Implementing M-Plane & M-Plane Passthrough in ORAN Studio Solution with S5040A

Reference Guide

U5040
Open RAN Studio
5G O-RAN Radio Unit Testing
DOWNLOAD YOUR NEXT INSIGHT
KEYSIGHT TECHNOLOGIES
© Keysight Technologies 2020
KEYSIGHT
S5040A
Open RAN Studio Player and Capture Appliance

KEYSIGHT TECHNOLOGIES

Contents

1. Management (M) Plane

KEYSIGHT
TECHNOLOGIES

Section 1.1: Overview

The Management Plane (M-Plane) is the part of a network system for configuration, monitoring, management and distribution of various services to all layers of the network stack and other parts of the system.

NOTE

Standardization with M-Plane

M-plane is the standardization of the configuration of O-RUs, without the need of any obscure and proprietary interfaces.

1.1.1: M-Plane characteristics

The M-Plane is used to:

- Handle operations in networks with multiple O-DUs and O-RUs.
- Define operations and functions to configure parameters required by the CUS-planes.

1.1.2: M-Plane architecture

The M-Plane architectures support two models:

- Hierarchical model

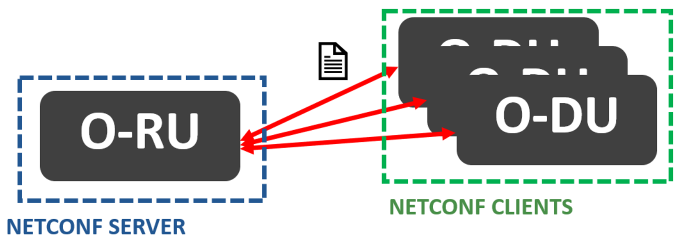


Figure 1 Block diagram for Hierarchical model of M-Plane architecture

- Hybrid model

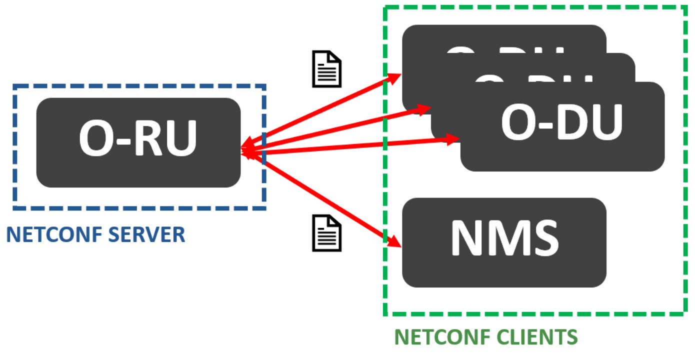


Figure 2 Block diagram for Hybrid model of M-Plane architecture

To know more, refer to the topic Hybrid Functionality on page 113.

For efficient management of network operations, the key features in M-Plane are:

- NETCONF (Network Configuration): Network protocol for management
- YANG (Yet Another Next Generation): Data exchange language in NETCONF

1.1.3: Network Configuration (NETCONF) protocol

NETCONF is a network management protocol developed and standardized by the IETF. It was published in 2006 as RFC 4741 and later revised in 2011 and published as RFC 6241.

NETCONF provides mechanisms to:

- Install configuration of network devices
- Manage configuration of network devices
- Delete configuration of network devices

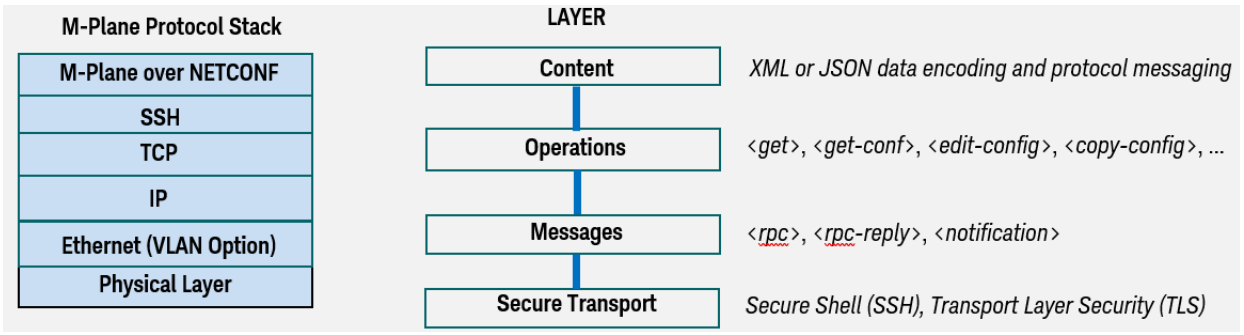

M-Plane Protocol Stack
M-Plane over NETCONF
SSH
TCP
IP
Ethernet (VLAN Option)
Physical Layer
LAYER
Content
Operations
Messages
Secure Transport
XML or JSON data encoding and protocol messaging
<get>, <get-conf>, <edit-config>, <copy-config>, ...
<rpc>, <rpc-reply>, <notification>
Secure Shell (SSH), Transport Layer Security (TLS)

Figure 3 Block diagram for M-Plane protocol stack and NETCONF layer

1.1.4: Yet Another Next Gen (YANG) model

YANG was published as RFC 6020 in October 2010 by Internet Engineering Task Force (IETF).

YANG is a data model language by itself and not an encoding data language such as XML or JSON, although instances of the YANG model can be encoded into these formats for transport over the network. This model is used in networks management protocols, that is, NETCONF, to define the structure of configuration and state data (also, YANG modules) on network devices.

The YANG models are tree-based, such as:

```
module (name.yang)
    container
        group
```

leaf

...

Refer to *Annex D* in the *O-RAN.WG4.MP.0-v0X.00* specification for latest YANG models.

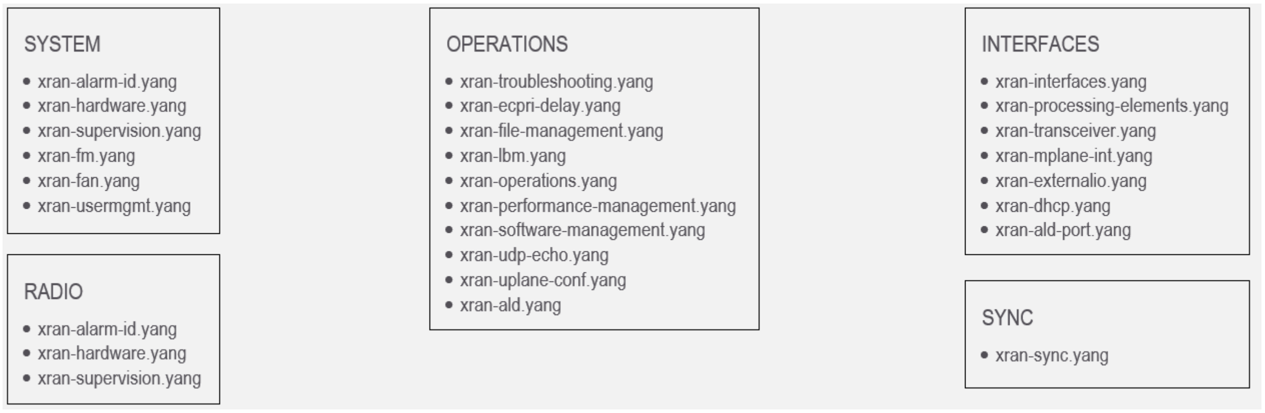


Figure 4 List of xRAN Fronthaul M-Plane YANG modules

1.1.5: M-Plane Functions

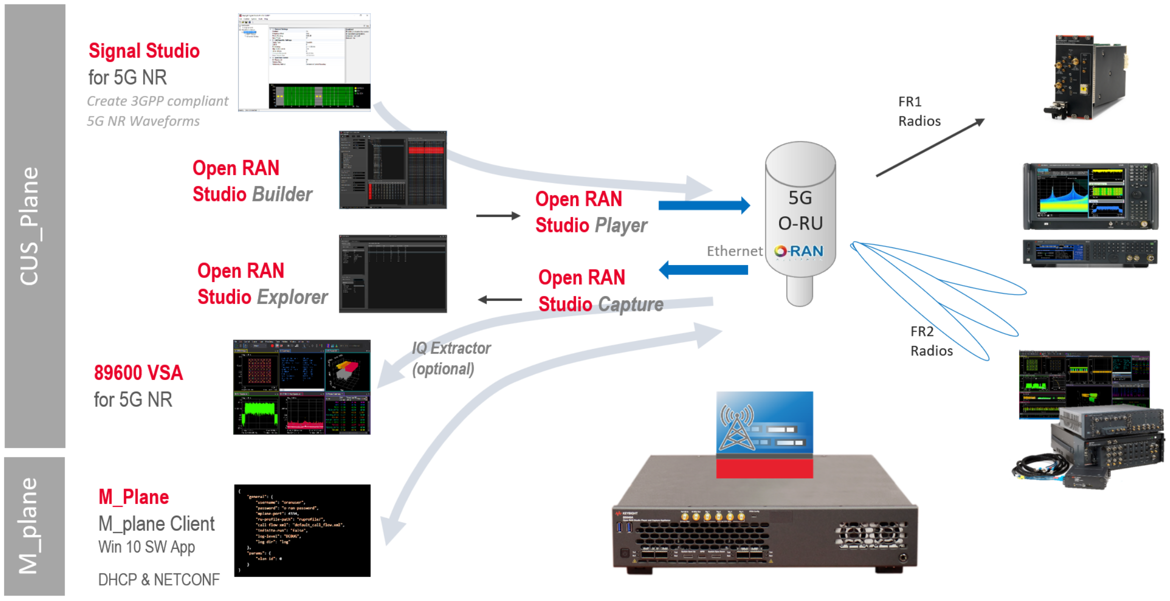
- Start Up installation
  - Establishes the M-plane connections between NETCONF clients and O-RU.
  - Setting of Transport Layer address via either manual, Dynamic Host Configuration Protocol (DHCP) and State-Less Address Auto-Configuration (SLAAC).
- SW Management
  - SW inventory
  - SW download
  - SW installation
  - SW activation
- Configuration Management
  - Sets the parameters required by O-RU for operation
  - Gets equipment status information

- Fault Management
  - Monitors potential errors and faults in the network
  - Contains notifications from O-RU such as ID, location of fault occurrence, locations affected by fault, severity, and so on

2. M-Plane in Keysight Open RAN Studio Solution

Section 2.1: Keysight Open RAN Studio Solution - Overview

Figure 5 shows a comprehensive solution, comprising of the S5040A Open RAN Studio Player and Capture Appliance and the U5040B Open RAN Studio software, for O-RU (Radio Unit) testing.



```
{
    "general": {
        "username": "oranuser",
        "password": "o-ran-password",
        "mplane-port": 4334,
        "ru-profile-path": "ruprofile/",
        "call-flow-xml": "default_call_flow.xml",
        "infinite-run": "false",
        "log-level": "DEBUG",
        "log-dir": "log"
    },
    "params": {
        "vlan-id": 0
    }
}
```

Figure 5 Key elements in the Open RAN Studio solution

2.1.1: Hardware Platform for Open RAN Studio Software

Keysight provides software applications running on a Windows based platform, including an FPGA controlled NIC card in the S5040A Appliance. The FPGA ensures time-deterministic generation and analysis. It also includes clock, trigger and PPS sharing to ensure that the entire system is operating under the same time and frequency reference.

System setup overview and features for the M-Plane client software	
Overview	▪ Host based emulation software ▪ M-Plane on independent Ethernet port ▪ Passthrough capabilities to merge M-Plane with CUS-plane (FPGA module acts as a switch)
Features	▪ DHCP server with DU profile ▪ M-Plane NETCONF client ▪ Configurable Call Flows

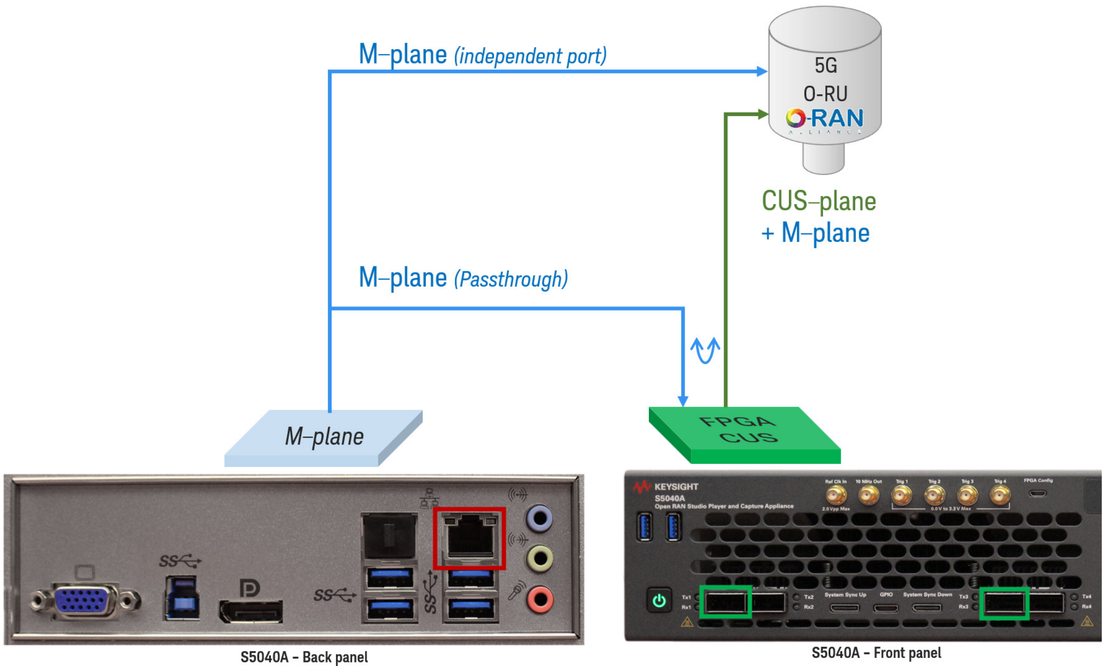


Figure 6 Setup for M-Plane client SW

NOTE

For M-Plane passthrough connections, the QSFP 1 must be connected to the O-RU and QSFP 3 must be connected to the Ethernet port (at the rear panel) of the S5040A Appliance. Figure 10 shows the connection diagram with such setup.

2.1.2: M-Plane Support in Open RAN Studio Solution

Supported M-Plane features	
Supported	▪ Based on O-RAN M-Plane Version 1.0 ▪ Most functions supported for Installation startup ▪ IPv4 and DHCP ▪ Hierarchical and Hybrid Models ▪ IPv6 ▪ VLAN ▪ ALD (partially supported) ▪ Extended Input/Output Ports ▪ LBM ▪ SSH and TLS

M-Plane Startup Sequence Support*

No.	Step	Support	Open RAN Studio components
1	Transport Layer Initialization	Yes	DHCP server
2	O-RU synchronization to primary reference clock	Yes	Open RAN Studio / S5040A Appliance
3	O-RU Calls home to NETCONF clients	Yes	NETCONF client
4	SSH Secure Shell Connection Established	Yes	NETCONF client
5	NETCONF Capability Discovery	Yes	NETCONF client
6	Optional provisioning of new management accounts	Yes	NETCONF client
7	Supervision of NETCONF connection	Yes	NETCONF client
8	Retrieval of O-RU Information	Yes	NETCONF client
9	SW Management	Yes	NETCONF client
10	CU-Plane transport connectivity check (LBM)	Yes	Open RAN Studio / S5040A Appliance

No.	Step	Support	Open RAN Studio components
11	U-Plane configuration	Yes	NETCONF client
12	Recovery of O-RU Delay Profile	Yes	NETCONF client
13	Fault Management	Yes	NETCONF client
14	Performance Management	Yes	NETCONF client
15	Retrieval of O-RU State	Yes	NETCONF client
16	Configuring the O-RU operational parameters	Yes	NETCONF client

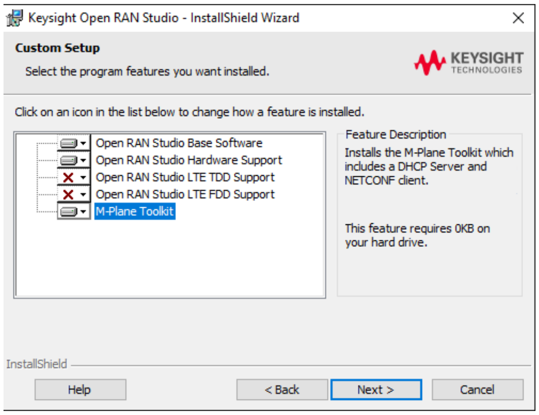
* Refer to Figure 1.4 in the ORAN-WG4-MP-0-V01.00 specification

2.1.3: M-Plane Features Available in ORAN Solution

Release Version	M-Plane features
2.0.10221.0 (U5040B Open RAN Studio software download page)	▪ M-Plane Client Software ▪ M-Plane FPGA Pass-through (on S5040A Appliance)
Future versions	▪ M-Plane conformance testing ▪ M-Plane Client API (to support automated testing and conformance)

2.1.4: Installing M-Plane Client Software

The M-Plane client software is installed, by default, with the U5040B Open RAN Studio software. You can download the software from the U5040B Open RAN Studio software download page. You may install the software either on the S5040A Appliance or on a Windows 10 64-bit PC. While the option to install “M-Plane Toolkit” is selected by default in both cases, the differences in the appearance of the “Custom Setup” window are shown in the images below. On the Windows PC, the M-Plane client is installed with the ‘offline’ mode of the U5040B Open RAN Studio software.


Keysight Open RAN Studio - InstallShield Wizard

Custom Setup

Select the program features you want installed.

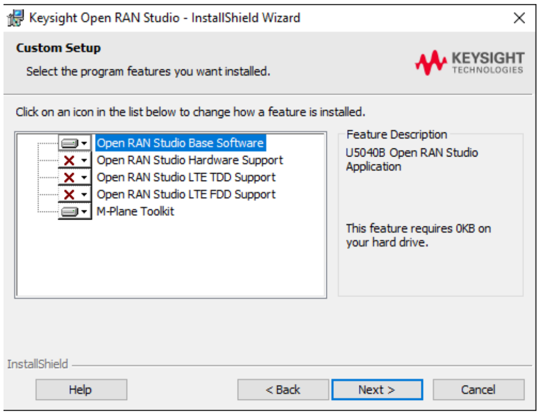
Click on an icon in the list below to change how a feature is installed.

- Open RAN Studio Base Software
- Open RAN Studio Hardware Support
- Open RAN Studio LTE TDD Support
- Open RAN Studio LTE FDD Support
- M-Plane Toolkit

Feature Description

Installs the M-Plane Toolkit which includes a DHCP Server and NETCONF client.

This feature requires 0KB on your hard drive.


Keysight Open RAN Studio - InstallShield Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

- Open RAN Studio Base Software
- Open RAN Studio Hardware Support
- Open RAN Studio LTE TDD Support
- Open RAN Studio LTE FDD Support
- M-Plane Toolkit

Feature Description

U5040B Open RAN Studio Application

This feature requires 0KB on your hard drive.

Figure 7 Custom setup on S5040A HW (left) and on a standard Win 10 PC (right)

2.1.5: Capabilities of the M-Plane client software

M-Plane capabilities	
DHCP server	1 IP address assignment 2 Includes M-Plane specific parameters ▪ Sends Option 43: Vendor Specific Information: encodes IP address of NETCONF controller ▪ Receives / Parses Option 60: Vendor Class Identifier (such as, "oran-ru/<vendor>")
NETCONF	1 Scripted through Call-flow.xml ▪ Transport and Handshake (IPv4) ▪ NETCONF session establishment ▪ NETCONF Subscription to Notifications ▪ M-Plane connection supervision ▪ Retrieve active alarm list ▪ Retrieve O-RU information elements (with and without filter) ▪ Edit / Merge Configuration - Create VLAN - Add Processing Element - Endpoint Carrier Configuration - Endpoint Carrier Link Configuration - Activate Carrier 2 Allows custom sequences, easy to modify

Section 2.2: Using M-Plane Software

The following block diagrams illustrate three scenarios, where M-Plane messages only are transmitted, the M-Plane software, which is installed with U5040B Open RAN Studio software, is used to transmit both M-Plane and CUS-Plane messages separately or M-Plane Passthrough is applied along with either a second NIC on the S5040A Appliance or an external PC.

Setup 1: Transmitting M-Plane messages only using Windows 10 PC

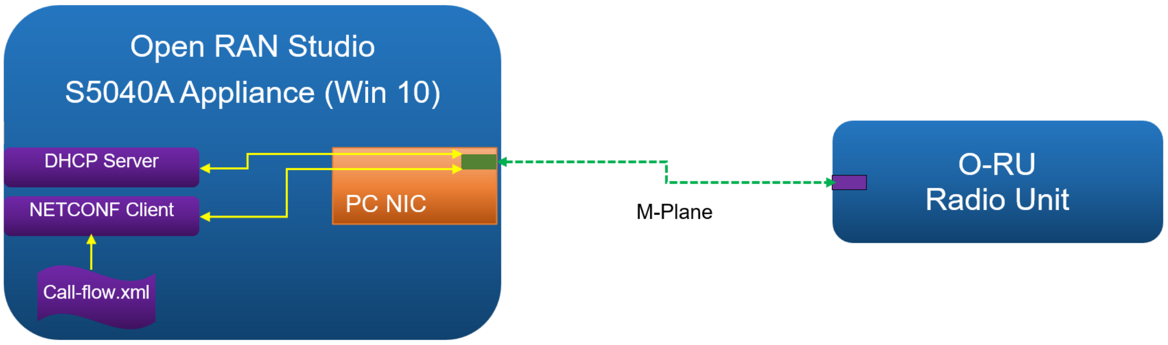


Figure 8 Transmitting M-Plane messages only using Win 10 PC

Setup 2: Transmitting M-Plane & CUS Plane messages using 2nd port on O-RU

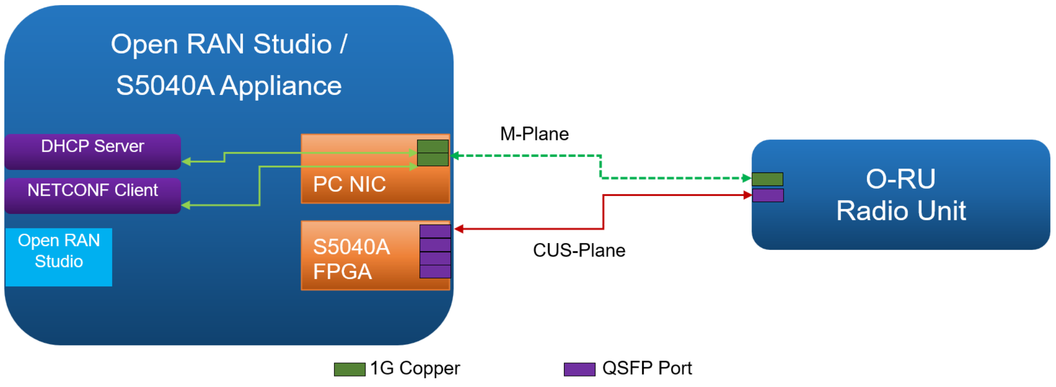


Figure 9 Transmitting M-Plane & CUS-Plane messages using M-Plane software

Setup 3: Transmitting CUS & M-Plane messages using M-Plane with FPGA Passthrough

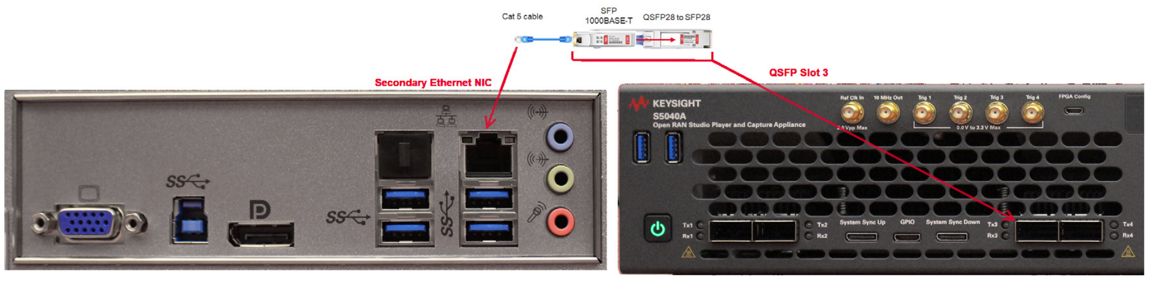


Figure 10 M-Plane Pass-through connection on S5040A Appliance with own NIC

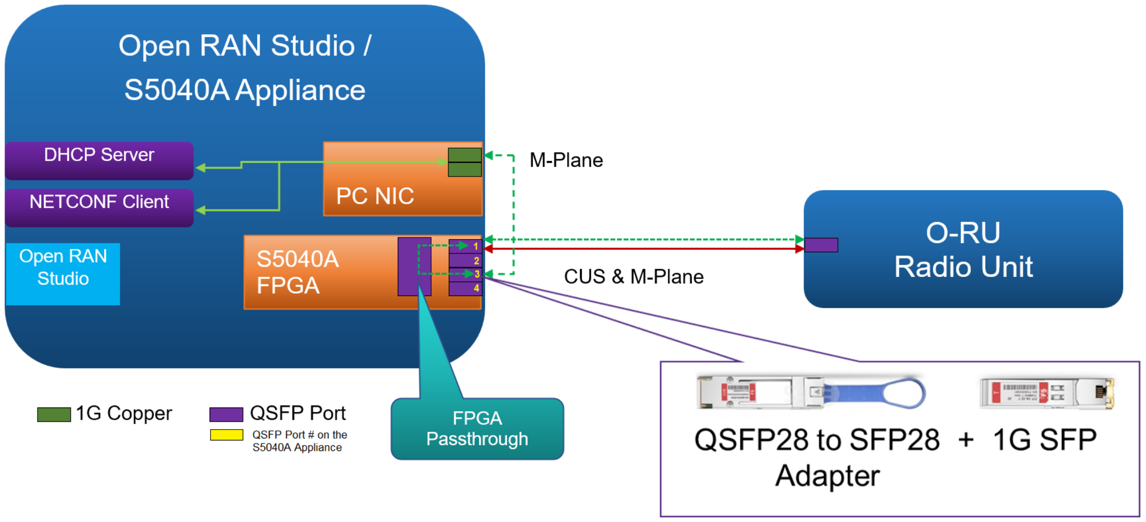


Figure 11 M-Plane Pass-through connection with secondary Ethernet NIC

The following block diagram depicts a system where the M-Plane Pass-through port has been used to establish connection with a remote machine hosting the M-Plane client. In this case, the NIC port of the remote hardware is connected to the S5040A Appliance as shown in the image below. Here, the O-RU is connected to QSFP port 1 of the S5040A Appliance Server. Both the CUS-Plane and M-Plane messages are transmitted over an Ethernet connection at 25G No FEC speed.

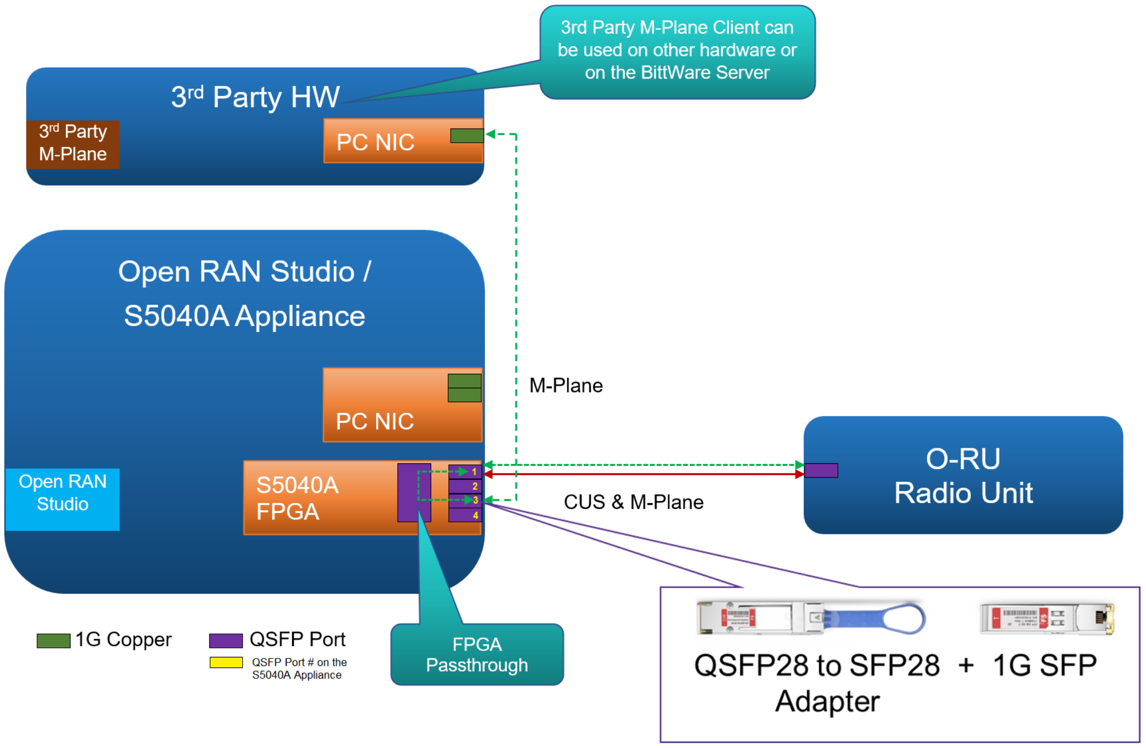


Figure 12 M-Plane Pass-through connection with external M-Plane source

Note that in the setup involving M-Plane with FPGA Passthrough, you must first insert a 1G SFP module into an QSFP28 to SFP28 adapter. Then, connect one end of a Cat-5 / Cat-6 Ethernet cable to the adapter and the other end to the secondary Ethernet NIC port on the S5040A Appliance.

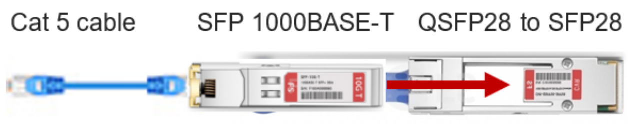
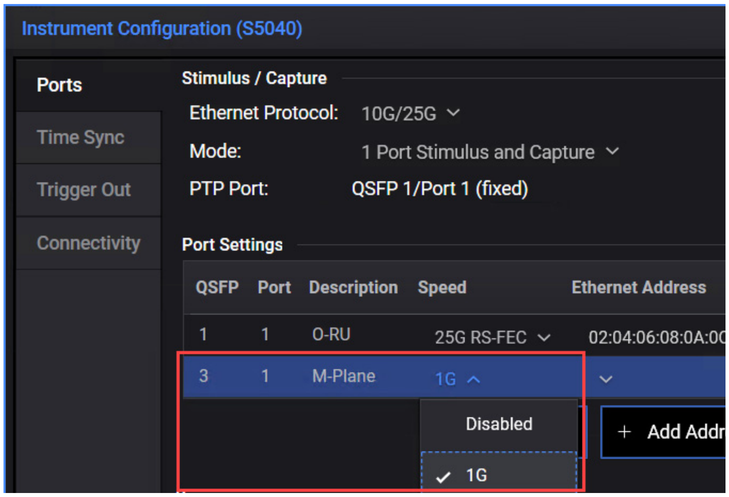


Figure 13 Setup requirements for M-Plane with FPGA Passthrough

In the U5040B Open RAN Studio software, you must ensure that M-Plane Passthrough Port is enabled under Instrument Configuration.

1 From the main menu of the U5040B Open RAN Studio software, click **Setup** > **Instrument Configuration...**.

2 In the **Port Settings** under the **Ports** tab, set the M-Plane **Speed** to **1G**.



Instrument Configuration (S5040)

Ports | Time Sync | Trigger Out | Connectivity

Stimulus / Capture

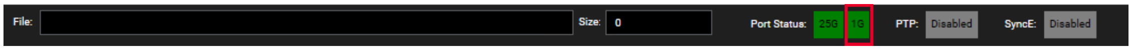
Ethernet Protocol: 10G/25G
Mode: 1 Port Stimulus and Capture
PTP Port: QSFP 1/Port 1 (fixed)

Port Settings

QSFP	Port	Description	Speed	Ethernet Address
1	1	O-RU	25G RS-FEC	02:04:06:08:0A:0C
3	1	M-Plane	1G	

Figure 14 Setup requirements for M-Plane with FPGA Passthrough

3 In the **Status** bar, verify the Link status, as highlighted below.



File: Size: 0 Port Status: 25G 1G PTP: Disabled SyncE: Disabled

Figure 15 Setup requirements for M-Plane with FPGA Passthrough

Section 2.3: M-Plane Configuration

To understand how to configure M-Plane, we must understand the configuration file structure.

2.3.1: Directory Structure of the M-Plane Client Software

By default, the configuration files for the M-Plane client software can be found in: *C:\ProgramData\Keysight\MPlane\Client*.

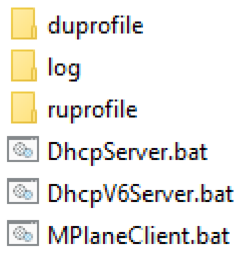

duprofile
log
ruprofile
DhcpServer.bat
DhcpV6Server.bat
MPlaneClient.bat

Figure 16 Directory structure of the M-Plane client

Folder / *File* name	Description
duprofile	Directory with DU configuration files
log	Directory with log files (this may not be created during the initial installation)
ruprofile	Directory with RU configuration files
DhcpServer.bat	Used to run a DHCP server, upon which, configuration can be provided to the O-RU
DhcpV6Server.bat	Used to run the DHCPv6 Server (for IPv6)
MPlaneClient.bat	Used to run the M-Plane NETCONF client

2.3.2: Flow of Configuration Files

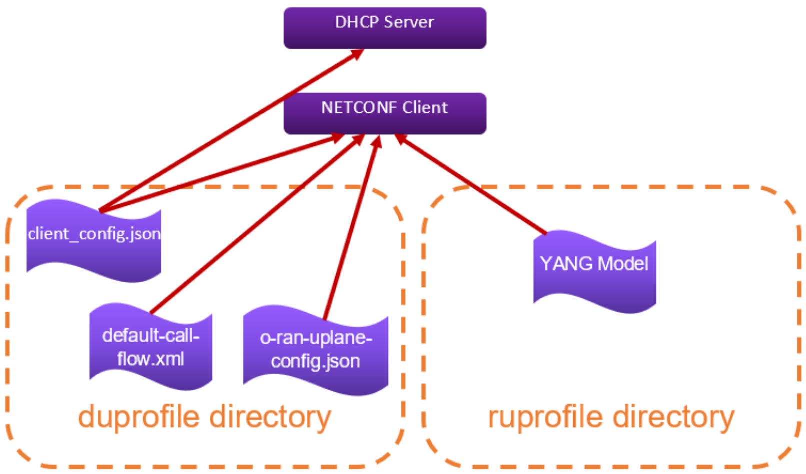


Figure 17 Flow of Configuration files in the M-Plane client

Config file name	Description
client_config.json	General client configuration
default-call-flow.xml	Defines the flow of actions that NETCONF client will perform, when the O-RU calls (it) home
o-ran-uplane-config.json	Defines the U-Plane configuration, which will be sent to the O-RU
YANG model	Described in Yet Another Next Gen (YANG) model.

Structure of the *client_config.json* file

```
{
  "general": {
    "username": "oranuser",
    "password": "o-ran-password",
    "infinite-run": "false",
    "log-level": "DEBUG",
    "log-dir": "log",
    "mplane-port": 4334,
    "call-flow-xml": "default_call_flow.xml"
  },
  "params": {
    "vlan-id": 0
  }
}
```

Parameters	Description
username	username for call-home session
password	password for call-home session
infinite-run	if true, the NETCONF client will keep the session open at the end of *default_call_flow.xml*
log-level	specifies logging level
log-dir	specifies directory for logging
mplane-port	TCP port for SSH / NETCONF session, default: 4334
call-flow-xml	name of file used for call flow script

Structure of the *o-ran-uplane-config.json* file

- This file defines the user plane carrier configuration, which is used in <edit-config/> operations.
- An example of this file populated can be found in *o-ran-uplane-conf.sample-filled.json*.

Snippet from the *default_call_flow.xml* file

```
<query>

    <get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

        <filter
xmlns:yanglib="urn:ietf:params:xml:ns:yang:ietf-yang-libra
ry"

                  type="xpath"

                  select="/yanglib:*//."/>

    </get>

    <create-subscription
xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">

        <stream>NETCONF</stream>

    </create-subscription>

    <supervision-watchdog-reset
xmlns="urn:o-ran:supervision:1.0"


action="start_supervision_thread">


<supervision-notification-interval>60</supervision-notific
ation-interval>

        <guard-timer-overhead>10</guard-timer-overhead>

    </supervision-watchdog-reset>

    <get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

        <filter
xmlns:o-ran-module-cap="urn:o-ran:module-cap:1.0"

                  type="xpath"

                  select="/o-ran-module-cap:*//."/>
        <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defau
lts">report-all</with-defaults>

    </get>

…

</query>
```

This file defines a sequential list of Remote Procedure Calls (RPCs) that the NETCONF Client will perform when the O-RU calls home. This normally follows the of the O-RU startup sequence per the M-Plane specification. However, nothing prohibits using different sequences.

Note that a different filename can be specified in *client_config.json*.

Section 2.4: Understanding M-Plane commands

To launch the **M-Plane Client Command Prompt** window, click **Start** > **Keysight M-Plane Toolkit** > **M-Plane Client Command Prompt**.

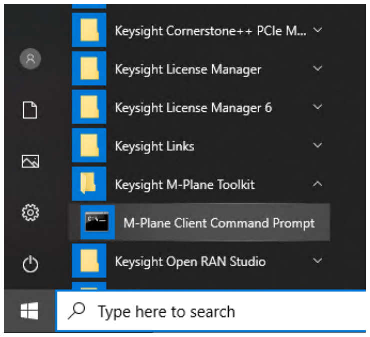
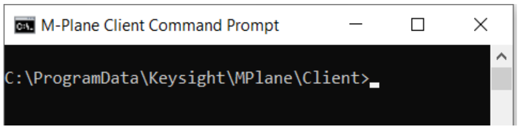


Figure 18 Launching M-Plane Client Command Prompt

The **M-Plane Client Command Prompt** window is launched, as shown in Figure 19.



```
C:\ProgramData\Keysight\MPlane\Client>_
```

Figure 19 Default view of M-Plane Client Command Prompt window

The root folder *C:\ProgramData\Keysight\MPlane\Client\* contains the configuration folders along with the *DhcpServer.bat* (to run the DHCP server) and *MPlaneClient.bat* (to run the NETCONF client) commands.

See M-Plane Configuration on page 23 in this document for more details.

2.4.1: Description of the M-Plane Commands

For DHCP Server

- Filename: DHCPServer.bat
- Usage:

  DhcpServer.bat --ip <IP-ADDRESS-TO-BIND-ON> --duprofile <du_directory>
- Example:

  DhcpServer.bat --ip 192.168.2.1 --duprofile duprofile
- Parameters

  --ip <IP-ADDRESS-TO-BIND-ON>

  - This is the IP Address of the windows Network Adapter that the DHCP Server will use. This should be a secondary Network Adapter configured with a static IP address.
  - WARNING! Running the DCHP server on the primary Adapter used to connect to a LAN may interfere with the DHCP server on the LAN and cause connectivity issues.
  - **NOTE**: There is currently a limitation that this IP address MUST end with **.1** through **.5**. This is because the DHCP server assumes those IP addresses are reserved for static IPs. DHCP clients will be given IP addresses **.6** or greater. Also, the subnet mask is assumed to be 255.255.255.0 (or /24 in CIDR notation).

  --duprofile <du_directory>

  - <du_directory> specifies the directory with configuration files for the O-DU emulation (DHCP Server / NETCONF client).

For DHCPv6 Server

- Filename: DhcpV6Server.bat
- Usage:
  - For Stateful:
    DhcpV6Server.bat -f PF_INET6 -t UDP -p 547 -a IP-ADDRESS-TO-BIND-ON -s STATEFUL
  - For SLAAC:
    DhcpV6Server.bat -f PF_INET6 -t UDP -p 547 -a IP-ADDRESS-TO-BIND-ON -s SLAAC
- Examples:

DhcpV6Server.bat -f PF_INET6 -t UDP -p 547 -a 192.168.56.1 -s STATEFUL

DhcpV6Server.bat -f PF_INET6 -t UDP -p 547 -a 192.168.56.1 -s SLAAC

- Parameters

  -f <family>

  - Valid values for <family> are PF_INET, PF_INET6, or PF_UNSPEC. Default value is PF_UNSPEC.

  -t <transport>

  - Valid values for <transport> are TCP or UDP. Default value is TCP.

  -p <port>

  - Port on which to bind. Default value is 547.

  -a <address>

  - This is the IP Address of the windows Network Adapter that the DHCPv6 Server will use. This should be a secondary Network Adapter configured with a static IP address.
    WARNING! Running the DCHPv6 server on the primary Adapter used to connect to a LAN may interfere with the DHCPv6 server on the LAN and cause connectivity issues.
    NOTE: There is currently a limitation that this IP address MUST end with **.1** through **.5**. This is because the DHCP server assumes those IP addresses are reserved for static IPs. DHCP clients will be given IP addresses **.6** or greater. Also, the subnet mask is assumed to be 255.255.255.0 (or /24 in CIDR notation).

  -s <state>

  - Method to be used for the IP address assignment in an IPv6 network. Valid values are SLAAC or STATEFUL. Default value is STATEFUL.

For NETCONF client

- Filename: MPlaneClient.bat
- Usage:

  MPlaneClient.bat --ip <IP-ADDRESS-TO-BIND-ON> --duprofile <du_directory> --ruprofile <ru_directory>
- Example:

  MPlaneClient.bat --ip 192.168.2.1 --duprofile duprofile --ruprofile ruprofile
- Parameters

  --ip <IP-ADDRESS-TO-BIND-ON>

  - This should be the same IP address used for the DHCP server (when using DHCP).
  - Specifies the IP Address that the NETCONF client will listen on (using the port number in <du_directory>/client_config.json, which is 4334, by default).

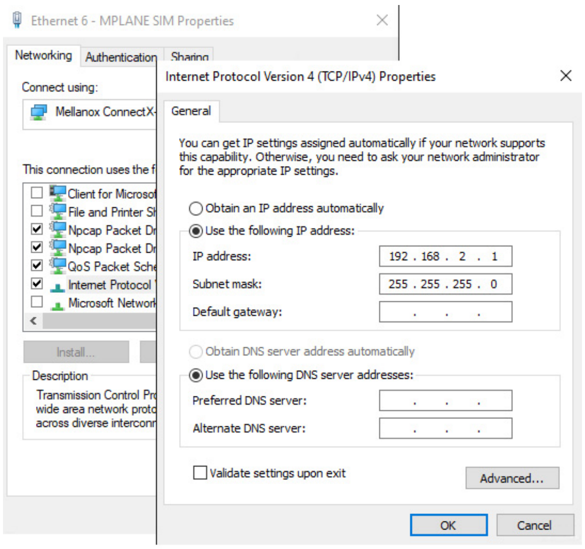
  --duprofile <du_directory>

  - This specifies the directory with configuration files for the O-DU emulation (DHCP Server / NETCONF client). Normally, this should be set to "duprofile/v1" (replace v1 with the proper M-Plane specification version used by the O-RU).

  --ruprofile <ru_directory>

  - This specifies the directory with the O-RU configuration profile. Normally, this should be set to "ruprofile/v1" (replace v1 with the proper M-Plane specification version used by the O-RU). Note that this directory contains the YANG Models for the O-RU and they should not be modified.

Example for DHCP Server and NETCONF Client



Ethernet 6 - MPLANE SIM Properties

Networking | Authentication | Sharing

Connect using:

Mellanox ConnectX-

This connection uses the f

- ☐ Client for Microsof
- ☐ File and Printer Sh
- ☑ Npcap Packet Dr
- ☑ Npcap Packet Dr
- ☑ QoS Packet Sche
- ☑ Internet Protocol
- ☐ Microsoft Network

Install...

Description

Transmission Control Pro wide area network protc across diverse intercon

Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

- ○ Obtain an IP address automatically
- ● Use the following IP address:

IP address:	192 . 168 . 2 . 1
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

- ○ Obtain DNS server address automatically
- ● Use the following DNS server addresses:

Preferred DNS server:	. . .
Alternate DNS server:	. . .

☐ Validate settings upon exit

Advanced...

OK | Cancel

Figure 20 Configuring DHCP Server + NETCONF Client

1 Configure Secondary Network Connection
   - Configure Internet Protocol Version 4 with:
      - IP address: 192.168.2.1
      - Subnet mask: 255.255.255.0
      - Default gateway: <empty>
   - Disable IPv6

2 Launch two **M-Plane Client Command Prompt** windows.
   - On window 1: Run the DHCP Server
      - DhcpServer.bat --ip 192.168.2.1 --duprofile duprofile/v1
   - On window 2: Run NETCONF Client
      - MPlaneClient.bat --ip 192.168.2.1 --duprofile duprofile/v1 --ruprofile ruprofile/v1

Section 2.5: Overview on RPC

The key elements in RPC are:

- get
- edit-config
- create-subscription
- supervision-watchdog-reset
- file-upload

2.5.1: RPC: get

Description

The "get" RPC retrieves running configuration information from the O-RU / NETCONF Server.

Sub-elements

- **Filter** (optional): specifies filtering criteria for the get RPC
  - Attribute: type = xpath | subtree
  - Attribute: select = subtree to select

```
EXAMPLE: get module capabilities

    <get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

        <filter
xmlns:o-ran-module-cap="urn:o-ran:module-cap:1.0"

                  type="xpath"

                  select="/o-ran-module-cap:*//."/>

        <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defau
lts">report-all</with-defaults>

    </get>
```

2.5.2: RPC: edit-config

Description

This RPC edits the existing configuration in the O-RU / NETCONF Server.

Attributes

- **xmlns**: defines the namespace
- **action** (optional): for some specific operations, an action is specified. See “Actions” for more details

Sub-elements

- **target**: specifies the datastore (that is, “running” or “startup”) to be modified.
- **default-operation**: specifies if the configuration should be a “merge” or a “replace”. In most cases, “merge” is used to merge into the existing configuration.
- **config**: specifies the configuration to merge.

```
Example: merge LBM configuration

    <edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

        <target>

            <running/>

        </target>

        <default-operation>merge</default-operation>

        <config>

           <md-data-definitions xmlns="urn:o-ran:lbm:1.0">

                <maintenance-domain>

                    <id>KEYS_xyz_LBM</id>

                    <name>level_7</name>

                    <md-level>7</md-level>

                </maintenance-domain>

            </md-data-definitions>
        </config>
    </edit-config>
```

2.5.3: RPC: create-subscription

Description

This RPC creates a subscription to the specified stream.

Attributes

- **xmlns**: defines the namespace for the notification to subscribe to.

Sub-elements

- **stream**: specifies the stream to subscribe to. This can be “NETCONF”, or “supervision-notification”.

```
Example: Subscribe to NETCONF stream
    <create-subscription
xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
        <stream>NETCONF</stream>
    </create-subscription>
```

2.5.4: RPC: supervision-watchdog-reset

Description

This RPC creates a subscription to the specified stream.

Attributes

- **xmlns**: defines the namespace for the supervision watchdog reset.
- **action**: this should be set to “start_supervision_thread”

Sub-elements

- **stream**: specifies the stream to subscribe to.

See *ORAN-WG4.MP.0-v01.00 section 3.6* for more details.

```
Example:

    <supervision-watchdog-reset
xmlns="urn:o-ran:supervision:1.0"

action="start_supervision_thread">

<supervision-notification-interval>60</supervision-notific
ation-interval>

        <guard-timer-overhead>10</guard-timer-overhead>

    </supervision-watchdog-reset>
```

2.5.5: RPC: file-upload

Description

This RPC requests a file upload.

Attributes

- **xmlns**: defines the namespace for the supervision watchdog reset.

Sub-elements

- **local-logical-file-path**: specifies the local file on the NETCONF server to upload.
- **remote-file-path**: specifies the URL, where the file on the O-DU/NMS must be uploaded
- **password**: specifies the password for the file upload

See *ORAN-WG4.MP.0-v01.00 section 9.2* for more details.

```
<file-upload xmlns="urn:o-ran:file-management:1.0">

    <!-- This will be sent if only we receive success in
start query-->

        <local-logical-file-path/>

        <!-- Add the correct remote file path and password
to make this successful in
troubleshooting-log-location.json file -->
```

```
        <remote-file-path/>
        <password>
            <password/>
        </password>
    </file-upload>
```

2.5.6: RPC Actions

The following table lists various RPC actions along with their description.

RPC Actions	Description
start_supervision_thread	Should be used with <supervision-watchdog-reset/> to start a thread that will handle the supervision-notification events.
wait_for_locked_block	Used in conjunction with getting <syncxmlns="urn:o-ran:sync:1.0"> to communicate, when the S-plane is locked. Note that this action pauses the script from executing other commands until the O-RU S-Plane is in a LOCKED state.
create_new_vlan	Should be used with <edit-config/> RPCs that add new VLANs.
add_processing_element	Should be used with <edit-config/> RPCs that add new processing elements.
endpoint_carrier_configuration	Should be used with <edit-config/> RPCs that modify endpoint carrier configurations.
endpoint_carrier_link_configuration	Should be used with <edit-config/> RPCs that modify endpoint carrier link configurations.
activate_carrier	Should be used with <edit-config/> RPCs that activate carriers.
log_management_thread	Should be used with <start-troubleshooting-logs/> RPCs.
endpoint_carrier_configuration_negative	Used in negative tests for endpoint carrier configuration.

Section 2.6: M-Plane configuration examples

2.6.1: Supervisor Watchdog setup

```
<create-subscription
xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
        <stream>NETCONF</stream>
    </create-subscription>
    <supervision-watchdog-reset
xmlns="urn:o-ran:supervision:1.0"

action="start_supervision_thread">

<supervision-notification-interval>60</supervision-notific
ation-interval>
        <guard-timer-overhead>10</guard-timer-overhead>
    </supervision-watchdog-reset>
```

2.6.2: Get Modules Capabilities

```
<get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <filter
xmlns:o-ran-module-cap="urn:o-ran:module-cap:1.0"
                  type="xpath"
                  select="/o-ran-module-cap:*//."/>
        <with-defaults
xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-defau
lts">report-all</with-defaults>
    </get>
```

NOTE

Similar examples are in the *default-call-flow.xml* file for the following procedures:

1. Getting Transceivers
2. Getting Interfaces
3. Getting Hardware
4. Getting U-Plane configuration
5. Various other examples

2.6.3: Test for Alarm Generation

```
<get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
action="wait_for_locked_thread">
        <filter type="subtree">
            <sync xmlns="urn:o-ran:sync:1.0">
                <sync-status/>
            </sync>
        </filter>
    </get>
```

2.6.4: Create new VLAN

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
                   action="create_new_vlan">
        <target>
            <running/>
        </target>
        <default-operation>merge</default-operation>
        <config>
```

```
            <interfaces
xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
                <interface>
                    <name>KEYS_if_xyz_vlan0</name>
                    <type
xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">i
anaift:l2vlan</type>
                    <mac-address
xmlns="urn:o-ran:interfaces:1.0">00:03:a2:0a:00:00</mac-ad
dress>
                    <vlan-id
xmlns="urn:o-ran:interfaces:1.0">2</vlan-id>
                    <base-interface
xmlns="urn:o-ran:interfaces:1.0">MAC0</base-interface>
                </interface>
            </interfaces>
        </config>
    </edit-config>
```

2.6.5: LBM Configuration

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <target>
            <running/>
        </target>
        <default-operation>merge</default-operation>
        <config>
           <md-data-definitions xmlns="urn:o-ran:lbm:1.0">
                <maintenance-domain>
                    <id>KEYS_xyz_LBM</id>
                    <name>level_7</name>
```

```
                        <md-level>7</md-level>
                    </maintenance-domain>
                </md-data-definitions>
            </config>
        </edit-config>
```

NOTE

After the LBM configuration has been applied to the O-RU, you can use the "Loopback Message (LBM)" feature in the Open RAN Studio GUI to test connectivity with the O-RU.
This feature is available in the Instrument Configuration > Connectivity tab of the Open RAN Studio GUI. When you enable this feature, it allows you to send Loopback request messages and receive Loopback responses from O-RU.

2.6.6: Configure mplane-info

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <target>
            <running/>
        </target>
        <default-operation>merge</default-operation>
        <config>
            <mplane-info
xmlns="urn:o-ran:mplane-interfaces:1.0">
                <searchable-mplane-access-vlans-info>
                    <vlan-range>
                        <lowest-vlan-id>1</lowest-vlan-id>
                      <highest-vlan-id>1</highest-vlan-id>
                    </vlan-range>
                </searchable-mplane-access-vlans-info>
                <m-plane-interfaces>
                    <m-plane-ssh-ports/>
                </m-plane-interfaces>
            </mplane-info>
        </config>
    </edit-config>
```

NOTE The <searchable-mplane-access-vlans-info> used in the script, is stored in the O-RU in reset persistent memory. So, this command should either be removed from the *default-call-flow.xml*, or adjusted for proper VLAN scan range.

2.6.7: Add Processing Element

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
                   action="add_processing_element">
        <target>
            <running/>
        </target>
        <default-operation>merge</default-operation>
        <config>
            <processing-elements
xmlns="urn:o-ran:processing-element:1.0">
<transport-session-type>ETH-INTERFACE</transport-session-t
ype>
                <ru-elements>
<name>KEYS_xyz_processing-element01</name>
                    <transport-flow>
<interface-name>KEYS_if_xyz_vlan0</interface-name>
                        <eth-flow>
<ru-mac-address>a0:ce:c8:17:7a:41</ru-mac-address>
                            <vlan-id>2</vlan-id>
```

```
<o-du-mac-address>38:AF:D7:D5:CE:EF</o-du-mac-address>
                        </eth-flow>
                    </transport-flow>
                </ru-elements>
            </processing-elements>
        </config>
    </edit-config>
```

2.6.8: Endpoint Carrier Configuration

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"

                   action="endpoint_carrier_configuration"

                 comment="All the user-plane-configuration
is filled using o-ran-uplane-conf.json file">

        <target>

            <running/>

        </target>

        <default-operation>merge</default-operation>

        <config>

            <user-plane-configuration
xmlns="urn:o-ran:uplane-conf:1.0">

            </user-plane-configuration>

        </config>

    </edit-config>
```

2.6.9: Endpoint Carrier Link Configuration

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"


action="endpoint_carrier_link_configuration"

                 comment="All the user-plane-configuration
is filled using o-ran-uplane-conf.json file">

        <target>

            <running/>

        </target>

        <default-operation>merge</default-operation>

        <config>

            <user-plane-configuration
xmlns="urn:o-ran:uplane-conf:1.0"/>
```

```
                </config>

            </edit-config>
```

2.6.10: Activate Carrier

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"

                   action="activate_carrier"

                 comment="All the user-plane-configuration
is filled using o-ran-uplane-conf.json file">

        <target>

            <running/>

        </target>

        <default-operation>merge</default-operation>

        <config>

            <user-plane-configuration
xmlns="urn:o-ran:uplane-conf:1.0"/>

        </config>

    </edit-config>
```

2.6.11: Log Management

```
<start-troubleshooting-logs
xmlns="urn:o-ran:troubleshooting:1.0"


action="log_management_thread"/>

    <file-upload xmlns="urn:o-ran:file-management:1.0">

    <!-- This will be sent if only we receive success in
start query-->

        <local-logical-file-path/>

        <!-- Add the correct remote file path and password
to make this successful in

             troubleshooting-log-location.json file -->
        <remote-file-path/>
        <password>
            <password/>
        </password>
    </file-upload>
```

2.6.12: Example: flow for M-Plane RPC configuration

The following steps show how to configure M-Plane RPC, based on *Fig. 24* in *ORAN-WG4-MP-0-V01.00* specification.

1. NETCONF Client / O-RU Controller sets parameters of objects on the NETCONF Server / O-RU

   ```
   <rpc> <edit-config><source><running/><…>
   ```

2. Upon receiving a value change request from the O-RU Controller, the O-RU automatically updates the values.
3. NETCONF Server / O-RU sends an acknowledgment to the NETCONF Client / O-RU Controller

   ```
   <rpc-reply> <OK>
   <rpc-reply> <rpc-error> <error-xxx>
   ```

Section 2.7: Configuring M-Plane VLAN

The S5040A Appliance supports VLANs on the PC NIC.

1 Navigate to www.intel.com.

Click here to directly access the web page for Intel's Network Adapter Drivers. This direct link may be subject to change in the future.

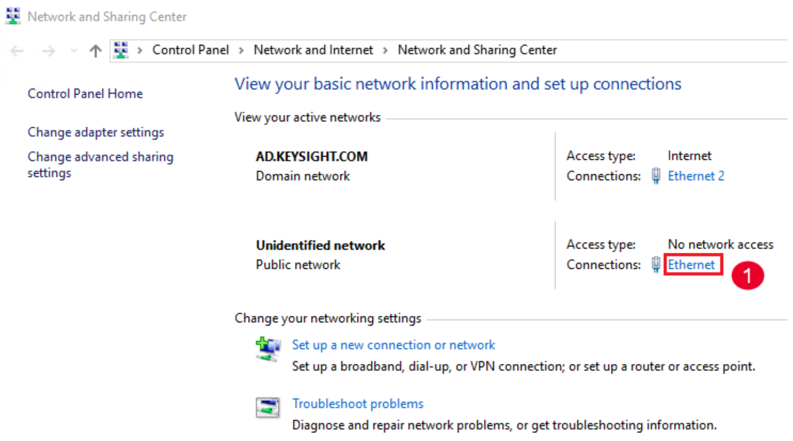
2 Download and install the latest Intel NIC driver and ProSet utilities for Windows. (Note that the latest versions available for download may be newer than the below-mentioned versions.)

- Wired_driver_27.2_x64.zip
- Wired_PROSet_27.2_x64.zip

3 Create the new VLAN on the secondary NIC in Windows on the S5040A Open RAN Studio Appliance. This can be either NIC, but should not be the NIC used for LAN access to the system. It should be the NIC used for M-Plane passthrough. The VLAN can be created in the VLAN tab of the adapter properties as shown in the following steps:

   i Open "Network and Sharing Center" window.

   ii Click the relevant "Ethernet" adapter.



Network and Sharing Center

Control Panel > Network and Internet > Network and Sharing Center

View your basic network information and set up connections

View your active networks

AD.KEYSIGHT.COM
Domain network
Access type: Internet
Connections: Ethernet 2

Unidentified network
Public network
Access type: No network access
Connections: Ethernet

Change your networking settings

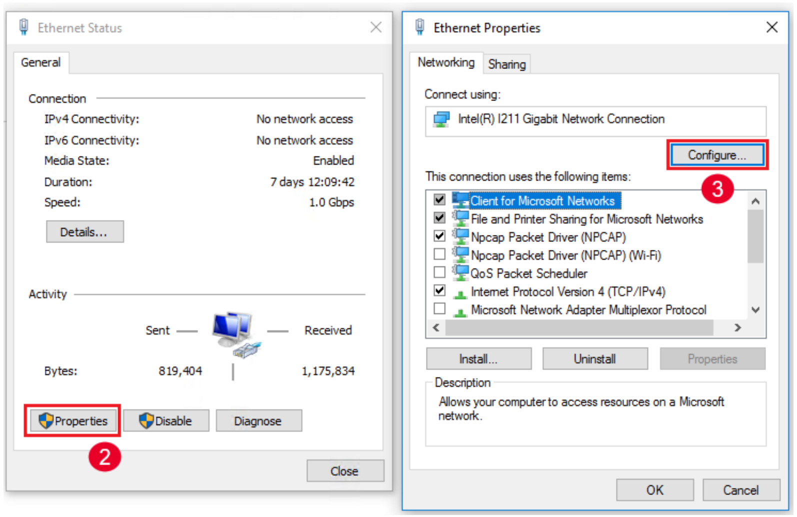
Set up a new connection or network
Set up a broadband, dial-up, or VPN connection; or set up a router or access point.

Troubleshoot problems
Diagnose and repair network problems, or get troubleshooting information.

Figure 21 Selecting proper Ethernet adapter for VLAN configuration

iii In the “Ethernet Status” dialog box that appears, click “Properties”.

iv In the “Ethernet Properties” dialog box that appears, click “Configure”.



Connection

IPv4 Connectivity:	No network access
IPv6 Connectivity:	No network access
Media State:	Enabled
Duration:	7 days 12:09:42
Speed:	1.0 Gbps

Activity

	Sent	Received
Bytes:	819,404	1,175,834

Connect using:

Intel(R) I211 Gigabit Network Connection

This connection uses the following items:

- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] Npcap Packet Driver (NPCAP)
- [ ] Npcap Packet Driver (NPCAP) (Wi-Fi)
- [ ] QoS Packet Scheduler
- [✓] Internet Protocol Version 4 (TCP/IPv4)
- [ ] Microsoft Network Adapter Multiplexor Protocol

Description

Allows your computer to access resources on a Microsoft network.

Figure 22 Dialog boxes to configure Ethernet adapter for VLAN configuration

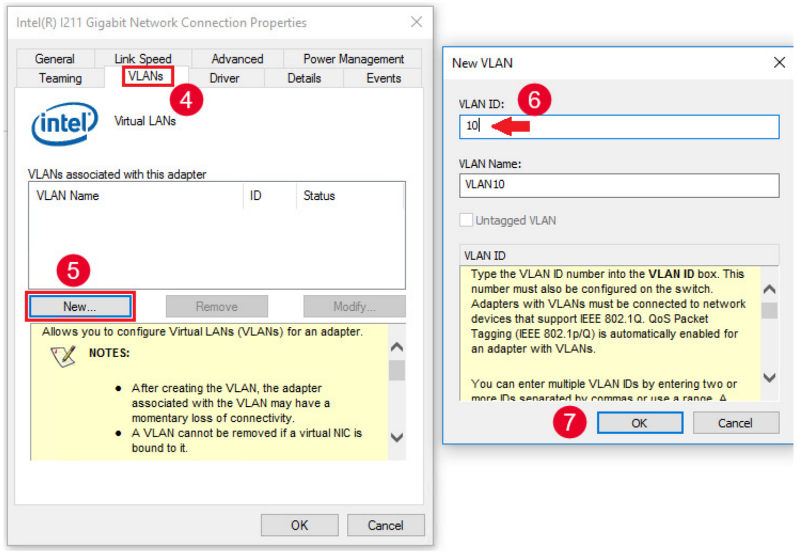
v In the “Network Connection Properties” for the selected Ethernet Adapter, click “VLANs” tab.

vi Click the “New...” button.

vii Modify the ‘VLAN ID’ and ‘VLAN Name’ fields.

For easy identification, it is suggested naming the VLAN as ‘Ethernet 2 VLAN 10’ or similar.

viii Click “OK”.



VLANs associated with this adapter

VLAN Name	ID	Status

Allows you to configure Virtual LANs (VLANs) for an adapter.

NOTES:

- After creating the VLAN, the adapter associated with the VLAN may have a momentary loss of connectivity.
- A VLAN cannot be removed if a virtual NIC is bound to it.

VLAN ID: 10

VLAN Name: VLAN10

☐ Untagged VLAN

VLAN ID

Type the VLAN ID number into the **VLAN ID** box. This number must also be configured on the switch. Adapters with VLANs must be connected to network devices that support IEEE 802.1Q. QoS Packet Tagging (IEEE 802.1p/Q) is automatically enabled for an adapter with VLANs.

You can enter multiple VLAN IDs by entering two or more IDs separated by commas or use a range. A

Figure 23 Dialog boxes to configure VLAN ID and name for VLAN configuration

ix Make sure there is NO IP assigned to the secondary NIC.

x Assign the IP address used for M-Plane to the virtual NIC created with the VLAN in the steps above. This IP address will be used with the “-ip” parameter in the DHCP server and netconf client.

NOTE **If your O-RU does not work when the IP 192.168.1.x is used for the M-Plane DHCP Server/Netconf client. Keysight recommends using a different IP address, such as 10.10.10.x, 192.168.2.x or any other subnet.**

4 The VLAN scan range must be edited in the default-call-flow.xml.

The default range is ‘1..1’, which is not good for most configurations. The example below shows how to change the range to ‘201..210’. Note that these values will be persisted in the O-RU automatically.

```
<edit-config
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <target>
            <running/>
        </target>
        <default-operation>merge</default-operation>
        <config>
            <mplane-info
xmlns="urn:o-ran:mplane-interfaces:1.0">
                <searchable-mplane-access-vlans-info>
                    <vlan-range>
                      <lowest-vlan-id>201</lowest-vlan-id>
<highest-vlan-id>210</highest-vlan-id>
                    </vlan-range>
                </searchable-mplane-access-vlans-info>
                <m-plane-interfaces>
                    <m-plane-ssh-ports/>
                </m-plane-interfaces>
            </mplane-info>
        </config>
    </edit-config>
```

3. Understanding O-RU M-Plane Conformance

Section 3.1: Overview

This section specifies the test cases and configurations that can be used to perform interoperability testing of O-RAN RU.

3.1.1: Test Environment

Figure 24 represents a schematic diagram, which depicts a test environment. Interfaces towards system under test (O-RU) are all standard specified interfaces.

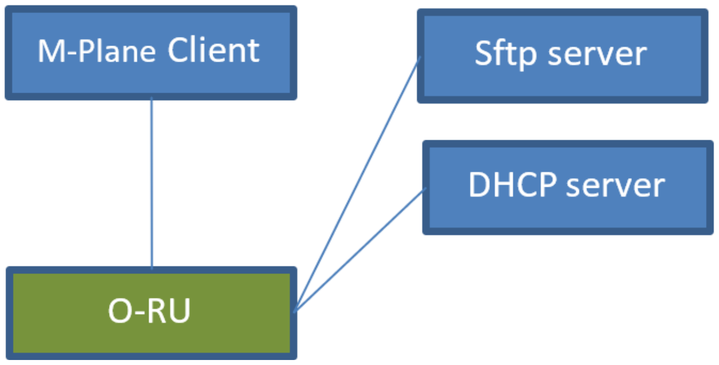


Figure 24 Test environment for M-Plane

Section 3.2: Understanding Installation and Execution Procedure

3.2.1: DHCP Server and M-Plane Client in O-DU

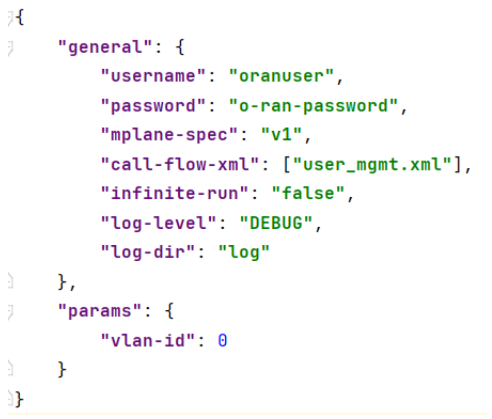
DHCP Server: DHCP server provides information of the transport layer address of the O-RU together with the identity of the NETCONF client. This identity encodes either the transport layer address or FQDN of the NETCONF client. If an FQDN is signaled, the O-RU shall use the DNS server address provided by the DHCP server to recover the IP address corresponding to FQDN of the NETCONF client. It is required as part of the "call home" procedure when the O-RU NETCONF client starts up. Further details are provided below including where the files are located and how to start the server.

M-Plane Client: It is the identity of Netconf client. Currently, we support M-Plane Client version 1 to version 6. M-Plane yang models are available corresponding to each M-Plane specification version and can be added or modified or deleted in each version, corresponding to the specification. Test cases provided in the conformance specifications are developed based with the YANG model supported in each version. This document refers to test cases for version 1 support.

- For DHCP, dhcp-server.exe file
- For M-Plane client, Keysight-5G-ORAN-M-PLANE-Client<version>.exe file
- DU profile present under the "profile/" folder contains configuration files, "client_config.json" for a successful and stateful mplane client run. The contents of client_config.json files are:
    - username: Username for the ssh login to RU.
    - password: Password for the ssh login to RU.
    - (optional) mplane-port: Port at which it will start listening for RU connection, most cases do not require a change; default is 4334.
    - (optional) ru-profile-path: Relative or Full Path to RU Profile directory containing RU jsons + Keysight mplane XML; by default, it is bundled with the mplane source.
    - (optional) call-flow-xml: Name of the call flow xml file to be used in configuring the RU. Should reside inside the DU profile dir; default is provided. You can only add to it, deletion is not recommended. Default is bundled with the duprofile called default_call_flow.xml.
    - (optional) infinite-run: "false" means client will stop after one connection with RU finishes; "true" means client will again start to listen if connection with RU closes. Default is false.

- optional; log-level: Level of logging wanted. Possible values are: ("TRACE", "DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"). Default is INFO.
- log-dir: Full or relative path to the directory where mplane client log files should be stored.

The sample config file used is displayed in Figure 25.



```
{
    "general": {
        "username": "oranuser",
        "password": "o-ran-password",
        "mplane-spec": "v1",
        "call-flow-xml": ["user_mgmt.xml"],
        "infinite-run": "false",
        "log-level": "DEBUG",
        "log-dir": "log"
    },
    "params": {
        "vlan-id": 0
    }
}
```

Figure 25 Client config file updated for user management

- Call flow xml file specifies steps of a test case. Call flow xml files are present in the folder 'profile/duprofile/'. Table 1 gives the list of test case with corresponding call flow xml file and input json file.

Table 1 List of test cases with Call Flow XML file and JSON file

#	Feature	Functional Test Case	JSON file	Call flow XML file
1	User Management	Sudo on Hybrid M-plane Architecture (positive case)	NA	3.1.8.1_3.1.8.6_Sudo_on_M-plane_architecture.xml
2		Access Control Sudo (negative case)		3.1.8.1_3.1.8.6_Sudo_on_M-plane_architecture.xml
3		Access Control NMS (negative case)		3.1.8.3_Access_Control_NMS.xml
4		Access Control FM-PM (negative case)		3.1.8.4_Access_Control_FM-PM.xml
5		Access Control SWM (negative case)		3.1.8.5_Access_Control_SWM.xml

#	Feature	Functional Test Case	JSON file	Call flow XML file
6	M-Plane Connection Supervision	M-Plane connection supervision (positive case)	NA	3.1.3_M-Plane_Connection_Supervision.xml
7		M-Plane Connection Supervision (negative case)		
8	Software Download, Install, Activate	O-RU Software Update and Install (positive case)	swm-download-location	3.1.6.1_3.1.7.1_O_RU_Software_Update_Install_Activation.xml
9		O-RU Software Update (negative case)		3.1.6.2_O_RU_Software_Update.xml
10		Software Activation without Reset		3.1.6.1_3.1.7.1_O_RU_Software_Update_Install_Activation.xml
11		Supplemental Reset after Software Activation		3.1.7.2_Supplemental_Reset_after_Software_Activation.xml
12	External IO	External Input Port State Detection	o-ran-io	3.1.9.1_External_Input_Port_State_Detection.xml
13		External Output Port State Control	o-ran-io	3.1.9.2_External_Output_Port_State_Control.xml
14	Fault Management	O-RU Alarm Notification Generation	NA	3.1.5_Fault_Management.xml
15		Retrieval of Active Alarm List		
16		Subscription to Notifications		
17		Manage Alarm Request		8.2_Manage_Alarms_Request.xml
18	Log Management	Troubleshooting	troubleshooting-log-location	11.2.1_Troubleshooting.xml
19		Trace		11.2.2_Trace.xml
20	Performance Management	Measurement Activation and Deactivation	o-ran-pm	7_Performance_Management.xml
21		Collection and Reporting of Measurement Result		
22		NETCONF process		
23		File Management process		

#	Feature	Functional Test Case	JSON file	Call flow XML file
24	File Management	File System Structure	file-retrieval	9_File_Management.xml
25		File Management Operation: upload		11.2.1_Troubleshooting.xml
26		File Management Operation: retrieve file list	file-retrieval	9_File_Management.xml
27		File Management Operation: download	file-download-location	9_File_Management.xml
28	Sync, ORU Configurability Tests	Sync Status Object/ORU Configurability positive and negative case		default_call_flow.xml
29	ALD Communication	ALD Communications Test		3.1.11.1_ALD_Communications_Test.xml
30	Default	Default Test	o-ran-uplane-conf	default_call_flow.xml
31	Configuration Management	6.2 Framework for optional feature handling		6_Configuration_Management.xml
32		6.3 M-Plane Operational State		
33		6.4.1 Introduction		
34		6.4.2 Subscribing to updates from an O-RU		
35	4.8 O-RU Adaptive Delay Capability		o-ran-delay	4.8_O-RU_Adaptive_Delay_Capability.xml
36	4.10_O-RU_Monitoring_of_CU_Plane_Connectivity			4.10_O-RU_Monitoring_of_CU_Plane_Connectivity.xml
37	3.1.1.1_Transport_Handshake_in_IPv4_Environment		Client_config	Default_call_flow.xml
38	3.1.1.2_Transport_Handshake_in_IPv4_Environment		Client_config	<Any xml>
39	Hybrid Functionality with call home			<Any xml>
40	Hybrid Functionality without call home			<Any xml>
41	Connectivity Check	Ethernet Connectivity Monitoring	NA	4.6_Ethernet_Connectivity_Monitoring.xml

NOTE For default_call_flow.xml, values must be filled in *o-ran-uplane-conf.json* file. The sample filled json for reference is “*o-ran-uplane-conf.sample-filled.json*”. It is available in the same json location.

3.2.2: SFTP Server setup

Run the steps below to install SFTP server in Windows for test cases such as File management and Software management.

- Install Cygwin from http://www.cygwin.com. Root installation directory is C:\cygwin.
  - During installation, select zip, unzip packages from Archive category followed with openssh and openssl from net package.
- Configure SSHD service
  - Navigate to C:\cygwin and edit Cygwin.bat and add the line:

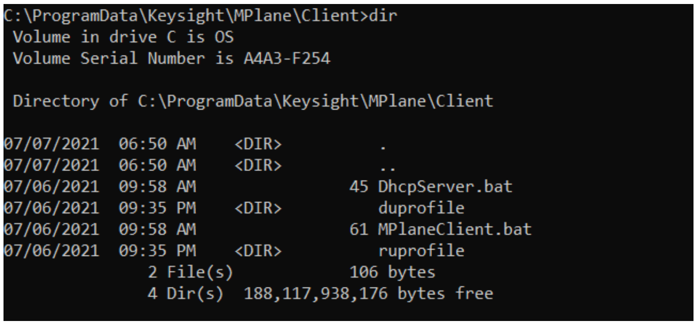
    ```
    set CYGWIN=binmode ntsec
    ```
  - Verify Cygwin is installed properly by executing `cygrunsrv -h`. This will display help options if Cygwin is installed properly.
  - Run `Cygwin.bat` and run the command `ssh-host-config`.
  - During questions prompted, enter *** Query: Enter the value of CYGWIN for the deamon: [ ] `binmode ntsec`.
- Running SSHD service
  - Run C:\cygwin\cygwin.bat and run `net start cygsshd`.

3.2.3: Executing and starting M-Plane client and server

Following assumptions are made to trigger execution and to start M-Plane client and server. Note that the M-Plane specification version used is V1.

1 Double click the exe file provided, which is “*MPlane_setup_full_1.0.30014.0.exe*” for installation.

2 Open the M-Plane command prompt and the folder structure is displayed as shown in Figure 26.



```
C:\ProgramData\Keysight\MPlane\Client>dir
 Volume in drive C is OS
 Volume Serial Number is A4A3-F254

 Directory of C:\ProgramData\Keysight\MPlane\Client

07/07/2021  06:50 AM    <DIR>          .
07/07/2021  06:50 AM    <DIR>          ..
07/06/2021  09:58 AM                45 DhcpServer.bat
07/06/2021  09:35 PM    <DIR>          duprofile
07/06/2021  09:58 AM                61 MPlaneClient.bat
07/06/2021  09:35 PM    <DIR>          ruprofile
               2 File(s)            106 bytes
               4 Dir(s)  188,117,938,176 bytes free
```

Figure 26 Folder structure displayed in M-Plane command prompt

Perform the following steps:

1. Update the call flow xml name in the *client_config.json* file in 'duprofile/v1/conformance-test-cases' folder as per each test case with reference to Table 1.
2. Start DHCP server in ODU in a command terminal:

   ```
   DhcpServer.bat --ip IP-ADDRESS-TO-BIND-ON --duprofile "duprofile/<version required>" # sample duprofile is provided in the dir ../profile
   ```

   The package is updated to support versions 1 to 8.

   Version required may range from v1 to v8 based on the need. Following are the sample configuration.

   ```
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v1"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v2"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v3"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v4"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v5"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v6"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v7"
   DhcpServer.bat -ip 192.168.56.1 -du "duprofile\v8"
   ```

3. Run M-Plane client in ODU in another windows command terminal:

   ```
   MPlaneClient.bat --ip IP-ADDRESS-TO-BIND-ON --duprofile "duprofile/v1" -ru "ruprofile\v1" # sample duprofile is provided in the dir ../profile
   ```

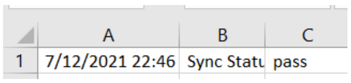
   The package is updated to support versions 1 to 8.

   Version required may range from v1 to v8 based on the need. Following are the sample configuration.

   ```
   MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v2" -ru "ruprofile\v2"
   ```

```
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v3" -ru
"ruprofile\v3"
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v4" -ru
"ruprofile\v4"
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v5" -ru
"ruprofile\v5"
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v6" -ru
"ruprofile\v6"
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v7" -ru
"ruprofile\v7"
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v8" -ru
"ruprofile\v8"
```

- The logs of M-plane client are stored in *C:\ProgramData\Keysight\MPlane\Client\log*. Additionally, the results are showed in csv format in the log folder.



	A	B	C
1	7/12/2021 22:46	Sync Statu	pass

Figure 27 Sample log file in CSV format

Section 3.3: O-RU Management Plane Features

O-RU M-plane conformance covers the following features:

3.3.1: DHCP Serv Transport and Handshake Test Scenario

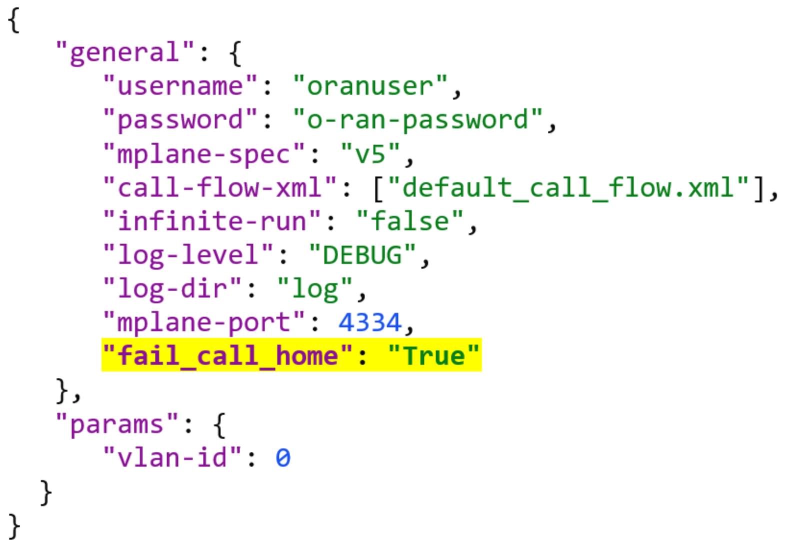
This is the scenario for DHCP connection establishment, which is, by default, covered with all IPV4 test cases. There are two flows, which are described in the steps below:

Steps to execute

1. Fill the client config file (shown in Figure 28 and Figure 29) under “du/profile/v1/” with call-flow-xml value as “default_call_flow.xml” as listed in Table 1.
2. Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that M-Plane server is running in O-RU.

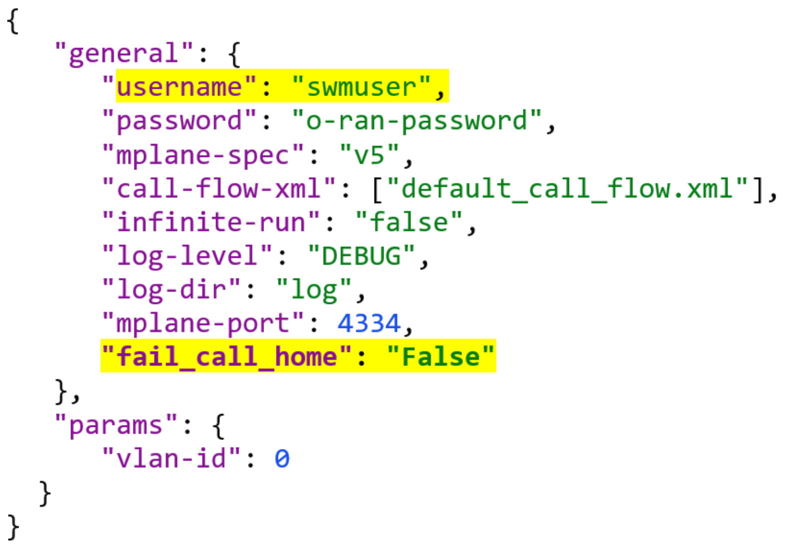
Result

1. Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.
2. Pass Logs for the execution is generated ‘\log’ folder.



```
{
   "general": {
      "username": "oranuser",
      "password": "o-ran-password",
      "mplane-spec": "v5",
      "call-flow-xml": ["default_call_flow.xml"],
      "infinite-run": "false",
      "log-level": "DEBUG",
      "log-dir": "log",
      "mplane-port": 4334,
      "fail_call_home": "True"
   },
   "params": {
      "vlan-id": 0
  }
}
```

Figure 28 Flow 1: client_config.json



```
{
   "general": {
      "username": "swmuser",
      "password": "o-ran-password",
      "mplane-spec": "v5",
      "call-flow-xml": ["default_call_flow.xml"],
      "infinite-run": "false",
      "log-level": "DEBUG",
      "log-dir": "log",
      "mplane-port": 4334,
      "fail_call_home": "False"
   },
   "params": {
      "vlan-id": 0
  }
}
```

Figure 29 Flow 2: client_config.json

3.3.2: Manage Alarm Request

List of Test cases

1. Subscription to Notifications

This is covered in Steps to execute on page 63.

3.3.3: M-Plane Connection Supervision

List of Test cases

1. M-Plane connection supervision (positive case)

2. M-Plane Connection Supervision (negative case)

For both the test cases 1 and 2,

Steps to execute

1. Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “supervision.xml” as listed in Table 1. There is no input json file for this test case.
2. Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that M-Plane server is running in O-RU.

Result

1. Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.
2. Pass Logs for the execution is generated ‘..\pkg_tmp\ Keysight_5G_ORAN_M-PLANE_Client\log’ folder.

Section 3.4: Retrieval of O-RU’s information elements

3.4.1: Fault Management

List of Test cases

1. O-RU Alarm Notification Generation

2. Retrieval of Active Alarm List

For both the test cases 1 and 2 and test case in Manage Alarm Request on page 62.

Steps to execute

1. Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “fm_call_flow.xml” as listed in Table 1. There is no input json file for this test case.
2. Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that M-Plane server is running in O-RU.

Result

1. Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.
2. Pass Logs for the execution is generated ‘\log’ folder.

8.2 Manage Alarms Request

For the test case 8.2,

Steps to execute

1. Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “8.2_manage_alarm_request.xml” as listed in Table 1. There is no input json file for this test case.
2. Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that M-Plane server is running in O-RU.

Result

1. Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.
2. Pass Logs for the execution is generated '\log' folder.

Section 3.5: O-RU Software Update

List of Test cases

1. O-RU Software Update and Install (positive case)

Steps to execute

1. Fill the client config file (shown in Figure 30) under "du/profile/v1/" with call-flow-xml value as "swm_management.xml" as listed in Table 1. The corresponding json file is swm-management-location. Input values to "product-code", "vendor-code", "build-id", "build-name", "build-version", "remote-file-path" and "password" in the json file before execution. "remote-file-path" shall be of the format: *sftp://username@hostname:22/path-to-file*.
2. Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that M-Plane server is running in O-RU.

Result

1. Successful rpc exchange of software download for multiple files and install message from client and reply message from server is seen in the Windows command terminal.
2. Ensure that "status" of the installed slot is valid and other parameters are unchanged.
3. Pass Logs for the execution is generated '..\pkg_tmp\Keysight_5G_ORAN_M-PLANE_Client\log' folder.

2. O-RU Software Update (negative case)

Steps to execute

1. Fill the client config file (shown in Figure 30) under "du/profile/v1/" with call-flow-xml value as "swm_management_negative.xml" as listed in Table 1. The corresponding json file is swm-management-location. Input values to "product-code", "vendor-code", "build-id", "build-name", "build-version", "remote-file-path" and "password" in the json file before execution. "remote-file-path" shall be of the format:

sftp://username@hostname:22/path-to-non-existent-file or a file containing integrity error.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Rpc exchanges of successful download and failed install message from client and reply message from server is seen in the Windows command terminal.

2 Ensure that “status” of the installed slot is invalid while other parameters are unchanged.

3 Pass Logs for the execution is generated ‘..\pkg_tmp\Keysight_5G_ORAN_M-PLANE_Client\log’ folder.

Section 3.6: O-RU Software Activation

List of Test cases

1. Software Activation without Reset

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “swm_management_negative.xml” as listed in Table 1. The corresponding json file is swm-management-location. Input values to “product-code”, “vendor-code”, “build-id”, “build-name”, “build-version”, “remote-file-path” and “password” in the json file before execution. “remote-file-path” shall be of the format: *sftp://username@hostname:22/path-to-non-existent-file* or a file containing integrity error.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Rpc exchanges of software activation message from client and reply message from server is seen in the Windows command terminal.

2 Ensure the parameters of currently activated slot without reset has “status” as valid, “running” as false and “active” as true whereas the slot that is still using software has “status” as valid, “active” as false and “running” as true.

7.1.2 Supplemental Reset after Software Activation

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as listed in Table 1.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Rpc exchanges of reset message from client and reply message from server is seen in the windows command terminal.
2 Ensure the slot which is currently activated has “status” as valid, “active” as true and “running” as true.
3 Since our RU server doesn’t support after reset check, it would have returned corresponding error message.

Section 3.7: Access Control

List of Test cases

1. Sudo on Hybrid M-plane Architecture (positive case)

Steps to execute

1 Fill the client config file (shown in Figure 30) under "du/profile/v1/" with call-flow-xml value as "user_mgmt.xml" as listed in Table 1.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges of reset message from client and reply message from server is seen in the windows command terminal.

2 Pass Logs for the execution is generated '..\pkg_tmp\Keysight_5G_ORAN_M-PLANE_Client\log' folder.

2. Access Control Sudo (negative case)

Description of Test Case

1 The existing user_mgmt call flow xml is used with sudo user.

2 The user_mgmt call flow xml is initially configured with default password.

3 Change the username password in the test case to see if it works.

Steps to execute

1 Fill the client config file (shown in Figure 30) under "du/profile/v1/" with call-flow-xml value as "user_mgmt.xml" as listed in Table 1.

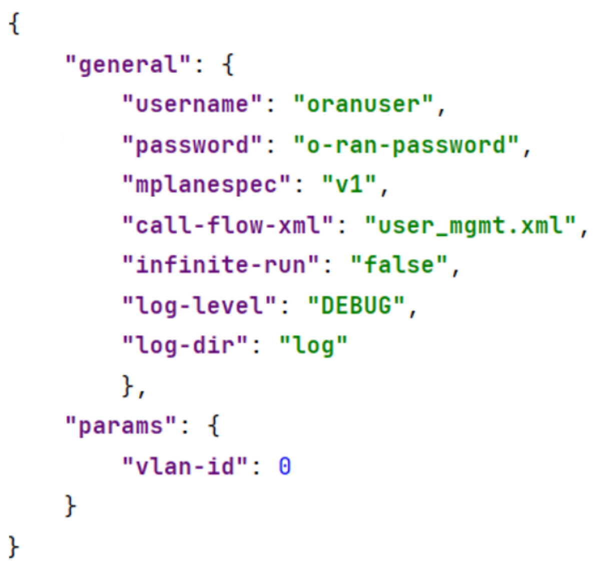
2 Start the dhcp server, M-Plane client and M-Plane server. See Executing and starting M-Plane client and server on page 57.

3 Stop the M-Plane server.

Result

1 Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.

2 Logs for the execution is generated ‘..\pkg_tmp\Keysight_5G_ORAN_M-PLANE_Client\log’ folder.



```
{
    "general": {
        "username": "oranuser",
        "password": "o-ran-password",
        "mplanespec": "v1",
        "call-flow-xml": "user_mgmt.xml",
        "infinite-run": "false",
        "log-level": "DEBUG",
        "log-dir": "log"
        },
    "params": {
        "vlan-id": 0
    }
}
```

Figure 30 Client config file updated for user management

3. Access Control NMS (negative case)

Description of Test Case

1 The nms_negative call flow xml is used with nms user and password.

2 When trying to use nms user and password, other than sudo user and password, the access denied message should be thrown from server side shown in Figure 37.

Steps to execute

1 Fill the client config file (shown in Figure 31) under “du/profile/v1/” with call-flow-xml value as “user_mgmt.xml” as listed in Table 1.

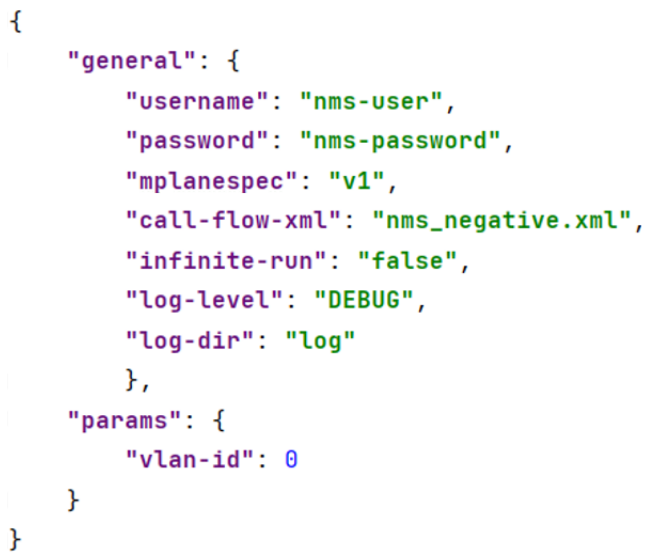
2 Start the dhcp server, M-Plane client and M-Plane server. See Executing and starting M-Plane client and server on page 57.

3 Stop the M-Plane server.

Result

1 Expected rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.

2 Logs for the execution is generated '..\pkg_tmp\Keysight_5G_ORAN_M-PLANE_Client\log' folder.



```
{
    "general": {
        "username": "nms-user",
        "password": "nms-password",
        "mplanespec": "v1",
        "call-flow-xml": "nms_negative.xml",
        "infinite-run": "false",
        "log-level": "DEBUG",
        "log-dir": "log"
        },
    "params": {
        "vlan-id": 0
    }
}
```

Figure 31 Client config file updated for user management xml

4. Access Control FM-PM (negative case)

Description of Test Case

1 The fmpm_negative call flow xml is used with fmpm user and password.

2 When trying to use nms user and password, other than sudo user and password, the access denied message should be thrown from server side similar to Figure 3.4.8.3.2.

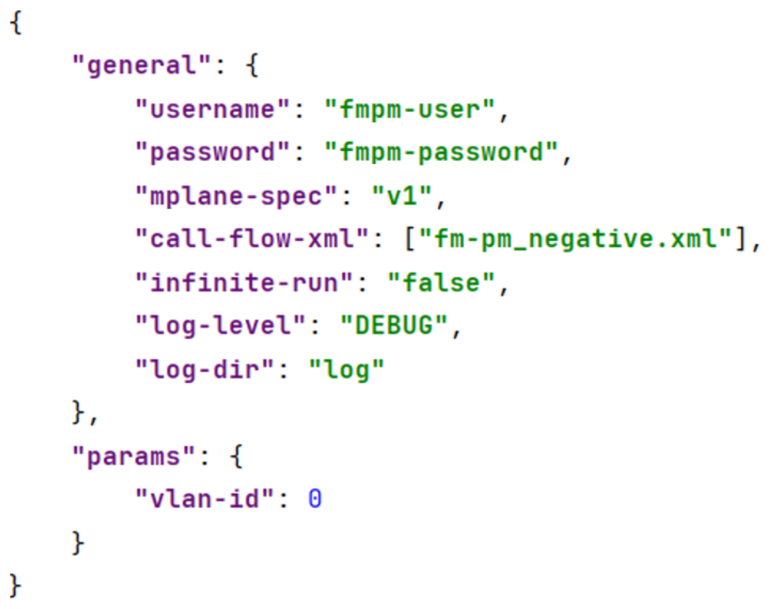
3 When trying to get read access to o-ran-processing-elements should also throw error in server side.

Steps to execute

1. Fill the client config file (shown in Figure 32) under “du/profile/v1/” with call-flow-xml value as “user_mgmt.xml” as listed in Table 1.
2. Start the dhcp server, M-Plane client and M-Plane server. See Executing and starting M-Plane client and server on page 57.
3. Stop the M-Plane server.

Result

1. Expected rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.
2. Logs for the execution is generated ‘..\pkg_tmp\ Keysight_5G_ORAN_M-PLANE_Client\log’ folder.



```
{
    "general": {
        "username": "fmpm-user",
        "password": "fmpm-password",
        "mplane-spec": "v1",
        "call-flow-xml": ["fm-pm_negative.xml"],
        "infinite-run": "false",
        "log-level": "DEBUG",
        "log-dir": "log"
    },
    "params": {
        "vlan-id": 0
    }
}
```

Figure 32 Client config file updated for fmpm xml

5. Access Control SWM (negative case)

Description of Test Case

1 The swm_negative call flow xml is used with swm user and password.

2 When trying to use swm user and password, other than sudo user and password, the access denied message should be thrown from server side similar to Figure 3.4.8.3.2.

3 When trying to get read access to o-ran-sync should also throw error in server side.

Steps to execute

1 Fill the client config file (shown in Figure 33) under “du/profile/v1/” with call-flow-xml value as “user_mgmt.xml” as listed in Table 1.

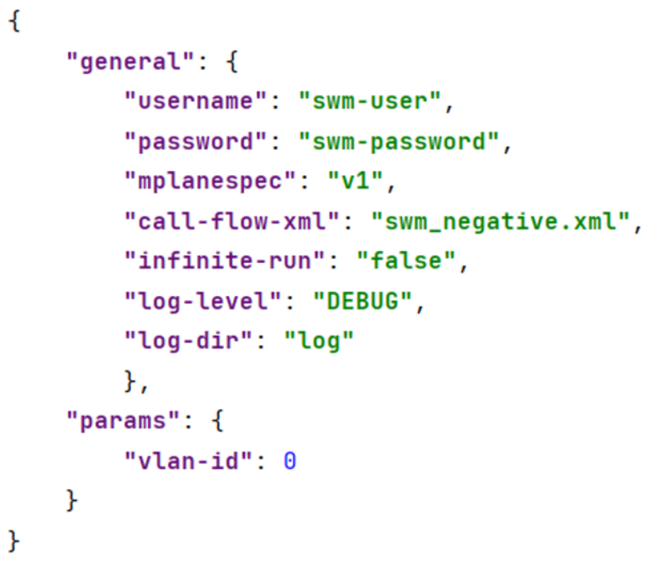
2 Start the dhcp server, M-Plane client and M-Plane server. See Executing and starting M-Plane client and server on page 57.

3 Stop the M-Plane server.

Result

1 Expected rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.

2 Logs for the execution is generated ‘..\pkg_tmp\Keysight_5G_ORAN_M-PLANE_Client\log’ folder.



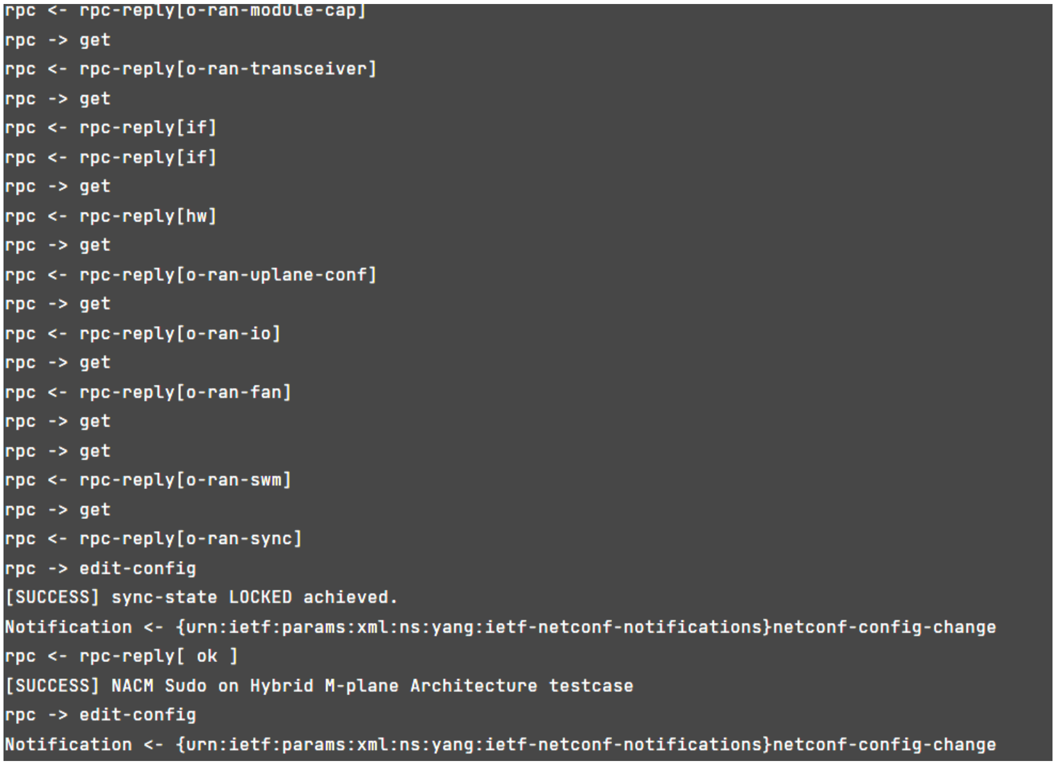
```
{
    "general": {
        "username": "swm-user",
        "password": "swm-password",
        "mplanespec": "v1",
        "call-flow-xml": "swm_negative.xml",
        "infinite-run": "false",
        "log-level": "DEBUG",
        "log-dir": "log"
        },
    "params": {
        "vlan-id": 0
    }
}
```

Figure 33 Client config file updated for swm xml

6. Sudo on Hierarchial M-plane Architecture (positive case)

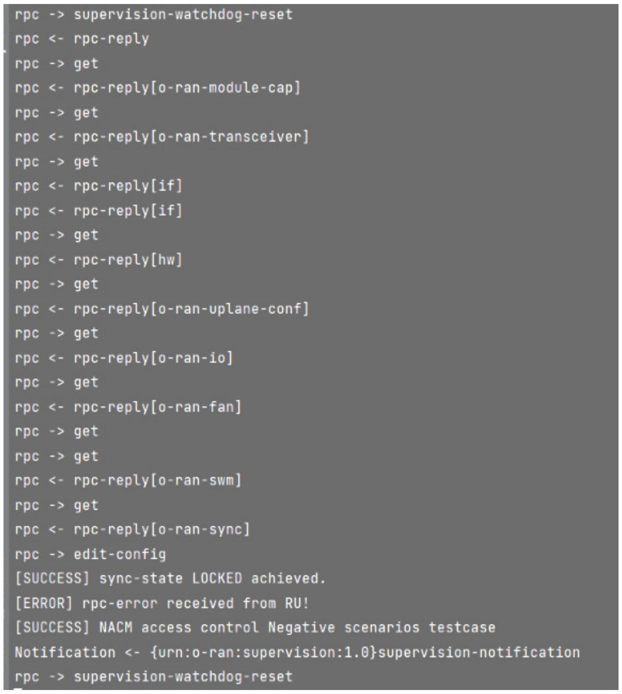
3.7.1: Snapshot of logs in M-plane client

The following images show the snapshot of user provisioning logs in the M-plane client:



```
rpc <- rpc-reply[o-ran-module-cap]
rpc -> get
rpc <- rpc-reply[o-ran-transceiver]
rpc -> get
rpc <- rpc-reply[if]
rpc <- rpc-reply[if]
rpc -> get
rpc <- rpc-reply[hw]
rpc -> get
rpc <- rpc-reply[o-ran-uplane-conf]
rpc -> get
rpc <- rpc-reply[o-ran-io]
rpc -> get
rpc <- rpc-reply[o-ran-fan]
rpc -> get
rpc -> get
rpc <- rpc-reply[o-ran-swm]
rpc -> get
rpc <- rpc-reply[o-ran-sync]
rpc -> edit-config
[SUCCESS] sync-state LOCKED achieved.
Notification <- {urn:ietf:params:xml:ns:yang:ietf-netconf-notifications}netconf-config-change
rpc <- rpc-reply[ ok ]
[SUCCESS] NACM Sudo on Hybrid M-plane Architecture testcase
rpc -> edit-config
Notification <- {urn:ietf:params:xml:ns:yang:ietf-netconf-notifications}netconf-config-change
```

Figure 34 M-plane Client log (Positive)

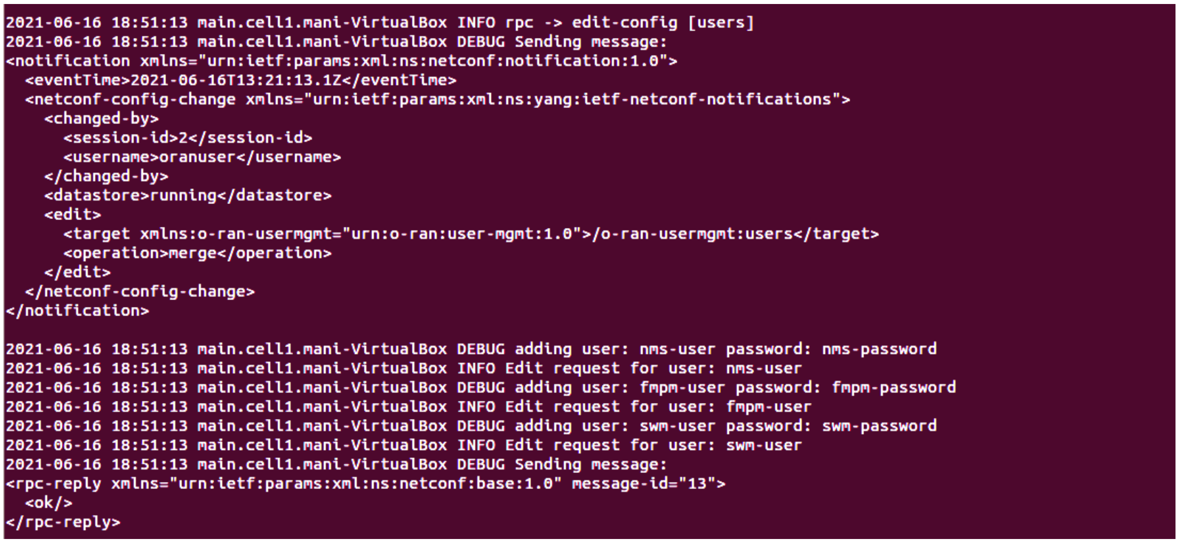


```
rpc -> supervision-watchdog-reset
rpc <- rpc-reply
rpc -> get
rpc <- rpc-reply[o-ran-module-cap]
rpc -> get
rpc <- rpc-reply[o-ran-transceiver]
rpc -> get
rpc <- rpc-reply[if]
rpc <- rpc-reply[if]
rpc -> get
rpc <- rpc-reply[hw]
rpc -> get
rpc <- rpc-reply[o-ran-uplane-conf]
rpc -> get
rpc <- rpc-reply[o-ran-io]
rpc -> get
rpc <- rpc-reply[o-ran-fan]
rpc -> get
rpc -> get
rpc <- rpc-reply[o-ran-swm]
rpc -> get
rpc <- rpc-reply[o-ran-sync]
rpc -> edit-config
[SUCCESS] sync-state LOCKED achieved.
[ERROR] rpc-error received from RU!
[SUCCESS] NACM access control Negative scenarios testcase
Notification <- {urn:o-ran:supervision:1.0}supervision-notification
rpc -> supervision-watchdog-reset
```

Figure 35 M-plane Client log (Negative)

3.7.2: O-RU Server side (anti) log

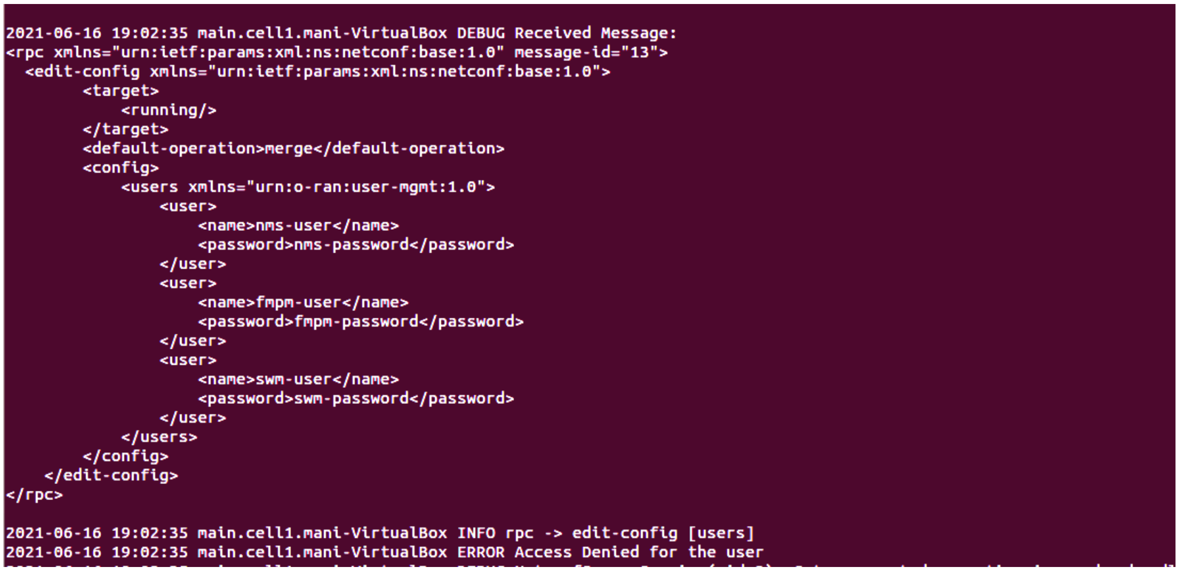
The following images show the snapshot of user provisioning logs in the M-plane client:



```
2021-06-16 18:51:13 main.cell1.mani-VirtualBox INFO rpc -> edit-config [users]
2021-06-16 18:51:13 main.cell1.mani-VirtualBox DEBUG Sending message:
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2021-06-16T13:21:13.1Z</eventTime>
  <netconf-config-change xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-notifications">
    <changed-by>
      <session-id>2</session-id>
      <username>oranuser</username>
    </changed-by>
    <datastore>running</datastore>
    <edit>
      <target xmlns:o-ran-usermgmt="urn:o-ran:user-mgmt:1.0">/o-ran-usermgmt:users</target>
      <operation>merge</operation>
    </edit>
  </netconf-config-change>
</notification>

2021-06-16 18:51:13 main.cell1.mani-VirtualBox DEBUG adding user: nms-user password: nms-password
2021-06-16 18:51:13 main.cell1.mani-VirtualBox INFO Edit request for user: nms-user
2021-06-16 18:51:13 main.cell1.mani-VirtualBox DEBUG adding user: fmpm-user password: fmpm-password
2021-06-16 18:51:13 main.cell1.mani-VirtualBox INFO Edit request for user: fmpm-user
2021-06-16 18:51:13 main.cell1.mani-VirtualBox DEBUG adding user: swm-user password: swm-password
2021-06-16 18:51:13 main.cell1.mani-VirtualBox INFO Edit request for user: swm-user
2021-06-16 18:51:13 main.cell1.mani-VirtualBox DEBUG Sending message:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="13">
  <ok/>
</rpc-reply>
```

Figure 36 User provisioning logs at O-RU side (Positive)



```
2021-06-16 19:02:35 main.cell1.mani-VirtualBox DEBUG Received Message:
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="13">
  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <target>
            <running/>
        </target>
        <default-operation>merge</default-operation>
        <config>
            <users xmlns="urn:o-ran:user-mgmt:1.0">
                <user>
                    <name>nms-user</name>
                    <password>nms-password</password>
                </user>
                <user>
                    <name>fmpm-user</name>
                    <password>fmpm-password</password>
                </user>
                <user>
                    <name>swm-user</name>
                    <password>swm-password</password>
                </user>
            </users>
        </config>
    </edit-config>
</rpc>

2021-06-16 19:02:35 main.cell1.mani-VirtualBox INFO rpc -> edit-config [users]
2021-06-16 19:02:35 main.cell1.mani-VirtualBox ERROR Access Denied for the user
```

Figure 37 User provisioning logs at O-RU side (Negative)

Section 3.8: External Input / Output Ports

3.8.1: External Input Port State Detection

3.8.2: External Output Port State Control

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with corresponding xml values as listed in Table 1. The corresponding json file is o-ran-io.json. Input the values to “input” and “output-settings” in external-io.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.

2 Pass Logs for the execution is generated in '\log' folder.

Section 3.9: O-RU Configurability

The M-Plane software supports various options with <edit-config /> to support configuring the O-RU.

See the following sections for examples:

- Create new VLAN on page 39
- LBM Configuration on page 40
- Configure mplane-info on page 41
- Add Processing Element on page 42
- Endpoint Carrier Configuration on page 44
- Endpoint Carrier Link Configuration on page 44
- Activate Carrier on page 45

Section 3.10: ALD Communications

ALD App Server: It is required for Netconf client RPCs to carry HDLC x` from Mplane client side. HDLC link needs to be first established by scanning HDLC bus, assigning address to the ALD, establishing link, send Antenna control messages, finally disconnect HDLC connection. ALD App is used to abstracting the HDLC/ALD related logic outside M-plane client. Mplane client and ALDApp interact over GRPC. The ald proto file was used for this purpose. Following scripts are from the ald prototype (ald.proto) file and the sample python files used:

Script from the ald.proto file:

```
syntax = "proto3";
package mplane;

service ALDService {
    rpc getALDPortDCControl(ALDPorts) returns (ALDPortDCControl);
    rpc requestALDPayload(ALDPorts) returns (ALDReq);
    rpc verifyALDResponse(ALDResp) returns (Verdict);
}

enum Connector {
    ANTENNA_CONNECTOR=0;
    RS485=1;
}

message ALDPort {
    string name=1;
    uint32 port=2;
    bool dc_control_support=3;
    bool dc_enabled_status=4;
    Connector supported_connector=5;
}

message ALDPorts {
    bool over_current_supported=1;
    repeated ALDPort ports=2;
}
```

```
message ALDPortDCControl{
    string name=1;
    bool dc_enabled=2;
}

message ALDReq {
    uint32 port=1;
    bytes msg=2;
}

message ALDResp {
    uint32 port=1;
    bytes msg=2;
}

message Verdict {
    bool result=1;
    bool ald_continue=2;
    ALDReq ald_req=3;
}
```

Script from the ald_server.py file:

```
import ald_pb2_grpc
from concurrent import futures
import logging
import ald_pb2
import grpc
i = 0
class ALDServicerImpl(ald_pb2_grpc.ALDServiceServicer):
    def __init__(self):
        self.port = 1
        self.name = "PORT1"
        ## below is the 3 message sent in hdlc format
        self.msg = ["12 00 81 80 18 05 04 00 05 34 30 06 04 00 03 6B 00
07 01 07 08 01 03", "12 00 81 80 18 05 04 00 05 34 30 06 04 00 03 6B 00
07 01 07 08 01 02", "12 00 81 80 18 05 04 00 05 34 30 06 04 00 03 6B 00
07 01 07 08 01 01"]
        self.status = 1
        self.ald_count = 0



    def getALDPortDCControl(self, request, context):
        print("received request for payload")
        for port in request.ports:
            print("port name {} id {} dc_control_status {}
dc_enabled_status {}".format(port.name, port.port,
port.dc_control_support, port.dc_enabled_status))
            return ald_pb2.ALDPortDCControl(name=self.name,
dc_enabled=True)


    def requestALDPayload(self, request, context):
        print("received request for payload : over-current-supported
{}".format(request.over_current_supported))
        for port in request.ports:
            print("port name {} id {} dc_control_status {}
dc_enabled_status {}".format(port.name, port.port,
port.dc_control_support, port.dc_enabled_status))
        return ald_pb2.ALDReq(port=self.port, msg=self.msg[i])


    def verifyALDResponse(self, request, context):
        verdict = ald_pb2.Verdict()
        if self.port == request.port and self.msg[i] == request.msg:
            print("port and msg match with requested value")
            if i < 3:
                i = i + 1
            else:
                i = 0
            verdict.result = True
            self.ald_count = self.ald_count + 1
            if self.ald_count < 3:
                verdict.ald_continue = True
                verdict.ald_req.port = self.port
                verdict.ald_req.msg = self.msg[i]
            else:
                verdict.ald_continue = False
        else:
            print("port and/or msg does not match")
            verdict.result = False
            verdict.ald_continue = False
        return verdict

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=1))
     ald_pb2_grpc.add_ALDServiceServicer_to_server(ALDServicerImpl(),
server)
    server.add_insecure_port('192.168.56.6:50051')
    server.start()
    server.wait_for_termination()


if __name__ == '__main__':
    logging.basicConfig()
    serve()
```

Script from the ald_pb2.py file:

```
# -*- coding: utf-8 -*-
# Generated by the protocol buffer compiler.  DO NOT EDIT!
# source: ald.proto
"""Generated protocol buffer code."""
from google.protobuf.internal import enum_type_wrapper
from google.protobuf import descriptor as _descriptor
from google.protobuf import message as _message
from google.protobuf import reflection as _reflection
from google.protobuf import symbol_database as _symbol_database
# @@protoc_insertion_point(imports)

_sym_db = _symbol_database.Default()
```

```
DESCRIPTOR = _descriptor.FileDescriptor(
  name='ald.proto',
  package='mplane',
  syntax='proto3',
  serialized_options=None,
  create_key=_descriptor._internal_create_key,
   serialized_pb=b'\n\tald.proto\x12\x06mplane\"\x8c\x01\n\x07\
x41LDPort\x12\x0c\n\x04name\x18\x01 \x01(\t\x12\x0c\n\x04port\x18\x02
\x01(\r\x12\x1a\n\x12\x64\x63_control_support\x18\x03 \x01(\x08\x12\
x19\n\x11\x64\x63_enabled_status\x18\x04 \x01(\x08\x12.\n\
x13supported_connector\x18\x05 \x01(\x0e\x32\x11.mplane.Connector\"J\
n\x08\x41LDPorts\x12\x1e\n\x16over_current_supported\x18\x01 \x01(\
x08\x12\x1e\n\x05ports\x18\x02 \x03(\x0b\x32\x0f.mplane.ALDPort\"4\n\
x10\x41LDPortDCControl\x12\x0c\n\x04name\x18\x01 \x01(\t\x12\x12\n\
ndc_enabled\x18\x02 \x01(\x08\"#\n\x06\x41LDReq\x12\x0c\n\x04port\
x18\x01 \x01(\r\x12\x0b\n\x03msg\x18\x02 \x01(\x0c\"$\n\x07\x41LDResp\
x12\x0c\n\x04port\x18\x01 \x01(\r\x12\x0b\n\x03msg\x18\x02 \x01(\x0c\
"P\n\x07Verdict\x12\x0e\n\x06result\x18\x01 \x01(\x08\x12\x14\n\x0c\
x61ld_continue\x18\x02 \x01(\x08\x12\x1f\n\x07\x61ld_req\x18\x03 \
x01(\x0b\x32\x0e.mplane.ALDReq*-\n\tConnector\x12\x15\n\x11\
x41NTENNA_CONNECTOR\x10\x00\x12\t\n\x05RS485\x10\x01\x32\xbd\x01\n\
nALDService\x12\x41\n\x13getALDPortDCControl\x12\x10.mplane.ALDPorts\
x1a\x18.mplane.ALDPortDCControl\x12\x35\n\x11requestALDPayload\x12\
x10.mplane.ALDPorts\x1a\x0e.mplane.ALDReq\x12\x35\n\
x11verifyALDResponse\x12\x0f.mplane.ALDResp\x1a\x0f.mplane.Verdictb\
x06proto3'
)


_CONNECTOR = _descriptor.EnumDescriptor(
  name='Connector',
  full_name='mplane.Connector',
  filename=None,
  file=DESCRIPTOR,
  create_key=_descriptor._internal_create_key,
  values=[
    _descriptor.EnumValueDescriptor(
      name='ANTENNA_CONNECTOR', index=0, number=0,
      serialized_options=None,
      type=None,
      create_key=_descriptor._internal_create_key),
    _descriptor.EnumValueDescriptor(
      name='RS485', index=1, number=1,
      serialized_options=None,
      type=None,
      create_key=_descriptor._internal_create_key),
  ],
  containing_type=None,
  serialized_options=None,
  serialized_start=451,
  serialized_end=496,
)
_sym_db.RegisterEnumDescriptor(_CONNECTOR)

Connector = enum_type_wrapper.EnumTypeWrapper(_CONNECTOR)
ANTENNA_CONNECTOR = 0
RS485 = 1




_ALDPORT = _descriptor.Descriptor(
  name='ALDPort',
  full_name='mplane.ALDPort',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
    _descriptor.FieldDescriptor(
      name='name', full_name='mplane.ALDPort.name', index=0,
      number=1, type=9, cpp_type=9, label=1,
      has_default_value=False, default_value=b"".decode('utf-8'),
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
      name='port', full_name='mplane.ALDPort.port', index=1,
      number=2, type=13, cpp_type=3, label=1,
      has_default_value=False, default_value=0,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
                                      name='dc_control_support',
full_name='mplane.ALDPort.dc_control_support', index=2,
      number=3, type=8, cpp_type=7, label=1,
      has_default_value=False, default_value=False,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
                                       name='dc_enabled_status',
full_name='mplane.ALDPort.dc_enabled_status', index=3,
      number=4, type=8, cpp_type=7, label=1,
      has_default_value=False, default_value=False,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
                                     name='supported_connector',
full_name='mplane.ALDPort.supported_connector', index=4,
      number=5, type=14, cpp_type=8, label=1,
      has_default_value=False, default_value=0,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
  ],
  extensions=[
  ],
  nested_types=[],
  enum_types=[
  ],
  serialized_options=None,
  is_extendable=False,
  syntax='proto3',
  extension_ranges=[],
  oneofs=[
  ],
  serialized_start=22,
  serialized_end=162,
)


_ALDPORTS = _descriptor.Descriptor(
  name='ALDPorts',
  full_name='mplane.ALDPorts',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
    _descriptor.FieldDescriptor(
                                  name='over_current_supported',
full_name='mplane.ALDPorts.over_current_supported', index=0,
      number=1, type=8, cpp_type=7, label=1,
      has_default_value=False, default_value=False,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
      name='ports', full_name='mplane.ALDPorts.ports', index=1,
      number=2, type=11, cpp_type=10, label=3,
      has_default_value=False, default_value=[],
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
  ],
  extensions=[
  ],
  nested_types=[],
  enum_types=[
  ],
  serialized_options=None,
  is_extendable=False,
  syntax='proto3',
  extension_ranges=[],
  oneofs=[
  ],
  serialized_start=164,
  serialized_end=238,
)


_ALDPORTDCCONTROL = _descriptor.Descriptor(
  name='ALDPortDCControl',
  full_name='mplane.ALDPortDCControl',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
    _descriptor.FieldDescriptor(
      name='name', full_name='mplane.ALDPortDCControl.name', index=0,
      number=1, type=9, cpp_type=9, label=1,
      has_default_value=False, default_value=b"".decode('utf-8'),
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
                                              name='dc_enabled',
full_name='mplane.ALDPortDCControl.dc_enabled', index=1,
      number=2, type=8, cpp_type=7, label=1,
      has_default_value=False, default_value=False,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,  file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
  ],
  extensions=[
  ],
  nested_types=[],
  enum_types=[
  ],
  serialized_options=None,
  is_extendable=False,
  syntax='proto3',
  extension_ranges=[],
  oneofs=[
  ],
  serialized_start=240,
  serialized_end=292,
)


_ALDREQ = _descriptor.Descriptor(
  name='ALDReq',
  full_name='mplane.ALDReq',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
    _descriptor.FieldDescriptor(
      name='port', full_name='mplane.ALDReq.port', index=0,
      number=1, type=13, cpp_type=3, label=1,
      has_default_value=False, default_value=0,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,  file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
      name='msg', full_name='mplane.ALDReq.msg', index=1,
      number=2, type=12, cpp_type=9, label=1,
      has_default_value=False, default_value=b"",
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
  ],
  extensions=[
  ],
  nested_types=[],
  enum_types=[
  ],
  serialized_options=None,
  is_extendable=False,
  syntax='proto3',
  extension_ranges=[],
  oneofs=[
  ],
  serialized_start=294,
  serialized_end=329,
)


_ALDRESP = _descriptor.Descriptor(
  name='ALDResp',
  full_name='mplane.ALDResp',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
    _descriptor.FieldDescriptor(
      name='port', full_name='mplane.ALDResp.port', index=0,
      number=1, type=13, cpp_type=3, label=1,
      has_default_value=False, default_value=0,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
      name='msg', full_name='mplane.ALDResp.msg', index=1,
      number=2, type=12, cpp_type=9, label=1,
      has_default_value=False, default_value=b"",
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
                  serialized_options=None,   file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
  ],
  extensions=[
  ],
  nested_types=[],
  enum_types=[
  ],
  serialized_options=None,
  is_extendable=False,
  syntax='proto3',
  extension_ranges=[],
  oneofs=[
  ],
  serialized_start=331,
  serialized_end=367,
)


_VERDICT = _descriptor.Descriptor(
  name='Verdict',
  full_name='mplane.Verdict',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
    _descriptor.FieldDescriptor(
      name='result', full_name='mplane.Verdict.result', index=0,
      number=1, type=8, cpp_type=7, label=1,
      has_default_value=False, default_value=False,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,  file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
       name='ald_continue', full_name='mplane.Verdict.ald_continue',
index=1,
      number=2, type=8, cpp_type=7, label=1,
      has_default_value=False, default_value=False,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,  file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
    _descriptor.FieldDescriptor(
      name='ald_req', full_name='mplane.Verdict.ald_req', index=2,
      number=3, type=11, cpp_type=10, label=1,
      has_default_value=False, default_value=None,
      message_type=None, enum_type=None, containing_type=None,
      is_extension=False, extension_scope=None,
            serialized_options=None,  file=DESCRIPTOR,
create_key=_descriptor._internal_create_key),
  ],
  extensions=[
  ],
  nested_types=[],
  enum_types=[
  ],
  serialized_options=None,
  is_extendable=False,
  syntax='proto3',
  extension_ranges=[],
  oneofs=[
  ],
  serialized_start=369,
  serialized_end=449,
)

_ALDPORT.fields_by_name['supported_connector'].enum_type = _CONNECTOR
_ALDPORTS.fields_by_name['ports'].message_type = _ALDPORT
_VERDICT.fields_by_name['ald_req'].message_type = _ALDREQ
DESCRIPTOR.message_types_by_name['ALDPort'] = _ALDPORT
DESCRIPTOR.message_types_by_name['ALDPorts'] = _ALDPORTS
DESCRIPTOR.message_types_by_name['ALDPortDCControl'] =
_ALDPORTDCCONTROL
DESCRIPTOR.message_types_by_name['ALDReq'] = _ALDREQ
DESCRIPTOR.message_types_by_name['ALDResp'] = _ALDRESP
DESCRIPTOR.message_types_by_name['Verdict'] = _VERDICT
DESCRIPTOR.enum_types_by_name['Connector'] = _CONNECTOR
_sym_db.RegisterFileDescriptor(DESCRIPTOR)

ALDPort = _reflection.GeneratedProtocolMessageType('ALDPort',
(_message.Message,), {
  'DESCRIPTOR' : _ALDPORT,
  '__module__' : 'ald_pb2'
  # @@protoc_insertion_point(class_scope:mplane.ALDPort)
  })
_sym_db.RegisterMessage(ALDPort)

ALDPorts = _reflection.GeneratedProtocolMessageType('ALDPorts',
(_message.Message,), {
  'DESCRIPTOR' : _ALDPORTS,
  '__module__' : 'ald_pb2'
  # @@protoc_insertion_point(class_scope:mplane.ALDPorts)
  })
_sym_db.RegisterMessage(ALDPorts)

ALDPortDCControl =
_reflection.GeneratedProtocolMessageType('ALDPortDCControl',
(_message.Message,), {
  'DESCRIPTOR' : _ALDPORTDCCONTROL,
  '__module__' : 'ald_pb2'
  # @@protoc_insertion_point(class_scope:mplane.ALDPortDCControl)
  })
```

```
_sym_db.RegisterMessage(ALDPortDCControl)

ALDReq = _reflection.GeneratedProtocolMessageType('ALDReq',
(_message.Message,), {
  'DESCRIPTOR' : _ALDREQ,
  '__module__' : 'ald_pb2'
  # @@protoc_insertion_point(class_scope:mplane.ALDReq)
  })
_sym_db.RegisterMessage(ALDReq)

ALDResp = _reflection.GeneratedProtocolMessageType('ALDResp',
(_message.Message,), {
  'DESCRIPTOR' : _ALDRESP,
  '__module__' : 'ald_pb2'
  # @@protoc_insertion_point(class_scope:mplane.ALDResp)
  })
_sym_db.RegisterMessage(ALDResp)

Verdict = _reflection.GeneratedProtocolMessageType('Verdict',
(_message.Message,), {
  'DESCRIPTOR' : _VERDICT,
  '__module__' : 'ald_pb2'
  # @@protoc_insertion_point(class_scope:mplane.Verdict)
  })
_sym_db.RegisterMessage(Verdict)



_ALDSERVICE = _descriptor.ServiceDescriptor(
  name='ALDService',
  full_name='mplane.ALDService',
  file=DESCRIPTOR,
  index=0,
  serialized_options=None,
  create_key=_descriptor._internal_create_key,
  serialized_start=499,
  serialized_end=688,
  methods=[
  _descriptor.MethodDescriptor(
    name='getALDPortDCControl',
    full_name='mplane.ALDService.getALDPortDCControl',
    index=0,
    containing_service=None,
    input_type=_ALDPORTS,
    output_type=_ALDPORTDCCONTROL,
    serialized_options=None,
    create_key=_descriptor._internal_create_key,
  ),
  _descriptor.MethodDescriptor(
    name='requestALDPayload',
    full_name='mplane.ALDService.requestALDPayload',
    index=1,
    containing_service=None,
    input_type=_ALDPORTS,
    output_type=_ALDREQ,
    serialized_options=None,
    create_key=_descriptor._internal_create_key,
  ),
  _descriptor.MethodDescriptor(
    name='verifyALDResponse',
    full_name='mplane.ALDService.verifyALDResponse',
    index=2,
    containing_service=None,
    input_type=_ALDRESP,
    output_type=_VERDICT,
    serialized_options=None,
    create_key=_descriptor._internal_create_key,
  ),
])
_sym_db.RegisterServiceDescriptor(_ALDSERVICE)

DESCRIPTOR.services_by_name['ALDService'] = _ALDSERVICE

# @@protoc_insertion_point(module_scope)
```

Script from the ald_pb2_grpc.py file:

```
# Generated by the gRPC Python protocol compiler plugin. DO NOT EDIT!
"""Client and server classes corresponding to protobuf-defined
services."""
import grpc

import ald_pb2 as ald__pb2



class ALDServiceStub(object):
    """Missing associated documentation comment in .proto file."""

    def __init__(self, channel):
        """Constructor.

        Args:
            channel: A grpc.Channel.
        """
        self.getALDPortDCControl = channel.unary_unary(
                '/mplane.ALDService/getALDPortDCControl',
               request_serializer=ald__pb2.ALDPorts.SerializeToString,

response_deserializer=ald__pb2.ALDPortDCControl.FromString,
                )
        self.requestALDPayload = channel.unary_unary(
                '/mplane.ALDService/requestALDPayload',
               request_serializer=ald__pb2.ALDPorts.SerializeToString,
                response_deserializer=ald__pb2.ALDReq.FromString,
                )
        self.verifyALDResponse = channel.unary_unary(
                '/mplane.ALDService/verifyALDResponse',
                request_serializer=ald__pb2.ALDResp.SerializeToString,
                response_deserializer=ald__pb2.Verdict.FromString,
                )


class ALDServiceServicer(object):
    """Missing associated documentation comment in .proto file."""

    def getALDPortDCControl(self, request, context):
        """Missing associated documentation comment in .proto file."""
        context.set_code(grpc.StatusCode.UNIMPLEMENTED)
        context.set_details('Method not implemented!')
        raise NotImplementedError('Method not implemented!')

    def requestALDPayload(self, request, context):
        """Missing associated documentation comment in .proto file."""
        context.set_code(grpc.StatusCode.UNIMPLEMENTED)
        context.set_details('Method not implemented!')
        raise NotImplementedError('Method not implemented!')

    def verifyALDResponse(self, request, context):
        """Missing associated documentation comment in .proto file."""
        context.set_code(grpc.StatusCode.UNIMPLEMENTED)
        context.set_details('Method not implemented!')
        raise NotImplementedError('Method not implemented!')


def add_ALDServiceServicer_to_server(servicer, server):
    rpc_method_handlers = {
           'getALDPortDCControl': grpc.unary_unary_rpc_method_handler(
                    servicer.getALDPortDCControl,
                    request_deserializer=ald__pb2.ALDPorts.FromString,

response_serializer=ald__pb2.ALDPortDCControl.SerializeToString,
            ),
            'requestALDPayload': grpc.unary_unary_rpc_method_handler(
                    servicer.requestALDPayload,
                    request_deserializer=ald__pb2.ALDPorts.FromString,

response_serializer=ald__pb2.ALDReq.SerializeToString,
            ),
            'verifyALDResponse': grpc.unary_unary_rpc_method_handler(
                    servicer.verifyALDResponse,
                    request_deserializer=ald__pb2.ALDResp.FromString,

response_serializer=ald__pb2.Verdict.SerializeToString,
            ),
    }
    generic_handler = grpc.method_handlers_generic_handler(
            'mplane.ALDService', rpc_method_handlers)
    server.add_generic_rpc_handlers((generic_handler,))


 # This class is part of an EXPERIMENTAL API.
class ALDService(object):
    """Missing associated documentation comment in .proto file."""

    @staticmethod
    def getALDPortDCControl(request,
            target,
            options=(),
            channel_credentials=None,
            call_credentials=None,
            insecure=False,
            compression=None,
            wait_for_ready=None,
            timeout=None,
            metadata=None):
          return grpc.experimental.unary_unary(request, target,
'/mplane.ALDService/getALDPortDCControl',
            ald__pb2.ALDPorts.SerializeToString,
            ald__pb2.ALDPortDCControl.FromString,
            options, channel_credentials,
             insecure, call_credentials, compression, wait_for_ready,
timeout, metadata)

    @staticmethod
    def requestALDPayload(request,
            target,
            options=(),
            channel_credentials=None,
            call_credentials=None,
            insecure=False,
            compression=None,
            wait_for_ready=None,
            timeout=None,
            metadata=None):
          return grpc.experimental.unary_unary(request, target,
'/mplane.ALDService/requestALDPayload',
            ald__pb2.ALDPorts.SerializeToString,
            ald__pb2.ALDReq.FromString,
            options, channel_credentials,
             insecure, call_credentials, compression, wait_for_ready,
timeout, metadata)

    @staticmethod
    def verifyALDResponse(request,
            target,
            options=(),
            channel_credentials=None,
            call_credentials=None,
            insecure=False,
            compression=None,
            wait_for_ready=None,
            timeout=None,
            metadata=None):
          return grpc.experimental.unary_unary(request, target,
'/mplane.ALDService/verifyALDResponse',
            ald__pb2.ALDResp.SerializeToString,
            ald__pb2.Verdict.FromString,
            options, channel_credentials,
             insecure, call_credentials, compression, wait_for_ready,
timeout, metadata)
```

Command to generate python supported files from prototype:

```
python -m grpc_tools.protoc -I./protos --python_out=.
--grpc_python_out=. ald.proto
```

ald_pb2.py and ald_pb2_grpc.py are the generated supported python files.

Format of XID files:

FI	GI	GL	PI	PL	PV	PI	PL	PV

XID Frame:

Table 2 Fields and Contents in the XID frame

Field	Content	Description
ADDR	12	Station address
CTRL	XID	Command
FI	0x81	Format identifier
GI	0x80	HDLC Parameter set
GL	18	Length of the parameter field (PI) in octets
PI	5	Maximum I Field length Transmit
PL	4	Length of the PV field in octets
PV	341040	Maximum I Field length Transmit in bits
PI	6	Maximum I Field length Receive
PL	4	Length of the PV field in octets
PV	224000	Maximum I Field length Receive in bits
PI	7	Maximum window size Transmit
PL	1	Length of the PV field in octets
PV	7	Maximum window size Transmit
PI	8	Maximum window size Receive
PL	1	Length of the PV field in octets
PV	3	Maximum window size Receive

For more information, refer to *Annexure B and C from 37462-g00 gpp* version.

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml with corresponding “xml” as listed in Table 1. There is no input json file.

2 Start the ALD server.

3 Start the dhcp server, M-Plane client. See Executing and starting M-Plane client and server on page 57.

4 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

Section 3.11: Log Management

List of Test cases

1. Troubleshooting

2. Trace

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml with corresponding “xml” as listed in Table 1.

2 Start the dhcp server, M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

Section 3.12: Connectivity Check

Test Case

- Ethernet Connectivity Monitoring

  The "Loopback Message (LBM)" feature of the Open RAN Studio software is used to test connectivity with the O-RU. Refer to the section LBM Configuration on page 40 to get an example of how to perform LBM configuration on the O-RU. After the LBM configuration has been applied to the O-RU, you can enable the LBM feature in the Instrument Configuration > Connectivity tab of the Open RAN Studio GUI. When you enable this feature, it allows you to send Loopback request messages and receive Loopback responses from O-RU.

  The corresponding call flow xml file for this test case is 4.6_Ethernet_Connectivity_Monitoring.xml and the execution steps are described below.

Steps to Execute

1. In the client config file (sample shown in Figure 30), specify the call-flow-xml value as 4.6_Ethernet_Connectivity_Monitoring.xml. This xml file is available under “duprofile/<version number>/” and contains the steps to execute for this test case.
2. Start the DHCP server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that the O-RU is running when the M-Plane client is started. If the O-RU is not running, the M-Plane client will wait until sleep secs time interval (100 sec) is expired. If the O-RU is not started before the expiry of this sleep interval, the M-Plane client will throw a Timeout error.

Result

1. If the test case is executed successfully, RPC exchanges (request message from client and reply message from server) are seen in the Windows command terminal.
2. Logs for the test execution are generated in C:\ProgramData\Keysight\MPlane\Client\log

Section 3.13: Performance Management

List of Test cases

1. Measurement Activation and De-activation

2. Collection and Reporting of Measurement Result

3. NETCONF process

For all the three test cases:

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value with corresponding “xml” as listed in Table 1. The corresponding json file is o-ran-pm.json. Input the values to “measurement-object”, “active”, “object-unit” and “report-info”. To test these cases, the value for “enable-sftp-upload” is input as “FALSE”.

2 Start the dhcp server, M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges request message from client and reply message from server is seen in the Windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

4. File Management process

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value with corresponding “xml” as listed in Table 1. The corresponding json file is o-ran-pm.json. Input the values to “measurement-object”, “active”, “object-unit” and “report-info”. To test these cases, the value for “enable-sftp-upload” is input as “TRUE”.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1. Successful rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.
2. Pass Logs for the execution is generated in '\log' folder.

Section 3.14: File Management

List of Test cases

1. File System Structure

2. File Management Operation: upload

See Log Management on page 103 for more information regarding these test cases listed above.

3. File Management Operation: retrieve file list

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “file-management.xml” as listed in Table 1. The corresponding json file is file-retrieval. Input values to “logical-path” and “file-name-filter” in the json file before execution.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges file retrieval message from client and reply message from server is seen in the Windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

4. File Management Operation: download

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “file-management.xml” as listed in Table 1. The corresponding json file is file-download-location. Input values to “local-logical-file-path”, “remote-file-path”, “password” and “certificate” in the json file before execution. “remote-file-path” shall be of the format:

sftp://username@hostname:22/path-to-non-existent-file or a file containing integrity error.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges file-download message from client and reply message from server is seen in the Windows command terminal.

2 Pass Logs for the execution is generated in '\log' folder.

Section 3.15: Synchronization Aspects

List of Test cases

1. Sync Status Object

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value as “default_call_flow.xml” as listed in Table 1.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

Section 3.16: O-RU Adaptive Delay Capability

Steps to execute

1. Fill the client config file (shown in Figure 30) under "du/profile/v1/" with `call-flow-xml` value as "4.8_O-RU_Adaptive_Delay_Capability.xml" as listed in Table 1.
2. Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.
3. Ensure that M-Plane server is running in O-RU.

Result

1. Successful rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.
2. Pass Logs for the execution is generated in '\log' folder.

Section 3.17: O-RU Monitoring of C/U Plane Connectivity

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml value to corresponding “4.10_O-RU_Monitoring_of_CU_Plane_Connectivity.xml” as listed in Table 1.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

Section 3.18: Configuration Management

List of Test cases

1. Framework for optional feature handling

2. M-Plane Operational State

3. Subscribing to updates from an O-RU

For all the three test cases:

Steps to execute

1 Fill the client config file (shown in Figure 30) under “du/profile/v1/” with call-flow-xml as “6_Configuration_Management.xml” as listed in Table 1.

2 Start the dhcp server and M-Plane client. See Executing and starting M-Plane client and server on page 57.

3 Ensure that M-Plane server is running in O-RU.

Result

1 Successful rpc exchanges of get message from client and reply message from server is seen in the windows command terminal.

2 Pass Logs for the execution is generated in ‘\log’ folder.

Section 3.19: Hybrid Functionality

Call-Home Functionality - DHCP server provides O-RU's transport layer address information together with the identity of the NETCONF client. O-RU performs transport layer resolution (DHCP, MAC, VLAN, IP, etc.) and recovers IP address(es) of O-RU controllers. O-RU, then begins synchronization of the O-RU against a Primary Reference Clock after which O-RU performs NETCONF Call Home to discovered O-RU controller(s). O-RU controller performs SSH connection establishment. O-RU and O-RU controller perform NETCONF capability discovery. O-RU controller performs optional provisioning of new management accounts (typically only performed once during pre staging).

Sample configuration:

```
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v2" -ru "ruprofile\
v2"
```

Without Call-Home Functionality - In this case, instead of the auto discovery of O-RU, the provisions are enabled in such a way that ruIP of the known RU is inputted to the execution parameters.

```
C:\ProgramData\Keysight\MPlane\Client>"C:\Program Files\Keysight\
MPlane\\Client\Keysight-5G-ORAN-M-PLANE-Client" -h

usage: Keysight 5G ORAN M-PLANE Client [-h] [-v] -ip IP -du DUPROFILE
[-ru RUPROFILE] [-ruIP RUIP]

optional arguments:

  -h, --help            show this help message and exit

  -v, --version         show program's version number and exit

  -ip IP, --ip IP       ip address of the interface

  -du DUPROFILE, --duprofile DUPROFILE

                        du profile path containing config file

  -ru RUPROFILE, --ruprofile RUPROFILE

                        ru profile path containing config file

  -ruIP RUIP, --ruIP RUIP

                        RU ip address of the interface
```

Sample configuration:

```
MPlaneClient.bat -ip 192.168.56.1 -du "duprofile\v2" -ru "ruprofile\
v2" -ruIP 192.168.56.7
```

References

The following documents were referred to, for the implementation of M-Plane and to prepare this document:

- ORAN-WG4.MP.0-v01.00 - O-RAN Alliance Working Group 4 Management Plane Specification
- RFC 6241, "Network Configuration Protocol (NETCONF)", IETF, June 2011
- RFC 7950, "The YANG 1.1 Data Modeling Language", IETF, August 2016

Contact us

For more information on Keysight Technologies' products, applications or services, contact your local Keysight office. The complete list is available at: www.keysight.com/find/contactus.