technische universität dortmund

Communication Networks Institute
Prof. Dr.-Ing. Christian Wietfeld

Additional information about the implementation of LENA-NB in the simulation framework ns-3

Note:
This information can be used for a better understanding of LENA-NB features.

For a full documentation of ns-3 and LENA, visit: https://www.nsnam.org/documentation/

Authors:
Tim Gebauer (tim.gebauer@tu-dortmund.de)
Pascal Jörke (pascal.joerke@tu-dortmund.de)

For citation, please refer to these publications:

https://doi.org/10.1145/3532577.3532600
https://doi.org/10.1109/VTC2022-Spring54318.2022.9860611

1 Implementation of an NB-IoT Simulation Framework

In the following chapter first Network Simulator 3 (ns-3) [1] and its Long Term Evolution (LTE) framework are briefly introduced. Afterwards a detailed analysis of the related work is performed. This work include both the attemps of implementing Narrowband Internet of Things (NB-IoT) in LTE Evolved Packet Core (EPC) Network Simulator (LENA) and research done based on NB-IoTs features. Then the performed modifications for NB-IoT are presented.

1.1 Introducing ns-3 and LENA

The ns-3 is a discrete event-driven simulation platform for communication networks written in C++. It targets research and educational use and is well known for its good reputation among researchers.

The creation of an ns-3 simulation is split up into multiple steps. Simulations in ns-3 are based on the concept of nodes that are linked to each other. On these nodes, different models can be applied. A model can thereby be a protocol layer like, e.g., User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) stacked on one another or a complete technology like Wi-Fi. To ease nodes' initialization and link different protocol layers and nodes, ns-3 introduces so-called helper classes, which automate these procedures. Further, containers are used to group several network nodes based on predefined aspects.

The behavior of different models is controlled by attributes that are exposed and can be adjusted. The simulated time is set before the simulation is started, and the simulator stops at the given end time. ns-3 simulations are usually predefined, and no external code is executed while the simulation is running. All models act on the parameters set at the beginning and are not directly controlled at runtime. Further delayed methods can be scheduled from within the model, e.g., a transmission delay between sending and receiving.

1.1.1 LTE EPC Network Simulator (LENA)

ns-3 already includes a mature LTE model called LENA. It implements the LTE radio protocol stack (Radio Resource Control (RRC), Packet Data Convergence Protocol (PDCP), Radio Link

Control (RLC), Medium Access Control (MAC), Physical (PHY)) on User Equipment (UE) and evolved Node B (eNB), an EPC model, and a realistic channel modeling. LENA implements every stack member as an individual entity (C++ class) and uses so-called Service Access Point (SAP)s to connect them figure 1.1. These SAPs divide into two groups, users and providers. User SAPs always implement the interactions performed by the lower stack member to a higher one. A possible example would be the MAC layer propagating an RLC-Protocol Data Unit (PDU) to the RLC layer. On the other hand, a provider SAP implements interactions for a higher layer to perform on a lower one, f.e. the RRC layer requesting the MAC layer to initiate a Random Access (RA) procedure. As already shown in the previous example, SAPs are not placed strictly between two layers but can instead extend over multiple layers, skipping the layer in between.

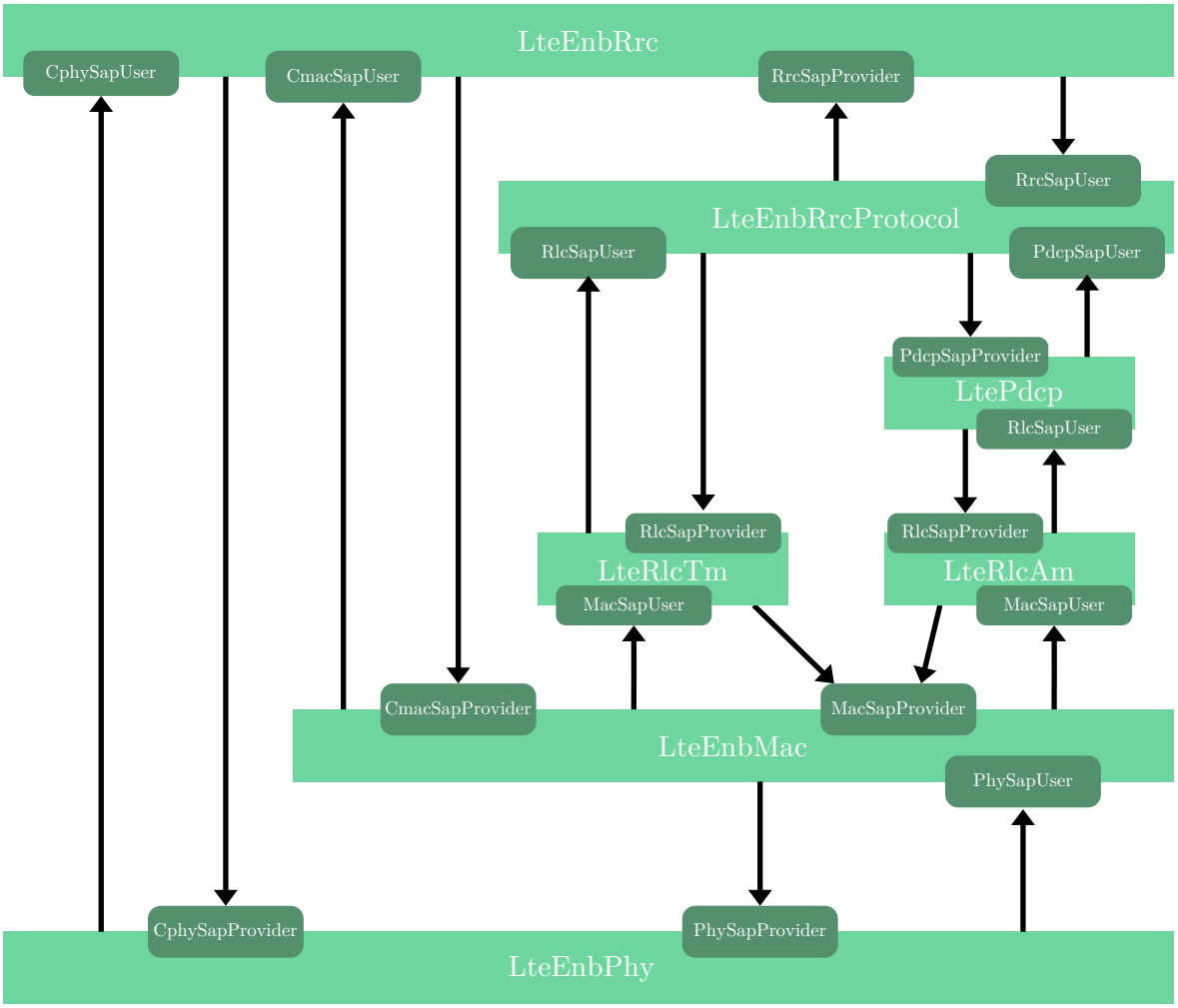

Figure 1.1: LENA Protocol Stack

1.2 Turning LENA into LENA-NB

In the following section the needed modifications of LENA for NB-IoT are documented. Additionally the new name LTE EPC Network Simulator Narrowband (LENA-NB) is proposed, indicating the major changes.

1.2.1 System Information Exchange

As a first step towards LENA-NB the control information like *Master Information Block Narrowband (MIB-NB)* and the *System Information Block (SIB)*s had to be implemented.

Legacy LENA differentiates between control messages and "actual" data. Control messages are implemented as ideal messages carrying data as member variables and are passed through the radio layers, bypassing standard packet processing. These control messages already include Downlink Control Information (DCI)s for downlink and uplink, the Buffer Status Report (BSR), Channel Quality Indicator (CQI) messages, the Random Access Channel (RACH) preamble, the Random Access Response (RAR), and the Master Information Block (MIB) and System Information Block 1 (SIB1). Thereby first for LENA-NB, control messages for the *MIB-NB*, *System Information Block 1 - Narrowband (SIB1-NB)*, and *System Information Block 2 - Narrowband (SIB2-NB)* were implemented.

The *SIB2-NB* was included due to its importance for the Random Access procedure in NB-IoT, carrying all information needed. The three control messages each contain a struct with the information specified in [2]. Their implemented structure is based on the Abstract Syntax Notation One (ASN.1) definition of [2]. Most of the information included in the *MIB-NB*, *SIB1-NB*, and *SIB2-NB* are specified in the *LteEnbRrc* layer. They are then passed to the *LteEnbPhy* layer whenever they are updated.

In the case of the *MIB-NB* and *SIB1-NB*, I implemented their transmission process purely into the *LteEnbPhy* layer. The beginning of a new transmission cycle of the *MIB-NB* and *SIB1-NB* can each be derived by the layers *StartFrame* method called at the beginning of any new frame. For the beginning of a *MIB-NB* transmission $SFN\%64 == 0$ is evaluated and for the *SIB1-NB* $SFN\%256 == 0$. While the *MIB-NB* is sent in every system frame, the *SIB1-NB* is sent in the *StartSubframe* method (called from *StartFrame*) after conditions evaluate to true. Both are then added to the *LteEnbPhy* layers control message queue (which holds all control messages sent in this subframe).

The *SIB2-NB*, on the other hand, is sent using the periodically called *SendSystemInformationNB* method of the *LteEnbRrc* layer. LENA allows the use of two RRC protocol types, ideal and real. RRC-messages are directly passed to the *LteUeRrc* layer without going through the other radio protocol stacks when using the ideal setting. The real RRC protocol implements the transmission of any RRC-message using the radio protocol stack. However, system information (like the *SIB2-NB*,

etc.) is still directly transmitted to the UEs RRC, but a typical RRC delay was added. This is not a problem because the theoretically used resources are still blocked in the later implemented scheduler.

On the UE side, the *MIB-NB* and *SIB1-NB* are received by the *LteUePhy* layer as generic control messages from the spectrum. The *LteUePhy* then determines the control messages type (*MIB-NB*, *SIB1-NB*, DCI, etc.) and then forwards it to the corresponding layer. In the case of *MIB-NB* and *SIB1-NB*, they are forwarded to the *LteUeRrc* layer, which evaluates them and applies the contained configurations (Uplink frequency, etc.) to the underlying protocol layers. The overall implementation is visualized in figure 1.2.

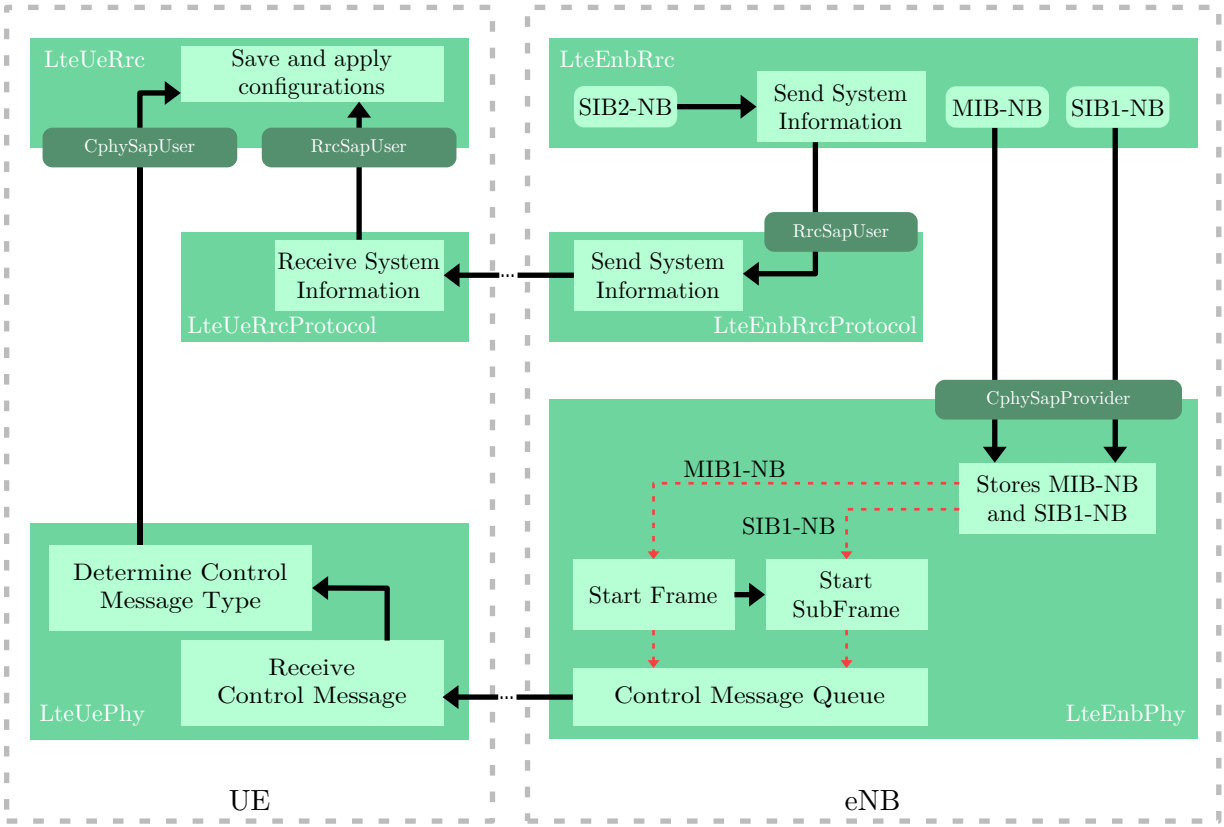

Figure 1.2: Transmission of System Information in LENA-NB

1.2.2 Random Access Procedure

Next I implemented the RA procedure of NB-IoT by modifying and further extending LENAs implemented RA procedure as presented in figure 1.3. After the *LteUeRrc* layer received its *SIB2-NB*, it applies the included Narrowband Physical Random Access Channel (NPRACH) config and initiates a RA procedure. Therefore it calls the *StartRandomAccessProcedureNB* method of the *LteUeMac*, which first evaluates if all necessary information is available. If this is the case, the *LteUeMac* first reset its previous counters regarding sent preambles, etc.

Next, it gets the current Reference Signal Receive Power (RSRP) from the *LteUePhy* layer, which is calculated based on the received *NPSS*. Following it determines the Coverage Enhancement (CE)-Level, to which it belongs. The eNB defines 3 CE levels using a transmitted threshold list of the *SIB2-NB*. Depending on the CE group, the MAC layer derives its random access parameters from the *SIB2-NB*s *RadioResourceConfig.nprachConfig* and starts the Random Selection of a NPRACH preamble.

At this point, a short insertion about the config structures of the system information. LENA bases its system information structures on the ASN.1 definition of [2]. Accordingly, ASN.1 Enumerated objects are implemented as C++-Enums. That way, users can only directly choose from valid options but have to be parsed every time they have to be evaluated mathematically.

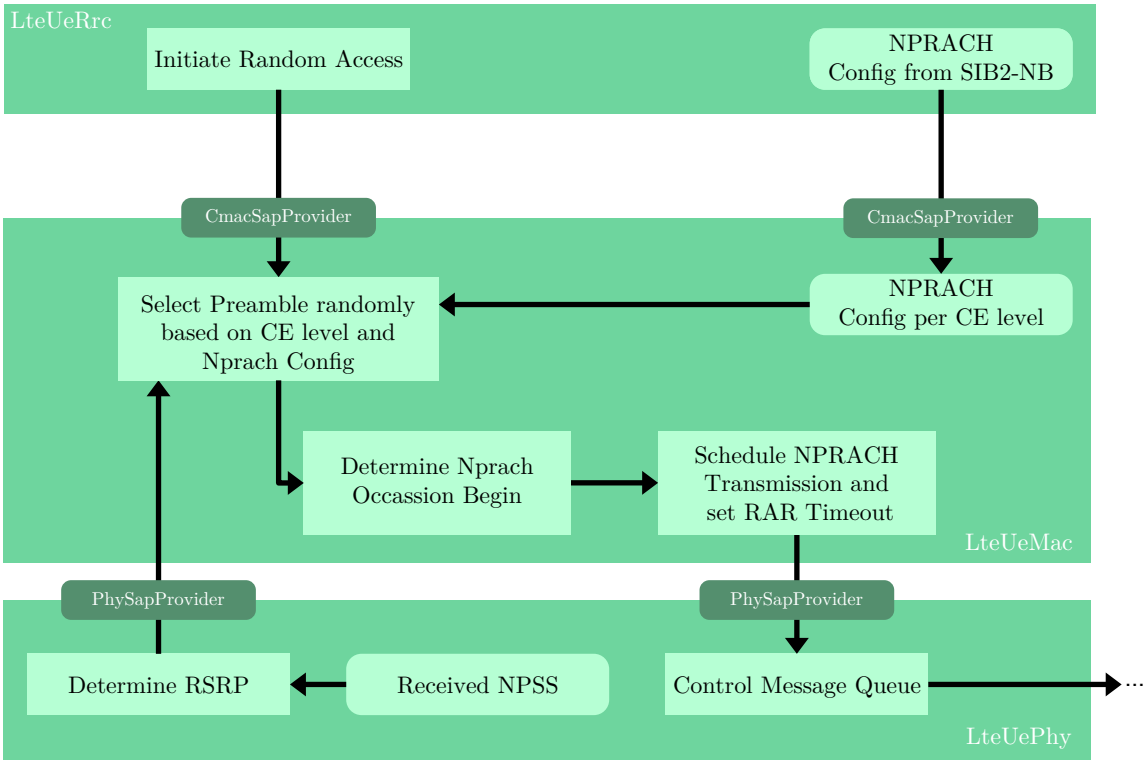

Figure 1.3: Implementation of the NPRACH in LENA-NB

The RA preamble id is selected randomly between 0 and the number of subcarriers of its set using a uniform distribution. Next, the beginning of the NPRACH occasion has to be determined:

$$Occassion = \left(SFN - \left(SFN \mod \left(\frac{Nprach_{Periodicity}}{10}\right)\right)\right) \times 10 + NPRACH_{offset} \quad (1.1)$$

In the unlikely case that the current subframe is equal to the beginning of the occasion, the NPRACH transmission is triggered right away. Otherwise, the offset to the next occasion is calcu-

lated with

$$SF_{wait} = \begin{cases} Ocassion - SF, & SF < Ocassion \\ (Nprach_{period} - (SF \mod Nprach_{Period})) + Nprach_{offset}, & SF > Ocassion \end{cases} \quad (1.2)$$

and the NPRACH transmission scheduled acorrdingly. When sending the NPRACH preamble the *LteUeMac* first determines the Random Access-Radio Network Temporary Identifier (RA-RNTI). Further, it calculates the duration of the complete NPRACH and schedules the transmission of the NPRACH to the end subframe accordingly. This simplification has no impact on the following random access procedure but reduces computation time drastically, which will be a reoccurring factor. Last, the *LteUeMac* determines the RAR timeout and schedules the corresponding method accordingly. The actual transmission of the preamble is handled by the *LteUePhy* layer, which queues an NPRACH containing the RA-RNTI, the Random Access Preamble Identifier (RAP-ID), and the subcarrier offset in its control message queue. The NPRACH control message is once again ideal and carries additional information for simplification reasons.

Next, the receiving process of the *LteEnbPhy* and *LteEnbMac* layer is presented in figure 1.4. The *LteEnbPhy* receives all control messages for a subframe bundled as a list of generic control messages and first determines the control message type. In the case of an NPRACH control message, it notifies the *LteEnbMac* layer about the reception and forwards the RAP-ID the subcarrier on which the NPRACH message was theoretically received and the RA-RNTI of the RA. The *LteEnbMac* layer then saves the RA-RNTI and the number of received preambles based on the used subcarrier. Further, the *LteEnbMac* checks every subframe if it is the end of an NPRACH occasion and, in this case, evaluates all NPRACH preambles received on this occasion. The preambles of all UEs that chose to use this occasion are received simultaneously, as propagation delays are not taken into account.

Usually, NB-IoT supports contention resolution for its random access. Still, for this implementation, it is assumed that if two UEs in the same CE-level choose the same occasion and RAP-ID, they interfere destructively, and the eNB cannot identify any preamble. Accordingly, at the end of the occasion, the *LteEnbMac* checks the list of preambles for duplicates and, in case, removes the corresponding entries because it theoretically never received the NPRACH. Next, for all preambles it received successfully, it begins by allocating temporary Cell-Radio Network Temporary Identifier (C-RNTI)s from the *LteEnbRrc*. At this point, the *LteEnbRrc* initializes an instance of its *UeManger*, which corresponds to the individual UE.

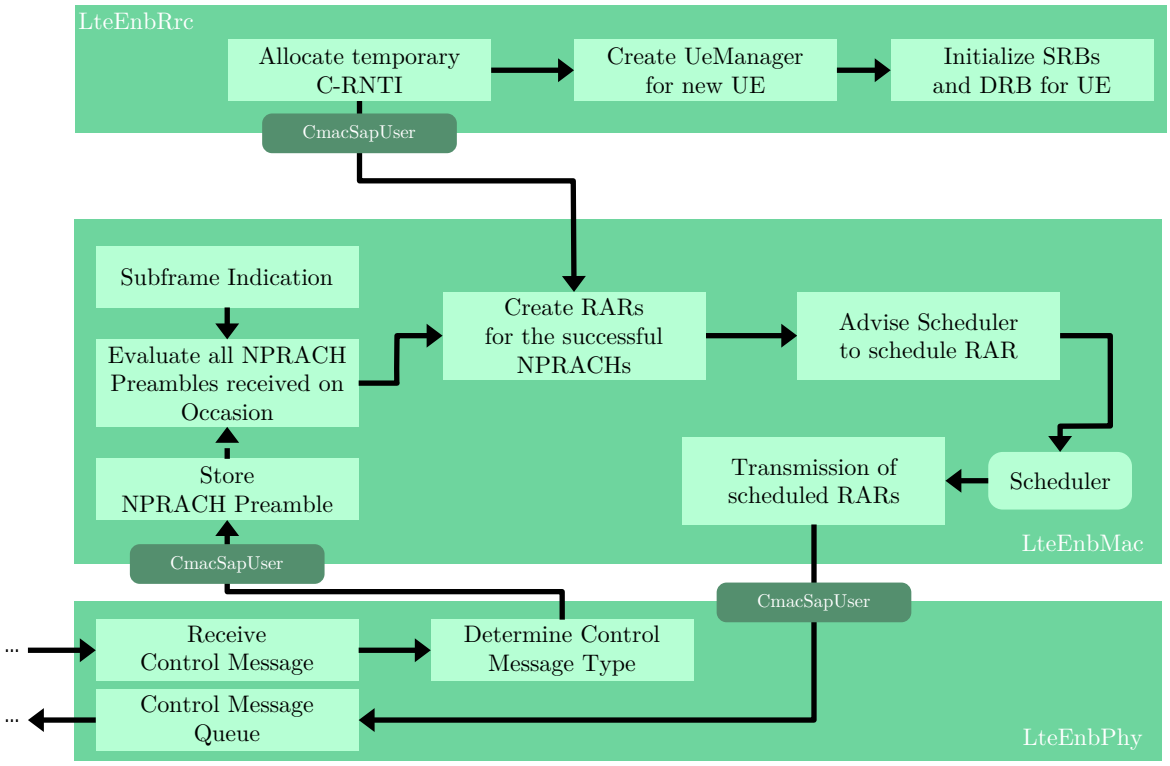

Figure 1.4: Implementation of the RAR processing in LENA-NB

This *UeManager* contains a state machine of the UE, initializes the Signalling Radio Bearer (SRB) and later Data Radio Bearer (DRB)s for the UE, and further handles every RRC level communication with this specific UE. When the *UeManager* was initialized successfully, the *LteEnbRrc* layer returns the temporary C-RNTI to the *LteEnbMac*, which then starts to assemble a MAC RAR Service Data Unit (SDU) for the corresponding UE. This SDU, implemented as a struct, consists of the temporary C-RNTI, the RAP-ID, and an uplink grant. Due to simplifications, further information is not included but can be easily added later. Multiple MAC RAR SDU can fit into a single MAC PDU due to their small size of just 48 bit [3]. Thus the *LteEnbMac* then uses the maximum size of a MAC PDU and determines the number of RAR SDU fitting into it. It also considers the size of the MAC PDU header and each RAR SDU, which all have the same size of 8 bit [3]. If there are more RAR SDU that can fit into a single PDU, another PDU is queued. Last, all PDUs are passed to the scheduler, which schedules their transmission based on the corresponding coverage enhancement level. Thus, for now, it is assumed that the DCIs and MAC PDUs were scheduled successfully, transmitted, and successfully received by each UE.

The last part of the RA procedure takes place in the UE. First, the *LteUePhy* identifies the RAR message. The UE has not received an identifier yet. Thus it can only be identified using the RA-RNTI and the RAP-ID. Accordingly, the UE first checks if the RAR includes its RA-RNTI. This RA-RNTI would have been scrambled into the message in an actual transmission. When the

RARs RA-RNTI corresponds to the UEs RA-RNTI, the UE starts to iterate over each RAR and checks for its RAP-ID. In case of successful reception, the *LteEnbPhy* forwards the RARs payload to the *LteEnbMac*, which then applies the temporary C-RNTI, notifies the *LteUeRrc* about the successful RA, and further triggers the RLCs TX opportunity based on the included uplink grant. Should the UE not receive a RAR in the given RA timeout, it triggers a new RA attempt until the maximum number of tries is reached. This maximum is again transmitted within the *SIB2-NB*, as well as the RA timeout.

1.2.3 Implementation of an NB-IoT Compliant Resource Scheduler

At the beginning of this work, we assumed that the existing LENA scheduler could have been slightly adjusted to suit the research approach. It implements the specification of an LTE scheduler Application Programming Interface (API) standardized by the FemtoForum (today Small Cell Forum) [4], which already includes multiple scheduling algorithms like Round-Robin, Proportional-fair, Maximum-throughput, etc.

Complications with the existing scheduler

On further inspection, various problems with the use of this scheduler were revealed, the most important being that the scheduler schedules only over a subframe. First, it checks the available Physical Resource Block (PRB)s in the current subframe and then assigns them based on its scheduling algorithm to the queued users. After that, it triggers the transmission of the corresponding DCIs, and the UE sends or receives the data instantaneously after receiving their DCI. Further, the overall scheduler complexity is kept down because it does not have to keep track of future transmissions.

While this is a proper approach for LTE scheduling, it does not suit NB-IoT. The first problem being that NB-IoT specifies guard times between the reception of a DCI and the following transmission, to allow lower complexity UEs to switch transmission modes or decode the information correctly. Further, NB-IoT introduces repetitions, which prolong the scheduled transmissions even further. According to this ”cross-subframe” scheduling, the scheduler must keep track of the whole uplink and downlink spectrum over multiple subframes. The highest number of repetitions specified in NB-IoT is 2048, resulting in more than 2 of transmission time. Further, the scheduler must be able to schedule the different kinds of Ressource Unit (RU) sizes and subcarrier spacings. Another important aspect is that an NB-IoT scheduler also has to schedule the UEs based on the predefined search spaces.

After further in-depth analysis, we decided that the adaption of the Femto forum scheduler to the proposed features would be beyond the scope of this thesis and require more effort than implementing a specific NB-IoT scheduler. Another strong argument for implementing a particular scheduler

is that one of the significant benefits of the Femto Forum scheduler (the multiple scheduling algorithms) might not apply to NB-IoT. An example of this would be the proportional-fair algorithm. It assigns resources based on the relative channel quality of a UE. For LTE, this is a logical approach due to the UEs moving dynamically and fast and large-scale fading changes over time. In NB-IoT, most use cases include stationary applications, in which the relative channel quality might not change in a scale, allowing proper scheduling. The implemented NB-IoT scheduler changed throughout this thesis multiple times. Accordingly, only the last and final state is presented in the following.

While implementing the NB-IoT scheduler, its interfaces were kept close to the interfaces of the Femto forum scheduler. Therefore, a *NpdcchMessage* called struct was defined, which is used to exchange scheduling information between the scheduler and the *LteEnbMac* layer. This struct contains the complete scheduled transmission information like search space type, the Narrowband Physical Downlink Control Channel (NPDCCH) format, the DCI type, scheduled subframes, etc. Using this information, the *LteEnbMac*-Layer can coordinate and trigger each transmission. The NB-IoT Scheduler has different input parameters. First, it receives the Buffer status reports of the eNB and the UEs from the *LteEnbMac* (figure 1.5).

BSR

While the BSR from the eNB is divided in the Logical Channel (LC)s and thereby more detailed, the UEs BSR is just the number of queued bytes, as specified in [BSR]. Since NB-IoT does not have an uplink control channel, the BSR control message of LENA can not be used.

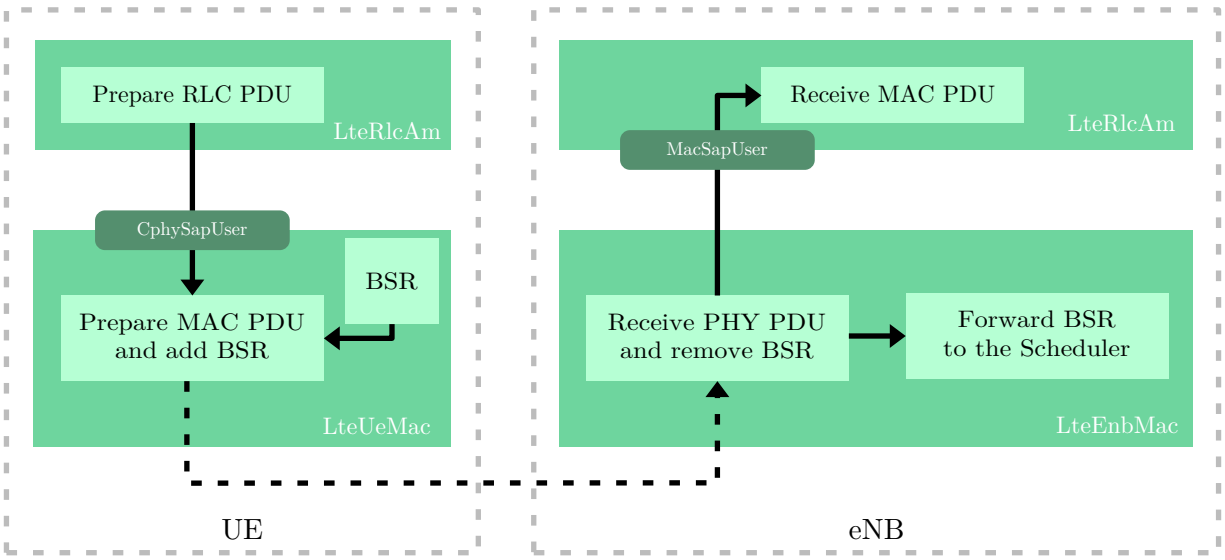

Figure 1.5: Implementation of the BSR in LENA-NB

Accordingly I implemented the typical BSR and Data Volume and Power Headroom Report (DPR)

procedure of NB-IoT for LENA-NB. In NB-IoT, the UEs BSR is transmitted as a MAC control element. Therefore two ns-3 *ByteTags* were created. A *ByteTag* can be added to any packet and later be identified by its type. The first represents the DPR MAC control element, mainly used for the BSR in the RRC connection setup/resume procedure. Further, the second *ByteTag* represents the generic BSR, which can be added to any MAC PDU. Both tags include mapping the queued bytes to the BSR index and are added to the transmitted PDU if necessary.

Scheduler Initialization

In the following the fully self implemented NB-IoT scheduler is discussed, beginning with its initialization, where it first receives the NPRACH configuration of the different CE levels and the complete *SIB2-NB*. The scheduler then stores all relevant scheduling delays, offsets, and the usable Hybrid Automatic Repeat Request (HARQ) subcarrier on initialization. Afterward it initializes a resource grid for both the uplink and downlink.

The scheduler currently works on a hyperframe-cycle basis, so the Downlink Resource-Grid includes $1024 hyperframes \times 1024 systemframes \times 10 subframes$ entries. The uplink grid supports only 15 kHz subcarrier spacing and consists of the number of downlink entries times the number of used subcarriers (12). Following the resource grid initialization, the scheduler starts blocking Narrowband Primary Synchronisation Channel (NPSS) and Narrowband Secondary Synchronisation Channel (NSSS), *MIB-NB*, *SIB1-NB*), *System Information - Narrowband (SI-NB)*, and RA occasions for the whole hyperframe-cycle. That way, the scheduler only has to consider the somewhat complex system resources scheduling once and later schedules messages based on the free resources.

Search space based scheduling

Further, the scheduler uses a search-space-based scheduling approach. Therefore the scheduler receives all potential search spaces configuration using a predefined *SearchSpaceConfig* struct, allowing the generalization of different search spaces. Also, different methods were implemented, allowing the conversion of the *SIB2-NB*s NPRACH config into the search space format of the scheduler. To keep track of each UE, the scheduler implements a UE Table mapping the UEs C-RNTI onto a struct containing all necessary information like RSRP, buffer sizes, etc. Since UEs can switch to other search spaces based on their current connection state, the scheduler further consists of a table mapping the UEs C-RNTI to the current search space. The UE config is first added to the scheduler on the scheduling request of the UEs RAR.

Central scheduling routine

The central scheduling routine of the self implemented NB-IoT scheduler (figure 1.6) depends on the *SubframeIndication* of the *LteEnbMac*. On each *SubframeIndication*, the *LteEnbMac* calls the scheduling routine with the current system frame and subframe number. Then the scheduler checks if the current subframe is the beginning of any of the present search spaces. If so, the scheduler starts the scheduling process of the corresponding search space. As the first part of the search space scheduling, RARs are prioritized, and the scheduler tries to schedule them (the specific scheduling approach will be explained further below) before any other UE in the search space. Next, the scheduler sorts all UEs in the search space based on a predefined scheduling algorithm. A simple round-robin scheduler is used in the current implementation based on its fairness. Then the scheduler starts looping through the sorted queue. When a UE has data to transmit, the scheduler tries to find potential resources as follows.

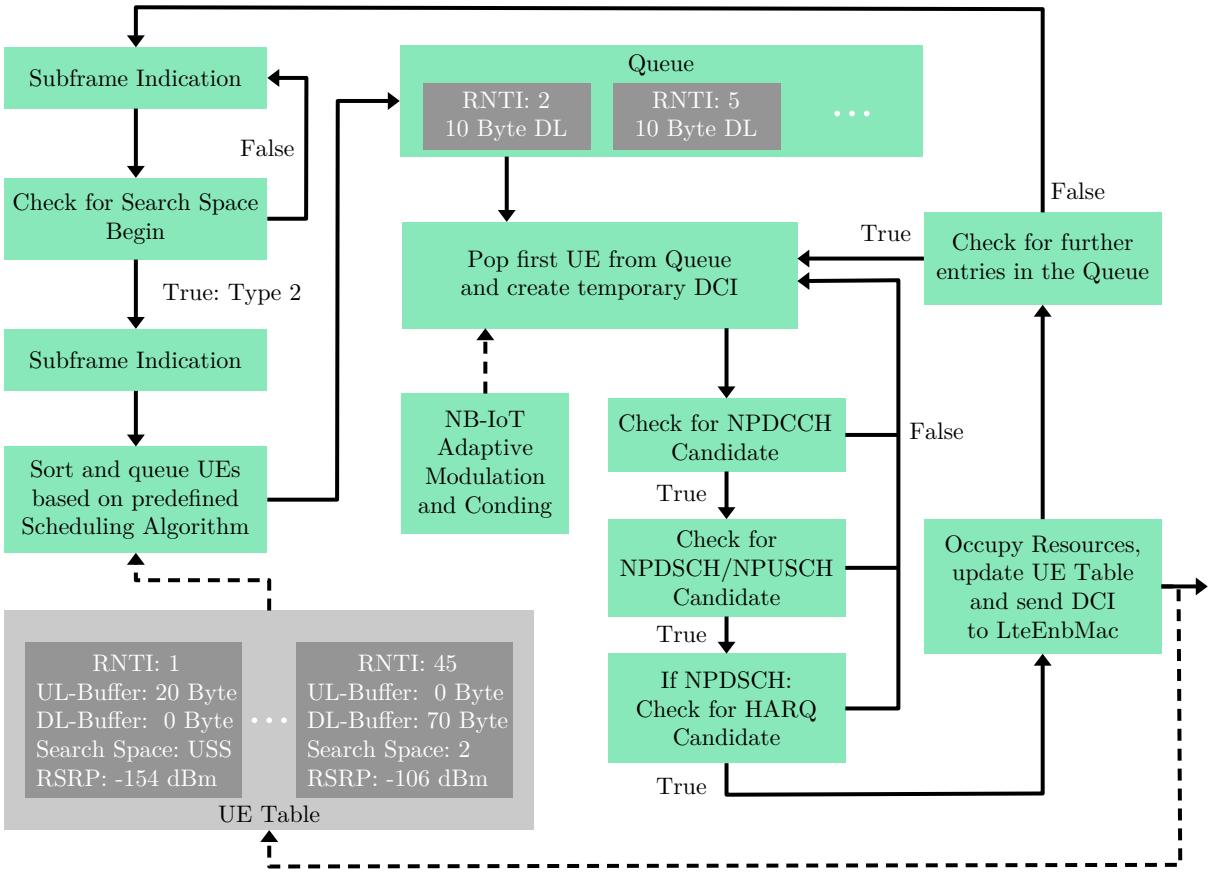

Figure 1.6: Flowchart of the implemented LENA-NB Scheduler

First, it creates a temporary DCI for the transmission, containing the theoretical number of subframes or RU, repetitions, and the Transport Block Size (TBS). In the first instance, the scheduler checks if the queued bytes do not exceed the maximum TBS for the given NB-IoT release. If

so, the scheduler limits the transmission to the maximum, and further data is queued for later scheduling. The repetitions of the DCI are based on [5], while the parameters for the Narrowband Physical Downlink Uplink Channel (NPUSCH) and Narrowband Physical Downlink Shared Channel (NPDSCH) are more detailed and based on the implemented NB-IoT Adaptive Modulation and Coding (AMC). This AMC is based on measurements created in [6] and determines the optimal configuration for the given coupling loss and needed TBS, where low a Block Error Rate (BLER) and a Transmission Time Interval (TTI) are prioritized. Using the DCI parameters, the scheduler first tries to fit the DCI itself into the given search space.

In the following, the the downlink scheduling is presented. Therefore the scheduler first determines R_{max} DL subframes that do not contain system resources. Within these subframes, it then checks for every potential beginning of a DCI candidate if it is followed by a number of not occupied subframes equal to the number of DCI repetitions. When multiple viable candidates are found, the scheduler selects the first of them. Assuming the scheduler found a viable DCI candidate, it tries a similar procedure for the NPDSCH resources, by first calculating the needed subframes ($N_{subframes} \times N_{repetitions}$). Then it loops through all possible offsets between the DCI and the NPDSCH transmission and checks if the number of following subframes necessary for the transmission (excluding system resources) are not occupied. For the DL, it is also important to note that the offset between the number of repetitions can differ. When both a viable DL DCI candidate and an NPDSCH candidate are found, the scheduler continues by checking for a potential HARQ candidate. This process is similar to the NPDSCH because the scheduler loops through all possible offsets and the available subcarriers. If the scheduler found a viable DL DCI, an NPDSCH candidate, and a HARQ opportunity, it occupies the resources, adjusts its UE table, and forwards the information to the *LteEnbMac* layer.

For the uplink, the scheduling is implemented similarly. The only differences are the different offsets between DCI N0 and the NPUSCH transmission, the number of subcarriers and the missing HARQ transmission. Further for now the scheduler only supports single tone transmission. The scheduling of RARs represents a special case in the NB-IoT scheduling. Then the scheduler schedules the resources for the downlink, it also has to schedule another uplink transmission for the uplink grant of Message 3 (MSG3). After finishing the scheduling process, the scheduler returns all scheduling information to the *LteEnbMac*.

1.2.4 Connection Resume Procedure

As the connection resume procedure is a main component of NB-IoT and LENA does not support it, next the resume procedure was implemented close the the already implemented setup procedure. The UE needs a resume ID transmitted in the *RrcConnectionRelease-NB* message to initiate the connection resume procedure, but LENA has not entirely implemented the connection release. Therefore the first step was to implement the release procedure.

As a start, a *DataInactivity* timer was introduced at the *UeManager*, which keeps track of the UEs transmissions. It is kept up to date by the *LteEnbMac*, which notifies the *LteEnbMac* whenever a new communication with the UE is scheduled and when it ended. On the inactivity notification, the *UeManager* starts the *DataInactivity* timer, which length can be set variably. The *UeManager* cancels the *DataInactivity* timer when it receives an activity notification before the timer expires. When the timer expires, the *UeManager* issues the transmission of the *RrcConnectionRelease-NB* message, including an allocated resume ID and the release cause "*rrc_Supsend*". This message once again was implemented bit-accurate to the ASN.1 definition.

To keep the simulation as realistic as possible, the UE-specific entities (SRB, DRB, etc.) must be saved, and its C-RNTI must be released. According to [2], the eNB has to save the entities after 10 s, but by LENAs implementation, this could lead to unexpected behavior, especially at higher repetitions and higher cell load. Thereby the duration until the eNB suspends its entities can be set dynamically. When the *LteEnbRrc* suspends the components of a UE, it begins by saving the *UeManager* instance in a table mapped to the resume ID. Further, it instructs the MAC-layer, the EPC, and the layers in between to save all components associated with the suspended C-RNTI. These are once again mapped to the resume ID.

Last, the *LteEnbRrc* cancels all pending events of the UE and removes the remaining information. When the UE receives the *RrcConnectionRelease-NB* message, it saves the received resume ID, notifies the upper layer about the released connection, and switches from the *CONNECTED* state to the *IDLE* state (or power-saving). When a suspended device later wants to reconnect to the cell, it also issues a RA. If the RA succeeds, the *LteUeRrc* then evaluates if it has a saved resume ID. In the case of a saved resume ID, it applies the newly received C-RNTI to its SRBs and DRBs. In theory, the DRBs are later, but the earlier initialization does not affect the performance. It then issues the transmission of the *RrcConnectionResumeRequest-NB* message containing the resume ID through SRB0.

On the reception of the *RrcConnectionResumeRequest-NB* message, the *LteEnbRrc* layer first checks its suspend table for the received resume ID. If it finds it, it resumes the *UeManager* and further advises every layer to resume the associated components and update their *C-RNTI* to the current one. Additionally, it has to delete the *UeManager* created by the RA procedure. Then the *UeManager* checks if it allows the resume procedure and, if so, transmits the *RrcConnectionResume-NB* message. Otherwise, it sends an *RrcConnectionSetup* or *RrcConnectionReject*, and the UE reacts accordingly. The *LteUeRrc* further responds with the *RrcConnectionResumeComplete-NB* message and thereby completes the resume procedure.

1.2.5 Power-Saving Features and Detailed Energy Model

The implementation of NB-IoT power-saving features goes hand in hand with the implementation of the connection resume procedure (figure 1.7). Therefore the state machines of the *LteUeRrc* and

the *UeManager* were further extended by two new states, the *IDLE_SUSPEND_EDRX* and the *IDLE_SUSPEND_PSM*. Both states can be enabled individually.

Table 1.1: Power Consumption of the Different BG96 Energy States

	Voltage [V]	Current [A]	Power [W]
PSM	3.8	3.9×10^{-6}	14.82×10^{-6}
DRX	3.8	1.56×10^{-3}	5.928×10^{-3}
eIDRX	3.8	0.81×10^{-3}	3.07×10^{-3}
Uplink	3.8	155×10^{-3}	589×10^{-3}
Downlink	-	-	80×10^{-3}
IDLE	3.8	0.81×10^{-3}	3.07×10^{-3}

Considering the features are enabled, both the *LteUeRrc* and the *UeManager* switch to the extended Discontious Reception (eDRX) state during the connection release procedure. Then a timer corresponding to T3324 is started for the eDRX mode, after which they move into the Power Saving Mode (PSM) state. The connection resume procedure is triggered when either the T3412 timer expires and the *LteUeRrc* tries a Tracking Area Update (TAU) or when the *LteUeRrc* receives data from its upper layer. More importantly, in addition to the power-saving states, a detailed energy model was implemented based on the ns-3 energy model.

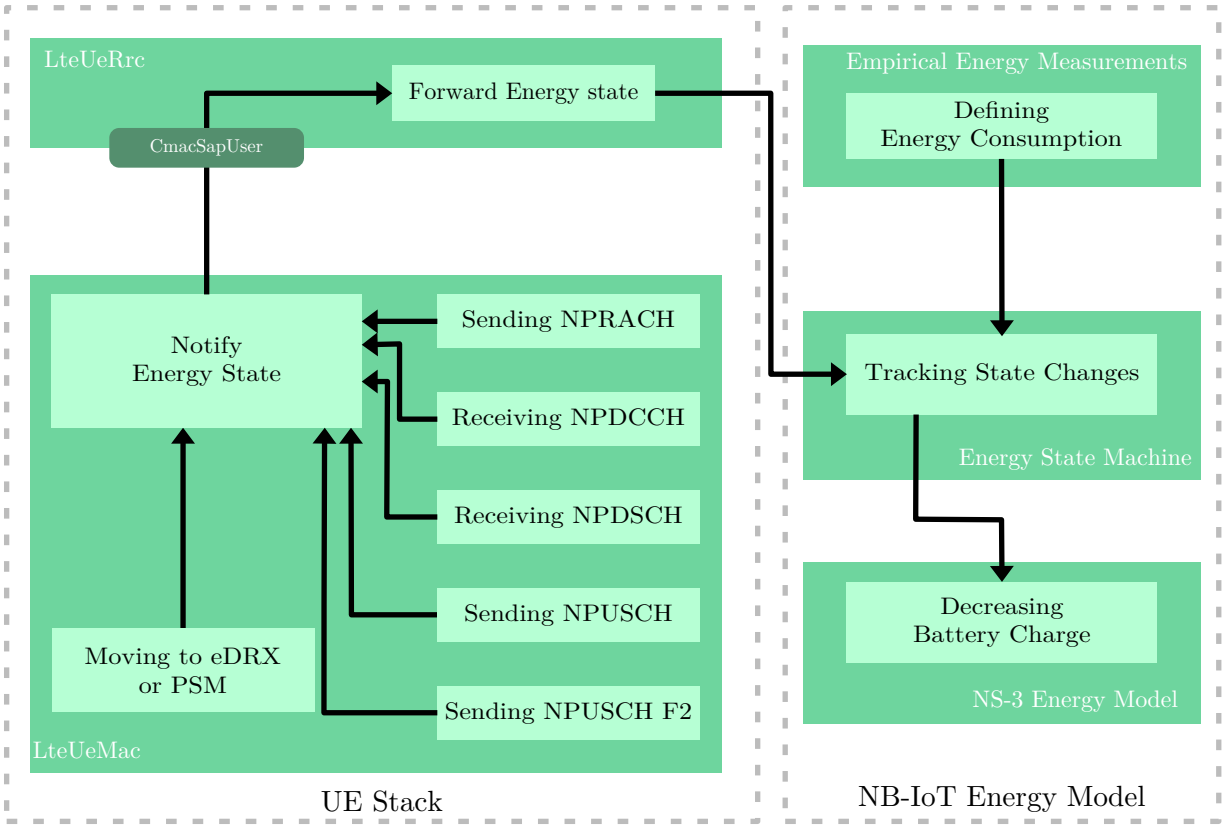

Figure 1.7: Implementation of the NB-IoT Energy Model

The energy model uses empirical energy measurements from actual LTE modules to allow a realistic representation. In this case, the Quectel BG96 [7] was used, but other models can be easily added. The table 1.1 shows the power consumption of the BG96 for each energy state. For a precise control of the individual power states, multiple entry points at the *LteUeMac* layers were implemented. These entry points include the DCI and packet reception, the random access, the NPDCCH decoding, etc.

When the theoretical power state of the UE changes, the corresponding layer notifies the state change to the energy model and also schedules the end of the current state. The energy model keeps track of the precise time the UE spends in which state on a subframe basis and thereby every time the energy models state changes, it determines the time spend in that state and then calculates the corresponding energy usage. Accordingly, the calculated energy consumption is then decreased from the ns-3 energy model.

1.2.6 Different Releases and Signaling Improvements

As stated in the [problem definition chapter], this thesis shall also analyze different NB-IoT releases and other signaling improvements.

Cellular IoT-Optimization

The first implemented NB-IoT improvement in LENA-NB is the Cellular Internet of Things Optimization (C-IoT-Opt). It allows user data to be transmitted over the NB-IoT control plane. While this allows lower complexity and lower device costs, the implementation focuses on appending user data to the *RRCConnection[Setup/Resume]Complete-NB*. Thereby, assuming a small payload size, one uplink message can be saved. To implement C-IoT-Opt, mainly the *LteUeRrc*, and the *LteEnbRrc* had to be modified. First, the *LteUeRrc* evaluates if it can use C-IoT-Opt and if the additional data would fit into the Message 5 (MSG5). Otherwise, the C-IoT-Opt would use the same messages as the typical setup/resume-procedure and thereby falls back to them. In theory, the transmission over the control plane would still differ from the data plane, but the main focus of the thesis is the reduced signaling overhead.

When MSG3 is passed to the *LteRlcTm*, the *LteUeRrc* also notifies the *LteUeMac* about the payload size of the MSG5. The *LteUeMac* then updates the DPR of MSG3 to advise the *LteEnbMac* to grant a bigger uplink opportunity. When sending MSG5, the upper-layer packet, saved in the *dedicatedInfoNas* field of the MSG5 struct, is appended to the ns-3 packet by the *LteEnbRrcProtocol*. It is not parsed as an ASN.1 octet string due to the high complexity of parsing an octet string of unknown size in a statically typed programming language like C++. On the *LteEnbRrcProtocol*, a trick can be used. It usually would not easily be possible to differentiate the append upper-layer data from the headers in front of it. Due to LENAs implementation and by sticking to its guiding principle in LENA-NB, every layer of the *LteEnb* protocol stack removes its related headers.

Accordingly, when the ns-3 packet reaches the *LteEnbRrc* layer, every NB-IoT header is removed, allowing the *LteEnbRrc* to pass the packet to its upper layers the same way as if it received it over a DRB. To not modify the following Evolved Packet System (EPS) stack, the *LteEnbRrc* layer adds a temporary EPS *ByteTag* to the packet, pretending it is a packet received over a DRB. The packet is then forwarded to the EPS.

Early Data Transmission

In Release 15, NB-IoT introduces the Early Data Transmission (EDT) feature. As a promising candidate for a significant overall performance improvement, it was also implemented into LENA-NB. Due to EDT using its own NPRACH resources, first, the *SIB2-NB* was extended by parameters like the EDT TBS, and the specific NPRACH resources. The configuration of the new *SIB2-NB* still takes place in the *LteEnbRrc*.

Next, the *LteUeRrc* was extended. When it is instructed to send a data packet, the *LteUeRrc* first evaluates if the packet would fit in the maximum TBS advertised from the new *SIB2-NB* and if it is even capable of using EDT. Then the *LteUeRrc* saves the packet and instructs the *LteUeMac* using a modified *StartConnectionNB* method, passing an EDT boolean, to initiate the modified RA. The *LteUeMac* then again evaluates its CE-level and first applies the standard NPRACH config. If the passed boolean implies that EDT should be used, the *LteUeMac* then adds the corresponding EDT parameters. The standard parameters must be applied first because then *SIB2-NB*s parameters for the NPRACH miss parameters like the maximum number of NPRACH attempts. Due to that, it was assumed that the standard parameters should be used for all missing parameters. Next, the *LteUeMac* performs a standard RA based on the EDT NPRACH resources.

When the *LteEnbMac* receives the NPRACH preamble, it has to evaluate if the preamble belongs to a regular random access or an EDT. Therefore it calculates the RA occasions of all NPRACH configs in the *SIB2-NB* and then compares them with the received preambles. When the *LteEnbMac* detects an EDT preamble, it triggers a normal RAR but increases its TBS to the one advertised in the *SIB2-NB*.

If the *LteUeMac* receives an uplink grant with TBS greater than 88 bit it can assume that the EDT is granted, and it notifies the *LteUeRrc* about the successful RA. Otherwise, if the *LteEnbMac* only grants a TBS of 88 bit, the *LteUeMac* assumes that it should use a standard connection setup/resume procedure instead. On the notification that the EDT was granted, the *LteUeRrc* prepares an *EarlyDataRequest* struct (again implement conform to its ASN.1 definition), fills in the necessary information, and appends the data packet again as the *dedicatedInfoNas* element. Then the *LteUeRrcProtocol* converts the struct to an *EarlyDataRequest-NB* message and transmits it using the SRB0.

On the *LteEnbRrc*, the reception procedure proceeds analogously to the C-IoT-Opt procedure. Further, it was assumed that the *LteEnbRrc* awaits a potential downlink message for the UE,

which is then appended to the *EarlyDataComplete-NB* message. For this implementation, it was assumed that the received data wouldn't exceed the downlink TBS. After the UE receives the *EarlyDataComplete-NB*, it moves directly back into its PSM.

1.2.7 Further NB-IoT specific modifications of LENA

Suspended attachment

Most of NB-IoTs applications are stationary, and UEs most likely only use the connection resume procedure daily. Accordingly, simulating the connection setup procedure in every simulation would be a significant overhead and prolong the overall simulation time. To allow the simulation to begin with "sleeping" devices directly, the *LteHelper* was modified to fake the connection setup procedure for the *LteEnbRrc* and the *LteUeRrc*.

First, the LteHelper passes the UEs International Mobile Subscriber Identity (IMSI) to the *LteEnbRrc* stack and initiates the creation of a corresponding *UeManger*, thereby bypassing the RA. Next, the *UeManger* was modified to execute the complete internal logical connection setup procedure without the actual message transfer. Last, the *UeManger* allocates a resume ID, moves into the PSM state, and returns the resume ID to the *LteHelper*.

The modifications to the *LteUe*'s stack turned out to be tricky because the System Information (SI) and DRB configurations of the *LteEnb* are first initialized at the actual beginning of the simulation. Still, the fake procedure is executed at the configuration state of the simulation. Accordingly, the next step of the procedure is scheduled into the first 20 ms of the simulation, allowing the *LteEnbRrc* and the EPC to be fully initialized and configured.

Further multiple interfaces were implemented, allowing the *LteHelper* to obtain the SIBs and the radio resource configs from the *LteEnbRrc*. In the last step, the *LteHelper* passes the resume ID, the cell id, the downlink Evolved Universal Mobile Telecommunications System (E-UTRA) Absolute Radio Frequency Channel Number (EARFCN), the radio resource configs, *SIB1-NB*, and the *SI-NB* to the *LteUeRrc*. The *LteUeRrc* then carries out the theoretical connection setup procedure, again without the actual message transfer, and then moves into the PSM.

2 Symbols

3GPP 3rd Generation Partnership Project

ACK Acknowledgement

AM Acknowledged Mode

AMC Adaptive Modulation and Coding

API Application Programming Interface

ARQ Automatic Repeat Request

ASN.1 Abstract Syntax Notation One

BLER Block Error Rate

BCCH Broadcast Control Channel

BSR Buffer Status Report

BSI Buffer Status Index

CCCH Common Control Channel

CE Coverage Enhancement

CIoT Cellular Internet of Things

C-IoT-Opt Cellular Internet of Things Optimization

CoAP Constrained Application Protocol

C-RNTI Cell-Radio Network Temporary Identifier

CP Cyclic Prefix

CQI Channel Quality Indicator

CRC Cyclic Redundancy Check

CSV Comma-Seperated Values

DAU Data Application Unit

DCCH Dedicated Control Channel

DCI Downlink Control Information

DPR Data Volume and Power Headroom Report

DRB Data Radio Bearer

DRX Discontinious Reception

DTCH Dedicated Transport Channel

DV Data Volume

EARFCN E-UTRA Absolute Radio Frequency Channel Number

eDRX extended Discontious Reception

EDT Early Data Transmission

eMTC enhanced Machine Type Communication

eNB evolved Node B

EPC Evolved Packet Core

EPS Evolved Packet System

E-UTRA Evolved Universal Mobile Telecommunications System

GCC GNU Compiler Collection

GPRS General Packet Radio Service

GPS Global Positioning System

GSM Global System for Mobile Communications

HARQ Hybrid Automatic Repeat Request

HSFN Hyper System Frame Number

IoT Internet of Things

IP Internet Protocol

IMSI International Mobile Subscriber Identity

KPI Key Performance Indicator

LC Logical Channel

LENA LTE EPC Network Simulator

LENA-NB LTE EPC Network Simulator Narrowband

LoRaWAN Long Range Wide Area Network

LPWAN Low Power Wide Area Network

LTE Long Term Evolution

LTE-M Long Term Evolution Machine Type Communication

LSB Least Significant Bit

M2M Machine-to-Machine

MAC Medium Access Control

MCL Maximum Coupling Loss

MCS Modulation and Coding Scheme

MIB Master Information Block

MIB-NB Master Information Block Narrowband

MME Mobility Management Entity

MQTT Message Queuing Telemetry Transport

MQTT-SN Message Queuing Telemetry Transport for Sensor Network

MSB Most Significant Bit

MSG1 Message 1

MSG2 Message 2

MSG3 Message 3

MSG4 Message 4

MSG5 Message 5

NACK Negative Acknowledgement

NAS Network Access Stratum

NB-IoT Narrowband Internet of Things

NDI New Data Indicator

NIDD Non IP Data Delivery

NPBCH Narrowband Physical Broadcast Channel

NPDCCH Narrowband Physical Downlink Control Channel

NPDSCH Narrowband Physical Downlink Shared Channel

NPRACH Narrowband Physical Random Access Channel

NPSS Narrowband Primary Synchronisation Channel

NPUSCH Narrowband Physical Downlink Uplink Channel

ns-3 Network Simulator 3

NSSS Narrowband Secondary Synchronisation Channel

OFDM Orthogonal Frequency Division Multiplexing

OFDMA Orthogonal Frequency Divison Multiple Access

PCAP Packet Capture

PDCCH Physical Downlink Control Channel

PDCP Packet Data Convergence Protocol

PGW Packet Data Network Gateway

PHR Power Headroom Report

PHY Physical

PLMN Public Land Mobile Network

PRB Physical Resource Block

PSM Power Saving Mode

PSS Primary Synchronization Channel

PTW Paging Transmission Window

QoS Quality of Service

QPSK Quadrature Phase Shift Keying

RA Random Access

RACH Random Access Channel

RAP Random Access Preamble

RAP-ID Random Access Preamble Identifier

RAR Random Access Response

RB Resource Block

RE Resource Element

RLC Radio Link Control

RRC Radio Resource Control

RA-RNTI Random Access-Radio Network Temporary Identifier

RTT Round Trip Time

RSRP Reference Signal Receive Power

RSRQ Reference Signal Received Quality

RU Ressource Unit

PDU Protocol Data Unit

SAP Service Access Point

SCS Subcarrier Spacing

SCEF Service Capability Exposure Function

SDU Service Data Unit

SDR Software Defined Radio

SFN System Frame Number

SF System Frame

SGW Serving Gateway

SI System Information

SI-NB System Information - Narrowband

SIB System Information Block

SIB1 System Information Block 1

SIB-NB System Information Block Narrowband

SIB1-NB System Information Block 1 - Narrowband

SIB2-NB System Information Block 2 - Narrowband

SIB3-NB System Information Block 3 - Narrowband

SRB Signalling Radio Bearer

SRB0 Signalling Radio Bearer 0

SRB1 Signalling Radio Bearer 1

SRS Sounding Reference Signal

srsRAN Software Radio System Radio Access Network

SRB1bis Signalling Radio Bearer 1 bis

TA Timing Advance

TAU Tracking Area Update

TB Transport Block

TBS Transport Block Size

TBSI Transport Block Size Index

TCP Transmission Control Protocol

TM Transparent Mode

TTI Transmission Time Interval

UDP User Datagram Protocol

UL Uplink

UE User Equipment

Bibliography

[1] Network Simulator 3. `https://www.nsnam.org`, 2008-2021.

[2] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification. Technical Specification (TS) 36.331, 3rd Generation Partnership Project (3GPP), 01 2021. Version 13.17.0.

[3] 3GPP. LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification. Technical Specification (TS) 36.321, 3rd Generation Partnership Project (3GPP), 07 2021. Version 13.9.0.

[4] FemtoForum. FAPI 2.0. `https://www.eurecom.fr/~kaltenbe/fapi-2.0/index.html`.

[5] Joachim Sachs, Y.-P. Eric Wang, Mårten Sundberg, Johan Bergman, and Olof Liberg. *Cellular Internet of Things*. 2017.

[6] Jonas Gibbels. Narrowband Internet of Things: Eine Analyse der Batterielaufzeit unter Einbezug des Signalisierungsaufwands. Masters thesis, TU Dortmund, oct 2020.

[7] Quectel. BG96 Hardware Design. 2017.