State management in coreless mobile networks

Frans Ojala

Masters thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, November 24, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Laitos — Institution — Department Department of Computer Science
Tekijä — Författare — Author Frans Ojala	
Työn nimi — Arbetets titel — Title State management in coreless mobile networks	
Oppiaine — Läroämne — Subject Computer Science	

Työn laji — Arbetets art — Level Masters thesis	Aika — Datum — Month and year November 24, 2016	Sivumäärä — Sidoantal — Number of pages 67+11

Tiivistelmä — Referat — Abstract

The number of mobile Internet users has skyrocketed and with the advent of the Internet of things we are reaching the limits of the current telecommunications standard 4G. The improvements and goals set for the next standard, 5G, are not trivial and research is in progress to reach them. Improvements across all involved technology fields is needed.

In this thesis we present a novel mobile network architecture—coreless mobile networks—and develop state management concepts, which we base on the analysis of the current 4G/LTE architecture. The coreless mobile network focuses on the redesign of the state management in mobile networks, more precisely, removal of state from 4G core network entities into an eternal ubiquitous data store. The architecture follows trends in current research, particularly network function virtualisation, software defined networking and mobile edge computing.

The new network architecture requires a data storage solution that is capable of functioning as the state store in the mobile network environment. Thus, we present an overview of promising data stores and evaluate their suitability. Further in this thesis we present the results of benchmarking the Apache Geode data store, as an example of a state management solution that could be leveraged in realising the coreless mobile network architecture. We discovered that the Apache Geode data store is, depending on configuration, capable of delivering the data model, consistency, high availability, scaling, throughput and latency that are required in our proposed architecture.

ACM Computing Classification System (CCS):
- Information systems~Distributed storage
- Information systems~Hierarchical storage management
- Networks~Middle boxes / network appliances
- Networks~Mobile networks

Avainsanat — Nyckelord — Keywords
4G/LTE, VNF, virtualisation, SDN, 5G, network architecture, distributed storage, benchmark

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

Abbreviations

The following abbreviations are used throughout the document.

3GPP: 3rd Generation Partnership Project
LTE: Long Term Evolution
SAE: System Architecture Evolution
GSM: Global System for Mobile Communications
GPRS: General Packet Radio Service
RAT: Radio Access Technology
RAN: Radio Access Network
(E)-UTRAN: (Evolved) Universal Terrestrial Radio Access Network
PLMN: Public Land Mobile Network
RRC: Radio Resource Control
EPS: Evolved Packet System
EPC: Evolved Packet Core
UE: User Equipment
EMM: Evolved Mobility Management
TA: Tracking Area
TAI: Tracking Area Identifier
GUTI: Globally Unique Temporary Identifier
C-RNTI: Cell Network Temporary Identifier
S1AP UE ID: S1 Application Protocol UE Identifier
ECGI: E-UTRAN Cell Global Identifier
NAS: Non-Access Stratum
AS: Access Stratum
TEID: Tunnel Endpoint Identifier
APN: Access Point Name
E-RAB: E-UTRAN Radio Access Bearer
DRB: Data Radio Bearer
UL/DL: Upload / Download
eNB: Evolved Node B
PDN: Packet Data Network
MME: Mobility Management Entity
S/P GW: Serving / Packet Data Gateway
HSS: Home Subscriber Server
PCRF: Policy and Charging Rules Function
IMSI: International Mobile Subscriber Identifier
TAU: Tracking Area Update
D2D, M2M: Device to Device, Machine to Machine (communication)
C-RAN: Cloud Radio Access Network
MIMO: Multiple Input Multiple Output
VNF: Virtualised Network Functions
SDN: Software Defined Networking
IoT: Internet of Things
MEC: Mobile Edge Computing
AD/DA: Analogue to Digital / Digital to Analogue (converter)
IP: Internet Protocol (address)
VoIP: Voice over IP
FTP: File Transfer Protocol
QoS: Quality of Service
ARP: Allocation and Retention Policy
AMBR: Aggregate Maximum Bitrate
QCI: QoS Class Identifier
TFT: Traffic Filter Template
CAPEX / OPEX: Capital Expenditure / Operational Expenditure
YCSB: *Yahoo!* Cloud Serving Benchmark

List of figures

List of tables

Appendices

1 Introduction

Wireless communication methods have stood the test of time. They have evolved from the text transmitting telegraph into the all-around computing platform in our pockets, the smartphone. Since the turn of the millennium the number of private individuals owning and using smartphones has skyrocketed. Presently there are about 3 billion smartphone subscriptions with the number expected to grow upwards of 6.1 billion by 2020.[1]

In the advent of the Internet of Things the number of connected devices will further increase by orders of magnitude.[2][3] 80% of traffic from smartphones is estimated to be data traffic to and from the Internet, increasing from today's 2.4 GB/month to 14 GB/month per smartphone in the USA alone.[1] IoT devices are expected to be generating traffic solely into the Internet primarily via radio communication. With these trends in sight it is foreseeable that the current state of wireless communication cannot support the amount of users and traffic [21]. To ameliorate the congested 3G and 4G mobile data networks much research is being poured into the developing of the next generation wireless 5G communication systems [45].

The pressure on 5G is as high as are expectations. The technical requirements planned for 5G in contrast to 4G/LTE systems contain improvements in several fields [41]:

- 1000 times higher data volume (per area),
- 10 - 100 times higher user data rate,
- 100 - 1000 time more connected devices,
- 10 times longer battery life (low power devices),
- 5 times reduced end-to-end latency.

Additionally in order to reach the goals and allow further development, according to Nokia,[4] the future 5G networks need to integrate multiple radio frequency access technologies, support heterogeneous backhaul and edge computing, virtualise core and radio access network functions, enable SDN technology and security in the network architecture from the very beginning.

Current proposals to combat the need for capacity include, but are not limited to, spectrum reuse and new spectra (including mm-wave) [32], multi-tier networks [36], device-to-device (D2D) communication [23], Cloud Radio Access Networks (C-RAN) [23] and massive Multiple Input Multiple Output (MIMO) [54] technologies. Latency improvements are sought from C-RAN, D2D and full-duplex communication, while network densification can be

[1]`https://www.ericsson.com/news/1925907` *(Referenced on 28.10.2016)*

[2]`http://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts/` *(Referenced on 8.11.2016)*

[3]`http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html` *(Referenced on 28.10.2016)*

[4]`http://resources.alcatel-lucent.com/asset/200004` *(Referenced on 28.10.2016)*

enabled by heterogeneous and multi-tier networks. Advanced services and applications, such as network caching and on-line gaming, can be supported by C-RAN, virtualised network functions (VNF) and machine-to-machine (M2M) communication. Battery life can be increased by wireless charging, energy harvesting and protocol redesign. IoT requires that the network support autonomous devices and applications, which can be reached with M2M communication and self-organising and cognitive networks created by software defined networking (SDN) [22].

A key problem on the road to 5G is the architecture that the current mobile wireless networking solution, 4G/LTE, contains as the core principle: the evolved packet core (EPC). The EPC is a resource constrained set of primitive network functions that together orchestrate the connectivity of user equipment (UE) and other devices. By analysing the roles and functions of the current EPC architecture we realised that the functions are convoluted and overlapping. More specifically, contrasting to SDN paradigms, the EPC contains mixed control plane and user plane functions [40].

Just one of the effects of this mix is a congested control plane. The signalling load of the EPC has been widely analysed and many solutions have been proposed, ranging from intelligent Tracking Area (TA) optimisation to protocol redesign particularly in the transition from 3G to 4G specification [4]. These actions, however, are not sufficient in providing the leaps of improvements sighted for the next generation of telecommunication services. At the high level there are two methodologies to solving current problems: the first is to move functionality to the cloud, create the C-RAN as discussed in section 3.3; The second is to move functionality to the edge of the network, termed Mobile Edge Computing, MEC, as discussed in section 3.4.

In this thesis we present a redesign of the 4G/LTE network architecture, coreless mobile networks, that we believe will enable the progressive enhancement on the 5G prospects mentioned earlier. The concept can be seen as an evolution of MEC, featuring among others, a fully virtualised EPC and an interconnecting data layer for state management. We present an analysis of the properties required of the data store and, in particular, the evaluation of a data storage system, the Apache Geode NoQSL data store, as an example of a data storage system that could be leveraged in the realisation of our proposed architecture.

This thesis is structured as follows: section 2 discusses the state of the art in the current 4G/LTE network and section 3 presents current and emerging technologies that enable the progression into 5G. Beginning from section 4 we present our mobile network architecture and continue to an analysis of existing solutions in the area of data stores in section 5. We present the evaluation of the Apache Geode system in section 6 which also details the results gathered in benchmarking the system. Finally, we discuss some aspects of our work in section 7 and conclude the thesis in section 8.

2 State of the art

We start off by briefly presenting the current state of the research community as it provides a perspective into the importance of the topic and its emphasis in the world. After, we delve into the current 4G system and see the functions of the telecommunications network at a high level.

2.1 A global research community

5G research is growing fast. This is indicated, for example, by the *4G Americas - Global Organisations Forge New Frontier for 5G* report on 5G activities.[5] We present a summary of come of the projects and research avenues:

China National Key Project on 5G: This is a government funded research activity, where the 2016–2017 key project includes projects in standardisation, AD/DA chipset prototyping, power amplification, bandwidth filtering, low frequency networks, low power & massive connect, high frequency systems, high density networks, new multiple access coding, and more.[6]

5G Forum is a Korean industry-academia coordinated program including establishing national policy, promoting R&D, fostering ecosystem support and enabling global collaboration.[7]

The association of radio industries and businesses (ARIB) in Japan was established in 2013 with the purpose of studying various aspects of mobile communication beyond 2020. Another significant effort in Japan is the Fifth Generation Mobile Communications Promotion Forum (5GMF) that conducts research and development on the subject.[8]

European Union Framework Project 8: (FP8) included the forming of the 5G Public Private Partnership (5G-PPP) the aim of which is to enable 5G collaboration in Europe. The 5G-PPP project is a large ongoing collaboration with academia and industry co-funded projects. White papers are being continually produced in all avenues of 5G.[9]

3rd Generation Partnership Project: (3GPP) continues its work toward new communication systems. Their work for 5G started in 2015 with the stage 1 as the service requirements, which will become 3GPP Release 14. Other major building blocks include studying service

[5] `http://www.4gamericas.org/files/8914/6774/6748/Global_Organizations_Forge_New_Frontier_of_5G_Final.pdf` *(Referenced on 28.10.2016)*

[6] `http://www.chinaeu.eu/wp-content/uploads/2016/04/mazhigang-5G-Progress-and-Cooperation-in-China-20160408.pdf` *(Referenced on 28.10.2016)*

[7] `http://www.5gforum.org` *(Referenced on 28.10.2016)*

[8] `http://5gmf.jp/en/` *(Referenced on 28.10.2016)*

[9] `https://5g-ppp.eu/white-papers/` *(Referenced on 28.10.2016)*

requirements for massive IoT, critical communications, enhanced mobile broadband and network operation. A major research area within 3GPP is the System Architecture (Working Group) which is scoped to resolve the system architecture for next generation mobile networks (including clean-slate or evolution), the support for new Radio Access Technologies (RATs) & multiple access technologies and the scenarios of migration.

The research landscape contains, in addition to those mentioned above, a large body of parties including 5G Americas, Alliance for Telecommunications Industry Solutions, GSM Association, IEEE 802.11, International Telecommunication Union (ITU-R), Next Generation Mobile Networks, Telecommunications Industry Association, Federal Communications Commission Technical Advisory Council, and numerous telecommunications operators and equipment vendors from around the world [39].[5] They are all involved in various aspects of research to bring the 5G to fruition.

2.2 System Architecture Evolution

Before we can introduce our proposed concept of coreless mobile networks we must first understand the current mobile telecommunications architecture. In this section we cover the relevant components and functions of the System Architecture Evolution (SAE), Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and finally the Evolved Mobility Management (EMM) scenario. This section is mostly based on [51].

2.2.1 Overview

System Architecture Evolution (SAE), also known as the Evolved Packet System (EPS), is the next evolutionary step of the General Packet Radio Service (GPRS) core network. Compared to the GPRS system the SAE is much simplified, has a flat all-IP protocol stack, supports higher throughput Radio Access Networks (RANs) and supports mobility to and from legacy systems in addition to supporting other non-3GPP wireless access technologies such as WiFi.

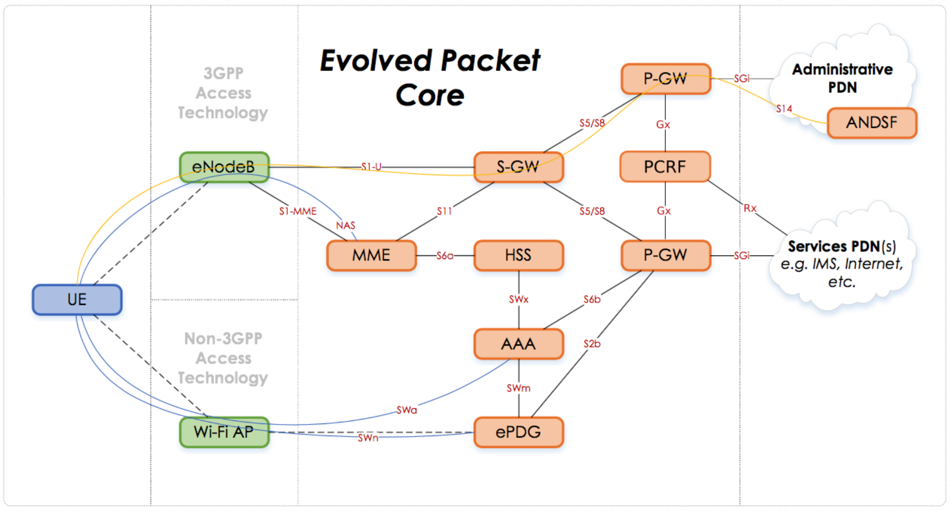
GPRS was an integrated part of the GSM network switching subsystem, which was mainly circuit switched. Perhaps the most notable evolution to the previous technology stack is the SAE all-IP architecture; in SAE all protocols and interfaces are packet switched. The aim of the SAE is to enable seamless IP connectivity to Packet Data Networks (PNDs) even when the UE is mobile. This is achieved with bearers that the the E-UTRAN, the access network, and the Evolved Packet Core (EPC), the core network, set up in coordination.

Bearers are associated with Quality of Service (QoS) parameters according to which the EPC and E-UTRAN prioritise and throttle the packet flows. As the user may have multiple bearers active, for example one for continuous FTP transfer and one for a Voice-over-IP (VoIP) session, the limited bandwidth available must be used wisely. The set up and management of bearers and other connectivity parameters in the network is achieved by standardised interfaces and protocols across the network elements. This allows network operators to choose their equipment from several vendors without the fear of vendor lock-in [51].

The system also provides security and privacy in order to protect the user and the network from attacks and fraudulent use. Various qualities of service, security and distinguishing one UE from another are achieved by a complex set of variables stored across the network elements. Together the variables are referred to as the UE *context* state. Along side the context the network operator stores data about the user itself and other static data, such as the International Mobile Subscriber Identifier (IMSI) and base encryption keys [13, 14, 15].

Figure 1 depicts the EPS network elements at a high level. The next sections cover the functional entities of the EPS in more detail. After establishing an understanding of the network, section 2.3 continues with the description of the events seen in the system.

Figure 1: The Evolved Packet System:[10] User Equipment in blue, the E-UTRAN in green and the Evolved Packet Core in orange.



[10]Figure adapted from `https://en.wikipedia.org/wiki/System_Architecture_Evolution`, licence CC BY-SA 4.0, *(Referenced on 28.10.2016)*

2.2.2 The edge: eNodeB

The Evolved Node B (eNB), commonly also called the base station, is the one and only network element in the E-UTRAN access network. The E-UTRAN itself is made up of a network of interconnected eNBs which are connected through the *X2* interface. The eNB connects the UE via radio link to the EPC, through the *S1* interface. Contrasting to the previous architecture there is no centralised controller coordinating normal user traffic making the architecture flat. The UE and eNB communicate via Access Stratum (AS) protocols, the details of which are out of the scope of this thesis but are detailed in [51].

Each eNB is responsible for managing a number of cells and the connectivity of the UE within the geographical area covered. Some of the core responsibilities of the eNB include [3]:

Radio resource control (RRC) which consist of radio bearers, admission control, radio mobility control, scheduling and dynamic allocation of resources.

Header compression makes the use of the radio resources more efficient by compressing packet headers which can bring significant overhead.

Security in the AS is achieved by encryption of all data sent over the radio link.

2.2.3 The core: evolved packet core

The EPC is *"responsible for the overall control of the UE and establishment of bearers"* [3]. This is achieved by the coordination of multiple entities within the core network. Each entity has been tasked with a particular area of management responsibilities. The main entities are the Mobility Management Entity (MME), the Packet Data Network (PDN) Gateway (P-GW) and the Serving Gateway (S-GW). In addition there are several other important entities that complement the functionality to the network. The following details the key entities of the network [51].

Mobility Management Entity: The MME is a control plane element and no actual user traffic flows through it. It is connected to the eNB via the S1-MME interface. The responsibilities of the EMM include the coordination of bearer establishment, management and teardown, the idle mobility of the UE including paging, tagging and retransmissions, and the choosing of a Serving gateway for the UE at initial attach and handover. Furthermore, the MME authenticates the attaching UE in coordination with the Home Subscriber Server.

Serving Gateway: The S-GW is much like a switch in a regular IP network, and its main responsibility is the forwarding of user traffic to and from the eNB and PDN gateway(s). It also buffers incoming data should the UE be in the idle state and in coordination with the MME wakes up the UE for it to receive the data. Another important responsibility of the S-GW is that it functions as a mobility anchor for the UE when it changes between eNBs.

PDN Gateway: The P-GW is the gateway to the operators' or external IP networks, the networks with services the user may wish to access. It is responsible for the IP address allocation for the UE for each connection to a different PDN and subsequently handles the mapping of the EPS bearers to and from IP addresses. Other related functions are the filtering of packets, charging rule enforcement and quality of service provisioning.

Home Subscriber Server: As the name suggests, the HSS stores information relating to the subscribed users. The stored information about the user may vary by operator, but at the high level the profile contains QoS parameters that relate to, for example, how fast the users Internet connection should be and how they are charged, available PDN connections, and encryption master keys. Additionally the HSS tracks which MME the UE is associated with.

Policy and Charging Rules Function: The responsibilities of the PCRF include making decisions about the policies and controlling the charging functions to be applied to the users data flow. The QoS decisions that the P-GW enforces originate in the PCRF.

Access Network Discovery and Selection Function: The role of the ANDSF is to provide the UE with information about nearby access networks, including 3GPP and non-3GPP standardised radio links such as WiFi.[11] The information can be used to guide the UE to switch to access networks with better reception or higher bandwidth.

2.3 The Evolved Mobility Management scenario

In this section we cover the basic Evolved Mobility Management scenario in the 4G/LTE system and detail the seven bare-bone cases around which the environment rotates. We describe the cases with special focus on the UE *context* state as it plays a central role in our concept of coreless mobile networks. The information in this section is mostly based on the documentation by Netmanias/NMC Consulting Group.[12] In particular we focus on three questions:

- What exactly does happen in the LTE network?

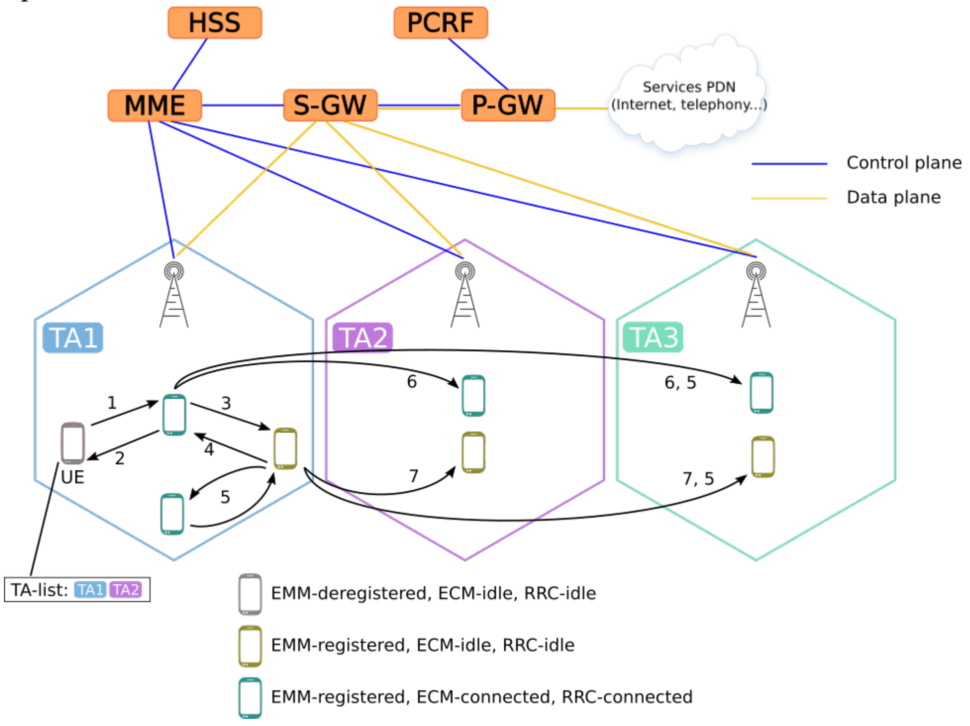
[11]`https://en.wikipedia.org/wiki/System_Architecture_Evolution` *(Referenced on 28.10.2016) (Referenced on 28.10.2016)*

[12]Eleven EMM Cases in an EMM Scenario: `http://www.netmanias.com/en/?m=view&id=techdocs&no=6002` *(Referenced on 28.10.2016)*

- How is the UE connected, how does it receive data and how is 'seamless connectivity' achieved?
- What is the UE context state and how does it change?

The scenario is depicted in figure 2, in it each of the seven barebone cases is indicated with and arrow from one *connection* state to another. The scenario assumes that a user has brought a cell phone and a subscription to a Public Land Mobile Network, PLMN, of a mobile network operator. The operator has installed the user subscription information into its HSS and PCRF components within its core network. In the descriptions of the EMM procedures the following notation is used: the UE connection state "EMM-deregistered, ECM-idle, RRC-idle" is referred to as the *detached* state; the UE connection state "EMM-registered, ECM-idle, RRC-idle" is referred to as the *idle* state; the UE connection state "EMM-registered, ECM-connected, RRC-connected" is referred to as the *active* state.

Figure 2: The EMM scenario:[12] 1: initial attach, 2: detach, 3: S1 release, 4: service request, 5: tracking area update, 6: handover, 7: cell reselection, 6,5: handover with tracking are update, 7,5: cell reselection with tracking area update.


HSS
PCRF
MME
S-GW
P-GW
Services PDN (Internet, telephony...)
Control plane
Data plane
TA1
TA2
TA3
1
2
3
4
5
6
7
6, 5
7, 5
UE
TA-list: TA1 TA2
EMM-deregistered, ECM-idle, RRC-idle
EMM-registered, ECM-idle, RRC-idle
EMM-registered, ECM-connected, RRC-connected

The UE *context* state can be categorised in the following categories: identifiers, location related, security, EPS bearers and QoS parameters. The identifiers pertain to uniquely identifying the UE at the global, the operator network, tracking area and cell levels. Each level has its own designated identifiers of which some have overlapping subcomponents. The UE location must also be tracked in order to deliver data to and from it. For this purpose there are several identifiers, which pertain to keeping the last known position of the UE known at the cell, tracking area and network levels. Communication within the EPS is encrypted and integrity protected at the control plane (NAS), and integrity protected at the user plane (AS). For this purpose the planes contain their own security context, the NAS- and AS-security contexts which include integrity and encryption keys and algorithms. EPS bearers are pipes through which traffic flowing to and from the UE is routed. Several bearers are needed in the system: at the radio interface the DRB, at the S1 interface the S1 TEIDs and at the S5 interface the S5 TEIDs. Together the DRB and S1 bearers are called the E-RAB. Each subscription has an accompanying charging policy and associated quality of service, these are contained in the last QoS categorisation. They are used in establishing charging rules, throttles, bearers and their management in order to deliver the quality of service the user subscribed to.

2.3.1 Initial attach

In the very beginning the user has not yet used the cell phone, henceforth as UE, and turns its power on. The UE is initially in the detached state and proceeds to do a cell search and synchronises with a base station. The UE will then attempt to attach to the PLMN by the *initial attach* procedure. During the initial attach several operations take place: the UE and the MME authenticate each other, the MME sets up default bearers and signalling connections according to the user profile stored in the HSS, the UE is assigned an IPs to use in connecting to the PDNs, and the UE is assigned the tracking area identifier (TAI-) list. After a successful attach procedure the UE enters the active state and user traffic will flow to and from the registered PDNs. Should the attach procedure fail the MME will notify the UE and it will remain in the detached state.[13]

The Initial attach procedure begins with the UE sending the MME an attach request message which includes an identifier. The identifier may be the IMSI, International Mobile Subscriber Identity, which will be used in the first attach to an MME that has not seen the UE before. The identifier may also be an old GUTI, the Globally Unique Temporary Identifier, that an MME has assigned to the UE at a previous attach. Depending on the type of attach, whether it be an attach to a new MME or new LTE network, or an old MME and so forth, the MME may go through all or a subset of subsequent operations:

[13]`http://www.netmanias.com/en/?m=view&id=techdocs&no=10441&page=2` *(Referenced on 28.10.2016)*

UE ID acquisition: Depending on whether the UE has been attached to the network and to which MME, the MME may request the UE to identify itself with the IMSI should the integrity check of the attach request message check fail.

Authentication: If the integrity check fails the MME will initiate the EPS-AKA authentication procedure. During the procedure the HSS generates the K_{ASME} which is sent to the MME which then performs the rest of the authentication procedure with the UE in behalf of the HSS.

NAS security setup: The MME and UE generate NAS security keys from an intermediate key derived from the master LTE key by the HSS. The master key never leaves the HSS nor the UE, but is used to derive intermediate integrity and encryption keys.

Location update: The MME notifies the HSS that the user subscribed to the network and sends the UE the TAI-list. The MME also downloads the user subscription profile, which includes, among others, registered PNDs and QoS parameters.

EPS session establishment: The MME then proceeds to set up, in coordination with other network elements, the required bearers for the user traffic. Default bearers include, but may be supplemented by others, the S1 and S5 download and upload bearers. During bearer setup the PCRF sets up the QoS policies into the P-GW which enforces them.

In summary, the entire context for successful and secure UE connectivity is set up during the initial attach procedure. Table 1 summarises the UE context before and after the procedure.

Table 1: UE context before and after initial attach. Added context in blue. See also the section on abbreviations.

Table 1: initial attach context.

	Before	**After**
UE identifiers	IMSI	IMSI GUTI IP C-RNTI eNB S1AP UE ID MME S1AP UE ID
UE location	-	ECGI TAI TAI-list MME ID
Security context	K_{master}	K_{master} NAS security AS security

Table 1 continued		
	Before	**After**
EPS bearers	APN	APN APN in use EPS bearer ID DRB ID E-RAB ID S1 TEID (UL/DL) S5 TEID (UL/DL)
QoS parameters	Subscriber profile Access profile	Subscriber profile Access profile QCI ARP UE-AMBR (UL/DL) APN-AMBR (UL/DL) TFT (UL)

2.3.2 Detach

The purpose of the detach procedure is to release the resources tied to the upkeep of UE connectivity. The detach procedure may be triggered by the network or the UE, but in either case the detach will be explicit or implicit. The key difference between the two cases is that in an explicit operation the network or the UE triggers the detach procedure due to, for example, a switch-off at the UE or a subscription cancellation event at the HSS. Whereas the implicit detach always involves the UE losing connectivity with the network because of signal loss. When the MME is no longer able to complete a tracking area update due to connectivity loss it starts the mobile reachable timer, the expiry of which triggers the implicit detach procedure. The end result of any detach procedure is that the UE will enter the detached state and all active resources are freed.[14] In most cases the network will retain some information from the last session to hasten the subsequent attach procedure. Table 2 summarises the changed UE context.

Table 2: UE context before and after detach. Removed context in red.

Table 2: detach context.		
	Before	**After**
UE identifiers	IMSI GUTI IP C-RNTI eNB S1AP UE ID MME S1AP UE ID	IMSI GUTI
UE location	TAI TAI-list MME ID ECGI	TAI (last visited) TAI-list MME ID

[14] `http://www.netmanias.com/en/?m=view&id=techdocs&no=6108` *(Referenced on 28.10.2016)*

Table 2 continued		
	Before	**After**
Security context	K_{master} NAS security AS security	K_{master} NAS security
EPS bearers	APN APN in use EPS bearer ID DRB ID E-RAB ID S1 TEID (UL/DL) S5 TEID (UL/DL)	APN
QoS parameters	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL)

2.3.3 S1 release

The case begins as the UE is in the active state and traffic is flowing to and from it. At some point, however, the user will stop using the UE and it will become inactive. After some time of inactivity the UE or the MME will trigger the release of S1 bearers and signalling connection in the S1 release procedure and the UE enters the idle state. In summary the S1 release may be triggered by user inactivity, RRC signalling integrity failure, UE initiated release, authentication failure or the UE associating with a disallowed closed subscriber group cell. After the release of S1 signalling connection and DRB the UE enters the idle state and all context is deleted at the eNB, freeing the resources. The purpose of the procedure is to release resources not needed in the idle state in order for the UE to conserve energy and the EPS, particularly the eNB, to conserve energy and resources.[15] Table 3 summarises the changed UE context.

Table 3: UE context before and after S1 release. Removed context in red.

Table 3: S1 release context.		
	Before	**After**
UE identifiers	IMSI GUTI IP C-RNTI eNB S1AP UE ID MME S1AP UE ID	IMSI GUTI IP
UE location	TAI TAI-list MME ID ECGI	TAI (last visited) TAI-list MME ID

[15]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6110` *(Referenced on 28.10.2016)*

Table 3 continued		
	Before	**After**
Security context	K_{master} NAS security AS security	K_{master} NAS security
EPS bearers	APN APN in use EPS bearer ID DRB ID E-RAB ID S1 TEID (UL/DL) S5 TEID (UL/DL)	APN APN in use EPS bearer ID E-RAB ID S1 TEID (UL) S5 TEID (UL/DL)
QoS parameters	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)

2.3.4 Service request

The UE has been in the idle state for a while. Now the user or a background application generates new traffic which must be delivered to the PDN. It is also possible that the UE receives new traffic originating in the PDN or other related service. The UE must be transferred from the idle state into the active state in order for it to receive the data, and so the service request procedure takes place. During the procedure the resources released in the S1 release procedure are reserved again and the S1 signalling connection and DRB are formed. The procedure involves three operations, the first is the setup of the RRC connection to allocate radio resources at the eNB, the second is an optional authentication step which is necessary if the NAS messages are not integrity protected or the check fails, and the third and final step is the establishment of the E-RAB which includes the S1 signalling connection and DRB setup. Should the new traffic originate in the PDN there is an additional step involved before any of the previously mentioned. That is, the MME must page the UE in order to notify it of the incoming data so the UE may wake up and start the service request procedure.[16] Table 4 summarises the changed UE context.

Table 4: UE context before and after service request. Added context in blue.

Table 4: service request context.		
	Before	**After**
UE identifiers	IMSI GUTI IP	IMSI GUTI IP C-RNTI eNB S1AP UE ID MME S1AP UE ID

[16]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6134` *(Referenced on 28.10.2016)*

Table 4 continued		
	Before	**After**
UE location	TAI TAI-list MME ID	TAI (last visited) TAI-list MME ID ECGI
Security context	K_{master} NAS security	K_{master} NAS security AS security
EPS bearers	APN APN in use EPS bearer ID E-RAB ID S1 TEID (UL) S5 TEID (UL/DL)	APN APN in use EPS bearer ID E-RAB ID S1 TEID (UL/DL) S5 TEID (UL/DL) DRB ID
QoS parameters	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)

2.3.5 Tracking area update

The EPS allows the UE to enter the idle state to conserve power and resources on both sides. This has the effect, that the location of the UE can no longer be tracked to the precision of a cell, like in the active state, because the UE, as it is only listening, does not have an active connection with a base station and eNB. However, even in the idle state the UE might receive traffic, such as a message, that needs to be delivered to it. For this purpose a mechanism was built into the EPS to allow the UE to be woken up: the paging procedure.

The paging procedure itself is quite simple: the MME sends a paging message to each cell that the UE should be in, tracked by the TAI-list, the eNBs and finally base stations send the message to the UE via the air interface that the UE is listening to. This is a broadcast message and all UE within the same geographical area will be listening to the broadcast. Should it contain the identifier of the particular UE, the UE will wake up and continue with the service request procedure.

The problem comes when decisions need to be made of the cells that will actually broadcast the paging message. This is the purpose of the concept of Tracking Areas, TAs. A TA may contain multiple cell sites and multiple eNBs, essentially covering a certain geographical area in which a UE may reside. Once the UE attaches to the network, the MME assigns the UE a set of TAs that it may enter while in the idle state without any additional procedures. This set is called the Tracking Area Identifier (TAI)-list, as in the figure 2. The TAI-list defines the cells through which the MME will send its paging messages should it have any, to the UE. It is a non-trivial optimisation problem that has had operators and equipment vendors alike searching for answers for a long time, how to most efficiently assign TAI-lists and the associated TAU-timer and paging procedures to minimise core network load and maximise battery life on the UE.

The procedure according to which the TAI-list of the UE, stored also at the MME, is updated is called the Tracking Area Update, TAU-procedure. There are two cases in which a TAU will be triggered: once the TAU-timer elapses notifying the UE and the MME that enough time has elapsed since the last TAU, and if the UE enters a TA that is not contained in the TAI-list assigned to it. In the simple case the TAU involves the UE to transition from the idle state to the active state with the exception that the DRB and S1 bearer are not set up. The only connectivity that is needed for the TAU is the RRC and S1 signalling connection to enable control message flow between the UE and the MME. The MME then assigns the UE with a new TAI-list and GUTI if necessary. Finally the UE acknowledges the update and proceeds to release the signalling connection and returns to the idle state.[17] The UE context remains largely unmodified, with the exception of the GUTI and TAI-list if the MME decides they need to be updated.

2.3.6 Handover

A handover is a mobility management procedure that ensures seamless connectivity of an active UE to the PDN when the UE moves from a cell in one TA to the next cell in another TA. Should the UE move to a TA that is not in its TAI-list the TAU procedure is initiated after the handover.[18] There are several types of handovers depending on which EPC entities are involved and change during the procedure. All of the cases will not be covered here, as they are outside the scope of this thesis, but the high level structure is explained.

Measurement configuration: The process begins when the UE associates with a base station and eNB which specifies which measurements the UE is to collect. This is transmitted in an *RRC connection reconfiguration* message that includes the specification of the trigger event, report interval, neighbour list to measure and so forth.

Measurement report: The UE takes measurements according to the specification. Upon a trigger event, which may be a weak signal, the UE sends a measurement report to the eNB. The eNB decides whether to initiate a handover.

Handover decision: The eNB, upon receipt of a measurement report, decides to initiate a handover to a target cell included in the report.

Handover preparation: Depending on the type of handover different entities are involved. In the case of an X2 handover, the source and target eNB communicate directly via the connecting X2 interface and they prepare the forwarding tunnel. In the case of an S1 handover the source eNB and the target eNB communicate and forward traffic via an indirect tunnel via the S-GW, the forwarding tunnel preparation is coordinated by the MME.

Handover execution: In the X2 handover the source eNB sends the UE context to the target eNB which allocates the resources to connect to the UE. The X2 enables a direct tunnel via which incoming traffic is forwarded from the source eNB to the target eNB onwards to the UE until the handover is complete. In the case of an S1 handover the indirect tunnel from the source eNB to the target eNB via the S-GW forwards the traffic thusly: the S-GW normally

[17] `http://www.netmanias.com/en/?m=view&id=techdocs&no=6193` *(Referenced on 28.10.2016)*

[18] `http://www.netmanias.com/en/?m=view&id=techdocs&no=6324` *(Referenced on 28.10.2016)*

forwards the traffic to the source eNB, which forwards it via the indirect tunnel back to the S-GW, which further forwards the traffic to the target eNB.

Handover completion: Finally the UE completes its connection to the target eNB. After, the forwarding tunnels are stripped down and traffic will flow normally again.

Tracking area update: If the UE moves to a cell which is in a TA that is not included in its TAI-list the TAU procedure is initiated immediately after the handover.

For further information and details the reader is encouraged to refer to the Netmanias documentation at [19], [20] and [21]. During handover there may be one or several extra bearers set up for the forwarding tunnels, but after completion the changes in context reflect the changed network entities. The context that changes during the X2 handover is highlighted in table 5.

Table 5: UE context before and after X2 handover. Changed context in green.

Table 5: handover context.

	Before	**After**
UE identifiers	IMSI GUTI IP C-RNTI eNB S1AP UE ID MME S1AP UE ID	IMSI GUTI IP C-RNTI eNB S1AP UE ID MME S1AP UE ID
UE location	TAI TAI-list MME ID ECGI	TAI (last visited) TAI-list MME ID ECGI
Security context	K_{master} NAS security AS security	K_{master} NAS security AS security
EPS bearers	APN APN in use EPS bearer ID DRB ID E-RAB ID S1 TEID (UL/DL) S5 TEID (UL/DL)	APN APN in use EPS bearer ID DRB ID E-RAB ID S1 TEID (UL/DL S5 TEID (UL/DL)

[19]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6224` *(Referenced on 28.10.2016)*

[20]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6257` *(Referenced on 28.10.2016)*

[21]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6286` *(Referenced on 28.10.2016)*

Table 5 continued		
	Before	**After**
QoS parameters	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)	Subscriber profile Access profile UE-AMBR (UL/DL) APN-AMBR (UL/DL) QCI ARP TFT (UL)

2.3.7 Cell reselection

The cell reselection is an idle-state procedure that ensures that the UE is able to listen to paging messages and subsequently wake up from idle. In the idle state the UE wakes up at end of each discontinuous reception cycle to measure the signal of the serving cell. When the UE moves with the user, the signal may become attenuated due to obstruction or distance. Once the signal drops below a certain threshold the UE starts listening to the broadcast of system information from nearby cells. The system information is used by the UE when it decides to change the cell it camps on.[22] In particular the UE checks the cell identifier which indicates the TA of the cell. Should the UE decide to camp on a cell that is not in its TAI-list the UE initiates the TAU procedure.[23] With the exception of the additional TAU procedure no context is changed.

2.4 Summary

We have discussed the EMM scenario in some detail. All in all, the management of the state of UEs, their context, is achieved with complex procedures involving large amounts of signalling. It is estimated that in regular conditions a UE may generate upwards of 500 signalling messages per hour in the EPC and "always-on" UEs will further increase the signalling load [4, 56]. While network capacity may not be a bottleneck, research has shown that disruptions in signalling and sporadic short bearer changes can amount to much decreased performance and throughput [50, 26]. The protocols are often interdependent, rendering their interactions complex. This increase in complexity has been shown to be problematic, incurring penalties in system operation [53].

Standardisation provides a path for vendors to create similar products and compete in their efficiency, it also removes some problems with vendor lock-in at the operator. It is, however, also a fundamental source of inelasticity in the architecture. While we are already on a path to software based architectures, there are still tremendous amounts of legacy hardware in use. Fixed routing and provisioning causes major load imbalances and makes it difficult for operators to obtain and provision resources to combat signalling storms and flash crowds [46]. It is therefore necessary to redesign control plane protocols and redesign the standardisation to allow for more software architectures which enable faster evolution and operator specialisation.

[22]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6322` *(Referenced on 28.10.2016)*

[23]`http://www.netmanias.com/en/?m=view&id=techdocs&no=6324` *(Referenced on 28.10.2016)*

3 Overview of important research topics

In this section we cover some of the key technologies that are proposed to enable the 5G prospects. Currently the technology trend in almost all fields is to move all possible components to the cloud. This is reflected also in 5G research, where the concepts of Cloud Radio Access Networks (C-RAN) and Heterogeneous C-RAN have received much attention. Another potentially game changing technology is massive multiple input multiple output, which aims to increase base station antenna count to the hundreds and UE antenna count into the tens.

3.1 Software defined networking

Since its introduction by Greenberg et al., software defined networking [19], SDN, has attracted mounds of interest in the research community. Subsequently the technology has advanced in leaps and bounds as is indicated also by the number of vendors currently supporting a popular SDN interface, OpenFlow.[24]

Traditionally networking infrastructure and paths therein have been configured by tedious manual labour through vendor specific APIs, typically resulting in vendor lock-in [44]. On top of the static network configuration the IP routing mechanism is run in a distributed fashion, whereby each router has a very limited and, by definition, outdated view of the network. In traditional networking the routing plane, where the actions for a packet are decided, and the actual forwarding of the packets in the user plane are intertwined. SDN separates the control and user planes allowing the network to be programmed with high level objectives rather than low level primitives [24].

In essence, SDN centralises the control plane of the network so that routing decisions are no longer made by routers themselves, but by a centralised entity which has a global view of the network. The so-called controller is aware of all of the network elements under its region of control by means of direct communication. Each router is therefore configured and monitored by the controller in real-time, so that each new flow for which a router does not have a rule dictating an action, triggers the router to ask the controller for a rule. The controller subsequently installs a rule to the requesting router and to all the routers that should also be made aware of said rule, typically all in the path of the flow. A rule at the high level, in OpenFlow for example, are of the form MATCH:ACTION, where MATCH is a set of wildcards and specifiers that are matched to the headers of incoming packets of a flow, and the ACTION dictates whether for instance to DROP, FORWARD or MODIFY the packet or its headers. The routers then continue routing the flow according to the newly installed rule [1].

SDN has been sought to enable easier management and operations to new virtualised architectures. There have been several promising papers released on the matter [44, 21, 35, 49]. Sama et al. propose that the OpenFlow protocol requires extensions in the match-fields and action types to enable efficient handling of the GTP-U TEIDs [49]. In his thesis, Mäki presents a prototype OpenFlow controller for handling of GTP-U flows [35]. He et al. propose a new request routing protocol and algorithm for the SDN enabled mobile core [21]. Hypervisors have originally been developed to enable efficient virtualisation of hardware resources. Now, as per [6], hypervisors are being brought into the SDN paradigm as enablers of network hardware virtualisation, monitoring and management.

[24]`https://www.opennetworking.org/sdn-resources/openflow` *(Referenced on 28.10.2016)*

Although promising, SDN is not without its flaws, as is pointed out by [57]. The survey concentrates on SDNs susceptibility to distributed denial of service attacks, DDoS, which have become on of the most extensively used methods of disrupting service availability on the public Internet. The loss of a controller due to a DDoS attack could bring down large sections of the network, amplifying the scope of service unavailability with contrast to traditional networks.

3.2 Virtualised network functions

Virtualisation is not a new topic in computer science. In fact, hardware virtualisation has been one of the driving forces behind cloudification of services during the past decade. Traditionally services have been run in datacentres owned by the company hosting the service, today those services are by and large are hosted within leased virtualised computing resources in third party operated datacentres. Network functions, however, have been lagging behind in virtualisation. While traditionally network functions have been implemented as hardware that needed to be physically plugged in to the network and manually configured into place, virtualised network functions are software that is running on generalised off-the-shelf hardware. Previously a firewall has been a specialised hardware component installed in a network node centre, but with VNF it can be dispensed and the software moved to be hosted in the cloud. The flows that need to be routed into and out of the network function can be routed with software defined routes that are nimble, elastic, rather than hardware routes in cables [24].

Virtualised network functions have attracted research with regards to the 5G aspects as well. Specifically of interest have been hypervisors [6], resource management [58] and information-centric networking [33]. These are all important research questions in light of the scoped 5G improvements; the VNFs must be cost and energy efficient, elastically scalable and versatile. As network functions are virtualised it becomes possible create new services and to chain them in ways not possible before. For instance, the next generation of mobile networks may employ completely different modes of operating the control plane and it can be seamlessly coupled with legacy protocols. By virtualising components of the current and next generation telecommunications networks it is expected that much savings in capital and operational expenditure can be achieved. How will VNFs cope with the demands of 5G? It is known that specialised hardware has traditionally been faster with higher throughput than software. Response time will become a critically important factor to be considered in the adaptation of VFNs [2].

3.3 Cloud radio access networks

Recent developments in virtualisation have spawned new ideas in telecommunications as well as Internet services. One of the most hailed concepts in 5G research has been the cloudification of the RAN. The Cloud Radio Access Network, C-RAN, centralises processing of the RAN into datacentres, where resources an be efficiently managed [28]. In essence all that is left at the edge of the mobile network are the Remote Radio Heads, RRH, which contain the antennas used for the actual radio link. While the datacentre provides an excellent platform for virtualisation, including flexible resource management and on-site hardware maintenance, there are some serious caveats in RAN centralisation [47].

Important advantages in centralisation of the RAN include, in addition to efficient resource management at the datacentre, enabling of efficient coordination of radio resource usage resulting in higher spectral efficiency including supporting massive MIMO [36], and further virtualisation of services into a conceptualised Radio Access Networks as a Service, RANaaS [48]. Centralisation also enables straightforward combination of small, medium and large cell networks and non-3GPP standardised wireless communication, heterogeneous networks, into the RAN [34].

The most problematic aspect of the C-RAN is the fronthaul [29]. The fronthaul concerns the transport of the digital radio signal from the Baseband Unit (BBU) in the datacentre to the RRH which converts it to analogue and finally to the UE via radio interface. The BBU is concerned with the signal processing of the digital signal and has typically been located at the base station for good reason. The resulting traffic from the RRH to the BBU is huge even compared to the backhaul, which comprises of the link between the edge RAN and the core network. The only technology available today to carry the fronthaul traffic is optical fibre which is expensive to deploy. In contrast, millimetre-wave is being sought to be a possible solution in this respect, not requiring expensive hardlines to deploy but with the potential capacity to carry the traffic [43, 2].

3.4 Mobile edge computing

Contrasting to C-RAN, mobile edge computing, MEC, brings the computing resources closer to the user, to the edge of the radio access network. Since the edge of the RAN is as close to the user as possible it can provide services with high bandwidth and low latency. Mobile edge computing has become a viable option as a basis to build cloud platforms with the advent of low cost and high capacity general purpose computing hardware. Deploying computing power at the edge has the disadvantage that its maintenance is more costly than at a datacentre. However, base stations and Evolved Node Bs (eNBs) need to be deployed and maintained in any case [42].

Being closer to the user the mobile edge can offer more acceleration compared to the centralised cloud. Caching of content, for instance, is seen as a major step in reducing core network load, access latency and object fetch time [38]. The effect of caching has been widely studied and is a common practice of content providers, such as Youtube, Vimeo and Facebook [7].[25] Compared to current caching techniques, where content delivery networks, CDNs, are regular servers geographically distributed and users connect to them via the Internet, caching at the edge of the RAN brings the cache servers to within one-hop distance of users [55].

Opening the mobile edge to third parties, allowing them to deploy applications has potentially other advantages as well. In addition to caching and optimised content delivery some highlighted use cases include video analytics, location services, IoT and augmented reality.[26] It has been suggested, that Over-the-Top players and independent software vendors could foster new market value and areas should they be allowed to deploy innovative applications on the mobile edge. Leasing the computing power and network resources could bring new revenue streams to the mobile network operator, alleviating their pressure on operational expenditure [42].

[25] `https://code.facebook.com/posts/1653074404941839/under-the-hood-broadcasting-live-video-to-millions/` *(Referenced 15.11.2016)*

[26] `http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing` *(Referenced on 28.10.2016)*

3.5 Massive multiple input multiple output

Multiple Input Multiple Output (MIMO) refers to the technique of attaching multiple antennas to both the transmission and reception components of a radio circuit. Current technology can make use of only a narrow band of frequencies at each antenna, so multiplying the number of antennas naturally increases the number of usable frequencies. Massive MIMO refers to increasing the amount of antennas to the hundreds. Massive MIMO can thus enable benefits in spectral and energy efficiency, noise and intra-cell interference can be more effectively mitigated and the medium access control layer can be simplified. The need for massive MIMO may be counteracted to some extent by full-duplex communication [54].

3.6 Spectrum reuse and new new bands

Spectrum reuse pertains more specifically to spectrum sharing. It is common today that licensed spectrum is bought and used by one operator at a time in a particular geographical location. At the same time the same spectrum could be free to be used in another geographical location, but policies and technology does not permit doing so. This wastes space on the spectrum and causes unnecessary congestion in those spectra that can be used. Wireless network virtualisation is a broad term that may, in addition to the aforementioned spectrum sharing, include infrastructure virtualisation and air interface virtualisation, for instance. In essence, just as with any virtualisation technique, it decouples hardware used to host and the running of services, allowing leasing and new business models [32, 23].

Nevertheless, we are running out of usable spectra. New frequencies are sought for, and specifically the millimetre (mm)-wave is seen to promise vast amounts of new capacity. Mm-wave bands are much wider and can accommodate over 200 times more spectra in the 30-300 GHz range than the current sub 3 GHz frequencies. The shorter wavelength allows smaller antennas, which in turn opens the path to massive MIMO. The major disadvantage of mm-waves is their short range, requiring deployment of pico-cell base stations with a maximum range of 100–200 metres [52].

3.7 Full-duplex communication

Traditional transceivers have been limited by the inability to transmit and receive on the same radio frequency simultaneously. With the advancements in antenna, digital baseband and interference cancellation technologies we have come closer to building systems that are able to overcome this limitation. Being able to transmit and receive in the same frequency channel at the same time would effectively double spectral efficiency. Full-duplex communication cloud potentially also solve problems with hidden terminals, loss of throughput and large end-to-end delays [52, 22].

3.8 Wireless charging

Increasing the battery life of wireless devices is one of the more challenging aspects in the range of 5G research. While computing chips and devices are becoming more powerful they need more energy. With recent advancements in system-on-chips, SOCs, and new processors the requirements for energy have been mitigated to some extent. Should small IoT devices continue the trend of current smartphones' battery life, their era might not be too long. Harvesting energy from the ambient light, heat and wind are popular research directions, but as their occurrence is varied they might not be able to provide energy in a stable enough fashion. Radio frequency energy harvesting has thus gained attention as the output power can be modulated and controlled. Transferring energy via RF signals is possible across long distances and could allow transmission of information at the same time [22].

3.9 Self organising and cognitive networks

Self organising and cognitive radio networks (SON, CR) are closely related to spectrum efficiency gains through sharing and virtualisation of resources. Contrasting to the C-RAN, where the optimisation of resource usage is achieved by centralised components, cognitive radio access networks achieve it by having the base stations actively monitoring spectra and adjusting to changes [54].

Self-organisation may be seen also as interconnected base stations cooperating to provide sufficient radio resources to a geographical area by activating of deactivating parts of the RAN. This will become possible particularly in areas with heterogeneous coverage, where small-cell and macro-cell coverage overlaps. During the night time, for example, when network usage may be significantly less than that of daytime high peaks, the small-cell areas may be deactivated, saving energy, while the areas are still being covered by a macro-cell base station [5]. For further reading on the topic, the reader is encouraged to turn to the excellent book on the topic by Hämäläinen et al.: [20].

3.10 Multipath Transmission Control Protocol

Current service acquisition model in the Internet and networks almost invariably rely on the Transmission Control Protocol, TCP. Most all implementations use a single path, or more precisely, a single network interface for communication. This is somewhat of a historical remnant, as the TCP stack was developed so that the data stream is tightly bound to the destination and source IP addresses. Today devices are ever more capable of connecting to the Internet via multiple access technologies, such as wired ethernet, WiFi, Bluetooth and 4G/LTE. Even as more access technologies have been taken into use the industry has not been keen on developing the techniques to efficiently use all of them, but most devices are limited to using a single network interface at a time [16].

Multipath TCP (MPTCP) enables a device to use multiple network paths, interfaces, for a connection to a service. It was standardised by IETF already in 2013. The more popular desktop operating systems implement MPTCP support in their TCP stack: Mac OS X, Free BSD, iOS 7 and Linux. However, its spread into use by services has been slow. For example, authors Mehani et al. studied the adaptation of MPTCP in the Internet [37]. By scanning the *Alexa Top 1M* list they found that only about 0.1% of IPs and domains were served by MPTCP capable hosts. Deployment of MPTCP is naturally up to service providers, but mobile network operators and operating system manufacturers need to support MPTCP in order to advance technology and improve the quality of service of wireless networks.

3.11 Summary

Future 5G networks will likely not see as big technological leaps in single areas as previous generations have. More so, the technology will likely consist of multiple smaller refinements and introductions of new concepts, which together will bring 5G to fruition. That being said, the technologies that are sought for are still in their infantile stages and more research is needed in order for them to mature and become deployable. This is especially the case with new protocols that are subject to standardisation.

4 Coreless mobile networks

The current 4G/LTE network is slow and as such it cannot be leveraged to achieve the planned advancements in 5G [29]. As mentioned earlier in section 2, the distribution of the control layer across several disparate entities causes a plethora of problems. Enter the concept of coreless mobile networks. The term derives from the idea, that, contrasting to the 4G/LTE architecture, all of the core network functionality is virtualised and moved away from the core network to the edge of the network. In this section we detail the concept, starting with its architecture in section 4.1. We present design challenges and possible solutions and a novel optimisation possibility. This is, as far as we know, the first concept of a completely virtualised and MEC based Evolved Packet Core.

4.1 Architecture of the network

As detailed in section 2.2 the EPS consists of several components, the eNB at the network edge providing the radio connectivity and the core network entities providing controlling functions and connectivity to the PDNs. Let us observe the core network entities as virtualised components, software, that can be deployed as VNFs. We can now place all the core network entities in which ever facility we like. Additionally, let us split the control plane and user plane functionalities in the core network components, not much unlike in [40]. Specifically the S-GW and P-GW contain control plane functions and forwarding functions on the user plane: the S1-U and S5/S8 interfaces carry user plane traffic. Now the core network entities functions are much alike controllers in an SDN environment. They program the forwarding rules for the user plane into SDN capable switches which forward actual user plane traffic.

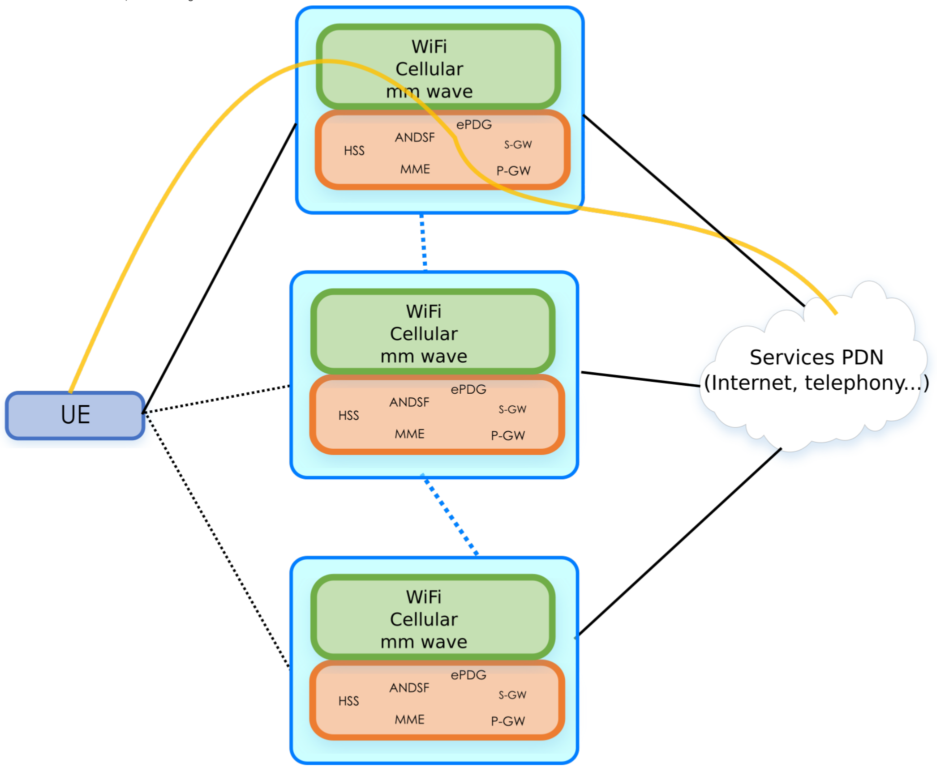
In the concept of the C-RAN, the components are placed and run in datacenters. However, as already discussed, the fronthaul becomes a tedious problem. Another solution is to place and run the virtualised core network components at the same site as the eNB. For example: the site has dedicated hardware for the eNB functionality as before, and additional general purpose hardware for the core network entities and SDN capable switches. The high level architecture is depicted in figure 3. The SDN capable switches forward user plane traffic and are programmed by the core network entities in the local site, smart-box.

Let us also observe that, since each site now has a dedicated set of core network entities, each cell reselection or handover would require much control plane actions in explicitly setting the new entities for the UE. We overcome this by logically unifying all of the core network entities: they read and write a ubiquitous[27] data set that contains the existing UE context which a new set of entities in another site use in building the bearers and identifiers in their process space. Thus the core network entities, that currently hold and manage much state, are made stateless and the context state is moved to be held in the data set. The sites can be connected by the *X2* interface, the operator backbone network, or even the public Internet provided the connection is secure. Note that, contrasting to the 4G architecture in figure 1, our scenario in figure 3 contains multiple radio access technologies in the eNB (in green), hosted alongside the core network entities (in orange) within the same smart-box (in cyan).

The change in architecture has several implications. Firstly the UE has unhindered access to the PDNs, as they are accessible immediately after the smart-box

[27]Ubiquitous to the core network entities, not to the world.

Figure 3: The coreless mobile architecture at a high level. The eNB site, smart-box, in cyan.



rather than via the dedicated EPC, resulting in lower latencies and higher throughput. Secondly, as the EPC entities are close to the UE and each other, even as close as in the same operating system and hence memory space, signalling to the EPC will be extremely fast since slow network connections aren't involved. This further increases throughput and decreases latency as changes in context data and routing are performed on site. Thirdly, as the virtualised EPC entities in each smart-box are now a logically unified across the entire operator network, the need to perform explicit handover procedures diminishes significantly. This has the effect, that a UE will never lose connectivity while it has a good signal to any cell served by the operator in question. In essence the UE can now move seamlessly within the operator network coverage area.

Furthermore, the proposed architecture need not be deployed in one go. The core network entities can still support connections to legacy systems, such as a centralised HSS, making the architecture less disruptive while still capable of supporting novel technologies. It is also noteworthy that the architecture follows along the themes of Mobile Edge Computing, MEC as described in section 3.4 as well as the concept of a *Shared Data Layer* by Nokia.[28] As mentioned, MEC has gained much interest since it can provide computing resources and very fast, low latency connectivity to

[28]Alcatel-Lucent (Nokia): Creating a new data freedom with the Shared Data Layer, at `http://resources.alcatel-lucent.com/asset/200238` *(Referenced on 28.10.2016)*

the UE and the Internet. It is therefore foreseeable that, with the adaptation of MEC, the hindrance of installing additional computing equipment at the eNB site would be alleviated.

It should be evident by now that the architecture, in essence, highlights the importance of the context data. After all, it is according to the UE context state that all user plane data is moved between the UE and the PDNs. While the 4G core network entities are already software and changes to them can be made with relative ease, it is not at all clear how the context data can be made ubiquitous. It is this question we have decided to address in this thesis. In summary, the key concepts in the coreless mobile network architecture are as follows:

Virtualised EPC: Virtualisation of the evolved packet core entities allows flexibility in their placement and enables dynamic provisioning.

Plane separation: Separating the functions of the control and user planes from the EPC components allows the building of an SDN controller with the functionality of the EPC control plane.

SDN switching: The aforementioned "EPC controller" decides and installs the forwarding rules into associated SDN-capable switches.

In-situ computing: The EPC controller, SDN switch and eNB are co-located in the eNB site, and all computing will be done locally. Contrasting to the centralised nature of the traditional EPC, this model is entirely decentralised. Each smart-box can be viewed as an access point into any connected PDN.

Unified components: UE context state is shared via a ubiquitous data store, available to all smart-boxes and their associated EPC controllers which are thus made stateless. From the perspective of the UE, while the eNB may change, the EPC is now a single entity and does not change.

4.2 A shared, distributed data store

The recent decades' developments in deploying and building clouds has brought with itself a new mode of storing data. Once centralised databases with strict ACID semantics were capable of providing the speed, throughput and easily understandable data models. However, their limits have been reached with the advent of the cloud, massive parallelisation and big data. Along have come distributed databases, some of which are more capable than others in providing the necessary *scalability* that is required of them. The field of distributed data stores has, in fact, come far enough to offer solutions in the most demanding of environments [10]. It is therefore natural to look at them as possible hosts for the context data. In this section we identify properties that are imposed on the data storage system by the architecture of coreless mobile networks.

Key-value data model: The UE context is comprised of identifiers, numbers and cryptographic keys. In particular, the UE context always contains at least one identifier which does not change during the lifetime of the subscriber: the IMSI. Another partially static identifier is the GUTI. These can be used as keys and the entire context as the value object. Thus the data model is most easily described in terms of keys and values, typical of noSQL data stores.

Consistency: The data, that the now virtualised core network entities fetch, needs to be consistent with the state of the UE. Specifically the data must be consistent with the most recent update of the neighbouring smart-box in order to achieve a seamless transition from the service of one smart-box to the service of another.

Availability: The data needs to be highly available to the entities. Should data not be available during a transition, it may result in unnecessary detach and attach procedures as the context is rebuilt.

Fast: Transitions in the network happen very fast and often, even more so in the advent of massive IoT. Therefore, the data store needs to support quasi-realtime access and update. It is possible that in-memory data stores are able to provide the required speeds.

Concurrent: Without saying it should be evident, that supporting multiple UE in multiple geographical locations requires that the data store is highly concurrent, allowing access to multiple data items at the same time.

Scalable: In order to be considered the data store should be highly scalable also to support the number of eNB sites in current 4G networks. The key prospects of 5G include that the network support ever more connected devices. Looking ahead of this, operators need room for expansion as well.

Lean: Updates to the context are necessary to keep the UE connected throughout the operator network coverage. However, this should not be achieved 'at any cost'. If the update procedures themselves occupy most of the network resources in the interconnect the purpose of the system is defeated. Updates should constitute only a fraction of the available bandwidth in order for the architecture to be meaningful.

4.3 High availability and consistency zoning

The context data that governs UE connectivity has a special property not seen in many other applications, and it arises especially in the coreless mobile network architecture. The UE is tightly bound to a geographic location, and can be assumed to move at a reasonable speed.[29] The UE connects to a nearby smart-box which serves the UE and makes changes to the context data as the state changes. We can now see that since the UE is tightly bound to a geographic location, so is the context data. In fact, it is for a similar purpose that the concept of tracking areas was conceived.

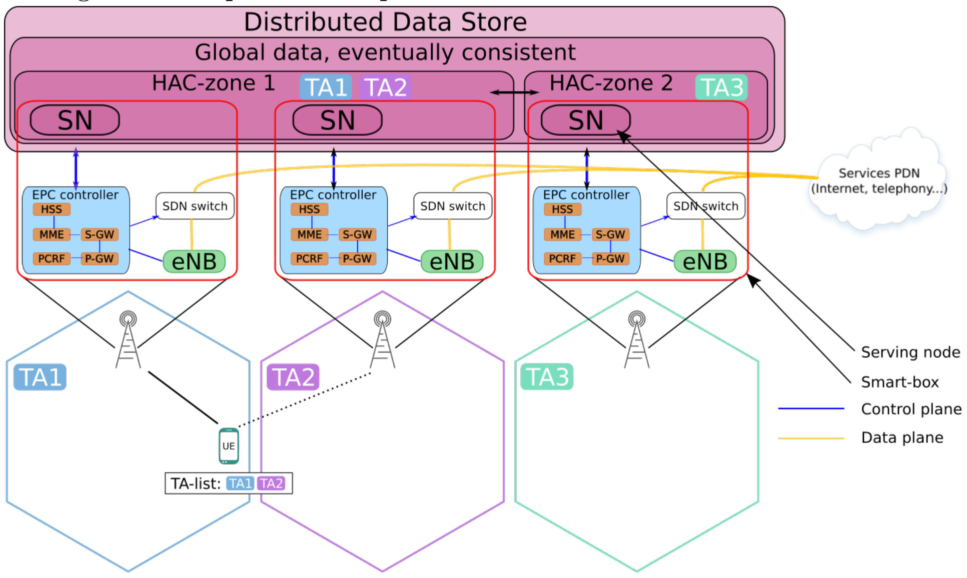
Binding the UE context data to a geographic location presents new possibilities for optimisation. More specifically, it should be possible to relax consistency and availability constraints on the data in locations far away from the UE while still retaining the data fully consistent and highly available near the UE. It is advantageous to any distributed system to contain as little distribution as possible, since the larger a system the more unstable it becomes. Thus, we can form *High availability and consistency zones*, henceforth as HAC-zones, that are geographical areas near a UE within which the UE context data is hosted by a small subset of the global servers and where the data is strongly consistent and highly available.

[29]For instance, we can assume it moves at a speed less than the speed of sound.

Outside of the HAC-zone the data will still be available, but for example, it might not contain the very latest update. However, the data must be consistent eventually so as to enable the operator to maintain coherent information on its subscribers. Information, such as data usage, in the end constitutes the revenue streams of the operator.

Enabling the new HAC-zone optimisation brings about a new requirement of the data store. It must support the configuration, creation and management of data sets with varying degrees of consistency and availability. It must also support restricting the data set to any set or subset of servers that may or may not reside in the same datacentre or geographical region. In order to enable the UE movement between HAC-zones the system must also support mechanisms of transferring data from one set of servers hosting one HAC-zone to another set hosting the next HAC-zone. An abstract view of the system can now be formed, depicted in figure 4.

Figure 4: Simplified example structure for coreless mobile networks.



4.4 Concerning the Evolved Mobility Management scenario

The implications on the EMM scenario and cases described in section 2.3 are mostly beneficial. The most notable difference is that where signalling events between the core network entities used to traverse the operator backbone network, in the coreless mobile network architecture they no longer do so. All signalling events are contained within the smart-box hosting the eNB and core network entities, and only fetches and updates to the distributed data storage will traverse the interconnect. This presents another optimisation possibility: how the updates are propagated. Several options exist, each with its own pro's and con's. A look at the different update mechanisms is provided in section 5.2. The intricacies of deployment are far beyond the scope of this thesis, but in summary, the changes in the EMM scenario are as follows:

Initial attach: The UE context does not exist, only the subscription profile is installed. The EPC controller in a smart-box fetches the profile from the data store and constructs the context normally and installs forwarding rules. The updated UE context is pushed into the data store.

Detach: The EPC controller in a smart-box has the up to date UE context as it is currently the serving site. It completes the detach operation normally and the updated UE context is pushed into the data store.

S1 release: The current smart-box is the serving site. The EPC controller completes the procedure normally and pushes the updated state into the data store.

Service request: Should the current smart-box not be the last active serving site the EPC controller will fetch the UE context from the data store HAC-zone, and rebuild it in its process space. From here the process continues similarly as in the last smart-box: the service request is processed by the EPC controller within the smart-box, forwarding rules are installed and the updated context is pushed into the data store.

Tracking area update: The relevance of TAU is questionable in the new architecture. However, if it should remain it could take the following form: if the current smart-box is not the last serving site the UE context is fetched and rebuilt it in the process space. The TAU procedure is completed normally and the updated context is pushed to the data store along with an update to the HAC-zone should it change.

Handover: Let us consider two cases:

Intra HAC: The UE moves to be served by a smart-box within the same HAC-zone. The new smart-box fetches the context from its HAC-zone, rebuilds it in its process space and installs the new rules within the switches. In principle the context could be pre-built and only the rules need changing.

Inter HAC: The UE moves from the service of a smart-box in a HAC-zone to the service of another smart-box within another HAC-zone. The new smart-box fetches the UE context from either the global data store or contacts the last HAC-zone to acquire the latest context. Network paths to the UE are updated.

Cell reselection: The UE is in the discontinuous reception mode, i.e. idle state, and has no active communication to the network. If the newly selected cell belongs to the same TA in the UE TAI-list no action is taken. Else the normal TAU procedure is triggered.

4.5 Summary

In this section we presented a novel architecture of coreless mobile networks at a high level. It involves the refactoring and virtualisation of the current 4G/LTE core network entities into a component that functions much like an SDN controller. The state of the UE context is moved from the network entities into a data set that is available at each EPC controller. While stateless themselves, the controllers can act upon the context state to provide the normal functionality of the EPC and install forwarding rules into SDN capable switches which forward the user plane traffic.

Though described here, the EPC controller need not be based on the 4G/LTE core network entities, but can incorporate any protocol with some refactoring. The architecture of the coreless mobile network can therefore support any technology for the control plane element, making it backward and forward compatible.

5 Related work in data storage solutions

As discussed in the introduction, the primary scope of this thesis is to discover a data storage solution that can be employed to manage the state in the coreless mobile network architecture, such that its requirements are met. In this section we look at properties of some of the more prominent distributed data storage solutions and evaluate their suitability in terms of our architecture.

5.1 Distributing data storage

When it becomes necessary to move away from the centralised database due to an architectural decision or due to the volume of the data, it becomes necessary to distribute the storage tier. Distribution, however, brings its own problems, namely the problem of dealing with CAP [18]. The CAP-theorem states that in any reasonably conceivable distributed computing system it is impossible to simultaneously achieve Consistency of data among all participants, Availability of data and Partition tolerance. In essence, one may choose two of the three properties while sacrificing one. Distributed data storage solutions are therefore designed with two of the aforementioned properties, and mitigate the effect of the third property.

For instance, a data store may have been designed to be highly available and to tolerate network partitions. Such a database may be ideal in a scenario where Internet users must be able to access some version of a web page. Said data store, however available, will not be fully consistent in its contents: during a network partition event when content is modified by user A the modification may not reach the partitioned set of servers and so user B will receive and out of date version of the web page. This results in an inconsistent view. Should user B proceed to modify the same content, the two updates will conflict. Such a conflict must be resolved in one manner or another, else the behaviour of the system is undefined.

Problems relating to distribution are well known, and there are well known techniques in mitigating the effects in any given configuration of the CAP-choices. Since the cloud is exceedingly often built on commodity off-the-shelf hardware at the scale of thousands and even millions of computers, hardware failure is considered the norm, rather than the exception. Data stores that aim to be useful in this sector of computing will have to have solutions to the distribution problems [31].

Despite that, numerous distributed storage systems exist [9], and in fact the number of potential data stores is far too big to be listed here,[30] thus only a handful are compared. It should be noted, that there are many more distributed storage systems beyond the noSQL category. The scope of data stores was limited to the chosen category as their data model most closely resembles the one desired by the coreless mobile network architecture. The data stores were chosen to provide an overview, a cross section rather than for completeness, of noSQL data stores and form a perspective around the requirements of our architecture.

Apache Cassandra: Originally developed by Facebook, this Apache top-level open source distributed data store has received a tremendous amount of interest in the open source community.[31] Cassandra was developed to be run on hundreds of cheap commodity hardware components and to provide high write throughput. Deploying a data store into hundreds of cheap hardware

[30]For a more comprehensive listing of noSQL data stores, the reader is advised to refer to `http://nosql-databases.org`.

[31]`http://cassandra.apache.org`, *Referenced on 1.11.2016*

means that components will be constantly failing. This has been taken as the norm and Cassandra has been developed with it in mind. It supports replication to achieve high availability and durability, and dynamic partitioning of data over servers in order to scale incrementally. Write operations consist of a write into a local disk commit log and an update to the in-memory data structure. A read operation checks the in-memory data first, then disk lookup by searching newest files first. Cassandra's data model is that of a column store: each item is a map indexed by a key, where the value is highly structured [31].

Memcached: An open source distributed in-memory object caching data store that has been originally intended as a web caching system to alleviate database loads. Memcached has a very simple key-value data model and a flat namespace. It does not offer replication for redundancy and data hosting servers don't synchronise, it is a least recently used caching infrastructure. Memcached takes unused memory in a system as its cache, and by pooling multiple servers' unused memories it can scale extremely well and have very low latencies. Clients accessing the cache query the appropriate server addressed by the hash of a key.[32]

Voldemort: In use at LinkedIn, for instance, this open source data store is a distributed hash table employing a simple key-value data model. It supports replication for redundancy, consistent hashing for partitioning and tunable consistency in either strict quorum or eventual consistency models. Persistence is achieved through a database connection to, for example MySQL. Data can be serialised by multiple mechanisms, including JSON, Protocol Buffers and Java serialisation.[33]

Google Spanner: Developed at Google, Spanner is not an open source data store, however, very interesting in its own right. It has been developed to support scaling into millions of machines across multiple datacentres around the world. Replication and consistency models can be configured by applications: which datacentres contain the data, distance from users, inter-replica distance and their count, and consistency requirements. An important feature is that Spanner provides externally consistent reads throughout the data by the means of a globally meaningful commit timestamp which is achieved by their TrueTime API. Spanner's data model is schematised semi-relational as a multi-version type database value storage: each value is addressed by a key and a timestamp from the TrueTime API. As a semi-relational data model, the Spanner employs a SQL-like query language [10].

Infinispan: Infinispan is, once again, an open source data storage system, which is based as a noSQL -like object store. It supports several modes of operation, including local, invalidation, replicated and distributed cache. Multiple discovery methods enable dynamic provisioning of the cluster across multiple geographic locations. Data partitioning is supported by tunable consistent hashing schemes and failure modes include reduced availability to keep consistency. Infinispan is a fully fledged data grid solution with features to use

[32]`https://memcached.org`, *Referenced on 1.11.2016*

[33]`http://www.project-voldemort.com/voldemort/`, *Referenced on 1.11.2016*

in many deployment scenarios.[34]

Apache Ignite: Another top-level project in the Apache community, Apache Ignite is an in-memory data- and compute grid for use with large data sets. Ignite's data storage system is key-value based, supports partitioning and replication, near caching, collocated processing, distributed ACID transactions, SQL-queries, persistence, and so on. Like Infinispan, Ignite is a data grid, but in addition a compute grid, which supports, among others, Apache Spark RDD's and in-memory MapReduce operations. These features make Ignite a compelling platform for data analytics.[35]

HBase: Apache HBase is another open source, scalable, fault tolerant, versioned, distributed noSQL database that has been modeled after the Google BigTable [8]. Unlike most data stores, HBase is built on top of another Apahce project, the HDFS,[36] which essentially makes HBase disk bound. HBase is built around strong consistency, but is massively scalable [17].[37]

Redis: An in-memory data store for use as a database, cache and message broker, Redis is also open source. Redis supports strings, hashes, lists, sets and ordered variants thereof as storable data. Where only a few noSQL stores have support for, Redis supports range queries. It also has configurable replication, Lua-scripting, LRU-eviction of data, and transactions. High availability and automatic failover in Redis are achieved via its Redis Sentinel distribution system, and it can be combined with Redis Cluster for partitioning.[38]

Amazon Dynamo: Developed by Amazon in 2007, this key-value store was one of the first to spark the *eventually consistent* data store boom. Among others, Dynamo uses consistent hashing as partitioning model, vector clock versioning and quorums for consistency resolution. It also employs gossip algorithms for failure detection. Like other distributed databases, also Dynamo was developed with the assumption that the hardware it is running on is constantly failing [11].

Apache Geode: The development of the Geode data management platform started already in 2002, when the project was conceived by GemStone. The subsequent commercial product, Gemfire by Pivotal, is now being brought into the Apache Software Foundation via an incubation phase. At heart Geode is an in-memory object storage system featuring, among others, synchronous or asynchronous replication, data partitioning and processing collocation, tunable consistency, configurable persistence, and so on. Data in the system is hosted by a *region* that can be dynamically hosted at any connected server. Very low latency in Geode is achieved with a single-hop architecture and in-order updates to a master host enable strong consistency in the distributed environment. Geode can detect and act on network partitions; in the event of a network failure the minority of the quorum will cease to function while the majority continues

[34]`http://infinispan.org`, *Referenced on 1.11.2016*
[35]`https://ignite.apache.org`, *Referenced on 1.11.2016*
[36]`http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`
[37]`http://hbase.apache.org`, *Referenced on 1.11.2016*
[38]`http://redis.io`, *Referenced on 1.11.2016*

normally. The Apache Geode system also supports delta updates via Geode's own Delta-interface.[39]

Figure 5 highlights some of the important properties of chosen data stores. We observe the following properties:

Consistency model: how consistently the same the data is across hosts, how is this achieved and to what extent. As alluded to previously, data consistency is paramount if multiple consumers of the data rely on it to make changes that rely on the up to date state of the data.

Availability model: in the event of network partitions, how available the data is for reading and writing. Sometimes data may not be available, and it is necessary to understand the circumstances in which it might happen, and if necessary find ways to correct the matter.

Partitioning model: is the data split to be hosted by multiple hosts, how is it achieved. When multiple hosts are used for the data, it is often more reasonable to split data into disjoint sets and store the sets on different hosts rather than store all data on each host. With large enough data volumes it is impossible to do so, and partitioning is the only method to scale the system to accommodate the data.

Distribution model: are all components the same (symmetrical), or is there a hierarchy of dedicated controlling and serving components. Depending on the requirements of the consumers of the data it may be more efficient to extract orchestration of the hosting members into separate components.

Data model: how the data is seen by the system. Traditional databases were built with the relational data model. Many modern systems support simpler and also more complex ways of modelling data. Exactly what model is used dictates the way that data is read and written in the data store.

Replication model: is the data stored at multiple hosts, how is it achieved. To achieve redundancy in the event of failures, it is necessary that the system has multiple copies of the same data. Keeping the data copies in the same state is dictated by the consistency model.

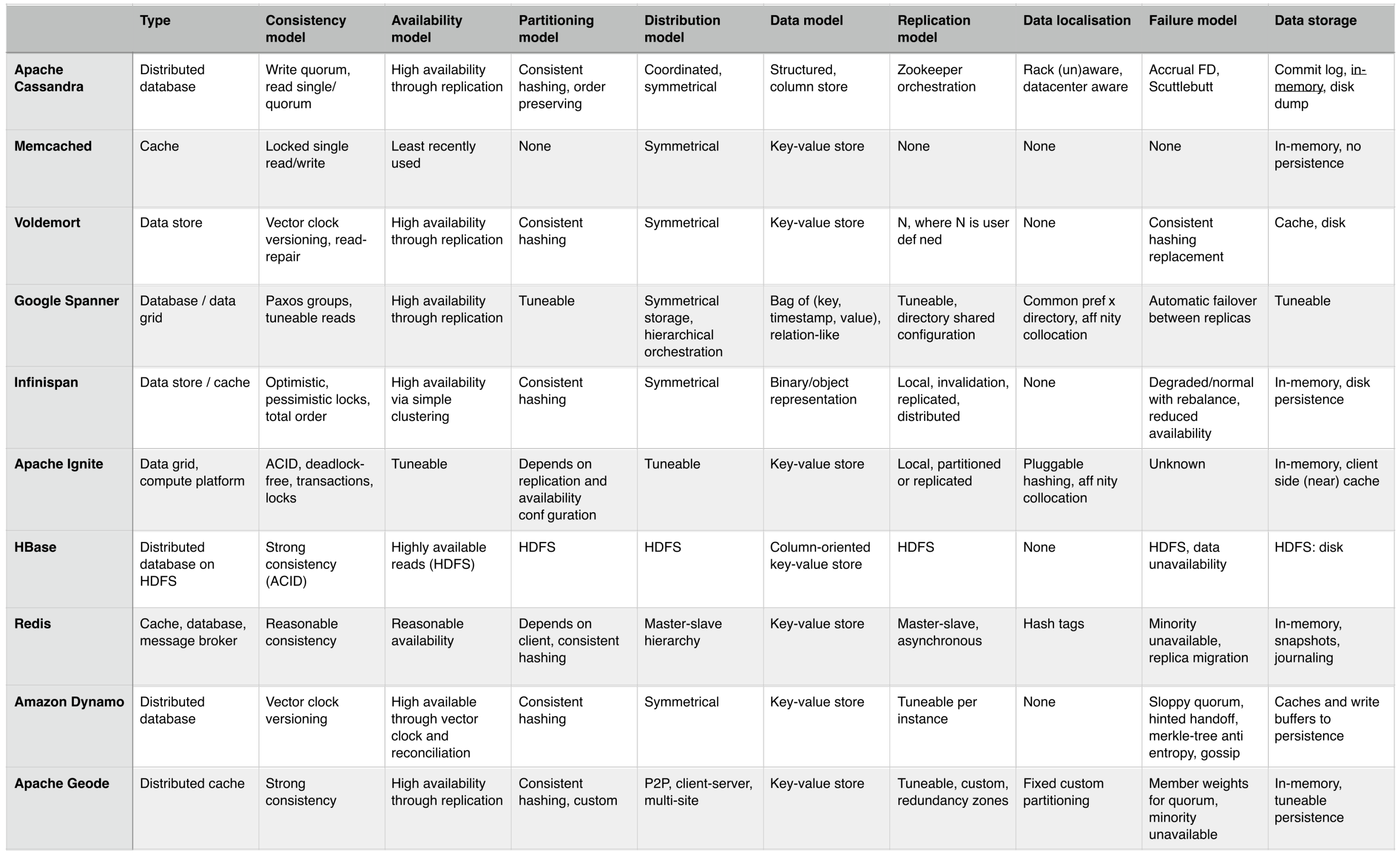
Data localisation: how the system supports locating data to hosts or geographic regions. To avoid placing redundant copies in the same physical machine or even rack in a datacentre it is necessary for the system to be aware of the underlying hardware. Should the data store be distributed across a wide geographical area it may become necessary to place data in specific regions for fast access or durability.

Failure model: how the system is built to operate in the event of data loss or disappearance of hosts due to e.g. network partitions. In distributed systems failures are common and mechanisms are needed to cope with them. For example if a network partition splits the system so that no communication can be completed, the different parts need to know how to function on their own with the information at hand.

Data storage: can the data be written to disk or some other persistent system, how. If a host has a CPU failure which is subsequently fixed, it may be more advantageous and fast for the host to read the data it was hosting from its local disk rather than transfer all of it via a slow network link.

[39]`http://geode.apache.org`, *Referenced on 1.11.2016*

Figure 5: Relevant properties of promising distributed data storage solutions.



	Type	Consistency model	Availability model	Partitioning model	Distribution model	Data model	Replication model	Data localisation	Failure model	Data storage
Apache Cassandra	Distributed database	Write quorum, read single/ quorum	High availability through replication	Consistent hashing, order preserving	Coordinated, symmetrical	Structured, column store	Zookeeper orchestration	Rack (un)aware, datacenter aware	Accrual FD, Scuttlebutt	Commit log, in-memory, disk dump
Memcached	Cache	Locked single read/write	Least recently used	None	Symmetrical	Key-value store	None	None	None	In-memory, no persistence
Voldemort	Data store	Vector clock versioning, read-repair	High availability through replication	Consistent hashing	Symmetrical	Key-value store	N, where N is user def ned	None	Consistent hashing replacement	Cache, disk
Google Spanner	Database / data grid	Paxos groups, tuneable reads	High availability through replication	Tuneable	Symmetrical storage, hierarchical orchestration	Bag of (key, timestamp, value), relation-like	Tuneable, directory shared configuration	Common pref x directory, aff nity collocation	Automatic failover between replicas	Tuneable
Infinispan	Data store / cache	Optimistic, pessimistic locks, total order	High availability via simple clustering	Consistent hashing	Symmetrical	Binary/object representation	Local, invalidation, replicated, distributed	None	Degraded/normal with rebalance, reduced availability	In-memory, disk persistence
Apache Ignite	Data grid, compute platform	ACID, deadlock-free, transactions, locks	Tuneable	Depends on replication and availability conf guration	Tuneable	Key-value store	Local, partitioned or replicated	Pluggable hashing, aff nity collocation	Unknown	In-memory, client side (near) cache
HBase	Distributed database on HDFS	Strong consistency (ACID)	Highly available reads (HDFS)	HDFS	HDFS	Column-oriented key-value store	HDFS	None	HDFS, data unavailability	HDFS: disk
Redis	Cache, database, message broker	Reasonable consistency	Reasonable availability	Depends on client, consistent hashing	Master-slave hierarchy	Key-value store	Master-slave, asynchronous	Hash tags	Minority unavailable, replica migration	In-memory, snapshots, journaling
Amazon Dynamo	Distributed database	Vector clock versioning	High available through vector clock and reconciliation	Consistent hashing	Symmetrical	Key-value store	Tuneable per instance	None	Sloppy quorum, hinted handoff, merkle-tree anti entropy, gossip	Caches and write buffers to persistence
Apache Geode	Distributed cache	Strong consistency	High availability through replication	Consistent hashing, custom	P2P, client-server, multi-site	Key-value store	Tuneable, custom, redundancy zones	Fixed custom partitioning	Member weights for quorum, minority unavailable	In-memory, tuneable persistence

5.2 Meeting the requirements

Throughout the compared systems we can see that well known techniques are being used in various areas of distributed data management. Each system has been constructed to serve as a relatively general data storage system supporting in-memory operations. With the exception of Memcached, which is meant to serve as a ‘front end’ cache, most all support replication for higher availability of data throughout the deployment zone. Consistent hashing [27] is employed in many systems as the model for partitioning the data among hosts, speaking for its usefulness also in choosing placement for redundant copies. Many systems reportedly have tweaked algorithms for choosing the hosting members in a partitioned setting due to the unbalanced nature of the naive consistent hashing algorithm.

Most systems also support some form of data persistence. It is an important factor as disk failure is rarely the sole cause of data loss. Often, it is faster to repair a system and restore some data from the disk rather than transferring all of it via the network. Off-site analysis is just as important and often impossible without proper persistence of data. The architecture of most systems is symmetrical or at least mostly symmetrical. For example, Google Spanner and Apache Cassandra employ symmetrical storage tier architectures with hierarchical orchestration layers. Most systems don’t have a strong basis in choosing geographical locations for data, rather, rack or datacentre awareness or usage patterns are metrics which can be used for locating data.

Consistency techniques are quite standard, as well. Many systems use quorums to decide whether a write (or read) was successful or whether the distributed system is still intact and is able to operate. It would seem two modes of operating are popular: one in which the partitioned systems keep operating as usual, implying eventual consistency, and another where the minority of the quorum of the partition stops functioning or at least has reduced capabilities, namely writes are disabled.

Looking back at the desired properties in section 4.2, we can see that some listed systems adhere to them better and some worse. However, where most systems don’t make it explicitly clear, the Apache Geode supports delta updates. In the attempt to keep network bandwidth usage minimal in update procedures it becomes crucial to update only necessary parts of a stored value. This can be achieved with delta updates: an item is updated and only the *difference* to the old item is sent over the network. While the implementation of delta update procedures are not extremely complex in principle, it would require significant effort to add the functionality to a system that does not support it. Procedures which resemble delta updates are achievable with some SQL-like query languages supported by some noSQL data stores, but with much increased complexity and decreased elasticity of the system: query strings are often static and are not capable of supporting multiple versions of objects in the same data store.

With the added option of easy geo-location of data and strong consistency that it is built for, the Apache Geode system was deemed most proper in the context of the coreless mobile architecture. It could be argued, that as the data is unavailable in the minority quorum the failure model is unfavourable in our proposed architecture and that a more high availability failure model should be adopted. However, we assume that the interconnecting network is stable and that there are redundant paths to keep it functioning properly even in the event of network partitions. To reiterate: we assume that the consistency of the UE context state in the local geographic region is more important than ultra-high availability.

6 Evaluation of a data store: Apache Geode

It is not wise to base trust solely on the described performance and suitability of a system as it may be, and often is, biased [12]. Rather, the system must be tested, evaluated, in order to ascertain its characteristics [9]. For the reasons presented in section 5.2 the Apache Geode distributed data store was chosen to be evaluated as a possible data store to manage the state in the coreless mobile network architecture. In this section we detail the benchmarking system in section 6.1, the workloads and cluster topology used in the evaluation in section 6.2, and finally the results in section 6.3.

Having decided on the system, the actual performance of Apache Geode needed to be understood. From the previous section on desired properties, section 4.2, the speed, scalability and leanness of the system remain to be established. Each property can be divided into subcategories. Speed encompasses the throughput and latency of the system. Additionally, the latency of the system to reach full consistency can be seen as a separate variable to measure. Scalability pertains to the question of how well the systems' speed stays constant when data, transactions and hosts increase in the system [9]. Leanness, as alluded to previously, has to do with the size of update procedures: the smaller and less frequent updates and data fetching are in terms of network usage—while still keeping the system consistent and available—the more lean the system is said to be.

However interesting, it was not possible to map all of the aforementioned properties simply due to time constraints. Therefore, arguably the most interesting aspects of the system, its throughput, latency and scaling (to an extent), were evaluated. The standard method to test such attributes of a system is to benchmark it with appropriate workloads. The *Yahoo!* cloud serving benchmark [9] was chosen for this purpose as it is viewed as an industry standard for benchmarking noSQL systems,[40] it is open source and provides a straightforward interface to extend.

6.1 *Yahoo!* cloud serving benchmark

The YCSB was developed to provide a relatively general purpose tool for benchmarking cloud serving systems. It was designed to be flexible in the definition of workloads and easily extensible to enable benchmarking of new data stores.

6.1.1 *Yahoo!* cloud serving benchmark architecture overview

The YCSB client, a Java program which runs the benchmarks, is depicted in figure 6. It consists of the following components [9]:

Workload definition: A file which contains definitions about the number of operations, for instance insert, update and read proportions. It also contains the number of data items to load into the database and the distributions of internal variables. May also contain database interface and other parameters. By defining new workload parameter files the user can create custom workloads. See appendix A for an example workload file used in our benchmarks.

Workload executor: The YCSB contains a default CoreWorkload executor, which can be used for many purposes. It reads the workload definition file and takes in other parameters optionally supplied at the command line. Using

[40]`http://blog.cloudera.com/blog/2015/08/ycsb-the-open-standard-for-nosql-benchmarking-joins-cloudera-labs/` *(Referenced on 28.10.2016)*

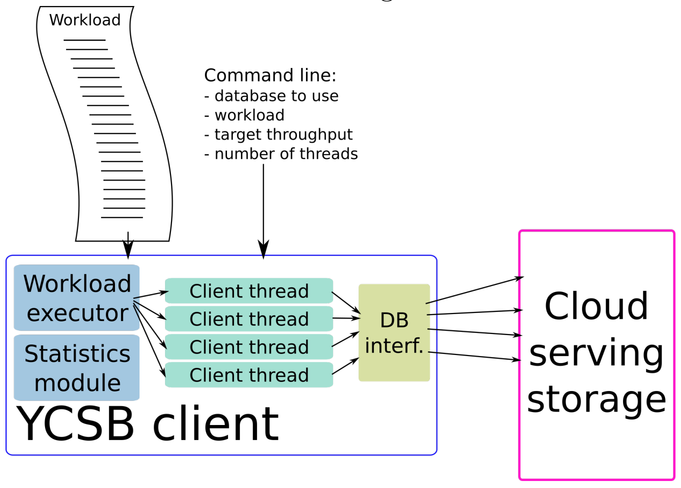
these properties it executes the loading or transaction phase of the workload, depending on the command. The executor was meant to be easily extensible: new workloads can be created by creating a Java class which extends the *Workload* abstract class. Our benchmark was designed in this way, as the full capabilities of the Apache Geode system were otherwise not available.

Statistics module: The statistics are gathered at each transaction automatically. The user can additionally define the collection of read-write latencies, for instance. The module collects results from individual threads so that only a single set of results are output per YCSB client.

Database interface: The YCSB client must somehow connect to the data-base. The database connection has been abstracted so that the workload definition file and other parameters may stay the same, but only the interface to the database is swapped from one to another. This way it is possible to produce comparable results. It makes the extension of the benchmarking tool easier, as well, since the inclusion of a new database requires only that a new database interface is written in Java.

Client thread: The client thread holds a Java-thread instance of the workload executor, which connects to the defined database, performs the workload, measures throughput and latency, and returns the results to the parent process. The parent process then aggregates and finally outputs results to the standard out and, optionally, files.

Figure 6: The *Yahoo!* cloud serving benchmark architecture [9].



6.1.2 Benchmark implementation details

To take advantage of Geode's delta update neither the existing database adapter nor the database interface in YCSB were sufficient. Instead, we implemented the benchmark as an entirely new Workload class, similar to the CoreWorkload. In the GeodeWorkload class the database connection is set up in-place, bypassing the database interface altogether. This allowed us to pass our own object, which implemented the Geode Delta-interface, to the database connection. The UE object was built around the cases described in the EMM-scenario detailed in section 2.3.

In addition to allowing use of our own objects, implementing the GeodeWorkload allowed us to implement our own latency measurements which are important as the standard CoreWorkload only measures read and write latencies separately, whereas all our operations are read-write combinations. Furthermore, it enabled introducing new transaction types beyond the simple Create, Read, Update, Delete (CRUD)-operations, and finally the programmatic combination of regions to demonstrate HAC-zoning became possible. Full source code is available for download from GitHub.[41]

Method 1. The GeodeWorkload functions as follows:

1. Initialise GeodeWorkload class: read properties from file and initialise variables.
2. Initialise client thread: read properties from file and initialise some more variables.
3. Loading phase: In this phase the data store is primed with objects to query in the transaction phase: call the *insert*-transaction method according to the count of UE objects to insert into the database. Use the identifiers to use as keys from the appropriate file and create UE objects.
4. Transaction phase: In this phase the data store is read and written to by the YCSB load generating threads: for each running thread continuously call the *doTransaction* method until the desired transaction count is reached.
   - 4.1 doTransaction: This is the method in which the case to be performed is chosen: query the *discreteOperationChooser* for a transaction type according to the probabilities set in the workload file. Call the designated transaction method.
   - 4.2 Transaction method: In this method the actual reading and writing the data store takes place. There are in total eight transaction methods, one for each case which do the following actions: 1. choose a random identifier to query from an in-memory list, 2. start a timer, 3. query the object from the database 4. transform it accordingly, 5. put it back into the database, 6. stop the timer and report the elapsed time.
5. Report collected statistics.

The UE object to store was built as close to the 3GPP specification [13, 14, 15] of the context data as possible with the knowledge at hand. At this stage the subscriber profile was not included in full as its description was convoluted to the untrained. However, in addition to some important identifiers, the encryption and integrity keys were included. The full UE object is included in appendix B. The methods were modelled after the cases in the EMM scenario. Implementing the Delta-interface required two additional methods: *toDelta* and *fromDelta*. They were modelled after the recommendations in Geode documentation using transient boolean values, which are not included in serialisation, to mark changed fields. In benchmarking the Geode system two versions of the UE object were used: one with Delta updates enabled, one without. This can be seen in the source code for the object in appendix B on lines 16 and 17. The data contained in the UE object implementation (not including overheads) is compared to the specification in table

[41] Branch: geode-updates, `https://github.com/Virta/YCSB/tree/geode-updates`

6. While the actual size of the object even after serialisation is not very easy to calculate precisely, we can calculate a good estimate by understanding the data structures used in variables. We can thus gain some insight into update sizes. As can be seen, the difference, that arises from having to use certain programming language dependent variables, to the specification is not big. Even overheads that double the object size in program memory—which are quite common in Java—should not constitute any problems in modern computing systems.

In addition to the seven cases in the EMM scenario, there remains one important procedure, or set of procedures, that is relevant but has not been touched on. That is session management,[42] which is used to activate, manage, and tear down dedicated bearers that carry traffic to the UE within the core network. Dedicated bearers are tunnels separate of the default bearer and have their own QoS parameters. They can be used to carry, for example, real-time traffic for VoIP which would else be incurred suboptimal QoS in the default bearer, resulting in lowered voice quality. The session management operation is important also because it procedures considerable, albeit not the most, traffic within the EPC, but also because it incurs changes to the UE context. It was deemed sufficient that Session management was implemented as a simple change in the default bearer context.

Table 6: UE context size in implementation vs. specification and involvement in case. *Static* stands for hard coded, *one-time* for generation on insert. In specification we assume one digit (D) to be one byte, 8 bits (b).

Table 6: UE context size			
Item	**Implementation**	**Specification**	**Case (table 7)**
MNC	3x16 b = 48 b	3Dx8 b = 24 b	Static
MCC	3x16 b = 48 b	3Dx8 b = 24 b	Static
PLMN ID	6 x 16 b = 96 b	MNC+MCC = 48 b	Static
MMEGI	1x16 b = 16 b	16 b	Static
MMEC	1x16 b = 16 b	8 b	Static
MMEI	2x16 b = 32 b	MMEGI+MMEC = 24 b	Static
GUMMEI	8x16 b = 128 b	PLMNID+MMEC = 56 b	Static
PGW ID	19x16 b = 304 b	FQDN(32 b)	Static
TAI size	1x32 b = 32 b	n/a	Static
IMEI	15x16 b = 240 b	15Dx8 b = 120 b	Static
MSIN	6x16 b = 96 b	10Dx8 b = 80 b	Static
IMSI	15x16 b = 240 b	15Dx8 b = 120 b	One-time
K_{master}	8x16 b = 128 b	128 b	One-time
K_{cipher}	8x16 b = 128 b	128 b	One-time
K_{enc}	8x16 b = 128 b	128 b	One-time
Status	1x8 b = 8 b	n/a	1 – 8
MTMSI	1x32 b = 32 b	32 b	1, 5
GUTI	8x16 b + 1x32 b = 160 b	GUMMEI+MTMSI = 80 b	1, 5
TIN	GUTI = 160 b	80 b	1, 5

[42] `http://www.3gpp.org/technologies/keywords-acronyms/96-nas`, *(Referenced on 1.11.2016)*

Table 6 continued			
Item	**Implementation**	**Specification**	**Case (table 7)**
IP	1x32 b = 32 b	32 b	1, 2
CRNTI	1x16 b = 16 b	16 b	1 – 4, 6
eNB UE S1AP	1x32 b = 32 b	32 b	1 – 4, 6
MME UE S1AP	1x32 b = 32 b	32 b	1 – 4
OLD eNB X2	1x32 b = 32 b	32 b	n/a
NEW eNB X2	1x32 b = 32 b	32 b	n/a
ECI	1x32 b = 32 b	28 b	1 – 4, 6
ECGI	PLMN ID + ECI = 128 b	52 b	1 – 4, 6
TAI	1x32 b = 32 b	32 b	1, 5
TAI list	15x32 b = 480 b	15Dx32 b = 480 b	1, 5
PDN ID	656 b	FQDN(100Dx8 b = 800 b)	1, 2
EPS bearer	1x8 b = 8 b	4 b	1, 2, 8
E-RA bearer	1x8 b = 8 b	4 b	1, 2, 8
DR bearer	1x8 b = 8 b	4 b	1 – 4. 6
S1 TEID UL	1x32 b = 32 b	32 b	1, 2, 8
S1 TEID DL	1x32 b = 32 b	32 b	1, 2, 8
S5 TEID UL	1x32 b = 32 b	32 b	1, 2, 8
S5 TEID DL	1x32 b = 32 b	32 b	1, 2, 8
K_{ASME}	16*16 b = 256 b	256 b	1, 2
K_{eNB}	16*16 b = 256 b	256 b	1 –4, 6
K_{NASint}	16*16 b = 256 b	256 b	1
K_{NASenc}	16*16 b = 256 b	256 b	1
K_{RRCint}	16*16 b = 256 b	256 b	1 – 4, 6
K_{RRCenc}	16*16 b = 256 b	256 b	1 – 4, 6
K_{UPenc}	16*16 b = 256 b	256 b	1 – 4, 6
Total	5’152 b	4’596 b	

6.2 Workload and cluster setup

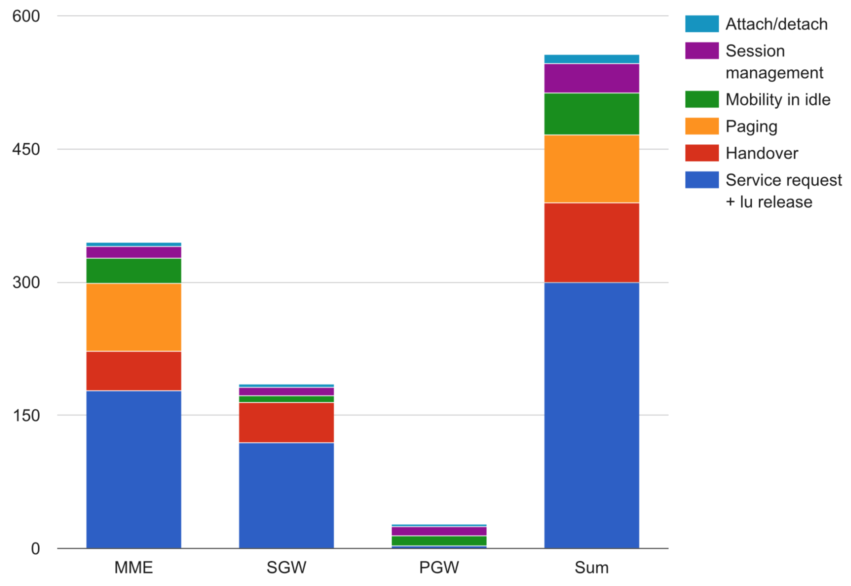
Our aim was to benchmark the Apache Geode system with workloads as close to those seen in the real world as possible. The UE object and transitions in context reflect this with respect to individual updates. In addition, we wanted to base the count and frequency of transitions close to a real world model. Combined, the two facets could provide information about the speed of the data store to accredit comparisons to an actual deployment.

6.2.1 Defining the workload

We resolved to replicate workload according to the signalling load distribution in a report by Alcatel-Lucent [4]. It reports the signalling load distribution within the EPC according to a 16 busy hour call attempts per subscriber and 15 eNB TA size traffic model. While this is a traffic model, we assume that it is reasonably close to a real world scenario. The distribution of signalling loads is reproduced in figure 7.

The counts of messages were approximated from the original graph because the actual numbers were not available. The components participating into the signalling in figure 7 were then translated into the cases of the EMM scenario as follows:

Figure 7: Signalling message load distribution in the EPC [4].



Attach / detach: Correspond directly to attach and detach cases. Totalling 11 messages.
Session management: Corresponds to the session management described in section 6.1.2 on page 38. Totalling 33 messages.
Mobility in idle: Corresponds to TAU and cell reselection with TAU. Totalling 47 messages.
Handover: Corresponds to handover and handover with TAU. Totalling 89 messages.
Service request + lu release: Correspond to service request and S1 release. Totalling 300 messages.
Paging: Does not correspond to any case, therefore omitted.
Total events: 480 messages.

The probabilities of context switches, i.e. occurrence of a case, were calculated from the distribution according to their respective proportions. Equation 1 details the calculation:

$$P_{case} = \frac{\sum n_{sc}}{n_{cc}} \cdot \frac{1}{N \cdot n_{occ}} \tag{1}$$

Where
P_{case} is the probability of a case.
n_{sc} is the count of signals in components where case occurs.
n_{cc} is the count of cases in a component.
n_{occ} is the count of occurrences of a case.
N is the total number of signals.

As an example, equation 2 details calculation of the probability of the initial attach case:

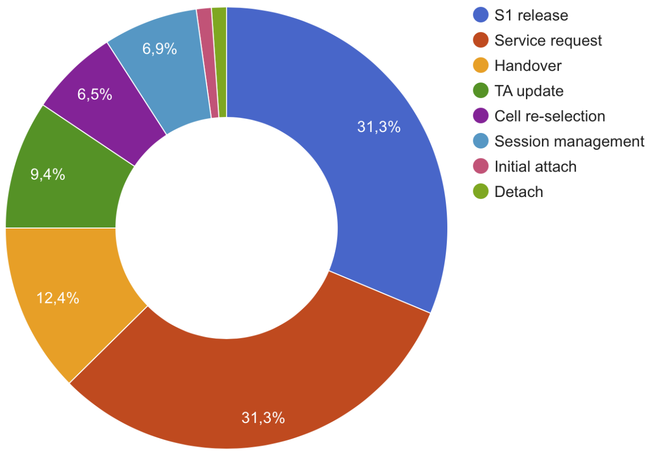
$$P_{attach} = \frac{11 signals}{2 cases/component} \cdot \frac{1}{480 \cdot 1 component} = 0.011 \tag{2}$$

The probabilities yielded by applying equation 1 to all cases are listed in table 7, and are depicted in figure 8. These probabilities were defined in the benchmark workload files for each case respectively. See appendix A. Table 7 also lists the approximate update size of each transition in UE context in our implementation. Update sizes are derived from table 6 by summing the changed context items for each case, respectively. As with table 6, it is difficult to accurately calculate the serialised delta update size, but the figures provide good estimates. The table also details the count of signalling messages for each case. They are needed in calculating the count of supported UE from the throughput achieved in our experimentation.

Table 7: Translated probabilities of a case occurrence and update size.

Table 7: case probabilities			
Case	**Probability**	**Update size**	**Signals**
1: Initial attach	$11/2 \cdot 1/480 = 0{,}011$	3'744 b	30
2: Detach	$11/2 \cdot 1/480 = 0{,}011$	2'400 b	15
3: Service request	$300/2 \cdot 1/480 = 0{,}313$	1'280 b	16
4: S1 release	$300/2 \cdot 1/480 = 0{,}313$	1'280 b	6
5: TA update	$(47+89)/3 \cdot 1/480 = 0{,}094$	736 b	16
6: Handover	$89/3 \cdot 1/480 \cdot 2 = 0{,}124$	1'088 b	17
7: Cell reselection	$47/3 \cdot 1/480 \cdot 2 = 0{,}065$	0 b	0
8: Session management	$33 \ / \ 480 = 0{,}069$	152 b	8

Figure 8: Probabilities of a case, transition in UE context.



6.2.2 Apache Geode cluster topology

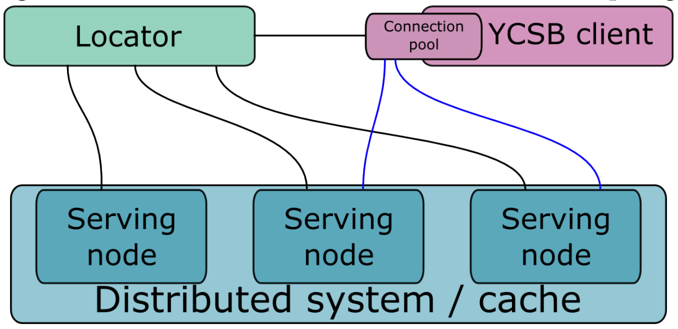
Apache Geode supports many different topologies for deployment.[43] We chose to deploy the cluster in the client-server model for our benchmarks for the following reasons: it matches the model of the coreless mobile network architecture where the EPC-controller is a client requesting services and the cluster provides them, it allows us to choose if the client should have a local cache, it has a better performance and

[43]`http://geode.apache.org/docs/guide/topologies_and_comm/topology_concepts/topology_types.html` *(Referenced on 8.11.2016)*

scales better. We did not deem it wise to employ a mode of operation where the EPC-controller itself would be a part of the data distribution framework, which would have been the case in the peer-to-peer topology, as it would tie the EPC-controller implementation to that of the data store connection.

Figure 9 provides an overview of the topology. The cluster is comprised of two types of processes: locators and serving nodes. The locators orchestrate the cluster. They connect to the serving nodes to receive load information and to orchestrate data balancing. Client applications connect to the locators to query connections to serving nodes, which are provided by the locator in a least-loaded first fashion. The connections returned to clients are pooled in their connection pool and are used for single-hop querying of data to and from the serving nodes. The cluster topology is the same for *replicated* and *partitioned* regions. A replicated region is a set of data that is replicated on every single serving node on the cluster, whereas a partitioned region is one where the data is partitioned to the serving nodes with a consistent hashing function. A partitioned region in Geode as configurable data redundancy.

Figure 9: Overview of the client-server cluster topology.



For the benchmarking, we used up to six Dell C6320 server machines, each containing 2 x Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz (48 logical cores with hyperthreading enabled), 256 GB DDR-4 dual rank 2133MHz main memory, 2x10Gbps NICs with bonding disabled, and 2x600GB 10 krpm disk storage in hardware RAID 1 configuration. All machines were installed with Ubuntu Linux 16.04, swap was disabled because of ample RAM. The network was reserved for the machines only, with very little other traffic, the average ping measured between any two machines was 150 μs. The Apache Geode cluster was built from the *incubating.M2*-source[44] with Oracle Java 1.8 as the compiler and runtime. The same Java version was used also for the compilation and running of our customised YCSB.

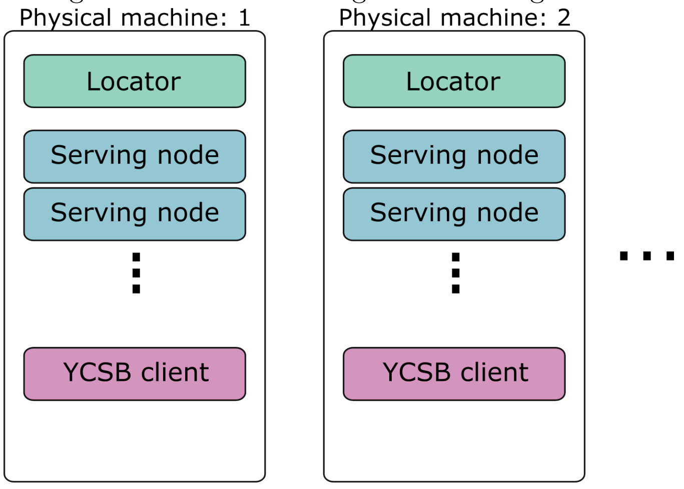
The benchmarking procedure itself consisted of a preliminary setup of the cluster, and an automated scaling benchmark procedure which was repeated each time a new machine was brought into the setup. The preliminary setup included starting one instance of locator and server processes on each machine used in the current benchmark, and initialising the data store with data. For the main body of benchmarks we used 1'000'000 UE instances and 3'000'000 transactions, these are defined in the YCSB workload files, an example is in appendix A. Method 2 describes the automated scaling benchmark procedure.[45] The resulting cluster configuration

[44]`https://github.com/apache/incubator-geode/tree/rel/v1.0.0-incubating.M2`

[45]For the detailed procedures see the source code in `https://github.com/Virta/YCSB/`

is depicted in figure 10. We scaled the number of load generating threads of YCSB incrementally to find the maximum throughput as it was not possible to achieve maximum throughput with a single load generating thread. While not strictly necessary, we included a locator on each physical machine for two reasons: it eased the automation of the scaling benchmark, and it provides extra redundancy for the clients pool requests in the event of a locator failure.

Figure 10: Benchmarking cluster configuration.



Method 2. Procedure for the automated scaling benchmark of Apache Geode:

```
1. Repeat for $T_n$=1 until $T_n >= T_{max}$:
    1.1 Run the benchmark with $T_n$ threads on each participating machine.
    1.2 Increment $T_n$ by scaling factor $I$.
2. If $S_{current} < S_{max}$ repeat step 3 for $I$ times and continue. Else stop.
3. For each machine:
    3.1 Start a new Geode serving node, with optional grouping parameter.
4. Initiate cluster rebalance operation.
5. Return to step 1.
```

Where:
T_n is the current thread count for YCSB.
T_{max} is the maximum desired YCSB thread count.
I is the scaling factor according to which thread count is incremented, and how many more serving nodes are started.
$S_{current}$ is the current number of Geode serving nodes per machine.
S_{max} is the maximum desired number of Geode serving nodes per machine.

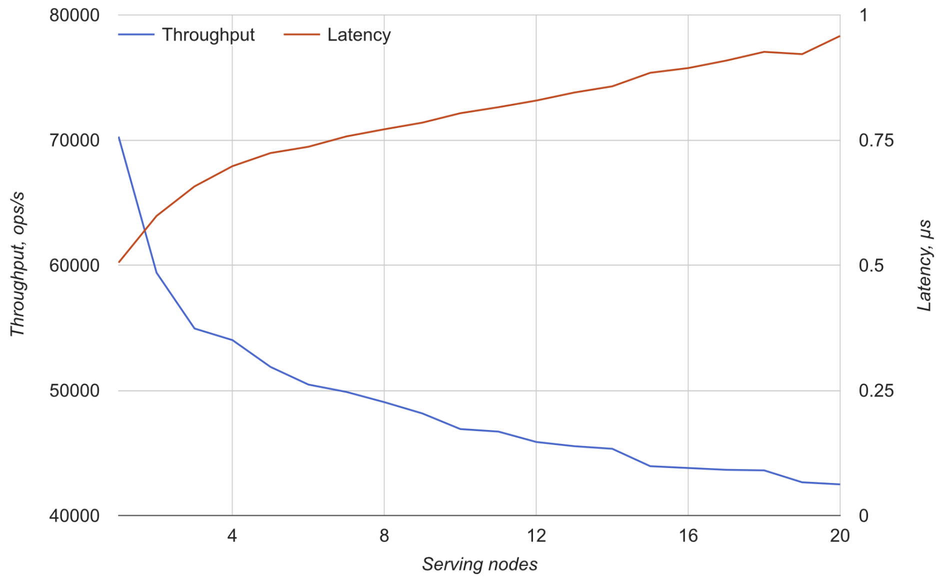
6.3 Results

The scaling benchmark was applied to the Apache Geode distributed storage system deployed in our local computing cluster. The benchmark scaled from 1 to 40 load generating threads for each combination of hardware machines and

`tree/geode-updates/GeodeBenchmarkingScripts/benchmarking`.

active serving nodes. The serving node count was incremented on each machine to discover how the system performs in this type of scaling. We present results of benchmarking the system in the following region configurations: simple partitioned region, partitioned region with three-way redundancy, replicated regions, and a programmatic combination of the two. The latter forms an example use case featuring a global region and a HAC-zone. We applied the benchmark symmetrically to both versions of the UE object, with Delta updates and without. The full result set including some preprocessing is available online.[46]

Figure 11: Maximum throughput and latency achieved on a single machine with a simple partitioned region with no redundancy.



6.3.1 Baseline

To understand the baseline throughput that the Apache Geode data store and our benchmarking system could achieve we ran a very simple benchmark. It consisted of a partitioned region with no redundancy (each data item is stored in the system only once), and one physical machine. In the setup we could observe how the system, while confined to one machine, performs in terms of latency and throughput on incrementing serving nodes. The benchmark used delta updates.

Figure 11 presents the result of benchmarking the simple system. We can clearly see how the throughput decreases and latency increases as we add more serving nodes to the system. The maximum throughput is observed with one serving node in the system at 70'272 ops/s, the average latency was 0.51 μs. The throughput decreased to 42'487 ops/s, latency was 0.96 μs. There are likely two phenomena that constitute the decrease in throughput. Firstly, there is a single locator in the system which must serve the benchmarking client with the appropriate connections

[46]https://github.com/Virta/Written-works/blob/master/GeodeBenchmarkingResults.tar.gz *(Referenced 24.11.2016)*

to serving nodes, it may become overburdened. Secondly, we increase the number of load generating threads from 1 to 40 and their presence on the same machine may induce resource contention. Any one machine has a limited amount of resources for which all processes are contending. Increasing the number of processes on the machine will eventually result in all resources being in use, whose effect is increased CPU context switching and memory access. Too much context switching and queuing in the operating system's process pool decreases the efficacy of all processes and therefore decreases throughput and increases latency. In fact, we observed that with 10 serving nodes and 20 threads the CPU utilisation was about 70% on all 48 logical cores, and it did not change significantly with more serving nodes and threads.

6.3.2 Partitioned regions

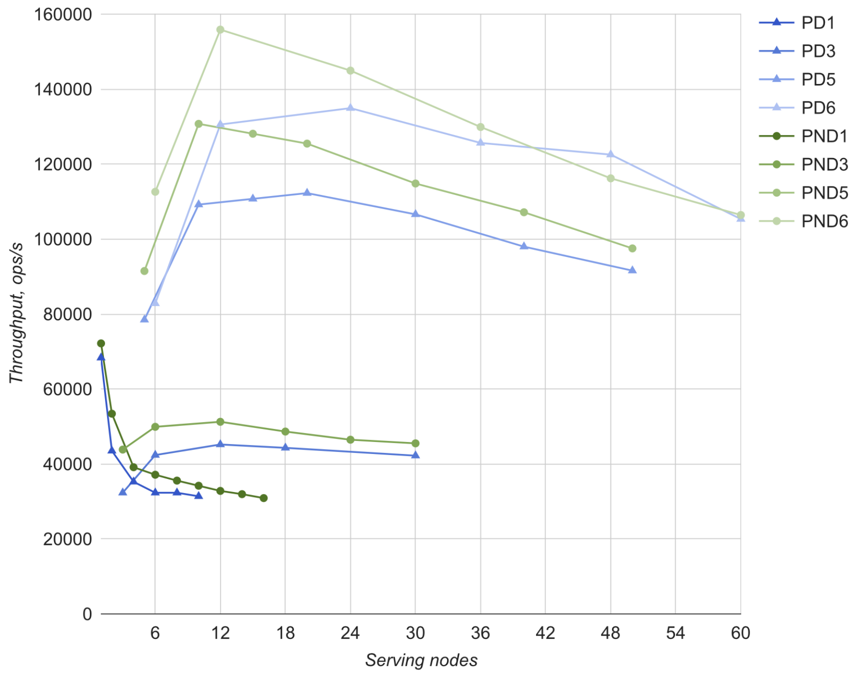
In Geode, a partitioned region with redundant copies of data contains a primary copy, which functions as the update anchor. All writes go to the primary from whence they are synchronously (not involving the original client) propagated to all secondaries. Reads go to any member hosting the data. In our benchmark we used a three-way redundancy policy, which Geode locators try to uphold in the event of a loss of serving nodes. Geode also tries to place the three copies of data into serving nodes on different physical machines. The behaviour can be enforced, and Geode will stop serving data if the policy if violated. We did not enforce unique hosts in our experimentation.

Figure 12 details the maximum throughput at each machine and serving node increment. From it can be seen how the throughput rapidly increases when adding more machines into the cluster. With only one machine in the cluster, RD1 and RND1, the throughput is good and, as in the simple partition experiment, it decreases quite rapidly with the addition of serving nodes, likely due to resource contention. After three physical machines in the cluster the throughput increases rapidly and peaks at 134'938 ops/s with Delta updates and with four serving nodes per machine, 24 in total. The delta update enabled cluster delivers 130'565 ops/s with 2 serving nodes per machine, 12 in total; at the same point (12 serving nodes total) the non-delta update peaks at 155'882 ops/s, delivering a 19,4% increase in throughput. Contrary to expectations the non-delta update mechanism performs constantly better. This is highly likely due to the overhead in applying the delta update to the data in the cluster. We discuss the possible cause more at length in section 7.3. The slight decline of throughput with increasing serving nodes is likely due to resource contention.

Figure 13 details how the client observed latencies evolved in the benchmark. The declining throughput for one machine is evident also in the latencies which increase in the very beginning, then stay constant. We are also observing expected behaviour in the five and six machine experiments: in the beginning the latencies decrease where the throughput is also peaking, and later the latency increases with the increment of serving nodes and declined throughput.

This behaviour is possibly due to the following phenomenon: with less serving nodes the likelihood of a data item to be residing on another machine is higher than with more serving nodes. For example: with five machines, one serving node per machine and three redundant data replicas, there are $\frac{5}{3} \cdot 1'000'000 = 600'000$ data items on the local machine assuming Geode was able to place them all in different machines (which is the default behaviour with best-effort). Assuming a totally even distribution (resulting from the random choosing of UE objects) 40% of objects are read over the slow network link. In theory, not including processing delays, this should lead to an average latency of about $\frac{60 \cdot 0.5\mu s + 40 \cdot 150\mu s}{100} = 60\mu s$. On the other hand, calculating the theoretical latency (including processing delays) from throughput, we arrive at $\frac{1'000'000\mu s}{100'000op(/s)} = 10\mu s$.

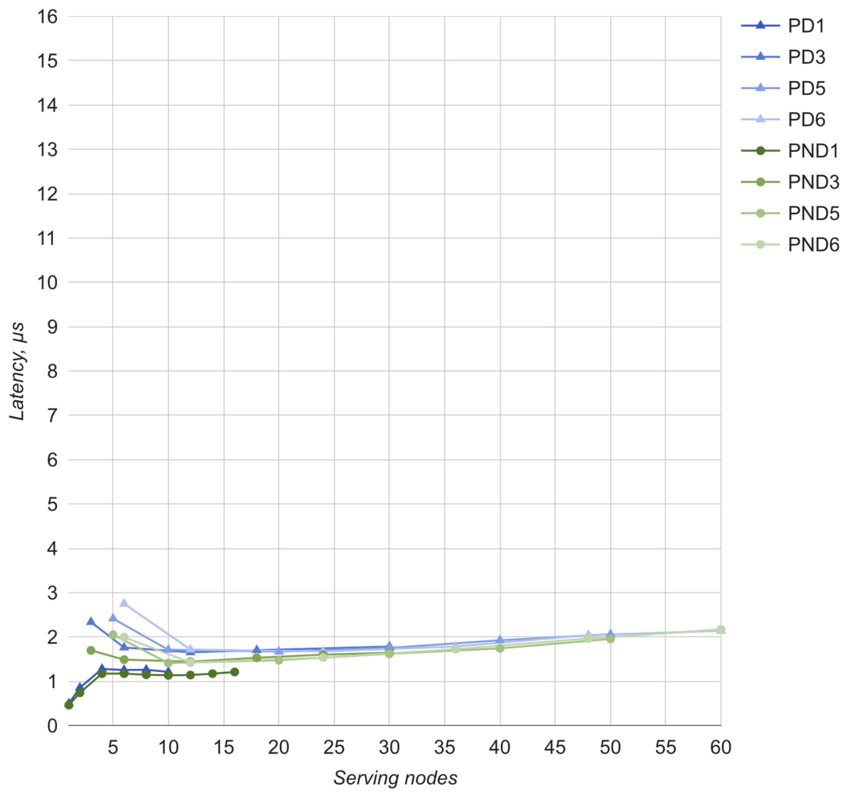
Figure 12: Maximum aggregate throughput for a partitioned region with redundancy starts from around 40’000 ops/s and scales to about 150’000 ops/s.



Label	**Update type**	**Machines**	**Serving nodes**
PD1	Delta	1	1–10
PD3	Delta	3	3–30
PD5	Delta	5	5–50
PD6	Delta	6	6–60
PND1	No delta	1	1–16
PND3	No delta	3	3–30
PND5	No delta	5	5–50
PND6	No delta	6	6–60

The source of these discrepancies is yet to be determined but it is entirely possible that there are further optimisations implemented in Geode that we are not aware of. For instance, a client may be allowed to return immediately after transmitting an update to the serving node, rendering the update local at first and Geode may propagate the update to the cluster after-the-fact. As our configuration contained one or more serving nodes per physical machine, it is very possible that the client is connected to the local serving node rather than one across a network link, making the connection from client to serving node very fast. At this stage we do not know if there is a mechanism for observing the clients’ connection to the serving node.

Figure 13: Average latencies for a partitioned region with redundancy. The latencies are mostly under 2μs, speaking for the in-memory operations.



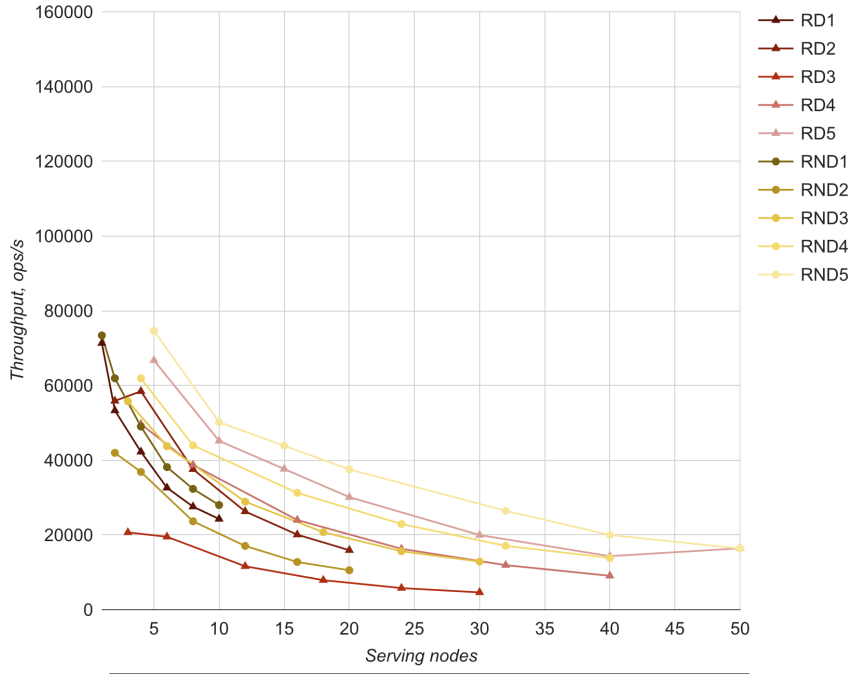
YCSB reported the latencies in the experiment to be, on average, below 2 μs and just above that even for 60 serving nodes in both delta and non-delta update mechanisms. The observed 99th-percentile latencies were reported to be 1 μs. As an in-memory data store Geode seems to be performing well according to this experiment.

6.3.3 Replicated regions

A replicated region in Apache Geode is a data set that is symmetrically stored at every serving node that hosts the region. Our benchmark was configured to use the *distributed-ack* update procedure, which states that on sending an update the client must receive an acknowledgement of reception from all serving nodes before continuing. The procedure is slower than no acknowledgement, but it increases cache consistency dramatically. It is also possible to configure Geode with global locking, but the mechanism is slow as it requires all serving nodes to lock the data item for the duration of the update.

The results are detailed in figure 14. Following the trend of the partitioned region result, the non-delta updated region performs better than the delta update in all but one (RD2 vs. RND2) setting. Experiment RND5 (with 5 machines) performs best throughput-wise, starting from 74'614 ops/s and declining to 16'369 ops/s. As expected, the throughput of the system gradually decreases with the increase of new serving nodes, as the count of update acknowledgements increases proportionally. The maximum throughput achieved at one serving node per physical machine is 66'722 ops/s for the delta update and 74'614 ops/s for non-delta updates. The maximum achieved throughput at 15 serving nodes without delta updates is 43'864 ops/s. Despite the relatively low and slow decline of throughput compared to partitioned regions, the overall result indicates that even replicated regions scale. At best, there is a 50% increase in throughput when scaling from RND4 to RND5.

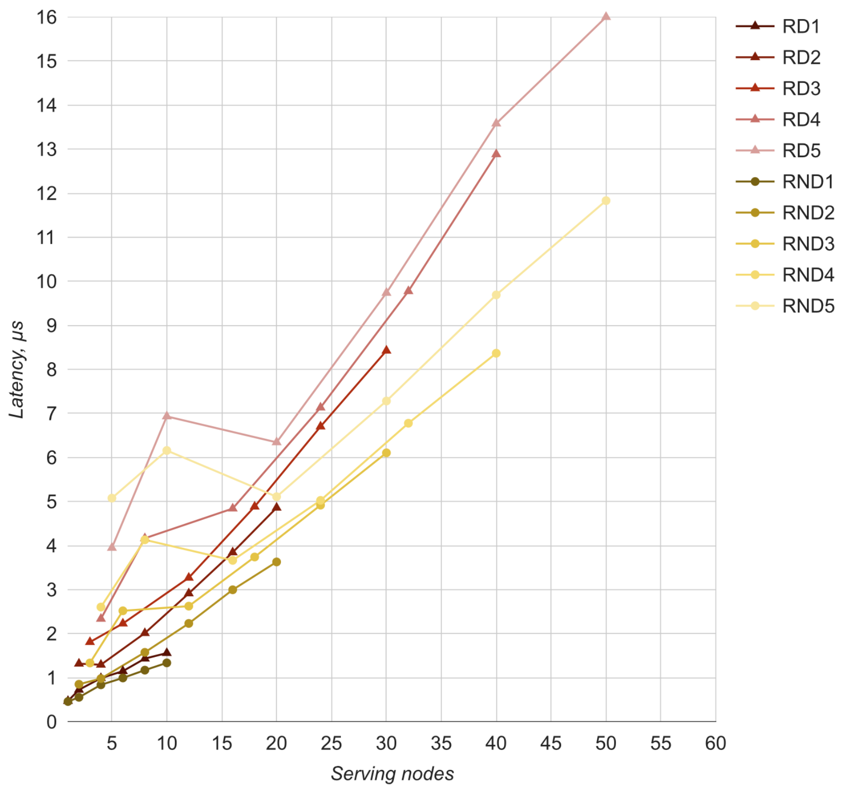
Figure 14: Maximum throughput for a replicated region. The throughput expectedly declines with the increment of serving nodes.



Label	Update type	Machines	Serving nodes
RD1	Delta	1	1–10
RD2	Delta	2	2–20
RD3	Delta	3	3–30
RD4	Delta	4	4–40
RD5	Delta	5	5–50
RND1	No delta	1	1–10
RND2	No delta	2	2–20
RND3	No delta	3	3–30
RND4	No delta	4	4–40
RND5	No delta	5	5–50

Figure 15 details the average latencies observed at the client during benchmarking the replicated region. The latencies behave as expected: with the increase of serving nodes the latency for operations steadily increase. YCSB reports latencies to steadily increase from under 2 μs with less than 5 serving nodes, to 16 μs with 50 serving nodes. The latencies for delta and non-delta update mechanisms followed the observed pattern where delta updated objects are slower than their non-delta counterparts. We did observe maximum latencies reaching 8'003 μs in the RD5 experiment with 33 load generating threads. However, the observed 99th-percentile latencies were reported to be no larger than 64 μs. The high maximum latency is likely due to resource contention as well as the high number of serving nodes and subsequent delays from replication acknowledgements.

Figure 15: Average latencies in a replicated region. The latencies increase almost linearly with the increase of serving nodes.

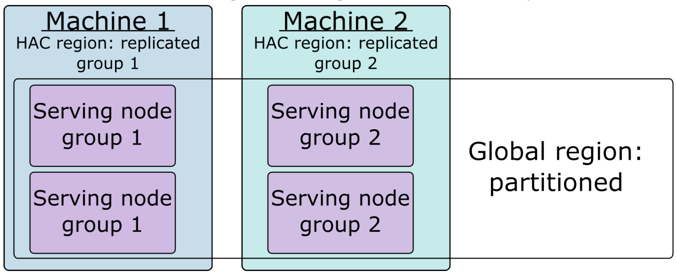


6.3.4 Programmatic combination: an example of high availability and consistency zoning

As a demonstration of a type of HAC-zoning we implemented the programmatic combination of the two types of regions already benchmarked: partitioned and replicated regions. First the 1'000'000 UE objects were split into 10 disjoint chunks of 100'000. Each chunk was assigned to a specific *member group*. In Geode a region can be defined to be hosted by a *group* of serving nodes (*members*). Only the serving nodes started with the specific group identifier host data in the said region. We assigned each physical machine with a group identifier and started each serving node with the identifier of the host machine. Each group was made host of a replicated region, within which the chunk of data is hosted. Further, all serving nodes were made host of a global region of type partitioned with three copy redundancy.

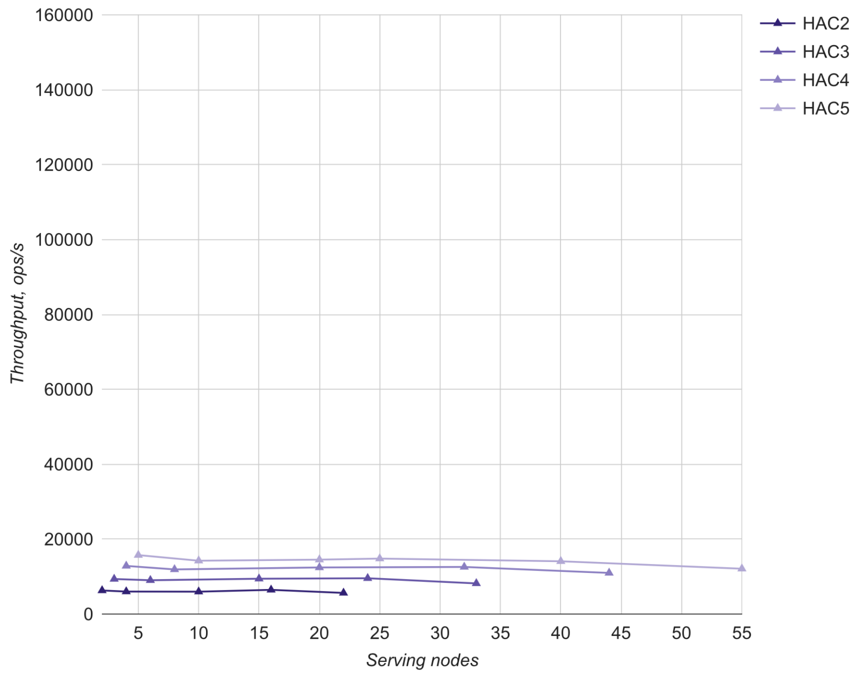
Next, we implemented extra logic in the GeodeWorkload transaction phase: each transaction fetches a random UE object from the group according to the group identifier of the physical machine that the YCSB client is on, modifies it according

Figure 16: HAC-zoning with Apache Geode. This example contains two machines, each containing one HAC-zone which are hosted by two serving nodes on each machine. The global region is hosted by all four serving nodes.



to the case type, the puts the object back into the same replicated region (same group) and additionally to the global partitioned region. The handover procedure was modified as follows: once a UE object had been fetched and mutated, a new group was randomly selected, a new connection pool was established to the new group and the object was put there and the global region, finally the UE object was removed from the old group. Figure 16 demonstrates the configuration.

Figure 17: Maximum throughput in programmatic HAC-zoning. The throughput is stable but much lower than in other benchmarks, likely due to implementation issues.

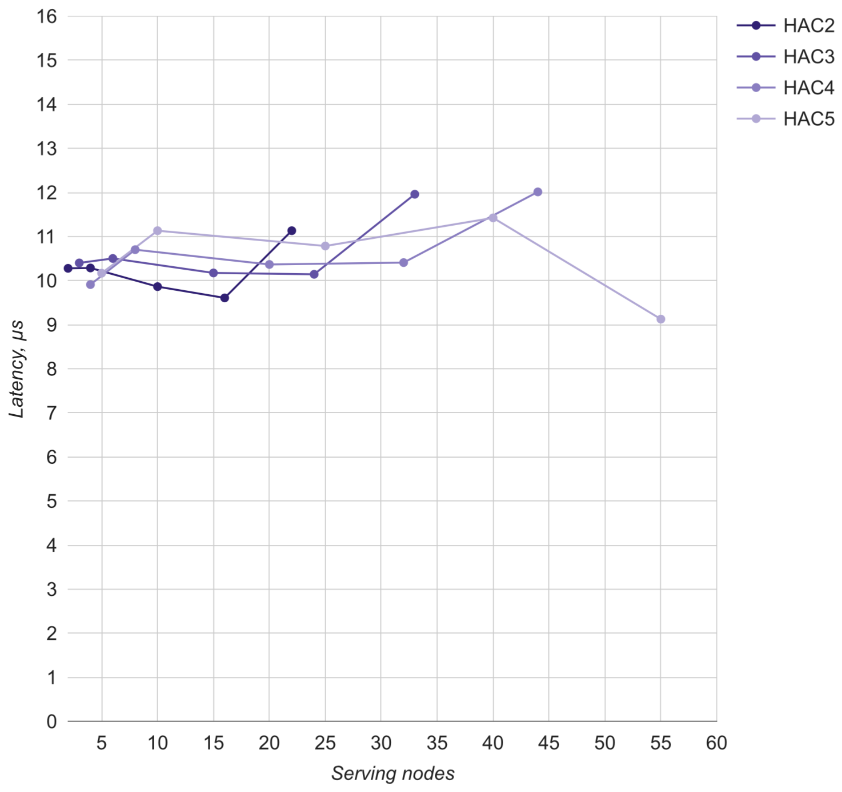


Label	Update type	Machines	Serving nodes
HAC2	Delta	2	2–22
HAC3	Delta	3	3–33
HAC4	Delta	4	4–44
HAC5	Delta	5	5–55

All other parameters in the benchmark were kept the same, with the exception that the experiment was started at two physical machines. We did not see value in benchmarking the system with only a single machine, because it would inevitably mean that our implementation of the handover case would not do anything useful. The serving nodes were instantiated similarly as in previous benchmarks but with the additional grouping parameter according to the physical machine.

The resulting throughput with delta updates in the experimental HAC-zone benchmark are depicted in figure 17. Even though relatively constant in throughput, the HAC-zoning performed poorly in comparison to the previous results. Even with five machines, HAC5, the throughput peaks only at 15'741 ops/s with one serving node, then continues in the same numbers until finally falling off to 12'103 ops/s. Despite that, the benchmark demonstrates that the system scales with the addition of machines: there is a 22% improvement in throughput from HAC4 to HAC5.

Figure 18: Average latencies in experimental programmatic HAC-zoning. The difference in percent in between the configuration is not large, which points to implementation issues.



The poor performance is most likely due to the implementation and artefacts thereof. In implementing the handover operation in this experiment we were forced to introduce some thread synchronisation into the GeodeWorkload. This was due to the structure of the implementation, where the next UE identifier is randomly chosen from a list in the local process space, the list is not thread specific. In order to make modification of the list thread safe and avoid exceptions during running the benchmark we introduced read-write locks around the list operations. As demonstrated in the results by the stable throughput, this had the unfortunate effect that throughput declined dramatically. Without the aforementioned locking, it is expected that the HAC-zone could perform as the replicated region because it is implemented as one.

The average latencies reported by YCSB have been gathered in figure 18. The steady throughput is quite evident from the graph, the latencies are roughly in the same numbers, and of the same order of magnitude. The effect of locking is evident because there is little development in throughput and latency throughout the experiment. The average latencies stay within 9–12 μs, the 99th-percentile is reported at most 22 μs while maximum latencies did not exceed 443 μs. The reported latencies include only the read, modify and update operations, and none of the other processing latencies in the GeodeWorkload program code.

6.3.5 Summary

In this section we presented the results of benchmarking Geode partitioned and replicated regions, and additionally the results from a programmatic combination of the two. The partitioned region with three copy redundancy demonstrated good scaling properties with the addition of new machines into the cluster, and retention of throughput when adding serving nodes. Replicated regions demonstrated expected properties of diminishing throughput when new serving nodes and machines were brought into the cluster. Between both the partitioned and replicated regions the non-delta update mechanism performed better than the delta counterpart, highly likely due to the processing overhead of delta updates.

Even though the resulting throughput in our experimental HAC-zoning is poor compared to the other two types of regions, it servers as a demonstration of the capabilities of programming with Apache Geode. It is likely that it could be significantly improved through redesign of the implementation and taking full advantage of Geode's advanced features. The main reason for lowered throughput comes from the introduction of thread synchronisation in our implementation. In principle it should follow the throughput of replicated regions, as it has been established as the bottleneck compared to partitioned regions.

6.4 Comparisons

In the time of writing this thesis only one source detailing the benchmark of the related GemFire data store was found that provides sufficient detail to accredit comparison. Zdata conducted a benchmark similar to ours, with the aim of discovering various aspects, including horizontal and vertical scaling characteristics of partitioned regions. The report concludes that in their experimentation the maximum sustained throughput reached 100'000 ops/s with read/write latencies under 2ms, and up to 25% throughput increase in horizontal scaling.[47] The throughput achieved in their benchmark is remarkably close to our result. However, there is a 1000-fold difference in latencies. Namely, where our results for partitioned regions indicate microsecond latencies the results of the Zdata benchmark indicate latencies in the millisecond range.

Comparing the result from our benchmark to that of other distributed storage systems we gain some insight to the relative performance of the Geode data store.

[47] `http://zdatainc.com/2015/02/gemfire-performance-evaluation-aws/` *(Referenced 10.11.2016)*

For example, Kuhlenkamp et al. used YCSB to benchmark the Cassandra and HBase data stores [30]. Their results show that at eight serving nodes on separate machines the maximum attainable throughput for a read intensive workload is under 50'000 ops/s for Cassandra and around 10'000 ops/s for HBase. Their respective latencies are reported at an average of 30 ms (above 200 ms at 99^{th}-percentile) and about 100 ms (above 1000 ms at 99^{th}-percentile). Both systems have been optimised for writes, and it is reflected in the results: at the same eight nodes, Cassandra achieves 75'000 ops/s and HBase about 150'000 ops/s for a write intensive workload. Their respective latencies are reported at an average 15 ms (110 ms at 99^{th}-percentile) and 1 ms (1 ms at 99^{th}-percentile). The results go to show that depending on the purpose of the data storage system, whether it be write or read optimised, has a tremendous impact on the performance of the system in different scenarios. Comparing our results, it would seem that the Apache Geode system can perform well in both read and write intensive workloads. The in-memory characteristic of the Apache Geode is shown in the resulting extremely low latencies.

7 Discussion

We have proposed and presented a high level description on a new mobile network architecture, the coreless mobile network. To support the architecture we have analysed the suitability of modern noSQL data stores as a unifying data layer for the EPC controllers residing in smart-boxes. We built a benchmarking system on the YCSB framework to assess the characteristics of one data store, the Apache Geode, and to provide a rough proof-of-concept of HAC-zoning. In this section we aim to discuss the interpretations that can be drawn from the results and to provide critique on some relevant aspects.

7.1 Interpreting the results

Looking back at the definition of the workload in section 6.2 and our results in section 6.3 we can calculate the theoretical number of active UE that our system could support. We fist transform the ops/s quantity into signals/s for each case according to equation 3. After summation of the signals from each case, we can divide the sum by the amount of signals per UE per busy hour traffic to yield total UE count per hour supported by our system, see equation 4. The resulting UE count at 5 machines with a total of 15 serving nodes (an approximation of 15 eNBs) with maximum throughput is summarised in table 8.

$$signals/sec_{case} = ops/sec \cdot P_{case} \cdot signals/case \tag{3}$$

Where P_{case} is the designated probability of a case, see table 7.

$$UE/h = \frac{\sum signals/sec_{case} \cdot 3600 sec/hour}{480 signals/UE/hour} \tag{4}$$

Table 8: Maximum theoretical count of supported UE with 5 machines, 15 serving nodes (simulated 15 eNB).

Table 8: supported UE count

Region type	**Throughput**	**Signals**	**UE / h**	**UE / eNB**
Partitioned, delta	110'739 ops/s	1'278'481 sig/s	9'588'613	639'241
Partitioned, no delta	128'108 ops/s	1'479'006 sig/s	11'092'551	739'503
Replicated, delta	37'652 ops/s	434'692 sig/s	3'260'192	217'346
Replicated, no delta	43'864 ops/s	506'409 sig/s	3'798'074	253'205
HAC, delta	14'526 ops/s	167'702 sig/s	1'257'770	83'851

As a part of their study, Jin et al. collected traces from a large ISP's LTE network [25]. The traces indicate that the maximum number of active users per base station was close to, but under 1000 UE. Contrasting to our results, if the measured throughput is actually attainable, the Apache Geode data store could in principle support several orders of magnitude more connected devices. Even the lowest result achieved in our programmatic HAC-zoning with a modest 15 simulated eNBs could in theory support up to 83'851 UE per eNB, over 80 times more active UEs than in the aforementioned report.

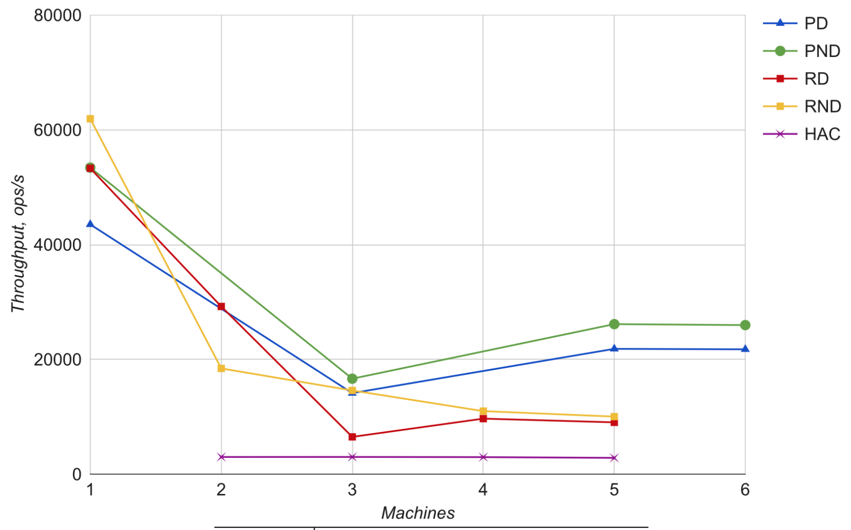
The limiting factor is therefore the latency with which data can be read from the storage. The latency results indicate that at least the partitioned region with 99th-percentile read-write latencies of 1 μs could deliver data fast enough. Even other region types could be leveraged in the right configuration to deliver sufficiently low latencies. It should be noted that the EPC controller needs to be able to read the data store faster than write to it, since it is more important for it to be able to set up the appropriate context in its process space. As we didn't measure read and write latencies separately it is difficult to draw clear conclusions on whether our implementations could provide sufficiently low read latencies even though the results indicate very low microsecond-scale latencies.

With these figures in mind, we can conclude that with the right optimisations and configurations applied in the data layer the scoped 100–1000-fold increase in connected devices could be achieved and perhaps even surpassed. Furthermore, should the interconnect behind the coreless mobile network architecture be flat and connection to the Internet unhindered, we postulate that the chances of reaching the scoped 10–100 data rate increase and 5-fold decrease in latency cloud be markedly improved.

7.2 Scaling Apache Geode

The throughput achieved at two serving nodes per machine is plotted in figure 19. It can be seen that all regions (not including HAC-zoning) start at roughly the same level of throughput. The partitioned regions experience a major drop off in throughput when the number of machines is increased to three, but rises and seems to stay relatively stable thereafter. The replicated region expectedly experiences incrementally dropping throughput with the addition of more machines to the cluster. The experimental HAC-zone is almost constant and we expect its throughput to be eventually bottlenecked by the performance of replicated regions.

Figure 19: Throughput per machine with two serving nodes per machine.



Label	Expanded label
PD	Partitioned region, Delta
PND	Partitioned region, No delta
RD	Replicated region, Delta
RND	Replicated region, No delta
HAC	HAC-zoning, Delta

How exactly the throughput scales beyond the observed is left to another experimentation, but we can make some predictions on its evolution. Taking into consideration the structure of updates in partitioned regions we can assume that the throughput that is reached at five machines will stay relatively constant, if not improve. The throughput per machine for the partitioned region is 21'842 ops/s and 26'149 ops/s for the delta and regular updates respectively. Assuming the throughput stays constant we can expect the system to reach 1'000'000 ops/s with 46 and 38 machines with delta and regular updates respectively.

For the replicated regions the behaviour is less predictable. We can certainly say that with the addition of new machines into the cluster the throughput will decrease incrementally. This is also evident from the aggregate throughput detailed in section 6.3.3. The throughput would seem to plateau at four machines, but we predict that this will not be the observed behaviour in the long run. In fact, taking into consideration the update mechanism in replicated regions (with the *distributed-ack* or global locking) we postulate that increasing the cluster size indefinitely will eventually render the system quite unusable. The size of replicated regions in deployment will therefore need careful consideration of required throughput and redundancy.

In our benchmarks we did not address the issues of server failure, serving node crash or network churn. Apache Geode has been built specifically around high availability and strong consistency, which were seen as the most important characteristics regarding the coreless mobile network architecture. We assumed that the operator network would be relatively stable and contain redundant paths, and that the deployment of a data store would contain more than enough redundant nodes for data storage. However, as mentioned, it is an optimisation problem to decide on the trade-off between excessive redundancy and high availability (and ultimately reliability). Figure 19 features a case with two serving nodes per physical machine. Assuming that the deployment scenario requires that each machine or smart-box site must contain at least one local serving node for maximum throughput and minimum latency, one could lose one of the two serving nodes and still function normally without seeing much effect on throughput or latency. It would critically important, however, to restart the serving node as fast as possible to resume normal operation because the loss of two serving nodes could be catastrophic. The amount of serving nodes could be increased to begin with, but again we return to the questions on optimisation.

7.3 Update mechanisms: delta versus regular

We explored the effect of delta updates on the throughput and latency of the Apache Geode data store. The delta update was implemented with the Geode internal *Delta*-interface according to recommendations in the documentation. The results in section 6.3 detail the difference and show that the delta update mechanism is consistently performing worse than the regular mechanism, where the entire object is sent over the network. The result seems surprising at first, but at a closer inspection is understandable.

The result is highly likely due to the processing overhead of the delta update mechanism and our small UE object: the entire object with all variables in use takes only 5'152 bits plus a small amount of overhead when serialised. Assuming that the serialised object takes 6'000 bits, then it can be transmitted in $6000 bits \div 1500 bits/packet = 4 packets$, with a standard maximum transmission unit of 1'500 bits omitting TCP and UDP overheads. The network connecting the machines in our experiments consisted of a single switch, two network hops. This has the effect that by the time a host starts sending the last packet of the four, the receiving host is already receiving the first three. Thus the total time to transmit all four packets is roughly the same as the time to transmit two single packets back-to-back. On the other hand, the vast majority of delta updates are less than 1'280 bits in size, fitting into a single packet to be sent over the network. The difference in time required to transmit the updates is small, almost negligible. Should the object be of much larger size overall while keeping small delta updates, the time to update via the regular mechanism would increase substantially as more bytes need to be sent over the network. Therefore in our experiment, even with updates sent over the network the time difference is not substantial enough to set the delta update mechanism far ahead. If the client is connected to a serving node on the local machine the difference is even smaller.

The delta update functions as follows. The first stage is the same for both delta and regular updates where the delta or the object is serialised and sent to the connected serving node. Geode then deserialises the serialised data in the serving node, calls the appropriate *fromDelta* method of the UE object to deserialise the serialised delta update, and finally re-serialises the updated object back into the data store. On the other hand the regular update mechanism only takes the received serialised object and replaces the old object with it. It is clear that there is significantly more processing delay in applying the delta update than the regular update, and this is likely causing the drop in throughput. It is important to note, however, that delta updates use potentially much less bandwidth (up to 75% less in our implementation) than non-delta update mechanism, allowing the network bandwidth to be used for other traffic. The decision to use either update mechanism needs to consider the deployment scenario and the network usage versus throughput desired of the system.

7.4 High availability and consistency zone implementations

Our experimental implementation of HAC-zoning suffered greatly from artefacts in the benchmark implementation. In the short period of time it was not possible to further study the more advanced features of Apache Geode. For example, Geode supports executing custom functions server side, in asynchronous or synchronous manners. Such functions could be leveraged to implement moving data between HAC-zones *behind-the-scenes*, and not explicitly in client code. Other useful features include moving data buckets and single data items explicitly between serving nodes, allowing for a fine grained control of data geo-location.

We implemented the experimental HAC-zoning as a replicated region combined with a partitioned region. In principle the HAC-zone need not be this type of region. For example it is possible to achieve high availability and consistency in the partitioned region type, and if network churn presents a significant problem the replication factor of the partitioned region can be configured higher. Nevertheless, there are multiple points available for optimisation even just in the case of Apache Geode.

7.5 Reliability of reported latencies

In our benchmarking we attempted to map the scaling characteristics of the Geode data store by i) involving incremental numbers of serving nodes, and ii) involving incremental numbers of physical machines in the Geode cluster. However, due to infrastructure concerns we were inadvertently restricted to at most seven physical machines. Additionally, the network infrastructure available in our experimentation was very stable, had no other load and the machines were within two switch hops of each other. This presents a serious limitation in predicting the full scaling characteristics of the system beyond our setup. It is difficult to predict how the system performs with hundreds or thousands of machines when network conditions are less than optimal. Therefore, it suffices to say that, while the results are definitely promising they are also inconclusive in the grand scheme.

The *Yahoo!* cloud serving benchmark has been taken seriously as an industry leading benchmark platform. It has also received critique regarding its implementation. In his blog, Wakart presents compelling evidence of the benchmarks' problem with coordinated omission.[48] Coordinated omission is a common problem relating to benchmark implementation, where some results are omitted from results. The phenomenon occurs when the benchmarking system experiences a stop-the-world event, such as garbage collection in Java programs. During garbage collection, which may last a comparatively long time, no actual processing is done even though in a real scenario requests would be made to the system under test. If not accounted for the benchmark system will record only a single instance of a large latency when in fact there would have been many. This significantly skews the end result of the benchmark.[49]

The effect of the coordinated omission is mitigated in our experimentation for two reasons: i) we tested the data store according to the maximum throughput setting, and ii) YCSB has received corrections to the code to mitigate the effect to an extent.[50] As said, our measurements were completed with maximising load and observing the achieved throughput. In this mode of operation the effect of stop-the-world events, while not absent, is minimal.

7.6 Accuracy of used traffic model

Some time into the benchmarking we came across another view of signalling load data, also by Nokia.[51] According to the report, which quotes data from a service provider in USA, the probabilities of a case occurrence are somewhat different. Specifically, the occurrence of session management and paging operations are significantly higher and the load from handover is significantly lower than in the data used to construct our workload. However, there is some room for interpretation in the new source: it does not explicitly state the service request and lu-release procedures that were included in the original, see figure 7; should said operations be included in the session management, then the picture is already much closer to our

[48]`http://psy-lob-saw.blogspot.fi/2015/03/fixing-ycsb-coordinated-omission.html` *(Referenced on 2.11.216)*

[49]`https://www.infoq.com/presentations/latency-pitfalls` *Referenced on 11.11.2016)*

[50]`https://github.com/brianfrankcooper/YCSB/blob/master/core/CHANGES.md` *(Referenced on 11.11.2016)*

[51]`https://insight.nokia.com/managing-lte-core-network-signaling-traffic` *(Referenced on 11.11.2016)*

original data. It is evident, that obtaining current data which accurately describes real world scenarios is difficult.

Lastly, despite the standard interfaces the implementation of any 4G/LTE system is largely vendor dependent, and the same will apply to 5G systems. Exactly how much space is required by a UE object and what form it will take is ultimately dependent on the implementation.

8 Concluding remarks

In this thesis we have presented the current state of the 4G/LTE mobile telecommunications network and key concepts in UE connectivity. Based on the technology, a novel network architecture was proposed. The coreless mobile network architecture was detailed, and its core properties analysed in the context of the requirements for the next generation 5G mobile networks. As state management was identified as the main purpose of the EPC entities, we presented and analysed data storage solutions that may be capable of hosting the state in the new architecture.

8.1 Contributions

By analysis of the current 4G/LTE architecture we realised that by virtualising the EPC entities and by bringing them into the same environment they could function much like an SDN controller. To make the envisioned controller scalable the UE context state needed to be separated from the entities into a ubiquitous data layer. This thesis thereafter focused on the data layer distribution. We analysed the requirements of the smart-box and based on the requirements we proposed that the Apache Geode data store could be leveraged as an example data storage solution. Finally, we benchmarked the Apache Geode data store with workloads attempting to model a real world scenario. The results indicate that depending on its configuration, Geode can indeed support the throughput and latency needed of the data layer.

8.2 Future work

While the results of benchmarking Geode are promising there is much room for optimisation. Of relevance is the final latency with which data needs to be available to the smart-box, and the latency at which any given configuration of a distributed data store can deliver data. Explicitly measuring latency of data accessed over the network would bring further insights into the latencies observed in our experimentation. We attempted to model some scaling properties but ultimately we were lacking in infrastructure, and as such the final scaling characteristics are yet to be discerned. Characterising the full scaling properties would require machines in the hundreds or even thousands, a feat that often is accomplished only at the deployment stage. Regarding Geode it is also of importance to discover more intricate possibilities of region combination into HAC-zoning.

Exactly how the smart-box or EPC controller is realised was beyond the scope of this thesis. There has been some research toward software defined control of the EPC which may be leveraged in this respect. A related topic is the propagation of flow rules into the interconnect or operator backbone network. For example, how deep into the backbone network are new flow rules installed, should there be regional master controllers?

Mobile edge computing may provide unprecedented throughput and latency for applications residing on the edge of the mobile network connecting with UE. Caching at the edge, for example, has been seen as one major possibility to reduce load in the core network. It is possible that the data layer used in hosting the UE context state could be used also in this respect, in caching and storing other data besides the context. The viability of using the data layer for more purposes should be researched.

Notes

1 `https://www.ericsson.com/news/1925907` *(Referenced on 28.10.2016)* . 1
2 `http://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts/` *(Referenced on 8.11.2016)* 1
3 `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html` *(Referenced on 28.10.2016)* 1
4 `http://resources.alcatel-lucent.com/asset/200004` *(Referenced on 28.10.2016)* 1
5 `http://www.4gamericas.org/files/8914/6774/6748/Global_Organizations_Forge_New_Frontier_of_5G_Final.pdf` *(Referenced on 28.10.2016)* . . . 3
6 `http://www.chinaeu.eu/wp-content/uploads/2016/04/mazhigang-5G-Progress-and-Cooperation-in-China-20160408.pdf` *(Referenced on 28.10.2016)* 3
7 `http://www.5gforum.org` *(Referenced on 28.10.2016)* 3
8 `http://5gmf.jp/en/` *(Referenced on 28.10.2016)* 3
9 `https://5g-ppp.eu/white-papers/` *(Referenced on 28.10.2016)* 3
10 Figure adapted from `https://en.wikipedia.org/wiki/System_Architecture_Evolution`, licence CC BY-SA 4.0, *(Referenced on 28.10.2016)* 5
11 `https://en.wikipedia.org/wiki/System_Architecture_Evolution` *(Referenced on 28.10.2016) (Referenced on 28.10.2016)* 7
12 Eleven EMM Cases in an EMM Scenario: `http://www.netmanias.com/en/?m=view&id=techdocs&no=6002` *(Referenced on 28.10.2016)* 7
13 `http://www.netmanias.com/en/?m=view&id=techdocs&no=10441&page=2` *(Referenced on 28.10.2016)* 9
14 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6108` *(Referenced on 28.10.2016)* 11
15 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6110` *(Referenced on 28.10.2016)* 12
16 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6134` *(Referenced on 28.10.2016)* 13
17 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6193` *(Referenced on 28.10.2016)* 15
18 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6324` *(Referenced on 28.10.2016)* 15
19 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6224` *(Referenced on 28.10.2016)* 16
20 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6257` *(Referenced on 28.10.2016)* 16
21 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6286` *(Referenced on 28.10.2016)* 16
22 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6322` *(Referenced on 28.10.2016)* 17
23 `http://www.netmanias.com/en/?m=view&id=techdocs&no=6324` *(Referenced on 28.10.2016)* 17
24 `https://www.opennetworking.org/sdn-resources/openflow` *(Referenced on 28.10.2016)* 18
25 `https://code.facebook.com/posts/1653074404941839/under-the-hood-broadcasting-live-video-to-millions/` *(Referenced 15.11.2016)* . . . 20

References

[1] ONF WHITE PAPER on software-defined networking: The new norm for networks. Whitepaper, Open Networking Foundation, 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[2] Abdelwahab, S., Hamdaoui, B., Guizani, M., and Znati, T. Network function virtualization in 5g. *IEEE Communications Magazine 54*, 4 (2016), 84–91.

[3] Alcatel-Lucent. The lte network architecture, 2009. A comprehensive tutorial.

[4] Alcatel-Lucent. Managing the signaling traffic in packet core, 2013. METHODS TO REDUCE THE SIGNALING LOAD IN 3G/LTE NETWORKS.

[5] Alsedairy, T., Qi, Y., Imran, A., Imran, M. A., and Evans, B. Self organising cloud cells: a resource efficient network densification strategy. *Transactions on Emerging Telecommunications Technologies 26*, 8 (2015), 1096–1107.

[6] Blenk, A., Basta, A., Reisslein, M., and Kellerer, W. Survey on network virtualization hypervisors for software defined networking. *IEEE Communications Surveys & Tutorials 18*, 1 (2016), 655–685.

[7] Casas, P., D'Alconzo, A., Fiadino, P., Bär, A., Finamore, A., and Zseby, T. When youtube does not work—analysis of qoe-relevant degradation in google cdn traffic. *IEEE Transactions on Network and Service Management 11*, 4 (2014), 441–457.

[8] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS) 26*, 2 (2008), 4.

[9] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 143–154.

[10] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS) 31*, 3 (2013), 8.

[11] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review 41*, 6 (2007), 205–220.

[12] DeWitt, D. J. The wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook*. 1991, pp. 119–165.

[13] ETSI, T. 123 003. *Digital cellular telecommunications system (Phase 2+).*

[14] ETSI, T. 123 008. *Digital cellular telecommunications system (Phase 2+).*

[15] ETSI, T. 123 401. *Digital cellular telecommunications system (Phase 2+).*

[16] Ford, A., Raiciu, C., Handley, M., and Bonaventure, O. Tcp extensions for multipath operation with multiple addresses. Tech. rep., 2013.

[17] George, L. *HBase: the definitive guide*. O'Reilly Media, Inc., 2011.

[18] Gilbert, S., and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News 33*, 2 (2002), 51–59.

[19] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review 35*, 5 (2005), 41–54.

[20] Hämäläinen, S., Sanneck, H., and Sartori, C. *LTE self-organising networks (SON): network management automation for operational efficiency*. John Wiley & Sons, 2012.

[21] He, J., and Song, W. Evolving to 5g: A fast and near-optimal request routing protocol for mobile core networks. In *2014 IEEE Global Communications Conference* (2014), IEEE, pp. 4586–4591.

[22] Hossain, E., and Hasan, M. 5g cellular: key enabling technologies and research challenges. *IEEE Instrumentation & Measurement Magazine 18*, 3 (2015), 11–21.

[23] Hossain, E., Rasti, M., Tabassum, H., and Abdelnasser, A. Evolution toward 5g multi-tier cellular wireless networks: An interference management perspective. *IEEE Wireless Communications 21*, 3 (2014), 118–127.

[24] Jain, R., and Paul, S. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine 51*, 11 (2013), 24–31.

[25] Jin, X., Li, L. E., Vanbever, L., and Rexford, J. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies* (2013), ACM, pp. 163–174.

[26] Kantola, R. *Performance of Handover in Long Term Evolution*. PhD thesis, Aalto University, 2011.

[27] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (1997), ACM, pp. 654–663.

[28] Kempf, J., Johansson, B., Pettersson, S., Lüning, H., and Nilsson, T. Moving the mobile evolved packet core to the cloud. In *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (2012), IEEE, pp. 784–791.

[29] Khan, F. Coreless 5g mobile network. *arXiv preprint arXiv:1508.02052* (2015).

[30] Kuhlenkamp, J., Klems, M., and Röss, O. Benchmarking scalability and elasticity of distributed database systems. *Proceedings of the VLDB Endowment 7*, 12 (2014), 1219–1230.

[31] Lakshman, A., and Malik, P. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review 44*, 2 (2010), 35–40.

[32] Liang, C., and Yu, F. R. Wireless network virtualization: A survey, some research issues and challenges. *IEEE Communications Surveys & Tutorials 17*, 1 (2015), 358–380.

[33] Liang, C., Yu, F. R., and Zhang, X. Information-centric network function virtualization over 5g mobile wireless networks. *IEEE Network 29*, 3 (2015), 68–74.

[34] Liu, Q., Wu, G., Guo, Y., and Zhang, Y. Energy efficient resource allocation for control data separation architecture based h-cran with heterogeneous fronthaul. *arXiv preprint arXiv:1511.01969* (2015).

[35] Mäki, J. O. A., et al. *Distributed OpenFlow Control-Plane for Mobile Network Gateways*. PhD thesis, University of Helsinki, 2013.

[36] Marotta, M. A., Kaminski, N., Gomez-Miguelez, I., Granville, L. Z., Rochol, J., DaSilva, L., and Both, C. B. Resource sharing in heterogeneous cloud radio access networks. *IEEE Wireless Communications 22*, 3 (2015), 74–82.

[37] Mehani, O., Holz, R., Ferlin, S., and Boreli, R. An early look at multipath tcp deployment in the wild. In *Proceedings of the 6th International Workshop on Hot Topics in Planet-Scale Measurement* (2015), ACM, pp. 7–12.

[38] Nygren, E., Sitaraman, R. K., and Sun, J. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[39] Orange, J. S.-B., Armada, A. G., Evans, B., Galis, A., and Karl, H. White paper for research beyond 5g. *Accessed 23* (2015), 2016.

[40] Osmani, L., Lindholm, H., Chemmagate, B., Rao, A., Tarkoma, S., Heinonen, J., and Flinck, H. Building blocks for an elastic mobile core. In *Proceedings of the 2014 CoNEXT on Student Workshop* (2014), ACM, pp. 43–45.

[41] Osseiran, A., Boccardi, F., Braun, V., Kusume, K., Marsch, P., Maternia, M., Queseth, O., Schellmann, M., Schotten, H., Taoka, H., et al. Scenarios for 5g mobile and wireless communications: the vision of the metis project. *IEEE Communications Magazine 52*, 5 (2014), 26–35.

[42] Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A., et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative* (2014).

[43] Peng, M., Wang, C., Lau, V., and Poor, H. V. Fronthaul-constrained cloud radio access networks: Insights and challenges. *IEEE Wireless Communications 22*, 2 (2015), 152–160.

[44] Pentikousis, K., Wang, Y., and Hu, W. Mobileflow: toward software-defined mobile networks. *IEEE Communications magazine 51*, 7 (2013), 44–53.

[45] Pirinen, P. A brief overview of 5g research activities. In *5G for Ubiquitous Connectivity (5GU), 2014 1st International Conference on* (2014), IEEE, pp. 17–22.

[46] Qazi, Z. A., Penumarthi, P. K., Sekar, V., Gopalakrishnan, V., Joshi, K., and Das, S. R. Klein: A minimally disruptive design for an elastic cellular core. In *Proceedings of the Symposium on SDN Research* (New York, NY, USA, 2016), SOSR '16, ACM, pp. 2:1–2:12.

[47] Rost, P., Bernardos, C. J., De Domenico, A., Di Girolamo, M., Lalam, M., Maeder, A., Sabella, D., and Wübben, D. Cloud technologies for flexible 5g radio access networks. *IEEE Communications Magazine 52*, 5 (2014), 68–76.

[48] Sabella, D., Rost, P., Sheng, Y., Pateromichelakis, E., Salim, U., Guitton-Ouhamou, P., Di Girolamo, M., and Giuliani, G. Ran as a service: Challenges of designing a flexible ran architecture in a cloud-based heterogeneous mobile network. In *Future Network and Mobile Summit (FutureNetworkSummit), 2013* (2013), IEEE, pp. 1–8.

[49] Sama, M. R., Contreras, L. M., Kaippallimalil, J., Akiyoshi, I., Qian, H., and Ni, H. Software-defined control of the virtualized mobile packet core. *IEEE Communications Magazine 53*, 2 (2015), 107–115.

[50] Sato, I., Bouabdallah, A., and Lagrange, X. Improving lte/epc signaling for sporadic data with a control-plane based transmission procedure. In *Wireless Personal Multimedia Communications (WPMC), 2011 14th International Symposium on* (2011), IEEE, pp. 1–5.

[51] Sesia, S., Toufik, I., and Baker, M. *LTE - The UMTS long term evolution.* Wiley Online Library, 2015.

[52] Talwar, S., Choudhury, D., Dimou, K., Aryafar, E., Bangerter, B., and Stewart, K. Enabling technologies and architectures for 5g wireless. In *2014 IEEE MTT-S International Microwave Symposium (IMS2014)* (2014), IEEE, pp. 1–4.

[53] Tu, G.-H., Li, Y., Peng, C., Li, C.-Y., and Lu, S. Detecting problematic control-plane protocol interactions in mobile networks. *IEEE/ACM Transactions on Networking 24*, 2 (2016), 1209–1222.

[54] Wang, C.-X., Haider, F., Gao, X., You, X.-H., Yang, Y., Yuan, D., Aggoune, H. M., Haas, H., Fletcher, S., and Hepsaydir, E. Cellular architecture and key technologies for 5g wireless communication networks. *IEEE Communications Magazine 52*, 2 (2014), 122–130.

[55] Wang, X., Chen, M., Taleb, T., Ksentini, A., and Leung, V. C. Cache in the air: exploiting content caching and delivery techniques for 5g systems. *IEEE Communications Magazine 52*, 2 (2014), 131–139.

[56] Widjaja, I., Bosch, P., and La Roche, H. Comparison of mme signaling loads for long-term-evolution architectures. In *Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th* (2009), IEEE, pp. 1–5.

[57] Yan, Q., Yu, F. R., Gong, Q., and Li, J. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials 18*, 1 (2016), 602–622.

[58] Yousaf, F. Z., and Taleb, T. Fine-grained resource-aware virtual network function management for 5g carrier cloud. *IEEE Network 30*, 2 (2016), 110–115.

A LTEworkload

```
# Copyright (c) 2012-2016 YCSB contributors. All rights reserved
    .
#
# Licensed under the Apache License, Version 2.0 (the "License")
    ; you
# may not use this file except in compliance with the License.
    You
# may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
    software
# distributed under the License is distributed on an "AS IS"
    BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
    or
# implied. See the License for the specific language governing
# permissions and limitations under the License. See
    accompanying
# LICENSE file.

# Yahoo! Cloud System Benchmark
# Workload Template: Default Values
#
# File contains all properties that can be set to define a
# YCSB session. All properties are set to their default
# value if one exists. If not, the property is commented
# out. When a property has a finite number of settings,
# the default is enabled and the alternates are shown in
# comments below it.
#
# Use of most explained through comments in Client.java or
# CoreWorkload.java or on the YCSB wiki page:
# https://github.com/brianfrankcooper/YCSB/wiki/Core-Properties

# The name of the workload class to use
workload=com.yahoo.ycsb.workloads.GeodeWorkload

# There is no default setting for recordcount but it is
# required to be set.
# The number of records in the table to be inserted in
# the load phase or the number of records already in the
# table before the run phase.
recordcount=1000000

# There is no default setting for operationcount but it is
# required to be set.
# The number of operations to use during the run phase.
operationcount=3000000

# The offset of the first insertion
```

```
insertstart=0

# The distribution used to choose the length of a field
fieldlengthdistribution=constant
#fieldlengthdistribution=uniform
#fieldlengthdistribution=zipfian

insertproportion=0
attach_proportion=0.01
detach_proportion=0.01
service_request_proportion=0.31
s1_release_proportion=0.31
tau_proportion=0.09
handover_proportion=0.12
cell_reselect_proportion=0.07
session_management_proportion=0.07

geode.locator=localhost[10334]

# The name of the database table to run queries against
table.name=usertable

# How the latency measurements are presented
measurmenttype=hdrhistogram+histogram
measurment.interval=both
hdrhistogram.fileoutput=true
hdrhistogram.output.path=hdrhistogram_
```

B UE.java

```
package com.yahoo.ycsb.workloads;
/**
 * Created by frojala on 19/08/16.
 */
import com.gemstone.gemfire.InvalidDeltaException;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

//public class UE implements Serializable {
public class UE implements com.gemstone.gemfire.Delta,
    Serializable {
  /**
   * Static entity codes, will never change in this benchmark.
   */
  private static final String MNC = "244";
  private static final String MCC = "921";
  private static final String PLMN_ID = MNC + MCC;
  private static final String MMEGI = "M";
  private static final String MMEC = "C";
  private static final String MMEI = MMEGI + MMEC;
  private static final String GUMMEI = PLMN_ID + MMEI;
  public static final String PGW_ID = "ubiquitous.internet";
  private static final int TAI_SIZE = 15;

  /**
   * Semi-static entity codes. Once generated on insert, will
      not be changed.
   */
  private String IMEI;
  private String MSIN;
  private String IMSI; // = PLMN_ID + MSIN;
  private String EPS_KEY; // = 128 bit (8 chars)
  private String Cipher_KEY; // = 128 bit
  private String Encryption_KEY; // = 128 bit

  /**
   * Dynamic entities, will experience change on a regular
      basis.
   */
  private byte status; // 0 = Detached, 1 = Attached, 2 = active
     , 3 = idle;
  private int M_TMSI;
  private String GUTI; // = GUMMEI + M_TMSI;
  private String TIN;
  private int IP;
  private short C_RNTI;
  private int eNB_UE_S1AP;
```

```
    private int MME_UE_S1AP;
    private int OLD_eNB_UE_X2;
    private int NEW_eNB_UE_X2;
    private int ECI;
    private String ECGI; // = PLMN_ID + ECI;
    private int TAI;
    private List<Integer> TAI_list;
    private String PDN_ID;
    private byte EPS_bearer;
    private byte E_RA_bearer;
    private byte DR_bearer;
    private int S1_TEID_UL;
    private int S1_TEID_DL;
    private int S5_TEID_UL;
    private int S5_TEID_DL;
    private String K_ASME; // = 256 bit (16 chars)
    private String K_ENB; // = 256 bit
    private String K_NASint; // = 256 bit
    private String K_NASenc; // = 256 bit
    private String K_RRCint; // = 256 bit
    private String K_RRCenc; // = 256 bit
    private String K_UPenc; // = 256 bit

    /**
     * The transients to support deltas.
     */
    private transient boolean master_ch;
    private transient boolean status_ch;
    private transient boolean M_TMSI_ch;
    private transient boolean GUTI_ch;
    private transient boolean TIN_ch;
    private transient boolean IP_ch;
    private transient boolean C_RNTI_ch;
    private transient boolean eNB_UE_S1AP_ch;
    private transient boolean MME_YE_S1AP_ch;
    private transient boolean OLD_eNB_UE_X2_ch;
    private transient boolean NEW_eNB_UE_X2_ch;
    private transient boolean ECI_ch;
    private transient boolean ECGI_ch;
    private transient boolean TAI_ch;
    private transient boolean TAI_list_ch;
    private transient boolean PDN_ID_ch;
    private transient boolean EPS_bearer_ch;
    private transient boolean E_RA_bearer_ch;
    private transient boolean DR_bearer_ch;
    private transient boolean S1_TEID_UL_ch;
    private transient boolean S1_TEID_DL_ch;
    private transient boolean S5_TEID_UL_ch;
    private transient boolean S5_TEID_DL_ch;
    private transient boolean K_ASME_ch;
    private transient boolean K_ENB_ch;
    private transient boolean K_NASint_ch;
    private transient boolean K_NASenc_ch;
    private transient boolean K_RRCint_ch;
```

```
  private transient boolean K_RRCenc_ch;
  private transient boolean K_UPenc_ch;

  public UE(String IMSI) {
    Random rand = new Random();
    this.IMEI = randomString(15, rand);
    this.IMSI = IMSI;
    this.MSIN = IMSI.substring(6);
    this.EPS_KEY = randomString(8, rand);
    this.Cipher_KEY = randomString(8, rand);
    this.Encryption_KEY = randomString(8, rand);
    this.status = 0; // default detached
  }

  public void initial_attach() {
    Random rand = new Random();
    this.status = 2;  /* Active */
                                                  this.status_ch = true;
    this.M_TMSI = rand.nextInt();
                                                   this.M_TMSI_ch = true;
    this.GUTI = GUMMEI + M_TMSI;
                                                    this.GUTI_ch = true;
    this.TIN = "";
                                                                  this.
        TIN_ch = true;
    this.IP = rand.nextInt();
                                                       this.IP_ch = true;
    this.C_RNTI = (short) rand.nextInt(Short.MAX_VALUE - 1);
             this.C_RNTI_ch = true;
    this.eNB_UE_S1AP = rand.nextInt();
                                             this.eNB_UE_S1AP_ch = true;
    this.MME_UE_S1AP = rand.nextInt();
                                             this.MME_YE_S1AP_ch = true;
    this.ECI = rand.nextInt();
                                                     this.ECI_ch = true;
    this.ECGI = PLMN_ID + ECI;
                                                     this.ECGI_ch = true;
    this.TAI = rand.nextInt();
                                                     this.TAI_ch = true;
    this.TAI_list = new ArrayList<>();
                                             this.TAI_list_ch = true;
    for (int i = 0; i < TAI_SIZE; i++) TAI_list.add(rand.nextInt
        ());
    this.PDN_ID = rand.nextInt() + ".apn.epc.mnc" + MNC + ".mcc"
        + MCC + rand.nextInt() + "3gppnetwork.org";
    this.PDN_ID_ch = true;
    this.EPS_bearer = (byte) rand.nextInt(Byte.MAX_VALUE - 1);
           this.EPS_bearer_ch = true;
    this.E_RA_bearer = (byte) rand.nextInt(Byte.MAX_VALUE - 1);
           this.E_RA_bearer_ch = true;
    this.DR_bearer = (byte) rand.nextInt(Byte.MAX_VALUE - 1);
            this.DR_bearer_ch = true;
    this.S1_TEID_UL = rand.nextInt();
                                              this.S1_TEID_UL_ch = true;
    this.S1_TEID_DL = rand.nextInt();
                                 this.S1_TEID_DL_ch = true;
    this.S5_TEID_UL = rand.nextInt();
                                 this.S5_TEID_UL_ch = true;
    this.S5_TEID_DL = rand.nextInt();
                                 this.S5_TEID_DL_ch = true;
    this.K_ASME = randomString(16, rand);
                               this.K_ASME_ch = true;
    this.K_ENB = randomString(16, rand);
                               this.K_ENB_ch = true;
    this.K_NASint = randomString(16, rand);
                              this.K_NASint_ch = true;
    this.K_NASenc = randomString(16, rand);
                              this.K_NASenc_ch = true;
    this.K_RRCint = randomString(16, rand);
                              this.K_RRCint_ch = true;
    this.K_RRCenc = randomString(16, rand);
                              this.K_RRCenc_ch = true;
    this.K_UPenc = randomString(16, rand);
                               this.K_UPenc_ch = true;
    this.master_ch = true;
  }

  public void detach() {
    this.status = 0; /* Detached */                this.
        status_ch = true;
    this.IP = 0;                                   this.IP_ch
        = true;
    this.C_RNTI = 0;                               this.
        C_RNTI_ch = true;
    this.eNB_UE_S1AP = 0;                          this.
        eNB_UE_S1AP_ch = true;
    this.MME_UE_S1AP = 0;                          this.
        MME_YE_S1AP_ch = true;
    this.ECI = 0;                                  this.ECI_ch
        = true;
    this.ECGI = PLMN_ID + ECI;                     this.
        ECGI_ch = true;
    this.PDN_ID = "";                              this.
        PDN_ID_ch = true;
    this.EPS_bearer = 0;                           this.
        EPS_bearer_ch = true;
    this.E_RA_bearer = 0;                          this.
        E_RA_bearer_ch = true;
    this.DR_bearer = 0;                            this.
        DR_bearer_ch = true;
    this.S1_TEID_UL = 0;                           this.
        S1_TEID_UL_ch = true;
    this.S1_TEID_DL = 0;                           this.
        S1_TEID_DL_ch = true;
    this.S5_TEID_UL = 0;                           this.
        S5_TEID_UL_ch = true;
    this.S5_TEID_DL = 0;                           this.
        S5_TEID_DL_ch = true;
    this.K_ASME = "";                    this.K_ASME_ch = true;
    this.K_ENB = "";                     this.K_ENB_ch = true;
    this.K_RRCint = "";                  this.K_RRCint_ch = true;
    this.K_RRCenc = "";                  this.K_RRCenc_ch = true;
    this.K_UPenc = "";                   this.K_UPenc_ch = true;
    this.master_ch = true;
  }

  public void S1_release() {
    this.status = 3; /* Idle */          this.status_ch = true;
    this.C_RNTI = 0;                     this.C_RNTI_ch = true;
    this.eNB_UE_S1AP = 0;                this.eNB_UE_S1AP_ch = true;
    this.MME_UE_S1AP = 0;                this.MME_YE_S1AP_ch = true;
    this.ECI = 0;                        this.ECI_ch = true;
    this.ECGI = PLMN_ID + ECI;           this.ECGI_ch = true;
    this.DR_bearer = 0;                  this.DR_bearer_ch = true;
    this.S1_TEID_DL = 0;                 this.S1_TEID_DL_ch = true;
    this.K_ENB = "";                     this.K_ENB_ch = true;
    this.K_RRCint = "";                  this.K_RRCint_ch = true;
    this.K_RRCenc = "";                  this.K_RRCenc_ch = true;
    this.K_UPenc = "";                   this.K_UPenc_ch = true;
    this.master_ch = true;
  }

  public void service_request() {
    Random rand = new Random();
    this.status = 2; /* Active */        this.status_ch = true;
    this.C_RNTI = (short) rand.nextInt(Short.MAX_VALUE - 1);    this.C_RNTI_ch = true;
    this.eNB_UE_S1AP = rand.nextInt();   this.eNB_UE_S1AP_ch = true;
    this.MME_UE_S1AP = rand.nextInt();   this.MME_YE_S1AP_ch = true;
    this.ECI = rand.nextInt();           this.ECI_ch = true;
    this.ECGI = PLMN_ID + ECI;
```

```
                                          this.ECGI_ch = true;
    this.DR_bearer = (byte) rand.nextInt(Byte.MAX_VALUE - 1);
            this.DR_bearer_ch = true;
    this.S1_TEID_DL = rand.nextInt();
                                   this.S1_TEID_DL_ch = true;
    this.K_ENB = randomString(16, rand);
                                this.K_ENB_ch = true;
    this.K_RRCint = randomString(16, rand);
                             this.K_RRCint_ch = true;
    this.K_RRCenc = randomString(16, rand);
                             this.K_RRCenc_ch = true;
    this.K_UPenc = randomString(16, rand);
                              this.K_UPenc_ch = true;
    this.master_ch = true;
  }

  public void tracking_area_update() {
    Random rand = new Random();
    this.M_TMSI = rand.nextInt();            this.M_TMSI_ch =
        true;
    this.GUTI = GUMMEI + M_TMSI;             this.GUTI_ch = true;
    this.TIN = GUTI;                         this.TIN_ch = true;
    this.TAI = rand.nextInt();               this.TAI_ch = true;
    this.TAI_list = new ArrayList<>();       this.TAI_list_ch =
        true;
    for (int i = 0; i < TAI_SIZE; i++) { TAI_list.add(rand.
        nextInt()); }
    this.master_ch = true;
  }

  public void handover() {
    // in intra-LTE the information entites are the same as in
        X2 handover. T
    // This is a mockup, as in the deployment scenario we
        envision there to be no need for this.
    Random rand = new Random();
    this.S1_TEID_DL = rand.nextInt();
                                   this.S1_TEID_DL_ch = true;
    this.DR_bearer = (byte) rand.nextInt(Byte.MAX_VALUE - 1);
            this.DR_bearer_ch = true;
    this.ECI = rand.nextInt();
                                          this.ECI_ch = true;
    this.ECGI = PLMN_ID + ECI;
                                          this.ECGI_ch = true;
    this.C_RNTI = (short) rand.nextInt(Short.MAX_VALUE - 1);
             this.C_RNTI_ch = true;
    this.eNB_UE_S1AP = rand.nextInt();
                                  this.eNB_UE_S1AP_ch = true;
    this.K_ENB = randomString(16, rand);
                                this.K_ENB_ch = true;
    this.K_RRCint = randomString(16, rand);
                             this.K_RRCint_ch = true;
    this.K_RRCenc = randomString(16, rand);
                             this.K_RRCenc_ch = true;
    this.K_UPenc = randomString(16, rand); this.K_UPenc_ch = true;
    this.master_ch = true;
  }

  public void session_management() {
    // we mock a session management act by changing the EPS, S1 and S5 TEID for creation of new bearers.
    Random rand = new Random();
    this.EPS_bearer = (byte) rand.nextInt(Byte.MAX_VALUE - 1); this.EPS_bearer_ch = true;
    this.S1_TEID_UL = rand.nextInt(); this.S1_TEID_UL_ch = true;
    this.S1_TEID_DL = rand.nextInt(); this.S1_TEID_DL_ch = true;
    this.S5_TEID_UL = rand.nextInt(); this.S5_TEID_UL_ch = true;
    this.S5_TEID_DL = rand.nextInt(); this.S5_TEID_DL_ch = true;
    this.master_ch = true;
  }

  public void cell_reselect() {
    // nothing happens.
  }

  public int getStatus() {
    return this.status;
  }

  public String getIMSI() { return this.IMSI; }

  private String randomString(int chars, Random rand) {
    String s = "";
    String hexa = "0123456789ABCDEF";
    for (int i = 0; i < chars; i++) {
      s += hexa.charAt(rand.nextInt(16));
    }
    return s;
  }

//  @Override
  public boolean hasDelta() {
    return master_ch;
  }

//  @Override
  public void toDelta(DataOutput out) throws IOException {
    out.writeBoolean(status_ch);          if (status_ch){ out.writeByte(status); this.status_ch = false; }
    out.writeBoolean(M_TMSI_ch);          if (M_TMSI_ch) { out.writeInt(M_TMSI); this.M_TMSI_ch = false; }
    out.writeBoolean(GUTI_ch);            if (GUTI_ch) { out.writeUTF(GUTI); this.GUTI_ch = false; }
    out.writeBoolean(TIN_ch);                    if (TIN_ch) {
        out.writeUTF(TIN); this.TIN_ch = false; }
    out.writeBoolean(IP_ch);                     if (IP_ch) { out
        .writeInt(IP); this.IP_ch = false; }
    out.writeBoolean(C_RNTI_ch);                 if (C_RNTI_ch) {
         out.writeShort(C_RNTI); this.C_RNTI_ch = false; }
    out.writeBoolean(eNB_UE_S1AP_ch);            if (
        eNB_UE_S1AP_ch) { out.writeInt(eNB_UE_S1AP); this.
        eNB_UE_S1AP_ch = false; }
    out.writeBoolean(MME_YE_S1AP_ch);            if (
        MME_YE_S1AP_ch) { out.writeInt(MME_UE_S1AP); this.
        MME_YE_S1AP_ch = false; }
    out.writeBoolean(OLD_eNB_UE_X2_ch);          if (
        OLD_eNB_UE_X2_ch) { out.writeInt(OLD_eNB_UE_X2); this.
        OLD_eNB_UE_X2_ch = false; }
    out.writeBoolean(NEW_eNB_UE_X2_ch);          if (
        NEW_eNB_UE_X2_ch) { out.writeInt(NEW_eNB_UE_X2); this.
        NEW_eNB_UE_X2_ch = false; }
    out.writeBoolean(ECI_ch);                    if (ECI_ch) {
        out.writeInt(ECI); this.ECI_ch = false; }
    out.writeBoolean(ECGI_ch);                   if (ECGI_ch) {
        out.writeUTF(ECGI); this.ECGI_ch = false; }
    out.writeBoolean(TAI_ch);                    if (TAI_ch) {
        out.writeInt(TAI); this.TAI_ch = false; }
    out.writeBoolean(TAI_list_ch);               if (TAI_list_ch)
         { out.writeInt(TAI_list.size()); for (int TAI:TAI_list)
        out.writeInt(TAI); this.TAI_list_ch = false; }
    out.writeBoolean(PDN_ID_ch);                 if (PDN_ID_ch) {
         out.writeUTF(PDN_ID); this.PDN_ID_ch = false; }
    out.writeBoolean(EPS_bearer_ch);             if (
        EPS_bearer_ch) { out.writeByte(EPS_bearer); this.
        EPS_bearer_ch = false; }
    out.writeBoolean(E_RA_bearer_ch);            if (
        E_RA_bearer_ch) { out.writeByte(E_RA_bearer); this.
        E_RA_bearer_ch = false; }
    out.writeBoolean(DR_bearer_ch);              if (DR_bearer_ch
        ) { out.writeByte(DR_bearer); this.DR_bearer_ch = false;
        }
    out.writeBoolean(S1_TEID_DL_ch);             if (
        S1_TEID_DL_ch) { out.writeInt(S1_TEID_DL); this.
        S1_TEID_DL_ch = false; }
    out.writeBoolean(S1_TEID_UL_ch);             if (
        S1_TEID_UL_ch) { out.writeInt(S1_TEID_UL); this.
        S1_TEID_UL_ch = false; }
    out.writeBoolean(S5_TEID_DL_ch);             if (
        S5_TEID_DL_ch) { out.writeInt(S5_TEID_DL); this.
        S5_TEID_DL_ch = false; }
    out.writeBoolean(S5_TEID_UL_ch);             if (
        S5_TEID_UL_ch) { out.writeInt(S5_TEID_UL); this.
        S5_TEID_UL_ch = false; }
    out.writeBoolean(K_ASME_ch);                 if (K_ASME_ch) {
         out.writeUTF(K_ASME); this.K_ASME_ch = false; }
    out.writeBoolean(K_ENB_ch);                  if (K_ENB_ch) {
        out.writeUTF(K_ENB); this.K_ENB_ch = false; }
```

```java
    out.writeBoolean(K_NASint_ch);              if (K_NASint_ch)
        { out.writeUTF(K_NASint); this.K_NASint_ch = false; }
    out.writeBoolean(K_NASenc_ch);              if (K_NASenc_ch)
        { out.writeUTF(K_NASenc); this.K_NASenc_ch = false; }
    out.writeBoolean(K_RRCint_ch);              if (K_RRCint_ch)
        { out.writeUTF(K_RRCint); this.K_RRCint_ch = false; }
    out.writeBoolean(K_RRCenc_ch);              if (K_RRCenc_ch)
        { out.writeUTF(K_RRCenc); this.K_RRCenc_ch = false; }
    out.writeBoolean(K_UPenc_ch);               if (K_UPenc_ch)
        { out.writeUTF(K_UPenc); this.K_UPenc_ch = false; }
    this.master_ch = false;
  }

//  @Override
  public void fromDelta(DataInput in) throws IOException,
      InvalidDeltaException {
    if (in.readBoolean()) this.status = in.readByte();
    if (in.readBoolean()) this.M_TMSI = in.readInt();
    if (in.readBoolean()) this.GUTI = in.readUTF();
    if (in.readBoolean()) this.TIN = in.readUTF();
    if (in.readBoolean()) this.IP = in.readInt();
    if (in.readBoolean()) this.C_RNTI = in.readShort();
    if (in.readBoolean()) this.eNB_UE_S1AP = in.readInt();
    if (in.readBoolean()) this.MME_UE_S1AP = in.readInt();
    if (in.readBoolean()) this.OLD_eNB_UE_X2 = in.readInt();
    if (in.readBoolean()) this.NEW_eNB_UE_X2 = in.readInt();
    if (in.readBoolean()) this.ECI = in.readInt();
    if (in.readBoolean()) this.ECGI = in.readUTF();
    if (in.readBoolean()) this.TAI = in.readInt();
    if (in.readBoolean()) {
      int size = in.readInt();
      this.TAI_list = new ArrayList<>();
      for (int i = 0; i < size; i++) this.TAI_list.add(in.
          readInt());
    }
    if (in.readBoolean()) this.PDN_ID = in.readUTF();
    if (in.readBoolean()) this.EPS_bearer = in.readByte();
    if (in.readBoolean()) this.E_RA_bearer = in.readByte();
    if (in.readBoolean()) this.DR_bearer = in.readByte();
    if (in.readBoolean()) this.S1_TEID_DL = in.readInt();
    if (in.readBoolean()) this.S1_TEID_UL = in.readInt();
    if (in.readBoolean()) this.S5_TEID_DL = in.readInt();
    if (in.readBoolean()) this.S5_TEID_UL = in.readInt();
    if (in.readBoolean()) this.K_ASME = in.readUTF();
    if (in.readBoolean()) this.K_ENB = in.readUTF();
    if (in.readBoolean()) this.K_NASint = in.readUTF();
    if (in.readBoolean()) this.K_NASenc = in.readUTF();
    if (in.readBoolean()) this.K_RRCint = in.readUTF();
    if (in.readBoolean()) this.K_RRCenc = in.readUTF();
    if (in.readBoolean()) this.K_UPenc = in.readUTF();
  }

}
```