Tracing the Execution Path in Mac80211

Pooja Parasuraman, Jonas Andre*, Stephan Günther*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: pooja.parasuraman@tum.de, andre@net.in.tum.de, guenther@tum.de

Abstract—**The purpose of this study is to trace the execution paths of a wireless packet, traversing within the Mac80211 subsystem. Eventhough the world progresses towards building high speed wireless networks by optimizing link and routing costs, the end system's processing capability remains a bottleneck in limiting the efficiency of networks. As first step in optimising per-packet processing at wireless endpoints, we analyse the existing architecture of the Mac80211 subsystem. We discuss the transmission path of IEEE802.11 packets with a focus on managed mode of operation. This paper presents the structure and design of the Linux Kernel and the functions through which IEEE802.11 packets traverse. An experiment is perfomed to extract real-time trace of packets using explicit kernel logs for comparing the results obtained from manual tracing of Linux source code.**

Index Terms—**Linux Kernel, Mac80211, Wireless Driver, IEEE802.11 WLAN**

1. Introduction

One of the most widely used protocols in the family of IEEE 802 Local Area Network standards is the wireless LAN protocol (`IEEE 802.11` [1]). `IEEE 802.11` standard defines the link layer and physical layer protocols for communication between wireless devices. This standard needs to be followed by all driver developers in order to facilitate interoperability.

The Linux operating system provides a generic framework for wireless device drivers called the Mac80211. It is a subsystem that interfaces between the kernel and the device driver for various functionalities with respect to the wireless network packets that pass through the subsystem. An `IEEE 802.11` wireless network interface card (NIC) can operate in one or many modes [2] as discussed below.
Master : NICs in Master mode act as Access Points (AP) and follow a hierarchy of operation. A connection to another wireless NIC is possible only if the latter operates in Managed mode. This mode can also be termed as AP mode or Infrastructure mode.
Managed : Managed mode NICs act as clients (also termed as slaves) and associates to an already created wireless network by a master card. Managed mode is the counter-part of Master mode. There is a strict master-slave hierarchy and hence a client NIC can only communicate with its own master. It is not possible for two client cards to interact between themselves directly.

There are other less commonly used modes in which a NIC can be configured. In Monitor mode, NICs can sniff all radio traffic on a particular channel for wireless network debugging and analysis. Promiscuous mode is similar to Monitor mode but the difference is the former mandates an association with an AP (active sniffing) while the latter supports passive sniffing. Ad-hoc mode is used in peer-to-peer networks. Mesh mode combines ad-hoc mode and routing. A NIC in Repeater mode extends the existing wireless networks for longer range of access. In Tunnelled Direct Link Setup (TDLS) mode, a direct secure fast path for data transfer between communicating peers is made possible in a hierarchical network. This facilitates faster media streaming and other data transfers.

This study is based on manual tracing using the open-source Linux Kernel source code with focus on Managed mode. Section 2 talks about the related research work performed in this area. Section 3 provides an overview of the Linux Kernel architecture with respect to the Wireless network stack. Section 4 talks about Mac80211 subsystem in detail. A special mode available in Mac80211 subsytem of Linux Kernel - the Fast Xmit mode - is discussed in Section 5. Finally, Section 6 explains the experiment conducted to trace `IEEE802.11` packets traversing the Mac80211 subsytem.

2. Related work

Vipin et al. analyses the implementation of `IEEE802.11` network stack in the Linux Kernel and its interaction with open source device drivers [3]. Lisovey et al. discuss about the feasibility of including a module in the Linux Kernel to enable wireless communication for vehicular environment. Vitalik et al. proposes a design for portable and pluggable mode for the Mac80211 subsystem with add-on features such as co-operative retransmission support [4]. All these papers, analyse the wireless network stack in the Linux Kernel and discuss whether the design of Mac80211 framework can be optimised and enhanced.

The work from multiple analysis, implementations and architectural papers have been used as a base for this paper in order to understand the architecture of the Mac80211 subsystem.

3. Linux Kernel and Mac80211

The Linux Kernel network stack is designed in a modular way with a clear separation between multiple entities. Figure 1 depicts how the wireless network stack is layered in the Linux Kernel and how interaction between the layers takes place. The core understanding of this architecture is obtained from [5] and [6].

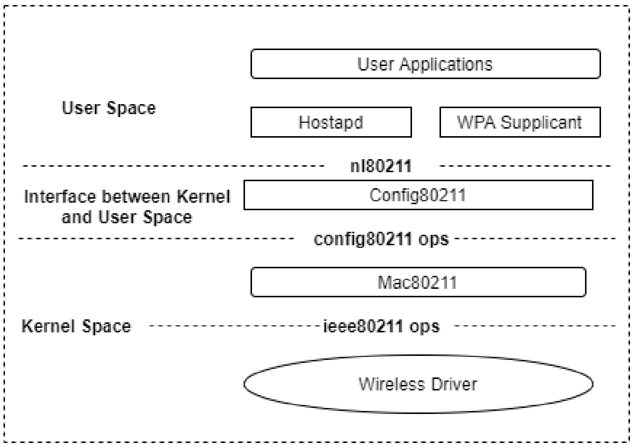


Figure 1: Kernel Wireless Stack

In order to better understand the architecture, we analyse each component using a top-down approach. The user space is composed of applications, such as `Hostapd` and `WPA supplicant`, that use wireless as their underlying technology. `Hostapd` facilitates configuration and control of Access Points, whereas `WPA supplicant` is a control interface to manage wireless clients. User space applications interact with the Linux Kernel using config80211 subsystem [7]. Config80211 is a configuration API acting as a bridge between user space applications and the underlying driver via the Mac80211 subsystem. Interaction between user space and the config80211 happens via the 802.11 netlink interface called nl80211 [8].

The Mac80211 subsystem [9] is a generic framework provided by the Linux Kernel. Wireless drivers use the Mac80211 framework to register callbacks for all packet processing functionalities. These are available in kernel space and they interact using config80211 ops and ieee80211 ops [8], which are the wireless configuration operations. Every wireless interface has its own set of configuration operations which is called from user space.

4. Mac80211 TX Path

This section provides a detailed schema of the Mac80211 subsystem [9] with respect to managed mode. Understanding of the information in this section is obtained by manual tracing backed by the experiment discussed in Section 6.

Figure 2 shows the path (functions) taken by a wireless packet from user space to the driver. These functions correspond to handling packets when the sending interface operates in managed mode.

A packet is originated by a user space application and sent to the kernel. The packet then passes through various OSI layers [10] of the kernel stack to add layer specific information. Packets are carried within a data structure called skb (socket buffer) [11] and are passed onto every layer in the kernel.

All the relevant information required for the higher layer OSI headers [10] is added to the skb by the Linux Kernel. The kernel then hands the skb to the Mac80211 subsystem along with the information of the interface via which the packet must be sent (out-

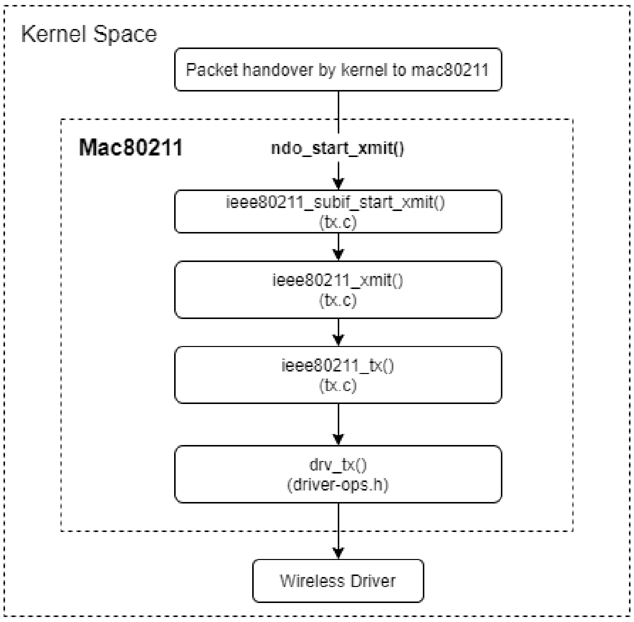


Figure 2: Mac80211 Control Flow

bound). This is achieved by a registered callback function (ndo_start_xmit()).

The corresponding registered function for ndo_start_xmit() with respect to managed mode is ieee80211_subif_start_xmit(). This function is responsible for the actual MAC layer processing and adding MAC and PHY information into the skb. After the required information is filled in, it is passed onto the ieee80211_xmit() function which handles the adjustment of skb headroom and sets the QoS header, if required. The skb is then passed onto the ieee80211_tx() function. Every interface has its own transmission (TX) queue. In order to transmit the skb into the correct interface, corresponding transmission queue of the outbound interface is selected. The skb is put into the outbound interface's TX queue. The skb will be dequeued and sent to the final point in the Mac80211 subsystem (drv_tx()) by a kernel tasklet named ieee80211_tx_pending. At the end of drv_tx() the control is transferred from the Mac80211 subsystem to the wireless driver.

ieee802111_subif_start_xmit

The majority of the MAC layer processing happens within the ieee80211_subif_start_xmit() function. Figure 3 shows the detailed flow diagram of control points in this function. The Mac80211 subsystem maintains a hash table with a list of known stations connected to the local device. The initial step is to retrieve the target station (STA) node from the hash using the target address from the skb data structure. In managed mode, various parameters must be verified to retrieve the target station node. It is checked if the target station is a TDLS peer to the local device. If it is a TDLS peer, then more checks are done to verify if the target station is an authenticated peer to send or receive data to/from the local device. If all the above conditions are satisfied, the station node is retrieved with target mac address as the key to the hash table. If the STA node is found successfully, the skb is attempted to be sent via a

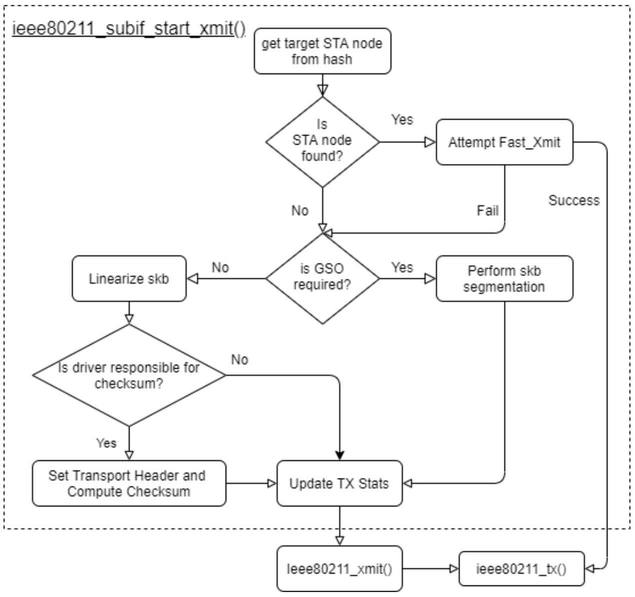


Figure 3: Flow Diagram of ieee80211_subif_start_xmit function

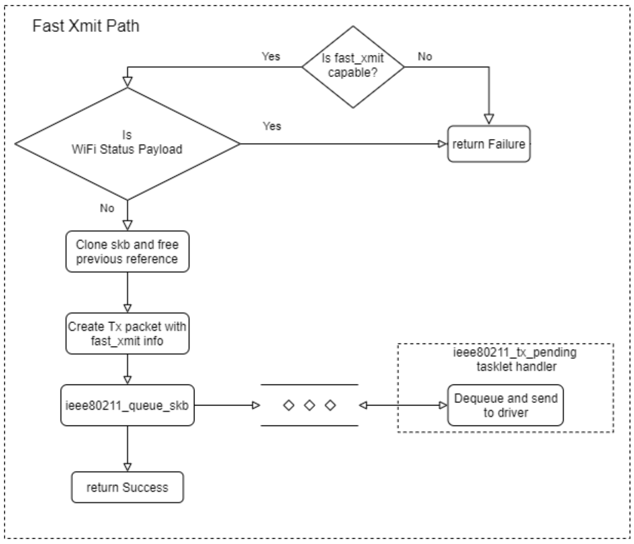
relatively faster execution path, called the fast_xmit path [?]. This is explained in detail in Section 5.

If the station node is not found in the hash table or the skb could not be sent via the fast_xmit path, then the skb takes regular slow path where detailed checks are made before sending the packet out through the interface. In this path, checks to verify if the skb must undergo Generic Segmentation Offloading (GSO) [12] is performed. GSO is a software-based technique to perform packet segmentation which is offloaded by wireless cards to drivers. If a skb is subject to GSO, then it will be segmented into multiple skbs based on the MSS segment size provided by the wireless card using gso_size config parameter. The segments will then be transmitted onto the interface.

If a skb does not require GSO, the skb must undergo linearization which is a process in which, a paged skb is converted to a linear skb [11]. A paged skb is used when the data that needs to be sent is larger than the MSS. One predominant use case of a paged skb is when a file system file content needs to be sent over a socket. Once, the skb is linearized, it is checked if the wireless card has offloaded the checksum [13] operation to the driver. The flag CHECKSUM_PARTIAL specifies if checksum needs to be verified in the software. If the flag is set, then the transport header offset is adjusted such that there is enough space for the checksum to be inserted. The checksum is calculated and copied into the skb after the header size is modified. With this, all the necessary processing is done and the TX statistics are updated for TX packet count and TX byte count for the interface. The skb is then sent to the ieee80211_xmit() function which is explained above.

5. Fast Transmit Path

This section discusses one of the interesting features of the Mac80211 subsystem - Fast Transmit Path [14]. The understanding of information provided in this section is based on manual code analysis with the open-source Linux distribution [8]. The Fast Transmit feature requires support from hardware. Figure 4 shows the list of functions a packet takes in Fast Transmit mode.



Fast Xmit Path
Is fast_xmit capable?
Yes
No
Is WiFi Status Payload
Yes
return Failure
No
Clone skb and free previous reference
Create Tx packet with fast_xmit info
ieee80211_queue_skb
ieee80211_tx_pending tasklet handler
Dequeue and send to driver
return Success

Figure 4: Flow Diagram of Fast Transmit Path

Fast Transmit is used as an optimization technique so that packets do not need to go through a long list of checks as discussed in Section 4. Before a packet is sent for further processing, the function ieee80211_check_fast_xmit() checks if the target station and the local device's underlying wireless card provide Fast Transmit support in their hardware. If Fast Transmit is possible, further checks to determine if the target station is an authorized peer are done to proceed with Fast Transmit. Once these checks are passed successfully, the 802.11 header and other information for packet processing is cached inside the STA node data structure, which is required during TX packet processing. This function is called whenever a new station is added or any state change happens at the station. Explicit calls to this function must be made to reset the information in case the fast_xmit path is no longer applicable for the station.

During TX packet processing, the STA node retrived from hash table is verified for the fast_xmit information. If the information exists (not NULL), then the skb is capable to be sent via the fast_xmit path, else the skb follows the regular path as discussed in Section 4. Once the skb enters the fast_xmit path, the protocol of the packet is checked. If the skb is a Wi-Fi Status message, then the skb is sent to be processed via the regular path, else it is further processed as a Fast Transmit packet.

The final packet is constructed using the previously cached 802.11 header information. Once the packet is ready to be sent out of the interface, a TX queue slot for the outbound interface is fetched and the packet is put into the queue. As discussed earlier, the ieee80211_tx_pending tasklet takes care of dequeuing the skb and passing it onto the driver.

6. Packet Tracing with Kernel Logs

We performed an experiment to trace `IEEE802.11` packets passing through the Mac80211 subsystem in the

`Linux Kernel`. Figure 5 shows the setup used for the experiment. The setup includes Raspberry Pi 4 and an external RT5572 Wireless adapter. The Raspberry Pi 4 is configured as an ethernet backhaul extender. A backhaul is an interface through which a device can connect to another existing network. This means that the Raspberry Pi 4 is now acting as a bridged wireless access point within our already existing local Ethernet network. The external RT5572 Wireless adapter is attached to the Raspberry Pi 4 and is considered as the Access Point in our experiment. The backhaul interface of the Raspberry Pi 4 is connected to a D-Link Router. We use an Android mobile phone as our client device which is connected to the RT5572 wireless adapter.

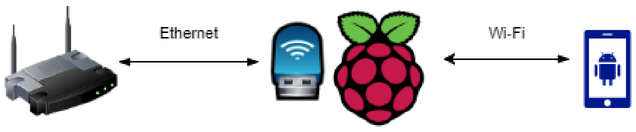
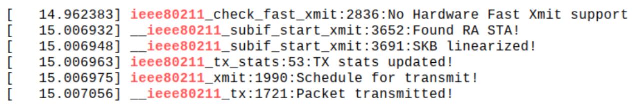


Figure 5: Experimental Setup

Explicit kernel logs were added to the open source Mac80211 subsytem source code to help us trace the packets within the Linux Kernel. We sent ICMP echo messages [15] from the Raspberry Pi to the Android phone. Figure 6 shows a sample trace captured during the experiment. The trace shows list of functions traversed by a single `IEEE802.11` TX packet from the Raspberry Pi 4.



```
[   14.962383] ieee80211_check_fast_xmit:2836:No Hardware Fast Xmit support
[   15.006932] __ieee80211_subif_start_xmit:3652:Found RA STA!
[   15.006948] __ieee80211_subif_start_xmit:3691:SKB linearized!
[   15.006963] ieee80211_tx_stats:53:TX stats updated!
[   15.006975] ieee80211_xmit:1990:Schedule for transmit!
[   15.007056] __ieee80211_tx:1721:Packet transmitted!
```

Figure 6: Packet Trace

As first step discussed in Section 4, the kernel hands over the TX packet to mac80211 subsytem. Then check to find if the target station is available in the cached memory (hash table) is done. The kernel log "Found RA STA" denotes that the station is already known to the local device and its information is available in the hash table. Now, the Mac80211 subsystem checks if the packet can take Fast Transmit path. Both the RT5572 adapter and the client device did not exhibit support for Fast Transmit. Hence the kernel log "No Hardware Fast Xmit support" appears in our trace. Hence, The Tx packet follows the regular slow path.

The next step is to check if the hardware has offloaded segmentation to the driver. The RT5572 adapter takes care of segmentation in the hardware and hence GSO path is not triggered. Alternately, the skb undergoes linearization. We introduced kernel logs to find if the Mac80211 subsystem is responsible for handling checksum. Since those logs did not appear in our packet trace, it is understood that the hardware takes care of calculating checksum before sending the packet into the network. In the final step of processing, the TX statistics are updated for the TX packet which is seen from the log "TX stats updated". From this point, the underlying IEEE802.11 device driver takes care of sending the packet to the target station.

The prepared skb is enqueued in the TX queue of the outbound interface by the device driver. After this, the tasket handler discussed in Section 4 dequeues the skb and transmits the packet onto the interface which is seen in the packet trace. With this the entire lifetime of a packet within the Mac80211 subsystem is traced.

7. Conclusion and Future Work

This paper provides a detailed walkthrough of the Mac80211 subsystem and its architecture focussing on packet processing in managed mode of operation. This papers also analyses the fast_xmit mode of transmission which helps in drastically reducing the per-packet processing overhead. An experiment to back the information obtained from manual tracing is performed and the complete packet trace is presented. Future work can include finding possible areas of optimizations and analysing the remaining features of the mac80211 subsystem in order to create a simpler and a pluggable version of the mac80211 subsystem. Furthermore, the `IEEE802.11` driver can be examined and a detailed study can be made on how packets traverse through wireless drivers.

References

[1] G. Hiertz, T. Denteneer, L. Stibor, Y. Zang, X. Costa-Pérez, and B. Walke, "The ieee 802.11 universe," *Communications Magazine, IEEE*, vol. 48, pp. 62 – 70, 02 2010.

[2] "Monitor mode," https://en.wikipedia.org/wiki/Monitor_mode, [Online: accessed 06-June-2021].

[3] M. Vipin and S. Srikanth, "Analysis of open source drivers for ieee 802.11 wlans," in *2010 International Conference on Wireless Communication and Sensor Computing (ICWCSC)*, 2010, pp. 1–5.

[4] V. Nikolyenko and L. Libman, "Coop80211: Implementation and evaluation of a softmac-based linux kernel module for cooperative retransmission," in *2011 IEEE Wireless Communications and Networking Conference*, 2011, pp. 239–244.

[5] D. C. Mur, "Linux wi-fi open source drivers," http://www.campsmur.cat/files/mac80211_intro.pdf, [Online: accessed 06-June-2021].

[6] J. M. Berg, "Mac80211 overview," https://wireless.wiki.kernel.org/_media/en/developers/documentation/mac80211.pdf, 2009, [Online: accessed 06-June-2021].

[7] "Linux 802.11 driver developer's guide," https://www.kernel.org/doc/html/v4.12/driver-api/80211/index.html, [Online: accessed 06-June-2021].

[8] "Linux kernel developer documentation," https://wireless.wiki.kernel.org/en/developers/documentation, [Online: accessed 06-June-2021].

[9] P. Salvador, S. Paris, C. Pisa, P. Patras, Y. Grunenberger, X. Perez-Costa, and J. Gozdecki, "A modular, flexible and virtualizable framework for ieee 802.11," in *2012 Future Network Mobile Summit (FutureNetw)*, 2012, pp. 1–8.

[10] J. Day and H. Zimmermann, "The osi reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.

[11] "How skbs work," http://vger.kernel.org/~davem/skb_data.html, [Online: accessed 06-June-2021].

[12] "Segmentation offloads in the linux networking stack," https://www.kernel.org/doc/Documentation/networking/segmentation-offloads.txt, [Online: accessed 06-June-2021].

[13] T. C. Maxino and P. J. Koopman, "The effectiveness of checksums for embedded control networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 1, pp. 59–72, 2009.

[14] "Fast transmit," https://elixir.bootlin.com/linux/latest/source/net/mac80211/tx.c#L2908, [Online: accessed 06-June-2021].

[15] "Internet Control Message Protocol," RFC 792, Sep. 1981. [Online]. Available: https://rfc-editor.org/rfc/rfc792.txt