Analysis of open source drivers for IEEE 802.11 WLANs

Conference Paper · February 2010

DOI: 10.1109/ICWCSC.2010.5415877 · Source: IEEE Xplore

CITATIONS
42

READS
6,213

2 authors:

Vipin M.
RPTU - Rheinland-Pfälzische Technische Universität Kaiserslautern Landau
5 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Srimanthula Srikanth
Pragati Engineering College
27 PUBLICATIONS **358** CITATIONS
SEE PROFILE

Preprint of an article that appeared in IEEE conference proceeding of ICWCSC 2010

Analysis of Open Source Drivers for IEEE 802.11 WLANs

Vipin M
AU-KBC Research Centre
MIT campus of Anna University
Chennai, India
vipintm@au-kbc.org

Srikanth S
AU-KBC Research Centre
MIT campus of Anna University
Chennai, India
srikanth@au-kbc.org

***Abstract*—The purpose of this study is to analyze the open source IEEE 802.11 wireless local area network (WLAN) stack implementation for further enhancement and implementations. We discuss the IEEE 802.11 WLAN implementation in the latest Linux kernel. This includes a functional breakdown of the driver and the overall flow of information via functions. We also survey the specific implementation methods used in the WLAN Linux stack. We 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.**

***Keywords*— Open source, Linux kernel, Network stack, IEEE802.11 WLAN**

I. INTRODUCTION

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 [1] [2]. 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.

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, wireless distribution system (WDS), virtual access point (VAP), virtual interface and monitor [3]. Peer to peer connections are made in ad-hoc mode. In this mode, each WLAN device communicates directly without any central coordinator. In infrastructure mode, an access point (AP) is the coordinator for every client. All traffic passes through the AP. The mesh mode is an extension of ad-hoc mode. The salient feature of this mode is multi hop transmission. WDS has a bridging similar to Ethernet and also provides repeater functionalities. VAP is a multiple virtual AP with 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 standard for infrastructure and ad-hoc is IEEE 802.11a/g/n, for mesh is IEEE 802.11s and for security is IEEE 802.11i.

II. LINUX KERNEL

Linux kernel is an open source operating system kernel. The Linux kernel has a modular architecture. Most of the drivers for the peripheral hardware are built as loadable modules. They are loaded when new hardware is attached or at boot time. When any module is loaded, they export their functions to the kernel space.

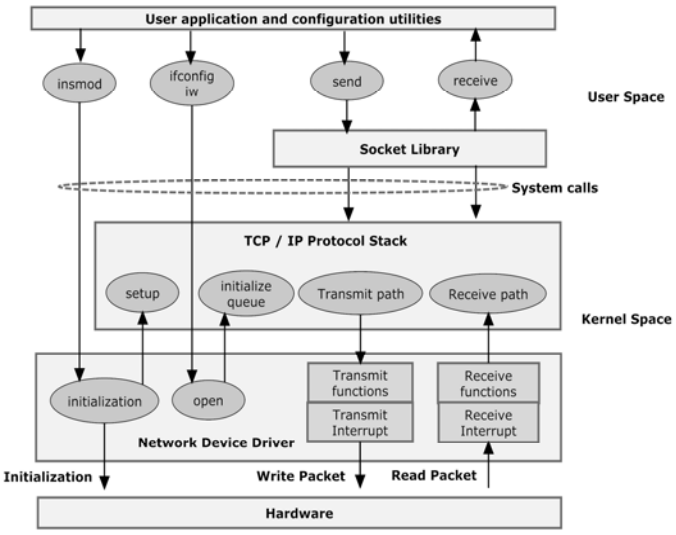

Figure 1. Linux kernel interaction

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 Fig. 1. The operations explained in the figure can be categorized as configuration/management, transmission path, and receive path [4].

Insmod is a user level function that loads any kernel module [5]. 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 transmit interrupt for physical transmission from hardware by 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.

Sections III describes evaluation of open source WLAN driver. Section IV, V and VI describes its major functionality namely control plane, SoftMAC, hardware driver. Sections VII discuss monitor mode and debugfs.

III. EVALUTION OF OPEN SOURCE DRIVERS

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 like PCI or USB. Wireless netdev is similar to an Ethernet interface for higher layer with extra features such as ad-hoc. 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 medium access control (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 didn't have the 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 more access to code. The two major IEEE 802.11 WLAN SoftMAC implementations are net80211 [6] and mac80211 [7].

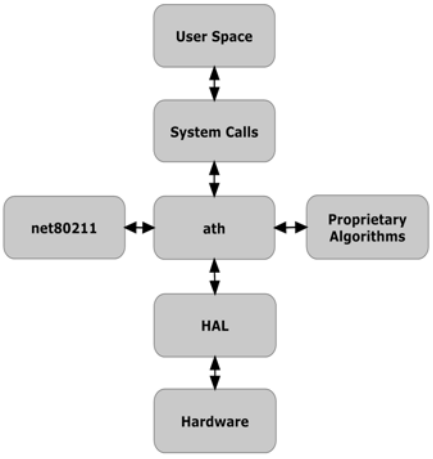

Figure 2. MADWiFi Structure

The original open source Atheros driver is developed for FreeBSD. MADWiFi is a modified version of it. MADWiFi drivers work with binary and open hardware abstraction layer (HAL) and net80211. MADWiFi driver is under dual license BSD and GPL v2 [3].

Fig. 2 shows the MADWiFi structure. net80211, ath, HAL and proprietary algorithms are in kernel space [6]. 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 frequency band and 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 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. The ath module 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 are implemented as loadable modules.

The new implementation of WLAN network device is as a part of Linux kernel source tree [8]. This implementation is based on SoftMAC.

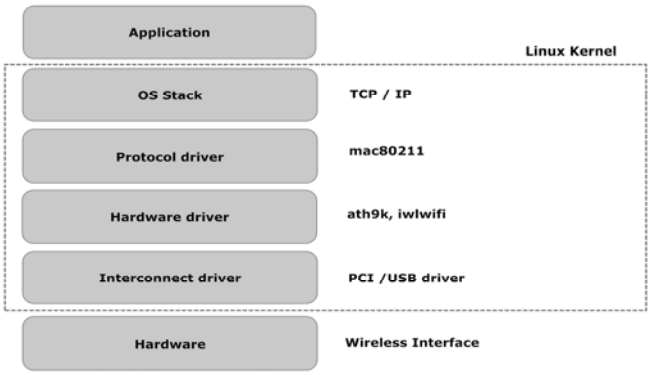

Figure 3. Linux kernel stack

In Fig. 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 based on capabilities. The softMAC handles most of the MAC functionality with respect to 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 [9], ath5k [10] and Intel iwlwifi [11]. Mac80211 and ath9k are considered in further discussion of WLAN devices in the Linux kernel.

In order to understand the flow of the WLAN driver and its structure, its functionality can be divided in to 3 parts.

- Control plane
- SoftMAC
- Hardware driver

IV. CONTROL PLANE

In WLANs, scanning, association and setting specific threshold values (RTS, Fragmentation etc) are initiated and controlled from the user space.

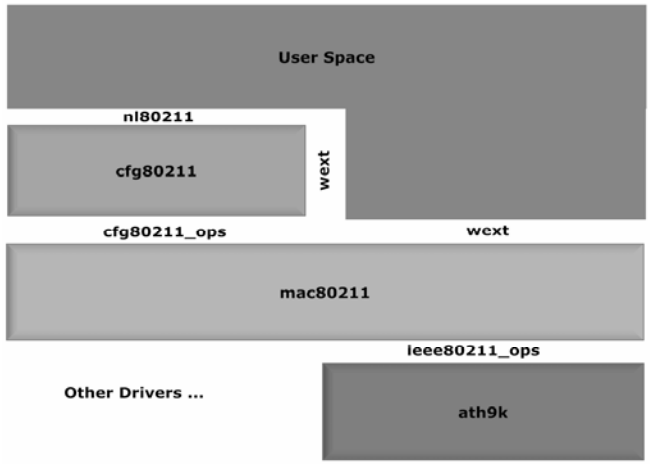

Figure 4. Structure of mac80211 – ath9k

In Fig. 4, the modular structure of mac80211 - ath9k WLAN driver is shown. It gives an abstracted view of the interaction between the various layers. cfg80211 [12], 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 [13] 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 (wext) for all configuration and management functionalities from user level [14]. 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 defined the semantics of commands than wireless-extension. Nl80211 is a complete redesigned of wireless settings flow.

A. *User Space*

The user space comprises of 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 [15], iw [16], wpa_supplicant [17] and hostapd [18]. NetworkManager is a general network management GUI application whereas iw, wpa_supplicant and hostapd are specific wireless command line tools.

These tools use nl80211 header defined system calls to interact with cfg80211, where nl80211 is 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 (wext) that are primitive user control functions.

NetworkManager use wpa_supplicant 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.

B. *Cfg80211*

This layer exists between the user space and protocol driver (mac80211). These set of APIs 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.

V. SALIENT FUNCTIONALITIES OF 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. It handles the following protocol functionalities:

- Transmission Path
- Receive Path

A. *Transmission Path*

The higher layer transfers the packet structure to the MAC by calling kernel public transmission function.

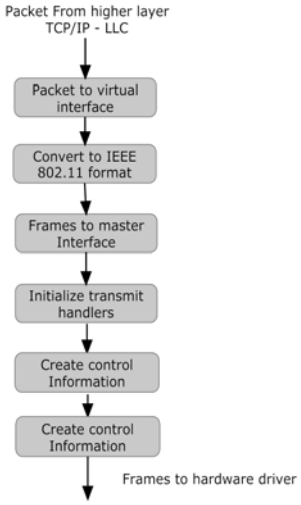

Figure 5. Functional flow for transmission in mac80211

In Fig. 5, the functional flow of the transmission path of WLAN protocol stacks mac80211 is shown. The higher layer packet will be converted to IEEE 802.11 frame format and initializes all its required buffers and headers. The transmit handlers - selects the key, transmission rate, inserts the sequence number (based on the hardware capability), selects the encryption algorithm, fragmentation, calculates the transmission time and generates control information for transmission.

B. *Receive Path*

Hardware driver ath sends the frame to the protocol driver mac80211 along with the hardware receive status information.

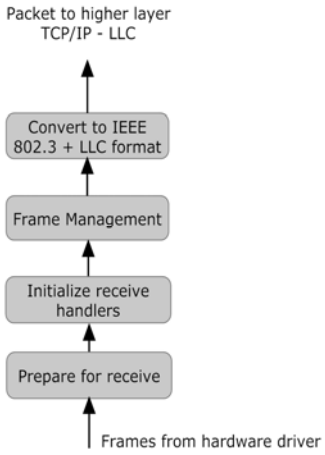

Figure 6. Functional flow for reseption in mac80211

In Fig. 6, the functional flow of frame from the hardware driver to higher layer logical link control (LLC) and TCP/IP is shown. In the receive path: mac80211 checks the type of the packet, receive status and prepares the receive handlers. Receive handler verifies the alignment of the packet for proper processing, decryption and defragmentation. The frames with aggregation, control frame (Block Acknowledgment), next management frame exchange are processed. The IEEE 802.11 frames are converted to IEEE 802.3 + LLC and sent to the higher layer.

VI. SALIENT FUNCTIONALITIES OF HARDWARE DRIVER

Ath9k is the device driver for Atheros IEEE 802.11n based wireless devices. It uses mac80211 as the protocol driver. For better understanding, the whole structure of wireless driver is explained based on its functional flow.

- Transmission Path
- Receive Path

A. *Transmission Path*

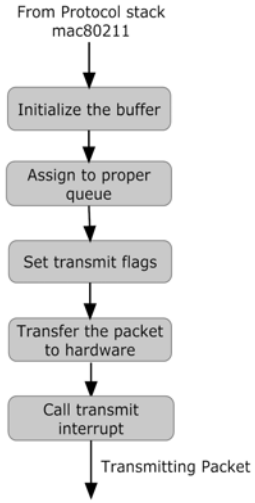

Figure 7. Functional flow for transmission in ath9k

In Fig. 7, the functional flow of the transmission path is shown. Since a packet is received at ath driver from mac80211, initialize the required buffers and maps it to the hardware queues. Transmit flags are assigned depending on the type of the packet, physical layer parameters and control information. The packets are transferred to the hardware through the interconnection driver using DMA. Transmit interrupt initiates for the transfer of the frame from the hardware and checks the status for reporting and retry.

B. *Receive Path*

In Fig. 8 the functional flow of reception of packet from hardware to the protocol stack is shown. When a packet is received by the hardware, it generates a receive interrupt. The ath function which is mapped to receive interrupt will generate required locks for fetching the packets from hardware and transferring it to protocol stack mac80211 with status information.

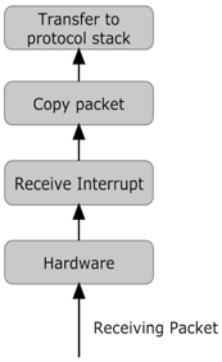

Figure 8. Functional flow for reception in ath9k

VII. Other Functions

A. Monitor mode

In monitor mode of operations, the interface does not join to any network. This mode is generally 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 [19]. Sniffing tools such as Wireshark [20] use Pcap [21] function to get these packets to the application layer.

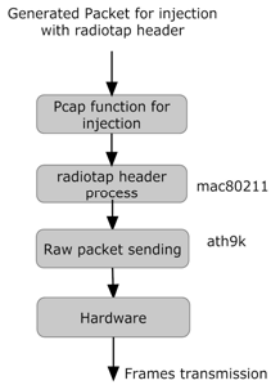

Figure 9. Functional flow of packet injection

The other purpose of monitor mode is packet injection. It is possible to inject random IEEE 802.11 MAC frames using the radiotap header and monitor mode WLAN network interface. Fig. 9 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 [22].

B. 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 [23].

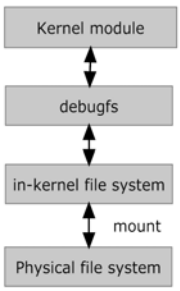

Figure 10. debugfs working model

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.

VIII. Conclusion

In this survey, we discussed the evolution and general implementation of WLAN drivers in the Linux kernel by considering the Atheros driver as an example. We discussed the functionalities of different components and traced out the important characteristics of the same. This study the will help the developers understand the details of driver implementation in the Linux kernel.

References

[1] http://standards.ieee.org/getieee802/ retrieved on 09/15/09.
[2] http://grouper.ieee.org/groups/802/11/ retrieved on 09/15/09.
[3] http://madwifi-project.org/wiki/About retrieved on 09/15/09.
[4] Klaus Wehrle, The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel, Prentice Hall, 2003.
[5] Jonathan Corbet, Linux Device Drivers, 3rd ed, O'Reilly, 2005.
[6] http://madwifi-project.org/wiki/DevDocs retrieved on 09/15/09.
[7] http://linuxwireless.org/en/developers/Documentation/mac80211 retrieved on 09/15/09.
[8] git://git.kernel.org/pub/scm/linux/kernel/git/linville/wireless-testing.git retrieved on 09/15/09.
[9] http://linuxwireless.org/en/users/Drivers/ath5k retrieved on 09/15/09.
[10] http://linuxwireless.org/en/users/Drivers/ath9k retrieved on 09/15/09.
[11] http://intellinuxwireless.org/?p=iwlwifi retrieved on 09/15/09.
[12] http://linuxwireless.org/en/developers/Documentation/cfg80211 retrieved on 09/15/09.
[13] http://linuxwireless.org/en/developers/Documentation/nl80211 retrieved on 09/15/09.
[14] http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html retrieved on 09/15/09.
[15] http://projects.gnome.org/NetworkManager retrieved on 09/15/09.
[16] http://linuxwireless.org/en/users/Documentation/iw retrieved on 09/15/09.
[17] http://hostap.epitest.fi/wpa_supplicant/ retrieved on 09/15/09.
[18] http://hostap.epitest.fi/hostapd/ retrieved on 09/15/09.
[19] http://www.radiotap.org/ retrieved on 09/15/09.
[20] http://www.wireshark.org/ retrieved on 09/15/09.
[21] http://www.tcpdump.org/pcap3_man.html retrieved on 09/15/09.
[22] http://sourceforge.net/apps/trac/w-meter/ retrieved on 09/15/09.
[23] http://kerneltrap.org/node/4394 retrieved on 09/15/09.