

QSDK - AP ARCHITECTURE

Premier Electronics Engineering & Manufacturing Company
Creating Opportunities & Transforming Lives

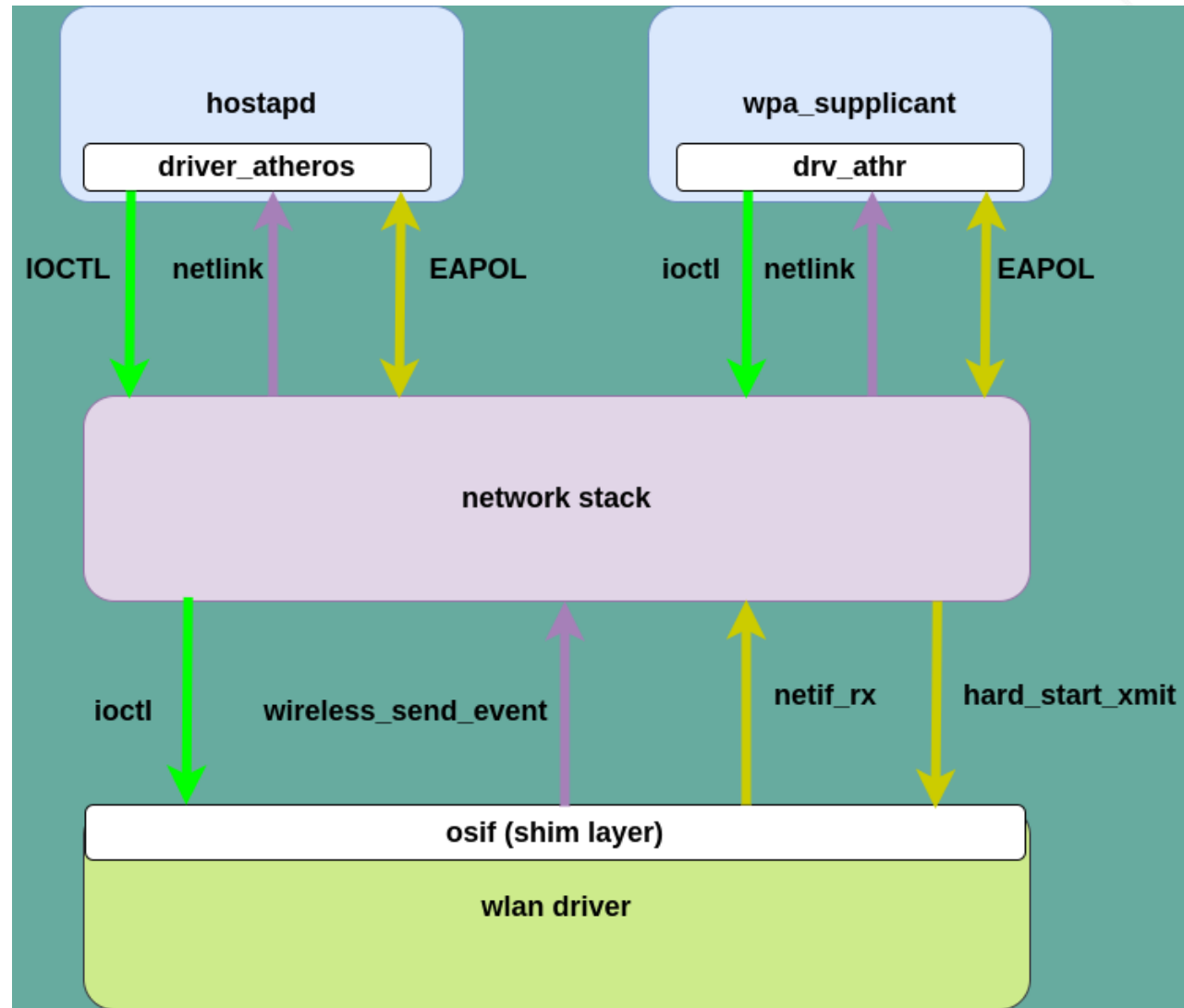
3- dec-20

WLAN AP SOFTWARE CLASSIFICATION

- Wireless Driver - resides in kernel space (qca).
- WLAN Applications (hostapd, wpa-supPLICANT, wifi script, iwconfig, iwpriv, uci etc).
- FIRMWARE - wifi core functionality mainly data path.

The AP software as whole have lot of other components - kernel, bridge, network stack, ethernet driver and whole lot of packages in order to provide various functionalities offered by the AP software. They may reside in kernel space or user space.

Different AP model available in QSDK - This classification of AP models is based on the way the wlan software is placed or we can say how the different WLAN core functionality are splitted across the host or the target platform.

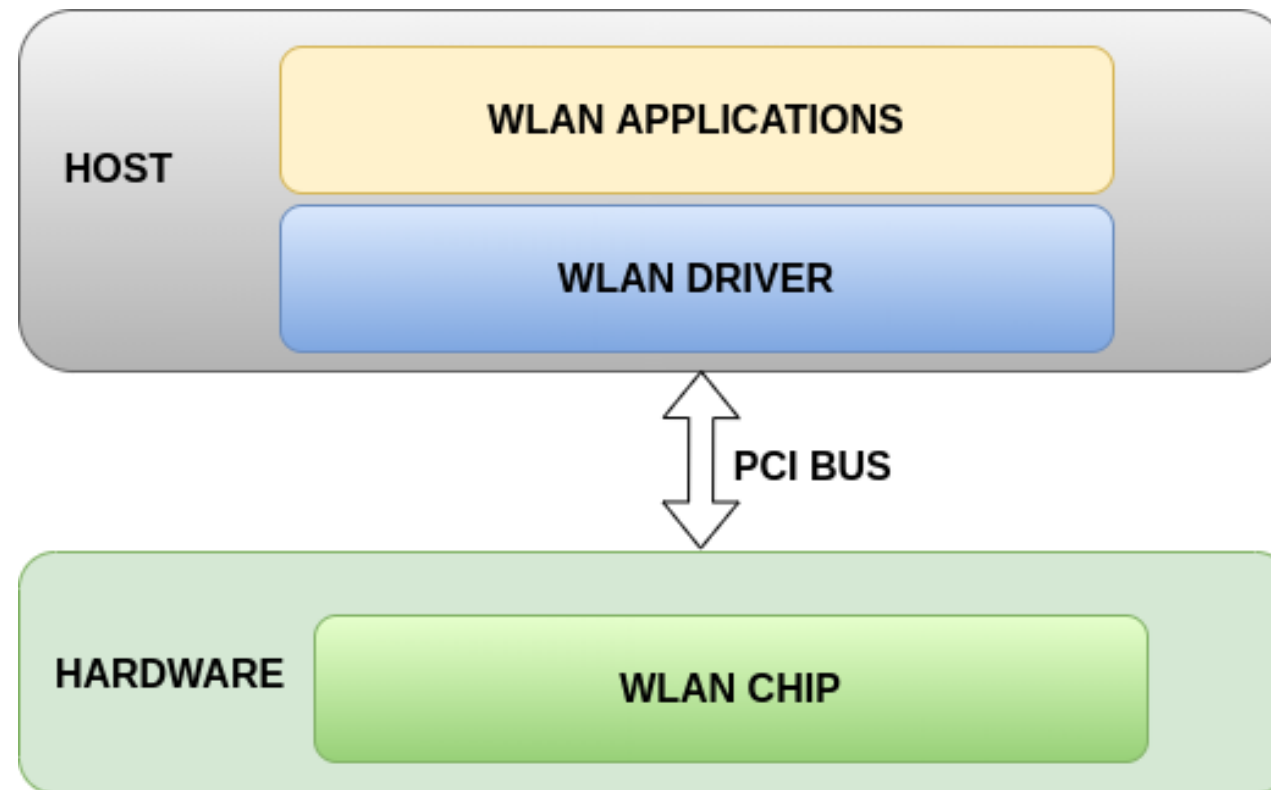


WLAN Driver interfacing with hostapd and the wpa-supPLICANT

Different feature set or Functionality performed by the WLAN driver.



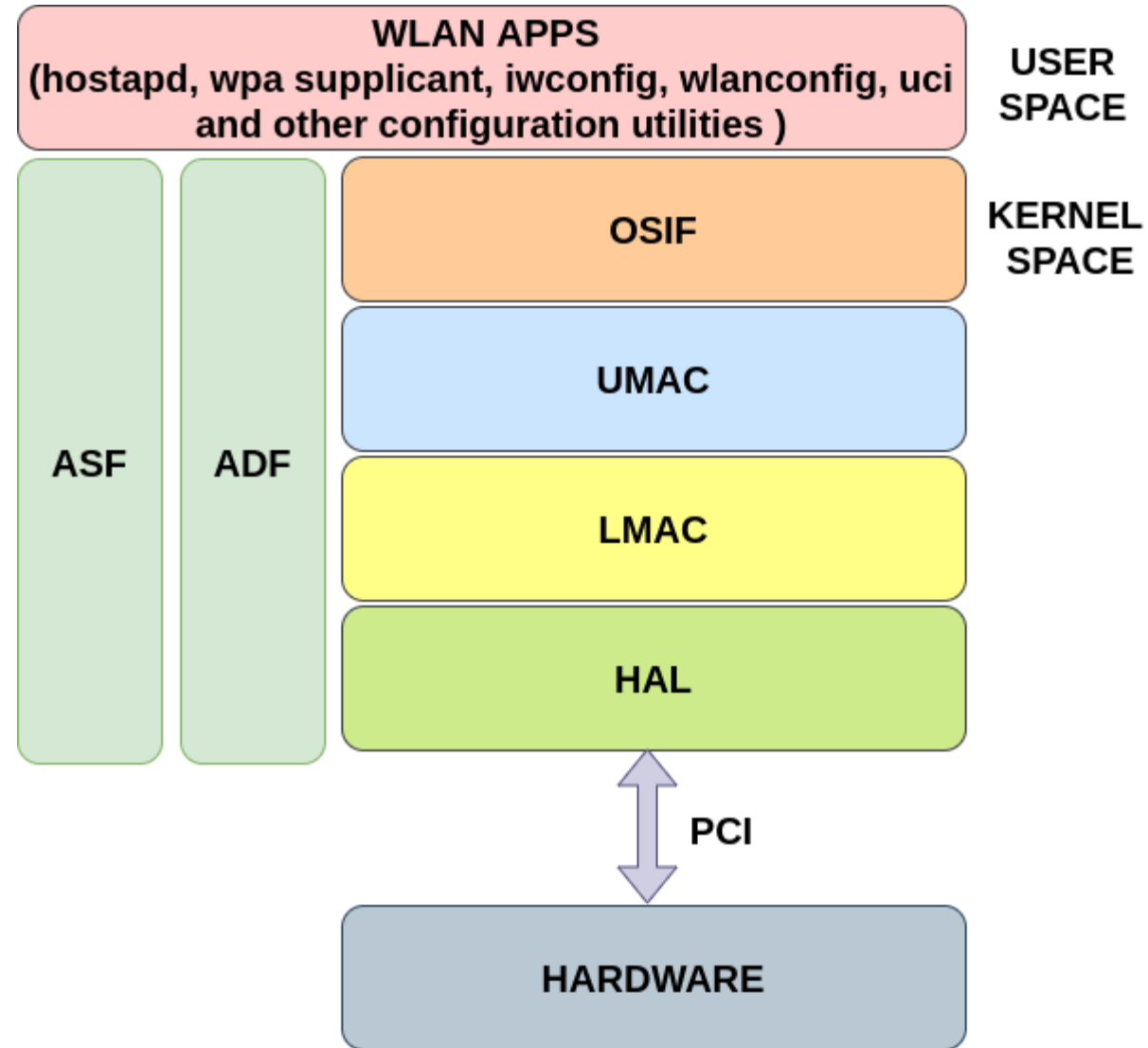
Direct Attach - Entire WLAN driver resides on the host CPU. The driver communicates with the HW through the PCI(AHB etc).



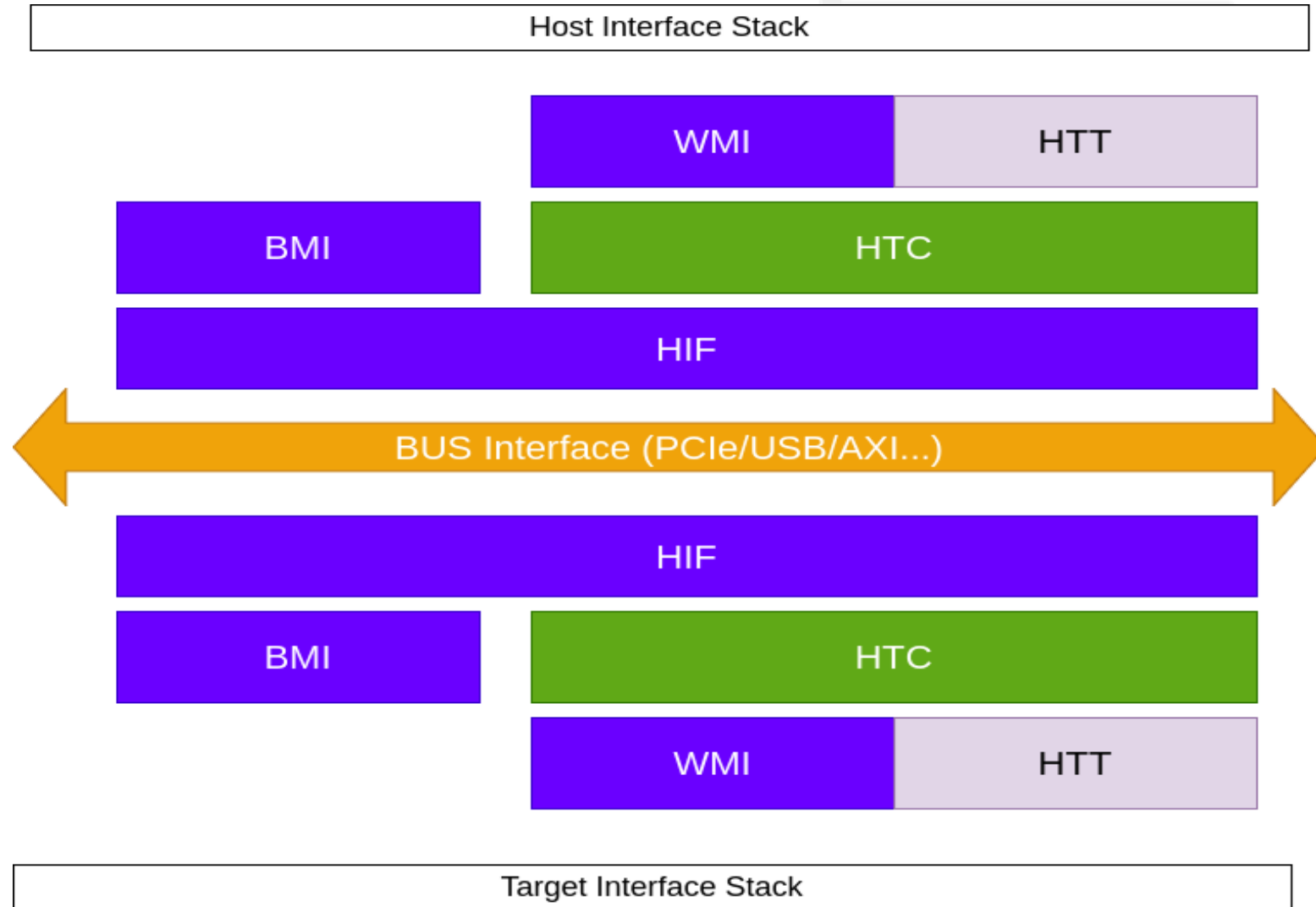
DIRECT ATTACH

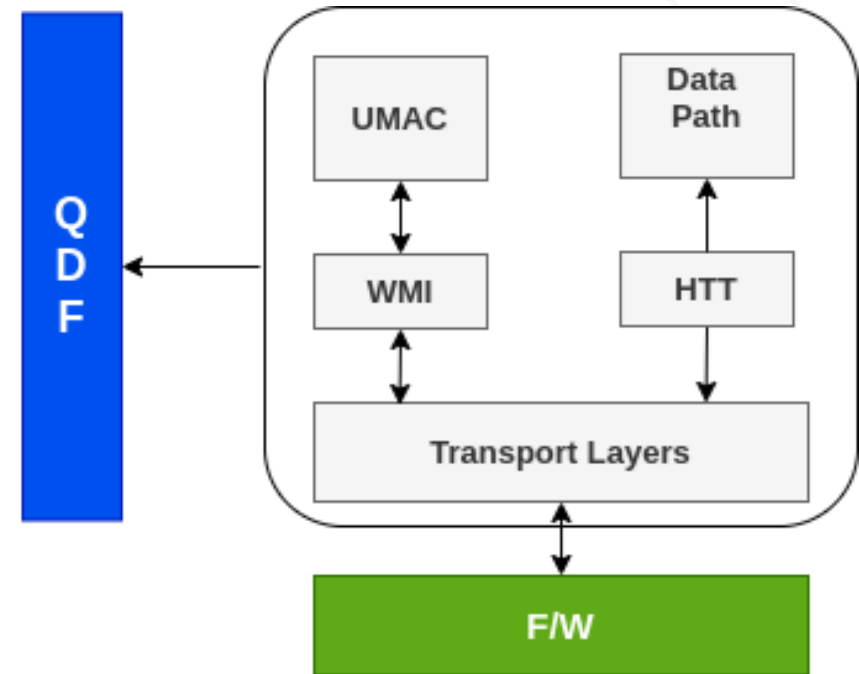
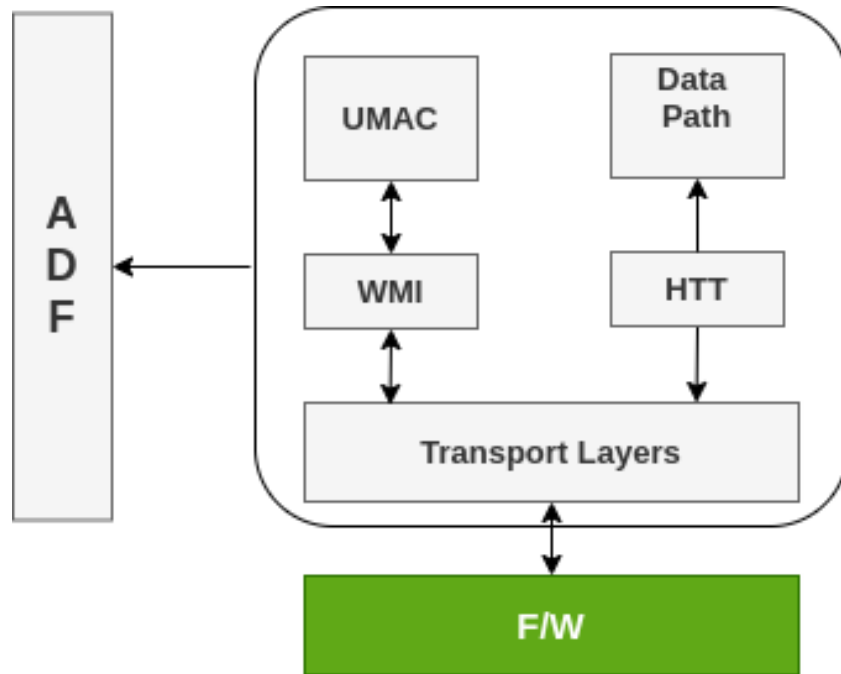
MODEL

WLAN Driver Architecture high level for Direct attach.



Partial offload Architecture





HIF - HOST INTERFACE LAYER

- Host interface layer (HIF) This layer abstracts the bus interface between the host and target, and provides a communication mechanism between the host and target.

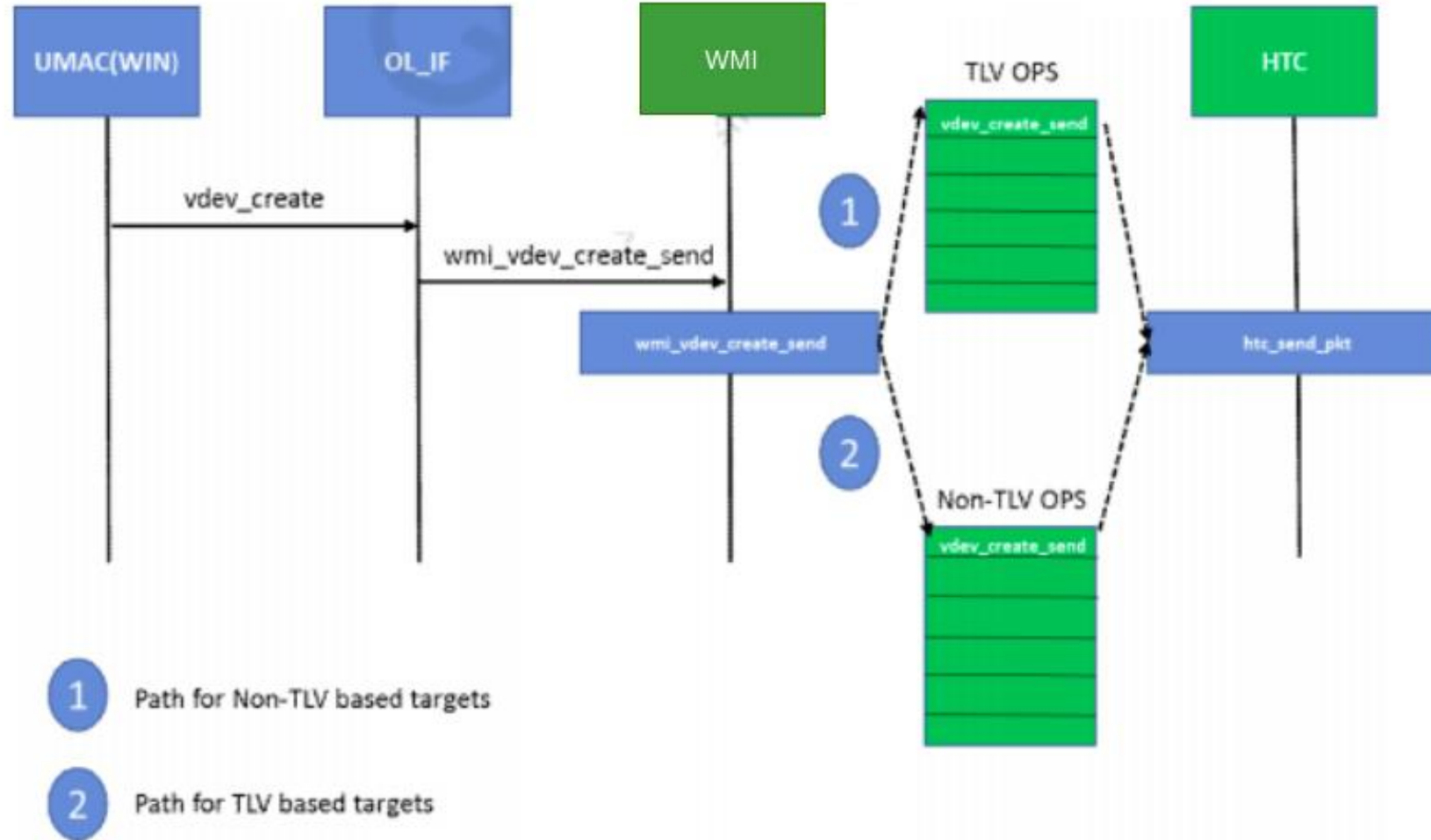
HIF Data Structures

- Bus independent - ***struct hif_softc*** , This is a common data structure for all the buses. It does not contain any bus-specific information
- Bus dependent - ***struct hif_pci_softc***.

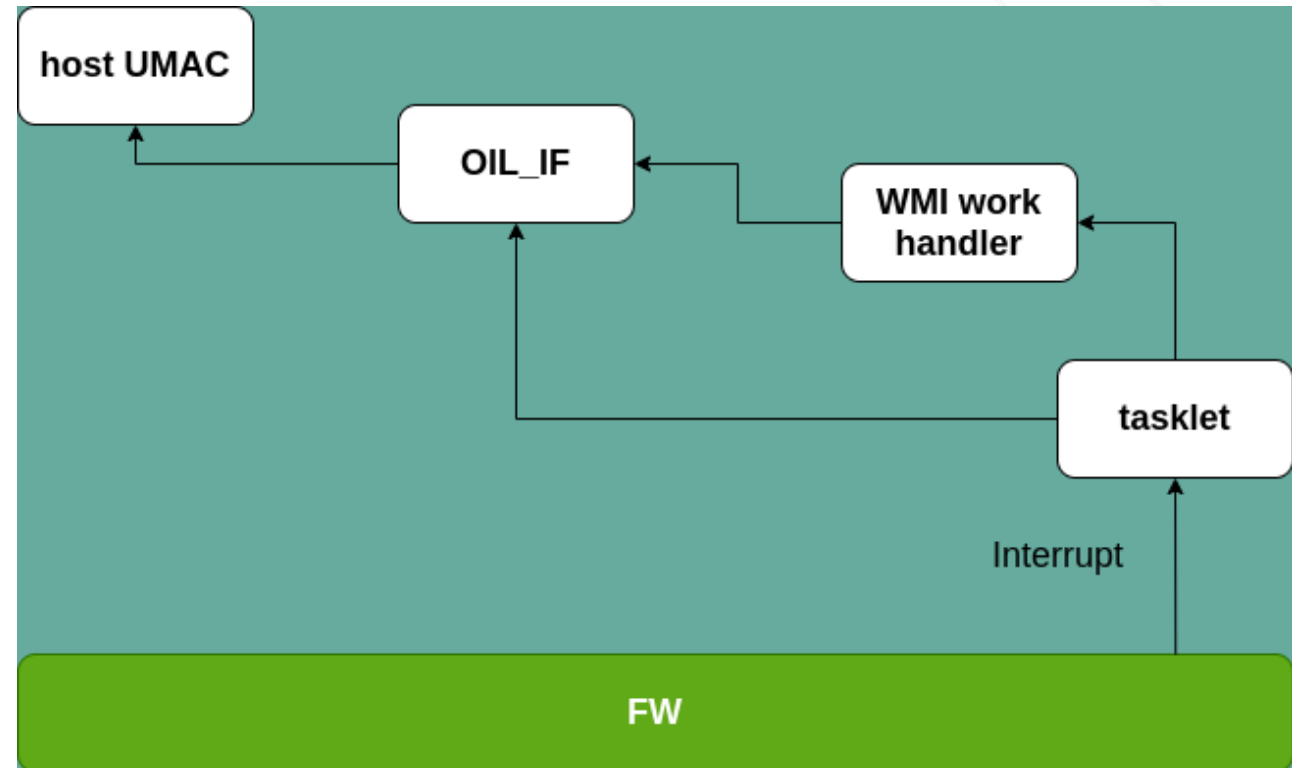
WMI LAYER

- WMI offers the control path(plain), any communication which is between host and target and vice versa done through WMI. All commands, events and MGMT packets received through the WMI.
- WMI Transmit is asynchronous in nature and there is no response associated in return. In case there is response associated with a command, they are handled as event from firmware.
- WMI does not hold any blocking context. This is responsibility of the upper layers to have a blocking context whenever the upper layer wants to handle the response in same context.
- WMI Type TLV or NON TLV is hidden from the caller and WMI type selection is based on the tagert type.

WMI command going to EP for VAP create (Tx path)



WMI Receive Path



WMI supports RX callback in following execution context -

- Default Tasklet context.
- Worker (process) context.

WMI initialization.

- It will happen during the device probe.
- ***wmi_unified_attach()***; /* wmi registration API */
- Context in which we need to handle the WMI event is generally decided during the registration time, it can be tasklet context for time sensitive events and work context for some other events.
- Generally mgmt packets are handled in work context.
- RX registration API will provide an argument to choose the context in which Upper layer wants this event to be handled. Based on the execution context requested, RX event callback will be called in that context.

HTT Layer and HTC layer

- Host Target Transport Layer (HTT) Host-to-target data path communications are done through the HTT interface.
- Sends frame descriptors to the firmware using HTC.
- On AP platforms, the “low latency” code path is used.
- Communication between the host and the target is through DMA entity called Copy Engine. The “Copy Engine” has 8 pipes. Each pipe in the Copy Engine, can be configured to talk either from host to target, or from target to host All notifications from target to host are in the form of interrupts.
- CE_pipe_config defined in `cmn_dev/hif/src/ce/ce_assignment.h`

PCI enumeration and probing.

ol_pci_probe(); /* As soon as qca_ol.ko is loaded. */
initialize bus related data structures.

hif_open.()

Allocation of HIF instance. /* ***struct hif_softc*** */ initializes the generic bus data structures.

struct hif_softc *scn; /* bus specific structure */

hif_bus_open(); does the ***hif_initialize_pci_ops();*** /* initialize the bus ops */

Here we took example of the PCI bus, it can be some other type of the bus also depending on the target bus.

- Once the bus specific initialization is done we need to setup the bus.
- ***hif_enable_pci*** - / *hif_initialize_pci_ops - list of PCI ops */
- ***pci_enable_device***. / * initialize the pci device before doing any operation*/
- ***pci_request_region***./* pci_request_region(pdev, BAR_NUM, "ath");*/
- ***pci_set_consistent_dma_mask***(pdev, DMA_BIT_MASK(64));
/* DMA addressing capabilities */
- ***pci_set_master***(pdev);
- ***mem = pci_iomap(pdev, BAR_NUM, 0);***
/ * create a virtual address cookie for the PCI BAR region */

managed version of API's during the registration.

for example ***pcim_enable_device***(pdev) instead of pci_enable_device();

hif_pci_bus_configure();

/ setup the CE (copy engine) and the interrupts */*

For example following is the call for the interrupts registration

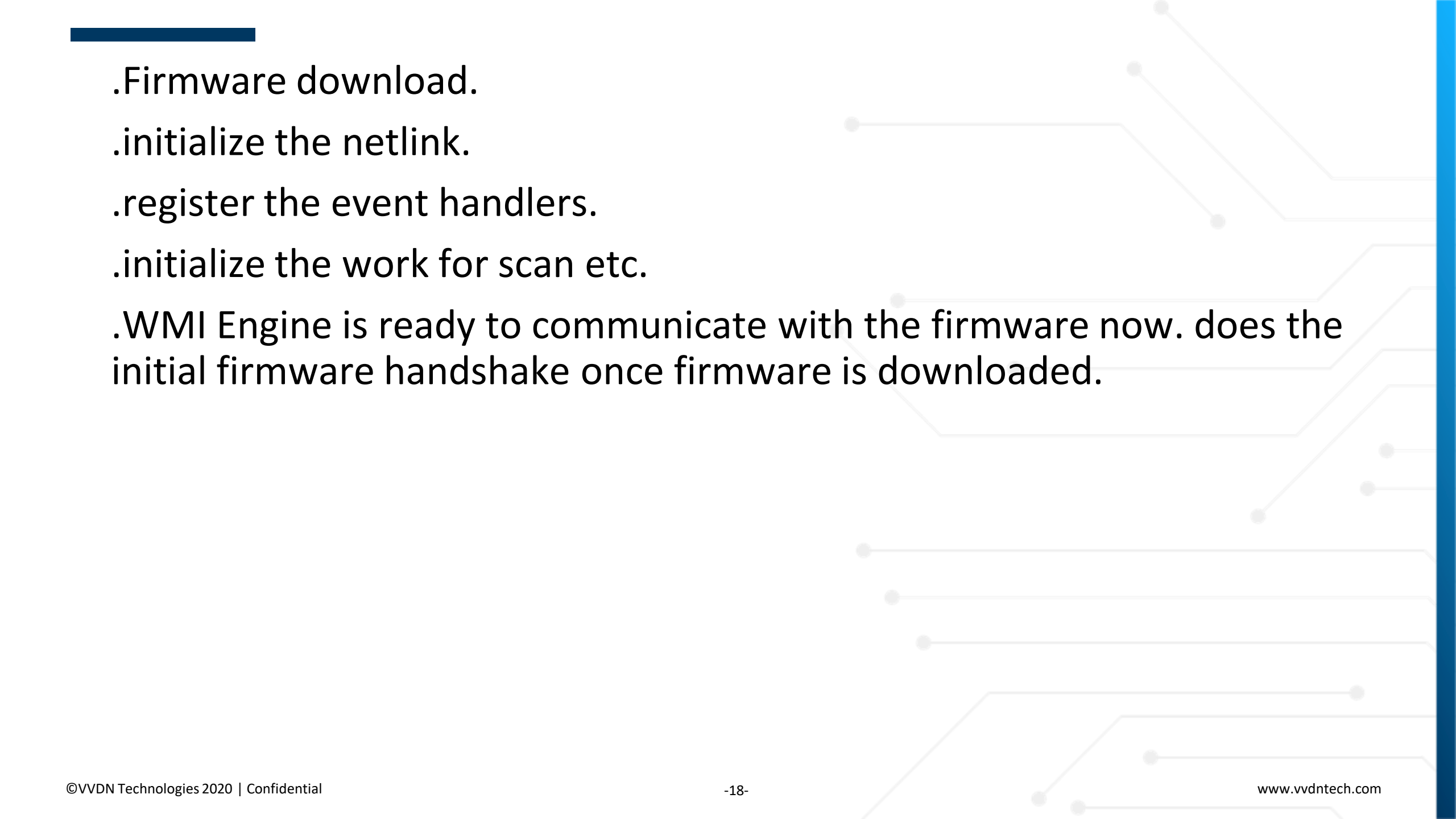
hif_pci_configure_legacy_irq();

ret = request_irq(sc->pdev->irq,

hif_pci_interrupt_handler, IRQF_SHARED,

"wlan_pci", sc);

wmi_unified_attach();

- 
- .Firmware download.
 - .initialize the netlink.
 - .register the event handlers.
 - .initialize the work for scan etc.
 - .WMI Engine is ready to communicate with the firmware now. does the initial firmware handshake once firmware is downloaded.

Device Registration to the network stack.

```
__ol_ath_attach();//
```

Till now we have setup the communication channel and device specific things now we need to register the device to the network stack.

```
dev = alloc_netdev(sizeof(struct ol_ath_softc_net80211), "wifi%d",  
ether_setup);
```

```
/* allocate the netdevice, struct ol_ath_softc_net80211 is the private  
data for the netdevice */
```

```
struct ieee80211com *ic; /* represents the radio */ is also allocated  
with ol_ath_softc_net80211.
```

- initialize the netdev->ops
- dev->open = ath_netdev_open;
- dev->stop = ath_netdev_stop;
- dev->hard_start_xmit = ath_netdev_hardstart;
- /* any extra headroom if required like in case of MESH etc can be set in the dev->headroom*/
- IEEE80211_ADDR_COPY(dev->dev_addr,ic->ic_myaddr);
- SET_NETDEV_DEV(dev, osdev->device);
- register_netdev(dev)); /* device is ready*/.
- Once the device registration is done we can do the virtual interface creation on top of that.

Data path

All notifications from target to host are in the form of interrupts.

1. checks for the exception /error interrupts/Data.
2. turns off interrupts.
3. schedules a delayed processing routine(napi or tasklet).

To understand the data path we need to understand the following concepts .

- 1.skB, skb->data, skb operations.
- 2.Tx and Rx descriptors.

3.DMA mapping coherent and streaming.

Generally the coherent mapping is used for the tx and rx descriptor ring buffers and their allocation is done through `dma_alloc_coherent()`;

It returns two values: the virtual address which you can use to access it from the CPU and `dma_handle` which you pass to the card.

They have the consistent DMA mapping throughout the driver.

Streaming DMA - `pci_map_single(pdev, skb->data)`; returns a single map region and once the DMA activity is done , we need to perform the `pci_unmap_single()`;

Once a buffer has been mapped, it belongs to the device. Until the buffer has been unmapped, the driver should not touch its contents in any way.

4.meta data

meta data is the info attached or control info with per packet by the host for the target in case of Tx and for Rx target attach a Rx meta data for the host. This is basically some brief info or description about the packet , which one party is sending to the other. This agreement is fixed during the initialization time.

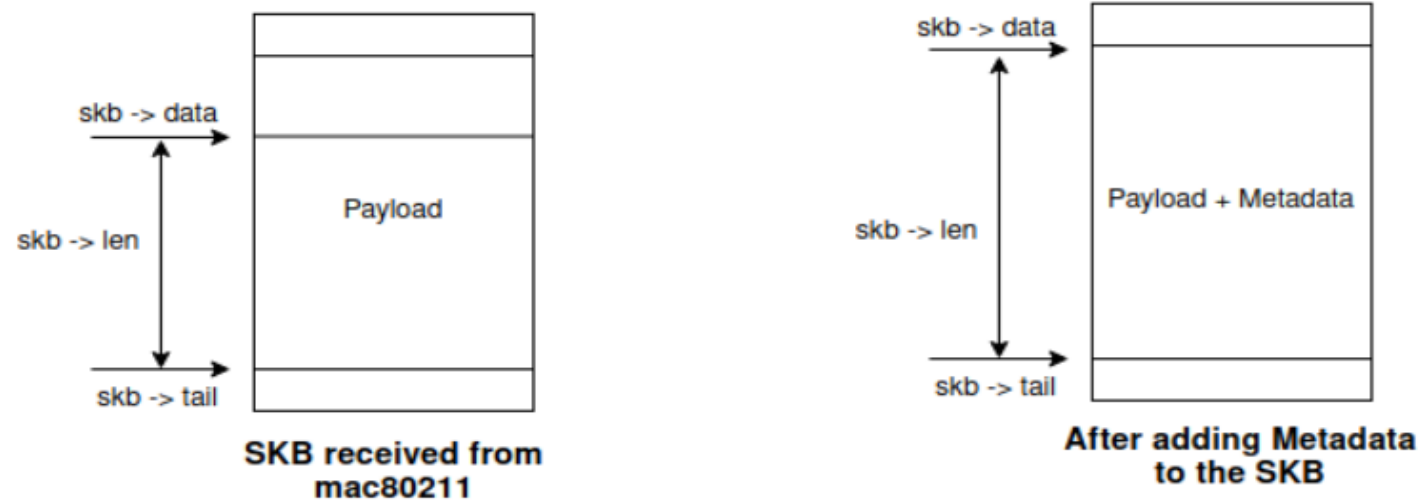
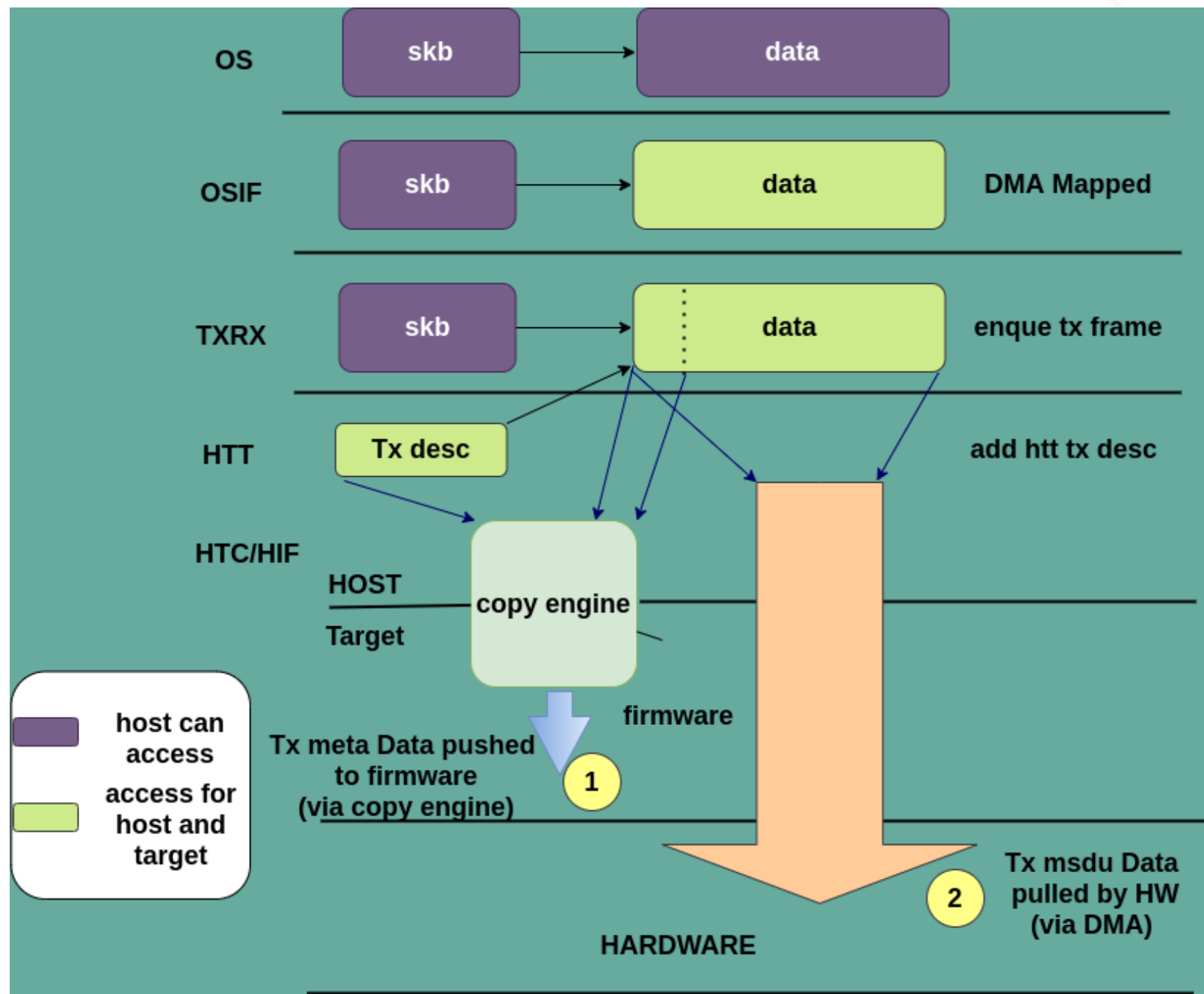


Figure 17. SKB with Metadata

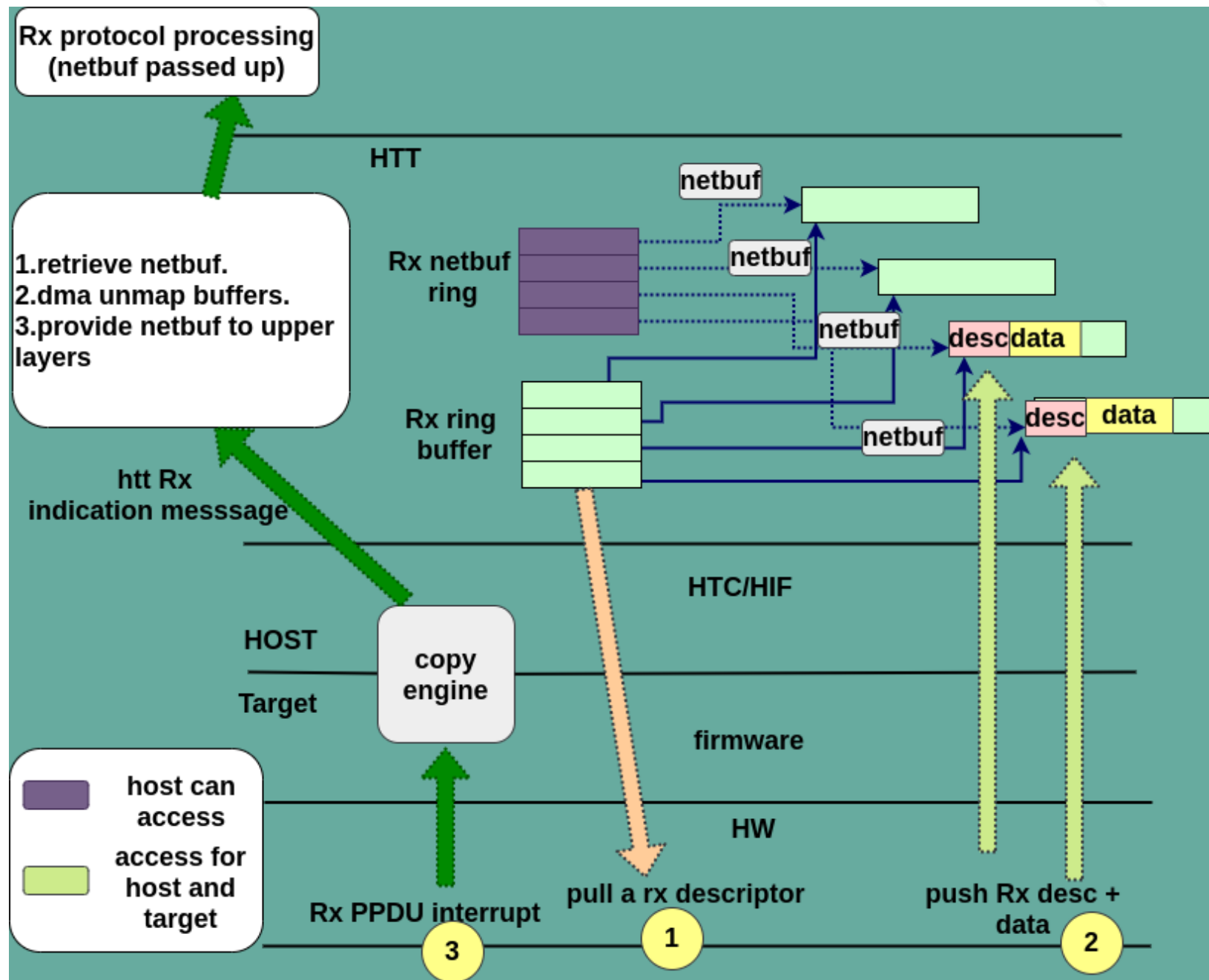
5. Ring buffer.

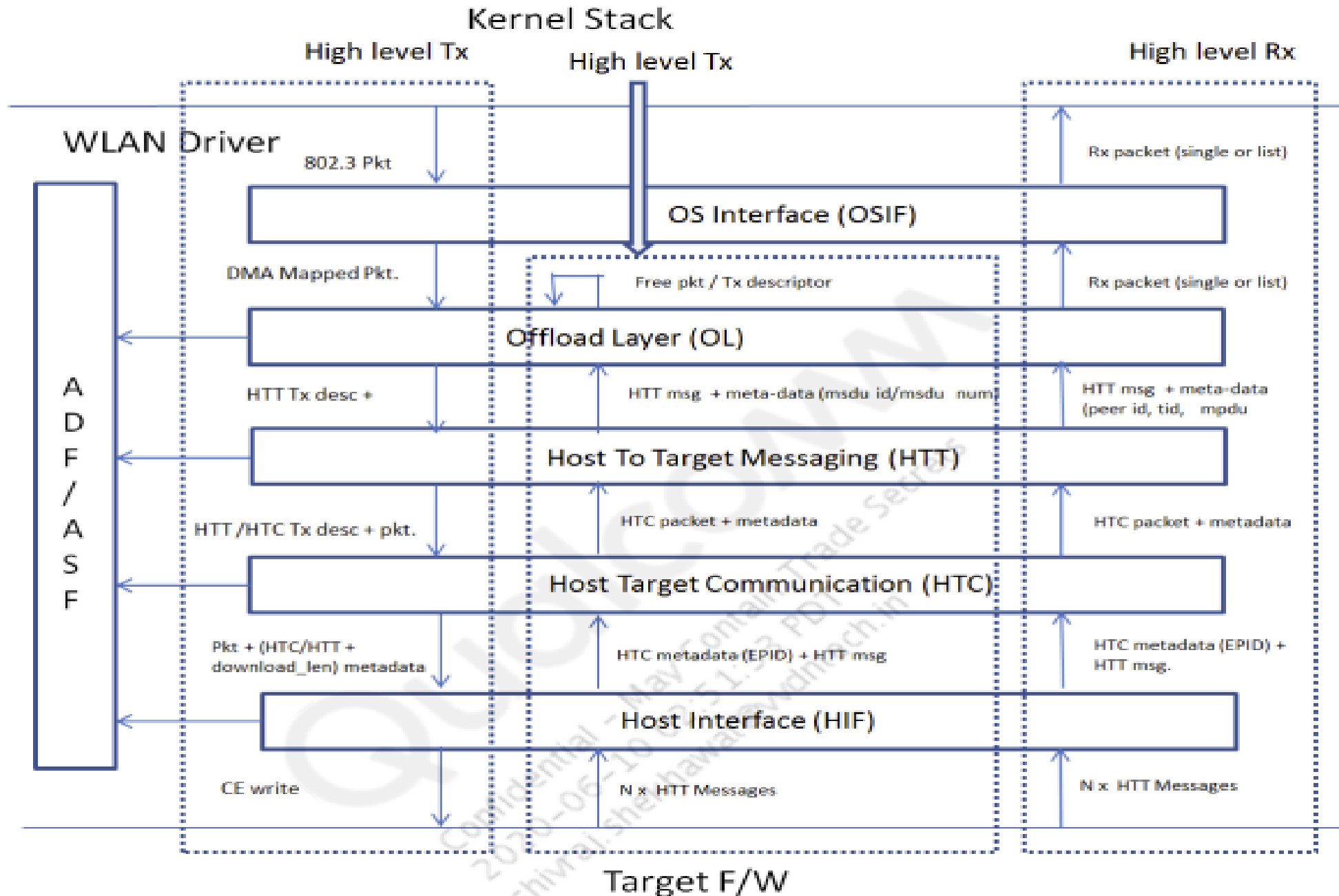
During the initialization time (after device probe) host reserves a DMA mapped region of the rx ring buffer. The number of the elements in the ring called the Rx descriptor count. This is the effective Rx packets available in the system. Whenever the HW receives a packet it does a DMA to the available ring buffer element (index) and write the same info in the rx descriptor table and notifies to the host by raising an interrupt.

Host then iterate through the ring buffer , DMA unmap the packet and send it to network stack after doing the processing. Later the ring buffer is replenished by a new SKB.



- HTT message downloads tx -meta data (descriptor) + MSDU's L2/L3 header(via Copy engine).
- MAC HW downloads MSDU Payload.
- CE 4 is used for the host to target of HTT + HTC descriptor and a portion of the data packet.
- For a given packet, the driver downloads 24 bytes of HTT/HTC header + a certain portion of the data-packet to the target. Even though the data-packet is DMA'ed by the MAC h/w directly from host memory, this partial download is required so that the target can parse the required packet headers for packet classification. This partial download len includes the L2 headers + 2 bytes of L3 headers (for IPv4 TOS / IPV6 Flow label) information. The download length is set at driver initialization time and has the following values: Ethernet packet 28 bytes Native 802.11 packet 44 bytes Raw 802.11 packet 50 bytes





Host Tx

DMA map the data buffer, for the packet or list of packets received from stack.

Set up meta-data required for target Set up the copy engine to do a transfer of the meta-data & part of the data packet.

Do Tx completion, which can happen in 2 steps:

Tx descriptor download complete

Tx packet complete

Only when both parts are completed, the packet is freed.

Target Tx

Packet classification into WMM access classes.

Aggregation per TID, aggregation setup & teardown.

Rate lookup. Indicate packet completion via HTT messages.

Host Rx

Post Rx buffers into which MAC hardware can DMA.

Discard the packets or reorder & deliver the packets up to the stack (depending on HTT message indication)

Target Rx Block Ack Window Management and sending of Block Ack.
Posting HTT Rx indication messages to the host .

The header encap/decap and the encryption/decryption (if security is enabled) is done by the MAC hardware

Different Types of Packets Received on the WLAN Driver.

- 802.3 packets.
- 802.11 packets.
- Raw packets.
- skb (wbuf can be linear or fragmented). In case skb is fragmented scatter gather needs to be used. support should be there in the Hardware side.
- SKB may or may not VLAN header.
- TSO , GRO , netdev->features.

Different Processing context in which WLAN Driver work.

- ISR.
- Softirq/Tasklet
- process context.



CREATE. INNOVATE. PRODUCE
#WEMAKEITHAPPEN

Premier Electronics Engineering & Manufacturing Company