



TAMPERE UNIVERSITY OF TECHNOLOGY

**RIKU KAURA-AHO**  
**SCALABLE IOT TRAFFIC GENERATION SYSTEM USING**  
**LTE NETWORK EMULATION**

Master of Science Thesis

Examiner: Professor Jarmo Harju  
Examiner and topic approved by the  
Faculty Council of the Faculty of  
Computing and Electric Engineering  
on 6 April 2016

# ABSTRACT

**RIKU KAURA-AHO:** Scalable IoT traffic generation system using LTE network emulation

Tampere University of Technology

Master of Science Thesis, 72 pages

May 2016

Master's Degree Programme in Signal Processing and Communications Engineering

Major: Communication Systems and Networks

Examiner: Professor Jarmo Harju

**Keywords:** cloud computing, IoT, load generation, LTE, network emulation, network virtualization, OpenStack, orchestration, testing, virtualization

The number of networked devices will grow exponentially in the coming years. It is estimated that up to 50 billion devices will be connected to the internet by 2020 and most of the growth comes from Internet of Things (IoT). Along with growth, new IoT devices will use network in more diverse ways and new specialized networking technologies are needed to fulfill new requirements.

Future networks are running more and more in programmable cloud environments which enable scalable provision of connections to new diverse devices and users. Though there are good estimations about the future, the practical solutions and standards are open.

This thesis will implement a tool called LoadGenerator to model large future networks. LoadGenerator uses LTE network emulation software, thus it can create an emulated 3GPP compatible network which can be connected to external LTE entities. LoadGenerator scales horizontally in cloud environment and a large scale cellular IoT network is presented as a case study. This proves that cloud environments can be used to run large scale network emulations with decent performance.

The modeling of future networks produces information which can be used in today's development work. Practical tests with a real size emulated network gives the possibility to try out new technologies which could solve the future problems. To test new scalable products, scalable testing methods are needed.

Telecommunications is developing fast and there are new technologies to keep eye on so that updated research information is available. Technologies, like IoT and cloud, are driving the development towards more programmable world. To assist with the change, this thesis presents network emulation as a tool, which combines the best features from software and hardware for more realistic testing also in large scale.

# TIIVISTELMÄ

**RIKU KAURA-AHO:** Skaalautuva IoT-liikenne generaattori LTE-verkkoemulaattorilla  
Tampereen teknillinen yliopisto

Diplomityö, 72 sivua

Toukokuu 2016

Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma

Pääaine: Communication Systems and Networks

Tarkastaja: professori Jarmo Harju

Avainsanat: IoT, kuorman generointi, LTE, OpenStack, orkestrointi, pilvilaskenta, testaus, verkkoemulaatio, verkkovirtualisointi, virtualisointi

Verkotettujen laitteiden lukumäärä tulee kasvamaan räjähdysmäisesti tulevina vuosina. On arvioitu, että jopa 50 miljardia laitetta on yhdistettynä internetiin vuonna 2020 ja suurin osa kasvusta tulee esineiden internetistä (IoT). Kasvun lisäksi uudet IoT-laitteet käyttävät verkkoja hyvin moninaisiin tarkoituksiin, jolloin tarvitaan myös uusia erikoistuneita verkkotekniikoita vastaamaan muuttuneita tarpeita.

Tulevaisuuden verkot toimivat yhä enemmän ohjelmoitavissa pilviympäristöissä, mikä antaa mahdollisuuden tuottaa yhteydet kasvavalle ja moninaistuvalla käyttäjäkunnalle. Tulevaisuudesta siis tiedetään melko paljon, mutta käytännön ratkaisuja ei ole vielä toteutettu eikä standardoitu.

Tässä diplomityössä toteutetaan kuormageneraattori, LoadGenerator, jota käytetään tulevaisuuden verkkojen mallintamiseen. LoadGenerator hyödyntää LTE-verkkoemulaattoria 3GPP-yhteensopivan verkon tai sen osan luomiseen, ja se voidaan kytkeä ulkoisiin LTE-elementteihin. Kuormageneraattori skaalautuu horisontaalisesti pilviympäristössä ja esimerkkinä mallinnetaan matkapuhelinverkkoa sekä suurta määrää IoT-laitteita. Tämä osoittaa, että pilvessä on mahdollista suorittaa suuria verkkoemulaatioita kohtuullisella suorituskyvyllä.

Tulevaisuuden verkkojen mallintaminen tuottaa tietoa, jota voidaan hyödyntää tämän päivän kehitystyössä. Käytännön kokeilut aidon kokoisessa emuloidussa verkossa antavat mahdollisuuden testata uusia tekniikoita, joilla tulevaisuuden ongelmia pyritään ratkaisemaan. Uusien skaalautuvien tuotteiden testaamiseen tarvitaan skaalautuvia testausmenetelmiä.

Tietoliikenteen kehitys on nopeaa ja kehitystä on seurattava, jotta uusista teknologioista on saatavilla ajantasaista tietoa. Uudet teknologiat, kuten IoT ja pilvet, muuttavat maailmaa entistä enemmän ohjelmoitavaksi. Muutoksen tukemiseksi tämä diplomityö esittelee verkkoemulaation työkaluna, joka yhdistää ohjelmistojen ja laitteiden parhaat puolet mahdollistaen realistisemmän testaamisen myös isossa mittakaavassa.

## PREFACE

This thesis is done for Nokia Networks. The creation of LoadGenerator began when I started as summer trainee in Nokia Networks in Espoo in May 2015. Janne Parantainen, my line manager, had the idea of the system and I started to implement it. The concept proved to be working already in the early phases but nothing similar was done before so a lot of time went on building, testing, and debugging the new environment.

Thesis process started in the Autumn 2015 and professor Jarmo Harju from Tampere University of Technology started as the thesis supervisor. With long academic experience professor Harju has been a steady source for comments that have made sure that this thesis has found its right form.

LTE Emulators development team has been great support for solving emulator related issues and they have given good comments about this thesis, too. Special thanks goes to Harri Povelainen for making so many patches and answering all the LTE and EMU related questions. Also, Juha-Matti Tilli has given very detailed feedback especially related to load generation and emulator performance.

During the year Tuomo Tikkanen has taught many Linux tricks, that have solved out multiple problems, and he has also built some of the support services and scripts for the environment. Petri Velin has created some nice visualizations for the demo front-end. Also, people in the Innovation Lab have provided a good environment for development by providing the LTE core elements and hardware to run LoadGenerator.

I want to thank all who have participated in the thesis process and the creation of LoadGenerator. Multiple elements and software have been combined together which has required the co-operation of many people. LoadGenerator started from the idea and there were no answers how to do the implementation. On the other hand, it is a dream for engineer to make ideas into reality which is our goal here in innovation programs.

Espoo, 24.05.2016

RIKU KAURA-AHO

# CONTENTS

<b>1. Introduction</b>	<b>1</b>
1.1 Goal and methods . . . . .	2
1.2 Thesis structure . . . . .	4
<b>2. Long Term Evolution (LTE)</b>	<b>5</b>
2.1 Architecture . . . . .	6
2.2 Interfaces and protocols . . . . .	7
2.3 Signaling and parameters . . . . .	8
2.4 Simulation and emulation . . . . .	12
2.5 LTE Emulators at Nokia . . . . .	14
2.6 LTE Emulators in LoadGenerator . . . . .	15
<b>3. Internet of Things (IoT) networking</b>	<b>18</b>
3.1 Current status and verticals . . . . .	19
3.2 Future estimations and network requirements . . . . .	20
3.3 Future 3GPP Releases . . . . .	21
3.4 Example protocols . . . . .	23
<b>4. System base used in the setup</b>	<b>25</b>
4.1 Current status . . . . .	26
4.2 Kernel-based Virtual Machine (KVM) and containers . . . . .	27
4.3 OpenStack open source cloud . . . . .	29
4.4 Hardware . . . . .	30
4.5 Networking stack . . . . .	31
4.5.1 Network virtualization . . . . .	32
4.5.2 OpenStack networking . . . . .	33
4.5.3 Offload functions . . . . .	35
4.5.4 Network performance . . . . .	36
<b>5. System installation and configuration</b>	<b>38</b>
5.1 Configuring hardware, network, and OpenStack . . . . .	40
5.2 Configuring guest instances . . . . .	42
5.3 LTE and EMU configuration . . . . .	43
5.4 Control network and scripts . . . . .	44
5.5 UE without real air interface . . . . .	46
<b>6. System usage and load generation</b>	<b>47</b>
6.1 Repository and support services . . . . .	47
6.2 Preboot Execution Environment (PXE) for host installation . . . . .	48

6.3	OpenStack Heat deployment . . . . .	50
6.4	Load generation . . . . .	52
6.4.1	Traffic modeling . . . . .	53
6.4.2	Traffic generation . . . . .	54
6.4.3	LTE transactions generation . . . . .	54
6.4.4	Measuring and visualization . . . . .	55
<b>7.</b>	<b>Tests and results</b>	<b>57</b>
7.1	U-plane load . . . . .	57
7.2	C-plane load . . . . .	58
7.3	Scalability . . . . .	59
7.4	Results . . . . .	61
7.5	Discussion . . . . .	62
<b>8.</b>	<b>Conclusions</b>	<b>64</b>
8.1	LoadGenerator as a tool . . . . .	64
8.2	Future work . . . . .	66
	<b>References</b>	<b>68</b>

## ABBREVIATIONS

10GE	Ten Gigabit Ethernet
2G	Second Generation
3G	Third Generation
3GPP	Third Generation Partnership Project
4G	Fourth Generation
5G	Fifth Generation
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
APN	Access Point Name
APP	Application
AWS	Amazon Web Services
C-plane	LTE control plane
CxO	C-suite Officers
D2D	Device-to-device
DES	Discrete-event simulation
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DPDK	Data Plane Development Kit
E2E	End-to-end
EMU	LTE Emulator
ENB	Evolved NodeB which is LTE base station
EPC	Evolved Packet Core
EPS	Evolved Packet System
Fixed IP	is OpenStack term for fixed internal IP

Floating IP	is OpenStack term for external IP address
Gb	gigabit
GB	gigabyte
GPRS	General Packet Radio Service
GRE	Generic Routing Encapsulation
GTP	GPRS Tunneling Protocol
GUI	Graphical User Interface
GUTI	Globally Unique Temporary Identifier
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
IMSI	International Mobile Subscription Identity
I/O	Input/Output
IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific, Medical
KVM	Kernel-based Virtual Machine
L1	Physical layer
L2	Data Link Layer
L3	Network layer
LAN	Local Area Network
LPWAN	Low Power Wide Area Network
LTE	Long Term Evolution
M2M	Machine-to-machine
MAC	Media Access Control



Mb	megabit
MB	megabyte
MCC	Mobile Country Code
ML2	Modular Layer 2
MME	Mobility Management Entity
MNC	Mobile Network Code
MQTT	MQ Telemetry Transport
MTC	Machine Type Communications
MTU	Maximum Transmission Unit
NFV	Network Functions Virtualization
NAT	Network Address Translation
NUMA	Non-uniform memory access
NIC	Network Interface Card
OVS	Open vSwitch
pCPU	Physical CPU
PDN	Packet Data Network
PF	Physical Function
PGW	PDN Gateway
PLMN	Public Land Mobile Network
pNIC	Physical NIC
PXE	Preboot Execution Environment
RAID	Redundant Array of Independent Disks
RAN	Radio Access Network
RAT	Radio Access Technology
RTC	Real-Time Communications

SAE	System Architecture Evolution
SCTP	Stream Control Transmission Protocol
SFTP	SSH File Transfer Protocol
SGW	Serving Gateway
SMS	Short Message Service
SR-IOV	Single Root Input Output Virtualization
SSH	Secure Shell
TAC	Tracking Area Code
TAI	Tracking Area ID
TAU	Tracking Area Update
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TEID	Tunnel Endpoint ID
TMSI	Temporary Mobile Subscriber Identity
ToR	Top-of-Rack
U-plane	LTE user plane
UEFI	Unified Extensible Firmware Interface
UDP	User Datagram Protocol
UE	LTE User Equipment
V2X	Vehicle-to-everywhere
vCPU	Virtual CPU
VF	Virtual Function
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNC	Virtual Network Computing

VNF	Virtual Network Function
vNIC	Virtual NIC
VoLTE	Voice over LTE
VXLAN	Virtual Extensible LAN
W3C	World Wide Web Consortium
WLAN	Wireless LAN
YAML	YAML Ain't Markup Language

## 1. INTRODUCTION

The number of networked devices will grow exponentially in the future and numbers as high as 50 billion networked devices by 2020 has been estimated [1]. New technologies have to be implemented to handle the growing network load and the different needs of the new devices. A scalable testbed, where large and realistic future networking scenarios could be created and studied, would be useful for innovation and implementation. First, the future must be estimated so that it can be turned into a system that is able to emulate it. What will be the main technologies that will power the future networks?

IBM study shows that cloud computing and services, mobile solutions, and Internet of Things (IoT) are the technologies that C-suite Officers think that are important in the near future. Results in Figure 1.1 clearly show that these three technologies rise above others [2], and they all are related to telecommunications.

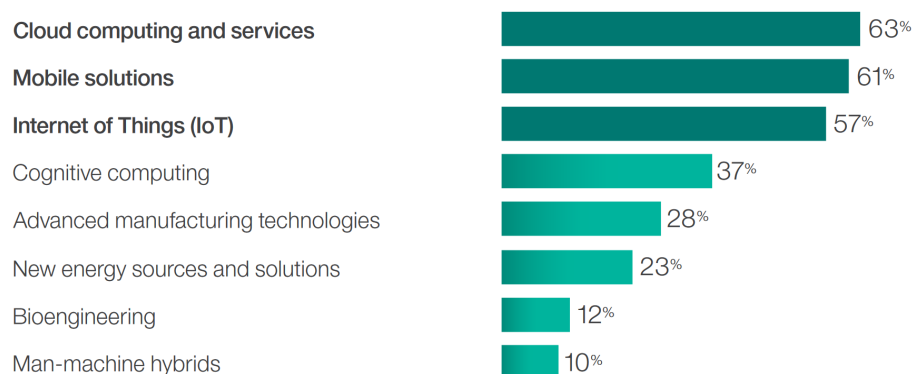


Figure 1.1: IBM study shows the important technologies in the near future according the C-suite Officers [2].

It is clear that the expectations for future networking are high, but it is unclear what actually will happen and when. The current state could be described that the information and communications technology industry is investing large sums to new technologies, e.g., IoT [3; 4], but the mass adoption is far away [5]. Though the growth and time estimations vary between the sources, it is clear that fundamental changes are to become with the future networks.

To narrow the thesis topic more to a practical and implementable level, Long Term Evolution (LTE) is selected to cover the mobility aspect. The recent advancements

in standardization show that LTE based technologies will be used for cellular IoT [6]. Cloud is the environment for future networks [7; 8], where also massive amount of programmable things can be ran. And last but not the least, IoT contains massive amount of different networked devices with a need for new types of connectivity.

These three technologies are combined and studied in this thesis to create a system that can be used to study future networks. The approach in this thesis is straightforward: A large emulated network consisting of millions of mobile devices and thousands of network elements is created in cloud. This system is named LoadGenerator and its main target usage is to represent part of a network which can be connected to real networks for load creation.

System like LoadGenerator has not been built before, thus also cloud environment has to be studied and configured to support large scale network emulation. Nokia's LTE Emulators software has not been scaled out in cloud environments as far as LoadGenerator targets are. These are the main open questions which need to be solved for successful implementation to reach the goal for large scale network emulation with LTE Emulators.

When the environment and LoadGenerator are set up, the research target is to emulate IoT devices and IoT optimized networks that can be measured and visualized.

## 1.1 Goal and methods

The goal for LoadGenerator is to generate events and traffic against real networks and elements. This can be achieved by using LTE Emulators developed by Nokia for Third Generation Partnership Project (3GPP) compatibility and open-source cloud environment for scalability. The system is a combination of emulated LTE elements, users and applications running in cloud. It can be connected and combined to any 3GPP compliant LTE element(s).

LoadGenerator can be compared to a large operator network and it can be used in product testing and verification. Also, demo creation is a key element to show and verify that technologies and products are working as promised. The system is not limited to IoT testing; but in this thesis IoT can be seen as an example use-case for the system.

Network emulation is a technique which combines the best sides of simulation and real-life testing. Software defined emulation running on top a of real working network provides controllable testing environment which is closer to real-life than pure simulation. In simulation everything is separated from real world which causes inaccuracy which can accumulate in large setups. In emulation real physical devices are used to keep the system closer to the real-world resulting more accurate results. [9].

Emulations are typically running in real-time and system functionality can be observed to demonstrate and study end-to-end (E2E) operation. Network emulation suits also very large test setups because separated elements can be multiplied and networked easily like in real world networking. These features are well suited for LoadGenerator and network emulation was chosen to be the modeling technique.

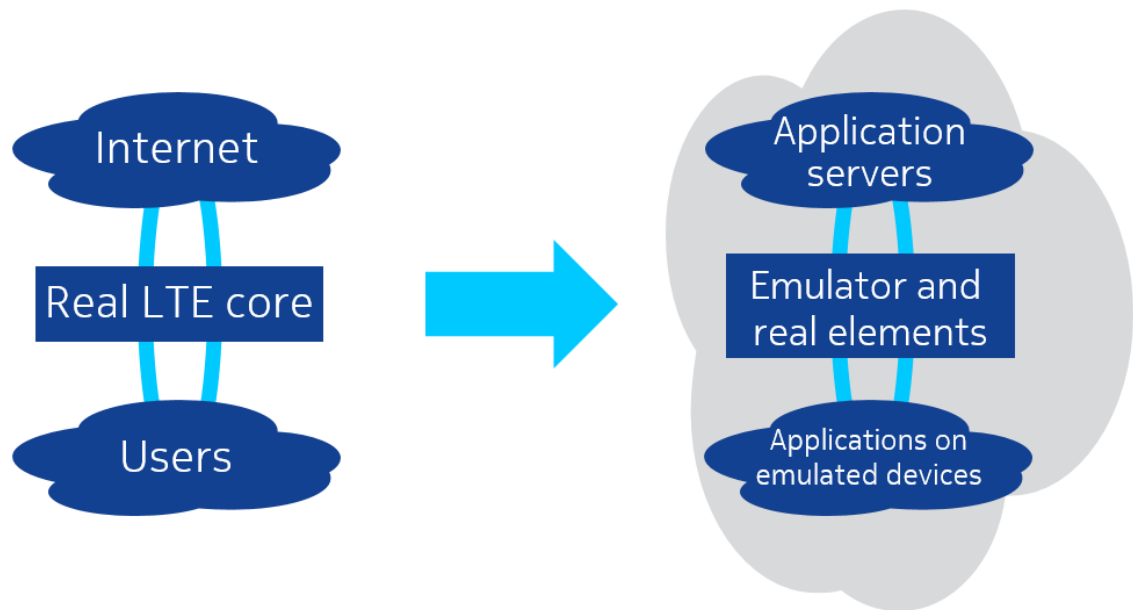


Figure 1.2: The goal for LoadGenerator is to present real like networks in a emulated test world running in cloud.

Scalable and fully programmable system enables a platform that can be used in large scale state-of-the-art and proof-of-concept testing. The main idea behind building the setup is to choose components and configurations so that goals will be met with existing open source components when available. This will speed up the development when less custom software implementations are used.

Some benefits in the emulated setup is the possibility to change LTE functions and settings freely without physical limitations. This enables agile and versatile testing. On the other hand, creating large network simulation or emulation is challenging without making any simplifications from real network. However, network emulation uses the full network protocol stack and thus inaccuracies are less likely compared to pure simulation. It can also reveal problems hidden in a real environment, like regular real life testing.

Work methods will focus on practical planning, developing and implementing the system. Theory from LTE, IoT and network simulation and emulation is also presented. Many of the solutions introduced are not optimal because the system will be a first of its kind, i.e., proof-of-concept, and product quality system is not a target. In overall, building a telecommunications grade cloud is a challenging topic.

Many of the components used in LoadGenerator are working in a cloud but are not designed for cloud environments. This thesis is not a detailed technical documentation about the system but it provides an overview of the components used and shows some example tests and results that the system can achieve mainly from the viewpoint of IoT. Another viewpoint for the thesis is to present some general problems related to telecommunications cloud.

## 1.2 Thesis structure

Thesis can be divided into three parts which are the protocols, theory, and definitions; system creation and installation; and the actual testing.

First chapters will introduce the protocols and entities which enable mobile LTE networking. LTE network provides connectivity for wireless mobile devices around the globe. Nokia's LTE Emulators is a software family that can emulate LTE network elements and is one of the main software components in this work. In this thesis the focus is on IP connectivity over LTE for IoT devices.

After LTE and IoT have been introduced, the thesis continues with a view to the building blocks of the system. LoadGenerator's environment consists of OpenStack cloud which is running on regular server and network hardware. Orchestration of the setup will be developed and introduced, and it is used for easy installation. In addition, hierarchical and horizontally scalable system built from separated components will need a control infrastructure which is created and used to carry out test scenarios.

After the basic installation of LoadGenerator has been developed and deployed, tests can be run. LoadGenerator can be configured to test all kind of cases and settings even outside of the LTE specifications. The downside of the system is the complexity, but the customizable system can be developed further to meet the new goals and scenarios.

This thesis does not present exact results from the point of LTE optimization for IoT. The created system can be tuned and optimized for performance testing but in the first phase, and in this thesis, the main idea is to explore and distribute this new concept of using large scale network emulation.

LoadGenerator is a practical implementation in a cloud with development software versions, thus it is hard to rule out external causes affecting the system performance. In short, the results are not comparable to real systems and production elements.

## 2. LONG TERM EVOLUTION (LTE)

LTE is the fourth generation (4G) of world wide cellular network. It is a successor in a long line of network engineering concepts including Global System for Mobile Communications (2G GSM) and Universal Mobile Telecommunications System (3G UTMS). LTE is a high speed pure packet access network which has evolved from UMTS but large differences exist between the generations. In this thesis LTE network will be running with only the elements that are needed for the basic IP connectivity. More comprehensive description for LTE can be found from many books, for example [10].

The LTE standardization comes from collaboration organization called 3GPP in form of Releases. The first version of LTE was frozen in Release 8 in 2008. As of May 2016 Releases up to 13 are frozen and Release 14 is under development. Frozen Releases mean that the list of items is decided but the details of some items are not yet finished.

Releases are being worked on parallel so that there are clear and stable milestones for developers to work on. Thus, the actual products and implementations based on the standards can be developed while last details are worked on and finished. Complete list of Releases and their statuses is available from 3GPP [11].

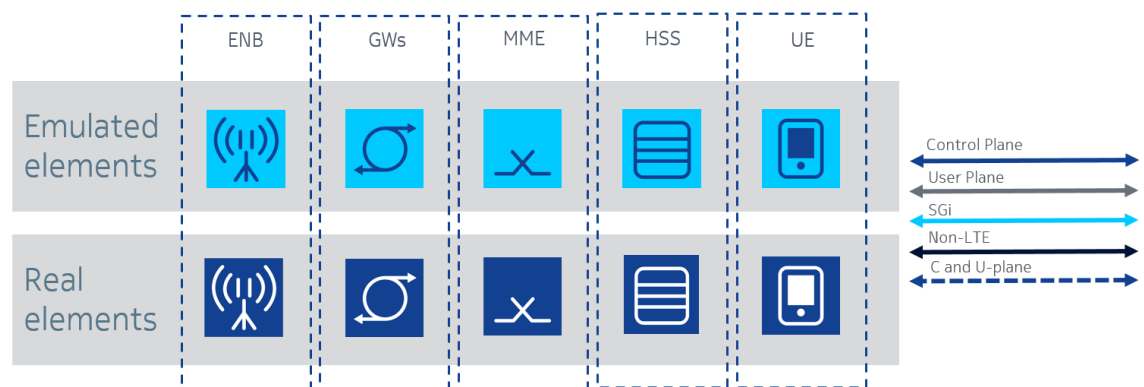


Figure 2.1: Explanations for the symbols and colors in use. Dark blue will be used for real elements and turquoise for emulated elements.

Before going into detailed networking figures, the figure notation is explained in Figure 2.1. These symbols and colors will be used consistently throughout this thesis.



## 2.1 Architecture

The main target in LTE is to provide high speed packet access for subscribers. The basic architecture of LTE is divided to two bigger parts which are Radio Access Network (RAN) and Evolved Packet Core (EPC). LTE RAN is often called with Evolved Universal Terrestrial Radio Access Network (E-UTRAN) to highlight the difference between the RAN generations.

To be precise, the term LTE refers to a radio technology, but it is often used to describe the whole system. The terms System Architecture Evolved (SAE) or Evolved Packet System (EPS) are better when radio is not in the focus. Nevertheless, this inaccuracy exists also in the name of LTE Emulators, and also this thesis uses term LTE to describe the whole system. In overall, all of these terms are pointing to same architectural compatibility with a bit different viewpoint.

In this work the focus is more on EPC. In pure LTE network, there are the following minimal elements to provide IP connectivity for the subscriber: Evolved NodeB (ENB) that acts as a base station, User Equipment (UE) which presents the end-user device, Mobility Management Entity (MME) which handles most of the control signaling work in the network, Home Subscriber Server (HSS) for authentication and subscriber database, Serving and Packet Data Network Gateway (SGW and PGW) for IP and service access. These LTE elements are drawn in Figure 2.2.

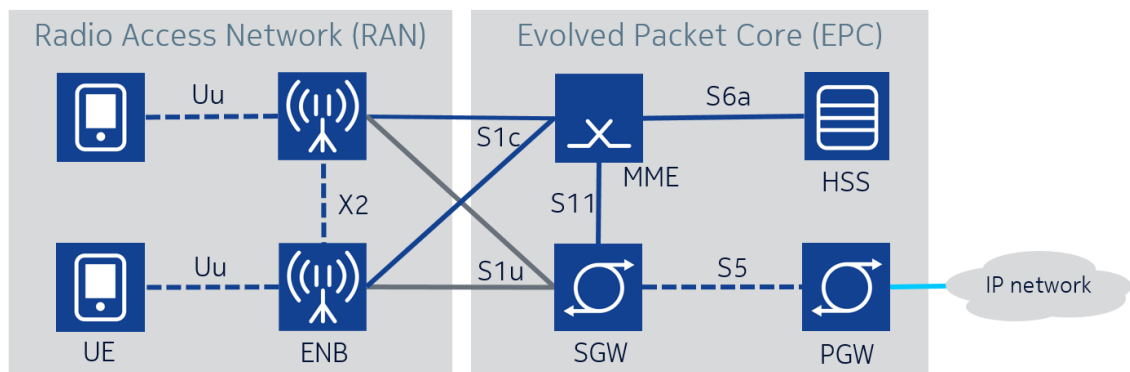


Figure 2.2: Minimal LTE architecture elements and interfaces with two UEs and ENBs. The network is scaled up by adding more RAN and EPC elements.

Real LTE networks are more complex because they are supporting connections to previous cellular generations and provide multiple services for the end-users, operators, and hardware vendors. The connection to previous cellular generations is needed to accomplish handovers between radio access technologies (RAT) for extended coverage and services. For example, Voice over LTE (VoLTE) is standardized but still voice calls are mostly going through GSM and UMTS. These connections and extra elements are not drawn or explained further here but, in short, they are similar to any other elements in the LTE network. They have standardized functions

and interfaces, and they can be connected also to LoadGenerator if needed.

## 2.2 Interfaces and protocols

The following section will introduce the interfaces and protocols connecting LTE elements in more detail. Understanding these protocols is important because we are emulating these interfaces and protocols. Resolving network issues in LTE is quite straightforward when line conversation can be compared to specifications and standards.

Table 2.1: LTE interfaces that are used to provide basic connectivity for UEs.

	ENB	MME	HSS	PGW
UE	Uu	-	-	-
ENB	X2	S1c	-	-
MME	S1c	S10	S6a	-
SGW	S1u	S11	-	S5

LTE entities are connected to each other with special interfaces which are listed in Table 2.1. Typically, S1c is used to connect ENB to MME and S1u to SGW, S11 connects MME and SGW, S6a is between HSS and MME, S5 between the PGW and SGW, and X2 is used between ENBs. These interfaces are also shown on Figure 2.2. Interfaces connecting ENBs and EPC elements are typically running on top of a fixed IP network.

Traffic in LTE networks is divided to user and control plane (U-plane and C-plane). U-plane carries user packet data through the LTE network. C-plane is used in signaling between the entities to enable connectivity around the globe for mobile devices. Signaling means that the elements have to instruct each other to create, modify and delete connections in the network. C-plane and U-plane are separated so that U-plane has short path through the network for lower latency and better throughput.

Stream Control Transmission Protocol (SCTP) is a transport layer protocol and it is designed to carry Public Switched Telephone Network (PSTN) signaling [12]. It provides reliable transfer like TCP (Transmission Control Protocol) [13] but it is message oriented like User Datagram Protocol (UDP) [14]. It more resilient, secure and redundant than TCP and UDP. Thus, SCTP is used with LTE C-plane messages, e.g., on S1c and S6a interfaces where large amount of signaling messages have to be exchanged reliably.

LTE U-plane is transported with GPRS Tunneling Protocol (GTP) which is running on top of UDP. GTP is divided to GTPv1-U [15] on U-plane and GTPv2-C [16] on C-plane. GTPv2-C is used on S11 interface where MME and SGW are changing messages. GTPv1-U is used on S1u and S5 to encapsulate the end-user packet data.

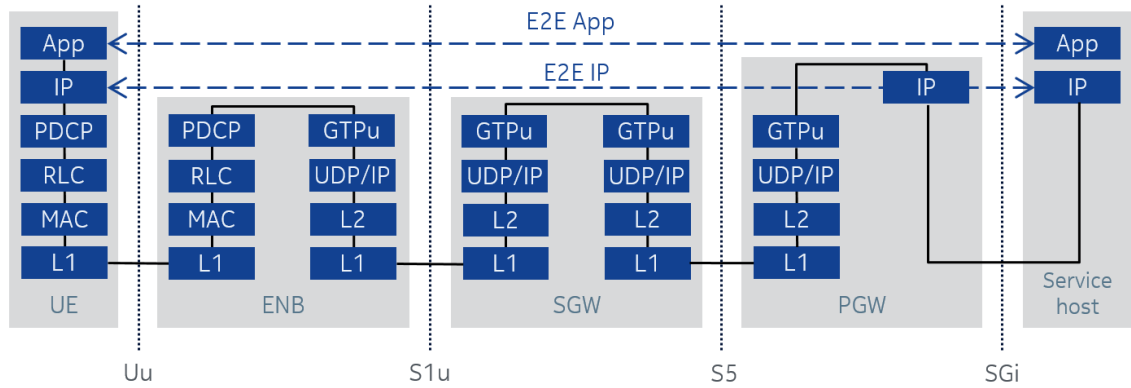


Figure 2.3: LTE user plane protocol stack with end-to-end connection.

Diameter is used on S6a between MME and HSS [17]. It is an authentication, authorization and accounting protocol which are the same functions that HSS is part of. HSS acts as a server for MME.

X2 protocol connects ENBs to neighboring ENBs. This enables handovers between the base stations without interruptions and also data exchange between ENBs. Uu interface is the physical radio interface between ENB and UE. Physical layer (L1) air interface is not emulated in LTE Emulators but air interface related parameters can be manually changed.

### 2.3 Signaling and parameters

Now when interfaces and protocols of LTE are known some signaling scenarios are presented to familiarize how the elements communicate. These signaling cases are also used in the emulated system.

Before the UE can attach (LTE term for connect) to the network all the elements and interfaces have to be up on the network side. MME initiates S6a connection to HSS. ENB initiates S1c connection to MME. Also large amount of configuration parameters have to match. 3GPP networks have always been strict on security and authentication so multiple identification parameters and settings are in use.

Figure 2.4 presents the identification codes in LTE. It also shows how they are used and what entity is provisioning them. Some of the codes are provisioned by the elements dynamically, whereas others are more static to identify logical divisions. These IDs also define the theoretical limits for scalability because most of them have fixed length. For example, MME-S1AP-UE-ID and TEID are 32 bit integers. Thus, the maximum number of sessions, that MME and GW can theoretically support, is  $2^{32}$  which is approx. 4,3 billion which is more than any real system can support.

However, not all parameters have so much capacity. For example, Aggregate Maximum Bit Rate is a 32 bit value and has a resolution of 1 bit/s [19] meaning that the maximum bit rate is 4,3 Gbit/s, whereas LTE-Advanced has downlink rates

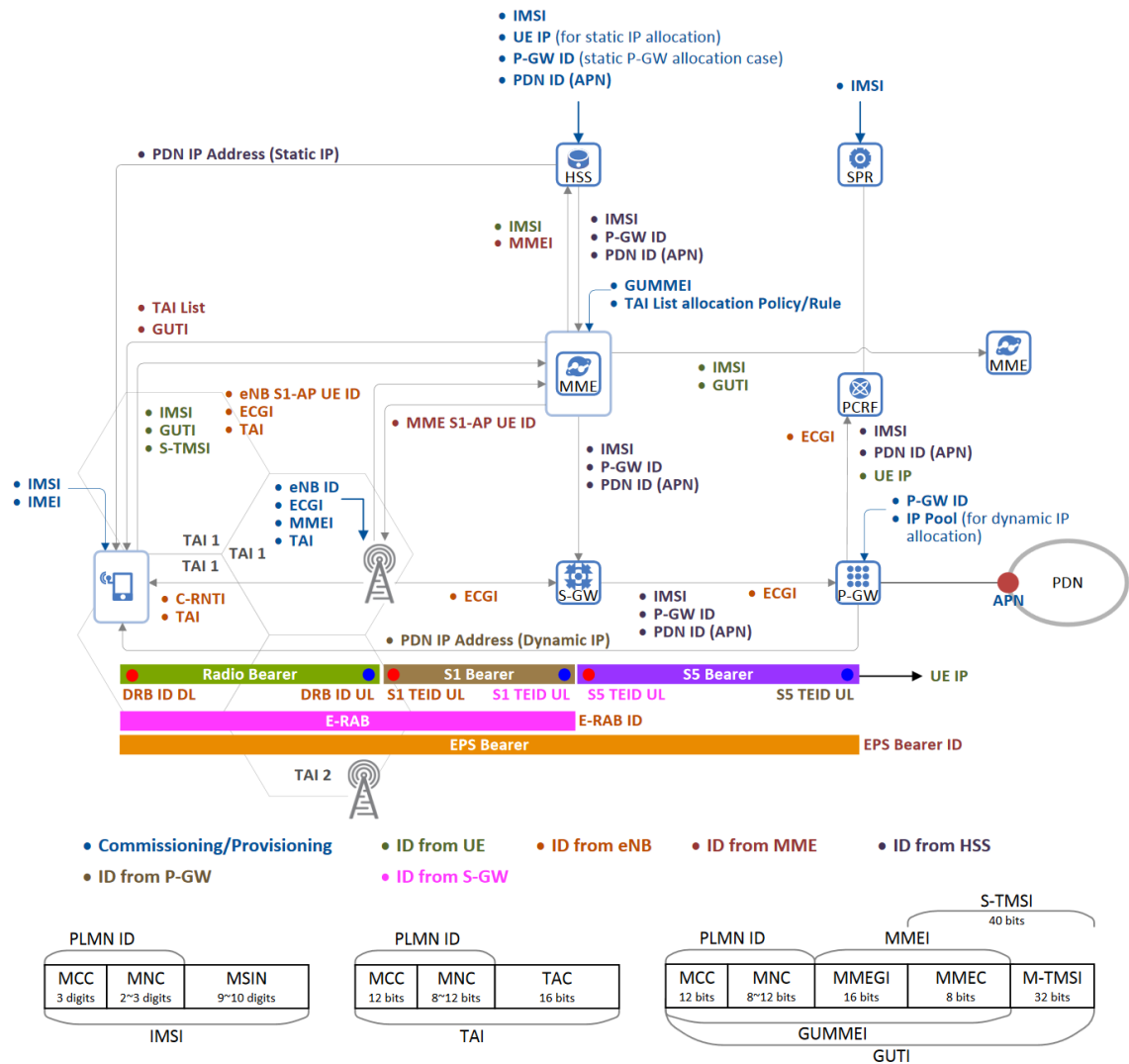


Figure 2.4: LTE uses multiple identifiers to identify elements locally and globally. Modified from [18].

up to 3 Gbit/s [20].

Mobile Network Code (MNC) and Mobile Country Code (MCC) will together form the Public Land Mobile Network (PLMN) code. This is also drawn in Figure 2.4, and PLML is used to separate different countries and operators from each others. On lower level, Tracking Area ID (TAI), which is formed from PLMN and Tracking Area Code (TAC), is used to physically map the location of the UE globally so that the network can reach the UE when needed. If these codes do not match between LTE entities, connections wont be accepted. In other words, these codes define the division to different networks.

International Mobile Subscription Identity (IMSI) will identify every subscriber in 3GPP network. IMSI consists of PLMN code and is filled to the length of 15 digits. Local codes are used after the connection has been made to hide the global

identity and prevent tracking. Temporary Mobile Subscriber Identity (TMSI) works in the area controlled by MME. Globally Unique Temporary UE Identity (GUTI) contains the PLMN code, MME code and M-TMSI to identify the MME and UE globally without revealing the original IMSI.

Initial attach procedure for UE is in Figure 2.5. Example corresponds to the emulated case which is the minimum required to set up the connections. Prerequisites are that MME and HSS have Diameter connection and ENB and MME have set up S1. From client-server point of view MME has to know HSS address and ENB has to know MME address. In real radio system UE has to find the ENB radio signal but this phase is skipped in this thesis. Also SGW and PGW have to be available for upcoming connections.

In the following scenario it is supposed that UE is not existing in the network, i.e., UE has no GUTI and IMSI is detached in the network, so no extra handovers or updates are made during the attach. In real life, UE is mobile and handovers are used to switch the connection to on top of the entities which can serve the UE best. Also roaming can be used to gain access via different operators but these scenarios are not explained here. Also the detailed operations inside entities are not explained.

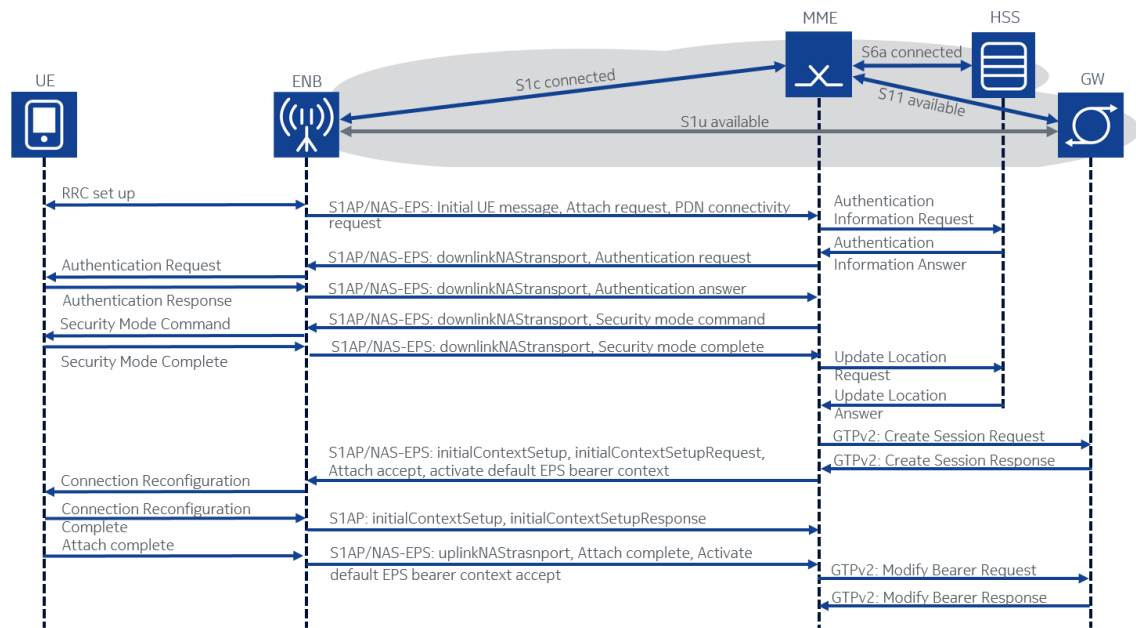


Figure 2.5: LTE signaling in clean and successful attach.

In Figure 2.5 UE is trying to attach to the network with Packet Data Network (PDN) access. There are no sessions or bearers for the UE in the network, thus the full path has to be signaled "open". This requires co-operation on multiple entities so that all handshakes are made securely. When parameters are not correct or no resources are available, the attach will fail.

First the UE connects to ENB which terminates the Uu interface, i.e., ENB is the endpoint for the protocol. ENB gathers the necessary information from the UE requests and starts the attach procedure to EPC. *Initial UE message*, *Attach request* and *PDN connectivity request* are sent from ENB to MME.

Based on the requests MME sends an *Authentication Information Request* to HSS to confirm that the UE is allowed to connect. If HSS sends an positive *Authentication Information Response*, MME sends *Authentication request* to ENB which passes it to the UE and UE replies with *Authentication Response*. MME verifies the response and sends *Security Mode Command* to UE and UE replies *Security Mode Completed*. MME updates the UE location to HSS which gives more subscriber details in response. If failure happens at any point MME sends Detach request to elements involved and the UE will be detached from the network.

When previous steps have been completed, there is enough information to open the GTP sessions to GWs. The GW information can be retrieved from multiple places. UE can request for specific Access Point Name (APN) in the initial attach or HSS can provide the APN value. MME then converts the APN name typically via Domain Name System (DNS) query to an address where the SGW and PGW are located. MME sends GTP *Create Session Request* to SGW via S11 interface and based on the request SGW opens the connection and retrieves the UE IP address from PGW.

If GTP creation is successful in SGW and PGW, SGW responds to MME with *Create Session Response* which includes the details that are needed to finish PDN connection set-up. MME passes these details to ENB and UE which reconfigure themselves according the details. UE will answer with complete messages which are passed to MME. MME then request to update the bearer on SGW with *Modify Bearer Request* and SGW replies with *Modify Bearer Response*. U-plane connection is established and UE has an IP address and connectivity trough ENB, SGW and PGW to external IP networks and other systems.

Source	Destination	Protocol	Length	Info
ENB	MME	S1AP/NAS-EPS	134	id-initialUEMessage, Attach request, PDN connectivity request
MME	HSS	DIAMETER	270	cmd=3GPP-Authentication-Information Request(318)
HSS	MME	DIAMETER	350	SACK cmd=3GPP-Authentication-Information Answer(318)
MME	ENB	S1AP/NAS-EPS	126	id-downlinkNASTransport, Authentication request
ENB	MME	S1AP/NAS-EPS	142	SACK id-uplinkNASTransport, Authentication response
MME	ENB	S1AP/NAS-EPS	102	id-downlinkNASTransport, Security mode command
ENB	MME	S1AP/NAS-EPS	138	SACK id-uplinkNASTransport, Security mode complete
MME	HSS	DIAMETER	274	cmd=3GPP-Update-Location Request(316)
HSS	MME	DIAMETER	1250	SACK cmd=3GPP-Update-Location Answer(316)
MME	SGW	GTPv2	243	Create Session Request
SGW	MME	GTPv2	145	Create Session Response
MME	ENB	S1AP/NAS-EPS	258	id-InitialContextSetup, InitialContextSetupRequest, Attach accept, Activate default EPS bearer context request
ENB	MME	S1AP	122	SACK id-InitialContextSetup, InitialContextSetupResponse
ENB	MME	S1AP/NAS-EPS	126	id-uplinkNASTransport, Attach complete, Activate default EPS bearer context accept
MME	SGW	GTPv2	89	Modify Bearer Request
SGW	MME	GTPv2	92	Modify Bearer Response

Figure 2.6: Wireshark packet capture from MME showing successful UE attach.

Figure 2.6 shows a real packet capture from MME ports handling the connections to HSS, ENB and SGW. Note that the entities which do not have direct connection

to MME are not present in this capture. In other words, they are communicating through other entities, e.g., ENB is acting on behalf of an UE. Packet captures are useful in network debugging because the conversation between the elements is strictly standardized and readable when protocols are familiar. In short, MME keeps track about connection states and makes the required operations to keep the network aligned.

Detach is basically the reverse attach. Some element in the network initiates the detach procedure, typically UE sends a detach request which is then forwarded to MME. MME then signals all the elements relevant that the created connection will be deleted by sending Delete Requests for sessions that were created for that specific connection.

Handovers between ENBs and idle stages are another typical scenario. In handover situation mobile device has moved so that it has better connectivity through some other ENB or some other core entity. Handovers enable "jumping" from element to another without connectivity losses. In real world handovers are mainly based on monitoring radio signal powers and error ratios to find the best cell for the UE. ENBs then have certain parameters, like tracking area IDs, which can be used to divide the physical network into logical areas which are served by specified core entity.

Idle stages are controlled by ENB which sets UE to idle if it has not sent data for a while. In idle stage no U-plane data can be transferred but the connection can be activated by paging from network side or by request from the UE itself. Paging is technique where MME sends signal via ENB to UE that network has something for the UE. In both cases, UE then sends *Service Request* to re-activate connection. UE locations in the network are periodically updated also during idle and the procedure is called Tracking Area Update (TAU).

This section introduced some basic LTE signaling. It can be seen that signaling in current cellular networks is complicated. It has many specific features which are not needed for all devices, especially from IoT point of view. This makes LTE a complicated system where a lot of resources are needed to keep the network running. Overhead signaling can use more bandwidth than the payload itself if low data rates are used. On the other hand, U-plane is quite simple and provides good performance. Thus, IP networks and client applications can be used more extensively than in previous cellular generations.

## 2.4 Simulation and emulation

Network simulations can be used in many situations from research to network debugging. Telecommunication suits simulations in a way that the packets and protocols are well defined in specifications, so they can also be simulated by following the

same details. In other words, network simulation and real life are typically quite close to each other and many times the algorithms found in simulations can also be implemented in real life.

Another thing that helps with simulations is that the network problems are already divided into smaller problems – into multiple protocols and layers which can be examined also separately. On the other hand, simulating the full stack becomes more and more problematic when layers are added, and it may not be enough to study only one part of the system. The end-to-end functionality is important for the network user, not a single protocol.

Typically simulations are mathematical models of real system and thus they are simplified from the real world. The benefit in simulation is that it can be run on a single PC without specific hardware - pure calculations without external elements. Typically discrete-event simulation (DES) is used in network modeling. [9]

In DES, events occur in a logical time order which differs from real networks where delays cause events happen at random times. Time, and thus propagation, is hard to simulate realistically which can lead to too optimal solutions. Some real-life factors may not exist or they can differ somehow in the simulation. This leads to inaccuracy and possible failures might appear in real implementation which did not exist in the simulation. [9]

*Ns* (open source) and *OPNET* (proprietary) are examples of software used in network simulations. They both support wide range of protocols and can be used to create simulations with detailed configurations. In this thesis the networking stack is customized and quite complicated because LTE networking is running on top of cloud networking. This leads to a situation where accurate simulation is not available and it would be very hard to implement.

Another issue with pure simulation is often the lack of real time execution and external connectivity. These features are needed if real networking elements are connected to the test setup. The goal in this thesis is to create load for real elements, so plain simulation software is not enough. LoadGenerator has to act like real elements would do.

Emulation inherits features from simulation and real world testing. In network emulation, some of the real devices are replaced with customizable virtual versions which have the full networking stack available, and they are typically running in real time. Simulation of time is not a problem when real network and propagation is used.

Emulation is also preferred on very large setups. Simulation algorithms have often difficulties to scale above certain point. In network emulation every element of the network can be existing in its own operational unit which can be run in parallel for scaling just like more ENBs and UEs can be added to a real network. This



means that emulation can scale further than simulation. Thus, accurate results can be gathered also from large test setups.

The targeted system in this thesis is clearly not only a simulation. The system will be connected to a real network and all the real-like networking elements are existing in a form of virtualized functions. They all have full connectivity and real time execution, but the system also has features like simulators, e.g., the environment can be modified and configured quite freely.

With these definitions and properties emulation is better suited for LoadGenerator than simulation. The term emulator also matches with the features of LTE Emulators, like the name suggests.

Both terms, simulation and emulation, are used in this thesis and it easy to mix them. The preferred term can depend from the viewpoint. In LoadGenerator the elements are emulated and in a big picture the emulated setup creates an artificial testbed. None of the elements can tell the difference if some part of the setup is real or emulated. The generation starts from real applications and flows through a 3GPP compatible LTE network in real time.

What happens on top of LoadGenerator is another case. Network user is often a unique human whose behavior has to simulated with some model: what applications will be used and how much traffic will be generated, is left open. Basically nothing prevents installing a real use-case application, e.g., some sensor software with random input data to send it through the system to test how the core handles the load. In short, devices can be emulated but human behavior is simulated.

LoadGenerator's target is to emulate LTE elements and create networks which can be connected like any other LTE network or element(s). This ability can be used to create a large network which then can be studied. The end-user application can be run on top of regular network stack meaning that multiple test scenarios can be created on top of LoadGenerator.

## 2.5 LTE Emulators at Nokia

LTE Emulators (EMUs) is a Nokia internal tool for emulating LTE entities. Emulators family covers all LTE entities from UE to SGi interface. Thus, emulated UE can be connected to the Internet through fully emulated LTE network and there is a support for both C-plane and U-plane traffic. The emulators are 3GPP compliant but the features differ from real entities used in production. The emulator entities are named like they are specified in 3GPP standards.

EMUs support low level signal tracing and macros for monitoring and creating events. These architectural features will limit how many connections and how much throughput EMU can support. Also resource usage can be high. Network expansion can be done by adding more parallel elements which can be mixed with real elements

to support larger loads.

Another limitation is that Layer 1 air interface is not emulated. However, it is simplified so that normal EPC related signaling is working and U-plane traffic can be transferred. EMUs also include performance test variants where functions are optimized to generate large amount of specialized load. Letters *PT* in the entity name stand for performance test variant.

EMUs operation is not explained in detail in this thesis. In short, EMUs can perform the same functions as real LTE entities with the limitations mentioned. It can be used to replace any slice of LTE network. This means that the whole network can be created fully emulated and RAN side can be horizontally scaled when real EPC is used.

## 2.6 LTE Emulators in LoadGenerator

LTE Emulators and LoadGenerator can be used to test multiple configurations. The main task for EMUs in this thesis is to emulate large amount of base stations (ENBs) and end-user devices (UEs). The target is to have millions of UEs and thousands of ENBs which can load real EPC elements. This scenario can be used, for example, to examine large IoT networks based on LTE.

Figure 2.7 shows an example configuration how the emulated and the real entities are connected. ENB(s) and UE(s) are emulated in the LTE EMU VM (Virtual Machine) and U-plane payload is generated in the UE VM. The U-plane insertion is an EMU feature which is not related to LTE specifications. EMUs can be run with or without U-plane source and by using this modular architecture there is an opportunity to connect different environments together which can sometimes be useful.

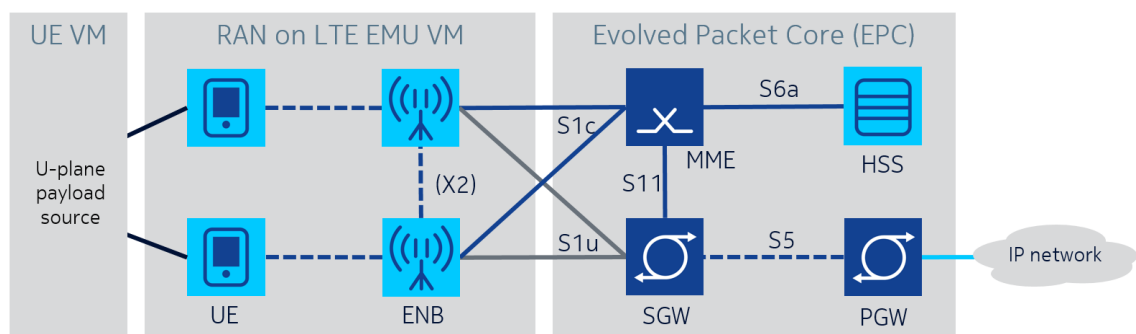


Figure 2.7: Example configuration and its connections.

In this thesis the end-user applications are running on a regular Linux distribution which is connected to an Ethernet broadcast domain. The Ethernet traffic, which is flowing between the UE VM and the LTE EMU VM, is transformed into LTE traffic by the EMU. In practice, one virtual network card (vNIC) is mapped to one LTE

UE. A vNIC is like any other network interface and any process in the operating system can use it.

LTE attach and detach are mapped to Dynamic Host Configuration Protocol (DHCP) Request and Release, and emulated UE holds the LTE specific information. Thus, basically any operating system or device capable of using DHCP is able to act as as LTE UE in this system and generate payload into the system. This mapping is drawn in Figure 2.8 which is compared to a real scenario. MME(s) and GW(s) can also be emulated but they are not used because they can not reach our targeted network size. On core side real MME and GWs are needed always to create LoadGenerator in large scale.

When the goal is to generate load against real core (MME and GW), EMU performance variants are useful. HSSPT is a simplified version of an HSS. It's minimized to only do necessary operations requested by other elements. In practice it only authenticates UEs and gives the UE profile to MME. This way HSSPT has a good performance to handle large amount of requests coming from real MME.

RANPT does the same to ENB by removing most of the extra signaling. These simplifications are close to the planned IoT optimization where network will be made simpler and more energy efficient by removing extra signaling. More on IoT optimization for LTE is presented in Chapter 3. If full LTE functionality is tested, normal (non PT) versions are used from the EMU. This will provide full 3GPP compatibility but reduces performance when more resources are used per ENB and UE.

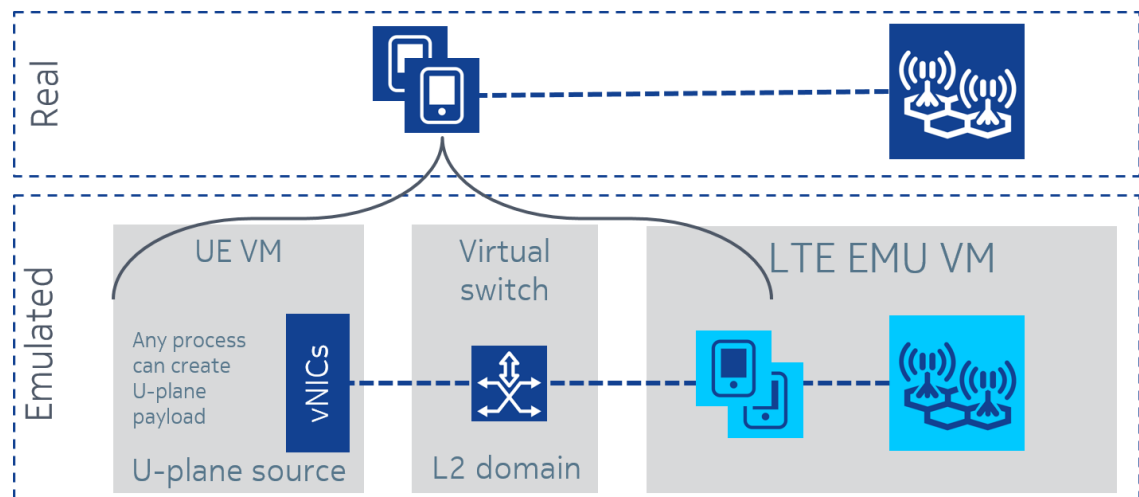


Figure 2.8: In the emulated setup any process can generate load into normal Ethernet domain which is then converted into LTE compatible form.

In this system Ethernet provides data link layer (L2) connectivity. One Ethernet domain is used to contain up to 5000 sources of U-plane which all have their own vNIC. This L2 Ethernet domain and entities connected to it can be seen analogous

to L1 RAN cell which has  $n$  UEs and  $m$  ENBs. L1 air interface is not emulated but the needed variables and procedures are available in emulated UE to describe changes on RAN so that full functionality can be tested from the core point of view.

For example, transmit and receiver powers are not simulated nor measured actively but handovers can be manually triggered. One L2 Ethernet domain is isolated into a VLAN where the cloud networking comes in to scale the setup further. Cloud networking with the virtual switching in Figure 2.8 is explained in more detail in Section 4.5.

One thing to emphasize is that EMU is consisting of processes which are running on a regular virtual machine. EMU processes don't have support for multi-threading but in a large setup the scalability comes from multiple deployments and processes instead of cloudified software.

### 3. INTERNET OF THINGS (IOT) NETWORKING

Internet of Things (IoT) has been in the center of hype for a couple of years. The term has topped Gartner's yearly hype cycle of emerging technologies two times in a row in 2014 and 2015 [21; 22]. Gartner estimates that IoT will reach plateau of productivity in 5 to 10 years meaning that even a decade can pass before IoT is widely accepted and used. But what is actually happening?

Definition for IoT is not exactly clear because IoT covers enormous amounts of different devices, applications and even social changes. IEEE is currently working on the definition for IoT and has found that many definitions are made and they usually lack in accuracy. IoT-like devices are not a new thing and many other terms have been used before IoT. [23]. Term Machine Type Communications (MTC) is also often used.

Nowadays term IoT is used almost everywhere which can be misleading when the same term is used with very different devices and capabilities. Loose term is easy to use and everyone has some idea what is a thing and what is the Internet. But from a technical point of view the definitions are problematic. What are the capabilities of a thing? What are the requirements to be connected to the Internet?

Terms that are close to IoT are also in use. Term machine-to-machine (M2M) emphasizes that things can communicate with each other. Device-to-device (D2D) and vehicle-to-everywhere (V2X) are similar and they will have many new applications. Term mesh or wireless sensor networks have also been used earlier but nowadays wireless sensors are clearly under IoT. This shows that IoT can be separated to multiple verticals depending on the use case.

Internet of Everything (IoE) is a term used by Cisco [24] and it gives understandable definition that everything is connected to Internet. Generally it can be seen that IoT is the phase where the number of networked devices and device types will grow exponentially. Further in the future this should lead into a situation where everything is connected – like IoE describes.

For telecommunications industry IoT means great changes because *things* differ from the current mobile devices. There are massive amount of things and they have to be simple, cheap and have long battery life which is quite the opposite to the current cellular devices, e.g., smart phones. This leads to new standards and

technologies which are currently forming up. IoT is a big phenomenon and it will happen but the question is when and how?

### 3.1 Current status and verticals

In this section the focus will be on cellular IoT networking and standardization which enables secure connectivity around the globe and prevents the incompatibility between the systems. Thus, global standardization accelerates general acceptance and usability of IoT, too. Also verticals are presented to summarize how IoT will be used.




	<b>SIGFOX</b>	<b>LoRa</b>	<b>5G (targets)</b>
			
Range (outdoor) MCL	<13km 160 dB	<11km 157 dB	<15km 164 dB
Spectrum Bandwidth	Unlicensed 900MHz 100Hz	Unlicensed 900MHz <500kHz	Licensed 7-900MHz shared
Data rate	<100bps	<10 kbps	<1 Mbps
Battery life	>10 years	>10 years	>10 years
Availability	Today	Today	beyond 2020

Figure 3.1: Comparison between SIGFOX, LoRa and 5G targets [25]. 3GPP technologies are listed in Figure 3.4.

Another general term for cellular IoT like networks is Low Power Wide Area Network (LPWAN). Figure 3.1 shows some competing technologies in a segment of LPWAN. These networks will have long range, low power, low data rate and very low cost. Current technologies available, e.g., LoRa and SIGFOX, are operating on unlicensed Industrial, Scientific and Medical (ISM) bands and they offer the ability to create local deployments before global connectivity is available through (3GPP) standardization as explained more in Section 3.2.

Finding "winners" from these technologies will take years but the main targets are the same. There will be big markets when it is estimated that the amount of networked devices will be tens of billions in the next decade. Fifth generation (5G) of cellular networks is targeted to launch in 2020 so operators are investing in LTE based technologies before that. IoT is also a big and long change, thus some of the features will be implemented into LTE and some features into 5G, and it is unknown how these giants will be aligned.

Currently it is clear that IoT will be used in multiple verticals. There will be and already is so many applications that it is likely that no single solution will be available to connect all of the cases. 3GPP is mainly working on licensed bands and global compatibility which will add extra costs to technologies supporting 3GPP standards. Also LPWAN networks will connect only part of the devices and unlicensed short range technologies are estimated to cover more cases. For example, public WLANs (Wireless LANs) can be found in places which have dense population. WLAN standards are also evolving towards IoT [26].

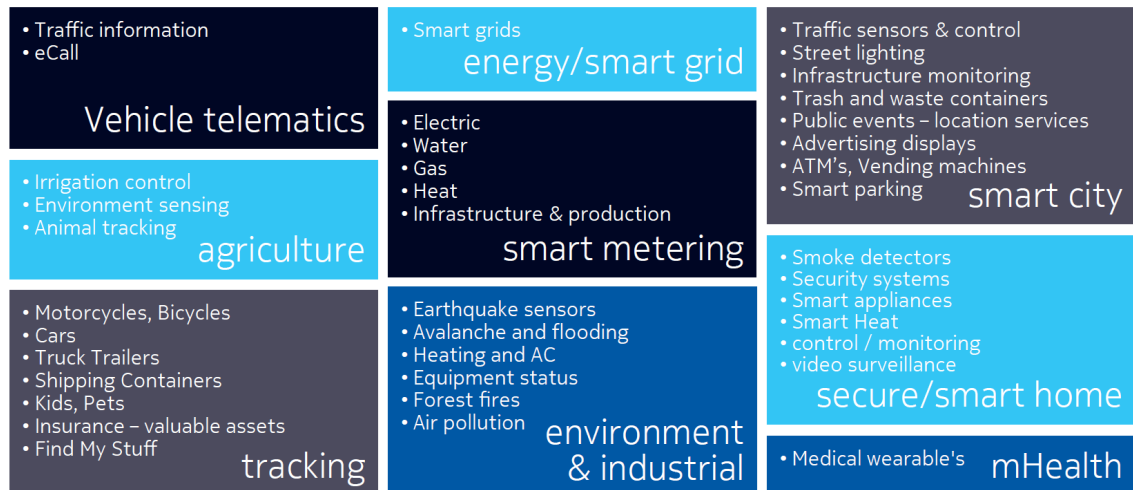


Figure 3.2: IoT vertical examples include long list of use cases [25].

Some of the verticals have proceeded closer to real life usage but many of them have not entered the markets. Large number of devices requires new kind of control layers to handle the devices, applications, and the new data has to be processed into more valuable form. In short, the whole package is currently open but some success stories have been made and there is room for new business.

From LoadGenerator point of view, all lightweight applications should be able to run on top of LoadGenerator. For simplicity, metering is selected to be an example in this thesis because the function of the meters is straightforward. A measurement is done and sent over the network and processed further. Meters and sensors will be everywhere in the future, thus emulating large number of those is one way to study future networks.

### 3.2 Future estimations and network requirements

The results from standardization process show that evolved LTE will be used for cellular IoT. Many of the LTE features are designed for high-speed access with low latency which differs from IoT requirements. Features that are not needed can be removed or optimized to IoT usage. Thus, the requirements for IoT devices must

be examined and estimated so that LTE optimization can be done efficiently.

The device count will grow but many of the devices won't use much bandwidth and will send data seldom. Long battery life requires that low power sleep stages must be used. During sleep no communications are made. From core point of view every device and session will reserve some amount of resources, thus static resource usage should be minimized. During idle stages some resources can be released also in the core and thus make resource usage more dynamic.

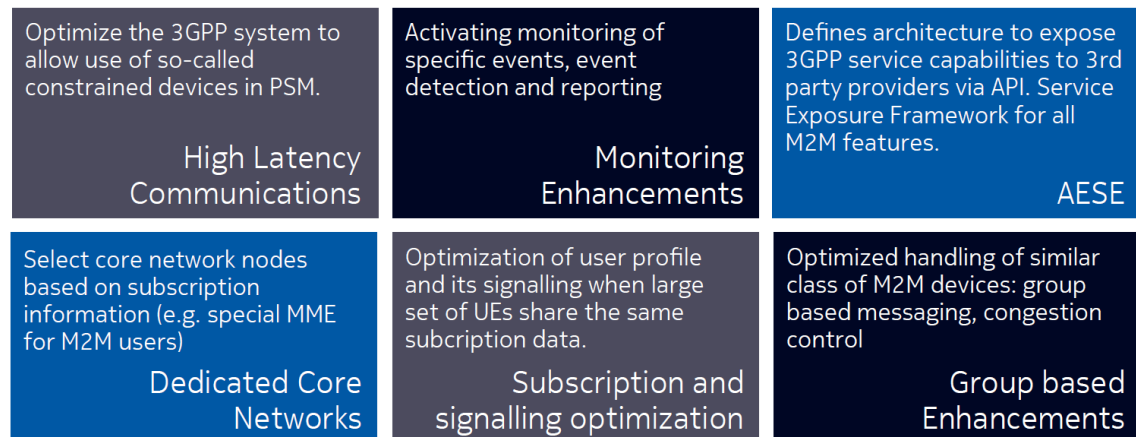


Figure 3.3: Techniques for IoT optimizations [25].

Real-time connections are not needed for all devices. This allows higher latency which can be exploited by using caches to distribute load more evenly over the time. Also higher utilization rate is possible when load is evenly distributed. Longer latency allows core to be further away, e.g., in large data-centers where more resources are available.

Many IoT devices are static and thus real-time handovers are not necessary. If IoT device moves to a new location it can re-attach to regain connectivity. Also active location updates, paging and TAU procedures can be adjusted to reduce unnecessary signaling. This simplifies the system which helps to reduce costs and saves battery and resources when signaling overhead is lower.

Reduced signaling mostly works on current core but simplified elements can also be used in core to save computing resources. Devices could use different core which would be optimized for the specific usage, e.g., MME, PGW and SGW could be implemented into "one box" and also non-IP connectivity could be used [6]. For example, Short Message Service (SMS) could be used.

### 3.3 Future 3GPP Releases

LoadGenerator will use LTE as a base, thus LTE evolution is examined more carefully. The target is to gather knowledge about future networks so that also these



networks could be emulated better. In other words, this topic gives more precise information about scenarios and settings that could be implemented to LoadGenerator to test the effect of these changes in practice.

Although this section focuses more on RAN specifications, the core network will be logically built on top of RAN selections. In practice, 3GPP standardization is the first concrete information about the future cellular networks. Another note on this topic is that although some standard is agreed it doesn't necessarily mean that it will be implemented in large scale if the markets won't find value from a standard.

LTE Cat0 was approved in 3GPP Release 12 but it includes only minimal changes which will probably be implemented by software updates, thus it does not fill all the IoT requirements. However, Cat0 can be seen as the first step to evolve LTE towards IoT and MTC and the upcoming IoT evolution is based on top of Cat0. Figure 3.4 shows a summary between the upcoming 3GPP standards which are introduced in Release 13. As a recap, Release 8 is the first LTE release and Release 13 is latest frozen release.

IoT has been under work in 3GPP for some time. In 2014 3GPP had a study item named "Cellular System Support for Ultra Low Complexity and Low Throughput Internet of Things". The objective was to study how cellular IoT could be implemented from radio point of view. Both GSM evolution and clean slate were evaluated. Result was that EC-GSM work item was approved to evolve GSM; eMTC (LTE Cat M1) and NB-IoT were approved to be evolved by the RAN Technical Specification Group (in 3GPP) in Release 13.

	eMTC (LTE Cat M1)	NB-IOT	EC-GSM-IoT
Deployment	In-band LTE	In-band & Guard-band LTE, standalone	In-band GSM
Coverage*	155.7 dB	164 dB for standalone, FFS others	164 dB, with 33dBm power class 154 dB, with 23dBm power class
Downlink	OFDMA, 15 KHz tone spacing, Turbo Code, 16 QAM, 1 Rx	OFDMA, 15 KHz tone spacing, TBCC, 1 Rx	TDMA/FDMA, GMSK and 8PSK (optional), 1 Rx
Uplink	SC-FDMA, 15 KHz tone spacing Turbo code, 16 QAM	Single tone, 15 KHz and 3.75 KHz spacing SC-FDMA, 15 KHz tone spacing, Turbo code	TDMA/FDMA, GMSK and 8PSK (optional)
Bandwidth	1.08 MHz	180 KHz	200kHz per channel. Typical system bandwidth of 2.4MHz [smaller bandwidth down to 600 kHz being studied within Rel-13]
Peak rate (DL/UL)	1 Mbps for DL and UL	DL: ~250 kbps UL: ~250 for multi-tone, ~20 kbps for single tone	For DL and UL (using 4 timeslots): ~70 kbps (GMSK), ~240kbps (8PSK)
Duplexing	FD & HD (type B), FDD & TDD	HD (type B), FDD	HD, FDD
Power saving	PSM, ext. I-DRX, C-DRX	PSM, ext. I-DRX, C-DRX	PSM, ext. I-DRX
Power class	23 dBm, 20 dBm	23 dBm, others TBD	33 dBm, 23 dBm

Figure 3.4: 3GPP IoT evolution summary from February 2016 [6].

The summary in Figure 3.4 shows that multiple technical changes are made to support IoT better and these three solutions also offer wide range of features to support multiple use-cases. For example, EC-GSM offers extended coverage, eMTC optimizes Cat 0 further and NB-IoT provides multiple deployment options.

Modem complexity affects the device prices directly when simpler chips are cheaper to manufacture. Narrowband and lower rates will be optimized to save battery with enhanced idle stages allowing *things* to sleep for hours if needed. Lower coding rates and multiple link powers will enable enhanced coverage.

From core point of view the biggest change will be the number of devices. 3GPP goal is to support 50000 devices per cell [6] for EC-GSM and NB-IoT, thus core elements will have to scale even more. NB-IoT and eMTC are introducing multiple optimizations on core side, too. For example, core architecture may be combined to single entity and data can be transferred over non-IP connection.

In overall, the variety of devices will grow and multiple techniques will be used to provide connectivity. Some devices are a lot more in idle state and some devices require very low latency. Thus, multiple solutions will probably be implemented on the core side also, e.g., edge computing can be applied to serve the device, and the user, as near as possible. Whereas large data centers could be used to serve devices with low priority traffic to optimize resource usage.

The last missing details in Release 13 are related to IoT but they will be finished before June 2016 [27]. The 3GPP work continues with Release 14 which is gaining momentum when Release 13 is closing and Release 14 has already a long list of items which can be found from [28]. Release 15 is also starting in June 2016 [11]. The completion of Releases 14 and 15 is years ahead but they can give some information about the areas which are important beforehand.

The upcoming features have to be implemented into emulations also. In practice, the LTE features of LoadGenerator are depending on LTE Emulators, thus the development of EMU will define what features are available in LoadGenerator in the future.

### 3.4 Example protocols

The mission of IoT is to connect all the things but what then? What will be the protocols to communicate with the things? IP connectivity is not the full networking stack and on top of IP there are application protocols. The path from specifications to standards and compliant devices will take time. It is also important to have new applications which take the advantage from the new things. To speed up IoT development also higher layer protocols have to be kept in mind. This section should give some examples which will be used in the future.

One interesting framework is webRTC [29]. It enables real time communication (RTC) between a browser and a thing for example. RTC is useful in scenarios where the thing is remotely controlled and sends information about its environment to the user via video for example. webRTC completes the networking stack and applications can be made inside a browser without any extra software or plugins.

In webRTC standardization World Wide Web Consortium (W3C) is responsible for the API and Internet Engineering Task Force (IETF) for protocol thus it is an open framework.

Also some quite old protocols for simple communications over unreliable network can be used for IoT. MQ Telemetry Transport (MQTT) is one popular protocol for transporting simple messages [30]. MQTT is open and free protocol thus it has been implemented by many industries. Its design principles have solved out the same problems that IoT networks and devices are now facing. Earlier devices and networks were simple and slow but IoT has packed this device type into a new form factor but some of the problems have stayed the same: how to handle slow unreliable connections?

The number of devices will also pose new problems. How to control large number of devices and how to process the gathered information? Some kind of platforms have to be developed to efficiently control, monitor, configure etc. large number of things. For example, how operator can manage the huge amount of devices, and how user is controlling his/hers devices? In short, there are also new needs for control protocols.

## 4. SYSTEM BASE USED IN THE SETUP

LoadGenerator has evolved to its current status through many steps. The implementation of the idea has proven to be possible already in early phase but the technical steps have been mostly into untested areas. Large emulated LTE network on top of cloud networking has many pieces that do not work without extra configuration. Before going to the current setup a short history is presented as an introduction. This section gives an overall description of the problems related to the setup.

First practical tests were done with VirtualBox on a regular laptop where a fully emulated setup was tested. The tests were successful and a basic configuration of virtual machines and network layout was found. Of course, the performance and the scalability were not good enough. Emulators need precise time information, whereas reading clock information in VirtualBox on Windows is slow. The next step was to move the setup to cloud environment where it could scale out.

The first cloud was a regular OpenStack based environment. It was not configured for tests which include traffic from unknown sources which are blocked by the security by default. When the environment configuration was done to fix security issues so that the rest of the cloud was operating normally, it was time to scale the setup by multiplying elements and creating scripts for event generation. When more events and more elements were added, things started to break down and fixes had to be made to go further. For example, EMUs needed multiple patches and default networking configuration needed expansion. Soon the concept of control network was added to control a large number of elements and virtual machines.

The cloud that was in use became small so a new and bigger cloud had to be built from a scratch. The bigger cloud was configured with LoadGenerator in mind so most of the cloud environment problems were solved during the cloud setup. Also orchestration was added to aggregate deployment. In the new environment the system has been scaled out to support millions of UEs and thousands of emulated entities fully emulated. It was time to integrate LoadGenerator with real MME and GWs to support even bigger setup.

The following Chapters will introduce the LoadGenerator setup in more detail. Figure 4.1 explains the graphics used in the following figures.

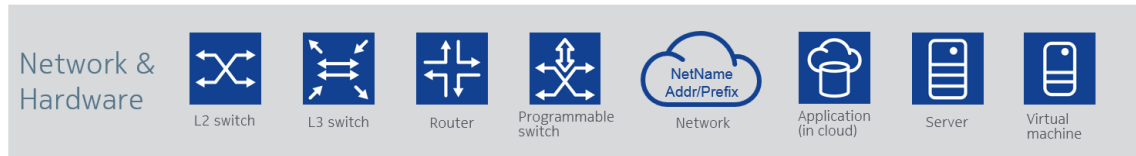


Figure 4.1: Network and hardware icons in use.

## 4.1 Current status

LoadGenerator is running in the innovation lab at Nokia Campus Espoo Finland. Currently 26 servers are allocated from two racks to build up two OpenStack clouds. Also some bare metal testing and support servers are allocated and installed. Splitting the setup in two has been preferred because one setup can be used for more stable testing and another one for experiments where outages might happen. This way more people can test their ideas without interrupting others too much. Real MME and GW are also running in the same lab.

Figure 4.2 shows the basic layout and connections between the racks in the laboratory. This chapter will introduce the building blocks of the system from hardware to virtualization on computing and networking which will enable OpenStack cloud. The focus will be on how standard hardware and software can be used in telecommunications and what problems there will be.

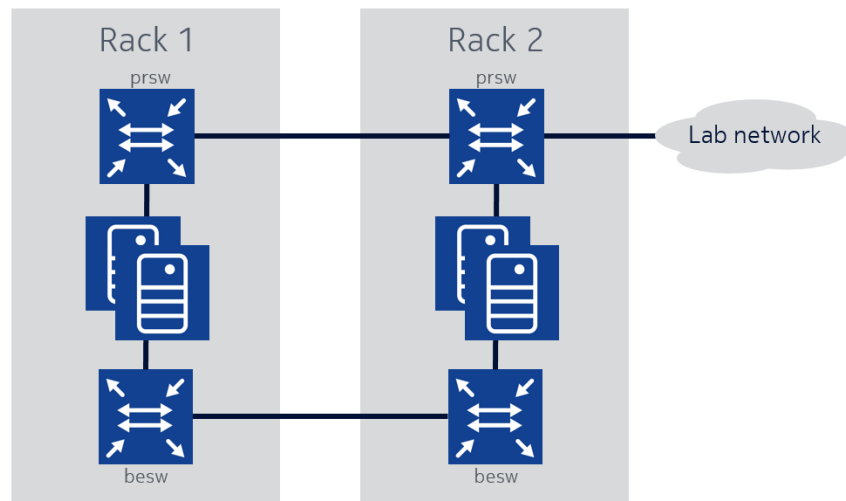


Figure 4.2: System is divided into two racks which are connected together and to the lab network. Physical layout supports the traffic flow through the servers.

Term *cloud* is quite loose and it is often added to some other technology to emphasize functionality over network and maybe hide some technical details in the way. More strict definition for *cloud computing* by NIST says: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a

shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [31].

With LoadGenerator, cloud computing will be used as a platform which offers an easy orchestration to rapidly deploy large masses of networks and instances which are turned into emulated LTE network. In practice, the selected system has to have good support for multiple networking modes and powerful orchestration tools. Also overhead in networking and virtualization has to be low when the protocol stack is complicated.

## 4.2 Kernel-based Virtual Machine (KVM) and containers

LoadGenerator has to be scalable, thus virtualization is applied to utilize hardware better. Main units in virtualization are virtual machines (VMs) and containers. Both techniques provide near native speeds and they can be used in multiple scenarios but there are also many differences.

This section introduces KVM VMs and (Docker) containers [32] focusing on Network Function Virtualization (NFV) related problems. Containers have many features that would be better suited for emulation but some mandatory features for LoadGenerator are missing. Nevertheless, containers are also introduced because it is likely that in the future they will be used more in telecommunications, too.

IBM's comparison between native, KVM and Docker Container environments provides a good overview of the subject [33]. The results show that containers typically have lower overhead than KVM. The study mentions that KVM overhead is especially visible in I/O-operations and low latency requirements, that are the features that Virtual Network Functions (VNFs) typically need: KVM can manage 10 Gb/s link speeds but special notice should be kept that the overhead can be an issue. Containers can perform better than KVM in operations requiring heavy I/O.

IBM's study also concludes that the difference between containers and VMs has narrowed down and both techniques have matured. Good example about the development, also mentioned in [33], is that the general rule to deploy infrastructure with VMs and platforms with containers is not so clear anymore.

The reason for the I/O differences is architectural. KVM uses hardware supported virtualization where isolation is done so that the stack is also growing vertically which is shown in Figure 4.3. The VM is running in a single process on host operating system's user space and this nesting causes performance penalty, especially with I/O operations which have to travel through two operating systems. On the other hand, this vertical addition improves security, when a new dedicated device is created instead of shared hardware resource usage.

KVM emulates the VM's devices, like NICs, thus the guest operating system does

not share them among other VMs. This gives more control for the guest but the emulation will grow the stack vertically and in practice adds overhead. KVM has options to pass-through host devices directly to VMs but in LoadGenerator’s case hardware resources are easily outnumbered if pass-through is used.

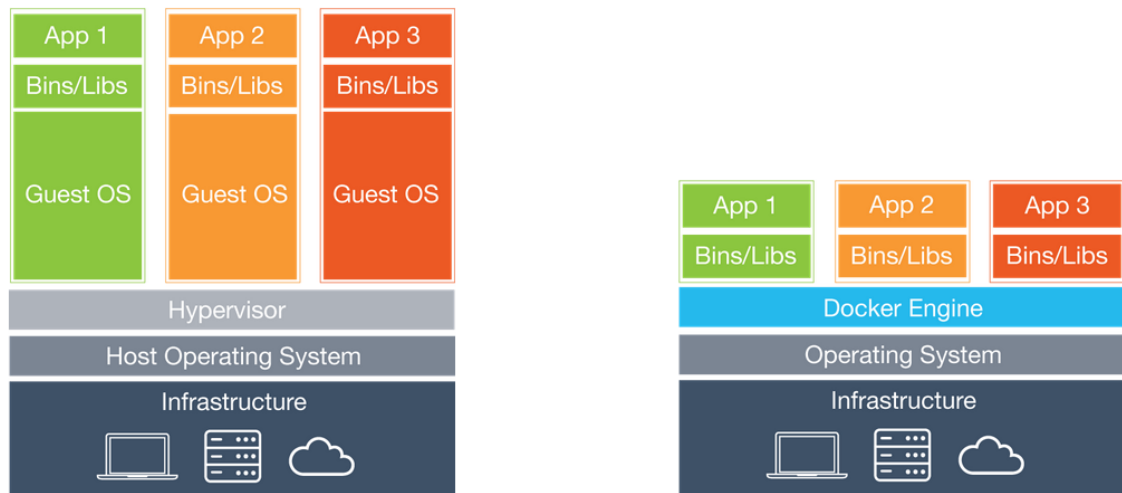


Figure 4.3: Container architecture has less vertical layers compared to VMs [32].

With containers the isolation is done more horizontally. Linux *cgroups* and *namespaces* are used to isolate processes from each other directly in the host kernel. Thus, all the processes are visible in host kernel where only software features are isolating them from the rest of the system. So processes running in containers have more direct access to host resources compared to KVM.

Containers are better suited in deploying large masses of microservices because they are lighter than virtual machines. Containers don’t have extra applications which are not used during the run-time. For example, one clear difference between containers and VMs is boot-up time: containers are up almost immediately and VMs typically take couple of ten seconds. Emulator process or VNF can be seen as a microservice and lots of them are needed in large scale emulation, thus containers are really interesting also from network emulation point of view.

In LoadGenerator’s case the problem with containers is networking. *Docker* 1.9 release in late 2015 introduced *libnetwork* which supports multinetting but still, e.g., SCTP [34] is not working between external and Docker overlay networks. Typical reason is that the protocol does not work over NAT which is typically used between external and Docker overlay networks. Host network mode in Docker is not an option when hardware should be divided between large number of devices.

Containers, and Docker, are developing rapidly and in the future containers will be used for NFVs more. When SCTP is working, LTE network emulation should be possible with containers that would enable smaller overhead compared to KVM.

Docker is also going towards very small footprint. In early 2016 Docker and Unikernel joined together. The main idea behind unikernel is to provide optimized kernel for the container. This will create even more light and optimized containers [35].

Because of the lack of SCTP support in container networking, KVM is left to be selected to be the virtualization service behind LoadGenerator. In the future, containers can be seen as a good candidate for network emulation. TCP and UDP based emulations could already be implemented especially if NAT is not an problem.

### 4.3 OpenStack open source cloud

OpenStack is a free open source cloud operating system. It converts standard hardware into cloud resources and services which are then turned into virtual machines, networks and in the end to applications and functions. OpenStack has a release cycle of six months and development is rapid.

OpenStack provides good orchestration tools and many networking options. For example, Open vSwitch (OVS), KVM and multiple other components can be used to virtualize networking and computing. In practice, OpenStack is a platform where LoadGenerator can easily be deployed on and decent performance can be expected. Thus, OpenStack is selected to be the base of the infrastructure. A short introduction to OpenStack is given next.

OpenStack consists of shared services which have certain responsibilities. The compute service is called *Nova*. For LoadGenerator good performance and scalability is needed so Nova is configured to use KVM for virtualization. Virtualization enables the sharing of one physical machine into multiple virtual machines.

*Neutron* provides networking service in OpenStack. It supports multiple drivers, networking modes and third party plugins. In short, Neutron can support nowadays almost any networking solution and fits well in our system where multi-networking is a must. Neutron and Nova work together so that Nova will create the VMs which have NICs that are bound to networks handled by Neutron.

Orchestration service is called *Heat* and it is a powerful tool which turns easy-to-use templates into ready-to-use networks and VMs. Heat is used for orchestration tool to deploy LoadGenerator. There are also multiple support services needed to run OpenStack: *Glance* is image service, *Keystone* is for authentication, *Horizon* is web interface for controlling OpenStack. Also databases, message brokers and other basic services are needed on the host machines.

Many third party automation tools are available to deploy OpenStack but they are not used here. With LoadGenerator host machines have CentOS 7 with OpenStack Kilo or Liberty that can be installed via Preboot Execution Environment (PXE). The chapter 5 will describe the installation in more detail.



To summarize the role of OpenStack with LoadGenerator: it is used to provide platform with orchestration and multiple networking modes. These features are used to easily deploy, connect and scale out LoadGenerator.

#### 4.4 Hardware

The servers in the racks have two Intel E5-2665 Xeon CPUs, 256 GB of RAM, three 2 TB hard-drives in RAID0 and Intel X520 dual port 10 Gb/s physical NIC (pNIC).

Intel E5-2665 Xeon is part of Romley architecture [36]. The most interesting knowledge of the platform for LoadGenerator is related to the networking capabilities of the system. The servers have only one pNIC with two ports. PCI Express (PCIe) lines are directly connected to CPUs which causes that NIC is directly accessible for only CPU0. CPU1 has to access NIC over the CPU0. CPU0 and CPU1 belong to different Non-Uniform Memory Access (NUMA) nodes.

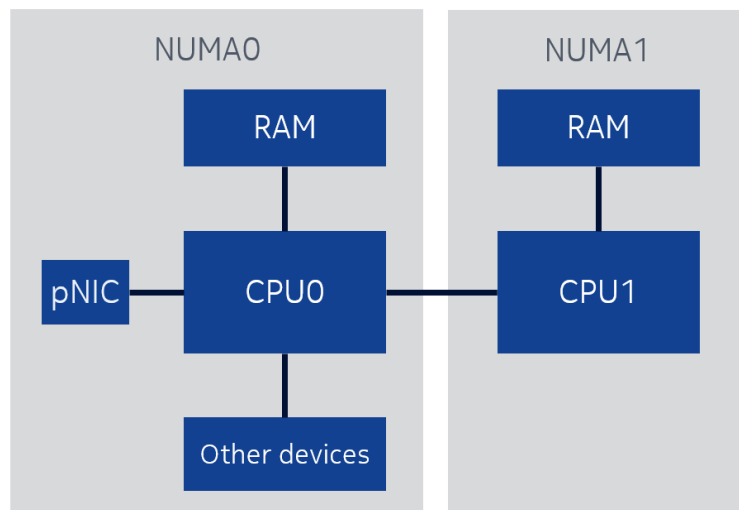


Figure 4.4: NUMA layout on the physical servers.

NUMA is used to describe the physical topology of the available resources. This will help kernels to use resources more efficiently. For example, Linux kernel has the knowledge of the NUMA costs in a matrix form which is the source for NUMA optimization. In practice, CPU's own memory is faster to access than the memory behind the other CPU. This is described by placing CPUs to different NUMA nodes.

In other words, crossing NUMA nodes causes performance hit. This is important to know in cloud environments. Especially VNFs take a big performance hit if they are not executed in the right place. One reason is that VNFs require real time execution and good I/O throughput. Virtualization hypervisors can try to converge to a state where the best performance can be achieved. However, manual CPU and hardware pinning is often needed to be able to reach the best performance.

In general, a guest instance, a VM, is just one process on host system. Hypervisor does not provide full host hardware resource information and, for example, NUMA information is dropped. This will create a problem for the hypervisor and for the guest kernel to schedule resources optimally. KVM has option to pin vCPUs to pCPUs and NUMA areas which can help with NUMA issues. Unluckily, the case is often so that pCPUs are over-committed with vCPUs and the static pinning does not solve the problem.

Host system tries to converge to an optimal stage by observing resource usage. Host services can move processes and memory used to an optimal location, i.e., other physical CPU core or NUMA area. There are issues related to this convergence: resource usage varies and optimal performance might be needed right from the start point, i.e., before converging happens.

```
# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 131044 MB
node 0 free: 112783 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 131072 MB
node 1 free: 115346 MB
node distances:
node  0  1
 0: 10 11
 1: 11 10
```

Figure 4.5: NUMA details including cost matrix. Compare to physical layout in Figure 4.4.

Top-of-rack (ToR) switches are HP 5900AF-48XG-4QSFP+ which have 48 10 Gb/s ports and four 40 Gb/s ports available with a switching capacity over 1,2 Tb/s. Both racks have a primary, *prsw*, and back-end, *besw*, switch which are connected to servers' pNICs' 10GE ports. Port1 is connected to the primary side and Port2 to the back-end side. Primary and back-end switches are connected to their counterparts on the other rack by 2×10 Gb/s link. Physical layout is drawn in Figure 4.2.

## 4.5 Networking stack

Networking stack can be divided into three groups: the physical layer with hardware and hardware related virtualizations, OpenStack networking with Neutron and OVS, and on the top are LTE Emulators. Networking stack might sound long and complex but the configuration is kept quite simple. The main goal is to have large number of networks that are isolated and LTE Emulators will make the connections between these networks.

Connections between LTE elements are typically done with routable addresses. This is due to the specifications where ports are strictly defined. In other words, Network Address Translation (NAT) can make connections between the elements very difficult. The use of routable addresses will consume a large amount of addresses in a large network. For example, every base station needs an IP address and we have thousands of base stations.

The network size goes so high that connecting LoadGenerator to external networks can be challenging. Thus, an internal network with private addressing is created and used. UE count and addressing is even more problematic when count is in millions. Some simplifications can be made if UEs don't need connectivity to external networks and NAT can be used for Internet access.

### 4.5.1 Network virtualization

This and the following section will describe some techniques that could be used for LoadGenerator and what effects they would have. Based on this evaluation selections for LoadGenerator networking are made. LoadGenerator will benefit from stable, efficient, scalable and non-hardware dependent network solutions. Hardware dependent solutions typically offer the best performance but they also limit hardware choices.

Most of the problems are how to share the limited physical resources. In practice, the question is how a VM is connected to the right network. The first step is to connect pNIC to multiple virtual NICs (vNICs) which are connected to the guest VMs.

Single Root Input Output Virtualization (SR-IOV) can be used to virtualize a device in a PCIe slot. It is hardware dependent virtualization. When SR-IOV is used, the device is divided into physical and virtual functions (PF and VFs) which appear as normal NICs in the operating system. For example, servers in LoadGenerator setup have pNICs that support SR-IOV: Intel x520 pNIC has two physical ports which both can support up to 64 VFs. These VFs can be directly assigned for single a VM and have near native speed.

Especially latency is significantly lower with SR-IOV compared to virtual switches. SR-IOV does not provide enough VFs for large scale usage but it provides a test environment where latency can be pushed down. This is important for example 5G testing where latency requirements are really low. Thus, SR-IOV can be only used in small scale testing with LoadGenerator. Some tests were done in OpenStack environment but MAC and VLAN spoof filtering [37] have caused some problems with LoadGenerator.

OVS is an open-source virtual multilayer switch. Typical use case for OVS is to connect pNICs and vNICs together so that networks are securely isolated, manage-

able and have a good performance and stability. OVS is widely used with OpenStack for example.

An IEEE accepted paper shows that OVS has better performance than Linux Bridge [38]. This comparison uses the same kind of network cards that are used in this work and results should be alike with LoadGenerator. The situation for Data Plane Development Kit (DPDK) [39] is the same as in the IEEE comparison: DPDK should provide best results but the implementation requires extra work and it has not been tested with LoadGenerator.

These results support the selection of OVS to be the connector between vNICs and pNICs. It is easy to use, stable and has the best performance from the virtual switches, and many platforms have support for it, including OpenStack.

### 4.5.2 OpenStack networking

There is no single way to configure OpenStack networking and the variety of options gives a possibility to have multiple solutions. This subsection explains some of the basic OpenStack networking terminology and operating modes available.

Neutron is the networking service in OpenStack, like introduced earlier. It is used to manage the lifecycle of the networks created by the cloud users. Figure 4.6 shows a basic deployment with the controller, network and compute hosts. A management network is used for internal communications between the controller and other host servers so that the services creating the cloud can communicate, and it is created during the cloud installation, thus Neutron will use it but won't manage it.

Neutron's role is to manage *tenant* and *provider* networks, and how they are connected to external networks. External network(s) will provide external connectivity but as the Figure 4.6 shows, there are some differences.

*Tenant* is a term for OpenStack project which can have multiple users. Thus, *tenant networks* are networks which are created and controlled by the project group members. For tenant networks, the networking host offers DNS, DHCP, NAT and network layer (L3) routing services. For every interface (eth in VMs), a *fixed IP* is assigned. Fixed IP is static, thus OpenStack cloud always has an IP address which can be used to communicate with the VM. VM can also have a *floating IP* which is assigned from the external network. To reach the VM from an external network the networking host will do NAT between fixed and floating IP.

*Provider networks* are networks where OpenStack does not provide all networking services. This enables the use of external network hardware and direct connections without NAT or networking host. Otherwise provider networks and tenant networks are alike. Tenant and provider networks can be chosen when VM is deployed, or they can also be attached and detached during VM run-time. OpenStack will make the connections between the chosen networks and VM's NIC(s).

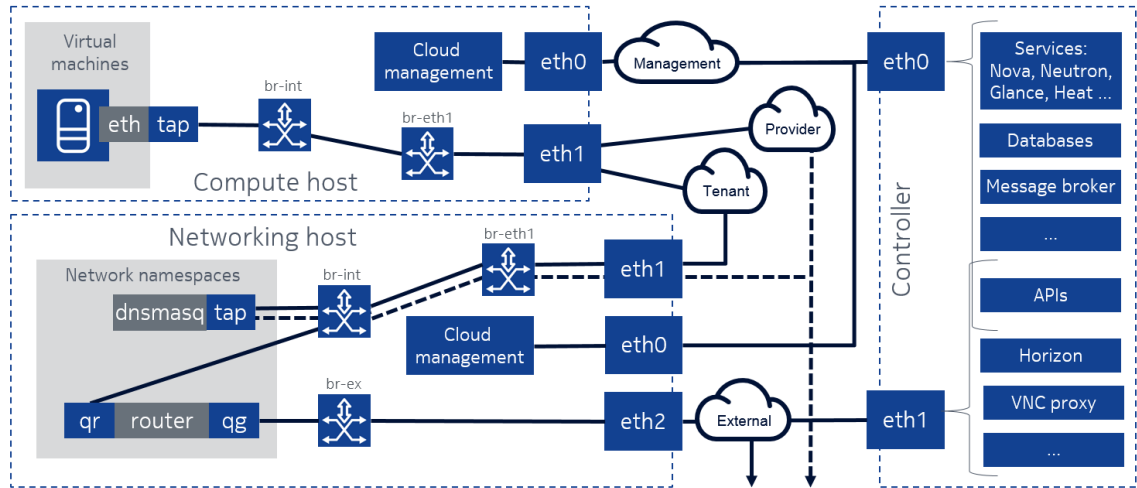


Figure 4.6: OpenStack networking example.

Neutron supports many drivers which expand the networking capabilities. In our setup Modular Layer 2 (ML2) plugin with OVS mechanism driver is used. OpenStack networks can be made using multiple techniques: ML2 and OVS support *flat*, *local*, *VLAN*, *GRE* and *VXLAN* networks. GRE and VXLAN are overlay networks which have the benefit of creation of L2 domains over L3 connections with plain switches.

In large data centers, it is useful when L2 doesn't have to be configured from point to point on network hardware. GRE and VXLAN cause overhead when encapsulation and decapsulation has to be made for traffic crossing different hosts. Offload functions on network hardware can be used to decrease the overhead CPU load also with overlay network protocols.

*Local* network type creates only local network inside the compute host. Local network could be used with LoadGenerator to deploy isolated networks to connect UE and EMU VMs but Heat orchestration does not support deploying local networks. Flat mode does not create isolated L2 domains so it can not be used in our case. VLAN separation needs support and configuration from network hardware. Separation is achieved by using VLAN tag which is 12 bit long and thus supports separation of 4096 L2 domains per switch supporting VLAN tagging. These features lead to the fact that VLAN has clear scalability and configuration issues in large data centers.

LoadGenerator will be deployed into two racks, with two ToR switches each, thus making of L1 connections is not a problem. With four switches it is possible to create over 16k ( $4 \times 2^{12}$ ) separate L2 domains when VLAN tenant network separation is selected. VLAN separation is light and it provides enough separated L2 domains analogous to LTE cells where UEs and base-stations are. One Ethernet switch typically has support for 8192 MAC addresses at least. Inside compute hosts, the

internal integration bridge (br-int) uses VLANs also to isolate networks, thus single compute host is also limited to support 4096 networks.

To estimate more about the setup scalability, the specification of ToR switch says that maximum MAC table size is 128k per device. These theoretical upper limits define the maximum amount of directly connected devices. In practice, LoadGenerator will have millions of UEs so they cannot be connected to ENBs via ToR switches. On the other hand, these calculations are just telling how to not to do the setup because physical switches are not the only switches in use.

To circumvent the problem of limited MAC addresses on physical switches, UEs and ENBs will be connected within the compute hosts, which is anyways logical solution to keep the data paths as short as possible. Thus, it will be *br-int* OVS bridge, that will be used to connect ENBs and UEs, thus having also most of the MAC load. OVS can handle a sufficient number of flows, thus it is capable of handling the traffic.

ENBs will be connected to EPC via provider networks and ToR switches will act as gateways for these networks. Thus, the MAC limit will apply to ENBs, but the number of ENBs will stay under the limit clearly. Another thing is how well the controller will scale, especially Neutron server, when large amount of networks and ports are deployed. On networking level the scalability will not be a problem if basic properties of all layers are kept in mind.

### 4.5.3 Offload functions

Multiple optimizations and offload features have been developed to improve networking performance. Most of the offloading features try to reduce CPU load with various operations. For example, a NIC can do checksum calculations just before/after the wire and rest of system doesn't have to care about checksums. Typically these features have a big performance improvement but with network functions they can cause problems.

NIC is typically thought to be last/first step of the network but with cloud environment it is rarely the case nowadays. Virtual networking has changed the game so that NIC is only bridging the physical and virtual network together. Many of the networking optimizations and offloads have been made for traditional networks long before large clouds have been built up, thus the compatibility issues can rise in surprising locations.

When routing and bridging is combined to optimization which alter packet headers, corruption is likely to happen. Also other modifications to the packets can cause multiple problems. Another typical case is when extra tag or encapsulation will grow the packet size over the maximum transmission unit (MTU) and fragmentation happens. For example, VLAN tag and GTP encapsulation add extra bytes

and this has to be known when adjusting MTUs.

LTE Emulators also have some packet capturing functions which are not compatible with most of the offload features which are present in the NICs. The more layers the networking stack has, the more probable it is that things can go wrong. On the other hand, hardware features are often needed to handle packets arriving at very fast rate, thus it is always balancing between the features and compatibility.

Offload settings can be controller and in practice incompatible offload features are turned off. *Ethtool* is a program on Linux to adjust the NIC settings. For example, large receive offload should not be used with bridging and routing [40].

#### 4.5.4 Network performance

Typically Linux systems have been strictly divided into user and kernel space. Both spaces, user and kernel, have certain features and limitations which are problematic especially with I/O virtualization. This is due to the logic how data is transferred between the spaces: copying the packet between the spaces is bad performance wise. With enormous amount of small packets the number of function calls and interrupts will use too much CPU cycles and throughput goes down.

From networking point of view, KVM can support multiple modes. By default the best performance can be achieved by using *Vhost-net*, which is shown in Figure 4.7. Vhost-net enables direct queries between guest NIC and host kernel space where the virtual *tap* device and networking bridge are running. This direct connection reduces CPU cycle usage but it can still be a bottleneck.

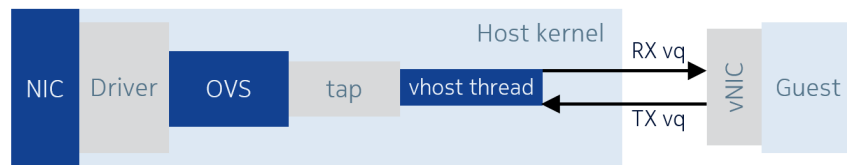


Figure 4.7: Vhost-net architecture.

LoadGenerator has been running on top of Vhost-net and OVS but this configuration is not very good from performance point of view. In the future specialized packet frameworks should be implemented into LoadGenerator because they enable better link performance between guests and host pNIC(s) with lower resource usage. The problem is that specialized networking frameworks are not yet fully matured.

These specialized packet frameworks typically enable direct communication between the user space and the pNIC by replacing slower kernel networking. DPDK [39] and Netmap [41] are examples from this group. They are typically installed directly from sources which is more complicated than using package repositories.

Another problem with these frameworks is the lower abstraction level. In practice, TCP/IP stack is missing and applications which are using operating system sockets are not working directly. On the other hand, these frameworks can be used for low level programming for best performance. One option to complete the stack is to use OpenDataPlane [42], which extends both frameworks with good performance.



## 5. SYSTEM INSTALLATION AND CONFIGURATION

This chapter will present how LoadGenerator can be installed and configured. This chapter is not a complete guide because LoadGenerator is not restricted to a single platform or configuration. In other words, a couple of example deployments are presented to show how the full-stack can be built. If needed, many of the components in the stack can be replaced easily when they are using standard networking.

The deployed layout varies from single EMU VM to a large set of emulated and real entities. Three scenarios are presented here: The first one is a fully emulated setup with three VMs and two networks. The second is a single non-virtualized server without U-plane payload capabilities with real MME and GW. The third one is the large scale setup with emulated RAN with U-plane payload mapping, and real EPC is connected, too. Note that from this point on, GW is referring to the combination of SGW and PGW which are implemented by single physical element.

The first setup in Figure 5.1 will support up to 5000 UEs, 50 ENBs, 1 MME, 1 HSS and 2 GWs. These numbers are software defined limitations in the EMU. Single fully emulated U-plane path (UE-ENB-GW) can support throughput over 1 Gbit/s depending on the configuration and hardware. This deployment can be used to study how LTE, emulators and the environment are working. This is especially recommended if they are not known beforehand. Fully emulated environment gives the possibility to test everything and it is a good environment for learning, i.e., one is not depending on external systems when all the elements are emulated.

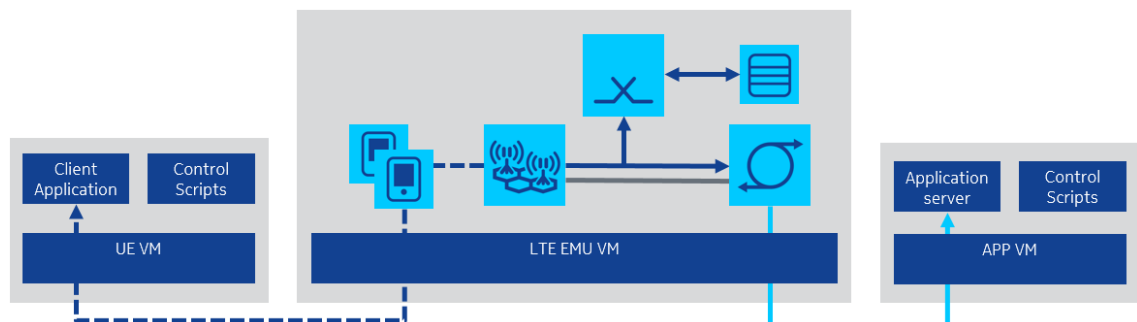


Figure 5.1: Fully emulated setup with three VMs and two networks.

The second setup in Figure 5.2 is a special case where C-plane load is generated

with minimum resources and it can scaled also on single non-virtualized host. This deployment shows how the scripts from LoadGenerator can be used to configure EMU also outside cloud environment, and how to create C-plane stress test setup to load real EPC.

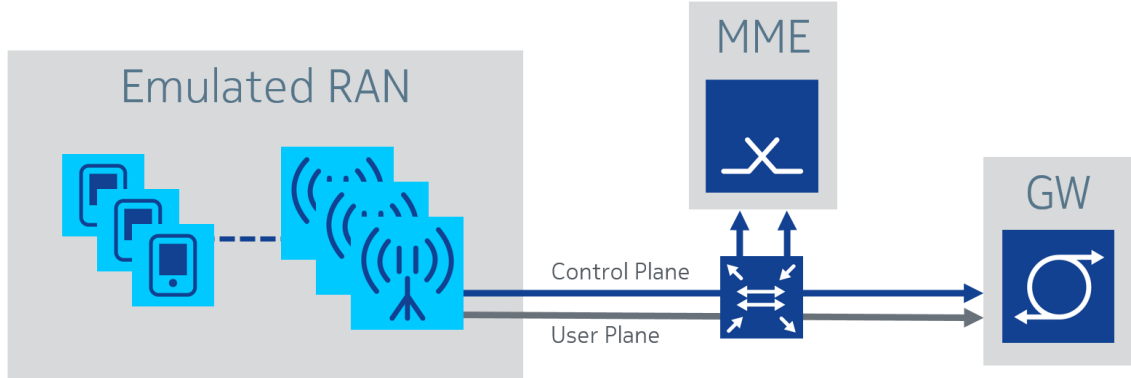


Figure 5.2: LoadGenerator only emulating RAN without U-plane payload.

The third deployment in Figure 5.3 shows how LoadGenerator can be used in a cloud environment and how it scales. This scenario will create large scale C and U-plane load to real elements. This is the combination of the first and second case with centralized control network and cloud computing.

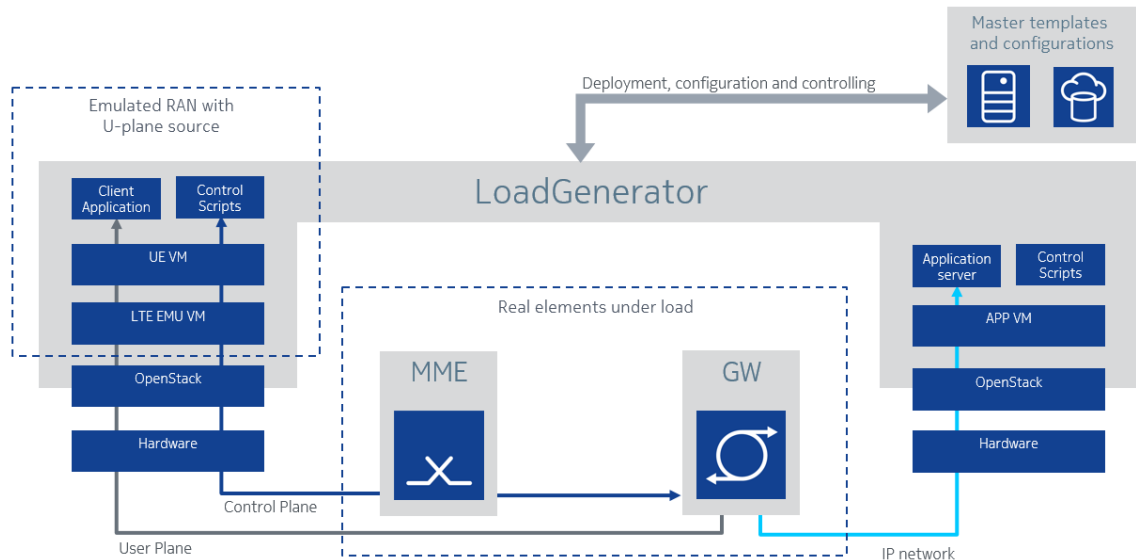


Figure 5.3: LoadGenerator building the full-stack.

These three cases are example templates which then can be used for easy deployment and customized further. Small scale setups are good for first time tests like checking E2E configuration of newly installed element. Large scale deployments are used to generate load and to create very large networks to model futuristic scenarios.

## 5.1 Configuring hardware, network, and OpenStack

Servers are updated to the latest Unified Extensible Firmware Interface (UEFI) and firmware versions available and maximum performance profile is enabled. Hard drives are assigned to single drive RAID0 (striped mode) configuration.

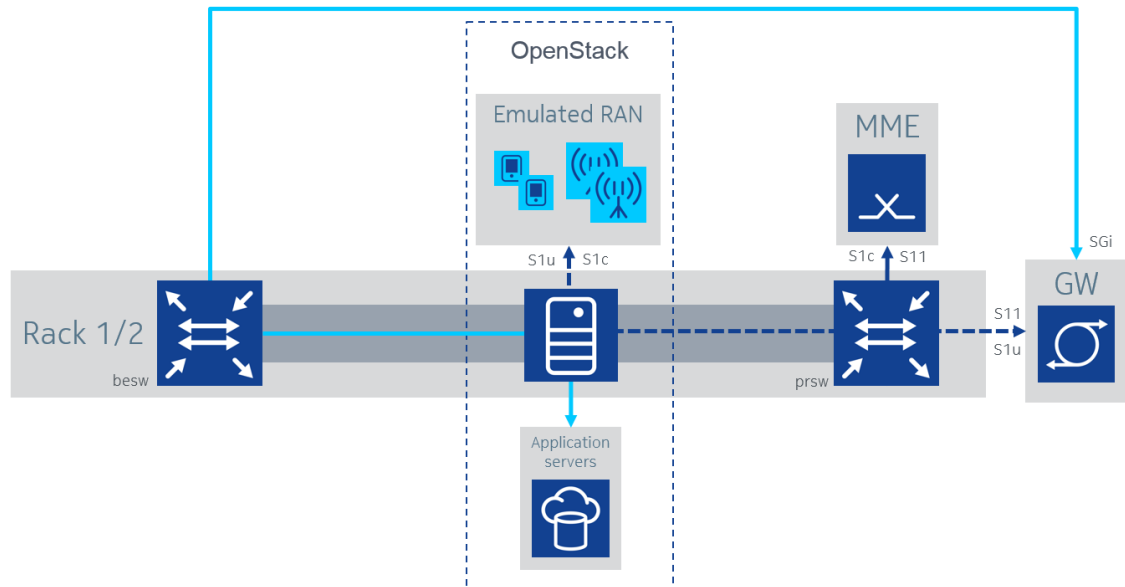


Figure 5.4: An example how external LTE elements can be physically connected to LoadGenerator.

The physical network connections are shown in Figure 5.4. Networking configuration is made so that primary switch (prsw) connects RAN to EPC. Back-end switch (besw) is connected to SGi side of the PGW. SGi is the interface connecting the PDN to PGW. Switches are configured to act as gateways for provider networks. In practice, IPs are assigned for the gateways and static routing is added. Tenant networks can use VLANs, defined in ML2 configuration, on trunk ports, thus they don't need to be configured to the switches.

During the host installation the primary switch port type is configured as *access port* to support PXE boot from specific VLAN. During the host installation the port connected to besw is configured to use OVS and is ready to support cloud networking. After the host installation is completed, OVS is added also to primary port. Cloud management network interface is configured as internal OVS port on the compute host. The end-configuration of OVS bridge-mappings is presented in Figure 5.5.

OpenStack networking is configured so that OVS is used to share the two physical network interfaces. Both ports have their own OVS bridges that are connected to a third bridge named br-int. These mappings and networking modes are defined in

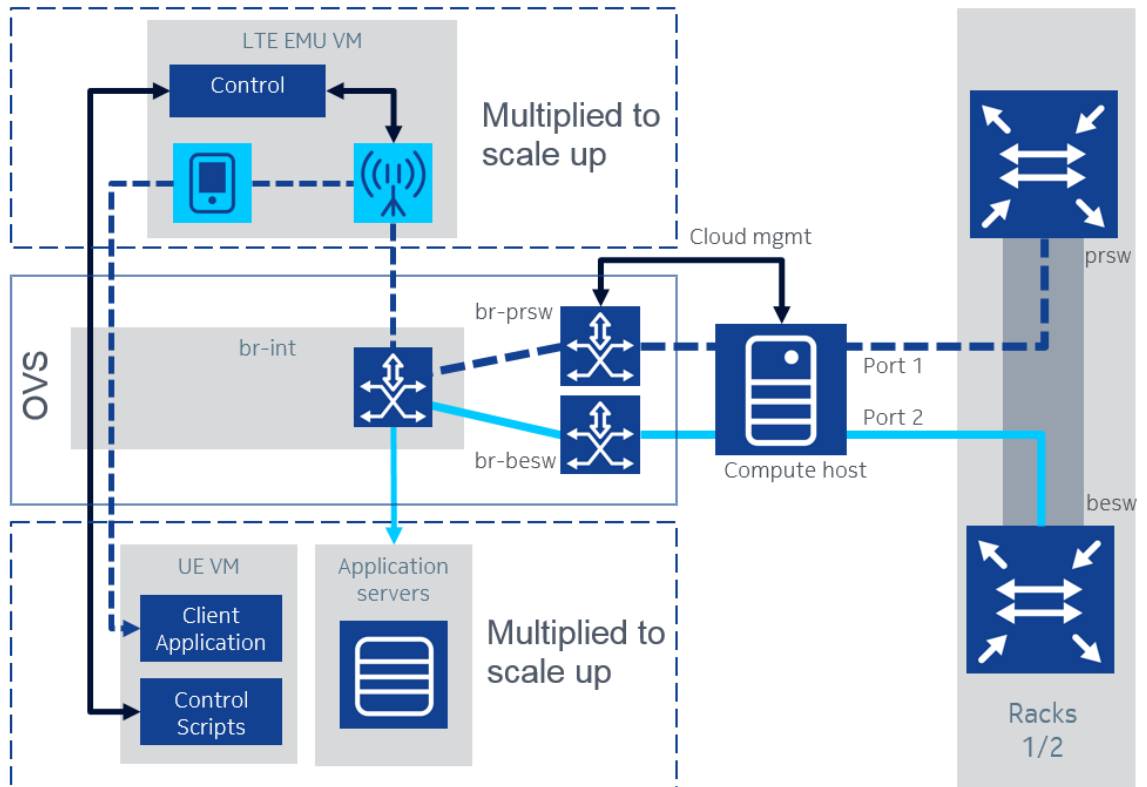


Figure 5.5: OpenStack OVS configuration consist of three OVS bridges which are controlled by Neutron.

Neutron’s configuration. OpenStack can deploy networks according to this configuration. In practice, this configuration gives the ability to use both physical networks via OpenStack networking. Prsw is used to contain LTE networking (C-plane and U-plane) and besw handles the IP network load coming from PGW’s SGi interface.

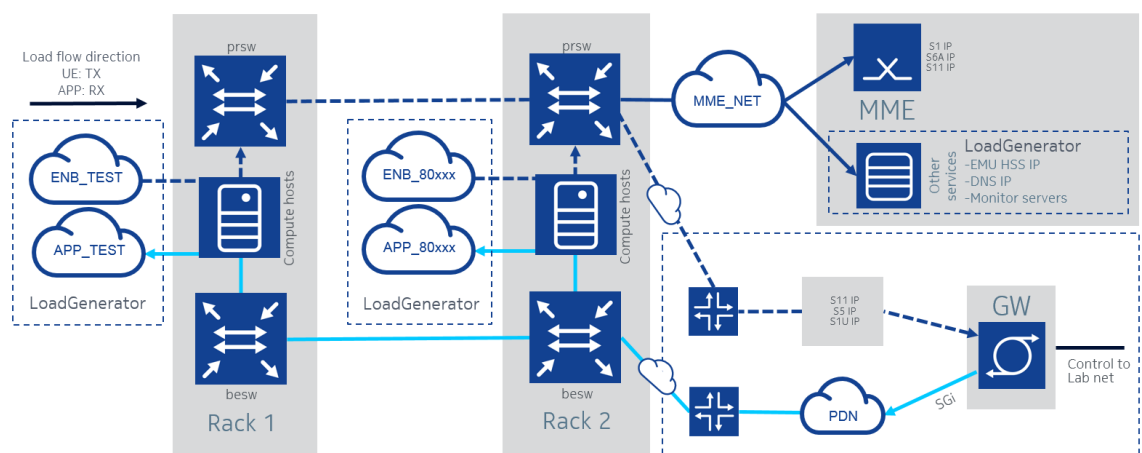


Figure 5.6: The overall setup and networking to external core.

The full network with both racks is presented in Figure 5.6. Rack 2 will handle the routing to external networks, i.e., Rack 2 switches are running with L3 configuration,

and VLANs for provider networks are passed directly to Rack 1.

To support unknown VNFs port security must be turned off. This can be achieved by using port security settings or by turning security off cloud-wide. Turning off security will also remove iptables-bridge from the OpenStack networking. Note that OpenStack Liberty implements ARP-spoofing protection via OVS flow rules by default where preceding releases do not. Security note: turning port security **off** can be compared to giving access to physical switch port which accepts all traffic. Thus, unsecure ports should not be given to an untrusted party.

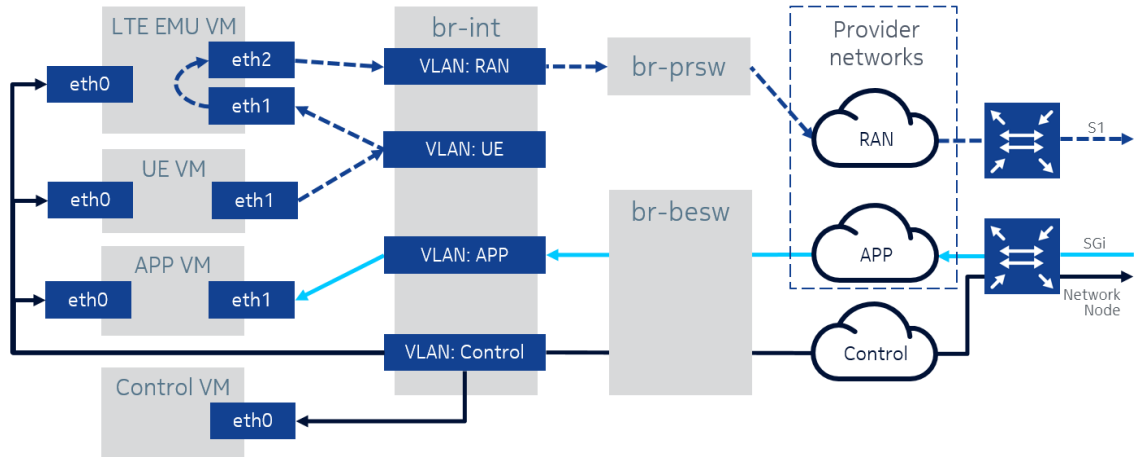


Figure 5.7: OpenStack networking configuration for LoadGenerator when external core is used.

After host, OVS, and Neutron configuration is made, the network scenario in Figure 5.7 can be deployed to support LoadGenerator. The arrows present a flow from UE to some service running on APP VM.

## 5.2 Configuring guest instances

Debian 8.2.0 is used as guest distribution. Installation image is configured and uploaded to image service Glance. Installation image has the necessary packages installed so it is ready to use after the boot. Heat installation template is used to configure remaining options related to different roles of the VMs.

LTE EMU VM is configured to run LTE Emulator. Instance has NICs that are connected to the UE, RAN and Control network which is drawn in Figure 5.7. If fully emulated GW is used, a NIC is added to connect the emulated PGW to PDN, e.g., APP network.

In this thesis, UE VM is the U-plane payload source for 5000 UEs, which are present in a form of macvlan type vNICs. UE VM can also generate some events by using DHCP for detach and attach, and send messages to EMU for idle and activation. If DHCP is used, the default dhclient is not light enough for hundreds

of parallel processes. Udhcpc is a lightweight DHCP client and it can be used to replace dhclient. Also MAC-table sizes are adjusted to support enough L2 neighbors. Typical L2 default limit is 1024 on Linux.

APP vM can be used to receive the payload coming from the UEs. In practice, this means that the applications running on APP VM are regular services. APP VM can also be left out if external services are preferred, e.g., for more performance.

### 5.3 LTE and EMU configuration

After the VMs and OpenStack networks have been deployed, it is up to the EMUs configuration how the LTE network will be formed on top. The basic LTE topology is chosen by deciding how many elements there will be and how they will be networked. The EPC has to have capacity to support RAN and all the elements have to be configured to support the selected scenarios.

LTE elements can be shared to multiple logical networks by using LTE identification codes. The ratios between the elements can be chosen quite freely, e.g., how many UEs per ENB and how many ENBs per MME etc. In real life, this is mostly related to product licensing, but of course there are technical limitations, too.

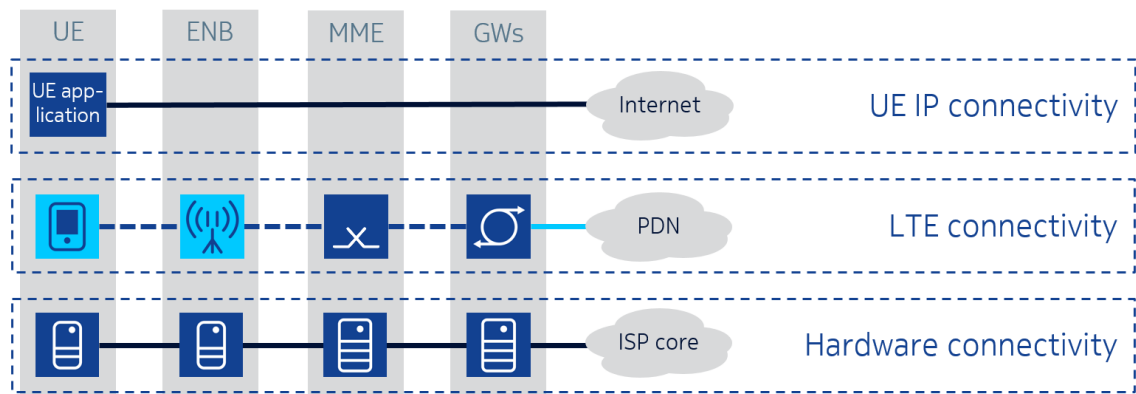


Figure 5.8: It is important to remember that LTE is only a one layer of connectivity. In LoadGenerator's case, the bottom hardware is consisting of VMs and servers.

In short, LTE configuration depends on the chosen case. If fully emulated setup is used, most of the default EMU parameters are fine and they can be adjusted more freely. If external entities are used, EMU has to match the device configuration or vice versa. To achieve compatibility, it may require the configuration of multiple networking parameters, not only the LTE parameters. It must be kept in mind that there are networks and layers under LTE and on top of LTE, that also affect the outcome. This is also drawn in the Figure 5.8.

In practice, the LTE network codes are adjusted so that test networks are in use. Plain IP parameters are configured and connectivity is verified so that entities



Figure 5.9: Typically, PLMNs and APNs can be used for logical division and elements need IP connectivity, too. In real life, there is a huge number of parameters that have to be set.

acting as servers (HSS, MME, GWs) are known beforehand where needed, and they can connect with each other. After plain IP connectivity is working and parameters are aligned, LTE connections can be established. It can be a challenge to configure element(s) manually, thus this step is mostly automatized in LoadGenerator. This automation is introduced in the next section.

## 5.4 Control network and scripts

LoadGenerator consists of hundreds of VMs, thus centralized control structure is needed for updates and commands. Current implementation is based on top of Secure Shell (SSH) which is not very convenient in large scale, but it is working when emulators are controlled with text based commands. Proper message broker and configuration database should be implemented in the longer run.

Master server is the central control point for LoadGenerator. In practice, LoadGenerator's configuration files can be retrieved via SSH File Transfer Protocol (SFTP) and commands are given through SSH. This general principle is drawn in Figure 5.10. There is a set of commands which have a predefined actions and scripts, but also custom commands can be sent to specific VM(s). Master server is also installation source during Heat deployment where SFTP is also used to retrieve the custom files.

SSH commands can be run in parallel with simple bash scripts or by using some program like *ansible*. Ansible uses SSH and offers easy-to-use configuration file for grouping the servers. In practice, both techniques can multiply single command to multiple servers, thus reducing the mechanical work. Control over SSH provides agent-less mode of operation, i.e., the servers under control don't need extra applications to receive the commands. SSH agent forwarding is typically used to

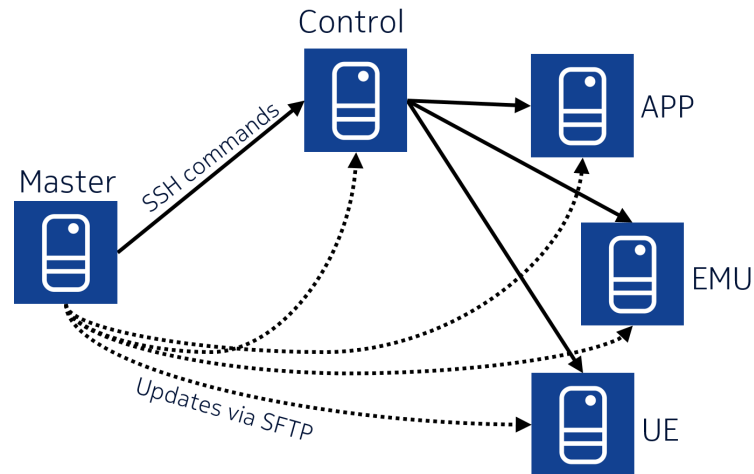


Figure 5.10: Master server controls other instances and provides repository for updates.

authenticate in the hierarchical chain.

Master server is used to share the updates like mentioned earlier. The basic idea is that the master server has a file structure where every VM type has its own folder and contents. These folders are packed into a zipped tarballs for the transfer and they are extracted on the destination VM. In other words, the exactly same contents are copied from the master server to every VM in the LoadGenerator. This enables easy updates to all VMs. Unique values are typically listed into a configuration files where the instance can identify itself and make actions according the configuration.

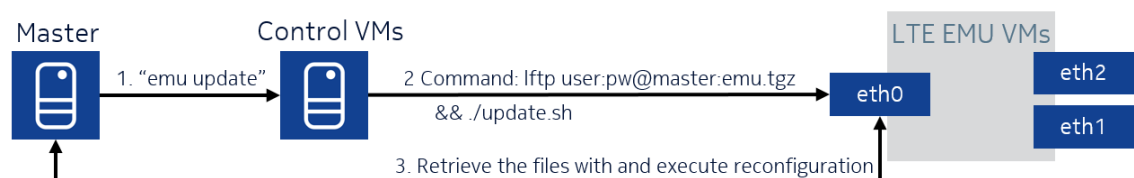


Figure 5.11: An example how update cycle is made. Master and control servers are used to distribute commands to target servers, which act according the given command.

The configuration of large number of LTE elements is automatized and this is drawn in Figure 5.12. In the Figure, `emu_config.sh` is a master script that generates initial values which are then applied to EMU VMs. The basic hierarchy is such that the initial values are stored on a master server. EMU will receive these values via control network and run necessary scripts to implement configuration based on the initial values.



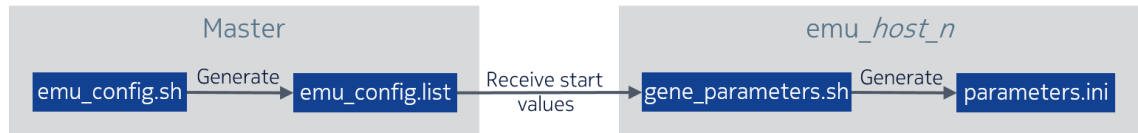


Figure 5.12: Parameters.ini is a file that is read when an emulator is started. It is generated locally on the VM but it receives some start values so that even a large number of EMU VMs can have a non-overlapping configuration.

## 5.5 UE without real air interface

UE process, responsible for the UE's LTE connection, is running on LTE EMU VM and it has to be controlled there. UE VM, the U-plane payload source, runs the actual end-user applications and does not have control over the LTE connection like real UE has. UE VM has to signal EMU in non-3GPP way. The missing L1 air interface will also cause that cell selection has is different from 3GPP specifications.

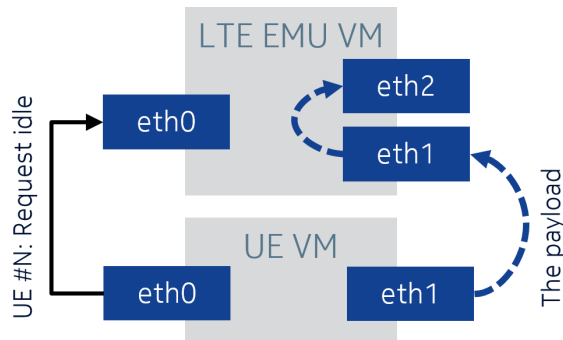


Figure 5.13: The eth0 interfaces are not used for LTE payload traffic but they can be used to control the emulator process, for example, idle states.

Another missing feature is that the emulated base stations don't have idle timers, thus the UE must control the idle stages via a non 3GPP compliant way, i.e., it will send a text command to EMU for going to idle or re-activating the connection. This is almost the opposite to real UE-ENB connection where the base-station will handle the idle states and UE cannot decide when it is in idle.

From core point of view, the EPC receives only requests and acts, and it does not matter from where the request has exactly originated. Thus by using the control connection, it is possible to circumvent the missing features from UE-ENB link when EMUs are used to test the EPC.

These limitations come from the EMUs' architecture which is designed for EPS testing and the physical radio part is left out. Nevertheless and as said earlier, EMUs have 3GPP compliant interfaces on EPC side and they support most of the the signaling scenarios which affect the EPC state.

## 6. SYSTEM USAGE AND LOAD GENERATION

After the basic configuration is decided the system can be scaled up. In practice, this is done by expanding the cloud by installing more compute hosts and then by deploying more guests where the emulation processes are running. Third dimension are the EMU processes themselves and how many elements and connections they will create. All of these three ways can be run separately to achieve the best mode of operation.

Basic idea in two first cases is to use predefined templates to automatize installation to save time on the mechanical steps. Configuring tens of host machines and thousands guest instances manually would be too slow and unreliable. Templates make testing more comfortable when failed setups can be re-installed and instances have exactly the same configurations.

The scalability of the EMU process is also depending on its configuration where the LTE values are defined. In practice, a configuration is made, EMUs are started to bring up ENBs and other core elements. After start-up UEs can be attached by using control network and scripts. EMU macros can also be used to create all kind of events to the network.

Installing the system is also very fast. Compute host installation takes around 10 minutes and the whole cloud can be set up under half an hour if preconfigured installation is used. Guest installation is also completed in minutes. Empty hardware can be turned into a large emulated network in a couple of hours if no problems occur.

### 6.1 Repository and support services

In this work, the system is installed on top of two clouds. These two clouds are in two separated racks and have 10 compute hosts each. To ease up maintaining and installing host machines, a management server is deployed. For network monitoring and debugging there are monitoring servers which have a GUI and a VNC server for remote graphical use. The management server acts as a gateway to cloud management network. It has templates for all host installations and proxy server to support the automatized host set-up.

As introduced, Master server is a VM which acts as a centralized control point and custom file repository. Code changes can be made on master server, pushed to

internal git repository, and/or packaged for deployment to instances. Every instance type has its own installation package on the master server which also has initial configuration values for the emulators.

*Redis* is an in-memory data structure store which can process high amount of events with low CPU usage [43]. Redis has also a feature for messaging: *Pub/Sub* is a service where clients can publish on and subscribe to channels in order to send and receive information without predefined source and destination. Redis combines features of database and message broker and everything is in memory with ACID features. This provides good performance for processing large data amounts. Redis client is available for multiple programming languages and *Pub/Sub* is used for VMs to report statistics.

## 6.2 Preboot Execution Environment (PXE) for host installation

LoadGenerator can be run in different environments but in every environment it needs an operating system underneath. In practice, the operating system must be installed no matter if LoadGenerator is running on non-virtualized or in cloud.

This section briefly describes how PXE network boot can be used to install and configure physical servers and OpenStack Compute host installation is presented as an example. The next step is presented in Section 6.3 where guest operating systems are installed in VMs on top of OpenStack.

With multiple physical servers, like a couple of tens with LoadGenerator, it is easier to use automatized host operating system installation instead of manual work. One typical way of doing this is to use PXE to boot from a network. If multiple hosts are assigned to the same role with similar configuration, installation templates will save many mechanical steps and reduce the risk of human errors. In other words, these techniques are needed to administrate data centers efficiently.

In LoadGenerator's case, a special server, called *management server*, provides DHCP, Trivial File Transfer Protocol (TFTP) and HTTP servers as the installation source for the environment. PXE clients, the physical servers under installation, are UEFI based and Intel X520 NIC PXE boot is used to retrieve boot instructions.

A boot program on the network card is launched and it requests IP address and more boot instructions via DHCP from the network, in this case from the management server. Clients are identified with hardware MAC addresses. The client receives DHCPOFFER that contains the IP address for the client and address for the TFTP server where the boot images are stored. The client loads the image and boots it, and CentOS installation starts.

Kickstart is the automated installation method for Red Hat based systems. Kick-

start will install the system according to a template, which is also downloaded from the management server. For example, OpenStack compute host installation has its own template which can be used multiple times to expand the cloud. Multiple templates can be defined for different roles and the selection can be made during the PXE boot. Figure 6.1 describes the overall process where a Compute host is installed and configured to register to the cloud controller.



Figure 6.1: When OpenStack cloud is expanded, new compute hosts can be installed by using network boot.

ToR switch configuration has to be made to enable a connection between the management server and the client. Typically a VLAN is allocated for the internal cloud management network where the management server and the selected hosts will be connected. The management server is connected to multiple networks, thus its port-type will be trunk in switch configuration. An interface is added to the management server network configuration for each VLAN.

PXE clients' ports will be access-type in the switches during the installation. After the installation is complete, it might be necessary to modify network settings according the planned configuration. For example, the interface used for the PXE boot can be handed over to OVS control which is not possible during the installation without interrupting the network connection.

In practice, the start-up time of a physical server in use is a couple of minutes and minimal CentOS installation takes under ten minutes. Installation time depends on selected packages and external repository speed. Internal repository could be installed to speed up package retrieval. The physical server might also need configuration changes, e.g., RAID storage and networking setup. The whole install process is typically between 10 to 20 minutes from the server start to installation complete. Automatized installation is fast, easy, and configurations are consistent across the hosts, which are good features in larger environments.

Although LoadGenerator hosts are installed with custom templates, there are also ready-to-use tools which can be used to ease the management of physical servers and use technique similar to those presented here. Ubuntu Metal as a Service is one example which can be used to manage physical servers, i.e., metal [44].

### 6.3 OpenStack Heat deployment

As introduced earlier, *Heat* is the OpenStack component for orchestration. Its target is to manage the lifecycle of resources found in OpenStack cloud [45]. In practice, Heat can create, configure, delete etc. most of the resources available in OpenStack, thus it is very powerful tool.

By using Heat, a large LoadGenerator setup can be deployed in hours. This can be compared to manual configuration of one instance which could take the same time, i.e., Heat enables fast and efficient scaling. The main trick is to convert code-like templates into usable resources.

Heat uses (YAML Ain't Markup Language) (YAML) formatted templates which contain Heat or Amazon Web Services (AWS) markup for descriptions. In other words, these templates contain the instructions what to create in a form that both, human user and Heat, can understand. The Heat format and available resources are defined in [46] and some example templates are shared in GitHub [47].

With LoadGenerator, Heat is used to deploy networks and VMs so that the network emulation is ready to run. Of course, Heat has no clue about LTE Emulators but it will be used to launch scripts that will do the EMU configuration part. Section 5.3 tells more about EMU configuration itself.

The basic unit with Heat is a *stack*. A stack is a combination of resources which are grouped together when the stack is created. Most of the Heat operations are targeted to a stack; like create, delete, modify, check, update etc. are typical operations. A stack is the total output of a template or a joint of templates. The same template can be used multiple times and only the parameters are changed which need to be changed are changed. For example, a typical scenario is to deploy database VM to another network or scale the database cluster with a new node. Heat can also be integrated to telemetry service for automatic scaling based on the load.

The basic usage of Heat requires that the service is installed in the cloud. When Heat is available, the first step is to create the *stack.yaml* file to describe what one wants to deploy. Once the template is ready, it can be given to Heat service, for example, via `heat-pythonclient` by giving command `'heat stack-create NAME -f stack.yaml -P "ext-net=EXTNETNAME param=VALUE ..."'` where `-P` defines optional parameters that are inserted into the template automatically.

A LoadGenerator deployment can consist of multiple stacks thus running the heat commands, with right parameters, can be a challenge. To solve this problem, stack controlling can be done via scripts. This is presented in Figure 6.2 where scripts, templates and cloud services are drawn. The arrows in the figure describe how these elements are related to each other.

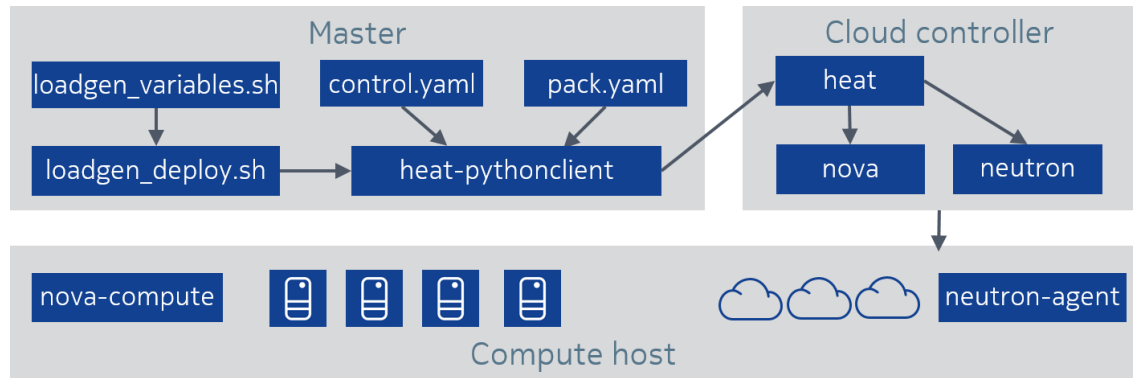


Figure 6.2: Scripts on Master are using Heat and Heat templates to deploy VMs and networks to compute hosts.

In practice, this layout is used to deploy LoadGenerator reliably and fast in large scale by using Heat underneath. The script called `loadgen_deploy.sh` provides a text user interface to insert variables, which can also be loaded automatically from `loadgen_variables.sh`. These variables are used to select the networks, compute hosts, etc. so that Heat will do the deployment in right place and with right size.

Second source for instructions are the templates and the first one, `control.yaml`, is for deploying a control network and a control VM, which are deployed once on every compute hosts used. The second template, `pack.yaml`, is for one emulator pack installation.

`loadgen_deploy.sh` will insert the given variables and template names into `heat stack-create` command. `heat-pythonclient` then combines the `.yaml` templates and given parameters, and makes the requests to a cloud controller. The controller then creates the networks and VMs according Heat's input by instructing the compute hosts.

After the controller and compute hosts have created the networks and VMs, which are connected together, the configuration may start. Some common configuration changes and updates can be made to the installation images already before Heat to reduce duplicate steps. However, Heat can be used to do the full configuration from plain cloud-image, but in LoadGenerator's case Heat is used to configure VMs to different roles.

For example, Heat could install a database to one VM and a web server to another VM. Heat could add a database user with password and transfer the login information to the web server, too. On the web server some application is installed with database access based on the given credentials, and the final output of the Heat template is the floating IP from where the application can be accessed. To get everything completed in the right order, wait conditions and signaling can be used. RackSpace offers some good tutorials, like this example, on their site [48].

How does Heat make this configuration? Heat uses *cloud-init* to configure VMs during the first boot. Cloud-init is a program installed in the (cloud) image of the chosen operating system. When VMs are booted, an image is loaded and cloud-init is requesting information from the network. In this phase, the configuration script from the template is inserted to the instance.

In LoadGenerator's case, the script in the template will do the configuration and receive files from the *Master* server which acts as our central repository for the custom files. *Cloud-init* can also send signals by using *curl* about the process success and failures so that even complicated chains can be made to verify that things are done in right order and required information is passed between VMs, and the user, too. Signals also help administrators to determine if the deployment has been successful or not.

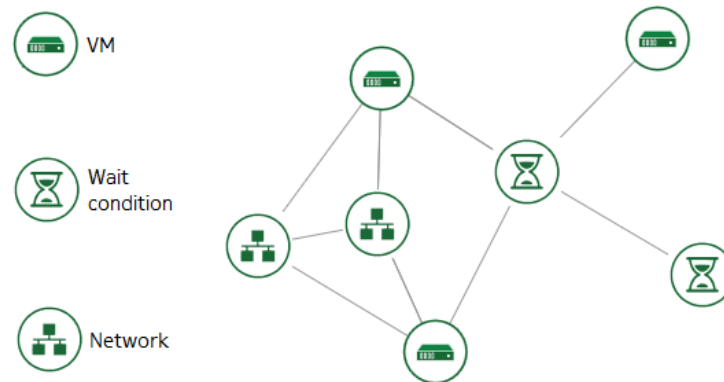


Figure 6.3: Graphical presentation of Heat-stack from Horizon. Everything is green, meaning successful, and create is completed.

The deployment output of pack.yaml, an EMU pack, typically contains three instances and two networks which are APP VM, APP network, EMU VM, UE network and UE VM. Figure 6.3 shows the output of pack.yaml from the Horizon dashboard. VMs are also connected to the control network created by the first template, control.yaml. Control network is used for control script networking and Internet access.

## 6.4 Load generation

Load in LTE networks is divided into two parts: C-plane and U-plane. C-plane mostly causes load to MME which is the central point for signaling in the network. Signaling does not cause so much load on networking layer but the signal messages have to be processed which requires computation time from the elements. This section presents some basic information related to load generation with LoadGenerator.

U-plane encapsulates the traffic to carry it through the LTE network, so network throughput and the number of sessions is the main performance values on U-plane. U-plane loads GWs the most because it terminates the GTP tunnels and connects UEs to external networks.

C-plane and U-plane are divided into different core elements, thus LoadGenerator setup can be different depending on the use case: payload on U-plane does not load MME, thus it possible to leave out the U-plane generation, when MME is tested. Dropping out U-plane source in LoadGenerator is possible and modular layout saves also resources when unnecessary components are not used.

After deployment, the system is ready for testing. System fine tuning and control is done by using control scripts and network. Operating system in use is regular Debian 8.2. installation so basically system can run any application as load. Load-Generator can be compared to a botnet of machines which can be controlled to cause traffic flooding

Every test should have a plan, method and goal to get results efficiently. The same applies with LoadGenerator which can be customized to specific test case. This thesis presents some deployments and test results in Chapter 7 but not all cases are covered or tested. This thesis is only using minimum LTE architecture but often the interest is in some support services which are monitoring the LTE elements, for example.

### 6.4.1 Traffic modeling

When emulation is used, most of the traffic can be generated by using existing network test applications. Cloud virtualization can run almost any operating system in a VM and the VM can run multiple applications. Single application can generate a huge U-plane load but it does not respond to real network where multiple users will share the capacity available. For more realistic traffic generation scenarios custom applications with proper parameters to adjust the behavior are needed.

One simple approach to model traffic is the method of record and playback: real traffic is recorded and only the headers are changed to get the traffic to a new destination, and the rate and count can also be modified. In this thesis the case-study is to mimic a large number of future IoT devices. Record and playback is not very convenient for LoadGenerator because the IoT devices are so simple that the payload is easy to generate in real-time. For other use cases record and playback should provide traffic which is very close to real life.

Another option to load U-plane is to study the payload and find out what is the distribution in packet sizes and how many destination-source tuples there are. With these details it is possible to generate dummy data which is sent over the network causing load which equals real network load. From network device's point of view



the payload content does not matter because only the headers are processed and payload is kept the same during transfer.

With LoadGenerator the scalability and simple input and output are important features. Most of the applications are made for testing one connection but with LoadGenerator there are millions of UEs using the network and every connection should have some kind of control and there are very limited resources available. Thus, customized application is needed to combine the needed features from different modeling techniques: one application will generate traffic to  $n$  UEs with customized payload. For simple IoT devices this is quite a simple problem: typical scenario is to send sensor data in a fixed payload periodically, e.g., electric meter sends usage data in every 15 minutes.

### 6.4.2 Traffic generation

As introduced earlier, GWs are the termination point for GTP tunnels coming from a large number of UEs and ENBs. LoadGenerator can scale horizontally to add more UEs and ENBs to load the GWs more. The system can also be scaled on per UE level where the running application defines how much payload is sent. These two parameters, the amount of GTP tunnels and the payload, define the load on GWs.

*Scapy* is a library for Python and it can be used to create custom packets. Scapy has functions which will form the packets according to the parameters provided. Scapy usage doesn't require low level knowledge of the protocol implementations: it is enough to tell what protocols to use and what will be the payload and options for the protocol. LoadGenerator has some example programs which use Scapy to generate the packets.

For low amount of TCP and UDP session with maximum throughput scenario, *iperf* [49] is a typical test application. *Iperf* has client-server model and it provides good statistics. EMU provides same kind of application like *iperf* but it can create a large number of streams and TCP and UDP sessions. These applications are good for generating massive U-plane load.

### 6.4.3 LTE transactions generation

LTE signaling is defined in the specifications and there are multiple configuration parameters which will change the operation of a network or some part of it. Modifying these options and optimizing the network for IoT devices will change the C-plane load. Most of the signaling is related in handling of the mobility of the UE and if IoT devices have less mobility, the signaling should also be reduced. One goal for LoadGenerator is to be able to emulate networks where IoT optimization have been

implemented.

LoadGenerator can be run with normal and simplified ENB (RANPT). With IoT tests the simplified ENB is more interesting and it scales much better. Simplified ENB supports attach, detach, context release which equals idle, service request which equals (re)activation of the bearer, tracking area update, detach and some other bearer modifications. The other functions which were not separately mentioned are not used in this thesis. The listed functions provide simple connectivity for IoT like device. Normal ENB extends support to full 3GPP compliance, e.g., hand-overs are supported.

As introduced, LTE Emulators do not emulate L1 air interface so the changes in mobility have to be launched with a workaround. This is not how real network operate but core does not see this difference. For example, the *context release* command is exactly the same even though UE controls when it is launched.

With the features of simplified ENB and with the own traffic generation application the following scenario can be implemented: application attaches  $n$  UEs in random intervals within defined time distribution, e.g., 5000 attaches in 10 minutes on single EMU pack. After the attach is complete UE sends first data packet to application server and goes idle according idle timer, e.g., 10 seconds. After 15 minutes the UE wants to send the next data packet and sends a *service request* to reactivate the bearer, sends the data packet and goes back to sleep. This loop can be left on to emulate some sensor network as a static background load.

The simplified ENB in the EMU has been designed to generate large amount of events to load the core on C-plane. There are some predefined EMU macros for load generation and EMU can be extended with external scripts. EMU can be controlled by giving text-based commands, thus it is easy to create all kind of events by using macros, scripts and manual commands.

#### 6.4.4 Measuring and visualization

Monitoring data is needed to measure and visualize the behavior of the element(s) under test. Real core elements, MME and GW, can provide good statistics about their usage but monitoring EMUs is not as straightforward. EMUs provide some internal statistics about C-plane events but U-plane statistics have to be gathered from operating system side.

Luckily, Linux has good statistics for network interfaces, thus it is logical to create a virtual network interface for every ENB to monitor the U-plane load. These vNIC statistics are read and throughput values are calculated. These measurements are published to Redis Pub/Sub channel from where values can be retrieved for further usage.

One demo visualization example is to generate locations for ENBs based on user

density data and put them on a map. Mapped ENBs are given IDs and they are grouped so that network segments are created. Given IDs will match the ones in the emulated system and close to real-time information can be seen on the map. For example, emulated ENB will publish its address and throughput to Redis and a map-dashboard will retrieve the information and show it on the map.

## 7. TESTS AND RESULTS

LoadGenerator can be used in multiple configurations and environments. This chapter will test the current features of LoadGenerator in fully emulated mode and with real MME and GW. Also small scale comparison between non-virtualized and Open-Stack deployments is made to estimate the virtualization overhead.

LoadGenerator is horizontally scalable, thus the maximum performance from single compute host is studied more carefully. To test the maximum load against real elements LoadGenerator will be scaled so that it can overload the real MME and GW. These tests will provide an estimation how much resources LoadGenerator will need to perform a specified loading scenario.

These results give an example what LoadGenerator can achieve but these results should not be taken as absolute values because they vary a lot depending on the configuration. To get the performance on the level that is presented here, should be possible without special optimization which could improve the performance a lot.

### 7.1 U-plane load

GW is the element handling the U-plane load, thus a test is made if LoadGenerator can overload GW. The load is generated by 495 UEs from 495 ENBs from five compute hosts and every UE has *iperf* client for traffic generation. The ratio of 1 UE active per 1 ENB is just for simplicity because *iperf* does not support multiple interfaces at once.

Iperf is a synthetic tool and TCP mode was used, thus every client tries to send or receive as much as they can. TCP is fair protocol, thus it will define how the networks capacity is shared. In these tests the network does not limit the throughput of single UE, but real GW supports traffic shaping and quality of service.

Figure 7.1 shows the statistics from the GW. In the first third UEs are only sending, thus uplink is fully utilized. In the middle third, the flow is altered by switching parameter *-d* on meaning that the iperf server will send data back to the clients resulting that both uplink and downlink are utilized. The last third is pure downlink. In all of the three cases the limiting factor is the GW throughput. The overload alarm was also reported by the GW monitoring application, which confirms the result.

When the results are compared to typical link speeds, saturating 1 gigabit link

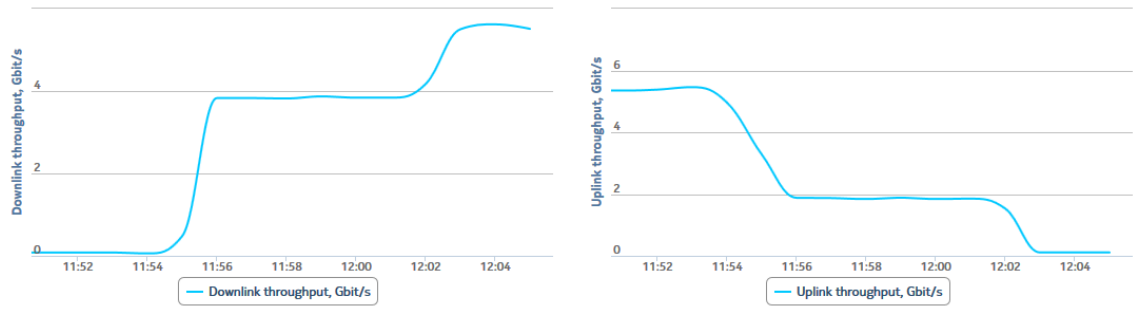


Figure 7.1: Maximum downlink and uplink measurement statistics where GW is the bottleneck.

is possible with single EMU process in right configuration. In non-virtualized environment close to 1,5 Gbit/s has been reached and in a OpenStack environment the number is close to 1 Gbit/s. These numbers are greater than LTE speeds nowadays and for IoT they offer a lot of throughput. A 10GE link has not been saturated because of the GW scalability.

## 7.2 C-plane load

Most of the C-plane load is expected to hit MME because it is the element handling the signaling between core elements. For example, in the attach procedure MME has to communicate with the ENB, HSS and GW to set up connection for the UE. In the next test the system is ramped up from zero to 2 million UEs.

As expected, during the attach stress-test MME was the bottleneck with a bit over 500 successful attaches per second meaning that the full system ramp-up to two million UEs is taking approximately one hour. Figure 7.2 shows the number of attached UEs in total on the left, and on the right there are number of activations (attaches) per second. In this test 693 ENBs were used and they all had 2890 UEs each. In total the maximum was 2002770 UEs.

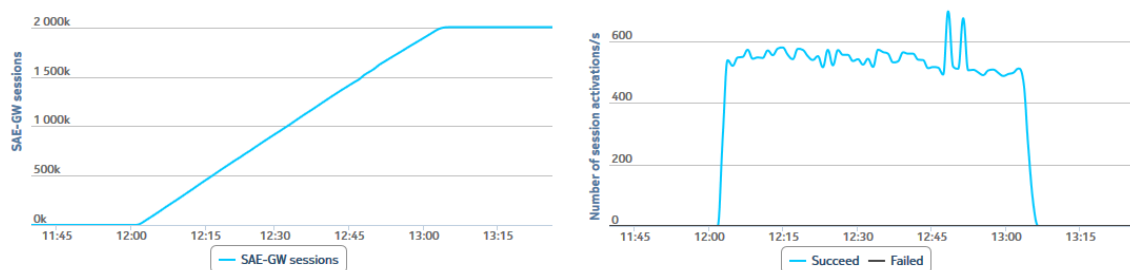


Figure 7.2: Attaching two million UEs takes approximately one hour and MME is the bottleneck.

In overall, generating plain C-plane load and causing overload to MME can be

done with a low amount of resources. On C-plane the environment, virtualized or non-virtualized, does not affect so much on the results because generating events is lightweight, especially with simplified emulator versions.

For example, simplified ENB is made so that it can also send a batch of events. Batch of 100 attaches can already cause the overload function discard some of the request in the real MME. In other words, the EPC will be the limiting factor on C-plane because the core signaling operations are much heavier compared to generating the request. A single simplified ENB can generate hundreds of attach request per second and when this is multiplied with the amount of ENBs the total number will be really huge.

Based on this evaluation LoadGenerator should be able to stress out any system available on C-plane with basic LTE produces like attach and detach. However, the performance on more detailed procedures has not been tested in large scale. The mapping between UE behavior simulation and event creation adds some overhead but in many cases the pure stress-testing without control over single UE may be preferred.

### 7.3 Scalability

The scalability of LoadGenerator consists of multiple dimensions. In short, hardware, virtualization, software, and the combination and configuration of these will affect the outcome. This section presents how the results change when some of the parameters are adjusted.

On cloud level LoadGenerator can be scaled by deploying more networks and VMs. This expansion can continue as long as the physical hosts have resources available. Figure 7.3 shows an example where each compute host has approximately 500000 UEs so the total number of UEs is close to 4 million. There are still some RAM unallocated but CPU over-commit starts to limit how much load a single host can generate. Also the compute host operating system shows that close to 100 GB of RAM is free even when the 500000 UEs are attached, thus there is also some space for optimization in the Heat templates.

LoadGenerator is horizontally scalable and adding more hardware will always increase the performance numbers. Thus, the most informative results are the numbers with a certain amount of RAM and CPU. However, it is hard to estimate how much a single process in a VM will consume the host system resources. This leads to a situation where the most accurate results are gathered by measuring specific unit as a whole, for example, how a single compute host performs under load.

Especially with high I/O load, the CPU load numbers are unreliable source. Study about OVS switch shows clearly that CPU cycle usage will grow linearly with the offered network load where as CPU load does not [38]. When CPU cycle usage

Hostname	Type	VCPUs (used)	VCPUs (total)	RAM (used)	RAM (total)	Local Storage (used)	Local Storage (total)	Instances
esclos81006.lgman.local	QEMU	608	32	203.5GB	251.8GB	2.5TB	5.4TB	304
esclos81007.lgman.local	QEMU	596	32	199GB	251.8GB	2.4TB	5.4TB	298
esclos81008.lgman.local	QEMU	596	32	199GB	251.8GB	2.4TB	5.4TB	298
esclos81009.lgman.local	QEMU	596	32	199GB	251.8GB	2.4TB	5.4TB	298
esclos81010.lgman.local	QEMU	596	32	199GB	251.8GB	2.4TB	5.4TB	298
esclos81011.lgman.local	QEMU	596	32	199GB	251.8GB	2.4TB	5.4TB	298
esclos81012.lgman.local	QEMU	598	32	199.5GB	251.8GB	2.4TB	5.4TB	299
esclos81013.lgman.local	QEMU	596	32	199GB	251.8GB	2.4TB	5.4TB	298

Displaying 8 items

Figure 7.3: Each compute host is hosting 99 emulator packs which can support almost 500000 UEs with U-plane support. In total, these eight compute hosts support close to 4 million UEs.

is the only reliable source, it is even harder to estimate the bottlenecks of the setup. Linux performance analysis program called *perf* can be used to analyze low level performance from the system and hardware counters, too.

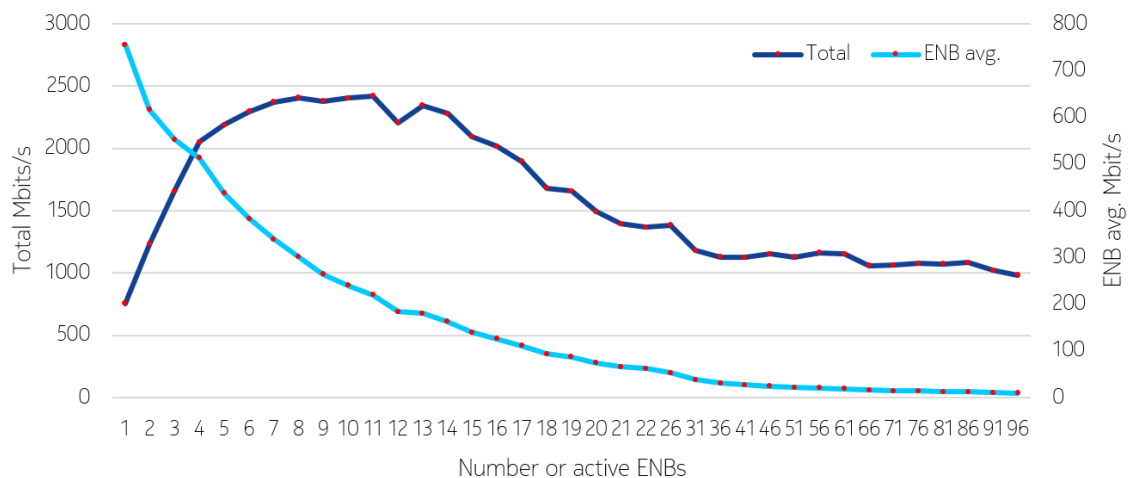


Figure 7.4: The scalability of U-plane on single compute host when the number of active ENBs is added.

Figure 7.4 shows how the number of ENBs per compute host affects the U-plane performance. Results are averages from three measurements from all the ENBs which were active and *iperf* was sending data from single UE per ENB, and the rest UEs were attached but not generating traffic. In total the host was running 99 ENBs with close to 5000 UEs attached on each ENB. The results show that single emulated ENB can achieve throughput of close to 800 Mbit/s in OpenStack

environment. The peak total throughput is close to 2,5 Gbit/s, thus with five to six compute hosts it should be possible to saturate a 10GE link.

Optimizing the system is also problematic when a single variable can make a huge difference in performance or scalability. Multiple default values can cause big problems in many places. For example, the cloud controller has to scale to support large number of ports and networks and compute host virtualization has to be able to share the resources efficiently. The worst scenario in networked services is typically an exponential growth in response times that can collapse the service waiting for the response.

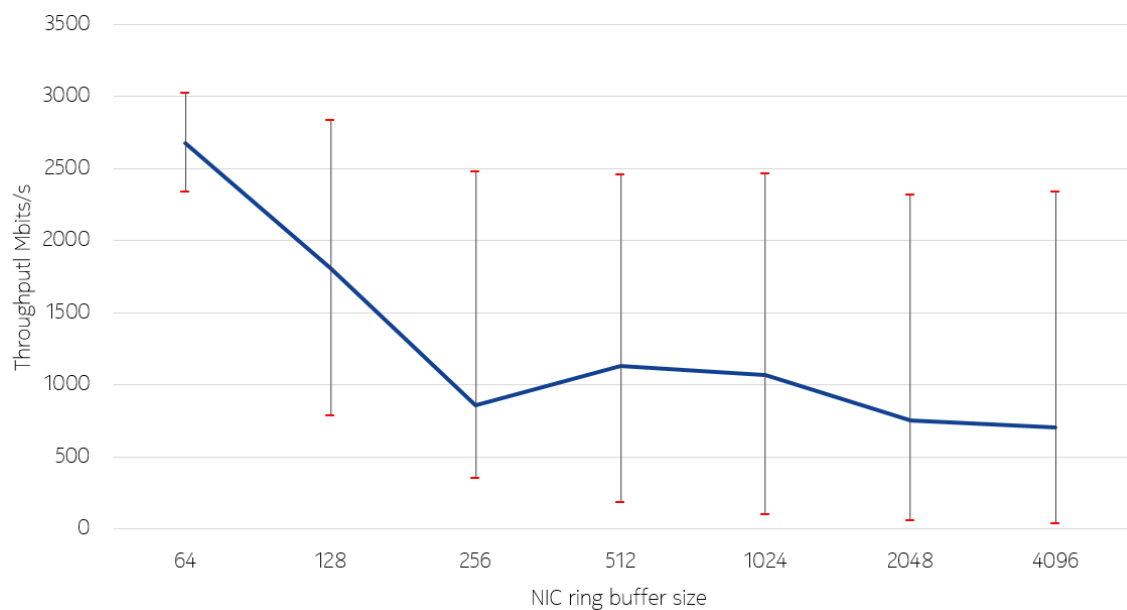


Figure 7.5: Compute host ring buffer size has huge effect on performance stability in the setup.

One low level configuration example is NIC buffer size. Figure 7.5 shows how basic buffer setting makes the system very unstable from a performance point of view. Each buffer size was tested three times over one minute window and then switched to next buffer size and this was repeated 34 times by a test script. The line presents the mean values from these measurements and low-high marks present the min-max range seen in the measurements.

## 7.4 Results

LoadGenerator and LTE Emulators can be deployed in multiple configurations as presented earlier. These configurations affect the system performance. In short, main configuration matrix consists of: 1) Load type: pure signaling or signaling and U-plane payload. 2) Environment: non-virtualized or virtualized. 3) Emulation level: fully emulated or partly emulated.



Pure signaling without U-plane payload connections will use very little resources and can be easily scaled on single machine to support millions of subscribers and thousands of base-stations if enough IP addresses are available.

U-plane source memory consumption varies according the payload application. For example, a compute host with 256 GB of RAM hosting 500000 UEs means that only a half a megabyte per UE is available. Compared to a smart-phone it is a tiny amount of RAM but for a simple IoT device it is plenty. In practice, payload applications have to use resources efficiently.

Table 7.1: Performance and resource consumption

	EMU RAM usage with 10000 UEs	Non-virtualized U-plane throughput	Virtualized U-plane throughput
Fully emulated	10 GB	1,2 Gbit/s	0,7 Gbit/s
With real EPC	30 MB	1,3 Gbit/s	0,8 Gbit/s

Table 7.1 shows an overview about memory consumption and U-plane performance in different configurations. The memory consumption value is an average from a larger setup and the throughput values are maximum values from single UE. In overall, emulated EPC is a big memory consumer and virtualization drops U-plane performance. On the other hand, virtualization offers a way to utilize host machine better by using multiple deployments that is not shown in the table. Scalability outperforms the losses of virtualization.

The performance of fully emulated setup depends also heavily on the amount of available CPUs when every LTE element can consume one CPU under heavy load. Also CPU pinning, Linux distribution, task prioritization etc. have a noticeable effect on the results, thus only a general rule should be extracted from these results. The absolute values may differ a lot between the environments.

## 7.5 Discussion

Horizontal scalability makes it possible to adjust setup size according the needs from small scale trials to stress testing, also in the future. However, the non-linearity in scalability has to kept in mind when scaling the system but it is possible to scale the system to support large scenarios. Standard server hardware is also cheaper than specialized testing tools and software components are open-source or developed in Nokia meaning that LoadGenerator can be taken use with low costs inside Nokia.

C-plane load generation requires much less resources compared to U-plane generation mainly for three reason. First, the performance variant (simplified) ENB is optimized for light operation. Second, the U-plane mapping adds over-head when network connections are created and connected to the emulator processes. Third,

the actual application generating the U-plane payload may not be optimized for the task.

Resource usage and performance also depends on the configuration size. When more elements are emulated, the more resource LoadGenerator uses, and the less CPU time is available per process if scaling is done on single host. Adding more hardware improves scalability, whereas external real elements reduce the load from LoadGenerator. External EPC is also needed to scale the core when emulated EPC can support only a limited size RAN.

Depending on the use case, the memory consumption varies from tens of kilobytes per UE on simple signaling scenarios to multiple megabytes per UE on complex scenarios with U-plane load. Decent server hardware with Linux host and guest systems should support throughput of a couple gigabits per second when EMUs are scaled properly. Cloud orchestration makes the installation and scaling easy but on the other hand standard cloud network stack can't offer the best performance.

To sum it up, LoadGenerator can be deployed and scaled easily. It can overload real LTE elements on C and U-plane, but the performance depends heavily on environment and configuration. Results confirm that LoadGenerator fulfills the targets that were set for the development.

## 8. CONCLUSIONS

Computing clouds offer scalability which was combined to network emulation in this thesis. The outcome, LoadGenerator, is a horizontally scalable tool for LTE network emulation. LoadGenerator has multiple use cases from basic research to LTE network element stress-testing.

This thesis also presented basic operation of LTE and the current status of cellular IoT. Cloud networking and orchestration were studied and used so that LoadGenerator's goals were met. The main technical problems were related how to scale and virtualize network efficiently with existing open-source components.

### 8.1 LoadGenerator as a tool

LoadGenerator is a proof-of-concept which has multiple limitations but the main features are working and the system can be developed forward. Thus, it can be said that the original idea has evolved into a tool which has a value: large LTE networks can be emulated and the system especially suits to IoT cases.

A programmable testbed, like LoadGenerator, can be customized for multiple use cases. Different scenarios can be configured to create live end-to-end demos, with real and emulated devices mixed. The results show that the concept is working in end-to-end configuration and it fulfills the basic goals that were set, and that the selected components are working. Next is the evaluation of how LoadGenerator can be used and what benefits it brings.

The basic feature for LoadGenerator is that it can achieve large loads on U- and C-plane and horizontal scalability provides capacity for future scenarios. The whole setup can be deployed on standard hardware with multiple configurations. It can be used to ease generation of single events to support development work but it can also be scaled up to support large scale stress testing. The ability to create load levels above production environment can be used in product verification.

A good testing tool is easy to use, has enough features, and can provide accurate results. LTE Emulators has not been designed for large scale load generation, but with some modifications the setup runs stably also in cloud environments. Deployment can be automatized and combined to familiar technologies like OpenStack orchestration to ease initial deployment. LTE configuration can be assisted with scripts that will spread selected configuration across the setup, but good LTE and

cloud knowledge is needed to solve out problems.

After cloud environment, networking, and support services are set up LoadGenerator can be deployed. EMU usage in a cloud environment can be compared to a Software as a Service model when local installations are not needed. When the network environment and configuration are deployed in sync, LoadGenerator offers the possibility to create ready-to-use service for its users. This extends the availability of LTE Emulators. Although LoadGenerator can configure itself quite well, changes in network configuration have to be told to the system by the user. Naturally LoadGenerator cannot remove all responsibility from the user but can greatly help by reducing mechanical steps. The detailed configuration is always up to the user and environment.

LTE Emulators have a basic configuration validator to find out certain basic configuration errors but it can not be used to validate large setups nor the cloud environment. The configuration validator can verify single configurations which might help in finding initial settings. If a setup is working with a couple elements in parallel, it should scale more if the same configuration logic is applied to new elements. In other words, find one working configuration and copy it to scale the setup. The same idea is behind the configuration scripts which automate mechanical steps during the configuration.

The fact that cellular networks have evolved through many standards and generations, with different options and settings, can cause surprises in compatibility. The method of trial and error is slow when the variable count is large. Sorting out compatibility problems can be challenging if low level debugging is needed. On the other hand, when real issues are found, they can be fixed, which is the purpose of testing. Already during the development of LoadGenerator, multiple features and bugs have been found from multiple elements.

The goal in this thesis was not to do exact validation or performance testing related to real LTE elements or IoT optimizations. Nevertheless, LoadGenerator and real elements can provide measurement data about the tested scenarios and information can be easily stored in a database or processed in real time. This data can be processed in multiple ways: test the functionality and maturity of new technologies and ideas to support the decision making, test real elements with realistic load, create visual demos about the functionality of network, etc. To sum it up, LoadGenerator has multiple use cases.

For innovation lab it is essential to have good tools. LoadGenerator adds the ability to load elements with realistic load which expands the available tool-set for the lab. This accelerates testing and innovation processes when it is possible to create scenarios to support stories behind the future technologies. End-to-end demos are essential in proofing of new concepts. When a lab with good resources

and tools is available, the work can concentrate more on the actual research and innovation.

For IoT LTE Emulators provide a modifiable environment where new functions can be added and old ones can be disabled. This gives a new dimension compared to simulations because real elements can be mixed with LoadGenerator and overall functionality can be tested, with the modifications existing in the system. The ability of early testing and error fixing helps to reduce costs in software development. In overall, the goal for LTE Emulators has been the ability to provide a testing pair, a functional element over a real interface during development when real devices are not yet existing for the tests.

Clouds and many other open-source technologies have entered the telecommunications industry and LoadGenerator is an example that all software in use is open-source except the LTE Emulators. Externally developed software requires monitoring and basic research to find out and use the right combination of technologies. In open-source community development is done in co-operation, i.e., everyone can contribute, and the results are shared to accelerate the overall process. On the other hand, the continuation of the work is not guaranteed and the goals might be different.

## 8.2 Future work

Future networks will be more diverse and more scalable, i.e., they will serve more devices and more specialized needs. This change has to taken into account during the development. However, realistic modeling of a network can be a complicated task; especially in large scale. To assist with the development, scalable research and testing tools should be developed to model futuristic networks.

This thesis used network emulation which is interesting technique to model networks that can be connected to real elements, too. Network emulation is less used compared to network simulation although network emulation combines some of the best sides of simulation and real life testing. Network emulation has potential to model future networks, also in very large scale, thus it could be used more in the future when network sizes are also growing.

LoadGenerator is an example tool which can support the evolution of telecommunication. The development of LoadGenerator will continue after this thesis is finished and the results have been promising so far. LoadGenerator definitely brings new opportunities inside Nokia although the life-cycle of LoadGenerator is open. From a scientific point of view, LoadGenerator could evolve into a system which could provide accurate measurements from different settings, technologies and network elements.

The scale-out of LTE Emulators in cloud resulted a very large network emula-

tion and it is clear that LoadGenerator has some potential to replace stress-testing tools, too. It is also possible to tune the system to do concept validation based on emulation, e.g., new IoT network optimizations. There are multiple use cases for LoadGenerator and the system has gained some interest inside Nokia.

The stress testing aspect is a never ending optimization game where the goal is to generate more, and more realistic traffic on the line with optimized resource usage. Also, new standards and features should be implemented into emulation as fast as possible. In short, these kind of tools will never be completed because they can always be optimized and developed further. Continuous integration should be applied to speed up development.

Another issue is the support for LTE Emulators themselves because it defines the LTE features. The concept of LoadGenerator could also be used in the future with 5G emulations to improve the next generation of network emulation inside Nokia. Currently it is hard to say what exactly happens with 5G and how it should be emulated. However, well-educated guesses can be made to develop 5G emulators already in 2016. Although IoT and especially 5G are not standardized terms, they are widely used and industry is investing these topics. LoadGenerator's goal is to support the development of these huge technological changes.

At the time when this thesis process ends, LoadGenerator is in a state that it could be taken into wider use with some limitations. LoadGenerator uses components which were fast to take in use, however they may not be optimal in the long run. This is often the case in innovation projects where the target is to prove the concept rather than make a ready-to-use tool. LoadGenerator should be refactored with proper software components to provide more stable, accurate and scalable tool for the future usage.

In overall, one sure thing is that the networks will grow bigger and bigger. Exponential growth will require new technologies and provide new research topics with multiple aspects. New topics can be found by monitoring open-source software and standards which are public. These public processes are like open windows to see what will likely happen in the future.

Bigger networks need also new type of tools for research. As said, network emulation seems to be one solution to model very large networks accurately. Higher link rates also impose new low level problems when latency is nearing to zero. This has to be taken into account when using software defined environments where strict time requirements can cause multiple problems. In practice, traditional network stack will have problems how to scale to very fast link rates like 10GE and above, which also affect network emulation.

New environments should be tested whether they can support network emulation. For example, scaling in a non-virtualized environment and building the setup on top

of containers should be studied. Specialized network frameworks are also interesting research topic which could lead to solving of multiple performance problems related to telecommunications, not to mention the possibilities of virtualized and software defined networking in overall.

In short, the programmable world has changed and will continue to change our lives. The pace is accelerating when more and more things are networked. Many small things can cause big problems when they are connected to old systems which have not been designed for massive scalability.

## REFERENCES

- [1] Statista. Internet of Things (IoT): number of connected devices worldwide from 2012 to 2020 (in billions). <http://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>, read 16.05.2016.
- [2] IBM. 2015. Redefining Boundaries: The Global C-suite Study. <http://public.dhe.ibm.com/common/ssi/ecm/gb/en/gbe03695usen/GBE03695USEN.PDF?>, read 28.12.2015.
- [3] Lima, J. 2015. Behold the 10 Biggest IoT Investments. Computer Business Review. <http://www.cbronline.com/news/internet-of-things/behold-the-10-biggest-iot-investments-4549522>, read 11.4.2016.
- [4] Gartner. 2015. Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015. <http://www.gartner.com/newsroom/id/3165317>, read 16.05.2016.
- [5] Lima, J. 2016. M2M, OT & CIoTO: 10 IoT trends looking to connect the industry this year. Computer Business Review. <http://www.cbronline.com/news/internet-of-things/m2m-ot-cioto-10-iot-trends-looking-to-connect-the-industry-this-year-4814241>, read 11.4.2016.
- [6] 3GPP. Progress on 3GPP IoT. [http://www.3gpp.org/news-events/3gpp-news/1766-iot\\_progress](http://www.3gpp.org/news-events/3gpp-news/1766-iot_progress), read 11.4.2016.
- [7] Xu, Z., Chen, Z., Sun, Q. 2013. Emerging of Telco Cloud. China Communications. <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6549261>.
- [8] OpenStack Foundation Report. 2016. OpenStack Foundation Report Accelerating NFV Delivery with OpenStack. <http://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf>, read 20.5.2016.
- [9] Beuran, R. ed. 2012. Introduction to Network Emulation. Pan Stanford Publishing.
- [10] Holma, H., Toskala, A. 2011. LTE for UMTS: Evolution to LTE-Advanced, 2nd Edition. Wiley.
- [11] 3GPP Portal. Releases. <https://portal.3gpp.org/#55934-releases>, read 2.5.2016.



- [12] Steward, R ed. 2007. RFC 4960 - Stream Control Transmission Protocol. <https://tools.ietf.org/html/rfc4960>.
- [13] Postel, J ed. 1981. RFC 793 - Transmission Control Protocol. <https://tools.ietf.org/html/rfc793>.
- [14] Postel, J ed. 1980. RFC 768 - User Datagram Protocol. <https://tools.ietf.org/html/rfc768>.
- [15] 3GPP. 3GPP TS 29.281 - General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U). <http://www.3gpp.org/DynaReport/29281.htm>.
- [16] 3GPP. 3GPP TS 29.274 - 3GPP Evolved Packet System (EPS); Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C); Stage 3. <http://www.3gpp.org/DynaReport/29274.htm>.
- [17] Fajardo, V ed. 2012. RFC 6733 - Diameter Base Protocol. <https://tools.ietf.org/html/rfc6733>.
- [18] NMC Consulting Group. LTE Identifiers. <http://www.nmcgroups.com/files/download/NMC.LTE%20Identifiers.v1.0.pdf>, read 12.5.2016.
- [19] 3GPP. 3GPP TS 29.214 - Policy and charging control over Rx reference point. <http://www.3gpp.org/DynaReport/29214.htm>.
- [20] 3GPP. LTE-Advanced. <http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced>, read 16.05.2016.
- [21] Gartner. 2014. Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business. <http://www.gartner.com/newsroom/id/2819918>, read 28.12.2015.
- [22] Gartner. 2015. Gartner's 2015 Hype Cycle for Emerging Technologies Identifies the Computing Innovations That Organizations Should Monitor. <http://www.gartner.com/newsroom/id/3114217>, read 28.12.2015.
- [23] Minerva, R., Biru, A. and Rotondi, D. Towards a definition of the Internet of Things (IoT). Telecom Italia and shared by IEEE. [http://iot.ieee.org/images/files/pdf/IEEE\\_IoT\\_Towards\\_Definition\\_Internet\\_of\\_Things\\_Revision1\\_27MAY15.pdf](http://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf), read 28.1.2016.
- [24] Evans, D. The Internet of Everything. Cisco. <https://www.cisco.com/web/about/ac79/docs/innov/IoE.pdf>, read 28.1.2016.

- [25] Nokia Networks. 2015. LTE-M - Optimizing LTE for the Internet of Things - white paper. [http://networks.nokia.com/sites/default/files/document/nokia\\_lte-m\\_-\\_optimizing\\_lte\\_for\\_the\\_internet\\_of\\_things\\_white\\_paper.pdf](http://networks.nokia.com/sites/default/files/document/nokia_lte-m_-_optimizing_lte_for_the_internet_of_things_white_paper.pdf), read 18.1.2016.
- [26] Wi-Fi Alliance. Wi-Fi HaLow. <http://www.wi-fi.org/discover-wi-fi/wi-fi-halow>, read 23.05.2016.
- [27] 3GPP. SA#71 - Release 13 frozen in Gothenburg. [http://www.3gpp.org/news-events/3gpp-news/1769-rel-13\\_tsg71](http://www.3gpp.org/news-events/3gpp-news/1769-rel-13_tsg71), read 2.5.2016.
- [28] 3GPP. RAN Evolution of LTE in Release 14. [http://www.3gpp.org/news-events/3gpp-news/1768-ran\\_rel14](http://www.3gpp.org/news-events/3gpp-news/1768-ran_rel14), read 2.5.2016.
- [29] WebRTC. <https://webrtc.org/>, read 18.5.2016.
- [30] MQTT. <http://mqtt.org/>, read 18.5.2016.
- [31] Mell, P., Grance, T. 2011. The NIST Definition of Cloud Computing. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>, read 30.10.2015.
- [32] Docker. What is Docker?. <https://www.docker.com/what-docker>, read 21.3.2016.
- [33] Felter, W., Ferreira, A., Rajamony, R., Rubio, J. [http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/\\$File/rc25482.pdf](http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf), read 27.1.2016.
- [34] Docker. Github. Proposal: exposing SCTP ports for container #9689. <https://github.com/docker/docker/issues/9689>, read 27.1.2016.
- [35] Marks, M. Unikernel Systems Joins Docker. <https://blog.docker.com/2016/01/unikernel/>, read 27.1.2016.
- [36] Intel. Intel Xeon Processor E5-2600 v2 Product Family with Intel C604 Chipset and Intel C602-J Chipset: Diagram. <http://www.intel.com/content/www/us/en/intelligent-systems/romley/xeon-e5-v2-c604-c602-j-chipset-ibd.html>, read 14.1.2016.
- [37] Kernel.org. Linux Base Driver for the Intel(R) Ethernet 10 Gigabit PCI Express Family of Adapters. <https://www.kernel.org/doc/Documentation/networking/ixgbe.txt>, read 27.1.2016.

- [38] Emmerich, P., Raumer, D., Wohlfart, F., Carle, G. Performance Characteristics of Virtual Switching. <http://www.net.in.tum.de/fileadmin/bibtex/publications/papers/Open-vSwitch-CloudNet-14.pdf>, read 27.1.2016.
- [39] DPDK.org. Homepage. <http://dpdk.org/>, read 27.1.2016.
- [40] Linux\* Base Driver for the Intel(R) Ethernet 10 Gigabit PCI Express Family of Adapters. <https://downloadmirror.intel.com/22919/eng/README.txt>, read 3.3.2016.
- [41] Netmap on GitHub. <https://github.com/luigirizzo/netmap>, read 17.5.2016.
- [42] OpenDataPlane.org. <http://www.opendataplane.org/>, read 17.5.2016.
- [43] Redis. <http://redis.io/>, read 17.5.2016.
- [44] Ubuntu. MAAS. <http://www.ubuntu.com/cloud/maas>, read 19.4.2016.
- [45] Heat. OpenStack. <https://wiki.openstack.org/wiki/Heat>, read 19.4.2016.
- [46] OpenStack. Resource Types. [http://docs.openstack.org/developer/heat/template\\_guide/openstack.html](http://docs.openstack.org/developer/heat/template_guide/openstack.html), read 19.4.2016.
- [47] OpenStack heat-templates on GitHub. <https://github.com/openstack/heat-templates>, read 19.4.2016.
- [48] Grinberg, M., OpenStack Orchestration In Depth, Part I: Introduction to Heat. <https://developer.rackspace.com/blog/openstack-orchestration-in-depth-part-1-introduction-to-heat/>, read 19.4.2016.
- [49] Iperf. <https://iperf.fr/>, read 17.5.2016.