
        for(int s=0; s<listOfNetworks.getLength() ; s++){
           mainch=0;
           mainpacket=0;
           Node WlanNode = listOfNetworks.item(s);
           if(WlanNode.getNodeType() == Node.ELEMENT_NODE){
              Element FirstWlanElement = (Element)WlanNode;
              //Node type
              Wlans[s].type=FirstWlanElement.getAttribute("type");
              //Node SSID
              NodeList SsidList =
FirstWlanElement.getElementsByTagName("essid");
              Element SsidElement = (Element)SsidList.item(0);
              if(SsidElement!=null){//SSID not null validation
              NodeList text1EList = SsidElement.getChildNodes();
              Wlans[s].essid=text1EList.item(0).getNodeValue();
              if (Wlans[s].essid.equals(my_wlan)) {
              my_wlan_pos=s;}
              }
              //Node bssid
              NodeList BssidList =
FirstWlanElement.getElementsByTagName("BSSID");
```

```
                Element BssidElement = (Element)BssidList.item(0);
                NodeList text2EList = BssidElement.getChildNodes();
                Wlans[s].bssid=text2EList.item(0).getNodeValue();

                //Node channel
                NodeList ChannelList =
FirstWlanElement.getElementsByTagName("channel");
                Element ChannelElement = (Element)ChannelList.item(0);
                NodeList text3EList = ChannelElement.getChildNodes();

Wlans[s].channel=Integer.parseInt(text3EList.item(0).getNodeValue());
                // Node packets
                //total
                NodeList PacketList =
FirstWlanElement.getElementsByTagName("total");
                Element PacketsElement = (Element)PacketList.item(0);
                NodeList text4EList = PacketsElement.getChildNodes();

Wlans[s].num_packets=Integer.parseInt(text4EList.item(0).getNodeValue());

                NodeList PacketdataList =
FirstWlanElement.getElementsByTagName("data");
                Element PacketsdataElement = (Element)PacketdataList.item(0);
                NodeList text13EList = PacketsdataElement.getChildNodes();

Wlans[s].data=Integer.parseInt(text13EList.item(0).getNodeValue());

                // nodo datasize
                NodeList DataList =
FirstWlanElement.getElementsByTagName("datasize");
                Element DataElement = (Element)DataList.item(0);
                NodeList text11EList = DataElement.getChildNodes();

Wlans[s].data_size=Integer.parseInt(text11EList.item(0).getNodeValue());

                //---------------------------------------------------------
                // Nodo Frequencies
                NodeList listOfFrequencies =
FirstWlanElement.getElementsByTagName("freqmhz");
                int u=0;
                for(int t=0; t<Wlans[s].num_packets;){

                Element FreqElement = (Element)listOfFrequencies.item(u);
                NodeList text12EList = FreqElement.getChildNodes();

                //Separacion del String frequencia
                String[] Textfreq = text12EList.item(0).getNodeValue().split (" ");
                int freq, freq_packets;
                freq = Integer.parseInt(Textfreq[0]);
                freq_packets = Integer.parseInt(Textfreq[1]);
```

```
                t=t+freq_packets;
                if(freq_packets>mainpacket){
                mainch=freq;
                mainpacket=freq_packets;
                }

                //Packets per channel distribution
               if (my_wlan_pos!=s){
                switch (freq) {
                    case 2412:  Channels[0].num_packets += freq_packets;
break;
                    case 2417:  Channels[1].num_packets += freq_packets;
break;
                    case 2422:  Channels[2].num_packets += freq_packets;
break;
                    case 2427:  Channels[3].num_packets += freq_packets;
break;
                    case 2432:  Channels[4].num_packets += freq_packets;
break;
                    case 2437:  Channels[5].num_packets += freq_packets;
break;
                    case 2442:  Channels[6].num_packets += freq_packets;
break;
                    case 2447:  Channels[7].num_packets += freq_packets;
break;
                    case 2452:  Channels[8].num_packets += freq_packets;
break;
                    case 2457:  Channels[9].num_packets += freq_packets;
break;
                    case 2462:  Channels[10].num_packets += freq_packets;
break;
                    case 2467:  Channels[11].num_packets += freq_packets;
break;
                    case 2472:  Channels[12].num_packets += freq_packets;
break;
                    default: Channels[13].num_packets += freq_packets; break;
                }
               }
                u++;
                }

                //-------------------------------------------------------------
                 // Nodo Wireless-Client
                NodeList listOfClient =
FirstWlanElement.getElementsByTagName("wireless-client");
                Wlans[s].num_clients=listOfClient.getLength();
                //nodo last snr
                //Last Signal
                NodeList Last_signal_dbmList =
FirstWlanElement.getElementsByTagName("last_signal_dbm");
```

```
                Element Last_signal_dbmElement =
(Element)Last_signal_dbmList.item(0);
                NodeList text5EList = Last_signal_dbmElement.getChildNodes();

Wlans[s].last_signal_dbm=Double.parseDouble(text5EList.item(0).getNodeValu
e());
                // Last Noise
                NodeList Last_noise_dbmList =
FirstWlanElement.getElementsByTagName("last_noise_dbm");
                Element Last_noise_dbmElement =
(Element)Last_noise_dbmList.item(0);
                NodeList text6EList = Last_noise_dbmElement.getChildNodes();

Wlans[s].last_noise_dbm=Double.parseDouble(text6EList.item(0).getNodeValu
e());
                //calculo last SNR

Wlans[s].last_snr=10*Math.log10((10*Math.exp(Wlans[s].last_signal_dbm/10))/(
10*Math.exp(Wlans[s].last_noise_dbm/10)));

                //nodo Best_snr
                //Max Signal
                NodeList Max_signal_dbmList =
FirstWlanElement.getElementsByTagName("max_signal_dbm");
                Element Max_signal_dbmElement =
(Element)Max_signal_dbmList.item(0);
                NodeList text7EList = Max_signal_dbmElement.getChildNodes();

Wlans[s].max_signal_dbm=Double.parseDouble(text7EList.item(0).getNodeVal
ue());
                // Min Noise
                NodeList Min_noise_dbmList =
FirstWlanElement.getElementsByTagName("min_noise_dbm");
                Element Min_noise_dbmElement =
(Element)Min_noise_dbmList.item(0);
                NodeList text8EList = Min_noise_dbmElement.getChildNodes();

Wlans[s].min_noise_dbm=Double.parseDouble(text8EList.item(0).getNodeValu
e());
                //calculo last SNR

Wlans[s].best_snr=10*Math.log10((10*Math.exp(Wlans[s].max_signal_dbm/10))
/(10*Math.exp(Wlans[s].min_noise_dbm/10)));

                //nodo Worst_snr
                //Min Signal
                NodeList Min_signal_dbmList =
FirstWlanElement.getElementsByTagName("min_signal_dbm");
                Element Min_signal_dbmElement =
(Element)Min_signal_dbmList.item(0);
```

```
                NodeList text9EList = Min_signal_dbmElement.getChildNodes();

Wlans[s].min_signal_dbm=Double.parseDouble(text9EList.item(0).getNodeValu
e());
                // Max Noise
                NodeList Max_noise_dbmList =
FirstWlanElement.getElementsByTagName("max_noise_dbm");
                Element Max_noise_dbmElement =
(Element)Max_noise_dbmList.item(0);
                NodeList text10EList = Max_noise_dbmElement.getChildNodes();

Wlans[s].max_noise_dbm=Double.parseDouble(text10EList.item(0).getNodeVal
ue());
                //calculo last SNR

Wlans[s].worst_snr=10*Math.log10((10*Math.exp(Wlans[s].min_signal_dbm/10)
)/(10*Math.exp(Wlans[s].max_noise_dbm/10)));

                // Networks per channel, Bytes per channel, data packets per
channel calculation
                if (Wlans[s].channel==0){
                   switch (mainch) {
                   case 2412:
Channels[0].num_networks+=1;Channels[0].data_size+=Wlans[s].data_size;Ch
annels[0].data_packets+=Wlans[s].data;Wlans[s].channel=1;      break;
                   case 2417:
Channels[1].num_networks+=1;Channels[1].data_size+=Wlans[s].data_size;Ch
annels[1].data_packets+=Wlans[s].data;Wlans[s].channel=2;      break;
                   case 2422:
Channels[2].num_networks+=1;Channels[2].data_size+=Wlans[s].data_size;Ch
annels[2].data_packets+=Wlans[s].data;Wlans[s].channel=3;      break;
                   case 2427:
Channels[3].num_networks+=1;Channels[3].data_size+=Wlans[s].data_size;Ch
annels[3].data_packets+=Wlans[s].data;Wlans[s].channel=4;      break;
                   case 2432:
Channels[4].num_networks+=1;Channels[4].data_size+=Wlans[s].data_size;Ch
annels[4].data_packets+=Wlans[s].data;Wlans[s].channel=5;      break;
                   case 2437:
Channels[5].num_networks+=1;Channels[5].data_size+=Wlans[s].data_size;Ch
annels[5].data_packets+=Wlans[s].data;Wlans[s].channel=6;      break;
                   case 2442:
Channels[6].num_networks+=1;Channels[6].data_size+=Wlans[s].data_size;Ch
annels[6].data_packets+=Wlans[s].data;Wlans[s].channel=7;      break;
                   case 2447:
Channels[7].num_networks+=1;Channels[7].data_size+=Wlans[s].data_size;Ch
annels[7].data_packets+=Wlans[s].data;Wlans[s].channel=8;      break;
                   case 2452:
Channels[8].num_networks+=1;Channels[8].data_size+=Wlans[s].data_size;Ch
annels[8].data_packets+=Wlans[s].data;Wlans[s].channel=9;      break;
```

```
                case 2457:
Channels[9].num_networks+=1;Channels[9].data_size+=Wlans[s].data_size;Ch
annels[9].data_packets+=Wlans[s].data;Wlans[s].channel=10;       break;
                case 2462:
Channels[10].num_networks+=1;Channels[10].data_size+=Wlans[s].data_size;
Channels[10].data_packets+=Wlans[s].data;Wlans[s].channel=11;       break;
                case 2467:
Channels[11].num_networks+=1;Channels[11].data_size+=Wlans[s].data_size;
Channels[11].data_packets+=Wlans[s].data;Wlans[s].channel=12;       break;
                case 2472:
Channels[12].num_networks+=1;Channels[12].data_size+=Wlans[s].data_size;
Channels[12].data_packets+=Wlans[s].data;Wlans[s].channel=13;       break;
                default:
Channels[13].num_networks+=1;Channels[13].data_size+=Wlans[s].data_size;
Channels[13].data_packets+=Wlans[s].data;Wlans[s].channel=14;       break;
                }
              }else{
                Channels[(Wlans[s].channel-1)].num_networks+=1;
                Channels[(Wlans[s].channel-1)].data_size+=Wlans[s].data_size;
                Channels[Wlans[s].channel-1].data_packets+=Wlans[s].data;
              }

              //End of Networks, Bytes and data packets per Channel

            }//end of if
          }//end of XML parse

          //calulation of max noise per channel
          for (int t=0; t<Wlans.length; t++ ) {
            if (Channels[Wlans[t].channel-
1].max_noise<Wlans[t].max_noise_dbm){
              Channels[Wlans[t].channel-
1].max_noise=Wlans[t].max_noise_dbm;
             }
            }
          //Calculate non acumulate data packets
          for (int t=0; t<Channels.length; t++ ) {
            if ((t+1)==Wlans[my_wlan_pos].channel){
            Channels[t].data_packets-=Wlans[my_wlan_pos].data;

            Channels[t].data_size-=Wlans[my_wlan_pos].data_size;
            Channels[t].num_networks--;
            }}

          //----------ADAPTATIVE LOGIC-------------------------------------
          if (loop_count==0){
            for (int t=0; t<Channels.length; t++ ) {
            Channels[t].data_packets_history=Channels[t].data_packets;
            Channels[t].packets_history=Channels[t].num_packets;
            }
```

```
        }else {
            for (int t=0; t<Channels.length;) {
            int history_aux = Channels[t].data_packets_history;
            Channels[t].data_packets_history=Channels[t].data_packets;
            Channels[t].data_packets-=history_aux;

            int history_aux1 = Channels[t].packets_history;
            Channels[t].packets_history=Channels[t].num_packets;
            Channels[t].num_packets-=history_aux1;

            Channels[t].quality_value=((Channels[t].num_networks*-
0.83866)+(Channels[t].data_packets*-0.57617))/(-1.41483);
            Channels[t].actual_capacity=(-
0.306616*Channels[t].quality_value)+18.968691;
            t+=5;
            }
            best_channel=Wlans[my_wlan_pos].channel;
            for (int t=0; t<Channels.length;) {
            if (Channels[t].quality_value<Channels[best_channel-1].quality_value)
{
                best_channel=t+1;
            }
            t+=5;}
            if(loop_count==1)
            {
           if(best_channel!=Wlans[my_wlan_pos].channel)
           {

           change_channel(best_channel);
           }

            }
            else {
                if (((1-(Channels[Wlans[my_wlan_pos].channel-
1].actual_capacity/Channels[best_channel-1].actual_capacity))*100)>=10){

                        change_channel(best_channel);

                }
            }
            System.out.printf("Working channel %d Capacity %.3f Best channel
%d Capacity %.3f\n",
Wlans[my_wlan_pos].channel,Channels[Wlans[my_wlan_pos].channel-
1].actual_capacity,best_channel,Channels[best_channel-1].actual_capacity);
        }

        //PRINT INFO PER CHANNEL

        System.out.println("Information Per Channel ");
```

```
          System.out.printf("Channel;   Interfering WLAN;  Rogue Data Packets;
Quality Value;     Capcity\n");
          for (int t=0; t<Channels.length; ) {
          System.out.printf("   %d;            %d;              %d;              %.3f;
%.3f\n",(t+1),Channels[t].num_networks,Channels[t].data_packets,Channels[t].q
uality_value, Channels[t].actual_capacity);
            t+=5;   }
          //---------------------------------------------------------------


          loop_count++;

        for (int t=0;t<Channels.length; t++ )
        {
          Channels[t].num_networks=0;
        Channels[t].num_packets=0;
        Channels[t].data_size=0;
        Channels[t].data_packets=0;
        Channels[t].max_noise=-200;
        Channels[t].quality_value=0;
        Channels[t].actual_capacity=0;
        my_wlan_pos=-1;
        }

          //--------------------DELAY-----------------------------------
          delay(60);
          //---------------------------------------------------------------
          }//fin while
      }// fin try
      catch (SAXParseException err) {
      System.out.println ("** Parsing error" + ", line "
           + err.getLineNumber () + ", uri " + err.getSystemId ());
      System.out.println(" " + err.getMessage ());

      }
      catch (SAXException e) {
      Exception x = e.getException ();
      ((x == null) ? e : x).printStackTrace ();

      }
      catch (Throwable t) {
      t.printStackTrace ();
      }
   }//end of main
}
```