ANALYSIS OF OPEN SOURCE DRIVERS FOR IEEE 802.11 WLANs

A THESIS

Submitted by

VIPIN M

in partial fulfilment for the award of the degree
of

MASTER OF SCIENCE (BY RESEARCH)

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING
ANNA UNIVERSITY CHENNAI
CHENNAI - 600 025

JUNE 2010

ANNA UNIVERSITY CHENNAI
CHENNAI - 600 025

BONAFIDE CERTIFICATE

Certified that this thesis titled **“Analysis of Open Source Drivers for IEEE 802.11 WLANs”** is the bonafide work of **Mr. M. VIPIN** who carried out the research under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion of this or any other candidate.

DR.S.SRIKANTH
(SUPERVISOR)
Member Research Staff
AU-KBC Research Centre
Madras Institute of Technology,
Anna University Chennai,
Chromepet,
Chennai-600 044, India

ABSTRACT

Wireless Local Area Network (WLAN) interfaces are common in personal computing devices such as PDAs, mobile phones, and laptops. IEEE 802.11 is the de facto standard used for WLAN devices. The standard has evolved over the last ten years. The WLAN medium access control (MAC) implementation has also taken shape over the years and open source implementation of WLAN drivers in Linux kernel is an evolving area.

Linux is a popular and stable open source operating system kernel implementation. The Linux kernel has a modular architecture and is comparatively easy to plug new components. The Linux kernel supports a vast category of network interfaces. A few years ago, the WLAN implementation was also added to this mainline kernel development. We discuss the general network implementation, architecture and operation for understanding the complex implementation of WLAN. The WLAN driver implementation in Linux kernel is explained further with a detailed discussion of the structure. The WLAN driver initially evolved from propriety to partially proprietary and then to fully open source over time.

The WLAN implementation in kernel is based on three layers - control plane, protocol stack and hardware driver. The control plane provides the group of functions for management and control of the WLAN interface and the hardware. Current implementation of the WLAN control plane is unified across different vendors, the same way vendors use a single protocol stack implementation. We discuss the control plane and the protocol stack functions for different operations. The detailed step by step operations on the frame in different layers are explained.

The process for different management functions are explained based on the three layers. This explanation traces the functional flow of different management functions from application layer to the hardware. The transmission and reception of data in WLAN with respect to different layers are discussed. The extra functions supported by the WLAN are further discussed.

ACKNOWLEDGEMENT

I thank **Dr. S. Srikanth,** my supervisor for being a good friend and a wonderful teacher. Much is attributed to his free spirit and timely checks that have kept me on track. I am indeed lucky to have him as my guide.

I would like to thank **Prof. C.N. Krishnan** for giving me this opportunity to do my M.S at AU-KBC Research Centre, MIT Campus Anna University Chennai. I also take this opportunity to thank **Prof. P. Manohar**, for the support to do my research work at AU-KBC Research Centre.

I take this opportunity to thank the members of the monitoring committee **Mr. Gangadharan** for his constant support and invaluable suggestions. I extend my sincere thanks to Dr. Jackson and Mr. Vasudevan for their invaluable technical inputs.

I convey my heartfelt thanks to my friends Selvam, Sarad, Murugesapandian, Mohamed Imran, Sathish, Sridhar, Muthusaravanan and all others, who had helped me.

VIPIN. M

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AP	Access Point
ACK	Acknowledgement
API	Application Program Interface
BSS	Basic Service Set
BSSID	Basic Service Set Identifier
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CCA	Clear Channel Assessment
CCK	Complementary Code Keying
DSSS	Direct Sequence Spread Spectrum
DCF	Distributed Coordination Function
ESS	Extended Service Set
FHSS	Frequency Hopping Spread Spectrum
HAL	Hardware Abstraction Layer
HT	High Throughput
ISM	Industrial, Scientific and Medical
IBSS	Independent Basic Service Set
I/O	Input / Output
IOCTL	Input Output Control
KOBJECT	Kernel Object
LAN	Local Area Network
LLC	Logical Link Control
MAC	Medium Access Control
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
MIMO	Multiple Input Multiple Output
NAV	Network Allocation Vector
NETDEV	Network Device
OFDM	Orthogonal Frequency Division Multiplexing

PPDU	PHY Protocol Data Unit
PCF	Point Codenation Function
SSID	Service Set Identifier
SOFTMAC	Software MAC
STA	Station
SYSFS	System File System
TID	Traffic Identifier
VAP	Virtual Accec Point
WEP	Wired Equvalent Privacy
WXT	Wireless Extenssions
WDS	Wireless Distrubution System
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network
WPA	Wi-Fi Protected Access

CHAPTER 1

INTRODUCTION

Wireless Local Area Network (WLAN) devices are common in all personal computing devices such as PDAs, mobile phones, and laptops. IEEE 802.11 is the de facto standard used for WLAN devices. The standard has evolved over the last ten years and has spawned into a number of subgroups such as 802.11b, 802.11a, 802.11g, and 802.11n. Each subgroup is an enhancement over the basic 802.11 in functionality and/or performance. The newly developed standards are backward compatible and support legacy systems. This makes it feasible for existing hardware to work with new products. This chapter discusses the history, general functions and operations for IEEE 802.11 based WLAN.

1.1 HISTORY

The first version of the standard IEEE 802.11 WLAN was released in 1997 and ratified in 1999 (IEEE 802.11-2009). It specified two bit rates (1 or 2 Mbit/s) and three physical layer technologies. It used infrared or microwave transmission technologies with frequency-hopping spread spectrum (FHSS) at 1 Mbit/s or 2 Mbit/s and direct-sequence spread spectrum operating (DSSS) at 1 Mbit/s or 2 Mbit/s. The microwave transmission over the Industrial Scientific Medical (ISM) frequency band at 2.4 GHz and some earlier WLAN technologies used lower frequencies, such as the U.S. 900 MHz ISM band in US. The IEEE 802.11b standard was defined as an extension to the previously one. It extended the data rates up to 11 Mbps, with the help of complementary code keying (CCK) modulation. The IEEE 802.11a uses the same frame format as the original standard, but physical layer is based on orthogonal frequency division multiplexing (OFDM). This operates in the 5 GHz band with a maximum data rate of 54 Mbit/s.

IEEE 802.11g was ratified in June 2003 and it works in the 2.4 GHz band, which uses the same OFDM based transmission scheme as 802.11a. It operates at a maximum physical layer bit rate of 54 Mbit/s (IEEE 802.11-2007). IEEE 802.11g is fully backwards compatible with IEEE 802.11b. IEEE 802.11b/g devices suffer interference from other products operating in the 2.4 GHz band. Later, QoS mechanisms were defined in IEEE 802.11e to support delay sensitive applications such as live video streaming and voice transmissions. The IEEE 802.11e standard is purely a MAC layer extension. IEEE 802.11n is a recent extension which improves upon the previous 802.11 standards. It supports data rate of up to 600 Mbps with better QoS and backward compatibility with legacy (IEEE 802.11a/b/g) devices by using multiple input multiple output (MIMO) and many other newer features for achieving the high data rates. A brief summary is given in Table 1.

Table 1.1 Evolution of 802.11

	802.11	**802.11b**	**802.11a**	**802.11g**	**802.11n**
Band of operation (GHz)	2.4	2.4	5	2.4	2.4 / 5
Min & Max Data rate (Mbps)	1 – 2	1 - 11	6 – 54	1 - 54	1 - 600
Channel Bandwidth (MHz)	22	22	20	20	20/40
Modulation	FHSS, DSSS	DSSS	OFDM	DSSS, OFDM	DSSS,OFDM, MIMO-OFDM
Radio Coverage (ft)	300	300	200	300	450

1.2 ARCHITECTURE

The building block of an 802.11 network is the basic service set (BSS), which is simply a group of stations (STAs) that can communicate with each other without using network cables (Matthew Gast 2005). The two network structures are independent basic service set (IBSS), which is simply a group of STAs that can communicate with each other and infrastructure networks, where an access point (AP) is used to coordinate the communications between the stations. IBSS network is often formed on a temporary basis, and is commonly referred to as an ad hoc network. In infrastructure networks, the AP acts as a bridge between the wireless network and the wired backbone network. The other network structures are wireless Distribution System (WDS) and wireless mesh network (WMN).

1.2.1 Independent basic service set

STAs in an IBSS communicate directly with each other as shown in Figure 1.1. These STAs must be in range to have direct communication. The smallest possible 802.11 network is an IBSS with two stations. IBSSs are composed of a small number of stations set up for a specific purpose and for a short period of time. IBSSs networks are referred to as ad hoc networks. One example is a network to support a meeting in a conference room.

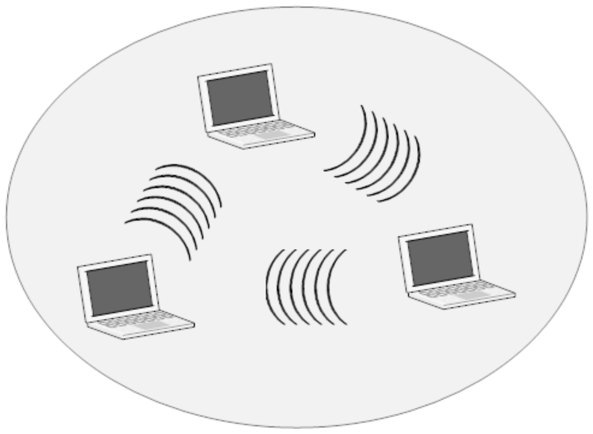

Figure 1.1 Independent basic service set

1.2.2 Infrastructure networks

Infrastructure networks use an AP as shown in Figure 1.2. The STAs are not allowed to directly communicate with each other and all the traffic must flow through the AP. The AP acts as a bridge between the wireless network and the wired backbone network. The IEEE standard does not provide any guidelines about the usage of backbone technology. However, most of the commercial products support Ethernet as a backbone network.

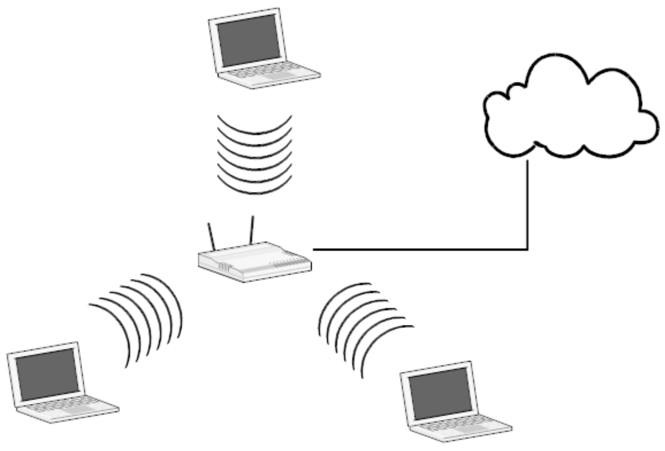

Figure 1.2 infrastructure network

The IEEE 802.11 standard places no limit on the number of mobile stations that an access point may serve. Implementation considerations may, of course, limit the number of mobile stations an access point may serve. In practice, however, the relatively low throughput of wireless networks is far more likely to limit the number of stations placed on a wireless network.

An Extended Service Set (ESS) is a set of one or more interconnected BSSs and integrated local area networks (LANs) that appear as a single BSS to the logical link control layer at any station associated with one of those BSSs.

1.2.3 Wireless Distribution System

WDS is a system that enables the wireless interconnection of access points in an IEEE 802.11 network. WDS is a proprietary implementation and the methods and supported functions vary vendor to vendor. Bridge and repeater are some of the major WDS modes of operation. In a bridge network two wireless AP act as an interconnecting device between two networks as shown in Figure 1.3. In the other mode of operation as repeater it allows a wireless network to be expanded using multiple APs without the need for a wired backbone to link them.

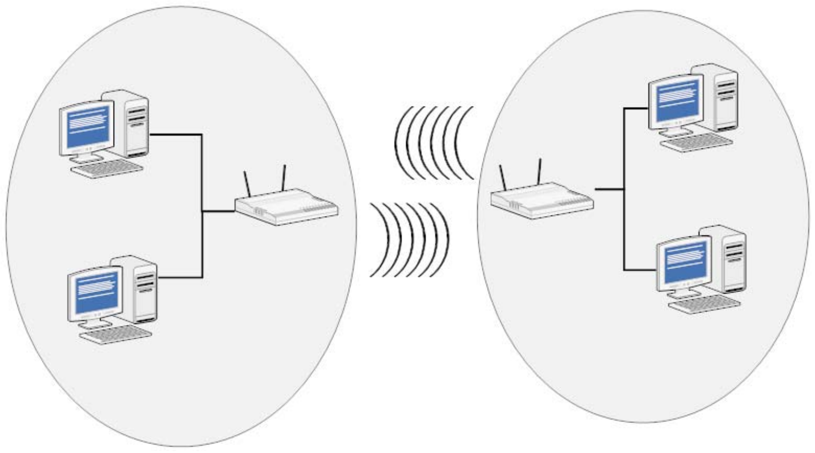

Figure 1.3 Wireless Distribution Systems

An AP can be either a main or relay. A main AP is typically connected to the wired Ethernet. A relay AP relays data between wireless STAs to the main AP. A relay AP accepts connections from wireless clients and passes them on to main AP.

1.2.4 Wireless mesh network

A WMN is a network made up of STAs organized in a mesh topology. A wireless mesh network can be seen as a special type of wireless ad-hoc network. A WMN is reliable and offers redundancy. When one node can no longer operate, the rest of the nodes can still communicate with each other, directly or through one or more intermediate nodes shown in Figure 1.4. The standard discuss about WMN is IEEE 802.11s.

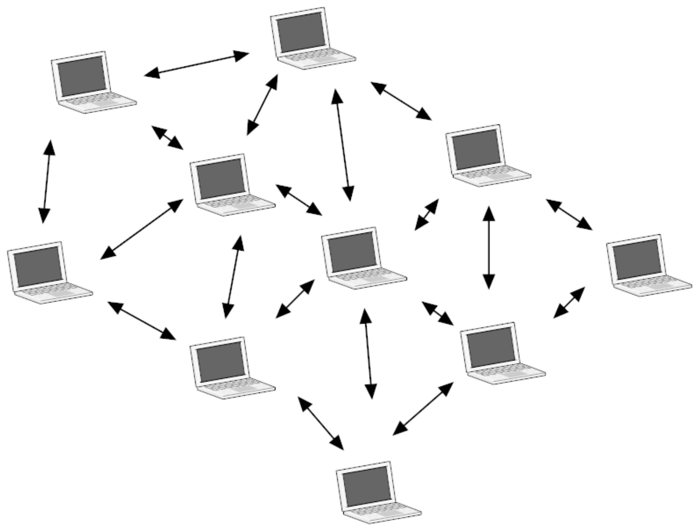

Figure 1.4 Wireless mesh network

1.3 MEDIUM ACCESS SCHEMES

IEEE 802.11 defines two medium access schemes. Distributed co-ordination function (DCF) which is mandatory and point co-ordination function (PCF) which is optional. Most of the commercial products implement DCF alone. In DCF, the channel access mechanism is distributed. The AP and STA have to content for the medium to get its chance for transmission. It is basically a carrier sense multiple access with collision avoidance (CSMA/CA) mechanism. Since collision detection is not possible due to the nature of wireless medium, collision avoidance method was adopted. The status of wireless medium is identified with the help of physical carrier sensing and virtual carrier sensing.

Clear channel assessment (CCA) is used at the PHY layer to monitor the status of the medium and report this to the MAC layer. Physical carrier sensing is done by any of the methods discussed below. The first method uses energy measurement in the wireless medium. It reports as a busy medium when it senses a signal having a power level of at least -85 dBm (IEEE 802.11-2007). The second method observes the PHY layer headers of WLAN frames in the current operating channel. Since WLAN is operating in the crowded unlicensed band, most vendors prefer the later option. Virtual carrier sensing is done by the MAC layer by announcing the required medium usage information through the header portion of the frame. This duration is also known as network allocation vector (NAV). Both carrier sensing methods have to report the medium status as idle; otherwise it is considered that the medium is busy.

1.4 MANAGEMENT FUNCTIONS

In IEEE 802.11 a set of operations such as scanning, authentication and association etc can be classified as management function.

1.4.1 Scanning

Scanning is among one of initial process in a WLAN network. By this process a STA discovers a network/AP and the details with respect to it. There are two forms of scanning are possible, they are passive scanning and active scanning.

In passive scanning the STA receive the broadcasted information (Beacon) and find the available networks. Once the station has discovered the AP through its received beacon The STA send probe request to the AP directly for additional information.

In active scanning, a STA transmits probe request frames on each of the channels where it is seeking a network. The Probe Request frame can include multiple details based on the type of scan it is performing. The name of the network service set identifier (SSID) is used when the STA scanning for the specific network. In other case broadcast SSID is used to find all available network in that channel. In the case of basic service set identifier (BSSID) this can be the MAC address of the specific AP or can be broadcast based on the scanning. In the case of broadcast Probe Request multiple APs may respond. Normal channel access procedures are used to avoid collisions.

1.4.2 Authentication

This is a process used by STA/AP for identifying each other. The original IEEE 802.11 supported two authentication methods the open system authentication and shared key authentication. In initial systems, wired equivalent privacy (WEP) is used for authentication. WEP has been shown to be insecure and the newer security techniques such as Wi-Fi Protected Access

(WPA) and WPA2 are now in use. IEEE 802.11i standard discussed about security in WLAN. WPA enforces IEEE 802.1X authentication and key-exchange and only works with dynamic encryption keys. Authentication is performed prior to association.

1.4.3 Association

This is the process by which a STA become part of a network or get connected to an AP. After this, a station is allowed to send data via an AP. Association provides a mapping between the STA and AP that allows messages within the distribution system (DS) to reach the AP with which the STA is associated and ultimately to the STA itself. At any given instant a STA may only be associated with a single AP.

This process is initiated by the STA. The STA send an Association Request to the AP. If the station is admitted, the AP responds with an Association Response. With the Association Request and Response exchange, the STA and AP exchange capability information (support for optional features) and the AP informs the STA of specific operating parameters within the network.

1.4.4 Reassociation

Reassociation is used in the case of mobility. This makes it possible to move from a presently associated AP to another within the same ESS for a STA. Reassociation may also be performed to change attributes of the STA association such as STA capability information. Reassociation is initiated by the STA by sending a reassociation request to the AP. The AP responds with a reassociation response.

1.4.4 Disassociation

Disassociation is the process of exiting from a network. This terminates an existing association and may be performed by either the STA or the AP. STA should attempt to disassociate when it leaves the network. In some cases this may not happen because of communication loss. In such scenario, a timeout mechanism allows the AP to disassociate the STA without a message exchange. To disassociate a STA from the BSS, the AP or STA sends a Disassociation frame. The other end acknowledges reception of the disassociation frame.

1.5 DATA/ACK FRAME EXCHANGE

In IEEE 802.11 a positive acknowledgement (ACK) in the form of a frame is used as transmission over a wireless medium is error prone. The basic mechanism by which this is achieved is to have the station that correctly receives a data frame addressed to it send an immediate, positive acknowledgement in the form of an ACK frame. If the station sending the data frame does not receive the ACK frame, it assumes the data frame was not received and may be retransmitted it. Broadcast and multicast data frames are directed to all or a subset of the stations in a WLAN and there is no ACK sent for these frames.

The number of retransmission attempts on a particular medium access control (MAC) Service Data Unit (MSDU) is limited. The transmitting station maintains a count of the number of retransmission attempts on an MSDU and when that count exceeds a configured retry limit the MSDU is discarded. To enhance the reliability with which acknowledgement feedback is provided, the ACK frame is modulated robustly, i.e. it is sent using a lower

PHY data rate than data frames sent to the same station. The additional overhead incurred with robust modulation is relatively small since the ACK frame itself is very short.

If MSDUs are large in size they are more prone to error or collisions can have a considerable reduction in performance. This can be reduced by using fragmentation. When fragmentation is used, large MSDUs are broken to smaller MPDUs. At low data rates, an unfragmented MSDU can occupy a large amount of air time. A bit error in the frame would result in the entire frame being retransmitted. With fragmentation the MSDU would be broken into smaller sections and each section encapsulated in an MAC Protocol Data Unit (MPDU). Each MPDU is sent in a separate PLCP Protocol Data Unit (PPDU) with the preamble of each PPDU providing a new channel estimate. A bit error would result in only the MPDU having errors segment being retransmitted.

1.6 CONTRIBUTIONS OF THE THESIS

The main focus of this thesis is in studying the structure of WLAN Linux kernel driver implementation. This includes a functional breakdown of the driver and the overall flow of information via functions. We survey the specific implementation methods used in the WLAN Linux stack and compare the legacy driver implementation with the newer Linux kernel implementation. For reference, the Atheros network device driver is taken as an example to discuss the WLAN structure, stack and driver implementation.

1.7 ORGANIZATION OF THE THESIS

The thesis is organized as follows. Chapter 2 briefly discusses the Linux kernel and the structure of it with respect to network. Chapter 3 describes the configuration and management functions of WLAN Linux kernel driver. Chapter 4 discusses the transmission and receive path of the WLAN Linux kernel driver. The conclusion is presented in Chapter 5.

CHAPTER 2

LINUX KERNEL DEVICE DRIVER

Linux kernel is the major open source UNIX like OS kernel implementation. Most of the drivers for the peripheral hardware are built as loadable modules. They are loaded when new hardware is identified or at boot time. When any module is loaded, they export their functions to the kernel space. Kernel space functions can be utilized by other kernel modules or user application through system calls. This chapter discusses Linux kernel structure specific to networking and the general structure of WLAN drivers.

2.1 LINUX KERNEL

Linux is a monolithic kernel. The Linux kernel has a modular architecture (Klaus Wehrle 2003). The Linux kernel can be further divided into three levels. The system call interface at the top level, which implements the basic functions such as read and write. At middle level is the kernel code, which can be defined as the architecture independent kernel code. This code is common to all of the processor architectures supported by Linux. The lowest level is the architecture dependent code. This code is the processor and platform specific for the different architectures. The Linux kernel is stable, efficient in terms of both memory and CPU usage. The most interesting aspect of Linux is that it can be compiled to run on a large number of processors and platforms with different architectural.

The important functions handled by Linux kernel are process management, memory management, file systems, device drivers and network.

2.1.1 Kernel module

Loadable kernel module is one of the important features of Linux, which allow it to extend its functionality at runtime. This means that we can add and remove functionalities to the kernel while the system is up and running. This pluggable code, which can be added to the kernel at runtime, is called a module. When the functionality of a module is no longer needed, it can simply be removed and the memory space used is freed. These modules can be of device drivers, file systems, network protocols and network drivers. Adding new functionality using kernel module require corresponding kernel interface to inform the rest of the kernel about the new components.

2.1.2 Device Driver

In any system other than processor and memory are the devices (Jonathan Corbet 2005). Any system operations to these devices are performed by code that is specific to the device being addressed. That code is called a device driver. The kernel must include in it a device driver for every peripheral present on a system. For example hard drive, keyboard and mouse.

Device drivers in the form of modules can be added or removed from Linux at any time. In Linux, network adapters are not treated as normal devices and so they are not listed in the */dev* directory.

2.1.3 Network Driver

The network driver identified as individual connections in Linux kernel. The Linux way to provide access to interfaces is by assigning a unique name to each network device (Example eth0 to an Ethernet network interface). The communication between the kernel and a network device driver is completely different from the other type of devices. The kernel calls functions related to packet transmission for sending packets.

Linux kernel network drivers also support a number of administrative tasks, such as setting addresses, modifying transmission parameters, and maintaining traffic and error statistics. The kernel API for network drivers gives functions for all the above operations. The network subsystem of the Linux kernel is designed independent of protocols. This is valid to both networking protocols (Internet protocol, etc.) and MAC protocols (Ethernet and IEEE 802.11, etc.). In Linux kernel, network driver is build to hide protocol issues and the physical transmission to be hidden from the protocol.

2.1.4 Device driver operations

There are a set of operation which is defined for any device drivers in Linux kernel. These operations are device registration, device driver unloading, and device operations. The device operations are open interface, close interface, packet transmission, and packet receive because we are discussing about network drivers.

Device registration is the set of operations occurring when a driver module is loaded into a running kernel. The kernel module requests resources and offers facilities. The driver should probe for its device and its hardware location, such as input/output (I/O) ports and IRQ line and register them. Network driver is registered by its module initialization function. The module initialization function inserts a data structure for each newly detected interface into a global list of network devices. All interfaces in a system are described by a network device structures (net_device) in Linux kernel.

The network device structure is a kernel object (kobject) and is reference counted and exported via system file system (sysfs). The memory for this structure allocated dynamically like other kernel structure by using functions (alloc_netdev). After the allocation of memory the registration of network device is carried out by calling corresponding function (register_netdev). The network device structure is one of the largest and cumbersome in Linux kernel. The next process is to setup the device. There are some general functions is to setup the device like *ether_setup* for Ethernet device.

Device driver unloading is the set of operations to be carried out when the module is unloaded. This includes cleanup function of memory and releasing of the network device structure. The memory cleaning including freeing the transmission and reception buffers.

Device methods are the set of operations that declares the functions acting on the network interface. Device methods for a network interface can be divided into two groups fundamental and optional. Fundamental methods include those that are needed to be able to use the interface and optional methods implement more advanced functionalities that are not strictly required. At the time of device setup, every driver export the operations support list to the kernel space. Some of the operations can be left NULL. The fundamental methods are explained herewith.

Opening operation is called when the module load time or Linux kernel boot and it probe for the interface. This assign address and other parameters before the interface carry packets by the kernel. The kernel opens or closes an interface in response to the user level commands like *ifconfig*.

When *ifconfig* is used to assign an address to the interface, it performs two tasks. First, it assigns the address by means of a socket I/O control set interface address input output control (ioctl). Then it sets the “interface up bit” in device flag by means of socket I/O control set interface flags ioctl to turn the interface on. When the interface is shutting down using ifconfig, the user level command ioctl is called to clear “interface up” bit in device flag.

Packet transmission operation is one of the most important tasks performed by network interfaces. The packet transferred to the kernel from the user level is transmitted through the physical interface by the kernel. As the kernel gets data to transmit, it calls the drivers transmission function. This function is generally made available through *hard_start_transmit* method. This function puts the data on an outgoing queue. The packet handled by the kernel is contained in a socket buffer structure (sk_buff). Each network packet belongs to a socket in the higher network layers. The input/output buffers of any socket are lists in the socket buffer structures. The same socket buffer structure is used throughout all the Linux network subsystems. The socket buffer passed to *hard_start_xmit* contains packet to send on the media. The socket buffer data points to the packet being transmitted and socket buffer length element gives length of the data in octets.

Packet reception is another important task for a network interface. Receiving data from the network is complex than transmitting it. A socket buffer must be allocated and handed off to the upper layers. There are two modes of packet reception that are implemented by network drivers. These methods are interrupt driven and polled. Most of the drivers implement the interrupt driven technique. Some drivers for high-bandwidth adapters may also implement the polled technique. A receiver function is called from the network interface interrupt handler after the hardware has received the packet, and it is already in the computers memory. Receiver function receives address to the data and the length of the packet. The network interface sends the packet and some additional information to the upper layers of networking functions. This code is independent of the way the data pointer and length are obtained. The first step is to allocate a buffer to hold the packet. The buffer allocation function is called with the length information (dev_alloc_skb). The data frames are processed the frame is send to the layer above through the socket buffer.

2.1.5 General structure of network driver

The user space applications interact with the kernel space modules through system calls. A simplified Linux kernel network stacks interaction between user applications, TCP/IP stack, network device driver and hardware is shown in Figure 2.1. The operations in the figure can be categorized as configuration/management, transmission path, and receive path (Klaus Wehrle 2003).

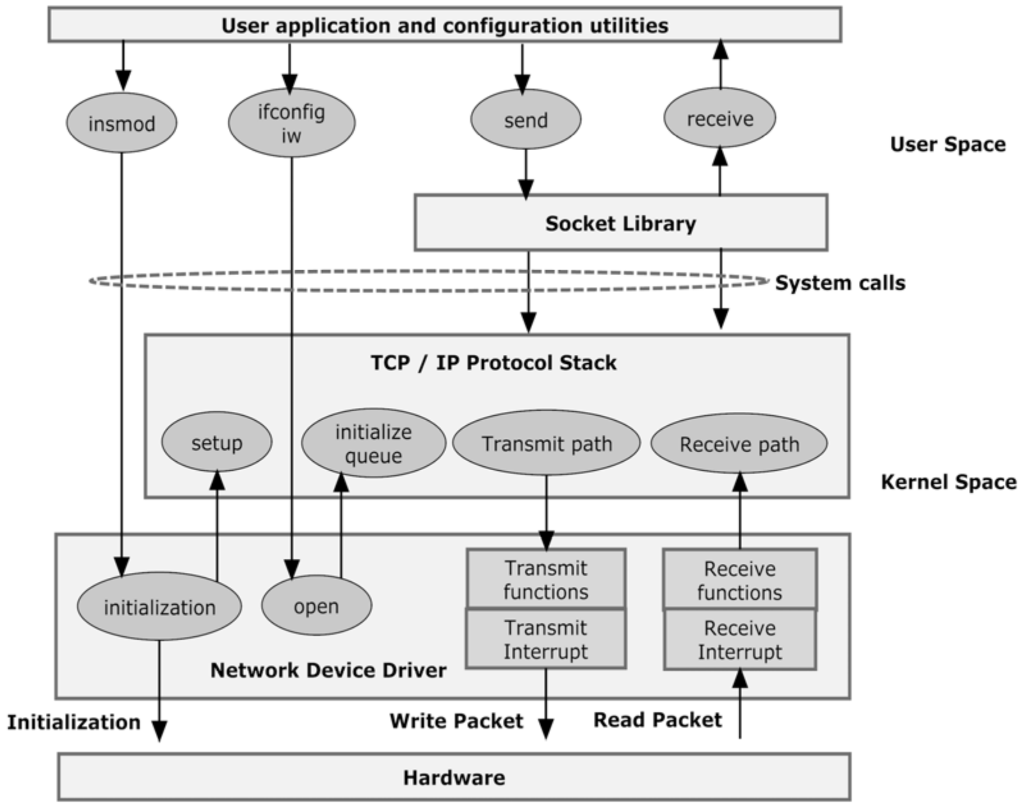

Figure 2.1 Linux kernel interaction

Insmod is a user level function that loads any kernel module (Jonathan Corbet 2005). The network device drivers may have multiple kernel modules based on functionalities such as hardware module and protocol module. *Insmod* calls initialization functions of each module. This initialization function initializes the hardware and registers the public functions used by other layers in the kernel network stack for transmission, reception and management. The configuration functions from the user level interact with the TCP/IP and the network driver (for configuring and initialization interface). For sending the packet from the user level, it uses the socket function which internally sends the packet to the TCP/IP stack and further to the network driver. The protocol modifications are made and calls the transmit interrupt for physical transmission from hardware through the network driver. The packet reception and process is initiated by the hardware interrupt generated as it receives a packet in hardware in the receive path.

2.2 WLAN DRIVER STRUCTURE

The WLAN driver in Linux kernel is structured similar to other network device (netdev) drivers. In general, wireless devices are connected through PCI or USB interfaces. The interconnecting driver varies based on the interface to wireless device (PCI or USB). Wireless netdev is similar to an Ethernet interface for higher layer with extra features.

In the case of software and hardware implementations, WLAN will be capable of handling all mandatory functions proposed by the standard. Depending on the vendor, they may implement some optional functions and proprietary features. The Linux kernel WLAN drivers developed by major vendors implement most of the mandatory functions while the others are still in development. The modes of operation supported by any WLAN device are ad-hoc, infrastructure, mesh, WDS, virtual access point (VAP), virtual interface and monitor (madwif-project.org 2009). VAP is a multiple virtual AP with a single hardware device. Virtual interface is multiple logical interfaces using a single physical interface. Monitor mode is used for passively sniffing the air interface. The network driver for Atheros WLAN hardware was initially started as an independent project and is now a part of the Linux kernel itself.

Earlier, network interface hardware used to handle the MAC and other lower layer protocols. The other implementation model was network device hardware and firmware. None of the above types of implementations provided the end user developer, the freedom for development because they did not have access to the firmware code. The next generation of network devices moved the MAC functionalities to software. The software MAC (SoftMAC) is loaded as a Linux kernel module. This implementation method gave the end user developer better access to code. The two major IEEE 802.11 WLAN SoftMAC implementations are *net80211* (madwif-project.org 2009) and mac80211 (linuxwireless.org 2009).

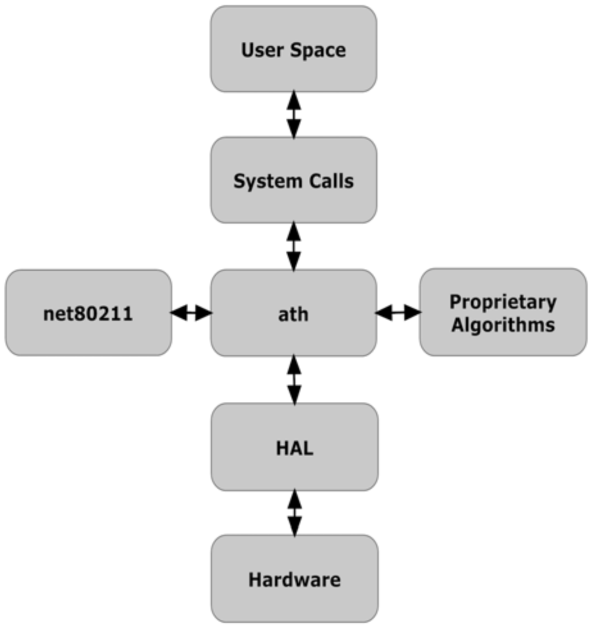

Figure 2.2 MADWiFi Structure

The original open source Atheros driver was developed for FreeBSD. MADWiFi is a modified version of it. MADWiFi drivers work with binary and open hardware abstraction layer (HAL) and net80211. Figure 2.2 shows the MADWiFi structure comprising of *net80211*, *ath*, HAL and proprietary algorithms which are in kernel space. MADWiFi driver is under dual license BSD and GPL v2.

HAL is a piece of software that handles the access to the WLAN hardware. This is unlike firmware, which loads into the onboard microcontroller. HAL executes in the host processor and provides application programming interfaces (APIs) to the driver in order to access the hardware. Atheros hardware is flexible and it is capable of working in a wide range of frequency spectrum. HAL restricts use of operation frequency band and transmission power based on the operating country. HAL acts as a wrapper around the hardware registries. This allows the driver to interact with hardware in the permissive manner. At present, some versions of HAL source code are open source. The newer versions of MADWiFi work with openHAL.

The original net802.11 is an IEEE 802.11 SoftMAC implementation of FreeBSD. The stack net80211 was suggested as the generic stack for IEEE 802.11 to the Linux kernel, but did not satisfy the criteria of a real Linux kernel network stack and was never picked. *net80211* supports a wide range of modes such as station (STA), AP, ad-hoc, monitor and WDS. The other device dependent codes are in ath module which includes Atheros network hardware dependent functions such as hardware initialization and interface configuration. Other functions are implemented as separate modules. For e.g., rate adaptation, sync scan etc. are implemented as loadable modules.

The new implementation of WLAN network device is a part of Linux kernel source tree (git.kernel.org 2009). This implementation is based on SoftMAC.

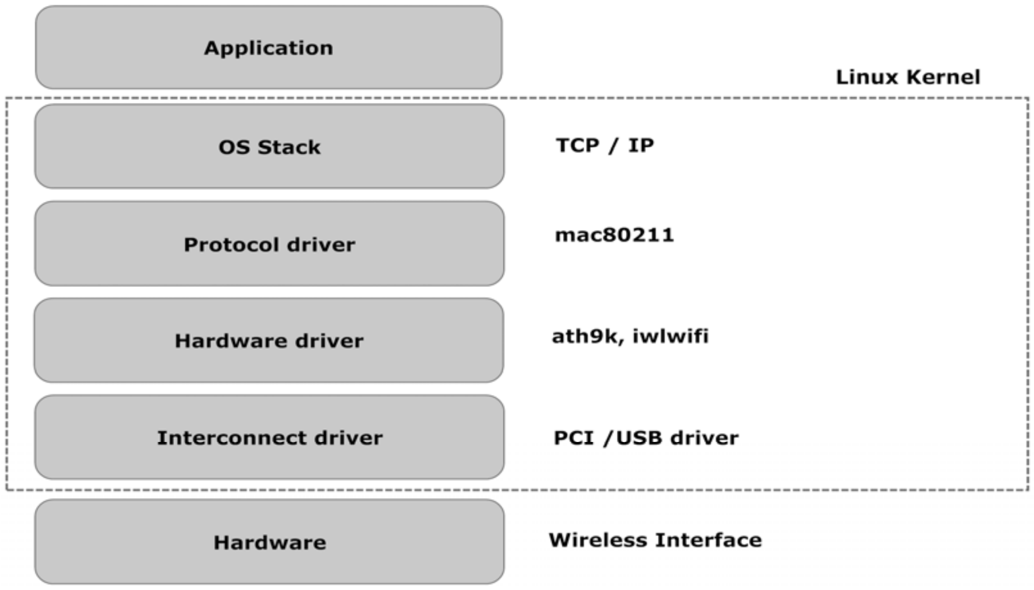

Figure 2.3 Linux kernel stack

In Figure 2.3, the layer diagram of a WLAN driver in the Linux kernel stack and higher layer protocols is shown. WLAN device drivers are divided into two modules (kernel components) namely hardware dependant module and protocol module (softMAC). The hardware dependant modules are different for each vendor and are based on capabilities. The softMAC handles most of the MAC functionality with respect to the IEEE 802.11 protocol. The functions in softMAC are used by hardware drivers amongst different vendor. In the case of WLAN, this softMAC implementation is known as mac80211. The hardware dependent drivers are Atheros ath9k, ath5k (linuxwireless.org 2009) and Intel *iwlwifi* (intellinuxwireless.org 2009) etc. Mac80211 and ath9k are considered in further discussion of WLAN devices in the Linux kernel for understanding.

In order to understand the WLAN driver, it can be partitioned by functional blocks and flow of operation.

The functional blocks can be split as follows:

- Control plane
- SoftMAC
- Hardware driver

And the flow of operation can be analyzed as follows:

- Configuration and management path
- Transmit and Receive path
- Special operations

This chapter introduces the Linux kernel and its general functions. The general network driver functions of kernel module are explained. Implementation of WLAN stack in Linux kernel is explained in more detail and the functional blocks are introduced. The configuration and management path operations are explained with control plane as they are user initiated operations in chapter 3. Transmission path and receive path are explained as a part of the software and hardware dependant driver in chapter 4.

CHAPTER 3

CONTROL PLANE

The control plane is the logical group of functions which are used to manage and configure the WLAN network interface. The WLAN interface supports all basic functions like any other network interface. In addition to that WLAN interface, it provides specific function such as scanning, association and setting specific threshold values (RTS, Fragmentation etc). All these operations are initiated and controlled from the user space. This chapter presents a functional flow of the control plane. The structure of control plane is changed as core WLAN implementation evolved. The present state for controlling WLAN interface is a mixture of new control plane and old wireless extension functions.

3.1 STRUCTURE OF CONTROL PLANE

In general control plane consists of libraries for interacting with the configuration functions of network interface and the control functions in the kernel. For WLAN, the user level interaction functions are wireless extensions (wxt) and netlink library for IEEE 802.11 (nl80211).

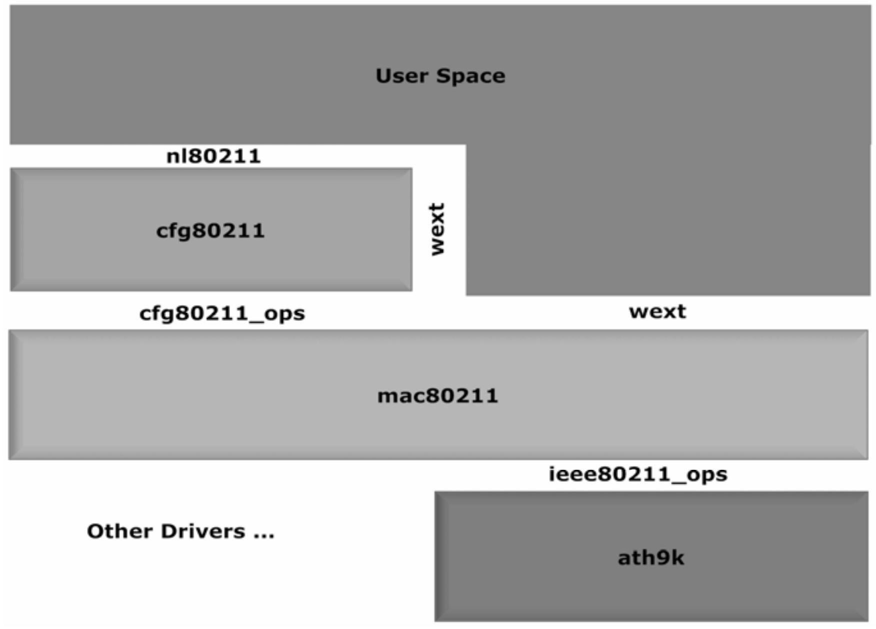

Figure 3.1 Control plane interactions with other layers

Figure 3.1, gives an abstract view of the interaction between the various layers. It gives the modular structure of mac80211 - ath9k WLAN driver. cfg80211 (linuxwireless.org 2009), mac80211 and ath9k are in kernel space and the applications for control and management are in the user space. The user space application makes use of the nl80211 (linuxwireless.org 2009) calls to interact with cfg80211 which in turn communicates with mac80211. The controls are initiated from a user space application and are in turn transferred through system calls.

MADWifi driver uses wireless-extension for all configuration and management functionalities from user level. There are no intermediate layers such as cfg80211. The configurations are made using system calls directly to the network device driver. Nl80211 and cfg80211 clearly define the semantics of commands when compared to wireless-extension.

The user space comprises of a graphical user interface (GUI) and command line network management application through which it controls and configures WLAN devices. This allows configuring physical layer and MAC layer parameters including security. For e.g. NetworkManager (projects.gnome.org 2009), *iw* (linuxwireless.org 2009), wpa_supplicant and hostapd (hostap.epitest.fi 2009). NetworkManager is a general network management GUI application whereas iw, wpa_supplicant and iwconfig are specific WLAN network device command line tools.

These tools use nl80211 header defined system calls to interact with cfg80211, where nl80211 is the new 802.11 netlink functions which is under development. A number of interactions from the user space to mac80211 are carried out through wireless-extensions that are primitive user control functions.

NetworkManager use *wpa_supplicant* (hostap.epitest.fi/wpa_suplicant 2009) internally to achieve wireless functionalities. In the case of *wpa_supplicant,* it uses new nl80211 netlink functions. Hostapd use nl80211 and radiotap functions to implement AP functions. nl80211 exists between the user space and protocol driver (mac80211) and it provide functions to interact with the WLAN device (hostap.epitest.fi/hostapd 2009). These set of functions perform sanity check and protocol translation to configure wireless devices. It provides functions for

- Device registration
- Regularity enforcement
- Station management
- Key management
- Mesh management
- Virtual Interface management
- Scanning

Device registration includes band, channel, bit rate, high throughput (HT) capabilities and supported interface modes. Regularity enforcement will ensure during the registration of cfg80211 that only the specified frequency channels permitted for that given country will be enabled. Station management include add, remove, modify stations and dump station details. These functions are part of AP capabilities.

In mesh path handling, mesh parameter set and retrieve are the functions provided for mesh management. Virtual interface management provides create, remove, change type and monitor flags. It also keeps track of the network wireless interface. Scanning allows user level initialization of scanning and reporting.

3.2 CONFIGURATION AND MANAGEMENT FUNCTION

WLAN interface configuration and management is possible from user level like any other network interface. Most of the wireless interface related management and configuration are selected from the user level.

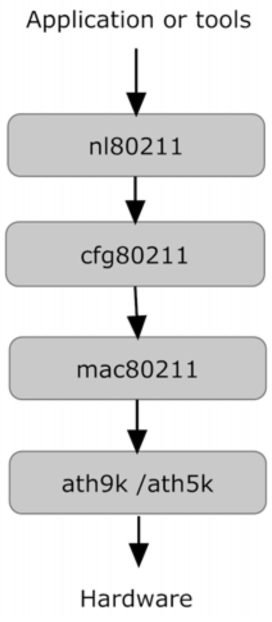

Figure 3.2 Control plane interactions with other layers

Flow from management application to hardware through nl80211, cfg80211, mac80211 and hardware driver is shown in Figure 3.2. Tools use nl80211 function for all types of management and configuration of WLAN devices. Some of these functions are still through wxt as previously discussed. Function calls from higher user layer carry out a set of operations in every layer. In the case of functions for hardware parameter configuration, it takes the parameters directly to hardware modification function than going through mac80211. The functional flow from user level for adding/deleting new virtual interface, scanning, authentication/association and setting up transmission power are used to explain the operation configuration and management functions.

3.2.1 Adding / Deleting an Interface

One of the most common user initiated operation is adding a new interface.

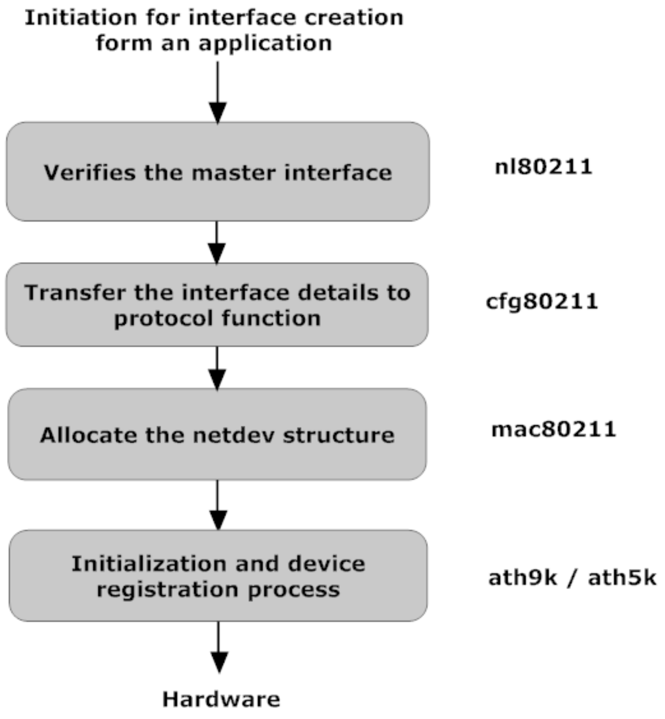
Initiation for interface creation form an application
Verifies the master interface
nl80211
Transfer the interface details to protocol function
cfg80211
Allocate the netdev structure
mac80211
Initialization and device registration process
ath9k / ath5k
Hardware

Figure 3.3 Function flow for interface creation

The functional flow from application to hardware through nl80211, cfg80211 and mac80211 is shown in Figure 3.3. User level tool use nl80211 function (nl80211_new_interface) to initiate adding a new virtual interface. Nl80211 function verifies the master interface and mode of operation and calls cfg80211 function (add_virtual_intf). Mac80211 function (ieee80211_add_iface) allocate the netdev structure, copy hardware address and other hardware related information, sets the operation mode dependant information and allocate buffer for the transmission data. Mac80211 use ath call back function (add_interface) for the hardware dependant functions. Any WLAN driver implementation will have a mapping for this function to equivalent hardware specific function (ath9k_add_interface or ath5k_add_interface). These functions do the necessary initialization and device registration process. It sets the operation mode for the newly created interface, increments the virtual interface count, setup the interrupt, enables

MIB and set the hardware capabilities based on the mode of operation. The Linux kernel functional flow from user level to hardware driver is shown in Figure Appendix 1.1.

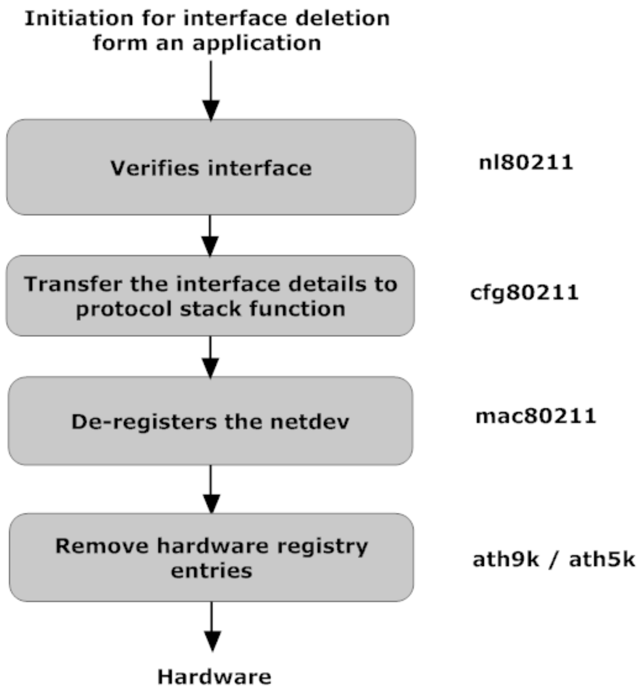
Initiation for interface deletion form an application
Verifies interface
nl80211
Transfer the interface details to protocol stack function
cfg80211
De-registers the netdev
mac80211
Remove hardware registry entries
ath9k / ath5k
Hardware

Figure 3.4 Function flow for interface deletion

Figure 3.4 show the functional flow for deleting the interface. In the case an existing virtual interface is given to delete, user level tools use nl80211 function (nl80211_del_interface). The nl80211 function internally calls cfg80211 function (del_virtual_intf) to interact with mac80211. Mac80211 function (ieee80211_dell_iface) de-registers the netdev structure. Mac80211, use call back function (remove_interface) for the hardware dependant functions (ath9k_remove_interface or ath5k_remove_interface) to remove hardware registry entries. If the interface is in power save, it is reset to normal and then stops all DMA transmission queue and reduces the virtual interface count. Function call flow for deleting interface is shown in Figure Appendix 1.2

3.2.2 Scanning

This user level process is used to identify the available networks joining.

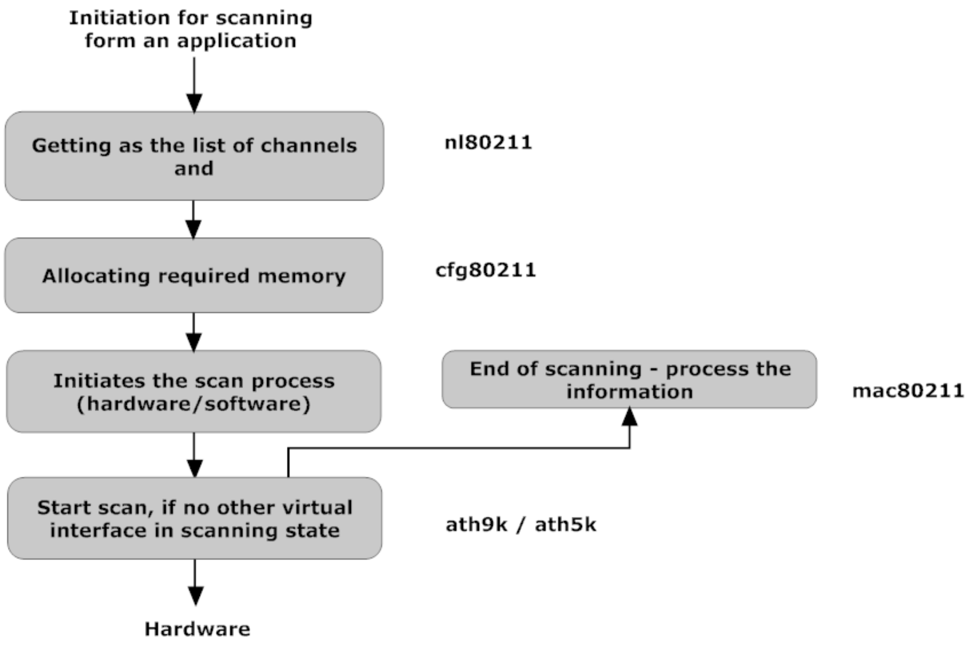
Initiation for scanning form an application
Getting as the list of channels and
nl80211
Allocating required memory
cfg80211
Initiates the scan process (hardware/software)
End of scanning - process the information
mac80211
Start scan, if no other virtual interface in scanning state
ath9k / ath5k
Hardware

Figure 3.5 Function flow for scanning

The flow from application to hardware through nl80211, cfg80211 and mac80211 is shown in Figure 3.5. Scanning is initiated by calling nl80211 function (nl80211_trigger_scan) which carries out primary operations such as getting as the list of channels and allocating required memory for scan result. Nl80211 function calls cfg80211 function (ieee80211_scan) for further operations. The mac80211 function check for any existing scanning operations before continuing with the current requested for scanning. This initiates the scan process, based on the bit set for hardware or software scan. In the case of Atheros, hardware scan is not implemented and software scan is used. The software scanning function (sw_scan_start) is mapped to a hardware driver function (ath9k_sw_scan_start or ath5k_sw_scan_start). In the software scan operation, it checks for the presence any other virtual interface in scanning state. Scan complete function (sw_scan_complete) is called at the end of scanning operation to pass the result to the user layer. Linux kernel functional call flow is shown in Figure Appendix 1.3.

3.2.3 Authentication and Association

This is a two essential step process involved in joining any wireless network.

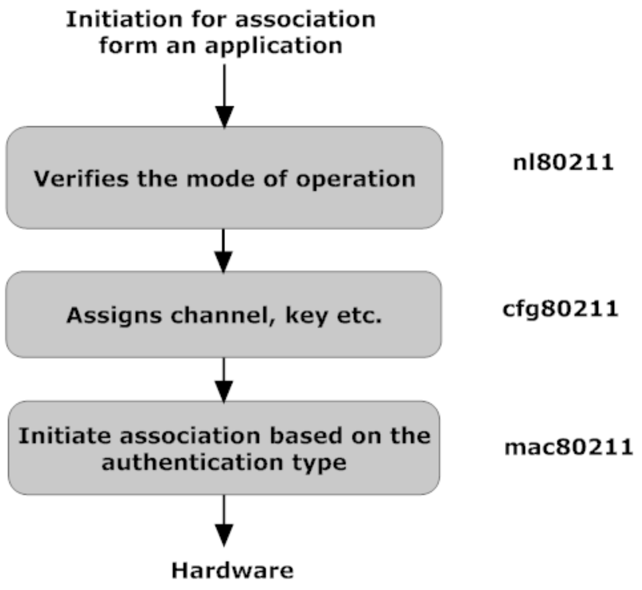
Initiation for association form an application
Verifies the mode of operation
nl80211
Assigns channel, key etc.
cfg80211
Initiate association based on the authentication type
mac80211
Hardware

Figure 3.6 Function flow for association

Functional flow for authentication and association is shown in Figure 3.6. These two processes are a sequence of operations that user level initiates by calling nl80211 functions (nl80211_authenticate and nl80211_associate). Nl80211 function verifies the mode of operation of the interface and parameters before initiating the process by calling cfg80211 functions (cfg80211_mlme_assoc and cfg80211_mlme_auth). Cfg80211 function assigns the necessary parameters such as channel and key before calling mac80211 functions (ieee80211_auth and ieee80211_assoc). Mac80211 functions initiate the association based on the authentication type. The authentication type is dependent on the security system used such as open authentication, WEP and WPA. Once the operation is successful it initiates the process for generating a working queue for the WLAN interface. Functional flow for Linux kernel is shown in Appendix 1.4.

3.2.4 Transmission Power

WLAN netdev allows changing the transmission power from user level.

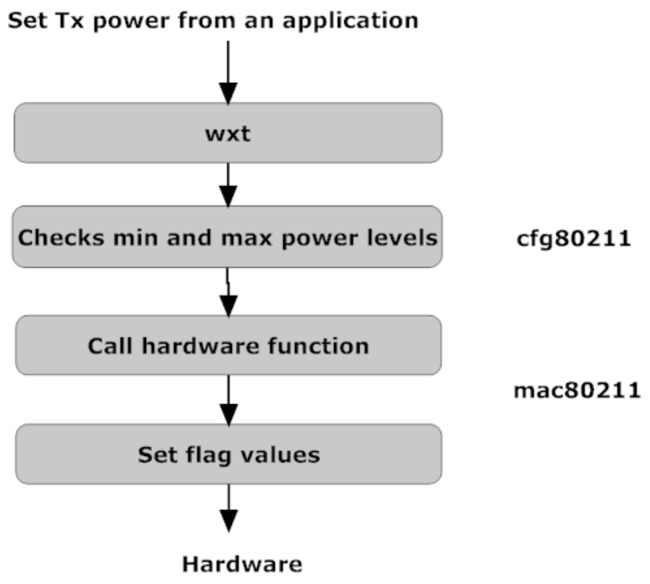

Figure 3.7 Function flow for setting transmission power

The functional flow from application to hardware through nl80211, cfg80211 and mac80211 is shown in Figure 3.7. From the user level, it uses wxt through cfg80211 function (cfg80211_wext_siwtxpower) for setting the transmission power. This cfg80211 function checks for the minimum and maximum power levels and call the mac80211 function (set_tx_power) for further processing. Mac80211 does not perform any functional operation as changing the transmission power is a hardware configuration as discussed earlier. The mac80211 calls the hardware configuration function (ieee80211_hw_config) to modify the transmission power with modified dbm value as its parameter. The hardware configuration function sets the values in appropriate flags to set the new power value through the interconnecting driver. The function flow from user level through kernel is shown in Figure Appendix 1.5.

This chapter discussed about the structure of control plane. Function flows are explained with respect to each layer for adding/deleting an interface, scanning, authentication/association and setting transmit power.

CHAPTER 4

SOFTMAC AND HARDWARE DRIVER

In Linux kernel, SoftMAC and hardware driver are built as separate modules. For WLAN, the SoftMAC module is standard and the hardware dependant driver changes based on the WLAN chipset. The major functional flows through these two layers are transmit and receive paths. In this chapter the functional flow for transmission path, receive path and special operations are explained.

4.1 SOFTMAC

SoftMAC supports IEEE 80211 a/b/d/g /n and s, different types of interfaces and QoS. The types of interfaces include STA, AP, monitor and mesh. Ath9k and Ath5k are the hardware device driver for Atheros IEEE 802.11n based wireless devices, which uses mac80211 as the protocol driver.

4.1.1 Transmission Path

The logical link control (LLC) layer initiates the transmission through by calling hard *xmit* function. The MAC protocol drivers *xmit* function (ieee80211_subif_start_xmit) is called to initiate the transmission.

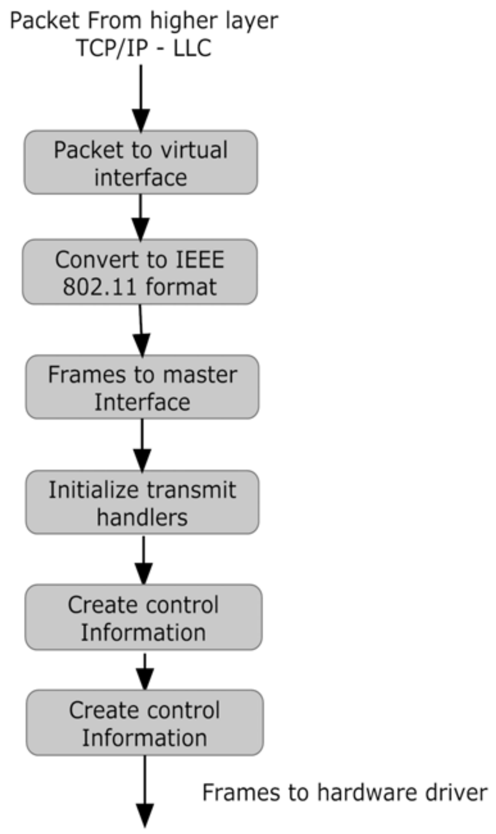
Packet From higher layer
TCP/IP - LLC
Packet to virtual interface
Convert to IEEE 802.11 format
Frames to master Interface
Initialize transmit handlers
Create control Information
Create control Information
Frames to hardware driver

Figure 4.1 Functional flow for transmission in mac80211

Figure 4.1 gives the functional flow of the transmission path of WLAN protocol stacks mac80211 to the hardware driver after calling the *xmit* function. The higher layer packet is converted to IEEE 802.11 frame format and initializes all its required buffers and headers. Higher layer uses the mac80211 function to start the transmission. This function takes device structure and *skbuf* as parameters. The device structure holds the device information and *skbuf* holds the data buffer to send. The higher layer functions are responsible for retries in case of a failure. This function handles in addition the ieee80211 header and sending of packets through the appropriate interface. The sub function checks the minimum length, mode of operation and allocates buffer length for control information based on the frame type. The mapping to the appropriate transmission queue is selected based on the QoS and type of frame in the queue selection function (ieee80211_select_queue). Based on the operating mode it converts the source and destination addresses in the ieee80211 header. This is achieved by copying the address fields to the packet structure based on the operation mode, such as AP, VLAN, mesh, station and ad-hoc. In the case of VLAN, data classifiers are used to determine the 802.1d tag.

Next, the master transmission function (ieee80211_master_start_xmit) checks if the packet generated is for an invalid/non-existing interface and if the type of the interface is unknown or incapable to transmit - then the function frees the packet buffer and returns zero. Next, the power save state is verified for the transmission. If the interface is mesh, then the transmission is based on the next hop. If the interface is in monitor mode, the packet header length is verified for injection. The MAC addresses are not checked here.

Next, the 802.11 packet is prepared for transmission (ieee80211_tx_prepare). The transmit handlers function (invoke_tx_handlers) - selects the key, transmission rate, inserts the sequence number, selects the encryption algorithm, fragmentation, calculates the transmission time and generates control information for transmission.

The select key function (ieee80211_tx_h_select_key) sets the key for different methods and NULL key for no encryption. This function also sets NULL if the frame belongs to management or control.

Function for selecting the rate for transmission (ieee80211_tx_h_rate_ctrl) initializes the required variables for rate control. These structures are used by the rate adaptation algorithms. In the case of multicast frames basic rate is selected. For control frame such as RTS and CTS the basic rate is selected when “rate bit” set in flags or a lesser rate than data rate (less than maximum rate value for control frames) is selected.

The mac80211 function is responsible for adding sequence numbers based on the frame type (ieee80211_tx_h_sequence) to the frame. Based on the hardware flag, this function set the sequence number for management and data frames. If the hardware sequence number flag is set, the protocol driver instructs the hardware to insert the sequence number. For QoS Data, the sequence number is set per STA per traffic id (TID).

The encryption function (ieee80211_tx_h_encrypt) selects the algorithm based on the key configuration flag value. This function use specific routines based on the encryption type, such as WEP, WPA and WPA2. Fragmentation function (ieee80211_tx_h_fragment) updates the duration, sequence and flags in the frame. The duration is calculated (ieee80211_tx_h_calculate_duration) based on transmission rate, type of frame, address used and frequency band. A detailed transmission function flow is shown in Figure Appendix 2.1.

4.1.2 Receive Path

Hardware driver *ath* sends the frame to the protocol driver. The hardware driver passes transmission status info along with frame to mac80211 function (ieee80211_rx). This wrapper function calls packet handler function (ieee80211_rx_handle_packet), which is the actual receive frame handler and carries out further action based on the packet type.

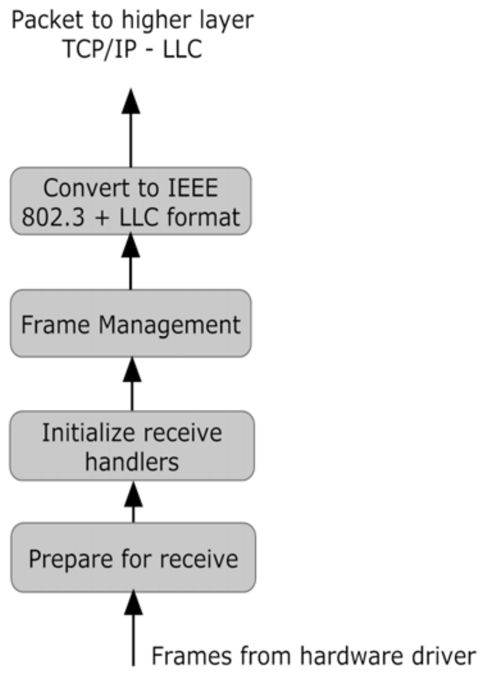
Packet to higher layer
TCP/IP - LLC
Convert to IEEE 802.3 + LLC format
Frame Management
Initialize receive handlers
Prepare for receive
Frames from hardware driver

Figure 4.2 Functional flow for reception in mac80211

In Figure 4.2, the functional flow of frame from the hardware driver to higher layer LLC and TCP/IP is shown. In the receive path mac80211 checks the type of the packet, receive status and prepares the receive handlers. The QOS control fields are processed by a parsing function (ieee80211_parse_qos) and it gets the TID and aggregation details from the further processing of the frame. Receive handler verifies the alignment of the packet for proper processing.

Next, the duplicate frame is checked (ieee80211_rx_h_check). The decryption (ieee80211_rx_h_decrypt) is carried out if the frame is encrypted. Reassembly of the frames for fragmented frames is done by the defragmentation function (ieee80211_reassemble_add). The data frames are converted to IEEE 802.3 with LLC before being sent to higher layer by mac80211 data management functions (ieee80211_data_to_8023). The frames with aggregation, control frame (Block Acknowledgment), next management frame exchange are processed by next frame functions (ieee80211_rx_h_ctrl).

4.2 HARDWARE DRIVER

For better understanding, the whole structure of wireless driver is explained based on its functional flow. The hardware driver manages the transmission/reception of data to the physical hardware through one of the interconnecting driver such as PCI or USB.

4.2.1 Transmission Path

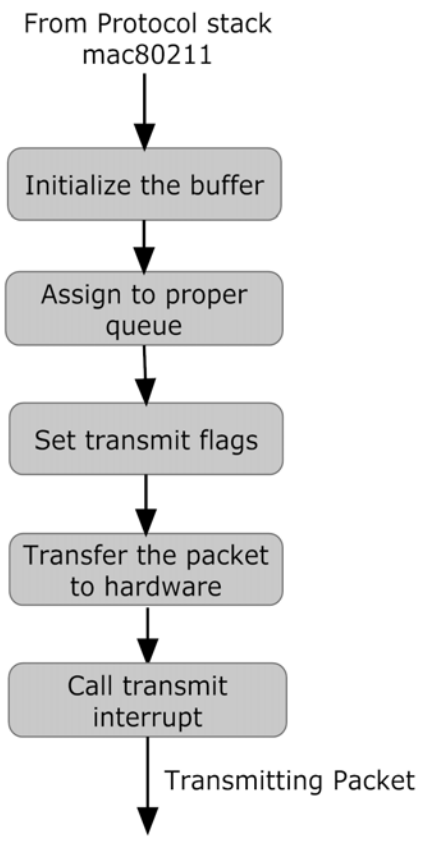

Figure 4.3 Functional flow for transmission in ath9k

Figure 4.3 gives the functional flow of the transmission path. After a packet is received at ath driver from mac80211, it initializes the required buffers. Ath sets up the transmit buffer using function transmit function (ath_tx_setup_buffer) and maps it to the hardware queues. Next, it checks the transmit queue availability and calls ath start transmission function (ath9k_tx). From the type of frame on the buffer (beacon, probe response, ps-poll) the transmission descriptor (transmit flags), physical layer parameters and control information are assigned.

The packets are transferred to the hardware through the interconnection driver using the DMA function (ath_tx_start_dma). Transmit interrupt initiates for transfer of the frame from the hardware and checks the status for reporting and retry.

4.2.2 Receive Path

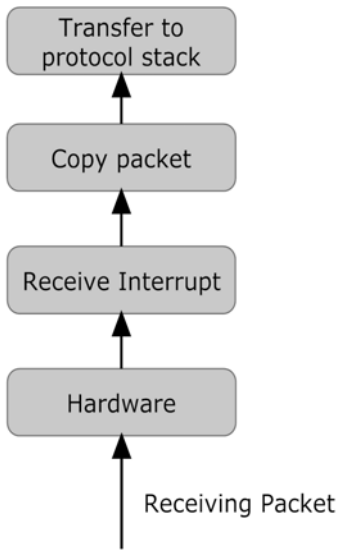

Figure 4.4 Functional flow for reception in ath9k

Figure 4.4 gives the functional flow of reception of packet from hardware to the protocol stack. When a packet is received by the hardware, it generates a receive interrupt. The *ath* function is mapped to receive interrupt of the hardware. The ath tasklet function (ath_rx_tasklet) generates required locks for fetching the packets from hardware and transferring it to protocol stack mac80211 with status information. This function further calls the function (ath_rx_send_to_mac80211) to transfer the frame structure to mac80211.

4.3 SPECIAL OPERATIONS

WLAN open source implementation supports AP, mesh and monitor as special supported operations. In this section the functions which are different from normal station operation is explained.

4.3.1 Monitor Mode

In the monitor mode of operations, the interface does not join any network. This mode is usually used for passive sniffing. The interface receives all packets in its listening channel, even though it may not be destined for it. The protocol driver mac80211 sends upstream, the unaltered IEEE 802.11 MAC packet with certain extra header information. The extra header radiotap includes physical layer information such as received channel, signal quality, signal to noise ratio, antenna and modulation scheme (radiotap.org 2009). Sniffing tools such as Wireshark (wireshark.org 2009) use the Pcap (tcpdump.org 2009) function to transfer these packets to the application layer.

In transmission path the *xmit* function calls the 802.11 frame prepare function based on the mode of operation. In the case of monitor mode, *xmit* function calls the radiotap transmit function (radiotap_tx). This function parses and removes all radiotap headers and initiates sending of the frame in injection mode. In the receive path, the hardware driver generates all required radiotap header information based on the information passed by the hardware.

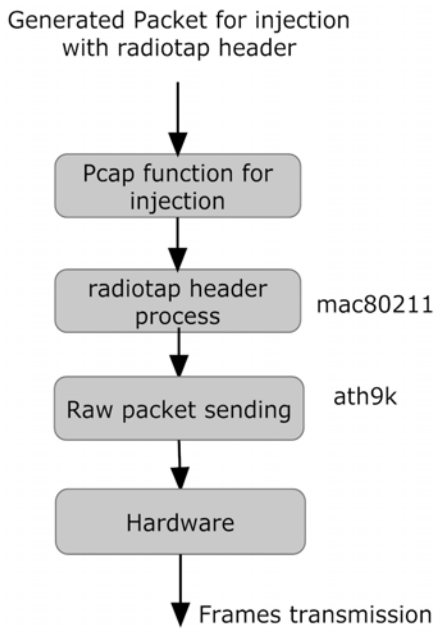

Figure 4.5 Functional flow of packet injection

Monitor mode is used for packet injection. It is possible to inject random IEEE 802.11 MAC frames using the radiotap header and monitor mode WLAN network interface. Figure 4.5 shows the functional flow of packet injection. This is possible by assembling the packet with minimum required radiotap header and sending it to the driver using kernel socket functions. W-meter is one of the open source tool, which is used for arbitrary frame injection.

4.3.2 AP Mode

The mac80211 and hardware drive work as an AP mode with support of HostAP utility. HostAP use monitor mode interface to receive the management frames by using nl80211 function (nl80211_create_iface). The management frames are sent using “frame injection” operation. The AP manages an array of its associated stations inside SAT information elements (sta_info). AP keeps each STA power save transmission buffer queue for unicasts and multicast/broadcasts queue.

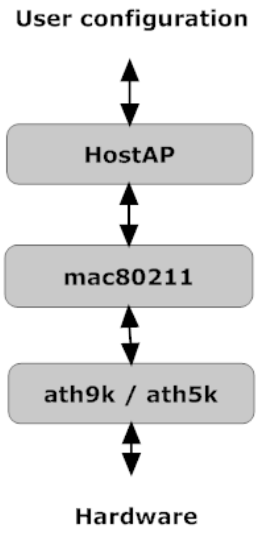

Figure 4.6 HostAP intraction between mac80211 and ath

HostAP uses the net80211 function to control and manage the WLAN interface. The protocol stack and the device driver are set to work in AP mode. This mode enables the HostAP application layer to handle the management frames and AP functionalities. Figure 4.6 shows the interaction between HostAP, mac80211 and ath. Bridge/route utilities and functions of a Linux kernel is used to set up a fully functional AP.

4.3.3 Mesh Mode

Minimum functionalities for a mesh started supporting in mac80211 protocol driver and the device driver from Linux kernel 2.6.26 (open80211s.org). This implements hybrid wireless mesh protocol (HWMP). The mesh allows fixing the next hop (mesh_path_fix_nexthop) of path as similar to IPv4 fixed path. In the mesh mode, when the initial packet is sent to another station, there is first a lookup in the mesh table. If there is no hit a path request (PREQ) is sent (mesh_path_req) as a broadcast. When the PREQ is received on all stations except the final destination, it is forwarded by mac80211 function (ieee80211_rx_h_mesh_fwding). When the PREQ is received on the final station, a path reply (PREP) is sent. If there is a failure on the way, a path error (PERR) is sent (mesh_path_error_tx). The airtime metric is calculated (airtime_link_metric_get) for the possible path and the list is selected.

4.3.4 Debugfs

This is an in-kernel file system designed to help kernel developers easily export debug data to user-space. The debugfs is an interactive system debugger and can be used to examine and change the values of kernel module variables.

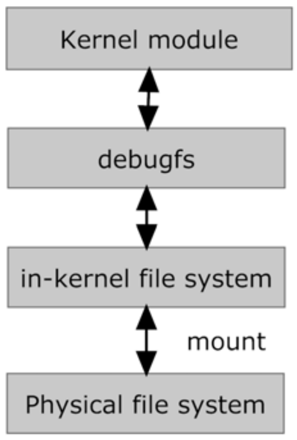

Figure 4.7 Debugfs kernel intraction

Debugfs make use of Kernel functions to create a file in the in-kernel file system and read/write the kernel module values to this file. The in-kernel file system is mounted to physical file system in order to examine and modify the kernel module values. The interaction between kernel module and device driver is shown in Figure 4.7.

The debugfs give information of interface statistics, rate control algorithm, fragmentation, retry and security for mac80211. DEBUGFS flags should be enabled for hardware driver and mac80211 in kernel to support debugfs. The files are created for each physical interface separately. Hardware driver debugfs functions are initiated as interrupt service. The debugfs give DMA, interrupt and rate table details for hardware driver.

This chapter discussed about the structure of SoftMAC and hardware driver. The functional flows for transmission/reception path, special operation such as monitor mode, AP, mesh and debugfs are explained. The step by step operations on the frame for transmission from the LLC to hardware and frame which received in hardware to LLC are explained.

CHAPTER 5

CONCLUSION

Chapter 1 discusses the evolution of IEEE 80211 WLAN standard. The architecture of WLAN is explained with IBSS, infrastructure, WDS and mesh networks. The medium access scheme of the IEEE 802.11 is discussed. Management functions such as scanning, authentication, association, reassociation and disassociation are explained.

Chapter 2 introduces the Linux kernel and its general functions. The general driver functions of kernel module are explained with registration, unloading, opening transmission and reception. The general network driver is explained with its function flow. Implementation of WLAN stack in Linux kernel is explained in detail with the functional blocks such as control plane, SoftMAC and hardware driver. Linux kernel modules like cfg80211, mac80211 and ath9k/ath5k are introduced with their corresponding functional blocks.

Chapter 3 discusses the structure of control plane. The interaction between nl80211, cfg80211, mac80211 and device driver are explained along with its operations. The general user applications and their operation with respect to the control plane layers are discussed. The process and function flows are explained with respect to each layer for adding/deleting an interface, scanning, authentication/association and setting transmit power.

Chapter 4 discusses the process of SoftMAC and hardware driver in different cases. The functional flows for transmission/reception path, special operation such as monitor mode, AP and mesh are explained. The step by step operations on the frame for transmission from the LLC to hardware and frame which received in hardware to LLC are also explained.

In this thesis, we reviewed and discussed the evolution and general implementation of WLAN drivers in the Linux kernel by considering the Atheros driver as a case study. We discussed the functionalities of different components and traced out its important characteristics of the same. This study is aimed to assist the developers in understanding the details of WLAN driver implementation in the Linux kernel.

APPENDIX 1

MANAGEMENT FUNCTIONS

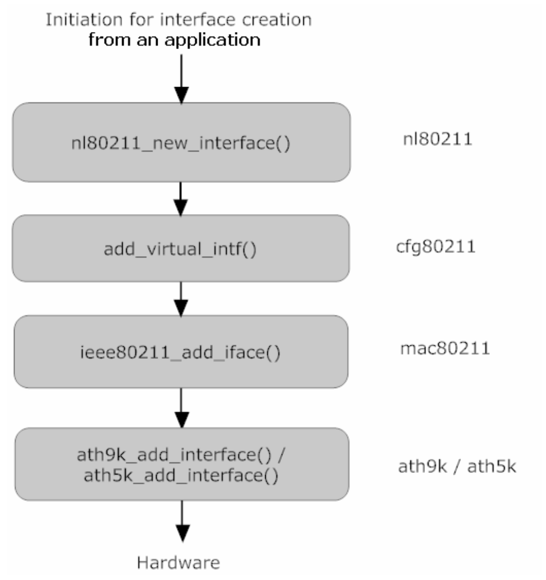

Figure A1.1 Functional calls for creating an interface

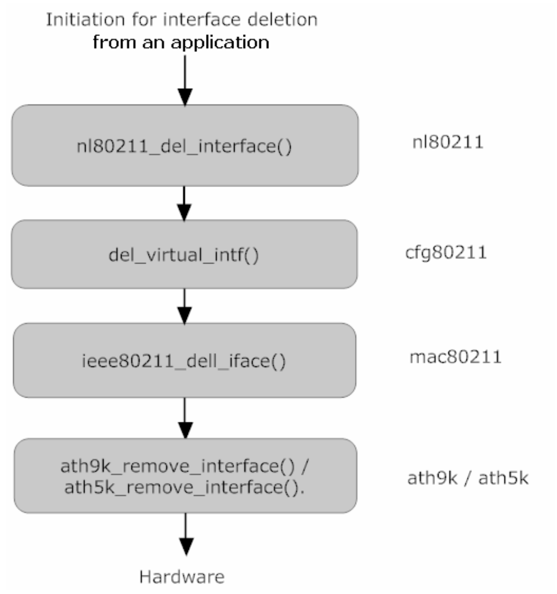

Figure A1.2 Functional calls for deletion of an interface

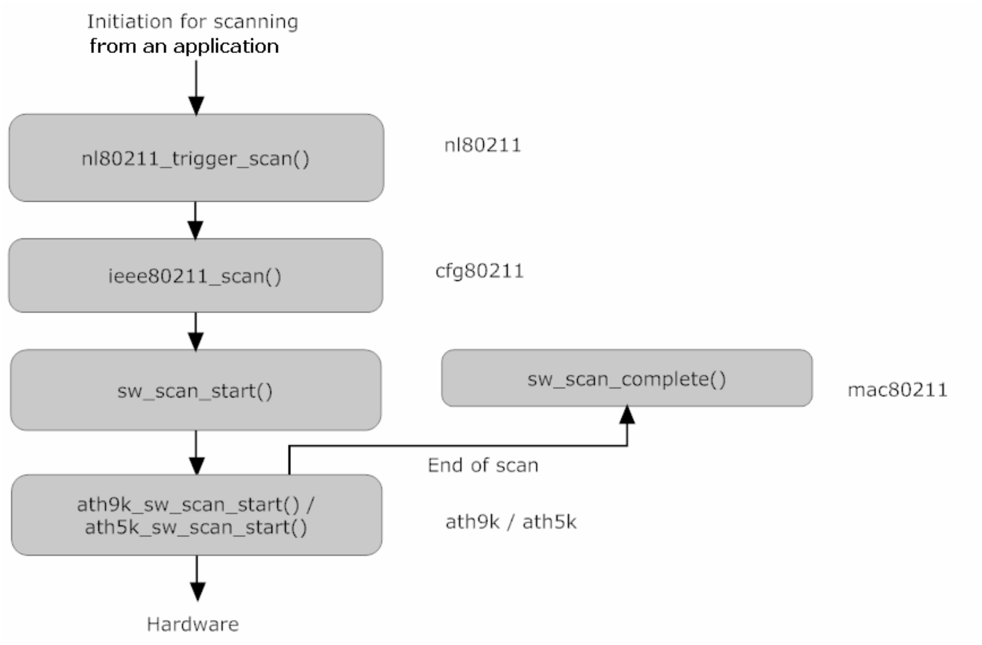

Figure A1.3 Functional calls for scanning process

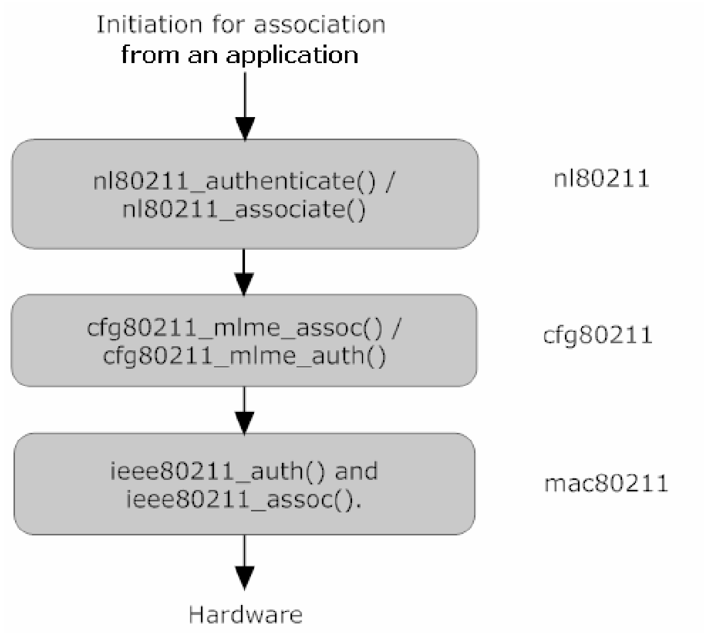

Figure A1.4 Functional calls for authentication and association process

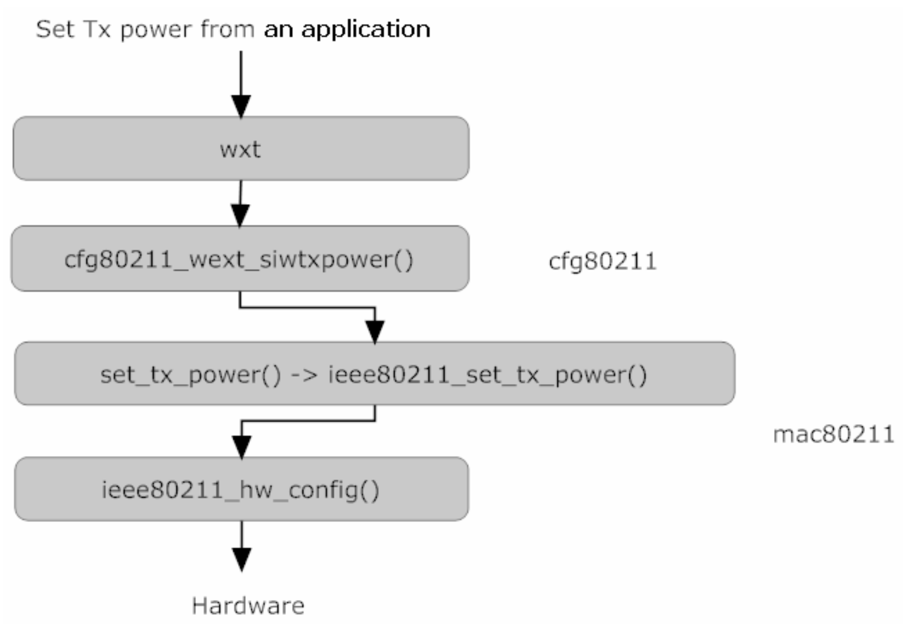
Set Tx power from an application
wxt
cfg80211_wext_siwtxpower()
cfg80211
set_tx_power() -> ieee80211_set_tx_power()
mac80211
ieee80211_hw_config()
Hardware

Figure A1.5 Functional calls for setting transmission power

APPENDIX 1

MAC80211

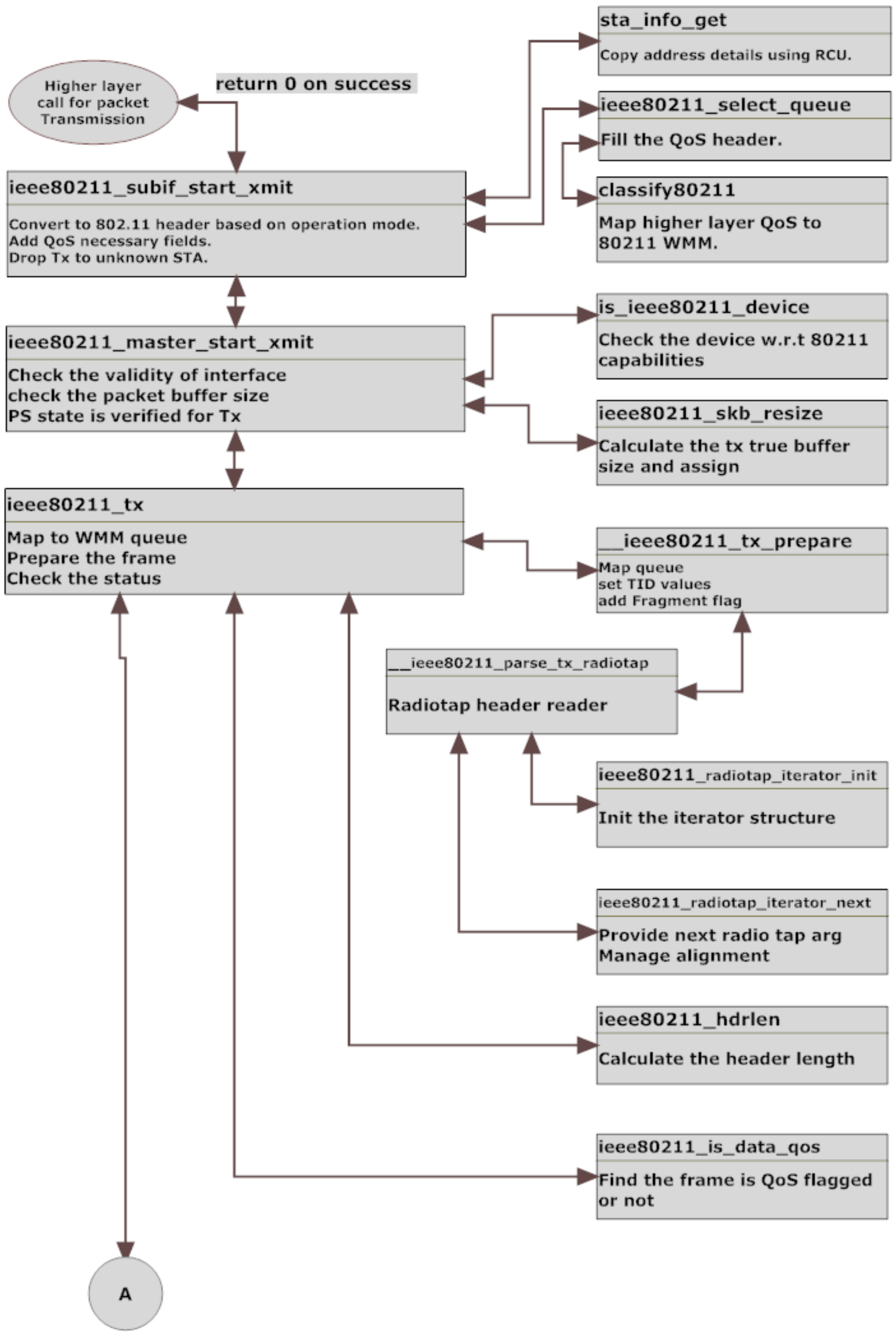
Higher layer call for packet Transmission
return 0 on success
ieee80211_subif_start_xmit
Convert to 802.11 header based on operation mode.
Add QoS necessary fields.
Drop Tx to unknown STA.
sta_info_get
Copy address details using RCU.
ieee80211_select_queue
Fill the QoS header.
classify80211
Map higher layer QoS to 80211 WMM.
ieee80211_master_start_xmit
Check the validity of interface
check the packet buffer size
PS state is verified for Tx
is_ieee80211_device
Check the device w.r.t 80211 capabilities
ieee80211_skb_resize
Calculate the tx true buffer size and assign
ieee80211_tx
Map to WMM queue
Prepare the frame
Check the status
__ieee80211_tx_prepare
Map queue
set TID values
add Fragment flag
__ieee80211_parse_tx_radiotap
Radiotap header reader
ieee80211_radiotap_iterator_init
Init the iterator structure
ieee80211_radiotap_iterator_next
Provide next radio tap arg
Manage alignment
ieee80211_hdrlen
Calculate the header length
ieee80211_is_data_qos
Find the frame is QoS flagged or not
A

Figure A2.1(a) Functional flows in mac80211 for transmission process

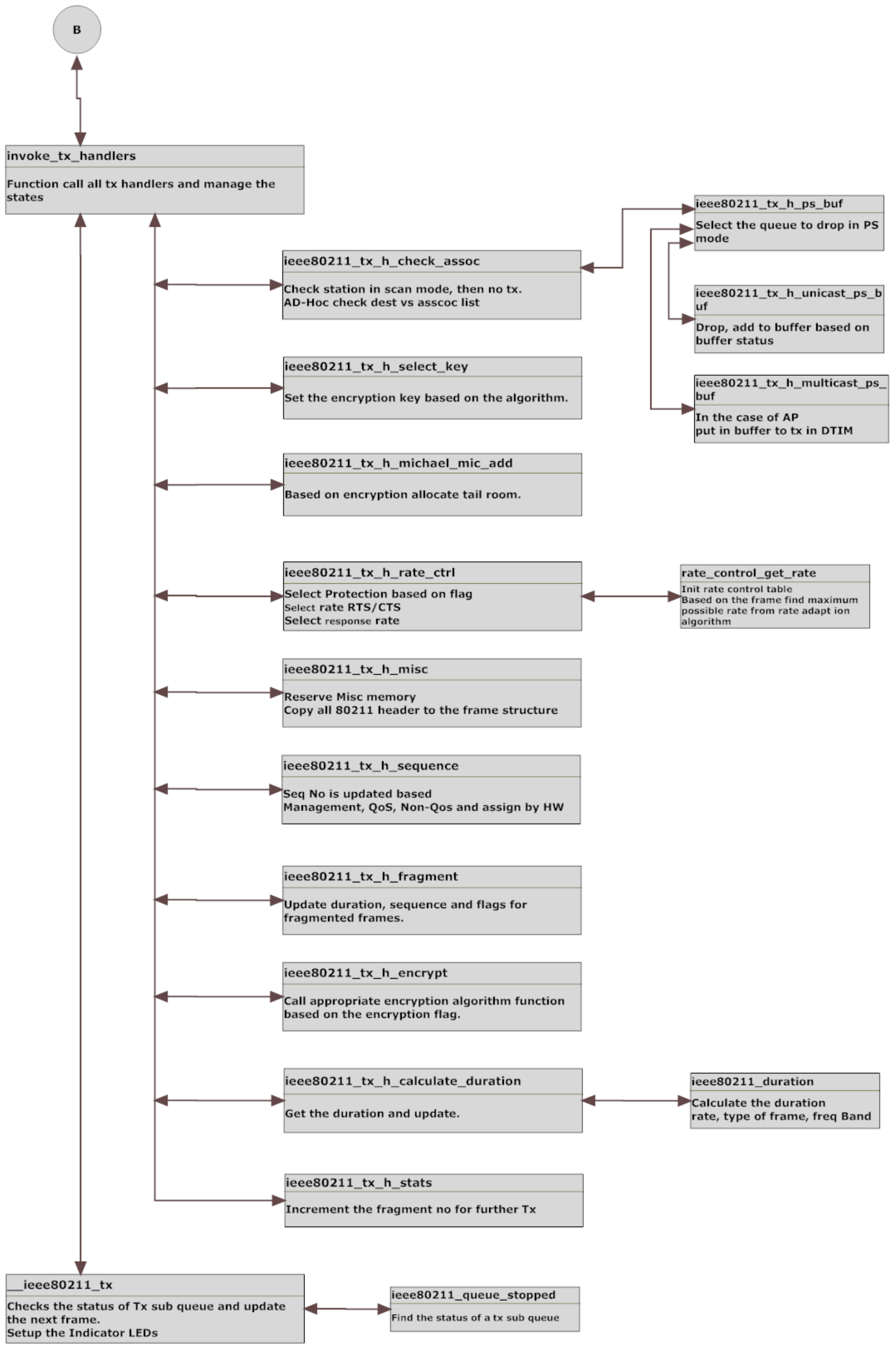
B
invoke_tx_handlers
Function call all tx handlers and manage the states
ieee80211_tx_h_check_assoc
Check station in scan mode, then no tx.
AD-Hoc check dest vs asscoc list
ieee80211_tx_h_ps_buf
Select the queue to drop in PS mode
ieee80211_tx_h_unicast_ps_buf
Drop, add to buffer based on buffer status
ieee80211_tx_h_multicast_ps_buf
In the case of AP
put in buffer to tx in DTIM
ieee80211_tx_h_select_key
Set the encryption key based on the algorithm.
ieee80211_tx_h_michael_mic_add
Based on encryption allocate tail room.
ieee80211_tx_h_rate_ctrl
Select Protection based on flag
Select rate RTS/CTS
Select response rate
rate_control_get_rate
Init rate control table
Based on the frame find maximum possible rate from rate adapt ion algorithm
ieee80211_tx_h_misc
Reserve Misc memory
Copy all 80211 header to the frame structure
ieee80211_tx_h_sequence
Seq No is updated based
Management, QoS, Non-Qos and assign by HW
ieee80211_tx_h_fragment
Update duration, sequence and flags for fragmented frames.
ieee80211_tx_h_encrypt
Call appropriate encryption algorithm function based on the encryption flag.
ieee80211_tx_h_calculate_duration
Get the duration and update.
ieee80211_duration
Calculate the duration
rate, type of frame, freq Band
ieee80211_tx_h_stats
Increment the fragment no for further Tx
__ieee80211_tx
Checks the status of Tx sub queue and update the next frame.
Setup the Indicator LEDs
ieee80211_queue_stopped
Find the status of a tx sub queue

Figure A2.1(b) Functional flows in mac80211 for transmission process

REFERENCES

1 IEEE 802.11 - 1999, Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE.

2 IEEE 802.11 - 2007 (Revision of IEEE Std 802.11-1999), Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE.

3 Matthew Gast (2005), 802.11 Wireless Networks: The Definitive Guide, Second Edition. O'Reilly Media.

4 http://madwifi-project.org/wiki/About retrieved on 09/15/09.

5 Klaus Wehrle (2003), The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel, Prentice Hall.

6 Jonathan Corbet (2005), Linux Device Drivers, 3rd ed, O'Reilly.

7 http://madwifi-project.org/wiki/DevDocs retrieved on 09/15/09.

8 http://linuxwireless.org/en/developers/Documentation/mac80211 retrieved on 09/15/09.

9 git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-testing.git retrieved on 09/15/09.

10 http://linuxwireless.org/en/users/Drivers/ath5k retrieved on 09/15/09.

11 http://linuxwireless.org/en/users/Drivers/ath9k retrieved on 09/15/09.

12 http://intellinuxwireless.org/?p=iwlwifi retrieved on 09/15/09.

13 http://linuxwireless.org/en/developers/Documentation/cfg80211 retrieved on 09/15/09.

14 http://linuxwireless.org/en/developers/Documentation/nl80211 retrieved on 09/15/09.

15 http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html retrieved on 09/15/09.

16 http://projects.gnome.org/NetworkManager retrieved on 09/15/09.

17 http://linuxwireless.org/en/users/Documentation/iw retrieved on 09/15/09.

18 http://hostap.epitest.fi/wpa_supplicant/ retrieved on 09/15/09.

19 http://hostap.epitest.fi/hostapd/ retrieved on 09/15/09.

20 http://www.radiotap.org/ retrieved on 09/15/09.

21 http://www.wireshark.org/ retrieved on 09/15/09.

22 http://www.tcpdump.org/pcap3_man.html retrieved on 09/15/09.

23 http://sourceforge.net/apps/trac/w-meter/ retrieved on 09/15/09.

24 http://kerneltrap.org/node/4394 retrieved on 09/15/09.

LIST OF PUBLICATION

1 **Vipin M,** Srikanth S. (2010), ‘Analysis of Open Source Drivers for IEEE 802.11 WLANs’ International Conference on Wireless Communication and Sensor Computing 2010. pp 66-70.