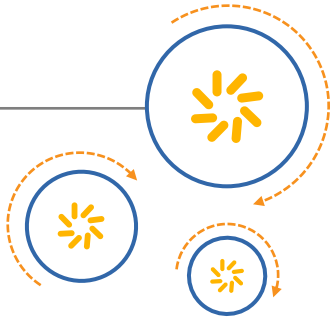
NOTICE REGARDING QUALCOMM ATHEROS, INC.

Effective June 2016, Qualcomm Atheros, Inc. (QCA) transferred certain of its assets, including substantially all of its products and services, to its parent corporation, Qualcomm Technologies, Inc. Qualcomm Technologies, Inc. is a wholly-owned subsidiary of Qualcomm Incorporated. Accordingly, references in this document to Qualcomm Atheros, Inc., Qualcomm Atheros, Atheros, QCA or similar references, should properly reference, and shall be read to reference, Qualcomm Technologies, Inc.

Qualcomm Technologies, Inc.



Wi-Fi Debug Framework for Android Nougat

User Guide

80-Y7674-152 Rev. B

July 29, 2016

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	November 2015	Initial release
B	July 2016	Updated document for Android-N

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document defines the Wi-Fi Debug Framework used in the Android Nougat release.

FCC NOTICE:

This kit is designed to allow:

(1) Product developers to evaluate electronic components, circuitry, or software associated with the kit to determine whether to incorporate such items in a finished product and

(2) Software developers to write software applications for use with the end product. This kit is not a finished product and when assembled may not be resold or otherwise marketed unless all required FCC equipment authorizations are first obtained. Operation is subject to the condition that this product does not cause harmful interference to licensed radio stations and that this product accepts harmful interference. Unless the assembled kit is designed to operate under part 15, part 18 or part 95 of the FCC's rules, the operator of the kit must operate under the authority of an FCC license holder or must secure an experimental authorization under part 5 of the FCC's rules.

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, **`copy a:*.* b:`**

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

2 Debug framework

2.1 Overview

The latest Android-N debug framework leverages the existing QCOM DIAG framework, and the CNSS_DIAG infrastructure, to ensure that it is scalable for logkit/RIDL/qxdm/Google-M debug infrastructure.

Logging infrastructure enables logging of both firmware and driver logs to the user space that can be retrieved via BUG REPORT. A developer UI option is provided to enable or disable the collection of logs and to choose the verbose level of logging infrastructure.

The framework has the capability to trigger BUGREPORT if certain events fail.

In the user-space, both Wi-Fi HAL and CNSS_DIAG services receive firmware and host logs. CNSS_DIAG sends a copy of the logs to QXDM.

If required, the CNSS_DIAG service, can also write the logs to a file that can be extracted by the user when there is any issue with the WLAN.

The Wi-Fi HAL stores the firmware, host logs, and other logs to respective ring buffers in the Wi-Fi HAL.

Each ring contains its own type of debug data. That data is encoded and written in a compressed format. See, Figure 3-2 for more details.

To extract logs, run the post-processing script on the bug-report file.

See Table 2-1 for more details about the types of logs, which include, driver logs, firmware logs, firmware memory dump, connectivity events, etc.

Table 2-1 Debug framework requirements

Service	Flow
WLAN connectivity events	Support for connectivity events and the existing QCOM events are transformed to Google format in the Android Nougat release. Connectivity events are generated from both HOST and Firmware, and copied to a ring buffer on the HOST. Any bugs triggered from the framework are collected and they become part of the bug report. Example Events: EVENT_WLAN_EAPOL EVENT_WLAN_PE EVENT_WLAN_ROAM EVENT_WLAN_GSCAN CNSS_DIAG handles the debug framework WLAN connectivity events, along with the existing WLAN events. Connectivity events route from both host and firmware to CNSS_DIAG. A ring buffer in the Wi-Fi HAL stores events.

Service	Flow
Firmware memory dump	The memory dump generates a firmware memory DRAM snapshot. Firmware determines the contents of the dump. Its size is currently restricted to 350 KBytes. Dump is triggered by the framework and invoked as a part of the bug report or a request from the framework.
Per packet stats	Per packets stats provide packet information at the firmware layer. The stats for a packet include the following: ▪ Packet type (TX/RX) ▪ Timestamp ▪ RSSI information ▪ Number of retries Use this packet information, along with the Wireshark, to identify latencies for any data path traffic.
WAKELOCK events	The WLAN driver tracks any WAKELOCK taken, released, and expired. The WAKELOCK ring is maintained in the Wi-Fi HAL. Suspend/resume events are notified to the framework and the same information is provided to framework upon request.
Critical/fatal events	FATAL events are considered critical and require immediate attention. FATAL events notify the framework to provide event information, to trigger an immediate bug report from the framework.
TX/RX Capture	After association, Framework captures the first 32 TX/RX packets. This is used in triaging most of the connection issues and to determine if any packets are dropped at framework or firmware or driver layers. It captures the following: • EAPOL • DHCP • ARP • AUTH frames

3 Architecture

The debug framework follows a generic architecture that enhances the existing CNSS_DIAG to the QXDM interface, and also supports the new logging requirements.

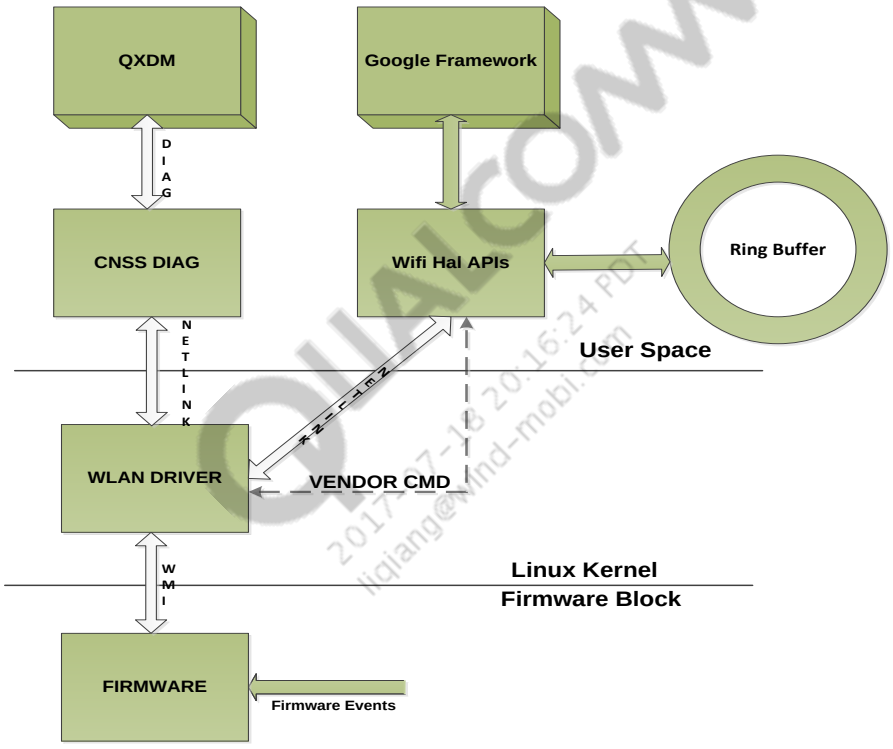


Figure 3-1 Generic debug framework depiction

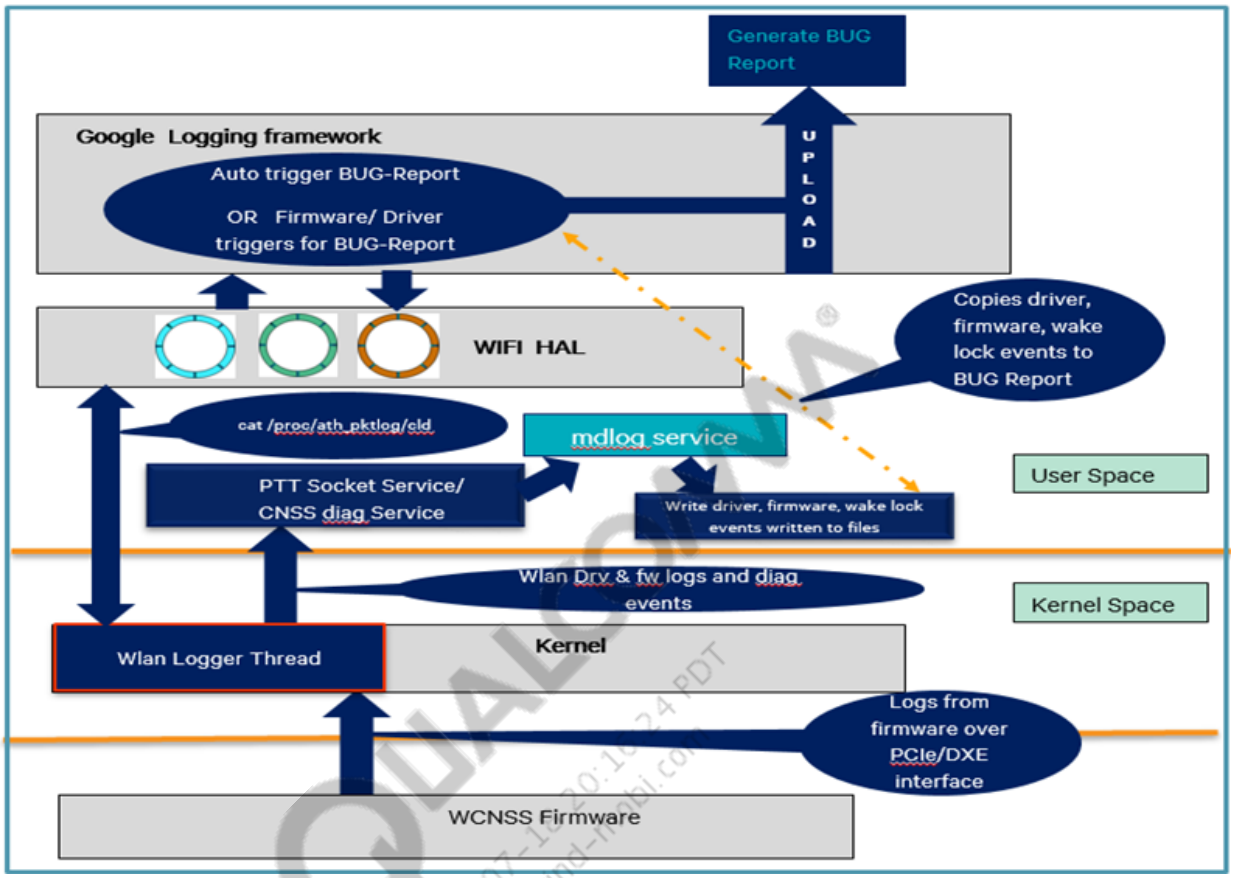


Figure 3-2 Debug framework architecture

4 Debug framework sequence flow

4.1 Initialization sequence

The framework, at the time of initialization, calls the wifi_get_ring_buffers_status to obtain the names and lists of supported buffers. When the Wi-Fi is operational, the framework calls the command, wifi_start_logging, to trigger the beginning of log collection.

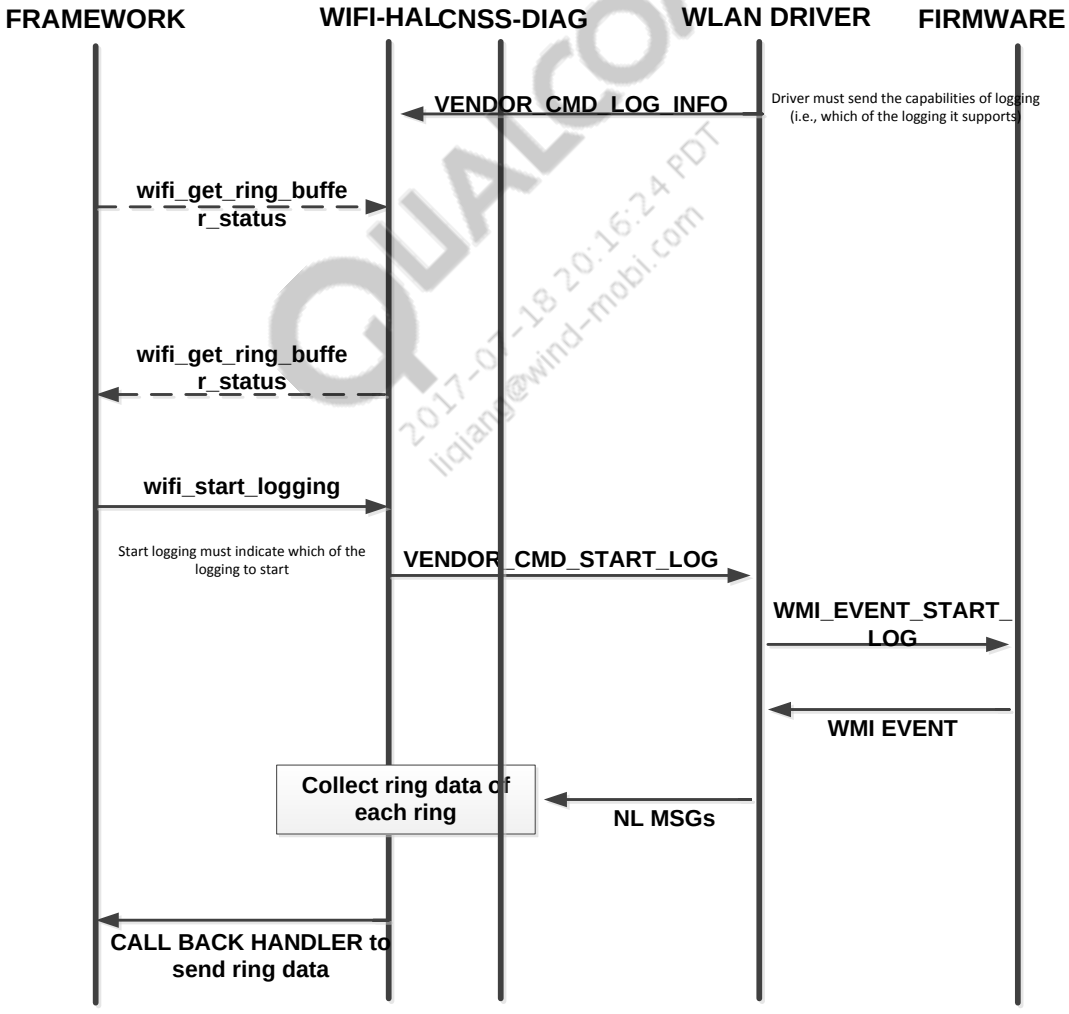


Figure 4-1 Generic command flow for logging

The wifi_get_ring_buffer_status gets the individual ring buffer information. This API status can be used to set/get the verbose levels for a given ring.

When conditions related to time interval or ring size are met, the ring data is automatically sent to the framework through the on_ring_buffer_data call back.

When a bug report is requested, the framework communicates to the driver to upload all the data. The framework subsequently initiates the wifi_get_ring_data and wifi_hal vendor commands to the WLAN driver and firmware.

The framework stores data uploaded by Wi-Fi HAL in separate files, with one stream of files per ring. Files are stored in pcapng format for easy merging and parsing with network analyzer tools.

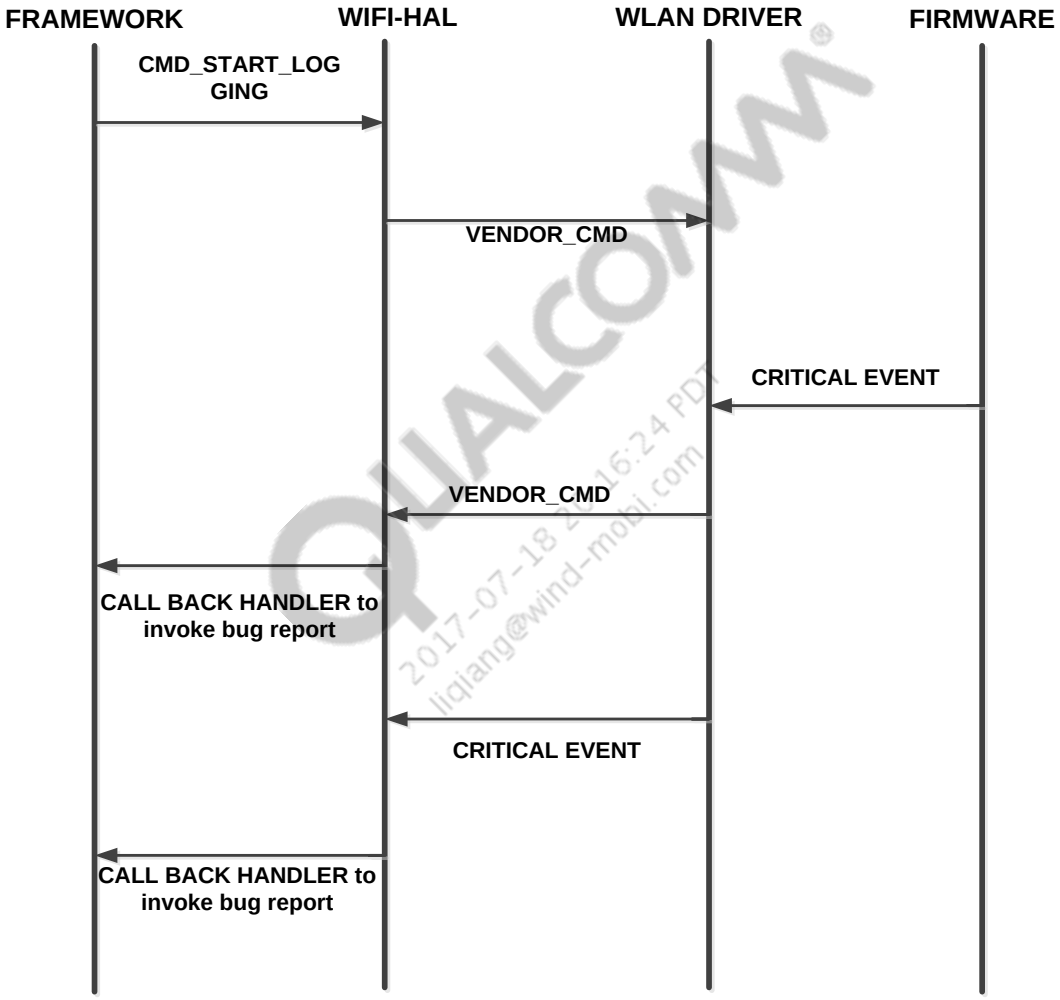


Figure 4-2 Command flow for critical events

4.2 Per packet stat flow

QCA6174 (A) chipsets have an existing packet logging infrastructure since the Android Lollipop (L) release. Firmware, through HTT messages, sends the content of each packet to the host driver. The "pktlogconf" tool then pulls the packet log through the /proc/ath_pktlog/cld entry. Using an external script, this packet dump can be converted to a readable format.

The driver periodically pushes the driver packet logs to the HAL ring buffers.

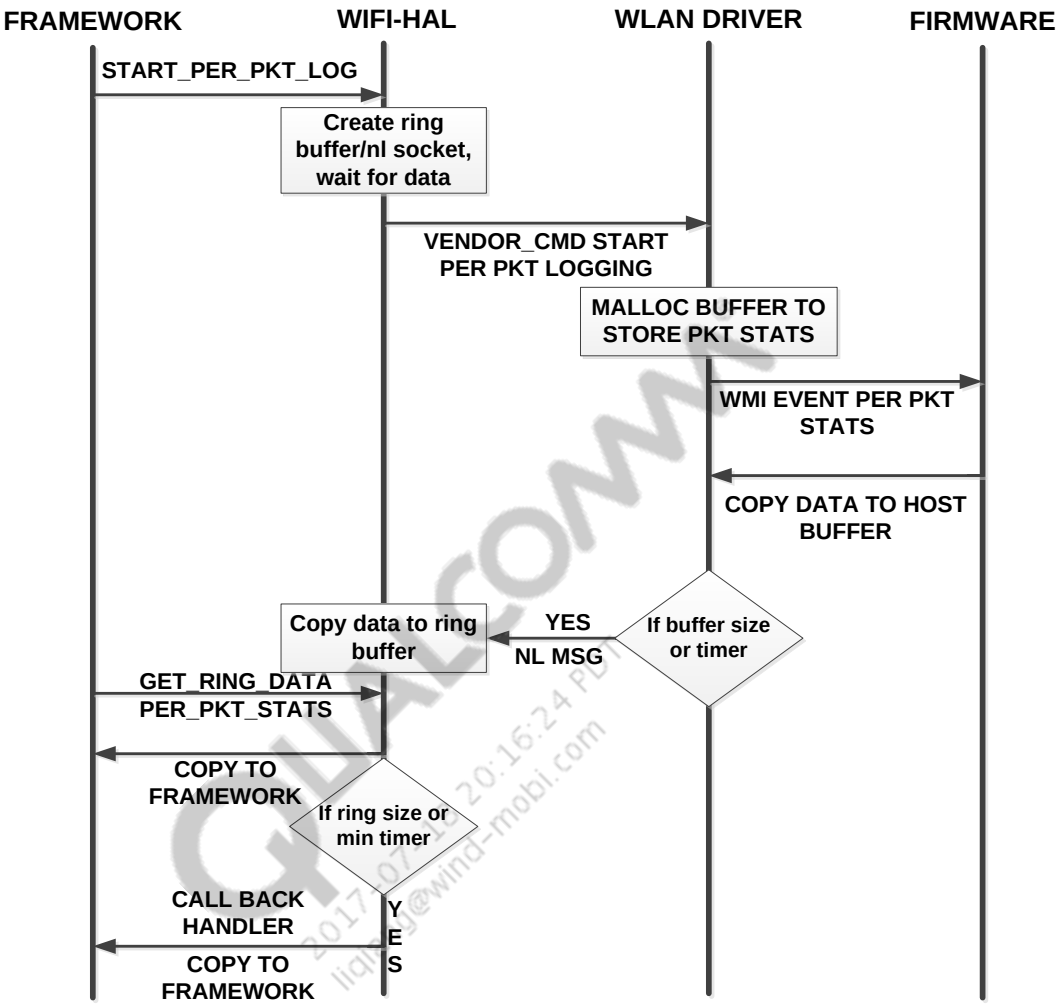


Figure 4-3 Per packet statistic flow

Figure 4-4 depicts the existing packet logging infrastructure for QCA6174 (A) chipsets.

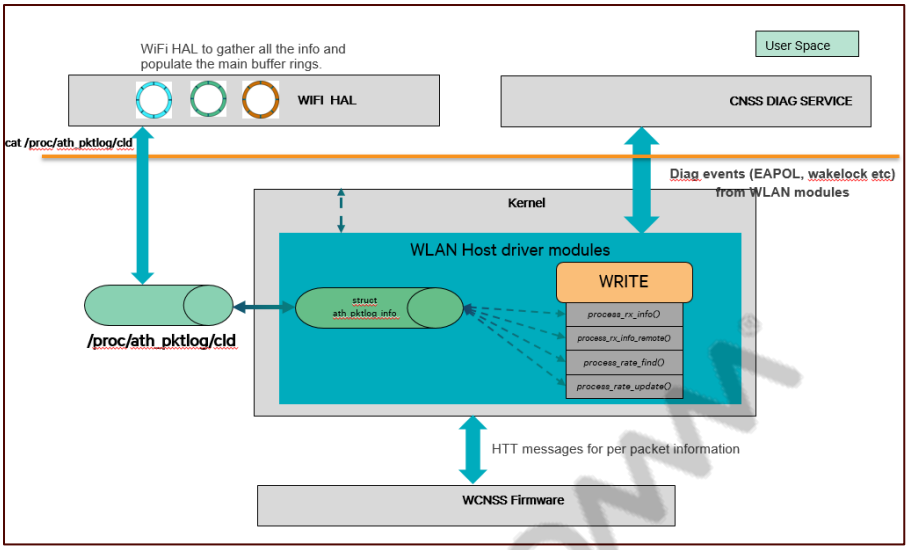


Figure 4-4 WLAN packet logging infrastructure

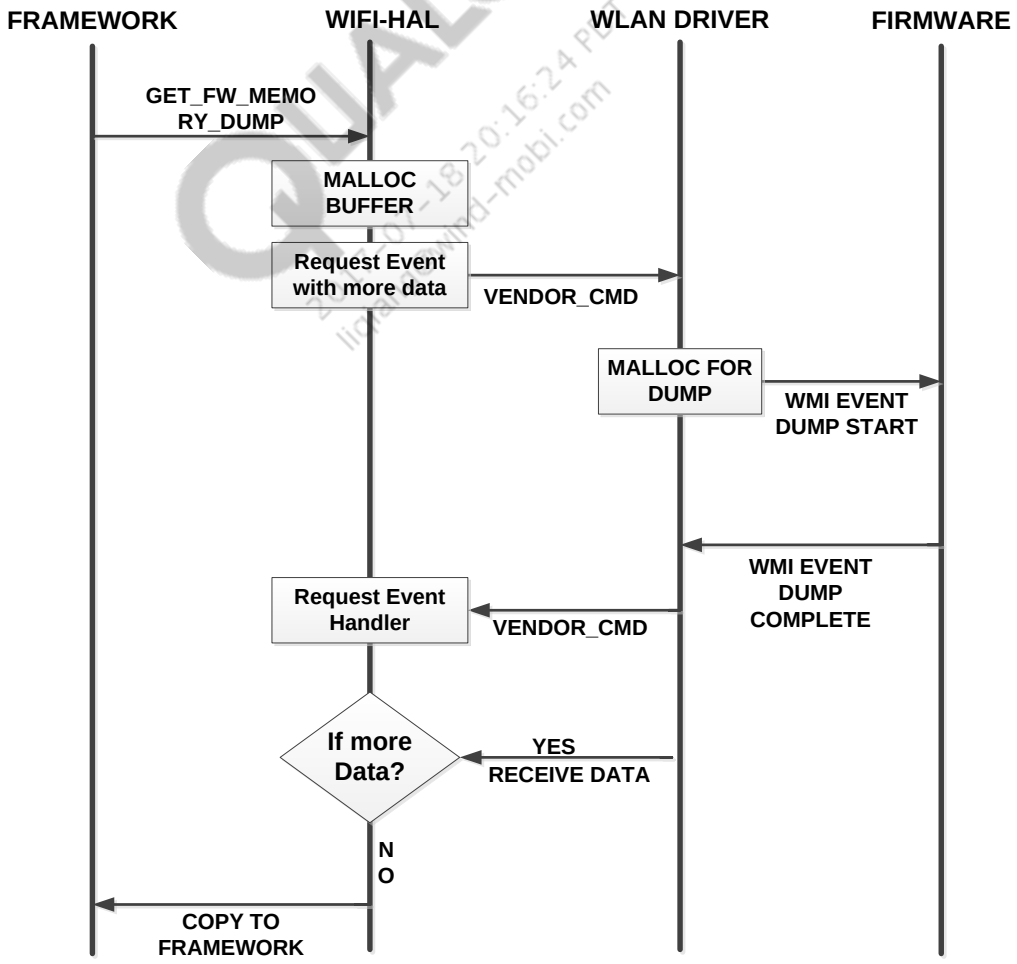


Figure 4-5 Flow for memory dump command

5 APIs and Event list

5.1 Supported framework/Wi-Fi HAL APIs

5.1.1 Android-N

```
wifi_error wifi_get_firmware_version(wifi_interface_handle iface,
                                     char *buffer, int buffer_size)

wifi_error wifi_get_driver_version(wifi_interface_handle iface, char
*buffer, int buffer_size);

wifi_error wifi_get_firmware_memory_dump(wifi_interface_handle iface,
wifi_firmware_memory_dump_handler handler);

wifi_error wifi_get_ring_buffers_status(wifi_interface_handle iface,
u32 *num_rings, wifi_ring_buffer_status *status);

wifi_error wifi_start_logging(wifi_interface_handle iface, u32
verbose_level, u32 flags,u32 max_interval_sec, u32 min_data_size, char
*ring_name);

wifi_error wifi_get_logger_supported_feature_set(wifi_interface_handle
iface, unsigned int *support);

wifi_error wifi_get_ring_data(wifi_interface_handle iface, char
*ring_name);
```

5.2 Connectivity event list

Header file is in
<AU_ROOT>/hardware/libhardware_legacy/include/hardware_legacy/wifi_logger.h

Table 5-1 Connectivity event list

Connectivity events	Description
WIFI_EVENT_ASSOCIATION_REQUESTED	Authentication/association events
WIFI_EVENT_AUTH_COMPLETE	
WIFI_EVENT_ASSOC_COMPLETE	
WIFI_EVENT_FW_AUTH_STARTED	

Connectivity events	Description
WIFI_EVENT_FW_ASSOC_STARTED	
WIFI_EVENT_FW_RE_ASSOC_STARTED	
WIFI_EVENT_DRIVER_SCAN_REQUESTED	Host triggered scan events
WIFI_EVENT_DRIVER_SCAN_RESULT_FOUND	
WIFI_EVENT_DRIVER_SCAN_COMPLETE	
WIFI_EVENT_DISASSOCIATION_REQUESTED	Disassociation/reassociation events
WIFI_EVENT_RE_ASSOCIATION_REQUESTED	
WIFI_EVENT_ROAM_AUTH_STARTED	Roam scan/reassociation events
WIFI_EVENT_ROAM_AUTH_COMPLETE	
WIFI_EVENT_ROAM_ASSOC_STARTED	
WIFI_EVENT_ROAM_ASSOC_COMPLETE	
WIFI_EVENT_ROAM_SCAN_STARTED	
WIFI_EVENT_ROAM_SCAN_COMPLETE	
WIFI_EVENT_ROAM_CANDIDATE_FOUND	
WIFI_EVENT_ROAM_SCAN_CONFIG	
WIFI_EVENT_CHANNEL_SWITCH_ANOUNCEMENT	Channel switch announcement
WIFI_EVENT_BT_COEX_BT_SCAN_START	Bluetooth co-ex scan/SCO/HID events
WIFI_EVENT_BT_COEX_BT_SCAN_STOP	
WIFI_EVENT_BT_COEX_BT_SCO_START	
WIFI_EVENT_BT_COEX_BT_SCO_STOP	
WIFI_EVENT_BT_COEX_BT_HID_START	
WIFI_EVENT_BT_COEX_BT_HID_STOP	
WIFI_EVENT_G_SCAN_CAPABILITIES	Gscan events
WIFI_EVENT_G_SCAN_CYCLE_STARTED	
WIFI_EVENT_G_SCAN_CYCLE_COMPLETED	
WIFI_EVENT_G_SCAN_BUCKET_STARTED	
WIFI_EVENT_G_SCAN_BUCKET_COMPLETED	
WIFI_EVENT_G_SCAN_STOP	
WIFI_EVENT_G_SCAN_RESULTS_AVAILABLE	
WIFI_EVENT_BLOCK_ACK_NEGOTIATION_COMPLETE	Aggregation events
WIFI_EVENT_BLOCK_ACK_NEGOTIATION_COMPLETE	
WIFI_EVENT_DRIVER_EAPOL_FRAME_TRANSMIT_REQUESTED	802.1x EAPOL events
WIFI_EVENT_DRIVER_EAPOL_FRAME_RECEIVED	

6 OEM/ODM Integration

6.1 Using the halproxydaemon test tool

The halproxydaemon is a test tool developed by Qualcomm®. The tool calls the debug framework APIs directly, without depending on the Android framework.

To learn how to use these APIs quickly, use the with the command-line test tool through the ADB.

To check the sample codes, do the following:

1. Turn on Wi-Fi
2. Enter the ADB shell with root permission.

In the following example, see the highlighted input for the query-supported feature set.

6.1.1 halproxydaemon log output:

```
MSM8996:/ # hal_proxy_daemon wifi_logger wlan0
hal_proxy_daemon wifi_logger wlan0
halProxyDaemon running.
hal_proxy_daemon: Version: AU_LINUX_ANDROID_LA.HB.1.1.1.05.00.02.063.069-
138-gac68be8
WifiLoggerTestSuite::WifiLoggerTestSuite: Created a WifiLogger Test Suite
with request_id:1523578976
**********************
Step 1: Enter Request ID for WiFiLogger Cmd
Step 2: Enter ID for WifiLogger Cmd, as follows:
    Type 1 for WifiLogger Get Driver Version
    Type 2 for WifiLogger Get Firmware Version
    Type 3 for WifiLogger Start Logging
    Type 4 for WifiLogger Stop Logging
    Type 5 for WifiLogger Get Ring Buffer Status
    Type 6 for WifiLogger Get Supported Feature Set
    Type 7 for WifiLogger Get Ring Data
    Type 8 for WifiLogger trigger Memory Dump
    Type 9 for WifiLogger Get Driver Memory Dump
```

```
    Type 10 for WifiLogger Get wake reason stats
    Type 11 for WifiLogger Start packet fate stats
    Type 12 for WifiLogger Get Tx packet fate stats
    Type 13 for WifiLogger Get Rx packet fate stats
    Type 1000 to exit.
**********************
**********************
Now Enter Request ID:
1
1
Step 2: Enter WiFiLogger Cmd ID:
6
6
executeCmd: Enter
wifiLoggerGetSupportedFeatureSet: Sending Get Supported Feature Set.
wifiLoggerGetSupportedFeatureSet: Received Wi-Fi logger with value:0.
Supported Feature Set: 0x17.
**********************
Now Enter Request ID:
```

On the following bitmap, 0x17 means that memory dump per packet stats, connect event, and wake lock are supported.

```
enum wifi_logger_supported_features
{
WIFI_LOGGER_MEMORY_DUMP_SUPPORTED = (1 << (0)),
WIFI_LOGGER_PER_PACKET_TX_RX_STATUS_SUPPORTED = (1 << (1)),
WIFI_LOGGER_CONNECT_EVENT_SUPPORTED = (1 << (2)),
WIFI_LOGGER_POWER_EVENT_SUPPORTED = (1 << (3)),
WIFI_LOGGER_WAKE_LOCK_SUPPORTED = (1 << (4)),
WIFI_LOGGER_VERBOSE_SUPPORTED = (1 << (5)),
WIFI_LOGGER_WATCHDOG_TIMER_SUPPORTED = (1 << (6)),
};
```

6.2 Using the QXDM tool

To use the QXDM tool, do the following:

1. Turn on Wi-Fi.
2. Connect the QXDM tool to the device under test.
3. Start capturing the logs.

6.2.1 QXDM snippet host logs

In the following example, see highlighted input for the WLAN-related logs.

```
{MSG [cld-diag-parser.c  1657] R0: [wlan_logging_th][00:11:17.861719] wlan:
[5686:E :VOS] vos_get_log_indicator: 2840: vos context initialization is in
progress LoadUnload: 1 LogP: 0 ReInit: 0 WLAN Reserved 10/Medium [4516/0001]
4199/11/24 01:34:15.630}

{MSG [cld-diag-parser.c  1657] R0: [kworker/0:0][00:11:17.878122] wlan: [4:
D:WDA] wma_rx_ready_event: Enter WLAN Reserved 10/Medium [4516/0001] 4199/11/24
01:34:15.631}

{MSG [cld-diag-parser.c  1657] R0: [kworker/0:0][00:11:17.878139] wlan:
[4:F :WDA] WMA <-- WMI_READY_EVENTID WLAN Reserved 10/Medium [4516/0001]
4199/11/24 01:34:15.631}

{MSG [cld-diag-parser.c  1657] R0: [kworker/0:0][00:11:17.878348] wlan: [4:
D:WDA] wma_update_target_ht_cap: ht_cap_info - 85b ht_rx_stbc - 1, ht_tx_stbc -
1, mpdu_density - 0 ht_rx_ldpc - 1 ht_sgi_20 - 1, ht_sgi_40 - 1 num_rf_chains -
2  WLAN Reserved 10/Medium [4516/0001] 4199/11/24 01:34:15.633}
```

6.3 Using the bugreport

1. Turn on Wi-Fi.
2. Collect the bugreport by issuing the `$ adb bugreport | tee bugreport.txt` command.

6.3.1 Bugreport snippet host logs:

```
<6>[  258.160032] wlan: [5688:F :WDA] P2P Device: removing self peer
8e:fd:f0:01:80:e4

<6>[  258.160840] TXRX: ol_txrx_peer_detach:peer ffffffc04d3f0800
(8e:fd:f0:01:80:e4)

<6>[  258.160927] wlan: [5688:E :WDA] wma_remove_peer: Removed peer with addr
8e:fd:f0:01:80:e4 vdevid 1 peer_count 0

<6>[  258.166250] TXRX: ol_rx_peer_unmap_handler: Remove the ID 72 reference to
peer ffffffc04d3f0800

<6>[  258.166780] TXRX: Deleting peer ffffffc04d3f0800 (8e:fd:f0:01:80:e4)

<6>[  258.169175] wlan: [5688:E :HDD] hdd_ndp_session_end_handler: 936: Adapter
is not in NDI mode

<6>[  258.170539] wlan: [5688:E :WDA] wma_unified_vdev_create_send: ID = 1 VAP
Addr = 8e:fd:f0:01:80:e4
```

Check for the WLAN logs as highlighted previously and the prints like VOS, WDA and WMA corresponds to host driver modules.

Connectivity stats and power events can be decoded from bugreport using QCOM proprietary parsing tool.

6.4 Power consumption impacts

In some cases there is a power penalty to memory dumps.

Power collapse for firmware does not happen between these two instances: The point when the scratch buffer updates begin up to the time when the host receives the SMD messages about the DMA operation status.

In such cases, scenario-based and firmware memory dumps introduce a power impact. So, in the event of a host or fatal scenario, invoking a firmware memory dump from the host or firmware flushing logic does not resolve the APPS CPU power state.

6.5 Sample codes for integration

The sample codes are in <AU_ROOT>/vendor/qcom/proprietary/wlan/utils/halproxydaemon/src/wifilogger_test.cpp file.

To determine what information to collect to help fix the issue, do the following:

1. Initialize the debug framework (see Section 4.1)
2. Decide which events trigger the bug-report mechanism.