WLAN Watch: A Step Towards The Study Of 802.11b Wireless LANs.

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Imtiaz Ur Rahaman M.

to the
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

April, 2003

Certificate

This is to certify that the work contained in the thesis entitled "*WLAN Watch: A Step Towards The Study Of 802.11b Wireless LANs.*", by *Imtiaz Ur Rahaman M.*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

April, 2003

(Dr. Pravin Bhagwat)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

802.11b is a MAC and PHY layer standard for wireless LANs. These networks are widely used and can be easily spotted in various universities, airports, railway stations and coffee shops. When compared to wired networks, these networks provide a meager bandwidth of 11Mbps. There is a need to study these networks to suggest any possible performance enhancements and improvements at the MAC and Application layers. However, there are no adequate software tools available to carry out these experiments. The non availability of such software tools which can assist the study of the 802.11b networks has served as the motivation for this work.

The aim of this thesis work is to build tools that can assist the study of the 802.11b networks and lead to possible performance enhancements of these WLANs. The study of these WLANs requires tools which can provide the signal and noise level of each packet which in turn could be used, for example, to find the signal level range in which most of the packets over air get corrupted and the signal level below which most of the packets get lost. To assist the study of the 802.11b networks, the tools should also help in establishing a relationship among packet size, packet corruption ratio, throughput, power of transmitter, etc.

The tool built as part of the thesis, the *WLAN Watch*, provides all this information among other things. *WLAN Watch* is generic and currently supports Cisco Aironet 350 Series and Intersil Prism-II NICs.

Acknowledgements

It is an immense pleasure to express my sincere gratitude towards my supervisor Dr. Pravin Bhagwat for his immaculate guidance. It would have never been possible for me to take this project to completion without his innovative ideas and his support and encouragement. I consider myself extremely fortunate to have had a chance to work under his supervision. In spite of his hectic schedule he was always approachable and took his time off to attend to my problems and give the appropriate advice. It has been a very enlightening and enjoyable experience to work under him.

I wish to thank whole heartily all the faculty members of the Department of Computer Science and Engineering for the invaluable knowledge they have imparted to me. I also extend my thanks to the CSE department and the Media Labs Asia group for providing excellent facilities.

I would like to thank my family for taking me to this stage in life. It was their blessings which always gave me courage to face all the challenges and made my endeavors easier. The guidance of my brothers, which I have received for my entire life and which I wish to receive for the rest of my life, has been spot on.

Finally, I owe many thanks to my fun loving juniors, wonderful batchmates and caring seniors for making my stay at IITK memorable. The acknowledgements would be incomplete if I do not mention the name of the person who not only has helped me through out my stay here but has been of great help to many batches that have passed out and that will pass out during his stay here at IITK. Thanks a lot Atul sir.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Growth of wireless LAN has been phenomenal since its inception. WLANs have been around for some time now and apart from being used in various universities for various experiments, are widely being used to provide wireless connectivity in offices, homes and public places including restaurants, airports and railway stations. Most of the times, the standard used is the IEEE 802.11b (also known as Wi-Fi or Wireless Ethernet). So the same hardware can be used throughout these different environments. The IEEE has two more higher data rate versions, 802.11a and 802.11g, which will increase data rates to the point where wireless LANs can compete with their wired equivalents.

IEEE 802.11b is the wireless equivalent of ethernet. It operates in the 2.4GHz frequency band and uses High Rate - Direct Sequence Spread Spectrum (HR/DSSS) as the physical layer. The HR/DSSS provides a maximum bandwidth of 11Mbps [1]. Though the standard mentions the bandwidth of 802.11b cards to be 11Mbps, the physical layer overhead cuts down the throughput, meaning the real rate of 802.11b would be far less than 11Mbps [4].

Unlike wired networks, in general, wireless communications experience higher error rate and packet loss. This further lowers the throughput in wireless networks. The error rate, packet loss, throughput and signal strength of received frames in wireless networks are influenced by various parameters including transmission rate, distance between transmitter and receiver, size of frames transmitted, mobility of wireless stations and the presence of other stations communicating in overlapping frequency spectrum.

As the 802.11b standard is widely being used, we have planned to work with these WLANs.

1.1 Motivation

There is a need to monitor and study the behavior of these wireless networks under various scenarios with different parameters. These studies would lead to better understanding of these networks under various environments and hence would lead to possible performance enhancements. However, the study of the effects of the varying wireless communication parameters on the error rate, packet loss etc., is a wide and complex field that is still being explored. For example, this field comprises of study and analysis of the performance of wireless networks under varying channel conditions, study of the effect of mobility on signal strength behavior, etc.

However, there are no software tools available which can assist in these studies. For example, there are no existing tools using which the error rate and packet loss can be modelled as a function of distance between transmitter and receiver. There are no tools which can provide the signal level range in which most of the packets get corrupted and the signal level below which most of the packets get lost. For the same, a tool is required which can provide signal level information at the packet level. There are no tools which can provide the effect of various wireless communication parameters on error rate, packet loss, signal strength and throughput.

Existing sniffing tools for wireless traffic, like kismet [5], log only uncorrupted packets and fail to log corrupted packets as the drivers do not pass corrupted packets up to the application space. This prevents the user from learning about error patterns.

However, there are plenty of simple tools which give current signal strength, for example, the wireless tools [6]. These tools again do not provide per packet signal strength. There are tools available to measure throughput for various types of traffic under linux. Netperf [9] is one such benchmark for throughput measurement.

This thesis work focuses on building a tool that would assist people in studying the 802.11b WLANs in above mentioned ways and suggesting possible performance enhancements for these networks.

1.2 Organization of the Thesis

The remainder of this report is organized as follows. In chapter 2, we provide some background on network monitoring and compare wireless and wired networks with respect to network monitoring. We mention the features of the desired tool and the various experiments that can be conducted using it. We also introduce the RFMON mode of the wireless NICs.

To make the tool support more than one type of network card, the possibility of building a generic driver for various 802.11b NICs and the possibility of standardizing the driver-application interface has been explored. The issues involved therein are discussed in chapter 3. Chapter 4, describes the architecture and other aspects of WLAN Watch tool built during the thesis. Chapter 5, describes an experiment carried out using the WLAN Watch tool. Finally, we discuss the conclusions in chapter 6.

The appendix A documents the steps for setting up drivers and other tools required to use the WLAN Watch tool.

Chapter 2

Background

Network monitoring is the process of sniffing the packets over the communication medium. The sniffed packets can further be filtered based on any given criteria in order to analyze the network traffic. The filtering criteria is generally based on the fields of the IP header.

In wired networks, the sniffing is done by putting the card into promiscuous mode. In this mode, the network card passes all the packets on the physical medium, destined to any station, up to the protocol stack. These packets are then filtered based on the specified criteria and are written to the disk. Post sniffing analysis is performed on these logged packets to gather the required information. The network monitoring tools like tcpdump [10] and ethereal [18] are widely used for sniffing on the wired networks. These tools scale well and perform efficient sniffing even on networks with high bandwidths like 100Mbps LANs.

Such sniffing tools can also be used for wireless networks to log the packets over air. However, unlike in wired networks, the traffic in wireless networks is affected by several external factors and various wireless communication parameters. The external factors which can affect the wireless traffic include the availability of other stations communicating in interfering frequencies, distance between transmitter and sniffer, channel between sniffer and transmitter, location of sniffer and environmental noise. The various wireless communication parameters that can affect the traffic include power of transmitter, antenna used by both transmitter and sniffer, frequency of transmitter, signal strength at the sniffer and frequency of the sniffing network card.

Hence, the analysis of traffic on wireless networks is much more complex than the wired networks. In wireless networks, all the external factors and the network parameters need to be logged along with the packets sniffed over air, in order to study the behavior of the network. The existing network monitoring tools like tcpdump, ethereal, etc., which were mainly designed for wired networks, do not log the external factors and the communication parameters for wireless networks. Hence, to study the wireless networks we need much more sophisticated tools which can take the communication parameters and the external factors into consideration.

2.1 Desired Tool Features

To study the wireless network behavior, one would need a tool which can help in modelling the error rate, packet loss, signal strength, throughput, etc., as a function of transmission rate, distance between transmitter and receiver, size of frames, mobility, location of trace collector and frequency of communication. The tool should also provide the user with error patterns in the corrupted frames, per packet signal and noise level, options to vary the packet size, transmission rate and channel (frequency) of communication. There are many vendors manufacturing the 802.11b NICs and it is desirable that the tool supports more than one network card. As wireless stations include laptops and handheld devices with limited capabilities, the tool should also perform efficiently on such machines with limited capabilities. The tool should also be scalable to other higher rate versions of wireless LANs like 802.11a and 802.11g.

2.1.1 Support for Experimentation

The tool should be used for collecting enough traces at various locations in a given area under various circumstances. Those traces can further be analyzed to characterize the behavior of the 802.11b networks. The desired tool should provide enough support to

- Study the effect of co-channel interference on error rate, packet loss, throughput, signal and noise levels.
- Study the effect of transmission rate on error rate, packet loss and throughput.
- Study the effect of distance between transmitter and receiver on error rate, packet loss, throughput, signal and noise levels.
- Study the effect of packet size on error rate, packet loss and throughput.
- Study the effect of mobility on error rate, packet loss, throughput, signal and noise levels.
- Choose and implement error correction mechanisms at the application space. Analyze the overheads and benefits of having error correction implemented at the MAC layer.
- Plot iso-throughput lines in the area of coverage of a given access point.

Finally, the tool should also assist in setting up the network in an optimal way ensuring required throughput at various locations in the range of a given access point. The above mentioned experiments would provide an insight on the behavior of 802.11b networks. Those could in turn be used to suggest any possible performance enhancements for these networks.

We have built a generic tool, WLAN Watch, with most of the above features and it currently supports Cisco Aironet 350 Series [7] and Intersil Prism-II [8] network cards. An experiment has been conducted using our tool to demonstrate its usefulness and show how it could be used further.

2.2 RFMON mode

In wired networks, the promiscuous mode of the network card is used to sniff all the packets on the physical medium. The MAC, IP and transport layer headers of these sniffed packets are further analyzed to study the wired networks. The information contained in these sniffed packets is necessary and sufficient for monitoring the behavior of wired networks.

However, the wireless traffic has additional properties associated with it. Each wireless frame is associated with a certain signal level, noise level and frequency. This information is also necessary for monitoring the behavior of wireless networks with respect to various parameters. The promiscuous mode of wireless network cards can not be used to provide this information with each sniffed packet. Figure 2.1 depicts the format of the packets sniffed in normal or promiscuous mode.

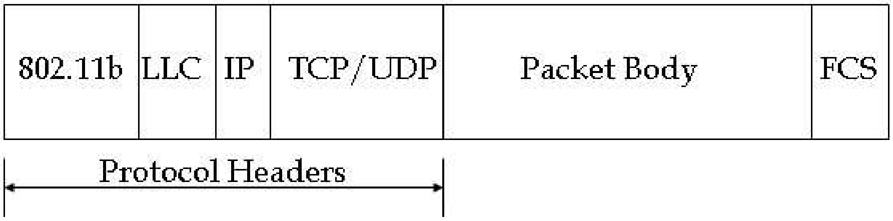


Figure 2.1: Frame format of packets in promiscuous mode

These packets would be passed to the user space through a raw socket interface. The additional information regarding the signal level, noise level, frequency, etc., can not be prepended to any incoming packet under promiscuous mode as doing so would disrupt the parsing of various headers by the layers of network protocol stack.

To enable the sniffing of packets and prepending of signal, noise and frequency information to each incoming frame, the wireless network cards have a special experimental mode of operation called RFMON mode. When in RFMON (Monitor) mode the network card passes up to the driver all the data frames along with all the management and control frames. The management and control frames are built and transmitted by the network card and are not passed to the driver in normal or promiscuous mode of operation. However, in the monitor mode, the network card does not transmit any frame and hence can not become a part of any Ad-Hoc or Managed wireless network. Hence the card can only be used for sniffing. This RFMON feature of the network card is used by the WLAN Watch tool in order to collect per packet information like signal and noise level.

All the frames sniffed in RFMON mode can be brought to the user space through a raw socket interface, in which the headers of the frames are not parsed by the different layers of the network protocol stack. Hence in this mode of operation, a special structure can be prepended to each incoming frame. This structure could include the information like signal level, noise level, frequency and rate at which the frame was transmitted.

The *wlan-ng* drivers for the Intersil Prism-II cards optionally prepend their own header (*prismheader*) to each of the sniffed frames. The *prismheader* consists of various pieces of information including *signal level*, *noise level*, *rate* and *channel*. Figure 2.2 shows the format of the packets sniffed through the Intersil Prism-II cards in RFMON mode, with the optional *prismheader* prepended.

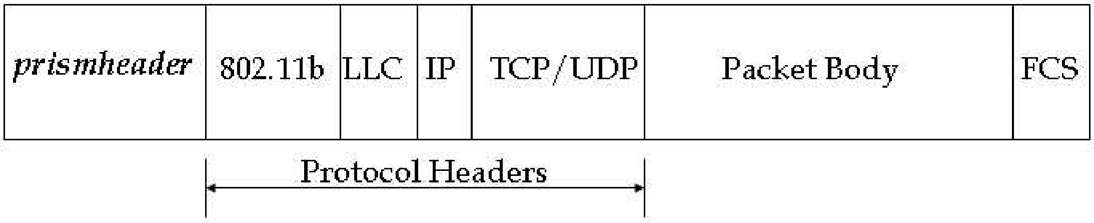
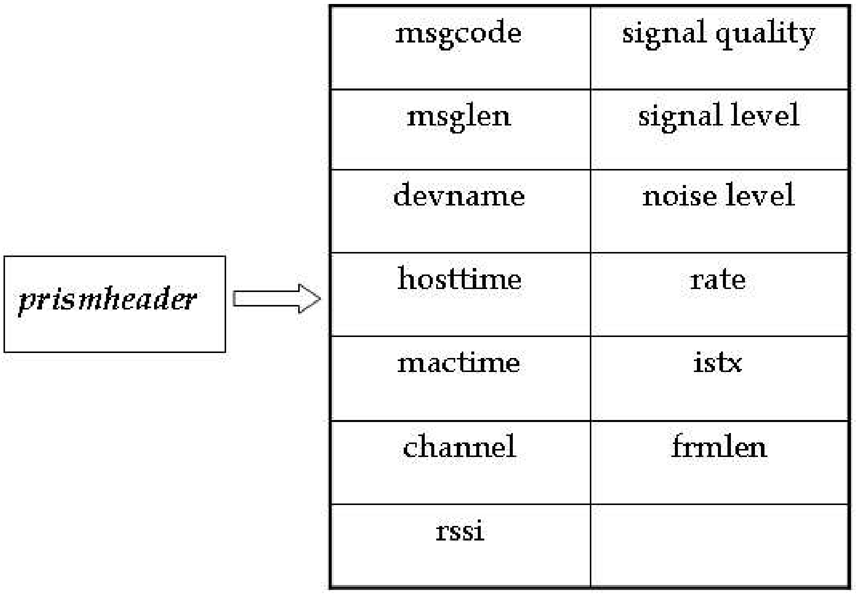


Figure 2.2: Frame format of packets in RFMON mode with prismheader prepended

Figure 2.3 shows the various fields of the *prismheader*. Among other things, it provides the signal level, noise level, rate and channel of the packet sniffed.



msgcode	signal quality
msglen	signal level
devname	noise level
hosttime	rate
mactime	istx
channel	frmlen
rssi	

Figure 2.3: Contents of the prismheader

Chapter 3

Design Considerations

802.11b network cards are being manufactured by several vendors like Cisco, Intersil, Atmel and TI. These chipsets have different firmwares and hence are subsequently seen as different hardware by the driver. It is desirable that our tool be generic and not be hardware or firmware dependent. Hence our tool should support more than one network card.

For making the tool generic, the possibilities of building a generic driver for various 802.11b NICs and standardizing the way the information is exported from the driver space to the application space have been explored. This chapter presents the networking stack architecture under linux before describing the above two ideas.

3.1 Networking Stack Architecture

The figure 3.1 depicts the various sub-systems involved in the architecture of any networking device under linux. The entire system can be divided into three broad categories - The Hardware, The Kernel Space and The User Space. At the Hardware level lies the network interface card. In our case it is the 802.11b NIC manufactured by several vendors like Intersil and Cisco. These NICs have set of instructions flashed onto the chips which help the driver communicate with the MAC chip. These instructions constitute the firmware. There are multiple interfaces using which the

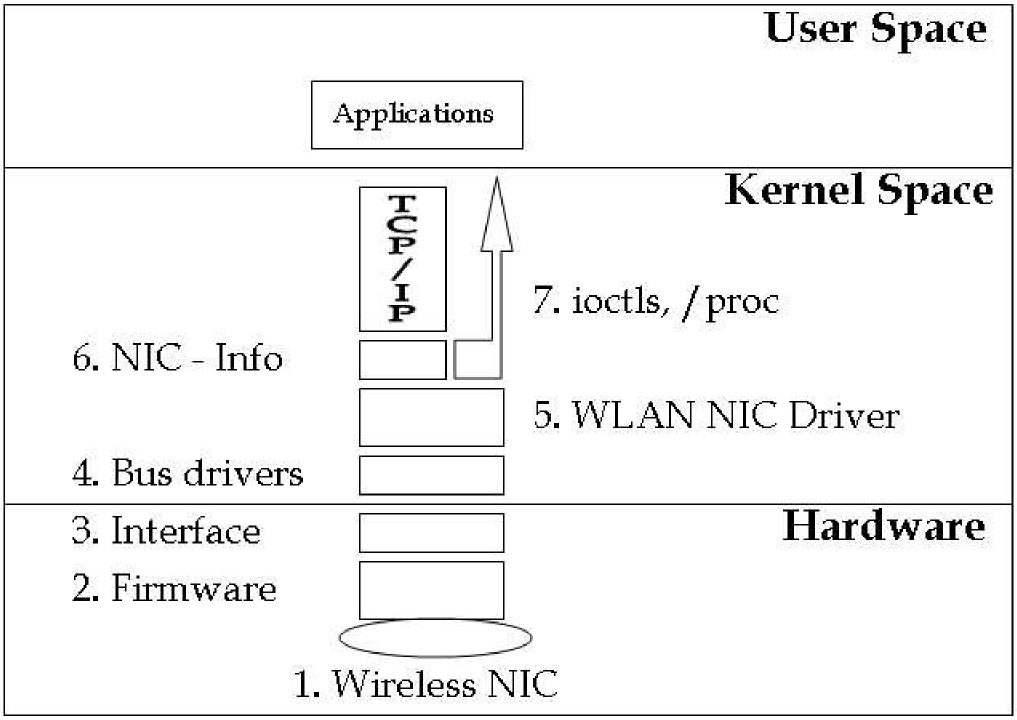


Figure 3.1: Networking Stack Architecture Under Linux

802.11b NICs can be interfaced with the computer system. The interfaces include PCMCIA, PCI, USB and PLX.

The driver of the networking device (WLAN NIC Driver) lies in the kernel space. It uses several kernel interfaces to register itself with low level drivers like the PCI, PCMCIA or USB drivers which also lie in the kernel space. The WLAN NIC driver is responsible for exchanging the packets between the upper layers and the NIC through standard kernel interfaces. The driver also gathers various pieces of information about the traffic dealt with by the NIC. This information is exported to the user space using standard interfaces like *ioctls* and */proc* file system.

User applications are developed at the application layer (The User Space).

3.2 Generic Device Driver

The idea of building a generic driver for various 802.11b NICs was explored. This section discusses the advantages and disadvantages of such a setup.

As RFMON mode is to be used for sniffing the packets, a generic driver can be built to support multiple cards like Cisco Aironet 350 series and Intersil Prism-II which support RFMON mode. In this mode, the generic driver can prepend its own header containing the signal, noise, rate and frequency information to each incoming packet. The generic driver can then log to the disk the packets sniffed and prepended with extra information. Such a setup of generic driver would be very efficient as the packets can be filtered and logged at the kernel level. An interface (*ioctl* or */proc*) is to be provided by the driver for allowing the user to specify the filtering criteria and the information that is to be prepended to each logged packet. For example, the user can specify to log only the data packets originating for a particular MAC. He can also specify to log only the signal and noise level information for each sniffed packet matching the criteria.

However, though this setup of generic driver is efficient it is associated with several issues.

3.2.1 Issues

Firstly, there are multiple vendors manufacturing the 802.11b MAC chips. Some of them are Intersil, Cisco, TI and Atmel. As of now, the Intersil and Cisco NICs have good support under linux. Secondly, the Intersil chips have several resellers who flash different firmwares onto the chips and sell under different labels. For example, Lucent, Intersil and Symbol NICs are known to have same underlying MAC chips but all have different firmwares. Thirdly, these NICs interface with the computer systems in various ways like PCMCIA, USB, PCI and PLX.

All these above issues make the development of a generic driver difficult. Also it is hard to keep pace with the changing firmware versions for a wide range of network cards. Moreover, it has been found that the drivers which are specifically meant for a particular vendor's NIC are more stable, well maintained and hence perform better. Such drivers take care of the changing firmware and provide better support for those particular NICs. For example, the *wlan-ng* driver from Absolute Value Systems (AVS) [12] for the Intersil chips is more stable than the *Orinoco* driver which claims to be generic. The *Orinoco* driver claims to support Lucent, Intersil Prism-II and Symbol cards but it is found that it is not fully functional with some Prism-II cards. The recent versions of *Orinoco* driver do not seem to handle the Symbol cards properly and there is also no support for USB. On the other hand, the *wlan-ng* and *HostAP* drivers for Prism-II cards are more tested and have more features.

Hence, it was observed that though the idea (of building a generic driver and making it log the sniffed packets matching user specified criteria) looks to be efficient, it has other stability and maintenance issues associated.

3.3 Standard Driver-Application Interface

Another possibility of making the tool generic is to patch several drivers to provide a standard interface to user applications. Hence such an architecture of standard driver-application interface would export sniffed packets matching user criteria and prepended with an extra piece of information through a standard interface (*ioctl* or */proc*). The existing drivers for Cisco Aironet 350 series and Intersil Prism-II, for example, provide different *ioctls* and */proc* file system entries for getting or setting card parameters like dropping the card into RFMON mode. These drivers also have different interfaces for exporting information about the traffic to applications. These drivers could be patched to have a common interface for configuring the cards and for providing the desired information about the traffic like the signal and noise level of each packet. These drivers could also provide a common interface to the user for specifying the sniffing criteria.

An improvement over the above scheme would be to have the sniffed packets logged at the kernel level. The standard interface can be provided only for configuring the card, specifying sniffing criteria and specifying the extra information that needs to be prepended to each sniffed packet before logging to the disk. But both these schemes would require good amount of modification of existing drivers.

The advantages and disadvantages of having a common interface have been discovered by closely studying the work done by Jean Tourrilhes. Wireless Extensions [13, 6] by Jean Tourrilhes is an effort in the direction of standardizing the driver-application interface for providing uniform ways for configuring a wireless network device.

Wireless Extensions and the issues related with such tools are discussed in the following subsections.

3.3.1 Wireless Extensions

The idea behind building the Wireless Extensions was to be able to manipulate any wireless networking device in a standard and uniform way. They provide the following interfaces for various pieces of information:

/proc/net/wireless

Designed to give some wireless specific statistics on each wireless interface in the system. This entry is in fact a clone of */proc/net/dev* which gives the standard driver statistics. For each device, the following information is provided:

1. ***Status* -** Its current state. This is a device dependent information.
2. ***Quality link* -** General quality of the reception.
3. ***Quality level* -** Signal strength at the receiver.
4. ***Quality noise* -** Silence level at the receiver.

5. ***Discarded nwid* -** Number of discarded packets due to invalid network id.
6. ***Discarded crypt* -** Number of packets unable to decrypt.
7. ***Discarded misc* -** Unused.

iwconfig

This tool is designed to configure the wireless specific parameters of the driver and the hardware. This is a clone of ifconfig used for standard device configuration. The following parameters are available:

1. ***freq or channel* -** The frequency or the channel sequence.
2. ***nwid* -** Network id or domain, to distinguish different logical networks.
3. ***protocol* -** The name of the *protocol* used on the air.
4. ***sens* -** This is the signal level threshold to trigger packet reception (sensitivity).
5. ***enc* -** The encryption or scrambling key used.

iwspy

Designed to test the Mobile IP support. Gathers quality information for a set of network addresses.

iwpriv

Device specific.

Drivers that support these Wireless Extensions include *Wvlan_cs*, *Orinoco*, *Wlan-ng*, *HostAP* and *Airo*. The following subsection discusses the issues involved in trying to standardize the information and the way it is exported from the driver to the applications.

The Wireless Extensions, however do not provide per-packet signal level and noise-level information. These tools only provide information about the wireless interfaces that are associated with some access point. They fail to work when the association of the wireless NIC is lost with the access point. Hence, these tools do not provide any information when the NICs are dropped into RFMON mode. Under RFMON mode, the NICs do not transmit any frame and hence can not be associated with an access point.

3.3.2 Issues

Firstly, the drivers of few NICs need to be modified in order to add the */proc* entries and appropriate *ioctls*. Secondly, the standardized interface provided by modifying various drivers would not be able to provide the robust functionalities as provided by the tool kits of various drivers. The drivers come with their own tool kits which allow the users to configure the NICs comfortably and to get and set plenty of parameters. For example, the *wlan-ng* driver provides the *wlanctl-ng* and *wlancfg* tools to read and write large number of parameters. The number of parameters that can be read or written are far more than what the Wireless Extensions provide. The Wireless Extensions, however, are just meant to configure the various cards in a standard way. Those do not give all the options that the tool kits like *wlanctl-ng* provide. Finally, support for Wireless Extensions is not available in newer drivers coming into the market. Newer drivers do not tend to implement the suggested standard interface and build their own tool kits for configuring the network cards and exporting the information to user space.

All these issues do not encourage us to modify the existing drivers to have a common interface for communicating with the driver. Hence a better design would be the one which does minimum possible modifications to the existing drivers.

The WLAN Watch tool is built on the existing tool kits provided by the *airo* and *wlan-ng* drivers. However, the *wlan-ng* driver has been modified to pass the corrupted packets up to the applications in RFMON mode of the card. This driver prepends an optional header to each incoming packet. This optional header contains signal level, noise level, rate and channel information among other things. The sniffed packets can be brought to the user space by opening raw sockets. In the existing version of the WLAN Watch tool, the *airo* driver has not been modified to pass the corrupted packets up to applications in RFMON mode. Also, the existing *airo* drivers do not prepend an optional information header to each incoming packet.

3.4 *wlan-ng* Driver Modifications

We have worked with the 0.1.16-pre8 version of the wlan-ng driver for Intersil Prism-II chipsets. The wlan-ng driver consists of two modules `p80211` and `prism2_*`. The `prism2_*` would be one of `prism2_cs`, `prism2_pci`, `prism2_usb` and `prism2_plx` depending on the type of network card adapter.

The `p80211` module contains code for handling the 802.11b services. This module interacts with the network layer of the protocol stack. It implements the code for configuring the network card. It contains the code for various *ioctls*. It passes up the packets received from the `prism2_*` module to the network layer of the protocol stack. Similarly, it receives the packets to be transmitted from the kernel and passes them down to the `prism2_*` module.

The `prism2_*` module implements the code for the Intersil Prism-II MAC specifications. It registers itself with the low level drivers like PCI, PCMCIA and USB drivers. It handles the interrupts from the network card. When ever any event like packet reception, packet transmission etc., occurs, the card raises an interrupt which is handled by the code in this module. For the event of packet reception, this module (interrupt handler) allocates a socket buffer and reads the packet from the card into this buffer. The pointer to this buffer is then given to the other module, `p80211`. The `p80211` module then passes it to the upper layers.

When ever an uncorrupted packet is received in the RFMON mode, the `prism2_*` module passes this packet to the upper layer by appending a four bytes of dummy FCS, with each byte being 0xFF. However, all corrupted packets received in this RFMON mode are dropped by the module. We have modified the driver at this point and have made it to pass even the corrupted packets up to the upper layers by appending a four bytes of dummy FCS, with each byte being 0x12. The WLAN Watch tool looks at this FCS at the application layer to find out if the packet is corrupted.

The functions of the driver which have been modified are `void hfa384x_int_rx()` and `void hfa384x_int_rxmonitor()`.

Chapter 4

WLAN Watch

The WLAN Watch tool is built on top of the existing drivers and the tool kits they provide. The minor modifications made to the *wlan-ng* driver were described in the previous chapter. This chapter discusses the various aspects of the WLAN Watch tool.

4.1 Overview

The figure 4.1 depicts the generic tool built on top of the existing drivers and their tool kits. The user space library below the generic tool provides a standard interface to the tool and internally uses the tool kits provided by the drivers to communicate with them.

The WLAN Watch is an user space tool. It supports Intersil Prism-II and Cisco Aironet 350 Series NICs. It requires *wlan-ng* driver for Prism-II cards and *airo* driver for the Aironet cards. As the *airo* driver has not been modified yet to pass up the corrupted packets, to conduct experiments which require to map the corrupted packets to their original ones, the *wlan-ng* driver (Prism-II NIC) is to be used. Also the current implementation of WLAN Watch does not provide per packet signal and noise level information for *airo* driver (Aironet 350 Series NICs). To conduct the experiments with WLAN Watch, an access point is required to which a station can

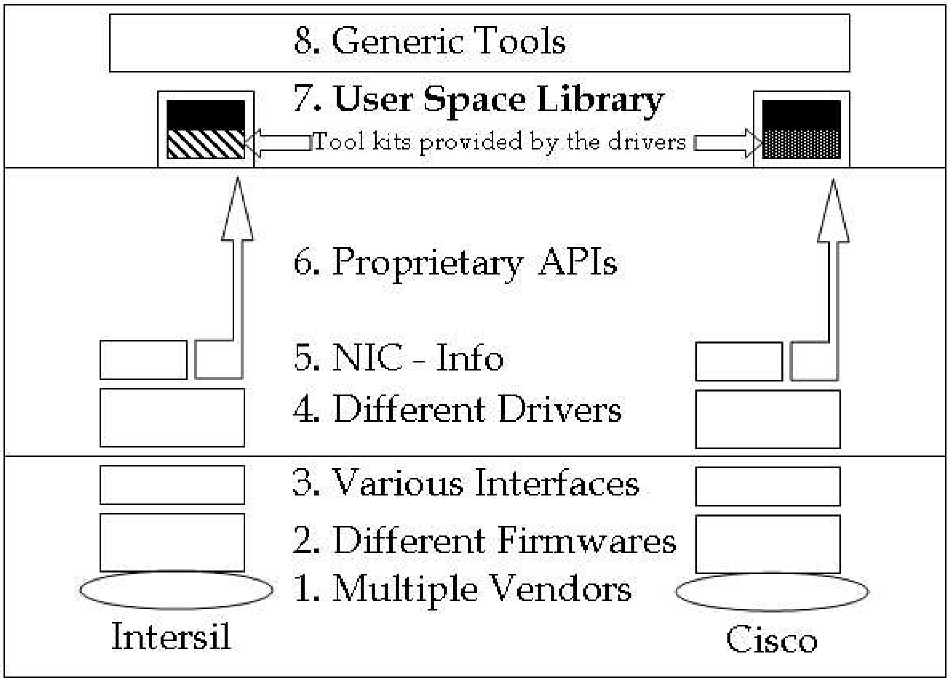


Figure 4.1: Generic User Space Tools

associate in infrastructure mode with WEP disabled. And finally, one requires root privileges to use the WLAN Watch tool.

4.2 Architectural Requirements

The idea behind the tool to be built is to be able to generate some traffic and sniff it at various locations in the area of coverage of the transmitter. For sniffing the wireless traffic, the sniffing machine (sniffer) is required to use the RFMON mode. But as the sniffer can not be a part of any network when in RFMON mode, one more interface is required on the sniffer through which the sniffer can communicate with the transmitter. This communication between the sniffer and the transmitter is required, firstly for the transmitter to control the sniffer and secondly for the sniffer to upload the sniffed packets.

The transmitter should also be able to specify sniffing criteria to the sniffer, like, the channel to sniff on and the MAC address whose packets need to be sniffed. The transmitter also should be able to reset the sniffer for sniffing, stop the sniffing after generating enough traffic and finally ask the sniffer to upload the sniffed log back to the transmitter. It should also be possible for moving the sniffer around to install it at various places to monitor the behavior of network at several places in the area of coverage of the transmitter.

4.3 Architecture of WLAN Watch

Due to the above mentioned requirements of the tool, the architecture of our tool - WLAN Watch, consists of a transmitter which has one wireless interface through which it can generate wireless traffic. Also, as traffic can only be generated when it is a part of a network, we have used an access point to which the transmitter could be connected in infrastructure mode. The access point and the sniffer are further connected to ethernet backbone which allows the sniffer to communicate with the transmitter. Hence our sniffer has two interfaces, one wireless and the other wired.

A TCP connection is established between the transmitter and the sniffer through the access point. The transmitter connects to the sniffer and resets the sniffer for sniffing. The transmitter then generates user specified traffic and also logs it to a local file, *TxLog*. This is being sniffed by the sniffer and the packets are logged to the *RxLog* file. After generating enough traffic, the transmitter asks the sniffer to stop sniffing and upload the log of the sniffed packets back. The positions of sniffer and transmitter can then be changed depending on the demands of the experiments. At the end of each session of transmission, sniffing and uploading of the sniffed log, the transmitter processes the *TxLog* and *RxLog* to display the statistics including the number of packets dropped at the sniffer, the number of packets duplicated, the number of packets corrupted and the signal and noise level information at the packet level.

Our tool is built on a client-server model in which the sniffer runs a server which accepts the connections from the clients (transmitters). This setup provides the flexibility of having any number of transmitters conduct the experiments. However, at any given instance of time, the server (sniffer) can be serving only one client (transmitter). Hence, the software of our WLAN Watch tool has two parts - the WLAN Watch Server and the WLAN Watch Client. The WLAN Watch Server is the part of the tool that is installed on the sniffer and the WLAN Watch Client is that part which is installed on the transmitter. These two parts are described in the sub sections to follow. The figure 4.2 depicts the WLAN Watch setup.

We have conducted an experiment using our tool. In this we have monitored the traffic at the access point. This experiment is detailed in the next chapter. To monitor the behavior of the traffic at the access point, the machine hosting WLAN Watch server is placed very close to the access point being used in the experiment. It

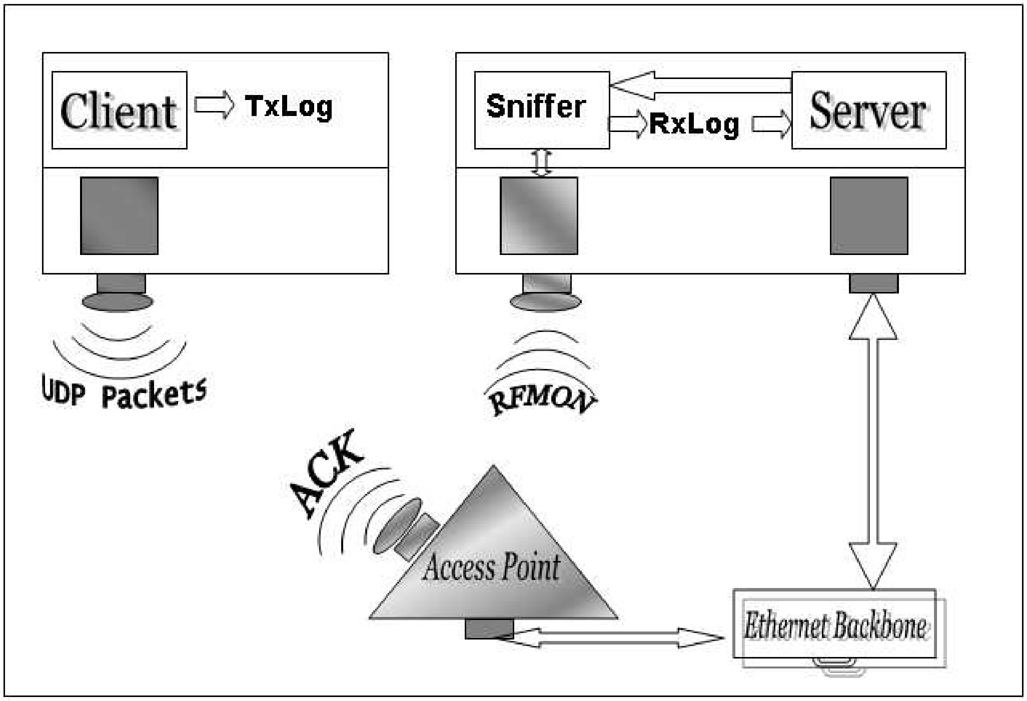


Figure 4.2: WLAN Watch Architecture

sniffs all the data packets originating from the WLAN Watch client's wireless interface that are being sent to the access point. As the NIC of the WLAN Watch client is associated to the access point under infrastructure mode, all the packets originating from the WLAN Watch client MAC are addressed to the access point at the MAC level. The WLAN Watch server, placed very close to the access point, tries to sniff these packets through its wireless interface dropped in RFMON mode as shown in the figure 4.2. The data packets being dumped by the WLAN Watch client and the data packets being sniffed by the WLAN Watch server are both logged. Hence this entire WLAN Watch setup provides a close picture of the actual wireless network behavior at the destination, the access point in this case.

4.3.1 WLAN Watch - Server

The station hosting the server has two network interfaces. One is the wireless interface that is dropped into the RFMON mode to sniff the packets on air. The other one is a wired interface through which the server and client communicate over a TCP connection. The task of the server is to receive the commands `START SNIFFING`, `STOP SNIFFING` and `IMPORT RxLOG` from client over the TCP connection and act accordingly.

The `START SNIFFING` command has two parameters. First parameter is the *channel number* on which the client would like the server to sniff the packets on air by dropping its NIC into RFMON mode and the second parameter is the *MAC address* of the NIC that puts the traffic on air, essentially the client's MAC address. Management and control frames are not logged on the sender side (client). This is due to the reason that the management and control frames are actually transmitted by the NIC without the involvement of the driver. Hence, though all the management, control and data frames are sniffed by the server, only the data packets are logged to the file *RxLog*. After sending this command, the client starts dumping UDP packets onto the air and also logs them to a local file, *TxLog*. These UDP packets are acknowledged at the MAC level by the access point to which the client's NIC is associated as shown in the figure 4.2. However, it should be noted that duplicate packets are logged at the server either due to the loss of ACKs from the access point to the client or the loss of frames at the access point. The packets lost at the server might actually not have got lost at the access point and similarly the packets corrupted at the server might not have got corrupted at the access point.

After dumping specified UDP packets onto the air, the client sends the command `STOP SNIFFING` to make the server stop sniffing on its wireless NIC. The *RxLog* is imported back to the client by sending the command `IMPORT RxLOG`. The client then compares the files *RxLog* and *TxLog* to present the various statistics. These files can further be processed to extract per packet signal and noise level.

4.3.2 WLAN Watch - Client

This is the part of the WLAN Watch tool that is installed on the transmitter. It forms UDP packets of user's interest and dumps them onto the air. The machine hosting client needs to have just one network (wireless) interface. The commands the client sends to the server were detailed in the above section.

UDP traffic is generated for conducting the experiments. A sequence number is embedded in the payload of each packet generated. For being able to map the corrupted packets to the original ones, this sequence number is repeated at several places in the payload of the packet. The various parameters associated with the UDP packets (traffic) that the user can modify include the destination IP address, source port, destination port, starting value of the sequence number, the size of each instance of the sequence number, the redundancy of the sequence number, the packet payload size, the number of packets to be generated and the contents of the packet payload.

Finally, at the end of each session, the client scans the traffic parameters specified by the user and the *RxLog* file to display the various statistics as depicted in the table 4.1.

```
TRAFFIC PARAMETERS: SPECIFIED TRAFFIC COUNT          :  3000
TRAFFIC PARAMETERS: UDP PACKET BODY (IN BYTES)       :  1000

RxLog: TOTAL PACKETS SNIFFED                         :  2903
RxLog: TOTAL CORRUPTED PACKETS                       :  0
RxLog: DUPLICATE PACKET : 502
RxLog: DUPLICATE PACKET : 1130
RxLog: DUPLICATE PACKET : 1144
RxLog: DUPLICATE PACKET : 1162
RxLog: DUPLICATE PACKET : 1629
RxLog: DUPLICATE PACKET : 1722
RxLog: TOTAL DUPLICATES FOUND                        :  6
RxLog: TOTAL PACKETS LOST                            :  106

RxLog: ERROR RATE - PERCENTAGE                       :  0.00
RxLog: PACKET LOSS - PERCENTAGE                      :  3.53
RxLog: THROUGHPUT                                    :  1355.517 Kbps
```

Table 4.1: Statistics

4.4 Performance

WLAN Watch is built for 802.11b wireless LANs. The bandwidth of these LANs is 11Mbps. The performance of our tool mainly depends on the performance of the WLAN Watch server (sniffer) as it has to sniff, filter and log the packets in real time. Hence, for measuring the performance of our tool, only the sniffer has been considered.

The performance of our tool has been very efficient when the WLAN Watch server (sniffer) was installed on a fast laptop with a Pentium-III processor, 256MB RAM and a fast disk. However, the performance of the sniffer has suffered a bit when the WLAN Watch server was installed on a slower desktop with a Pentium-II processor, 192MB RAM and a slower disk. On such a desktop, the sniffer was not able to sniff, filter and log packets fast enough to keep pace with the transmitter. It would be interesting to note the performance of our tool with the higher data rate versions of the IEEE such as the 802.11a networks. These networks provide a data rate of upto 54Mbps. Lastly, as the sniffed packets are brought to the user space for filtering and logging, the performance of our sniffer would not be as good as the setup in which the driver does the filtering and logging at the kernel space.

Chapter 5

WLAN Watch: Experimentation

An experiment has been conducted in the CSE department of IIT Kanpur, using the WLAN Watch tool, to demonstrate the usefulness of the tool. This experiment shows how WLAN Watch helps in deriving a relationship among the signal level, distance between transmitter and sniffer, error rate and packet loss.

This experiment is meant to demonstrate how the WLAN Watch tool opens up doors for further experimentations and study of 802.11b wireless LANs. This chapter details the scenarios in which the experiment was conducted and the graphs that were plotted. Further study has to be done on these graphs and several such traces need to be collected using our tool to model the error rate and signal level with respect to the distance between the transmitter and the sniffer.

5.1 Site Of Experimentation

The figure 5.1 shows the site where the experiment was conducted. The triangle in the figure depicts the access point (AP), the square just beside the triangle depicts the machine hosting WLAN Watch server (sniffer) and the circles stand for the machine hosting WLAN Watch client (transmitter). The dashed lines indicate the glass windows. The solid lines stand for the walls and doors. The AP and the server were placed in the room CS316. The client was moved around and samples were taken at several places listed below:

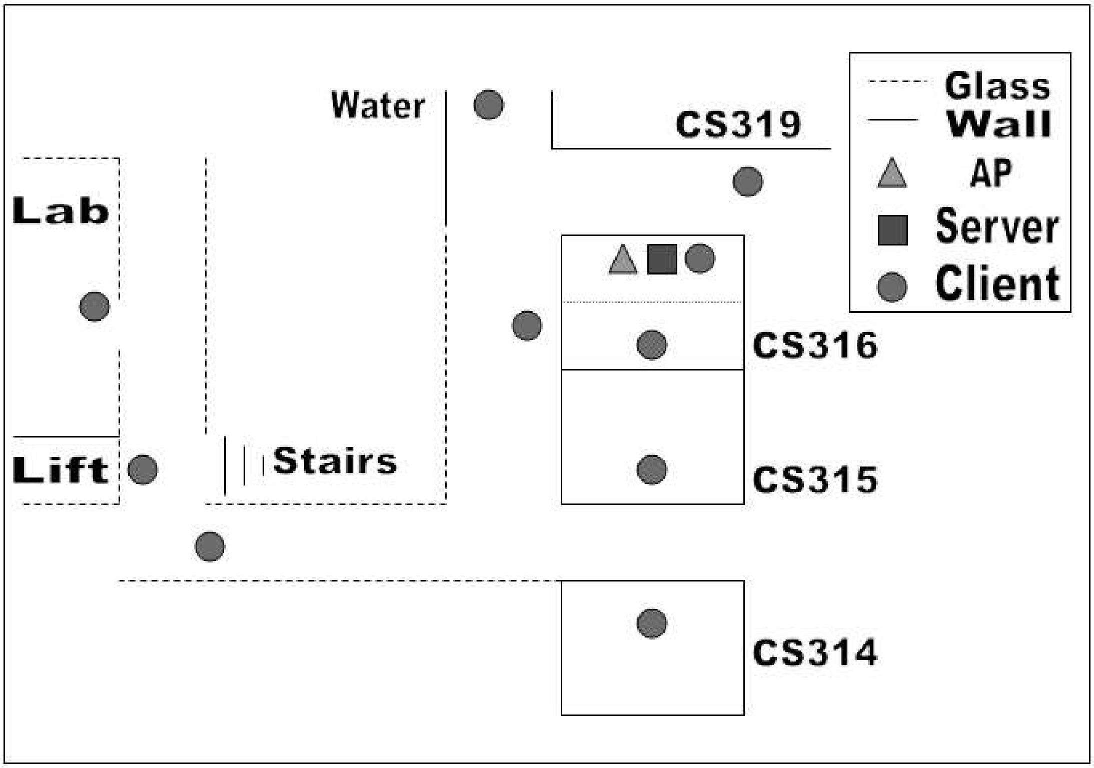


Figure 5.1: Site Of Experimentation

1. Just beside the AP and the server.
2. Inside CS316, five meters away from the AP.
3. Just outside CS316.
4. Near the water.
5. Outside CS319.
6. Inside CS315.
7. Inside CS314.

8. Near the stairs.

9. Outside the lift.

10. At the entrance of the lab.

5.2 Experimentation Details

Traces have been collected at various points as shown in the figure 5.1. The following information from the *RxLog* has been used for this experiment:

1. Per packet signal level.

2. Count of packets sniffed by the server.

3. Count of corrupted packets sniffed.

4. Count of uncorrupted packets sniffed.

Using the above information extracted from the *RxLog*, the two tables 5.1 and 5.2 (one for corrupted and the other one for uncorrupted packets) were constructed. Each graph plotted has two curves, one for corrupted packets and the other for uncorrupted packets.

A signal level is prepended by the driver to each incoming packet. This signal level is read from the network card. However, the actual units of the signal level depends on the firmware of the card. The mapping between the signal level value for the card (firmware) used and dBm scale was not available when this report was written. However, a mapping can be established between the signal level values provided by the firmware and the dBm scale using other tools which provide the signal on dBm scale. The signal level provided by the firmwares are positive integers which lie in the range 27-154 for Intersil Prism-II chipsets [14]. However, the firmware of the card used in this experiment has given signal level values beyond and below the range 27-154.

Signal Level Interval	**% Of Traffic**
10-14	0.14
15-19	0.00
20-24	0.14
25-29	0.14
30-34	0.42
35-39	0.28
40-44	0.42
45-49	1.25
50-54	1.11
55-59	0.28
60-64	0.56
65-69	0.14
70-74	0.00
75-79	0.00
80-84	0.00
85-89	0.00
90-94	0.00
95-99	0.00

Table 5.1: Corrupted Packets: Signal Level Interval Vs % Of Traffic

Signal Level Interval	**% Of Traffic**
10-14	0.00
15-19	0.14
20-24	0.00
25-29	0.00
30-34	0.00
35-39	0.97
40-44	2.51
45-49	10.31
50-54	19.78
55-59	16.02
60-64	23.54
65-69	14.35
70-74	3.90
75-79	2.37
80-84	0.42
85-89	0.42
90-94	0.28
95-99	0.14

Table 5.2: Uncorrupted Packets: Signal Level Interval Vs % Of Traffic

In our experiment, the entire signal level range was divided into smaller intervals of width 5. In the table 5.1 for corrupted packets, the first column shows the intervals of width 5. The second column shows what percentage of corrupted packets were found in the corresponding interval. Similar information is tabulated for uncorrupted packets in table 5.2.

Finally, the graphs were drawn with the (mean of) signal level intervals on X-axis and the percentage of traffic on Y-axis. Note that the abscissa of each point in the graphs is the mean of the signal level interval to which it belongs to. Hence, for example, if the percentage of corrupted packets in the signal level interval 15-19 is 6.23, then the graph plots a corresponding point at the coordinates (17, 6.23) as 17 is the mean of the interval 15-19.

5.3 Graphs

This section depicts the graphs plotted for all the traces collected at various places in the site.

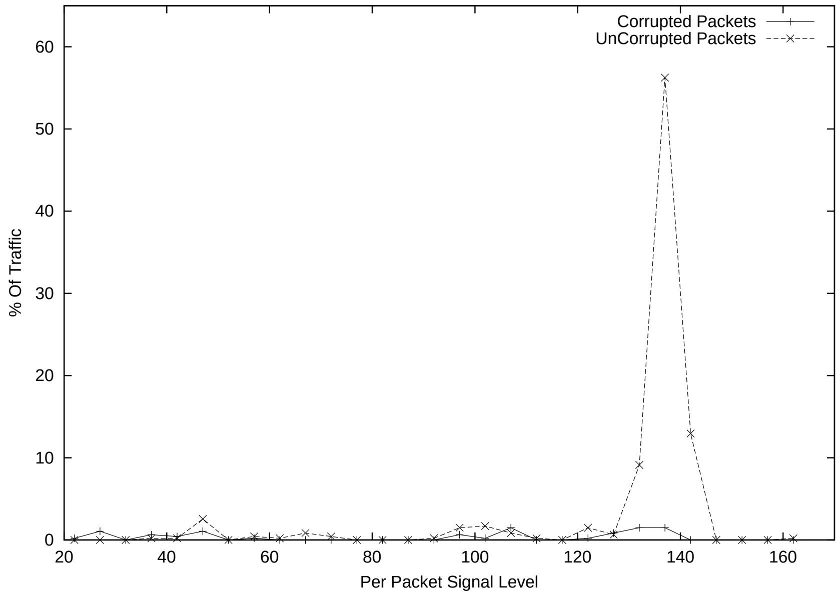


Figure 5.2: Just Beside The Access Point

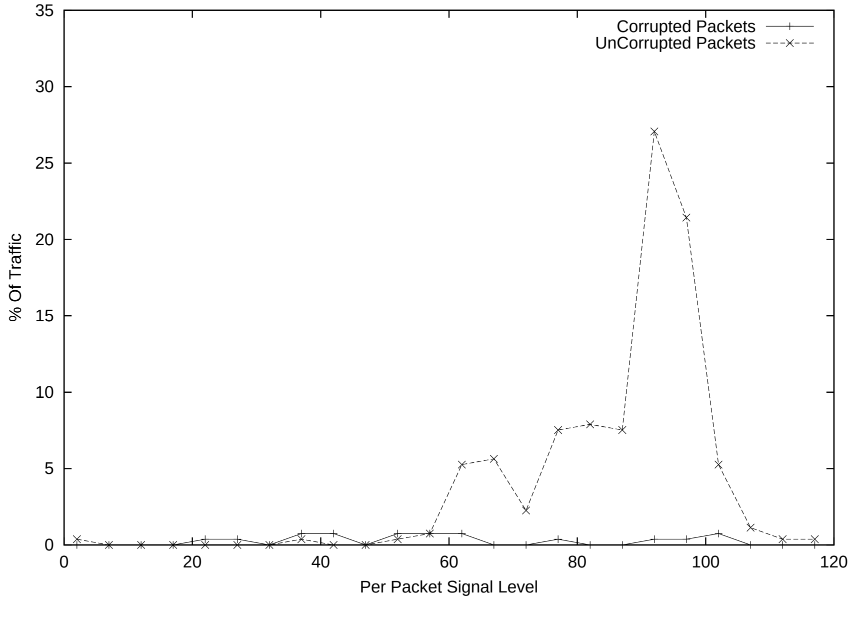


Figure 5.3: CS316 - 5 meters from the AP

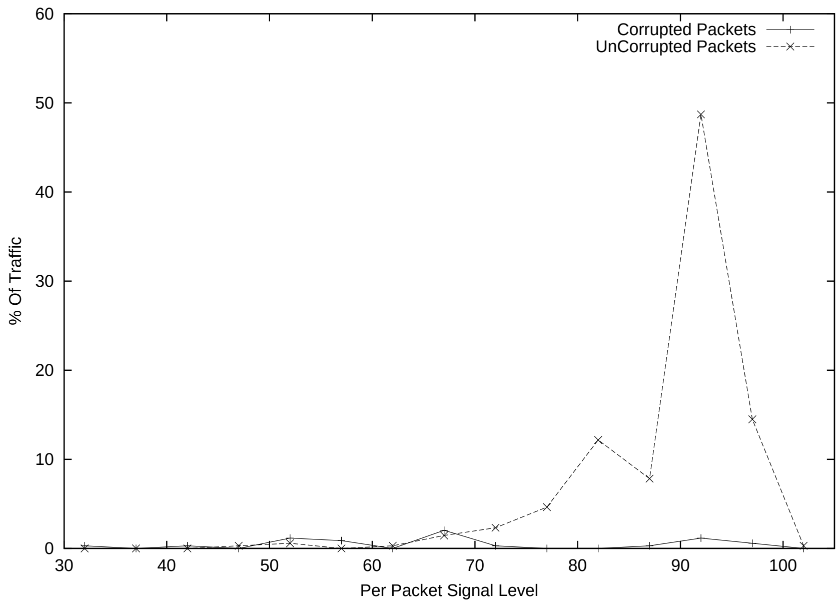


Figure 5.4: Just Outside CS316

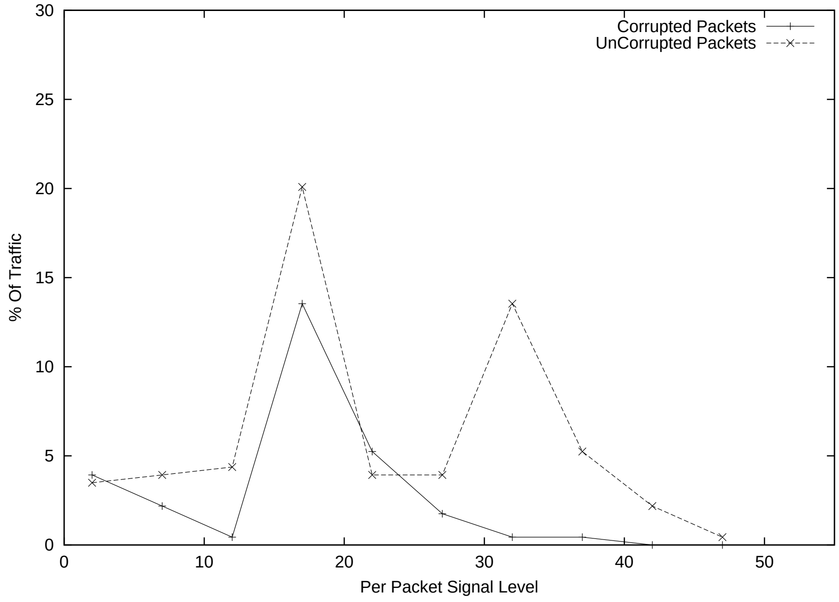


Figure 5.5: Near The Water

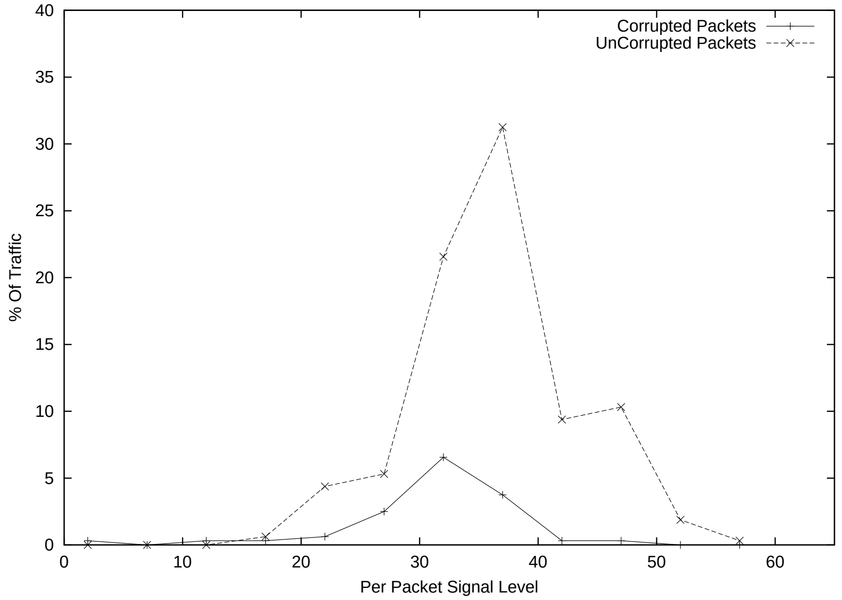


Figure 5.6: Outside CS319

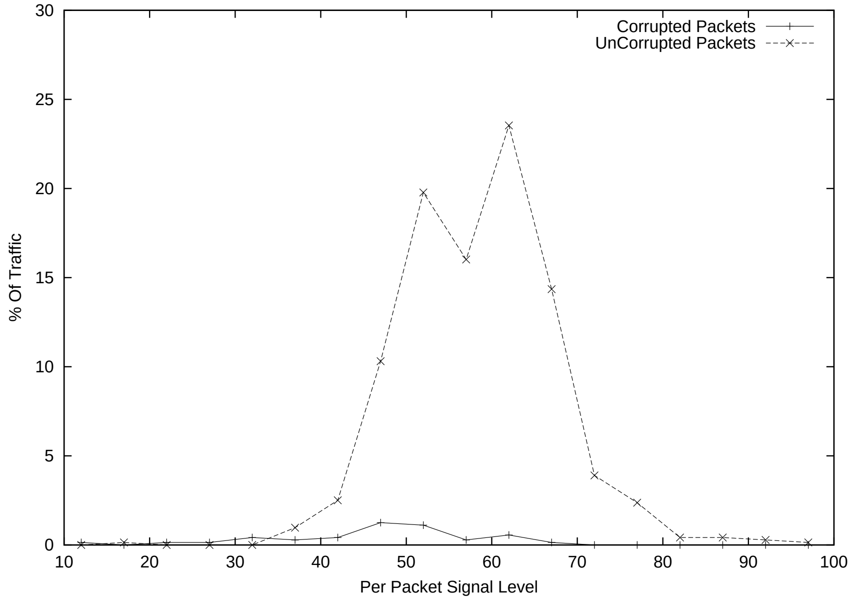


Figure 5.7: Inside CS315

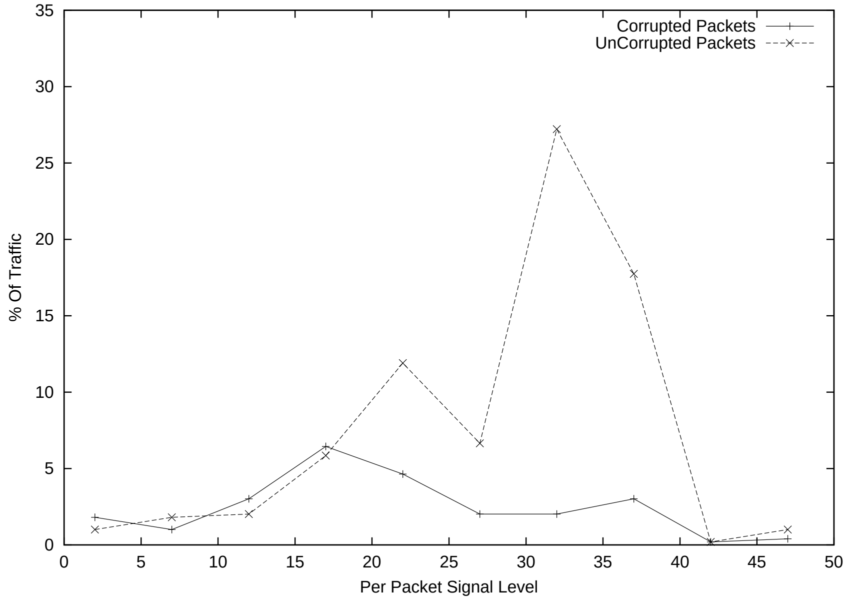


Figure 5.8: Inside CS314

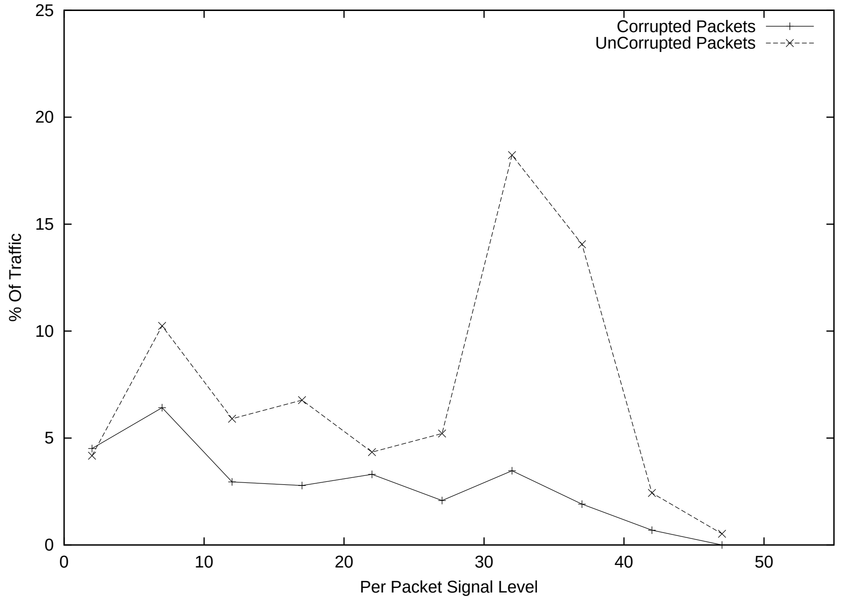


Figure 5.9: Near The Stairs

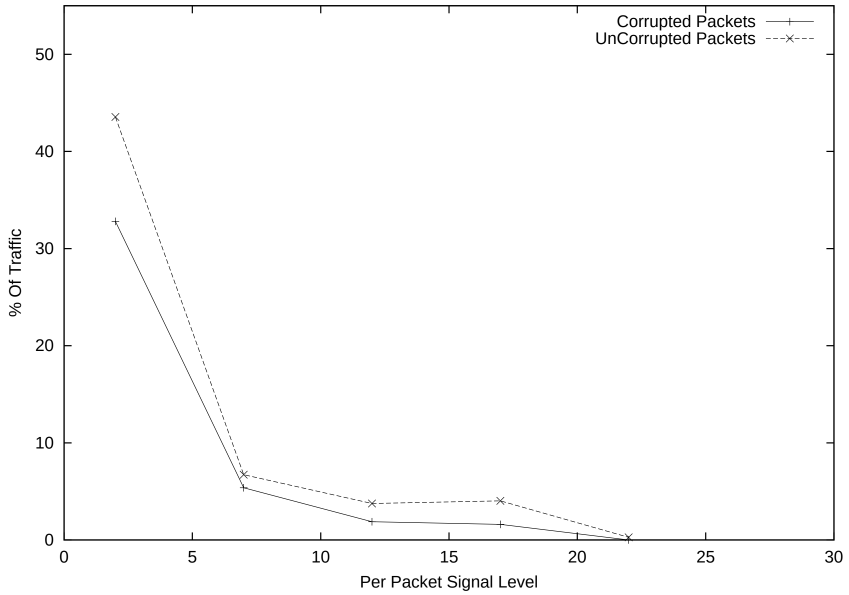


Figure 5.10: Outside The Lift

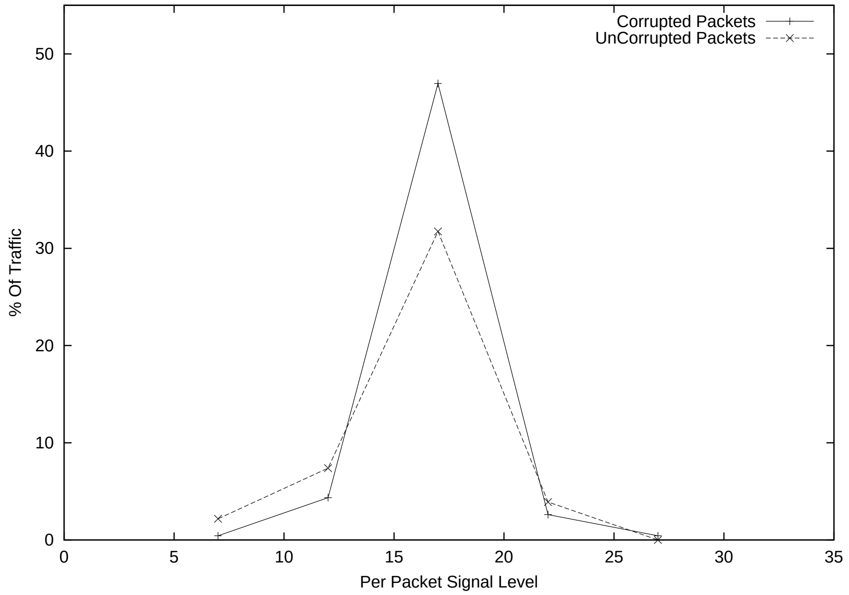


Figure 5.11: At The Lab Entrance

Chapter 6

Conclusions

In this thesis, we have built and described the generic WLAN Watch tool that allows the behavior of 802.11b wireless networks to be studied. Using our tool, the behavior of these networks can be analyzed by varying several parameters like channel (frequency), transmission rate, distance between transmitter and receiver, size of frames, mobility of wireless stations and presence of other wireless stations functioning in various channels. We have also conducted an experiment to demonstrate the usefulness of the WLAN Watch tool. The current version of our tool supports Cisco Aironet 350 Series and Intersil Prism-II network cards. The tool can be easily extended to support other NICs which have good support under Linux and which have the experimental RFMON mode.

We conclude that the various features desired in any generic wireless monitoring tool can be found in our tool. It provides the platform for modelling the error rate, packet loss, signal strength, throughput, etc., as a function of transmission rate, distance between transmitter and sniffer, size of frames, mobility and frequency of communication. It provides the user with per packet signal and noise levels. It also provides the options to vary the packet size and channel for conducting various experiments.

Though our tool is generic, while used for sniffing, the Cisco Aironet card drivers do not provide per packet signal, noise levels and do not pass the corrupted packets up to the upper layers. However, these cards can still be used for transmitting the packets on the machine running WLAN Watch client. The performance of our tool has been observed on desktops and laptops. It has been found to perform better when installed on laptops. However, the performance of our tool on handheld devices has not been observed. Our tool has performed well for the 11Mbps standard of IEEE 802.11b and it would be interesting to note its performance on higher data rate standards.

Appendix A

Setting up drivers and other tools

The *WLAN Watch* tool was built on a machine hosting RedHat 7.3 (linux kernel version 2.4.18-3). So, a kernel version not earlier to it is recommended.

A.1 *linux-wlan-ng* driver setup

This section describes the steps required to setup the driver for Prism-II cards from Intersil. The *wlan-ng* driver does not pass the corrupted frames up. The driver has been modified a bit for passing up the corrupted frames as well.

The README included in the driver package describes the steps to install the driver. However, the required sections from the README are repeated below along with some of my comments, just for convenience.

A.1.1 Prerequisites

To build *linux-wlan-ng* you will need:

1. Configured kernel source code for the kernel you are running. Ideally, this will be the resulting tree after building your own kernel. Configured means that you have at least run "`make config`", "`make menuconfig`", or "`make xconfig`". If you are trying to build *linux-wlan-ng* for a previously existing

kernel binary (one you did not build yourself), look for help on the mailing lists because it can be tricky.

2. The good David Leffler identified that if you are having difficulty with *_netlink_* symbols, you may have a problem with `"make clean"` in the kernel tree. Do a `"make mrproper"` followed by `"make config"` and the rest of the kernel build process. `"make mrproper"` does a more thorough cleaning of the kernel tree. For more info, look for David's comments in the *linux-wlan-user* mailing list.

3. If you are building a driver for a PCMCIA card, you *might* also need the configured PCMCIA source code for the pcmcia_cs subsystem you are currently running. You can obtain the latest pcmcia-cs package from sourceforge.net. Please go through the PCMCIA-HOWTO included in the downloaded package to configure the pcmcia-cs package. Again for simplicity, I am repeating what needs to be done usually:

   (a) Unpack `pcmcia-cs-X.Y.Z.tar.gz` in `/usr/src`.

   (b) Run `"make config"` in the new `pcmcia-cs-X.Y.Z` directory.

   (c) Run `"make all"` and then `"make install"`.

   This step should not be required for latest kernel versions. Hence only do this if you face any problems with the drivers in the kernel.

A.1.2 Building *linux-wlan-ng*

1. Untar the package using the command :
   `tar -xvzf linux-wlan-ng-X.Y.Z.tar.gz`

2. Make sure you have configured kernel and (optionally) pcmcia sources on your system. Note that if you are only building the prism2_pci, prism2_plx, or prism2_usb drivers you don't need the pcmcia-cs source tree.

3. To configure the *linux-wlan-ng* package, run `"make config"`. The following set of questions will be asked. The default answer is in braces. Just press <Enter> to select the default answer:

- Build Prism2.x PCMCIA Card Services (_cs) driver? (y/n) [y]: Select "y" if you want to build the Prism PCMCIA driver. If you select "n", the PCMCIA related questions below will not be asked.
- Build Prism2 PLX9052 based PCI (_plx) adapter driver? (y/n) [y]: Select "y" if you want to build the Prism driver for PLX PCI9052 PCI or PCMCIA adapter based solutions.
- Build Prism2.5 native PCI (_pci) driver? (y/n) [y]: Select "y" if you want to build the Prism driver for Prism2.5 ISL3874 based native PCI cards. This includes PCI add-in cards and the mini-pci modules included in some notebook computers (but not all, some use internal USB modules).
- Build Prism2.5 USB (_usb) driver? (y/n) [y]: Select "y" if you want to build the Prism driver for Prism2.5 ISL3873 based USB adapters. This includes USB add-on modules and the internal modules included in some notebook computers.
- Linux source directory [/usr/src/linux]: The config script will attempt to automagically find your kernel source directory. If found, the kernel source directory will be presented as the default selection. If the default selection is wrong, you may correct it here.
- pcmcia-cs source dir [/usr/src/pcmcia-cs-3.1.29]: If the "_cs" driver is selected above, the configure script will attempt to present a reasonable default for the pcmcia source directory. If the presented directory is incorrect, you may change it here. If the "_cs" driver is not selected, this prompt will not appear.
- PCMCIA script directory [/etc/pcmcia]: If the "_cs" driver is selected, this prompt allows you to change the location where the pcmcia scripts will be installed. Only do this if you have installed the rest of the pcmcia_cs scripts to a non-default location.
- Alternate target install root directory on host []: This prompt allows you to specify an alternative root directory for the install process.
- Module install directory [/lib/modules/2.2.20]: Select where you want the

driver modules to be installed. The script constructs a default location using the output of `uname`. If you have not yet installed the kernel you will run *linux-wlan* with, and the new kernel has a different version string, you will need to change this value.

- Prefix for build host compiler? (rarely needed) []: When cross-compiling or using different compilers for kernel and user-mode software, it is sometimes (but rarely) necessary to specify a different compiler prefix to use when compiling the tools that are built to run on the build host during the *linux-wlan-ng* build process.
- Build for debugging (see doc/config.debug) (y/n) [y]: This option enables the inclusion of debug output generating statements in the driver code. Note that enabling those statements requires the inclusion of *insmod/modprobe* command line arguments when loading the modules. See the document doc/config.debug for more information.

4. To build the package, run "`make all`"
5. To install the package, run "`make install`" (as root).
6. If the above "`make install`" fails due to the non availability of the directory structure `/usr/local/man/man1`, then create it accordingly. In my case it was "`mkdir /usr/local/man`" and "`mkdir /usr/local/man/man1`".

A.2 *airo* driver setup

This section describes the steps required to setup the driver for Cisco Aironet cards. There are two versions of the *airo* driver in the latest kernel sources and both get compiled and installed into the `/lib/modules` directory. Note that the *airo* drivers installed in `/lib/modules/2.4.18-3custom/kernel/drivers/net/pcmcia/` do not support **RFMON** mode. The modules installed in `/lib/modules/2.4.18-3custom/kernel/drivers/net/wireless/` do support **RFMON** mode. So, to make sure that the modules in the `wireless` subdirectory get loaded when *modprobe*d, I chose to copy the *airo** modules installed in `wireless` subdirectory into the `pcmcia` subdirectory. NOTE: By default, *modprobe* loads the modules in pcmcia subdirectory for the distribution I have.

It would be helpful to note that these drivers can put the card into **RFMON** mode:

- The drivers available in kernel 2.4.18
- The drivers available in Kernel 2.4.19
- CVS drivers from `airo-linux.sourceforge.net`. But they exhibit lockups, the card stops reporting packets, etc.

Drivers that definitely do not support **RFMON** mode:

- Drivers from `Cisco.com`
- Drivers in `pcmcia-cs` package

A.3 Miscellaneous Tools

In addition to the drivers, the following tools are required.

Wireless Tools

Though not strictly required, it is good to have them. Get the latest version of the wireless tools from Jean Tourrilhes's site. Refer the bibliography.

Aironet Client Utility

A tool used for easily configuring the Cisco Aironet NICs.

Glade

Glade is not usually required to use the *WLAN Watch* tools. However, it would be needed to modify the *WLAN Watch* Client or Server GUI.

Autoconf and Automake

Make sure that *autoconf* and *automake* are installed on the machine. These are required by the applications generated by Glade.

Set PATH

Make sure the tools *wlanctl-ng*, *ifconfig* and *modprobe* are in the PATH and *tcpdump* is also installed.

Bibliography

[1] Bob O'Hara, Al Petrick. *IEEE 802.11 Handbook A Designer's Companion*. Standards Information Network, IEEE Press.

[2] Giao T. Nguyen, Randy H. Katz, Brain Noble, Mahadev Satyanarayanan. *A trace-based approach for modelling wireless channel behavior*, ACM Press, Series-Proceeding-Article, 597-604.

[3] Luis Munoz, Marta Garcia, Johnny Choque, Ramon Aguero and Petri Mahonen. *Optimizing Internet Flows over IEEE 802.11b Wireless Local Area Networks: A Performance-Enhancing Proxy Based on Forward Error Correction*, IEEE Communications Magazine, December 2001.

[4] Andy Dornan. *Emerging Technology: Wireless Lan Standards*, Network-Magazine.com, 02-06-2002.
http://www.networkmagazine.com/article/NMG20020206S0006

[5] Kismet - 802.11 sniffer.
http://www.kismetwireless.net/

[6] The Wireless Extensions.
http://www.hpl.hp.com/personal/Jean_ Tourrilhes/Linux/Tools.html

[7] Cisco Aironet 350 Series.
http://www.cisco.com/en/US/products/hw/wireless/ps458/index.html

[8] Intersil Prism-II.
http://www.intersil.com/design/prism/ser-pii-11mbps.asp

[9] Netperf benchmark.
http://www.netperf.org/netperf/NetperfPage.html

[10] Tcpdump - Network Protocol Analyzer.
http://www.tcpdump.org

[11] Ethereal - Network Protocol Analyzer.
http://www.ethereal.com

[12] AVS - Driver developers for Intersil Prism-II chipsets.
http://www.linux-wlan.com/

[13] Wireless LAN resources for Linux.
http://www.hpl.hp.com/personal/Jean_ Tourrilhes/Linux/index.html

[14] Intersil. *PRISM Driver Programmer's Manual.*
CHOICE-Intersil, Version 2.10, 2001-08-31.

[15] W. Richard Stevens. *UNIX Network Programming.*
Prentice Hall, 1990.

[16] Alessandro Rubini. *Linux Device Drivers.*
O'Reilly & Associates, Inc., 1st edition, 1998.

[17] Linux Device Drivers by Alessandro Rubini - online.
http://www.xml.com/ldd/chapter/book/index.html

[18] Ethereal - Network protocol analyzer.
http://www.ethereal.com/

[19] Wireless LANs in use.
http://www.hpl.hp.com/personal/Jean_ Tourrilhes/Linux/Linux.Wireless.usage.html

[20] The Aironet Client Utility.
http://www.cisco.com/univercd/cc/td/doc/product/wireless/airo_ 350
/350cards/windows/legacy/scg/pc_ ch1.htm

[21] Glade - GUI builder.
http://glade.gnome.org/index.html

[22] David A Rusling. *The Linux Kernel.*
http://en.tldp.org/LDP/tlk/tlk.html

[23] The Linux Kernel HOWTO.
http://www.tldp.org/HOWTO/Kernel-HOWTO.html

[24] Cross-Referencing Linux.
http://lxr.linux.no

[25] Mailing lists : *linux-wlan-ng* driver.
linux-wlan-devel@lists.linux-wlan.com
linux-wlan-user@lists.linux-wlan.com

[26] Mailing lists : *airo* driver.
airo-linux-gen80211@lists.sourceforge.net
airo-linux-general@lists.sourceforge.net

[27] Mailing list : kismet sniffer.
wireless@kismetwireless.net

[28] Mailing list : Glade.
glade-users@ximian.com