UiO : Department of Informatics
University of Oslo

Measuring Performance of Web Content Delivery Over Mobile Networks

Using NorNet Edge Infrastructure

Diako Kezri

Master's Thesis Spring 2015

Measuring Performance of Web Content Delivery Over Mobile Networks

Diako Kezri

18th May 2015

Abstract

We explored the performance of the web content delivery over the Internet for two Norwegian MBB networks (Telenor and Netcom) using the NorNet Edge measurement infrastructure. The performance was evaluated by measuring the impact of several factors including the radio signal conditions, the ingresspoint placement and the networking topology in cellular networks when accessing the first byte of the web content of the top 50 Alexa websites. The time to get the first byte can be decreased by 80 percent, if the web content is placed close to the ingresspoint. The ingresspoint placement affects the network delay for domestic subscribers by only 10 milliseconds. There is no clear differences in time to get the first byte between 3G or 4G technologies. The time to connect to the web content (to query DNS and establish a TCP connection) is only 20 percent of the time spent to receive the first byte of the web content. The poor radio signal conditions can increase the time to get the first byte of the web content by 100 milliseconds.

Acknowledgment

I would like to express my special thanks of gratitude to the following people who supported me during this thesis and my entire period of master study at Oslo and Akershus University College of Applied Sciences and University of Oslo:

- My deep gratitude to my supervisor; Ahmed Elmokashefi at Simula Research Laboratory, as a great professor and also a wonderful person. Thanks for all his supports, helps, motivation, encouragements and guidelines to overcome the challenges faced throughout this thesis.
- Special thanks to my internal supervisor; Paal Engelstad as a knowledgeable professor, for his support, thoughtful advices and encouragements during this thesis.
- Special thanks to Hårek Haugerud as a great person and knowledgeable professor who coordinates this master study program at the university of Oslo, and helped us a lot beside our studies during these two years of master study.
- Thanks to Kyrre Begnum as a knowledgeable professor, for his great classes and for all technical and scientific knowledge that he thought us.
- Thanks to Ismail Hassan for his help and so many technical stuff that he taught us in this master program.
- Thanks to Anis Yazidi for all his help and advices during this master degree.
- Thanks to all my friends and classmates for their support and their kind friendship during this master degree program.
- I would like to say thanks to the University of Oslo, and to the University College in Oslo and Akershus for offering me this master degree program.
- I would like to thank my beloved family in my country and in Norway, especially my fantastic wife for their all support.

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 3
- 1.2 Thesis Outline . . . 3

2 Background **5**
- 2.1 Mobile Cellular Network . . . 5
  - 2.1.1 Generations of Cellular technologies . . . 5
  - 2.1.2 Ec / Io . . . 9
  - 2.1.3 LTE Networks (4G) . . . 11
- 2.2 Interdomain connectivity . . . 13
  - 2.2.1 Autonomous System Number (ASN) . . . 13
- 2.3 NorNet Research Testbed . . . 14
  - 2.3.1 NorNet Edge architecture . . . 15
- 2.4 Measurement Tools . . . 18
  - 2.4.1 ICMP . . . 19
  - 2.4.2 Ping . . . 19
  - 2.4.3 Traceroute . . . 20
  - 2.4.4 Nslookup . . . 21
  - 2.4.5 Curl . . . 22
  - 2.4.6 Whois . . . 23
- 2.5 Related work . . . 24

3 Methodology and Approach **25**
- 3.1 Objectives . . . 25
- 3.2 Measurement setup . . . 25
- 3.3 Measurement Procedure . . . 26
- 3.4 Tools and scripting language . . . 27
- 3.5 Collecting and recording data . . . 27
  - 3.5.1 Httpfetcher.py . . . 27
  - 3.5.2 Traceroute.py . . . 29
  - 3.5.3 Plots.py and Makeplots.py . . . 31
  - 3.5.4 Database.py and utils.py . . . 32

4 Results and Analysis **33**
- 4.1 Ingress point placement . . . 33
- 4.2 Network Delay . . . 34
  - 4.2.1 Node-to-ingress delay . . . 34

4.2.2 End-to-end delay . . . 35
4.2.3 Node-to-ingress delay ratio to end-to-end delay . . . 37
4.3 Routing path . . . 38
4.4 HTTP performance delay . . . 40
4.5 Time to connect to content provider's server in Netcom (3G and 4G) . . . 41
4.5.1 Time to start transfer the first byte of Internet content in Netcom (3G and 4G) . . . 42
4.6 Time to connect to the content provider in Telenor and Netcom 44
4.6.1 Time to start transfer the first byte of web content . . . 46
4.6.2 Radio conditions . . . 50

5 Discussion and Future work **55**
5.1 Evaluation of the network topology . . . 55
5.2 Evaluation of the Network delay . . . 55
5.3 Evaluation of HTTP performance under accessing web content 56
5.4 Evaluation of radio conditions . . . 56
5.5 Limitations . . . 56
5.6 Future Work . . . 56

6 Conclusion **59**

List of Figures

1.1 Global Mobile Data Traffic, 2014 to 2019 [28] 1
1.2 The UMTS MBB Network simplified Infrastructure 2

2.1 The simplified LTE Network Infrastructure 12
2.2 The cellular Network Core layout 13
2.3 An overview of NorNet Edge infrastructure 15
2.4 NorNed Edge UFO-board with four USB modems 16
2.5 NorNet Edge Backend [29] 17
2.6 Screenshut of NorNet Edge Visualization website [39] 18

3.1 The NNE nodes deployed in measurements [34] 26

4.1 Boxplot of the node-to-ingress delay and end-to-end delay in second (Telenor) 36
4.2 Boxplot of the node-to-ingress delay and end-to-end delay in second (Netcom) 36
4.3 The CDF of the node-to-ingress delay ratio to end-to-end delay (Telenor) 37
4.4 The CDF of the node-to-ingress delay ratio to end-to-end delay (Netcom) 38
4.5 The routing path between MBB operator and the content provider 39
4.6 Overview of the NNE nodes and regions used for measurements. 41
4.7 Boxplot of the time to connect in second (Netcom 3G) 42
4.8 Boxplot of the time to connect in second (Netcom 4G) 42
4.9 Boxplot of the time to start transfer in second (Netc 3G) . . . 43
4.10 Boxplot of the time to start transfer in second (Netcom 4G) 43
4.11 Boxplot of the time to connect in second (Telenor) 44
4.12 Boxplot of the time to connect in second (Netcom) 45
4.13 CDF of the time to connect in second (Telenor) 45
4.14 CDF of the time to connect in second (Netcom) 46
4.15 Boxplot of the time to start transfer in second (Telenor) . . . 46
4.16 Boxplot of the time to start transfer in second (Netcom) . . 47
4.17 CDF of the time to start transfer in second (Telenor) 48
4.18 CDF of the time to start transfer in second (Netcom) 48
4.19 CDF of the time to start transfer in second for top 3 websites (Telenor) 49

4.20 CDF of the time to start transfer in second for top 3 websites (Netcom) 49

List of Tables

2.1 Generations of Cellular mobile networks 7
2.2 Lists of characters and description that can appear in the traceroute output 21

3.1 Sample data collected in Curl measurements recorded in database. 29
3.2 Sample data collected in the traceroute measurements and recorded in database. 30

4.1 Statistics calculation of node-to-ingress RTT delays in milliseconds (Telenor) 34
4.2 Statistics calculation of node-to-ingress RTT delays in millisceconds (Netcom) 35
4.3 Statistics calculation of end-to-end RTT delay (Telenor) . . . 35
4.4 Statistics calculation of node-to-end RTT delay (Netcom) . . 35
4.5 RSRP and RSRQ conditions in 4G 51
4.6 Median of the signal conditions in dBm (Telenor 3G) 51
4.7 Median of the signal conditions in dBm (Telenor 4G) 52
4.8 Median of the signal conditions in dBm (Netcom 3G) 52
4.9 Median of the signal conditions in dBm (Netcom 4G) 53

Chapter 1

Introduction

Internet users are increasingly depending on mobile cellular networks for browsing web contents, streaming media, social network activities,and making VOIP calls irrespective of where they are and when they are using their mobile devices. Therefore, it is important that ISPs (Internet service providers) provide a reliable mobile network services with good performance for their subscribers and mobile users.

According to a report from the *Cisco Visual Networking Index* [27], Global mobile data traffic reached 2.5 exabytes per month at the end of 2014, up from 1.5 exabytes per month at the end of 2013. Figure 1.1 shows that overall mobile data traffic is expected to grow to 24.3 exabytes per month by 2019, nearly a tenfold increase over 2014.

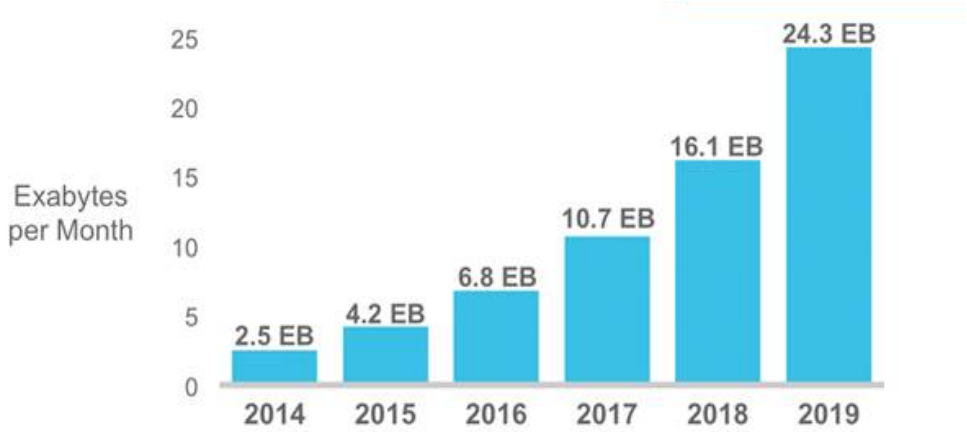

Figure 1.1: Global Mobile Data Traffic, 2014 to 2019 [28]

Despite the high popularity of mobile cellular networks, their performance remains unpredictable and at times questionable. This lack of predictability have encouraged regulators, network operators, and researchers to setup large measurement infrastructures and campaigns to gain insights into the different factors that impact the end-to-end performance of mobile networks. At the same time, content providers and operators have been

working on optimizing content placement and transport protocols to improve users experience.

This work studies the performance of web content delivery over two mobile broadband networks in Norway: Telenor and Netcom. Using existing measurement data from the NorNet project as well as a set of controlled experiment, it will isolate and study the impact of several factors on the performance of end-to-end web content delivery. These factors include the last-mile radio, the layout of the mobile core network and the wide-area routing configuration.

A cellular network also known as MBB network consists of two main components are shown in Figure 1.2: The Radio Access Network (RAN) and the Core Network(CN), more details described in chapter 2. These two components corporate together in order to connect end-users to web content. The factors that influence the performance of end-to-end web content delivery depend on the availability, stability and reachability of both components in MBB networks.

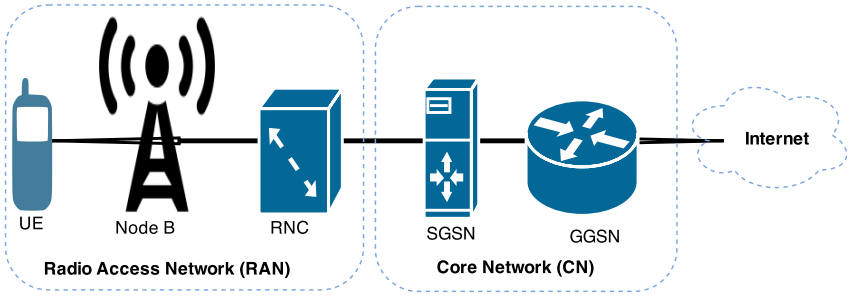

Figure 1.2: The UMTS MBB Network simplified Infrastructure

Last-mile radio refers to any telecommunication technology which links local mobile users to a cellular network by delivering radio signals between them [32]. The word "last mile" is used as a metaphor for describing the length of the last mile's link that may be more or less than a mile.[18].
The last-mile has a significant impact on the connectivity of the radio access network part. The radio signal conditions, the availability of connections, and the number of connection failures which my be caused by congestion or central failures of mobile network are the main metrics related to this component of MBB networks. [8]

Mobile network core layout refers to the carrier's topology of MBB networks and determines where traffic from mobile users enters the Carrier's core network[4]. The number of ingress points (the entry point between mobile users and the Internet) of mobile carriers, and their distance to the

mobile users have a significant impact on the network latency for the content delivery to mobile users.

Routing configuration has also impact on the performance of content delivery networks. Routers on the Internet path between end-to-end users and the web content providers play a significant role for network availability and stability which both has a direct influence on the performance of the end-to-end communication in a cellular network.[37]
The peering agreement between ASes is another reason of causing path inflation, e.g. the lack of presence of the carrier network in the same area of mobile users, or even the location where the carrier's network peers with the content provider's network as well. [4]

1.1 Problem statement

This study focuses on measuring the performance of the content delivery for mobile users in a cellular network by using NorNet Edge infrastructure. Using the NorNet Edge (NNE) measurement infrastructure for conducting the experiments provides a real-world and reliable test-environment which strives to increase the value of measurements and evaluation of mobile cellular networks. It tries to address the following questions in our research:

- *Breakdown web access end-to-en delay into its principal components.*
- *Identify and evaluate the placement of MBB carrier's ingress point.*
- *Investigating the impact of interdomain connectivity*

1.2 Thesis Outline

This rest of thesis has the following layout structure:

- **Chapter 2** (Background) provides a brief description over mobile cellular networks (MBB), an overview on NorNet measurement infrastructure, It describes the NorNet Edge component in more details and how it works. Later this chapter introduces several networking tools which are used for conducting experiments.
- **Chapter 3** (Methodology and approach) consists the methodology and explains the plan for conducting the experiments.
- **Chapter 4** (Results and analysis) describes the results and the analysis of the important metrics and parameters in methods for collecting data.

- **Chapter 5** (Discussions and Future work) dissects in depth the results, evaluates the them and explains the problems encountered in the experiments, and finally it discusses the future work.
- **Chapter 6** (Conclusion) Finally the questions in the problem statement section are asked clearly in this chapter.
- **Appendices** contains the scripts used for conducting the experiments. In addition some graphs and configuration files are displayed here.

Chapter 2

Background

This chapter introduces the mobile cellular network, the NordNet edge measurement infrastructure and several tools, applications and technologies which are used in the later chapters.

2.1 Mobile Cellular Network

A mobile cellular network also known as mobile broadband (MBB) is a high speed Internet broadband service which provides the wireless Internet access in a larger geographic coverage for end-users through different types of devices such as mobile phones, laptops, and tablets.

2.1.1 Generations of Cellular technologies

A new mobile generation is appeared approximately every 10 years since the first generation of mobile technology (1G) was introduced in 1981. The cellular concept was introduced in 1G technology also known as AMPS (Advanced Mobile Phone System) [9, 38], the first generation (1G) refers to the analogue transmission type of cellular technology which all mobile devices broadcasting analogue signals, therefor 1G made the large scale wireless communication possible.

The first wireless Internet access, which is referred to as 2G was introduced in 1991. 2G replaced the analogue technology with digital communication and improved the wireless communication quality. The third generation (3G) of cellular networks, also called wide area cellular network, enabled the high speed Internet access and video telephony. It is regarded as the first Mobile Broadband (MBB) networks. The 3G and 4G which have higher speeds become available respectively in 2001 and 2010. Among these generations 2G and 3G are widely used. Nowadays 4G is becoming more popular especially in developed countries.

Mobile broadband uses a spectrum between 225MHz and 3700 MHz, and there are various of standards which are used in different cellular technologies. For instance, Global System for Mobile (GSM), General Packet Radio Service(GPRS) and Enhanced Data Rate of GSM Evolution (EDGE) are known standards used by 2G technology where GPRS (2.5G) and EDGE (2.75G) are the enhanced GSM. Another standard which 2G partially using is the Code Division Multiple Access (CDMA). 2G networks can transmit non-voice data in form of Short Message Service (SMS), and Multimedia Message Service (MMS). 2G is also connected to the Internet for email and web browsing, but accessing the Internet is painfully slow because it can only transfer data around 14.4Kbps and up to 48kbps

3G MBB networks uses two main standards such as the Universal Mobile Telecommunication Services (UMTS) and CDMA2000. It also uses other standards such as the Wide band Code Division Multi Access (WCDMA), the High Speed Packet Access (HSPA), the Evolved High Speed Packet Access (HSPA+), the Dual Cell HSDPA (DC-HSDPA), and the Evolution-Data Optimized version of CDMA2000 (CDMA2000 EV-DO). Nowadays 3G is the de-facto minimum requirement for using a smartphone, it can support transfer speeds from 2Mbps and up to 14.4Mbps.

The 4G technology is based on the standards such as Long Term Evolution (LTE), and Worldwide Operability for Microwave Access (WiMax). 4G promises data transmission from 10Mbps up to 1Gbs. 4G promises higher data rates and expanded multimedia services, higher quality of service (QoS) and higher security

The 5G technology is not specified yet, and it is still in planning phase. It is is expected to have much higher network capacity and to provide multiple giga-bits per-second data rate for mobile users. It is also expected that 5G technology to operate in a very large bandwidth(multiple giga hertz) and to provide all_IP based model for wireless and mobile networks interoperability.[42]

Table 2.1 lists the generations of cellular mobile networks from 1G to 5G, with a brief information such as definition, Speed, technology, and some features about them.

Generation	Definition	Speed	Technology	Features
1G	Analog	2.4Kbps	AMPS,NMT,TACS	Voice only , no data service, Limited Capacity, poor availability
2G	Digital signals, Circuit data	14.4 kbps	TDMA,CDMA,GSM	Voice and short text, better quality and capacity
2.5G	Digital, Packet data	Up to 48kbps	EDGE,GPRS	Voice, data, email and web browsing
3G	Digital,Broadband, Packet data	Up to 2Mbps	IMT2000,FDMA, UMTS,CDMA2000, EV-DO	Voice, data, MMS, video streaming, smart phones
3.5G	Packet data	Up to 14.4Mbps	HSDPA,HSUPA, HSPA+, DC-HSDPA	Voice, data, video and TV steaming , interconnectivity
4G	Digital,Broadband, Packe data, All IP Network	Up to 1Gbps	WiMAX,LTE, Wi-Fi	Voice, data, video, Higher Definition quality
5G	Not yet	Multiple Gbps	Not yet	Higher speed, efficient use of bandwidth, support for Wireless World Wide Web (WWWW)

Table 2.1: Generations of Cellular mobile networks

2.1.1.1 UMTS networks (3G)

The UMTS network uses Wideband CDMA (WCDMA) [1] to carry the radio transmissions and often the system is referred by the name WCDMA. The UMTS employs a 5 MHz channel bandwidth and is designed for both, voice and the Internet data, compared to GSM networks which can only transmit voice. The UMTS network is compatible with 2G network and it can support multiple networks such as GSM, GPRS, EDGE, WCDMA, HSPA and the enhanced versions of them.

As shown in Figure 1.2 in chapter 1, the UMTS network is divided into two main components [8]: The Radio Access Network (RAN) and The Cor Network (CN), each consists of different components and functionalists.The RAN part is responsible for radio signal related functionality of calls, where the CN part is responsible for switching, routing and connecting to Internet.The RAN component of UMTS network

consists of the following elements:

- **User Equipment (UE)** e.g a mobile-phone, tablets and etc, links the mobile user to the radio interface.
- **Node B** is the term used to denote the base station transceiver. It consists of both transmitter and receiver for communicating with an user equipment (UE).
- **Radio Network Controller(RNC)** is responsible for controlling the Node base stations that are connected to it, RNC is also responsible for the encryption/decryption of user data sent from or to a User Equipment (UE). it is connected to the Serving GPRS Support Node (SGSN) in the Packet Switched element of the Core Network.

The CN component of UMTS network consists of two elements:

- **Serving GPRS Support Node (SGSN)** is responsible for delivering data packets from and to a Node base station in its geographical area. In addition, it is responsible for mobility and session management, for packet routing and transfer, authentication and billing functions.
- **Gateway GPRS Support Node (GSSN)** is the central element in a UTMS packet switched network. It is actually the link between Radio Network Controller (RNC) and external packet switched networks. Thus GSSN is like a sophisticated router that connects mobile networks to the Internet. In operation, when the GGSN receives data addressed to a specific user, it first checks if user is active, then forwards data to SGSN for that particular UE.

2.1.1.2 UMTS Enhanced technologies

There are several evolutionary technologies for UMTS based 3G networks which provide a higher data transfer speed [9]:

- **3.5G, High Speed Download Packet Access (HSDPA)** is a packet based data service in WCDMA downlink with data transmission up to 10 Mbps, it operates over a 5MHz bandwidth..
- **3,75G, High speed Uplink Packet Access (HSUPA)** is the uplink evolution technology in UMTS/WCDMA. It boost the uplink up to 5.74Mbps.
- **Evolved HSPA (HSPA+)** provides data rates up to 42Mbps in the downlink and up to 11 Mbps in the uplink, each link operates over 5MHz bandwidth.
- **High Speed Packet Access (HSPA)** HSPA is combination of HSUPA and HSDPA for both downlink and uplink speed enhancement.

- **Dual Carriers/Dual Channels HSPA (DC-HSDA)** is a Dual Channel form of HSPA+ that provides a greater downlink and uplink by using two 5MHz bandwidth spectrum in parallel instead of one. It provides data rates up to 42Mbps in downlink direction and 5.8Mbps in uplink direction.

2.1.1.3 Resiver Signal Strength Indicator (RSSI)

In a wireless networking environment the Receiver Signal Strength Indicator (RSSI) is a measure of the strength of the current radio signals received by the mobile equipments antenna. [31, 44]. RSSI can be used internally by the wireless networking card to indicate the power level being received by the user equipment's antenna which determines when a packet of information is ready to be sent. RSSI may be measured in mW (Milli-Watts) or dBm (Decibel-Milliwatts).

There is no standardization of RSSI parameters, Different producers providing the wireless networking cards and chipset have their own definition for the range of actual power and the range of minimum to maximum RSSI value (0 - RSSI_MAX). The higher the number, the better the signal. The exact number vary between cellular carriers. However , -40 dBm to -70 dBm usually indicates an excellent coverage area to the mode.

2.1.2 Ec / Io

The Ec/Io or the noise ratio is a measure of the quality of the signal from the tower to the modem and indicates the signal to noise ratio. The Ec/Io is the ratio of the received/good energy to the interference/bad energy, and it is measured in decibels (dB). If there is no noise level, so the Ec/I0 equals 0. Once the Ec/Io is below -5dB, it means that your connection is going to suffer. Several factors such as power suppliers, bad cabling, trees, hills, building, walls, wrong antenna polarization, and congestion at the power can cause a higher Ec/Io value.

2.1.2.1 Cell ID (CID) and Local Area Code (LAC)

A cellular network is a combination of overlapping small geographic areas/cells to coverage the bandwidth for a mobile user. In a UMTS network Cell ID (CID) is a unique number to identify each NodeBs. In UMTS and LTE networks a valid CID has a value range from 0 to 268435455. [6]

A local area is a set of NodeBs overlapping at their edge to optimize the signaling. In UMTS networks, multiple NodeBs are managed by a Radio Access Controller (RAN). A local Area Code (LAC) is a unique number to identify each local areal. Since there are large number of local areas, changing the location from one local area to another one, requires a mobile node to update it's network provider frequently.

2.1.2.2 UMTS channels structure

Th UMTS channels are categorized into three channels: [1]:

- **Logical channels** is responsible for defining the ways to transfer data, and what is transferred.
- **Transport channels** in corporation with logical channels to define how to transfer data.
- **physical channels** is responsible for carrying the payload data and controls the physical characteristics of the signal.

The transport channels are involved in how to transfer data, they consist of the following channels:

- **Dedicated Transport channel (DCH)** This is used to transfer data to a particular UE, which each UE has it's own DCH for both download and upload links.
- **Broadcast channel (BCH)** broadcast information to the UE in the cell in order to enable UE to identify the network and the cell.
- **Forward Access Channel (FACH)** carries data to the UEs that already are registered in the system, an UE can have more than one FACH per cell to carry data packets.
- **Paging channel (PCH)** alerts the UE for incoming calls, SMS, data sessions and etc.
- **Random Access Channel (RACH)** carries requests for services from an UE trying to access the system.
- **Uplink Common Packet Channel (CPCH)** is responsible for enabling additional capability beyond that of the RACH for an EU.
- **Downlink Shared Channel (DSCH)** for sharing among several users for large data such as data from web browsing etc.

2.1.2.3 UMTS handover

The RNC element of the RAN makes decisions about handover by monitoring continually the information about the signals being received by both the UE and the NodeB, when a specific link is below a given level and a better radio signal is available, it initiates a handover. The UE in this monitoring process is involved by measuring the Received Signal Code Power (RSCP) and Received Signal Strength Indicator (RSSI) and returns the information to Node A and then to RNC.

For any communication between the UE and the Internet in a cellular network such as UMTS networks, and before the transmission of any data, a user equipment (EU) has to attach itself to the network and establish a

Packet Data Protocol (PDP) context with the Gateway GPRS Service Node (GGSN). The PDP context contains information about the user's session and the IP address. When the PDP context is successfully established, the state of Radio Resource Controller (RRC) is checked by Radio Network Controller (RNC), so RNC assigns a shared (low bit rate) or a dedicated (high bit rate) radio channel for the user depending on the user's data traffic, thus RRC sets different state for the user equipment. [8]:

- CELL_PCH state if the user is not sending any data.
- CELL_FACH state when the user needs shared channel.
- CELL_DCH state when the user needs high bit rate channel.

The Core Network (CN) component of UMTS networks is responsible for routing of data traffic, and uses a hierarchical structure for both internal and external networks including the Internet. It uses GTP (GPRS Tunneling Protocol) when it works with TCP/IP to external networks and the Internet since all traffic goes through the GGSN which also known as ingress points.

In cellular networks the BTSs/NodeBs are widely distributed in order to provide a good radio signal coverage for mobile users, whereas the network carriers and service providers have a small number of GGSNs, each covering different size of geographical regions.

2.1.3 LTE Networks (4G)

The fourth generation of cellular networks, became available in 2010, is basically the extension to the 3G technologies with wider bandwidth and more services than 3G technologies. 4G offers new frequency bands, new transmission technology and with no backwards compatible. The 4G network is also known as all-IP packets switching network, multi-carrier transmission support, and ultra high speed. It uses a scalable channel bandwidths of 5–20 MHz, optionally up to 40 MHz.

The LTE 4G [9, 15] is expected to provide 100Mbps communication for mobiles users, and up to 1Gbs over fixed stations. It consists two fundamental componenets as depicted in Figure 2.1: The Evolved Radio Access Network (eRan) and the Evolved Packet Core (EPC).

The Evolved Radio Access Network (eRAN) is responsible for radio communication between eNodeB and Evolved Packet Core (EPC). It has no controller component like BSC in 2G or RNC in 3G UMTS networks. It consists of two elements [46]:

- **UE** is the User equipment such as smartphones, tablets and etc, it links the mobile user to the radio interface.
- **eNode B** in LTE MBB networks is directly connected to the network routers. there is no more intermediate controller as BSC in 2G, or

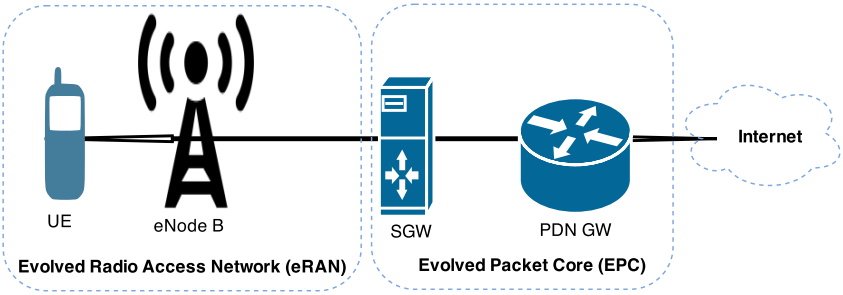

Figure 2.1: The simplified LTE Network Infrastructure

RNC in 3G. The architecture of LTE networks is simpler than the previous MBB networks, which means simplified network operation that provides better performance over the radio access interface by decreasing data transmission latency.

The Evolved Packet Core (EPC) is responsible for packet switching, routing and connection to external and Internal networks, and also to the Internet. It consists of the following elements:

- **Serving Gateway (SGW)** acts as a router between eNodeBs and PDNGW. Data packets are routed through this point from and to eNodeBs.
- **Packet Data Network Gateway (PDNGW)** communicates with external networks and the Internet. PDN GW has the same role as Serving GPRS Supporting Node (SGSN) in UMTS networks.
- **Mobility Management Entity (MME)** controls plane functions such as user authentication, user session, tracking area update. MME is linked to HSS which records the user subscription information in a database.

In Cellular Networks, when a user equipment (UE) moves from one cell to another cell and performs the cell selection or cell reselection and performs the handover, it needs to measure the signal strength and quality of all neighbor cells. In LTE networks, the user equipment has to measure two parameters related to the radio signals:

- **RSRP (Reference Signal Received Power)** It measures the average received power over the resource elements that carry reference signals in the certain frequency bandwidth of a specific cell.
- **RSRQ (Reference Signal Received Quality)**: It indicates the quality of received reference signal.
- **RSSI (Reference Signal Strength Indicator)**: It measures the average total received power observed by the the UE, and indicates the power from serving and non-serving cells, adjacent channel interference, etc.

RSRP is an RSSI type measurements and it is applicable in both RRC_Idle and RRC_connected modes in the procedure of cell selection or cell reselection, whereas RSRQ is only used in RRC_Idle mode. The RSRQ provides additional information when the RSRP is not sufficient to make a reliable decision for cell reselection or handover.

2.2 Interdomain connectivity

The end-to-end path in the routing data traffic can also be affected by how ASs and service providers are have relationship or peering as is depicted in Figure 2.2. A client request for Internet content can take a longer path due to the agreements between them [13].

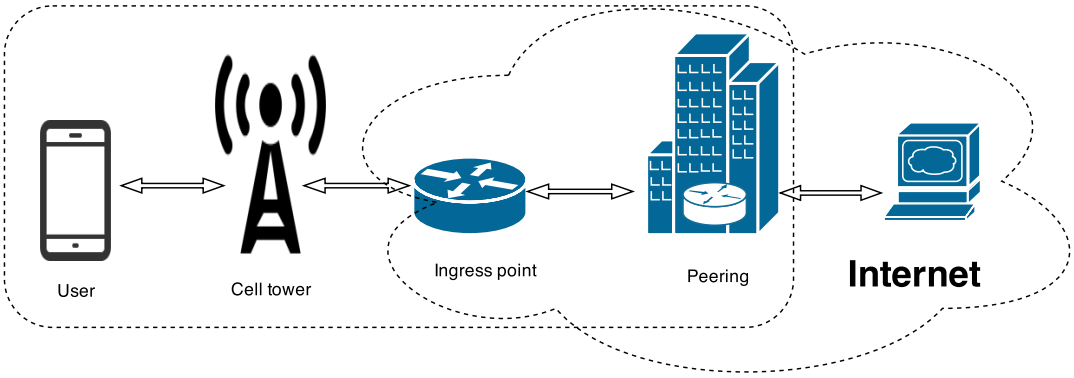

Figure 2.2: The cellular Network Core layout

2.2.1 Autonomous System Number (ASN)

An AS (Autonomous System) is a group of IP networks operated by one or more network operator(s) which has a single external routing policy. ASes use Exterior Routing Protocol such as the Exterior Gateway Protocol (EGP) to exchange routing information between them.

A public AS has a unique and global number (ASN), associated with it. So this number as an identifier of AS, is used for the exchange of exterior routing information between neighbor ASes. There are two types of ASNs: public ASN, and private ASN.

A public ASN is needed only when a AS is exchanging routing information with other ASes on the Internet, thus all routes belong to the AS are visible on the Internet. A private AS is used only if an AS communicates via the Broad Gateway Protocol (BGP) with a single provider, in this case the routing policy will not be visible on the Internet.

The carrier networks such as Telenor has the public ASN 2119, where as Netcom has the public ASN 12929.

2.3 NorNet Research Testbed

NorNet is an open, large scale and multi-homed Internet testbed distributed geographically across various locations. NorNet is built and operated by the Simula Research Laboratory and it is funded by the Research Council of Norway.[17]

NorNet provides researchers a programmable testbed for doing their measurements and experimental networking research [45]. The key element of NorNet testbed is to provide a real-world Internet testbed for research on Clouds, networking and MBB networks . Nowadays more and more applications rely on Internet connectivity, and the current Internet conductivities are not as robust as the should be. In order to provide a robust Internet connectivity and to make networks more robust, it is necessary that networks should be equipped by redundancy and multi-path transport. This idea was the motivation of building a multi-homing network which is connected to multiple Internet Service Providers(ISPs) simultaneously, and using different access technologies. NorNet consists of two components: NorNet Core and NorNet Edge.

NorNet Core [16, 19] is the wired part of NorNet [21]. Currently, it consists of 14 programmable sites which 11 sites of them is distributed over all parts of the country of Norway, one site is placed in Essen of Germany, one site is located in Karlstad of Sweden and one site is located in Hainan of China. Each site has at least two Internet connections which are connected to different Internet Service Providers.

NorNed Edge is the second part and also the complementary part of the NorNet testbed. It provides a dedicated infrastructure for measurements and experiments in Mobile Broadband Networks (MBB networks)[29]. NorNet Edge has the following features:

- Currently, NorNet Edge consists of more than 100 measurement nodes which together provide an entire networking.
- It is distributed nation-wide across various locations in Norway which includes major cities and even remote islands.
- Each Node is powered by a fully programmable computer running a standard Linux distribution for maintaining and conducting experiments.
- Each node has often access to wired and wireless networks and is connected to at least two MBB networks with different Internet access technologies such as 3G and up to 4G.
- Networking tools and programs are installed on each Nodes which provides meta data information such as cell ID, connection mode, and radio signal conditions.

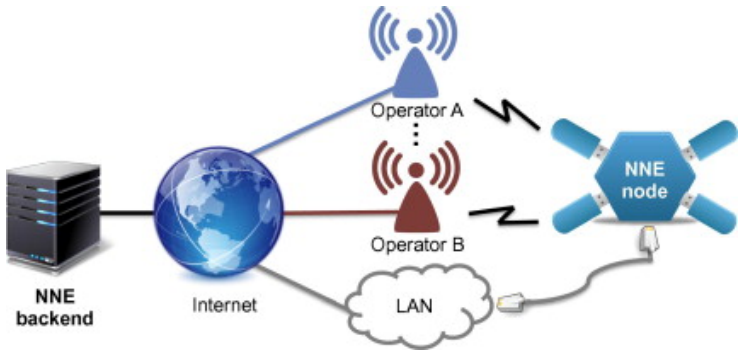

Figure 2.3: An overview of NorNet Edge infrastructure

2.3.1 NorNet Edge architecture

The NorNed Edge [29] consists of two main components: a large set of nodes (over 100 nodes) running a standard Linux Distribution, and a central backend-system consists of a set of servers responsible for management, configuration, monitoring, deploying nodes and for visualization of the status of nodes. Another task the backend-system doing is to managing the experiments and measurements on nodes. Both components are described in more details in sections [2.3.1.1] and [2.3.1.2].

2.3.1.1 NorNet Edge node (UFO-board)

Fig 2.6 shows an NNE UFO-board which is connected to up to four USB modems. An UFO-board consists of the following componenets and tools:

1. A Samsung S5PV210 Cortex A8 which has 1 GHz processor, 512 MB RAM, 512 NAND flash memory, 16 GB SD card storage, it also has one fast Ethernet port and 7 USB ports.

2. Each NNE node is powered by a Linux Debian distribution with a 3.11.x kernel as its operative system.

3. The UFO-board also has 1-4 UMTS modems for providing up to 4G Internet connections. All UMTS modes are of type Huawei E353-u2 3G modems and they supports GSM technology up to HSPA+ (3.75 G) and LTE (4G). One advantage of this kind of modems is that they can collect and report meta-data on network connection mode and submode, cell ID and signal strength.

4. In addition to UMTS modems, each NNE node is equipped by 1 CDMA modem which is of type EV-DO modems. It provides connection to Ice which operates in a different Frequency Band (450 MHz).

5. One WLAN is optionally available for some NNE nodes which is useful to provide Wi-Fi connections in nodes or to turn nodes to Wi-Fi access points for Internet connections by using MBB connections.

6. NNE nodes are equipped with a set of tools and programs, which provides developers and researchers easy access to nodes, uploading files, performing some operation to conduct measurements and to collect data about the state of MBB connections. The following tools and applications are installed on all NNE nodes:

    - **SSH**: the secure shell (SSH) used for accessing the NNE nodes remotely from other hosts or servers.
    - **MULTI**: is a command line network manager with multi-homing functionality which configures routing tables when multiple interfaces are running in parallel.
    - **usbmodem-listner**: is a daemon program written in Python which is responsible for modem management, cellular connection management, metadata management.

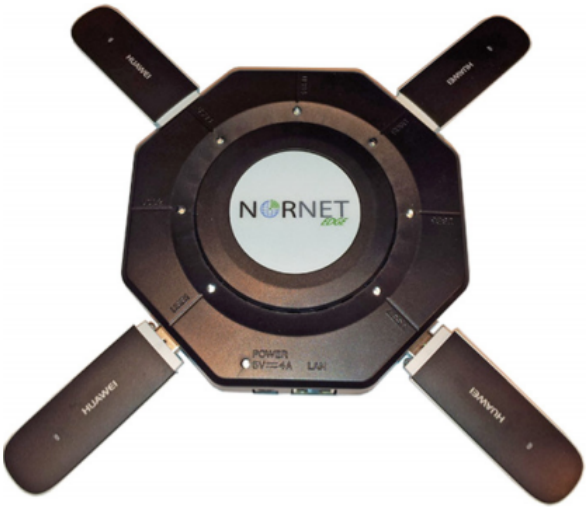

Figure 2.4: NorNed Edge UFO-board with four USB modems

2.3.1.2 Backend system

The backend system as stated before, consists of a set of servers in order to maintain nodes and to visualize the status on nodes in order to monitor them. Figure 2.5 shows an overview of the backend system and it's components used in NorNet Edge.

- **Puppet** is an open source configuration management utility [30], is used for managing, updating and maintaining the NNE nodes. It is useful for developers and researchers to control and manage servers/nodes centrally from a single server.

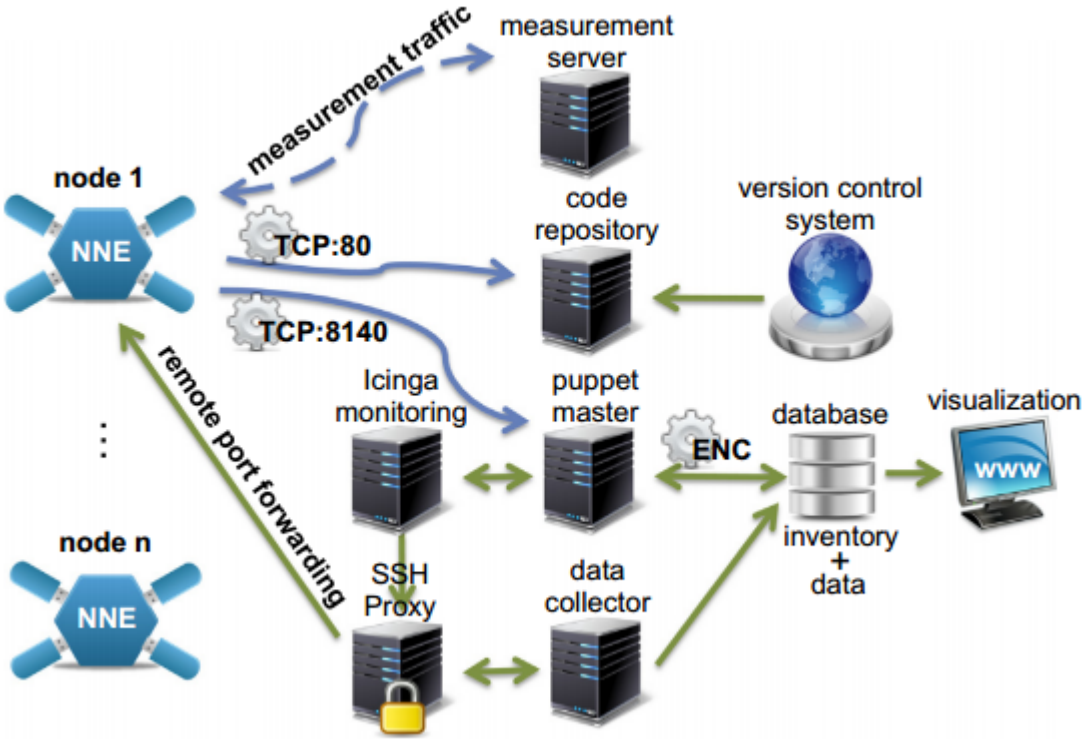

Figure 2.5: NorNet Edge Backend [29]

- **Icinga** [24] is an open source network and computer system monitoring application.It is a fork of the *Nagios*[33] system monitoring application developed in 2009. It is commonly used for visualization the service status, network maps, reports, logs of hosts. In the NorNet Edge system backend Icinga reports the status of the node's resources such as disk-space, CPU-Usage,Memory and etc.

- **Mysql Database** [2] is most widely used as open-source relational database management system(RDBMS). It contains information about NNE nodes like location, address,contact person, and administrative messages. In addition it records data collected from measurement experiments.

- **Version control** records changes to the source code and files of measurement applications and other software over time.

- **Repository and deploying server** is interactively connected to the Version control and Puppetmaster server . It retrieves new versions of measurement applications, new packages and other software from the Version control, then the Puppetmaster deploy it on the relevant NNE nodes.

- **Data collector** Inserts the collected measurement data from NNE nodes into Mysql database. The data collector can also perform

some analytical operations on the collected data before saving them in database.

- **Measurements server** runs the measurements need to be run actively from the serverside. Although from the Measurement server, other experiments can be run remotely on the Internet.

- **Virtualization of NNE nodes** Figure 2.6 shows the screen-shot of the website, created by NorNet project, for visualization the real-time status information of all NNE nodes including the status of MBB connections of each node. A node may consist of several operators which can be filtered by choosing a special operator for displaying the status information of it. The status of the MBB networks related to each nodes can be displayed in different collors e.g. red color means the connections is unavailable.
  The visualization website provides researchers and experimenters to get a realtime information on their network and measurement nodes. In addition, it shows the performance of different MBB operators that can be a good point for individuals that are interested in assessing and comparing the performance of different MBB operators.

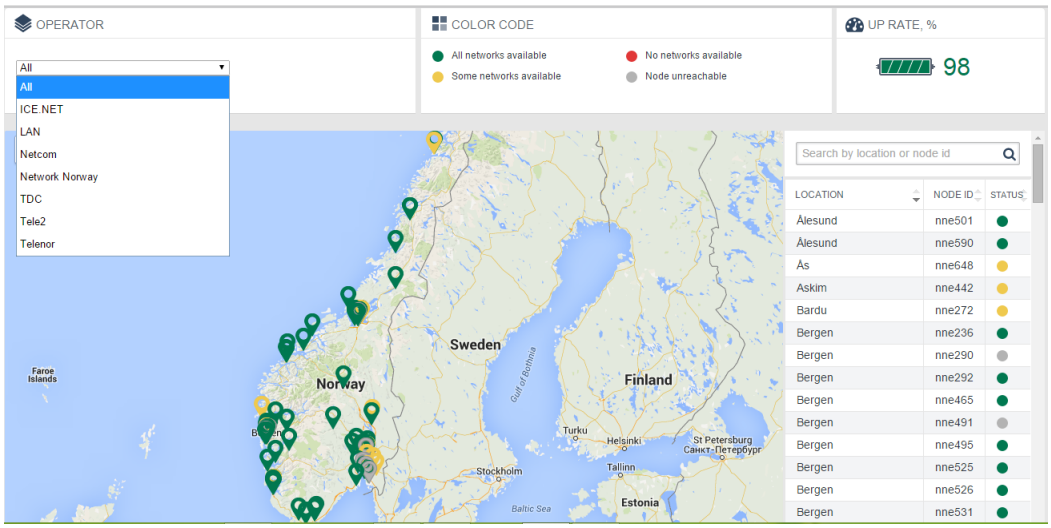

Figure 2.6: Screenshut of NorNet Edge Visualization website [39]

2.4 Measurement Tools

This section gives a description on protocols and utilities used for conducting the experiments, discovering and measuring the Network topology and routing of a cellular network.

2.4.1 ICMP

ICMP is a fundamental part of the TCP/IP protocol suite, which operates in the Internet layer of TCP/IP. ICMP is widely used in the Internet to diagnose and detect the network problems. ICMP sends a message notification known as ICMP error message to inform that the network devices about a detected failure, for example, while a network's router detects a network problem under forwarding an IP packet, it will usually sends an ICMP error massage to the source of packet to inform it a network problem is happening. it will also send an ICMP error massage when a host or server is not available or could not be reached in the internet.[25].

Contrary to the transport protocols such as TCP and UDP protocols, ICMP is not used for transferring data between end-points. All ICMP messages are contained in a standard IP packet. Each ICMP packet uses 8 bytes of IP header and different size of payload in order to send ICMP messages. The ICMP header is divided into type, code and checksum. [20]

ICMP generates many messages which are identified by "type" field which is a digit. some of known messages are as below:

- *Echo reply* (type: 0)
- *Time exceeded* (type: 11)
- *Destination unreachable* (type: 3)

Many of these ICMP types have digit "code" fields e.g. some of the "code" fields of "Destination unreachable" are:

- code 0: Net Unreachable
- code 1: Host Unreachable
- code 2: Protocol Unreachable
- code 3: Port Unreachable
- code 6: Destination Network Unknown
- code 7: Destination Host Unknown

ICMP has no authentication method, thus it may be used by attackers in order to perform som attacks such as *Denial of Service*(Dos), *Ping flood*, *Ping of death*, and *ICMP tunneling*.

2.4.2 Ping

Ping is a networking utility written by Mark Muuss in 1983, which is used by network and system administrators to determine whether or not a specific host/server is accessible over Internet protocol. Ping is an ICMP based

tool and works by sending a packet(s) from a source address to a destination address and waiting for a reply, if the destination address was able to send back a response at a specified time out, it means that the host is available. It is freely available in all Unix/Linux and Windows systems. It helps to troubleshooting networking issues.

Ping [26] may also be used for calculating the lost rate of packets and for testing the latency/delay time which is known as Round Trip Time (RTT) in communication between two hosts. Ping can be issued by predefining the network interface e.g. (-I eth0), IP address, the number of bytes e.g. (-s 1400), the number of packets e.g. (-c 3), time to live and etc from a host.

Since Ping is an ICMP based tool, and due to ICMP vulnerabilities which it may be used by attackers, some operating systems, routers and firewalls may filter or restrict Ping, which results in efficiency reduction of using Ping by network operators.

2.4.3 Traceroute

Traceroute is a networking utility, written by Van Jacobson[10], that traces packets from a computer to a host over Internet and discover the route that packet travel through to their destination. The Traceroute output shows how many hops the packet requires to reach the host and how long each hop takes. Traceroute is widely used by network operators to troubleshooting network problems such as routing failure, routing misconfiguration and network latency [7].

Traceroute [26] works by sending a sequence of three User Datagram Protocol (UDP) packets to an invalid port address at the host remote host, each UPD has a Time To live (TTL) field value set to one. The TTL value of one causes the datagram to timeout when it reaches the first router through the path, thus the router replies with an ICMP Time Exceeded Message (TEM) indicating that the datagram has been expired. Another Three UDP packets is sent again, each with a TTL value set to two, which causes the second router in path replies with an ICMP Time Exceeded Message (TEM) message is occurred by the first sequence of UDP packets. This process continues until the packet reaches the actual destination host, since these UDP packets are trying to access an invalid port address at the destination host, an ICMP Unreachable message is returned by destination host indicating that the port is unreachable, which terminates the Traceroute program is finished. The Round-Trip-Times (RTT) of the packets received from the successive host on the routing path is recorded in milliseconds; the sum of the mean times in each hop indicates the total time used to establish the connection.

Recording the source of each ICMP Time Exceeded Message provides a trace the packet took in order to reach the destination host. Traceroute can be issued with "-m" option up to 255 hops in Traceroute command.

The default number of packets is set to four which can be changed by "-q" option e.g. (-q 10). In order to specify the network interface, the "-i" option e.g (-i eth0) can be used.

Traceroute has some backdraws and limitations as [22]:

- Traceroute is not able to discover backward path which is knows as *Unidirectional*, the path from destination host to the source host, which is different from the path from host to destination.
- Traceroute is not able to avoid the Internet path loadbalancing in routers, which many be done by ISPs in order to increase performance of their network. This issue may result in discovering false links by Traceroute.[43]
- A large number of Traceroute probes to the same network interface may lead to an issue known as *redundancy* in which the router consider the Traceroute probes as Denial of Service (Dos) attack.
- Traceroute is not able to reveal hidden routers such as *Multiprotocol LabeL Switching* (MPLS). [14]
- Some routers restricted the Traceroute probes and they reply with a limited number of ICMPs e.g only one ICMP message per second, thus Traceroute is not able to discover this kind of routers known as *Anonymous Routers*, therefor Traceroute displays a asterisk "*" in the output.

A list of characters whit the description of them is shown in Table 2.2

Character	**Description**
	For each node, the round-trip time in milliseconds for the specified number of probes
*	The probe timed out
A	Administratively prohibited (example, access-list)
Q	Source quench (destination too busy)
I	User interrupted test
U	Port unreachable
H	Host unreachable
N	Network unreachable
P	Protocol Unreachable
T	Timeout
?	Unknown packet type

Table 2.2: Lists of characters and description that can appear in the traceroute output

2.4.4 Nslookup

Nslookup is networking tool for querying the Domain Name System (DNS) to obtain domain name, or mapping IP-address and any other specific

DNS records. Nslookup is used by network and system administrators to troubleshoot and DNS related problems.Following DNS records can be queried by Nslookup:

- IP address (A record): displays the IP-address of the DNS server for a given domain name.
- name server (NS record): it maps the domain name to a list of DNS servers authoritative for that domain, which means that it provides the name servers which are associated to a given domain name.
- mail exchange (MX record): obtains a list of mail exchange server for a given domain name.
- Start of authority (SOA record): provides information such as the authoritative information about the domain, the mail address of the domain admin, and the serial number of domain.

2.4.5 Curl

Curl [40] is a network application used in command line for transferring data with URL syntax for instance to transmit and receive HTTP requests and responses. Curl is free and open software that compiles and run under variety of operative systems such as Linux distributions, Mac OS X and Microsoft Windows.

It supports protocols such as DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMTP, SMTPS, Telnet and TFTP.

curl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, HTTP/2, cookies, user+password authentication (Basic, Plain, Digest, CRAM-MD5, NTLM, Negotiate and Kerberos), file transfer resume, proxy tunneling.

Curl offers proxy support, user authentication, ftp upload, HTTP-Post, HTTP-GET, SSL connection, cookies, file transfer resuming and etc. Curl has around 150 flags and options that can be used for issuing commands on a remote host, for instance with "-I" sends a head request to an URL address on Internet, "-v" or verbose option shows the header request information, "-s" option silences the Curl's progress output, with "-o" followed by path to a file, it redirects the output to file.

There are several available variables can be used with Curl command in order to customize the output from response headers are as following [41]:

- **http_code** : A numerical code obtained from last retrieved HTTP(s) page.

- **time_pretransfer**: The time in seconds it took from the start until the file begins to be transferred.
- **time_starttransfer**: The time in seconds it took from the start until the first byte is just about to be transferred.
- **time_connect**: The time in seconds Curl took from start until the connection to the remote host was completed.
- **time_total**: The total time in seconds,which displayed in millisecond resolution, that full operation lasted.
- **redirect_url**: It will show the actual URL a redirect would take to when the option "-L" is not used in the HTTP request.
- **url_effective**: The URL that was obtained last.
- **size_download**: The total amount of bytes that were downloaded.
- **size_request**: The total size of bytes that were sent in HTTP request.
- **speed_download**:The average download speed measured by Curl to complete download.
- **num_redirects**: Number of redirects that followed in the request.

2.4.6 Whois

Whois [11]is a query/response protocol which is widely used for query databases that hold information about Internet resources such as Autonomous System Number (ASN), domain names, IP addresses allocations and etc. It is used by network and system administrators and Internet operators to discover and reveal individuals or entities responsible for network operations on the Internet. [12]

Whois protocol was first described by **Ken Harrenstein and Vice White** in 1982. The latest and most significant update of Whois protocol was published in 2004. It has both web-based and commandline services . It operates via the communication between a Whois server and a Whios client. A Whois server listens on TCP port 43 for requests from Whois clients. The Whois client sends a text request to the Whois server and the Whios server replies with text content. The Whois server terminates its connection when the output is received by the client.

Whois is a simple protocol without any authentication mechanism between the end-points, and the output structure is not user-friendly which transform it to one of the complex protocols to work with.

2.5 Related work

A research by Dziugas Baltrunas et al [8] measured the reliability of MBB networks in Norway using NorNet Edge infrastructure with dedicated nodes. They used end-to-end measurements in order to identify reliability, failures and performance problem of different operators. Their results showed that networks differ in connections stability, packet loss patterns, ability to support applications.

A study by Kiriakos Zarifis et al [4] provided the taxonomy of causes for path inflation of mobile client traffic in the USA. They used a dataset collected from mobile devices subscribed by four major carrier's in the USA.They identified the reasons behind the causes and quantified them based on their impacts. Their study showed that *lack of carrier's ingress points* or *lack of provider peering points* can cause lengthy latency in mobile traffic. Although they had found that evolving the carrier's and provider's topology would improve the routing.

Another research done by John P.Rule et al[3] on celluar DNS for content replica selection in 4G networks. They measurement dataset gathered from mobile devices using four major USA's carriers and two other carriers from South Korea. Their work showed that due to cellular DNS's poor localization, using the public DNS has equal or even better performance over 75% of the time.

This study differs from the previous studies in combining all major factors that influence the performance of web content over cellular mobile networks. Several factors such as the last-mile radio, the network topology, the routing path and DNS resolvers will be studied through this work. The experiments will be performed using the NorNet testbed with dedicated nodes of 5 major network carriers and service providers in the country of Norway.

Chapter 3

Methodology and Approach

This chapter explains measurement setup, measurement procedure, data collection, and the assumption and limitations in experiments and measurements to achieve the final goal.

3.1 Objectives

As mentioned in chapter 1, this work aims to study the performance of the content deliver over three Mobile Broadband networks (MBB networks) in Norway using NorNet Edge testbed. In the experiments, the last-mile radio signals, the ingress point of MBB networks, and the path inflation types are studied and evaluated in order to compare the performance of three major MBB networks in Norway based on measurements.

3.2 Measurement setup

NorNet Edge (NNE) is flexible platform for measurements and experiments in MBB networks is used to evaluate the properties of the network interacted with the transport/application layer protocols in experiments. NNE provides a large number of nodes distributed geographically and connected to multiple MBB networks.

There are two mobile broadband operators Telenor and Netcom in Norway used for carrying out the measurements in this study.Both Telenor and Netcom are operating on their own Radio Access Network (RAN) with nation-wide coverage. Telenor and Netcom have 2G, 3G, and 4G coverage nation-wide in Norway.

The measurements are carried out using 16 NNE nodes distributed in different geographical regions over Norway, as depicted in Figure 3.1. NNE nodes were chosen from different places in Norway to simulating the nation-wide coverage of MBB networks. NNE nodes are connected through modems to 1-3 MBB networks for accessing the Internet.

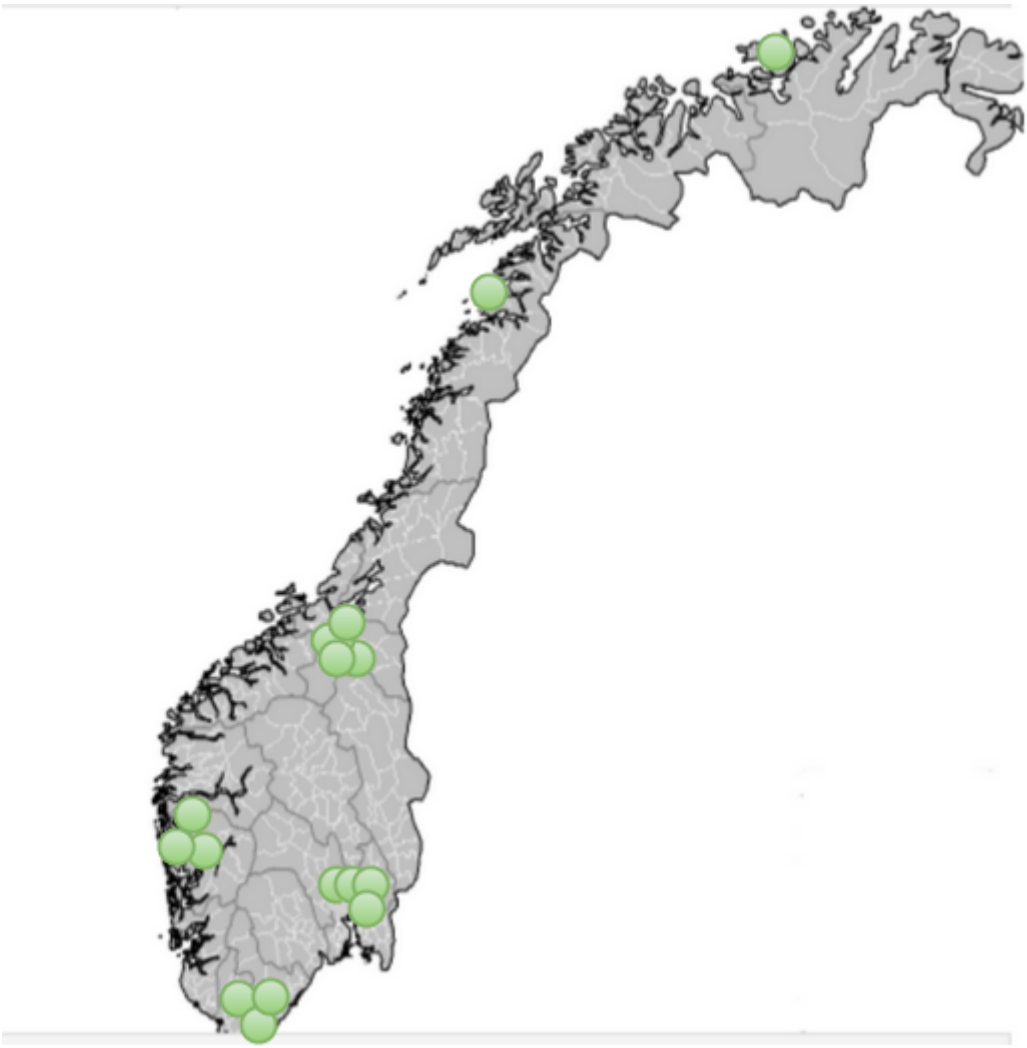

Figure 3.1: The NNE nodes deployed in measurements [34]

3.3 Measurement Procedure

The experiments will be carried out firstly by collecting appropriate data on NNE nodes, later extracting desired data from collected data. The experiments will be repeated for one week to collect data at different time periods such as during ordinary days and nights, and in the weekend. Running experiments in different periods of time gives more reliable data due to various data traffic and radio signal conditions.

The measurements are carried out by sending different types of data packets such as Ping, Traceroute, and HTTP request from NNE nodes to a list of top 50 websites collected from **Alexa.com** ranking system categorized in different regions across the world. Using vast numbers of end-to-end points will give different routing path, network latency and connectivity, and robust results.

3.4 Tools and scripting language

Python [36] is chosen as programming language to carry out the measurements. Python is an open source interpreted programming language for general purpose. It is a cross-platform and object oriented programming language which has a large and comprehensive standard library especially for developing scripts relevant to network and system administration. .

Matplotlib[23] is an open source plotting library written by John D. Hunter. Matplotlib can be used in Python scripts, web application servers and graphical user interface toolkits. It allows students and researchers to visualize mathematical functions, numerical data and signal processing.

The networking tools such as Ping, Traceroute, Curl, and Whois were described in chapter 1, are used in the Python scripts and with the measurements.

3.5 Collecting and recording data

Five Python scripts are developed and deployed on NNE nodes which are as following:

3.5.1 Httpfetcher.py

This script are executed on NNE nodes by accepting two arguments (the measurement instance id and the network interface) to specify which MBB connection it should run. There are more than one MBB connections on NNE nodes, so the script runs for all MBB connections by specifying their network interface. The script runs in an infinite loop until 7 days, it sends HTTP_GET request to the index file of the top 50 websites ranked in *Alexa.com* [5] and records the output in separated files, then export them to the database. It carries out the following operations for each of the top 50 websites from each MBB connections on the nodes:

1. assign a dedicated radio transport channel for the MBB connection, specified in the argument, by sending a Ping request packet of size 1500 bytes.
2. run the "nne-query-metadata" program for gathering the metadata information for that MBB connection.
3. send the HTTP request to that website.
4. run again the "nne-query-metadata" program for gathering the metadata information for that MBB connection.
5. record the metadata information and the HTTP response in a temporary file located in "/tmp" folder.

6. run the operations 1-5 were repeated for all the top 50 websites.

The **httpfetcher.py** script generates an output of 46 columns in database which contains two type metrics: the metadata metrics on nodes and the HTTP_Get metrics when access to the web content of the top 50 websites. Some examples of the columns from both metrics are represented here:

Metadata metrics

The metadata data information contains several metrics related to the last-mile radio when NNE nodes accessing the content on the Internet. Among these metrics:

- **node_id**: The nodes identification number.
- **network_id**: The operators identification number.
- **ma_mode**: The connection mode (3G,4G).
- **ma_rssi**: This column is the Arbitrary Strength Unit (ASU) as an integer value proportional to the RSSI value measured on NNE nodes from the metadata collector.It will be converted to dBm in order to get The radio signal strength indicator (RSSI).
- **ma_rsrp**: The Reference Signal Received Power, The average received power of a single resource element.
- **ma_ecio**: The Ec/Io or the noise ratio is a measure of the quality of the signal from the tower to the modem and indicates the signal to noise ratio.
- **ma_rsrq**: The Reference Signal Received Quality.
- **ma_rscp**: The Received Signal Code Power, the power measured by a receiver on a user equipment. It is the sum of RSSI and Ec/No, both values in dBm.

HTTP GET metrics

It contains information on the HTTP performance when accessing a web content on the MBB networks from NNE nodes. The response to HTTP request from the service providers server contains the following HTTP parameters collected in **Curl** measurements which described in chapter 2:

- **c_cite**: The top 50 websites.
- **c_http_code**: The HTTP code in HTTP Get request.
- **c_t_connect**: The amount of time in seconds for connect to the content providers server. It is the time to connect is the time to establish the TCP connection to the content provider's server by sending HTTP Get request from NNE nodes, it occurs after the DNS lookup for mapping the IP address to domain name is performed.

- **c_t_starttransfer**: The amount of time in seconds to start to transfer the first byte from the content provider's server. It is the time to start transferring the first byte of the requested resource from the nodes. The time to start the first byte takes in place after the DNS lookup and the end-to-end connection is established.

A sample of the **httpfetcher.py** output recorded in database is shown in Table 3.1:

node_id	network_id	ma_mode	ma_rssi	c_site	c_http_code	c_t_connect	c_t_starttransfer
321	2	5	15	bing.com	200	0.169	0.601
424	1	6	19	ask.com	200	0.891	1.484
452	2	5	10	gooogle.com	200	0.116	0.603
465	1	6	15	apple.com	200	0.163	0.6
582	1	6	15	msn.com	200	0.385	0.84
599	3	6	26	yahoo.com	200	0.223	1.027
645	1	6	14	163.com	200	0.109	0.494
702	3	6	31	youtube.com	200	0.135	1.28
706	3	6	17	twitter.com	200	0	0.268
707	3	6	25	qq.com	200	0.125	0. 56

Table 3.1: Sample data collected in Curl measurements recorded in database.

3.5.2 Traceroute.py

This script are executed on NNE nodes by accepting three arguments (the cell identification number, the network interface for MBB connection, and the time-sleep value). The script records information about the hops and time spent to reach each hop in the routing path between the NNE nodes and the service providers (top 50 websites) in files, then exports the data to a database. The script run in an infinite loop until one week, for performing the following operations:

1. start with the first website listed in top 50 websites.
2. run traceroute program by sending 10 packets to the website.
3. record result in a temporary file (the name of files is a combination of the measurements instance_Id and timestamp) located under "/tmp" folder.
4. run the operations 1-3 for all top 50 websites.

The website **robustenett.no** [39] is used to map each NNE node to it's geographic location, MBB operator and measurements identification. Thus

before extracting and logging the result into database, this metrics are added to each measurements and then saved in database.

Table 3.2 shows a sample of the Traceroute measurements recorded in database :

M_Id	Node	Region	ISP	website	ASN_occurr	IngDely	ProvDelay
6528	582	Oslo	Telenor	google.com	"2119": 4, "224": 1	39.207	15.475
6528	582	Oslo	Telenor	adcash.com	"2119": 5, "15169": 11	19.772	158.609
6528	582	Oslo	Telenor	youtube.com	"2119": 4, "224": 1	28.182	24.183
6528	582	Oslo	Telenor	facebook.com	"2119": 5, "32934": 7	39.746	150.044
6528	582	Oslo	Telenor	google.com	"2119": 4, "224": 1	32.333	37.76
6528	582	Oslo	Telenor	fc2.com	"2119": 5, "3257": 2, "16509": 2	23.1	627.229
6528	582	Oslo	Telenor	msn.com	"2119": 5, "8075": 16	34.005	234.956
6528	582	Oslo	Telenor	bing.com	"2119": 5, "8075": 1	28.22	41.565
6528	582	Oslo	Telenor	blogspot.com	"2119": 5, "15169": 2	36.687	25.384
6528	582	Oslo	Telenor	imgur.com	"2119": 5, "3356": 1	37.865	37.346

Table 3.2: Sample data collected in the traceroute measurements and recorded in database.

The following measurement parameter are collected by the **traceroute.py** scripts on NNE nodes. Each raw in database table contains the following metrics for traceroute measurements:

- **M_Id**: The measurements id which is unique number to identify the measurement's node and MBB operator.
- **Node**: The NNE nodes identification number.
- **Region**: The geographic location of NNE nodes.
- **ISP**: The MBB operator(s) on each Node (NNE are connected to multiple MBB operators)
- **Website:** The web content provider from top 50 websites used in the measurements.

- **ASN_occurr:** The Autonomous System Number of the IP addresses in traceroute hops, and the occurrences of each IP-address.
- **IngDelay (Ingress delay):** The RTT time measured in milliseconds, when the traceroute packet enters the ingress point of the MBB operator.
- **ProvDelay (content provider delay):** The RTT time measured in milliseconds, when the traceroute packet enters the content provider's network .
- **All Hops:** The IP address of all hops/routers through the Internet path between end-to-end points (the NNE nodes and the service provider's network).
- **Hop's RTT:** The delay time of each hop in the Internet path between endpoints.

3.5.3 Plots.py and Makeplots.py

The script **plots.py** implements the **Matplotlib** library and contains the generic functions for creating bar charts, boxplots, and plots which accept several parameters such as dataset, title, and labels for X and Y axises.

The script **makeplots.py** runs the functions written in **plots.py** to create plots and graphs in order to visualizes the desired data for analysis. The following graphs is chosen to be plotted from the extracted data in measurements:

- The tables for statistics analysis of values such as min, max, mean, median, standard deviation, 75 percentile, 95 percentile of results.
- The boxplots of RTT delays for each operators in different graphs.
- The Cumulative Density Function (CDF) of RTT delays fraction for operators in one graph.
- The boxplots of the median values of the time to connect for nodes using MBB connection for each operators in different graphs
- The boxplots of the median values of the time to start transfer the first byte in web access to the content providers, for each operators in different graphs.
- The Cumulative Density Function (CDF) of the time to connect ratio to the time to start transfer for operators in one graph.
- The Table of the median values of the radio signal conditions, such as RSSI, RSCP, Ec/Io, RSRP, and RSRQ.

3.5.4 Database.py and utils.py

The script **database.py** is used for creating database, tables, inserting data into the tables, and reading records from the tables. The **utils.py** contains the python functions/methods that can be reused in **traceroute.py** and **curl.py** scripts.

Chapter 4

Results and Analysis

The chapter presents the results of the experiments and the analysis that has been performed on the collected data.

As mentioned in chapter 3, the measurements carried out using NNE Edge of Simula Laboratory were divided into two parts, thus there are two set of measurement results as following:

- The Traceroute measurements consisted of and 101254 measurements which were recorded in database. From these Traceroute measurements in database, there were 50386 measurements for Telenor, and 39732 measurements for Netcom. Each traceroute measurement was performed by running Traceroute command from a NNE node towards one of the top 50 Alexa websites using a MBB connection (3G and 4G) from Telenor or Netcom.

- The Curl measurements consisted of 1736364 measurements recorded in database, 757832 measurements for Telenor and 396948 measurements for Netcom. Each record in database consisted of both the metadata information of the NNE nodes and the HTTP performance metrics under accessing the web content from the nodes using Curl command.

4.1 Ingress point placement

To determine the approximate location of the carrier's ingress point, the IP address of the first hop in the routing path was resolved using the Whois command and mapped to the AS number and the owner of it. If the first hop's IP address was a private address e.g (10.0.0.0/24 or 192.168.0.0/24), so the IP-address of the next hop was chosen as the public IP address of the MBB operator and etc.

4.2 Network Delay

The Round-Trip-Time (RTT) value was measured in order to calculate the network delay in two cases: from the NNE node to the ingress point (node-to-ingress delay) and from the NNE node to the content provider (end-to-end delay).

For analyzing the network delay, it was chosen to group the NNE nodes by their geographic regions in Bergen, Oslo, and Trondheim since the impact of routing path between nodes and the content provider was of interest in order to measure the delay caused by the routing path.

4.2.1 Node-to-ingress delay

In Telenor, it was observed that the median node-to-ingress delays for the nodes located in Oslo was around 20 milliseconds, whereas for nodes located in Trondheim and Bergen was around 30 milliseconds which is increased 10 milliseconds compared to nodes in Oslo. The 75 percentile of the node-to-ingress delays in all regions were distributed by 10 milliseconds ingress compared to the median values. The 95 percentile of delays was equally and under 40 milliseconds distributed for nodes located in Oslo and Bergen, whereas it was 10 milliseconds more for nodes located in Trondheim as is shown in Table 4.1.

Region	**Min**	**Max**	**Mean**	**Median**	**Std**	**75 percentile**	**95 percentile**
Bergen	20	100	30	30	10	40	40
Oslo	10	160	30	20	10	30	40
Trondheim	10	21	30	30	10	40	50

Table 4.1: Statistics calculation of node-to-ingress RTT delays in milliseconds (Telenor)

A slightly different results of min, max, mean, 75 and 95 percentile of node-to-ingress delays was observed using Netcom connection. The median of node-to-ingress delays for nodes in Oslo was around 20 milliseconds, whereas it was 30 milliseconds for nodes in Bergen and Trondheim.The 75 percentile of node-to-ingress delays for all regions was observed equally and around 30 milliseconds. The 95 percentile of node-to-ingress delays were distributed under 30 milliseconds for nodes in Oslo, whereas it was 40 milliseconds for nodes in other regions as is shown in Table 4.2. These results showed that the ingress point of both operators is located in Oslo.

Region	Min	Max	Mean	Median	Std	75 percentile	95 percentile
Bergen	20	48	30	30	10	30	40
Oslo	10	50	20	20	10	30	30
Trondheim	20	18	30	30	10	30	40

Table 4.2: Statistics calculation of node-to-ingress RTT delays in milliseconds (Netcom)

4.2.2 End-to-end delay

Table 4.3 and 4.4 show the statistics of end-to-end delays for Telenor and Netcom respectively. In the results of end-to-end delays for Telenor, it was observed that the median end-to-end delays for nodes in Oslo was around 150 milliseconds, whereas it was around 160 milliseconds for nodes in the other regions. The 75 percent of end-to-end delays for nodes in Oslo was distributed under 230 milliseconds, whereas it was under 240 milliseconds for nodes in the other regions. The 95 percent of end-to-end delays was distributed under 400, 410, and 390 milliseconds for nodes in Oslo, Bergen , and Trondheim, respectively .

Region	Min	Max	Mean	Median	Std	75 percentile	95 percentile
Bergen	20	3940	190	160	130	240	410
Oslo	20	2810	180	150	120	230	400
Trondheim	20	1940	190	160	120	240	390

Table 4.3: Statistics calculation of end-to-end RTT delay (Telenor)

The median end-to-end delays for nodes in Oslo was around 140 milliseconds and for nodes in the other regions was around 150 milliseconds. The 75 percent of end-to-end delays was distributed under 240 milliseconds for nodes in Oslo and Trondheim, whereas it was 270 for nodes in Bergen. The 95 percent of end-to-end delays for nodes in all regions was distributed with almost doubled values for nodes in all regions compared to 75 percentile value.

Region	Min	Max	Mean	Median	Std	75 percentile	95 percentile
Bergen	30	1300	210	150	160	270	520
Oslo	10	4170	190	140	150	240	470
Trondheim	30	3550	200	150	150	240	460

Table 4.4: Statistics calculation of node-to-end RTT delay (Netcom)

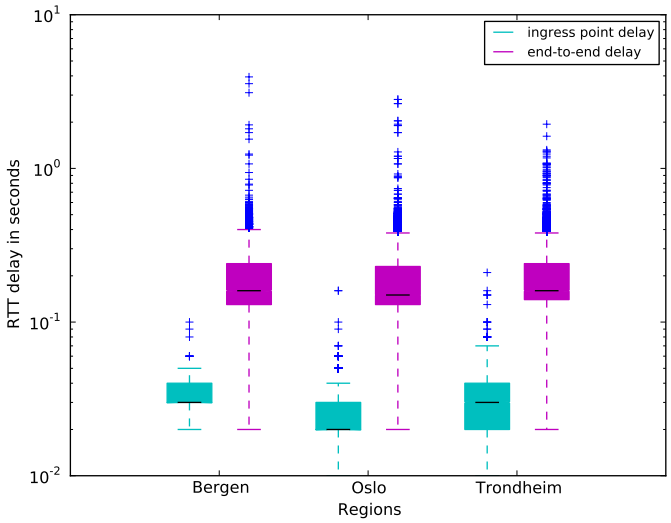

Figure 4.1: Boxplot of the node-to-ingress delay and end-to-end delay in second (Telenor)

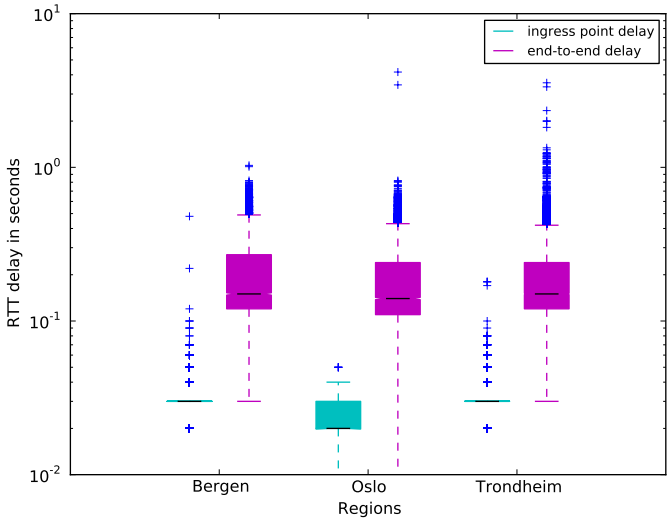

Figure 4.2: Boxplot of the node-to-ingress delay and end-to-end delay in second (Netcom)

The distribution of node-to-ingress delays and end-to-end delays for nodes in different regions of Norway for both operators is depicted in the boxplot [4.1,4.2]. The RTT delays distribution in case of from nodes to the ingress point is reasonably much lower than the RTT distribution in case of from nodes to the content providers since the ingress point is located in Oslo in both operators.

If RTT delays observed with their spread, Netcom had less spread of end-to-ingress delays in Bergen and Trondheim, while both operators have the same spread in Oslo. The skewness of boxplots here indicates that data distribution are not uniform between different quartile range. Looking at the 95 percentile values, Netcom performed better than Telenor in Trondheim, whereas Telenor performed better in Oslo. Comparing the spread and the 95 percentile values of the end-to-end delays in both operators, Telenor performed slightly better than Netcom in all regions.

4.2.3 Node-to-ingress delay ratio to end-to-end delay

The plots in Figures 4.3 and 4.4 show the Cumulative Distribution Function (CDF) of ingress to end-to-end delay ratio in Telenor and Netcom, respectively. Comparing RTT delay fraction for Telenor and Netcom, in over 60 percent of the traceroute fraction, only 20 percent of RTT delay was observed between the node and ingress point, whereas 80 percent of delay was from end-to-end delay in both operators.

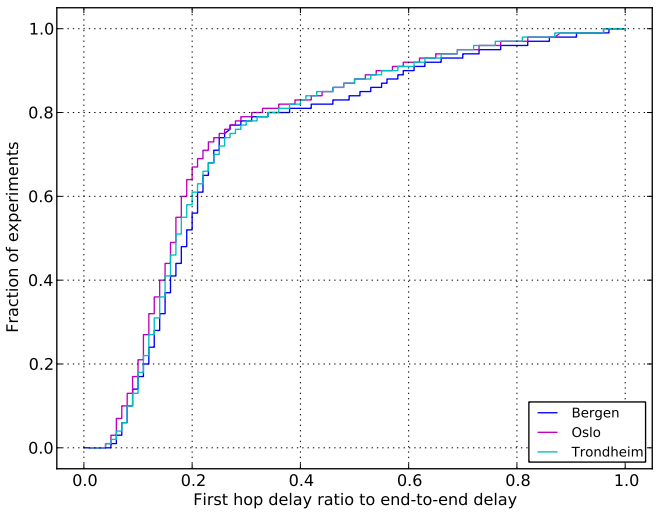

Figure 4.3: The CDF of the node-to-ingress delay ratio to end-to-end delay (Telenor)

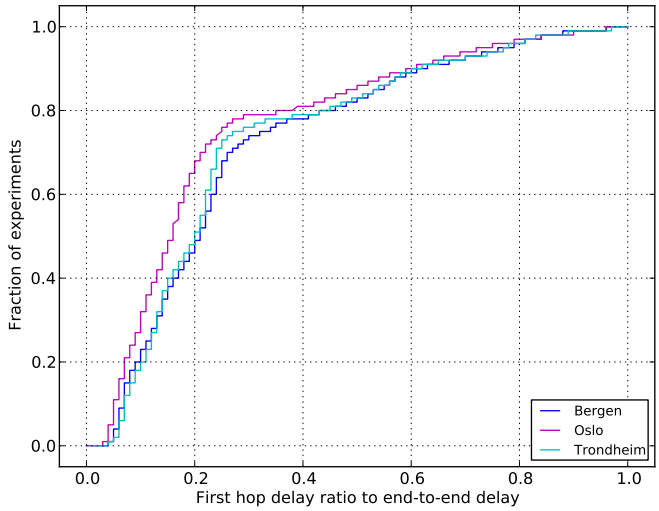

Figure 4.4: The CDF of the node-to-ingress delay ratio to end-to-end delay (Netcom)

4.3 Routing path

The MBB operators may send the Internet traffic for accessing the web content to/from their subscribers either by peering directly with the service and content providers or by passing the Internet traffic through a transit ASs, or even they have local copies or local CDNs inside their networks.

In the end-to-end routing path of each operator from the Traceroute measurements, the ASN occurrences and the number of hops needed to reach the content provider were studied and a separate table were created for routing path using Telenor and Netcom connection.

Figure 4.5 shows three different cases of routing path between the NNE nodes and the content providers:

1. There was only the operator's ASN in the routing path.

2. The content provider were directly linked to the MBB operator's network.

3. There were more than two different ASNs in the routing path.

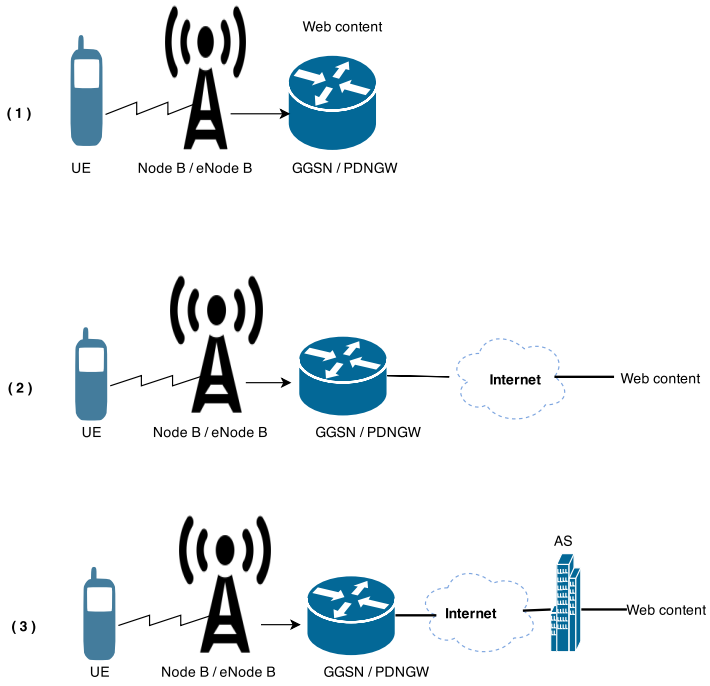

Figure 4.5: The routing path between MBB operator and the content provider

In the routing path results for Telenor as operator, it was observed that the Internet path to from NNE nodes to the content providers such as Bing.com, Linkedin.com, Wikipedia.org, Youtube ended to the Telenor's network. For other content providers hosted in the USA such as Google.com, microsoft.com, Amazon.co, Ebay.com, Instagram.com, Stackoverflow.com, and Netflix.com. it was observed that Telenor passed the Internet traffic either directly to the content providers network or to the ASN-3356 which owned by **Level 3 Communications, Inc.,USA** which in 44 of the 50 content providers. In addition, between 3 and 5 different ASN in the routing path from the NNE nodes to the other top 50 websites mostly hosted in China such as 163.com, Baidu.com, Gmw.com, Sohu.com, Tmall.com depending on their geographic location

The routing path results are approximately similar for Netcom AS Norway which is part of the **TELIANET TeliaSonera International Carrier**. The Internet path to content providers such as Bing.com, Google.com, Paybal.com, Ebay.com, Youtube.com, Paybal.com and Linkedin.com ended to the Netcom's network, and for content providers such as Apple.com,

Amazon.com, Yahoo.com, and Twitter.com. The Internet path from Netcom ended directly to these network. For other content providers depending on their geographic location across the world, between 3 and 5 different ASNs are crossed in order to reach the content provider's network. The **Level 3 Communications, Inc.,USA** which has ASN-3356 was observed in 14 cases of the Internet path using Netcom as operator.

4.4 HTTP performance delay

As mentioned in chapter 3, the Curl measurements were carried out on NNE nodes in order to collect the metadata information and the HTTP performance of both MBB operators when accessing web content by sending HTTP request to the index file of top 50 websites.

Since this study is measuring and analyzing the HTTP performance for accessing a small size of content over the Internet, so the analysis in this section are based on only two important HTTP metrics of Curl results, the time-to-connect to the content provider (time-to-connect) and the time to start transfer the first byte of web content (time-to-start-transfer).

With time-to-connect metric, the time spent to establish the TCP connection from nodes to the content provider's server were measured. The time-to-start-transfer the first byte as the second metric of the HTTP Get was important in this study due to measure the time were spent from nodes in order to access the web content over the Internet. By comparing the results of the time-to-start-transfer with the results of the time-to-connect, the time used for reaching the ingress point and the content providers server were measured.

By analyzing the Curl measurements for both operators, it was observed that the number of nodes in Telenor using 3G connection was only two nodes compared to Netcom which had more nodes using 3G connection (7 nodes). In both operators, more number of nodes using 4G connection was observed, 10 and 11 nodes using 4G connection for Telenor and Netcom, respectively.

Table 4.6 describes the Norway regions, the number of NNE nodes in each region, and the MBB operators available on each node.

Region	NNE Node	Telenor	Netcom
Oslo	582	✔	✔
	452	✔	✔
	545	✔	✔
	712	✔	✔
Trondheim	599		✔
	645	✔	✔
	721	✔	✔
	485		
Bergen	750	✔	✔
	465	✔	✔
	755	✔	✔
Hammerfest	715	✔	✔
Bodø	424	✔	
Kristiansand	321	✔	✔
	706		✔
	707		✔

Figure 4.6: Overview of the NNE nodes and regions used for measurements.

4.5 Time to connect to content provider's server in Netcom (3G and 4G)

Figures 4.7 and 4.8 show the results of the median time-to-connect for nodes using Netcom 3G and 4G connection, respectively. The median time-to-connect for all nodes in both 3G and 4G, as it can be seen in the boxplots was between 100 and 200 milliseconds, except node 582 which had median time-to-connect of 300 milliseconds using Netcom 3G connection.

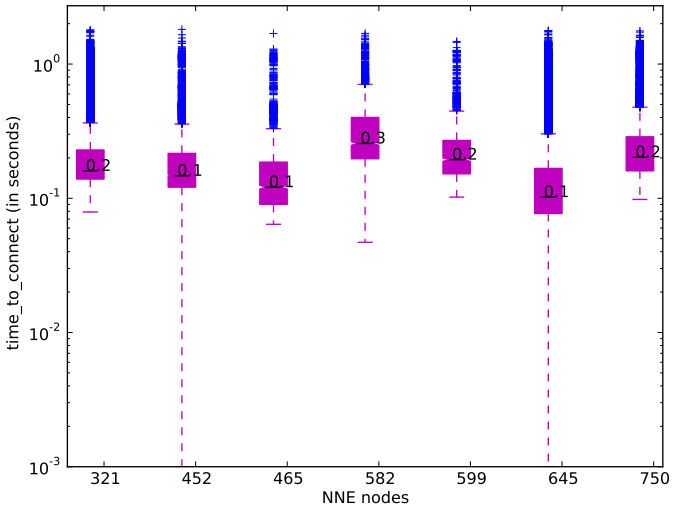

Figure 4.7: Boxplot of the time to connect in second (Netcom 3G)

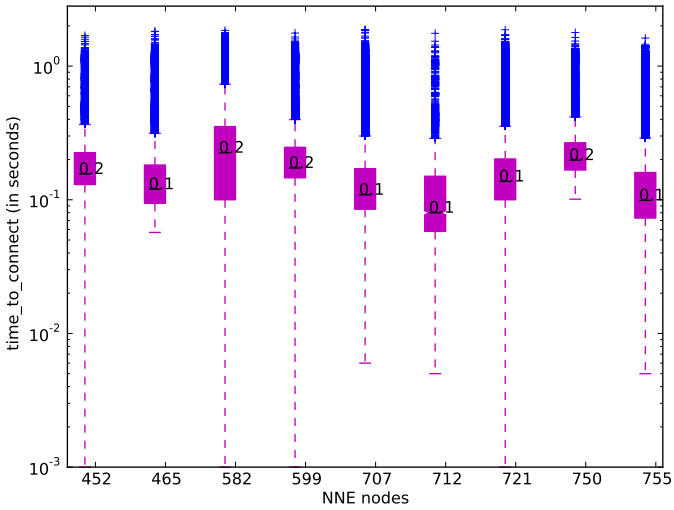

Figure 4.8: Boxplot of the time to connect in second (Netcom 4G)

4.5.1 Time to start transfer the first byte of Internet content in Netcom (3G and 4G)

The results of the median time-to-start-transfer of the first byte when accessing the web content. The boxplots in Figures 4.9 and 4.10 show very similar results of the median time-to-start-transfer between 500-700 milliseconds in both 3G and 4G connections from Netcom. So there is no clear differences between 3G and 4G connections for time-to-connect.

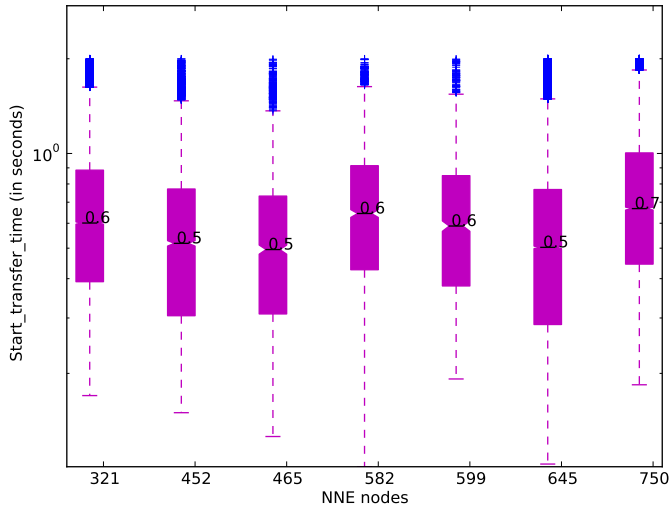

Figure 4.9: Boxplot of the time to start transfer in second (Netc 3G)

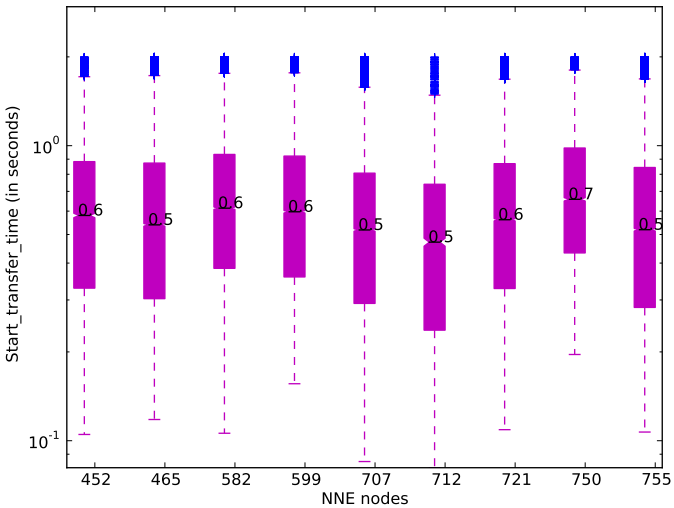

Figure 4.10: Boxplot of the time to start transfer in second (Netcom 4G)

4.6 Time to connect to the content provider in Telenor and Netcom

The boxplots in Figures 4.11 and 4.12 show the distribution of the median time-to-connect in scale of seconds to the content provider's server from the NNE nodes for both Telenor and Netcom. All nodes using Telenor connection had a similar median value of the time-to-connect between 200 milliseconds except from three nodes (750, 582 and 424) had the median time-to-connect of 300 seconds which was unexpected.

The results of the median time-to-connect for the nodes using Netcom showed a slightly lower median time-to-connect in Telenor. The median values of 100 milliseconds was observed on nodes (321, 452, 582, 599, and 750) while the median values of 200 milliseconds were observed on the rest of nodes using the Netcom connections. The inter-quartile range in both operator is very similar.

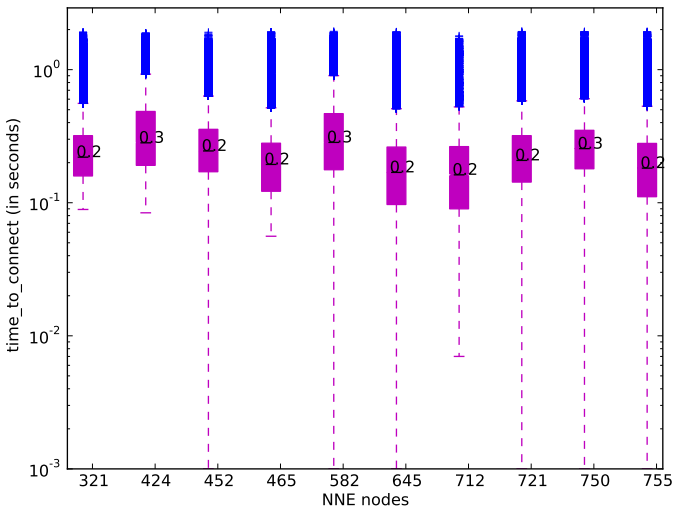

Figure 4.11: Boxplot of the time to connect in second (Telenor)

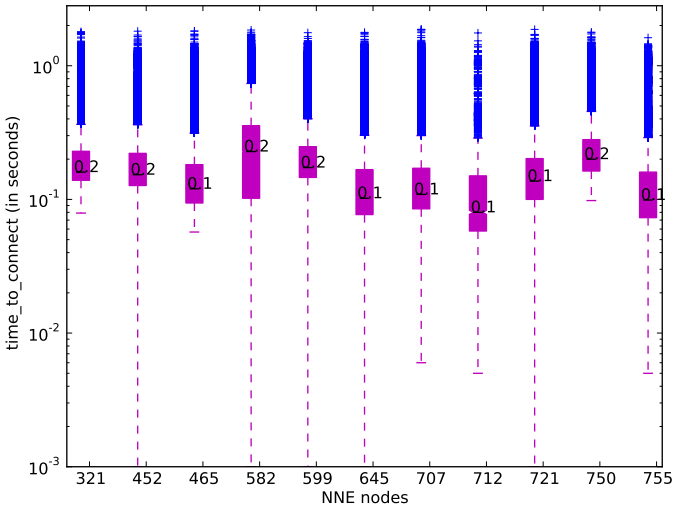

Figure 4.12: Boxplot of the time to connect in second (Netcom)

Figure 4.13 shows the CDF of the median time-to-connect in the scale of seconds for nodes using Telenor as MBB connection. The figure shows that up to 50 percent of cases for all nodes located in different places in Norway, the median time-to-connect is under 250 milliseconds.

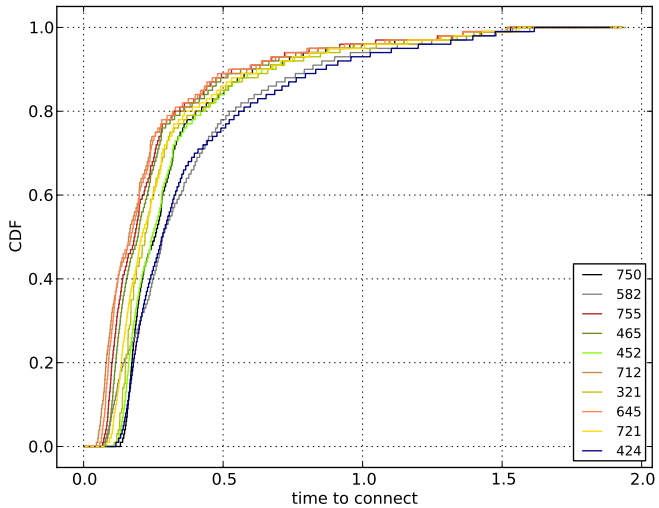

Figure 4.13: CDF of the time to connect in second (Telenor)

It can be also seen that up to 80 percent of cases in the mentioned node the median time-to-connect is under 500 milliseconds.

The results of the median time-to-connect for nodes using Netcom connection is shown in Figure 4.14. The results showed that the median

time-to-connect in Netcom is slightly shorter than what experienced in Telenor. It can be seen that Netcom has very similar the median time-to-connect up to 40 percent of cases.

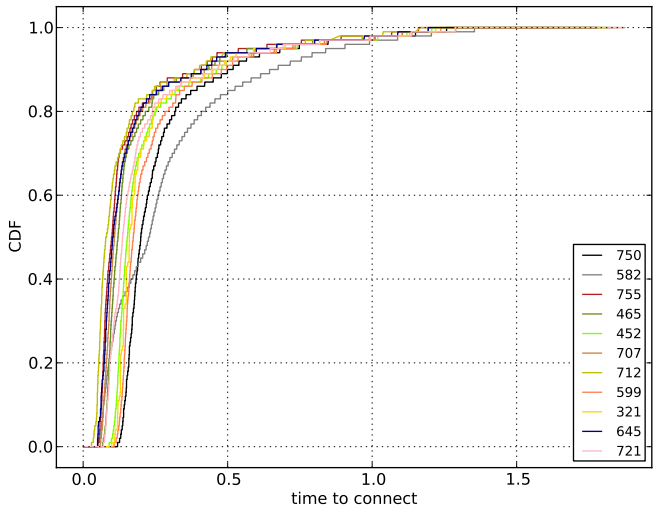

Figure 4.14: CDF of the time to connect in second (Netcom)

4.6.1 Time to start transfer the first byte of web content

The boxplots in Figures 4.16 and 4.15 shows that how the median time-to-start-transfer were distributed in both operators.

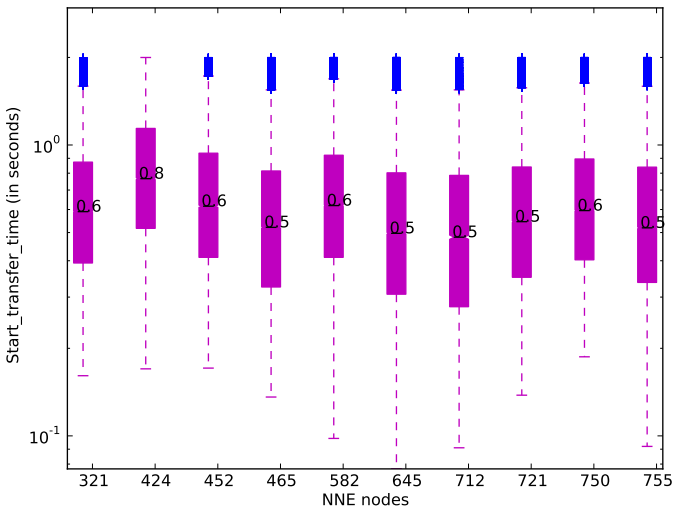

Figure 4.15: Boxplot of the time to start transfer in second (Telenor)

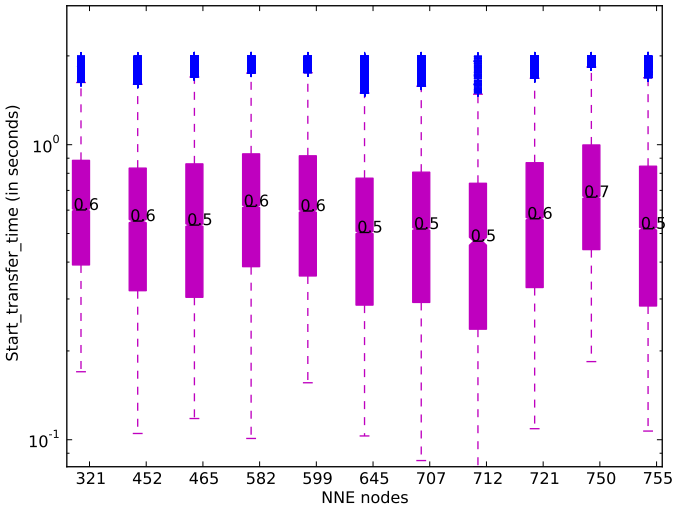

Figure 4.16: Boxplot of the time to start transfer in second (Netcom)

As it can be seen in the figures, both operators had almost very similar distribution of the time-to-start-transfer around 500 and 600 milliseconds for nodes using both operators. Only two nodes had slightly different median values: In Netcom, node 750 had the median time-to-start-transfer of 700 milliseconds, and in Telenor node 424 had the median time-to-start-transfer of 800 milliseconds.

Figure 4.17 and 4.18 show the CDF of the median time-to-start-transfer for nodes in both operators. The results showed that the median values were scattered equally in all cases for both operators, except from two nodes had the different median of the time-to-start-transfer, in Telenor node 424 and in Netcom node 750, which was not expected.

It can be seen in the graph that in up to 80 percent of the cases in both operators, the median time-to-start-transfer was under 1000 milliseconds.

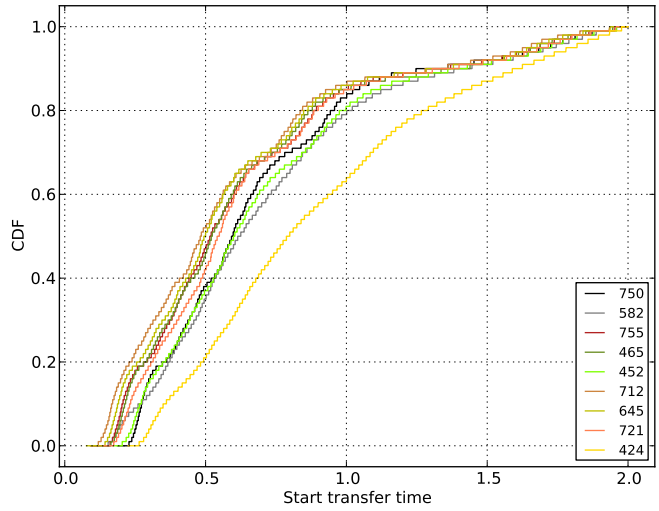

Figure 4.17: CDF of the time to start transfer in second (Telenor)

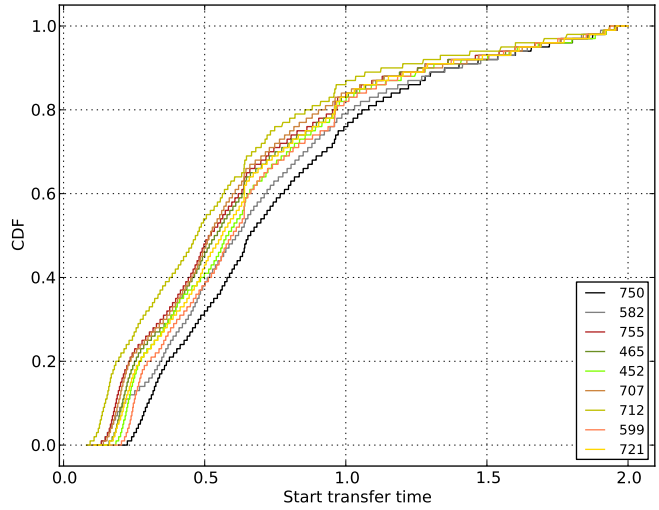

Figure 4.18: CDF of the time to start transfer in second (Netcom)

4.6.1.1 Time to start transfer the first byte of web content of top 3 websites (Google.com,Apple.com, and Bing.com)

In this section the results of the median time-to-start-transfer are analyzed when the web access to only top 3 content providers: Google.com, Apple.com, and Bing.com. The reason for analyzing the time-to-start-transfer of them was to discover how much of time is used in order to reach the web content of those content providers with their web content closest to the ingress point.

The CDF plots of the median time-to-start-transfer for top 3 content providers for nodes in both operators are shown in Figures 4.19 and 4.20. The results showed that in 20 percent of cases, the nodes in both operators had spent less than 200 milliseconds in order to reach the web content of these top three content providers. It can also be seen in the plot that in 95 percent of the cases the median time to reach the web content of them were less than 300 milliseconds.

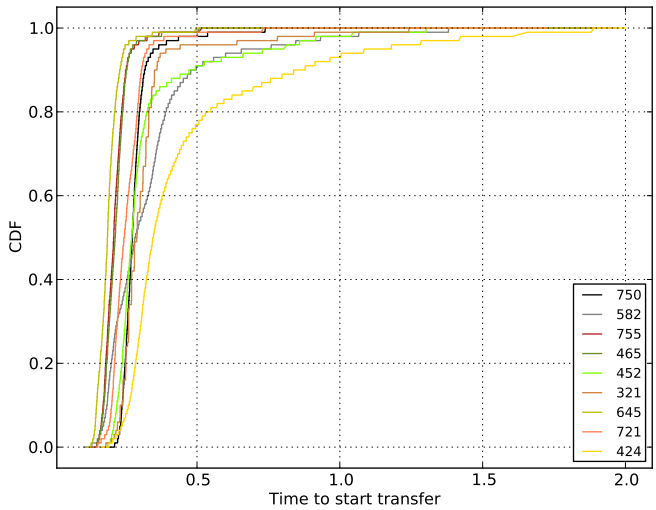

Figure 4.19: CDF of the time to start transfer in second for top 3 websites (Telenor)

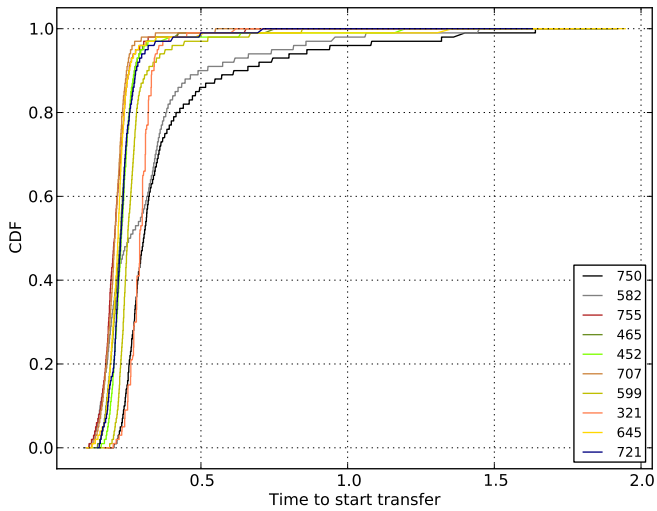

Figure 4.20: CDF of the time to start transfer in second for top 3 websites (Netcom)

4.6.2 Radio conditions

In order to understand what may cause the bad connection condition under the HTTP Get results, the median values of the RSSI, RSCP,Ec/Io, RSRP, and RSRQ were analyzed for the nodes using the MBB connection from both operators.

As mentioned in chapter 2, the RSSI measures the total received signal power which includes both the good signal power and the interfaces/noise power, so a high RSSI does not indicate a good radio condition. The ma_rssi metrics from the metadata information collected in the Curl measurements in in ASU (Arbitrary Signal Unit), thus it needed to b converted from ASU to RSSI in scale of millidecibel by using this formula:

1 dBm = (2 * ASU) - 113

Then the median RSSI values for all nodes were categorized into five groups:

- 1 bars: -103 dBm and lower
- 2 bars: -98 dBm to -102 dBm
- 3 bars: -87 dBm to -97 dBm
- 4 bars: -78 dBm to -86 dBm
- 5 bars: -77 dBm to higher

In 3G connection the Ec/Io measures the signal quality (the signal ratio to noise) which is for the connection stability, the median Ec/Io values were grouped in three categories:

- Good (0 dB> Ec/Io > -8 dB)
- Medium (-8 dB> Ec/Io >-15 dB)
- Bad (-15dB> Ec/Io <-33 dB)

RSCP values in 3G connection for both operators were categorized in three groups:

- Sufficient (RSCP > -99)
- Poor (-115 < RSCP <-100)
- No coverage (RSCP < -115)

In 4G connection the RSRP and RSRQ represents the signal power and the signal noise, respectively. They were categorized in four groups:

Condition	RSRP	RSRQ
Excellent	RSRP >= -80	RSRQ >= -10
Good	-90 < RSRP < -80	-15 < RSRQ < -10
Midle	-100 < RSRP < -90	-20 < RSRQ < -15
Poor	RSRP =< -100	RSRQ =< -20

Table 4.5: RSRP and RSRQ conditions in 4G

The results of the median radio signal conditions for nodes using Telenor 3G connection are shown in Table 4.6. As mentioned before in this section, there were only two nodes (321 and 424) using Telenor 3G connection. Node 321 had the median RSSI of -73.0 dBm and node 750 had the median RSSI of -91.0 dBm. Node 424 had also the median RSCP of -96.0 dBm and the median Ec/Io of -5.0 dBm. Node 424 had lowest median RSCP and Ec/Io compared to node 321, it had -96.0 dBm and -5.0 dBm for RSCP and Ec/Io, respectively.

The only 3G connection was available on node 321 which had the median RSCP, and Ec/Io of -78 dBm and -2.0, respectively. The other nodes were using the 4G connection mode of Telenor connection.

Table 4.7 shows the median values of the RSSI, the signal power (RSRP) and the signal noise/quality (RSRQ) for nodes using Telenor 4G connections. The median RSSI values between -75.0 dBm and -85.0 dBm were observed for all nodes. The result shows that the median RSSI for nodes was between -75.0 and 89.0 dBm, the highest that nodes had the median RSRP in range -98.0 and -114.0 dBm, the highest median RSRP was observed for node 721 which was -98.0 dBm and the lowest median RSRP was observed for node 452 which was -114.0 dBm. The median RSRQ for nodes using 4G connection varied between -4.0 dBm and -20.0 dBm, the lowest median RSRQ was observed for node 424 which was -20.0 dBm, and the highest median RSRQ was observed for node 465 which was -4.0 dBm. The result showed that node 424, 452 and 712 using Telenor 4G connection had suffered of poor signal quality.

Node	RSSI	RSCP	Ec/Io	RSRP	RSRQ
321	-79.0	-78.0	-2.0	-	-
424	-91.0	-96.0	-5.0	-	-

Table 4.6: Median of the signal conditions in dBm (Telenor 3G)

Node	RSSI	RSCP	Ec/Io	RSRP	RSRQ
424	-75.0	-	-	-112.0	-20.0
452	-89.0	-	-	-114.0	-10.0
465	-83.0	-	-	-106.0	-4.0
582	-81.0	-	-	-106.0	-5.0
645	-83.0	-	-	-109.0	-7.0
712	-83.0	-	-	-113.0	-11.0
721	-73.0	-	-	-98.0	-6.0
750	-85.0	-	-	-106.0	-6.0
755	-75.0	-	-	-103.0	-7.0

Table 4.7: Median of the signal conditions in dBm (Telenor 4G)

Table 4.8 shows the results of the median radio signal conditions for nodes using Netcom connection 3G. The results shows that the median RSSI for nodes was between -73.0 dBm and -89.0 dBm. Node 750 and 452 had the lowest median RSSI, in both -93.0 dBm. The results, shows that the median RSCP was between -73.0 dBm and -103.0 dBm, the lowest median RSCP were -103.0 and -102.0 dBm which were observed for nodes 452 and 750, respectively. The median Ec/Io was between -1.0 and -9.0 dBm for nodes. the highest Ec/Io was -1.0 dBm which was observed for node 321. The lowest Ec/Io was -9.0 dBm for node 452. The results showed that node 452 and 752 had suffered of poor signal power.

Node	RSSI	RSCP	Ec/Io	RSRP	RSRQ
321	-81.0	-80.0	-1.0	-	-
452	-93.0	-103.0	-9.0	-	-
465	-79.0	-84.0	-4.0	-	-
582	-87.0	-94.0	-7.0	-	-
599	-71.0	-73.0	-4.0	-	-
645	-85.0	-93.0	-8.0	-	-
750	-93.0	-102.0	-8.0	-	-

Table 4.8: Median of the signal conditions in dBm (Netcom 3G)

Table 4.9 shows the median signal condition for nodes using Netcom 4G connection. As it can be seen in the table, the median RSSI for nodes were observed between -71.0 dBm and -87.0 dBm, The lowest median RSSI was observed for node 750 with the median RSSI of -87.0 dBm, and the highest median RSSI was -61.0 dBm which was observed for node 707. The median RSRP for nodes was observed between -89.0 dBm and -118.0 dBm, the lowest median RSRP was -118.0 dBm was observed for node 750. The median RSRQ for nodes was between -6.0 dBm and -11.0 dBm, the lowest median RSRQ was -11 which were observed for nodes 750 and 452. The results showed that nodes have good signal quality but nodes 750 had poor signal power.

Node	RSSI	RSCP	Ec/Io	RSRP	RSRQ
452	-71.0	-	-	-97.0	-11.0
465	-79.0	-	-	-105.0	-6.0
582	-77.0	-	-	-105.0	-8.0
599	-67.0	-	-	-94.0	-6.0
707	-61.0	-	-	-89.0	-6.0
712	-67.0	-	-	-93.0	-6.0
721	-79.0	-	-	-107.0	-7.0
750	-87.0	-	-	-118.0	-11.0
755	-79.0	-	-	-106.0	-7.0

Table 4.9: Median of the signal conditions in dBm (Netcom 4G)

So the poor radio conditions on nodes 424 in Telenor and Node 750 in Netcom had a significant impact on their HTTP Get performance. As the median time to connect and the median time to start transfer the first byte were increased by approximately 100 milliseconds for these nodes.

Chapter 5

Discussion and Future work

This chapter discusses and evaluates our results.

5.1 Evaluation of the network topology

The analysis of the Traceroute results, showed that the web access from nodes towards the top 50 content providers can take three different routing paths based on where the web content is located. In addition, it was observed that the web content of some popular content providers such as Bing.com, Google.com, and Apple.com were located inside the MBB operator's network or was closest to the ingress point of MBB operators.

5.2 Evaluation of the Network delay

The analysis of the ingress point placement, showed that having only one ingress point (GGSN / PDNGW) in a small country like Norway that is not very wide compared to the countries such as USA or Russia, would not significantly affect the latency from different regions across Norway. In the results, it was observed that the delay from nodes to ingresspoint for nodes located in Bergen and Trondheim was similar and around 20 milliseconds compared to Oslo that was 20 milliseconds, which mean that only 10 milliseconds different between regions in Norway.

The network delay between the operator's ingresspoint and the content providers varied based on where the web content was located. The results of the node to ingresspoint delay ratio to the end-to-end delay showed that that only 20 percent of network latency for accessing the web content is caused by the distance between nodes and the ingress point, the remaining 80 percent is because of the distance between the ingress point and the content provider.

5.3 Evaluation of HTTP performance under accessing web content

The results from time to connect and time to start transfer the first byte of the web content in the HTTP Get request to the content providers, showed that there were no significant differences between operators in the time to connect and time to start transfer the first byte. The median time spent to query the DNS and establish TCP connection with the content provider's server (time to connect) was between 100 and 200 milliseconds, which is about 20 percent of the whole time spent to get the first byte which was between 500 and 600 milliseconds.

5.4 Evaluation of radio conditions

The results of the radio signal conditions for nodes, showed that the poor radio signals related to the signal power and signal quality affected the HTTP Get performance on nodes. The results showed that nodes that had poor radio signals, had also the higher median values of the time to connect and time to start transfer the first byte. The poor radio signal conditions can increase the time to get the first byte of the web content by 100 milliseconds

5.5 Limitations

There were also some limitations in this study. Firstly, the NNE nodes are stationary nodes, which do not provides mobility to carry out the measurements. So the performance of content delivery were measured only in a static environment. In mobility of user equipments it is expected that the radio signals strength changes frequently which results in more often connectivity loss, connection failure and even delay in transmission of data. In addition to the mobility of users, the coverage of MBB network in other public places like in the subways, under grounds, tunnels etc is not compared in this study.

5.6 Future Work

Due to the time constraint, this work could not cover some other research possibilities on the performance of the content delivery over the Internet, so some aspects of the work remains as an inspiration for future work related to the performance of MBB networks.

The HTTP Get performance was measured using TCP/IP protocol to only get the first byte of the web content as a very small size of data. As a future work, the HTTP Get performance can be measured in other scenarios for accessing content over the Internet such as the media streaming

and file transfer.

The combination of multiple MBB operators may provide a more reliable communication, and better performance for subscribers. So the performance of content delivery over Internet using multiple path and protocols such as MPTCP is highly recommended as a future work.

In the future, we also plan to investigate the performance of the current proposals for optimizing web access such as **Spdy**, **HTTP2** [35] and the different choices for DNS servers placement.

Chapter 6

Conclusion

According to the evaluation results and the analysis presented in chapter 4 and 5, The following conclusions were obtained:

It is shown that the networking topology has significant impact on the end-to-end web performance for the mobile subscribers, so that the network delay can be decreased by 80 percent if the web content is hosted inside the MBB network or near the ingresspoint. In addition, the MBB networks have ingresspoint located in Oslo. For a small country like Norway, the ingress point placement does affect the network delay only by 10 milliseconds increased for the regions that are far away from the ingresspoint.

Regarding the web performance by accessing a small size of the web content, there was no clear differences in time to get the first byte between MBB networks using 3G or 4G technologies.

In addition, it is shown that the time to connect to the web content (to query DNS and establish a TCP connection) is only 20 percent of the time spent to receive the first byte of the web content.

It is also shown that the poor radio signal conditions including the signal power and the signal quality would increase the time to get the first byte of the web content by 100 milliseconds.

Appendices

Appendix 1: Top 50 Alexa websites

Top 50 websites ranked by Alexa.com

```
google.com
facebook.com
youtube.com
baidu.com
yahoo.com
wikipedia.org
amazon.com
twitter.com
taobao.com
yandex.ru
qq.com
linkedin.com
live.com
sina.com.cn
weibo.com
tmall.com
ebay.com
blogspot.com
hao123.com
bing.com
reddit.com
sohu.com
tumblr.com
imgur.com
wordpress.com
instagram.com
pinterest.com
msn.com
apple.com
paypal.com
microsoft.com
aliexpress.com
xvideos.com
imdb.com
fc2.com
alibaba.com
stackoverflow.com
vk.com
ask.com
360.com
netflix.com
163.com
adcash.com
go.com
```

```
craigslist.org
never.com
diply.com
gmw.cn
xhamster.com
rakuten.co.jp
```

Appendix 2: Developed scripts

utils.py

```
#! /usr/bin/env python
import subprocess
import sys
from  time import gmtime,strftime
import datetime
import socket
import io
import commands
import netifaces as ni
import signal
import shutil
import os

def filename(path,experiment,ts,instanceId):
    return "%s%s_%s.sdat.%s"%(path,experiment,str(instanceId),ts)

def filename2(path,experiment,instanceId):
    ts=strftime("%Y%m%d%H%M%S",gmtime())
    return "%s%s_%s_%s"%(path,experiment,str(instanceId),ts)

def logger(path,instanceId,str):
    logfile="%serror.logs"%path
    log=open(logfile,"a")
    ts=gettime()
    log.write("%s %s %s\n"%(ts,instanceId,str))

def gettime():
    time=strftime("%Y-%m-%d %H:%M:%S",gmtime())
    return time

def gettimeF():
    time=strftime("%Y%m%d%H%M%S",gmtime())
    return time

def curlparams():
    params= '''<http_code>%{http_code}</http_code>
                    <redirect_url>%{redirect_url}</redirect_url>
                    <url_effective>%{url_effective}</url_effective>
                    <total_time>%{time_total}</total_time>
                    <speed>%{speed_download}</speed>
                    <total_size>%{size_download}</total_size>
                    <number_connection>%{num_connects}</number_connection>
                    <size_request>%{size_request}</size_request>
                    <time_connect>%{time_connect}</time_connect>
                    <time_pretransfer>%{time_pretransfer}</time_pretransfer>
                    <time_starttransfer>%{time_starttransfer}
                    </time_starttransfer>
            '''
    return params
```

```
def copyfileTo(f1,f2):
    shutil.copy2(f1,f2)

def makeDir(dir):
    if not os.path.exists(dir):
           os.mkdir(dir,0755)

def pwd():
    currentdir=os.path.dirname(os.path.realpath(__file__))
    return currentdir

def top50sites():
    currdir=pwd()
    filename=currdir+"/top50sites.txt"
    file=open(filename,"r")
    lines=file.readlines()
    list=[]
    for l in lines:
        sp=l.strip()
        list.append(sp)
    return list

def floatf(value):
    return "%.2f"%value

def executecommand(list,filename):
    fileout=open(filename,"a")
    subprocess.Popen(list,stdout=fileout, \
                                          stderr=subprocess.STDOUT)

def myhostname():
    hostname=socket.gethostname()
    return hostname

def timestamp():
    ts= datetime.datetime.now()
    return ts

def regreplace(pattern,strtorep,line):
     #pattern=r"\d+\.\d+\.\d+\.\d+"
     match=re.search(pattern,line)
     res=""
     if match:
        m= match.group()
        res=re.sub(m,strtorep,line,1)
     return res

def writeoutput(filename,str):
    file=open(filename,"a")
    file.write("%s\n"%str)
    file.close()

def getAllifaces():
```

```
    return ni.interfaces()

def getnodeiface():
    allifaces=getAllifaces()
    return allifaces[-1]

def getnodeIP():
    allifaces=getAllifaces()
    ppp= ni.ifaddresses(allifaces[-1])
    return ppp[2][0]['addr']

def hostResolver(hostname):
    s=subprocess.check_output(['host',hostname])
    return s.split()[3]

def getMCCmns(iface):
    mapping={}
    mapping["ppp1"]=24202
    mapping["ppp0"]=24201
    mapping["ppp2"]=24007
    mapping["ppp3"]=24205
    mapping["ppp5"]=24206
    if iface in mapping:
        return str(mapping[iface])
    else:
        return 0

def getIP(host):
    hostname="www."+host
    ipaddr= socket.getaddrinfo("wwww.facebook.com",80,0,0,\
                                                    socket.IPPROTO_TCP)
    return ipaddr

def nodeInfo():
    hostname=myhostname()
    nodeiface=getnodeiface()
    nodeip=getnodeIP()
    MCCmns=getMCCmns(nodeiface)
    print "\n\nHostname: %s"%hostname
    print "NodeInterface: %s"%nodeiface
    print "Node IP: %s"%nodeip
    print "MCCmns: %s"%MCCmns
    print "pid number: %d"%os.getpid()

def curlargs():
    arg=sys.argv
    if (len(arg)<3):
       print "please type instanceId and interface:"
       sys.exit(0)
    else:
       instanceId= arg[1]
       interface=arg[2]
       MCCmns=getMCCmns(interface)
    return (instanceId,interface)
```

```
def traceargs():
    arg=sys.argv
    if (len(arg)<4):
      print "please type instanceId, network interface and timesleep:"
      sys.exit(0)
    else:
      instanceId=arg[1]
      interface=arg[2]
      timesleep=arg[3]
      MCCmns=getMCCmns(interface)
    return (instanceId,interface,timesleep)

def curlargsD():
    arg=sys.argv
    if 'start' in arg:
       if (len(arg)<4):
          print "please type instanceId and interface:"
          sys.exit(0)
       else:
          instanceId= arg[2]
          interface=arg[3]
          return (instanceId,interface)

    else:
        return 0
```

traceroute.py

```
  #!/usr/bin/env python
import subprocess
import re
import time
import sys
import socket
import geoip
import netifaces as ni
import utils
import signal
import os
import daemon
import lockfile

temp="/tmp/"
measurements="/home/diako/scripts/"
running = True

class tracerouteClass:

    def __init__(self):
         self.args=utils.traceargs()
         self.hostname=utils.myhostname()
         self.instanceId=self.args[0]
         self.interface=self.args[1]
```

```
      self.timesleep=int(self.args[2])
      self.nodeip=utils.getnodeIP()

def handler(self,signum=None,frame=None):
    sig= "Traceroute signal handler:%s"%str(signum)
    time.sleep(1)
    running=False
    os.killpg(os.getpgrp(),9)
    sys.exit(0)

def signalexec(self):
     signal.signal(signal.SIGTERM, self.handler)
     signal.signal(signal.SIGINT,self.handler)

def copyfiles(self,tracelog):
   utils.copyfileTo(tracelog,measurements)
   os.remove(tracelog)

def traceroute(self,host):
  start=utils.gettime()
  cmd=[]
  cmd.append("traceroute")
  #cmd.append("-s")
  #cmd.append(self.nodeip)
  cmd.append("-q")
  cmd.append("10")
  cmd.append("-n")
  cmd.append("-i")
  cmd.append(self.interface)
  cmd.append(host)
  p=subprocess.Popen(cmd,stdout=subprocess.PIPE,\
        stderr=subprocess.PIPE)
  stdout,stderr=p.communicate()
  result=stdout.splitlines()
  error=p.returncode
  res="<d>\n"
  res+=("<site>%s</site>\n"%host)
  starttime="<starttime>%s</starttime>\n"%str(start)
  res+=starttime
  res+="<traceroute>\n"
  for line in result:
      res+="%s\n"%line.strip()
  end=utils.gettime()
  endtime="<endtime>%s</endtime>\n"%str(end)
  res+="<errorcode>%s</errorcode>\n"%str(error)
  res+="</traceroute>\n"
  res+=endtime
  res+="</d>\n"
  return res

def doWork(self):
  sites=utils.top50sites()
  utils.makeDir(measurements)
  #counter=0
```

```
      while running:
           #ts=utils.gettimeF()
           tracelog=utils.filename2(temp,"trace",\
                                       self.instanceId)
           utils.writeoutput(tracelog,"<root>")
           for host in sites:
                if running is False:
                     return
               res=self.traceroute(host)
               utils.writeoutput(tracelog,res)
           utils.writeoutput(tracelog,"</root>")
           self.copyfiles(tracelog)
           time.sleep(self.timesleep)

    def run(self):
         self.signalexec()
         self.doWork()

if __name__=="__main__":
     obj=tracerouteClass()
     obj.run()
```

httpfetchter.py

```
#!/usr/bin/env python
import subprocess
import sys
import socket
import utils
import time
import signal
import os
from time import gmtime,strftime
import daemon
import lockfile

temp="/tmp/"
measurements="/home/diako/scripts/temp/"
running = True
pingurl="173.254.3.58"

class httpfetcherClass():

    def __init__(self):
         self.args=utils.curlargs()
         self.hostname=utils.myhostname()
         self.instanceId=self.args[0]
         self.interface=self.args[1]
         self.nodeip=utils.getnodeIP()

    def handler(self,signum=None,frame=None):
         sig= "The httpfetcher signal handler:%s\n"%str(signum)
         time.sleep(1)
         running=False
         os.killpg(os.getpgrp(),9)
```

```
     sys.exit(0)

def signalexec(self):
     signal.signal(signal.SIGTERM, self.handler)
     signal.signal(signal.SIGINT,self.handler)

def copyfiles(self,httplog):
   utils.copyfileTo(httplog,measurements)
   os.remove(httplog)

def ping(self):
           pingcmd="ping -I %s -s 1500 %s > /dev/null"% \
                                    (self.interface,pingurl)
       sub=subprocess.Popen(pingcmd,shell=True, bufsize=-1)

def metadata(self,startend,httpfile):
    MCCmns=utils.getMCCmns(self.interface)
        metadatacmd="nne-query-metadata"
    ts=utils.gettime()
    sub=subprocess.Popen(metadatacmd,shell=True, \
                  stdout=subprocess.PIPE,stderr=subprocess.STDOUT)
    lines=sub.stdout.readlines()
    httpfile.write("<%s>"%startend)
    res="<time>%s</time>"%ts
    for line in lines[1:]:
        l=line.strip().split(',')
        if MCCmns == l[2]:
           res+="<%s>%s</%s> "%(l[3],l[4],l[3])
    httpfile.write(res)
    httpfile.write("</%s>"%startend)

def check(self,url):
    try:
         p=subprocess.check_output("curl -s --interface %s -f %s"%\
                     (self.interface,url),shell=True,\
                     stderr=subprocess.STDOUT)
    except subprocess.CalledProcessError as e:
           return e.returncode


def curl(self,url,httpfile):
    params=utils.curlparams()
    cmd=[]
    cmd.append('curl')
    cmd.append('-sL')
    cmd.append('--interface')
    cmd.append(self.interface)
    cmd.append(url)
    cmd.append('-w')
    cmd.append(params)
    cmd.append('-r')
    cmd.append('0-100')
    cmd.append('-o')
    cmd.append('/dev/null')
```

```
        start=utils.gettime()
        proc=subprocess.Popen(cmd,stdout=subprocess.PIPE)
        output=proc.stdout.readlines()
        httpfile.write("<site>%s</site><start>%s</start>"%(url,start))
        for l in output:
            httpfile.write("%s"%l.strip())
        end=utils.gettime()
        #df=end-start
        errcode=self.check(url)
        if errcode is None:
           err='0'
        else:
           err=str(errcode)
        httpfile.write("<curl_code>%s</curl_code><end>%s</end>"%(err,end))
    def allData(self,url):
       timestamp=utils.gettimeF()
       utils.makeDir(measurements)
       httplog=utils.filename(temp,"http",timestamp,self.instanceId)
       httpfile=open(httplog,"a")

       self.ping()
       ts=utils.gettime()
       httpfile.write('\ %s\ \t%s\t0\t'%(ts,self.instanceId))
       httpfile.write("<d>")
       self.metadata("metadatabefore",httpfile)
       httpfile.write("<curldata>")
       self.curl(url,httpfile)
       httpfile.write("</curldata>")
       self.metadata("metadataafter",httpfile)
       httpfile.write("</d>")
       httpfile.write("\n\n")
       httpfile.close()
       self.copyfiles(httplog)

    def doWork(self):
       sites=utils.top50sites()
       counter=1
       start=utils.timestamp()
       while running:
            for s in sites:
                 if running is False:
                      return
                 self.allData(s)

    def run(self):
         self.signalexec()
         self.doWork()

if __name__== "__main__":
    obj=httpfetcherClass()
    obj.run()
```

plot.py

```
#! /usr/bin/env python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
#matplotlib.use("agg")

colors="byckm"
class plots:

      def __init__(self):

      def datatuples(self,operatorsdata):
          N =len(operatorsdata)
          xaxis=np.arange(1,N+1)
          yaxis=[float(y) for (a,b,y) in operatorsdata]
          labels = [l.replace('.com','') for (l,a,b) in data]
          return (xaxis,yaxis,labels)

      def appenditem(self,list1,list2,list3,list4,index):
          res=[]
                  res.append(list1[index])
                  res.append(list2[index])
                  res.append(list3[index])
                  res.append(list4[index])
          return res
      def formatratio(self,str):
          f="%.2f"%str
          return float(f)

      def  cdffromlist(self,list):
                  counter=1
                  cs = []
                  size=len(list)
                  sorted=np.sort(list)
                  while counter<=size:
                        prob=float(1)/size
                        if(len(cs))> 0:
                           sum=prob+cs[len(cs)-1]
                           cs.append(sum)
                        else:
                                cs.append(prob)
                        counter+=1
                  cs= [self.formatratio(x) for x in cs]
                  return cs

      def sec(self,data):
          d=[(d/1000) for d in data]
          return d


      def fractiondelayplot(self,xlabel,ylabel,title,filename,data):
              cdf1=self.cdffromlist(data[0])
              cdf3=self.cdffromlist(data[1])
```

```
        cdf4=self.cdffromlist(data[2])
    plt.plot( data[0],cdf1, 'b',label="Bergen")
    plt.plot( data[1],cdf3, 'm',label="Oslo")
    plt.plot( data[2],cdf4, 'c',label="Trondheim")

    plt.margins(0.05,0.05)
    plt.grid(True)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend(loc='lower right',prop={'size':10})

        plt.savefig("%s.pdf"%filename)
         plt.close("all")

def CDFplot(self,xlabel,ylabel,filename,data):
            markers=['o','+','*','.','d','|','h','H','p','s']
            clr=open("colors.txt","r").readlines()
            clr=[c.strip() for c in clr]
            count=0
            for k,v in data.items():
                        values=np.sort(v)
                        cdf=self.cdffromlist(v)
                  cl=clr[count]
                        plt.plot( values,cdf,color=cl,label="%s"%k)
                        count+=1
            plt.margins(0.05,0.05)
            plt.grid(True)
            plt.xlabel(xlabel)
            plt.ylabel(ylabel)
           #plt.xlim(0,12)    #xlim of start_tranfertime
           #plt.xlim(-21,1)   #xlim for RSRQ
            plt.title("")
            plt.legend(loc='lower right',prop={'size':10})
            plt.savefig("%s.pdf"%filename)
            plt.close("all")

def set_box_color(self,bp, color):
   plt.setp(bp['boxes'], color=color)
   plt.setp(bp['whiskers'], color=color)
   plt.setp(bp['caps'], color=color)
   plt.setp(bp['medians'],linewidth=1, color='k')

def groupboxplots(self,xlabel,ylabel,title,filename,telenor,netcom):
   labels=["Telenor", "Netcom"]
   plt.figure()
   bp1=plt.boxplot(telenor, positions=np.array(xrange\
             (len(telenor)))*2.0-0.4,widths=0.6 ,notch=True,\
             patch_artist=True)
   bp2=plt.boxplot(netcom, positions=np.array(xrange\
              (len(netcom)))*2.0+0.4,widths=0.6 ,notch=True,\
              patch_artist=True)
       self.set_box_color(bp2, 'm')
```

```
        plt.plot([], c='c', label='ingress point delay')
        plt.plot([], c='m', label='end-to-end delay')
    plt.legend(loc='upper right',prop={'size':10})
            plt.xticks(xrange(0, len(labels) * 2, 2),labels)
            plt.xlim(-2, len(labels)*2)
            plt.title(title)
            plt.xlabel(xlabel)
            plt.ylabel(ylabel)
            plt.savefig("%s.pdf"%filename)
            plt.close("all")

def groupboxplotsregion(self,xlabel,ylabel,title,filename,d1,d2):
    labels=["Bergen","Oslo","Trondheim"]
    plt.figure()
        print labels
    bp1=plt.boxplot(d1, positions=np.array(xrange(len(d1)))*2.0-0.4,\
                widths=0.6 ,notch=True,patch_artist=True)
    bp2=plt.boxplot(d2, positions=np.array(xrange(len(d2)))*2.0+0.4, \
                    widths=0.6 ,notch=True, patch_artist=True)
    self.set_box_color(bp1, 'c')
        self.set_box_color(bp2, 'm')

        plt.plot([], c='c', label='ingress point delay')
        plt.plot([], c='m', label='end-to-end delay')
    plt.legend(loc='upper right',prop={'size':10})
            plt.xticks(xrange(0, len(labels) * 2, 2),labels)
            plt.xlim(-2, len(labels)*2)
            plt.title(title)
            plt.xlabel(xlabel)
            plt.ylabel(ylabel)
            plt.yscale("log")
            plt.savefig("%s.pdf"%filename)
            plt.close("all")

def boxplot(self,xlabel,ylabel,filename,labels,listoflists):
    data_to_plot=listoflists
    bp1=plt.boxplot(data_to_plot,positions=np.array(xrange(len\
              (data_to_plot)))*2.0-0.3, widths=0.6 ,notch=True,\
              patch_artist=True)
          self.set_box_color(bp1, 'm')

    for line in bp1['medians']:
            x, y = line.get_xydata()[1]
            plt.text(x, y, '%.1f' % y, horizontalalignment='center')

            plt.margins(0.5,0.5)
            plt.xticks(xrange(0, len(labels) * 2, 2),labels)
            plt.title("")
            plt.xlabel(xlabel)
            plt.ylabel(ylabel)
            plt.yscale("log")
                    plt.savefig("%s.pdf"%filename)
            plt.close("all")
```

makeplots.py

```
#!/usr/bin/env python
import os
from  plots import plots
import numpy as np
from scipy.stats import norm
from collections import defaultdict
from collections import Counter
import collections

dir="traceroutegraphs/"
curlpath="httpgraphs/"
class runplotsClass:

      def __init__(self):
           if not os.path.exists(dir):
              os.makedirs(dir)
           if not os.path.exists(curlpath):
              os.makedirs(curlpath)
           self.path=dir
           self.curlpath=curlpath
       self.plots=plots()

      def mean_sd_percentile(self,dataset):
           min=self.formatratio(np.min(dataset))
           max=self.formatratio(np.max(dataset))
           mean=self.formatratio(np.mean(dataset,axis=0))
           median=self.formatratio(np.median(dataset))
           sd= self.formatratio(np.std(dataset,axis=0))
           sd= self.formatratio(np.var(dataset,axis=0))
           Q1=self.formatratio(np.percentile(dataset,25))
       Q2=self.formatratio(np.percentile(dataset,75))
           Q3=self.formatratio(np.percentile(dataset,95))
           print "size: %d\n"%len(dataset)
           res="%s & %s & %s & %s & %s & %s & %s & %s"% \
            (min,max,mean,median,sd,Q1,Q2,Q3)
           return res

      def tep(self,file):
           f=self.curlfile(file)
           dic=self.curlstarttotalregions(file)
           print file
           for k,v in dic.items():
                 size=len(v)
                 per20=self.formatratio(np.percentile(v,20))
                     per80=self.formatratio(np.percentile(v,80))
                 print k,size,per20,per80

           w=[]
           t=[]
           with open(f,"r") as f:
```

```
          lines=f.readlines()
          for line in lines:
               l=line.split('|')
               si=l[1].strip()
               st=l[3].strip()
               if(float(st))< 0.26:
                   w.append(si)
               if(float(st))> 0.85:
                   t.append(si)

def operatoranalytical(self,filename):
           data=self.readingressdelayallregions(filename)
           ingress=[]
           provider=[]
           for d in data[0]:
                  ingress.append(self.mean_sd_percentile(d))
           for d in data[1]:
                   provider.append(self.mean_sd_percentile(d))

           res=(ingress,provider)
           return res

def formater(self,str):
       f= "%.2f"%(float(str)/1000)
       return float(f)

def filename(self,filename):
       file="%s%s"%(self.path,filename)
       return file

def curlfile(self,filename):
       file="%s%s"%(self.curlpath,filename)
       return file

def readfile(self,file):
          data=[]
          with open(self.filename(file),"r") as f:
            lines=f.readlines()
            labels=[x.split('|')[0] for x in lines]
            ingress=[self.formater(x.split('|')[1]) for x in lines]
            contentprovider=[self.formater(x.split('|')[2])\
                     for x in lines]
      return (labels,ingress,contentprovider)

def formatratio(self,i):
          str="%.2f"%float(i)
          return float(str)

def readingressdelayallregions(self,file):
            oi=[]
            bi=[]
            ki=[]
            ti=[]
            op=[]
```

```
            bp=[]
            kp=[]
            tp=[]
            lines=open(self.filename(file),"r").readlines()
            for l in lines:
                  l=l.strip().split('|')
                  if "Oslo" in l[0]:
                          oi.append(self.formater(l[1]))
                          op.append(self.formater(l[2]))
                  elif "Bergen" in l[0]:
                          bi.append(self.formater(l[1]))
                          bp.append(self.formater(l[2]))
                  elif "Trondheim" in l[0]:
                          ti.append(self.formater(l[1]))
                          tp.append(self.formater(l[2]))
            ingress=[bi,oi,ti]
            provider=[bp,op,tp]
            return (ingress,provider)

def divide(self,a,b):
       d= float(a)/float(b)
       return str(d)
def delayfractionregions(self,file):
    o=[]
    b=[]
    k=[]
    t=[]
    lines=open(self.filename(file),"r").readlines()
    for l in lines:
                  l=l.strip().split('|')
                  if "Oslo" in l[0]:
                          o.append(self.formatratio(\
                          self.divide(l[1],l[2])))
                  elif "Bergen" in l[0]:
                          b.append(self.formatratio(\
                          self.divide(l[1],l[2])))
                  elif "Trondheim" in l[0]:
                          t.append(self.formatratio(\
                          self.divide(l[1],l[2])))

            o=np.sort(o)
            b=np.sort(b)
            k=np.sort(k)
            t=np.sort(t)
    fraction=(b,o,t)
        return fraction

def curlstarttotalregions(self,file):
            st=defaultdict(list)
            td=defaultdict(list)
                    lines=open(self.curlfile(file),"r").readlines()
                    for l in lines:
                                l=l.strip().split('|')
                          node=l[0].strip()
```

```
                        sttransfer=l[3].strip()
                    st[node].append(float(sttransfer))
            ost=collections.OrderedDict(sorted(st.items()))
    return ost


def timeconnect(self,file):
    dict=defaultdict(list)
    lines=open(self.curlfile(file),"r").readlines()
    for l in lines:
          l=l.strip().split('|')
          node=l[0].strip()
          site=l[1].strip()
          s=l[3].strip() # IN mreged 3G_4G files is 2
          c=l[2].strip() # IN mreged 3G_4G files is 2
          #frac=float(c)/float(s)
          if (float(c)>float(s)):
              dict[node].append(s)
    orssi=collections.OrderedDict(sorted(dict.items()))
    return orssi

def RSSI(self,file):
    dict=defaultdict(list)
    lines=open(self.curlfile(file),"r").readlines()
    for l in lines:
          l=l.strip().split('|')
          node=l[0].strip()
          ASU=l[2].strip() # IN mreged 3G_4G files is 2
          if ASU <> 'NULL':
              RSSI=(2*int(ASU))-113
             dict[node].append(RSSI)
    orssi=collections.OrderedDict(sorted(dict.items()))
    return orssi


def MSRP(self,file):
    dict=defaultdict(list)
    lines=open(self.curlfile(file),"r").readlines()
    for l in lines:
          l=l.strip().split('|')
          node=l[0].strip()
          msrp=l[2].strip() # IN mreged 3G_4G files is 2
          if msrp <> 'NULL':
             dict[node].append(int(msrp))
    orssi=collections.OrderedDict(sorted(dict.items()))
    return orssi

def MSRQ(self,file):
    dict=defaultdict(list)
    lines=open(self.curlfile(file),"r").readlines()
    for l in lines:
          l=l.strip().split('|')
          node=l[0].strip()
          msrq=l[3].strip() # IN mreged 3G_4G files is 2
```

```
            if msrq <> 'NULL':
               dict[node].append(int(msrq))
        orssi=collections.OrderedDict(sorted(dict.items()))
        return orssi

    def MRSCP(self,file):
        dict=defaultdict(list)
        lines=open(self.curlfile(file),"r").readlines()
        for l in lines:
              l=l.strip().split('|')
              node=l[0].strip()
              msrp=l[1].strip() # IN mreged 3G_4G files is 2
              if msrp <> 'NULL':
                 dict[node].append(msrp)
        orssi=collections.OrderedDict(sorted(dict.items()))
        return orssi


    def mediansignals(self,file,index):
        dict=defaultdict(list)
        lines=open(self.curlfile(file),"r").readlines()
        for l in lines:
              l=l.strip().split('|')
              node=l[0].strip()
              val=l[index].strip()
              val=int(val)
              #vv=(2*val)-113
              dict[node].append(int(val))
        co=collections.OrderedDict(sorted(dict.items()))
        labels=[]
        values=[]
        for k,v in co.items():
            if len(v)>1000:
                  labels.append(k)
                  values.append(np.median(v))
         return (labels,values)

    def operatorAllregionsbotxplot(self,filename,fileread,title):
          filenametosave=self.filename("%s"%filename)
        data=self.readingressdelayallregions(fileread)
        self.plots.groupboxplotsregion("Regions","RTT delay\
       in seconds" ,title,filenametosave,data[0],data[1])


    def Allregionsfractionplot(self,filename,fileread,title):
          filenametosave=self.filename("%s"%filename)
        data=self.delayfractionregions(fileread)
        self.plots.fractiondelayplot("First hop delay ratio\
                 to end-to-end delay"  ,"Fraction of experiments"\
                 ,title,filenametosave,data)

    def curlstarttransferdboxplot(self,filename,fileread):
          filenametosave=self.curlfile("%s"%filename)
        data=self.curlstarttotalregions(fileread)
```

```
    labels=[]
    values=[]
    for k,v in data.items():
        if len(v)>1000:
              labels.append(k)
              values.append(v)
     self.plots.boxplot("NNE nodes"," Start_transfer_time\
          (in seconds)" ,filenametosave,labels,values)


def curlstarttimefractionplot(self,filename,fileread):
      filenametosave=self.curlfile("%s"%filename)
    dict=self.curlstarttotalregions(fileread)
    data={}
    for k,v in dict.items():
        if len(v)>1000:
              data[k]=v
    self.plots.CDFplot("Start transfer time ","Fraction \
             of experiments",filenametosave,data)

def curltimeconnectboxplot(self,filename,fileread):
      filenametosave=self.curlfile("%s"%filename)
    dict=self.timeconnect(fileread)
    labels=[]
    values=[]
    for k,v in dict.items():
        if len(v)>1000:
            print k, v
              labels.append(k)
              values.append(v)
     self.plots.boxplot("NNE nodes"," time_to_connect (in seconds)"\
            ,filenametosave,labels,values)


def timetoconnectplots(self,filename,fileread):
      filenametosave=self.curlfile("%s"%filename)
    dict=self.MSRP(fileread)
    data=dict
      data={}
    for k,v in dict.items():
        if len(v)>1000:
              data[k]=v
    self.plots.CDFplot("Time to connect ratio to time to start\
        transfer","Fraction of experiments",filenametosave,data)

def RSSICDFplots(self,filename,fileread):
          filenametosave=self.curlfile("%s"%filename)
    dict=self.RSSI(fileread)
        data=dict
          data={}
        for k,v in dict.items():
           if len(v)>1000:
                 data[k]=v
            self.plots.CDFplot("RSSI","Fraction of \
```

```
                                experiments",filenametosave,data)
          print "%s crated\n"%filenametosave

      def MSRPCDFplots(self,filename,fileread):
                filenametosave=self.curlfile("%s"%filename)
          dict=self.MSRP(fileread)
              data=dict
                data={}
              for k,v in dict.items():
                 if len(v)>1000:
                         data[k]=v
                  self.plots.CDFplot("RSRP","Fraction of \
                                experiments",filenametosave,data)
          print "%s crated\n"%filenametosave

      def MSRQCDFplots(self,filename,fileread):
                filenametosave=self.curlfile("%s"%filename)
          dict=self.MSRQ(fileread)
              data=dict
                data={}
              for k,v in dict.items():
                  if len(v)>1000:
                         data[k]=v
                  self.plots.CDFplot("time to connect","Fraction of \
                               experiments",filenametosave,data)
          print "%s crated\n"%filenametosave

if __name__=="__main__":
     obj=runplotsClass()
         # in order to run plots call the functions making plot
         #obj.MSRQCDFplots(......)
```

database.py

```
#!/usr/bin/env python
import xml.etree.ElementTree as et
import sqlite3 as lite
import subprocess
import os
import re
import time
from cymruwhois import Client
from collections import OrderedDict
import json
import shutil
import ast

path="traceroutefiles/"
pathtomove="/Users/dikon/Documents/plots/finishedfiles/"


class extractor:
```

```
def __init__(self):
    self.path=path
    #self.tracefiles=allfiles

def conn(self):
    return lite.connect('experiments.db')

def createHTML(self,filename):
     html="""
         <html>
            <head>
              <style>
               .TableRow td{
                       border-bottom: 2px solid rgb(167,167,167);
                         border-top: 2px solid rgb(167,167,167);
                }
                .TableRaw tr{
                       margin-top:5px;
                }
                span {margin-left: 4px;}
             </style>
            </head>
            <body>
              <center>
                    <h1>%s</h1>
                    <table cellspacing="1">
                      %s
                    </table>
              </center>
            </body>
         </html>
         """
     result=self.internetpath(filename)
     with open(filename+".html","w") as f:
         f.write(html%(filename,result))
         f.close()

def internetpath(self,file):
    str=''
    with open(file,'r') as f:
         lines=f.readlines()
         for l in lines:
             str+='<tr class="TableRow">'
              sp=l.strip().split('|')
             site=sp[1]
             str+="<td>%s</td>"%site
             tracepath=eval(sp[2])
             str+='<td>'
             for k,v in tracepath.iteritems():
                 if not 'NA' in k:
                    q=self.queryTable("select asn,owner from \
                                      whois where asn=%s"%k )
                    str+="<span>%s : </span><span>%s ;</span><span>\
                          %s hops</span><br/>"%(q[0],q[1],v)
```

```
                str+="</td>"
                str+="</tr>"
        return str

    def nodeoperatorsmapping(self):
        lines=self.readfile("cooperators.txt")
        m={}
        for l in lines[1:]:
              line=l.split()
              if line[3]=="1":
                m[line[0]]=(line[1],line[2],'Telenor',line[4])
              elif line[3]=="2":
                m[line[0]]=(line[1],line[2],'Netcom',line[4])

        return m
def mapper(self,key):
         m=self.nodeoperatorsmapping()
         return m[key]

    def createTraceTable(self):
        con=self.conn()
        with con:
             cur=con.cursor()
             cur.execute("DROP TABLE IF EXISTS traceroute")
             cur.execute("CREATE TABLE traceroute (M_Id INT,\
                 timestamp TEXT,node INT,region TEXT,ISP TEXT, \
                 connection TEXT, website TEXT,asn_occurr TEXT,\
                 ingDelay REAL, provDelay REAL, \
                  hops TEXT, delays TEXT);")
             print "Table traceroute createds"

    def createWhoisTable(self):
        con=self.conn()
        with con:
             cur=con.cursor()
             cur.execute("DROP TABLE IF EXISTS whois")
             cur.execute("CREATE TABLE whois (ip TEXT,asn INT,owner TEXT);")
             print "Table whois createds"


    def formatdelay(self,ms):
        list=[float(l.strip()) for l in ms]
        minimum= min(list)
        return str(minimum)

    def hopsoccurence(self,rsn):
         uniqASN={}
        str=""
        if r.asn not in uniqASN:
            uniqASN[r.asn]=1
        else:
            uniqASN[r.asn]+=1
        return uniqASN
```

```
def whoisMany(self,iplist):
     c=Client()
     ips=[x for x in iplist if not x.startswith('10') and \
         not x.startswith('192.168')]
      uniqASN=OrderedDict()
     str=""
     for ip in ips:
          raws=self.fetchWhois(ip)
          if raws == 0:
             whois=self.whoisSub(ip)
             if whois <> 0:
                asn=whois[0]
                ip=whois[1]
                owner=whois[2]
                  if asn not in uniqASN:
                   uniqASN[asn]=1
                else:
                    uniqASN[asn]+=1
                self.insertTableWhois(ip,asn,owner)
          else:
                asn=raws[1]
                ip=raws[0]
                owner=raws[2]
                  if asn not in uniqASN:
                   uniqASN[asn]=1
                else:
                    uniqASN[asn]+=1
     res= json.dumps(uniqASN)
     return res

def whoisSub(self,ip):
     cmd=[]
     cmd.append("whois")
     cmd.append("-h")
     cmd.append("whois.cymru.com")
     #cmd.append(" -v ")
     cmd.append(ip)
     sub=subprocess.Popen(cmd ,stdout=subprocess.PIPE)
     res= sub.stdout.readlines()
     if len(res)>1:
             sp= res[1].split('|')
             asn=sp[0].strip()
             ip=sp[1].strip()
              owner=sp[2].strip().replace('\n','')
             tup=(asn,ip,owner)
             #print tup
             return tup
     else:
         return 0
     sub.wait()

 def searchhopsanddelay(self,hoplines):
     ip_pattern=r"\d+\.\d+\.\d+\.\d+"
     delay_pattern=r"\s\d+\.?\d+\s"
```

```
    tup=()
    both=[]
    dict={}
    #hops=hoplines
    hops=hoplines.split("\n")
    for line in hops[2:]:
          m1=re.search(ip_pattern,line)
          m2=re.findall(delay_pattern,line)
          if m1 and m2:
                 ip=m1.group()
                  minvalue=self.formatdelay(m2)
                 tup=(ip,minvalue)
                 both.append(tup)
                 dict[ip]=minvalue
    return (both,dict)


def insertTableTrace(self,tuples):
    con=self.conn()
    with con:
         cur=con.cursor()
         cur.executemany("INSERT INTO traceroute \
                VALUES(?,?,?,?,?,?,?,?,?,?,?,?)",(tuples,))
         #print "New raws added to traceroute"

def insertTableWhois(self,ip,asn,owner):
    con=self.conn()
    with con:
         cur=con.cursor()
         cur.execute("INSERT INTO whois VALUES(?,?,?)",(ip,asn,owner))
         #print "New raws added to whois"

def fetchWhois(self,query):
    con=self.conn()
    with con:
         con.row_factory = lite.Row
         cur=con.cursor()
         cur.execute("SELECT * FROM whois WHERE ip=?",(query,))
         raw=cur.fetchone()
         if raw == None:
             return 0
         else:
             return raw


def commitdb(self):
      self.conn().commit()
     self.conn().close()

def readfile(self,filename):
    f=open(filename,"r")
    lines=f.readlines()
    return lines
```

```
def xmlparser(self,filename):
    parser= et.parse(filename)
    return parser.getroot()
    #return parser

def siteAndhopstext(self,filename):
    root=self.xmlparser(filename)
    dict={}
    for child in root.iter("d"):
        site=child.find('site').text
        tracehops=child.find('traceroute').text
        dict[site]=tracehops
    return dict

def getAll(self,filename):
    tup=()
    sp= filename.split("_")
    measureid=sp[1]
    timestamp=sp[2]
    map=self.mapper(measureid)
    node=map[0]
    region=map[1]
    ISP=map[2]
    connection=map[3]
    tracedata=self.siteAndhopstext(filename)
    for site,tracehops in tracedata.iteritems():
        if len(tracehops)>100:
            hopsanddelay=self.searchhopsanddelay(tracehops)
            both=hopsanddelay[0]
            print len(both)
            if len(both)>0:
                keyvals=hopsanddelay[1]
                hops=[x[0] for x in both]
                delays=[x[1] for x in both]
                ingressdelay=delays[0]
                providerdelay=delays[-1]
                allhops=','.join(hops)
                alldelays=','.join([str(d) for d in delays])
                list=[]
                whoisinfo=self.whoisMany(hops)
                #whoisstr= whoisinfo[0]
                asnoccurrences=  whoisinfo
                tuples=(int(measureid),timestamp,int(node),region,ISP\
                   ,connection,site,asnoccurrences,float(ingressdelay),\
                   float(providerdelay),allhops,alldelays)
                self.writeout(tuples)
                self.insertTableTrace(tuples)
                self.commitdb()
                print "New rows added to traceroute table\n"
    self.logger(filename)
    time.sleep(1)

def movefile(self,p1,p2):
    shutil.move(p1,p2)
```

```
    def logger(self,filename):
        with open("logs.txt","a") as f:
             str="%s is completed.\n" %filename
             f.write(str)
             print str
        f.close()

    def writeout(self,tuple):
        with open("finalltraces.txt","a") as f:
             print >> f, tuple
        f.close()

    def queryTable(self,query):
        cursor=self.conn().cursor()
        try:
            cursor.execute(query)
            #results=cursor.fetchall()
            results=cursor.fetchone()
            if results <> None:
                return results
        except:
            print "Error: unable to fetch data by \
                   executing the query: %s"%query
            #self.db().close()

    def dowork(self):
        counter=1
        for root,dirs,files in os.walk(self.path):
            for f in files:
                if f.startswith("trace"):
                   print f , counter
                   tf=os.path.join(root,f)
                   self.getAll(tf)
                   counter+=1
                   p2="%s%s"%(pathtomove,f)
                   self.movefile(tf,p2)
    def run(self):
        self.dowork()


obj=extractor()
#obj.tabledata()
#obj.createTraceTable()
#obj.createWhoisTable()

#obj.run()
#obj.createHTML("telenor.internetpath")
#obj.createHTML("netcom.internetpath")
```

Bibliography

[1] Adrio Communications Ltd : *Resource and analyse for electronic engineers*. URL: http://www.radio-electronics.com/info/cellulartelecomms/umts/umts_wcdma_tutorial.php.

[2] Mysql AB: *relational database management system*. URL: http://www.mysql.com.

[3] John P.Rula et al. 'Behind the curtain-Cellular DNS and content replica selection'. In: *Internet Computing, IEEE* 5.5 (Sept. 2001). ISSN: 1089-7801. DOI: 10.1109/4236.957902.

[4] Kyriakos Zarifis et al. *Diagnosing path Inflation of mobile client traffic*. Tech. rep. Juni. 2012.

[5] *Alexa Internet Inc*. 1996 - 2015. URL: http://www.alexa.com.

[6] M. Amirijoo et al. 'Neighbor cell relation list and measured cell identity management in LTE'. In: *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. Apr. 2008, pp. 152–159. DOI: 10.1109/NOMS.2008.4575129.

[7] B. Augustin, T. Friedman and R. Teixeira. 'Multipath tracing with Paris traceroute'. In: *End-to-End Monitoring Techniques and Services, 2007. E2EMON '07. Workshop on*. 2007, pp. 1–8. DOI: 10.1109/E2EMON.2007.375313.

[8] Dziugas Baltrunas, Ahmed Elmokashfi and Amund Kvalbein. 'Measuring the Reliability of Mobile Broadband Networks'. In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. IMC '14. Vancouver, BC, Canada: ACM, 2014, pp. 45–58. ISBN: 978-1-4503-3213-2. DOI: 10.1145/2663716.2663725. URL: http://doi.acm.org/10.1145/2663716.2663725.

[9] Mudit Ratana Bhalla and Anand Vardhan Bhalla. 'Generations of mobile wireless technology: A survey'. In: *International Journal of Computer Applications (0975–8887)* 5.4 (2010).

[10] S. Branigan et al. 'What can you do with Traceroute?' In: *Internet Computing, IEEE* 5.5 (Sept. 2001), pp. 96–. ISSN: 1089-7801. DOI: 10.1109/4236.957902.

[11] Team Cymru. *Whois:* URL: http://www.team-cymru.org/IP-ASN-mapping.html#whois.

[12] Leslie Daigle. 'WHOIS protocol specification'. In: (2004).

[13] Wei Dong, Zihui Ge and Seungjoon Lee. '3G meets the internet: understanding the performance of hierarchical routing in 3G networks'. In: *Proceedings of the 23rd International Teletraffic Congress*. International Teletraffic Congress. 2011, pp. 15–22.

[14] Benoit Donnet et al. 'Revealing MPLS tunnels obscured from traceroute'. In: *ACM SIGCOMM Computer Communication Review* 42.2 (2012), pp. 87–93.

[15] K. Doppler et al. 'Device-to-device communication as an underlay to LTE-advanced networks'. In: *Communications Magazine, IEEE* 47.12 (Dec. 2009), pp. 42–49. ISSN: 0163-6804. DOI: 10.1109/MCOM.2009.5350367.

[16] T. Dreibholz and E.G. Gran. 'Design and Implementation of the NORNET CORE Research Testbed for Multi-homed Systems'. In: *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*. Mar. 2013, pp. 1094–1100.

[17] Thomas Dreibholz. *NorNet – An Open, Large-Scale Testbed for Multi-Homed Systems*. Invited Talk at Swinburne University, Centre for Advanced Internet Architectures (CAIA). Melbourne, Victoria/Australia, 30th Jan. 2014. URL: https://www.simula.no/sites/www.simula.no/files/publications/files/caia2014-presentation-web.pdf.

[18] Denis Fawler. 'The last mile: making the broadband connection'. In: *ACM New York, NY, USA* 4 (Mars. 2000). ISSN: 1091-3556 EISSN. DOI: 10.1145/330894.330900.

[19] Forough Golkar. 'Measuring and Comparing the Stability of Internet Paths over IPv4 & IPv6'. In: (2014).

[20] F Gont. 'ICMP Attacks against TCP'. In: *Internet Engineering Task Force (IETF)* (July. 2010). ISSN: 2070-1721.

[21] Ernst Gunnar Gran, Thomas Dreibholz and Amund Kvalbein. 'NorNet Core – A multi-homed research testbed'. In: *Computer Networks* 61 (2014). Special issue on Future Internet Testbeds – Part I, pp. 75–87. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.bjp.2013.12.035. URL: http://www.sciencedirect.com/science/article/pii/S1389128613004489.

[22] Rahul Hiran, Niklas Carlsson and Phillipa Gill. 'Characterizing large-scale routing anomalies: a case study of the China Telecom incident'. In: *Passive and Active Measurement*. Springer. 2013, pp. 229–238.

[23] John D. Hunter. *Matplotlib*. URL: http://matplotlib.org.

[24] Icina: *System monitoring*. URL: https://www.icinga.org/.

[25] ICMP: *(Internet Control Message Protocol)*. January. 2015. URL: http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[26] Cisco Inc: *Understanding the Ping and Traceroute Commands*. November. 2006. URL: http://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-software-releases-121-mainline/12778-ping-traceroute.html.

[27] Cisco Inc: *Visual Networking Index*. URL: http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html.

[28] Cisco Inc: *Visual Networking Index*. February. 2015. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html?CAMPAIGN=VNI+2014&COUNTRY_SITE=us&CREATIVE=PR+to+Mobile+VNI+WP&POSITION=PR&REFERRING_SITE=Press+release.

[29] Amund Kvalbein et al. 'The Nornet Edge platform for mobile broadband measurements'. In: *Computer Networks* 61 (2014). Special issue on Future Internet Testbeds – Part I, pp. 88–101. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.bjp.2013.12.036. URL: http://www.sciencedirect.com/science/article/pii/S1389128613004490.

[30] Puppet Labs: *Configuration management*. URL: https://puppetlabs.com/.

[31] G. Lui et al. 'Differences in RSSI readings made by different Wi-Fi chipsets: A limitation of WLAN localization'. In: *Localization and GNSS (ICL-GNSS), 2011 International Conference on*. June 2011, pp. 53–57. DOI: 10.1109/ICL-GNSS.2011.5955283.

[32] Carlos d M. Cordeiro et al. *The last mile: Wireless Technologies for Broadbands and home Network*. Tech. rep. University of Cincinnati, OH-USA: Center for distributed and Mobile communication, ECECS, 2005.

[33] Nagios: *System monitoring*. URL: http://www.nagios.org/.

[34] *Norway Map*. URL: http://mapsof.net/map/norway-municipalities.

[35] Jitu Padhye and Henrik Frystyk Nielsen. *A comparison of SPDY and HTTP performance*. Tech. rep. Citeseer, 2012.

[36] Guido van Rossum. *Python*. URL: http://www.python.org.

[37] Liia Sarjakosk. 'Challenges of Mobile Peer-to-Peer Applications in 3G and MANET Environments'. In: *Publications in Telecommunications Software and Multimedia Teknillisen korkeakoulun tietoliikenneohjelmistojen ja multimedian julkaisuja* (Spring 2005). ISSN: 1455-9749.

[38] Abhi Sharma. 'An intro to Cellular communication Generations'. In: *Open Scholar Library* (2013). URL: https://www.academia.edu/3099956/Generations_of_Wireless_Communication._From_0G_to_5G_Abhi.

[39] NorNet Edge Simula. *Resilient Networks Mobile Broadband Measurements*. URL: http://robustenett.no/mape.

[40] Daniel Stenberg: *cURL*. 1977. URL: http://curl.haxx.se.

[41] Daniel Stenberg: *cURL*. 1977. URL: http://curl.haxx.se/docs/manpage.html.

[42] Pratik Sule and Anish Joshi. 'Architectural Shift from 4G to 5G Wireless Mobile Networks'. In: (2014).

[43] Paris Traceroute: *Paris Traceroute*. URL: http://www.paris-traceroute.net.

[44] J. Uribe and W. Fu. *Receiver signal strength indicator*. US Patent 7,630,695. Dec. 2009. URL: https : / / www . google . com / patents / US7630695.

[45] NorNet visualizaiton: *Resilient Networks Mobile Broadband Measurements*. URL: http://robustenett.no/map.

[46] Kimio Watanabe and Mamoru Machida. 'Outdoor LTE Infrastructure Equipment (eNodeB)'. In: *FUJITSU Sci. Tech. J* 48.1 (2012), pp. 27–32.