


LOLA	Project N° 248993	WP5	 Validation Results for WP4 Algorithms on Testbed 1 V1.0 2013-06-01-2013-06-30
------	-------------------	-----	---



<b>Project Number:</b>	<b>Project Acronym:</b>	<b>Project Title:</b>
248993	LOLA	Achieving Low-Latency in Wireless Communications

<b>Instrument:</b>	<b>Thematic Priority</b>
STREP	The network of the future

<b>Workpackage:</b>
WP5

<b>Deliverable or Document Title:</b>
Validation Results for WP4 Algorithms on Testbed 1

<b>Keywords :</b>
OpenAirInterface, traffic modeling, KPI, LTE, M2M, CBA, latency, performance evaluation, experimentation.

<b>Organization name of lead contractor for this deliverable:</b>	<b>Document version:</b>
EURE	V1.0

<b>Partners (organisations) contributing to this document :</b>
EURE, TuV, MTS

<b>Contractual Delivery Date:</b>	<b>Actual Delivery Date:</b>
2013-03-31	2013-06-3

<b>Start date of project:</b>	<b>Duration:</b>
2010-01-01	39 months

Dissemination level ( Project co-funded by the European Commission within the Seventh Framework Programme)		
<b>PU</b>	Public	<b>X</b>
<b>PP</b>	Restricted to other programme participants (including the Commission	
<b>RE</b>	Restricted to a group defined by the consortium (including the Commission)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission)	

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4 Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

## Revision History

The following table is a record of the main modifications done to the document since its creation.

Revision	Date	Author (Organisation)	Description
V0.1	2013-03-31	EURE	ToC
V0.2	2013-04-15	MTS	Annex A - detailed results for delay model
V0.3	2013-04-22	MTS	Annex A – TCP and UDP processing delay analysis added
V0.4	2013-05-15	EURE	Contribution of to section 2
V0.5	2013-05-21	EURE	Contribution to section 3
V0.6	2013-05-22	TuV	Contribution to section 2
V0.7	2013-05-31	EURE	Contribution to section 3, and results
V1.0	2013-06-03	EURE	Review, minor corrections, and release the final version

# Table of Contents

1. EXECUTIVE SUMMARY	5
2. TESTBED1 COMPONENTS AND CONFIGURATION	6
2.1. OpenAirInterface platform and its Overall Configuration.....	6
2.2. Emulation Platform and Process (revisited).....	7
2.3. Selected WP4 Algorithm .....	8
2.3.1. General idea of contention based access .....	9
2.3.2. CBA Software architecture(updated).....	10
2.3.3. Implementation .....	12
2.4. Application Traffic model (revisited) .....	13
2.5. Key Performance Indicators (revisted) .....	17
2.6. Precise Latency Calculation Methodology (revisted).....	19
2.6.1. Network GPS synchronization.....	20
2.7. Network Delay Model (updated) .....	20
2.8. Towards the Large Scale Experimentation.....	23
2.8.1. PHY abstraction .....	23
2.8.2. Parallelism Methodology.....	33
3. VALIDATION RESULTS	37
3.1. CBA implementation validation .....	37
3.1.1. RRC layer.....	37
3.1.2. MAC layer.....	38
3.1.3. PHY layer .....	42
3.2. Latency results .....	44
3.2.1. Experimentation with different type of traffic .....	46
3.2.2. Experimentation with different backoff window size .....	49
3.2.3. Experimentation with different number of CBA groups .....	51
4. CONCLUSION	54

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

5. ACRONYMS AND DEFINITIONS	55
5.1. Acronyms .....	55
6. REFERENCES	56
ANNEX A UDP AND TCP MEASUREMENT ON LIVE HSPA NETWORK (MTS/TuV) – DETAILED RESULTS FOR DELAY MODEL	57

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

## 1. EXECUTIVE SUMMARY

This deliverable specifies the implementation and performance evaluation for the selected WP4 algorithm, contention based access (CBA), on the OpenAirInterface platform (<http://www.openairinterface.org/>).

The OpenAirInterface (OAI) platform is an open-source hardware/software development platform created at EURECOM for innovation in the area of digital radio communications. We elaborate the main OAI components, its overall configuration as well as its emulation process. We highlight the traffic pattern and the key performance indicators described in D3.5 and D3.6. We further present the efforts made toward a large-scale emulation, namely phy abstraction, and parallelism.

The contention based access method is proposed in [1], which is used to reduce to redundant signalling in regular scheduling. The general idea of CBA is that a UE sends a packet without dedicated resource allocated by eNB. Instead, a UE sends a packet on a common resource, i.e. multiple UEs contend for a resource to send packets. We explain the contention based access method, its software architecture, and its modifications to the LTE protocol stack in section 2.3.

The validation results for the implementation and performance evaluation of CBA are provided in two categories. Firstly, we present the signalling validation results for the implementation of CBA on the OAI platform. The signalling introduced by CBA is validated in RRC layer, MAC layer, and PHY layer, respectively. Secondly, we provides emulation results used to compare CBA with the regular scheduling method used in LTE, as well as the results aimed to evaluate the effects of different parameters (traffic pattern, backoff window size and number of CBA groups) on the CBA performance.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

## 2. TESTBED1 COMPONENTS AND CONFIGURATION

The testbed1 is design to experimentally validate the three fundamental objectives of LOLA [2]-[3]. In particular, it allows deriving the traffic model (WP3) for the selected realtime M2M/gaming applications (WP2) and validating the innovations in the access stratum L1/L2 (WP4). The performance of the proposed contention-based random access method described in D4.2 will be tested over a mixed human and machine traffic models. This testbed allows a repeatable and controlled set of experimentations for a large scale emulated network in laboratory environment. It fills the gap between the simulation and real experimentation by providing the baselines for protocol validation, performance evaluation and system testing. In this section, we will briefly explain the components and configuration of the testbed 1 as well as the traffic model, KPI, network synchronization for accurate one-way delay calculation. Furthermore, we present the efforts made toward large-scale experimentation using realistic and efficient PHY abstraction method and software parallelism applied to the testbed 1.

### 2.1. OpenAirInterface platform and its Overall Configuration

OpenAirInterface is an open-source platform for experimentation in wireless systems with a strong focus on cellular technologies such as LTE and LTE-Advanced. The platform comprises both hardware and software components and can be used for simulation/emulation as well as real-time experimentation. It comprises the entire protocol stack from the physical to the networking layer. The objective of this platform is to fill the gap between the simulation and real experimentation by providing the baselines for protocol validation, performance evaluation and pre-deployment system test.

OpenAirInterface comprises a highly optimized C implementation all of the elements of the 3GPP LTE Rel 8.6 protocol stack for UE and eNB (PHY, MAC, RLC, RRC, PDCP, NAS driver). Apart from real-time operation of the software modem on a hardware target, the full protocol stack can be run in emulation. The OpenAirInterface emulation environment allows for virtualization of network nodes within physical machines and distributed deployment on wired Ethernet networks. Nodes in the network communicate via direct-memory transfer when they are part of the same physical machine and via multicast IP over Ethernet when they are in different machines. In the first case the emulator can either be run with the full PHY layer or with PHY abstraction while in the latter case nodes interface at layer 2. The rest of the protocol stack (MAC and RLC) for each node

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

instance uses the same implementation, as would the full system. Each node has its own IP interface that can be connected either to an application or a traffic generator. The emulator also comprises a simple mobility model and channel models including path loss, shadow fading and stochastic small scale fading.

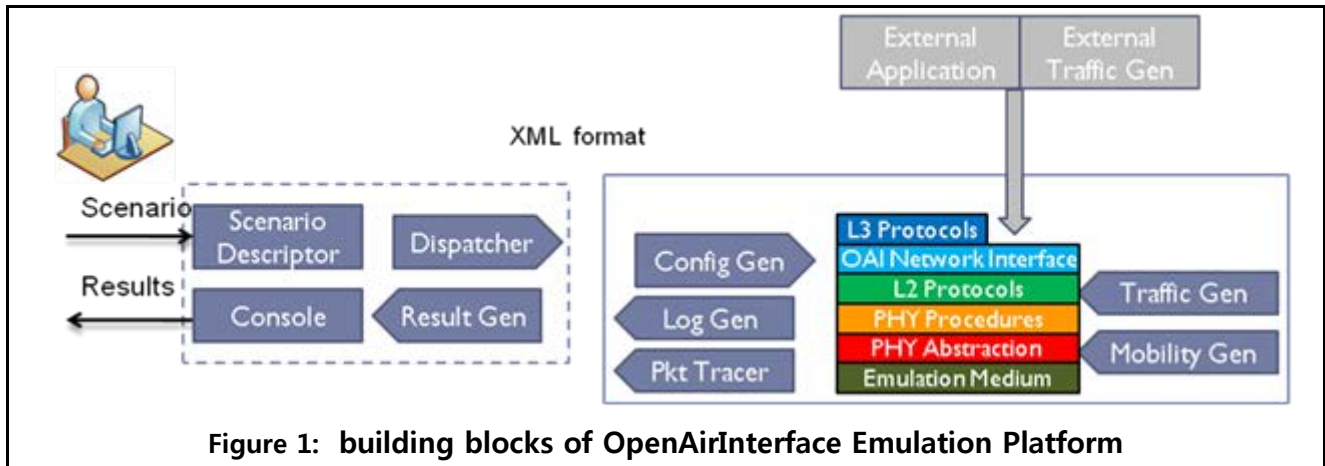
The testbed 1 experimentation is performed on the top of OpenAirInterface emulation platform. The hardware platform is a laptop equipped with a quad-core CPU running OAI emulator and protocol stack using Linux on Ubuntu 12.04. We carried out one-way delay (OWD), jitter, and goodput measurements in the soft realtime mode for LTE operating in TDD frame configuration 3 for 5MHz bandwidth, in a simple cellular network topology composed of one eNB (enhanced-NodeB) and 6 UEs (User Equipment). The rest of the network including mobile core network, IP backbone, and application server are emulated in terms of additional latency as the purpose of the experiment is to measure the end-to-end OWD in the data-plane. We make use of OAI scenario descriptor to layout the experiment such that the reproducibility is preserved and results can be regenerated. The simulation is run for 10 seconds (i.e. 1000 LTE TDD frames). We applied the traffic pattern of M2M application characterized by small and large constant sized packets with random inter-arrival times in uplink. The data rate is between 10 Kbps 20 kbps for most cases, and thus the RTT should be below 50ms to avoid any accident for realtime M2M application.

## 2.2. Emulation Platform and Process (revisited)

Figure 1 shows the high level software architecture of the emulated platform itself and the main building blocks. The user scenarios are described using baseline scenario descriptor and then encoded into xml format and dispatched across OpenAirInterface hardware platform according to some predefined rules. Then, the config generator block translates the high level scenarios into low level configuration files understandable for traffic/mobility generator, UE/eNB protocol stack, phy abstraction and emulation transport medium. The real applications are attached on top of some emulated UE/eNB and the remaining traffic pattern and mobility model will be generated automatically. The behavior of the wireless medium is modeled using a PHY abstraction unit which emulates the error events in the channel decoder and provides emulated measurements from the PHY in real-time [4]. The remainder of the protocol stack for each node instance uses a complete implementation as would a full-RF system. The Log generator is in charge of recording the emulator activities according to the user defined log level, while the packet

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

trace generator captures the frame (potentially extract on-the-fly the relevant field). The result generator produces the user defined test results using the outcome of log generator and packet trace generator [5].



The process of emulation has three phases, which are performed sequentially as follows:

1. **Input:**
  - a. Description of application scenario
  - b. Initialization and configuration of all the blocks
2. **Execution:**
  - a. Protocols: PHY procedures, L2 protocols, traffic generator, and packet tracer
  - b. Connectivity: PHY abstraction, channel model, and mobility model
  - c. Emulation data transport: IP multicast, shared memory
3. **Output:**
  - a. Protocol validations and execution logs
  - b. Performance evaluation metrics
  - c. System testing

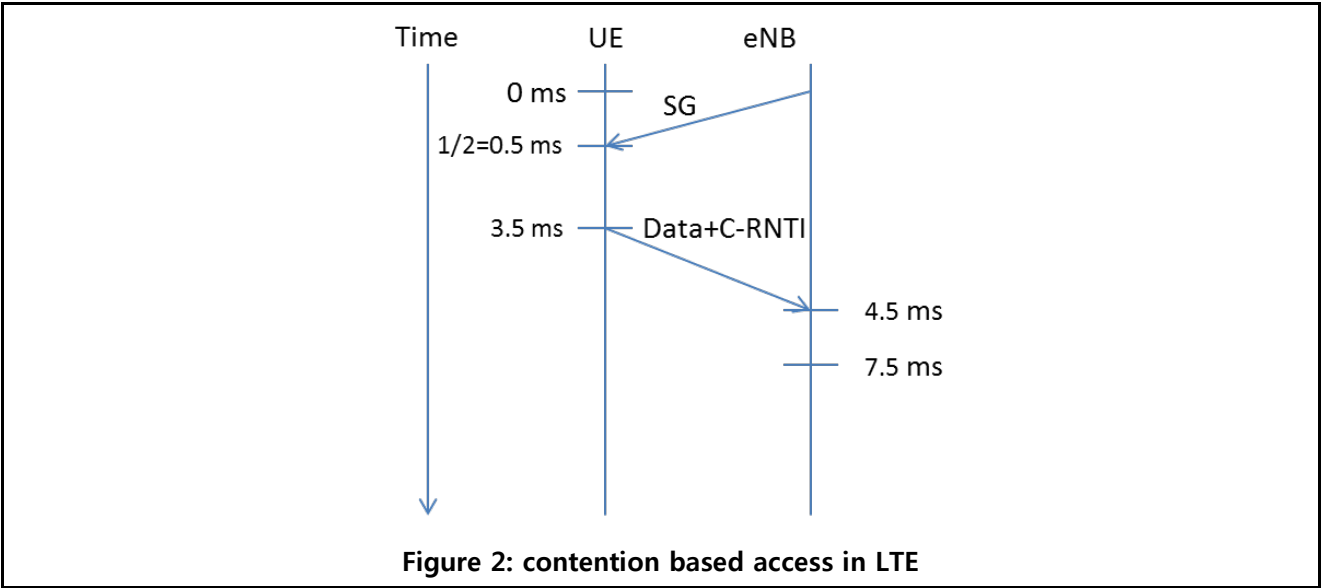
### 2.3. Selected WP4 Algorithm

Among the various applications provided by LTE, M2M and online gaming are the most promising applications. However, the regular mobile networks are designed for human-to-human (H2H) communications, targeting the voice/multimedia transmissions with a continuous flow of packets, which are not suitable for such application characterized by low data rate delay-sensitive sporadic traffic. To enable an efficient and low latency uplink access for sporadic traffic, a contention based access (CBA) method is proposed in D4.2 [1][6-7].

The following subsections describe the idea of CBA, its software architecture and its integration with the testbed 1.

### 2.3.1. General idea of contention based access

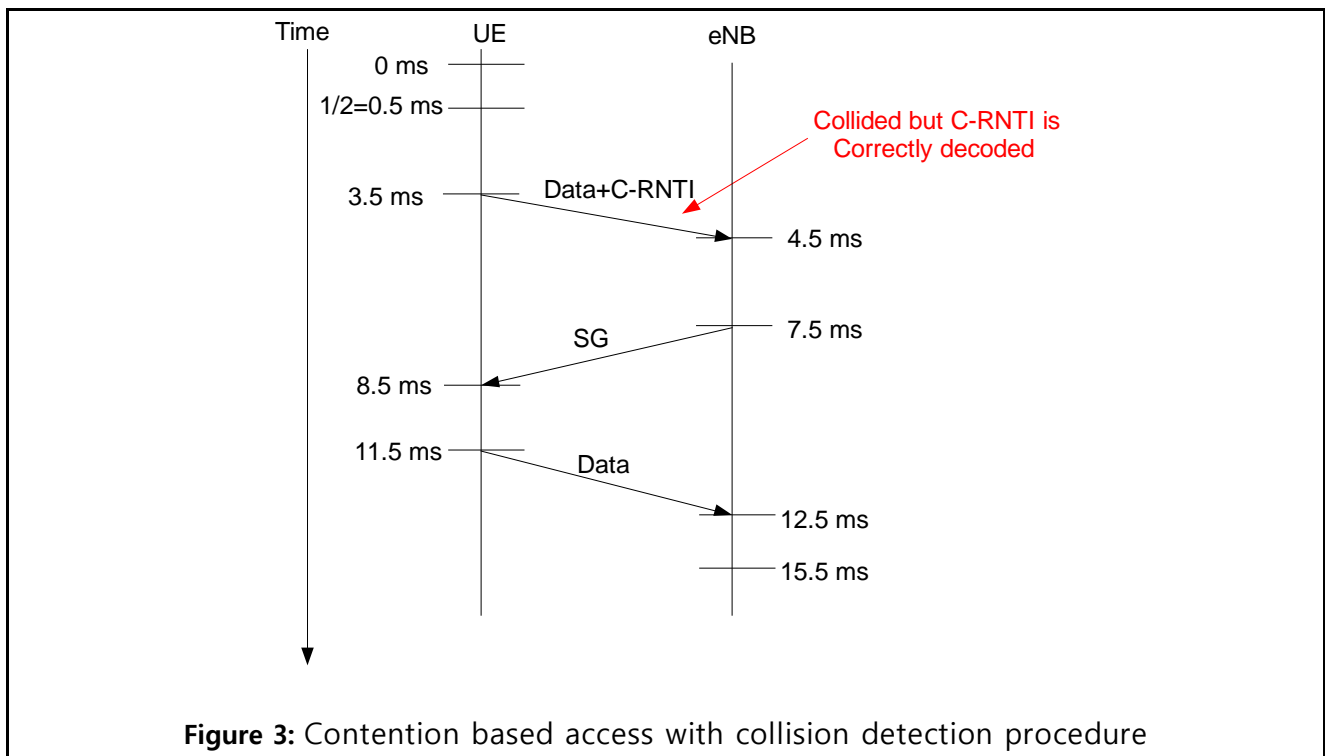
The main feature of contention based access is that UE is not assigned with dedicated resource. Instead, the resource is allocated for all or a group of UEs. A UE randomly selects its resource and sends a data packet on it. The procedure for CBA is shown in Figure 2. First, the eNB informs UEs about the resource allocation information for CBA via the scheduling grant (SG) information which cost 0.5ms provided that the CBA resource is available in each subframe. Then, after decoding the SG information which cost 3ms, the UE selects its resource block randomly and sends the data packet . The channel latency for this packet scheduling procedure is 7.5ms (not including the time waiting for the ACK information), which is much smaller than 22.5ms of the regular scheduling case.



As the CBA resources are allocated for all or a group of UEs, collision happens when multiple UEs within a group select the same resource. To address the problem of collision,

in our method each UE sends its identifier, C-RNTI, along with the data on the randomly selected resource. The C-RNTI is of very small size, therefore it can be transmitted with the most robust modulation and channel coding scheme (MCS) without huge overhead. MU-MIMO detection is used at the eNB side to decode packets; the highly protected C-RNTIs

from different UEs might be decoded even if they are sent on the same resource. If the C-RNTI of a UE are successfully decoded while its data payload is lost, dedicated resource is allocated for that UE by eNB. With the allocated resource, the UE sends its data packet; the latency for this procedure is 15.5ms as shown in Figure 3, which is less than latency of the regular scheduling case.



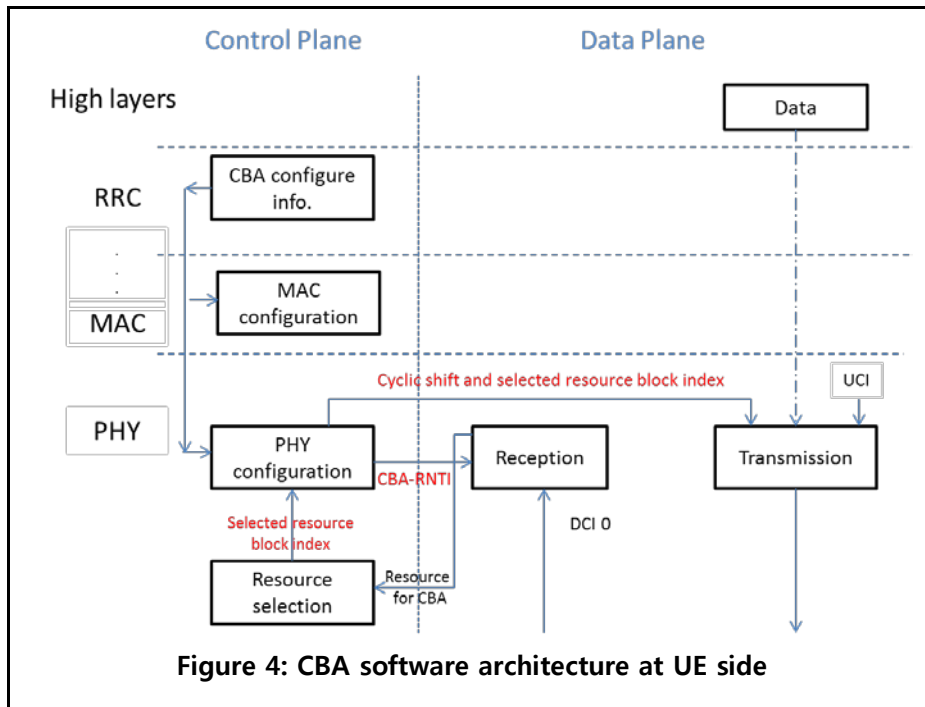
### 2.3.2. CBA Software architecture(updated)

The software architecture for CBA at the UE side is shown in Figure 4.

- The UE uses the received RRC message to configure its MAC and PHY layers. With RNTI allocated for CBA transmission, the UE decodes the DCI 0 information to locate the resource allocated for CBA.
- With the resource allocation information for CBA, the resource selection module randomly selects resource and passes the result to physical configuration module.
- The physical configuration module sets the proper parameters for transmission using the cyclic shift information obtained from the RRC message and the results from the resource selection module.

- Finally the transmission module sends the data and UCI information following the instructed configuration.

Since the resource is not UE specific, collision may happen when multiple UEs select the same resource. If the UE does not receive the ACK information 8ms after the initial transmission, the UE starts a retransmission following the same procedure as described above.



The software architecture for CBA at the eNB side is shown in Figure 5.

- The eNB sends the CBA configuration information to UEs through the RRC message. In addition to that, the eNB also performs resource allocation for CBA, for which the related resource allocation information is sent with DCI 0 and the CRC parity bits of DCI 0 is scrambled by RNTIs allocated for CBA transmission.
- As for the reception, the ACK information is sent to the transmission module for the correct received CBA packets such that it can be sent on the PHICH channel. In contrast to that, for the incorrect received packet (both data and control information are corrupted) the error information is sent the CBA resource allocation module so as to proper resource can be allocated for the UEs.

- The successfully decoded C-RNTIs of the collided UEs are sent to regular scheduling module and hence the specific resource can be allocated for those UEs and signalled through the DCI 0 information.

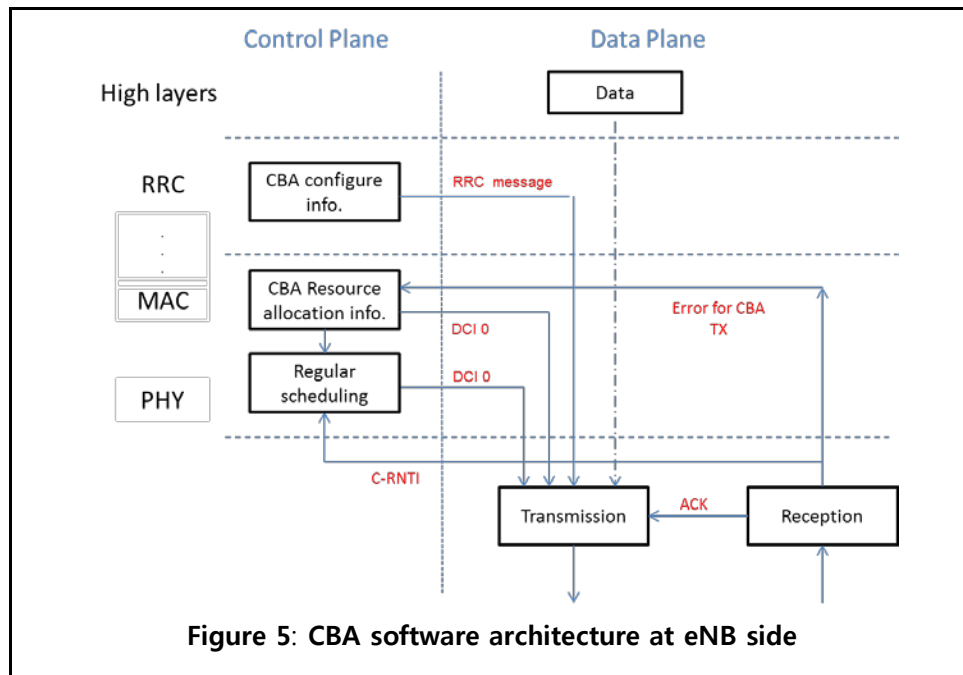


Figure 5: CBA software architecture at eNB side

### 2.3.3. Implementation

To implement CBA in the testbed 1, the following modifications to the LTE standard should be carried out.

- **RRC signaling to informs UE about CBA-RNTI (eNB side)**

The CBA-RNTI, which is used by a UE to decode the resource allocated information for CBA, is allocated by eNB during the RRC connection reconfiguration procedure for the data radio bearer (DRB) establishment. To implement this procedure, the CBA-RNTI has be added to the RadioResourceConfigDedicate information element. It should be mentioned that the CBA-RNTI is not UE specific. Instead, all UEs or a group of UEs have a common CBA-RNTI configured by RRC signaling.

- **Resource allocation for CBA (eNB side)**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

eNB allocates resource for CBA. With this resource allocation result, the eNB generates the DCI 0 information, which can be decoded by UEs to locate the CBA resource.

- **Locate the CBA resource (UE side)**

A UE uses its CBA-RNTI to decode the DCI 0 information to locate the resource allocated for CBA.

- **CBA data transmission (UE side)**

A UE sends the data packet on the allocated CBA resource informed by eNB. Moreover, to help eNB decode the packet sent on the CBA resource, a UE should inform eNB about the modulation and coding scheme (MCS) for the data payload. To do that, a UE sends MCS, C-RNTI along with the data in a subframe. The MCS for the bit stream of uplink MCS and C-RNTI (16 bits for C-RNTI and 4 bits for MCS) is fixed and known to eNB.

- **CBA data reception (eNB side)**

As the CBA data format is different from the legacy format, a eNB should decode the CBA data in a special way. For the data sent on the CBA resource, the eNB should be aware of that part of payload is the UL-SCH data while the other part is MCS and C-RNTI.

- **CBA HARQ procedure (eNB side)**

For a successfully received packet, the eNB sends the ACK in a standardized way (no modification is needed). For an erroneous packet, no NACK is sent by eNB.

## 2.4. Application Traffic model (revisited)

In the context of the LOLA project the OAI was extended with a program module for the integration of a traffic generator. The tool is part of the current open air interface source code tree, which can be downloaded at <http://www.openairinterface.org>.

The tool is a module of the OpenAirInterface written in C programming language, which allows for the generation of random traffic patterns which are characterized by random processes of packet size and packet inter-arrival time with given probability density

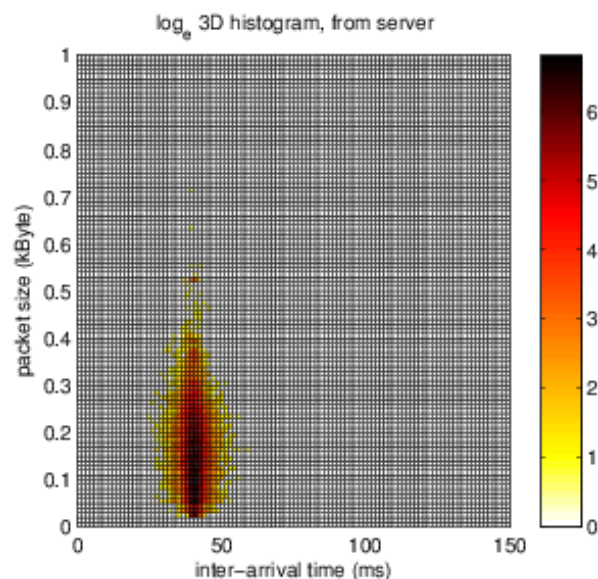
LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

functions, auto-correlation functions and cross-correlation function. The respective modelling approach is commonly deployed for Video, Online-gaming and VoIP traffic patterns. The tool is optimized for low computational complexity.

In the following we will describe the parameters used for two different test cases, namely online gaming and autopilot.

#### 2.4.1. OpenArena

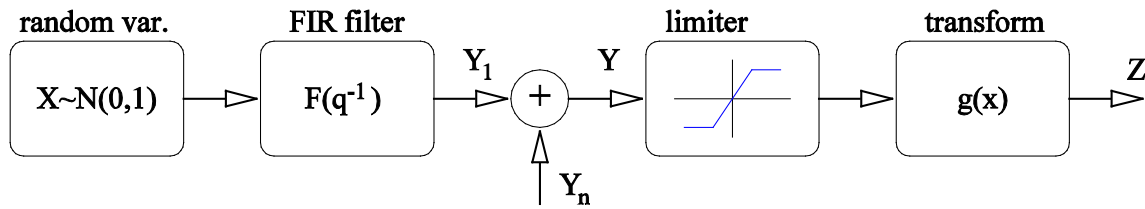
The first traffic source selected for the demonstration is OpenArena. The application representing the group online gaming is "Open Arena". This game is an open source project derived from the game Quake3 Arena written and published by id-software. The traffic patterns of this application have been analyzed in D 3.3 in this project. The application traffic is encapsulated into UDP packets for network transmission. The application itself takes care about packet loss and reordering on its own. The traces for the derived traffic generator have been collected at the IP layer. An example is shown here in Figure 6 where the packet size of the downlink is depicted over the inter arrival time between two consecutive packets. It is clearly visible that the parameter inter arrival time has a very small spread while the according packet size may vary, for further reading please refer to D 3.3 of the Lola project.



**Figure 6:** Packet Size vs Interarrival Time (OA, DL) (source: D3.3)

Detailed analysis showed that the traffic traces contained non neglect able long time correlations and also very long tailed ACF functions. In the following deliverable D3.5 of the Lola project we developed a new approach for a traffic model capable to reassemble

these values. It is capable of modeling not only the distributions of packet size and rate but also the autocorrelation functions between consecutive packets. The elements of the model used in the test are depicted in Figure 7.



**Figure 7: Online Gaming Network Traffic Model (Source D3.5)**

The parameterization of the main parameters for the model in case of Open Arena traffic is the following (Source D3.5):

- **Packet size PDF in the downlink direction:** This PDF can be found in Fig. 8, D3.3. We modeled it by transforming a Gaussian distribution with zero mean and unit variance by a polynomial, as described above. The polynomial is  $g(x) = 140 + 75x + 40x^2 - 5.7x^3 + 1.6x^4 + 0.7x^5 - 0.31x^6$ , with values in Bytes.
- **Packet size PDF in the uplink direction:** It can be observed in Fig. 9, D3.3. Since it is sharp and concentrated around its mean of 42 Bytes, with a standard deviation of 3.3 Bytes, it is modeled constant with its mean.
- **Packet inter-arrival time PDF in the downlink:** It can be found in Fig. 8, D3.3. Since it is sharp and concentrated around its mean of 40ms with a standard deviation of 2.8ms, we model it as constant with its mean.
- **Packet inter-arrival time PDF in the uplink direction:** It can be found in Fig. 9, D3.3. Since it is sharp and concentrated around its mean of 11ms with a standard deviation of 0.7ms, we model it as constant with its mean.
- **Packet size ACF the downlink direction:** This ACF is shown in Fig. 30 in D 3.5. It is induced by the FIR filter, which in this case has the coefficients of  $f(z) = 0.41 \cdot \delta(z) + 0.083 \cdot \exp(0.05 \cdot z) + 0.047 \cdot \exp(0.002 \cdot z)$  and a length of 500 taps.

#### 2.4.2. Autopilot

The second traffic source selected in the demonstration setup is the Autopilot scenario. This traffic pattern represents an M2M application exchanging data between car and infrastructure. The scenario was defined in D 2.1 Chapter 4.1. The traffic pattern itself is discussed in D3.3 and a traffic model is presented in D3.5.

The Autopilot scenario focuses on vehicle collision detection and avoidance (especially on highways) and urgency measures taken in case of accident. The scenario concerns mainly

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

the pre-crash sensing, warning and collision detection based on car telemetry in exchange with a backend system. The backend system is sending out notifications to all vehicles in the close surrounding of the actual event updating the customers about new hazards on the road.

The latency requirements derived in D 2.1 were based on the average breaking distance at 100 km/h at normal road conditions combined with the reaction time for the other drivers in the close vicinity of the accident. The derived end to end delay resulted into 30 ms as the total latency budget for the whole system. In the following we will refer to this value, for more details please see D 2.1, chapter 4.1.1 and 4.1.2.

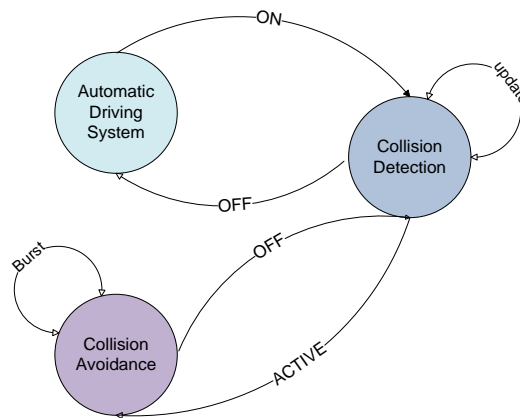
In contrast to the online gaming applications the M2M traffic patterns for the scenarios defined in D 2.1 could not be measured from real applications. Therefore we developed an M2M traffic frame work in D3.5 based on Markov chains representing the typical states of an M2M application. The actions identified in D 2.1 are:

- The UE connected to a mobile network (ACTIVE state)
- The client request Auto-pilot registration
- The Auto pilot registration confirmed
- The client sends data (GPS, speed, time) continuously every 20 (25) ms
- The Application server sends the emergency signals (warning and actuator commands) when needed (the delay of these messages is critical).

These states where transformed following the M2M traffic generation framework into the following states:

- Automatic Driving System,
- Collision Detection,
- Collision Avoidance.

The state diagram with the according state transmissions is depicted in Figure 8.



**Figure 8: Autopilot Reference Model**

The parameters we used in the demonstration test are given in Table 1 (D3.5).

**Table 1:** Parameters for M2M model Auto-pilot

Traffic	Avg. N	$P_{OH}$	$P_{PL}$	$F_{Keepalive}$ $F_{Heartbeat}$	$F_{Trigger}$	$F_{EventThreshold}$
Keep-alive (cars)	50	50	1KB	10ms-100ms	NA	NA
Keep-alive (server)	50	50	1kB	1s	NA	NA
Burst (server)	50	50	1-2 KB	NA	NA	10ms

This traffic generation tool will be used for the upcoming results in the demonstration chapter.

## 2.5. Key Performance Indicators (revisted)

In order to quantify the improvements introduced in WP4 of the Lola project we use key performance indicators (KPI) discussed in D3.6. In the following we describe the KPIs selected for the demonstration setup and the implementation into the hardware setup itself.

### 2.5.1. Generic Technical KPI Metrics for LOLA

The following parameters reflect the technical KPI values based on 3GPP and IETF proposals and the input of WP4 in D 3.6 of the Lola project. These parameters are:

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

- One-way delay and loss,
- Round-trip delay,
- Delay variation.

The first generic KPI is the one-way-delay (OWD). The OWD is defined as the time a datagram travels between a source A and a destination B. More details can be found in D3.6.

The second parameter is the time a packet needs to travel from A to B and back. The value reflects the reaction time a certain setup provides to the applications. This is interesting especially in cases where the delay is not symmetric.

The last parameter is the second order statistic of the delay parameter, the variation. A high variation indicates a high probability of large deviation from the mean value of the delay parameter, e.g., and outage due to buffer stalling. In a combination with the optimizations and implementations done in WP3, WP4 and WP5 the resulting technical KPI are:

- KPI1: Average One Way Delay (Direction, Datarate, Users, Traffic Pattern) The delay is extended for the number of users of the application, the datarate and the traffic pattern used.
- KPI2: Jitter of One Way Delay (Direction, Datarate, Users, Traffic Pattern) The delay is extended for the number of users of the application, the datarate and the traffic pattern used.

### 2.5.2. Online Gaming KPI Metric for LOLA

The application online gaming is a constant interaction of one gamer with a central server station. A generic technical KPI is not suited to capture the nonlinear impact in this interactions, e.g., a tresh hold of delay which if exceeded leads to an unplayable result for the human.

Such scenarios can be covered by subjective quality measures, like Mean Opinion Score (MOS), Perceptual Speech Quality Measure (PSQM) and Perceptual Evaluation of Video Quality (PEVQ). The MOS is expressed as a single number in the range 1 to 5, where 1 is lowest perceived application quality, and 5 is the highest perceived application quality measurement. The MOS is generated by averaging the results of a set of standard, subjective tests where a number of candidates rate the experienced application quality of test scenario, e.g., test game, over the communications medium being tested, e.g., LTE with and without the Lola improvements of WP4. A candidate is required to give each scenario a rating from 5 to 1, see D 3.6 for more details.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

The mapping technical sound KPIs to so call mean opinion scores is based on such test runs with many candidates. The KPI we have selected for online gaming is based on a MOS score based on a mapping of evaluated MOS numbers onto a delay and jitter function:

$$MOS_{gaming} = -0,00000587 * x^3 + 0.00139 * x^2 - 0,114 * x + 4,37,$$

$$X = 0,104 * RTT_{avg} + Jitter_{avg}.$$

### 2.5.3. M2M KPI Metrics for LOLA

The M2M scenario is differs from the online gaming given in the previous chapter. First in this application only machines exchange data, therefore a direct use of technical KPIs is possible. However the number of nodes is not neglectible small for the cell itself. In fact the expected nodes per cell in the current discussions in the standardization range up to 10000 per cell. In D 3.6 we also identified the different operation states of an UE in the mobile LTE network as an important factor, e.g., if there are no pre allocated resources for a random M2M packet the LTE UE has to request these values causing additional delay, see D 3.5. for measurement results and D 4.2 for theoretical analysis and improvements. Based on these result the following metrics for the M2M KPIs have been selected:

The generic metric is:

KPI\_M2M\_1 = IPTD(Number of Clients, Operation Mode, Activation),

KPI\_M2M\_2 = IPDV(Number of Clients, Operation Mode, Activation),

KPI\_M2M\_3 = IPLR(Number of Clients, Operation Mode, Activation).

## 2.6. Precise Latency Calculation Methodology (revisted)

The measurement of latency between different points in a distributed network is challenging as the measurement nodes need to measure at different network-places in a time synchronized way. A recording node must have the same clock as the others, and must be synchronized. As the delay in both directions is considered to be asymmetric and strongly varying, different nodes cannot estimate their clock offset by using round-trip time measurements. Any node must be driven by the same clock. This is possible by the use of the GPS system, which has an atomic clock time base accessible via GPS receivers offering an puls per second (PPS) output. The achievable precision by GPS timing is 20ns.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

### 2.6.1. Network GPS synchronization

The time synchronization of the measurement setup is based on GPS receivers. Each receiver is delivering an PPS impulse to an interface of the demonstration unit. Based on these impulses the system can be tuned to a precision of up to 100 $\mu$ s in time. It is therefore possible to do distributed simulations and demonstration on different PC without the risk of running async in the setup.

The details of the setup are given in D 3.2 of the Lola project. And a possible implementation is shown in D 3.3. The possible precision of the setup is analyzed in D 3.2 in detail and shows a performance of well below 100 $\mu$ s.

The setup has been implemented to the Open Air Interface testbed. The PPS impulse of the units is distributed via cable to the serial interfaces of each PC. The test setup used is the one introduced in D 3.2 and depicted here in the following Figure 9.

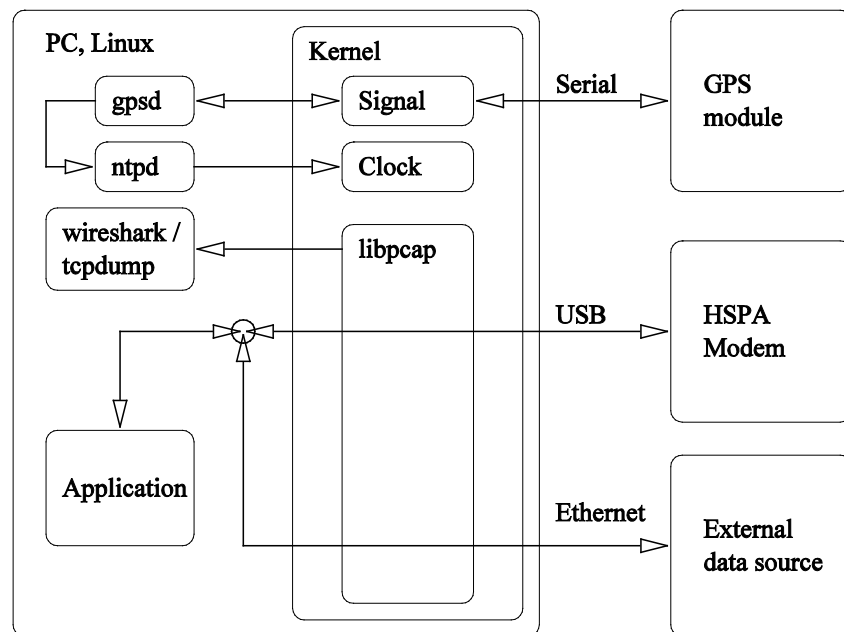


Figure 9: Detailed Test Setup

### 2.7. Network Delay Model (updated)

According to measurements described in the Annexes to this document, A.1 and A.2, the network delay model for an HSPA network may be extracted.

For the backbone, the average two-way delays are those recorded at the firewall. Server processing delay needs to be subtracted for the 1<sup>st</sup> TCP packet (see Annex A.2). Then, the statistics is as follows:

**Table 2: Two-way delay for the backbone**

Backbone delay	1st TCP packet (without average server processing delay)	2nd TCP packet
test1	0.0270	0.0238
test2	0.0267	0.0224
test3	0.0252	0.0232
test4	0.0270	0.0228
test5	0.0257	0.0250
test6	0.0267	0.0239
test7	0.0277	0.0234
test8	0.0279	0.0236
test9*	0.0355	0.0197
test10	-	-

\*For phone test9, the instance of 15s RTT for the 2nd packet due to several retransmissions is excluded from statistics.

From Table 2 we may deduce that the average delay for the 1<sup>st</sup> TCP packet is around 26.7ms, and for the second around 23.5ms. The statistics for phone test9 should be taken with precaution, since for this phone, due to the sporadic traffic pattern, we have much less packets recorded than for other phones, i.e. the sample is not large. The missing value for phone test10 is due to rebooting of the phone after the firewall tracing has started (filtering by its initial IP address was applied, see Annex A.2).

For the core network (GGSN, proxy if used, firewall), the average two-way delay may be calculated by subtracting RTTs recorded at the firewall from those recorded at the Gn interface. Here, we have an extremely small statistical sample at the Gn interface, especially for phones test9 and test10 with sporadic patterns, and instances of retransmissions caught at the firewall (long trace), but not at the Gn (3-5 minutes trace), which may lead to negative values. Missing values for phone test10 are due to missing firewall RTT values explained above, and for phones test2 and test3 due to unsynchronized Gn trace recording at two branches (described in Annex A.2).

Direct Tunnel functionality was deployed in the network, so the data in the user plane logically went directly from the RNC to the GGSN.

**Table 3: Two-way delay for the core network**

Core delay	1st TCP packet	2nd TCP packet
test1	0.0048	0.0031
test2	-	-
test3	-	-
test4	0.0087	0.0063
test5	-0.0045	0.0047

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

test6	0.0117	0.0037
test7	0.0146	0.0033
test8	0.0182	0.0091
test9*	-0.0175	-0.0014
test10	-	-

\*For phone test9, the instance of 15s FW RTT for the 2nd packet due to several retransmissions is excluded from statistics, as well as the value of 3.3s Gn RTT for the 1st packet.

From the Table 3 we may deduce that the latency in the core is around several milliseconds, and larger values may be attributed to retransmissions and small sample at the Gn interface.

For the access network, including the transport path between the RNC and GGSN, which is short in our case (RNC, SGSN and GGSN being physically close) and may be neglected, the average two-way delays are calculated for the TCP case by subtracting average RTTs recorded on the Gn interface from those recorded on phones. Again, the small sample at the Gn interface has influence, but not as much as for the statistics in Table 2, as the delays in the access network are much longer than in the core. Missing values for phones test2 and test3 are due to unsynchronized Gn trace recording at two branches (described in Annex A.2).

For UDP, for comparison reasons, since there was no tracing at the Gn interface not at the firewall, the access network statistics is shown by subtracting average core two-way delay in Table 2 and average backbone two-way delay in Table 2 (server processing delay is mostly less than 2ms) from the average RTTs recorded on phones.

The statistics is as follows:

**Table 4: Two-way delay for the access network**

<b>Access delay</b>	<b>1st TCP packet</b>	<b>2nd TCP packet (main payload)</b>	<b>UDP packet (without average core RTT 0.005s and average backbone RTT 0.025s)</b>
test1	0.1168	0.1220	0.0880
test2	-	-	0.0913
test3	-	-	0.1014
test4	0.0743	0.1009	0.1090
test5	0.1040	0.2011	0.2266
test6	0.0693	0.1944	0.2498
test7	0.0746	0.2020	0.2067
test8	0.0966	0.1986	0.3183
test9*	1.3145	0.4212	1.6865
test10	1.3349	0.3333	1.5718

\*For phone test9, the instance of 3.3s Gn RTT for the 1st packet due to retransmission was excluded from statistics.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

From the Table 4 we may deduce that the access network shows strong selectivity concerning latency towards specific traffic patterns. 1<sup>st</sup> TCP packet is around 70 B long, and the second carries the main payload of nominal length, according to modelled statistical distribution of packet size. 1<sup>st</sup> TCP packet bears the impact of accessing the network for long inter-arrival times (phones test9 and test10) i.e. phones that are often switched to the Idle state, and the impact of assigned traffic channel in UTRAN. 2<sup>nd</sup> TCP packet bears the impact of assigned traffic channel and the impact of packet length, as it carries the main payload. The UDP packet suffers from all impacts – the assigned traffic channel, going from Idle to RRC Connected state for phones test9 and test10, and packet length.

Firstly, the phones with large inter-arrival times (test9 and test10) have largest delays. In TCP case, 1<sup>st</sup> packet delay shows the impact of accessing the network and assigned traffic channel, while the 2<sup>nd</sup> packet statistics shows the impact of assigned traffic channel (mostly common channels due to low throughputs) and packet length. Statistics for the UDP packet shows all these impacts.

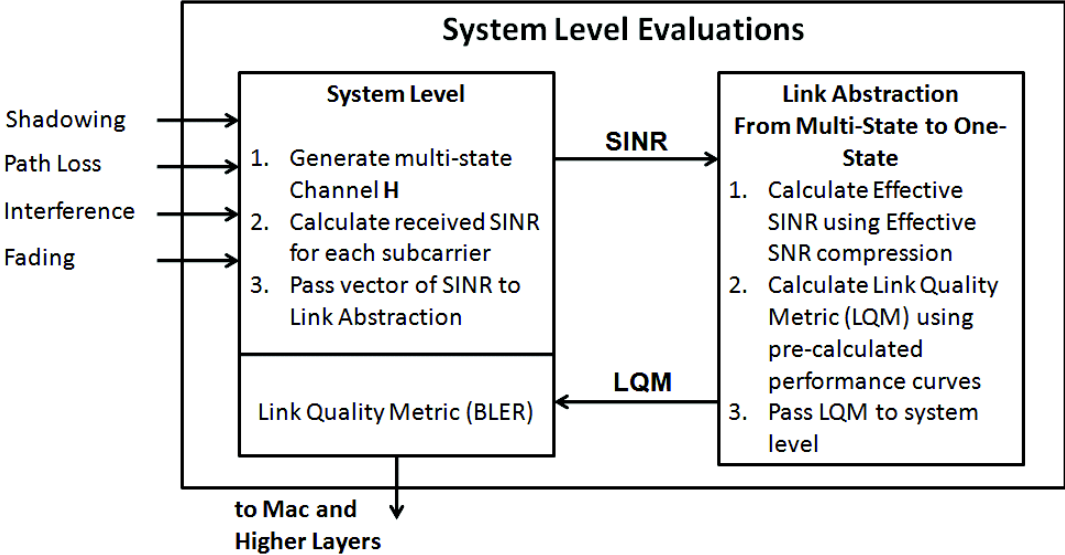
Further, statistics for the 2<sup>nd</sup> TCP packet, carrying the main payload (see Annex A.2), and for the UDP packet, shows the impact of packet length on latency – phones test5-test8 with 1 kB packets have around 100 ms longer two-way delay than packets test1-test4 with packets mostly smaller than 240 B in average. Group of phones test1-test8 is the group with rather faster traffic, and these phones never end-up in Idle state, so we may consider the influence of packet length almost independently from the influence of inter-arrival time.

## 2.8. Towards the Large Scale Experimentation

### 2.8.1. PHY abstraction

PHY abstraction, also referred to as link-to-system mapping and link prediction, provides such an interface between system level simulators and link level simulators for the large scale system simulations. This interface is normally a metric representing the quality of an instantaneous physical link (channel) between the eNodeB (LTE acronym for base station) and the connected UEs (LTE acronym for mobile station) by taking into account other important parameters of the system. These parameters as shown in Figure

10 may include the knowledge about power and resource allocation to the specific UE, number of spatial layers, modulation and coding scheme (MCS), and mainly channel characteristics, i.e., path loss, shadowing, fading, interference etc.

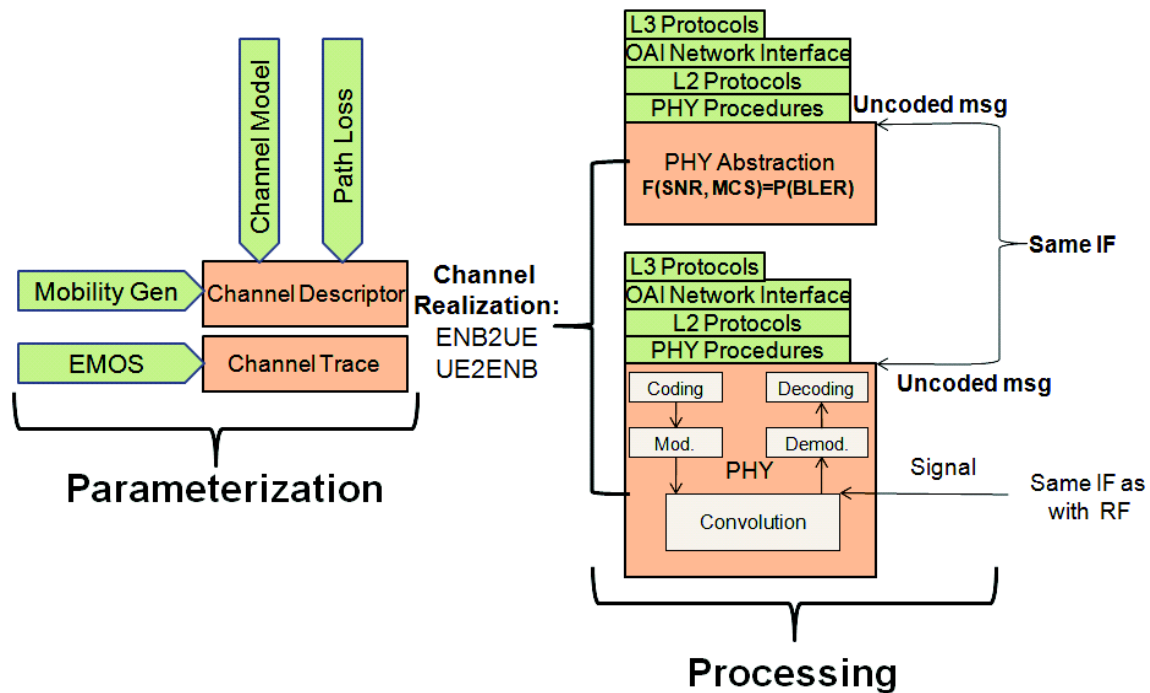


**Figure 10: Link Abstraction in System Performance Evaluation**

PHY abstraction is rather trivial for the frequency flat channels as the simple averaging of channel qualities is sufficient for link quality mapping but for highly frequency selective channels the performance evaluation is not that trivial. This is mainly because of the smaller coherence bandwidth than that of the signal bandwidth giving rise to the multi-state channel at the receiver [4].

### 2.8.1.1. PHY Abstraction in OpenAirInterface

In OpenAirInterface the required parameters for large scale system simulations are highly dynamic and can be generated either by the specialized tools already included in the simulator, i.e., openair traffic generator and openair mobility generator, or these parameters can be specified explicitly in great details for the specific scenarios and evaluations. The use of PHY abstraction in OpenAirInterface system simulator is explained in Figure 11.



**Figure 11:PHY Abstraction in System Performance Evaluation in OpenAirInterface**

It can be seen from the Figure 11 that there are two important steps in any evaluation using OpenAirInterface, parameterization and processing. It is important to note that parameterization is independent of the knowledge about the PHY abstraction. The output (channel realizations) from parameterization step is given to the processing where the comparison between using the full PHY and PHY abstraction is shown. It can be seen that in the case of PHY abstraction there is no coding, decoding or other complex operations involved from the transceiver chain at the physical layer (L1) only. The main purpose of the physical layer is to inform the higher layers about the status of the decodability of data packet. If the decoding is successful then the higher layers are notified about it. However in the case of using PHY abstraction this is achieved by predicting a link quality metric in terms of block error probability from the instantaneous channel realizations across all of the subcarriers. After the BLER is calculated using PHY abstraction, a random number between 0 and 1 is generated which is compared with this BLER for taking the decision on the successful or unsuccessful transmission. Then the outcome of this probabilistic experiment is passed to the higher layers which perform their tasks independent of the knowledge about the PHY abstraction.

### 2.8.1.2. Abstraction Methods

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

The most important step in PHY abstraction is to calculate the effective SINR in a way that it is able to transform the multi-state channel in to a single state channel. For this purpose two most studied link abstraction methodologies are the expected effective SINR mapping (EESM) and mutual-information based effective SINR Mapping (MIESM). In both of the two methods the basic scheme is effective SINR mapping which at first maps the varying SINRs of a codeword to an effective SINR ( $\gamma_{\text{eff}}$ ) value which is then used to read the equivalent BLER from the AWGN performance curves of a particular modulation and code scheme (MCS).

$$\gamma_{\text{eff}}(\beta_1, \beta_2) = \beta_1 I^{-1} \left[ \frac{1}{N} \sum_{n=1}^N I \left( \frac{\gamma_n}{\beta_2} \right) \right] \quad (1)$$

Where  $N$  is the number of channel symbols in a codeword and  $I(\gamma_n)$  is a mapping function which transforms SINR of each channel symbol to some "information measure" where it is linearly averaged over the codeword. Then these averaged values are transformed back to SNR domain.  $\beta_1$  and  $\beta_2$  are called calibration factors and they are there to compensate for different modulation orders and code rates.

For the EESM the mapping function  $I(\gamma_n)$  is calculated using Chernoff Union bound of error probabilities, i.e.,

$$I(\gamma_n) = 1 - \exp(-\gamma_n) \quad (2)$$

$$\gamma_{\text{eff}}(\beta_1, \beta_2) = -\beta_1 \ln \left[ \frac{1}{N} \sum_{n=1}^N \exp \left( -\frac{\gamma_n}{\beta_2} \right) \right] \quad (3)$$

For the mutual information based methods the approximations of mapping function and the reverse mapping functions come from the mutual information for discrete QAM constellation, i.e.

$$I_{M_1}(\gamma_j) = \log M_1 - \frac{1}{M_1} \sum_{x_1 \in \chi_1} \mathcal{E}_{z_1} \log \frac{\sum_{x_1' \in \chi_1} \exp \left[ -|\gamma_j(x_1 - x_1') + z_1|^2 \right]}{\exp[-|z_1|^2]} \quad (4)$$

where  $\chi_1$  is the set of the QAM constellation points with  $|\chi_1| = M_1$  and  $z_1 \in \mathcal{CN}(0,1)$ .

For the PHY abstraction  $\gamma_{\text{eff}}(\beta_1, \beta_2)$  can be calculated using (4), the respective information function for EESM and MIESM, and the received SINR expressions from (1), (2) and (3) for each of the LTE transmission modes. This  $\gamma_{\text{eff}}$  then can be used to obtain the BLER from the previously calculated AWGN performance curves corresponding to the specific MCS, i.e.,

$$\text{BLER}(\gamma, \text{MCS}) \simeq \text{BLER}_A(\gamma_{\text{eff}}(\beta_1, \beta_2), \text{MCS}) \quad (5)$$

Where  $\gamma$  represents the  $N \times 1$  vector of  $\gamma_n$  and  $\text{BLER}_A$  represents the AWGN block error rate obtained for specific MCS. If these values are not calibrated or wrongly calibrated then PHY

abstraction becomes unreliable as using false  $\gamma_{\text{eff}}$  for obtaining the BLER can result in either overestimation or underestimation of error probability. Therefore the adjustment factors are to be calculated very carefully. The selection of specific MCS depends on the quality of instantaneous channel. The eNodeB obtains the information about the quality of instantaneous channel from the uplink control information using the Channel Quality Indicator (CQI).

### 2.8.1.3. Link Level Calibration

In order to train and test the PHY abstraction for system level evaluations, first it had to be validated through link level simulator, therefore, we used Eurecom's OpenAirInterface link level simulator and used both ESM methods for the calculation of  $\gamma_{\text{eff}}(\beta_1, \beta_2)$ . For this paper we used ideal channel estimation with 8-tap Rayleigh channel model with the delay spread of 1e-6 seconds. Further parameters for the link level simulator are given in Table 5.

**Table 5: Simulation Parameters for Link Level Simulator**

Transmission mode	1, 2, 6
Transmission bandwidth	5 MHz
FTT size	512
Subcarrier spacing	15 KHz
Useful subcarriers	300
Subframe length	1 ms
Cyclic shift	Normal
Physical resource blocks	25
channel	8-tap Rayleigh Channel Model
Delay spread	1e-6 second
Channel estimation	Ideal
Decoder	Max-log Map
MCS	0 – 22

For each of the transmission mode and each of the MCS, we performed link level simulations for a large number of different channel realizations. We kept the channel constant during each of the channel realization and simulated the system for 10000 packets or 5000 erroneous packets with random AWGN noise. From these simulations we saved the  $\text{BLER}_{m, \text{MCS}}$  and other required parameters necessary for the link abstraction. The next important step is to calibrate the adjustment factors. The calibration of these factors should be performed with such a channel model which can provide it with high frequency selectivity that is why we chose

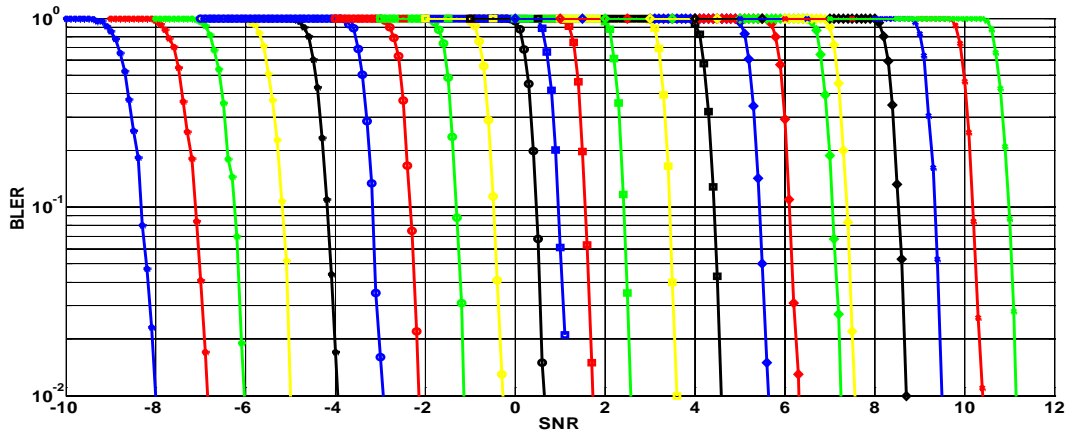
LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Rayleigh Channel model and then we performed this step over large number of channel and noise realization to find adjustment factors such that

$$(\beta_1, \beta_2) = (\beta_1, \beta_2) \argmin [\text{MSE}] \quad (6)$$

$$\text{MSE} = \sum_{i=1}^{N_{\text{ch}}} |\text{BLER}_A(\gamma_{\text{eff}}(\beta_1, \beta_2), \text{MCS}) - \text{BLER}_{m, \text{MCS}}|^2 \quad (7)$$

where MSE is the mean squared error,  $N_{\text{ch}}$  is the number of different channel realizations,  $\text{BLER}_A(\gamma_{\text{eff}}(\beta_1, \beta_2), \text{MCS})$  is the predicted block error rate from the respective AWGN curve and from the  $\gamma_{\text{eff}}(\beta_1, \beta_2)$  calculated using (4) for both of the ESM methods, and  $\text{BLER}_{m, \text{MCS}}$  is the error rate from the  $N_{\text{ch}}$  channel realizations. To obtain the  $\text{BLER}_{\text{testA}}(\gamma_{\text{eff}}(\beta_1, \beta_2), \text{MCS})$  we performed AWGN link level simulations for all MCS of LTE and stored these AWGN SNR-BLER performance curves to be used for the prediction of BLER during the optimization of adjustment factors  $\beta_1$  and  $\beta_2$ . These are shown in Figure 12.



**Figure 12: AWGN Link Performance Curves in LTE with 5 MHz Bandwidth for MCS 0 - 22**

#### 2.8.1.4. Results

After calibration we applied both of the ESM PHY abstraction techniques on the saved outputs of the link level simulations. In the following we provide the tables where we present the calibration factors with minimum MSE for both of the ESM PHY abstraction techniques and it can be seen that the MSE is very low for both techniques. Table 6, Table 7 and Table 8, present  $\beta_1$ ,  $\beta_2$  and MSE for both EESM and MIESM for MCS 0-22 for transmission mode 1, 2 and 6 respectively. It can be seen that the optimal calibration factors for the MIESM and EESM are very much different than the unity. Therefore even for the case of MIESM, calibration is required to reach a good level of accuracy. Further these results show that if both of the ESM methods are calibrated correctly then the performance of both ESM methods becomes almost the same. Therefore one can use any one of the two techniques in the system level simulator with proper calibration.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

**Table 6 LTE Transmission Mode 1 - Calibration Factors and Mean Squared Error (MSE) Values for EESM and MIESM abstraction**

MCS	EESM			MIESM		
	$\beta_1$	$\beta_2$	MSE	$\beta_1$	$\beta_2$	MSE
0	2.51867	2.49808	0.00391	1.11876	1.12934	0.00498
1	0.48797	0.48677	0.01437	0.37201	0.37072	0.01334
2	0.51811	0.51144	0.00526	0.37550	0.37219	0.00509
3	1.15845	1.14284	0.00891	0.94532	0.93101	0.00862
4	0.79600	0.79522	0.00578	0.57989	0.58077	0.00568
5	0.77935	0.77683	0.00493	0.52590	0.52793	0.00475
6	0.80905	0.79431	0.00708	0.54849	0.54037	0.00685
7	0.80876	0.79535	0.00842	0.53399	0.52787	0.00750
8	0.80563	0.81064	0.00919	0.51646	0.52099	0.00933
9	0.84083	0.82789	0.00576	0.52998	0.52262	0.00483
10	1.91969	1.85813	0.01039	0.64432	0.63280	0.01114
11	1.95414	1.90448	0.00708	0.59618	0.58269	0.00781
12	2.35630	2.26533	0.01159	0.48836	0.48043	0.00911
13	2.48046	2.41211	0.01111	0.49270	0.48302	0.01071
14	2.41703	2.37478	0.01093	0.36865	0.36376	0.00720
15	2.99274	2.92132	0.01046	0.46138	0.45064	0.00684
16	2.84205	2.87155	0.02203	0.40809	0.41069	0.03005
17	5.27602	5.02621	0.03266	0.26393	0.25466	0.01751
18	5.75658	5.42143	0.03246	0.29350	0.27973	0.01616
19	6.48658	6.08509	0.03809	0.25203	0.24131	0.03036
20	7.79768	7.18738	0.02389	0.37087	0.33508	0.02685
21	7.78316	7.46862	0.06827	0.29065	0.27502	0.07025
22	7.49917	7.51829	0.07799	0.26116	0.25681	0.11458

**Table 7 LTE Transmission Mode 2 - Calibration Factors and Mean Squared Error (MSE) Values for EESM and MIESM abstraction**

MCS	EESM			MIESM		
	$\beta_1$	$\beta_2$	MSE	$\beta_1$	$\beta_2$	MSE
0	0.51505	0.51133	0.00469	0.71378	0.70281	0.00623
2	0.72291	0.71876	0.00205	0.55335	0.55028	0.00204
3	0.69201	0.68623	0.00354	0.48285	0.48147	0.00344
4	0.72305	0.71997	0.00282	0.49984	0.50057	0.00250
5	0.69175	0.68011	0.00263	0.49064	0.48318	0.00243
6	0.74296	0.71927	0.00433	0.49418	0.48104	0.00383
7	0.79333	0.76209	0.00280	0.53226	0.51240	0.00305
8	0.79476	0.76746	0.00326	0.51416	0.49643	0.00258
9	0.77145	0.72608	0.00322	0.47562	0.44854	0.00286
10	1.79586	1.69821	0.00606	0.58539	0.55756	0.00413
11	1.63912	1.57082	0.00369	0.41666	0.39708	0.00408
12	1.90221	1.82649	0.00694	0.44454	0.42387	0.00691
13	2.31685	2.17287	0.00871	0.39421	0.37011	0.00785
14	2.66179	2.44757	0.00491	0.37893	0.34935	0.00330
15	2.29127	2.17064	0.02918	0.27565	0.26303	0.03556

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

16	3.11592	2.86310	0.00963	0.44559	0.40529	0.00740
17	4.46069	4.08881	0.01218	0.16501	0.15051	0.01651
18	5.21514	4.62933	0.02739	0.22538	0.20013	0.04562
19	5.96413	5.21403	0.02564	0.23530	0.20244	0.04945
20	6.46220	5.54003	0.07693	0.20975	0.17879	0.07800
21	7.60083	6.46778	0.06793	0.25174	0.21238	0.07145
22	9.87314	8.17512	0.04159	0.32418	0.26681	0.04350

**Table 8 LTE Transmission Mode 6 - Calibration Factors and Mean Squared Error (MSE) Values for EESM and MIESM abstraction**

MCS	EESM			MIESM		
	$\beta_1$	$\beta_2$	MSE	$\beta_1$	$\beta_2$	MSE
0	0.43114	0.42671	0.00391	0.71458	0.69798	0.00669
1	0.57761	0.56792	0.00349	0.47890	0.46943	0.00347
2	0.69824	0.69631	0.00221	0.53920	0.53792	0.00227
3	0.70350	0.69631	0.00364	0.55563	0.54833	0.00386
4	0.72077	0.71818	0.00285	0.49747	0.49825	0.00253
5	0.71127	0.69330	0.00147	0.48994	0.47943	0.00173
6	0.70513	0.68993	0.00390	0.46907	0.46111	0.00394
7	0.80003	0.76490	0.00381	0.52612	0.50511	0.00321
8	0.81582	0.77642	0.00311	0.52780	0.50203	0.00284
9	0.78684	0.74239	0.00354	0.49624	0.46719	0.00402
10	1.81199	1.73499	0.00551	0.45132	0.44134	0.01067
11	1.80174	1.72851	0.00372	0.41129	0.39574	0.00192
12	2.34708	2.19196	0.00706	0.46224	0.43891	0.00359
13	2.59607	2.41612	0.00578	0.46088	0.43443	0.00632
14	2.57389	2.38144	0.00385	0.39457	0.36445	0.00351
15	2.81933	2.59327	0.00866	0.39905	0.36611	0.00594
16	2.70396	2.53965	0.02087	0.35318	0.33006	0.02625
17	4.93805	4.43685	0.01588	0.24389	0.21788	0.01424
18	4.84900	4.46593	0.01656	0.18978	0.17297	0.01656
19	6.80857	5.92575	0.01480	0.29295	0.25357	0.01200
20	7.04470	6.27926	0.04026	0.27118	0.23895	0.05122
21	9.33852	8.03885	0.01717	0.33666	0.28843	0.02017
22	9.96321	8.28751	0.06238	0.35912	0.29355	0.05003

#### 2.8.1.5. System Level Validation

Link level results show that our approach for the ESM PHY abstraction is very much accurate and any of the two methods can be used in system level simulators. Therefore we decided to implement EESM in the OAI system level simulator for large scale evaluations. This system level simulator implements the full protocol stack for different transmission modes of LTE. We wanted to show that how the link abstraction can provide us with 1) low complexity and speed 2) scalability 3) applicability and most importantly 4) accuracy. To show all these we performed system level simulations for different transmission modes both with full PHY and PHY

abstraction. The underlying scenario consists of a system with one eNodeB and two UEs. We specified the system scenarios through parameters file and ran the simulator for 500 frames. The important point for the system level simulations are,

Abstraction is only implemented for the downlink shared channel (DLSCH) and there is no abstraction for uplink (UL) and Control channels. Scheduler gives most of the resources to the user with better feedback CQI. Full Buffer traffic is produced and the eNodeB can select between MCS 0-22 for the downlink (DL) communication. During the simulations we calculated both the accumulated averaged throughput of system over given number of frames and also the execution time for the simulation. To show that using PHY abstraction is less complex and it speeds up the evaluation process we stored the execution time for system simulations of same scenarios with full PHY and PHY abstraction. We stored these times under Linux operating systems when there was no other application running but the simulation only. We found out that simulations with abstraction took extremely short time than that of with full PHY. The calculated speedup factor for PHY abstraction was found to be around 30 when compared to the time for full PHY. Table 9 shows the execution time for the simulation and it is clear from the results that PHY abstraction speeds up the process very drastically.

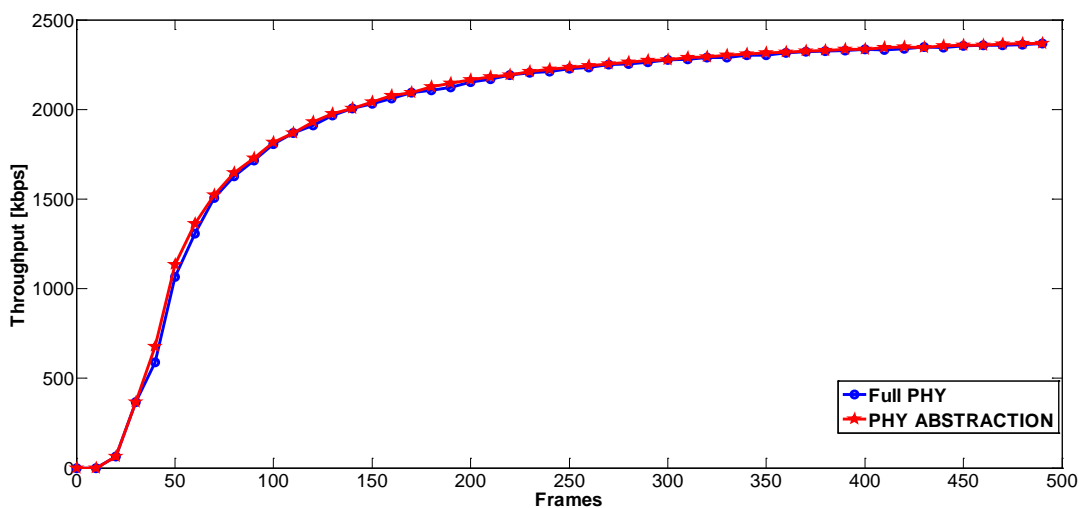
**Table 9: Simulation times different transmission Modes**

		Time in minutes and seconds	
		Full PHY	PHY Abstraction
TM 1	Total time	2m26.602s	0m6.794s
	user CPU time	2m25.633s	0m6.480s
	system CPU time	0m0.924s	0m0.328s
TM 2	Total time	4m1.607s	0m9.085s
	user CPU time	3m59.079s	0m8.753s
	system CPU time	0m1.940s	0m0.364s
TM 6	Total time	2m19.320s	0m7.027s
	user CPU time	2m18.473s	0m6.752s
	system CPU time	0m0.824s	0m0.300s

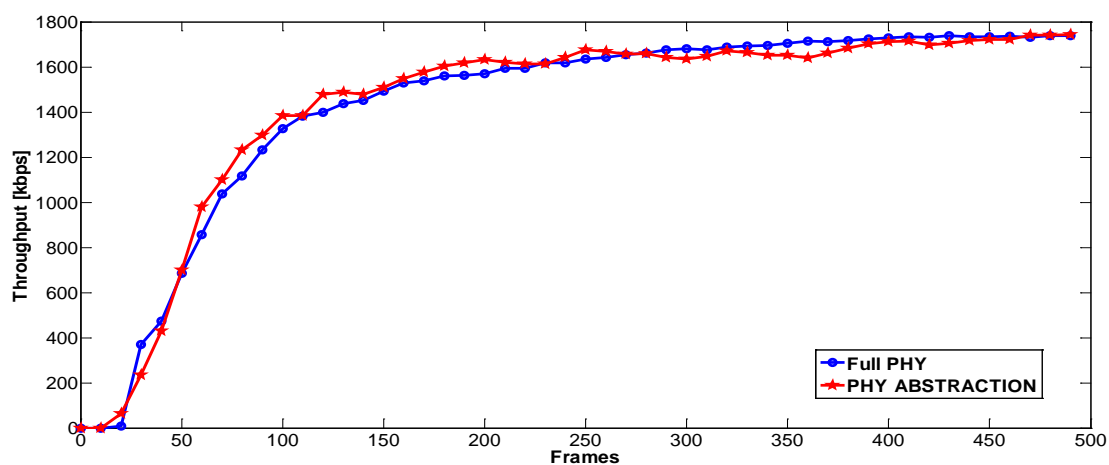
The next important thing to demonstrate is the realism of abstraction in system level evaluations. By realism we mean that the simulations with PHY abstraction should produce the results similar to the simulations with full PHY. This is shown by plotting the accumulated average throughput of the system over a given number of frames in Figure 13 - Figure 15 for transmission mode 1, 2 and 6 respectively. It is very clear that performance of both full PHY and PHY abstraction is very much close to each other and provide the same system

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

throughput. Another important aspect to note is that although we calibrated the adjustment factors with Rayleigh channel model but to show the applicability of PHY abstraction in diverse channel models we used different channel models for the simulations of these transmission modes. For example simulation for the TM 1 was performed with 8-tap Ricean channel, simulation for TM 2 with 8-tap Rayleigh channel and simulation for TM 6 with single tap Ricean channel. It is clear that the calibrated factors for Rayleigh channel are also applicable to other channel models thus giving rise to its applicability. In the end we shall like to discuss that although we performed these simulations with small number of users but still it shows the significant advantages of using PHY abstraction over full PHY. Further it can be straight forwardly inferred that in the case of more UEs in the system, the gains achieved from PHY abstraction will be even significant while maintaining the realism of evaluations.

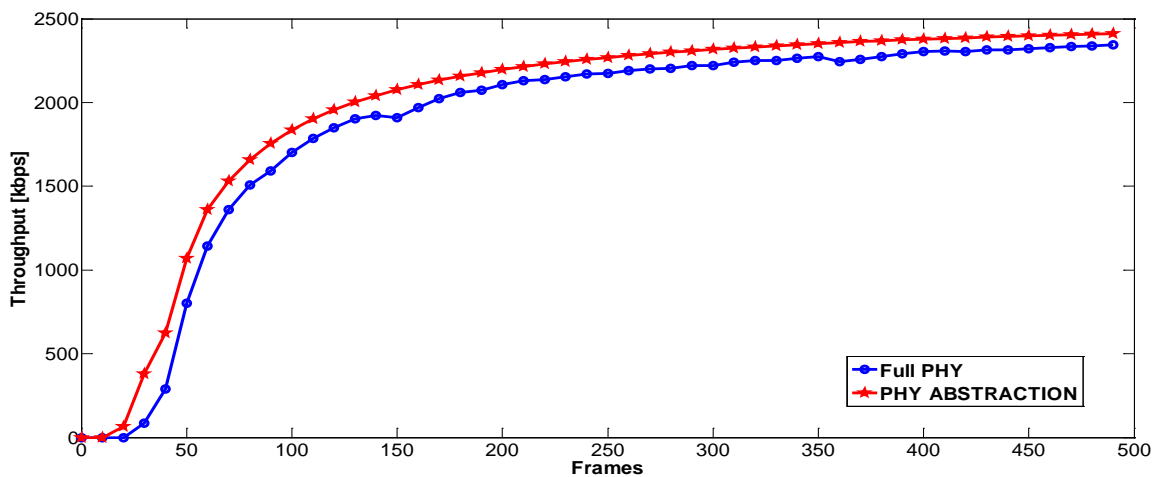


**Figure 13: LTE Transmission Mode 1 - Accumulated average system throughput over given number of frames**



**Figure 14: LTE Transmission Mode 2 - Accumulated average system throughput over given number of frames**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--



**Figure 15: LTE Transmission Mode 6 - Accumulated average system throughput over given number of frames**

### 2.8.2. Parallelism Methodology

The previous profiling work described in Deliverable D5.2, showed that the sequential execution of the emulator seriously limits the expected scalability and efficiency, and thus limit the scalability of the OAI in-lab system validation platform. Therefore, in the first attempt, we parallelize its routines according to a logical separation which addresses four independent entities: emulation, eNB, UE and channel (section 3.4.1 of D5.2). While such architecture provides a reasonable gain compared to the sequential one, additional profiling work demonstrated that the hardware usage rate and the efficiency remains under the expectations. In fact, emulated nodes may present significant load imbalance which is directly reflected on the global performance of threads representing the nodes. Therefore, we rather consider smaller entities that we call "job" which represent different actions that must be performed by each emulated node. The following four job types are available for a given node:

1. **JOB1:** Application traffic generator
2. **JOB2:** PDCP operation
3. **JOB3:** PHY/MAC procedures
4. **JOB4:** Channel realization

The emulator will rely on a pool of worker threads to execute the totality or a part of those jobs, which are controlled and synchronized by a master thread. By breaking down a node operations into different jobs and attribute jobs to different worker threads, significant improvement and the load balancing among the worker threads can be achieved. This is

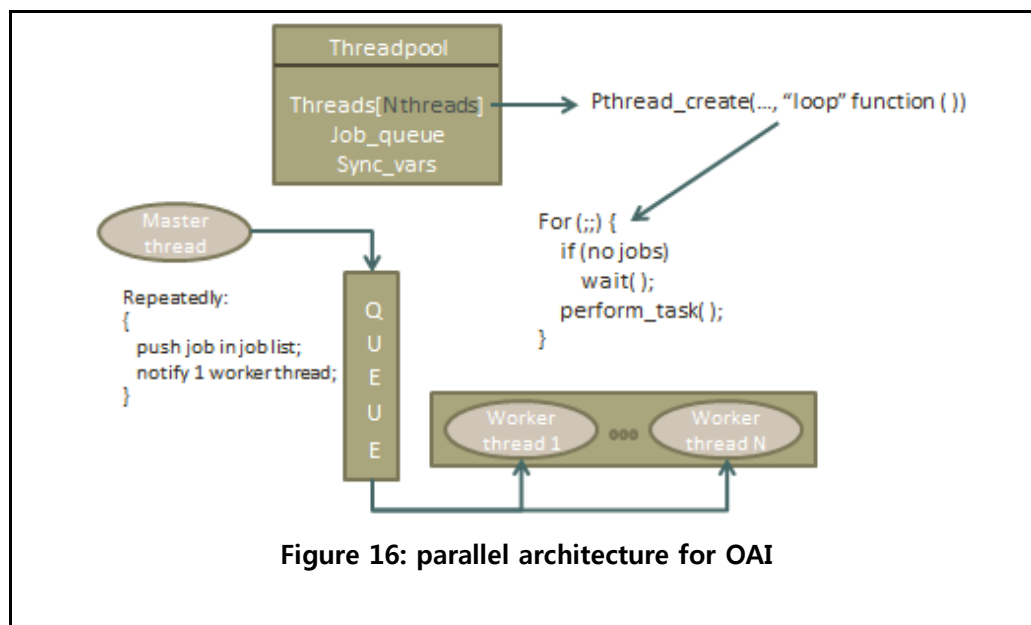
LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

because nodes are not directly associated with a given worker thread, and that if a node is overloaded, its load will be distributed to multiple worker threads.

The architecture is based on a master-worker model as shown in Figure 16 [8], where the workers are continuously waiting for jobs to perform and the master is executing the simulation until reaching a block of instructions which is going to be parallelized. Such a block must have the following for loop form:

```
for (UE_id=0;UE_id<NB_UE_INST;UE_id++) { example of a block to be split into jobs
    do_DL_sig(r_re0, r_im0, r_re, r_im, s_re, s_im, eNB2UE, enb_data, ue_data, next_slot,
    abstraction_flag, frame_parms,UE_id); //Realization of downlink channel
}
```

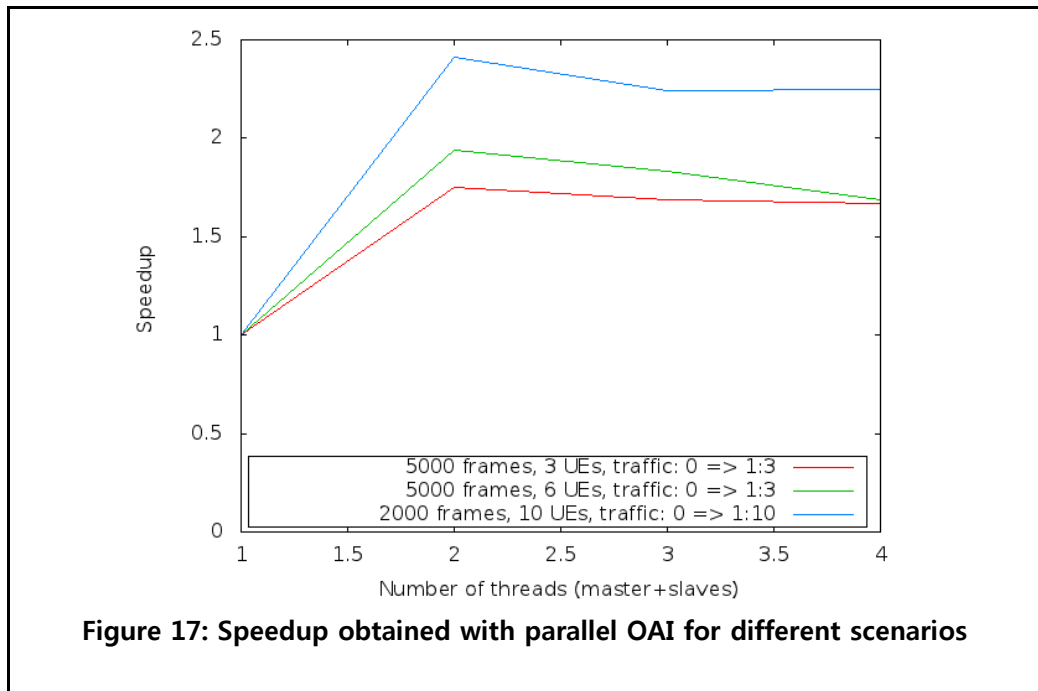
An important requirement is that the parallel execution of the loop body must not affect the correctness of the simulation, for instance, it must be free of race conditions. At this point, the master creates as many jobs as node multiplicity ( $NB\_UE\_INST$  in the above example) and pushes them in the job queue. Each new job being pushed in the queue automatically activates a waiting worker to execute the job. The number of workers is fixed when the thread pool is created. The optimal value for it depends on the average number of jobs that can be executed in parallel on the one hand and the available computing resources on the other hand. Increasing this number does not always imply a faster execution. For example, if for some reason a particular periodic task (the effective example is *do\_UL\_sig*, the function which performs the uplink channel realization) is heavy and could not be parallelized, this task becomes the system bottleneck by implying a long idle time. In such cases, one worker is enough to reach the highest efficiency (see Figure 17), but for a drastic performance improvement, those constraints should be removed.



More generally, to fully leverage the master-worker architecture, the emulator and the protocol stack has to be analyzed to reveal any race conditions.

To assess the current version, we found that the realization of the downlink channel (performed by the function *do\_DL\_sig*) task is a good candidate to apply our method on, since it represents an important part of the computing overhead (around 50% according to the last profiling) and is the safest task as far as parallel execution is concerned. Representing this task through multiple parallel jobs, we achieved a noticeable speedup which varies between *1.67* and *2.41*.

In the 3 tested scenarios, we gradually increased the node number and traffic load while varying the number of thread workers. We notice that the speedup increases with the number of nodes and traffic load. However, increasing the number of slaves (workers) does not produce the expected result even with a high number of nodes. As explained above, *do\_UL\_sig* task becomes the system's bottleneck since it is heavy, sequential and periodic (about 5 slots per frame). A local speedup is realized each time that a *do\_DL\_sig*-job series is executed in parallel. However, this speedup is cancelled by the long idle time during which the master executes a *do\_UL\_sig* task and the job queue is empty. Furthermore, adding new threads brings more overhead due to their management and synchronization, which worsens the global performance. This is why we obtain the most efficient result with 2 threads (one master and one worker).



Thus, the proposed parallelism model scales with the number of nodes and could allow us to experiment up to several tens of nodes while maintaining the real-time operation.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

### 2.8.2.1. Discrete Event Generator

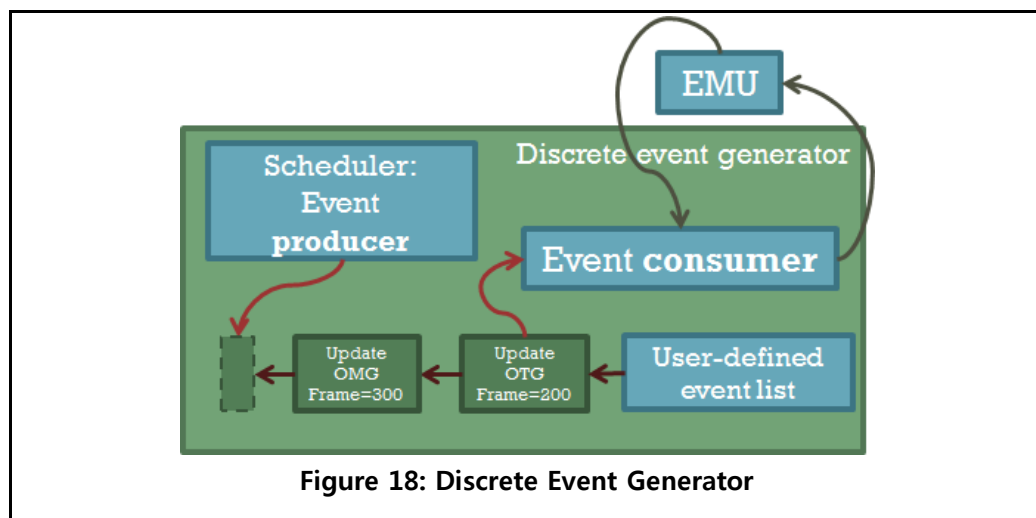
The discrete event generator (DEG) enables more customized but high-level simulation control and therefore enhances the scalability potential of the emulator. In fact, in very large scale scenarios, it is even more important to dispose of more control and monitoring possibilities in order to experiment special cases. These new possibilities are introduced in the form of user-defined events and the reconfigurability when targeting large-scale experimentation. The DEG consists of a Scheduler, an event consumer and an Event List for user-defined events as shown in Figure 18. Below, a simplified example of main function is shown, using DEG.

```
int main (int argc, char *argv[]) {
    Mobility *mobility_frame_10;
    Application_Config *application_frame_20;
    //Here make modifications on the mobility and traffic new models
    //mob_frame_10 -> ...
    //application_frame_30 -> ...
    schedule(ET_OMG, 10, NULL, mobility_frame_10); //schedule OMG model update for frame=10
    schedule(ET_OTG, 15, NULL, application_frame_20);
    schedule_end_of_simulation(FRAME, 100);

    run(argc, argv);
    return 0; //launch simulation
}
```

The simulation events can be divided into two categories:

- Regular events: periodic events which periodicity is always the same. For example: (i) time advancement or (ii) the event which triggers OTG (OpenAir Traffic Generator) state machine. These events are meant to ensure simulation progression and must not interfere with the simulation logic. Example (ii) does not affect OTG model nor triggers a packet generation. All what it does is to put into action OTG state machine.
- User-defined events: Any other events such as a modification in OTG or OMG model. These events are stored in a time-ordered event list.



At each frame, the EMU process calls the event consumer, which extracts the events related to current frame from the event list, checks their content and executes the corresponding code.

### 3. VALIDATION RESULTS

#### 3.1. CBA implementation validation

In this section, we demonstrate that modifications, which is required to enable CBA, to the LTE protocol stacks is implemented in our OpenAirInterface platform (<http://www.openairinterface.org/>).

##### 3.1.1. RRC layer

RRC layer is used to configure lower protocol layers including: PDCP, RLC, MAC and PHY layer. As mentioned in the last section, to enable CBA in LTE, the RRC layer at the eNB side should configure the CBA dedicated RNTIs for each UE such that a UE can use these RNTIs to decode the resource allocated for CBA transmission.

Firstly, we set the number of CBA group to 1 and number of UE to 6. Figure 19 shows that: (1) the eNB initializes four CBA RNTIs for possible use and the number of active CBA group is 1; (2) the eNB allocates the RNTI (fff4) to the six UEs in one group. Figure 20 shows after the RRC connection setup, every UE receives a CBA dedicated RNTI. It can be seen that the eNB correctly configure CBA dedicated RNTI for the five UEs in a CBA group and each UE in the CBA group successfully receives the configured CBA dedicated RNTI.

```
[RRC][D][eNB 0] Initialization of 4 cba_RNTI values (fff4 fff5 fff6 fff7) num active groups 1
[RRC][D][eNB 0] Frame 21: cba_RNTI = fff4 in group 0 is attributed to UE 0
[RRC][D][eNB 0] Frame 38: cba_RNTI = fff4 in group 0 is attributed to UE 1
[RRC][D][eNB 0] Frame 53: cba_RNTI = fff4 in group 0 is attributed to UE 2
[RRC][D][eNB 0] Frame 79: cba_RNTI = fff4 in group 0 is attributed to UE 3
[RRC][D][eNB 0] Frame 95: cba_RNTI = fff4 in group 0 is attributed to UE 4
[RRC][D][eNB 0] Frame 118: cba_RNTI = fff4 in group 0 is attributed to UE 5
```

**Figure 19: RRC layer signalling to initialize and to configure RNTIs for CBA (eNB side)**

```
[RRC][D][UE 0] Frame 21: radioResourceConfigDedicated received CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 1] Frame 38: radioResourceConfigDedicated received CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 2] Frame 53: radioResourceConfigDedicated received CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 3] Frame 79: radioResourceConfigDedicated received CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 4] Frame 95: radioResourceConfigDedicated received CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 5] Frame 118: radioResourceConfigDedicated received CBA_RNTI = fff4 for group 0 from eNB 0
```

**Figure 20: RRC layer signalling to acknowledge the received RNTI for CBA (UE side)**

To improve the resource efficiency and provide better QoS, there can be multiple CBA groups in a cell, which is also implemented in our platform. There are various UE grouping methods: grouping based on UE's location, grouping based on UE's QoS requirement, grouping based on UE's channel quality, etc. Here we set the number of CBA group to 2, and we use a simple grouping method: UEs with even index are grouped

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

in the first CBA group while UEs with odd index are grouped in the second CBA group. Moreover, the number of UE is set as 4.

Figure 21 shows that the eNB allocates UE 0 and UE 2 to the CBA group 0 with RNTI fff4 while UE 1 and 3 is attributed to the CBA group 1 with RNTI fff5, which verifies our method. Figure 22 demonstrates the received RNTI in the two CBA groups: the UE 0 and 2 receives RNTI fff4 for the CBA group 0 while UE 1 and 3 receives RNTI fff5 for the CBA group 1.

```
[RRC][D][eNB 0] Initialization of 4 cba_RNTI values (fff4 fff5 fff6 fff7) num active groups 2
[RRC][D][eNB 0] Frame 21: cba_RNTI = fff4 in group 0 is attributed to UE 0
[RRC][D][eNB 0] Frame 38: cba_RNTI = fff5 in group 1 is attributed to UE 1
[RRC][D][eNB 0] Frame 52: cba_RNTI = fff4 in group 0 is attributed to UE 2
[RRC][D][eNB 0] Frame 79: cba_RNTI = fff5 in group 1 is attributed to UE 3
```

**Figure 21: RRC layer signalling to initialize and to configure RNTIs for CBA with two groups (eNB side)**

```
[RRC][D][UE 0] Frame 21: radioResourceConfigDedicated reveived CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 1] Frame 38: radioResourceConfigDedicated reveived CBA_RNTI = fff5 for group 1 from eNB 0
[RRC][D][UE 2] Frame 52: radioResourceConfigDedicated reveived CBA_RNTI = fff4 for group 0 from eNB 0
[RRC][D][UE 3] Frame 79: radioResourceConfigDedicated reveived CBA_RNTI = fff5 for group 1 from eNB 0
```

**Figure 22: RRC layer signalling to acknowledge the received RNTI for CBA with two groups (UE side)**

### 3.1.2. MAC layer

The eNB uses the RRC layer signaling for CBA to configure its MAC layer such that resource can be allocated for CBA transmission. Here we set the number of CBA group to 2 and number of UE to 3. From Figure 23, we can see that the MAC layer receives the RNTIs allocated for UE 0, 1, and 2, and 4.

```
[MAC][D][eNB 0] configure CBA groups 0 with RNTI fff4 for UE 0 (total active cba groups 2)
[MAC][D][eNB 0] configure CBA groups 1 with RNTI fff5 for UE 1 (total active cba groups 2)
[MAC][D][eNB 0] configure CBA groups 0 with RNTI fff4 for UE 2 (total active cba groups 2)
[MAC][D][eNB 0] configure CBA groups 1 with RNTI fff5 for UE 3 (total active cba groups 2)
```

**Figure 23: Configure for MAC layer with CBA (eNB side)**

One of the main task for MAC layer is to perform resource allocation. Here we propose a simple resource allocation method for CBA. The general idea for this resource allocation scheme is that: the resource for CBA transmission is fixed and it is allocated between all the CBA groups; the ratio of resource allocated for one CBA group equals to the ratio of the number of active UE in this group to the number of active UE in all groups. The

pseudo code for the implementation for this resource allocation method is shown in Figure 24.

**Input:** total resource for CBA:  $N_{CBA}$

**Output:** amount of allocated resource:  $N_i$

1. For the CBA group  $i$ , find the number of active of UE in this group  $W_i$  and calculate its weight  $r_i$ : the ratio of the number of active UE in this group to the number of active UE in all groups.
2. For the CBA group  $i$ , while there is remaining resource, calculate the required resource for this group as  $N_i$ :  $N_i = r_i^* N_{CBA}$
3. Adjust the value of  $N_i$  to the nearest value which is supported in LTE resource allocate table.

**Figure 24: Resource allocation for CBA**

Here we set the CBA group to 2 and number of UE to 4. Figure 25 shows the logs when the UE executing the second step in CBA resource allocation. It can be seen that

- From frame 22 to 23, only one UE in CBA group 0 is connected to eNB, therefore the required resource block equals to total available resource block.
- From frame 29 to 32, another UE in CBA group 1 is connected to eNB, therefore the required resource block for the CBA group 0 is 11, which is the half of the available resource block.
- At frame 41, as there are one UE in each group, the required resource block for the CBA group 1 is also the half of the available resource block.
- At frame 54, one more UE of CBA group 0 connects to eNB, therefore the required resource for CBA group is 15, which is about the 2/3 of the available resource.
- At frame 55, since there are 2 UEs in CBA group 0 and 1 UE in CBA group 1, hence the required resource block for CBA group is 7, which is about the 1/3 of the available resource.
- At frame 83, another UE in CBA group 1 gets connected to eNB, therefore the number of UE in CBA group 0 and 1 are equivalent. As a result, the required resource block for CBA group 0 and 1 is 11, which is approximately half of the available resource.

```

[MAC][D][eNB 0] Frame 22, subframe 8: cba group 0 weight/granted_ues 1/1 available/required rb (23/23), num resources 1
[MAC][D][eNB 0] Frame 22, subframe 9: cba group 0 weight/granted_ues 1/1 available/required rb (23/23), num resources 1
[MAC][D][eNB 0] Frame 23, subframe 8: cba group 0 weight/granted_ues 1/1 available/required rb (23/23), num resources 1
[MAC][D][eNB 0] Frame 23, subframe 9: cba group 0 weight/granted_ues 1/1 available/required rb (23/23), num resources 1
.
.
.
[MAC][D][eNB 0] Frame 29, subframe 9: cba group 0 weight/granted_ues 1/2 available/required rb (23/11), num resources 1
[MAC][D][eNB 0] Frame 31, subframe 9: cba group 0 weight/granted_ues 1/2 available/required rb (23/11), num resources 1
[MAC][D][eNB 0] Frame 32, subframe 8: cba group 0 weight/granted_ues 1/2 available/required rb (23/11), num resources 1
[MAC][D][eNB 0] Frame 32, subframe 9: cba group 0 weight/granted_ues 1/2 available/required rb (23/11), num resources 1
.
.
.
[MAC][D][eNB 0] Frame 41, subframe 8: cba group 1 weight/granted_ues 1/2 available/required rb (23/11), num resources 1
.
.
.

```

**Figure 25: Logs for the second step operation in CBA resource allocation (eNB side)**

Figure 26 demonstrates the logs when performing the third step in CBA resource allocation. The allowed resource block size for CBA transmission is {1,2,3,4,5,6,8,9,10,12,15,16,18,20,24,25,27,30,32,36,40,45,48,50,54,60,72,75,80,81,90,96 ,100}. We can find that:

- From frame 22 to 23, the required resource block is 23, therefore 20 resource block is allocated for CBA group 0 with RNTI fff4, which yields 3 remaining resource blocks.
- From frame 29 to 32, the required resource block is 11, therefore 12 resource block is allocated for CBA group 0 with RNTI fff4, which leaves 11 resource blocks.
- At frame 41, the required resource block for CBA group 1 is 11. However, since 12 resource blocks has been allocated to CBA group 0, therefore 10 resource blocks are allocated to CBA group 1, which leaves 1 resource block.
- At frame 54, the required resource block size for CBA group 0 is 15, which is exactly one of the allowed resource block size. Therefore, 15 resource block is allocated.
- At frame 55, the required resource block size for CBA group 1 is 7. Since the remaining resource block size is 8, therefore 8 resource block is allocated.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

- At the frame 83, the eNB first allocates 12 resource block to CBA group 0 and then allocates 10 resource block to CBA group 1 though CBA group 0 and 1 both requires 11 resource blocks.

```
[MAC][D][eNB 0] Frame 22, subframe 8: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/23/20/3), rballoc 173, nCCE (21/0)
[MAC][D][eNB 0] Frame 22, subframe 9: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/23/20/3), rballoc 173, nCCE (21/0)
[MAC][D][eNB 0] Frame 23, subframe 8: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/23/20/3), rballoc 173, nCCE (21/0)
[MAC][D][eNB 0] Frame 23, subframe 9: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/23/20/3), rballoc 173, nCCE (21/0)
.
.
.
[MAC][D][eNB 0] Frame 29, subframe 9: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/11/12/11), rballoc 276, nCCE (21/0)
[MAC][D][eNB 0] Frame 31, subframe 9: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/11/12/11), rballoc 276, nCCE (21/0)
[MAC][D][eNB 0] Frame 32, subframe 8: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/11/12/11), rballoc 276, nCCE (21/0)
[MAC][D][eNB 0] Frame 32, subframe 9: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/11/12/11), rballoc 276, nCCE (21/0)
.
.
.
[MAC][D][eNB 0] Frame 41, subframe 8: CBA 1 rnti fff5, total/required/allocated/remaining rbs (23/11/10/1), rballoc 238, nCCE (17/4)
.
.
.
[MAC][D][eNB 0] Frame 54, subframe 8: CBA 0 rnti fff4, total/required/allocated/remaining rbs (23/15/15/8), rballoc 298, nCCE (21/0)
```

**Figure 26: Logs for the third step operation in CBA resource allocation (eNB side)**

The resource allocation message is sent by eNB through DCI 0. After decoding the resource allocation information with CBA dedicated RNTI, a UE has the CBA transmission opportunity. Figure 27 shows that a UE receives the CBA transmission opportunity from eNB

```
[MAC][D][UE 0] frame 69 subframe 3 CBA transmission opportunity, tbs 81
[MAC][D][UE 1] frame 73 subframe 3 CBA transmission opportunity, tbs 26
[MAC][D][UE 0] frame 79 subframe 3 CBA transmission opportunity, tbs 65
[MAC][D][UE 2] frame 80 subframe 3 CBA transmission opportunity, tbs 65
[MAC][D][UE 1] frame 83 subframe 3 CBA transmission opportunity, tbs 26
[MAC][D][UE 0] frame 89 subframe 2 CBA transmission opportunity, tbs 65
```

**Figure 27: UE gets the CBA transmission opportunity**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

### 3.1.3. PHY layer

PHY layer is configured by RRC layer with CBA dedicated RNTI. With this RNTI, a UE can decode the DCI 0 to extract the resource allocation information for CBA. Figure 28 shows that each UE receives the DCI information for CBA transmission: UE 0 and 2 receives the DCI destined for CBA group 0 with RNTI fff4 while UE 1 and 3 receives the DCI destined for CBA group with RNTI fff5.

```
[PHY][D][UE 2] frame 176, subframe 9: received DCI 0 with RNTI=fff4 (C-RNTI:367c, CBA_RNTI fff4) and format 0!
[PHY][D][UE 3] frame 176, subframe 9: received DCI 0 with RNTI=fff5 (C-RNTI:a7a4, CBA_RNTI fff5) and format 0!
[PHY][D][UE 0] frame 177, subframe 8: received DCI 0 with RNTI=fff4 (C-RNTI:bb02, CBA_RNTI fff4) and format 0!
[PHY][D][UE 1] frame 177, subframe 8: received DCI 0 with RNTI=fff5 (C-RNTI:1346, CBA_RNTI fff5) and format 0!
[PHY][D][UE 2] frame 177, subframe 8: received DCI 0 with RNTI=fff4 (C-RNTI:367c, CBA_RNTI fff4) and format 0!
[PHY][D][UE 3] frame 177, subframe 8: received DCI 0 with RNTI=fff5 (C-RNTI:a7a4, CBA_RNTI fff5) and format 0!
[PHY][D][UE 0] frame 177, subframe 9: received DCI 0 with RNTI=fff4 (C-RNTI:bb02, CBA_RNTI fff4) and format 0!
[PHY][D][UE 1] frame 177, subframe 9: received DCI 0 with RNTI=fff5 (C-RNTI:1346, CBA_RNTI fff5) and format 0!
```

**Figure 28: PHY layer receives the CBA dedicated DCI (UE side)**

After receiving the resource allocation method, a UE can send the CBA on the allocated resource. Figure 29 shows that the packet used for CBA transmission is prepared at UE 0, 1 and 2.

```
[PHY][N][UE 1] Frame 174, subframe 3: CBA data is prepared
[PHY][N][UE 0] Frame 179, subframe 2: CBA data is prepared
[PHY][N][UE 3] Frame 179, subframe 3: CBA data is prepared
[PHY][N][UE 2] Frame 180, subframe 3: CBA data is prepared
[PHY][N][UE 1] Frame 183, subframe 2: CBA data is prepared
```

**Figure 29: PHY layer prepares the data for CBA transmission (UE side).**

As the MCS for CBA transmission is not specified by eNB. Instead, a UE determine it MCS by itself. Therefore, to enable the CBA packet can be decoded by eNB, a UE should send its MCS used for CBA transmission. In addition to that, the UE's C-RNTI is also sent in uplink, whereby the eNB can allocate resource for a UE whose C-RNTI is detected. To comply with the uplink signalling used in LTE, the control information is sent in UCI, which is originally used to send CQI report. Figure 30 shows that each UE fills the UCI with its C-RNTI and MCS.

```
[PHY][I] fill uci for cba rnti 1346, MCS 2
[PHY][I] fill uci for cba rnti a7a4, MCS 2
[PHY][I] fill uci for cba rnti 1346, MCS 2
[PHY][I] fill uci for cba rnti bb02, MCS 2
[PHY][I] fill uci for cba rnti bb02, MCS 2
[PHY][I] fill uci for cba rnti bb02, MCS 2
[PHY][I] fill uci for cba rnti 367C, MCS 2
```

**Figure 30: UE fills the UCI**

At the eNB side, as a eNB has the knowledge about the resource allocated for CBA transmission, therefore for these subframes where CBA resource is allocated a eNB tries

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

to decode the packets on CBA dedicated resource blocks. Figure 31 shows that the eNB is checking if a CBA packet is sent on the dedicated resource.

```
[PHY][D][eNB 0][PUSCH 1] frame 173 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 2 with cba rnti fff4 mode PUSCH
[PHY][D][eNB 0][PUSCH 1] frame 174 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 0 with cba rnti fff4 mode PUSCH
[PHY][D][eNB 0][PUSCH 1] frame 174 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 1 with cba rnti fff5 mode PUSCH
[PHY][D][eNB 0][PUSCH 1] frame 174 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 2 with cba rnti fff4 mode PUSCH
[PHY][D][eNB 0][PUSCH 1] frame 175 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 0 with cba rnti fff4 mode PUSCH
[PHY][D][eNB 0][PUSCH 1] frame 175 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 1 with cba rnti fff5 mode PUSCH
[PHY][D][eNB 0][PUSCH 1] frame 175 subframe 3 Checking PUSCH/ULSCH CBA Reception for UE 2 with cba rnti fff4 mode PUSCH
[PHY][D][eNB 0][PUSCH 0] frame 176 subframe 2 Checking PUSCH/ULSCH CBA Reception for UE 0 with cba rnti fff4 mode PUSCH
[PHY][D][eNB 0][PUSCH 0] frame 176 subframe 2 Checking PUSCH/ULSCH CBA Reception for UE 1 with cba rnti fff5 mode PUSCH
```

**Figure 31 eNB attempts to receive CBA packets**

To decode data packet of a CBA packet, a eNB should first decode the control information for a CBA transmission. The control information includes MCS used for uplink transmission and the UE's RNTI which can be used by eNB to extract UE's identity such that dedicate resource can be allocated to it if possible. Figure 32 shows that the eNB decodes the UCI sent from UE 1 with C-RNTI 1346, UE 0 with C-RNTI bb02, UE 2 with C-RNTI 367c, and UE 3 with a7a4.

[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	1346
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	a7a4
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	1346
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	1346
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	bb02
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	bb02
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	367c
[PHY][D][eNB]	UCI	for	CBA	:	mcs	2	crnti	1346

**Figure 32: UCI decoding for CBA transmission (eNB side)**

If there is a CBA transmission on the dedicated resource, the eNB will decode the packet. Figure 33 shows that the eNB receives the CBA transmission from UE 1 with C-RNTI 1346, UE 0 with C-RNTI bb02, and UE 2 with C-RNTI 367c.

```
PHY[I][eNB 0] Frame 173, Subframe 2 : received ULSCH SDU from CBA transmission, UE (1,1346), CBA (group 1, rnti fff5)
[PHY][I][eNB 0] Frame 173, Subframe 3 : received ULSCH SDU from CBA transmission, UE (1,1346), CBA (group 1, rnti fff5)
[PHY][I][eNB 0] Frame 174, Subframe 3 : received ULSCH SDU from CBA transmission, UE (1,1346), CBA (group 1, rnti fff5)
[PHY][I][eNB 0] Frame 179, Subframe 2 : received ULSCH SDU from CBA transmission, UE (0,bb02), CBA (group 0, rnti fff4)
[PHY][I][eNB 0] Frame 179, Subframe 3 : received ULSCH SDU from CBA transmission, UE (0,bb02), CBA (group 0, rnti fff4)
[PHY][I][eNB 0] Frame 180, Subframe 3 : received ULSCH SDU from CBA transmission, UE (2,367c), CBA (group 0, rnti fff4)
[PHY][I][eNB 0] Frame 183, Subframe 2 : received ULSCH SDU from CBA transmission, UE (1,1346), CBA (group 1, rnti fff5)
[PHY][I][eNB 0] Frame 183, Subframe 3 : received ULSCH SDU from CBA transmission, UE (1,1346), CBA (group 1, rnti fff5)
[PHY][I][eNB 0] Frame 184, Subframe 3 : received ULSCH SDU from CBA transmission, UE (1,1346), CBA (group 1, rnti fff5)
```

**Figure 33: CBA packet is received at eNB**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

It would also be possible that a collision happens when multiple UEs select the same resource block. In this case, if the C-RNTI is detected by eNB, some dedicated resource is allocated for the collided UE such that the collides UE can send its scheduling request or data on the allocated resource. Figure 34 shows that collision is detected for UE 3, and the eNB allocate resource for UE3 to send it scheduling request (SR).

```
[PHY][N][phy_procedures_eNB_RX] [eNB 0] Frame 93 subframe 3 : CBA collision detected for UE3 for group 1, set the SR for this UE
[PHY][N][phy_procedures_eNB_RX] [eNB 0] Frame 103 subframe 3 : CBA collision detected for UE3 for group 1, set the SR for this UE
[PHY][N][phy_procedures_eNB_RX] [eNB 0] Frame 223 subframe 3 : CBA collision detected for UE3 for group 1, set the SR for this UE
[PHY][N][phy_procedures_eNB_RX] [eNB 0] Frame 233 subframe 3 : CBA collision detected for UE3 for group 1, set the SR for this UE
[PHY][N][phy_procedures_eNB_RX] [eNB 0] Frame 243 subframe 3 : CBA collision detected for UE3 for group 1, set the SR for this UE
```

**Figure 34: Collision is detected and dedicated resource is allocated for CBA**

### 3.2. Latency results

We have conducted three experimentations with one eNB and 6 UEs with on mobility in an area of 500x500. The traffic is generated by our *oaisim* traffic generator tool.

The following setup is common to all three experimentations. We use free space path loss exponent 2 with AWGN channel.

```
<TOPOLOGY_CONFIG>
  <AREA>
    <X_m>500</X_m>
    <Y_m>500</Y_m>
  </AREA>
  <MOBILITY>
    <UE_MOBILITY>
      <RANDOM_UE_DISTRIBUTION>
        <NUMBER_OF_NODES>6</NUMBER_OF_NODES>
      </RANDOM_UE_DISTRIBUTION>
      <UE_MOBILITY_TYPE>STATIC</UE_MOBILITY_TYPE> <!-- STATIC -->
    </UE_MOBILITY>
    <eNB_MOBILITY>
      <RANDOM_eNB_DISTRIBUTION>
        <NUMBER_OF_CELLS>1</NUMBER_OF_CELLS>
      </RANDOM_eNB_DISTRIBUTION>
      <eNB_MOBILITY_TYPE>STATIC</eNB_MOBILITY_TYPE>
    </eNB_MOBILITY>
  </MOBILITY>
</TOPOLOGY_CONFIG>
```

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

#### <ENVIRONMENT\_SYSTEM\_CONFIG>

```

<FADING>
  <FREE_SPACE_MODEL_PARAMETERS>
    <PATHLOSS_EXPONENT>2.0</PATHLOSS_EXPONENT>
  </FREE_SPACE_MODEL_PARAMETERS>
  <SMALL_SCALE>AWGN</SMALL_SCALE>
</FADING>
<UE_FREQUENCY_GHz>1.9</UE_FREQUENCY_GHz>

```

#### </ENVIRONMENT\_SYSTEM\_CONFIG>

#### <EMULATION\_CONFIG>

```

<EMULATION_TIME_ms>10000</EMULATION_TIME_ms>
<PERFORMANCE_METRICS>
  <THROUGHPUT>enable</THROUGHPUT>
  <LATENCY>enable</LATENCY>
  <OWD_RADIO_ACCESS>enable</OWD_RADIO_ACCESS>    </PERFORMANCE_METRICS>
<LOG> <!-- set the global log level -->
  <LEVEL>debug</LEVEL>
  <VERBOSITY>medium</VERBOSITY>
</LOG>
<SEED_VALUE>0</SEED_VALUE>

```

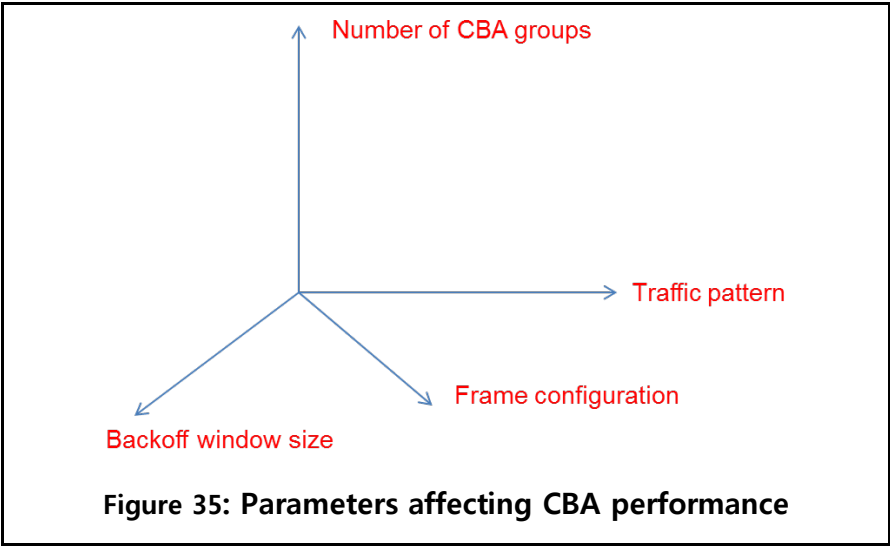
#### </EMULATION\_CONFIG>

In this section, we evaluate the effect of different parameters on the CBA performance. There are three types of parameters that we investigate as shown in Figure 35:

- Traffic pattern: There are various kinds of traffic generated by machine type communications (MTC). Here we investigate three type of traffic: SCBR (small constant bit rate), MCBR (medium constant bit rate), and BCBR (big constant bit rate). The SCBR traffic is related to the un-correlated traffics which are generated by different UEs in a large span of time, for example: periodical traffic from sensors (humidity, temperature, etc.) in a greenhouse. While the BCBR traffic accounts for the correlated traffic which are generated by different UEs in a small span of time, for example: event driven (ED) traffic generated by sensors in a security system. The MCBR traffic represents medium- related traffic generated by UEs in a medium span of time. It is obvious that, compared to the SCBR traffic, more collisions happens for BCBR traffic as more packets are generated by UEs simultaneously.
- Backoff window size: Backoff is used UE to avoid collision. However, larger backoff counter increases the time a UE has to defer before a CBA transmission. Therefore, how to set an optimal backoff window size is not obvious.
- Number of CBA group: Allocating UEs into larger number of group reduces the number of transmissions in one group. However, the resource for a group also

becomes smaller, which increases collision. Here, we investigate its effect on CBA performance.

It has to be mentioned that the frame format (FDD or TDD), and in case of TDD frame format, the TDD frame uplink configuration also has the effect on the CBA performance.



### 3.2.1. Experimentation with different type of traffic

We conduct our experiments with three types of traffic: SCBR (small constant bit rate), MCBR (medium constant bit rate), and BCBR (big constant bit rate). For the SCBR traffic, the main feature is:

```

Transport Protocol: UDP;
IP : IPV4;
Inter-departure time (IDT) distribution : FIXED;
IDT_min: uniform_dist((i+1)*30, 1000)) //i is index of UE
Packet size (PS) distribution : FIXED;
PS_min: 32;
  
```

For the MCBR traffic, its main feature is:

```

Transport Protocol: UDP;
IP : IPV4;
Inter-departure time (IDT) distribution : FIXED;
IDT_min: uniform_dist((i+1)*30, 500)) //i is index of UE
Packet size (PS) distribution : FIXED;
PS_min: 64;
  
```

For the BCBR traffic, its main feature is

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Transport Protocol: UDP;  
 IP : IPV4;  
 Inter-departure time (IDT) distribution : FIXED;  
 IDT\_min: uniform\_dist((i+1)\*30, 250)) //i is index of UE  
 Packet size (PS) distribution : FIXED;  
 PS\_min: 128;

We can see that the average packet size increases from SCBR (32 bits) traffic to BCBR (128 bits). For the inter departure time (IDT), it is uniform distributed over a defined range. It is obvious that the BCBR traffic has the smallest IDT, which means more packets are generated for BCBR traffic. The traffic of UE can be set by the use of following configuration.

```

<APPLICATION_CONFIG>
  <PREDEFINED_TRAFFIC>
    <SOURCE_ID>1:6</SOURCE_ID> <!-- valid formats are: -->
    <APPLICATION_TYPE>scbr (or mcbr, or bcbr)</APPLICATION_TYPE>
    <DESTINATION_ID>0</DESTINATION_ID> <!-- valid formats are: -->
  </PREDEFINED_TRAFFIC>
</APPLICATION_CONFIG>

```

Figure 36 shows the latency (average latency over the 6 UEs) comparison with CBA and regular scheduling under different type of traffic. We can see that the maximum latency is obtained when using BCBR traffic for CBA and regular scheduling. This is reasonable as there are more packets with BCBR traffic, therefore it takes more time to be scheduled by eNB for the regular scheduling time and it incurs more collisions for CBA which also increases latency. Moreover, we also find that the latency of CBA is less than that of regular scheduling for SCBR and MCBR traffic, while it is larger than the latency of regular scheduling for the BCBR traffic. The reason for this phenomenon is that: For the SCBR and MCBR traffic, the collision rate is low (less than 5%) as the IDT is large for these two traffics. Therefore, CBA outperforms the regular scheduling as it reduces the redundant signalling in the regular scheduling. However, the collision rate becomes higher (20%) for the BCBR traffic as IDT is getting smaller. Hence, the latency of CBA becomes higher than the regular scheduling, which suggests that the performance of CBA is highly degraded when the collision rate is high.

Figure 37 presents the delay jitter for CBA and regular scheduling with different types of traffic. We can see that the delay jitter increases from 6.8 ms to 20 ms for regular

scheduling when changing the traffic from SCBR to BCBR. In contrast, the jitter increases sharply from 10 ms to 39 ms when replacing the SCBR traffic with BCBR traffic. This is because, the collision rates is increased by the use of BCBR traffic, which hence enlarges the delay jitter. One thing has to be noted that the delay jitter for CBA is less that for the regular scheduling when using MCBR traffic.

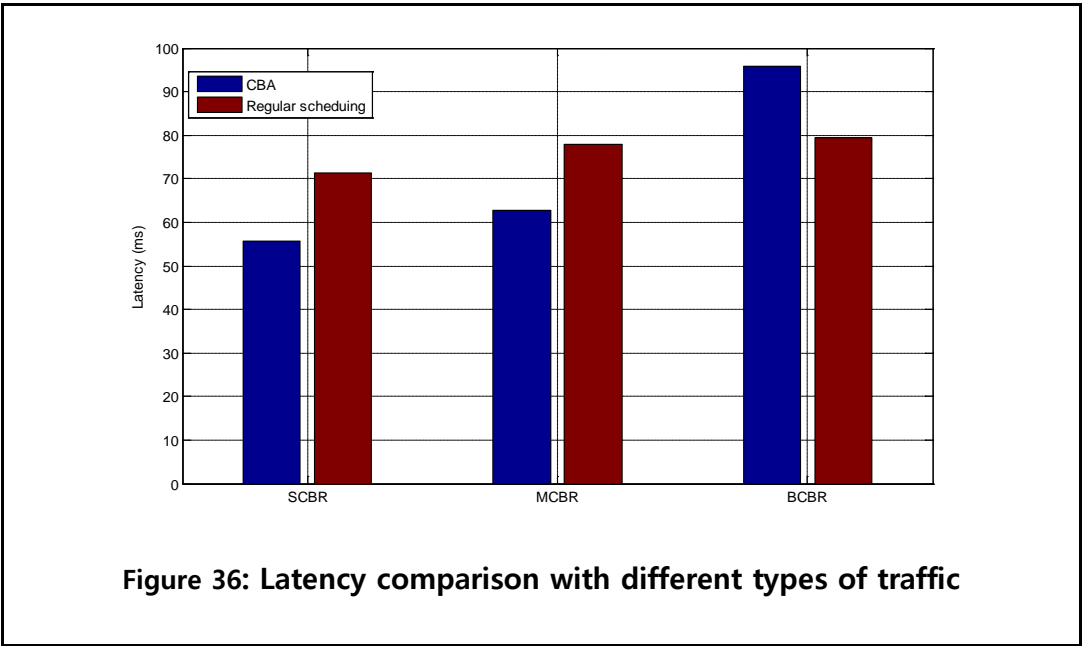
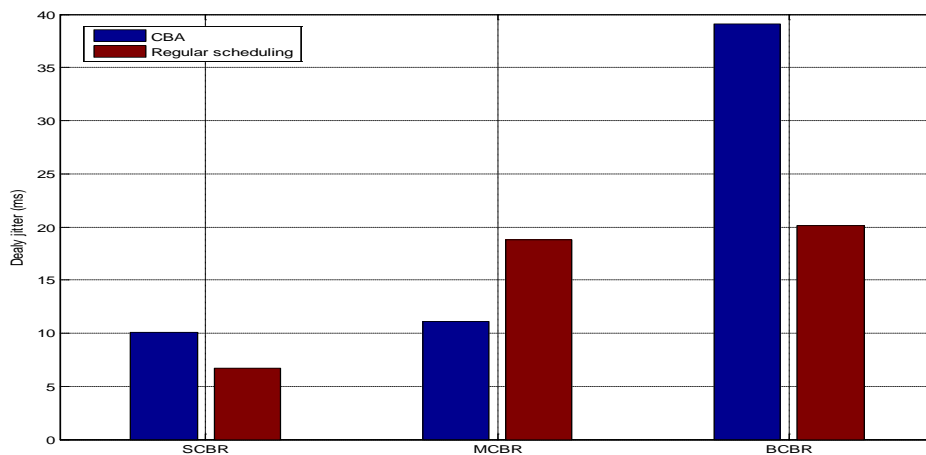
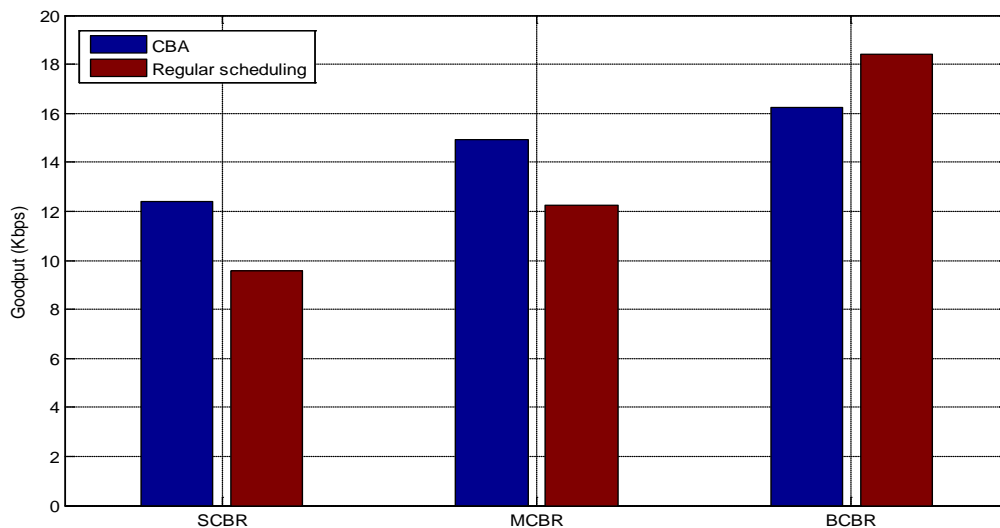


Figure 38 demonstrates the goodput comparison with CBA and regular scheduling under different types of traffic. It can be found that the goodput of CBA is larger than that of regular scheduling for SCBR and MCBR traffic, while it is smaller than goodput of regular scheduling for SBR traffic. This is because of the latency comparison between CBA and regular scheduling that we explained in the last paragraph. In addition to that, we also find that though the latency for BCBR traffic is higher than that of SCBR and MCBR as shown in Figure 36, the goodput for BCBR is larger than that of SCBR and MCBR, which is also reasonable as the packet size of BCBR traffic is 128 bits (much larger than SCBR and MCBR packet).



**Figure 37: Delay jitter comparison with different types of traffic**



**Figure 38: Good comparison with different types of traffic**

### 3.2.2. Experimentation with different backoff window size

With CBA, a UE sends its packet on the allocated CBA resource without redundant signaling incurred by regular scheduling. However, as the CBA resource is not UE dedicated, collision happens when multiple UEs use the same resource, which increases latency. To alleviate collision, backoff is employed for a CBA transmission. Concretely, a UE selects a backoff counter uniform distributed over  $[1, W]$  after a CBA transmission. This backoff counter is decreased by one every frame. A UE can trigger a CBA transmission when the backoff counter equals 0. We conduct experiments with different backoff

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

counter to investigate its effect. The traffic used in our experiments is configured as following.

```

Transport Protocol: UDP;
IP : IPV4;
Inter-departure time (IDT) distribution : FIXED;
IDT_min: uniform_dist((i+1)*30, 250)) //i is index of UE
Packet size (PS) distribution : FIXED;
PS_min: 128

```

```

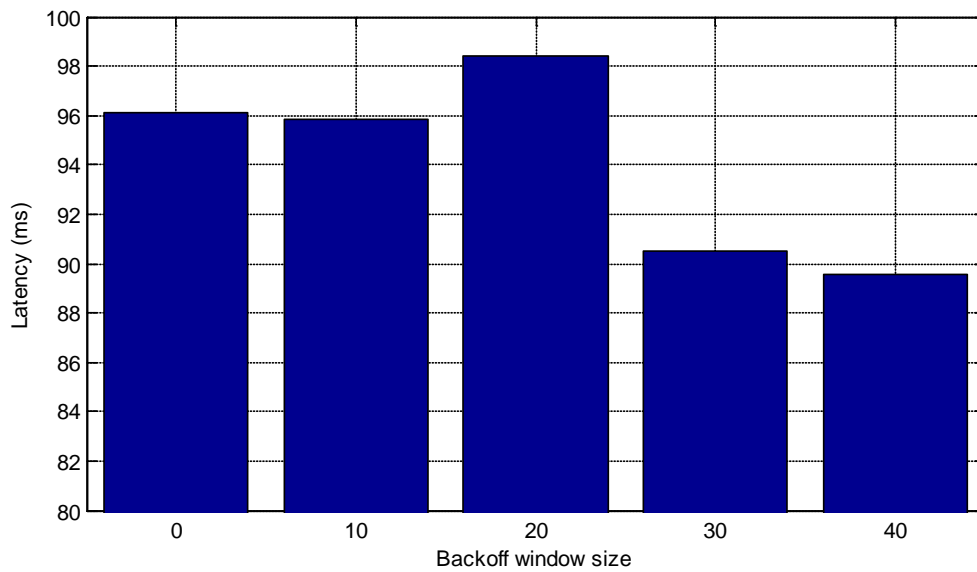
<APPLICATION_CONFIG>
  <PREDEFINED_TRAFFIC>
    <SOURCE_ID>1:6</SOURCE_ID> <!-- valid formats are: -->
    <APPLICATION_TYPE>BCBR <APPLICATION_TYPE>
    <DESTINATION_ID>0</DESTINATION_ID> <!-- valid formats are: -->
  </PREDEFINED_TRAFFIC>
</APPLICATION_CONFIG>

```

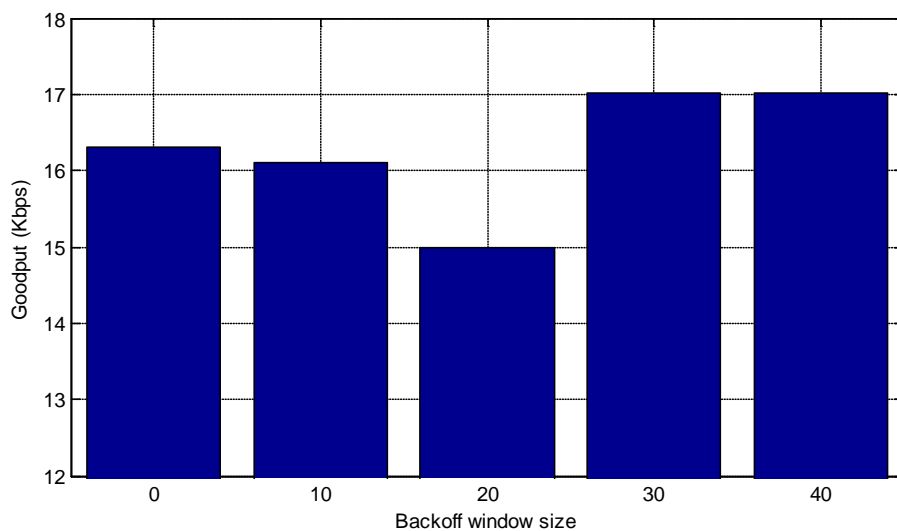
Figure 39 shows the latency comparison with different backoff counter window size. We can see that the latency increases 96 ms to 98 ms when the backoff window size increases from 0 to 20. This because though the collision rate decreases when backoff window size increases from 0 to 20, a UE has to wait more time before sending a packet which increases latency. In contrast to that, the latency decreases to 89ms when the backoff window increases from 20 to 40. This is also reasonable because the collision rate decreases, which reduces the latency.

Figure 40 compares the goodput of different backoff window size. The maximum goodput is achieved when the backoff counter equals 40, which is consistent with the result shown in Figure 39.

Using a larger backoff window reduces collision rate. However, a UE also has to wait longer time before transmission. Therefore, a good tradeoff between collision rate reduce and waiting time increase should be selected such that the latency can be minimized.



**Figure 39: Latency comparison with different backoff window size**



**Figure 40: Goodput comparison with different backoff window size**

### 3.2.3. Experimentation with different number of CBA groups

Another factor which may the performance of CBA is the number of CBA group. Here, we conduct experiments with different number of CBA groups to investigate its effect. The configuration for the traffic is as following.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Transport Protocol: UDP;  
 IP : IPV4;  
 Inter-departure time (IDT) distribution : FIXED;  
 IDT\_min:  $\text{uniform\_dist}((i+1)*30, 250)$  //i is index of UE  
 Packet size (PS) distribution : FIXED;  
 PS\_min: 128;

```

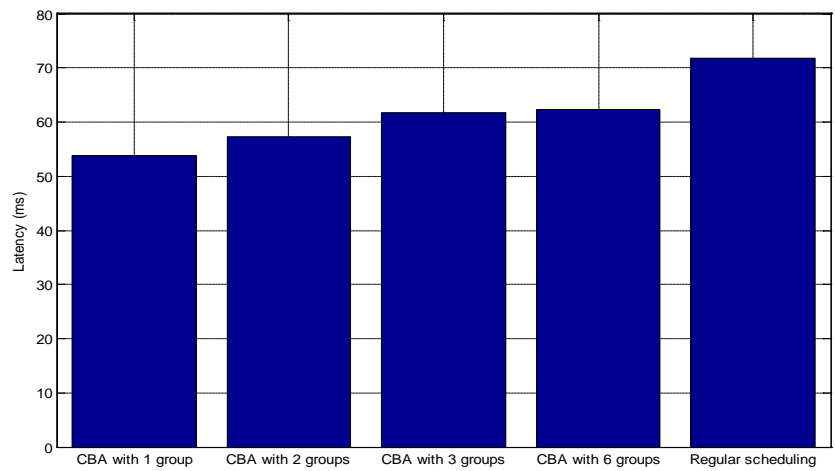
<APPLICATION_CONFIG>
  <PREDEFINED_TRAFFIC>
    <SOURCE_ID>1:6</SOURCE_ID> <!-- valid formats are: -->
    <APPLICATION_TYPE>BCBR <APPLICATION_TYPE>
    <DESTINATION_ID>0</DESTINATION_ID> <!-- valid formats are: -->
  </PREDEFINED_TRAFFIC>
</APPLICATION_CONFIG>

```

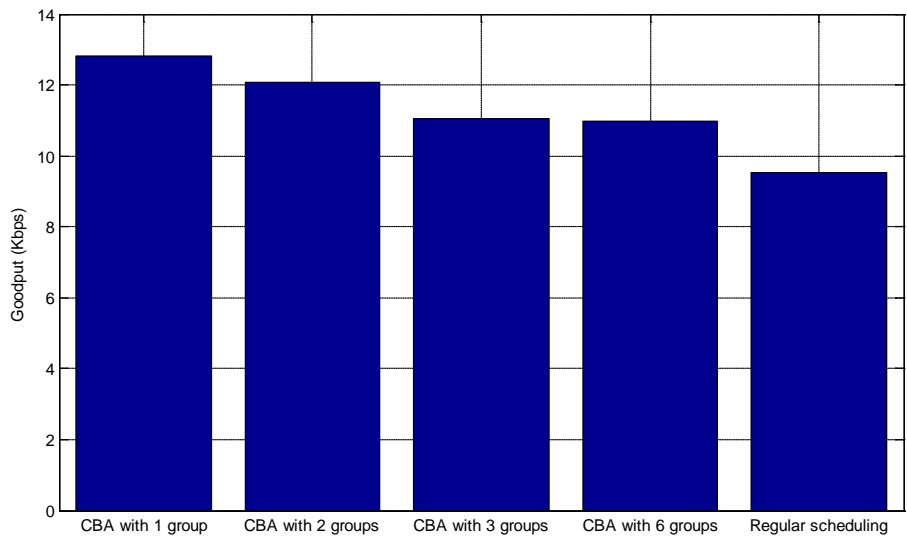
Figure 41 compares the latency with different number of CBA group as well as regular scheduling. We can see that the latency increases from 53 ms to 61 ms as CBA group number increases from 1 to 6. This is because allocating UEs into 6 groups reduces the number of UE in one CBA group (from 6 to 1), which may reduce the collision rate. However, as the CBA resource is fixed, the resource allocated for one CBA group decreases when using 6 CBA groups, which in turn increase the collision rate.

Figure 42 demonstrates the goodput when setting number of CBA group. We can find that setting CBA group as 1 achieves the best goodput (12.4 Kbps) as it obtains the smallest latency which is shown in Figure 41.

Therefore, how to set the number of CBA group is not so obvious. Using larger number of CBA group decreases the number of UE but also reduces the amount of resource in one CBA group. In contrast, using smaller number of CBA group increases the number of UE in a group and also increases the amount of resource in a group. To achieve the best performance of CBA, the number of CBA group should be carefully selected.



**Figure 41: Latency comparison with different number of CBA group**



**Figure 42: Latency comparison with different number of CBA group**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

## 4. CONCLUSION

In this deliverable we summarize the implementation and performance evaluation results of the contention based access (CBA) method on the OpenAirInterface (OAI) platform created by EURECOM. We elaborate the component, configuration and emulation process for the OAI platform. The idea of contention based access (CBA), its software architecture, and the modifications to the LTE protocol are also detailed.

We conducted extensive emulations to validate the implementation for the CBA method on the OAI platform. The signaling results in RRC, MAC, and PHY layer show that CBA is properly implemented on OAI. Concretely, the emulation results show that CBA works as following: (1) the eNB configures the CBA-RNTI through RRC signaling; (2) an UE receives this RRC message and configure it PHY and MAC layer; (3) the eNB allocates resource for CBA transmission; (4) an UE send a packet on the CBA resource; (5) the UE tries to decode CBA packets; (6) if a C-RNTI is received, dedicated resource is allocated for the related UE.

We also carried out different experimentation to compare CBA with the regular scheduling and to evaluate the effect of different parameters on the CBA performance. We find that when the traffic is sporadic (SCBR and MCBR) CBA outperforms the regular scheduling method. However, the performance of CBA degrades for the BCBR traffic whereby the traffic is intense and of large size. We also notice that the backoff window size and number of CBA group have great effect on the CBA performance. To achieve the best performance of CBA, the backoff window size and number of CBA group should be carefully selected. Specially, for the configuration of backoff window size, a goodput tradeoff between waiting time increase and collision rate reduce should be selected. As for adjustment of CBA group number, the suitable CBA number of group should decreases the collision rate for a CBA transmission.

## 5. ACRONYMS AND DEFINITIONS

### 5.1. Acronyms

Acronym	Defined as
3GPP	3 <sup>rd</sup> Generation Partnership Project
AP	Auto-Pilot
ENB	Enhanced Node B
CBA	Contention-based Random Access
DCI	Downlink Control Information
GW	Gateway
GPU	Graphical Processing Unit
LTE	Long Term Evolution
LTE-A	LTE-Advanced
IDT	Inter-Departure Time
MAC	Medium Access Control
M2M	Machine to Machine
MSC	Message Sequence Chart
OTG	Openair Traffic Generator
OA	OpenArena
OWD	One-Way Delay
PHY	PHYsical (layer)
PS	Packet Size
PDU	Protocol Data Unit
TF	Team Fortress
VB	Virtual Bike
UE	User Equipment
UCI	Uplink Control Information

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

## 6. REFERENCES

- [1] LOLA Project (Achieving Low-Latency in Wireless Communications), "D4.2 Enhancements to Framing and Low-Layer Signalling v1.0", August, 2011.
- [2] Ben Romdhanne, Bilel; Navid; Nikaein; Raymond, Knopp; Christian, Bonnet, "OpenAirInterface large-scale wireless emulation platform and methodology", PM2HW2N 2011, Florida, USA.
- [3] Ben Romdhanne, Bilel; Nikaein, Navid; Bonnet, Christian, "Coordinator-master-worker model for efficient large scale network simulation", SIMUTOOLS 2013, 6th International ICST Conference on Simulation Tools and Techniques, March 5-7, 2013, Cannes, France.
- [4] Latif, Imran; Kaltenberger, Florian; Nikaein, Navid; Knopp, Raymond, "Large scale system evaluations using PHY abstraction for LTE with OpenAirInterface", EMUTOOLS 2013, Cannes, France.
- [5] Hafsaoui, Aymen; Nikaein, Navid; Bonnet, Christian, " Analysis and experimentation with a realistic traffic generation tool for emerging application scenarios", EMUTOOLS 2013, Cannes, France.
- [6] Zhou, Kaijie; Nikaein, Navid; Knopp, Raymond; Bonnet, Christian, "Contention based access for machine-type communications over LTE", VTC 2012-Spring, IEEE 75th Vehicular Technology Conference, May 6-9, 2012, Yokohama, Japan.
- [7] Zhou, Kaijie; Nikaein, Navid; Knopp, Raymond, "Dynamic resource allocation for machine-type communications in LTE/LTE-A with contention-based access", WCNC 2013, IEEE Wireless Communications and Networking Conference, April 7-10, 2013, Shanghai, China.
- [8] Perumalla, K. Parallel and distributed simulation: traditional techniques and recent advances. In Proceedings of the 38th conference on Winter simulation, 2006.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

## ANNEX A UDP AND TCP MEASUREMENT ON LIVE HSPA NETWORK (MTS/TUV) – DETAILED RESULTS FOR DELAY MODEL

These following measurements are a follow-up to measurements performed within D3.5. These measurements are used to derive an accurate and realistic delay model for the mobile cellular system from terminal to the server including firewall, APN, and proxy. The measurements are performed for both TCP and UDP traffic.

### A.1 UDP measurements in MTS network

#### A.1.1 Measurement setup

Measurement setup is topologically the same as for measurements described in D3.5, as well as test phones (Huawei U8500).

The 3G/HSPA network has been modernized meanwhile, and its configuration changed – besides the increase of channel elements and licenses for simultaneous HS users on the subject NodeB, the TTI in the uplink is reduced to 2ms (before it was 10ms).

Serving NodeB, BGU44, is of following characteristics in this test case:

- 256/256 CE UL/DL activated
- eUL activated (and HSDPA)
- two carriers, HS traffic going to both carriers
- license for 32 simultaneous HSDPA users in each cell
- 2ms TTI in UL

All other setup parameters are the same as in Test case 3 of D3.5, and further.

Trace recording was performed on phones (Shark application). Network cell statistics is gathered, as well as application reports from phones.

Since there was no possibility to measure one-way delay, due to lack of synchronization between tracing points, the Traffic Generator application has been changed in such way that for every received UDP packet server application generates a "fake ACK". Client application, upon receipt of "fake ACK", calculates the "RTT" and records it in its report. If

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

the “fake ACK” does not arrive within 3 seconds, the application considers it lost and generates a new wait time and a new packet. Application report also contains the counter of sent and lost packets. As these fake ACKs are generated at the application layer, no “RTT” can be calculated through Wireshark without creating a custom dissector that would decode the data in the UDP packet. Such dissector was not developed, and thus, trace recording at other network interfaces was not performed.

The “default” APN gprswap was used (uses proxy, Service Awareness, GGSN 1, CheckPoint firewall).

This test case is intended to show the differences in the RTT compared to TCP case. Since the NodeB has been modernized, having more processing power and smaller TTI in the UL, the comparison with previously performed measurements (described in D3.5) is just illustrative. Both UDP and TCP measurements (described in A.2) are performed in order to dissect portions of RTT belonging to different parts of network, and get statistics to be used in testbed1. UDP measurements are given first, for later comparison.

RTTs are expected to be smaller than for previous TCP cases in D3.5, for all traffic patterns, not only because of the upgrade, but also because of UDP characteristics. Being a connectionless protocol, UDP does not establish and maintain a session. There are no retransmissions of packets, so the actual traffic on the link is closer to the defined patterns and RTTs for “successful” packets are smaller. There is no ordering of packets, so there’s no processing of out-of-order packets nor drops that might occur in the core (service-aware GGSN features). UDP has a smaller header compared to TCP, thus having smaller packets for same data content, which can also influence the RTT.

### A.1.2 Measurement parameters

All measurement parameters are the same as in test case 3 of D3.5, the only change is for time distribution for phones 9 and 10, and the transport protocol used - UDP. For emulated GPS Keep Alive messages, inter-arrival time distribution should be Uniform (1,60)s, but since it lead to frequent application failures in past measurements, Uniform (1,25s) is applied (as in cases 7-11 of D3.5), since it was checked empirically that this was the widest range for which application worked without interruption.

The parameters are shown in the following tables:

<b>Server:</b>	89.216.116.166
<b>Port:</b>	1234

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

**Table 10: Server IP address and port**

Name	Telephone No	Application/Protocol	Settings
test1	0641069123	OA, UL, UDP	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s
test2	0641069116	TF, UL, UDP	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s
test3	0641069117	OA,DL, UDP	Gauss (0,16836;0,08381)kB*, Uniform(0,041;0,047)s
test4	0641069118	TF, DL, UDP	Gauss (0,23511;0,07748)kB*, Uniform(0,039;0,046)s
test5	0641069119	M2M, BR, UL, UDP	Constant(1)kB, Uniform(0,1;0,5)s
test6	0641069120	M2M, BR, DL, UDP	Constant(1)kB, Uniform(0,1;0,5)s
test7	0641069121	M2M, AP, UL, UDP	Constant(1)kB, Uniform(0,025;0,1)s
test8	0641069122	M2M, AP, DL, UDP	Constant(1)kB, Uniform(0,999;1,001)s
test9	0641069115	M2M, TT(GPS Keep Alive), UL, UDP	Constant(0,5)kB, Uniform(1;25)s**
test10	0641069124	M2M, TT(GPS Keep Alive), UL, UDP	Constant(0,5)kB, Uniform(1;25)s**

\* Gaussian distribution was taken as an approximate, since LogNormal generated packets of around 1028 bytes all the time in Test Case 1. See deliverable D3.5 for explanation.

\*\* No possibility for Uniform(0,00977;1)kB packet size distribution, Constant was taken instead; Time distribution should be (1,60)s, but due to application failures in previous test cases, (1,25)s was taken instead, as an empirical limit for smooth operation.

**Table 11: Simulation parameters for UDP test case**

Test Case 12 was performed 11.15-13.00 on May 15<sup>th</sup>, 2012.

### A.1.3 Measurement results

Relevant notes on application behaviour are given in the following table.

Name	Application/Protocol	Settings	From time	To time	Remark
test1	OA, UL, UDP	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s	11:14	12:58	
test2	TF, UL, UDP	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s	11:15	12:58	
test3	OA,DL, UDP	Gauss (0,16836;0,08381)kB, Uniform(0,041;0,047)s	11:16	12:56	Application restarted at 11:20, was failing 11:21, 11:35, 11:40, 11:42, 11:46, 11:52, 11:55, 12:08, 12:23, 12:31, 12:41, 12:45, 12:47, 12:51
test4	TF, DL, UDP	Gauss (0,23511;0,07748)kB, Uniform(0,039;0,046)s	11:17	12:56	Application failed at 11:19, 11:35, 12:56
test5	M2M, BR, UL, UDP	Constant(1)kB, Uniform(0,1;0,5)s	11:17	12:58	
test6	M2M, BR, DL, UDP	Constant(1)kB, Uniform(0,1;0,5)s	11:18	12:59	
test7	M2M, AP, UL, UDP	Constant(1)kB, Uniform(0,025;0,1)s	11:16	12:58	
test8	M2M, AP, DL, UDP	Constant(1)kB, Uniform(0,999;1,001)s	11:18	13:00	

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>		
------	-------------------	-----	--	--	--

test9	M2M, TT(GPS Keep Alive), UL, UDP	Constant(0,5)kB, Uniform(1;25)s	11:13	12:57	
test10	M2M, TT(GPS Keep Alive), UL, UDP	Constant(0,5)kB, Uniform(1;25)s	11:15	12:57	

**Table 12: Notes on application behaviour for UDP test case**

#### A.1.3.1 Phone RTT

As mentioned above, the RTT measured by the client application on the phone is the period from the time the original UDP packet is sent to the time its corresponding "fake ACK" is received back.

The statistics is calculated from application reports concerning generated packet size, time between packets and the calculated RTTs. Yet, a more thorough analysis was done through Wireshark revealing that for some phones "fake ACKs" are all of same data content, i.e. they do not refer to a specific packet, so delays affect RTT calculated by application – received "fake ACK" may be for some previous packet and there is no way application could determine that. Seems that the "fake ACK" is implemented depending on the length of the main packet, so phones test5-test10, with constant size packets, have same "fake ACK" data content through all two hours of testing, while other phones have a Gaussian distribution of "fake ACK" data content.

Due to problems stated above, another approach was taken. RTTs cannot be calculated directly through Wireshark, but the traces are exported to Excel, and RTTs are calculated for series of values packet-"fake ACK", for as long as we have an uninterrupted flow of successive pairs packet-"fake ACK". As soon as we get two successive packets originated from the client, no RTTs are calculated, from that point on, since we later cannot determine which "fake ACK" belongs to which packet. Two successive client-originated packets mean that either the ACK for the first is really lost, either it is delayed more than 3s (the application waits for the fake ACK 3s and then considers it lost, but it may arrive later). In this way, no delay longer than 3s is taken into account, but we may assume that this has no greater impact on results, due to large number of packet exchanged and small number of instances where delay is longer than 3s.

If we compare data from application reports with time difference between client packets and returned server fake ACKs in Shark traces (beginning of traces), we may conclude that they match (a recorded UDP „fake RTT" corresponds closely to the time difference between sending a packet and receiving a „fake ACK" packet back). This proves that the client application processing delay is negligible, which is also exactly shown in the TCP

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

test case (A.2). As indicated, the application waits for the “fake ACK” and then generates a new packet with new wait time, so the traffic pattern does not follow size/time distributions exactly, but very closely, unlike in the TCP case. Packet size is nominal size plus UDP header.

Anyhow, in the way of processing traces described above, the possibility of an error is reduced to zero (there are no retransmissions), but the statistical sample is significantly smaller, with no excessive delays taken into account (as the first instance of delay larger than 3s occurs, no further packets are taken into statistics, as explained above).

In the following table, the statistics is given, with the number of samples (pairs packet-“fake ACK”) that the statistics is calculated for:

	test1	test2	test3	test4	test5
Count of RTT	194	78	1021	1246	1314
Average of RTT (ms)	<b>118.08</b>	<b>121.37</b>	<b>131.44</b>	<b>139.07</b>	<b>256.63</b>
Max of RTT (ms)	2214.14	1928.25	2712.82	2838.57	2933.95
Min of RTT (ms)	76.72	76.68	67.36	78.12	189.98

	test6	test7	test8	test9	test10
Count of RTT	531	1352	260	54	57
Average of RTT (ms)	<b>279.86</b>	<b>236.78</b>	<b>348.32</b>	<b>1716.55</b>	<b>1601.81</b>
Max of RTT (ms)	2082.35	2925.48	2963.14	2786.88	2883.28
Min of RTT (ms)	221.04	193.37	188.63	150.20	152.46

**Table 13: Statistics for UDP phone RTT taken from Shark traces**

It is also interesting to see the average values next to traffic characteristics:

Name	Settings	Application/ Protocol	Traffic			Average UDP phone RTT
			Avg. packet size (bytes)	Average time between packets (s)	Max throughput (kbps)	
test1	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s	OA, UL, UDP	40	0.086	6.68	118.08
test2	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s	TF, UL, UDP	75	0.0365	33.27	121.37
test3	Gauss (0,16836;0,08381)kB, Uniform(0,041;0,047)s	OA,DL, UDP	170	0.044	94.32	131.44
test4	Gauss (0,23511;0,07748)kB, Uniform(0,039;0,046)s	TF, DL, UDP	240	0.0425	117.39	139.07
test5	Constant(1)kB, Uniform(0,1;0,5)s	M2M, BR, UL, UDP	1024	0.3	80.00	256.63

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>		
------	-------------------	-----	--	--	--

test6	Constant(1)kB, Uniform(0,1;0,5)s	M2M, BR, DL, UDP	1024	0.3	80.00	279.86
test7	Constant(1)kB, Uniform(0,025;0,1)s	M2M, AP, UL, UDP	1024	0.0625	320.00	236.78
test8	Constant(1)kB, Uniform(0,999;1,001)s	M2M, AP, DL, UDP	1024	1	8.01	348.32
test9	Constant(0,5)kB, Uniform(1;25)s	M2M, TT(GPS Keep Alive), UL, UDP	512	13	4.00	1.716.55
test10	Constant(0,5)kB, Uniform(1;25)s	M2M, TT(GPS Keep Alive), UL, UDP	512	13	4.00	1.601.81

**Table 14: Traffic characteristics vs. UDP phone RTT**

Comparing these statistics with results from Test Case 3 of D3.5, first we observe significantly lower RTTs. Average RTTs are multiple times smaller than in case 3, except for phones 9 and 10, with very sporadic traffic patterns, whose RTTs are of the same order in two test cases.

These results are expected, as noted above, not only due to increase in processing power of the NodeB and TTI reduction to 2ms, but also because of UDP characteristics, compared to TCP case. However, sporadic traffic patterns of phones 9 and 10 still have rather same, large average RTTs of 1.7s.

Smallest average RTTs are those of phones 1 and 2, around 120ms, then for phones 3 and 4, around 130-140ms.

It is clearly visible that traffic patterns of phones test5-test8 have more than 100ms longer RTTs, between 240ms and 350ms, than those of phone test1-test4. If we compare traffic parameters, we may deduce that this is due to the influence of packet length on RTT. Large packets have bigger RTTs. This will be confirmed in the TCP case as well. In this group of phones, we further see the influence of inter-arrival time – phone test8 with largest inter-arrival time, practically 1s constantly, has the biggest RTT, while phone test7, with smallest wait time gets the smallest RTT.

These results again confirm that sporadic traffic patterns have large RTTs, for UDP as well, as they are given random-access channels, which are shared and have high access times.

In addition, very long generated „wait time“ (time between packets), if larger than the value of corresponding inactivity timers (in this case, up to 22s), results in phone going to the Idle state. This means that for sending the next packet, phone must first establish the RRC connection again. This situation occurs with phones 9 and 10, and additional time is

being spent on signalling. Introducing the URA\_PCH state in the network would help reduce this excessive signalling for phones with large inter-arrival times.

In the end, just as an illustration, statistics from application reports is given:

	<i>test1</i>	<i>test2</i>	<i>test3</i>	<i>test4</i>	<i>test5</i>
Average of UDP RTT (ms)	346,76	397,93	224,62	233,85	343,82
	<i>test6</i>	<i>test7</i>	<i>test8</i>	<i>test9</i>	<i>test10</i>
Average of UDP RTT (ms)	345,75	325,99	384,05	1.658,19	1.682,22

**Table 15: Application report UDP RTTs, not valid due to “fake ACK” ambiguous implementation**

We may see that they are larger than those from Shark traces, except for phones with large inter-arrival times – test8, test9 and test10. From application reports, for instance for test1, we see that RTTs are mostly around 100ms, until the first packet is recorded lost (i.e. “fake ACK” delay was more than 3s). The RTT for the next packet is more than 1s, and then all RTTs are 250ms and higher, as the order is disrupted. As a result, phones with small inter-arrival times have larger RTTs in application reports than in Shark trace analysis, due to “fake ACK” implementation. On the other hand, for phones with large inter-arrival times average RTTs are alike, relative difference not being bigger than 5%, as individual RTTs are high themselves. Anyhow, values calculated from application reports are much smaller than in Test case 5 of D3.5.

## A.2 TCP measurements in MTS network

### A.2.1 Measurement setup

Measurement setup is topologically the same as for measurements described in D3.5, as well as test phones (Huawei U8500).

The network has been modernized meanwhile, and the NodeB configuration changed – besides the increase of channel elements and licenses for simultaneous HS users, the TTI in the uplink is reduced to 2ms (before it was 10ms).

Serving NodeB, BGU44, is of following characteristics in this test case:

- 256/256 CE UL/DL activated
- eUL activated (and HSDPA)
- two carriers, HS traffic going to both carriers

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

- license for 32 simultaneous HSDPA users in each cell
- 2ms TTI in UL
- HSPA+ and Dual Carrier functionalities activated (but phones do not support these features)

All other setup parameters are the same as in Test case 3 of D3.5, and further.

Trace recording was performed on phones (Shark application), at the Gn interface and within the firewall (Wireshark application). Firewall traces were taken by direct filtering by IP addresses of phones. Network cell statistics is gathered, as well as application reports from phones.

The “default” APN gprswap was used (uses proxy, Service Awareness, GGSN 1, CheckPoint firewall).

This test case is intended to show the differences in the RTT compared to UDP case, as well as to dissect RTT to portions belonging to different parts of the network, in order to provide delay statistics for testbed1. The NodeB has been modernized compared to D3.5 measurements, having more processing power and smaller TTI in the UL.

RTTs are expected to be smaller than for previous TCP cases in D3.5, for all traffic patterns, due to network upgrade.

## A.2.2 Measurement parameters

All measurement parameters are the same as in Test case 3 of D3.5, the only change is for time distribution for phones 9 and 10. For emulated GPS Keep Alive messages, inter-arrival time distribution should be Uniform (1,60)s, but since it lead to frequent application failures in past measurements, Uniform (1,25s) is applied (as in cases 7-11 of D3.5), since it was checked empirically that this was the widest range for which application worked without interruption.

The parameters are shown in the following tables:

<b>Server:</b>	89.216.116.166
<b>Port:</b>	1234

**Table 16: Server IP address and port**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Name	Telephone No	Application/Protocol	Settings
test1	0641069123	OA, UL, TCP	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s
test2	0641069116	TF, UL, TCP	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s
test3	0641069117	OA,DL, TCP	Gauss (0,16836;0,08381)kB*, Uniform(0,041;0,047)s
test4	0641069118	TF, DL, TCP	Gauss (0,23511;0,07748)kB*, Uniform(0,039;0,046)s
test5	0641069119	M2M, BR, UL, TCP	Constant(1)kB, Uniform(0,1;0,5)s
test6	0641069120	M2M, BR, DL, TCP	Constant(1)kB, Uniform(0,1;0,5)s
test7	0641069121	M2M, AP, UL, TCP	Constant(1)kB, Uniform(0,025;0,1)s
test8	0641069122	M2M, AP, DL, TCP	Constant(1)kB, Uniform(0,999;1,001)s
test9	0641069115	M2M, TT(GPS Keep Alive), UL, TCP	Constant(0,5)kB, Uniform(1;25)s**
test10	0641069124	M2M, TT(GPS Keep Alive), UL, TCP	Constant(0,5)kB, Uniform(1;25)s**

\* Gaussian distribution was taken as an approximate, since LogNormal generated packets of around 1028 bytes all the time in Test Case 1. See deliverable D3.5 for explanation.

\*\* No possibility for Uniform(0,00977;1)kB packet size distribution, Constant was taken instead; Time distribution should be (1,60)s, but due to application failures in previous test cases, (1,25)s was taken instead, as an empirical limit for smooth operation.

**Table 17: Simulation parameters for TCP test case**

Test Case 14 was performed 10.27-12.00 on June 22<sup>nd</sup>, 2012.

## A.2.3 Measurement results

Relevant notes on application behaviour are given in the following table.

Name	Telephone No	Application /Protocol	Settings	From time	To time	Remark
test1	0641069123	OA, UL, TCP	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s	10:27	12:00	
test2	0641069116	TF, UL, TCP	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s	10:27	12:00	
test3	0641069117	OA,DL, TCP	Gauss (0,16836;0,08381)kB, Uniform(0,041;0,047)s	10:27	12:00	Application failed at 10:29,10:32, 10:38,10:39,10:46,10:51,10:56, 10:57,11:17,11:21,11:29,11:32, 11:51,11:56
test4	0641069118	TF, DL, TCP	Gauss (0,23511;0,07748)kB, Uniform(0,039;0,046)s	10:27	12:00	Application failed at 11:27, 11:31.
test5	0641069119	M2M, BR, UL, TCP	Constant(1)kB, Uniform(0,1;0,5)s	10:27	12:00	
test6	0641069120	M2M, BR, DL, TCP	Constant(1)kB, Uniform(0,1;0,5)s	10:27	12:00	
test7	0641069121	M2M, AP, UL, TCP	Constant(1)kB,	10:27	12:00	

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

			Uniform(0,025;0,1)s			
test8	0641069122	M2M, AP, DL, TCP	Constant(1)kB, Uniform(0,999;1,001)s	10:27	12:00	
test9	0641069115	M2M, TT(GPS Keep Alive), UL, TCP	Constant(0,5)kB, Uniform(1;25)s	10:27	12:00	Application failed and restarted at 11:20.
test10	0641069124	M2M, TT(GPS Keep Alive), UL, TCP	Constant(0,5)kB, Uniform(1;25)s	10:27	12:00	Application failed at 11:17. Restarted at 11:21. Failed at 11:30, restarted.

**Table 18: Notes on application behaviour for TCP test case**

#### A.2.3.2 TCP flow – phone, Gn interface and firewall

In order to analyze the flow of TCP packets through subject interfaces more thoroughly, excerpts from Shark traces are given for the chosen phone i.e. traffic pattern – phone test9 is chosen, because of constant packet size, easy to follow, as well as because of large wait times resulting in large RTTs, distinguishing it along with test10 from other phones.

Several packets exchanged between client at test9 and server are given in the following tables, seen by Shark application on the phone and Wireshark applications used to capture packets at Gn interface and the firewall.

Same packets are marked with the same colour. Packets captured at the firewall are triplicated, as the same packet passes through the tracing point three times, which is interpreted by Wireshark as retransmissions, but eventually only one packet of the three exits to the backbone network.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Phone trace test9\*:

No.	Time	Source	Destination	Protocol	Length	The RTT to ACK the segment was	Info
353	10:43:03.038178	10.130.81.223	89.216.116.166	TCP	70		40518 > search-agent [PSH, ACK] Seq=33025 Ack=1281 Win=5880 Len=4 TSval=163536 TSecr=6668697
354	10:43:04.926066	89.216.116.166	10.130.81.223	TCP	66	1,887888	search-agent > 40518 [ACK] Seq=1281 Ack=33029 Win=16640 Len=0 TSval=6671308 TSecr=163536
355	10:43:04.926688	10.130.81.223	89.216.116.166	TCP	578		40518 > search-agent [PSH, ACK] Seq=33029 Ack=1281 Win=5880 Len=512 TSval=163725 TSecr=6671308
356	10:43:05.075513	89.216.116.166	10.130.81.223	TCP	86	0,148825	search-agent > 40518 [PSH, ACK] Seq=1281 Ack=33541 Win=16128 Len=20 TSval=6671324 TSecr=163725
357	10:43:05.076071	10.130.81.223	89.216.116.166	TCP	66	0,000558	40518 > search-agent [ACK] Seq=33541 Ack=1301 Win=5880 Len=0 TSval=163740 TSecr=6671324
358	10:43:14.891318	10.130.81.223	89.216.116.166	TCP	70		40518 > search-agent [PSH, ACK] Seq=33541 Ack=1301 Win=5880 Len=4 TSval=164721 TSecr=6671324

\* Colour indicates individual packets at different tracing points.

**Table 19: Phone trace for phone test9, excerpt**

Gn trace test9\*\*:

No.	Time	Source	Destination	Protocol	Length	The RTT to ACK the segment was	This is an ACK to the segment in frame	Info
6601	10:43:04.254811	10.130.81.223	89.216.116.166	GTP <TCP>	106	26,567014	4983	40518 > search-agent [PSH, ACK] Seq=5673 Ack=221 Win=2940 Len=4 TSval=163536 TSecr=6668697
6629	10:43:04.471333	89.216.116.166	10.130.81.223	GTP <TCP>	102	0,216522	6601	search-agent > 40518 [ACK] Seq=221 Ack=5677 Win=65 Len=0 TSval=6671308 TSecr=163536

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

6630	10:43:04.471340	89.216.116.166	10.130.81.223	GTP <TCP>	102			search-agent > 40518 [ACK] Seq=221 Ack=5677 Win=65 Len=0 TSval=6671308 TSecr=163536
6645	10:43:04.613674	10.130.81.223	89.216.116.166	GTP <TCP>	614			40518 > search-agent [PSH, ACK] Seq=5677 Ack=221 Win=2940 Len=512 TSval=163725 TSecr=6671308
6646	10:43:04.626239	89.216.116.166	10.130.81.223	GTP <TCP>	122	0,012565	6645	[TCP Retransmission] search-agent > 40518 [PSH, ACK] Seq=221 Ack=6189 Win=63 Len=20 TSval=6671324 TSecr=163725
6647	10:43:04.626268	89.216.116.166	10.130.81.223	GTP <TCP>	122			[TCP Retransmission] search-agent > 40518 [PSH, ACK] Seq=221 Ack=6189 Win=63 Len=20 TSval=6671324 TSecr=163725
7552	10:43:15.652313	89.216.116.166	10.130.81.223	GTP <TCP>	102			[TCP ACKed lost segment] search-agent > 40518 [ACK] Seq=241 Ack=6193 Win=63 Len=0 TSval=6672494 TSecr=164721
7563	10:43:15.795232	89.216.116.166	10.130.81.223	GTP <TCP>	122			[TCP ACKed lost segment] search-agent > 40518 [PSH, ACK] Seq=241 Ack=6705 Win=67 Len=20 TSval=6672508 TSecr=164910
7599	10:43:16.114365	10.130.81.223	89.216.116.166	GTP <TCP>	106	12,161088	6582	[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=6189 Ack=241 Win=2940 Len=4 TSval=164721 TSecr=6671324

\*\*This trace is obtained by chronological merging of two traces from two branches of the Gn interface.

**Table 20: Gn trace for phone test9, excerpt**

FW trace test9\*\*\*:

No.	Time	Source	Destination	Protocol	Length	The RTT to ACK the segment was	Info
159350	10:00:50.307074	10.130.81.223	89.216.116.166	TCP	70		40518 > search-agent [PSH, ACK] Seq=33025 Ack=1281 Win=5880 Len=4 TSval=163536 TSecr=6668697
159351	10:00:50.307081	10.130.81.223	89.216.116.166	TCP	70		[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=33025 Ack=1281 Win=5880 Len=4 TSval=163536 TSecr=6668697
159352	10:00:50.307086	10.130.81.223	89.216.116.166	TCP	70		[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=33025 Ack=1281 Win=5880 Len=4 TSval=163536 TSecr=6668697
159398	10:00:50.523587	89.216.116.166	10.130.81.223	TCP	66	0,216513	search-agent > 40518 [ACK] Seq=1281 Ack=33029 Win=16640 Len=0 TSval=6671308

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

							TSecr=163536
159399	10:00:50.523592	89.216.116.166	10.130.81.223	TCP	66		[TCP Dup ACK 159398#1] search-agent > 40518 [ACK] Seq=1281 Ack=33029 Win=16640 Len=0 TSval=6671308 TSecr=163536
159400	10:00:50.523596	89.216.116.166	10.130.81.223	TCP	66		[TCP Dup ACK 159398#2] search-agent > 40518 [ACK] Seq=1281 Ack=33029 Win=16640 Len=0 TSval=6671308 TSecr=163536
159434	10:00:50.665945	10.130.81.223	89.216.116.166	TCP	578		40518 > search-agent [PSH, ACK] Seq=33029 Ack=1281 Win=5880 Len=512 TSval=163725 TSecr=6671308
159435	10:00:50.665952	10.130.81.223	89.216.116.166	TCP	578		[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=33029 Ack=1281 Win=5880 Len=512 TSval=163725 TSecr=6671308
159436	10:00:50.665958	10.130.81.223	89.216.116.166	TCP	578		[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=33029 Ack=1281 Win=5880 Len=512 TSval=163725 TSecr=6671308
159440	10:00:50.678488	89.216.116.166	10.130.81.223	TCP	86	0,012543	search-agent > 40518 [PSH, ACK] Seq=1281 Ack=33541 Win=16128 Len=20 TSval=6671324 TSecr=163725
159441	10:00:50.678493	89.216.116.166	10.130.81.223	TCP	86		[TCP Retransmission] search-agent > 40518 [PSH, ACK] Seq=1281 Ack=33541 Win=16128 Len=20 TSval=6671324 TSecr=163725
159442	10:00:50.678497	89.216.116.166	10.130.81.223	TCP	86		[TCP Retransmission] search-agent > 40518 [PSH, ACK] Seq=1281 Ack=33541 Win=16128 Len=20 TSval=6671324 TSecr=163725
159458	10:00:50.745868	10.130.81.223	89.216.116.166	TCP	66	0,06738	40518 > search-agent [ACK] Seq=33541 Ack=1301 Win=5880 Len=0 TSval=163740 TSecr=6671324
159459	10:00:50.745874	10.130.81.223	89.216.116.166	TCP	66		[TCP Dup ACK 159458#1] 40518 > search-agent [ACK] Seq=33541 Ack=1301 Win=5880 Len=0 TSval=163740 TSecr=6671324
159460	10:00:50.745878	10.130.81.223	89.216.116.166	TCP	66		[TCP Dup ACK 159458#2] 40518 > search-agent [ACK] Seq=33541 Ack=1301 Win=5880 Len=0 TSval=163740 TSecr=6671324
161474	10:01:02.166366	10.130.81.223	89.216.116.166	TCP	70		40518 > search-agent [PSH, ACK] Seq=33541 Ack=1301 Win=5880 Len=4 TSval=164721 TSecr=6671324
161475	10:01:02.166375	10.130.81.223	89.216.116.166	TCP	70		[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=33541 Ack=1301 Win=5880 Len=4 TSval=164721 TSecr=6671324
161476	10:01:02.166380	10.130.81.223	89.216.116.166	TCP	70		[TCP Retransmission] 40518 > search-agent [PSH, ACK] Seq=33541 Ack=1301 Win=5880 Len=4 TSval=164721 TSecr=6671324

\*\*\* Packets in FW trace are triplicated due to multiple passes through the tracing point, and are recognized by Wireshark as retransmissions.

**Table 21: Firewall trace for phone test9, excerpt**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

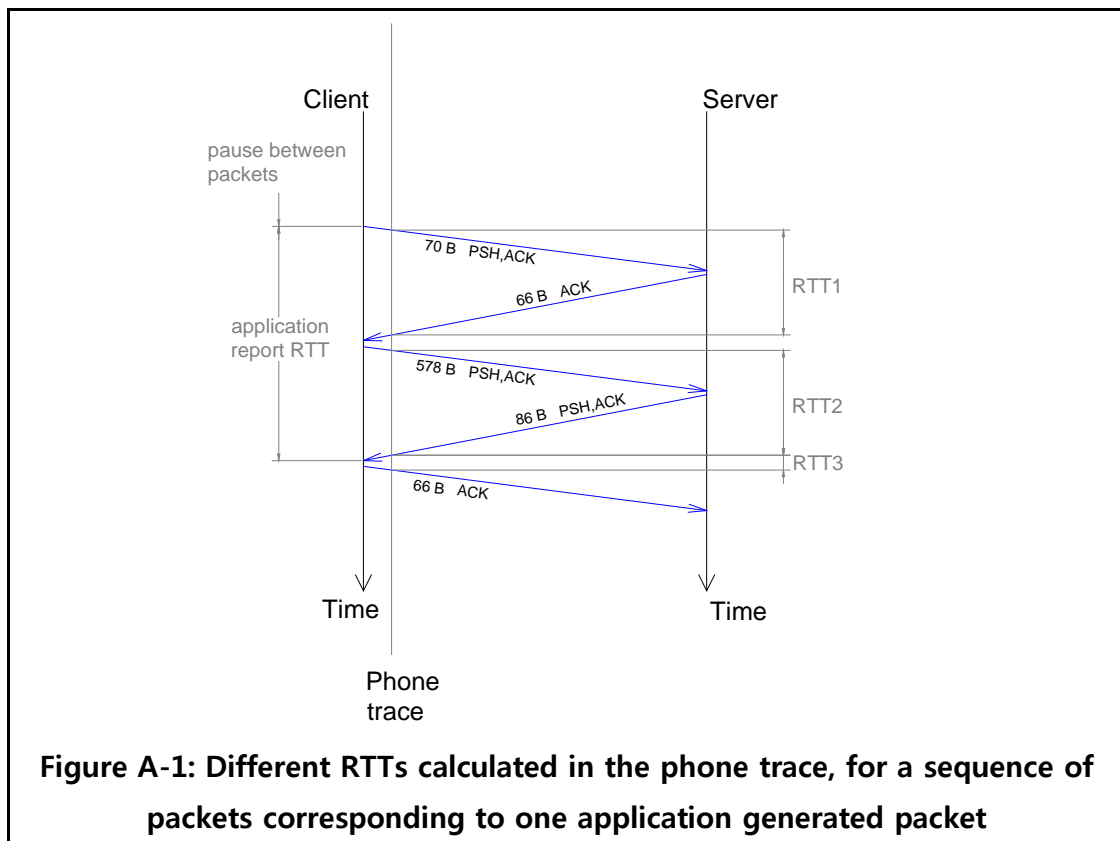
Looking at test9 phone trace, first we observe that after the initial TCP handshake procedure, and final client ACK, for each original packet of 512 bytes sent by the application, we see a 70B PSH,ACK packet from the client, ACKed with 66B packet from the server, then the 578B PSH,ACK packet containing the original (application) 512B packet. Then follows the server's 86B PSH,ACK response, followed by a 66B client ACK. This sequence is repeated for every packet sent by the application.

Here, we distinguish several RTTs in the sequence. „**First RTT**“ recorded by Wireshark is the RTT to the receipt of server's acknowledgement to the first 70B packet. In the phone trace table above we see that this RTT is 1.888s. „**Second RTT**“ of 0.149s in the phone trace table is for the server's response to the 578B packet (main payload). The last RTT recorded in the sequence contains the application processing delay, since it is the time, recorded at the phone, between the receipt of server's ACK and sending the client's acknowledgement for the received ACK to the server. This last recorded RTT in the sequence should be ignored for the phone trace, since it represents delay within the phone, not the network delay.

Looking at first two phone RTTs, we also see that the first RTT is much longer than the second one. This is, as explained in D3.5, due to network behaviour for long inter-arrival times between packets. In this case, the time between the last message exchanged between client and server (**client ACK**) for the previous packet and the first message for the current packet is around 24s – this exceeds the value of inactivity timer of the Channel Switching function in underlying WCDMA network plus round-trip time, so the UE goes into the IDLE state, and for sending the new packet it has to establish the RRC connection again. Going through the trace, we observe that for smaller inter-arrival times, this „first RTT“ is much smaller, since the UE maintains RRC connection.

The RTTs recorded in the application report are in fact sum of these first two RTTs for each application packet, i.e. sequence of two TCP packets and two ACKs. The third RTT, as mentioned, is related to the phone and client application.

The figure below (**Figure A-1**) depicts different RTTs recorded for a sequence of packets exchanged between client and server for one application packet.



For phone test9, we have the following statistics:

	"First RTT"	"Second RTT"
Average of The RTT to ACK the segment was (s)	1,5325	0,4395
Max of The RTT to ACK the segment was (s)	6,4416	15,9085
Min of The RTT to ACK the segment was (s)	0,2799	0,1371
StdDev of The RTT to ACK the segment was (s)	0,8455	1,2074

**Table 22: Phone test9 RTT statistics by parts of TCP sequence**

Taking into account first and second RTTs all together (including the ACK in the TCP handshake), the statistics is as follows:

	RTT
Count of The RTT to ACK the segment was	373
Average of The RTT to ACK the segment was (s)	0,9973
Max of The RTT to ACK the segment was (s)	15,9085
Min of The RTT to ACK the segment was (s)	0,1371
StdDev of The RTT to ACK the segment was (s)	1,1909

**Table 23: Phone test9 RTT statistics for all packets together**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

If we do the statistics on sums of first and second RTTs, i.e. on overall RTT for the application packet, excluding the ACK in the TCP handshake, we get the following:

„Overall“ RTT statistics, Shark trace for test9	
Count of overall (sum) RTT	186
Average of overall (sum) RTT (s)	1,9898
Max of overall (sum) RTT (s)	17,9195
Min of overall (sum) RTT (s)	0,4180
StdDev of overall (sum) RTT (s)	1,4066

**Table 24: Phone test9 statistics for overall (sum) RTT of the TCP sequence**

These values correspond to the values recorded by the first application report (until 11:16), including the exact number of packets:

test9 RTT statistics from application report, until 11:16	
Count of RTT (ms):	186
Average of RTT (ms):	1991,04
Max of RTT (ms):	17921
Min of RTT (ms):	419
StdDev of RTT (ms):	1406,50

**Table 25: Phone test9 RTT statistics obtained from application report**

Shark trace does not go further, while the application created another report after re-establishing the TCP connection (application failed at 11:20, and was restarted).

Maximum RTT recorded (15.9s) is due to several retransmissions of the packet (with main payload – 578B): there were 2 retransmissions after the original packet, and the RTT is calculated relatively to the departure of the first, original packet.

It is interesting to see the statistics for client ACKs as well – they are a measure of application processing delay (time between server’s PSH, ACK arriving to the phone and client’s ACK leaving the phone):

	Client ACK RTT (phone)
Average of The RTT to ACK the segment was (s)	0,00063
Max of The RTT to ACK the segment was (s)	0,00187
Min of The RTT to ACK the segment was (s)	0,00027
StdDev of The RTT to ACK the segment was (s)	0,00024

We see that the application processing delay is negligible (<2ms).

Looking at the Gn trace, we first observe that packet lengths are somewhat bigger, as on this interface we have TCP encapsulated into GTP i.e. few more headers

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

added (GTP, UDP, IP), 36 bytes long. Second, sequence and ACK numbers are altered (but not the timestamps TSval and TSecr).

In lines 6601 and 7599 we see that RTT is calculated for PSH,ACK packets going from the client to the server, and these are the only two instances occurring in test9 Gn trace. These RTTs should be ignored, as they are calculated in the wrong „place“, due to the loss of client ACK messages preceding these two client PSH,ACK messages. Instead of calculating RTT for client ACK, being lost, Wireshark calculated RTT for the succeeding PSH,ACK message having same Seq, Ack and TSecr numbers as expected ACK. Since the client ACK and the succeeding PSH,ACK message belong to two sequences of packets, there's a pause between them (wait time in the application between generation of packets), which is consequently included into the false RTT.

For the Gn interface, it is important to note that the Gn trace is obtained by chronologically merging two traces captured at two branches of the Gn interface (capturing was done on switches of these two branches). Laptops used for capturing were not synchronized. We see retransmissions that are not originated from the phone (lines 6630, 6647), ACKs to lost segments, server's PSH,ACK messages interpreted as retransmission (line 6646). All this will be further analyzed in the section with Gn trace statistics. Also, one should have in mind that these traces last only up to 5 minutes, unlike phone and FW traces.

At this interface, the client ACK RTT may be of interest. As explained above, a sequence of packets exchanged for every application packet has three RTTs – two belong to server's ACKs acknowledging client's packets, while the last belongs to client's ACK to server's PSH,ACK message. This „third“ ACK was ignored for phone trace, containing processing delay in the phone, but at the Gn interface, this ACK contains two-way delay in the mobile network, from the Gn interface to the phone and back, including the processing delay in the phone. So, this RTT tells us about the mobile „pre-Gn“ network, including the access network delay, delay on the IuB, in the RNC, and on the way to the GGSN (RNC, SGSN and GGSN are physically close); while the other two tell us about the delay from the Gn interface to the server and back.

The firewall (FW) trace follows the phone trace, only packets are triplicated, as the FW „bounces“ them through the tracing point three times.

Although we just saw that generated TCP packets do not follow the size/time distributions exactly, valid conclusions concerning latency statistics, as well as network behaviour, may be drawn, which will be shown in subsequent sections.

### A.2.3.3 Phone RTT

#### A.2.3.3.1 RTT measured by TGen application itself (.txt reports)

The delay statistics calculated from application reports is given in the following table.

	<i>test1</i>	<i>test2</i>	<i>test3</i>	<i>test4</i>	<i>test5</i>
Count of RTT (ms)	8787	10864	9836	10295	6155
Average of RTT (ms)	<b>501.00</b>	<b>428.14</b>	<b>437.34</b>	<b>442.83</b>	<b>558.89</b>
Max of RTT (ms)	3,509.00	4,513.00	8,922.00	3,284.00	9,088.00
Min of RTT (ms)	429.00	351.00	351.00	357.00	466.00
StdDev of RTT (ms)	92.35	192.86	233.04	154.56	343.17
First packet RTT	3,509.00	418.00	379.00	425.00	511.00

	<i>test6</i>	<i>test7</i>	<i>test8</i>	<i>test9</i>	<i>test10</i>
Count of RTT (ms)	6303	8452	3425	336	326
Average of RTT (ms)	<b>534.35</b>	<b>548.63</b>	<b>577.30</b>	<b>1,971.15</b>	<b>1,914.85</b>
Max of RTT (ms)	6,984.00	5,606.00	7,595.00	17,921.00	7,319.00
Min of RTT (ms)	462.00	469.00	476.00	419.00	420.00
StdDev of RTT (ms)	168.26	249.79	363.67	1,112.66	776.10
First packet RTT	6,984.00	4,533.00	2,267.00	443.00	1,275.00

**Table 26: RTT statistics taken from application reports**

As in previous test cases, phones with sporadic traffic (test9 and test10) have large RTTs (1.9-2.0s in average), while other phones have average RTTs spanning from 428ms to 577ms. These RTTs, recorded by application, are not the image of network RTT, i.e. they do not represent the network response to a single packet sent by some time/size statistical distribution. They represent the overall RTT, sum of two RTTs for 4 TCP packets exchanged between client and server for one application packet of nominal size (according to packet size distribution).

It is also interesting to see RTTs recorded for first packets in application reports, for all phones – they differ and are not related to traffic distributions, but are of rather

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

random nature, depending on current conditions and the ability of radio access network to assign resources in the very moment.

For purposes of comparison, RTTs obtained from application reports in Test Case 3 (TCP, D3.5) are given below:

	<i>test1</i>	<i>test2</i>	<i>test3</i>	<i>test4</i>	<i>test5</i>
Average of RTT (ms)	651.30	595.91	776.35	833.50	985.63
	<i>test6</i>	<i>test7</i>	<i>test8</i>	<i>test9</i>	<i>test10</i>
Average of RTT (ms)	788.72	863.19	871.38	1,758.76	1,897.47

**Table 27: Average RTTs from application reports in Test case 3**

We see that the upgrade of NodeB led to significantly lower RTTs for all traffic patterns except those of phones 9 and 10.

#### *A.2.3.3.2 RTT measured by Shark application on the phone*

The statistics taken from Shark traces captured on phones is given below:

Traffic						Phone RTT			
Name	Settings	Application/ Protocol	Avg. packet size (bytes)	Average time between packets (s)	Max throughput (kbps)	Average phone RTT first packet in sequence	Average phone RTT second packet in sequence (main payload)	Average phone RTT (all packets)	Average phone client ACK (processing delay) RTT
test1	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s	OA, UL, TCP	40	0.086	6.68	0.3486	0.1489	0.2489	0.0012
test2	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s	TF, UL, TCP	75	0.0365	33.27	0.3117	0.1138	0.2129	0.0010
test3	Gauss (0,16836;0,08381)kB, Uniform(0,041;0,047)s	OA,DL, TCP	170	0.044	94.32	0.3091	0.1243	0.2184	0.0011
test4	Gauss (0,23511;0,07748)kB, Uniform(0,039;0,046)s	TF, DL, TCP	240	0.0425	117.39	0.3100	0.1300	0.2203	0.0011
test5	Constant(1)kB, Uniform(0,1;0,5)s	M2M, BR, UL, TCP	1024	0.3	80.00	0.3252	0.2308	0.2784	0.0012
test6	Constant(1)kB, Uniform(0,1;0,5)s	M2M, BR, DL, TCP	1024	0.3	80.00	0.3077	0.2220	0.2653	0.0012
test7	Constant(1)kB, Uniform(0,025;0,1)s	M2M, AP, UL, TCP	1024	0.0625	320.00	0.3169	0.2287	0.2731	0.0014
test8	Constant(1)kB, Uniform(0,999;1,001)s	M2M, AP, DL, TCP	1024	1	8.01	0.3427	0.2313	0.2873	0.0011
test9	Constant(0,5)kB, Uniform(1,25)s	M2M, TT(GPS Keep Alive), UL,	512	13	4.00	1.5325	0.4395	0.9973	0.0006

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>						
------	-------------------	-----	--	--	--	--	--	--	--

		TCP							
test10	Constant(0,5)kB, Uniform(1;25)s	M2M, TT(GPS Keep Alive), UL, TCP	512	13	4.00	1.5562	0.3558	0.9713	0.0008

**Table 28: TCP phone RTT statistics from Shark traces**

Average phone RTT for one application packet (average by sums of two RTTs), as shown in previous section, is up to 2ms smaller than the RTT recorded by application, so this column is omitted.

We notice that the average RTT for first packet in sequence is larger than the RTT for the second packet in sequence. We may also notice that in the group of phones with rather „fast“ traffic (test1-test8), largest RTTs for second packet (main payload) are recorded for phones test5-test8 whose generated packets are the largest (marked blue in **Table 22**). Packet size, as suspected, also influences the RTT (bigger packets have larger RTTs), but not as much as the increased wait time between packets.

Concerning high average 1<sup>st</sup> TCP packet RTTs for phones test1-test7 and even test8, that have rather small inter-arrival time, explanation does not lie in packet size, as it is rather small (70B). Also, looking at TCP flow just as a series of packets and some times between them, no conclusion can be made, as the generated inter-arrival time (time between two application packets, i.e. last TCP packet in one sequence and the first in successive sequence) is sometimes bigger, sometimes smaller than the time (RTT) between two TCP packets in a sequence. Also, if we process the results from Test cases 7-11 of D3.5 in the same way, we see that they also show large RTTs for 1<sup>st</sup> TCP packets, meaning this occurs regardless of core network features (different APNs). So, the place to look is some server processing for the first TCP packet, some wake-up time of the server, which will be further validated through Gn and firewall trace analysis, and explored in Section A.2.3.7.

The influence of accessing the network for patterns with large inter-arrival times may also be seen comparing the traces of test9 and test10 with traces from Cases 7-11 – network modernization led to smaller RTTs for 1<sup>st</sup> TCP packet of phones with large inter-arrival times.

We may compare the TCP results, from Shark traces, with average RTTs for the UDP case, given in A.1.3.1. UDP results are given again for comparison reasons:

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>	
------	-------------------	-----	--	--

	<i>test1</i>	<i>test2</i>	<i>test3</i>	<i>test4</i>	<i>test5</i>
Average of UDP RTT (ms)	118,08	121,37	131,44	139,07	256,63
	<i>test6</i>	<i>test7</i>	<i>test8</i>	<i>test9</i>	<i>test10</i>
Average of UDP RTT (ms)	279,86	236,78	348,32	1716,55	1601,81

**Table 29: UDP phone RTT**

First one should have in mind that the first TCP packet is only 70B long, and for large inter-arrival times, it takes on itself the influence of accessing the network. The second bears the influence of generated packet length and current network conditions. In UDP, the only packet sent bears all impacts.

Second, UDP traces had to be processed manually, as explained in A.1.3.1, due to "fake ACK" ambiguous implementation, and the taken trace excerpt does not take into account long delays.

Comparing the TCP and UDP RTTs, the following conclusions are drawn:

- First TCP packet vs. UDP: UDP results are much smaller than those for first TCP packet (70B long), except for phones test9 and test10. This means that the 1<sup>st</sup> TCP packet suffers some impact that UDP packet does not. The reason should be sought in server processing for TCP case, as indicated earlier in this section. For phones test 9 and test10, TCP RTT is smaller, as here the influence of long packet accessing the network in UDP case dominates over small packet access plus server processing delay in TCP case.
- Main payload TCP packet vs. UDP: For first 8 phones RTTs are of the same order, TCP RTTs being slightly smaller, and even pronouncedly smaller for phones test5-test8 with larger packets. The reason for this behaviour, TCP and UDP packets being similar in size, may lie in inter-arrival time, the access network, or server processing for UDP packets. Inter-arrival time for 2<sup>nd</sup> TCP packet is the last RTT, as the packet is sent immediately after the ACK for the 1<sup>st</sup> is received, while for UDP packets wait time is the last "RTT" plus nominal wait time (see A.1.3.1). For phones test9 and test10, TCP RTTs are about 4 times smaller, as expected due to access procedure UDP packets have to pass.

#### A.2.3.4 RTT measured by Wireshark on the Gn interface (Gn RTT)

At this point, it is first important to have in mind that the Gn trace comprises only about 3-5 minutes of transmission, due to the large throughput at the interface, even with filtering, and the limitations of capturing methods used. As mentioned above, Gn trace is obtained from two branches of the Gn interface, with laptops that were not synchronized, by chronological merging of two traces (each containing several files).

Further, trace from branch one lasts 5 minutes, and from branch two 3 minutes, with no time synchronization:

Switch 1 trace:	10:40:21-10:45:00
Switch 2 trace:	10:42:51-10:45:34

**Table 30: Duration of taken Gn traces**

Taking into account retransmissions that might occur between the GGSN and the RNC in the user plane (direct tunnel), buffering at both sides, as well as ordering of packets, possible drops of out-of-order packets etc, occurring at the GGSN due to Service Aware functionalities, these traces are not so straightforward for analysis. Also, computing abilities of laptops used may lead to packet drop i.e. disability to record a packet as the new one arrives too fast.

For instance, for phone test9 we first have a lot of server's PSH,ACK and ACK messages, covering the time span of 3 minutes, acknowledging packets that seem to never have passed (nor they are recorded further in the trace). Then there is a part with all packets present, but with some non-causal acknowledgements (first there is an ACK recorded, after comes the packet to which that ACK belongs, then again the ACK) – obviously, client's packet must have passed or we wouldn't have server's response recorded.

The situation is similar with all phones, in some traces we have only client's packets recorded in the end, without server's response-messages, but in the phone trace we see that these have arrived to the phone.

As an illustration, message flow on the Gn interface follows. If we mark 5 TCP messages exchanged for 1 application packet as 1-1, 1-2, 1-3, 1-4 and 1-5, for the first application packet, and similarly 2-1..2-5 for the second etc, and take an excerpt from the Gn trace and corresponding messages from the phone trace, we get the following:

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Phone trace								
No.	Time	Source	Destination	Protocol	Length	The RTT to ACK the segment was	Packet-message No.	Meaning
373	10:44:07.171453	10.130.81.223	89.216.116.166	TCP	70		<b>1-1</b>	
374	10:44:09.104961	89.216.116.166	10.130.81.223	TCP	66	1,933508	<b>1-2</b>	ack to 1-1
375	10:44:09.105454	10.130.81.223	89.216.116.166	TCP	578		<b>1-3</b>	
376	10:44:09.246258	89.216.116.166	10.130.81.223	TCP	86	0,140804	<b>1-4</b>	ack to 1-3
377	10:44:09.246889	10.130.81.223	89.216.116.166	TCP	66	0,000631	<b>1-5</b>	cl.ack to 1-4
378	10:44:30.611513	10.130.81.223	89.216.116.166	TCP	70		<b>2-1</b>	
379	10:44:32.632058	89.216.116.166	10.130.81.223	TCP	66	2,020545	<b>2-2</b>	
380	10:44:32.632741	10.130.81.223	89.216.116.166	TCP	578		<b>2-3</b>	
381	10:44:32.781653	89.216.116.166	10.130.81.223	TCP	86	0,148912	<b>2-4</b>	
382	10:44:32.782364	10.130.81.223	89.216.116.166	TCP	66	0,000711	<b>2-5</b>	
383	10:44:50.711871	10.130.81.223	89.216.116.166	TCP	70		<b>3-1</b>	
384	10:44:52.635451	89.216.116.166	10.130.81.223	TCP	66	1,92358	<b>3-2</b>	
385	10:44:52.635976	10.130.81.223	89.216.116.166	TCP	578		<b>3-3</b>	
386	10:44:52.836603	89.216.116.166	10.130.81.223	TCP	86	0,200627	<b>3-4</b>	
387	10:44:52.838364	10.130.81.223	89.216.116.166	TCP	66	0,001761	<b>3-5</b>	
388	10:45:08.648419	10.130.81.223	89.216.116.166	TCP	70		<b>4-1</b>	
389	10:45:13.423121	10.130.81.223	89.216.116.166	TCP	70		<b>4-1re</b>	4-1 retransmitted
390	10:45:15.089976	89.216.116.166	10.130.81.223	TCP	66	6,441557	<b>4-2</b>	
391	10:45:15.090824	10.130.81.223	89.216.116.166	TCP	578		<b>4-3</b>	
392	10:45:15.746821	89.216.116.166	10.130.81.223	TCP	86	0,655997	<b>4-4</b>	
393	10:45:15.747349	10.130.81.223	89.216.116.166	TCP	66	0,000528	<b>4-5</b>	

**Table 31: Message flow in phone trace, example**

Gn trace									
No.	Time	Source	Destination	Protocol	Length	The RTT to ACK the segment was	Packet-message No.	Interpreted by Wireshark as	Meaning
11855	10:44:07.982929	89.216.116.166	10.130.81.223	GTP <TCP>	102		<b>1-2</b>	ack to lost segment	
11865	10:44:08.125547	89.216.116.166	10.130.81.223	GTP <TCP>	122		<b>1-4</b>	ack to lost segment	
11890	10:44:08.440116	10.130.81.223	89.216.116.166	GTP <TCP>	106		<b>1-1</b>	re	
11906	10:44:08.654951	89.216.116.166	10.130.81.223	GTP <TCP>	102	0.2148	<b>1-2</b>	ack to 1-1	
11907	10:44:08.654957	89.216.116.166	10.130.81.223	GTP <TCP>	102		<b>1-2</b>		duplicate ack to 1-1
11926	10:44:08.779399	10.130.81.223	89.216.116.166	GTP <TCP>	614		<b>1-3</b>	re	

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

11938	10:44:08.858999	10.130.81.223	89.216.116.166	GTP <TCP>	102	0.7335	1-5	ack to 1-4	
13765	10:44:31.503602	89.216.116.166	10.130.81.223	GTP <TCP>	102		2-2	ack to lost segment	
13783	10:44:31.658890	89.216.116.166	10.130.81.223	GTP <TCP>	122		2-4	ack to lost segment	
13803	10:44:31.954486	10.130.81.223	89.216.116.166	GTP <TCP>	106		2-1	re	
13821	10:44:32.175137	89.216.116.166	10.130.81.223	GTP <TCP>	102	0.2207	2-2	ack to 2-1	
13822	10:44:32.175141	89.216.116.166	10.130.81.223	GTP <TCP>	102		2-2		duplicate ack to 2-1
13839	10:44:32.313296	10.130.81.223	89.216.116.166	GTP <TCP>	614		2-3	re	
13847	10:44:32.392942	10.130.81.223	89.216.116.166	GTP <TCP>	102	0.7341	2-5	ack to 2-4	
15366	10:44:51.502930	89.216.116.166	10.130.81.223	GTP <TCP>	102		3-2	ack to lost segment	
15375	10:44:51.693472	89.216.116.166	10.130.81.223	GTP <TCP>	122		3-4	ack to lost segment	
15395	10:44:51.948208	10.130.81.223	89.216.116.166	GTP <TCP>	106		3-1	re	
15416	10:44:52.174208	89.216.116.166	10.130.81.223	GTP <TCP>	102	0.2260	3-2	ack to 3-1	
15417	10:44:52.174213	89.216.116.166	10.130.81.223	GTP <TCP>	102		3-2		duplicate ack to 3-1
15428	10:44:52.346516	10.130.81.223	89.216.116.166	GTP <TCP>	614		3-3	re	
15430	10:44:52.446294	10.130.81.223	89.216.116.166	GTP <TCP>	102	0.7528	3-5	ack to 3-4	
16505	10:45:09.934647	10.130.81.223	89.216.116.166	GTP <TCP>	106		4-1		
16622	10:45:13.032516	10.130.81.223	89.216.116.166	GTP <TCP>	106		4-1re		retransmission from phone
16634	10:45:13.241017	89.216.116.166	10.130.81.223	GTP <TCP>	102	3.3064	4-2	ack to 4- 1re	
16635	10:45:13.241022	89.216.116.166	10.130.81.223	GTP <TCP>	102		4-2	duplicate ack	
16712	10:45:15.186744	10.130.81.223	89.216.116.166	GTP <TCP>	614		4-3		
16713	10:45:15.206761	89.216.116.166	10.130.81.223	GTP <TCP>	122	0.0200	4-4	ack to 4-3	
16714	10:45:15.206770	89.216.116.166	10.130.81.223	GTP <TCP>	122		4-4	re	
16725	10:45:15.366161	10.130.81.223	89.216.116.166	GTP <TCP>	102	0.1594	4-5		

**Table 32: Message flow in Gn trace, example**

On the firewall, the message flow is the same as on the phone.

We see from the tables above that at the Gn first we see the server's ACK to an unseen message, after that passes the original message, and then we have the server's ACK repeated, twice.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

The explanation for non-causality may be found in the method of capturing. Analyzing the traces from two branches of the Gn interface, we notice that for majority of phones, packets from one phone went over both branches of the Gn. This was not the case in previous test cases, where normally we had ALL the traffic from several phones on one branch, and all the traffic from the other phones on the second branch. In order to obtain the full Gn trace for one phone, we merge two traces chronologically, and due to lack of time synchronization during capture - we get non-causal packets.

Some packets interpreted as „retransmissions“ may also be the result of this traffic flow over both Gn branches for the same phone. The same packet may be traced on both branches – these are the interpreted „retransmissions“. Plus, we have true retransmissions occurring over one branch. Then, packets that seem never to have passed, probably passed before the capture on a corresponding branch – have in mind that 5 minutes trace from one branch and 3 minutes trace from the other do not overlap completely, not only according to start and stop times from two traces, but also because of the offset between two referent clocks.

Therefore, the overall (merged traces) statistics taken from Gn traces is not valid for calculating average RTT. Parts of traces for individual phones may be taken (if possible – we only have few minutes of trace), with normal flow and correctly interpreted packets, to draw some valid statistics.

By analyzing the separate traces from two branches, we may describe them as follows, in order to decide which statistics to take:

Name	Switch 1 trace	Switch 2 trace	Result for merged traces chronologically	Statistics should be taken for
test1	Only server messages	Mixed messages	May be used	Switch 2 trace
test2	Only client messages	Only server messages	Invalid.	-
test3	Only client messages	Only server messages	Invalid.	-
test4	Only server messages	Mixed messages	May be used	Switch 2 trace
test5	Only server messages	Mixed messages	May be used	Switch 2 trace
test6	Only server messages	Mixed messages	May be used	Switch 2 trace
test7	Whole trace.	-	To be used.	Switch 1 trace (whole)
test8	Only server messages	Mixed messages	May be used	Switch 2 trace
test9	Only server messages	Mixed messages	May be used	Switch 2 trace
test10	Only server messages	Mixed messages	May be used	Switch 2 trace

**Table 33: Phone traffic occurring at different Gn branches**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Then, the statistics for the Gn interface RTT is as follows:

Traffic					Gn RTT - statistics taken according to upper table				
Name	Settings	Avg.pac ket size (bytes)	Average time between packets (s)	Max throughp ut (kbps)	Average Gn RTT first packet in sequence	Average Gn RTT second packet in sequence (main payload)	Average Gn RTT (all packets)	Average Gn RTT for 1 application packet	Average Gn client ACK (Gn to phone and back) RTT
test1	Gauss (0,04121;0,004497)kB, Uniform(0,069;0,103)s	40	0.086	6.68	0.2318	0.0269	0.1298	0.2578	0.1284
test2	Gauss (0,07473;0,013085)kB, Uniform(0,031;0,042)s	75	0.0365	33.27	-	-	-	-	-
test3	Gauss (0,16836;0,08381)kB, Uniform(0,041;0,047)s	170	0.044	94.32	-	-	-	-	-
test4	Gauss (0,23511;0,07748)kB, Uniform(0,039;0,046)s	240	0.0425	117.39	0.2357	0.0291	0.1305	0.2607	0.0749
test5	Constant(1)kB, Uniform(0,1;0,5)s	1024	0.3	80.00	0.2212	0.0297	0.1266	0.2465	0.0729
test6	Constant(1)kB, Uniform(0,1;0,5)s	1024	0.3	80.00	0.2384	0.0276	0.1302	0.2647	0.0731
test7	Constant(1)kB, Uniform(0,025;0,1)s	1024	0.0625	320.00	0.2423	0.0267	0.1329	0.2661	0.0798
test8	Constant(1)kB, Uniform(0,999;1,001)s	1024	1	8.01	0.2461	0.0327	0.1381	0.2690	0.0726
test9*	Constant(0,5)kB, Uniform(1;25)s	512	13	4.00	0.6043*	0.0183*	0.3789*	0.8529*	0.1051
test10	Constant(0,5)kB, Uniform(1;25)s	512	13	4.00	0.2213	0.0225	0.1219	0.2205	0.0811

\*For phone test9 we have just a few packet sequences recorded properly. Average RTTs are larger due to one retransmission of first packet with 3.3s RTT – without it, for instance, the average RTT for the 1<sup>st</sup> packet would be 0.218s, and for one application packet would be 0.2366s i.e. of the order of other phones' RTTs.

**Table 34: TCP Gn interface RTT statistics from Shark traces**

Again we see large RTT for first packets in sequence, and small RTTs for the second, with main payload, which supports the conclusion about significant server processing delay for the 1<sup>st</sup> TCP packet. Average RTTs for the 1<sup>st</sup> packet are of the same order, meaning there is no selectivity in the core and the backbone concerning traffic patterns. The same is with statistics for 2<sup>nd</sup> TCP packets.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Average client ACK RTTs suggest that the “backward” delay, to the phone and back, is less than or around 100ms.

#### A.2.3.5 RTT measured by Wireshark on the firewall (FW RTT)

The statistics for the RTT measured at the firewall is given in the following table:

Traffic				FW RTT				
Name	Avg. packet size (bytes)	Average time between packets (s)	Max throughput (kbps)	Average FW RTT first packet in sequence	Average FW RTT second packet in sequence (main payload)	Average FW RTT (all packets)	Average FW RTT for 1 application packet*	Average FW client ACK (FW to phone and back) RTT
test1	40	0.086	6.68	0.2270	0.0238	0.1252	0.2500	0.1251
test2	75	0.0365	33.27	0.2267	0.0224	0.1083	0.2485	0.0764
test3	170	0.044	94.32	0.2252	0.0232	0.1237	0.2480	0.0782
test4	240	0.0425	117.39	0.2270	0.0228	0.1246	0.2490	0.0749
test5	1024	0.3	80.00	0.2257	0.0250	0.1243	0.2498	0.0815
test6	1024	0.3	80.00	0.2267	0.0239	0.1250	0.2498	0.0750
test7	1024	0.0625	320.00	0.2277	0.0234	0.1247	0.2500	0.0799
test8	1024	1	8.01	0.2279	0.0236	0.1238	0.2511	0.0826
test9	512	13	4.00	0.2355	0.1044**	0.1693	0.3393**	0.0821
test10	512	13	4.00	-	-	-	-	-

\* This is the average value for the sums of RTTs for the first and second TCP packet sent for one application packet.

\*\* As all values include retransmissions, in case of phone test9 large average RTT is due to a case of 3 retransmissions having 15s RTT – without this instance, the average FW RTT for the 2<sup>nd</sup> packet would be 0.0197s, and for the application packet would be 0.2554s, i.e. similar to other phones.

**Table 35: TCP firewall RTT statistics from Shark traces**

For capturing on the firewall, filtering by phone IP addresses was applied. Phone test10 got one IP address in the beginning of measurements, by which the filter was applied, but later was rebooted and got another IP address – this is the reason why we do not have the FW trace for test10.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Looking at maximum RTTs, large values (1s and more, up to 15s) are due to retransmissions, but these values occur in several instances, i.e. they do not influence much the average RTT.

Also, we do not notice any increase in the FW RTT for the „second“ TCP packet, main payload, for phones with large packets, test5-test8, like we did for phone RTTs – the size of the packet does not influence the RTT in the backbone.

Again, we see that the average RTTs for the 1<sup>st</sup> TCP packet are around 200ms larger than those for the 2<sup>nd</sup> TCP packet, which again implies some server processing delay. RTTs for the 1<sup>st</sup> packet are around 230ms, and for the 2<sup>nd</sup> around 25ms, again with no selectivity concerning traffic pattern.

It is also interesting to look at duplicate ACKs – messages of 78B sent by server as a response to a retransmission from the client. The statistics for server’s „78B“ messages, i.e. server’s ACKs to retransmitted packets, is as follows:

78B server's ack to retransmission	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10
FW RTT (s)	0.0208	0.0183	0.0259	0.0191	0.0233	0.0438	0.0242	0.0214	0.0526	-

**Table 36: Delay statistics for server’s ACKs to retransmitted packets**

Although we have a small sample for these messages, again we see that the RTT from the firewall is small, here between 18 and 52ms.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

#### A.2.3.6 Comparison of RTT statistics at different points (phone, Gn interface, FW)

Phone RTT					Gn RTT					FW RTT				
Name	Average phone RTT first packet in sequence	Average phone RTT second packet in sequence (main payload)	Average phone RTT (all packets)	Average phone client ACK (processing delay) RTT	Average Gn RTT first packet in sequence	Average Gn RTT second packet in sequence (main payload)	Average Gn RTT (all packets)	Average Gn RTT for 1 application packet	Average Gn client ACK (Gn to phone and back) RTT	Average FW RTT first packet in sequence	Average FW RTT second packet in sequence (main payload)	Average FW RTT (all packets)	Average FW RTT for 1 application packet*	Average FW client ACK (FW to phone and back) RTT
test1	0.3486	0.1489	0.2489	0.0012	0.2318	0.0269	0.1298	0.2578	0.1284	0.2270	0.0238	0.1252	0.2500	0.1251
test2	0.3117	0.1138	0.2129	0.0010	-	-	-	-	-	0.2267	0.0224	0.1083	0.2485	0.0764
test3	0.3091	0.1243	0.2184	0.0011	-	-	-	-	-	0.2252	0.0232	0.1237	0.2480	0.0782
test4	0.3100	0.1300	0.2203	0.0011	0.2357	0.0291	0.1305	0.2607	0.0749	0.2270	0.0228	0.1246	0.2490	0.0749
test5	0.3252	0.2308	0.2784	0.0012	0.2212	0.0297	0.1266	0.2465	0.0729	0.2257	0.0250	0.1243	0.2498	0.0815
test6	0.3077	0.2220	0.2653	0.0012	0.2384	0.0276	0.1302	0.2647	0.0731	0.2267	0.0239	0.1250	0.2498	0.0750
test7	0.3169	0.2287	0.2731	0.0014	0.2423	0.0267	0.1329	0.2661	0.0798	0.2277	0.0234	0.1247	0.2500	0.0799
test8	0.3427	0.2313	0.2873	0.0011	0.2461	0.0327	0.1381	0.2690	0.0726	0.2279	0.0236	0.1238	0.2511	0.0826
test9*	1.5325	0.4395	0.9973	0.0006	0.6043	0.0183	0.3789	0.8529	0.1051	0.2355	0.1044	0.1693	0.3393	0.0821
test10	1.5562	0.3558	0.9713	0.0008	0.2213	0.0225	0.1219	0.2205	0.0811	-	-	-	-	-

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

\*For phone test9 we have just a few packet sequences recorded properly for the Gn interface, and one instance of 1<sup>st</sup> packet retransmissions influencing the average values. For the FW trace, large average RTT is due to a case of 3 retransmissions having 15s RTT – without this instance, the average FW RTT for the application packet would be 0.2554s, i.e. similar to other phones.

**Table 37: Comparison of RTT statistics at different points**

Another view of these results, for comparing purposes, is to align specific RTTs (for first packet in sequence, second, etc.) by tracing points.

Name	Average RTT for first packet in sequence		
	Phone	Gn	FW
test1	0.3486	0.2318	0.2270
test2	0.3117	-	0.2267
test3	0.3091	-	0.2252
test4	0.3100	0.2357	0.2270
test5	0.3252	0.2212	0.2257
test6	0.3077	0.2384	0.2267
test7	0.3169	0.2423	0.2277
test8	0.3427	0.2461	0.2279
test9	1.5325	0.6043	0.2355
test10	1.5562	0.2213	-

**Table 38: Average RTT for 1<sup>st</sup> TCP packet in sequence, at different points**

For the first packet in sequence, we see that at the FW we already have around 230ms generated delay, regardless of the type of traffic. As mentioned before, in the section about the FW trace, analysis of other messages (second packet, „78B“ messages) leads us to a conclusion that the backbone itself generates around 20-50ms delay, while the rest may be attributed to server processing of the first packet in the sequence of TCP packets exchanged for one application packet.

On the Gn interface, we notice that the delay is similar (for some phones somewhat less than at FW, but that's because of small statistical sample at the Gn). Only for phone test9 we have larger delay in the core, but as explained before, this is because of one retransmission of the first packet with 3.3s RTT (and there are just 7 sequences in the sample).

Finally, looking at phone values, we see that the access part with the RNC generated around 100ms RTT, except for phones test9 and test10, where the access generated more than a second RTT. As explained before, this is due to the access network response to sporadic traffic, where these phones get random access channels, and even go to the Idle state.

We may also compare this access delay with the Gn values for client ACKs (**Table 28: TCP Gn interface RTT statistics from Shark traces**Table 28) – comprising the delay from the Gn to the phone and back – we see the match, a delay of around 100ms (70-130ms).

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

Name	Average RTT for second packet in sequence (main payload)		
	Phone	Gn	FW
test1	0.1489	0.0269	0.0238
test2	0.1138	-	0.0224
test3	0.1243	-	0.0232
test4	0.1300	0.0291	0.0228
test5	0.2308	0.0297	0.0250
test6	0.2220	0.0276	0.0239
test7	0.2287	0.0267	0.0234
test8	0.2313	0.0327	0.0236
test9	0.4395	0.0183	0.1044
test10	0.3558	0.0225	-

**Table 39: Average RTT for 2<sup>nd</sup> TCP packet in sequence, at different points**

For the second packet in sequence (carrying the main payload of nominal packet size), RTT at the FW is around 25ms, with the exception of phone test9 with 104ms, due to a triple retransmission with 15s RTT. At the Gn, average RTTs are generally of the same order (for test9, the Gn average is smaller due to a trace length of up to 5 minutes, not comprising the instance of 3 packet retransmissions that influenced the FW average). For phone RTTs, we notice a 100ms larger delay for bigger packets of phones test5-test8, and much larger delay for phones test9 and test10, with sporadic traffic, again indicating delay in the access for such traffic patterns. The delay in the access for 2<sup>nd</sup> packets, in case of sporadic traffic, is smaller than for 1<sup>st</sup> packets, as the 1<sup>st</sup> packets suffer the delay of accessing the network in case of Idle state, as explained before.

Name	Average RTT for 1 application packet		
	Phone*	Gn	FW
test1	0.5010	0.2578	0.2500
test2	0.4281	-	0.2485
test3	0.4373	-	0.2480
test4	0.4428	0.2607	0.2490
test5	0.5589	0.2465	0.2498
test6	0.5344	0.2647	0.2498
test7	0.5486	0.2661	0.2500
test8	0.5773	0.2690	0.2511
test9	1.9711	0.8529	0.3393

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

test10	1.9148	0.2205	-
--------	--------	--------	---

\*for phone averages, results from application reports are taken

**Table 40: Average RTTs for one application packet at different points**

Looking at average values for the application packets (i.e. the average of the sum of RTTs for first two packets sent by the client, initial TCP packet and the packet with main payload), this 4-way delay in the access is around 200-300ms for test1-test8, and almost 2 seconds for phones test9 and test10.

#### A.2.3.7 Server processing delay

In order to prove that the server processing delay for the 1<sup>st</sup> TCP packet in a sequence sent for one application packet is significant, around 200ms, as deduced from traces in previous sections, another round of measurements has been performed. Traffic traces were taken at the server side for TCP case, as well as phone traces.

Processing of server trace was somewhat difficult, as on the server side the Network Address Translation (NAT) is performed. All packets coming from test phones are seen coming from one IP address (internal address of the NAT gateway), to internal server IP address, different from one specified externally, but going to one, specified, port, designating our server application. Packets from individual phones can be traced by their inner TCP data – Len, TSval, TSecr.

The statistics was made for the whole communication of server application with 10 test phones, provided that if standard deviation is high, calculation by individual phones shall be done.

The statistics for trace taken at the server is the following:

Server trace, TCP	Average RTT [s]	Max RTT [s]	Min RTT [s]	St.dev. of RTT [s]
Server processing delay for 1st TCP packet	0.20051	0.29791	0.19019	0.00479
Server processing delay for 2nd TCP packet	0.00027	0.01809	0.00006	0.00023
Network delay for client final ACK	0.14277	9.81529	0.05925	0.16034

**Table 41: Delay statistics for server-side trace, TCP**

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

We see that the average delay of processing within the server for the first TCP packet sent for one application packet is around 200ms, with the standard deviation of only 4ms. This proves conclusions deduced in previous sections, that the server has some “wake-up” time for the first TCP packet, regardless of the traffic pattern of individual phones.

From the table above, we also see that the processing delay for the second TCP packet is mostly less than 1ms.

On the other hand, network delay for client final ACK varies a lot – this is the time between server’s PSH,ACK message acknowledging the main (2<sup>nd</sup>) packet, and the client’s final ACK, i.e. it contains delay on the route server-phone and back.

For comparison reasons, a new round of UDP measurements was performed, with trace recording on the server and on phones. Server trace was processed in a similar way as described for TCP case, by applying a filter and exporting relevant records to Excel. Further, it was processed similar as explained for UDP measurements in A.1, manually calculating RTTs i.e. server processing delay. The results are as follows:

Server trace, UDP	Average RTT [s]	Max RTT [s]	Min RTT [s]	St.dev. of RTT [s]
Server processing delay for UDP	0.00045	0.04646	0.00012	0.00080

**Table 42: Delay statistics for server-side trace, UDP**

It is clear that in UDP case server processing mostly has small influence on RTT, being 0.4ms in average with 0.8ms standard deviation, but there are some instances of high processing delay going up to 46ms.

#### A.2.3.8 Conclusions

Measurements in D5.3 were done in a modernized network, with enough resources, while measurements in D3.5 were performed in somewhat strained network. Main conclusions that can be taken from the analysis performed within D3.5 and D5.3 are as follows:

- The size of inter-arrival time (wait time between packets) influences RTT most strongly. This influence is due to inherent properties of the radio access network:

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

- Sporadic traffic with low throughput gets FACH/RACH channels – these channels are shared, with a collision risk, offering high access times.
  - Very large inter-arrival time results in UE going to the Idle state. Network is loaded with excessive signalling as for every new packet UE has to establish the RRC connection again.
- Packet size also influences the RTT, to a lesser extent. Large packets have bigger RTTs, and this influence is visible in the access part, while core network and the backbone do not show any.
- Server processing delay is about 200ms for the first packet in a sequence of TCP packets sent for one application packet (server wake-up time). Client application processing delay is negligible, less than 2ms.
- For the shown TCP flow:
  - Statistics for the first packet shows the influence of random access and RRC establishing for long inter-arrival times, and the influence of assigned access channel for shorter inter-arrival times, but does not show the influence of packet length, since the first packet is always 70 B long, while the main payload is in the second. If the first packet would be with main payload, average RTTs would be bigger for some phones. First packet RTTs also show server processing delay of avg. 200ms.
  - Statistics for the second packet shows the network delay once the UE is in RRC connected state (either Cell\_FACH or Cell\_DCH), i.e. shows the impact of assigned access channel, as well as the impact of packet length on delay. Packets are of nominal size plus TCP header.
  - Statistics for the client ACK may be taken for verification – for phone RTT, it is the server processing delay; for Gn and FW RTT, it represents two-way delay to the phone and back, of a small packet (66B), having in mind that this is the last message in sequence, so phones are for sure in the RRC Connected state
- For UDP, packets are exchanged more closely according to the time/size distributions. Packet length is increased for UDP header, and time between packets is the last RTT, maximally 3s, plus generated wait time. RTT statistics, with small remarks (see A.1.3), shows all relevant effects of the network.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

- Large number of UEs with M2M traffic, but also Online Gaming traffic, leads to 3G/HSPA network accessibility degradation, affecting even voice services

Relevant statistics (two-way delay) per parts of the network may be taken by looking at the difference of corresponding RTTs:

- for the backbone – FW RTT minus server processing delay
- for the GGSN, proxy, firewall – FW RTT minus Gn RTT
- for the access, including RNC and internal transport network to the SGSN and GGSN – phone RTT minus Gn RTT, comparable with Gn RTT for client ACK (client ACK RTT represents latency for a small packet when in RRC Connected state); FW RTT for client ACK should be taken into account for verifying Gn RTT, since Gn samples are small

Using HSPA network with enough resources, average two-way delay generated in different parts of the test path is as follows:

- backbone – RTT mostly 23-25ms; not selective concerning packet length and time between packets
- GGSN, proxy, firewall – few milliseconds, up to 8ms; not selective
- access, including RNC and internal transport network to the GGSN – without server processing delay, 75-120ms for Online Gaming simulations and 0.1-1.3 s for M2M simulations; highly sensitive concerning time between packets, and sensitive to packet length

#### A.2.3.9 Statistics to be taken for tuning up the testbed parameters

Should the same TGen application with TCP be used for the testbed, TCP statistics is to be used for comparison. With some remarks (see A.1), UDP statistics, showing all relevant network impacts, should be used if other traffic generator application is applied.

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1 V1.0</b>
------	-------------------	-----	--

### A.3 Real M2M application – illustrative example concerning measurements performed in D3.5 and D5.3

The example that follows is illustrative, stressing main problems with massive M2M deployment in HSPA network and proving the results obtained in D3.5, Test Case 2, low-network-resources case.

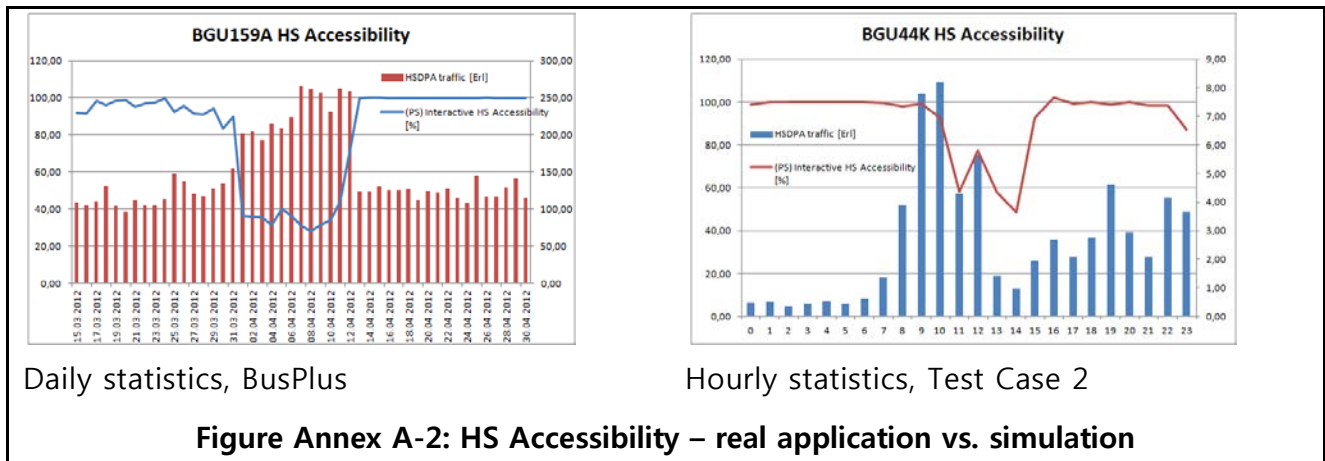
The subject real M2M application called BusPlus is an application deployed by public transportation company in Belgrade (GSP Belgrade). Telekom Srbija provided only the connectivity for this application, i.e. modems, without prior knowledge of the traffic patterns. Clients in buses send data on validated tickets and GPS position to the remote server, and the GSP company later uses this data for optimizing bus frequency on particular lines.

After the deployment, degradation of certain KPIs was observed in the network, and investigation showed that this happened during the night in the cells covering bus garages. Although with different traffic pattern than those simulated in D3.5, the application led to degradation of same KPIs – mainly accessibility, affecting even voice accessibility, as in D3.5.

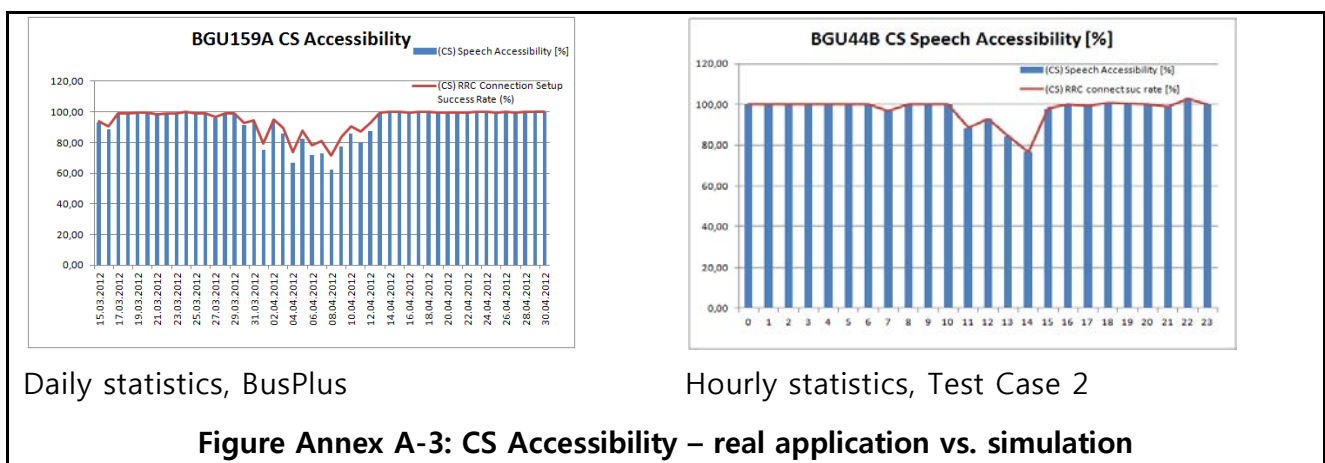
The investigation showed small traffic, but with a lot of connections. The problem occurred at night in the vicinity of bus garages as drivers forgot to put the devices in offline mode, so all clients were sending GPS update info every 30s, and certain garages host around 300 vehicles. 30s is enough time for client UEs to go to the Idle state in this network. So what happened is that all these buses had to establish the RRC connection again for each message. This application is not latency-critical, but the influence on the network is huge.

Even the modernized network (2ms TTI UL, more CEs in NodeB, more licenses for HS users) cannot support such massive number of connections and signalling.

Figure Annex A-2 shows, side-by-side, graphs of HS accessibility for BusPlus application and for Test Case 2. BusPlus statistics is given on daily level, and Test Case 2 statistics on hourly level. From 1<sup>st</sup> of April, when GSP switched modems to 3G only, until 12<sup>th</sup> of April, when they were asked to switch to GSM, in order to protect voice service on 3G, a significant growth of HSDPA traffic in Erl was observed, showing huge number of connections, as well as the related downfall of HS accessibility.



Next, as in Test Case 2, this influenced even the voice service, which is shown in Figure Annex A-3. The CS Accessibility downfall with BusPlus is mainly due to congestion on signalling channels, not due to lack of HW&SW resources. CS services are degraded, even though voice has priority. In Case 2 of D3.5, low-resources refers to low number of CEs, low number of licenses for simultaneous HS users, etc, and this was the main reason for rejecting new users to enter RRC Connected state.



For most critical cells optimization was done, reducing the number of codes assigned for HS traffic so the signalling channels could use them. This is a compromise between large number of connections of M2M users and HS traffic for throughput-demanding “regular” users.

A significant rise in uplink interference due to large number of users was also observed with BusPlus, but the traffic increase was relatively low compared to Test Case 2 – during

LOLA	Project N° 248993	WP5	Validation Results <b>for WP4</b> <b>Algorithms on Testbed 1</b> V1.0
------	-------------------	-----	--

simulation we had traffic patterns with higher datarate (Online Gaming plus M2M), while in BusPlus we had only a message every 30s.