Chapter 5
Cellular-Networks Simulation Using SimuLTE

Antonio Virdis, Giovanni Nardini, and Giovanni Stea

5.1 Introduction

With the advent of their fourth-generation deployment, known as Long Term Evolution (LTE), cellular networks have undergone a massive increase in popularity, due to their large bandwidth, ubiquitous coverage, and built-in features. More interestingly, they have progressively shifted from single-service to general-purpose access networks, capable of supporting diverse packet-based services simultaneously. Such a paradigm shift has been accompanied by a parallel one in the related research: research discussing physical-layer issues (i.e., waveforms, signal propagation models, receiver algorithms, coding and modulation, etc.) has lately been complemented by research related to Medium Access Control (MAC) layer issues, like resource allocation, admission control, user-side power saving techniques, performance guarantees, as well as the one dealing with the performance of services offered through cellular networks, from mobile web browsing to smart Internet of Things (IoT). Currently, cellular networks are being considered as a viable alternative to other technologies, such as WiFi for traditional mobile applications, IEEE 802.15.4 and Long Range (LoRa) [5] for sensors and smart things, IEEE 802.11p for vehicular networks, and Asymmetric Digital Subscriber Line (ADSL) for home Internet access. The main advantages of an LTE network are: being infrastructure-based and operator-managed, which relieves service users from the need of deploying and managing a (service-specific) infrastructure of their own, or making do with the lack of one. The fact that it operates on licensed spectrum, guaranteeing absence of external interference, and its built-in features for security, mobility management, and terminal-side power saving. Current trends,

A. Virdis · G. Nardini (✉) · G. Stea
University of Pisa, Pisa, Italy
e-mail: antonio.virdis@unipi.it; g.nardini@ing.unipi.it; giovanni.stea@unipi.it

A. Virdis, M. Kirsche (eds.), *Recent Advances in Network Simulation*,
EAI/Springer Innovations in Communication and Computing,
https://doi.org/10.1007/978-3-030-12842-5_5

such as the progression towards fifth-generation (5G) access and the standardization of Multi-access Edge Computing (MEC), all concur to foresee that the role of cellular networks in next-generation communications will increase, incorporating new key features like Device-to-Device (D2D) communications and a tighter coupling between communication and computation resources.

Evaluating the performance of cellular networks poses several challenges. The fact that LTE includes a whole stack of layered protocols, each one having buffers and timers, which interact with other features (such as power saving at both the base station and the terminal), intrinsically defies analytical modeling. On the other hand, building prototypes to do live measurements incurs non-trivial difficulties: despite the fact that several efforts have been done to realize open-source frameworks for LTE experimentation on Commercial Off-The-Shelf (COTS) hardware (e.g., OpenAirInterface [6]), licensed spectrum makes live experimentation difficult, and prototypes can only scale so much as for number of users and base stations, transmission and computing power, and available bandwidth. This leaves simulation as the ideal performance evaluation technique, trading some accuracy for a large gain in experiment manageability. Several simulators of cellular networks have been developed so far. Some are link-level simulators (e.g., [2, 4]). These do an extensive job of modeling the physical layer of a link, with little or no interest in what is above it. They are good for evaluating signal propagation, spectral efficiency, the impact of transmission techniques such as Multiple Input Multiple Output (MIMO) or beamforming, and interference management, but they are generally unsuitable to understand issues related to resource scheduling, protocol interaction, or application-level performance. A different approach is instead that of system-level simulators, where some modeling details (e.g., signal propagation) are abstracted in favor of a stronger focus on the interplay and communication among several complex submodels (e.g., protocol layers). Among system-level LTE simulators, LTE-EPC Network simulAtor (LENA) [1] is part of the Network Simulator 3 (ns-3) framework, and it focuses on the design and testing of Self-Organizing Network (SON) algorithms and solutions.

This chapter presents SimuLTE, a system-level simulator based on OMNeT++ for 4G LTE and Long Term Evolution Advanced (LTE-A) networks. SimuLTE is especially focused on the LTE data plane. Most of the control-plane functions are abstracted by using an oracle, called the *Binder*, which the various modules can query to obtain information which would otherwise require control-plane protocols and elements. SimuLTE includes all the protocols of the LTE stack. It models the effects of the physical layer by computing the received Signal-to-Interference-plus-Noise-Ratio (SINR) at receivers, taking into account all the simultaneous transmitters. This allows one to use it to test, for instance, interference-coordination schemes. It allows both infrastructure-mode communications, where the endpoints of communications are always one User Equipment (UE) and the evolved Node B (eNB), and network-assisted D2D communications, where both endpoints are UEs, and the eNB is in charge of resource scheduling. For the latter, both one-to-one and one-to-many communications are modeled. Within SimuLTE, the LTE functions are confined to a Network Interface Card (NIC), to facilitate the setup of

mixed scenarios, where LTE coexists with other layer-2 technologies (like WiFi), or to use LTE as a layer-2 technology in application scenario simulations (e.g., communicating vehicles).

The rest of this chapter is organized as follows: Sect. 5.2 describes the cellular-networks background, to introduce the reader to the necessary terminology and acronyms. Section 5.3 describes the modeling of an LTE network done within SimuLTE. Section 5.4 presents two tutorials on hot topics in LTE research: interference-coordination techniques at the MAC-level and D2D communications, respectively.

5.2 Background

An LTE network is composed of the Evolved Packet Core (EPC) part and the Radio Access Network (RAN) part, as shown in Fig. 5.1. The EPC is an Internet Protocol (IP)-based network that includes entities performing core functionalities for the network operator, such as Mobile Management Entity (MME), Home Subscriber Server (HSS), and Serving Gateway (SGW). The Packet Data Network Gateway (PGW) provides the EPC with the connectivity to the Internet. The RAN is composed of base stations, called eNBs, which are in charge of resource allocation, and UEs, e.g., smartphones or any device capable of connecting to an LTE network. Downlink (DL) transmission occurs from the eNB to the UEs, and Uplink (UL) ones in the opposite direction. As far as data-plane transmissions are concerned, in the DL an eNB receives IP packets from the EPC, processes them through the LTE protocol stack, which includes fragmentation and reassembly, and sends them on the air.

While LTE occupies the layer 2 of the Open Systems Interconnection (OSI) stack, its functions are split among three protocols, namely (from top to bottom) the Packet Data Convergence Protocol (PDCP), the Radio Link Control (RLC), and

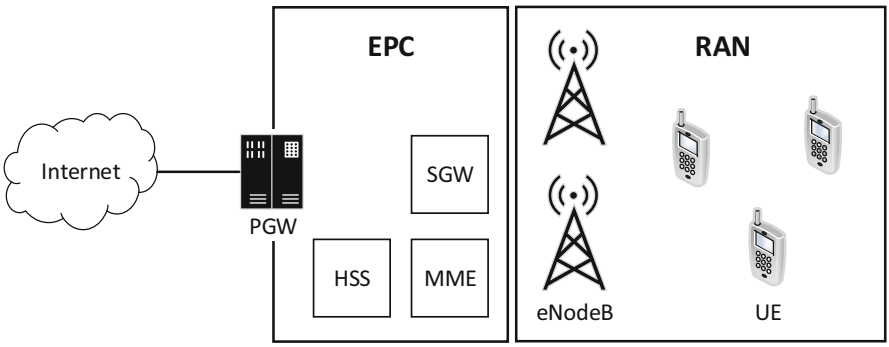

Fig. 5.1 Architecture of the LTE network

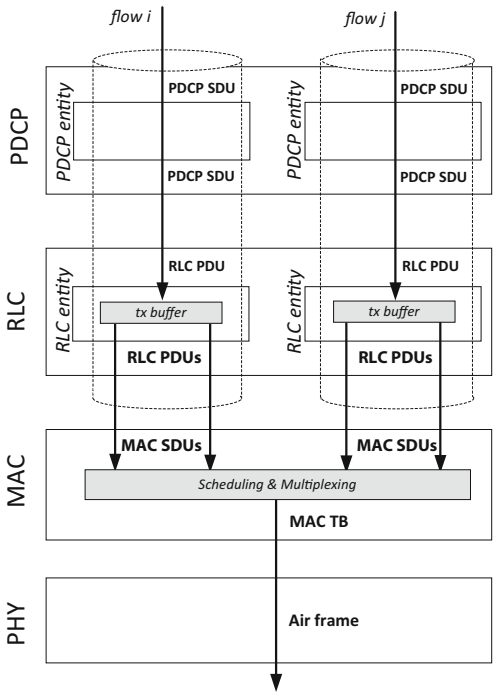

Fig. 5.2 Top-down traversal of the LTE protocol stack

the MAC. The downstream flow of data within the LTE stack, i.e., the one that an IP packet would undergo on transmission, is shown in Fig. 5.2. At each layer, data comes from the upper layer in the form of Service Data Units (SDUs) and goes to the lower layer as Protocol Data Units (PDUs). A PDCP entity maintains numbering and ciphering information, among others. Each PDCP SDU is assigned a PDCP Sequence Number (SN) and ciphered so that only the peering PDCP entity can decode it. Data from the PDCP is sent to the RLC. It is buffered in the RLC transmission buffer, until sent down in the form of RLC PDUs, upon request from the MAC layer. A flow is allocated to a PDCP and a RLC entity at a node, and its PDCP/RLC entities at the transmitter and receiver are synchronized. The RLC can work in one of three modes: Transparent Mode (TM), Unacknowledged Mode (UM), or Acknowledged Mode (AM). The first one does not perform any operation; hence, a RLC SDU corresponds exactly to a RLC PDU. The second one performs segmentation/concatenation of SDUs on transmission, as well as reassembly, duplicate detection, and reordering of PDUs on reception. The third one adds an Automatic Repeat-reQuest (ARQ) mechanism on top of that to ensure reliable delivery of RLC PDUs.

The MAC layer performs scheduling and multiplexing of RLC PDUs coming from different flows and encapsulates them into MAC Transport Blocks (TBs), which are sent over the physical layer. At every Transmission Time Interval (TTI), which is 1 ms in LTE, the MAC scheduler at the eNB assembles the DL and the

UL subframes. The latter include a fixed number of Resource Blocks (RBs), e.g., 50 in a 10 MHz deployment. In the DL, the MAC scheduler selects which UEs will receive information, how large their TB will be, and which RBs it will occupy. The number of bits carried in each RB depends on the Modulation and Coding Scheme (MCS) used by the eNB for that TB. The MCS is determined by the Channel Quality Indicator (CQI) reported by the UE, which depends on the measured SINR. A higher CQI warrants more bits per RB, hence a higher throughput. The MAC layer includes a Hybrid Automatic Repeat-reQuest (H-ARQ) function for error recovery: a receiving UE acknowledges (via ACKs) (or disacknowledges via NACKs) a MAC TB four TTI after its transmission, and the eNB can schedule a predefined number of retransmissions at any future TTI (asynchronous H-ARQ).

UL scheduling mirrors the DL one: the eNB allocates transmission grants to UEs having backlogged data, specifying which RBs they can use, using which modulation. However, since data is physically stored at the UEs, a signaling method is needed for a UE to signal to the eNB its intention to transmit. UEs can append quantized Buffer Status Reports (BSRs) to their scheduled data, to inform the eNB of their residual backlog. UEs can also signal the presence of backlog using an out-of-band Random Access (RAC) procedure. Simultaneous RAC requests from different UEs may collide, in which case UEs undergo a backoff procedure before attempting another RAC. The eNB responds to a RAC request by scheduling the UE in a future TTI. If unanswered, RAC requests are reiterated. The handshake for UL transmissions is summarized in Fig. 5.3. It consists of up to three parts: a RAC request, usually followed by a (small) grant that the eNB gives to the UE, large enough to contain a BSR, and finally a (larger) grant that the eNB can give to the UE once it knows its backlog.

Unlike the DL ones, H-ARQ processes are synchronous, i.e., retransmissions are scheduled exactly eight TTIs after the previous attempt.

In the latest LTE-A releases, new functions have been introduced to address increasing traffic demand and the requirements of new services (e.g., IoT and vehicular communications). For example, Coordinated MultiPoint (CoMP) transmission and reception and D2D communications are two of the main innovations. CoMP addresses the problem of mitigating the effects of inter-cell interference, since all the eNBs in an LTE network share the same spectrum. In particular, CoMP Coordinated Scheduling (CoMP-CS) and CoMP Joint Transmission (CoMP-JT) techniques aim at improving the performance of cell-edge UEs. This is achieved by allowing neighboring eNBs to exchange coordination information using a common

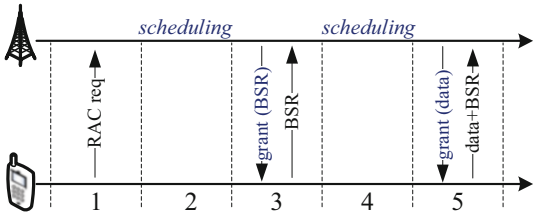

Fig. 5.3 Handshake for the scheduling of uplink UE traffic

logical interface, called *X2*. On the other hand, D2D allows UEs in proximity to communicate directly, without traversing the conventional infrastructure path through the eNB (i.e., one UL and one DL transmission). This allows the UEs to reduce latency and exploit better link quality. Although the standard focuses on *one-to-many* D2D communication (one D2D transmission can be decoded by all UEs in proximity), *one-to-one* D2D communication between two UEs is also envisaged, especially in the research community. In network-controlled D2D, control functions are still handled by the eNB: considering resource allocation as an example, UEs have to request transmission grants from the eNB, using the same handshake procedure as for standard UL communication. Given the short distance between the endpoints of D2D communications (possibly reducing UEs' transmission power), the eNB can also allocate the same RBs to multiple D2D flows simultaneously if their mutual interference is low, thus enabling *frequency reuse* and serving more traffic with the same spectrum.

5.3 Structure of the SimuLTE Simulator

In this section, we describe the general structure of SimuLTE. We first describe the main nodes composing a simulated LTE network in terms of their structure and interconnections. We then move to the core of the architecture, i.e., the NIC card, which implements the actual communication among nodes. Finally, we discuss the available main functionalities, detailing where they are located in the codebase and providing insights on design choices.

Figure 5.4 shows a simplified view of the project's folder structure. The *simulations* and *test* folders contain exemplary simulations together with the respective networks and fingerprint-based tests, which are used for verification purposes. The

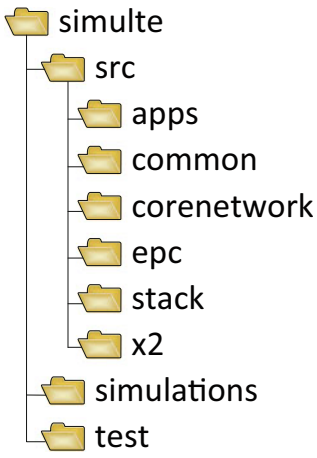

Fig. 5.4 Simplified representation of the SimuLTE-Project folders structure

src folder contains all the source code and is further divided into subfolders, which identify specific portion of the codebase. The *apps* folder contains four traffic models, for Constant Bit Rate (CBR), Voice-over-IP (VoIP), Video on Demand (VoD), and alert traffic. The *common* folder includes utility functions and definitions of LTE-related parameters. The *corenetwork* folder contains the definition of all nodes in the simulator, which are described in detail in the next section. *epc* and *x2* folders contain the definition of the entities involved, respectively, in the EPC and in X2 communications. Finally, the *stack* folder contains the files defining the structure of the LTE NIC card and all its internal layers and modules, as discussed in Sect. 5.3.2.

5.3.1 Nodes

As we explained in Sect. 5.2, the two main nodes of a simulated LTE network are eNBs and UEs, which handle all data-plane traffic through the LTE network. SimuLTE adds a third node, namely, the Binder, which acts as the "oracle" of the network, keeping track of communication associations, and which is also used to abstract control-plane operations. Nothing prevents one to implement some of these control procedures through the exchange of control messages in the LTE network, in any case. Finally, one or more nodes can optionally be added to realize EPC-related operations, e.g., for handover management. The general structure of a simulated network is shown in Fig. 5.5.

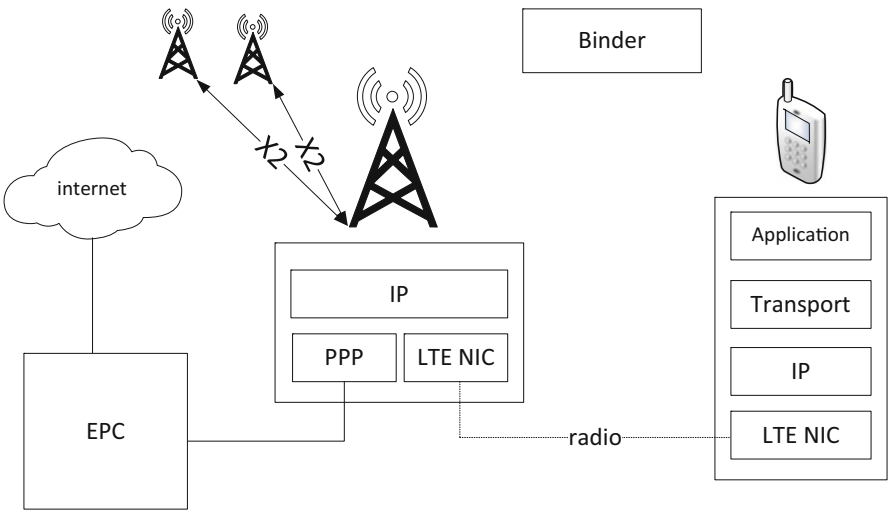

Fig. 5.5 High-level view of the main simulator nodes

5.3.1.1 Evolved Node B

The eNB is a communication relay for each transmission to/from UEs. It has a Point-to-Point Protocol (PPP) interface towards the EPC, and an LTE NIC card to communicate with UEs. Moreover, there is an array of PPP interfaces that implement the X2 connections with neighboring eNBs. The architecture of the X2 interface will be discussed in more detail in Sect. 5.3.3.2. The eNB does not generate traffic on the data plane, thus does not contain any application-layer module for communication with UEs.

5.3.1.2 User Equipment

The UE node models the behavior of an LTE-based cellphone. Its structure is inspired from INET's `StandardHost`, having multiple Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) applications, UDP and TCP modules implementing the respective transport layer protocols. Each TCP/UDP application is one end of a connection, the other end of which may be located within another UE or anywhere else in the simulated network. SimuLTE provides models of real-life applications (e.g., VoIP and VoD), but it can include any TCP/UDP-based OMNeT++ application. The IP module is taken from the INET package as well. Finally, an LTE NIC card is used for communication with the eNB or with other UEs in case of D2D communications.

5.3.1.3 Binder

The Binder is a simple module that stores information about every LTE-related node within the system. Its main purpose is to cover all the operations that are not modeled as a message exchange, either to simplify the model or to speed up the simulation process. This includes references to nodes, communication associations among eNBs and UEs, peering associations for D2D pairs, etc.

Each module registers to the Binder during its `initialize()` function. Listing 5.1 shows an exemplary registration process performed by the `IP2LTE` submodule (which is explained in Sect. 5.3.2.5) residing in either a UE or an eNB. In both cases, the module hierarchy is navigated to obtain a pointer to the module. The `getBinder()` function is a utility function that returns a pointer to the Binder. The `registerNode()` function is a member function of the Binder class. It stores the OMNeT++ Identifier (ID) of the module and associates it with its node type, its serving eNB specified via the Network Topology Description (NED) language in case of UE registration, and a unique `macNodeId`. The latter value will be used to obtain information on the node during the simulation.

Listing 5.1 Example of a node registration to the Binder within the `IP2LTE` module

```
if (nodeType_ == UE)
{
    cModule *ue = getParentModule()->getParentModule();
    getBinder()->registerNode(ue, nodeType_, ue->par("masterId"));
}
else if (nodeType_ == ENODEB)
{
    cModule *enodeb = getParentModule()->getParentModule();
    MacNodeId cellId = getBinder()->registerNode(enodeb, nodeType_);
}
```

5.3.2 LTE NIC

The LTE NIC module implements the LTE stack within eNBs and UEs. Figure 5.6 shows its internal structure and connections with other modules, namely, one between the UE and the eNB through an air channel and one with an IP module. The LTE NIC module is built by extending INET's `IWirelessNic` interface, to ease its deployment into INET nodes. This is one of the key points that allows the creation of complex scenarios, e.g., where LTE NICs are included in cars (e.g., Chap. 11) or used to build hybrid connectivity scenarios (e.g., Chap. 13), where nodes are equipped with both WiFi and LTE interfaces. With reference to Fig. 5.6, each of the NIC submodules models a corresponding part of the LTE protocol stack. Following the OMNeT++ paradigm, data-plane communications between the different layers of the protocol stack take place only via message exchange, hence ensuring a tight control over module interactions. The NIC structure is the same for both UE and eNB, the only exception being the `FeedbackGenerator`, only implemented in the UE. It is responsible for creating channel feedback that is then managed by the Physical Layer (PHY) module.

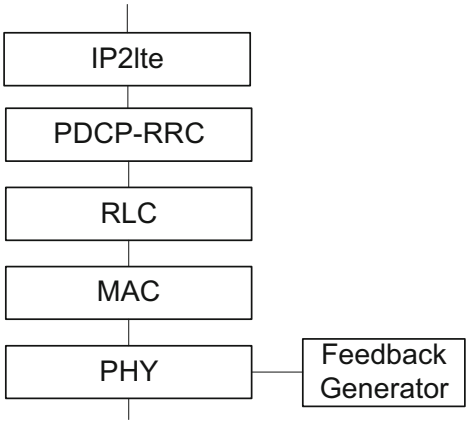

Fig. 5.6 Internal structure of the LTE NIC module

Each layer is implemented as a base module, specifying interfaces and main parameters. Base modules are then inherited to define specific versions for eNB and UEs, with additional parameters and gates. The same approach is followed for the implementation of the module behavior via C++ classes. In the following subsections, we describe each submodule and its functionalities.

5.3.2.1 PHY

The PHY module resides at the bottom of the LTE protocol stack and implements physical layer related functions, such as channel-feedback reporting and computation, simulation of the air channel, and data transmission and reception. It also stores the node's PHY parameters, like the transmission power and the antenna profile (i.e., whether transmissions are omni-directional or anisotropic). This control over transmission parameters allows one to define so-called heterogeneous scenarios, composed of macro-, micro-, and pico-eNBs, each one having its one radiation profiles.

PHY modules at both the eNB and the UE are associated with a module that models the behavior of the physical channel as perceived by the node itself. Channel modeling is implemented in a hierarchical manner: first, a base module called `LteChannelModel` defines two main functions `getSINR()` and `error()`. The first one, `getSINR()`, should compute the SINR of a given transmission. The second one, `error()`, should check if a packet has been corrupted during transmission. In the following, we describe the implementation of this module, which is available in the current implementation of SimuLTE. However, one can easily obtain its own version of the channel model by implementing the above interface.

Part of the tasks related to physical-layer procedures on the UE side is handled by a module called `FeedbackGenerator`. The latter measures the status of the channel and generates feedback, which is then sent to the eNB in the form of CQIs. The feedback-generation process can be configured to work aperiodically, i.e., on demand from the eNB, or periodically, with a configurable period. The feedback-generation process models channel measurements using the functions provided by the `LteChannelModel` module, in particular the `getSINR()` function. This function returns a vector of SINR values, one for each RBs available in the system bandwidth, which are then processed to generate CQIs. These can be either *wideband*, one value for the whole bandwidth, or *per-band*. In the latter case, the user can configure the number of RBs that compose a band.

As discussed above, the physical LTE channels, such as the Physical Uplink Control Channel (PUCCH), the Physical Downlink Control Channel (PDCCH), and the Physical Random Access Channel (PRACH), are not modeled down to the level of Orthogonal Frequency Division Multiplex (OFDM) symbols. We implemented the functions associated to said channels either via control messages sent between the eNB and UE nodes or via function calls through the Binder. However, any limitation or constraints imposed to the system by those channels (e.g., max. number

of UEs that can be scheduled simultaneously on the PDCCH) can be simulated by imposing constraints on the simulated message flow. Our modeling choice is to limit both memory and processor usage while retaining a good level of simulation accuracy.

As far as data flow is concerned, in the downstream MAC-PDUs received from the MAC layer are encapsulated in packets of the type `LteAirFrame`. Packets are marked with a type (i.e., data or control), a set of transmission parameters (e.g., transmission power and position, number of used RBs, etc.), and are sent to the destination module using the `sendUnicast()` function. In the upstream, instead, a received `LteAirFrame` is checked for its type and processed consequently: control packets are assumed to be correctly received and are forwarded to the upper layer. Data packets are instead checked for correct reception using the `error()` function of `LteChannelModel`, then decapsulated and sent to the upper layer. Note that while decapsulated packets are always forwarded upstream, they are marked with the result of the `error()` function, which will be then evaluated at the upper layers.

SimuLTE provides an implementation of the `LteChannelModel` base module, which is called `LteRealisticChannelModel`. In this implementation, each value of the SINR is computed as:

$$\text{SINR} = P_{RX}^{\text{eNB}} \Big/ \left(\sum\nolimits_i P_{RX}^i + N\right), \tag{5.1}$$

where P_{RX}^{eNB} is the power received from the serving eNB, P_{RX}^i is the power received from the ith interfering eNB, and N is the Gaussian noise. Furthermore, P_{RX} is computed as $P_{RX} = P_{TX} - P_{loss} - F - S$, where P_{TX} is the transmission power, P_{loss} is the path loss due to the eNB-UE distance, which also depends on the frequency where the considered RB lies, and F and S are the attenuation due to fast and slow fading, respectively [3]. Computing the SINR on each RB allows one to take into account interference on a per-RB basis, e.g., taking into account the transmissions from neighboring eNBs. This can be used to evaluate interference-coordination mechanism, notably the CoMP.

The `error()` function, instead, evaluates the correct `LteAirFrame` reception by analyzing the CQI used at transmission and the current channel status. These two values are used together with a set of realistic Block Error Rate (BLER) curves to obtain an error probability $X \in [0, 1]$. Finally, a uniform random variable P_{err} is sampled and the packet is assumed to be corrupted if $X < P_{err}$.

As stated above, custom models of the channel can be created and used within SimuLTE. Two main approaches are possible in this respect:

1. Create a new module that extends the `LteChannelModel` base module, i.e., redefining the `getSINR()` and `error()` functions.
2. Create a new module that inherits from `LteRealisticChannelModel`, redefining the functions for path loss, fading, and shadowing computation.

5.3.2.2 MAC

Most of the intelligence of LTE nodes is implemented in the MAC module. The main tasks of this module comprise buffering packets coming from lower layers (PHY) and requesting data for transmission from upper ones (RLC), encapsulating MAC-SDUs into MAC-PDUs and vice versa, handling and storing channel feedback, performing scheduling, and Adaptive Modulation and Coding (AMC). Most of the operations related to the flow of packets are the same at the UE and the eNB. Scheduling and channel-feedback management, instead, are performed differently on the two nodes.

Figure 5.7 shows a high-level view of the layer structure. The main functions of the MAC layer are executed periodically at a 1 ms pace. This behavior is implemented by the `LteMacBase` module, which schedules a self-message each millisecond and handles it via the `handleSelfMessage()` function. The latter is redefined to obtain specialized versions for the eNB and UE, respectively, by the `LteMacEnb` and `LteMacUe` classes. Figure 5.8 shows an overview of the operations performed by the `handleSelfMessage()` function in the two cases. They both start with the decapsulation of the successfully decoded PDU from the H-ARQ buffers in reception. In the upstream, MAC-PDUs coming from the PHY are stored into H-ARQ buffers, where they are then checked for correctness, decapsulated, and then forwarded to the RLC in the form of MAC-SDUs.

On the eNB side, UL connections are scheduled for transmission according to a configurable policy. Scheduling decisions are notified to the UEs via grant messages, i.e., the PDCCH is not simulated. Similarly, DL scheduling is performed by selecting which connection to serve, and how much data to send. For each scheduled connection, MAC-SDUs are then requested to the RLC layer and encapsulated into MAC-PDUs. The latter are finally stored in the H-ARQ buffers and forwarded to the PHY for transmission. The structure of H-ARQ buffers will be explained later on.

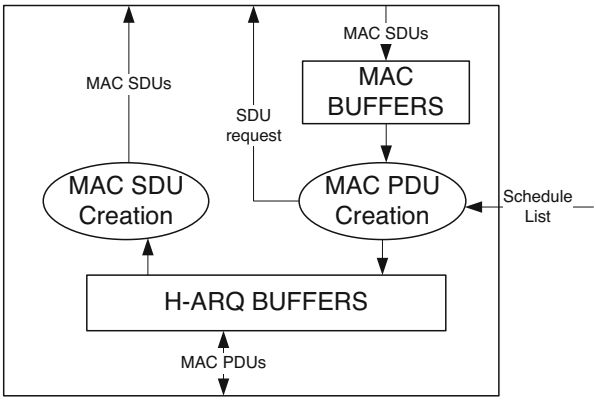

Fig. 5.7 High-level view of the MAC layer structure

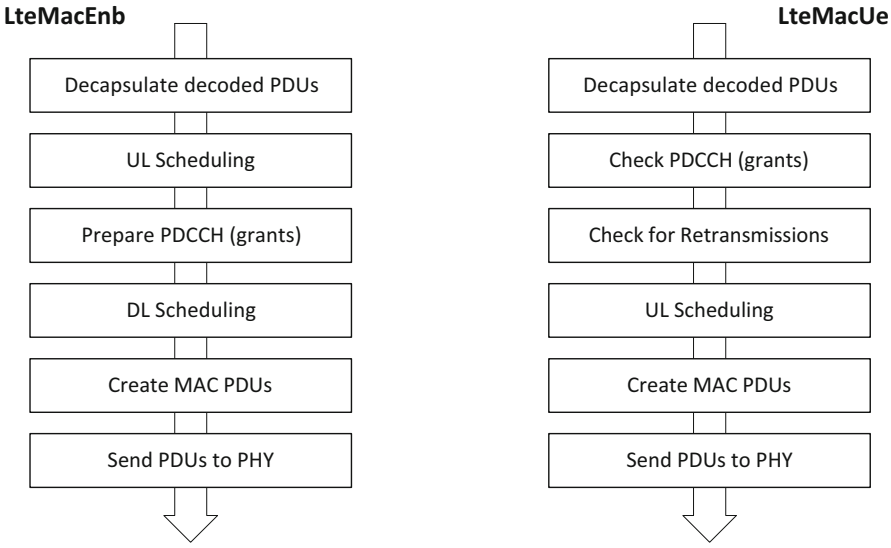

Fig. 5.8 Main MAC-level operations

On the UE side, instead, each UE checks if any grants have been received and decides which local connection will be able to use the granted resources, if any. If no resources are available, it will perform a resource request via the RAC procedure, which is again implemented through messages generated by the MAC module. Then, the UE will check its H-ARQ buffers to see if any transmission is expected for this TTI[1] and will proceed with the scheduling of new transmissions otherwise.

H-ARQ buffers are used to store MAC-PDUs that are being sent and received. There is one set of such buffers for transmissions and one for receptions, to which we will refer as *TxHbuff* and *RxHbuff*, respectively, and as *Hbuff* to denote both.

A transmitted MAC-PDU is stored in the *TxHbuff* until it is received correctly or the maximum number of retransmissions is reached. Correct reception is notified via H-ARQ feedback messages. A received MAC-PDU is instead stored in the *Rxbuff* until its decoding process has been completed. On the eNB side, H-ARQ buffers store MAC-PDU information for each H-ARQ process, for each connected UE in both downlink and uplink. In Fig. 5.9 we show the general structure of a *Hbuff*. The latter contains K buffers, one for each active connection to another node. An eNB has thus as many buffers as connected UEs in each direction, whereas a UE has up to one per direction, as it communicates directly only with its serving eNB. This architecture is slightly modified in case of D2D communications, where a UE has one additional buffer for each peering UE.

[1] Please note that UL transmissions are synchronous.

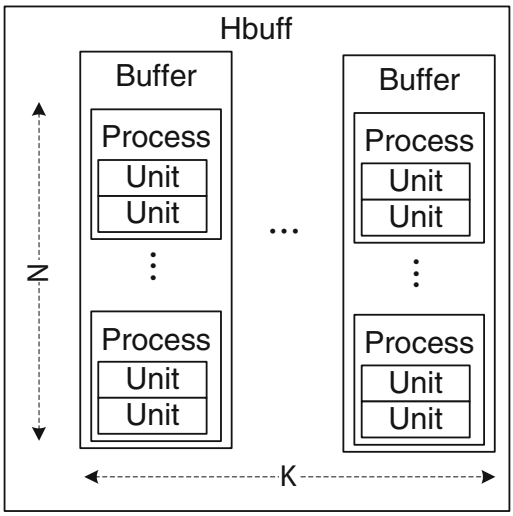

Fig. 5.9 Internal structure of H-ARQ buffers

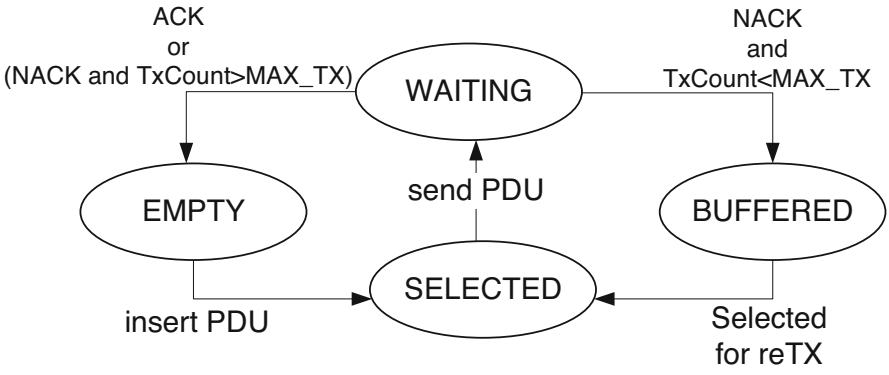

Fig. 5.10 High-level view of H-ARQ operations

Each buffer is composed of N processes (usually $N = 8$), one per H-ARQ process, and each process contains two *units*, to support single-user MIMO. Units are the actual data structure containing the information related to a transmitting/receiving MAC-PDU. The way unit status is stored depends on the feedback management procedure, resulting in different data structures for *Tx*- and *Rx-Hbuffs*. Figure 5.10 shows the finite-state automaton for transmissions (the one for receptions is similar, mutatis mutandis). Acknowledgment (ACK) and Negative-Acknowledgment (NACK) are the reception of the corresponding H-ARQ feedback. $TxCount$ is the transmission counter, which is increased in the `SELECTED` state and then reset in the `EMPTY` one. $maxTx$ is the maximum number of transmissions before a PDU is discarded.

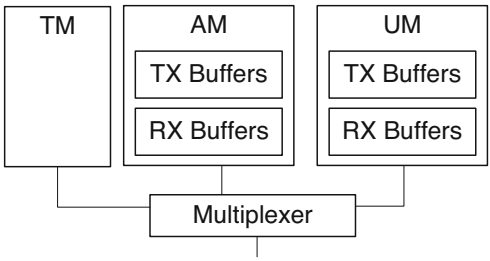

Fig. 5.11 High-level view of the architecture for the three RLC modes

5.3.2.3 RLC

This module implements RLC operations, which are identical for the eNB and the UE. It performs multiplexing and demultiplexing of MAC-SDUs to/from the MAC layer, implements the three RLC modes TM, UM, and AM as defined in 3GPP-TS 36.322, and forwards packets from/to the PDCP-Radio Resource Control (RRC) to/from the proper RLC mode entity. Figure 5.11 shows the general structure of the RLC module. As we can see, there is one different gate for each RLC mode, which are connected towards the PDCP-RRC one. The TM submodule forwards packets transparently and has no buffer, whereas AM and UM have their own set of transmission and reception buffers, one for each connection associated to the according RLC mode.

5.3.2.4 PDCP-RRC

The PDCP-RRC module models operations of the highest layer of the LTE protocol stack. It receives data from the `IP2LTE` module (in the downstream direction) and from the RLC one (in the upstream). In the first case, the PDCP-RRC performs a RObust Header Compression (ROHC) of the received packet and assigns it a Logical Connection Identifier (LCID). The latter uniquely identifies the connection to which the packet belongs. It is obtained from the 4-tuple composed of <*sourceIPAddr, destIPAddr, sourcePort, destPort*>. The packet is then encapsulated in a PDCP-PDU and forwarded to the proper RLC port, according to the selected RLC mode. In the upstream, a packet coming from the RLC is first decapsulated, then its header is decompressed, and the resulting PDCP SDU is finally sent to the upper layer.

5.3.2.5 IP2LTE

The `IP2LTE` module acts as an interface between the network layer (i.e., IP) and the LTE NIC. In the downstream, it receives layer-3 datagrams and extracts both source/destination IP addresses and port numbers. The latter are written into a `ControlInfo` object that is attached to the message before it is sent to the lower layers, as shown in Listing 5.2. This allows the PDCP-RRC module to obtain

the above 4-tuple without inspecting the packet and to associate a LCID to the flow, as explained in the previous section. In the upstream, `IP2LTE` only forwards the message from PDCP-RRC to the network layer without further processing. Moreover, the `IP2LTE` is responsible for registering the LTE NIC to the Binder and to the interface table of the network layer during the initialization.

Listing 5.2 Creation of the `ControlInfo` object

```
1 FlowControlInfo *controlInfo = new FlowControlInfo();
2 controlInfo->setSrcAddr(srcAddr.getInt());
3 controlInfo->setDstAddr(destAddr.getInt());
4 controlInfo->setSrcPort(srcPort);
5 controlInfo->setDstPort(dstPort);
6 [...]
7 datagram->setControlInfo(controlInfo);
8 send(datagram,stackGateOut_);
```

5.3.3 Main Functions

We now discuss the scheduling, inter-eNB, and D2D-communication operations.

5.3.3.1 Scheduling

As discussed in Sect. 5.2, resource scheduling is the process of deciding how to allocate RBs to UEs, in both the DL and UL subframes. This process is realized at the MAC layer and performed by the eNBs on each TTI. In SimuLTE, scheduling operations are implemented at the eNB by the `LteSchedulerEnb` class, which defines all the operations that are common to the DL and UL, such as data structure initialization, allocation management via the `Allocator`, and statistics collection. The `Allocator` is a C++ class that keeps the information about the RBs' occupation. The core of this class is the `schedule()` function, whose main operations are shown in Fig. 5.12. These are common to both the DL and UL scheduling. First, the data structures containing the allocation decisions of the previous TTI are cleared, and the per-UE modulation and coding information is updated using the most recent channel information. Then, the actual scheduling is performed, processing RAC requests (UL only), retransmissions, and first transmissions. The way these two last operations are performed defines the scheduling policy, i.e., how contention among UEs is managed. Finally, statistics on the allocation (e.g., the number of allocated RBs) are computed and emitted to the simulation environment.

The scheduling hierarchy is shown in Fig. 5.13 (left part of the figure). The `LteSchedulerEnbUl` and `LteSchedulerEnbDl` classes extend the base class `LteSchedulerEnb` by implementing the `rtxSchedule()` method, which handles RAC requests and manages retransmissions. Scheduling policies are instead implemented by extending the `LteScheduler` class. The latter's

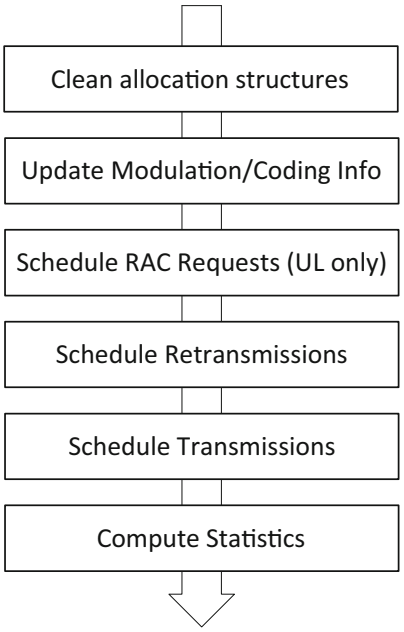

Fig. 5.12 Depiction of the main scheduling operations

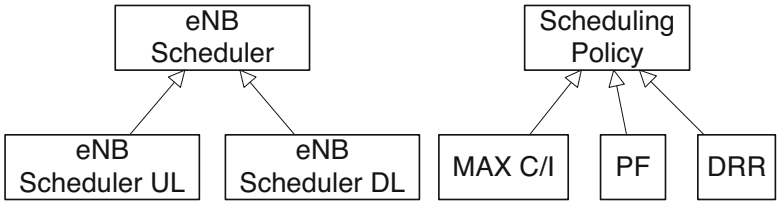

Fig. 5.13 High-level representation of the scheduling hierarchy

main function, called `prepareSchedule()`, is responsible for applying the scheduling policy on backlogged connections and building a schedule list that associates each connection to the number of allocated RBs. To build the schedule list, the scheduling policy examines backlogged connections one at a time and polls the `Allocator`, via the `requestGrant()` member function, to check whether there are RBs available for the given connection. The latter obtains the amount of bytes that can be transmitted in a single RB for that connection from the AMC module. Finally, the schedule list is passed to the MAC layer, which enforces it by fetching the data from the connections' RLC buffers and constructing the MAC-PDUs.

Three well-known scheduling policies are included in the current release, namely, Maximum Carrier-over-Interference (MaxC/I), Proportional Fair (PF), and Deficit Round Robin (DRR), as depicted in Fig. 5.13 (right part of the figure). The scheduling policy can be modified by changing the `schedulingDisciplineDl` and `schedulingDisciplineUl` parameters of the MAC layer, as follows:

```
1 *.eNodeB.lteNic.mac.schedulingDisciplineDl = "MAXCI"
2 *.eNodeB.lteNic.mac.schedulingDisciplineUl = "MAXCI"
```

As far as RBs are concerned, SimuLTE allows one to group RBs into logical bands, i.e., logical groups of RB that are considered as the minimum scheduling unit by the scheduler. Let us consider the following exemplary configuration:

```
1 **.deployer.numRbDl = 20
2 **.deployer.numRbUl = 20
3 **.deployer.numBands = 20
```

The first two lines define the number of available RBs in the DL and UL subframes, respectively. The third one defines the total number of logical bands among which the RBs are divided into. In this case, we will end up with 20 logical bands with 1 RB each, i.e., one RB for each band. The scheduling policy will work on bands rather than the RBs, different mappings can hence be used to modify the way the scheduler accesses resources.

Moreover, the scheduling policy can ask the `Allocator` to restrict the set of RBs that can possibly be allocated to a connection, by using the `BandLimit` concept. The latter is a data structure stored at the MAC layer of every eNB that specifies, for each UE and RB, the amount of bytes (if any) that can be allocated to that UEs connections in that RB. This concept can be exploited when enforcing interference-coordination mechanisms, for which we provide an example in Sect. 5.4.1.

5.3.3.2 Inter-eNB Communications

Interactions among eNBs are crucial to support functionalities like handover and interference coordination, which are the subject of several research works. eNBs interact via the X2 interface, which is modeled in SimuLTE. Within the eNB, we model the X2 protocol stack depicted in Fig. 5.14, where the `LteX2App` handles the communication with one peering eNB and runs on top of Stream Control Transmission Protocol (SCTP) as the transport protocol. The task of the `LteX2App` is to pass messages originated from the LTE NIC to the peering eNB and vice versa. The two directions are managed by two different inner modules, namely, `X2AppServer` and `X2AppClient`. If the eNB peers with multiple eNBs, one `LteX2App` is needed for each connection. `LteX2App` is transparent to the kind of messages to be sent on behalf of modules within the LTE protocol stack. Within the latter, `X2User` entities are base modules that can be extended in order to implement

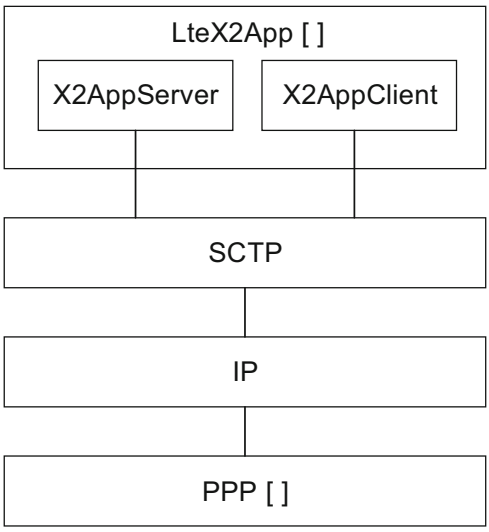

Fig. 5.14 High-level view of the X2 stack

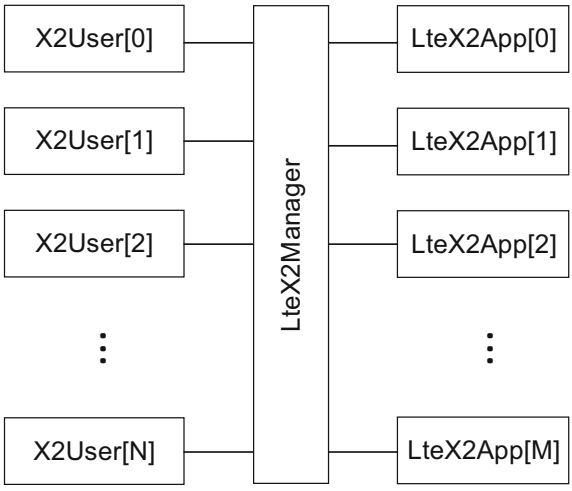

Fig. 5.15 High-level view of the X2 manager

a functionality that exploits the X2 to perform its task. `LteX2App` and `X2User` modules are transparent to each other and the interface between them is provided by the `LteX2Manager`, as shown in Fig. 5.15. Listing 5.3 shows the code of its `handleMessage()` function, which calls different handlers based on the origin of the message.

Listing 5.3 The role of the `LteX2Manager`

```
void LteX2Manager:handleMessage(cMessage *msg)
{
    cPacket* pkt = check_and_cast<cPacket*>(msg);
    cGate* incoming = pkt->getArrivalGate();
    if (strcmp(incoming->getBaseName(), "dataPort") == 0) // from LTE stack
    {
        EV << "LteX2Manager:handleMessage - Received message from LTE stack" <<
              endl;
        fromStack(pkt); // call handler
    }
    else  // from X2
    {
        int gateIndex = incoming->getIndex();
        EV << "LteX2Manager:handleMessage - Received message from X2, gate " <<
              gateIndex << endl;
        fromX2(pkt); // call handler
    }
}
```

As an example, consider the CoMP-CS function implemented within SimuLTE: in order to mitigate possible inter-cell interference, neighboring eNBs exchange scheduling information via X2 and avoid allocating the same RBs. This is done by the `LteCompManager`, which extends the `X2User` module. Therefore, the `LteCompManager` can interact with the LTE protocol stack via direct method calls and then sends its messages to the `LteX2Manager`, which in turn passes them to the peering eNBs. According to the architecture explained above, if eNB *i* has to send a CoMP message to neighboring eNB *j* and *k*, the `LteCompManager` only needs to send one message to the `LteX2Manager` including the list of destinations. The `LteX2Manager` then identifies the `LteX2App` modules handling the X2 connections to *j* and *k* and passes one copy of the message to each for the transmission over the X2 interface.

5.3.3.3 D2D Operations

SimuLTE supports both *one-to-one* and *one-to-many* D2D communications. In the one-to-one case, a D2D flow consists of a peering connection between two UEs. The Binder keeps a data structure containing the peering relationships between D2D-capable UEs, i.e., which pairs of UEs can communicate directly. For each pair of D2D endpoints, the Binder also stores whether the flow actually uses the direct path or the conventional infrastructure path through the eNB. The communication mode can be either static or selected dynamically by the `D2DModeSelection` module residing within each eNB. On the other hand, there are no peering relationships for one-to-many communications. Messages transmitted by a UE include a `multicastGroupID` field and only UEs enrolled within the multicast group can try to decode the transmission.

Processing of D2D flows is carried out by specialized, D2D-enabled versions of the LTE NIC (both the UE and the eNB sides), which inherit functionalities from the base ones. To this aim, `LteNicUe` and `LteNicEnb` modules are extended by `LteNicUeD2D` and `LteNicEnbD2D`, respectively. The latter, in turn, defines

specialized versions of each layer, e.g., `LteMacUeD2D` and `LteMacEnbD2D` for the MAC layer.

At the PHY layer, `getSINR_D2D()` and `error_D2D()` functions are provided to compute the SINR and check transmission errors of D2D flows. As far as scheduling is concerned, frequency reuse among D2D flows is accomplished by extending the `Allocator` module so that it can associate more than one UE to each allocated RB. Frequency-reuse-enabled scheduling policies can then be implemented on top of this general structure. Moreover, the eNB also needs to know whether to schedule H-ARQ retransmissions for D2D flows, although (N)ACKs are exchanged between UEs without involving the eNB. Thus, we allow the receiving UE to send a copy of the (N)ACKs to the eNB, too. The MAC layer of the eNB uses such information to update a data structure that *mirrors* the status of each *TxHbuffs*. RLC layer does not need additional D2D-related functionalities, whereas the PDCP layer takes care of associating downstream packets to either D2D or infrastructure mode exploiting the Binder.

5.4 Tutorials

We now describe two case studies, namely: (1) an analysis of the problem of inter-eNB interference coordination and (2) the mode selection for D2D. We use a tutorial approach in both cases, starting with the problem statement, then guiding the reader to the simulation-setup process, including the network definition and the parameters configuration, and prompting some possible modifications to the code.

5.4.1 Tutorial 1: Interference Coordination

We consider the downlink of a multicell network, where eNBs serve their UEs while sharing frequency resources. In such a scenario, CoMP-CS can be enforced to mitigate inter-cell interference. In this context, eNBs participating in the coordination send their load information to one eNB, chosen as coordinator, through the X2 interface. In turn, the coordinator periodically runs an algorithm to assign non-overlapping sets of usable RBs to each eNB and communicates them which RBs they can/cannot use in the subsequent scheduling operations. Thus, the objective of this tutorial is to show how to configure X2 connections and change the coordination algorithm implemented by the eNBs.

5.4.1.1 Network Definition

We consider the `MultiCell_X2Mesh` network depicted in Fig. 5.16, which is composed of three eNB connected to each other via the X2 interface. In particular,

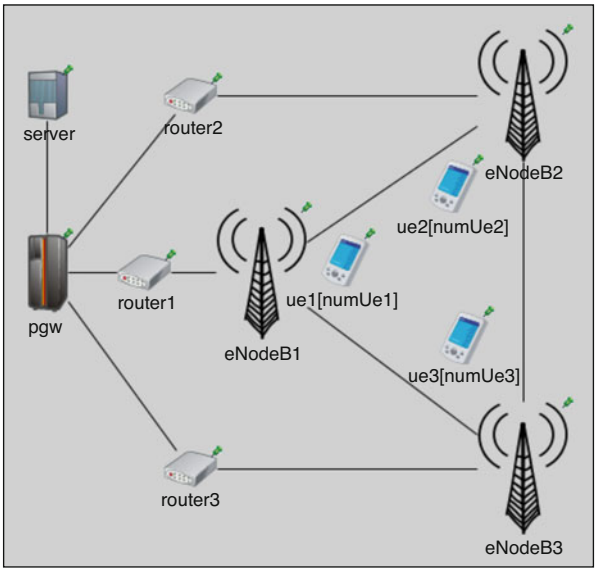

Fig. 5.16 Network definition for the scenario `MultiCell_X2Mesh`

full-mesh topology is established between eNBs; hence, point-to-point connections are provided between any pair of eNBs. Moreover, each eNB serves a set of UEs, respectively, `ue1[*]`, `ue2[*]`, and `ue3[*]`. Parameters `numUe1`, `numUe2`, and `numUe3` specify the number of UEs served by each eNB.

5.4.1.2 Parameters Configuration

With reference to the snippet of the *omnetpp.ini* file in Listing 5.4, we first configure X2-related parameters for this tutorial.

Listing 5.4 Configuration of X2-related parameters

```
# one x2App per peering eNodeB
*.eNodeB*.numX2Apps = 2

# X2 Server-side ports (x2App[0]=5000, x2App[1]=5001, ...)
*.eNodeB*.x2App[*].server.localPort = 5000 + ancestorIndex(1)

################## Peering configuration ###################
# - eNodeB1%x2ppp0 <--> eNodeB2%x2ppp0
# - eNodeB1%x2ppp1 <--> eNodeB3%x2ppp0
# - eNodeB2%x2ppp1 <--> eNodeB3%x2ppp1
############################################################
*.eNodeB1.x2App[0].client.connectAddress = "eNodeB2%x2ppp0"
*.eNodeB1.x2App[1].client.connectAddress = "eNodeB3%x2ppp0"
*.eNodeB2.x2App[0].client.connectAddress = "eNodeB1%x2ppp0"
*.eNodeB2.x2App[1].client.connectAddress = "eNodeB3%x2ppp1"
*.eNodeB3.x2App[0].client.connectAddress = "eNodeB1%x2ppp1"
*.eNodeB3.x2App[1].client.connectAddress = "eNodeB2%x2ppp1"
```

As explained in Sect. 5.3.3.2, the `LteX2App` module inside the eNB is responsible for maintaining the connection with one peering eNB. In order to build a full-mesh topology, we need to provide $N - 1$ `LteX2App` modules within each eNB. To do this, we set the `*.numX2Apps` parameter to 2 in our example.

Then, we need to configure both the server- and the client-side of each `LteX2App`. For the server-side, we need to ensure that `LteX2Apps` within the same eNB are bound to different port numbers; otherwise, local forwarding would not be possible. A simple way to achieve this is to assign incremental port numbers to every `LteX2App`, as shown at line 5. On the other hand, the client-side of each `LteX2App` must be configured so that it will connect to the IP address of the peering eNB. This is accomplished by setting the `connectAddress` parameter using symbolic addresses. Since each eNB has several X2 interfaces, it is necessary to specify the full name of the desired interface using the format `<module%interface>`. Note that the interface part of the address must specify the gate index, too.

Once the X2 is ready, it can transport every kind of message between eNBs, including CoMP messages. CoMP functionalities are disabled by default; hence, we need to activate them by setting the `compEnabled` parameter to true. This is exemplified in the *ini* configuration file below.

Listing 5.5 Configuration of CoMP-related parameters

```
1  ################ CoMP configuration ##################
2  *.eNodeB*.lteNic.compEnabled = true
3
4  # Master configuration
5  *.eNodeB1.lteNic.compManager.compNodeType = "COMP_CLIENT_COORDINATOR"
6  *.eNodeB1.lteNic.compManager.clientList = "2 3"
7
8  # Slaves configuration
9  *.eNodeB*.lteNic.compManager.coordinatorId = 1
10
11 # CoMP algorithm
12 *.eNodeB*.lteNic.compManagerType = "LteCompManagerProportional"
13
14 # Scheduling policy
15 *.eNodeB*.lteNic.mac.schedulingDisciplineDl = "MAXCI_COMP"
```

Since we modeled CoMP-CS algorithms according to the master–slave paradigm, we need to specify the role of every eNB participating in the coordination. Assume that the coordinator's role is co-located with eNodeB1. Then `eNodeB1`'s CoMP manager gets the value `COMP_CLIENT_COORDINATOR` as its `compNodeType`. On the other hand, `eNodeB2` and `eNodeB3` get the `COMP_CLIENT` default value. Moreover, the coordinator needs to know the IDs of the client eNBs, whereas the latter have to be configured with the ID of the coordinator. Parameter `compManagerType` allows one to instantiate different user-defined CoMP algorithms. SimuLTE comes with a simple example that we will introduce in the next subsection. Finally, a CoMP-enabled scheduling policy has to be selected for the downlink scheduler.

5.4.1.3 Modifying the Code

Coordinated-scheduling operations are defined within the `LteCompManager` module. According to our model, at every TTI, the eNBs participating in the coordination run a local algorithm that computes the required number of RBs and sends those requests to the coordinator via the X2 interface. Then, at every coordination period, the coordinator runs the coordination algorithm based on the input received by the clients and sends the results back to the clients. `provisionalSchedule()` and `doCoordination()` functions are virtual methods. Thus, one can define its own coordination algorithm, by implementing a new module derived from the `LteCompManagerBase` class, redefining the `provisionalSchedule()` and `doCoordination()` functions.

Listing 5.6 Modifying the CoMP algorithm

```
1  void LteCompManagerBase:runClientOperations()
2  {
3      EV<<"LteCompManagerBase:runClientOperations - node "<<nodeId_<<endl;
4      provisionalSchedule();
5      X2CompRequestIE* requestIe = buildClientRequest();
6      sendClientRequest(requestIe);
7  }
8
9  void LteCompManagerBase:runCoordinatorOperations()
10 {
11     EV<<"LteCompManagerBase:runCoordinatorOperations - node "<<nodeId_<<endl;
12     doCoordination();
13     // for each client, send the appropriate reply
14     std:vector<X2NodeId>:iterator cit = clientList_.begin();
15     for (; cit != clientList_.end(); ++cit)
16     {
17         X2NodeId clientId = *cit;
18         X2CompReplyIE* replyIe = buildCoordinatorReply(clientId);
19         sendCoordinatorReply(clientId, replyIe);
20     }
21
22     if (nodeType_ == COMP_CLIENT_COORDINATOR)   // local reply
23     {
24         X2CompReplyIE* replyIe = buildCoordinatorReply(nodeId_);
25         sendCoordinatorReply(nodeId_, replyIe);
26     }
27 }
```

SimuLTE provides the exemplary `LteCompManagerProportional` CoMP module, which inherits functionalities from `LteCompManagerBase` and overrides the `doCoordination()` function. According to this policy, the master eNB partitions the number of available RBs among eNBs in a proportional fashion based on their requested RBs. Slave eNBs can then use only the assigned subset of the available RBs when doing their scheduling. Listing 5.7 shows that the set of usable bands in the next TTIs is received within the master's reply. This information is then used to pilot the `BandLimit` data structure mentioned in Sect. 5.3.3.1.

Listing 5.7 Handling of CoMP master's reply

```
void LteCompManagerProportional:handleCoordinatorReply(X2CompMsg* compMsg)
{
    while (compMsg->hasIe())
    {
        X2InformationElement* ie = compMsg->popIe();

        if (ie->getType() != COMP_REPLY_IE)
            throw cRuntimeError("LteCompManagerProportional:
                  handleCoordinatorReply - Expected COMP_REPLY_IE");

        // parse reply message
        X2CompProportionalReplyIE* replyIe =
            check_and_cast<X2CompProportionalReplyIE*>(ie);
        std:vector<CompRbStatus> allowedBlocksMap =
            replyIe->getAllowedBlocksMap();

        UsableBands usableBands = parseAllowedBlocksMap(allowedBlocksMap);
        setUsableBands(usableBands);

        delete replyIe;
    }
}
```

5.4.1.4 Results

We now discuss the results obtained by simulating the network from Fig. 5.16. We consider 500 m as the inter-eNB distance and randomly deploy a varying number of UEs per eNB. Each UE is the destination of a CBR data flow; hence, it runs one `CbrReceiver` application on top of UDP. Flows originate at the server that has one `CbrSender` application per UE, each of them sending a 40 B packet every 20 ms. `CbrSender` and `CbrReceiver` applications are defined in the *apps* folder. The available number of RBs is set to 25, corresponding to a 5 MHz bandwidth system and MaxC/I is employed as the scheduling policy. We run five independent repetitions for each scenario configuration.

We compare the results obtained with and without interference coordination provided by the CoMP algorithm described in Sect. 5.4.1.3, to show that the latter improves the system fairness in terms of UE throughput. To this aim, we obtain the application-layer throughput by extracting the `cbrReceivedThroughput` statistics from the simulations results and process it to produce the *Lorenz curve* depicted in Fig. 5.17, in a scenario with 30 UEs per eNB. This curve provides a graphical representation of the cumulative portion of the throughput (on the y-axis) achieved by the cumulative portion of the UEs (on the x-axis). The bisector represents the ideal case, where all the UEs obtain the same throughput; hence, the more a curve is close to the bisector, the more the system is fair. Figure 5.17 shows that the scenario where CoMP is enabled guarantees more fairness among UEs than the scenario with no interference coordination.

This is due to the improvements of the channel quality for UEs close to the cell edge, which are more protected from the interference produced by non-serving eNBs. This argumentation is supported by MAC-level metrics provided by

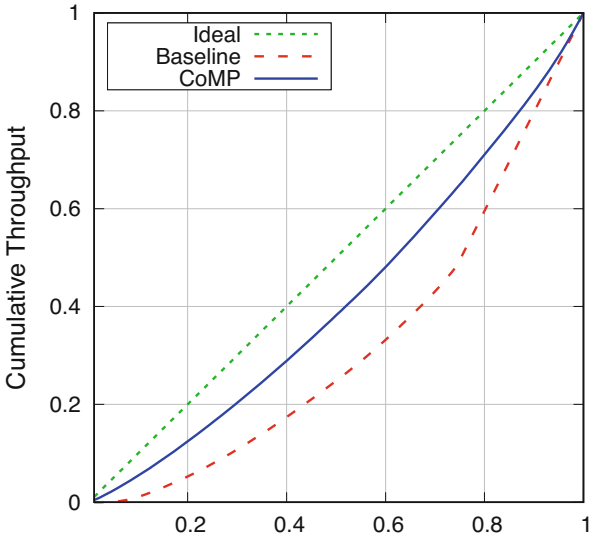

Fig. 5.17 Lorenz curve, 30 UEs per eNB

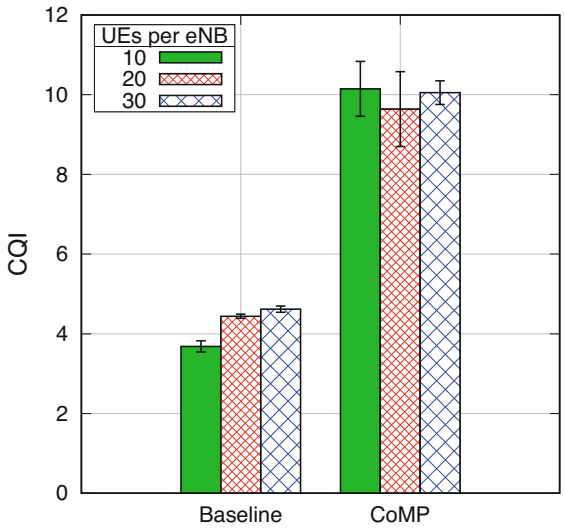

Fig. 5.18 Average CQI with increasing number of UEs

SimuLTE, such as the `averageCqiDl` one, which is shown in Fig. 5.18. The latter reports the average CQI used by the eNBs for transmitting in the DL subframe, with an increasing number of UEs per eNB. Besides improving throughput, better channel quality also allows the eNBs to reduce resource utilization. Figure 5.19 reports the `avgServedBlocksDl` statistic, i.e., the average number of RBs

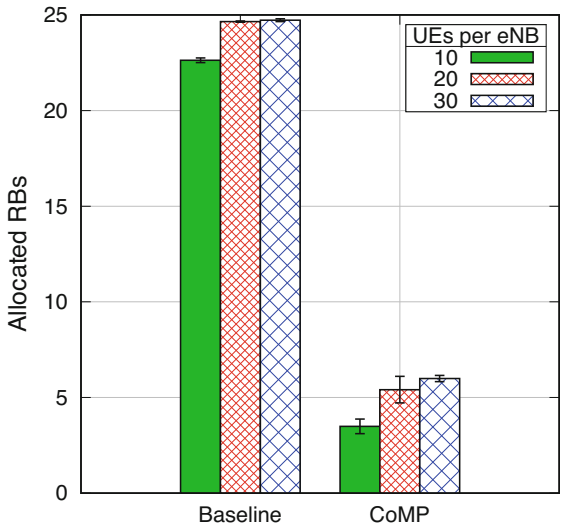

Fig. 5.19 Average number of allocated RBs with increasing number of UEs

occupied by one eNB on each TTI. While the subframe is basically saturated in the baseline case, only a small number of RBs is used when CoMP is enabled.

5.4.2 Tutorial 2: D2D Communication

This tutorial describes the configuration of a scenario where two UEs are capable of D2D communications and their serving eNB can switch their actual communication mode from direct to infrastructure mode and vice versa. In this scenario, the eNB periodically runs a decision algorithm that selects the communication mode that ensures the best channel quality for the D2D-capable flows. We show which parameters can be tuned in such a scenario and how to implement a new mode-selection policy.

5.4.2.1 Network Definition

We consider the `SingleCell_D2D` network, which is composed of one eNB. As Fig. 5.20 shows, UEs are divided into three groups, namely, `ueCell`, `ueD2DTx`, and `ueD2DRx`, which, respectively, represent conventional cellular UEs, transmitting D2D UEs, and receiving D2D UEs. The number of UEs can be configured using the corresponding NED parameters `numUeCell`, `numUeD2DTx`, and `numUeD2DRx`.

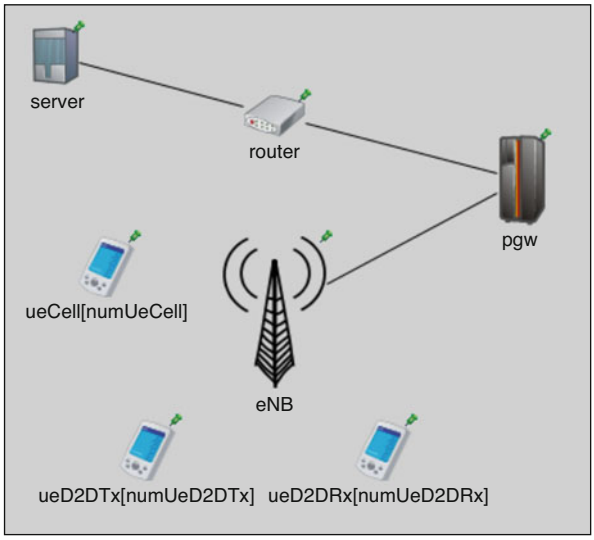

Fig. 5.20 Network definition for the D2D communication scenario

5.4.2.2 Parameters Configuration

With reference to the *omnetpp.ini* file portion shown in Listing 5.8, we first need to enable D2D capabilities on both the eNB and the UEs by setting the D2D-capable version of the LTE NICs, using the parameter `nicType`. Then, we specify the `d2dInitialMode` parameter for `ueD2DTx[0]`, i.e., the communication mode used at the beginning of the simulation. In this example this parameter is set; hence, `ueD2DTx[0]` performs transmissions using the D2D path. Regarding the MAC layer, we need to specify the D2D mode for the AMC module, which extends the default one supporting UL/DL communications only. We also allow D2D flows to transmit using either fixed modulation or the CQI reported periodically. In the former case, we need to set the `usePreconfiguredTxParams` parameter and specify `d2dCqi` in a range between 1 and 15. In the latter case, we set `enableD2DCqiReporting`, whereas `usePreconfiguredTxParams` is disabled and `d2dCqi` is ignored. For the purposes of this tutorial, the second mode is used. Going down to the PHY layer, it is possible to select different transmission power for UL and D2D communications through `ueTxPower` and `d2dTxPower` parameters, expressed in dBm. By default, mode-selection functionality is disabled at the eNB. We can enable it by specifying the name of the module implementing the mode-selection algorithm with `d2dModeSelectionType` parameter. Since mode selection is performed periodically by the eNB, it is possible to specify the period duration through the `modeSelectionPeriod` parameter, expressed in seconds. In the next subsection, we will show how to implement customized algorithms.

Listing 5.8 Configuration of D2D-related parameters

```
# Enable D2D for the eNB and the UEs involved in direct communications
*.eNB*.nicType = "LteNicEnbD2D"
*.ueD2D*[*].nicType = "LteNicUeD2D"

# Set the initial communication mode
*.ueD2DTx[0].lteNic.d2dInitialMode = true

# Select CQI reporting mode for D2D transmissions
*.eNB.lteNic.mac.amcMode = "D2D"
*.eNB.lteNic.phy.enableD2DCqiReporting = true
**.usePreconfiguredTxParams = false
**.d2dCqi = 7

# Select Tx Power
*.ueD2DTx[0].lteNic.phy.ueTxPower = 26 # in dBm
*.ueD2DTx[0].lteNic.phy.d2dTxPower = 20 # in dBm

# Enable Mode-selection algorithm
*.eNB.lteNic.d2dModeSelectionType = "D2DModeSelectionBestCqi"
*.eNB.lteNic.d2dModeSelection.modeSelectionPeriod = 1s
```

5.4.2.3 Modifying the Code

The module responsible for selecting the transmission mode of D2D-capable flows is `D2DModeSelectionBase`. The latter provides basic functionalities for periodic mode-selection operations and it can be extended to realize the preferred policy. As shown in Listing 5.9, this module periodically schedules a self-message, which serves as a trigger for calling the `doModeSelection()` function. The latter implements the actual mode-selection algorithm. Since it is a virtual function, one can build its own module extending the base one and redefining the behavior of `doModeSelection()`. SimuLTE provides an example module, called `D2DModeSelectionBestCqi`, which selects either UL or D2D for a flow based on the best CQI value reported for the two links. After the execution of `doModeSelection()`, the decisions are notified to the UEs involved in D2D-capable communications.

Listing 5.9 Modifying the mode-selection algorithm

```
void D2DModeSelectionBase:handleMessage(cMessage *msg)
{
    if (msg->isSelfMessage())
    {
        if (strcmp(msg->getName(),"modeSelectionTick") == 0)
        {
            // run mode selection algorithm
            doModeSelection();
            // send switch notifications to selected flows
            sendModeSwitchNotifications();
            scheduleAt(NOW+modeSelectionPeriod_, msg);
        }
        else
            throw cRuntimeError("D2DModeSelectionBase:handleMessage -
                  Unrecognized self message %s", msg->getName());
    }
```

```
    else
        delete msg;
}
```

The message including a switch notification traverses the whole LTE stack at the UE side in the upstream direction. This way, each layer is able to perform switching-related tasks. Listing 5.10 refers to a snippet of the `handleMessage()` function in `LteMacUeD2D` module. If the message is recognized as a switch notification, the corresponding handler `macHandleD2DModeSwitch()` is invoked. The MAC-layer handler is responsible for clearing buffers and terminating ongoing H-ARQ processes. However, one can re-implement this handler (as well as higher-layer handlers) to provide advanced switching operations, e.g., to avoid packet loss.

Listing 5.10 Modifying mode-switching handler at the UEs

```
if (incoming == down_[IN])
{
    UserControlInfo *userInfo = check_and_cast<UserControlInfo *>(pkt->
          getControlInfo());
    if (userInfo->getFrameType() == D2DMODESWITCHPKT)
    {
        EV<<"LteMacUeD2D:handleMessage - Received packet "<<
        pkt->getName()<<" from port "<<pkt->getArrivalGate()->getName()<<endl;

        // message from PHY_to_MAC gate (from lower layer)
        emit(receivedPacketFromLowerLayer, pkt);

        // call handler
        macHandleD2DModeSwitch(pkt);
        return;
    }
}
```

5.4.2.4 Results

In order to demonstrate the effects of mode switching, we consider the network from Fig. 5.20 and simulate one pair of D2D UEs. One UE is the transmitter and one is the receiver of a CBR data flow, sending one packet every 20 ms. We make the packet length vary from 500 B to 1000 B to assess the performance of the D2D flow with different traffic loads. The two UEs are 300 m away from the eNB and they swing back and forth in a straight line at a speed of 3 m/s. Such a path allows the direct link between them to experience different channel quality during the simulation, whereas the channel quality for UL stays constant. This way, we can simulate a scenario where the eNB implements the aforementioned `D2DModeSelectionBestCqi()` policy, making the flow bounce between the direct link and the infrastructure one. We compare this scheme with the scenario where mode selection is disabled and the flow uses always the direct link.

In this dynamic scenario, OMNeT++ vectors are useful to evaluate the behavior of the system during the simulation. The left side of Fig. 5.21 reports the CQI used by the transmitting UE, which experiences large variations due to the change in

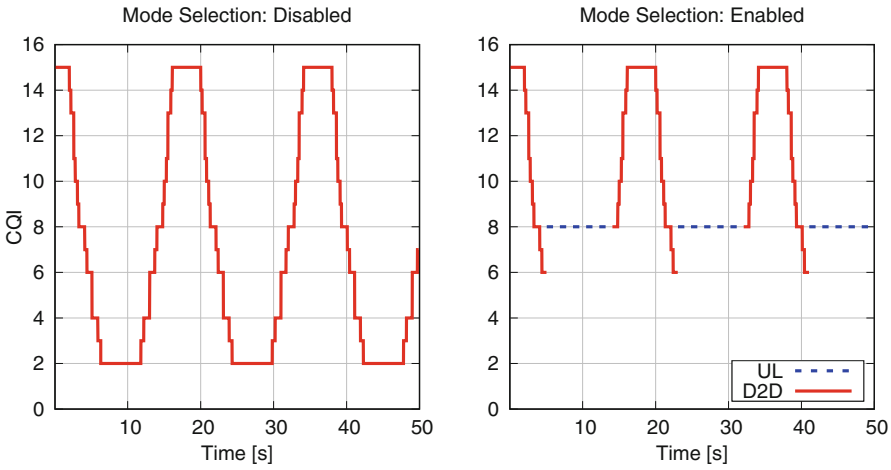

Fig. 5.21 CQI with mode selection disabled (left) and enabled (right)

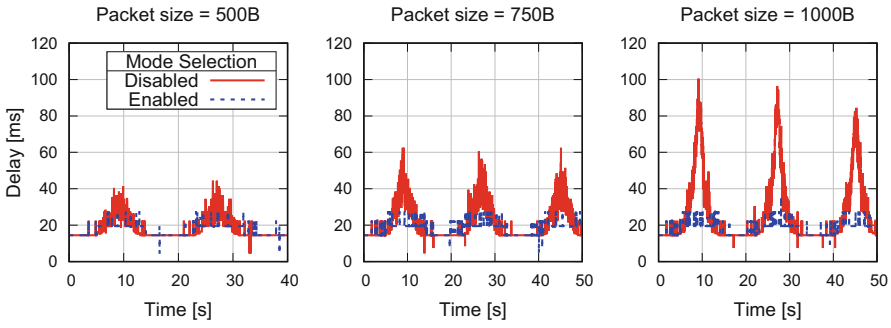

Fig. 5.22 Latency of CBR packets with increasing packet length

the distance between the UEs. The right part of Fig. 5.21, instead, shows the CQI used when mode selection is employed: the chart is obtained by putting together `averageCqiD2D:vector` and `averageCqiUl:vector` statistics, since the flow periodically switches from the direct link to the infrastructure one and vice versa. As a result, we observe that enabling mode selection allows the flow to use better CQI, hence use better modulation.

This affects the latency of the flow, as shown in Fig. 5.22. The latter reports the `cbrFrameDelay:vector` statistic with different packet lengths. When no mode selection is active, the latency of the flow drastically increases at the points where the CQI is smaller. This behavior is more pronounced when the traffic load increases due to larger packets. On the other hand, the configuration with mode selection allows the flow to keep the latency small.

References

1. Baldo, N., Miozzo, M., Requena-Esteso, M., Nin-Guerrero, J.: An open source product-oriented LTE network simulator based on ns-3. In: Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '11, pp. 293–298. ACM, New York (2011). https://doi.org/10.1145/2068897.2068948
2. Bouras, C., Diles, G., Kokkinos, V., Kontodimas, K., Papazois, A.: A simulation framework for evaluating interference mitigation techniques in heterogeneous cellular environments. Wirel. Pers. Commun. **77**(2), 1213–1237 (2014). https://doi.org/10.1007/s11277-013-1562-5
3. Jakes, W. (ed.): Microwave Mobile Communications. Wiley, New York (1975)
4. Mehlführer, C., Wrulich, M., Ikuno, J.C., Bosanska, D., Rupp, M.: Simulating the long term evolution physical layer. In: 2009 17th European Signal Processing Conference, pp. 1471–1478 (2009)
5. Sanchez-Iborra, R., Sanchez-Gomez, J., Ballesta-Viñas, J., Cano, M.D., Skarmeta, A.F.: Performance evaluation of LoRa considering scenario conditions. Sensors **18**(3), 772 (2018). https://doi.org/10.3390/s18030772
6. Virdis, A., Iardella, N., Stea, G., Sabella, D.: Performance analysis of openairinterface system emulation. In: 2015 3rd International Conference on Future Internet of Things and Cloud, pp. 662–669 (2015). https://doi.org/10.1109/FiCloud.2015.77