DALI ns-3 DC Extension – User Guide

Contents

User Guide: DALI – ns-3 Solution

DALI offers a new ns-3 module under the path "src/dali", which contains the new files generated by the DALI implementation. In the examples folder, we can find the ns-3 application DaliLteDcExperimentation, which provides extensive configuration options that allows testing and familiarize with DALI settings and its integration with NI API.

In this section, we will present the structure and characteristics of this ns-3 application. Nevertheless, the application includes comments that describes the purpose of different sections of the code. In order to understand the description in this section, a basic knowledge of C++ object oriented programing and ns-3 is recommended, which can be obtained through online ns-3 tutorial [1].

For DALI - ns3 implementation, the DC setup consists of two eNB nodes and two UE nodes (a Master and a Secondary in each case) executed in separate ns-3 instances, referred as MeNB, SeNB, mUE, and sUE, respectively. Additionally, a Remote Host and an EPC node need to be configured on MeNB ns-3 instance. A link is established to communicate PDCP and RLC layers of different eNB nodes through X2 interface and a newly implemented DALI UeDcx interface for communication between UEs.

DALI – ns3 implementation allows the Dual Connectivity execution in three hardware topology options as detailed below, and the choice of the topology setting and host-instance associations can be run using the only one application provided by DALI: DaliLteDcExperimentation. To allow the separation of one LTE network to different ns-3 instances, we used the "fake node" concept: every instance contains the full DC setup, but depending on the hardware topology chosen, specific nodes are allowed to communicate to the other instances through FdNetDevice installation on top of S1, X2, or Dcx interfaces. The different topology options are:

- One instance execution, as seen in Figure 1, includes all nodes of DC setup in one instance. The communication of interfaces is done through FdNetDevices (orange boxes), and the LTE channel is the simulated one provided by ns-3.
- Two instances execution, as seen in Figure 2, separates Master side (Remote Host, EPC, MeNB and mUE) and Secondary side (SeNB and sUE) in two instances/hosts. The communication of interfaces is done through FdNetDevices connected to a real network and the LTE channel is the simulated one provided by ns-3.
- Four instances execution with NI API integration, as seen in Figure 3, assigns Remote Host, EPC and MeNB to one host, and mUE, SeNB and sUE to three other hosts. The communication of interfaces is done through FdNetDevices connected to a real network and the LTE channel is provided by the ns3 - NI API implementations with SDR, and "Over the air" connection.

For Downlink, a client is installed in Remote Host and a Server is installed in mUE. For Uplink, a client is installed in mUE and a Server is installed in Remote Host.

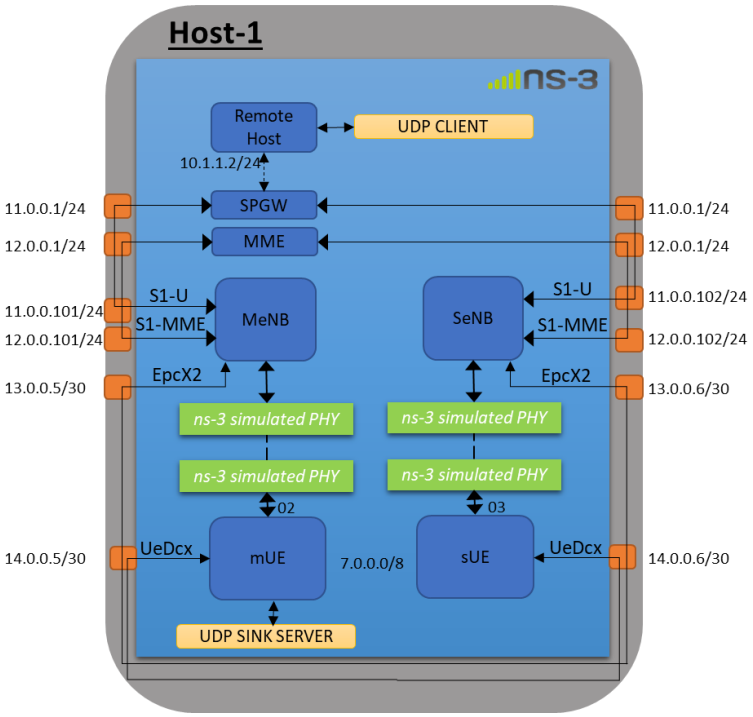

Figure 1. DALI Dual Connectivity one instance scenario

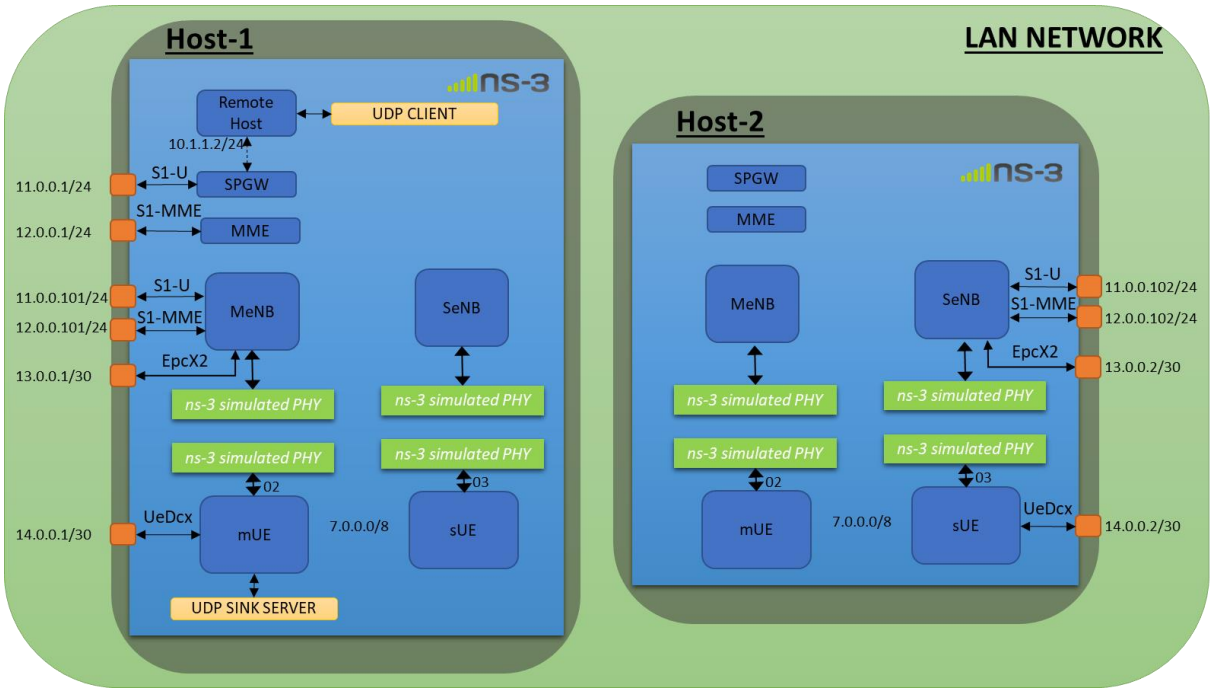

Figure 2. DALI Dual Connectivity two instances scenario

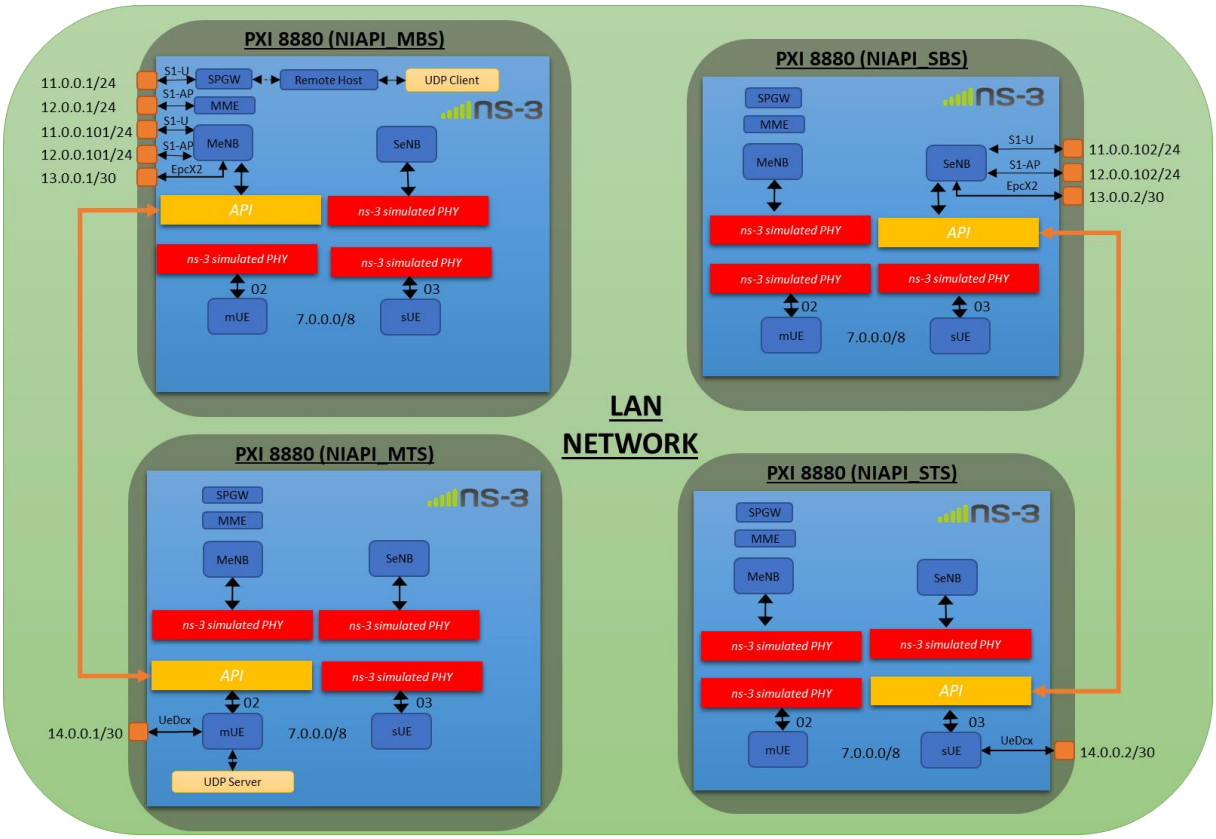

Figure 3. DALI Dual Connectivity four instances scenario, NI API integrated

Thanks to the modular design of ns3, DALI implementation does not interfere with any other possible configurations and scenarios that the specific ns3.26 NI version could have offered before, hence the new DALI ns-3 version can be used to integrate other functionalities that does not require DC.

In the following, this user guide first provides a DALI-ns3 installation guideline, then a basic description of the application execution, and finally a description of the coding of the application and the layer modifications performed. All these steps have been demonstrated through an online demo, the recording of which is published for public access at https://youtu.be/Ggr2q6KyQRI.

ns3 – DALI installation guide

The following steps need to be completed in every host to allow DC scenarios execution[1].

- Install ns3 prerequisites:
  https://www.nsnam.org/wiki/Installation
- Clone DALI-ns3 code (an access to the repository is required):
  $git clone https://github.com/ni/Orca-Dali-ns3
- Set up ns-3 by compiling modules including DALI, you will need to be a sudo user or root:
  $cd Orca-Dali-ns3
  $make all
  $sudo make install_ni

[1] Four instances execution with NI API integration needs at least one additional Host with NI Application Framework and PXI controllers for SDRs, as explained in detail in [2].

- Check the name of the network interface that is going to be used:
  $netstat –i or ifconfig
- Configure host interfaces for ns3-network interaction:
  $sudo ip link set <interface name> promisc on
- Verify the basic execution, as exemplified in Figure 4:
  $ns3.26-dali-lte-dc-experimentation-optimized --fdDeviceName="<interface name>"

```
-------- NS-3 Configuration -------------
Running LTE node as:        ENB and UE
LTE API:                    disabled
LTE UDP Loopback:           disabled
TapBridge:                  disabled
Logging:                    enabled
DALI Dual Connectivity:     disabled
DALI Throughput Evaluation: NI APP / UDP DOWNLINK
NI Module Version:          0.1.0

Init NI modules ...
Start logging thread to file /tmp/Log_Lte_BS.txt

-------- NS-3 Topology Information ------
Number of LTE UE devices   = LTE.UE.RRC: IDLE_START --> IDLE_WAIT_MIB (IMSI=1, RNTI=0)1

Number of LTE eNB devices  = 1
LTE Net GW IP Addr         = 10.1.2.2
LTE EPC PGW IP Addr        = 7.0.0.1
LTE UE#1 IP Addr           = 7.0.0.2
Client IP Addr             = 10.1.2.2
Server IP Addr             = 7.0.0.2

[>] Start simulation
LTE.UE.RRC: IDLE_WAIT_MIB --> IDLE_CAMPED_NORMALLY (IMSI=1, RNTI=0)
LTE.UE.RRC: IDLE_CAMPED_NORMALLY --> IDLE_WAIT_SIB2 (IMSI=1, RNTI=0)
LTE.UE.RRC: IDLE_WAIT_SIB2 --> IDLE_RANDOM_ACCESS (IMSI=1, RNTI=0)
LTE.UE.RRC: IDLE_RANDOM_ACCESS --> IDLE_CONNECTING (IMSI=1, RNTI=1)
LTE.ENB.RRC: calling InitialUeMessage, IMSI: 1, RNTI: 1
EPC.ENB.APPLICATION: calling m_s1apSapEnbProvider->SendInitialUeMessage
LTE.EPC.S1AP.ENB: Sending 41 bytes as InitialUeMessage to socket(1), mmeIpAddr: 12.0.0.1
LTE.ENB.RRC: INITIAL_RANDOM_ACCESS --> CONNECTION_SETUP (IMSI=1, RNTI=1)
LTE.UE.RRC: IDLE_CONNECTING --> CONNECTED_NORMALLY (IMSI=1, RNTI=1)
LTE.ENB.RRC: CONNECTION_SETUP --> CONNECTED_NORMALLY (IMSI=1, RNTI=1)
LTE.ENB.RRC: CONNECTED_NORMALLY --> CONNECTION_RECONFIGURATION (IMSI=1, RNTI=1)
LTE.ENB.RRC: CONNECTION_RECONFIGURATION --> CONNECTED_NORMALLY (IMSI=1, RNTI=1)
NI.CLIENT: sent 1000 bytes to 7.0.0.2 Uid: 1328 Sequence number: 0
NI.SERVER: received 988 bytes from 10.1.2.2 Uid: 1338 Sequence Number: 0
NI.CLIENT: sent 1000 bytes to 7.0.0.2 Uid: 1621 Sequence number: 1
NI.SERVER: received 988 bytes from 10.1.2.2 Uid: 1624 Sequence Number: 1
NI.CLIENT: sent 1000 bytes to 7.0.0.2 Uid: 1798 Sequence number: 2
NI.SERVER: received 988 bytes from 10.1.2.2 Uid: 1800 Sequence Number: 2
NI.CLIENT: sent 1000 bytes to 7.0.0.2 Uid: 1951 Sequence number: 3
NI.SERVER: received 988 bytes from 10.1.2.2 Uid: 1957 Sequence Number: 3
NI.CLIENT: sent 1000 bytes to 7.0.0.2 Uid: 2100 Sequence number: 4
NI.SERVER: received 988 bytes from 10.1.2.2 Uid: 2103 Sequence Number: 4
[#] End simulation

Received packets: 5 / Lost packets: 0

De-init NI modules ...
Terminating logging thread...

-------- Program end! -------------------
```

Figure 4. DaliLteDcExperimentation application basic execution check output

Basic description of application execution

DaliLteDcExperimentation example offers different modes of execution and various parameters that can be configured according to the evaluation purpose and host characteristics using ns3 command line attributes. Once compiled and installed, the list of parameters and their descriptions can be seen by executing this example with --help option as shown in Figure 5. Next, we describe these parameters, their possible values, description, and for which type of scenario they are relevant. All the parameters that are passed as words must be inserted between double quotes, and they are case sensitive. Number values must be inserted without units. The default values of all parameters are shown within square brackets.

```
dali_ns3@dalins3-OptiPlex-5040:~$ ns3.26-dali-lte-dc-experimentation-optimized --help
ns3.26-dali-lte-dc-experimentation-optimized [Program Arguments] [General Arguments]

Program Arguments:
    --packetSize:                   size of application packet sent [1000]
    --numPackets:                   number of packets generated [5]
    --interval:                     interval (milliseconds) between packets [500]
    --simTime:                      Duration in seconds which the simulation should run [10]
    --transmTime:                   Time in seconds when the packet transmission should be scheduled [5]
    --niApiEnableTapBridge:         Enable/disable TapBridge as external data source/sink [false]
    --niApiEnableLogging:           Set whether to enable NIAPI_DebugLogs [true]
    --niApiDevMode:                 Set whether the simulation should run as Master or Secondary, and BS or Terminal [NIAPI_BSTS]
    --niApiLteEnabled:              Enable NI API for LTE [false]
    --niApiLteLoopbackEnabled:      Enable/disable UDP loopback mode for LTE NI API [false]
    --daliDualConnectivityEnabled:  Enable/disable DALI Dual Connectivity configuration [false]
    --fdDeviceName:                 Interface name for emu FdDevice communication [eth1]
    --dualConnectivityLaunchTime:   Time in seconds when DALI Dual Connectivity packet transmission should be scheduled [4]
    --usePdcpInSequenceDelivery:    Enable/disable DALI PDCP In-Sequence reordering function [false]
    --daliTransportProtocol:        Set whether the simulation should use, NI (application service), UDP or TCP as transport protocol [NI]
    --daliLinkDirection:            Set simulation traffic direction, UPLINK / DOWNLINK [DOWNLINK]
    --dataRate:                     Data transmission Rate (Application layer desired rate for UDP) [60Mbps]
    --tcpVariant:                   Transport protocol to use: TcpNewReno, TcpHybla, TcpHighSpeed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno
, TcpBic, TcpYeah, TcpIllinois, TcpWestwood, TcpWestwoodPlus, TcpLedbat  [TcpNewReno]
```

Figure 5. DaliLteDcExperimentation application parameter options

--packetSize: The size of the application packet (payload) sent, the default value of which is 1000 bytes. The upper limit for the packet size is 1400 bytes, in order to not surpass the maximum Ethernet frame size, which is 1522 bytes, where 1500 bytes of it are the payload. Exceeding this upper limit causes packet fragmentation, which is not supported by FdNetDevice, for which case the packets would not be transmitted between the interfaces. The lower limit is restricted by the processing capacity of the host, where managing creation and transmission of packets of less than 250 bytes compromised the Real-Time execution requirements.

--numPackets: The number of packets generated. The default value is 5 packets. This parameter is considered only when NI client/server application is used. This value represents the total number of packets to be transmitted and for which NI statistics are shown at the end of the execution. Hence, it is important to consider the settings of simulation time and the interval between packets so that there will be time for all packets to be transmitted.

--interval: The interarrival time (in milliseconds) of the packets. Default value is 500 milliseconds. This parameter is considered only when NI client/server application is used.

--simTime: Duration in seconds for which the simulation should run. Default value is 10 seconds.

--transmTime: Time in seconds when the first packet transmission is scheduled. Default value is 5 seconds. This interval of time is useful to consider equipment connection to LTE network and other configurations before any client starts sending packets across the network. It is important to plan these time settings such that simTime – transmTime ≥ Total Time where packets would be transmitted.

--niApiEnableTapBridge: Enable/disable TapBridge as external data source/sink. Default value is “false”. This only can be used when eNB and UE are installed in different hosts, i.e. NIAPI_BS, NIAPI_TS, NIAPI_MBS, NIAPI_MTS niApiDevModes.

--niApiEnableLogging: Set whether to enable NIAPI_DebugLogs. Default vale is “true”. Useful Log messages are shown for debugging purpose. It is recommended to be disabled when a large amount of packets are transmitted, e.g., for throughput evaluations.

--niApiDevMode: Set whether the simulation should run as Master or Secondary, and BS (Base Station) or TS (Terminal) Default value is “NIAPI_BSTS”. DALI – ns3 implementation defines three hardware topology configurations that can be executed, and correspondent niApiDevMode would be:

- One instance execution, as seen in Figure 1, “NIAPI_BSTS”.
- Two instances execution, as seen in Figure 2, “NIAPI_MBSTS” for Master side, and “NIAPI_SBSTS” for secondary side.

- Four instances execution with NI API integration, as seen in Figure 3, “NIAPI_MBS”, “NIAPI_MTS”, “NIAPI_SBS”, and “NIAPI_STS”.

Note that the legacy NI configuration without DALI Dual Connectivity execution modes are still available: “NIAPI_BS”, “NIAPI_TS”, and “NIAPI_BSTS”.

--niApiLteEnabled: Enable NI API for LTE. Default value is “false”. Enabling this parameter activates NIAPI for DALI implementation.

--niApiLteLoopbackEnabled: Enable/disable UDP loopback mode for LTE NI API. Default value is “false”. Allows to evaluate LTE NI API without SDR hardware, it can be a development environment with at least two hosts, to separate Four instances execution in Master side (NIAPI_MBS, NIAPI_MTS) and Secondary side (NIAPI_SBS, NIAPI_STS).

--daliDualConnectivityEnabled: Enable/disable DALI Dual Connectivity configuration. Default value is “false”. Mandatory to be set to “true” for DALI DC evaluation.

--fdDeviceName: The interface name for EmuFdNetDevice communication. Default value is “eth1”. Indispensable to set correctly for the host where the application is executed.

--dualConnectivityLaunchTime: Time in seconds when DALI Dual Connectivity is started, i.e., DC control signalling is scheduled. Default value is 4 seconds. At this time, DALI configuration will be launched and, if completed, split bearer will be set up. When the throughput is being evaluated, it is recommended to set this value before transmTime, since a flooding of data packets can result in a loss of the configuration messages and the Dual Connectivity setup may not be completed.

--usePdcpInSequenceDelivery: Enable/disable DALI PDCP In-Sequence reordering function. Default value is “false”. This is especially useful for TCP evaluations, but it can be also used for all other applications available since it is executed within PDCP layer.

--daliTransportProtocol: Set whether the simulation should use, NI (application service), UDP or TCP as transport protocol. Default value is “NI”.

--daliLinkDirection: Set simulation traffic direction, UPLINK / DOWNLINK. Default value is “DOWNLINK”

--dataRate: Data transmission rate for UDP application. Default value is “60Mbps”. This value is considered only when UDP is configured as traffic application. Note that the unit is bytes per second.

--tcpVariant: Transport protocol to use: TcpNewReno, TcpHybla, TcpHighSpeed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno, TcpBic, TcpYeah, TcpIllinois, TcpWestwood, TcpWestwoodPlus, TcpLedbat. Default value is “TcpNewReno”. This parameter is considered when TCP is configured as traffic application.

As seen, all parameters have a default value. That is why, it is not necessary to define all parameters in some scenarios. For the basic tests that can be run to verify the execution of the available hardware topology configurations for DALI, the following terminal commands can be used.

- Legacy NI operation (EPC + eNB + UE):

  $ns3.26-dali-lte-dc-experimentation-optimized --fdDeviceName="<device_name>"

- One instance execution, as seen in Figure 1:

  $ns3.26-dali-lte-dc-experimentation-optimized --fdDeviceName=="<device_name>"

  --daliDualConnectivityEnabled="true"

- Two instances execution, as seen in Figure 2:

Host 1:

$ns3.26-dali-lte-dc-experimentation-optimized --fdDeviceName="<device_name>"

--daliDualConnectivityEnabled="true" --niApiDevMode="NIAPI_MBSTS"

Host 2:

$ns3.26-dali-lte-dc-experimentation-optimized --fdDeviceName="<device_name>"

--daliDualConnectivityEnabled="true" --niApiDevMode="NIAPI_SBSTS"

- Four instances execution with NI API integration, as seen in Figure 3:

MeNB

Command: ns3.26-dali-lte-dc-experimentation-optimized

Arguments: --niApiDevMode="NIAPI_MBS" --niApiLteEnabled="true"

--daliDualConnectivityEnabled="true" --fdDeviceName="<device_name>"

SeNB

Command: ns3.26-dali-lte-dc-experimentation-optimized

Arguments: --niApiDevMode="NIAPI_SBS" --niApiLteEnabled="true"

--daliDualConnectivityEnabled="true" --fdDeviceName="<device_name>"

mUE

Command: ns3.26-dali-lte-dc-experimentation-optimized

Arguments: --niApiDevMode="NIAPI_MTS" --niApiLteEnabled="true"

--daliDualConnectivityEnabled="true" --fdDeviceName="<device_name>"

sUE

Command: ns3.26-dali-lte-dc-experimentation-optimized

Arguments: --niApiDevMode="NIAPI_STS" --niApiLteEnabled="true"

--daliDualConnectivityEnabled="true" --fdDeviceName="<device_name>"

DALI code details and key layer modifications

DaliLteDcExperimentation file is located inside the examples folder of the new ns3 DALI module, with the name dali-lte-dc-experimentation.cc.

The code highlights of the key helpers' class interactions and main program sections within simulation application and helpers are presented in Appendix A.

The comments in the code make it simple to follow it. As can be seen, DALI-specific configurations are executed only if relevant flags are activated, by wrapping the baseline code with "if" statements.

The split bearer is a key feature of DC, which has been implemented in PDCP layer in line with the 3GPP specifications. Although Release 13 is the one indicated in the ORCA Open Call, there is no significant changes for E-UTRA DC in Releases 14 or 15. Hence, we can state that the implemented functionalities are Release 15 compliant. In the following we explain certain details about Split Bearer that is expected to be very important for the experimenters:

- Traffic load distribution between Master or Secondary link: By default, the Split bearer is configured when Dali Dual Connectivity is enabled. To control the traffic flow, a different class has to be modified based on the link direction: for Downlink, DaliEnbPdcp.cc file, and for Uplink, DaliUePdcp.cc. The simple flow control implemented in DALI for split bearer alternately forwards the PDCP PDUs via master (local stack) and secondary (remote stack) LTE links. This is controlled by the next statement located in the DoTransmitPdcpSdu() method:

  m_sendOverSecondary = !m_sendOverSecondary;

  As seen, m_sendOverSecondary flag is the one in charge of determining if the local stack or remote stack is used. By default, m_sendOverSecondary flag is set to false when the object is created. The code line above can be replaced with a new algorithm to determine how many packets are sent via one or another side.
- Improvement of reordering function in PDCP layer: DALI introduces a simple reordering function based on sequence numbers (SNs) of PDCP PDUs. It uses a simple implementation of reordering window (RW). The reordering window starts in the next expected SN, and ends at the length RW. One last consideration that has to be taken in account is that SN is a cyclic sequence, i.e. when the maximum possible SN is assigned, it continues with 0. In DALI implementation, a buffer will keep the out-of-order PDU received, until a packet outside of the RW arrives. In the latter case, the algorithm will unload the buffer until the new packet can enter into the RW. Note that DALI reordering function do not implement a timer-based out-of-order PDU handling. The method DoReceivePdu have been modified to implement in-order delivery, and can be activated though the flag UseInSequenceDelivery, configured as an attribute of the object DaliLteHelper. This algorithm can be extended, to consider all the required characteristics of it based on 3GPP specifications.

Single UE with Dual Connectivity capabilities. The PDCP layers of eNB and UE exhibit similar functions, and could have been integrated in just one file/class. But separation of these classes considers a future development of one UE stack with Dual Connectivity capabilities, that is, to communicate with eNBs over different RAT technologies at the same time. For this, DaliUePdcp class needs to extended and new relevant functions are necessary to implement specific functionalities of eNB and UE. Moreover, currently, NI API allows one LTE SDR per Host, which is one of the main reasons of the actual (dual stack) implementation of DALI.

References

[1] ns-3 Tutorial. [online]. Available in: https://www.nsnam.org/docs/tutorial/html/

[2] Provisioning a Real-Time Controller or USRP Stand-Alone Device for LabVIEW Communications [online]. Available: http://www.ni.com/tutorial/54622/en/