contents

What is QMAP

QMAP's full name is QUALCOMM Multiplexing and Aggregation Protocol.

When using GobiNet or QMI_WWAN, only one Physical Network Card can be created by default, so only one PDN data call can be set up. However, multiple virtual Network Cards can be created by using IP multiplexing protocol over one Physical Network card, and customers can setup multiple PDN data calls.

When using GobiNet or QMI_WWAN, only one IP Packet in one URB can be transferred, so when there are high throughput and frequent URB interrupts, the Host CPU will become overloaded. However, IP aggregation protocol can be used to transfer multiple IP Packets in one URB with increased throughput by reducing the number of URB interrupts.

When QMAP disabled, GobiNet or QMI_WWAN directly transfer IP Packet over USB BUS.
When QMAP enabled, GobiNet or QMI_WWAN transfer QMAP Packet over USB BUS.

EC21/EC25/EG06/EP06/EM06/EG12/EP12/EM12/EG16/EG18/EM20/RG500 all support QMAP.

IP Aggregation by QMI_WWAN

qmi_wwan_q.c is driver like qmi_wwan.c, can work on all Linux kernel that's version larger than or equal to V3.4, and can coexistence with qmi_wwan.c.

Please follow next steps to use IP Aggregation.

1. Porting qmi_wwan_q.c to your kernel.
   [KERNEL]/drivers/net/usb/Makefile

```
obj-y      += qmi_wwan_q.o
obj-$(CONFIG_USB_NET_QMI_WWAN) += qmi_wwan.o
```

2. Modify macro QUECTEL_WWAN_QMAP in qmi_wwan_q.c to 1.

```
/*
    Quectel_WCDMA&LTE_Linux_USB_Driver_User_Guide_V1.9.pdf
    5.6.  Test QMAP on GobiNet or QMI WWAN
    0 - no QMAP
    1 - QMAP (Aggregation protocol)
    X - QMAP (Multiplexing and Aggregation protocol)
*/
#define QUECTEL_WWAN_QMAP 1
```

And the dmesg log as next:

```
[240330.371864] usbcore: registered new interface driver qmi_wwan_q
[240332.720418] usb 2-1.2: new high-speed USB device number 21 using sunxi-ehci
[240332.840162] usb 2-1.2: New USB device found, idVendor=2c7c, idProduct=0435
[240332.840188] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[240332.840206] usb 2-1.2: Product: LTE Module
[240332.840223] usb 2-1.2: Manufacturer: Quectel, Incorporated
[240332.888377] qmi_wwan_q 2-1.2:1.4: cdc-wdm0: USB WDM device
[240332.888408] qmi_wwan_q 2-1.2:1.4: Quectel modules work on RawIP mode
[240332.892181] qmi_wwan_q 2-1.2:1.4 wwan0: register 'qmi_wwan_q' at usb-sunxi-ehci-1.2, WWAN/QMI device, 2e:75:af:df:48:50
[240332.892214] qmi_wwan_q 2-1.2:1.4: rx_urb_size = 4096
```

rx_urb_size indicate the MAX size of QMAP Packet.
rx_urb_size of EC20&EC25 is 4Kbytes. rx_urb_size of EM06&EM12 is 16Kbytes, rx_urb_size of EM20&RG500 is 32Kbytes.

3. Use Quectel-CM to setup data call

```
# ./quectel-CM &
[06-04_03:52:20:259] WCDMA&LTE_QConnectManager_Linux&Android_V1.3.3
[06-04_03:52:20:260] ./quectel-CM profile[1] = (null)/(null)/(null)/0, pincode = (null)
[06-04_03:52:20:262] Find /sys/bus/usb/devices/2-1.2 idVendor=2c7c idProduct=0435
[06-04_03:52:20:262] Find /sys/bus/usb/devices/2-1.2:1.4/net/wwan0
[06-04_03:52:20:262] Find usbnet_adapter = wwan0
[06-04_03:52:20:262] Find /sys/bus/usb/devices/2-1.2:1.4/usbmisc/cdc-wdm0
[06-04_03:52:20:262] Find qmichannel = /dev/cdc-wdm0
[06-04_03:52:20:265] qmap_mode = 1, muxid = 0x81, qmap_netcard = wwan0
[06-04_03:52:20:285] cdc_wdm_fd = 7
[06-04_03:52:22:358] Get clientWDS = 17
[06-04_03:52:22:390] Get clientDMS = 1
[06-04_03:52:22:422] Get clientNAS = 3
[06-04_03:52:22:454] Get clientUIM = 1
[06-04_03:52:22:486] Get clientWDA = 1
[06-04_03:52:22:518] requestBaseBandVersion AG35CEVAR05A06T4G
[06-04_03:52:22:550] qmap_settings.rx_urb_size = 4096
[06-04_03:52:22:678] requestGetSIMStatus SIMStatus: SIM_READY
[06-04_03:52:22:710] requestGetProfile[1] cmnet///0
[06-04_03:52:22:742] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_03:52:22:774] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[06-04_03:52:22:838] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_03:52:22:870] requestSetupDataCall WdsConnectionIPv4Handle: 0x86e9a010
[06-04_03:52:22:934] requestQueryDataCall IPv4ConnectionStatus: CONNECTED
[06-04_03:52:22:966] ifconfig wwan0 up
[06-04_03:52:22:981] busybox udhcpc -f -n -q -t 5 -i wwan0
udhcpc: started, v1.27.2
udhcpc: sending discover
udhcpc: sending select for 10.197.44.37
udhcpc: lease of 10.197.44.37 obtained, lease time 7200
[06-04_03:52:23:308] /etc/udhcpc/default.script: Resetting default routes
[06-04_03:52:23:320] /etc/udhcpc/default.script: Adding DNS 211.138.180.2
[06-04_03:52:23:320] /etc/udhcpc/default.script: Adding DNS 211.138.180.3
```

IP Multiplexing by QMI_WWAN

Please follow next steps to use IP Multiplexing.

1. Porting qmi_wwan_q.c to your kernel.
   [KERNEL]/drivers/net/usb/Makefile

```
obj-y      += qmi_wwan_q.o
obj-$(CONFIG_USB_NET_QMI_WWAN) += qmi_wwan.o
```

2. Modify macro QUECTEL_WWAN_QMAP in qmi_wwan_q.c to 4.

```
/*
      Quectel_WCDMA&LTE_Linux_USB_Driver_User_Guide_V1.9.pdf
      5.6.  Test QMAP on GobiNet or QMI WWAN
      0 - no QMAP
      1 - QMAP (Aggregation protocol)
      X - QMAP (Multiplexing and Aggregation protocol)
*/
#define QUECTEL_WWAN_QMAP 4
```

And the dmesg log as next:
Qmi_wwan_q will create four network adapter named as wwan0.X.
wwan0.1 for data call setup on PDN-1
wwan0.2 for data call setup on PDN-2
wwan0.3 for data call setup on PDN-3
wwan0.4 for data call setup on PDN-4

```
[244838.067277] usbcore: registered new interface driver qmi_wwan_q
[244844.970420] usb 2-1.2: new high-speed USB device number 22 using sunxi-ehci
[244845.090167] usb 2-1.2: New USB device found, idVendor=2c7c, idProduct=0435
[244845.090193] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[244845.090212] usb 2-1.2: Product: LTE Module
[244845.090229] usb 2-1.2: Manufacturer: Quectel, Incorporated
[244845.129638] qmi_wwan_q 2-1.2:1.4: cdc-wdm0: USB WDM device
[244845.129666] qmi_wwan_q 2-1.2:1.4: Quectel modules work on RawIP mode
[244845.135800] qmi_wwan_q 2-1.2:1.4 wwan0: register 'qmi_wwan_q' at usb-sunxi-ehci-1.2, WWAN/QMI device, 2e:75:af:df:48:50
[244845.140199] net wwan0: qmap_register_device wwan0.1
[244845.142132] net wwan0: qmap_register_device wwan0.2
[244845.143211] net wwan0: qmap_register_device wwan0.3
[244845.144309] net wwan0: qmap_register_device wwan0.4
[244845.144329] qmi_wwan_q 2-1.2:1.4: rx_urb_size = 4096
```

3. Run quectel-qmi-proxy as next.

```
root@cqh6:~# ./quectel-qmi-proxy -d /dev/cdc-wdm0 &
Will use cdc-wdm /dev/cdc-wdm0
qmi_proxy_init enter
qmi_proxy_loop enter thread_id 548403548656
link_prot 2
ul_data_aggregation_protocol 5
dl_data_aggregation_protocol 5
dl_data_aggregation_max_datagrams 10
dl_data_aggregation_max_size 4096
ul_data_aggregation_max_datagrams 16
ul_data_aggregation_max_size 4096
qmi_proxy_init finished, rx_urb_size is 4096
local server: quectel-qmi-proxy sockfd = 4
qmi_start_server: qmi_proxy_server_fd = 4
```

4. Use Quectel-CM to setup data call

   Argument '-n X' indicate which PDN want to setup data call on.

   For example: you can setup two data calls on PDN-1 and PDN-2 by next command:

   #./quectel-CM –n 1 &

   #./quectel-CM –n 2 &

   #

   Next log is setup data call on PDN-2.

```
# ./quectel-CM -n 2 &
[06-04_05:13:16:473] WCDMA&LTE_QConnectManager_Linux&Android_V1.3.3
[06-04_05:13:16:473] ./quectel-CM profile[2] = (null)/(null)/(null)/0, pincode = (null)
[06-04_05:13:16:475] Find /sys/bus/usb/devices/2-1.2 idVendor=2c7c idProduct=0435
[06-04_05:13:16:476] Find /sys/bus/usb/devices/2-1.2:1.4/net/wwan0
[06-04_05:13:16:476] Find usbnet_adapter = wwan0
[06-04_05:13:16:476] Find /sys/bus/usb/devices/2-1.2:1.4/usbmisc/cdc-wdm0
[06-04_05:13:16:476] Find qmichannel = /dev/cdc-wdm0
[06-04_05:13:16:477] qmap_mode = 4, muxid = 0x82, qmap_netcard = wwan0.2
[06-04_05:13:16:477] connect to quectel-qmi-proxy sockfd = 7
[06-04_05:13:16:478] cdc_wdm_fd = 7
[06-04_05:13:16:535] Get clientWDS = 17
[06-04_05:13:16:567] Get clientDMS = 1
[06-04_05:13:16:599] Get clientNAS = 3
[06-04_05:13:16:631] Get clientUIM = 1
[06-04_05:13:16:663] requestBaseBandVersion AG35CEVAR05A06T4G
[06-04_05:13:16:791] requestGetSIMStatus SIMStatus: SIM_READY
[06-04_05:13:16:823] requestGetProfile[2] ///0
[06-04_05:13:16:855] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_05:13:16:887] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[06-04_05:13:16:950] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_05:13:16:983] requestSetupDataCall WdsConnectionIPv4Handle: 0x86e513b0
[06-04_05:13:17:047] requestQueryDataCall IPv4ConnectionStatus: CONNECTED
[06-04_05:13:17:079] ifconfig wwan0 up
[06-04_05:13:17:089] ifconfig wwan0.2 up
[06-04_05:13:17:100] busybox udhcpc -f -n -q -t 5 -i wwan0.2
udhcpc: started, v1.27.2
udhcpc: sending discover
udhcpc: sending select for 10.247.97.242
udhcpc: lease of 10.247.97.242 obtained, lease time 7200
[06-04_05:13:17:449] /etc/udhcpc/default.script: Resetting default routes
[06-04_05:13:17:465] /etc/udhcpc/default.script: Adding DNS 211.138.180.2
[06-04_05:13:17:465] /etc/udhcpc/default.script: Adding DNS 211.138.180.3
```

IP Aggregation by GobiNet

Please follow next steps to use IP Aggregation.

1. Modify variable qmap_mode in GobiUSBNet.c to 1.

```
/*
	Quectel_WCDMA&LTE_Linux_USB_Driver_User_Guide_V1.9.pdf
	5.6.  Test QMAP on GobiNet or QMI WWAN
	0 - no QMAP
	1 - QMAP (Aggregation protocol)
	X - QMAP (Multiplexing and Aggregation protocol)
*/
static uint __read_mostly qmap_mode = 1;
module_param( qmap_mode, uint, S_IRUGO | S_IWUSR );
```

And the dmesg log as next:

```
[247029.696762] GobiNet: Quectel_WCDMA&LTE_Linux&Android_GobiNet_Driver_V1.5.0
[247029.697179] usbcore: registered new interface driver GobiNet
[247034.030445] usb 2-1.2: new high-speed USB device number 24 using sunxi-ehci
[247034.150166] usb 2-1.2: New USB device found, idVendor=2c7c, idProduct=0435
[247034.150194] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[247034.150212] usb 2-1.2: Product: LTE Module
[247034.150229] usb 2-1.2: Manufacturer: Quectel, Incorporated
[247034.200189] GobiNet 2-1.2:1.4 usb0: register 'GobiNet' at usb-sunxi-ehci-1.2, GobiNet Ethernet Device, 2e:75:af:df:48:50
[247034.201517] creating qcqmi0
[247044.490691] GobiNet::QMIWDASetDataFormat qmap settings qmap_enabled=5, rx_size=4096, tx_size=4096
```

2. Use Quectel-CM to setup data call

```
# ./quectel-CM &
[06-04_05:41:33:828] WCDMA&LTE_QConnectManager_Linux&Android_V1.3.3
[06-04_05:41:33:828] ./quectel-CM profile[1] = (null)/(null)/(null)/0, pincode = (null)
[06-04_05:41:33:830] Find /sys/bus/usb/devices/2-1.2 idVendor=2c7c idProduct=0435
[06-04_05:41:33:830] Find /sys/bus/usb/devices/2-1.2:1.4/net/usb0
[06-04_05:41:33:830] Find usbnet_adapter = usb0
[06-04_05:41:33:831] Find /sys/bus/usb/devices/2-1.2:1.4/GobiQMI/qcqmi0
[06-04_05:41:33:831] Find qmichannel = /dev/qcqmi0
[06-04_05:41:33:831] qmap_mode = 1, muxid = 0x81, qmap_netcard = usb0
[06-04_05:41:33:881] Get clientWDS = 7
[06-04_05:41:33:913] Get clientDMS = 8
[06-04_05:41:33:945] Get clientNAS = 9
[06-04_05:41:33:977] Get clientUIM = 10
[06-04_05:41:34:009] requestBaseBandVersion AG35CEVAR05A06T4G
[06-04_05:41:34:137] requestGetSIMStatus SIMStatus: SIM_READY
[06-04_05:41:34:169] requestGetProfile[1] cmnet///0
[06-04_05:41:34:201] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_05:41:34:233] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[06-04_05:41:34:297] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_05:41:34:330] requestSetupDataCall WdsConnectionIPv4Handle: 0x86e6d550
[06-04_05:41:34:392] requestQueryDataCall IPv4ConnectionStatus: CONNECTED
[06-04_05:41:34:426] ifconfig usb0 up
[06-04_05:41:34:439] busybox udhcpc -f -n -q -t 5 -i usb0
udhcpc: started, v1.27.2
udhcpc: sending discover
udhcpc: sending select for 10.239.220.134
udhcpc: lease of 10.239.220.134 obtained, lease time 7200
[06-04_05:41:34:759] /etc/udhcpc/default.script: Resetting default routes
[06-04_05:41:34:775] /etc/udhcpc/default.script: Adding DNS 211.138.180.2
[06-04_05:41:34:775] /etc/udhcpc/default.script: Adding DNS 211.138.180.3
```

IP Multiplexing by GobiNet

Please follow next steps to use IP Multiplexing.

1. Modify variable qmap_mode in GobiUSBNet.c to 4.

```
/*
      Quectel_WCDMA&LTE_Linux_USB_Driver_User_Guide_V1.9.pdf
      5.6.  Test QMAP on GobiNet or QMI WWAN
      0 - no QMAP
      1 - QMAP (Aggregation protocol)
      X - QMAP (Multiplexing and Aggregation protocol)
*/
static uint __read_mostly qmap_mode = 4;
module_param( qmap_mode, uint, S_IRUGO | S_IWUSR );
```

And the dmesg log as next:
Qmi_wwan_q will create four network adapter named as usb0.X.
usb0.1 for data call setup on PDN-1
usb0.2 for data call setup on PDN-2
usb0.3 for data call setup on PDN-3
usb0.4 for data call setup on PDN-4

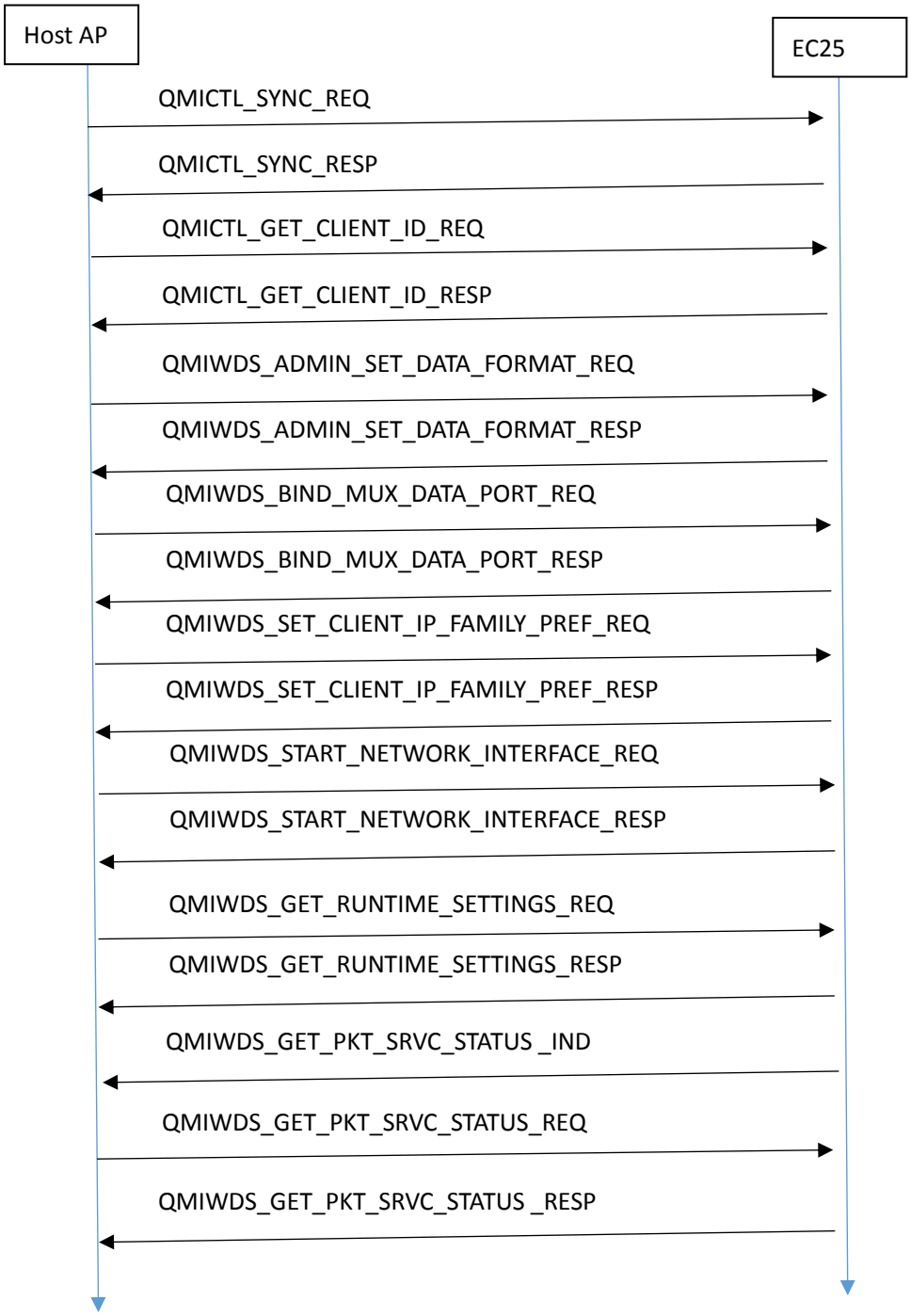
```
[247365.213865] GobiNet: Quectel_WCDMA&LTE_Linux&Android_GobiNet_Driver_V1.5.0
[247365.214209] usbcore: registered new interface driver GobiNet
[247369.390423] usb 2-1.2: new high-speed USB device number 25 using sunxi-ehci
[247369.510166] usb 2-1.2: New USB device found, idVendor=2c7c, idProduct=0435
[247369.510193] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[247369.510212] usb 2-1.2: Product: LTE Module
[247369.510229] usb 2-1.2: Manufacturer: Quectel, Incorporated
[247369.560313] GobiNet 2-1.2:1.4 usb0: register 'GobiNet' at usb-sunxi-ehci-1.2, GobiNet Ethernet Device, 2e:75:af:df:48:50
[247369.561690] creating qcqmi0
[247369.563154] GobiNet::qmap_register_device usb0.1
[247369.564861] GobiNet::qmap_register_device usb0.2
[247369.566014] GobiNet::qmap_register_device usb0.3
[247369.567818] GobiNet::qmap_register_device usb0.4
[247379.851678] GobiNet::QMIWDASetDataFormat qmap settings qmap_enabled=5, rx_size=4096, tx_size=4096
```

2. Use Quectel-CM to setup data call
   Argument '-n X' indicate which PDN want to setup data call on.
   For example: you can setup two data calls on PDN-1 and PDN-2 by next command:
   #./quectel-CM –n 1 &
   #./quectel-CM –n 2 &
   #
   Next log is setup data call on PDN-2.

```
# ./quectel-CM -n 2 &
  [06-04_05:52:13:869] WCDMA&LTE_QConnectManager_Linux&Android_V1.3.3
[06-04_05:52:13:869] ./quectel-CM profile[2] = (null)/(null)/(null)/0, pincode = (null)
[06-04_05:52:13:870] Find /sys/bus/usb/devices/2-1.2 idVendor=2c7c idProduct=0435
[06-04_05:52:13:870] Find /sys/bus/usb/devices/2-1.2:1.4/net/usb0
[06-04_05:52:13:870] Find usbnet_adapter = usb0
[06-04_05:52:13:870] Find /sys/bus/usb/devices/2-1.2:1.4/GobiQMI/qcqmi0
[06-04_05:52:13:870] Find qmichannel = /dev/qcqmi0
[06-04_05:52:13:870] qmap_mode = 4, muxid = 0x82, qmap_netcard = usb0.2
[06-04_05:52:13:882] Get clientWDS = 7
[06-04_05:52:13:914] Get clientDMS = 8
[06-04_05:52:13:946] Get clientNAS = 9
[06-04_05:52:13:978] Get clientUIM = 10
[06-04_05:52:14:010] requestBaseBandVersion AG35CEVAR05A06T4G
[06-04_05:52:14:138] requestGetSIMStatus SIMStatus: SIM_READY
[06-04_05:52:14:170] requestGetProfile[2] ///0
[06-04_05:52:14:202] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_05:52:14:234] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[06-04_05:52:14:298] requestRegistrationState2 MCC: 460, MNC: 0, PS: Attached, DataCap: LTE
[06-04_05:52:15:002] requestSetupDataCall WdsConnectionIPv4Handle: 0x86e4e3e0
[06-04_05:52:15:066] requestQueryDataCall IPv4ConnectionStatus: CONNECTED
[06-04_05:52:15:098] ifconfig usb0 up
[06-04_05:52:15:118] ifconfig usb0.2 up
[06-04_05:52:15:130] busybox udhcpc -f -n -q -t 5 -i usb0.2
udhcpc: started, v1.27.2
udhcpc: sending discover
udhcpc: sending select for 10.220.124.20
udhcpc: lease of 10.220.124.20 obtained, lease time 7200
[06-04_05:52:15:468] /etc/udhcpc/default.script: Resetting default routes
[06-04_05:52:15:480] /etc/udhcpc/default.script: Adding DNS 211.138.180.2
[06-04_05:52:15:480] /etc/udhcpc/default.script: Adding DNS 211.138.180.3
```

QMAP QMI Flow

If the customer develop QMI Tool and QMAP Driver by self, can refer to next chapters.



QMAP QMI Details

1. QMICTL_SYNC_REQ
   Only need send one time when EC25 boot up.
   SYNC_REQ will reset ALL QMI State in EC25, it means release all QMI Client and ALL data Call.

2. QMICTL_SYNC_RESP

3. QMICTL_GET_CLIENT_ID_REQ
   Should get QMI_WDA and QMI_WDS 's CLIENT ID.
   If multiple PND data calls, and IPV4 and IPV6 data calls required. Must get ONE CLIENT for E DATA CALL.
   For example:
   IPV4 Call on PDN-1 use QMI_WDS Client-1.
   IPV6 Call on PDN-1 use QMI_WDS Client-2.
   IPV4 Call on PDN-2 use QMI_WDS Client-3.
   IPV6 Call on PDN-2 use QMI_WDS Client-4

4. QMICTL_GET_CLIENT_ID_RESP

5. QMIWDS_ADMIN_SET_DATA_FORMAT_REQ
   Only need send one time when module boot up

tag	length	value	Description
0x10	uint8	0x00	Configured QOS data format. Values: • 0 – QOS flow header is not present(Default) • 1 – QOS flow header is present
0x11	uint32	0x02	Link protocol used by the client: • 0x01 – 802.3 Ethernet mode (Default) • 0x02 – IP mode
0x12	uint32	0x05	Uplink (UL) data aggregation protocol to be used for uplink data transfer. Values: • 0x05 – UL QMAP is enabled
0x13	uint32	0x05	Downlink (DL) data aggregation protocol to be used for downlink data transfer. Values: • 0x05 – DL QMAP is enabled
0x15	uint32	64	Maximum number of datagrams in a single aggregated packet on downlink. The value applies to all downlink data aggregation protocols when downlink data aggregation is enabled. Zero means no limit.
0x16	uint32	0x8000	Maximum size in bytes of a single aggregated packet allowed on downlink. The value applies to all downlink data aggregation protocols when downlink data aggregation is enabled.
0x17	uint32	0x02	Peripheral endpoint type. Values: • DATA_EP_TYPE_HSUSB (0x02) – High-speed universal serial bus • DATA_EP_TYPE_PCIE (0x03) – Peripheral

			component interconnect express
	uint32	0x04	Peripheral interface number.

6. QMIWDS_ADMIN_SET_DATA_FORMAT_RESP

tag	length	value	Description
0x16	uint32		Downlink Data Aggregation Max Size

Must make sure 'size for rx urbs' must larger than or equal to this TLV.
For Linux USBNET Driver. It is
include\linux\usb\usbnet.h

```
struct usbnet {
    size_t rx_urb_size;  /* size for rx urbs */
}
```

7. QMIWDS_BIND_MUX_DATA_PORT_REQ

tag	length	value	Description
0x10	uint32	0x02	Peripheral endpoint type. Values: • DATA_EP_TYPE_HSUSB (0x02) – High-speed universal serial bus • DATA_EP_TYPE_PCIE (0x03) – Peripheral component interconnect express
	uint32	0x04	Peripheral interface number.
0x11	uint8		Mux ID of the PDN to which the client binds. The default value is 0. 0x81 for PDN-1 0x82 for PND-2 …… 0x8X for PND-X For example: If you want to setup IPV4/IPV6 data call on PDN-X, you should first get a QMI_WDS Client, and this client must bind to MUX ID 0x8X by this QMI.
0x13	uint32	0x01	Type of the client that requests the binding. Values: • WDS_CLIENT_TYPE_RESERVED(0) – Reserved • WDS_CLIENT_TYPE_TETHERED(1) – Tethered

When QMAP enabled, qmap_hdr will insert before IP Packet,

```
struct qmap_hdr {
    uint8 cd_rsvd_pad;
    uint8 mux_id; //same as TLV 0x11
    uint16 pkt_len; //Length of IP Packet.
} __packed;
```

8. QMIWDS_BIND_MUX_DATA_PORT_RESP

9. QMIWDS_SET_CLIENT_IP_FAMILY_PREF_REQ

tag	length	value	Description
0x01	uint8		IP Family Preference • WDS_IP_FAMILY_IPV4 (0x04) – IPv4 • WDS_IP_FAMILY_IPV6 (0x06) – IPv6

10. QMIWDS_SET_CLIENT_IP_FAMILY_PREF_RESP

11. QMIWDS_START_NETWORK_INTERFACE_REQ

tag	length	value	Description
0x30	uint8	0x00	technology preferences: • 0x00 – 3GPP • 0x01 – 3GPP2
0x14	string	strlen()	Context Access Point Node (APN) Name
0x17	string	strlen()	Username to use during data network authentication
0x18	string	strlen()	Password used during data network authentication.
0x15	uint8		Authentication preference • 0 – None • 1 – PAP • 2 – CHAP
0x19	uint8		IP family preference • 4 – IPv4 • 6 – IPv6

If 'apn/user/password/auth' already set by AT Command at+cgdcont and at+qicgsp, can be ignored in this QMI.

12. QMIWDS_START_NETWORK_INTERFACE_RESP

tag	length	value	Description
0x01	uint32		Packet Data Handle The handle identifying the call instance providing packet service. The packet data handle must be retained by the control point and specified in the STOP_NETWORK_INTERFACE message issued when the control point is finished with the packet data session.

13. QMIWDS_GET_RUNTIME_SETTINGS_REQ

tag	length	value	Description
0x10	uint32	0x2310	Requested Settings Set bits to 1, corresponding to requested information. All other bits must be set to 0. If the values are not available, the corresponding TLVs are not returned in the response. Absence of this mask TLV results in the device returning all of the available information corresponding to bits 0 through 12. In cases where the information from bit 13 or greater is required, this TLV with all the necessary bits set must be present in the request.

			Values: • Bit 0 – Profile identifier • Bit 1 – Profile name • Bit 2 – PDP type • Bit 3 – APN name • Bit 4 – DNS address • Bit 5 – UMTS/GPRS granted QoS • Bit 6 – Username • Bit 7 – Authentication Protocol • Bit 8 – IP address • Bit 9 – Gateway information (addressand subnet mask) • Bit 10 – PCSCF address using a PCO flag • Bit 11 – PCSCF server address list • Bit 12 – PCSCF domain name list • Bit 13 – MTU • Bit 14 – Domain name list • Bit 15 – IP family • Bit 16 – IM_CM flag • Bit 17 – Technology name • Bit 18 – Operator reserved PCO

14. QMIWDS_GET_RUNTIME_SETTINGS_RESP

tag	length	value	Description
0x15	uint32		Primary DNS Address
0x16	uint32		Secondary DNS Address
0x1E	uint32		IPv4 Address
0x20	uint32		IPv4 Gateway Address
0x21	uint32		IPv4 Subnet Mask
0x25	struct { uint8 IPV6Address[16]; uint8 PrefixLength; } __attribute__ ((packed))		IPv6 Address
0x26	struct { uint8 IPV6Address[16]; uint8 PrefixLength; } __attribute__ ((packed))		IPv6 Gateway Address
0x27	struct { uint8 IPV6Address[16]; uint8 PrefixLength; } __attribute__ ((packed))		Primary IPv6 DNS Address
0x28	struct { uint8 IPV6Address[16]; uint8 PrefixLength; } __attribute__ ((packed))		Secondary IPv6 DNS Address
0x29	uint32		MTU

Must get MTU from this QMI RESP, and make sure it is same as your net card‘s MTU setting.

When no data exits, will get QMI RESP as next

{02, 0004, 01 00 0f 00 }

0x000f means QMI_ERR_OUT_OF_CALL

15. QMIWDS_GET_PKT_SRVC_STATUS_IND

tag	length	value	Description
0x01	uint8		Packet Service Status Current link status. Values: • WDS_CONNECTION_STATUS_DISCONNECTED (0x01) – Disconnected • WDS_CONNECTION_STATUS_CONNECTED (0x02) – Connected • WDS_CONNECTION_STATUS_SUSPENDED (0x03) – Suspended • WDS_CONNECTION_STATUS_AUTHENTICATING (0x04) – Authenticating
	uint8		Indicates whether the network interface on the host must be reconfigured. Values: • 0 – Not necessary to reconfigure • 1 – Reconfiguration required

When data call state change (for example, terminated by Network Operator). EC25 will auto and immediately report this QMI IND.

16. QMIWDS_GET_PKT_SRVC_STATUS_REQ

17. QMIWDS_GET_PKT_SRVC_STATUS_RESP

tag	length	value	Description
0x01	uint8		Packet Service Status Current link status. Values: • WDS_CONNECTION_STATUS_DISCONNECTED (0x01) – Disconnected • WDS_CONNECTION_STATUS_CONNECTED (0x02) – Connected • WDS_CONNECTION_STATUS_SUSPENDED (0x03) – Suspended • WDS_CONNECTION_STATUS_AUTHENTICATING (0x04) – Authenticating
	uint8		Indicates whether the network interface on the host must be reconfigured. Values: • 0 – Not necessary to reconfigure • 1 – Reconfiguration required

QMAP Header and Data Packet

```
struct qmap_hdr {
        uint8 cd_rsvd_pad;
        uint8 mux_id;
        uint16 pkt_len;
} __packed;
```

- When QMAP enabled, QMAP header is prepended to an IP packet, each IP packet has its own QMAP header.
- Following QMAP header is used for data packets:
  1. C/D bit – 1 bit; set to 0
  2. RESERVED – 1 bit; set to 0; must be ignored by the receiver
  3. MUX_ID – 8 bits; indicates which PDN the correspond IP Packet belong to.
     For details, refer to QMIWDS_BIND_MUX_DATA_PORT_REQ
  4. PAD – 6 bits; indicates number of bytes padded to achieve a minimum of 4 byte alignment
  5. PACKET_LEN_WITH_PADDING – 16 bits; total packet length in bytes including padding, Length is from the start of the IP header