SYSLIB Unit Test

Rev 1.7

Texas Instruments, Incorporated
20450 Century Boulevard
Germantown, MD 20874 USA

Document License

Contributors to this document

Revision Record	
Document Title: **Software Design Specification**	
Revision	Description of Change
1.0	Initial Draft
1.1	Updated for alpha-9
1.2	Updated for alpha-9 Drop5
1.3	Updated for alpha-10
1.4	Updated for alpha-12
1.5	Updated for 4.0.1.0
1.6	Updated for 4.0.3.0
1.7	Updated for 4.0.4.0

Note: Be sure the Revision of this document matches the QRSA record Revision letter.

1. Contents

1 Introduction

The document provides the description of the unit tests for the SYSLIB. This includes the descriptions on the test setup and instructions on each test execution.

2 Prerequisites

The SYSLIB package supports both the DSP and ARM execution realms. Each module provides the appropriate unit for the realm as applicable.

The section documents in details the test prerequisites that need to be completed

Prerequisites:

1. Please ensure that the tools specified in the MCSDK release package have been installed. For ARM development please ensure that the MCSDK RT `devkit` is installed.
2. Please manually install any MCSDK patches (if applicable) as documented in the release notes.
3. Please upgrade the kernel & SYSLIB RMv2 DTB files
4. Build the ARM & DSP Unit Tests. This is documented in the Release Notes.
5. Download the SYSLIB applications and configuration files. Please refer to the Device specific section in the release notes. Certain SYSLIB applications have device specific configuration files.

2.1 DTB File upgrade

The SYSLIB package modifies the default MCSDK kernel DTB file. The following procedure can be used for this purpose:-

1. SYSLIB releases the kernel DTS files under the SYSLIB_INSTALL_PATH/ti/runtime/resmgr/dts/<device>
2. TFTP all the DTS and DTSI files from the directory on the board
3. First time only create a temporary directory as follows:-

```
mkdir /mnt/boot
```

4. Use the following commands to create and update the kernel DTB file

```
mount -t ubifs ubi0_0 /mnt/boot
rm /mnt/boot/uImage-k2hk-evm.dtb
dtc -I dts -O dtb k2hk-evm.dts > /mnt/boot/uImage-k2hk-evm.dtb
umount /mnt/boot
```

```
```

5. Use the following commands to create and update the SYSLIB RMv2 DTB files

```
tftp 192.168.1.10 -g -r runtime/resmgr/dts/k2h/policy_dsp_arm_syslib.dts
tftp 192.168.1.10 -g -r runtime/resmgr/dts/k2h/global-resource-list.dts
dtc -I dts -O dtb policy_dsp_arm_syslib.dts > policy_dsp_arm_syslib.dtb
dtc -I dts -O dtb global-resource-list.dts > global-resource-list.dtb
```

6. Reboot the box

2.2 SYSLIB Applications

The section documents the various SYSLIB applications which are provided and how these need to be executed

2.2.1 Resource Manager

All unit tests rely on the RESMGR services and require the resource manager to be operational. The RESMGR server is provided as an executable which is a part of the standard release package.

Please refer to the Getting started section in the UG for more details on the setup of the DTB files for the RM Server as well as the kernel.

Download the resource manager server located in the following directory

```
SYSLIB/ti/apps/resmgr_server/resmgr_server_<device>.out
```

The resource manager executable file can be downloaded onto the target. The resource manager requires the following kernel modules to be inserted:

- HPLIB
- UIO

These modules are a part of the default MCSDK file system. Please refer to the MCSDK UG for the exact location of these modules. The UIO module is loaded by default; the HPLIB module will need to be loaded manually. The following command can be used to load the HPLIB module

```
insmod /lib/modules/3.10.10-rt7/extra/hplibmod.ko
```

```
```

NOTE: The HPLIB module location is subject to change with kernel upgrades.

In order for the IPv6 Tunnels to work the IPSEC manager kernel module also needs to be loaded. This module will allow ESP6 packets to flow through the system.

```
insmod /lib/modules/3.10.10-rt7/extra/ipsecmgr_mod.ko
```

NOTE: The IPSEC manager module location is subject to change with kernel upgrades.

Once the modules have been loaded; the resource manager server can be launched. The resource manager server can be configured using command line options. Please refer to the RESMGR documentation for more details about the individual command line options. The following is an example command sequence which can be used to launch the RESMGR server

```
./resmgr_server_<device>.out -n Rm_Server -g global-resource-list.dtb -s 17 -p
policy_dsp_arm_syslib.dtb &
```

2.2.2 SOC Initialization

SOC Initialization is a system service which needs to be executed along with the RESMGR Server before any other test can be executed

Download the SYSLIB SOC Initialization application located in the following directory

```
SYSLIB/ti/apps/soc_init/soc_init_<device>.out
```

NOTE: Please ensure that the resource manager server is running before launching the SYSLIB SOC Initialization application.

Sample configuration files are located in the `ti/apps/soc_init` directory. The following is an example to start the execution of the SOC Initialization application on K2H.

```
./soc_init_k2h.out -r Rm_System -c ./soc_k2h.conf
```

2.2.3 NETFP Master

NETFP master is a system service which needs to be executed along with the RESMGR Server and SOC Initialization application before any other test can be executed

Download the NETFP master located in the following directory

```
SYSLIB/ti/apps/netfp_master/netfp_master_<device>.out
```

NOTE: Please ensure that the resource manager server is running before launching the NETFP master.

The NETFP master configuration can be specified using a configuration file. Sample configuration files are located in the `ti/apps/netfp_master/netfp.conf`.

The following is an example to start the execution of the NETFP master on K2L

```
./netfp_master_k2l.out -r Rm_System -c ./netfp_k2l.conf &
```

2.2.4 Name Proxy

Name proxies are required to execute tests which showcase coexistence between the ARM and DSP realms. The Name proxy on ARM is provided as an executable which is a part of the standard release package.

Download the name proxy located in the following directory

```
SYSLIB/ti/apps/name_proxy/name_proxy_<device>.out
```

NOTE: Please ensure that the resource manager server is running before launching the Name proxy.

The name proxy can be configured using command line options. Please refer to the Name proxy documentation for more details about the individual command line options. The following is an example to start the execution of the name proxy.

```
./name_proxy_<device>.out -n NameServer_LTE9A -c Rm_LTE9A -i 1 -l 0 -r 1 -a
0xA001F000 &
```

2.2.5 NETFP Server

NETFP servers is required to execute the NETFP tests (ARM only, DSP & ARM coexistence)

Download the NETFP server located in the following directory

```
SYSLIB/ti/apps/netfp_server/ netfp_server_<device>.out
```

NOTE: Please ensure that the resource manager server is running before launching the NETFP server.

The NETFP server configuration can be specified using a configuration file. Sample configuration files are located in the `ti/apps/netfp_server/netfp_LTE9A.conf`.

The following is an example to start the execution of the NETFP server. Please refer to the NETFP Server documentation for the command line options.

```
./netfp_server_<device>.out -n NetfpServer_LTE9A -i 1 -r Rm_LTE9A -c
netfp_LTE9A.conf &
```

2.2.6 NETFP Proxy

NETFP Proxy is required to configure the NetFP server and enable the application to create 3GPP LTE channels.

1. **Download the NETFP Proxy**
   Download the NETFP Proxy located in the following directory to the EVM file system

```
SYSLIB/ti/apps/netfp_proxy/ netfpproxy_<device>.out
```

NOTE: Please ensure that the NETFP server is running before launching the NETFP proxy.

2. **Download the NETFP Proxy Plugin**

   The OAM or Management application can communicate with NETFP Proxy via Proxy's Plugin interface to issue policy offload commands, cascading commands etc. A sample Plugin can be found in the following location:

```
ti/apps/netfp_proxy/test/netfp_proxy_plugin/netfpproxy_plugin_k2h.so
```

3. **Start the NETFP Proxy**

   The following is an example to start the execution of the NETFP Proxy. Please refer to the NETFP Proxy documentation for the command line options.

```
./netfpproxy_<device>.out –p /home/root/netfpproxy_plugin_k2h.so –d 1 -s
NetfpServer_LTE9A –c Netfp_Client_LTE9A_NET_PROXY -r Rm_LTE9A -i 1 &
```

3 NETFP Master

The NETFP Master Unit tests the NETFP Master messaging infrastructure:-

Realm Test Applicability

DSP	☐
ARM	☒
DSP-ARM Coexistence	☐

The NETFP Master Unit tests for ARM are built and are located in the following directory:-

```
SYSLIB/ti/apps/netfp_master/test/test_netfp_master_<device>.out
```

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master

5. Execute the ARM unit test executable

Test Summary:

The unit test provides a simple menu based command which can be used to send and receive messages to the NETFP Master application. The following features are tested:

- Get Interface Map
- Set Interface Map
- Get Reassembly statistics
- Get Interface List
- Port Mirror
- Port Capture

NOTE: Register/Deregister Notifications and Wait for updates are used internally by development. These have been exposed as a part of the test application but their behavior is for internal SYSLIB development. These messages are not exposed to the end customers and should not be used by them.

4 PKTLIB

The PKTLIB Unit tests the PKTLIB API to ensure API sanity and benchmarking results.

Realm Test Applicability

DSP	☒
ARM	☒
DSP-ARM Coexistence	☐

4.1 DSP

Execution Instructions:

The PKTLIB Unit tests on the DSP are for 2 cores.

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Load the DSP PKTLIB Core0 Executable
6. Load the DSP PKTLIB Core1 Executable
7. Execute the DSP cores.

Test Summary:

The unit tests will test the following features:

- Heap creation/deletion
- Shared/Local Heaps
- Super Heaps
- Packet allocation/cleanup
- Merge
- Cloning
- Split & Split2 functionality

4.2 ARM

The PKTLIB Unit tests for ARM are built and are located in the following directory:-

```
SYSLIB/ti/runtime/pktlib/test/test_pktlib_<device>.out
```

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Execute the ARM unit test executable

Test Summary:

The unit tests will test the following features:

- Heap creation/deletion
- Shared/Local Heaps
- Super Heaps
- Packet allocation/cleanup
- Merge
- Cloning
- Split & Split2 functionality

5 MSGCOM

The MSGCOM Unit tests the MSGCOM API to ensure API sanity. MSGCOM Virtual channel tests record benchmarking information

Realm Test Applicability	
DSP	☒
ARM	☒
DSP-ARM Coexistence	☒

5.1 DSP

5.1.1 Core-Core Test

Execution Instructions:

The MSGCOM Unit tests are executed on 2 cores.

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Load the DSP MSGCOM Core0 Executable
6. Load the DSP MSGCOM Core1 Executable
7. Execute the DSP cores.
8. CLI command menu will be displayed
9. Select option 1 for MSGCOM DSP Core to Core

Test Summary:

The unit tests will test the following features:

- Queue Channels non-blocking mode
    - No interrupt
    - Accumulated interrupt
    - Direct Interrupt
- Queue Channels blocking mode
    - Accumulated interrupt
    - Direct Interrupt
- Queue DMA Channels non-blocking mode
    - No interrupt
    - Accumulated interrupt
    - Direct Interrupt

- Queue DMA Channels blocking mode
    - Accumulated interrupt
    - Direct Interrupt
- Call back function tests
- Virtual channel testing
- Channel Deletion

5.1.2 Virtual Channels

The MSGCOM Unit tests are executed on 2 cores.

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Load the DSP MSGCOM Virtual Core0 Executable
6. Load the DSP MSGCOM Virtual Core1 Executable
7. Execute the DSP cores.
8. CLI command menu will be displayed. Select all the test combinations
9. Benchmark information is reported at the end of the tests

Test Summary:

The unit tests will test the following features:

- Virtual channels are created on a physical channel with different channel configurations. The following channel configurations are tested:-
    - Queue Channel in accumulated interrupt mode
    - Queue Channel in direct interrupt mode
    - Queue DMA Channel in accumulated interrupt mode
    - Queue DMA Channel in direct interrupt mode
- Physical and virtual channel deletion

5.2 ARM

The MSGCOM Unit tests for ARM are built and are located in the following directory:-

```
SYSLIB/ti/runtime/msgcom/arm/test_msgcom_<device>.out
```

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch the MSGCOM unit test

Test Summary:

The unit tests will test the following features:

- Queue Channels non-blocking mode
  - No interrupt
  - Direct Interrupt
- Queue Channels blocking mode
  - Direct Interrupt
- Queue DMA Channels non-blocking mode
  - No interrupt
  - Direct Interrupt
- Queue DMA Channels blocking mode
  - Direct Interrupt
- Channel Deletion

5.3 DSP-ARM coexistence

The MSGCOM Unit tests for DSP-ARM are built and are located in the following directory:-

```
SYSLIB/ti/runtime/msgcom/dsp_arm/test_dsp_arm_msgcom_<device>.out
```

5.3.1 ARM Writer: DSP Reader

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application

4. Launch the NETFP Master
5. Launch the NAME Proxy
6. Load the DSP Core0 Executable
7. Load the DSP Core1 Executable
8. Execute the ARM unit test executable

```
./test_dsp_arm_msgcom_<device>.out
```

9. Execute the DSP executable
10. CLI command menu will be displayed
11. Select option 2 for MSGCOM DSP Core to Core tests

Test Summary:

The unit tests will test the following features:

- Queue DMA Channels non-blocking mode
    - No interrupt
    - ARM is able to send valid messages to the DSP
- Channel Deletion

5.3.2 ARM Reader: DSP Writer

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch the NAME Proxy
6. Load the DSP Core0 Executable
7. Load the DSP Core1 Executable
8. Execute the DSP executable
9. CLI command menu will be displayed
10. Select option 3 for MSGCOM DSP Core to Core tests
11. Execute the ARM unit test executable

```
./test_dsp_arm_msgcom_<device>.out
```

Test Summary:

The unit tests will test the following features:

- Queue DMA Channels non-blocking mode
    - No interrupt
    - DSP is able to send valid messages to the ARM
- Channel Deletion

5.4 Stress

The MSGCOM Stress Unit tests for DSP-ARM are built and are located in the following directory:-

```
SYSLIB/ti/runtime/msgcom/dsp_arm/test_dsp_arm_stress_msgcom_<device>.out
```

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch the NAME Proxy
6. Load the DSP Core0 Executable
7. Load the DSP Core1 Executable
8. Execute the ARM executable
9. Execute the DSP executable
10. CLI command menu will be displayed
11. Select the test to execute

Test Summary:

The unit tests will test the following features:

- Queue DMA Channels in Direct Interrupt Mode with non-blocking
- Messages are sent from DSP to ARM and vice versa and it records the round trip latency
- Ensures that the user space interrupts are not lost and there are no kernel lock ups under heavy stress.

6 Name Proxy

The Name Proxy Unit tests the Name API to ensure API sanity.

Realm Test Applicability

DSP	☐
ARM	☐
DSP-ARM Coexistence	☒

6.1 DSP-ARM coexistence

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch the name proxy
6. Load the Name DSP Core0 Executable
7. Load the Name DSP Core1 Executable
8. Execute the ARM unit test executable

```
./test_name_<device>.out
```

9. Execute the DSP executables

Test Summary:

The unit tests will test the following features:

- Named resource services between the DSP and ARM
  - Create named resources
  - Modify named resources
  - Delete named resources
- Default process

7 NETFP

The NETFP Unit tests the NETFP API to ensure API sanity and also provide benchmarking information

Realm Test Applicability	
DSP	☒
ARM	☒
DSP-ARM Coexistence	☒

7.1 Prerequisite

Ensure that the NETFP configuration file is modified to account for the following:-

- `ENODEB_MAC_ADDRESS` & `ENODEB_IP_ADDRESS` in the file has the IP address & MAC address of the ARM. For IPv6 testing please ensure that the `ENODEB_IP6_ADDRESS0` is also populated. Please ensure that the same IP addresses are configured on the eth0 interface on the EVM.
- `PDN_IP_ADDRESS` & `PDN_MAC_ADDRESS` to be the IP address & MAC Address of the UDP & GTPU Traffic Generator. For IPv6 testing please ensure that the `PDN_IP6_ADDRESS0` is also populated.
- `PDN2_IP_ADDRESS`, `PDN2_MAC_ADDRESS and PDN2_IP6_ADDRESS are needed for testing sockets with wildcarding fast path.`
- Copy the `test_vectors` folders from the NETFP test directory to `C:\test_vectors`.

Sample configuration file is provided in the following directory `ti/runtime/netfp/test` directory. Use this as a template and modify the file as per your test setup.

For DSP applications; ensure that the configuration file is located in the following directory: `"C:\netfp.dat".`

For ARM applications; executing on the EVM; ensure that the configuration file is located in the following directory: `/home/root`

7.2 Traffic Generation

Any traffic generator can be used to send packets to the NETFP unit test. The SYSLIB package provides the source for a sample packet generator utility: `sndPkt.c` (IPv4) and `sndPkt6.c` (IPv6); which can be found in the `netfp/utils` directory

This is designed to send and receive GTPU and UDP packets. By default, this utility can be built on Linux using gcc. To build this utility on Windows, define the flag **_WIN32** and build the utility in Microsoft Visual Studio IDE.

NOTE: Tests which require a traffic generating utility will display this information on the console

7.2.1 Scapy

In order to test certain scenarios (which are **not** easily generated using standard sockets) the open source package `scapy` has been used. The package can be downloaded onto a standard Linux machine.

Sample scripts are available in the `SYSLIB_INSTALL_PATH/ti/runtime/netfp/utils` directory. These scripts would need to be modified as they use hardcoded IP and MAC addresses.

The scripts can be used to test the following features:-

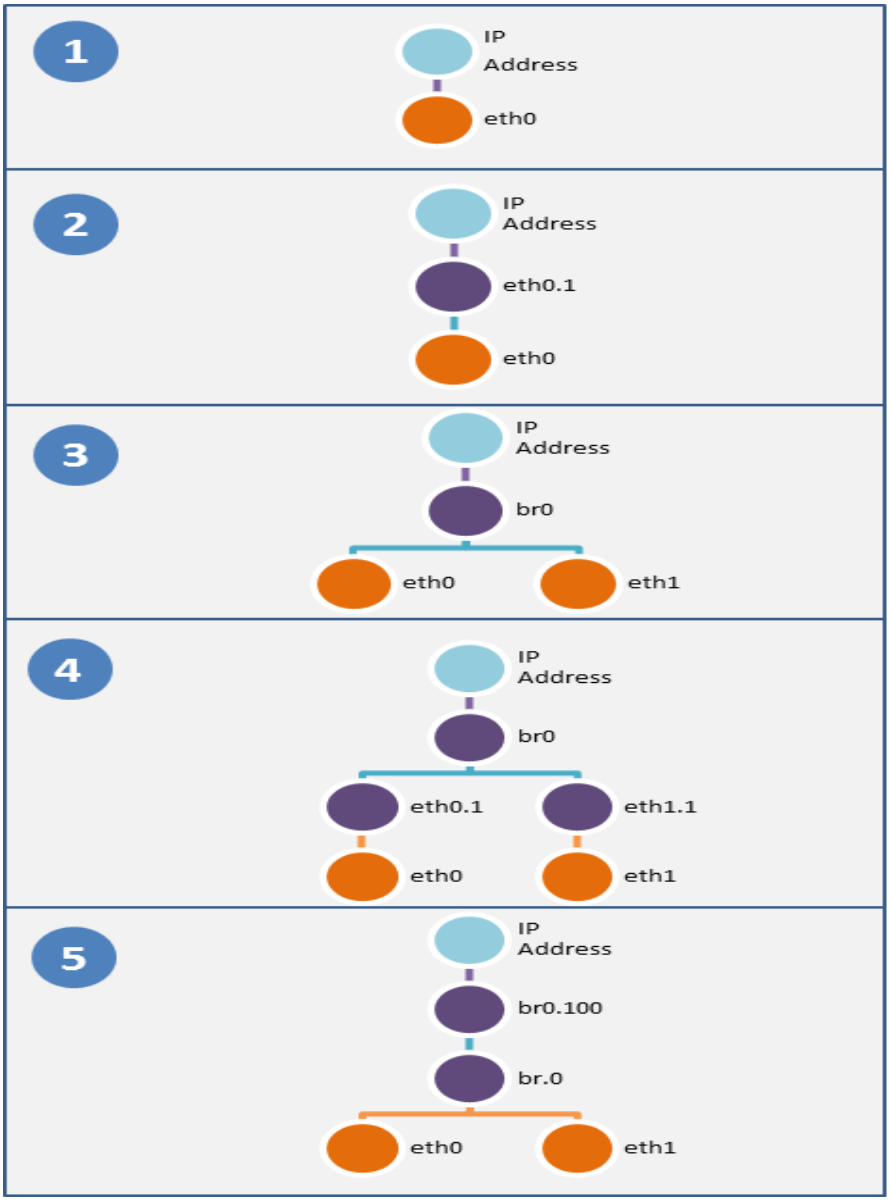
- Overlapping Fragments
- IPv6 Extension Headers
- ICMP Fragmentation needed
- IPv6 Packet Too Big Error

Please refer to the **SCAPY** documentation for more details.

7.3 NETFP supported Interfaces

The following network setups are supported by NETFP.

1. IP address on physical Interface
2. IP address on virtual LAN
3. IP address on a bridge
4. IP address on a bridge with virtual LAN
5. IP address on a virtual bridge



7.4 DSP

These set of tests execute the NETFP server in the context of the DSP Core0. DSP Core1 is the NETFP client.

Execution Instructions:

1. Load & Insert the kernel modules

2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Load the NETFP DSP Core0 Executable
6. Load the NETFP DSP Core1 Executable
7. Execute the DSP cores.
8. CLI command menu is displayed. Select the test option and follow the onscreen instructions.
9. IPSEC Tunnels [IPv6 in IPv6 & IPv4 in IPv4] are created `setkey` utility. The configuration files are located in the `ti/runtime/netfp/utils` folder. Please ensure that the tunnels are created on the Linux machine before executing the traffic generating utility.
10. If test with Wildcarding Fast Path , strongswan.conf need to be updated on EVM. The sample file is available under netfp/util/conf directory.
11. For re-establishment test, Use the sndPkt_burst utility to generate bursty traffic.
    - To initiate fast path re-establishment, set gInitiateFPReestablishment to 1.
    - To initiate fast path re-establishment reject, set gInitiateFPReestablishment to 2.
    - To initiate slow path re-establishment, set gInitiateSPReestablishment to 1.
    - To initiate slow path re-establishment reject, set gInitiateSPReestablishment to 2
12. For HandOver test, Use the sndPkt_burst utility to generate bursty traffic.
    - To initiate source HandOver, set gInitiateSourceHO to 1.
    - To initiate fast path target HandOver, set gInitiateFPTargetHO to 1.
    - To initiate slow path target HandOver, set gInitiateSPTargetHO to 1.

13. For the Basic Handover & Basic Reestablishment Test
    - Use the Send Packet Burst Utility application to generate N number of packets of fixed size 128 bytes. [This is also documented in the Test execution]. For example:

```
./sndpkt_burst.out 192.168.1.2 2152 16 128 0xdead12 255
```

    - Enter the number N when prompted
    - For Reestablishment; the test will also prompt for M packets which need to be handled; the remaining packets will be dropped. Thus M should always be less than N
    - Handover is lossless so no packets should be dropped.
    - The tests will validate the countC and ensure that the statistics are correct and there is no packet ordering.

14. For the Priority Marking

    Use external tool for capturing packets from wire. Wireshark will not show the VLAN tag. Alternatively create a netfp build which disables HW fragmentation.

When HW fragmentation is disabled, the port capture feature can be used to check the outgoing packets.

Unit Test Summary:

The unit tests will test the following features:

- NETFP client and server initialization and startup functionality
- Reassembly
- Send/Receive UDP Data
  - IPv4 non secure mode
  - IPv6 non secure mode
  - IPv4 in IPv4 tunnel [Hardcoded keys]
  - IPv6 in IPv6 [Hardcoded keys]
- 3GPP Services
  - SRB Encode/Decode
  - DRB Encode/Decode.
  - GTPU Control Payload
  - Fast Path Re-establishment
  - Slow Path Re-establishment
  - Source HandOver
  - Fast Path target HandOver
  - Slow Path target HandOver
- Priority Marking

7.5 ARM

These set of tests execute the NETFP server on ARM and the NETFP client is a process on the ARM.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch the NETFP Server
6. If running "Source routing" test, please make sure you have setup the test environment using "`netfp/utils/conf/syslib_duc_lte_untrust_9`" for secure source routing setup or using "`netfp/utils/conf/syslib_duc_lte_trust_10`" for Non-secure source routing setup.

7. If test with Wildcarding Fast Path , strongswan.conf need to be updated on EVM. The sample file is available under `netfp/util/conf` directory.
8. Launch the NETFP Proxy. Offload secure/non-secure policies using NETFP Proxy command shell interface. The test code assumes use plane policies are offloaded with following names:
    a. Ingress user plane : `"Ingress_SPID_IPv4_UP"`
    b. Egress user plane: `"Egress_SPID_IPv4_UP"`.
    c. Wildcarded Ingress user plane: `"IngressWC_SPID_IPv4_UP"`
    d. Engress user plane for wild carding fast path testing: `"EgressWC_SPID_IPv4_UP"`

    Note:
    i. This step is required only if you are planning on running the "Socket", "Wildcarding", "Encode DRB", or "Source routing" tests. The "Basic" and "SRB" tests do not need this step to be run.
    ii. If testing shared SA with Linux, use option "—shared" for command "offload_sp"

9. To test L3 QoS, create an interface via the command shell or by offloading the policies. Configure L3 QoS using command shell. Reconfigure L3 QoS to make sure the changes are propagated. Delete the interface and make sure data is dropped.
10. To test to the VLAN Egress Priority mapping, change the VLAN priority mapping, flush using the command shell. Make sure the changes are propagated.
11. Execute the NETFP Client test application

```
./test_netfp_<device>.out
```

12. CLI command menu is displayed
13. For re-establishment test, Use the sndPkt_burst utility to generate bursty traffic. Re-establishment is initiated automatically for every 10 packets received.
14. For HandOver test, Use the sndPkt_burst utility to generate bursty traffic. HandOver is initiated automatically once 100 packets are received.
15. Netfp Proxy neighbor status change tests are run manually.
    a. Set the neighbor status to FAILED state by deleting the neighbor cache entry. Make sure the changes are propagated to the Server and route recalculation is performed. Data must be dropped until the neighbor entry goes back to REACHABLE or STALE state.

```
ip neighbor del <Next hop IP address> dev <interface name>
```

Pinging the neighbor will cause the entry to go back to REACHABLE or STALE state. Make sure the changes are propagated to the Server and data resumes.

b. Detection of neighbor MAC address change. Use the following to change the neighbor MAC address.

```
ip link set dev eth1 down
ip link set dev eth1 address <new mac address>
ip link set dev eth1 up
ifconfig eth1 10.10.10.1
ifconfig eth1:1 192.168.1.1
arping -A 10.10.10.1 -I eth1 -c 1
```

Make sure the new MAC address is used by all the routes, Outbound Fast paths and Outbound SAs. Data should resume with the new MAC address.

16. For the Basic Handover & Basic Reestablishment Test
    - Use the Send Packet Burst Utility application to generate N number of packets of fixed size 128 bytes. [This is also documented in the Test execution]. For Example:

```
./sndpkt_burst.out 192.168.1.2 2152 16 128 0xdead12 255
```

    - Enter the number N when prompted
    - For Reestablishment; the test will also prompt for M packets which need to be handled; the remaining packets will be dropped. Thus M should always be less than N
    - Handover is lossless so no packets should be dropped.
    - The tests will validate the countC and ensure that the statistics are correct and there is no packet ordering.

17. For the Priority Marking

    Use external tool for capturing packets from wire. Wireshark will not show the VLAN tag. Alternatively create a netfp build which disables HW fragmentation. When HW fragmentation is disabled, the port capture feature can be used to check the outgoing packets.

Unit Test Summary:

The unit tests will test the following features:

- NETFP client and server initialization and startup functionality
- Send/Receive UDP Data
- 3GPP Services

    - SRB Encode/Decode
    - DRB Encode/Decode
    - GTPU Control Payload
    - Slow path and Fast path Re-establishment
    - Route Recalculation for IPv4 and IPv6 secure/non-secure tunnels.
    - Interface creation, deletion and configuring L3 QoS via Netfp proxy
    - Source and slow/fast path Target HandOver
- Path MTU
- Priority Marking

7.6 DSP-ARM

These set of tests execute the NETFP server on ARM and the NETFP client execute on both the DSP and ARM.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch the NETFP Server
6. Launch the NETFP Proxy
7. Load the NETFP DSP-ARM Core0 Executable
8. Load the NETFP DSP-ARM Core1 Executable
9. Execute the DSP cores.
10. Execute the NETFP Client test application

```
./test_dsp_arm_netfp_<device>.out <instNo> <portNum>
```

11. If test with Fast Path Wildcarding, strongswan.conf need to be updated on EVM (copy the file to /etc directory). The sample file is available under netfp/util/conf directory.
12. Execute the IPSEC `do_setup` scripts to setup the security policy. The scripts are provided in the `ti/runtime/netfp/utils/conf` folder. Please ensure that the scripts are executed on the EVM and Linux PC. The folder provides scripts for various test scenarios (Non-Secure mode, secure mode etc.)
13. Offload the required security policy to the NETFP Server using the `cmd_shell_<device>.out`

14. Enter the security policy identifiers on the DSP Core0 prompt also. All the DSP cores and ARM processes will wait till the security policy numbers are entered on the DSP cores also.
15. NAT-T test needs additional Kernel modules. Please refer to EVM script under `ti/runtime/netfp/utils/conf/ syslib_duc_lte_untrust_1_natt` folder for more details. Sample script is also provided to use Ubuntu as a NAT box.

NOTE: Port number 0 is a special case. This is used to kill the "eth0" interface and it in turn implies that all the routing entries and fast path are no longer valid on the DSP or ARM. This is used to test the NETFP event management and to ensure propagation of events from the ARM to DSP.

7.6.1 Socket Statistics

The NETFP module maintains statistics for each socket. The test should also verify and ensure that the socket statistics counters are valid. The socket module is the same between the DSP/ARM and so to verify this please perform the following steps on the CCS:-

- Select DSP Core0 or Core1
- In the CCS Expression Window add the socket Handle and typecast it to `Netfp_Socket`
- Expand the structure and it is possible to see the "stats"
- Send data to the socket using the packet generating utility.
- Capture the data on the Wireshark and ensure that the counters are in Sync

There are 3 types of data packets which should be sent:
1. Packet Size less than Ethernet Minimum size (60 bytes on the wire)
2. Packet between Ethernet Minimum Size and Interface MTU
3. Packet > Interface MTU

Unit Test Summary:

The unit tests will test the following features:

- NETFP Clients and services to coexist between the DSP and ARM realms
- UDP Data can be received and sent by all the NETFP clients
    - Non Secure IPv4
    - Non Secure IPv6
    - IPv6 in IPv4 Tunnels
- NETFP DSP Clients provide reassembly service for all packets.
- Event Management
- Statistics validation and counters
- NETFP Fast Path Wild Carding
- NETFP NAT transversal

7.7 Path MTU

The Path MTU feature along with aging has been tested with the NETFP ARM Unit Test with the Socket Test. The default PMTU ageing timer is set to be 10 minutes.

7.7.1 Non secure Fast Path

Test1: PMTU Message

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies
- Generate SCAPY Packet to set the PMTU to be 1200

Expected Result:

- Fast Path MTU should drop to 1200 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the PMTU Timeout
- Fast Path MTU should go back to 1500
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

7.7.2 Secure Fast Path

Test1: Inner IP PMTU message

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies
- Generate SCAPY Packet to set the Inner IPv4 PMTU 1200

Expected Result:

- Fast Path MTU should drop to 1200 [Verify through the NETFP Server Display]
- PMTU of the SA should be -1 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the PMTU Timeout
- Fast Path MTU should go back to 1500
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

Test2: Outer IP PMTU message

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies
- Generate SCAPY Packet to set the Outer IPv4 PMTU 1200

Expected Result:

- Fast Path MTU should be 1200 [Verify through the NETFP Server Display]
- PMTU of the SA should be 1200 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the PMTU Timeout
- Fast Path MTU should go back to 1500
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

Test3: Inner IP PMTU > Outer IP PMTU (Inner PMTU is sent before Outer PMTU)

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies
- Generate SCAPY Packet to set the *Inner IPv4 PMTU 1300*
- Wait for 1 minute
- Generate SCAPY Packet to set the *Outer IPv4 PMTU 1200*

Expected Result:

- Fast Path PMTU should be 1200 [Verify through the NETFP Server Display]
- PMTU of the SA should be 1200 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the Inner PMTU Timeout
- No socket notification since there was no change to the fast path PMTU
- Wait for the Outer PMTU Timeout
- Fast Path PMTU should go back to 1500
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

Test4: Inner IP PMTU > Outer IP PMTU (Outer PMTU is sent before Inner PMTU)

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies

- Generate SCAPY Packet to set the *Outer IPv4 PMTU 1200*
- Wait for 1 minute
- Generate SCAPY Packet to set the *Inner IPv4 PMTU 1300*

Expected Result:

- Fast Path MTU should be 1200 [Verify through the NETFP Server Display]
- PMTU of the SA should be 1200 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the Outer PMTU Timeout. Since the Inner IP PMTU is never used we don't track it's age
- Fast Path MTU should go back to 1500
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

Test5: Inner IP PMTU < Outer IP PMTU message (Inner PMTU is sent before Outer PMTU)

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies
- Generate SCAPY Packet to set the *Inner IPv4 PMTU 1200*
- Wait for 1 minute
- Generate SCAPY Packet to set the *Outer IPv4 PMTU 1300*

Expected Result:

- Fast Path PMTU should be 1200 [Verify through the NETFP Server Display]
- PMTU of the SA should be 1300 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the Inner PMTU Timeout
- Fast Path PMTU should go to 1300
- Socket Notified with the Path MTU Change reason and the MTU should be 1300
- Wait for the Outer PMTU Timeout
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

Test6: Inner IP PMTU < Outer IP PMTU (Outer PMTU is sent before Inner PMTU)

Configuration:

- Interface MTU is set to 1500
- Execute the ARM Socket Test and Offload the security policies
- Generate SCAPY Packet to set the *Outer IPv4 PMTU 1300*
- Wait for 1 minute
- Generate SCAPY Packet to set the *Inner IPv4 PMTU 1200*

Expected Result:

- Fast Path PMTU should be 1200 [Verify through the NETFP Server Display]
- PMTU of the SA should be 1300 [Verify through the NETFP Server Display]
- Socket Notified with the Path MTU Change reason and the MTU should be 1200
- Wait for the Outer PMTU Timeout
- No socket notification since there was no change to the Fast Path PMTU
- Wait for the Inner PMTU Timeout
- Socket Notified with the Path MTU Change reason and the MTU should be 1500

8 NETFP Master

The NETFP Master tests are used to test and ensure the configuration of the NETFP Master and to test the messaging interface which is exported by the NETFP Master.

Realm Test Applicability

DSP	☐
ARM	☒
DSP-ARM Coexistence	☐

8.1 Reassembly Enable

The test is used to ensure if the NETFP master configuration of "reassembly" sub-module. If the reassembly is enabled in the NETFP Master then all fragmented packets are reassembled by the master. Applications receive the reassembled packets.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the "`reassembly_handling`" is set to 1 in the NETFP Master configuration file
5. Launch the NETFP Master
6. Execute the NETFP ARM project [Socket]; follow the steps mentioned above
7. Send a packet (Matching the destination port) > MTU and make sure that the application receives the packet and echoes it back
8. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

```

```

9. Select the reassembly statistics and make sure the statistics indicate that the stats are incremented
10. Ensure that the NETFP Master Debug interface is operational

```
kill -10 <netfp_master_pid>
```

8.2 Reassembly Disable

The test is used to ensure if the NETFP master configuration of "reassembly" sub-module. If the reassembly is disabled in the NETFP Master then fragmented packets will not be reassembled and the fast path communication will be broken

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the "`reassembly_handling`" is set to 0 in the NETFP Master configuration file
5. Launch the NETFP Master
6. Execute the NETFP ARM project [Socket]; follow the steps mentioned above
7. Send a packet (Matching the destination port) > MTU and make sure that the application does not receive the packet.
8. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

9. Select the reassembly statistics; this will return an error since the reassembly submodule is disabled
10. Ensure that the NETFP Master Debug interface is operational

```
kill -10 <netfp_master_pid>
```

8.3 L2 QoS Enable

The test is used to ensure if the NETFP master configuration of "EQOS" sub-module. If the QOS is enabled then all the packets sent by the NETFP applications will pass through the L2 shaper.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the "`enable_eQOS`" is set to 1 in the NETFP Master configuration file
5. Launch the NETFP Master
6. Execute the NETFP ARM project [Socket]; follow the steps mentioned above. Use the IPSEC untrust_1 script for the test; since we want to ensure that the packet is marked correctly.
7. Send a packet and make sure that the application receives the packet and echoes it back
8. Capture the packets using Wireshark:
    a. By default the socket priority is set to 2; the Inner DSCP is set to be the same as the socket priority so it should be set to 2

    b. The `innerToOuterDSCP_map` will map the inner DSCP to Outer DSCP; refer to the NETFP Master's configuration file for the Outer DSCP value.

    c. The `dscp_map` configuration will map the outer DSCP to the QOS flow/Queue to be used. NOTE: The `dscp_map` section specifies the queue offset relative to the `base_queue`.

    d. The DTS File:-

```
SYSLIB_INSTALL_PATH/runtime/resmgr/dts/k2h/keystone-qostree.dtsi
```

The file above has information about the queue which is to be used. After adding the `queue_offset` to the `base_queue`; lookup the queue number in this file.

e. Use the `SYSLIB_INSTALL_PATH/runtime/netfp/utils/qos.sh` to determine if the packet has been pushed to the correct queue. NOTE: Since the L2 QOS is only enabled right now. Only those statistics are incremented

```
source ./qos.sh
```

9. Ensure that the NETFP Master Debug interface is operational

```
kill -10 <netfp_master_pid>
```

8.4 L2 QoS Disable

The test is used to ensure if the NETFP master configuration of "EQOS" sub-module. If the QOS is disabled then all the packets sent by the NETFP applications will bypass the L2 shaper.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the "`enable_eQOS`" is set to 0 in the NETFP Master configuration file
5. Launch the NETFP Master
6. Execute the NETFP ARM project [Socket]; follow the steps mentioned above. Use the IPSEC untrust_1 script for the test; since we want to ensure that the packet is marked correctly.
7. Send a packet and make sure that the application receives the packet and echoes it back
8. Capture the packets using Wireshark:
    a. By default the socket priority is set to 2; the Inner DSCP is set to be the same as the socket priority so it should be set to 2

    b. The `innerToOuterDSCP_map` will map the inner DSCP to Outer DSCP; refer to the NETFP Master's configuration file for the Outer DSCP value.

    a. Use the `SYSLIB_INSTALL_PATH/runtime/netfp/utils/qos.sh` to ensure that the packet has **NOT** been pushed to the L2 queue.

```
source ./qos.sh
```

9. Ensure that the NETFP Master Debug interface is operational

```
kill -10 <netfp_master_pid>
```

8.5 Setting configuration

The test is used to ensure that it possible to configure using the messaging interface the various QOS parameters. The NETFP Master allows the runtime configuration of the following parameters:

- **Routing Mode:** This can be changed between DSCP and DP-BIT. This is equivalent to changing the `<routing_mode>` entry in the configuration file.
- **DSCP Mapping:** This allows changing the DSCP to L2 QoS Queue. This is equivalent to changing the `<dscp_map>` entry in the configuration file.
- **VLAN Mapping:** This allows changing the VLAN Priority bits to L2 QoS Queue. This is equivalent to changing the `<vlan_map>` entry in the configuration file.
- **Default Host Priority:** This allows changing the default host priority which is used to determine the priority of all non-IP packets generated by the Host. This is equivalent to changing the `<default_host_priority>` entry in the configuration file.
- **Default Forwarding Priority:** This allows changing the priority of non-IP packets when they are forwarded to a specific physical interface by the NETCP. This is equivalent to changing the `<default_forwarding_priority>` entry in the configuration file.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

6. Select the test options matching the use cases described above.

7. Ensure that the configuration succeeds and there are no errors reported by the test application.
8. Verify if the configuration succeeds on the NETFP master by dumping the configuration using the following:-

```
kill -10 <netfp_master_pid>
```

8.6 L3 QoS

The test is used to ensure if the NETFP master configuration of L3 QOS.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the "`enable_eQOS`" is set to 1 in the NETFP Master configuration file
5. Launch the NETFP Master
6. Execute the NETFP ARM project [**Socket L3 Shaper**]; follow the steps mentioned above. Use the IPSEC untrust_1 script for the test; since we want to ensure that the packet is marked correctly.
7. Send a packet and make sure that the application receives the packet and echoes it back
8. Capture the packets using Wireshark:
    a. By default the socket priority is set to 2; the Inner DSCP is set to be the same as the socket priority so it should be set to 2.

    b. By default the test pushes the packets to `Queue 8079` which as per the kernel DTS file is the `hp-cos7`.

    c. The `innerToOuterDSCP_map` will map the inner DSCP to Outer DSCP; refer to the NETFP Master's configuration file for the Outer DSCP value.

    d. The `dscp_map` configuration will map the outer DSCP to the QOS flow/Queue to be used. NOTE: The `dscp_map` section specifies the queue offset relative to the `base_queue`.

    e. The DTS File:-

```

```

```
SYSLIB_INSTALL_PATH/runtime/resmgr/dts/k2h/keystone-qostree.dtsi
```

The file above has information about the queue which is to be used. After adding the `queue_offset` to the `base_queue`; lookup the queue number in this file.

f. Use the `SYSLIB_INSTALL_PATH/runtime/netfp/utils/qos.sh` to determine if the packet has been pushed to the correct queue. NOTE: Since the test is executed with **L2 and L3 QOS both the statistics should be incremented**.

```
source ./qos.sh
```

9. Ensure that the NETFP Master Debug interface is operational

```
kill -10 <netfp_master_pid>
```

8.7 Port mirroring

The test is used to ensure if the NETFP master configuration of port mirroring is operational. Port mirroring allows the NETCP to send inbound/outbound packets to another switch port for debugging

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the interface eth1 is up and running
5. Connect a PC to "`eth1`" and execute Wireshark
6. Launch the NETFP Master
7. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

8. Select the port mirroring option
9. Follow the menu options and select the destination port to be eth1. The port number can be read from the NETFP master configuration file.
10. Now send data to the EVM using ping. Ensure that the Wireshark capture on the PC is able to capture the data
11. The test can be run for Ingress/Egress directions. Validate and ensure that the mirroring can be enabled/disabled

8.8 Port capturing

The test is used to ensure if the NETFP master configuration of port capturing is operational. Port capturing allows the NETCP to send inbound/outbound packets to a queue using a specific flow for debugging

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Ensure that the interface eth1 is up and running
5. Connect a PC to "`eth1`" and execute Wireshark
6. Launch the NETFP Master
7. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

8. Select the port capturing option
9. Follow the menu options and select a destination queue.
10. Now send data to the EVM using ping.
11. There is a test CCS Project utility called "`queueDump`" located in the `SYSLIB_INSTALL_PATH/ti/runtime/netfp/utils` folder which allows the contents of the queue to be dumped. This can be used to validate and ensure that the packets have been captured correctly.
12. The test can be run for Ingress/Egress directions. Validate and ensure that the mirroring can be enabled/disabled

8.9 Ethernet rule management

The test is used to test adding/deleting Ethernet rules through NETFP master. Ethernet rule is used to program PA LUT1-0 to enable features such as cascading.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Setup two interfaces in a bridged mode using following commands as an example:

```
ifconfig eth0 0.0.0.0 up
ifconfig eth1 0.0.0.0 up
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
ifconfig br0 x.x.x.x up
```

5. Ensure that the interface eth0 , eth1 and br0 are up and running
6. Connect one PC1 to "eth0" and one PC2 to "eth1". Both PCs are configured in the same subnet as EVM br0.
7. Launch the NETFP Master with updated netfp.conf
8. Adding rules in netfp_ethrule.dat file as needed according to the above setup (add rules needed, since all the rules in the file will be processed and used to program LUT 1-0)
9. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

10. Select the add Ethernet rule option
11. Sending traffic between PC1 and PC2 with bridge forwarding disabled

```
ebtables -P FORWARD DROP
```

12. Select the del Ethernet rule option to delete all the rules added.

8.10 Ethernet rule management with ARM socket Testing

The test is used to test coexist of Ethernet rules added through NETFP master and Fast Path/socket created through NETFP clients. With Ethernet rules (should not conflict with Fast Path) , traffic for fast path should go through as usual.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch name Proxy

5. Setup two interfaces in a bridged mode using following commands as an example:

```
ifconfig eth0 0.0.0.0 up
ifconfig eth1 0.0.0.0 up
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
ifconfig br0 x.x.x.x up
```

6. Ensure that the interface eth0 , eth1 and br0 are up and running
7. Connect one PC1 to “eth0” and one PC2 to “eth1”. Both PCs are configured in the same subnet as EVM br0.
8. Setup IPsec (example in bridge mode is provided for syslib_duc_lte_trust_5 : `netfp\utils\conf\syslib_duc_lte_trust_5\evm\do_setup_bridge`) to setup secure policy
9. Launch the NETFP Master
10. Launch the NETFP Server and NETFP proxy
11. Adding rules in `netfp_ethrule.dat` file as needed according to the above setup (all the rules in the file will be processed and used to program LUT 1-0)
12. Offload the security policies, example is as follows:

```
offload_sp --sp_id 89 --sp_name Egress_SPID_IPv4_UP
offload_sp --sp_id 96 --sp_name Ingress_SPID_IPv4_UP
```

13. Execute the test NETFP Master application:

```
./test_netfp_master_<device>.out
```

14. Select the add Ethernet rule option to add desired Ethernet Rules.
15. Sending traffic between PC1 and PC2
16. Open telnet window to launch the NETFP ARM test

```
./test_netfp_<device>.out
```

17. Select “socket” test
18. Using packet generator to send traffic to socket create by the test

9 Debug & Trace (DAT)

The DAT Unit tests the DAT APIs to ensure API sanity and also provide benchmarking information

Realm Test Applicability

DSP	☒
ARM	☒
DSP-ARM Coexistence	☒

9.1 Prerequisite

Ensure that the NETFP configuration file is modified to account for the following:-

- `ENODEB_MAC_ADDRESS` & `ENODEB_IP_ADDRESS` in the file has the IP address & MAC address of the ARM. For IPv6 testing please ensure that the `ENODEB_IP6_ADDRESS0` is also populated.
- `RNC_IP_ADDRESS` & `RNC_MAC_ADDRESS` to be the IP address & MAC Address of the UDP & GTPU Traffic Generator. For IPv6 testing please ensure that the `RNC_IP6_ADDRESS0` is also populated.

Sample configuration file is provided in the following directory `ti/runtime/netfp/test` directory. Use this as a template and modify the file as per your test setup.

For DSP applications; ensure that the configuration file is located in the following directory: `"C:\netfp.dat"`.

For ARM applications; executing on the EVM; ensure that the configuration file is located in the following directory: `/home/root`

9.2 DSP test

The test projects execute DAT server and client on DSP core0 and core1.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master

5. Launch Name proxy on ARM
6. Launch NETFP server on ARM
7. Execute the DSP cores.
8. CLI command menu is displayed. Select the test option and follow the onscreen instructions.

Unit Test Summary:

The unit tests will test the following features:

- DAT client and server initialization and startup functionality
- DAT producer and consumer creation
- DAT producer with/without debug streaming

9.3 DSP-ARM producer-consumer test

The test projects execute the DAT server on ARM and the DAT client execute on both the DSP and ARM. It has several sub-test that covers different feature for DAT.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch Name proxy on ARM
6. Launch NETFP server on ARM
7. Launch DAT server on ARM
8. Execute the DSP projects from arm_dsp_consumer.
9. CLI command menu is displayed on CCS console. Select the test option and follow the onscreen instructions.
10. Launch ARM test program.

```
./test_dat_<device>.out
```

11. Select from the tests(1-4) from the following tests:

```
1. Verbosity test
2. ARM Producer test
3. Consumer test(UDP port 10000)
4. PM consumer
5. Memory logging test for DSP producers (no consumer)
6. Memory logging for ARM producer (no consumer)
```

Unit Test Summary:

DAT Clients and services to coexist between the DSP and ARM realms

The unit tests will test the following features:

1. Verbosity Test
    a. Test Verbosity creation and modification at runtime at different level(class, common component, individual component)
    b. It also has a CLI to change verbosity level. Follow the onscreen instruction for verbosity test.
    c. Note: “Mask” is always in hex format.

2. ARM Producer Test
    a. Test producer creation and deletion on ARM
    b. Test producer-consumer communication
    c. The DSP debug stream logs are shipped on UDP port 51235. Logs can be observed/captured from Wireshark. Use System Analyzer to analyze the logs.
    d. Cuia.rta.xml and cuia.uia.xml are provide under dat/test/metadataFiles/cuia directory. A sample UIA configfile – armOnly.usmxml file is also provided. Please refer to “cUIA_Linux_Example_Program_User_Guide.html” under CUIA package on How to setup this file.

3. Producer-consumer Test
    a. Test producer (UIA and General Purpose) and consumer creation on DSP and ARM.
    b. UIA log Sync events generation
    c. Test producer-consumer communication
    d. The logs from consumers are shipped on UDP port 10000. Use System Analyzer to analyze the logs.
    e. Dynamically created loggers use log instance id 32-39. Please copy the content of dynamicLogger.rta.xml to the generated rta.xml from core0/core1 Projects (core0_pe66.rta.xml, core1_pe66.rta.xml) and increase the “loggers-length” by 8. Refer to the following for an example

```
<loggers-length>9</loggers-length>
```

4. PM consumer Test
    a. Test PM consumer with 1 consumer buffer
    b. Buffers are consumed every 1 second and counters from the PM buffer are printed on Linux console. The counter should be stabilized around 800 after 3 buffers.

9.4 DSP-ARM memory logging Test

The test projects execute the DAT server on ARM and the DAT client execute on both the DSP and ARM. It has 2 tests that covers DAT memory logging feature.

Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Launch Name proxy on ARM
6. Launch NETFP server on ARM
7. Launch DAT server on ARM
8. Execute the DSP projects from arm_dsp_memLogging.
9. CLI command menu is displayed on CCS console. Select the test option and follow the onscreen instructions.
10. Launch ARM test program.

```
./test_dat_<device>.out
```

11. Select from the tests(5-6) from the following tests:

```
1. Verbosity test
2. ARM Producer test
3. Consumer test(UDP port 10000)
4. PM consumer
5. Memory logging test for DSP producers (no consumer)
6. Memory logging for ARM producer (no consumer)
```

Unit Test Summary:

DAT Clients and services to coexist between the DSP and ARM realms

1. Memory logging for DSP(Test case 5)/ARM(Test case 6) producers
    a. Test Memory logging for producers created on DSP/ARM
    b. Test Producer controller created on ARM
    c. Test Producer controller APIs to get producer information, start/stop memory logging etc.
    d. Generated logs are saved into log files under /tmp with producer name in the file name. Each producer will generate its own file.
    e. Logs are post-processed by using perl script under dat\utils\memLoggingScript\memLog.pl
    A batch file (memLog.bat) is also available to read list of logs and generate one bin file.
    f. Analyzed the bin file using System Analyzer. UIA config file creation should be the same as logs shipped through debug streaming.

9.5 Wireshark Dissector for UIA

The UIA Wireshark Dissector is available under:-

```
SYSLIB_INSTALL_PATH\ti\runtime\dat\utils\UIAdissector
```

This can be used to analyze the UIA stream through Wireshark.

10 LTE Release 9 Demo

The LTE Demo tests the Domain and Root functionality along with the other SYSLIB services to ensure proper API behavior

Realm Test Applicability

DSP	☒
ARM	☒
DSP-ARM Coexistence	☒

The tests ensure that the RAT master on the ARM is able to communicate with the DSP cores and the domain API are able to instantiate the RAT master provisioned services. The demo also allows the shutdown of a cell. The demo also illustrates the coexistence of 2 RAT cells (A and B).

10.1 Prerequisite

1. Please ensure that the NETFP & DAT Server Rel9A and Rel9B configuration files are available on the target for the demo to work. These files are located in the `ti/apps/netfp_server` and `ti/apps/dat_server` directory.
2. Due to a bug in MCSDK S/W, the current release version of the demo supports only one instance of NETFP Proxy to be run. Hence, to demonstrate the coexistence of 2 RAT cells, the demo code by default has been compiled to NOT use NETFP Proxy. NETFP Proxy can be however used with one RAT cell at a time. To try this option out, please recompile the L2 and master code of Rel 9 demo with `USE_NETFP_PROXY` defined.

10.2 Build Instructions:

Please ensure that the build environment is correctly configured. Please ensure that the SYSLIB_DEVICE is configured for the correct device.

NOTE: The demo is supported only for K2H

To build the Demo Master Code; please use the following command:-

```
make demoMaster
```

In order to build the rel9 demo; which includes all the 8 DSP Cores; please use the following command:

```
make rel9
```

10.3 Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Setup all the Strongswan IPSec configuration files required to create secure /non-secure policies on the EVM and the Security Gateway (SecGw).
6. Load the Core0-Core7 Rel9 demo applications (for K2H) and execute the DSP cores
7. Execute the RAT master application

```
./demo_master_<device>.out
```

8. CLI command menu is displayed
9. Select the Release 9 menu selection and follow the CLI to start/stop RAT cells.
10. Using the CLI command menu offload the secure/non-secure User plane (UP) policies to NETFP.

    NOTE: To figure out the policy Ids to use for offloads, please use '`ip -s xfrm policy`' command and find the ingress (dir = in) and egress (dir=out) UP policies to offload and locate their '`index`' field.

11. The demo expects the following naming convention to be followed during policy offloads:- When offloading egress UP policy, specify the policy name to be '`Egress_SPID_IPv4_UP`'. When offloading ingress UP policy, specify the policy name to be '`Ingress_SPID_IPv4_UP`'. The demo by default uses the same policies for FAPI logging as well.

12. Once done offloading the Ingress/Egress UP policies, choose '3' option from CLI command menu to exit the offload menu and return to main menu.

NOTE:

a. Steps 7-10 above are required ONLY IF the demo code has been compiled with `USE_NETFP_PROXY` defined.

b. The demo code by default uses the following IP addresses for User plane/FAPI Fast path creations.

```
eNodeB User plane IP address :- 192.168.1.2

PDN Gw IP address: 192.168.1.1

FAPI Logger IP address:  192.168.1.1
```

If using a different set of IP addresses, please make sure you modify the following parameters in the demo code located at `ti/demo/lte/rel9/l2/l2_dp.c` and `ti/demo/lte/rel9/l2/fapi.c`.

Modify `ti/demo/lte/rel9/l2/l2_dp.c` for:

```
/* eNodeB IPv4 Address. Used for GTPU Fast path creation */
uint8_t eNodeBIPAddress[4] = {192, 168, 1, 2};

/* eNodeB MAC Address. Used only when NetFP Proxy is not used */
uint8_t eNodeBMACAddress[6] = {0x08, 0x00, 0x28, 0x32, 0x99, 0xa3};

/* PDN Gw IPv4 Address. Used for GTPU Fast path creation */
uint8_t pdnGwIPAddress[4] = {192, 168, 1, 1};

/* PDN Gw MAC Address. Used only when NetFP Proxy is not used */
uint8_t pdnGwMACAddress[6] = {0x08, 0x00, 0x27, 0xb2, 0x0c, 0x65};
```

Modify `ti/demo/lte/rel9/l2/fapi.c` for:

```
/* eNodeB IPv4 Address. Used for FAPI Fast path creation */
uint8_t eNodeBIPAddress[4] = {192, 168, 1, 2};

/* Logger IPv4 Address. Used for FAPI Fast path creation */
uint8_t loggerIPAddress[4] = {192, 168, 1, 1};

/* Logger MAC Address. Used only when NetFP Proxy is not used */
uint8_t loggerMACAddress[6] = {0x08, 0x00, 0x27, 0xb2, 0x0c, 0x65};
```

Also please make sure to recompile the L1, L2 demo code using CCS and download the updated .out files to all the DSP cores.

10.4 Test Summary:

The unit tests will test the following features:

- Root functionality: Root master & slave (on DSP only) operations with the exchange of information between the entities
- Domain initialization & cleanup usage
- RAT cells with exclusive resource usage. Each RAT cell has access to its own set of independent resources which are not shared and which can be allocated & cleaned up without affecting the other RAT cell.
- FAPI Tracing Library which sends out the FAPI messages on the Ethernet and can be captured. **NOTE:** Please modify the IP address & MAC address in the `l2_dp.c` and `fapi.c` file as per your test setup.
- DAT Tracing support which streams out the DAT UIA messages on the Ethernet and can be captured.
- LTE User plane channel created with GTP-U Id: 0xdead12 and 0xdead16 for LTE cells 9A and 9B respectively . Using a packet generator one, can send/receive GTP-U packets to the above mentioned GTP-U Id.

11 LTE Release 10 Deployment1 Demo

The tests ensure that the RAT master on the ARM is able to communicate with the DSP cores and the L2 application on ARM.

11.1 Prerequisite

1. Please ensure that the NETFP & DAT Server Rel10 Deployment1 configuration files are available on the target for the demo to work. Sample files are

`ti/apps/netfp_server/netfp_LTE10.conf` and `ti/apps/dat_server/dat_LTE10.conf`

2. Ensure that the IPSEC configuration files are setup and the tunnels (Secure or non-secure) are setup before the demo application is started. The test will prompt for the policy identifiers to be offloaded.

   Use the following command to get the policy identifier to be offloaded once the IPSEC has been started on the EVM:-

```
ip –s xfrm policy
```

11.2 Build Instructions:

Please ensure that the build environment is correctly configured. Please ensure that the SYSLIB_DEVICE is configured for the correct device.

NOTE: The demo is supported for K2H, K2L

To build the Demo Master Code; please use the following command:-

```
make demoMaster
```

In order to build the rel10 Deployment1 demo; which includes 4 DSP Cores & the L2 application executing on ARM; please use the following command:

```
make rel10D1
```

11.3 Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server
3. Launch the SOC Init application
4. Launch the NETFP Master
5. Using the CLI command menu offload the secure/non-secure User plane (UP) policies to NETFP. Refer to the test prerequisite on how to get the policy identifier.

6. Load the Core0-Core3 Rel10 demo applications (for K2H) and execute the DSP cores
7. Execute the Rel10 L2 demo application

```
./demo_rel10d1_l2_<device>.out
```

8. Execute the RAT master application

```
./demo_master_<device>.out
```

9. CLI command menu is displayed
10. Select the release 10 deployment1 menu selection and follow the CLI to start/stop RAT cells.
11. The demo expects the following naming convention to be followed during policy offloads:-

    - Egress UP policy, Policy name is `'Egress_SPID_IPv4_UP'`.
    - Ingress UP policy, Policy name is `'Ingress_SPID_IPv4_UP'`

    In the demo; the FAPI Tracing uses the same policy identifiers

12. Once done offloading the Ingress/Egress UP policies, choose `'3'` option from CLI command menu to exit the offload menu and return to main menu.

    NOTE: The demo code by default uses the following IP addresses for User plane/FAPI Fast path creations.

```
eNodeB User plane IP address: 192.168.1.2

PDN Gw IP address: 192.168.1.1

FAPI Logger IP address:  192.168.1.1
```

    If using a different set of IP addresses, please make sure you modify the following parameters in the demo code located at `ti/demo/rel10_d1/l2/l2_dp.c` and `ti/demo/rel10_d1/l2/fapi.c`.

    Modify `ti/demo/rel10_d1/l2/l2_dp.c` for:

```
/* eNodeB IPv4 Address. Used for GTPU Fast path creation */
uint8_t eNodeBIPAddress[4] = {192, 168, 1, 2};

/* PDN Gw IPv4 Address. Used for GTPU Fast path creation */
uint8_t pdnGwIPAddress[4] = {192, 168, 1, 1};
```

Modify `ti/demo/rel10_d1/l2/fapi.c` for:

```
/* eNodeB IPv4 Address. Used for FAPI Fast path creation */
uint8_t eNodeBIPAddress[4] = {192, 168, 1, 2};

/* Logger IPv4 Address. Used for FAPI Fast path creation */
uint8_t loggerIPAddress[4] = {192, 168, 1, 1};
```

Also please make sure to recompile the demo code and download the updated demo L2 application to EVM.

11.4 Test Summary:

The unit tests will test the following features:

- Root functionality: Root master & slave (on DSP & ARM) operations with the exchange of information between the entities
- Domain initialization & cleanup usage
- RAT cells with exclusive resource usage. Each RAT cell has access to its own set of independent resources which are not shared and which can be allocated & cleaned up without affecting the other RAT cell.
- FAPI Tracing Library which sends out the FAPI messages on the Ethernet and can be captured.
- DAT Tracing support which streams out the DAT UIA messages on the Ethernet and can be captured.
- LTE User plane channel created with GTP-U Id: 0xdead12. Using a packet generator one, can send/receive GTP-U packets to the above mentioned GTP-U Id.

12 LTE Release 10 Deployment2 Demo

The tests ensure that the RAT master on the ARM is able to communicate with the DSP cores executing the L2 and L1. There is a central RAT database present on ARM which is used by all the DSP cores.

12.1 Prerequisite

1. Please ensure that the NETFP & DAT Server Rel10 Deployment2 configuration files are available on the target for the demo to work. Sample files are `ti/apps/netfp_server/netfp_LTE10_D2.conf` and `ti/apps/dat_server/dat_LTE10_D2.conf`

2. Ensure that the IPSEC configuration files are setup and the tunnels (Secure or non-secure) are setup before the demo application is started. The test will prompt for the policy identifiers to be offloaded.

   Use the following command to get the policy identifier to be offloaded once the IPSEC has been started on the EVM:-

```
ip –s xfrm policy
```

12.2 Build Instructions:

Please ensure that the build environment is correctly configured. Please ensure that the SYSLIB_DEVICE is configured for the correct device.

NOTE: The demo is supported only for K2H

To build the Demo Master Code; please use the following command:-

```
make demoMaster
```

In order to build the rel10 Deployment2 demo; which includes 8 DSP Cores; please use the following command:

```
make rel10D2
```

12.3 Execution Instructions:

1. Load & Insert the kernel modules
2. Launch the resource manager server

3. Launch the SOC Init application
4. Launch the NETFP Master
5. Using the CLI command menu offload the secure/non-secure User plane (UP) policies to NETFP. Refer to the test prerequisite on how to get the policy identifier.
6. Load the Core0-Core7 Rel10 demo applications (for K2H) and execute the DSP cores
7. Execute the RAT master application

```
./demo_master_<device>.out
```

8. CLI command menu is displayed
9. Select the release 10 deployment2 menu selection and follow the CLI to start/stop RAT cells.
10. The demo expects the following naming convention to be followed during policy offloads:-

    - Egress UP policy, Policy name is `'Egress_SPID_IPv4_UP'`.
    - Ingress UP policy, Policy name is `'Ingress_SPID_IPv4_UP'`

    In the demo; the FAPI Tracing uses the same policy identifiers

11. Once done offloading the Ingress/Egress UP policies, choose `'3'` option from CLI command menu to exit the offload menu and return to main menu.

    NOTE: The demo code by default uses the following IP addresses for User plane/FAPI Fast path creations.

```
eNodeB User plane IP address :- 192.168.1.2

PDN Gw IP address :- 192.168.1.1

FAPI Logger IP address :-  192.168.1.1
```

    If using a different set of IP addresses, please make sure you modify the following parameters in the demo code located at `ti/demo/rel10_d2/l2/l2_dp.c`.

    Modify `ti/demo/rel10_d1/l2/l2_dp.c` for:

```
/* eNodeB IPv4 Address. Used for GTPU Fast path creation */
uint8_t eNodeBIPAddress[4] = {192, 168, 1, 2};

/* PDN Gw IPv4 Address. Used for GTPU Fast path creation */
uint8_t pdnGwIPAddress[4] = {192, 168, 1, 1};

/* Logger IPv4 Address. Used for FAPI Fast path creation */
uint8_t loggerIPAddress[4] = {192, 168, 1, 1};
```

Also please make sure to recompile the demo code and download the updated demo L2 application to EVM.

12.4 Test Summary:

The unit tests will test the following features:

- Root functionality: Root master & slave (on DSP & ARM) operations with the exchange of information between the entities
- Domain initialization & cleanup usage
- RAT cells with exclusive resource usage. Each RAT cell has access to its own set of independent resources which are not shared and which can be allocated & cleaned up without affecting the other RAT cell.
- FAPI Tracing Library which sends out the FAPI messages on the Ethernet and can be captured.
- DAT Tracing support which streams out the DAT UIA messages on the Ethernet and can be captured.
- LTE User plane channel created with GTP-U Id: 0xdead12 (Core0) and GTP-U Id: 0xdead16 (Core4). Using a packet generator one, can send/receive GTP-U packets to the above mentioned GTP-U Identifiers.