ns-3 Public Safety Communications documentation

Release psc-1.0

National Institute of Standards and Technology (NIST)

Aug 17, 2018

CONTENTS

This is documentation for *ns-3* models relating to public safety communications, extracted from the overall *ns-3 Model Library* documentation.

This document is written in reStructuredText for Sphinx and is maintained in the `doc/psc-models` directory of the PSC repository.

CHAPTER

ONE

PUBLIC SAFETY COMMUNICATIONS

1.1 Public Safety Communications Overview

ns-3 support for public safety communications (PSC) is based on relatively new capabilities for 4G LTE systems introduced in 3GPP Release 12 and later releases. Device-to-Device (D2D) Proximity Services (ProSe) communications are terms relating to features allowing for UEs to discover each other, synchronize, and communicate with each other, with or without the use of an eNodeB. These services make use of a so-called *sidelink* channel between UEs.

Support for public safety communications is distributed among the following four *ns-3* modules:

1. `psc`: (this module) Support for models and scenarios that are specific to public safety communications.
2. `lte`: Support for ProSe (sidelink communications).
3. `buildings`: Support for pathloss models including building effects, as defined by 3GPP with relevance to public safety scenarios.
4. `antenna`: Parabolic antenna model as described in 3GPP document TR 36.814

Documentation for the PSC features implemented in the *ns-3* `lte`, `buildings`, and `antenna` modules is provided in the respective module documentation. This chapter documents the *ns-3* `psc` module.

At present, the majority of the code related to public safety communications is found in the ProSe implementation in the `lte` module. This is because the ProSe services of sidelink communications, discovery, and synchronization are deeply connected to the LTE models and difficult to factor into a separate module. Features intrisic to the low-level operation of ProSe in LTE are found in the `lte` module.

There is no support for legacy public safety communications such as land mobile radio system (LMRS), and only IPv4 (and not IPv6) is presently supported.

Future extensions to this module or other related modules such as LTE are planned for the following features:

1. mission-critical push-to-talk (MCPTT) and possibly other public safety applications
2. UE-to-network relay
3. public safety scenario support code and example programs

1.1.1 Acknowledgments

Public safety communications features are based on development led by the Wireless Networks Division of the U.S. National Institute of Standards and Technology, described in publications (*[NIST2016]* and *[NIST2017]*). Users of the D2D features of *ns-3* are requested to cite *[NIST2017]* in academic publications based on these models.

The following individuals are authors of the public safety communications extensions:

- Aziza Ben-Mosbah (aziza.ben.mosbah@gmail.com)

- Fernando J. Cintron (fernando.cintron@nist.gov)
- Samantha Gamboa (samantha.gamboa@nist.gov)
- Richard Rouil (richard.rouil@nist.gov)

This release was ported from an earlier release, based on ns-3.22, to the ns-3.29 release, and was integrated with the LTE and buildings module, and extended to include additional example programs and tests, by CTTC and the University of Washington. This work was performed under the financial assistance award 70NANB17H170 from U.S. Department of Commerce, National Institute of Standards and Technology.

1.2 UDP Group Echo Server

The class `ns3::UdpGroupEchoServer` implements a group echo server that echoes received UDP datagrams to a set of clients. The policy of generating replies can be tailored through the use of attributes.

The relevance of this model to public safety communications is that such scenarios often require a many-to-many group communications application, and existing *ns-3* applications are not suitable to generate such traffic.

The UDP group echo implementation is authored by Fernando J. Cintron (fernando.cintron@nist.gov) and is derived from the UdpEchoServer found in the *ns-3* applications module.

1.2.1 Model Description

The implementation is provided in the following files:

- `src/psc/model/udp-group-echo-server.{h,cc}` The model itself
- `src/psc/helper/udp-group-echo-helper.{h,cc}` Helper code for configuration

Additionally, a simple example is provided at `src/psc/examples/example-udp-group-echo.cc`.

The model is based on the `ns3::UdpEchoServer`, but differs in that the existing server only handles one client, while the group echo server handles one or more clients. The behavior of the `ns3::UdpEchoServer` can be reproduced (i.e., it is a special case of this object).

The model works as follows. The UdpGroupEchoServer is an *ns-3* application, listening for UDP datagrams on a configured UDP port. Upon receipt of a datagram for the first time from a client, the server records the client and a timestamp for when the packet was received. The server then decides whether to forward the packet back to one or more clients, on a pre-configured 'EchoPort'. The set of possible clients is built dynamically based on received packets.

There are a few configurable policies:

1. The server may be configured to echo only to the client that originated the packet (similar to the `UdpEchoServer`).
2. The server may be configured to echo to a group of clients including the sender. Furthermore, the sending client may be excluded from the response.
3. The server may be configured to echo to a group of clients including the sender, so long as the server has heard from each client within a configurable timeout period. Furthermore, the sending client may be excluded from the response.

Attributes

The following is the list of attributes:

- `Port`: Port on which the server listens for incoming packets, default value of 9.
- `EchoPort`: Port on which the server echoes packets to client, default value of 0.
- `Mode`: Mode of operation, either no group session (reply to sender only), timeout limited session (replicate to all clients for which a packet has been received from them within the configured timeout period), and infinite session (reply to all known clients). The default is no group session.
- `Timeout`: Inactive client session expiration time, default of zero seconds.
- `EchoClient`: Whether the server echoes back to the sending client, default value of true.,

Trace sources

The model also provides a `Rx` trace source for all received datagrams.

1.2.2 Usage

A simple example based on CSMA links is provided in the file `src/psc/examples/example-udp-group-echo.cc`.

```
//   *    *
// n0   n1   ... n(nExtra)  n(1+nExtra)
//  |    |    |    |         |
//  ==========================
//     LAN 10.1.2.0
```

By default, node n0 is the client and node n1 is the server, although additional nodes can be added with the `--nExtra` argument. If there are more nodes, the highest numbered node is the server.

Each client is configured with an on-off traffic generator that sends traffic at random times to the server. The following program options exercise some of the configuration of the server:

```
$ ./waf --run 'example-udp-group-echo --PrintHelp'

Program Options:
  --nExtra:      Number of "extra" CSMA nodes/devices [0]
  --echoClient:  Set EchoClient attribute [true]
  --mode:        Set Mode attribute (InfSession|NoGroupSession|TimeoutLimited)␣
↪[InfSession]
  --timeout:     Set Timeout attribute [+0.0ns]
  --verbose:     Tell echo applications to log if true [true]
  --enablePcap:  Enable PCAP file output [false]
  --time:        Simulation time [10]
```

By default, the number of extra CSMA nodes/devices is zero, so there will be only one client and one server. By running with the existing defaults, the program will configure the server to echo back to the client, use the ‘InfSession’ mode of operation (to echo without considering timeout value) and a timeout value of 0 (which is not used in this mode). Logging is also enabled, and running the program with defaults yields output such as:

```
$ ./waf --run 'example-udp-group-echo'
...

8.7006 server received 41 bytes from 10.1.2.1 port 49153
Client found; old timestamp: 8.6806
New timestamp: 8.7006
8.7006 number of clients: 1
```

```
---------------------------------------------------------
               Client     Session
---------------------------------------------------------
         10.1.2.1:49153          0
=========================================================
8.7006 server sent 41 bytes to 10.1.2.1 port 49153
```

This shows that node 10.1.2.1 sent data at time 8.7006 that was echoed back to the client. Running with the option `--nExtra=2` shows:

```
$ ./waf --run 'example-udp-group-echo --nExtra=2'

...

9.99949 server received 41 bytes from 10.1.2.1 port 49153
Client found; old timestamp: 9.97949
New timestamp: 9.99949
9.99949 number of clients: 3
---------------------------------------------------------
               Client     Session
---------------------------------------------------------
         10.1.2.1:49153          0
         10.1.2.2:49153 0.00647334
         10.1.2.3:49153 0.00426833
=========================================================
9.99949 server sent 41 bytes to 10.1.2.1 port 49153
9.99949 server sent 41 bytes to 10.1.2.2 port 49153
9.99949 server sent 41 bytes to 10.1.2.3 port 49153
```

Here, one packet sent leads to three packets echoed, and the data under the 'Session' column shows the time in seconds since the last packet was received from each client. In mode 'InfSession', there is no explicit timeout, but if we set a timeout to be something small, such as 5ms, and set the mode to 'TimeoutLimited', we can suppress the response to node 10.1.2.2 because it has last polled the server over 6ms ago.

```
$ ./waf --run 'example-udp-group-echo --nExtra=2 --timeout=5ms --mode=TimeoutLimited'

...

9.99949 server received 41 bytes from 10.1.2.1 port 49153
9.99949 number of clients: 3
---------------------------------------------------------
               Client     Session
---------------------------------------------------------
         10.1.2.1:49153          0
         10.1.2.2:49153 0.00647334 **Session Expired!**
         10.1.2.3:49153 0.00426833
=========================================================
9.99949 server sent 41 bytes to 10.1.2.1 port 49153
9.99949 server sent 41 bytes to 10.1.2.3 port 49153
```

As seen above, the session has expired to 10.1.2.2 with this timeout and mode setting, so the echo response is suppressed.

The C++ code for setting the server is fairly standard *ns-3* syntax and container/helper-based code, as exemplified below:

```
uint16_t serverPort = 9;
// Attributes 'timeout', 'mode', 'echoClient' may be set above
```

```
UdpGroupEchoServerHelper echoServer (serverPort);
echoServer.SetAttribute ("Timeout", TimeValue (timeout));
echoServer.SetAttribute ("Mode", StringValue (mode));
echoServer.SetAttribute ("EchoClient", BooleanValue (echoClient));
ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma - 1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (simTime));
```

CHAPTER

TWO

LTE D2D MODELS

This chapter is an excerpt of the LTE module documentation chapter, containing the D2D-related models that have been added to the *ns-3* LTE module.

2.1 Design Documentation

2.1.1 LTE Sidelink

Overview

This section describes the *ns-3* support for LTE Device to Device (D2D) communication based on the model published in *[NIST2016] [NIST2017]*.

3GPP introduced D2D Proximity Services (ProSe) in release 12 to allow ProSe enabled devices to exchange information directly, i.e., without traversing the eNB. Figure *Non-Roaming Reference Architecture* shows new ProSe entities and the new interfaces added to the network *[TS23303]*.

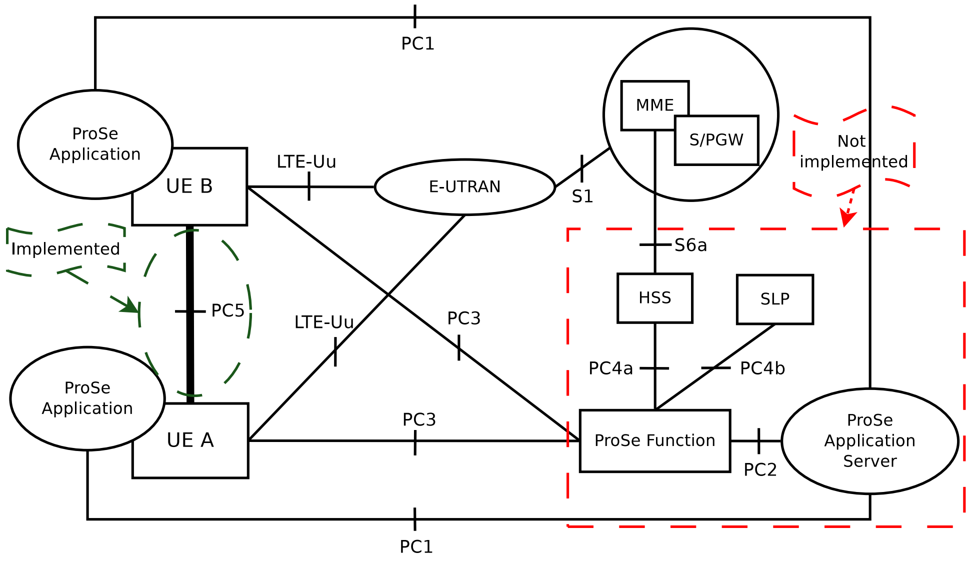


Fig. 2.1: Non-Roaming Reference Architecture

At the time of writing this documentation only the new radio interface, i.e., PC5 is implemented. This interface is also known as Sidelink at physical layer. The model supports all the three following LTE D2D functionalities defined under ProSe services:

1. Direct communication
2. Direct discovery
3. Synchronization

These LTE D2D functionalities can operate regardless of the network status of the UEs. Thus, four scenarios were identified by 3GPP *[TR36843]*:

1A. Out-of-Coverage

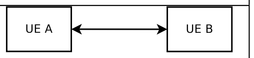
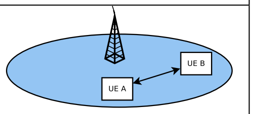
1B. Partial-Coverage

1C. In-Coverage-Single-Cell

1D. In-Coverage-Multi-Cell

At this stage, the model has been tested for scenario 1A and 1C. The Table *ns-3 LTE Sidelink supported/tested scenarios* gives more information about the support of these two scenarios for all the three ProSe services.

Table 2.1: ns-3 LTE Sidelink supported/tested scenarios

#	Description	UE A	UE B	Direct Communication	Direct Discovery	Synchronization	Example
1A	Out-of-Coverage	Out-of-Coverage	Out-of-Coverage	Yes RA = Mode 2	Yes RA = Type 1	Yes Autonomous synchronization	
1C	In-Coverage Single Cell	In-Coverage	In-Coverage	Yes RA = Mode 1 RA = Mode 2	Yes RA = Type 1	Yes Network synchronization	
RA = Resource Allocation							

The model is developed in a way that simulating the above ProSe services are not interdependent. In the following, we describe all the changes introduced in the *ns-3* LTE architecture and its protocol stack to realize Sidelink functionality.

Architecture

eNB architecture

There is no change in the eNB data (fig-ca-enb-data-plane), control (fig-ca-enb-ctrl-plane) plane and neither in its PHY/channel (fig-lte-enb-phy) model.

UE architecture

There is no change in the UE data (fig-ca-ue-data-plane) and control (fig-ca-ue-ctrl-plane) plane. However, in order to receive packets transmitted by other UEs in the uplink channel a new instance of the `SpectrumPhy` class, as shown in Figure *PHY and channel model architecture for the UE*, is added to the PHY/channel architecture of the UE.

NAS

In LTE module, the NAS layer functionality is provided by `EpcUeNas` class. In addition to its existing capabilities, this class has been extended to support LTE ProSe services. In particular, direct communication and direct discovery.

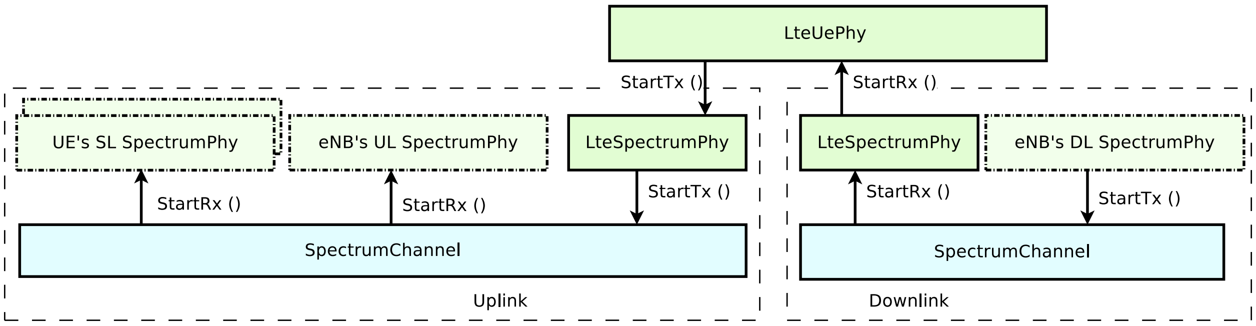


Fig. 2.2: PHY and channel model architecture for the UE

For direct communication, it supports the functions to activate/deactivate Sidelink bearers. Since, there is no EPS bearer for Sidelink communication, the existing TFT can not be used to map IP packets to a Sidelink bearer. Therefore, a new type of TFT, called `LteSlTft` is implemented. It maps IP packets to the Sidelink bearers based only on their destination IP address. Moreover, the `Send` function has been extended for both UE NAS "Active" and "Off" states. In the active state, if there is a Sidelink bearer established between the source and the destination UE, it utilizes Sidelink bearer to send the packet, thus, prioritizing a Sidelink bearer over normal LTE uplink bearer. On the other hand, in off state in which previously UE was unable to send the packets, is now able to use a Sidelink bearer, if established.

For the direct discovery, it conveys the information, e.g., list of discovery applications and the nature of the application, i.e., announcing or monitoring to `LteUeRrc` class.

RRC

The RRC layer has been modified to support all the three ProSe services. Among all the new changes a major modification, which is common in both eNB and UE RRC is the addition of two new classes called, `LteSlEnbRrc` and `LteSlUeRrc`. Both the classes serves a common purpose of holding Sidelink resource pool (i.e, communication and discovery) configuration done through the functions added to the `LteHelper` class. Figures *Relationship between LteEnbRrc and LteSlEnbRrc* and *Relationship between LteUeRrc and LteSlUeRrc* show the relationship between `LteEnbRrc<-->LteSlEnbRrc` and `LteUeRrc<-->LteSlUeRrc` classes.

Note: Only the key responsibilities of the new classes are shown. For complete view of these classes please refer to their class APIs.

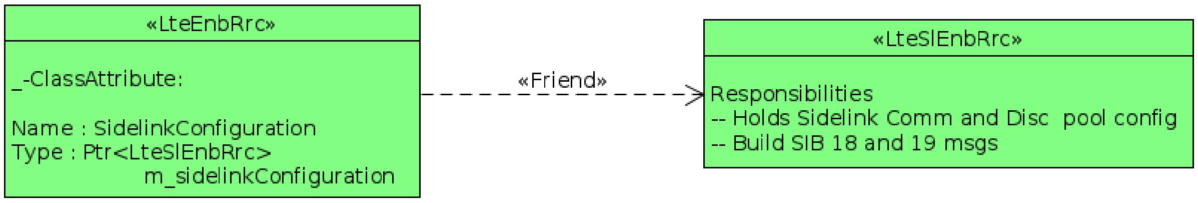


Fig. 2.3: Relationship between LteEnbRrc and LteSlEnbRrc

The following Figure *ns-3 LTE Sidelink pool configuration flow* shows the interaction among the classes to configure a Sidelink pool.

The Sidelink pools, for both in-coverage and out-of-coverage scenarios are configured through the user's simulation script. Moreover, all the RRC SAP classes, i.e., control and data including the `LteRrcSap` class have been extended to support Sidelink functionalities.

In the following we will explain the remaining modifications specifically introduced in the eNB and UE RRC layer.

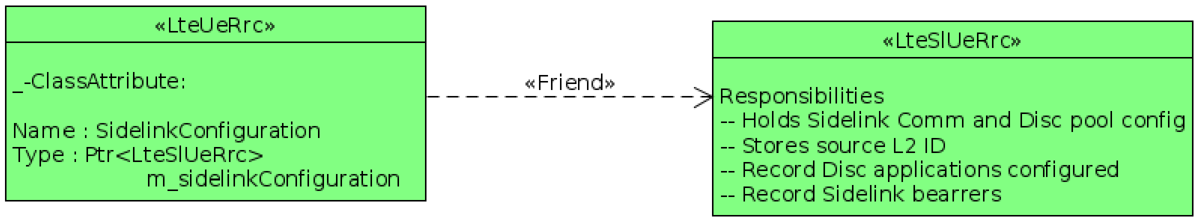

«LteUeRrc»
_-ClassAttribute:
Name : SidelinkConfiguration
Type : Ptr<LteSlUeRrc>
m_sidelinkConfiguration
«Friend»
«LteSlUeRrc»
Responsibilities
-- Holds Sidelink Comm and Disc pool config
-- Stores source L2 ID
-- Record Disc applications configured
-- Record Sidelink bearrers

Fig. 2.4: Relationship between LteUeRrc and LteSlUeRrc

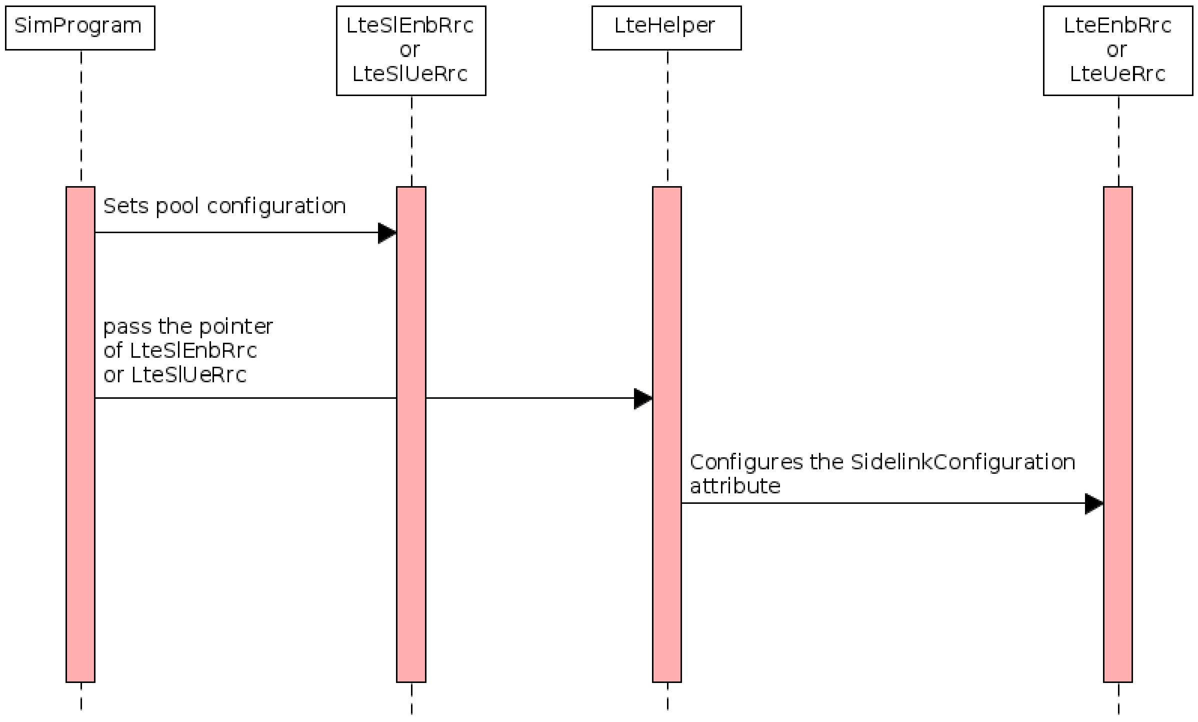

SimProgram
LteSlEnbRrc or LteSlUeRrc
LteHelper
LteEnbRrc or LteUeRrc
Sets pool configuration
pass the pointer of LteSlEnbRrc or LteSlUeRrc
Configures the SidelinkConfiguration attribute

Fig. 2.5: ns-3 LTE Sidelink pool configuration flow

eNB RRC

To support in-coverage Sidelink scenarios, SIB18 and SIB19 were added to broadcast Sidelink resource pool configuration for communication and discovery respectively. For simplicity, we use the same periodicity (default : 80 ms) of all the SIB messages, which could be configured by changing the attribute `SystemInformationPeriodicity` value. The resource pools are configured through `LteHelper` as discussed earlier. Similar to other SIB messages, SIB18 and SIB19 are defined in `LteRrcSap class`. By receiving these SIB messages, a UE can deduce the type of ProSe service an eNB can support. The eNB is also now capable of processing `SidelinkUeInformation` messages sent by the UEs [TS36331]. In general, this type of message contains information related to the frequency UE is intended to use for Sidelink, resources required by the UE for Sidelink communication or discovery and a list of destination identities, i.e., group ids. We note that, **at this moment only one Tx/Rx pool and one group per UE** is supported. In response to the `SidelinkUeInformation`, eNB sends an `RrcConnectionReconfiguration` message containing the resource allocation information as per the supported ProSe services. In case of in-coverage Sidelink communication, resource allocation is performed as per MODE1, which is also reffered as `Scheduled` mode *[TR36843]*. In this mode, the `RrcConnectionReconfiguration`, as per the standard, includes the specifications of the pool to be used and the timing specification for Sidelink Buffer Status Report (SL-BSR) transmission and re-transmission. On the other hand, if the pool is for MODE 2, i.e., `UeSelected` mode, it only includes the dedicated pool specifications.

For the in-coverage discovery, only `Type1`, i.e. UE selected resource allocation is supported.

UE RRC

The `LteUeRrc` class has been extended to support all the ProSe services for both, in-coverage and out-of-coverage mode. To highlight all the modifications, lets discuss them in context of each ProSe service.

Sidelink direct communication

The UE RRC now supports the creation of Sidelink bearers for both in-coverage and out-of-coverage scenarios. A new function `DoActivateSidelinkRadioBearer` is implemented for this purpose. If the configuration of the TFT used indicates the nature of the Sidelink radio bearer as``Bidirectional``, the creation of Sidelink bearers for Tx and Rx, which is to populate the Tx/Rx pool configuration along with the list of destinations to the lower layers, occurs at the same time. Only the creation of PDCP/RLC instances for TX bearer is done in `DoActivateSidelinkRadioBearer` function. The creation of PDCP/RLC instances for Rx occurs upon the reception of first Sidelink packet [TS36300]. For the in-coverage case, the bearer establishment procedure involves the communication with the eNB by sending `SidelinkUeInformation` message for the sake of resource allocation as shown in Figures *ns-3 LTE Sidelink in-coverage radio bearer activation (Tx)* and *ns-3 LTE Sidelink in-coverage radio bearer activation (Rx)*.

For the out-of-coverage scenario, the Sidelink bearer activation and the resource allocation is done by the UE autonomously. Figure *ns-3 LTE Sidelink out-of-coverage radio bearer activation (Tx)* and *ns-3 LTE Sidelink out-of-coverage radio bearer activation (Rx)* show sequence diagrams of the out-of-coverage Sidelink bearer activation for TX and RX respectively.

As mentioned earlier, the UE RRC is now also capable of processing new SIB18 message and the Sidelink direct communication configuration received in `RrcConnectionReconfiguration` message. For in-coverage scenarios, resource allocation can be done in MODE 1 and MODE 2. In MODE 1, the resources are set to "Scheduled" and the eNB is responsible of scheduling the exact number of resources (i.e., with the help of the scheduler), while in MODE 2, resources are "UeSelected" and a UE receives only one dedicated resource pool through `RrcConnectionReconfiguration` message to choose the resources from [TS36331]. By standard the SIB18 message is used as an indicator that an eNB supports ProSe communication services, thus, the UE transmits `SidelinkUeInformation` if it has received SIB18 message. A UE in "IDLE_CAMPED_NORMALLY" or in

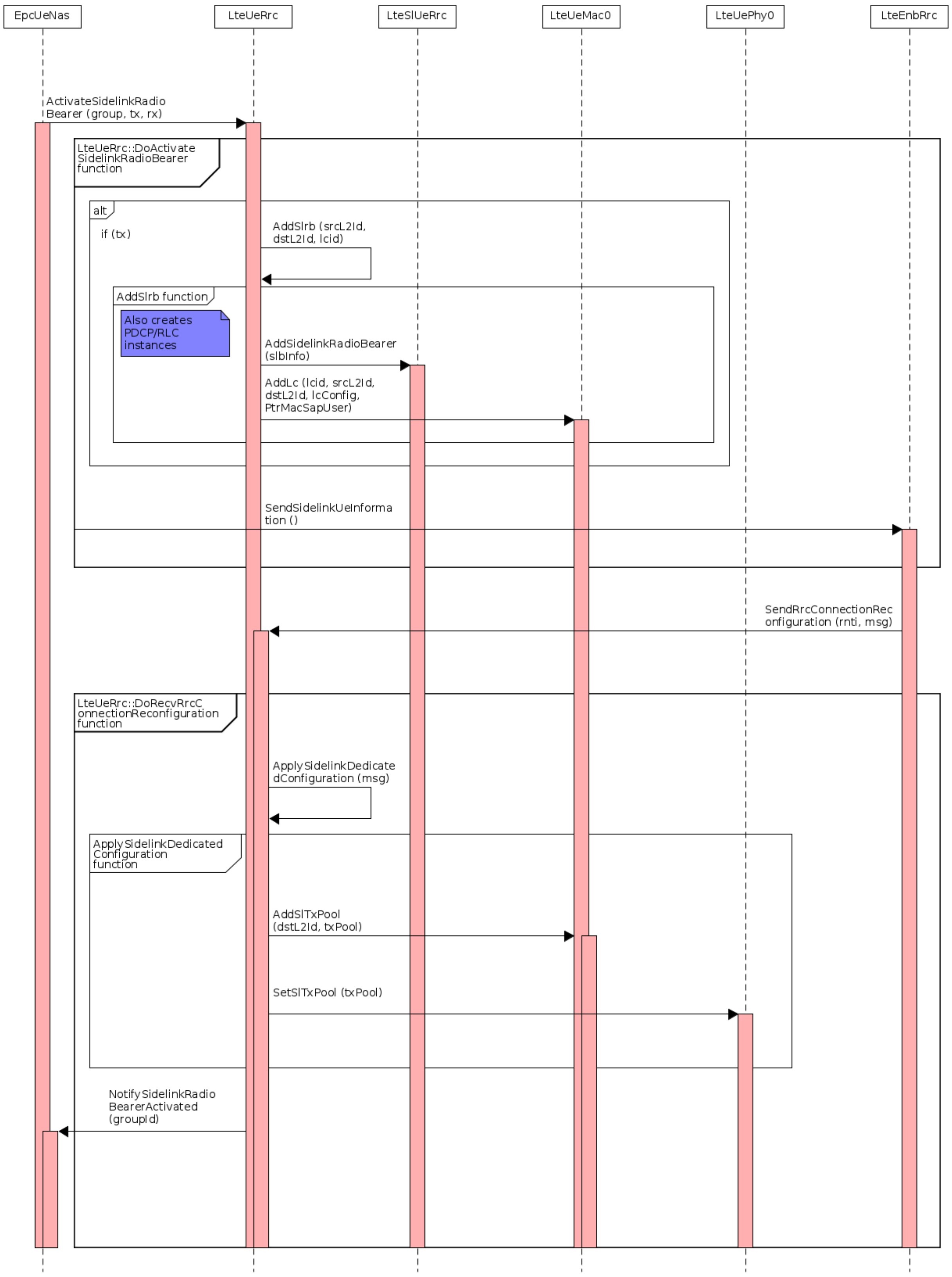

EpcUeNas
LteUeRrc
LteSlUeRrc
LteUeMac0
LteUePhy0
LteEnbRrc
ActivateSidelinkRadio Bearer (group, tx, rx)
LteUeRrc::DoActivate SidelinkRadioBearer function
alt
if (tx)
AddSlrb (srcL2Id, dstL2Id, lcid)
AddSlrb function
Also creates PDCP/RLC instances
AddSidelinkRadioBearer (slbInfo)
AddLc (lcid, srcL2Id, dstL2Id, lcConfig, PtrMacSapUser)
SendSidelinkUeInformation ()
SendRrcConnectionReconfiguration (rnti, msg)
LteUeRrc::DoRecvRrcConnectionReconfiguration function
ApplySidelinkDedicatedConfiguration (msg)
ApplySidelinkDedicated Configuration function
AddSlTxPool (dstL2Id, txPool)
SetSlTxPool (txPool)
NotifySidelinkRadio BearerActivated (groupId)

Fig. 2.6: ns-3 LTE Sidelink in-coverage radio bearer activation (Tx)

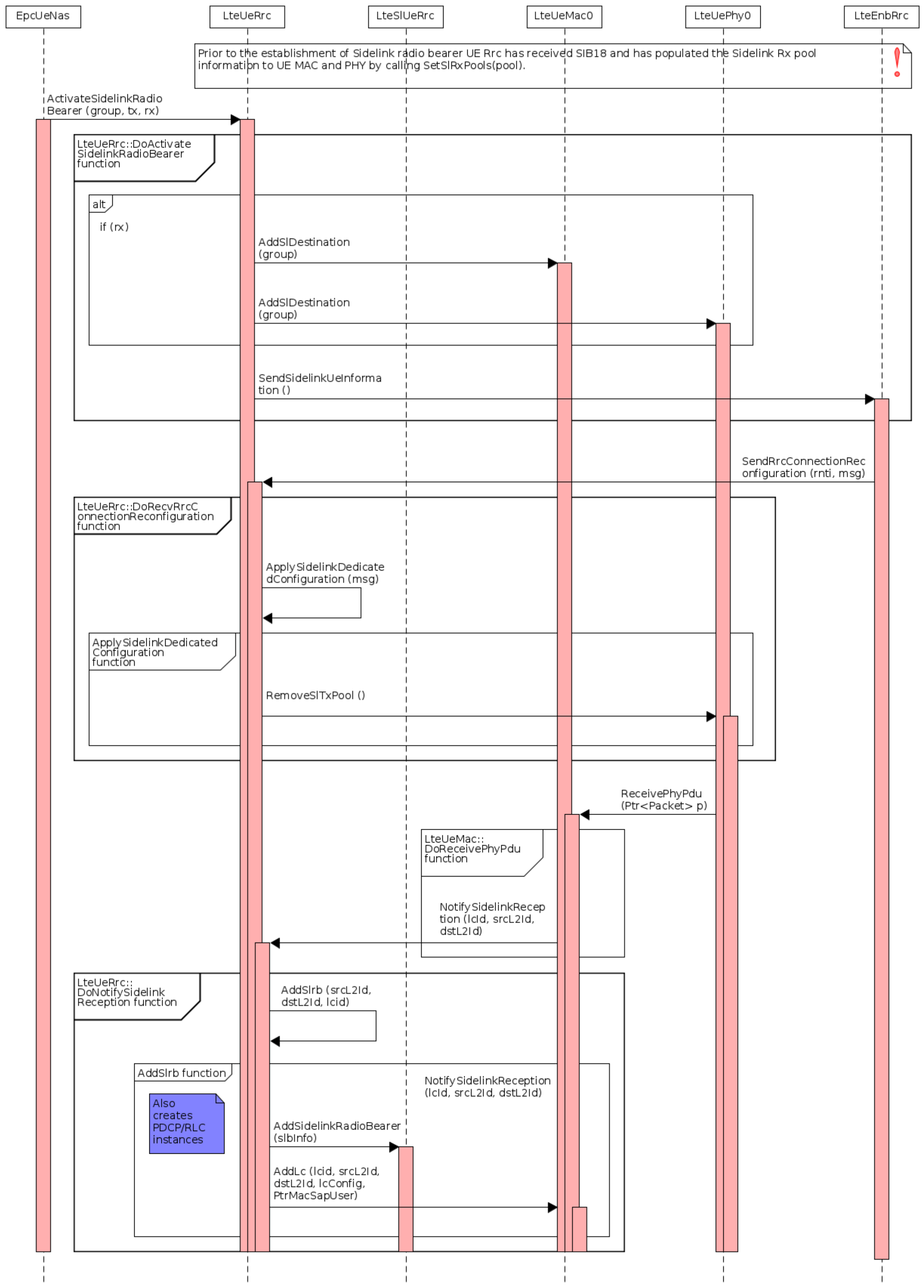

EpcUeNas
LteUeRrc
LteSlUeRrc
LteUeMac0
LteUePhy0
LteEnbRrc
Prior to the establishment of Sidelink radio bearer UE Rrc has received SIB18 and has populated the Sidelink Rx pool information to UE MAC and PHY by calling SetSlRxPools(pool).
ActivateSidelinkRadio Bearer (group, tx, rx)
LteUeRrc::DoActivate SidelinkRadioBearer function
alt
if (rx)
AddSlDestination (group)
AddSlDestination (group)
SendSidelinkUeInformation ()
SendRrcConnectionReconfiguration (rnti, msg)
LteUeRrc::DoRecvRrcConnectionReconfiguration function
ApplySidelinkDedicatedConfiguration (msg)
ApplySidelinkDedicated Configuration function
RemoveSlTxPool ()
ReceivePhyPdu (Ptr<Packet> p)
LteUeMac:: DoReceivePhyPdu function
NotifySidelinkReception (lcId, srcL2Id, dstL2Id)
LteUeRrc:: DoNotifySidelink Reception function
AddSlrb (srcL2Id, dstL2Id, lcid)
AddSlrb function
Also creates PDCP/RLC instances
NotifySidelinkReception (lcId, srcL2Id, dstL2Id)
AddSidelinkRadioBearer (slbInfo)
AddLc (lcid, srcL2Id, dstL2Id, lcConfig, PtrMacSapUser)

Fig. 2.7: ns-3 LTE Sidelink in-coverage radio bearer activation (Rx)

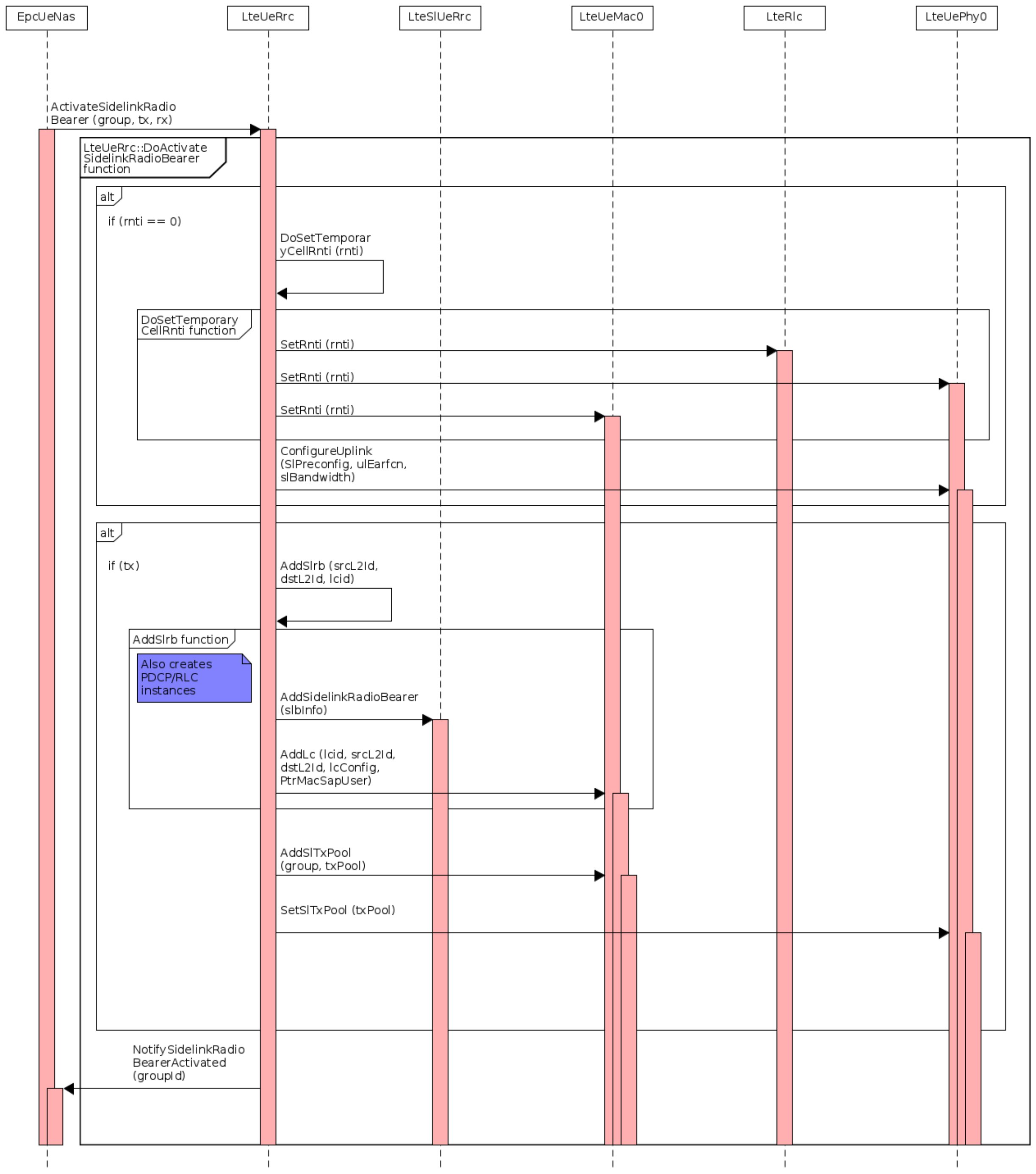

EpcUeNas
LteUeRrc
LteSlUeRrc
LteUeMac0
LteRlc
LteUePhy0
ActivateSidelinkRadio Bearer (group, tx, rx)
LteUeRrc::DoActivate SidelinkRadioBearer function
alt
if (rnti == 0)
DoSetTemporar yCellRnti (rnti)
DoSetTemporary CellRnti function
SetRnti (rnti)
SetRnti (rnti)
SetRnti (rnti)
ConfigureUplink (SlPreconfig, ulEarfcn, slBandwidth)
alt
if (tx)
AddSlrb (srcL2Id, dstL2Id, lcid)
AddSlrb function
Also creates PDCP/RLC instances
AddSidelinkRadioBearer (slbInfo)
AddLc (lcid, srcL2Id, dstL2Id, lcConfig, PtrMacSapUser)
AddSlTxPool (group, txPool)
SetSlTxPool (txPool)
NotifySidelinkRadio BearerActivated (groupId)

Fig. 2.8: ns-3 LTE Sidelink out-of-coverage radio bearer activation (Tx)

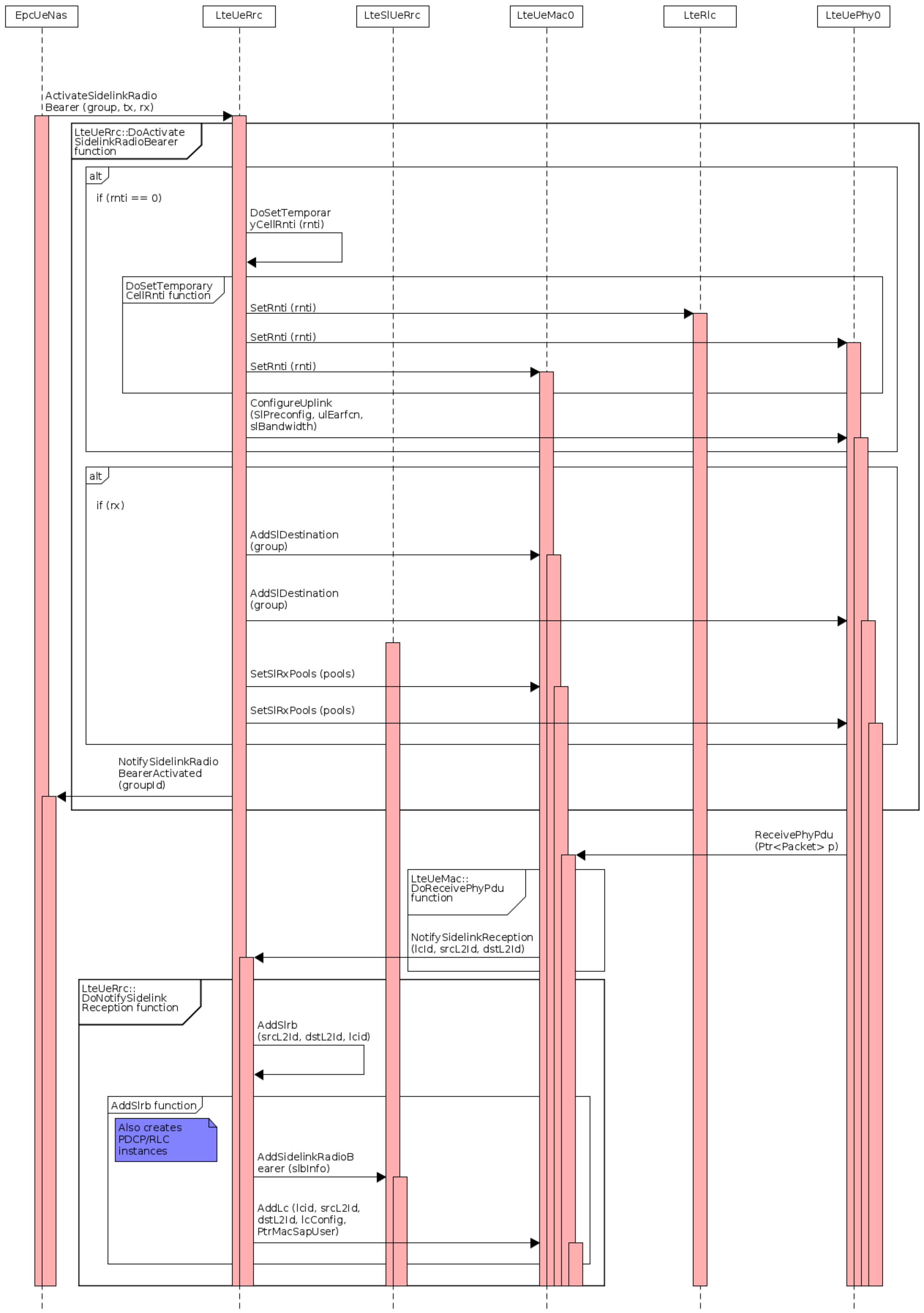

EpcUeNas
LteUeRrc
LteSlUeRrc
LteUeMac0
LteRlc
LteUePhy0
ActivateSidelinkRadio Bearer (group, tx, rx)
LteUeRrc::DoActivate SidelinkRadioBearer function
alt
if (rnti == 0)
DoSetTemporar yCellRnti (rnti)
DoSetTemporary CellRnti function
SetRnti (rnti)
SetRnti (rnti)
SetRnti (rnti)
ConfigureUplink (SlPreconfig, ulEarfcn, slBandwidth)
alt
if (rx)
AddSlDestination (group)
AddSlDestination (group)
SetSlRxPools (pools)
SetSlRxPools (pools)
NotifySidelinkRadio BearerActivated (groupId)
ReceivePhyPdu (Ptr<Packet> p)
LteUeMac:: DoReceivePhyPdu function
NotifySidelinkReception (lcId, srcL2Id, dstL2Id)
LteUeRrc:: DoNotifySidelink Reception function
AddSlrb (srcL2Id, dstL2Id, lcid)
AddSlrb function
Also creates PDCP/RLC instances
AddSidelinkRadioB earer (slbInfo)
AddLc (lcid, srcL2Id, dstL2Id, lcConfig, PtrMacSapUser)

Fig. 2.9: ns-3 LTE Sidelink out-of-coverage radio bearer activation (Rx)

"CONNECTED_NORMALLY" is able to receive SIB18 messages, upon which, Rx pools are updated and also populated to the MAC and PHY layers. On the other hand, if the UE is out-of-coverage it uses a pre-configured pool and MODE 2 for resource allocation.

Sidelink direct discovery

For the Sidelink discovery, `LteUeRrc` class now supports creation/removal of discovery applications for both in-coverage and out-of-coverage scenarios. This procedure triggers the process of conveying the list of applications and the resource pool configuration to the lower layers (Tx or Rx depending on type of discovery application). Similar to the direct communication, for in-coverage scenario, this process triggers the communication with the eNB by transmitting the `SidelinkUeInformation` message for resource allocation, followed by the eNB `RrcConnectionReconfiguration` message. An in-coverage UE only transmits `SidelinkUeInformation` if it has received SIB19 message. We note that, for in-coverage scenario, "Scheduled" mode is not supported yet. The model supports only `Type1`, i.e., "UeSelected" resource allocation. An in-coverage UE can receive a list of multiple pools (only 1 pool is supported now) with the criteria of selection either `RANDOM` or `RSRSPBased` via the `RrcConnectionReconfiguration` message. A UE in "IDLE_CAMPED_NORMALLY" or in "CONNECTED_NORMALLY" is able to receive SIB19 messages, upon which, Rx pools are updated and also populated to the MAC and PHY layers. On the other hand, in out-of-coverage scenarios, the UE only uses the pre-configured pool.

Note: At this stage only one pool is supported

Sidelink synchronization

The `LteUeRrc` class holds the main implementation for performing Sidelink synchronization. In particular, it supports

- Activation/Deactivation of Sidelink Synchronization Signal (SLSS)
- The configuration of the SLSS
- Delivering of the configured SLSS to phy layer for transmission
- Reception and L3 filtering of the SLSS from other UEs
- Selection of a Synchronization Reference UE (SyncRef)
- Notification of SyncRef change to the lower layers

By setting the attribute `UeSlssTransmissionEnabled == true` enables the transmission of SLSS. The purpose of synchronization process is to align the frame and subframe number of a UE with a SyncRef. An SLSS from a SyncRef is considered detectable if;

$$SRSRP_{strongestSyncRef}[dBm] - MinSrsrp[dBm] > syncRefMinHyst[dB] \tag{2.1}$$

$MinSrsrp$ is the minimum threshold for detecting SLSS configured using `MinSrsrp` attribute of `LteUeRrc` class and $syncRefMinHyst$ is the pre-configured hysteresis value, which could be configured from the user's simulation script. In case more than one SyncRefs are detected during a scanning period (Refer to the UE phy section below for details), a SyncRef with the highest Sidelink RSRP (S-RSRP) is selected. Upon re-selection of the SyncRef, if a better SyncRef is detected the UE selects this new SyncRef if;

$$SRSRP_{NewStrongestSyncRef}[dBm] - SRSRP_{OldStrongestSyncRef}[dBm] > syncRefDiffHyst[dB]$$

where $syncRefDiffHyst$ is a threshold representing how higher the S-RSRP of the newly detected SyncRef should be than the currently selected SyncRef's S-RSRP to consider a change. On the other hand, if the S-RSRP of the previously selected SyncRef satisfies the (2.1) and its S-RSRP is greater than a pre-configured $syncTxThreshOoC$ (i.e. out-of-coverage synchronization threshold) [TS36331] the SyncRef is considered as valid till the next re-selection.

In case, the previously selected SyncRef is not valid anymore and no other SyncRef has been detected, the UE itself becomes a SyncRef by choosing SLSS-ID and the Sidelink synchronization offset indicator. Different from the standard [TS36331], the current implementation for the sake of simplicity selects a SLSS-ID as `SLSS-ID = IMSI * 10`, while the synchronization offset indicator, as per the standard, is randomly selected between the two pre-configured offsets, i.e., `syncOffsetIndicator1` and `syncOffsetIndicator2`. We note that, when a Synchronization process is followed by ProSe group communication or discovery, a UE could only be able to receive a message from other UE, if they have same SLSS-ID and the group id is known to the receiving UE. However, when simulating only ProSe communication or discovery the SLSS-ID of all the UEs are initialized to 0 and the UE receives the transmission if the group id is known. Moreover, the transmission of the SLSS is subjective to the data transmission by the UE. At the beginning of every Sidelink Control (SC) period, a notification by the UE MAC layer is sent to the UE RRC, indicating if there is data to be transmitted. The function `DoNotifyMacHasSlDataToSend` and `DoNotifyMacHasNoSlDataToSend` of `LteUeRrc` class serve this purpose. If there is no data to be transmitted the UE stops transmitting the SLSS.

PDCP

The `LtePdcp` class has been extended to include ProSe source Layer 2 ID and the ProSe Layer-2 Group ID to identify the PDCP/RLC pair to be used in the receiving UE. The reason is that in Sidelink, each radio bearer is associated with one PDCP entity. And, each PDCP entity is associated to one RLC entity, since it uses RLC-UM mode for the transmission / reception [TS36323]. Thus, given the nature of Sidelink communication, i.e., one to many, it may happen that multiple UEs transmit to a single UE, assigning LCIDs independently. It may happen that two UEs select same LCID for the same group. Therefore, for a receiving UE to distinguish between two Sidelink Radio bearers (SLRBs) with the similar LCIDs requires additional identifiers.

RLC

Similar to the PDCP layer, the RLC layer now has ProSe source Layer 2 ID and the ProSe Layer-2 Group ID. This is achieved by extending the structure of `TransmitPduParameters` of `LteMacSapProvider` class. As mentioned earlier, only RLC-UM mode is used for Sidelink. Moreover, `LteRlcUm` class is also extended in accordance to the specification define in section 7.1 of [TS36322]. It says that "For RLC entity configured for Sidelink Traffic Channel (STCH), the state variables `VR(UR)` and `VR(UH)` are initially set to the SN of the first received PDU". A new attribute named "ChannelType" is introduced, which is set to type `STCH` for Sidelink.

MAC

Among the following eNB and UE specific enhancements for ProSe, one of the important change, irrespective of the eNB or UE implementation, is the limitation to the maximum frame number, which is now set to 1024. This is because all the computations to determine correctly the boundaries of ProSe communication and discovery periods and their resource allocation considers the maximum frame number to be 1024. This limitation also helps us to implement the rollover constraints more accurately, e.g., to have a configurable offset to be applied at the beginning of each communication and discovery periods. This change mainly impacted all the schedulers. In particular, every scheduler maintains a map of uplink resource allocation using the SFnSF (i.e., combination of frame and subframe number) as an unique key to update the PUSCH based CQIs. By limiting the frame number to 1024, means that a combination could repeat after 10 sec of simulation, thus, a c++ map would not allow to insert new allocation information with the same SFnSF combination. To handle this, we appropriately remove any old allocation information before inserting a new one.

eNB Mac

`LteEnbMac` has been extended to support in-coverage ProSe communication, i.e., Mode 1 resource allocation. The SAP between the `LteEnbRrc` and `LteEnbMac` is extended to receive the pool configuration to be conveyed to the scheduler to schedule the resources. If the type of resource allocation for the configured resource pool is `Scheduled` the eNB MAC is now capable of receiving Sidelink Buffer status Report (SLBSR) from the UEs. To achieve this, the `structure MacCeListElement_s` in `ff-mac-common.h` file is extended to support BSR of type SLBSR. The reporting of the BSR in uplink is the same as shown in Figure fig-ca-uplink-bsr. To handle the scheduling of Sidelink resources for MODE 1, a scheduler `RrSlFfMacScheduler` based on the existing Round Robin implementation is also provided. Thus, making it the only scheduler, for the time being, in LTE module capable of supporting Sidelink and normal LTE resource scheduling. It also makes sure that Sidelink and the uplink resources are not scheduled in the same subframe.

For Sidelink discovery, as stated before, the "Scheduled" mode is not implemented yet.

UE Mac

The `LteUeMac` is extended to support all the three ProSe services. In the following we explain these extensions in context of each ProSe service.

Sidelink direct communication

The UE MAC is now capable of receiving the RLC buffer status notifications with extended Sidelink identifiers, i.e., laye 2 source and destination ids. A tuple `SidelinkLcIdentifier` of `LteUeMac` holds the LCID, SrsL2Id and DstL2Id, which is then used as a key for a c++ map storing all the buffer status notifications from RLC layer. For "Scheduled" resources, i.e., MODE 1, a UE is now capable of processing the Sidelink DCI (i.e., DCI 5 [TS36212]) message containing the resource allocation information for PSCCH and PSSCH transmissions. A UE for transmitting PSCCH uses the PSCCH resource indicated in the SL DCI from the eNB. To ensure the reliability of Sidelink Control information (SCI), each control message is transmitted twice using two different subframe, each utilizing 1 RB belonging to the configured resource block pool [TS36213]. Thus, supporting 2 PSCCH transmissions per grant. While the PSSCH (i.e. Data) transmissions are computed asper the current frame and subframe number and the information including in the SL DCI, i.e., Time Resource Pattern Index (iTrp), Starting RB index and the total number of RBs to be used. Since there are no HARQ feedbacks in Sidelink PSSCH, each transport block is transmitted using 4 HARQ transmissions (i.e., RV = 4). Hence, for each grant every transport block is transmitted using 4 subframes.

On the other hand, for the "UeSelected" resource scheduling, i.e., MODE 2, `LteUeMac` now have following new attributes:

- Ktrp
- SetTrpIndex
- SlGrantMcs
- SlGrantSize

Where Ktrp and SetTrpIndex are used to specify the number of active subframes and the index of the TRP respectively [TS36213]. The last two attributes are used to specify the MCS and the number of RBs (per subframe) to be used for Sidelink transmissions. Different from the "Scheduled" mode, in "UeSelected" mode a UE chooses a random PSCCH resource among the available resources and similarly transmits twice using two subframes. While for transmitting PSSCH it uses a pre-configured pool configuration to determine RBs and subframe for 4 PSSCH transmissions. Moreover, upon the reception of a PHY PDU the `LteUeMac` class is now able to notify UE RRC to establish a Sidelink radio bearer for the reception.

Sidelink direct discovery

In case of direct discovery, the `LteUeMac` class is responsible of creating and scheduling the discovery messages. It is `LteUeMac` class where ProSe APP code is assigned to each announcing discovery application and then is embedded in the discovery messages. We note that, the discovery messages are modeled as control messages. Each discovery message can be retransmitted up to 3 times. The number of retransmissions is provided by the eNB with the pool configuration if the UE is in-coverage, or is specified by the pre-configured pool configuration for out-of-covergae scenarios. Each discovery message occupies 2 contiguous RBs in a subframe belonging to the assigned pool.

Sidelink synchronization

For the Sidelink synchronization, `LteUeMac` receives notification from `LteUePhy` upon change of time, i.e, change of frame and subframe number when a new SyncRef is selected.

PHY

eNB PHY

The LTE module of *ns-3* at the time of writing this documentation does not support Radio Link Failure (RLF) functionality. Therefore, to simulate out-of-coverage scenarios a workaround has been implemented to disable eNB PHY layer. An eNB can be disable by configuring the attribute "DisableEnbPhy" of `LteHelper`.

UE PHY

As shown in Figure *PHY and channel model architecture for the UE*, to receive the Sidelink transmission in the uplink channel the `LteUePhy` includes an additional `LteSpectrumPhy` instance. In particular, the function `DoConfigureUplink`, which is called by `LteUeRrc` is responsible of adding this new instance to the channel. The `LteUePhy` class is extensively edited to support all the three ProSe services.

Sidelink direct communication

The `LteUePhy` is now capable of receiving and transmitting Sidelink control messages. In particular, DCI format 5, and SCI format 0 is now supported [TS36212]. The UE PHY receives the information related to the Sidelink communication pool from the UE RRC in order to compute the boundaries of an SC period to correctly map the frame/subframe and the RBs associated with the PSCCH and PSSCH. At the beginning of each subframe, it checks if the Sidelink transmission pool is added, it make sure to initialize it in order to compute exactly the frame and subframe number of the next SC period. This is needed to remove the Sidelink transmission grant associated to the previous SC period, since a grant is valid only for one SC period.

Sidelink direct discovery

For the discovery, the `LteUePhy` now also supports the transmission and reception of the discovery messages. The discovery message structure is implemented as per *[TS24334]* with two additional identifiers, i.e., UE RNTI and the PSDCH resource number. We note that, the discovery message is modeled as a broadcast control message, thus, is received by every UE. Similar to the direct communication, the UE PHY receives the information related to the Sidelink discovery pool from the UE RRC in order to compute the boundaries of a discovery period to correctly map the frame/subframe and the RBs associated with the PSDCH. At the beginning of each subframe, it checks if the discovery transmission pool is added, it make sure to initialize it in order to compute exactly the frame and subframe number of the next discovery period. A UE upon the reception of a discovery message, simply passes it to the MAC layer, which in turn delivers it to the RRC for further filtering.

Sidelink synchronization

The `LteUePhy` is extended to support Sidelink synchronization functionality. In particular, it supports:

- Transmission and reception of SLSS
- Scheduling of SLSS scanning periods
- Measuring S-RSRP and reporting it to the RRC layer
- Synchronizing to the chosen SyncRef
- Performing change of timing
- Scheduling of the SyncRef re-selection process

In the current implementation, all the UEs by default are perfectly synchronized, i.e., all the UEs in a simulation upon being initialized pick the same frame and subframe 1 to start with. Therefore, to simulate synchronization and to make every UE to pick a random frame and subframe number a new attribute "UeRandomInitialSubframeIndication" is introduced. We note that, in our model a UE being synchronized to a SyncRef refers to a state, where both the UEs have similar SLSS-ID and are aligned on frame and subframe level. The SLSS is represented by MIB-SL, which is modeled as a broadcast control message. MIB-SL is implemented as per the standard, [TS36331], with additional metadata fields: SLSSID, MIB-SL creation time, MIB-SL reception time and the reception offset. The S-RSRP of the SLSS received by a UE is computed as explained in the section sec-phy-ue-measurements. Only those SLSS are considered detected, which have the S-RSRP greater than the configured minimum S-RSRP and for which MIB-SL has been decoded successfully. The minimum S-RSRP is configured through the attribute "MinSrsrp". The modeled synchronization process can be categorized by three sequential processes:

1. Scanning
2. Measurement
3. Evaluation

To tailor these processes, the following new attributes have been introduced in the `LteUePhy` class.

- UeSlssScanningPeriod
- UeSlssInterScanningPeriodMin
- UeSlssInterScanningPeriodMax
- UeSlssMeasurementPeriod
- UeSlssEvaluationPeriod
- NSamplesSrsrpMeas

For more information about these attributes the reader is referred to the API of `LteUePhy` class. After the scanning process, only 6 detected SycRefs with the highest S-RSRP are measured during the measurement period. Moreover, if a UE receives multiple SLSS from different UEs but have the same SLSS-ID and the reception offset, they are considered as different S-RSRP samples of the same SyncRef. The start of the measurement and evaluation processes are subject to the detection of at least 1 SynchRef during the scanning process. At least one SyncRef selection process within 20 seconds is scheduled as per the standard [TS36331]. The MIB-SL is transmitted with a fixed periodicity of 40 ms [TS36331]. The `LteUeRrc` class is responsible of scheduling and delivering the MIB-SL to `LteUePhy` class. Once a suitable SyncRef is selected, the change of timing is performed upon subframe indication, i.e., before the next subframe to avoid any miss alignments. We also note that the transmission of a MIB-SL has priority over the transmission of SCI, if they are scheduled in the same subframe. On the other hand, an MIB-SL can not be transmitted if a SyncRef scanning or measurement process is in progress. The `LteUeMac` and `LteUeRrc` class are notified about the successful change of timings using the existing SAP interfaces, i.e., `LteUeCphySapUser` and `LteUePhySapUser` respectively.

LteSpectrumPhy

As mentioned before, the `LteUePhy` class includes an additional `LteSpectrumPhy` instance to receive Sidelink transmissions from other UEs. Therefore, new functions are implemented in `LteSpectrumPhy` class to relay the received Sidelink transmission to the `LteUePhy` class. To model the half duplex for Sidelink, since it is difficult for the UE to receive and transmit at the same time using the same frequency, a new attribute "CtrlFullDuplexEnabled" is introduced. This attribute is "false" by default, thus, all the Sidelink simulations are performed using half duplex mode. To notify `LteUePhy` about the reception of SLSS a new function `SetLtePhyRxSlssCallback` is implemented, which is hooked to the `ReceiveSlss` fucntion of `LteUePhy` through a callback, while installing UE devices in the `LteHelper` class. For other Sidelink signals, i.e., control and data the already implemented callbacks are utilized.

The `LteSpectrumPhy` class is still responsible for evaluating the TB BLER, however, with the introduction of new physical Sidelink channels, i.e., PSCCH, PSSCH, PSDCH and PSBCH a new physical error model `LteNistErrorModel` is implemented. This error model uses the BLER vs SNR curves for LSM from *[NISTBLERD2D]*, for all the Sidelink physical channels. In particular, these curves are obtained by extending the LTE toolbox in Matlab and performing Monte Carlo simulations by considering AWGN channel and SISO mode for transmission. In the following, we plot these curves for each Sidelink physical channel.

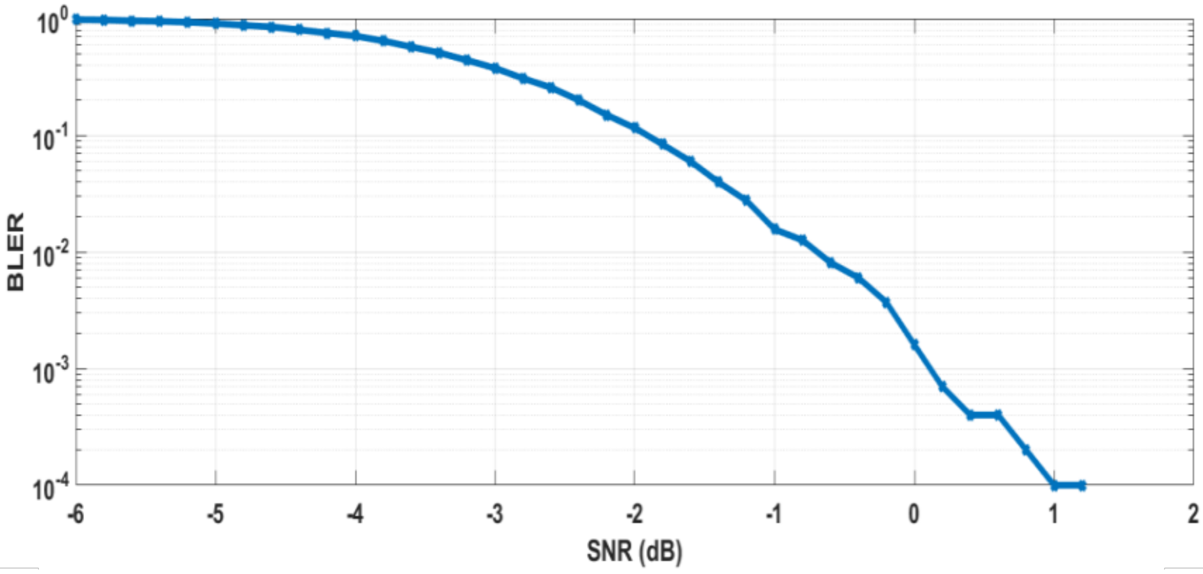


Fig. 2.10: BLER vs SNR of PSCCH (First transmission)

Different from the already existing error models, e.g., `LteMiErrorModel`, which uses the Mutual Information Based Effective SINR (MIESM) technique for soft combining process sec-data-phy-error-model, the `LteNistErrorModel` uses a weighted averaging algorithm explained in *[NISTBLERD2D]* for this purpose. Therefore, a new class `LteSlHarq` is implemented to store the SINR values of each Sidelink transmission that is used for soft combining process of the retransmission. It is to be noted that, as per the standard, no HARQ feedback is available for any Sidelink physical channels. Moreover, to handle interference in D2D scenarios, since all the Sidelink physical channels employ broadcast solution, a new interference model is implemented. This is needed because unlike LTE implementation in which unwanted signals are filtered on the basis of a cell id embedded in transmitted/received `LteSpectrumSignalParameters`, for D2D scenarios two signals received at the same time could be intended for a single UE. This overlapping of signal could occur in out-of-coverage scenarios, where two UEs pick an identical resource to transmit. To tackle such conditions two new classes, `LteSlInterference` and `LteSlChunkProcessor` are implemented.

The `LteSpectrumPhy` class, at the time of reception maintains a vector to store the information about the received signal(s). These signals including the overlapping ones (i.e., they occur at the same time and have equal duration)

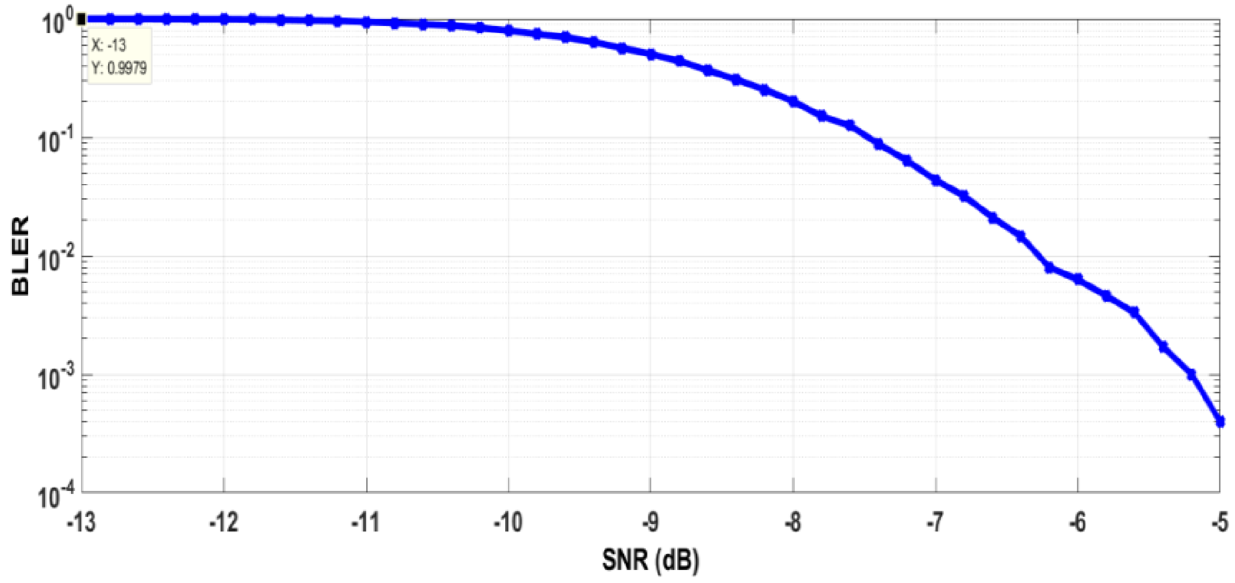


Fig. 2.11: BLER vs SNR of PSBCH

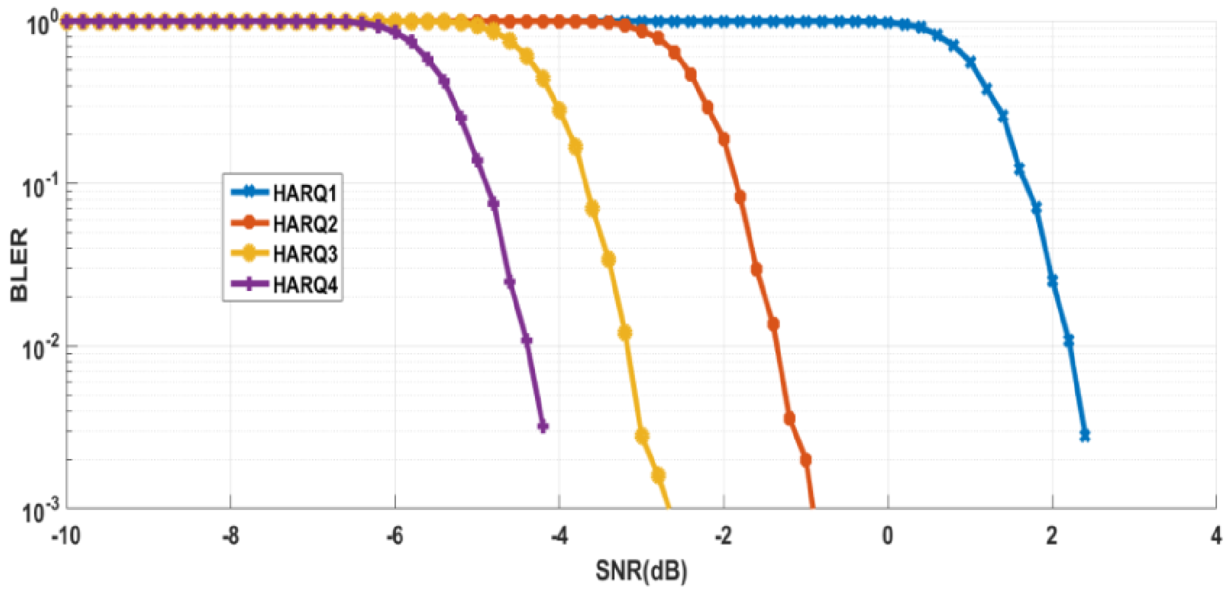


Fig. 2.12: BLER vs SNR of PSDCH

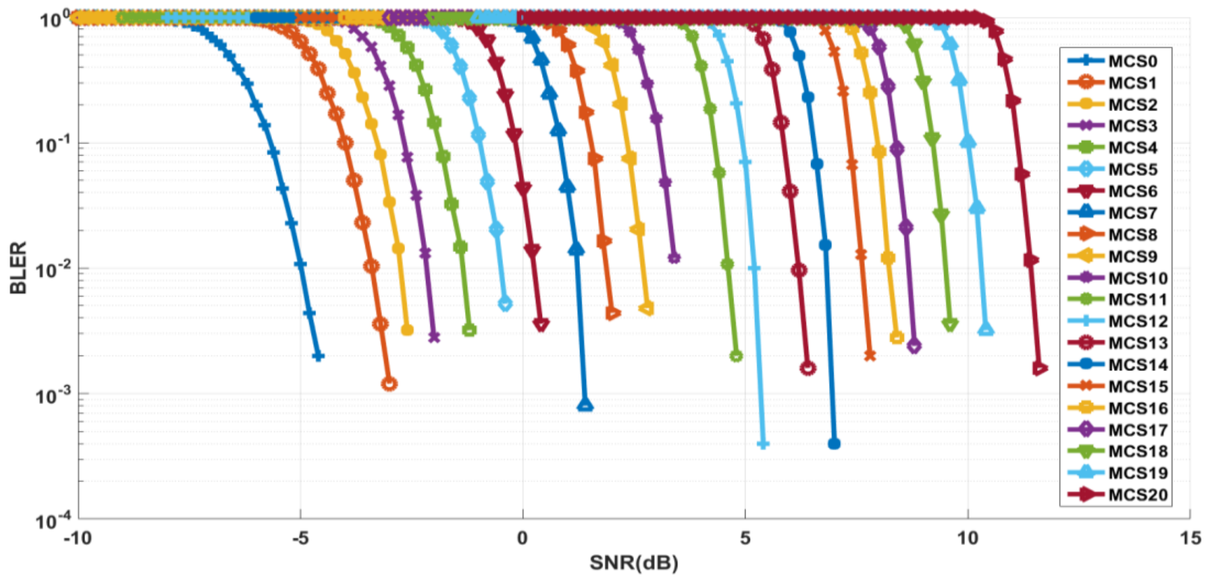


Fig. 2.13: BLER vs SNR of PSSCH (HARQ 1)

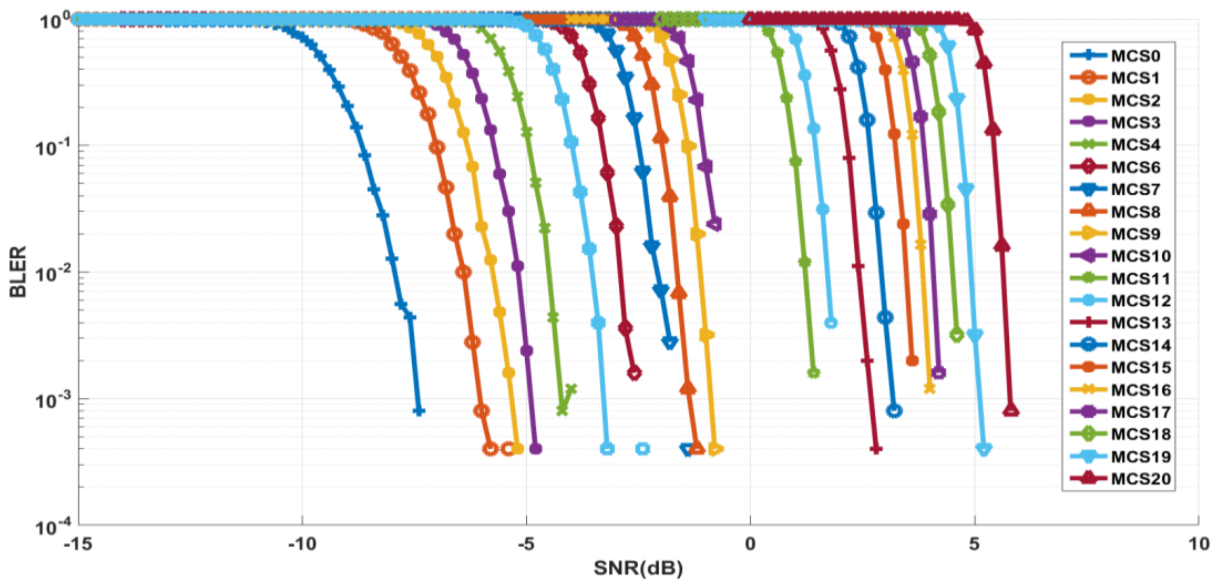


Fig. 2.14: BLER vs SNR of PSSCH (HARQ 2)

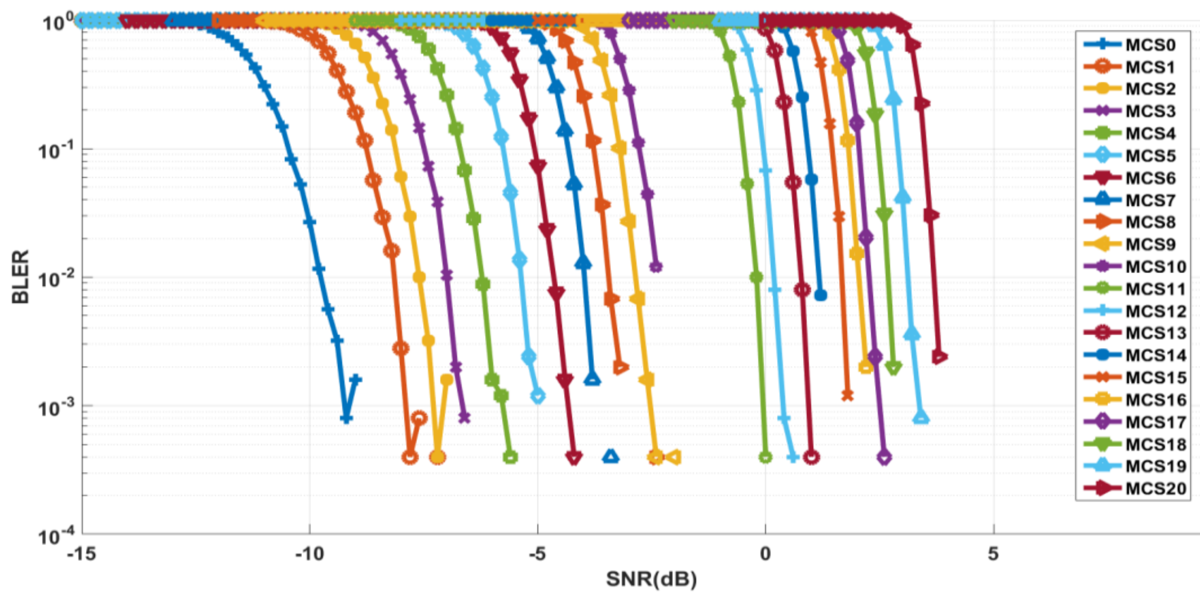


Fig. 2.15: BLER vs SNR of PSSCH (HARQ 3)

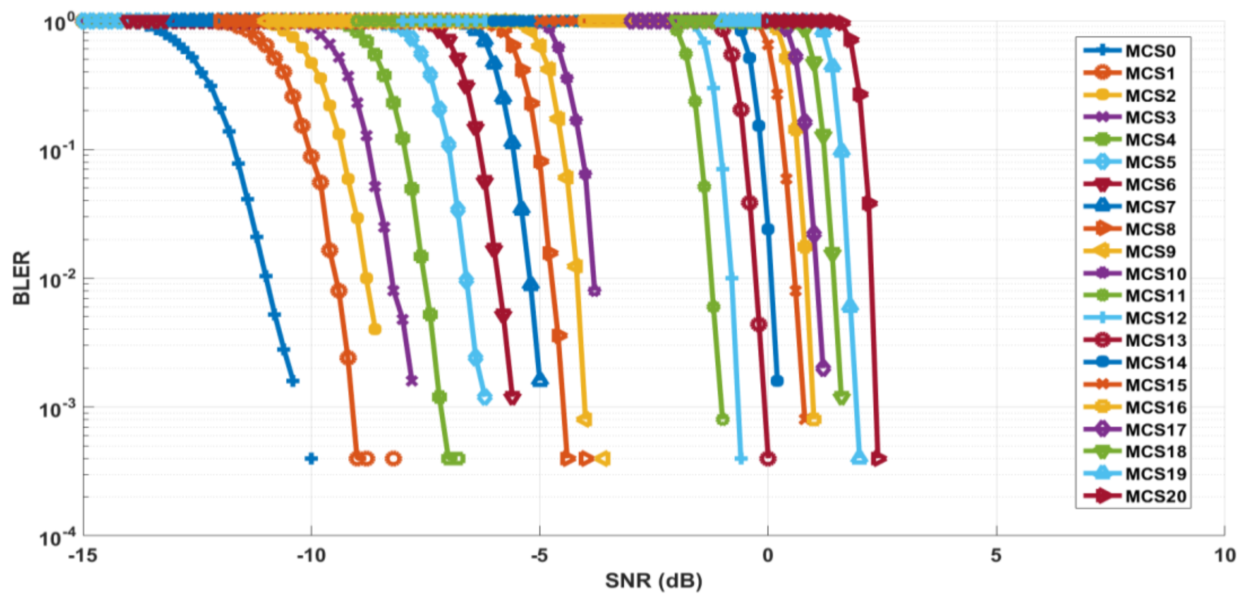


Fig. 2.16: BLER vs SNR of PSSCH (HARQ 4)

are passed to `LteSlInterference` class, which maintains the indexing of each signal by storing them into a vector. Once the signal(s) duration is elapsed, a call to `ConditionallyEvaluateChunk` function is invoked. This function iterates over this vector and computes the perceived SINR, interference, and the power of each signal, which are then passed to the respective `LteSlChunkProcessor` instance along with their signal id. Similarly, each `LteSlChunkProcessor` instance maintain its assigned values by storing them in a vector as per the signal id. For example, an `LteSlChunkProcessor` initialized to store the computed SINR values will maintain a vector containing the SINR of each overlapping signal as per their signal id. In the end, each vector is passed to the respective function, which are hooked through a callback in `LteHelper` class at the time of installing the UE device. For instance, the `UpdateSlSinrPerceived` function of `LteSpectrumPhy` receive the vector of perceived SINR values.

Based on the BLER computation in `LteSpectrumPhy`, the messages received on Sidelink physical channels can be divided into three types.

1. The messages whose BLER computation is performed without HARQ, e.g., SCI and MIB-SL
2. The messages whose BLER computation is performed with HARQ and follow the control information, e.g., Sidelink data
3. The messages whose BLER computation is performed with HARQ but do not have the control information, e.g., Discovery messages

For first type of messages, we compute the average SINR per RB for each received TB and sort them in the descending order of the SINR. Then, we try to decode them by computing the BLER with the help of `LteNistErrorModel`. For the second type of signals, we have the transport block information available in `m_expectedSlTbs` given by `LteUePhy` class. With this information, we perform a TB to SINR index mapping and retrieve the SINR of the expected TB from the perceived SINR vector. Then, this SINR and the HARQ info (if it is the retransmission) along with other parameters are used to compute the BLER of this message. Finally, the third type of messages, similar to the first type, are decoded by first storing them in desending order of the SINR, since we do not have the prior information about the TB. The only difference, is that the discovery messages can be retransmitted up to 3 times, therefore, we maintain the HARQ history of each transmission and use it along with other parameters for computing the BLER. We also note that for type 1 and type 3 messages, if the received TBs collide, i.e., they use the same RBs, the UE will try to decode the sorted TBs one at a time, and if any of the TB is decoded the remaining TB(s) are marked corrupted, thus, are not received by the UE. Alternatively, by setting the `DropRbOnCollisionEnabled` attribute all the colliding TBs can be dropped irrespective of their perceived SINR.

2.1.2 Frequency Hopping

The D2D model also supports the frequency hopping on Sidelink PSSCH for the “UeSelected” resource scheduling, i.e., MODE 2. At the time of writing this documentation, only the *inter-subframe* hopping mode, with constant (i.e., Type 1) and pseudo-random (Type 2) is supported.

Note: The documentation for this section will be extended in the later release of the D2D code. Users, interested to gain further information are referred to *[NISTFREQHOPP]*

2.1.3 Helpers

Four helper objects are used to setup simulations and configure the various components. These objects are:

- `LteHelper`, which takes care of the configuration of the LTE/Sidelink radio access network, as well as of coordinating the setup and release of EPS and Sidelink radio bearers and start/stop ProSe discovery. The `LteHelper` class provides both the API definition and its implementation.
- `EpcHelper`, which takes care of the configuration of the Evolved Packet Core. The `EpcHelper` class is an abstract base class, which only provides the API definition; the implementation is delegated to the child classes in order to allow for different EPC network models.

- `CcHelper`, which takes care of the configuration of the `LteEnbComponentCarrierMap`, basically, it creates a user specified number of `LteEnbComponentCarrier`. `LteUeComponentCarrierMap` is currently created starting from the `LteEnbComponentCarrierMap`. `LteHelper:InstallSingleUeDevice`, in this implementation, is needed to invoke after the `LteHelper:InstallSingleEnbDevice` to ensure that the `LteEnbComponentCarrierMap` is properly initialized.
- `LteSidelinkHelper`, this helper class is provided to ease the burden of a user to configure public safety scenarios involving Sidelink. In particular, it uses other helper classes, e.g., LteHelper to activate/deactivate Sidelink bearers, create groups for Sidelink broadcast or groupcast communication and Lte3gppHexGridEnbTopologyHelper to associate UEs to a Sidelink group or an eNB using wrap around method.

Note: Wrap-around functionality is not fully supported yet.

It is possible to create a simple LTE-only simulations by using the `LteHelper` alone, or to create complete LTE-EPC simulations by using both `LteHelper` and `EpcHelper`. When both helpers are used, they interact in a master-slave fashion, with the `LteHelper` being the Master that interacts directly with the user program, and the `EpcHelper` working "under the hood" to configure the EPC upon explicit methods called by the `LteHelper`. The exact interactions are displayed in the Figure *Sequence diagram of the interaction between LteHelper and EpcHelper.*.

2.2 User Documentation

Similarly, for the Sidelink the LTE model currently supports the output to file of PHY and MAC level KPIs. You can enable them in the following way:

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();

// configure all the simulation scenario here...

lteHelper->EnableSlPscchMacTraces ();
lteHelper->EnableSlPsschMacTraces ();

lteHelper->EnableSlRxPhyTraces ();
lteHelper->EnableSlPscchRxPhyTraces ();

Simulator::Run ();
```

The Sidelink MAC level KPIs are gathered at UE MAC and are written into two files, first one for PSCCH and the second for PSSCH. These KPIs are basically the trace of resource allocation for the Sidelink control and data transmissions for every SC period in MODE 1 and MODE 2. For PSCCH KPIs the format is the following:

1. Simulation time in milliseconds at which the SC period starts (**Note : PSCCH transmission start time will depend on the PSCCH resource index**)
2. Cell ID
3. unique UE ID (IMSI)
4. Sidelink-specific UE ID (RNTI)
5. Current frame number
6. Current subframe number
7. PSCCH resource index
8. Frame number for the first PSCCH transmission

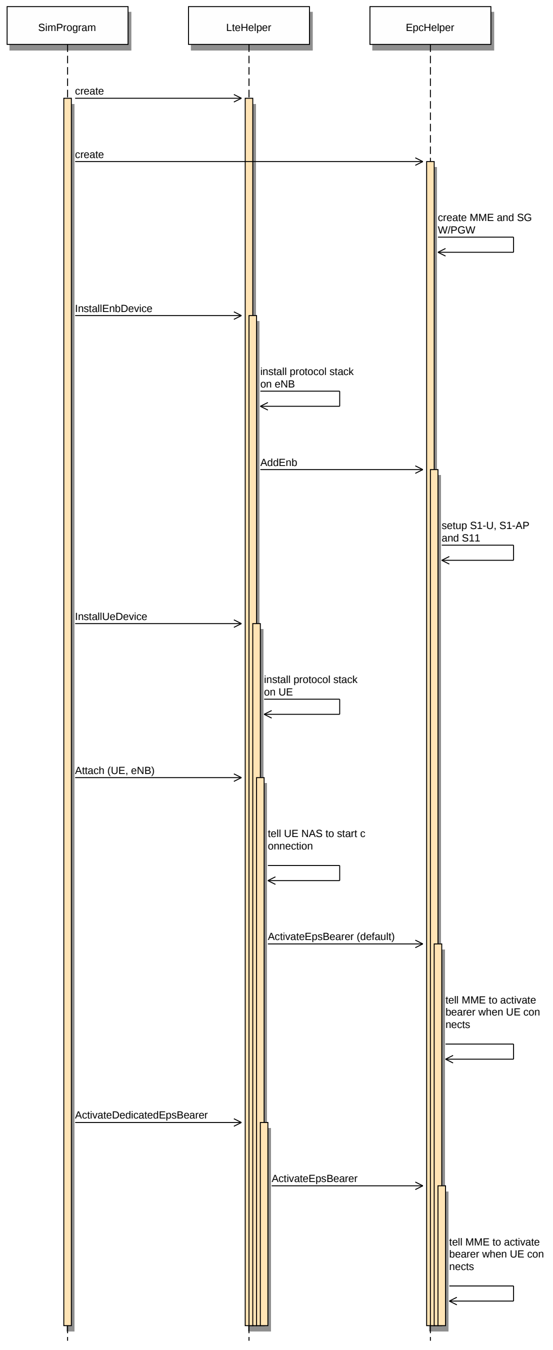

SimProgram
LteHelper
EpcHelper
create
create
create MME and SG W/PGW
InstallEnbDevice
install protocol stack on eNB
AddEnb
setup S1-U, S1-AP and S11
InstallUeDevice
install protocol stack on UE
Attach (UE, eNB)
tell UE NAS to start c onnection
ActivateEpsBearer (default)
tell MME to activate bearer when UE con nects
ActivateDedicatedEpsBearer
ActivateEpsBearer
tell MME to activate bearer when UE con nects

Fig. 2.17: Sequence diagram of the interaction between `LteHelper` and `EpcHelper`.

9. Subframe number for the first PSCCH transmission
10. Frame number for the second PSCCH transmission
11. Subframe number for the second PSCCH transmission
12. MCS
13. Transport block size
14. Index of the first RB used for PSSCH (i.e., Data)
15. Total number of RBs allocated for PSSCH
16. Time Resource Pattern Index (iTRP) used for PSSCH

while for PSSCH MAC KPIs the format is:

1. Simulation time in milliseconds at which the PSSCH transmission starts
2. Cell ID
3. unique UE ID (IMSI)
4. Sidelink-specific UE ID (RNTI)
5. Frame number at which the SC period starts
6. Subframe number at which the SC period starts
7. Frame number at which the PSSCH transmission starts
8. Subframe number at which the PSSCH transmission starts
9. Current frame number
10. Current subframe number
11. MCS
12. Transport block size
13. Index of the first RB used for PSSCH
14. Total number of RBs allocated for PSSCH

The names of the files used for Sidelink MAC KPI output can be customized via the ns-3 attributes `ns3::MacStatsCalculator::SlUeCchOutputFilename` and `ns3::MacStatsCalculator::SlUeSchOutputFilename`.

Sidelink PHY KPIs are distributed in two different files whose names are configurable through the following attributes

1. ns3::PhyRxStatsCalculator::SlPscchRxOutputFilename
2. ns3::PhyRxStatsCalculator::SlRxOutputFilename

In the SlPscchRx file, the following content is available:

1. Simulation time in milliseconds
2. Cell ID (Fixed to 0 for Sidelink)
3. unique UE ID (IMSI)
4. Sidelink-specific UE ID (RNTI)
5. MCS
6. Transport block size in bytes
7. PSCCH resource index

8. Total number of RBs for PSSCH
9. Index of the first RB used for PSSCH
10. Time Resource Pattern Index (iTRP) used for PSSCH
11. Hopping value
12. Layer 2 group destination id
13. Correctness in the reception of the TB (correct = 1 : error = 0)

And Finally, in SlRx file the parameters included are:

1. Simulation time in milliseconds
2. Cell ID (Fixed to 0 for Sidelink)
3. unique UE ID (IMSI)
4. Sidelink-specific UE ID (RNTI) **Note: For PSDCH reception, it is the RNTI of the UE transmitting discovery messages.**
5. Layer of transmission
6. MCS
7. Transport block size in bytes
8. Redundancy version
9. New Data Indicator flag
10. Correctness in the reception of the TB (correct = 1 : error = 0)
11. Average perceived SINR per RB in linear units

Note: This file is used to store the above parameters for both PSSCH and PSDCH reception.

The Sidelink physical error model consists of the data, control, and the discovery error model. All of them are active by default. It is possible to deactivate them with the ns3 attribute system, in detail:

```
Config::SetDefault ("ns3::LteSpectrumPhy::SlCtrlErrorModelEnabled", BooleanValue␣
↪(false));
Config::SetDefault ("ns3::LteSpectrumPhy::SlDataErrorModelEnabled", BooleanValue␣
↪(false));
Config::SetDefault ("ns3::LteSpectrumPhy::SlDiscoveryErrorModelEnabled", BooleanValue␣
↪(false));
```

Besides the error models, one more attribute, i.e., “DropRbOnCollisionEnabled” is also introduced only for the Sidelink transmissions. This is implemented by keeping in mind the scenarios in which the resources are autonomously scheduled by a UE, which increases the probability of two UEs choosing the same RBs to transmit. Therefore, causing a collision between the TBs. By using this attribute, a user can choose to drop such collided TBs. It can be configured as follows:

```
Config::SetDefault ("ns3::LteSpectrumPhy::DropRbOnCollisionEnabled", BooleanValue␣
↪(true));
```

A new directory “d2d-examples” in the LTE module examples directory holds the examples for simulating ProSe services, i.e., direct communication, direct discovery, and synchronization. These examples can be divided into the following two categories:

1. Simple examples

These examples covers in-coverage and out-of-coverage ProSe communication using a simple scenario. The examples included in this category are the following:

- lte-sl-in-covrg-comm-mode1
- lte-sl-in-covrg-comm-mode2
- lte-sl-out-of-covrg-comm

2. Detailed examples

These examples can be used to simulate out-of-coverage ProSe communication, discovery and synchronization. In particular, these examples covers the simulation scenarios used for the study published in *[NIST2017]*. The users interested in the extensive simulation campaigns described in this study, are referred to a "README.txt" file for more information. The examples included in this category are:

- wns3-2017-pscch
- wns3-2017-pssch
- wns3-2017-synch
- wns3-2017-discovery

The Sidelink example scripts follow the same rules as writing an LTE simulation script, however, there are additional configurations required to simulate Sidelink. Following are some rules of thumb for writing Sidelink scripts;

- Sidelink simulations require EPC.
- eNB should be disabled to simulate out-of-coverage scenarios.
- Sidelink pools should be configured before the start of the simulation.
- `Lte3gppHexGridEnbTopologyHelper` can not be use without initializing eNB nodes and appropriate antenna model.
- It is possible to avoid the initialization of eNB nodes in out-of-coverage scenarios, if hexagonal ring topology is not used.

In general, all the D2D examples are highly parameterizable and could be divided in three parts:

1. Configuration of LTE and Sidelink default parameters, e.g., the ones configured by calling `Config::SetDefault` function
2. Topology configuration
3. Sidelink pool configuration

In the following, we will discuss all the examples from the "Simple examples" category and the two examples from "Detailed examples" category, which are wns3-2017-synch and wns3-2017-discovery. We choose wns3-2017-synch, since it also simulates the PSCCH and PSSCH.

This example simulates an in-coverage MODE 1 ProSe communication by using the following scenario.

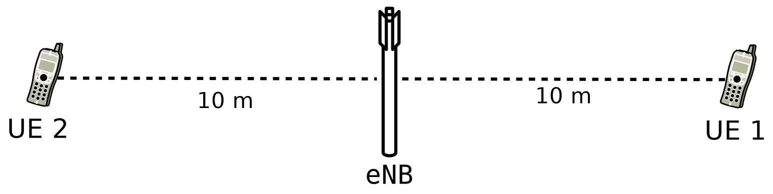


Fig. 2.18: Sidelink simple in-coverage scenario

Both the UEs are in-coverage of the eNB where UE1 sends the data to UE2 via Sidelink by using the resources assigned by the eNB. A user can configure the simulation time and the output of NS logs of the specified classes by using the corresponding global variables in the simulation script. For example, a user can run the simulation as follows:

```
./waf --run "lte-sl-in-covrg-comm-mode1 --simTime=7 --enableNsLogs=false"
```

The simulation time is in seconds. Moreover, this example only supports IPV4 traffic flows.

The simulation script begins with the configuration of the parameters of the Sidelink scheduler as follows.

- Configure the Sidelink scheduler :

```
Config::SetDefault ("ns3::RrSlFfMacScheduler::Itrp", UintegerValue (0));
Config::SetDefault ("ns3::RrSlFfMacScheduler::SlGrantSize", UintegerValue (5));
```

The `ns3::RrSlFfMacScheduler` is a very simple round robin scheduler, which uses a fixed TRP index value and number of RBs to be allocated to a UE.

- Configure the frequency and the bandwidth :

```
Config::SetDefault ("ns3::LteEnbNetDevice::DlEarfcn", UintegerValue (100));
Config::SetDefault ("ns3::LteUeNetDevice::DlEarfcn", UintegerValue (100));
Config::SetDefault ("ns3::LteEnbNetDevice::UlEarfcn", UintegerValue (18100));
Config::SetDefault ("ns3::LteEnbNetDevice::DlBandwidth", UintegerValue (50));
Config::SetDefault ("ns3::LteEnbNetDevice::UlBandwidth", UintegerValue (50));
```

We use the LTE Band 1 for both LTE and Sidelink communication.

- Configure the error models :

```
// For PSSCH
Config::SetDefault ("ns3::LteSpectrumPhy::SlDataErrorModelEnabled",
                                                     BooleanValue (true));

// For PSCCH
Config::SetDefault ("ns3::LteSpectrumPhy::SlCtrlErrorModelEnabled",
                                                     BooleanValue (true));

Config::SetDefault ("ns3::LteSpectrumPhy::DropRbOnCollisionEnabled",
                                                     BooleanValue (false));
```

- Configure the transmit power of the eNB and the UEs :

```
Config::SetDefault ("ns3::LteUePhy::TxPower", DoubleValue (23.0));
Config::SetDefault ("ns3::LteEnbPhy::TxPower", DoubleValue (30.0));
```

The powers configured are in dBm.

- Instantiating LTE, EPC and Sidelink helpers :

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
Ptr<PointToPointEpcHelper>  epcHelper = CreateObject<PointToPointEpcHelper> ();
lteHelper->SetEpcHelper (epcHelper);

Ptr<LteSidelinkHelper> proseHelper = CreateObject<LteSidelinkHelper> ();
proseHelper->SetLteHelper (lteHelper);
```

- Configuring the pathloss model :

```
lteHelper->SetAttribute ("PathlossModel",
                             StringValue ("ns3::Cost231PropagationLossModel"));
```

- Enabling the Sidelink :

```
lteHelper->SetAttribute ("UseSidelink", BooleanValue (true));
```

Note : Attribute ‘‘UseSidelink‘‘ must be set before installing the UE devices.

- Configuring the scheduler :

```
lteHelper->SetSchedulerType ("ns3::RrSlFfMacScheduler");
```

- Creating the eNB and UE nodes and setting their mobility :

```
NodeContainer enbNode;
enbNode.Create (1);
NodeContainer ueNodes;
ueNodes.Create (2);

Ptr<ListPositionAllocator> positionAllocEnb =
                                     CreateObject<ListPositionAllocator> ();
positionAllocEnb->Add (Vector (0.0, 0.0, 30.0));
Ptr<ListPositionAllocator> positionAllocUe1 =
                                     CreateObject<ListPositionAllocator> ();
positionAllocUe1->Add (Vector (10.0, 0.0, 1.5));
Ptr<ListPositionAllocator> positionAllocUe2 =
                                     CreateObject<ListPositionAllocator> ();
positionAllocUe2->Add (Vector (-10.0, 0.0, 1.5));

//Install mobility
MobilityHelper mobilityeNodeB;
mobilityeNodeB.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityeNodeB.SetPositionAllocator (positionAllocEnb);
mobilityeNodeB.Install (enbNode);

MobilityHelper mobilityUe1;
mobilityUe1.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityUe1.SetPositionAllocator (positionAllocUe1);
mobilityUe1.Install (ueNodes.Get (0));

MobilityHelper mobilityUe2;
mobilityUe2.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityUe2.SetPositionAllocator (positionAllocUe2);
mobilityUe2.Install (ueNodes.Get (1));
```

- Installing LTE devices to the nodes :

```
NetDeviceContainer enbDevs = lteHelper->InstallEnbDevice (enbNode);
NetDeviceContainer ueDevs = lteHelper->InstallUeDevice (ueNodes);
```

This example simulates an in-coverage scenario, therefore, the eNB will be configured with a pre-configured dedicated Sidelink pool. As mentioned in *RRC*, in this scenario the `LteSlEnbRrc` class will be responsible for holding this per-configured pool configuration. The pool configuration starts by setting a flag in `LteSlEnbRrc` as an indication that the Sidelink is enabled. It is configured as follows:

```
Ptr<LteSlEnbRrc> enbSidelinkConfiguration = CreateObject<LteSlEnbRrc> ();
enbSidelinkConfiguration->SetSlEnabled (true);
```

For configuring Sidelink parameters, the “setup” structure of the field “commTxResources-r12” of IE “SL-Preconfiguration” defined in the standard [TS36331] is converted into a C++ structure. This example uses this structure to configure the pool parameters for Sidelink control. The pool configuration is done by using the `LteSlPreconfigPoolFactory` in the following manner,

```
LteRrcSap::SlCommTxResourcesSetup pool;

pool.setup = LteRrcSap::SlCommTxResourcesSetup::SCHEDULED;
//BSR timers
pool.scheduled.macMainConfig.periodicBsrTimer.period =
                                             LteRrcSap::PeriodicBsrTimer::sf16;
pool.scheduled.macMainConfig.retxBsrTimer.period = LteRrcSap::RetxBsrTimer::sf640;
//MCS
pool.scheduled.haveMcs = true;
pool.scheduled.mcs = 16;
//resource pool
LteSlResourcePoolFactory pfactory;
pfactory.SetHaveUeSelectedResourceConfig (false); //since we want eNB to schedule

//Control
pfactory.SetControlPeriod("sf40");
pfactory.SetControlBitmap(0x00000000FF); //8 subframes for PSCCH
pfactory.SetControlOffset(0);
pfactory.SetControlPrbNum(22);
pfactory.SetControlPrbStart(0);
pfactory.SetControlPrbEnd(49);

pool.scheduled.commTxConfig = pfactory.CreatePool ();

uint32_t groupL2Address = 255;

enbSidelinkConfiguration->AddPreconfiguredDedicatedPool (groupL2Address, pool);
lteHelper->InstallSidelinkConfiguration (enbDevs, enbSidelinkConfiguration);
```

The resources for data are computed by the scheduler on the basis of the scheduler's attributes configured in the start of this simulation script and the above pool configuration.

Similarly, for the UEs we need to enable the Sidelink in `LteSlUeRrc` by setting a flag and a pre-configuration object, which will be initialized with the pool configurations once the UE receives an `RrcConnectionReconfiguration` message from the eNB, as shown in Figures *ns-3 LTE Sidelink in-coverage radio bearer activation (Tx)* and *ns-3 LTE Sidelink in-coverage radio bearer activation (Rx)*:

```
//pre-configuration for the UEs
Ptr<LteSlUeRrc> ueSidelinkConfiguration = CreateObject<LteSlUeRrc> ();
ueSidelinkConfiguration->SetSlEnabled (true);
LteRrcSap::SlPreconfiguration preconfiguration;
ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);
lteHelper->InstallSidelinkConfiguration (ueDevs, ueSidelinkConfiguration);
```

- Installing the IP stack on the UEs and assigning IP address :

```
//Install the IP stack on the UEs and assign IP address
InternetStackHelper internet;
internet.Install (ueNodes);
Ipv4InterfaceContainer ueIpIface;
ueIpIface = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ueDevs));

// set the default gateway for the UE
Ipv4StaticRoutingHelper ipv4RoutingHelper;
for (uint32_t u = 0; u < ueNodes.GetN (); ++u)
  {
    Ptr<Node> ueNode = ueNodes.Get (u);
    // Set the default gateway for the UE
    Ptr<Ipv4StaticRouting> ueStaticRouting =
                  ipv4RoutingHelper.GetStaticRouting (ueNode->GetObject<Ipv4>␣
↪());
    ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (),␣
↪1);
  }
```

The above configuration is similar to any usual LTE example with EPC.

- Attaching the UEs to the eNB :

```
lteHelper->Attach (ueDevs);
```

- Installing applications and activating Sidelink radio bearers :

```
Ipv4Address groupAddress ("225.0.0.0"); //use multicast address as destination
OnOffHelper sidelinkClient("ns3::UdpSocketFactory",
                          Address ( InetSocketAddress (groupAddress, 8000)));
sidelinkClient.SetConstantRate (DataRate ("16kb/s"), 200);

ApplicationContainer clientApps = sidelinkClient.Install (ueNodes.Get (0));
//onoff application will send the first packet at :
//(2.9 (App Start Time) + (1600 (Pkt size in bits) / 16000 (Data rate)) = 3.0 sec
clientApps.Start (slBearersActivationTime + Seconds (0.9));
clientApps.Stop (simTime - slBearersActivationTime + Seconds (1.0));

ApplicationContainer serverApps;
PacketSinkHelper sidelinkSink ("ns3::UdpSocketFactory",
                      Address (InetSocketAddress (Ipv4Address::GetAny (), 8000)));
serverApps = sidelinkSink.Install (ueNodes.Get (1));
serverApps.Start(Seconds (2.0));
```

In this example, an “OnOff” application is installed in the UE 1, which sends a 200 byte packet with the constant bit rate of 16 kb/s. On the other hand, the UE 2 is configured with the “PacketSink” application.

The Sidelink radio bearers are activated by calling the `ActivateSidelinkBearer` method of `LteSidelinkHelper` as follows:

```
//Set Sidelink bearers
Ptr<LteSlTft> tft = Create<LteSlTft> (LteSlTft::BIDIRECTIONAL,
                                          groupAddress, groupL2Address);
proseHelper->ActivateSidelinkBearer (Seconds (2.0), ueDevs, tft);
```

The `ActivateSidelinkBearer` method takes the following three parameters as input,

1. Time to activate the bearer
2. Devices for which the bearer will be activated
3. The Sidelink traffic flow template

The `tft` in this example is an object of `LteSlTft` class created by initializing its following parameters,

1. Direction of the bearer, i.e., “Transmit”, “Receive” or Bidirectional
2. An IPv4 multicast address of the group
3. Sidelink layer 2 group address (used as Sidelink layer 2 group id)

- Activating the Sidelink traces :

```
lteHelper->EnableSlPscchMacTraces ();
lteHelper->EnableSlPsschMacTraces ();

lteHelper->EnableSlTxPhyTraces ();
lteHelper->EnableSlRxPhyTraces ();
lteHelper->EnableSlPscchRxPhyTraces ();
```

The above code will enable all the Sidelink related traces. We note that, the user can also use the classical function “LteHelper::EnableTraces ()”, but this will also output the LTE traces.

Upon the completion of the simulation, the following trace files can be found in the repository’s root folder,

- SlUeMacStats.txt
- SlSchUeMacStats.txt
- SlPscchRxPhyStats.txt
- SlRxPhyStats.txt

For more information related to the above files please refer to the sec-sidelink-simulation-output section.

- UePacketTrace.tr

  The information in this file is obtained by using the traces “TxWithAddresses” and “RxWithAddresses” of `OnOff` and `PacketSink` application, respectively. Following table shows the snippet of the data from this file for the two UEs,

Table 2.2: UePacketTrace.tr

Time (sec)	tx/rx	NodeID	IMSI	PktSize (bytes)	IP[src]	IP[dst]
3	tx	2	1	200	7.0.0.2:49153	225.0.0.0:8000
3.08893	rx	3	2	200	7.0.0.2:49153	7.0.0.3:8000

The first row shows that the UE 1 with node id 2 and IMSI 1 transmits a multicast packet of 200 bytes and the UE 2 receives the packet transmitted by the UE 1. As per the client’s application data rate and ON time, i.e., 16 kb/s and 2 seconds, respectively, a total of 20 packets are sent and received by the transmitting and the receiving UE.

This example simulates an in-coverage MODE 2 ProSe communication using the same scenario *Sidelink simple in-coverage scenario*, used for previous example. The only difference is that the resources are not scheduled by the eNB, instead, the UE 1 selects the resources autonomously from a pool specified by the eNB through `RrcConnectionReconfiguration` message. Besides using the same scenario, this example also has the same application, therefore, in the following, we will discuss only those configurations, which are MODE 2 specific. A user can run the simulation as follows:

```
./waf --run "lte-sl-in-covrg-comm-mode2 --simTime=7 --enableNsLogs=false"
```

The simulation time is in seconds. Similarly, this example only supports IPV4.

- Configuring parameters for PSSCH resource selection :

```
// Fixed MCS and the number of RBs
Config::SetDefault ("ns3::LteUeMac::SlGrantMcs", UintegerValue (16));
Config::SetDefault ("ns3::LteUeMac::SlGrantSize", UintegerValue (5));

// For selecting subframe indicator bitmap
Config::SetDefault ("ns3::LteUeMac::Ktrp", UintegerValue (1));
//use default Trp index of 0
Config::SetDefault ("ns3::LteUeMac::UseSetTrp", BooleanValue (true));
```

The above parameters of `LteUeMac` class enable the UE to select the time (i.e., frame/subframe) and the frequency (i.e., RBs) resources autonomously. If the attribute "UseSetTrp" is false, a UE will select the TRP index randomly from the range of values depending on the KTRP value [TS36213].

- Configure the error models :

```
// For PSSCH
Config::SetDefault ("ns3::LteSpectrumPhy::SlDataErrorModelEnabled",
                                                      BooleanValue (true));

// For PSCCH
Config::SetDefault ("ns3::LteSpectrumPhy::SlCtrlErrorModelEnabled",
                                                      BooleanValue (true));

Config::SetDefault ("ns3::LteSpectrumPhy::DropRbOnCollisionEnabled",
                                                      BooleanValue (false));
```

Along with the error model configuration, by setting the "DropRbOnCollisionEnabled" attribute all the TBs collided in time and frequency are dropped. This attribute could be of particular interest for the users, e.g., to study the impact of collisions in MODE 2. In this example, this attribute has no impact as only one UE is acting as a transmitter. It is included just to make the users aware of this attribute.

- Configuring the scheduler :

```
There is no need to configure the scheduler, since this example uses MODE 2 for
resource selection.
```

The Sidelink pool configuration is similar to the MODE 1 configuration, e.g., the configuration for Sidelink control is the same. However, the following additional configurations are needed for in-coverage MODE 2.

```
LteRrcSap::SlCommTxResourcesSetup pool;

pool.setup = LteRrcSap::SlCommTxResourcesSetup::UE_SELECTED;
pool.ueSelected.havePoolToRelease = false;
pool.ueSelected.havePoolToAdd = true;
pool.ueSelected.poolToAddModList.nbPools = 1;
pool.ueSelected.poolToAddModList.pools[0].poolIdentity = 1;

//Data
pfactory.SetDataBitmap(0xFFFFFFFFFF);
pfactory.SetDataOffset(8); //After 8 subframes of PSCCH
pfactory.SetDataPrbNum(25);
pfactory.SetDataPrbStart(0);
pfactory.SetDataPrbEnd(49);
```

Compared to MODE 1, the pool resources are now indicated as "UE_SELECTED" along with other parameters. Moreover, now we need to configure the pool parameters related to the data, i.e., PSSCH. In addition to the subframe indicator bitmap specified by KTRP and iTRP, Mode 2 introduces another level of subframe filtering for the subframe pool via "DataBitmap" to limit the number of possible values for iTRP. The PSSCH transmission occurs on the filtered subframes after applying TRP bitmap on this "DataBitmap". Users interested to learn about how it is done are referred to SidelinkCommPoolPsschTestCase.

Finally, at the end of the simulation the trace files, similar to the previous example, can be found in the repository's root folder. One important difference, compared to MODE 1, is that in the trace files "SlUeMacStats.txt" and "SlPscchRxPhyStats.txt" the parameters, e.g., PSCCH resource index and the starting RB for PSSCH are randomly selected by the UE for every SC period. Furthermore, a similar amount of packets, i.e. 20, are sent and received by the UEs.

This example simulates an out-of-coverage MODE 2 ProSe communication by using the following scenario.

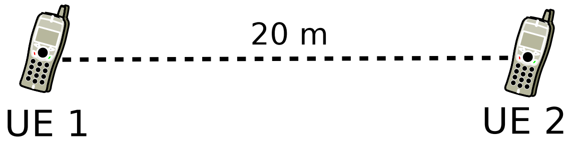


Fig. 2.19: Sidelink simple out-of-coverage scenario

This example allows a user to configure the simulation time and the output of NS logs of the specified classes by using the corresponding global variables in the simulation script. For example, a user can run the simulation as follows:

```
./waf --run "lte-sl-out-of-covrg-comm --simTime=7 --enableNsLogs=false"
```

The simulation time is in seconds, and at this stage, only IPV4 is supported.

- Configuring parameters for PSSCH resource selection :

  ```
  // Fixed MCS and the number of RBs
  Config::SetDefault ("ns3::LteUeMac::SlGrantMcs", UintegerValue (16));
  Config::SetDefault ("ns3::LteUeMac::SlGrantSize", UintegerValue (5));

  // For selecting subframe indicator bitmap
  Config::SetDefault ("ns3::LteUeMac::Ktrp", UintegerValue (1));
  //use default Trp index of 0
  Config::SetDefault ("ns3::LteUeMac::UseSetTrp", BooleanValue (true));
  ```

The above parameters of `LteUeMac` class enable the UE to select the time (i.e., frame/subframe) and the frequency (i.e., RBs) resources autonomously. If the attribute “UseSetTrp” is false, a UE will select the TRP index randomly from the range of values depending on the KTRP value [TS36213].

- Configure the frequency :

  ```
  uint32_t ulEarfcn = 18100;
  uint16_t ulBandwidth = 50;
  ```

  Here, it is not necessary to configure the EARFCNs and the bandwidth of the UE and eNB for the two reasons. First, in this example we will not instantiate the eNB node, thus, setting these attributes would have no impact. Second, both the UEs will use only the Sidelink to communicate, therefore, the EARFCN and the bandwidth are specified in the pool configuration. At this stage, the above two variables are initialized to be used later to configure the pathloss model and the Sidelink pool. Similar, to the previous simple examples, we use the LTE Band 1 for Sidelink communication.

- Configure the error models :

  ```
  // For PSSCH
  Config::SetDefault ("ns3::LteSpectrumPhy::SlDataErrorModelEnabled",
                                                       BooleanValue (true));

  // For PSCCH
  Config::SetDefault ("ns3::LteSpectrumPhy::SlCtrlErrorModelEnabled",
                                                       BooleanValue (true));

  Config::SetDefault ("ns3::LteSpectrumPhy::DropRbOnCollisionEnabled",
                                                       BooleanValue (false));
  ```

Along with the error model configuration, by setting the “DropRbOnCollisionEnabled” attribute all the TBs collided in time and frequency are dropped. This attribute could be of particular interest for the users, e.g., to study the impact of collisions in MODE 2. In this example, this attribute has no impact as only one UE is acting as a transmitter. It is included just to make the users aware of this attribute.

- Configure the transmit power for the UEs :

```
Config::SetDefault ("ns3::LteUePhy::TxPower", DoubleValue (23.0));
```

The power configured is in dBm.

- Instantiating LTE, EPC and Sidelink helpers :

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();
Ptr<PointToPointEpcHelper>  epcHelper = CreateObject<PointToPointEpcHelper> ();
lteHelper->SetEpcHelper (epcHelper);

Ptr<LteSidelinkHelper> proseHelper = CreateObject<LteSidelinkHelper> ();
proseHelper->SetLteHelper (lteHelper);
```

- Enabling the Sidelink :

```
lteHelper->SetAttribute ("UseSidelink", BooleanValue (true));
```

Note : Attribute ‘‘UseSidelink‘‘ must be set before installing the UE devices.

- Configuring the pathloss model and bypass the use of eNB nodes

```
(1)

  lteHelper->SetAttribute ("PathlossModel",
                                   StringValue ("ns3::Cost231PropagationLossModel
↪"));

(2)

  lteHelper->Initialize ();

(3)

  double ulFreq = LteSpectrumValueHelper::GetCarrierFrequency (ulEarfcn); //18100
  NS_LOG_LOGIC ("UL freq: " << ulFreq);
  Ptr<Object> uplinkPathlossModel = lteHelper->GetUplinkPathlossModel ();
  Ptr<PropagationLossModel> lossModel = uplinkPathlossModel->
                                                 GetObject<PropagationLossModel>␣
↪();
  NS_ABORT_MSG_IF (lossModel == NULL, "No PathLossModel");
  bool ulFreqOk = uplinkPathlossModel->
                            SetAttributeFailSafe ("Frequency", DoubleValue␣
↪(ulFreq));

  if (!ulFreqOk)
    {
      NS_LOG_WARN ("UL propagation model does not have a Frequency attribute");
    }
```

The use of eNB nodes can be bypassed by using the above commands strictly in the order they are listed. The command “lteHelper->Initialize ()” basically performs the channel model initialization of all the component carriers. Therefore, it is necessary to configure any desired pathloss model before issuing this command. The commands in step 3 are to properly configure the frequency attribute of the pathloss model used, which is normally done in `InstallSingleEnb` method of `LteHelper`.

- Creating the UE nodes and setting their mobility :

```
NodeContainer ueNodes;
ueNodes.Create (2);

Ptr<ListPositionAllocator> positionAllocUe1 =
                                     CreateObject<ListPositionAllocator> ();
positionAllocUe1->Add (Vector (0.0, 0.0, 1.5));
Ptr<ListPositionAllocator> positionAllocUe2 =
                                     CreateObject<ListPositionAllocator> ();
positionAllocUe2->Add (Vector (20.0, 0.0, 1.5));

//Install mobility
MobilityHelper mobilityUe1;
mobilityUe1.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityUe1.SetPositionAllocator (positionAllocUe1);
mobilityUe1.Install (ueNodes.Get (0));

MobilityHelper mobilityUe2;
mobilityUe2.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityUe2.SetPositionAllocator (positionAllocUe2);
mobilityUe2.Install (ueNodes.Get (1));
```

- Installing LTE devices to the nodes :

```
NetDeviceContainer ueDevs = lteHelper->InstallUeDevice (ueNodes);
```

This example simulates an out-of-coverage scenario, therefore, both the UEs are configured with a pre-configured Sidelink communication pool. As mentioned in *RRC*, in this scenario the `LteSlUeRrc` class will be responsible for holding this per-configured pool configuration. The pool configuration starts by setting a flag in `LteSlUeRrc` as an indication that the Sidelink is enabled. It is configured as follows:

```
Ptr<LteSlUeRrc> ueSidelinkConfiguration = CreateObject<LteSlUeRrc> ();
ueSidelinkConfiguration->SetSlEnabled (true);
```

For configuring Sidelink communication pre-configured pool parameters, the IE “SL-Preconfiguration” defined in the standard [TS36331] is converted into a C++ structure and similar to the basic LTE layer 3 messages it can be found in `LteRrcSap` class. This example uses this structure to configure the Sidelink communication pool parameters. The pool configuration is done by using the `LteSlPreconfigPoolFactory` in the following manner,

```
LteRrcSap::SlPreconfiguration preconfiguration;

preconfiguration.preconfigGeneral.carrierFreq = ulEarfcn; //18100
preconfiguration.preconfigGeneral.slBandwidth = ulBandwidth; // 50 RBs
preconfiguration.preconfigComm.nbPools = 1;

LteSlPreconfigPoolFactory pfactory;

//Control
pfactory.SetControlPeriod("sf40");
pfactory.SetControlBitmap(0x00000000FF); //8 subframes for PSCCH
pfactory.SetControlOffset(0);
pfactory.SetControlPrbNum(22);
pfactory.SetControlPrbStart(0);
pfactory.SetControlPrbEnd(49);

//Data
pfactory.SetDataBitmap(0xFFFFFFFFFF);
pfactory.SetDataOffset(8); //After 8 subframes of PSCCH
pfactory.SetDataPrbNum(25);
pfactory.SetDataPrbStart(0);
pfactory.SetDataPrbEnd(49);

preconfiguration.preconfigComm.pools[0] = pfactory.CreatePool ();

ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);
lteHelper->InstallSidelinkConfiguration (ueDevs, ueSidelinkConfiguration);
```

In addition to the subframe indicator bitmap specified by KTRP and iTRP, Mode 2 introduces another level of subframe filtering for the subframe pool via “DataBitmap” to limit the number of possible values for iTRP. The PSSCH transmission occurs on the filtered subframes after applying TRP bitmap on this “DataBitmap”. Users interested to learn about how it is applied are referred to SidelinkCommPoolPsschTestCase.

- Installing the IP stack on the UEs and assigning IP address :

```
InternetStackHelper internet;
internet.Install (ueNodes);
Ipv4InterfaceContainer ueIpIface;
ueIpIface = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ueDevs));
Ipv4StaticRoutingHelper ipv4RoutingHelper;
for (uint32_t u = 0; u < ueNodes.GetN (); ++u)
  {
    Ptr<Node> ueNode = ueNodes.Get (u);
    // Set the default gateway for the UE
    Ptr<Ipv4StaticRouting> ueStaticRouting =
                   ipv4RoutingHelper.GetStaticRouting (ueNode->GetObject<Ipv4>␣
↪());
    ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (),␣
↪1);
  }
```

- Installing applications and activating Sidelink radio bearers :

```
uint32_t groupL2Address = 255;

//Set Application in the UEs
Ipv4Address groupAddress ("225.0.0.0"); //use multicast address as destination
OnOffHelper sidelinkClient("ns3::UdpSocketFactory",
                           Address ( InetSocketAddress (groupAddress, 8000)));
sidelinkClient.SetConstantRate (DataRate ("16kb/s"), 200);

ApplicationContainer clientApps = sidelinkClient.Install (ueNodes.Get (0));
//onoff application will send the first packet at :
//(2.9 (App Start Time) + (1600 (Pkt size in bits) / 16000 (Data rate)) = 3.0 sec
clientApps.Start (slBearersActivationTime + Seconds (0.9));
clientApps.Stop (simTime - slBearersActivationTime + Seconds (1.0));

ApplicationContainer serverApps;
PacketSinkHelper sidelinkSink ("ns3::UdpSocketFactory",
                       Address (InetSocketAddress (Ipv4Address::GetAny (),␣
↪8000)));
serverApps = sidelinkSink.Install (ueNodes.Get (1));
serverApps.Start(Seconds (2.0));

//Set Sidelink bearers
Ptr<LteSlTft> tft = Create<LteSlTft> (LteSlTft::BIDIRECTIONAL,
                                                     groupAddress,␣
 ↪groupL2Address);
proseHelper->ActivateSidelinkBearer (Seconds (2.0), ueDevs, tft);
```

In this example, an “OnOff” application is installed in the UE 1, which sends a 200 byte packet with the constant bit rate of 16 kb/s. On the other hand, the UE 2 is configured with the “PacketSink” application.

The Sidelink radio bearers are activated by calling the `ActivateSidelinkBearer` method of `LteSidelinkHelper` as follows:

```
//Set Sidelink bearers
Ptr<LteSlTft> tft = Create<LteSlTft> (LteSlTft::BIDIRECTIONAL,
                                                    groupAddress, groupL2Address);
proseHelper->ActivateSidelinkBearer (Seconds (2.0), ueDevs, tft);
```

The `ActivateSidelinkBearer` method takes the following three parameters as input,

1. Time to activate the bearer
2. Devices for which the bearer will be activated
3. The Sidelink traffic flow template

The `tft` in this example is an object of `LteSlTft` class created by initializing its following parameters,

1. Direction of the bearer, i.e., “Transmit”, “Receive” or Bidirectional
2. An IPv4 multicast address of the group
3. Sidelink layer 2 group address (used as Sidelink layer 2 group id)

- Activating the Sidelink traces :

```
lteHelper->EnableSlPscchMacTraces ();
lteHelper->EnableSlPsschMacTraces ();

lteHelper->EnableSlTxPhyTraces ();
lteHelper->EnableSlRxPhyTraces ();
lteHelper->EnableSlPscchRxPhyTraces ();
```

The above code will enable all the Sidelink related traces.

Upon the completion of the simulation, the following trace files can be found in the repository’s root folder,

- SlUeMacStats.txt
- SlSchUeMacStats.txt
- SlPscchRxPhyStats.txt
- SlRxPhyStats.txt

For more information related to the above files please refer to the sec-sidelink-simulation-output section.

- UePacketTrace.tr

  The information in this file is obtained by using the traces “TxWithAddresses” and “RxWithAddresses” of `OnOff` and `PacketSink` application, respectively. Following table shows the snippet of the data from this file for the two UEs,

Table 2.3: UePacketTrace.tr

Time (sec)	tx/rx	NodeID	IMSI	PktSize (bytes)	IP[src]	IP[dst]
3	tx	1	1	200	7.0.0.2:49153	225.0.0.0:8000
3.08893	rx	2	2	200	7.0.0.2:49153	7.0.0.3:8000

The first row shows that the UE 1 with node id 1 and IMSI 1 transmits a multicast packet of 200 bytes and the UE 2 receives the packet transmitted by the UE 1. As per the client's application data rate and ON time, i.e., 16 kb/s and 2 seconds, respectively, a total of 20 packets are sent and received by the transmitting and the receiving UE.

In the following we will discuss the detailed examples mentioned above, however, we will mainly discuss about the topology and the Sidelink configuration, since the example scripts already contain the details of the default attributes and also, most of them are already covered by the previous examples.

The scenario in this example is composed of 1 hexagonal cell site divided into 3 sectors. Each sector has 1 UE, i.e., 3 in total, which are randomly dropped. These UEs are then grouped in a single group by choosing randomly one UE as a transmitter while the other 2 UEs act as receivers.

In the following, we walk through the example line by line and discuss each important topology configuration. However, some configuration parameters are skipped, which are already elaborated in the previous sections of LTE. Therefore, it is highly recommended for the new users to go through the basic LTE configuration before digging into D2D examples.

- Randomizing the frame and subframe number of the UEs :

```
Config::SetDefault ("ns3::LteUePhy::UeRandomInitialSubframeIndication",
                                                       BooleanValue (unsyncSl));
```

As mentioned in the design documentation *Sidelink synchronization*, by default all the UEs are perfectly synchronized, i.e., all the UEs in a simulation upon being initialized pick the same frame and subframe 1 to start with. Therefore, to simulate synchronization and to make every UE to pick a random frame and subframe number the attribute "UeRandomInitialSubframeIndication" should be set.

- Instantiating `LteSidelinkHelper` and setting `LteHelper` :

```
Ptr<LteSidelinkHelper> proseHelper = CreateObject<LteSidelinkHelper> ();
proseHelper->SetLteHelper (lteHelper);
```

- Instantiating `Lte3gppHexGridEnbTopologyHelper` and setting `LteHelper`

```
Ptr<Lte3gppHexGridEnbTopologyHelper> topoHelper =
                                  CreateObject<Lte3gppHexGridEnbTopologyHelper>␣
↪();
topoHelper->SetLteHelper (lteHelper);
```

- Disabling the eNB for out-of-coverage scenario :

```
lteHelper->DisableEnbPhy (true);
```

Note : Call to disable eNB PHY should happen before installing the eNB devices

- Configure the parameters for hexagonal ring topology :

```
topoHelper->SetNumRings (numRings); // 1 Ring
topoHelper->SetInterSiteDistance (isd); // 500 m
topoHelper->SetMinimumDistance (10); // in meter
topoHelper->SetSiteHeight (32); // in meter
```

The code above will set the parameters to build 1 ring, i.e., 1 hexagonal site, which will have 500 meters of inter-site distance between the sites, if the number of rings is more than 1. The minimum distance between an eNB and the UEs is set to 10 m and the eNB is at the height of 32 m above the ground.

- Creating eNB nodes :

```
NodeContainer sectorNodes;
sectorNodes.Create (topoHelper->GetNumNodes ());
```

Call to `topoHelper->GetNumNodes ()` will return 3, which in turns would create 3 sector nodes, since, it requires 3 eNBs to cover one hexagon.

- Fixing eNB mobility and installing eNB devices :

```
MobilityHelper mobilityeNodeB;
mobilityeNodeB.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityeNodeB.Install (sectorNodes);
NetDeviceContainer enbDevs = topoHelper->
                                  SetPositionAndInstallEnbDevice (sectorNodes);
```

The call `SetPositionAndInstallEnbDevice` will compute the position of each site and the antenna orientation of each eNB. The antenna orientation of each eNB in a hexagon is 30, 150 and 270 degrees, respectively [TS25814] *[TR36814]*. As mentioned before, even if the eNB PHY is disabled, the hexagonal topology requires to instantiate the eNB nodes to form the sectors of an hexagon. It is also to be noted that the use of antenna models, e.g., `Parabolic3dAntennaModel`, `ParabolicAntennaModel`, or `CosineAntennaModel` is needed to configure antenna orientation. Moreover, the function `SetPositionAndInstallEnbDevice` is also responsible to call `InstallEnbDevice` of `LteHelper` for each eNB in this scenario.

- Creating UE nodes :

```
NodeContainer ueResponders;
ueResponders.Create (ueRespondersPerSector * sectorNodes.GetN ());
```

This will create 3 UE nodes in total, 1 for each sector of hexagonal site.

- Enabling Sidelink

```
lteHelper->SetAttribute ("UseSidelink", BooleanValue (true));
```

Note : Attribute ‘‘UseSidelink‘‘ must be set before installing the UE devices.

- Fixing UE mobility and installing UE devices :

```
MobilityHelper mobilityResponders;
mobilityResponders.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
topoHelper->DropUEsUniformlyPerSector (ueResponders);
```

This will place 1 UE per sector by choosing their position randomly with in the sector, taking into account the configured minimum eNB to UE distance of 10 m. This will also install the UE devices by calling `InstallUeDevice` of `LteHelper`. The installation of the IP stack and IP address assignment is done in a similar fashion as explain in sec-evolved-packet-core

- Creating groups of UEs :

```
double ulEarfcn = enbDevs.Get (0)->GetObject<LteEnbNetDevice> ()->GetUlEarfcn ();
double ulBandwidth = enbDevs.Get (0)->GetObject<LteEnbNetDevice> ()->
                                                        GetUlBandwidth ();

std::vector < NetDeviceContainer > createdgroups;
```

```
createdgroups = proseHelper->AssociateForBroadcast (ueTxPower, ulEarfcn,
                                                    ulBandwidth, ueRespondersDevs,
                                                    -112, numGroups,
                                                    LteSidelinkHelper::
                                                                SLRSRP_PSBCH);
```

The `AssociateForBroadcast` function will basically form the specified number of groups, i.e., 1 in this example, by choosing the transmitting UE randomly from the total number of UEs. After choosing the transmitting UE, the receiving UEs of the group are selected if the Sidelink RSRP, calculated as per the method defined in [TS36214], between the transmitting UE and the receiving UE is higher than -112 dBm. In this example, UE node 4 with IMSI 1 is selected as a transmitter while remaining 2 UEs will act as receivers. It is to be noted that `AssociateForBroadcast` is only responsible for forming the groups and not installing any application.

All the UEs in a group are stored in a `NetDeviceContainer`, such that the first UE in this container is the transmitter of the group. Finally, this `NetDeviceContainer` is stored in a vector (createdgroups), which is used to install appropriate application in UE devices.

- Installing applications and activating Sidelink radio bearers :

```
**Note : Only IPV4 is supported at this stage**

// Client Application :

 uint32_t groupL2Address = 0x00;

 Ipv4AddressGenerator::Init (Ipv4Address ("225.0.0.0"), Ipv4Mask ("255.0.0.0"));
 Ipv4Address groupRespondersIpv4Address = Ipv4AddressGenerator::
                                           NextAddress (Ipv4Mask ("255.0.0.0"));

 std::vector < NetDeviceContainer >::iterator gIt;
 for (gIt = createdgroups.begin (); gIt != createdgroups.end (); gIt++)
  {
   //Create sidelink bearers
   //Use Tx for the group transmitter and Rx for all the receivers
   //split Tx/Rx
   NetDeviceContainer txUe ((*gIt).Get (0));
   NetDeviceContainer rxUes = proseHelper->
                                    RemoveNetDevice ((*gIt), (*gIt).Get (0));
   Ptr<LteSlTft> tft = Create<LteSlTft> (LteSlTft::TRANSMIT,
                                 groupRespondersIpv4Address, groupL2Address);

   //Sidelink bearer activation
   proseHelper->ActivateSidelinkBearer (Seconds (1.0), txUe, tft);
   tft = Create<LteSlTft>
              (LteSlTft::RECEIVE, groupRespondersIpv4Address, groupL2Address);
   proseHelper->ActivateSidelinkBearer (Seconds (1.0), rxUes, tft);

     .
     .
     .

   UdpEchoClientHelper echoClientHelper
                            (groupRespondersIpv4Address, grpEchoServerPort);
   clientRespondersApps = echoClientHelper.Install ((*gIt).Get (0)->GetNode ());

     .
     .
     .
```

```
    groupL2Address++;
    groupRespondersIpv4Address = Ipv4AddressGenerator::
                                        NextAddress (Ipv4Mask ("255.0.0.0"));
   }

// Server Application :

    PacketSinkHelper clientPacketSinkHelper ("ns3::UdpSocketFactory",
                            InetSocketAddress (Ipv4Address::GetAny (), echoPort));
    ApplicationContainer clientRespondersSrvApps =
                                  clientPacketSinkHelper.Install (ueResponders);
```

In the previous step we obtained the vector `createdgroups`, containing a group formed by one transmitter UE and 2 receiver UEs. Now, it is easy to anticipate that the client application will be installed in the transmitter UE and the receiver UEs will have server application. In this example, a user can configure any of the two client applications, i.e., `OnOff` and `UdpEchoClientApplication`, where the later one is the default client application.

The Sidelink radio bearers for Tx (for transmitting UE) and Rx (for receiving UEs) are activated by calling the *ActivateSidelinkBearer‘* method of `LteSidelinkHelper`.

Finally, a `PacketSink` application is installed as a server application in the receiving UEs.

This example simulates an out-of-coverage scenario, therefore, all the UEs will be configured with a pre-configured Sidelink pool. As mentioned in *RRC*, in this scenario the `LteSlUeRrc` class will be responsible for holding this per-configured pool configuration. The pool configuration starts by setting a flag in `LteSlUeRrc` as an indication that the Sidelink is enabled. It is configured as follows:

```
Ptr<LteSlUeRrc> ueSidelinkConfiguration = CreateObject<LteSlUeRrc> ();
ueSidelinkConfiguration->SetSlEnabled (true);
```

For configuring Sidelink pre-configured pool parameters, the IE “SL-Preconfiguration” defined in the standard [TS36331] is converted into a C++ structure and similar to the basic LTE layer 3 messages it can be found in `LteRrcSap` class. This example uses this structure to configure the Sidelink pool parameters for control, data and synchronization in the following manner,

```
LteRrcSap::SlPreconfiguration preconfiguration;

preconfiguration.preconfigGeneral.carrierFreq = 23330;
preconfiguration.preconfigGeneral.slBandwidth = 50;
preconfiguration.preconfigComm.nbPools = 1;

//control
preconfiguration.preconfigComm.pools[0].scCpLen.cplen = LteRrcSap::SlCpLen::NORMAL;
preconfiguration.preconfigComm.pools[0].scPeriod.period = LteRrcSap::
                                                     PeriodAsEnum (slPeriod).period;
preconfiguration.preconfigComm.pools[0].scTfResourceConfig.prbNum = pscchRbs;
preconfiguration.preconfigComm.pools[0].scTfResourceConfig.prbStart = 3;
preconfiguration.preconfigComm.pools[0].scTfResourceConfig.prbEnd = 46;
preconfiguration.preconfigComm.pools[0].scTfResourceConfig.offsetIndicator.offset = 0;
preconfiguration.preconfigComm.pools[0].scTfResourceConfig.subframeBitmap.
                                                          bitmap = pscchTrpNumber;

//data
preconfiguration.preconfigComm.pools[0].dataCpLen.cplen = LteRrcSap::SlCpLen::NORMAL;
preconfiguration.preconfigComm.pools[0].dataHoppingConfig.hoppingParameter = 0;
preconfiguration.preconfigComm.pools[0].dataHoppingConfig.numSubbands = LteRrcSap::
                                                           SlHoppingConfigComm::ns4;
```

```
preconfiguration.preconfigComm.pools[0].dataHoppingConfig.rbOffset = 0;

preconfiguration.preconfigComm.pools[0].trptSubset.subset = std::bitset<3> (0x7);
preconfiguration.preconfigComm.pools[0].dataTfResourceConfig.prbNum = 25;
preconfiguration.preconfigComm.pools[0].dataTfResourceConfig.prbStart = 0;
preconfiguration.preconfigComm.pools[0].dataTfResourceConfig.prbEnd = 49;
preconfiguration.preconfigComm.pools[0].dataTfResourceConfig.offsetIndicator.
                                                              offset = pscchLength;
preconfiguration.preconfigComm.pools[0].dataTfResourceConfig.subframeBitmap.bitmap =
                                                     std::bitset<40>(0xFFFFFFFFFF);

preconfiguration.preconfigComm.pools[0].scTxParameters.alpha = LteRrcSap::
                                                              SlTxParameters::al09;
preconfiguration.preconfigComm.pools[0].scTxParameters.p0 = -40;
preconfiguration.preconfigComm.pools[0].dataTxParameters.alpha = LteRrcSap::
                                                              SlTxParameters::al09;
preconfiguration.preconfigComm.pools[0].dataTxParameters.p0 = -40;

//Synchronization
preconfiguration.preconfigSync.syncOffsetIndicator1 = 18;
preconfiguration.preconfigSync.syncOffsetIndicator2 = 29;
preconfiguration.preconfigSync.syncTxThreshOoC = syncTxThreshOoC;
preconfiguration.preconfigSync.syncRefDiffHyst = syncRefDiffHyst;
preconfiguration.preconfigSync.syncRefMinHyst = syncRefMinHyst;
preconfiguration.preconfigSync.filterCoefficient = filterCoefficient;
```

Finally, the configured pool is stored in `LteSlUeRrc` class by calling `SetSlPreconfiguration` function

```
ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);
```

Then, by calling the `InstallSidelinkConfiguration` method of `LteHelper` class it configures the `LteUeRrc` attribute named "SidelinkConfiguration" of a UE, which is nothing but a pointer to the `LteSlUeRrc` object used to configure the pool:

```
lteHelper->InstallSidelinkConfiguration (ueRespondersDevs, ueSidelinkConfiguration);
```

This interaction of classes to populate the Sidelink pool is also depicted in *ns-3 LTE Sidelink pool configuration flow*.

Alternatively, the pool configuration for control and data can also be done in a semi-automatic way by using the `LteSlPreconfigPoolFactory`. This would ease the configuration of the pool by only changing the parameters of interest and leaving the rest as default. For example, the above configuration can also be done as follows,

```
preconfiguration.preconfigGeneral.carrierFreq = 23330;
preconfiguration.preconfigGeneral.slBandwidth = 50;
preconfiguration.preconfigComm.nbPools = 1;

LteSlPreconfigPoolFactory pfactory;

//Control
pfactory.SetControlPrbStart (3);
pfactory.SetControlPrbEnd (46);

//Data
pfactory.SetDataOffset (pscchLength);

//Synchronization
preconfiguration.preconfigSync.syncOffsetIndicator1 = 18;
preconfiguration.preconfigSync.syncOffsetIndicator2 = 29;
preconfiguration.preconfigSync.syncTxThreshOoC = syncTxThreshOoC;
preconfiguration.preconfigSync.syncRefDiffHyst = syncRefDiffHyst;
preconfiguration.preconfigSync.syncRefMinHyst = syncRefMinHyst;
preconfiguration.preconfigSync.filterCoefficient = filterCoefficient;

preconfiguration.preconfigComm.pools[0] = pfactory.CreatePool ();

ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);

lteHelper->InstallSidelinkConfiguration (ueRespondersDevs, ueSidelinkConfiguration);
```

Upon the completion of the simulation, the information related to the topology and the data related to the synchronization process gathered through the traces is written to the following files,

- nPositions.txt

  This file stores the position of all the nodes, i.e., eNBs and the UEs in the simulation.

- prose-connections.txt

  This file stores the information, e.g., Node id, and IMSI of the transmitting and the corresponding receiving UEs.

- FirstScan.txt

  This file stores the start time (in ms) of the first scanning period and the IMSI of each UE nodes. The start time for each UE is randomly chosen between 2000 ms and 4000 ms defined by the variables "firstScanTimeMin" and "firstScanTimeMax" in the simulation script.

- SyncRef.txt

  The information in this file is gathered by listening to the trace "ChangeOfSyncRef" of `LteUeRrc`.

Table 2.4: SyncRef.txt

Time (ms)	IMSI	prev SLSSID	prev RxOffset	prev FrameNo	prev SframeNo	curr SLSSID	curr RxOffset	curr FrameNo	curr SframeNo
0	1	0	0	0	0	141969	0	0	0
0	2	0	0	0	0	199946	0	0	0
0	3	0	0	0	0	105029	0	0	0
20025	1	141696	0	985	5	10	0	985	5
20585	2	199946	0	161	5	10	3	17	5
21567	3	105029	0	856	2	10	3	115	7
50000	1	10	0	910	10	0	0	0	0
50000	2	10	0	910	10	0	0	0	0
50000	3	10	0	910	10	0	0	0	0

  It is to be noted that the data in first and last three rows of above table are written by this simulation script to mark the start and end of the simulation. From row 4 we can observe that the UE with IMSI 1 becomes the SyncRef, this can be deduced by looking at its SLSS-ID, which is `SLSS-ID = IMSI * 10` as explained in *RRC*. On the other hand, UEs with IMSI 2 and 3 selects IMSI 1 as their SyncRef and use the same SLSS-ID, as shown in row 5 and 6.

- pscr-ue-pck.tr

  The information in this file is obtained by using the traces "TxWithAddresses" and "RxWithAddresses" of `UdpEchoClient` and `PacketSink` application respectively. Following table shows the snippet of data from this file for all the three UEs,

Table 2.5: pscr-ue-pck.tr

Time (ms)	tx/rx	NID	IMSI	UEtype	size (bytes)	IP[src]	IP[dst]
21690	t	4	1	resp	40	7.0.0.2:49153	225.0.0.1:8000
21690	r	5	2	resp	40	7.0.0.2:49153	7.0.0.3:8000
21690	r	6	3	resp	40	7.0.0.2:49153	7.0.0.4:8000

The first row shows that the first UE with node id 4 and IMSI 1 acts as a transmitter and sends a multicast packet of 40 bytes. Similarly, the other two UEs act as receiver and receive the packet transmitted by the first UE.

- TxSlss.txt

  This file stores the information gathered by listening to the trace "SendSLSS" of `LteUeRrc`. Following is the snippet of the information stored in this file.

Table 2.6: TxSlss.txt

Time (ms)	IMSI	SLSSID	txOffset (ms)	inCoverage	FrameNo	SframeNo
20040	1	10	29	0	986	9
20080	1	10	29	0	990	9
20120	1	10	29	0	994	9
20160	1	10	29	0	998	9

It can be observed that UE with IMSI 1 being a SyncRef is transmitting SLSS with a fixed periodicity of 40 ms. Moreover, at the time when this UE chose to become a SyncRef it randomly selects one of the two configured offsets, i.e., syncOffsetIndicator1 and syncOffsetIndicator2. In this example, it has selected syncOffsetIndicator2, which is set to 29 ms.

This example deploys 10 out-of-coverage UEs, distributed randomly within an area of 100 m x 100 m. This example script also illustrates how an out-of-coverage Sidelink related simulation can be performed without instantiating the eNB nodes.

Compared to the topology of the synchronization example that we discussed before, this example is very simple. For instance, it does not use hexagonal topology or form groups of UEs, i.e, it does not use `Lte3gppHexGridEnbTopologyHelper` and `LteSidelinkHelper`. The `LteHelper` and `PointToPointEpcHelper` are initialized in the same way like any other LTE example with EPC. Also, the call to disable the eNB PHY can also be skipped, since it does not instantiates the eNB nodes. Moreover, this example only focuses on simulating ProSe out-of-coverage discovery, therefore, there is no need to configure the IP of the UEs since discovery is purely a MAC layer application where the messages are filtered on the basis of ProSeAppCode.

The first Sidelink related configuration in this example is to enable the Sidelink.

- Enabling Sidelink

```
lteHelper->SetAttribute ("UseSidelink", BooleanValue (true));
```

Note : Attribute ''UseSidelink'' must be set before installing the UE devices.

- Work-around to bypass the use of eNB nodes

```
(1)

  lteHelper->SetAttribute ("PathlossModel",
                                   StringValue ("ns3::FriisPropagationLossModel
↪"));

(2)

  lteHelper->Initialize ();
```

```
(3)

  double ulFreq = LteSpectrumValueHelper::GetCarrierFrequency (23330);
  NS_LOG_LOGIC ("UL freq: " << ulFreq);
  Ptr<Object> uplinkPathlossModel = lteHelper->GetUplinkPathlossModel ();
  Ptr<PropagationLossModel> lossModel = uplinkPathlossModel->
                                        GetObject<PropagationLossModel>␣
↪();
  NS_ABORT_MSG_IF (lossModel == NULL, "No PathLossModel");
  bool ulFreqOk = uplinkPathlossModel->
                        SetAttributeFailSafe ("Frequency", DoubleValue␣
↪(ulFreq));

  if (!ulFreqOk)
    {
      NS_LOG_WARN ("UL propagation model does not have a Frequency attribute");
    }
```

The use of eNB nodes can be bypassed by using the above commands strictly in the order they are listed. The command “lteHelper->Initialize ()” basically performs the channel model initialization of all the component carriers. Therefore, it is necessary to configure any desired pathloss model before issuing this command. The commands in step 3 are to properly configure the frequency attribute of the pathloss model used, which is normally done in `InstallSingleEnb` method of `LteHelper`.

- Creating the UE node, fixing their mobility, and installing UE devices :

```
NodeContainer ues;
ues.Create (nbUes);

//Position of the nodes
Ptr<ListPositionAllocator> positionAllocUe =
                                    CreateObject<ListPositionAllocator>␣
↪();

for (uint32_t u = 0; u < ues.GetN (); ++u)
  {
    Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable> ();
    double x = rand->GetValue (-100,100);
    double y = rand->GetValue (-100,100);
    double z = 1.5;
    positionAllocUe->Add (Vector (x, y, z));
  }

// Install mobility

MobilityHelper mobilityUe;
mobilityUe.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobilityUe.SetPositionAllocator (positionAllocUe);
mobilityUe.Install (ues);

NetDeviceContainer ueDevs = lteHelper->InstallUeDevice (ues);
```

The above commands will simply create 10 UEs and position each UE at a random location in an area of 100 m x 100 m. The UE devices are installed in a usual way by calling `InstallUeDevice` of `LteHelper` class.

- Configuring discovery applications and starting the discovery process :

```
std::map<Ptr<NetDevice>, std::list<uint32_t> > announceApps;
std::map<Ptr<NetDevice>, std::list<uint32_t> > monitorApps;
for (uint32_t i = 1; i <= nbUes; ++i)
  {
    announceApps[ueDevs.Get (i - 1)].push_back ((uint32_t)i);
    for (uint32_t j = 1; j <= nbUes; ++j)
      {
        if (i != j)
          {
            monitorApps[ueDevs.Get (i - 1)].push_back ((uint32_t)j);
          }
      }
  }



for (uint32_t i = 0; i < nbUes; ++i)
  {
    // true for announce
    Simulator::Schedule (Seconds (2.0), &SlStartDiscovery,
                             lteHelper, ueDevs.Get (i),
                             announceApps.find (ueDevs.Get (i))->second,␣
↪true);

    // false for monitor
    Simulator::Schedule (Seconds (2.0), &SlStartDiscovery,
                             lteHelper, ueDevs.Get (i),
                             monitorApps.find (ueDevs.Get (i))->second,␣
↪false);
  }
```

Given the broadcast nature of the discovery process, the applications are configured such that each UE will be able to transmit discovery messages with one ProSeAppCode, i.e., one announce app and will be able to monitor/receive the discovery messages from all other UEs (Notice the two `for` loops at the beginning). The UE device index in the `NetDeviceContainer` is used as a ProSeAppCode, which is later in `LteUeRrc` is converted into a 184 bitset *[TS23003]*. Finally, at 2 sec the method "SlStartDiscovery" implemented in this example calls the `StartDiscovery` of `LteHelper` class to start the discovery process.

This example simulates an out-of-coverage scenario, therefore, all the UEs will be configured with a pre-configured Sidelink discovery pool. As mentioned in *RRC*, in this scenario the `LteSlUeRrc` class will be responsible for holding this per-configured pool configuration. The pool configuration starts by setting a flag in `LteSlUeRrc` as an indication that the Sidelink discovery is enabled. It is configured as follows:

```
Ptr<LteSlUeRrc> ueSidelinkConfiguration = CreateObject<LteSlUeRrc> ();
ueSidelinkConfiguration->SetDiscEnabled (true);
```

For configuring Sidelink discovery pre-configured pool parameters, the IE "SL-Preconfiguration" defined in the standard [TS36331] is converted into a C++ structure and similar to the basic LTE layer 3 messages it can be found in `LteRrcSap` class. This example uses this structure to configure the Sidelink discovery pool parameters in the following way,

```
LteRrcSap::SlPreconfiguration preconfiguration;

preconfiguration.preconfigGeneral.carrierFreq = 23330;
preconfiguration.preconfigGeneral.slBandwidth = 50;
preconfiguration.preconfigDisc.nbPools = 1;

preconfiguration.preconfigDisc.pools[0].cpLen.cplen = LteRrcSap::SlCpLen::NORMAL;
preconfiguration.preconfigDisc.pools[0].discPeriod.period = LteRrcSap::
                                                    SlPeriodDisc::rf32;
preconfiguration.preconfigDisc.pools[0].numRetx = 0;
preconfiguration.preconfigDisc.pools[0].numRepetition = 1;
preconfiguration.preconfigDisc.pools[0].tfResourceConfig.prbNum = 10;
preconfiguration.preconfigDisc.pools[0].tfResourceConfig.prbStart = 10;
preconfiguration.preconfigDisc.pools[0].tfResourceConfig.prbEnd = 49;
preconfiguration.preconfigDisc.pools[0].tfResourceConfig.offsetIndicator.offset = 0;
preconfiguration.preconfigDisc.pools[0].tfResourceConfig.subframeBitmap.
                                        bitmap = std::bitset<40>(0x11111);
preconfiguration.preconfigDisc.pools[0].txParameters.txParametersGeneral.alpha =
                                         LteRrcSap::SlTxParameters::al09;

preconfiguration.preconfigDisc.pools[0].txParameters.txParametersGeneral.p0 = -40;

preconfiguration.preconfigDisc.pools[0].txParameters.txProbability =
                            SidelinkDiscResourcePool::TxProbabilityFromInt (txProb);

NS_LOG_INFO ("Install Sidelink discovery configuration in the UEs...");
ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);
lteHelper->InstallSidelinkConfiguration (ueDevs, ueSidelinkConfiguration);
```

Alternatively, the discovery pool configuration can also be done in a semi-automatic way by using the `LteSlDiscResourcePoolFactory`. This would ease the configuration of the pool by only changing the parameters of interest and leaving the rest as default. For example, the above configuration can also be done as follows,

```
preconfiguration.preconfigGeneral.carrierFreq = 23330;
preconfiguration.preconfigGeneral.slBandwidth = 50;
preconfiguration.preconfigComm.nbPools = 1;

LteSlDiscPreconfigPoolFactory pfactory;
pfactory.SetDiscPrbEnd (49);

preconfiguration.preconfigDisc.pools[0] = pfactory.CreatePool ();

ueSidelinkConfiguration->SetSlPreconfiguration (preconfiguration);
lteHelper->InstallSidelinkConfiguration (ueDevs, ueSidelinkConfiguration);
```

Upon the completion of the simulation a user can find the following files containing the information of the simulated discovery process.

- discovery_nodes.txt

  This file stores the position of all the UEs in the simulation.

- discovery-out-announcement-mac.tr

  This file logs the discovery message transmissions (announcements) by listening to the "DiscoveryAnnouncement" trace of `LteUeMac` class. Following is the snippet from this file,

Table 2.7: discovery-out-announcement-mac.tr

Time (s)	IMSI	RNTI	ProSeAppCode
2.236	1	1	1
2.236	3	3	3
2.236	5	5	5
2.236	9	9	9

- discovery-out-announcement-phy.tr

  Similar to the information in "discovery-out-announcement-mac.tr" file this file contains the information about the discovery announcements but with the perspective of phy layer. In particular, it is obtained by listening to the "DiscoveryAnnouncement" trace of `LteUePhy` class. Following is the snippet from this file,

Table 2.8: discovery-out-announcement-phy.tr

Time (s)	IMSI	CellId	RNTI	ProSeAppCode
2.24	1	0	1	1
2.24	3	0	3	3
2.24	5	0	5	5
2.24	9	0	9	9

By comparing the traces from UE MAC and UE PHY, one can notice that the phy layer transmits a particular discovery messages exactly after 4 ms of the processing delay between UE MAC and PHY. The CellId is 0 because of out-of-coverage scenario.

- discovery-out-monitoring.tr

  This file contains the information about the reception of discovery messages by each UE. The data is obtained by listening to the "DiscoveryMonitoring" trace of `LteUeRrc` class. Following is snippet from this file,

Table 2.9: discovery-out-monitoring.tr

Time (s)	IMSI	CellId	RNTI	ProSeAppCode
2.24093	2	0	2	3
2.24093	2	0	2	9
2.24093	2	0	2	1
2.24093	2	0	2	5

From the above table it can be observed that the UE with IMSI 2 has received the discovery messages with ProSeAppCode of 3,9,1,5. In other words, this UE has discovered 4 other UEs transmitting their discovery announcements using the ProSeAppCode of 3, 9, 1, 5 respectively.

CHAPTER

THREE

BUILDINGS MODULE

This chapter is an excerpt of the Buildings module documentation chapter, containing the 3GPP-related models that have been added to the *ns-3* buildings module.

3.1 Design documentation

3.1.1 3GPP aligned propagation loss models

In the following we describe all the propagation loss models, which are compliant with 3GPP standards. In particular, following propagation loss models are implemented:

- Hybrid3gppPropagationLossModel
- IndoorToIndoorPropagationLossModel
- OutdoorToIndoorPropagationLossModel
- OutdoorToOutdoorPropagationLossModel
- ScmUrbanMacroCellPropagationLossModel
- UrbanMacroCellPropagationLossModel

Hybrid3gppPropagationLossModel

The `Hybrid3gppPropagationLossModel` pathloss model is a combination of the following pathloss models:

- IndoorToIndoorPropagationLossModel
- OutdoorToIndoorPropagationLossModel
- OutdoorToOutdoorPropagationLossModel
- UrbanMacroCellPropagationLossModel

This wrapper class is created to make it easier to evaluate the pathloss in different environments, e.g., Macro cell, D2D outdoor, indoor, hybrid (i.e. outdoor to indoor) and with buildings. The following pseudo-code illustrates how the different pathloss models are integrated in `Hybrid3gppPropagationLossModel`:

```
if (Macro Cell Communication)
   then
     L = UrbanMacroCell
else (D2D communication)
   if (NodeA is outdoor)
     then
```

```
      if (NodeB is outdoor)
         then
           L = OutdoorToOutdoor
         else
           L = OutdoorToIndoor
   else (NodeA is indoor)
      if (NodeB is indoor)
         then
           L = IndoorToIndoor
         else
           L = OutdoorToIndoor
```

IndoorToIndoorPropagationLossModel

The model is defined by 3GPP for D2D indoor to indoor scenarios *[TR36843] [TR36814]*. It considers LOS and NLOS scenarios for 700 MHz frequency (Public Safety use cases) by taking into account the shadowing according to a log-normal distribution. For the case when UE is inside the same building as hotspot the standard deviation is 3 dB and 4 dB for LOS and NLOS, respectively. For the scenario when UE is in a different building the standard deviation is 10 dB.

UE is inside a different building as the indoor hotzone

$$L_{\mathrm{NLOS}}[\mathrm{dB}] = max(131.1 + 42.8\log_{10}(R), 147.4 + 43.3\log_{10}(R))$$

UE is inside the same building as the indoor hotzone

$$\begin{aligned} L_{\mathrm{LOS}}[\mathrm{dB}] &= 89.5 + 16.9\log_{10}(R) \\ L_{\mathrm{NLOS}}[\mathrm{dB}] &= 147.4 + 43.3\log_{10}(R) \end{aligned}$$

where the probability of LOS is given as:

$$Prob(R) = \begin{cases} 1 & \text{if } R \leq 0.018Km \\ e^{\frac{-(R-0.018)}{0.027}} & \text{if } 0.018Km \leq R \leq 0.037Km \\ 0.5 & \text{if } R \geq 0.037Km \end{cases}$$

According to the standard *[TR36843]*, the pathloss for 700 MHz band is computed by applying $20\log_{10}(f_c)$ to the pathloss at 2 GHz as follows,

$$LOSS[\mathrm{dB}] = \quad LOSS + 20\log_{10}(\tfrac{f_c}{2}) \quad \text{if } 0.758GHz \leq f_c \leq 0.798GHz \quad .$$

where

f_c : frequency [GHz]

R : distance between the hotspot and UE [Km]

OutdoorToIndoorPropagationLossModel

This model is implemented for outdoor to indoor scenarios as per the specifications in [TR36843]. The model supports both Line-of-Sight (LOS) and Non Line-of-Sight (NLOS) scenarios by taking in to account the shadowing according to a log-normal distribution with standard deviation of 7 dB for both the scenarios.

The pathloss equations used by this model is:

$$L_{\mathrm{LOS}}[\mathrm{dB}] = PL\_B1\_tot(d_{in} + d_{out}) + 20 + 0.5d_{in}$$

$$L_{\mathrm{NLOS}}[\mathrm{dB}] = PL\_B1\_tot(d_{in} + d_{out}) + 20 + 0.5d_{in} - 0.8h_{ms}$$

PL_B1_tot is computed as follows,

$$PL\_B1\_tot(d_{in} + d_{out}) = max(PL_{freespace}(d), PL\_B1(d_{in} + d_{out}))$$

where $PL_{freespace}$ is free space path loss from Eq. 4.24 in [winner].

$$PL_{freespace} = 20\log_{10}(d) + 46.4 + 20\log_{10}(\frac{f_{\mathrm{c}}}{5})$$

and PL_B1 is the path loss from Winner + B1 channel model for LOS and NLOS scenarios in hexagonal layout [winnerfinal]:

For LOS

$$PL\_B1_{\mathrm{LOS}}[\mathrm{dB}] = \begin{cases} 22.7\log_{10}(d_{in} + d_{out}) + 27 + 20\log_{10}(f_{\mathrm{c}}) + LOS_{offset} & \text{if } 3m \leq d \leq d_{\mathrm{BP}} \\ 40\log_{10}(d_{in} + d_{out}) + 7.56 - 17.3\log_{10}(h'_{\mathrm{bs}}) - & \text{if } d_{\mathrm{BP}} \leq d \leq 5000m \\ 17.3\log_{10}(h'_{\mathrm{ms}}) + 2.7\log_{10}(f_{\mathrm{c}}) + LOS_{offset} & \end{cases}$$

where the LOS_{offset} is 0 dB and the breakpoint distance is given by:

$$d_{\mathrm{BP}} \approx 4h'_{\mathrm{bs}}h'_{\mathrm{ms}}(\frac{f_{\mathrm{c}}[Hz]}{c})$$

the LOS probability is computed as follows:

$$P_{\mathrm{LOS}} = min(\frac{18}{d}, 1)(1 - e^{\frac{-d}{36}}) + e^{\frac{-d}{36}}$$

and the effective antenna height of the eNB and UE is computed as:

$$h'_{\mathrm{bs}} = h_{\mathrm{bs}} - 1$$

$$h'_{\mathrm{ms}} = h_{\mathrm{ms}} - 1$$

For NLOS

The model supports frequency bands of 700 MHz for Public Safety and 2 GHz for general scenarios in NLOS. The pathloss equations used are the following:

for 700 MHz:

$$PL\_B1_{\mathrm{NLOS}}[\mathrm{dB}] = \begin{cases} (44.9 - 6.55\log_{10}(h_{\mathrm{bs}}))\log_{10}(d_{in} + d_{out}) + 5.83\log_{10}(h_{\mathrm{bs}}) + & \text{if } 3m \leq d \leq 2000m \\ 16.33 + 26.16\log_{10}(f_{\mathrm{c}}) + NLOS_{offset} & \end{cases}$$

for 2 GHz:

$$PL\_B1_{\text{NLOS}}[\text{dB}] = \begin{cases} (44.9 - 6.55\log_{10}(h_{\text{bs}}))\log_{10}(d_{in} + d_{out}) + 5.83\log_{10}(h_{\text{bs}}) + & \text{if } 3m \leq d \leq 2000m \\ 14.78 + 34.97\log_{10}(f_{\text{c}}) + NLOS_{offset} \end{cases}$$

where the $NLOS_{offset}$ is 5 dB.

The remaining parameters used in the above equations are:

f_{c} : frequency [GHz]

d : distance between the eNB and UE [m]

d_{in} : distance from the wall to the indoor terminal [m]

d_{out} : distance between the outdoor terminal and the point on the wall that is nearest to the the indoor terminal [m]

h_{bs} : eNB antenna height above the ground [m]

h_{ms} : UE antenna height above the ground [m]

$h^{'}_{\text{bs}}$: effective antenna height of the eNB [m]

$h^{'}_{\text{ms}}$: effective antenna height of the UE [m]

LOS_{offset} : line-of-sight offset

$NLOS_{offset}$: non line-of-sight offset

c : speed of light in vacuum (3x10$^{8}m/s$)

OutdoorToOutdoorPropagationLossModel

This propagation loss model is defined by 3GPP for Device to Device (D2D) outdoor to outdoor scenario *[TR36843]*. The model supports both LOS and NLOS scenarios by taking in to account the shadowing according to a log-normal distribution with standard deviation of 7 dB for both the scenarios.

The pathloss equation used by this model is:

$$PL\_B1\_tot(d) = max(PL_{freespace}(d), PL\_B1(d))$$

where $PL_{freespace}$ is free space path loss from Eq. 4.24 in *[winner]*.

$$PL_{freespace} = 20\log_{10}(d) + 46.4 + 20\log_{10}(\frac{f_{\text{c}}}{5})$$

and PL_B1 is the path loss from Winner + B1 channel model *[winnerfinal]* for hexagonal layout and is given by:

$$L_{\text{LOS}}[\text{dB}] = \begin{cases} 22.7\log_{10}(d) + 27 + 20\log_{10}(f_{\text{c}}) + LOS_{offset} & \text{if } 3m \leq d \leq d_{\text{BP}} \\ 40\log_{10}(d) + 7.56 - 17.3\log_{10}(h^{'}_{\text{bs}}) - 17.3\log_{10}(h^{'}_{\text{ms}}) + 2.7\log_{10}(f_{\text{c}}) + LOS_{offset} & \text{if } d_{\text{BP}} \leq d \leq 5000m \end{cases}$$

where the breakpoint distance is given by:

$$d_{\text{BP}} \approx 4h^{'}_{\text{bs}}h^{'}_{\text{ms}}(\frac{f_{\text{c}}[Hz]}{c})$$

The implemented model supports two range of frequency bands 700 MHz and 2 GHz in NLOS scenarios. The pathloss equations are the following:

for 700 MHz:

$$L_{\text{NLOS}}[\text{dB}] = \begin{cases} (44.9 - 6.55\log_{10}(h_{\text{bs}}))\log_{10}(d) + 5.83\log_{10}(h_{\text{bs}}) + & \text{if } 3m \leq d \leq 2000m \\ 16.33 + 26.16\log_{10}(f_{\text{c}}) + NLOS_{offset} \end{cases}$$

for 2 GHz:

$$L_{\text{NLOS}}[\text{dB}] = \begin{cases} (44.9 - 6.55\log_{10}(h_{\text{bs}}))\log_{10}(d) + 5.83\log_{10}(h_{\text{bs}}) + & \text{if } 3m \leq d \leq 2000m \\ 14.78 + 34.97\log_{10}(f_{\text{c}}) + NLOS_{offset} \end{cases}$$

and the probability of LOS is:

$$P_{\text{LOS}} = min(\frac{18}{d}, 1)(1 - e^{\frac{-d}{36}}) + e^{\frac{-d}{36}}$$

According to the standard while calculating Winner + B1 pathloss the following values shall be used

$$h_{\text{bs}} = h_{\text{ms}} = 1.5m$$

$$h'_{\text{bs}} = h'_{\text{ms}} = 0.8m$$

$$LOS_{offset} = 0dB$$

$$NLOS_{offset} = -5dB$$

where

f_{c} : frequency [GHz]

d : distance between the eNB and UE [m]

h_{bs} : eNB antenna height above the ground [m]

h_{ms} : UE antenna height above the ground [m]

h'_{bs} : effective antenna height of the eNB [m]

h'_{ms} : effective antenna height of the UE [m]

LOS_{offset} : line-of-sight offset

$NLOS_{offset}$: non line-of-sight offset

c : speed of light in vacuum (3x10$^8 m/s$)

We note that, the model returns a free space path loss value if the distance between a transmitter and a receiver is less than 3 m.

ScmUrbanMacroCellPropagationLossModel

This propagation loss model is based on the specifications defined for 3GPP Spatial Channel Model (SCM) *[TR25996]* for NLOS urban macro-cell scenario. The pathloss is based on the modified COST231 Okumura Hata urban propagation model for frequencies ranging from 150 – 2000 MHz. The model also considers shadowing according to a log-normal distribution with standard deviation of 8 dB, as defined in the standard *[TR25996]*.

The pathloss expression used by this model is:

$$L[\text{dB}] = (44.9 - 6.55\log_{10}(h_{\text{bs}}))\log_{10}(\frac{d}{1000}) + 45.5 + (35.46 - 1.1(h_{\text{ms}}))\log_{10}(f_{\text{c}}) - 13.82\log_{10}(h_{\text{bs}}) + 0.7(h_{\text{ms}}) + C$$

where

f_{c} : frequency [MHz]

h_{bs} : eNB antenna height above the ground [m]

h_{ms} : UE antenna height above the ground [m]

d : distance between the eNB and UE [m]

C : Constant factor

The value of $C = 3dB$ for urban macro-cell scenario.

UrbanMacroCellPropagationLossModel

This propagation loss model is developed and documented by 3GPP in *[TR36814]*. The implemented model covers an urban macro-cell scenario for the frequency range of 2 - 6 GHz with different antennas, building heights and street widths. It is designed for both LOS and NLOS scenarios by taking in to account the shadowing according to a log-normal distribution with standard deviation of 4 dB and 6 dB, for LOS and NLOS, respectively.

The pathloss expressions used by this model are:

$$L_{\mathrm{LOS}}[\mathrm{dB}] = \begin{cases} 22\log_{10}(d) + 28 + 20\log_{10}(f_{\mathrm{c}}) & \text{if } 10m \le d \le d_{\mathrm{BP}} \\ 40\log_{10}(d) + 7.8 - 18.0\log_{10}(h_{\mathrm{bs}}^{'}) - 18.0\log_{10}(h_{\mathrm{ms}}^{'}) + 2\log_{10}(f_{\mathrm{c}}) & \text{if } d_{\mathrm{BP}} \le d \le 5000m \end{cases}$$

$$L_{\mathrm{NLOS}}[\mathrm{dB}] = \begin{cases} 161.04 - 7.1\log_{10}(W) + 7.5\log_{10}(h) - & \text{if } 10m \le d \le 5000m \\ (24.37 - 3.7(\frac{h}{h_{\mathrm{bs}}})^2)\log_{10}(h_{\mathrm{bs}}) + (43.42 - 3.1\log_{10}(h_{\mathrm{bs}}))(\log_{10}(d) - 3) + & \\ 20\log_{10}(f_{\mathrm{c}}) - (3.2 - (\log_{10}(11.75h_{\mathrm{ms}}))^2 - 4.97) & \end{cases}$$

where the breakpoint distance is given by:

$$d_{\mathrm{BP}} \approx 4h_{\mathrm{bs}}^{'}h_{\mathrm{ms}}^{'}(\frac{f_{\mathrm{c}}[Hz]}{c})$$

The probability of LOS is given by:

$$P_{\mathrm{LOS}} = min(\frac{18}{d}, 1)(1 - e^{\frac{-d}{63}}) + e^{\frac{-d}{63}}$$

and the effective antenna heights of the eNB and UE are computed as:

$$h_{\mathrm{bs}}^{'} = h_{\mathrm{bs}} - 1$$

$$h_{\mathrm{ms}}^{'} = h_{\mathrm{ms}} - 1$$

and the above parameters are

f_{c} : frequency [GHz]

d : distance between the eNB and UE [m]

h : average height of the building [m]

W : street width [m]

h_{bs} : eNB antenna height above the ground [m]

h_{ms} : UE antenna height above the ground [m]

$h_{\mathrm{bs}}^{'}$: effective antenna height of the eNB [m]

$h_{\mathrm{ms}}^{'}$: effective antenna height of the UE [m]

c : speed of light in vacuum (3x$10^8 m/s$)

The model returns 0 dB loss if the distance between a transmitter and a receiver is less than 10 m. Therefore, a user should carefully deploy the UEs, such that, the distance between an eNB and a UE is 10 m or above.

CHAPTER

FOUR

ANTENNA MODULE

This chapter is an excerpt of the Antenna module documentation chapter, containing the parabolic antenna models that have been added to the *ns-3* antenna module.

4.1 Design documentation

4.1.1 Provided models

In this section we describe the PSC-related antenna radiation pattern models that are included within the antenna module.

ParabolicAntennaModel

This model is based on the parabolic approximation of the main lobe radiation pattern. It is often used in the context of cellular system to model the radiation pattern of a cell sector, see for instance *[R4-092042a]* and *[Calcev]*. The antenna gain in dB is determined as:

$$g_{dB}(\phi, \theta) = -\min \left(12 \left(\frac{\phi - \phi_0}{\phi_{3dB}} \right)^2, A_{max} \right)$$

where ϕ_0 is the azimuthal orientation of the antenna (i.e., its direction of maximum gain), ϕ_{3dB} is its 3 dB beamwidth, and A_{max} is the maximum attenuation in dB of the antenna. Note that this radiation pattern is independent of the inclination angle θ.

Parabolic3dAntennaModel

Another 3GPP-defined antenna model is the `Parabolic3dAntennaModel`, drawn from 3GPP TR 36.814 *[TR36814]*. The model, for 3-sector cell sites with fixed antenna patterns, is defined in Table A.2.1.1-2, 3GPP Case 1 and 3 (Macro-cell). Both a horizontal and vertical antenna pattern is defined, and an equation for combining the two methods is provided. So in contrast to `ParabolicAntennaModel`, different horizontal and vertical configuration parameters are required. The attributes `HorizontalBeamwidth`, `Orientation`, `MaxHorizontalAttenuation`, `VerticalBeamwidth`, and `MaxVerticalAttenuation` are configured with the suggested default values. In addition, attributes for mechanical and electrical tilt are defined; these help to adjust the azimuth angle with respect to the reference system of the antenna.

4.2 Testing Documentation

In this section we describe the PSC-related test suites included with the antenna module that verify its correct functionality.

The unit test suite `parabolic-antenna-model` checks that the `ParabolicAntennaModel` class works properly. Several test cases are provided that check for the antenna gain value calculated at different directions and for different values of the orientation, the maximum attenuation and the beamwidth. The reference gain is calculated by hand. Each test case passes if the reference gain in dB is equal to the value returned by `ParabolicAntennaModel` within a tolerance of 0.001, which accounts for the approximation done for the calculation of the reference values.

The unit test suite `parabolic-3d-antenna-model` is based on the `ParabolicAntennaModel` tests. A sequence of test cases at different directions is defined:

1. test horizontal plane with a 60 deg beamwidth; gain is -20dB at +-77.460 degrees from boresight
2. test positive orientation with a 60 deg beamwidth; gain is -10dB at +-54.772 degrees from boresight
3. test negative orientation and different beamwidths with a 80 deg beamwidth; gain is -20dB at +- 73.030 degrees from boresight
4. test vertical plane
5. test tilt

The reference gain is calculated by hand. Each test case passes if the reference gain in dB is equal to the value returned by `Parabolic3dAntennaModel` within a tolerance of 0.001, which accounts for the approximation done for the calculation of the reference values.

BIBLIOGRAPHY

[TS24334] 3GPP TS 24.334 "Proximity-services (ProSe) User Equipment (UE) to ProSe function protocol aspects; Stage 3"

[TS23303] 3GPP TS 23.303 "Technical Specification Group Services and System Aspects; Proximity-based services (ProSe); Stage 2"

[TS23003] 3GPP TS 23.003 "Technical Specification Group Core Network and Terminals; Numbering, addressing and identification; V15"

[TR36814] 3GPP TR 36.814 "E-UTRA Further advancements for E-UTRA physical layer aspects"

[TR36843] 3GPP TR 36.843 "Study on LTE Device to Device Proximity Services; Radio Aspects"

[NIST2016] Rouil, R., Cintrón, F.J., Ben Mosbah, A. and Gamboa, S., "An LTE Device-to-Device module for ns-3 ", in Proceedings of the Workshop on ns-3, 15-16 June 2016, Seattle (Washington).

[NIST2017] Rouil, R., Cintrón, F.J., Ben Mosbah, A. and Gamboa, S., "Implementation and Validation of an LTE D2D Model for ns-3", in Proceedings of the Workshop on ns-3, 13-14 June 2017, Porto (Portugal).

[NISTBLERD2D] J. Wang, R. Rouil "BLER Performance Evaluation of LTE Device-to-Device Communications. Technical Report. National Institute of Standards and Technology, Gaithersburg, MD.",.

[NISTFREQHOPP] Cintrón, F.J., "Performance Evaluation of LTE Device-to-Device Out-of-Coverage Communication with Frequency Hopping Resource Scheduling",.

[TR25996] 3GPP TR 25.996 v6.1.0 "Spatial channel model for Multiple Input Multiple Output (MIMO) simulations"

[winnerfinal] D5.3 "WINNER+ Final Channel Models"

[winner] D1.1.2 V1.2 "WINNER II Channel Models"

[itur] Draft new Report ITU-R M.[IMT.EVAL] "Guidelines for evaluation of radio interface technologies for IMT-Advanced, Document 5/69-E."

[Balanis] C.A. Balanis, "Antenna Theory - Analysis and Design", Wiley, 2nd Ed.

[Chunjian] Li Chunjian, "Efficient Antenna Patterns for Three-Sector WCDMA Systems", Master of Science Thesis, Chalmers University of Technology, Göteborg, Sweden, 2003

[Calcev] George Calcev and Matt Dillon, "Antenna Tilt Control in CDMA Networks", in Proc. of the 2nd Annual International Wireless Internet Conference (WICON), 2006

[R4-092042a] 3GPP TSG RAN WG4 (Radio) Meeting #51, R4-092042, Simulation assumptions and parameters for FDD HeNB RF requirements.