User Data Repository Design Supporting Internet of Things

Anteneh Assen Adem

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 11.04.2018

Thesis supervisor:
Prof. Raimo Kantola

Thesis advisor:
Dr. Jose Costa-Requena

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author:	Anteneh Assen Adem
Title of the Thesis:	User Data Repository Design Supporting Internet of Things
Date:	Language: English Number of pages:9+55

Department of Communications and Networking

Professorship: ELEC3029 - Communications Engineering

Supervisor: Prof. Raimo Kantola

Instructor: Dr. Jose Costa-Requena

Mobile devices comprise smart phones, sensors and actuators which are expected to grow exponentially. Current mobile networks have been designed considering User Equipment (UE) associated to a single user and is mostly used for people to people communication. However, machine to machine communications have different requirements which have not been considered until recent 3GPP standard specifications. The Internet of Things (IoT) devices are autonomous, large in number and often lacking user interface. This creates a challenge in managing them. Different IoT management solutions has been proposed and implemented to solve the challenge. 3GPP started addressing this need in Narrow-Band IoT and Machine Type Communication specifications. In this thesis we analyze proposed standards for managing device subscriptions such as the User Data Convergence (UDC). Given the limitations of UDC for managing IoT devices an extension that enables an operator network to support the management of IoT devices is proposed. The solution is a User Data Repository (UDR) extended with a data model and new design that facilitates IoT device management. A prototype of the design was developed and tested to check the feasibility. The results show the solution could work under the right setup.

Keywords: IoT management, UDR,

Preface

I would like to thank my advisor Dr. Jose Costa-Requena and my supervisor Prof. Raimo Kantola. Their guidance and patience has helped me a lot in improving and finishing this thesis. I also want to thank Aalto university for giving me the study opportunity and the support I got during my study.
Above all I want to thank my parents for their support and the positive role they have played in my life.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
APN	Access Point Name
AUC	Authentication Center
AVP	Attribute Value Pair
CADDOT	Context-aware Dynamic Discovery of Things
CS	Circuit Switching
CIoT	Cellular IoT
CP	Control Plane
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
DNS	Domain Name System
DS	Directory Server
EC-GSM-IoT	Extended Coverage GSM IoT
EPS	Evolved Packet System
ETSI	European Telecommunication Standard Institute
FE	Front End
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HLR	Home Location Register
HSS	Home Subscriber Server
HTTP	Hypertext Transport Protocol
IEEE	Institute of Electrical and Electronics Engineers
IMS	IP Multimedia Subsystem
IMSI	International Mobile Subscriber Identity
IoT	Internet of Things
ITU	International Telecommunication Union
LDAP	Lightweight Directory Access Protocol
LTE	Long-Term Evolution
LTE-M	LTE Cat-M1
MME	Mobility Management Entity
MSC	Mobile Switching Center
MTC	Machine Type Communication
NB-IoT	Narrow Band IoT
PDN	Packet Data Network
PS	Packet Switching
RFID	Radio-frequency Identification
SCEF	Service Capability Exposure Function
SCTP	Stream Control Transmission Protocol
SGSN	Serving GPRS Support Node
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol

UDC	User Data Convergence
UDR	User Data Repository
UE	User Equipment
ULA	Update Location Answer
ULR	Update Location Request
UML	Unified Modeling Language
UP	User Plane
UPnP	Universal Plug and Play
URL	Unified Resource Locator
USIM	Universal Subscriber Identity Module
VoLTE	Voice over Long-Term Evolution
VLR	Visitor Location Register
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1. Introduction

This chapter states the problems this thesis is attempting to solve. It presents briefly the limitation of current technologies in solving the problems and proposes a solution. Then it ends by presenting how the rest of the thesis is structured.

1.1 Motivation

In current mobile networks User Equipment (UE) is the mobile device connected to the network. The UE is associated to a single user and is mostly used for people to people communication. However, the latest Ericsson report indicates that “of the 29 billion connected devices by 2022, 18 billion will be IoT (or machine-to-machine) devices” [1]. In 5G networks, machine to machine communications will produce a significant part of the traffic in the system. Therefore, 5G mobile network is envisioned as a key enabler of industrial internet and machine to machine communications. The concept of UE will include not only personal mobile devices but also sensors and actuators that have different communication requirements. One main difference is that IoT devices will often not have a user interface. This creates the problem of configuring the devices manually. Moreover, a single user could have several personal UE devices in addition to a lot of IoT devices under his subscription. Managing all these devices (i.e. configuring and updating their software), is a non-trivial task. IoT creates the need to process and manage the subscription data of these devices in bulk. The future mobile network can play a key role in solving these problems. It can facilitate the bulk subscription and over the air configuration of IoT devices.

In the past the Home Location Register (HLR) in 2G/3G and later Home Subscriber Server (HSS) in 4G were the only entities that stored and provided all the UE information for authentication, location identification and service provisioning. These systems are designed to store UE data for people to people communication. These systems can be redesigned to store data by taking into consideration the needs of IoT devices.

3GPP has taken the first steps to redesign these storage systems and has defined the concept of User Data Convergence (UDC). It separates the UE data storage from the application logic that uses the data. The UDC requires a flexible data storage design and simplicity of introducing new applications that access the data. The UDC concepts could be used for managing the IoT devices and their subscription data.

1.2 Objective and scope

The objective of this thesis is to design a user data storage system for future mobile networks that besides managing the traditional HSS data takes into consideration the needs of IoT devices. This system is designed based on the UDC concept. It tries to solve the following IoT device related problems.

- Bulk subscription of IoT devices
- Over the air configuration and software update of IoT devices

The proposed design behind the UDC concept consists of a User Data Repository (UDR) for Evolved Packet System (EPS) and IP Multimedia Subsystem (IMS) users that includes UE and IoT device subscription data. In addition, an EPS HSS Application software is implemented based on the UDC concept to populate the IoT subscription data into the UDR.

1.3 Structure

The rest of the thesis is structured in 5 chapters. Chapter 2 reviews the existing data repositories used in the mobile networks in the past such as HLR, HSS. Then this chapter presents the concept of UDC. Chapter 3 describes the design and prototype implementation of UDC. Chapter 4 collects performance analysis and limitations of the current UDC when used for managing end user and device information. In this chapter initial testing of UDC with different EPCs, Narrow Band IoT (NB-IoT) sensors is performed to identify further limitations in the current design when managing large number of connected devices. This chapter proposes a solution for supporting end user services and applications but also M2M communication. In this chapter a solution on how to extend the usage of UDC to facilitate the deployment of M2M communications is detailed. Finally, Chapter 5 provides conclusions and next steps.

2. Background

This Chapter provides an overview of the current systems used for user data storage and handling in mobile networks. The chapter explains the functionality of the HSS and HLR used for storing user data in 2G, 3G. 4G mobile and IMS systems. The limitations of these systems are also presented and the new UDC component proposed by 3GPP to overcome those limitations is described in this chapter. In addition, some basic description about information model and data model is presented.

2.1 User data repositories in mobile networks

The user data repositories in mobile networks were originally targeted to store user credentials for authentication and authorization. These repositories have been evolving over time and this section describes those repositories starting from the HLR used in 2G followed by the HSS used in 3G and 4G later.

2.1.1 HLR/AUC

In 2G and 3G mobile networks, some permanent user data are stored in a network element called HLR and Authentication Center (AUC). Home Location Register (HLR) stores all permanent subscriber data (i.e. subscription information) and some temporary subscriber data (e.g. Visitor Location Register (VLR) Number) for Circuit Switching (CS) and Packet Switching (PS) domain of mobile networks. In addition to storing subscriber data the HLR provides logical functions such as access authorization, mobility management, call establishment support and facilitates a host of services [2]. The HLR is located in the core network as shown in the Figure 1 together with other network elements like MSC and SGSN. The HLR communicates with the Mobile Switching Center (MSC) and Serving GPRS Support Node (SGSN) with standard interfaces define in [3].

Authentication center stores authentication information that is used to authenticate subscribers of the PS and CS domain. The AUC transfers the authentication and ciphering data of the visiting user to the other network elements through the HLR, see Figure 1. The protocol used over the interface between the HLR and AUC, H-interface, is nonstandard [3]. The term HLR/AUC is used to refer to an entity which performs both the functionalities of an HLR and AUC

The protocols used on the interfaces depicted in Figure 1, except for the H-interface, are all standard protocols.

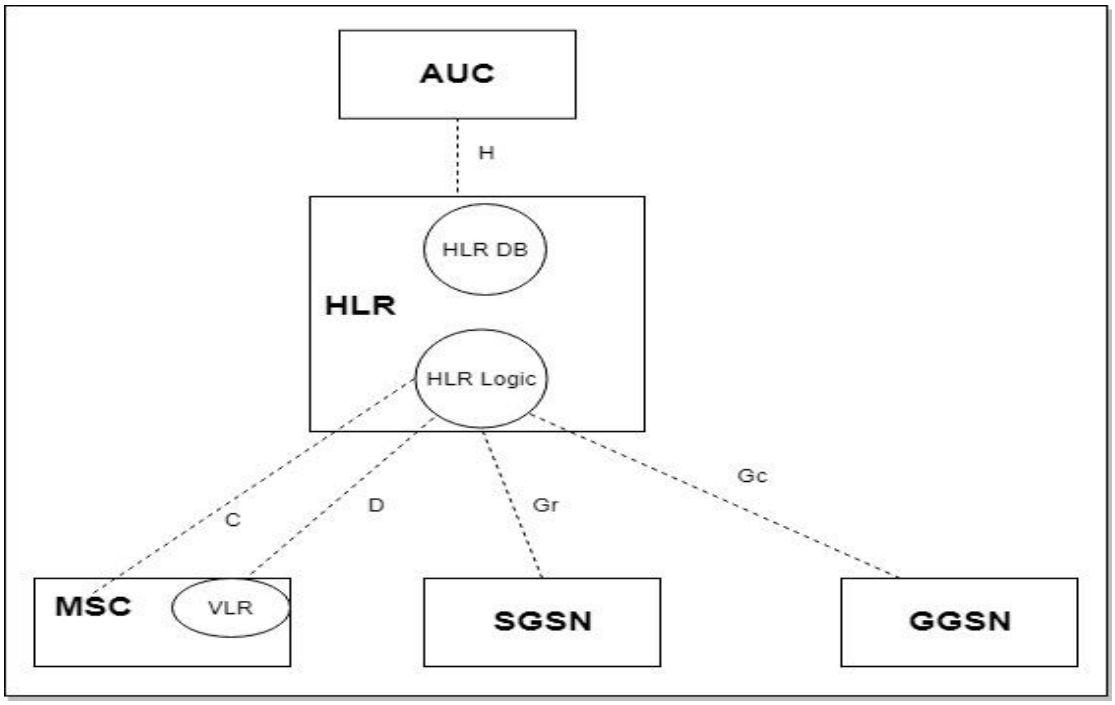


Figure 1. Common Interfaces between HLR and CS and PS network elements

2.1.2 HSS

The mobile networks evolved and in the next releases 3G and 4G the HLR becomes the Home Subscriber Server (HSS). The HSS is the heterogeneous master database that stores user related information like user identification, user security information, user location information and user service profile information. The CS and PS domain HLR/AUC is a subset of the HSS. In addition to CS and PS domain users, HSS stores data for IMS domain users. While storing heterogenous information, the HSS hides the heterogeneity of the information from Application servers that access it. [3]. In addition to storing user data, HSS provides logical functionalities such as mobility management, user security information generation, service, and access authorization for the CS, PS and IMS domain. It has a standard interface to enable network elements found in the above-mentioned domains to communicate with it. Figure 2, shows the logical functionalities of the HSS and standard interfaces between HSS and other network elements. It also shows the location of the HSS in the core network and interfaces with the rest of 3G and 4G network nodes.

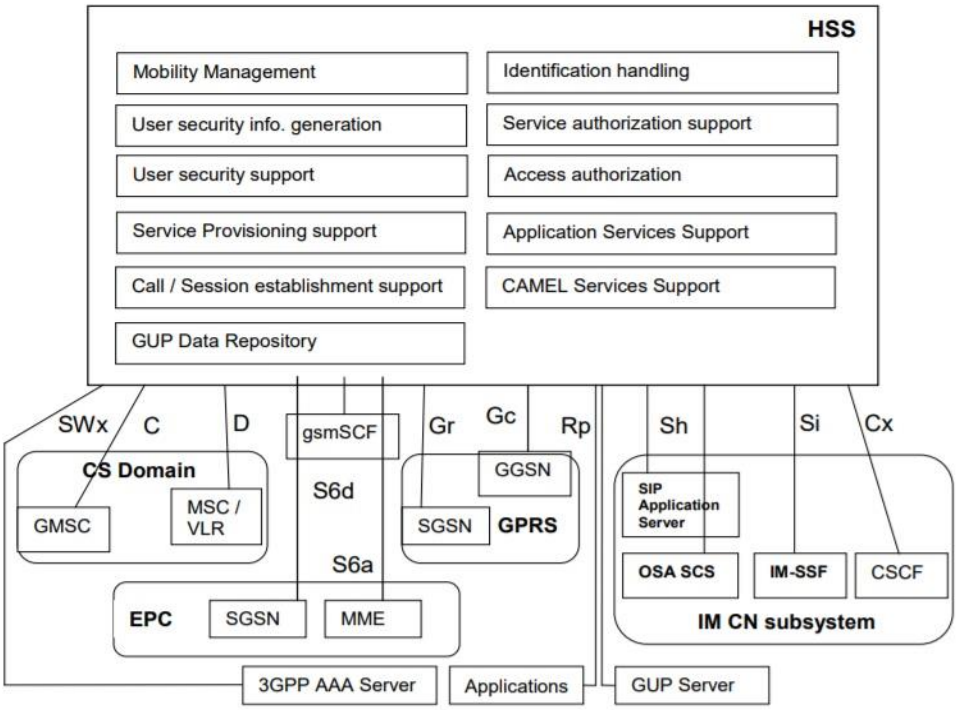


Figure 2. HSS logical functions and Interfaces [3]

2.2 UDC

In 4G, the lack of convergence for storing user and service information was identified. Thus, when adding IMS or Voice over Long-Term Evolution (VoLTE) services a new HSS was proposed to store only the service information while another HSS was used to store basic user security information. Therefore, over the past releases of mobile networks the data has been scattered in different repositories. Currently user data is stored in different network elements like the HLR, HSS, AUC and application servers. As described in the previous section, these network elements are responsible for both the storage and use of subscription and security data. Due to the lack of separation and a standard interface between the application logic that accesses the data and the data storage system, it is difficult or impossible to introduce new applications that provide some services by accessing the stored user data. In addition to this, the scattering of user data over multiple network elements and the existence of multiple reference points to these network elements complicates user data management, user data view and user data mining.

In order to mitigate the above-mentioned problems and more, 3GPP has introduced the concept of User Data Convergence (UDC) [4]. In this concept the user data storage and the application logic that accesses and uses the data are separated, see Figure 3. User data that used to be stored in network element like HSS and HLR/AUC

is converged and stored in a new entity called User Data Repository (UDR). The main building blocks of UDC are UDR, Application Front Ends and the Ud interface.

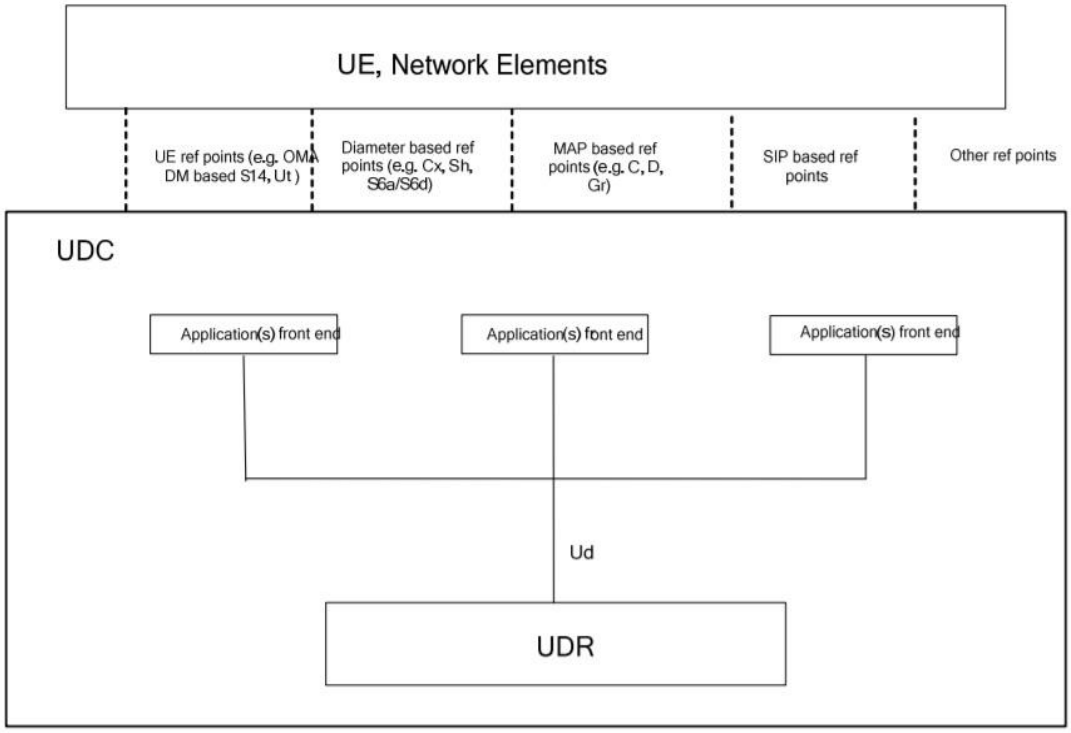


Figure 3. UDC Architecture [5]

2.2.1 UDR

“The User Data Repository (UDR) is a functional entity that acts as a single logical repository that stores converged user data” [5]. User data like subscription data that used to be traditionally stored scattered in different network elements like HSS, HLR and some Application servers is logically converged and stored in the UDR.

Application Front Ends should only access the data that is required by their application logic. To achieve this, UDR should perform authentication of Application Front Ends and authorization of data access based on the data the FEs are trying to access. The information model developed for the storage system should enable a separate view of the converged user data stored for each Application Front End. This means Application FEs will only access the part of the converged user data that contains the data relevant to them. The information model of the UDR should also be flexible enough to allow integration of a new Application Front End information or modification of information model of existing Application Front End without affecting operation of other Front Ends.

2.2.2 Application Front Ends

Application Front Ends are systems that only implement the application logic and store or access their data from the UDR. They might store the data temporarily during processing of some request but will discard it after completion of the operation. The network elements whose data storage and application logic would be separated could keep the application logic and become Application Front End (FE), e.g. an HSS could store the data in UDR and become HSS Application Front End. Figure 4 shows an example of an Application FE interaction with a UDR to get Authentication data of a user.

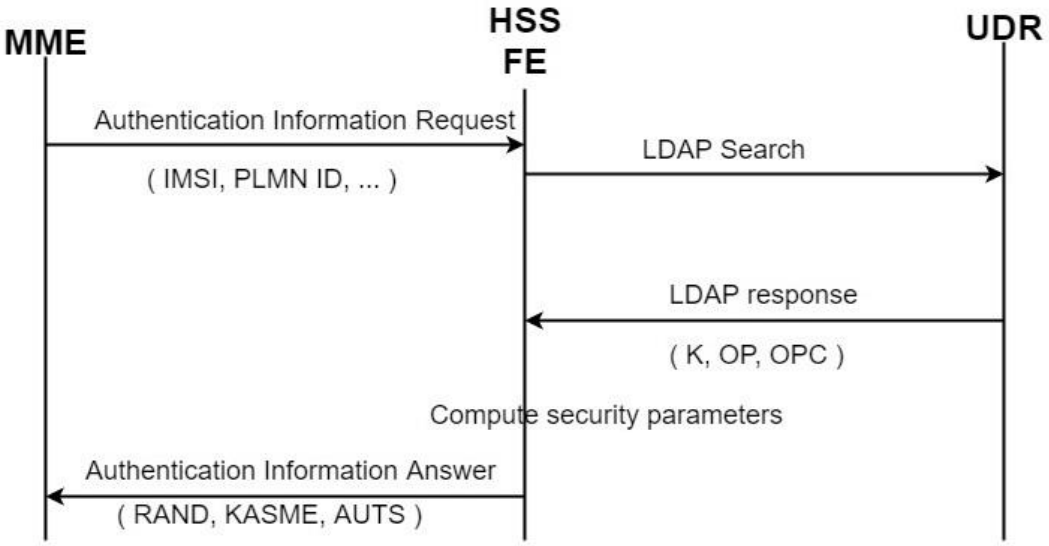


Figure 4. Example of Application FE message flow with UDR

2.2.3 Ud interface

As can be seen on Figure 3, the interface between the Application Front ends and the UDR is call the Ud interface. Using this interface Application FEs can read, write, modify or delete data stored in the UDR. Additionally, Application FEs can subscribe for notifications of change of data stored in the UDR. Two protocols are chosen to be used on this interface by 3GPP [6]. One is Lightweight Directory Access Protocol (LDAP) [7], which is used for the purpose of retrieving, adding, deleting and modifying data stored in the UDR, see Figure 5. The other is Simple Object Access Protocol (SOAP) [8], which is used by the Application FEs to subscribe and unsubscribe for notification of change of data, see Figure 6 and Figure 7. UDR use it to send notification data to Application FEs that subscribe to notification.

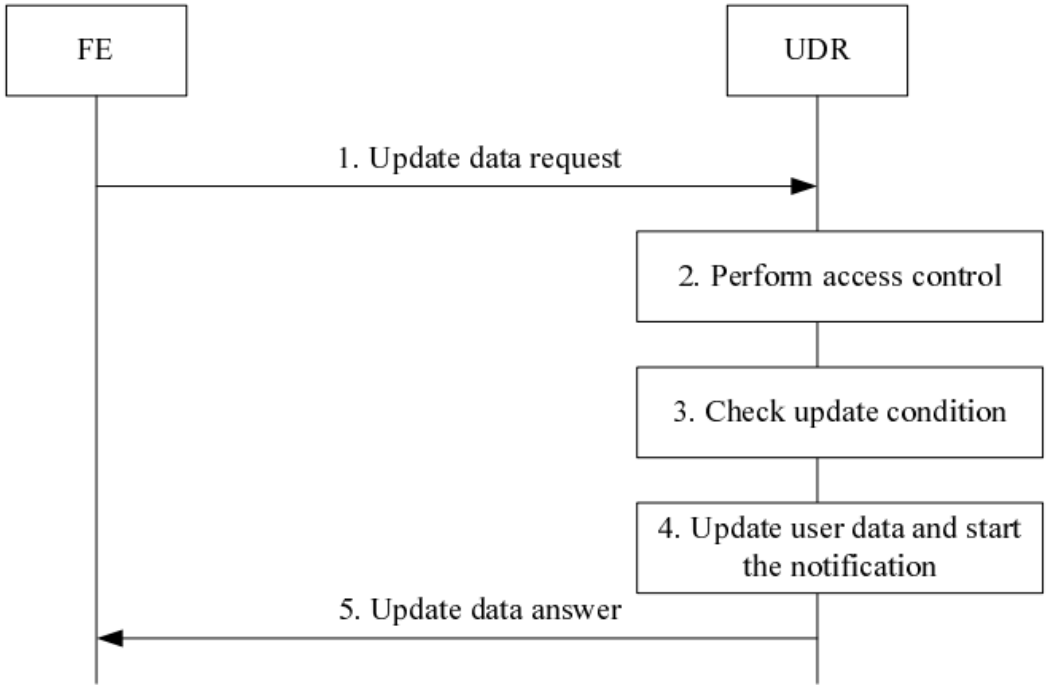


Figure 5. LDAP update data operation on Ud interface [5]

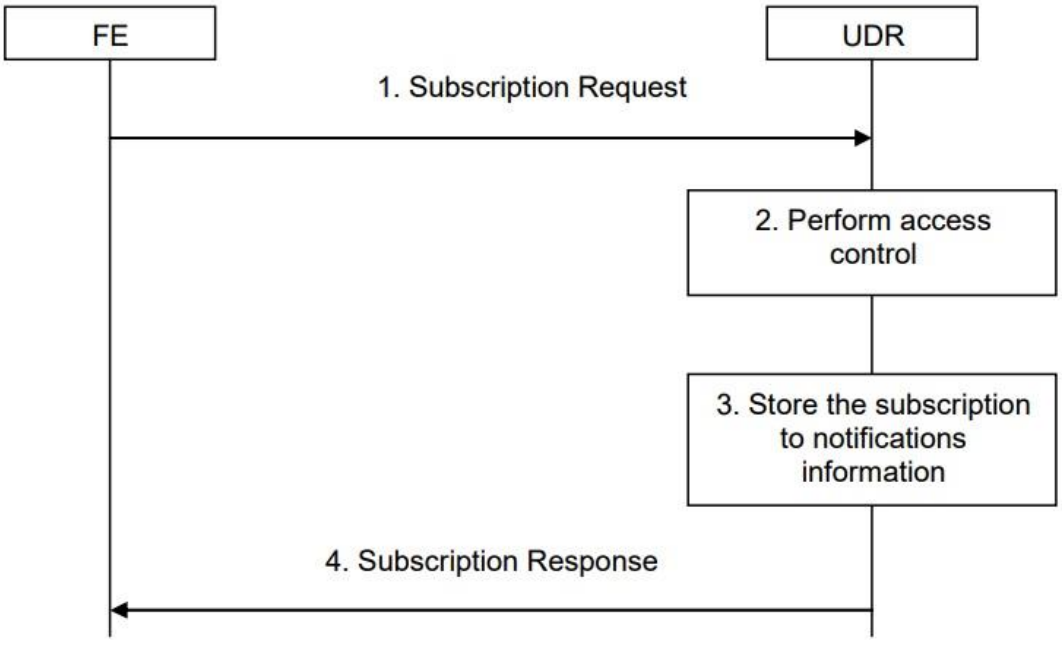


Figure 6. Subscription to Notification on Ud interface [5]

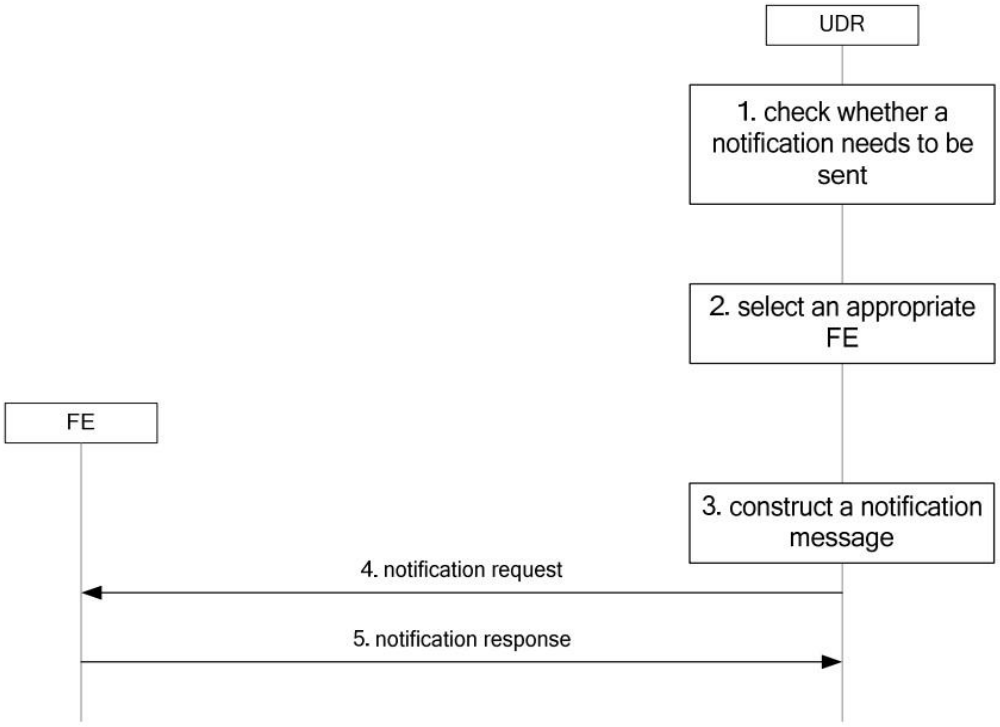


Figure 7. Notification Request on Ud interface [5]

2.2.4 LDAP

The Lightweight Directory Access Protocol (LDAP), which is defined in [7], is an application protocol used by LDAP clients to interact with a Directory server (LDAP server [9]), see Figure 8. This protocol enables an LDAP client to bind to an LDAP server and search, add, delete, modify a directory entry stored in the server.

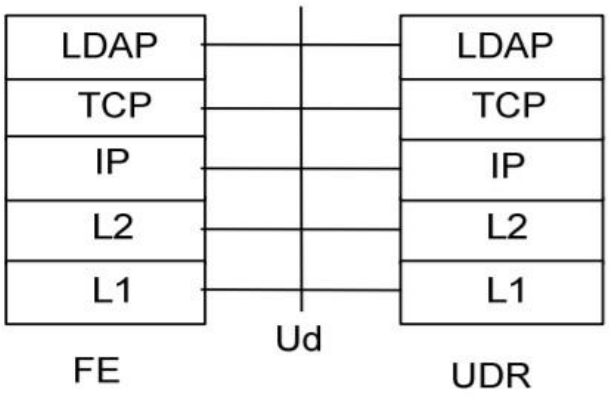


Figure 8. LDAP protocol stack [6]

LDAP servers could return a referral when the requested data is stored in another server. Referral is an LDAP Uniform Resource Locator (URL) that contains host name, port number and optionally a Distinguished Name (DN) on another server [10]. The client uses this information to make another query to the other server. An LDAP server could be configured to contact the other server and return the result to the client [11].

A directory entry is composed of attribute types, attribute values, object class and a distinguished name, which is used to identity the entry uniquely within the Directory. Most directory entries contain information related to an object. As an example, the 'security-parameter' associated with a single Evolved Packet System (EPS) subscription can be considered as an object where the entry for this object will hold the secret key(K) and the International Mobile Subscriber Identity(IMSI) of the EPS subscriber.

The data model of information stored in a Directory is a hierarchical tree structure where the vertices of the tree are the objects. Objects could have hierarchical relation. So, entries could have hierarchical relation based on the objects the entries contain information for. As an example, a country object can have city and language object under it. So, entries that contain city and language information about a country are placed under an entry that contains the country information. In Figure 9, the country entry c=Finland is on top of the tree, and the entries cn=cities and cn=language which contain information about the cities and languages in country Finland is placed under it.

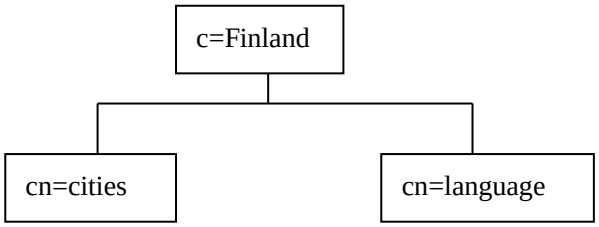


Figure 9. Example of hierarchical tree structured directory entries

LDAP is characterized as a write-once-read-many-times service [12]. This means LDAP is optimized for storage of data that could be read many times but updated rarely. The nature of most of the data, like subscription data of users to be stored in the UDR shares this characteristic of write-once-read-many-times. In terms of security, LDAP servers provide access control at individual object and individual attribute level [13]. Application FEs may be required to have access rights to only certain attributes in an entry. In terms of open interface access, LDAP is a standard protocol that enables any LDAP standard client communicate with any LDAP server [14]. In UDC, the UDR should be accessible to different Application FEs that are authorized to access it to simplify creation of new services [4]. For this purpose, the data access interface should be open and standardized.

As shown in Table 1, an LDAP Server has some advantages over Relational Database in meeting the requirements of the UDC concept. It could be accessed by a standard wire protocol, LDAP. It is possible to restrict data access at attribute level. This allows a fine-grained access control of data accessed by Application Front Ends.

Table 1. Some comparison between LDAP Server and Relational Database

LDAP server	**Relational Database**
Read optimized [12]	Better for frequently updated data
Standardized local and remote data access methods. [12]	No standardized remote data access methods. [12]
Access control up to object and attribute level. More fine-grained access control. [13]	Access control up to Column level. [13]
Hierarchical data organization.	Flat tables. No hierarchical data organization.

A work done in [15] investigated LDAP as a back-end technology to serve the static portions of subscriber data for both HLR and HSS. The result of the investigation shows the performance is good when the LDAP Directory Server stores the subscriber data in memory cache.

2.2.5 SOAP

Simple object access protocol (SOAP) is a lightweight structured message exchange protocol and messages are encoded in Extensible Markup Language (XML) format. A SOAP message is an XML document containing Envelope, Header, Body and Fault elements. SOAP relies on application protocols like Hypertext Transport Protocol (HTTP) to exchange messages, see Figure 10. When SOAP uses HTTP to exchange messages, the SOAP message is contained in the HTTP message body.

3GPP [6] has chosen this protocol for the exchange of Subscribe and Notify messages between Application Front Ends and the UDR. But a work done in [16] argued that SOAP messages are bulkier in comparison to its RESTful counterparts. In addition, it argued that using SOAP and LDAP protocol on Ud interface is not well suited for UDC. Instead it proposes using a single protocol called oData [17].

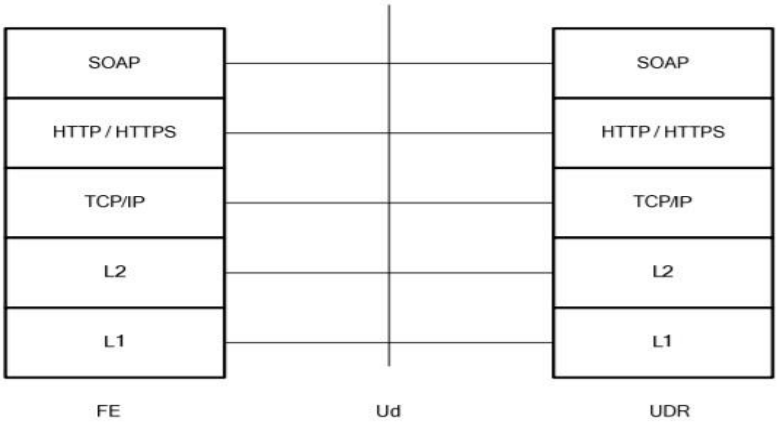
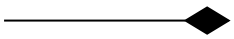
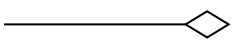


Figure 10. SOAP protocol layer [6]

2.2.6 Information model

Generally, an information model is used to show the relationship between entities in a system. "An Information Model provides the framework for organizing your content so that it can be delivered and reused in a variety of innovative ways." [18]. Information modeling requires a careful analysis of the managed objects to understand the relation between the objects and the features, or attributes, of the objects. One common modeling language used for information modeling is Unified Modeling Language (UML) [19]. It defines notations to be used for representation of objects and the relation between objects. Table 2 shows some of the basic notations defined in UML.

Table 2. Some UML Notations

Notation	**Meaning**
	Represents Composition. An object on the diamond edge has an object on the non-diamond edge. An instance of an object, which is placed on the non-diamond edge, can exist only when the instance of an object placed on the diamond edge exist.
	Represents Aggregation. An object on the diamond edge has an object on the non-diamond edge. An instance of an object, which is placed on the non-diamond edge, can exist independently.
0..1	None or one of the instance of the information object
0..*	None or multiple instance of the information object
0..N	None or up to N instance of the information object
1	Only one instance of the information object

An information model, as it is at conceptual level, does not concern itself with implementation of the model. But it will be used by implementors as a guide to create data models which are used by implementors.

2.2.7 Data model

A data model is defined at a lower level abstraction. It provides detailed information that is used for implementation. Since data models depend on the environment where the model is implemented, different data models can be constructed from a single information model.

2.3 Internet of Things

A number of computing devices are being connected to the internet, creating the Internet of Things (IoT). "The term IoT was initially proposed to refer to uniquely identifiable interoperable connected objects with radio-frequency identification (RFID) technology" [20]. But there are many definitions provided for IoT by different standardization organizations.

A work done in [21] aimed to provide an all-inclusive definition of IoT that addresses all the IoT's features. To come up with such definition, they have considered definitions provided by organizations like European Telecommunications Standards Institute (ETSI), International Telecommunication Union (ITU), Institute of Electrical and Electronics Engineers (IEEE) and other organizations. In their work they have mentioned IEEE defines IoT as: "A network of items—each embedded with sensors—which are connected to the Internet.". While ETSI defines a similar concept called Machine Type Communication (MTC) as: "Machine-to-Machine (M2M) communications is the communication between two or more entities that do not necessarily need any direct human intervention."

IoT communication technologies have become mature and widespread. IoT communication technologies can be categorized into short-distance and wide area network communication technologies. For short distance communication Zigbee, Wi-Fi, Bluetooth, Z-wave and other technologies can be used. And for wide area network communication, among other wide area network technologies, technologies standardized by 3GPP like Long-Term Evolution (LTE) can be used. [22]

2.3.1 3GPP on IoT

UDC is intended to solve the problem of having different repositories to store user and service data. However, 5G networks are posing new requirements and challenges for data repositories. 5G aims to address limitations of previous 2G, 3G and 4G standards and be a potential key enabler for IoT [23]. The 5G networks have requirements beyond providing connectivity to end users. 5G is targeting users and machines with a wider set of requirements in terms of authentication, service data and user provisioning information. The concept of Internet of Things (IoT) envisions a connection of billions of heterogeneous devices communicating with each other and

other non IoT devices autonomously to provide a variety of services. Most traditional communication is triggered or supervised by a human sitting in front of a personal computer or using some communication device like a smartphone. Because of this, most existing networks, networking technologies and services are designed based on the needs and behaviors of humans.

There are a variety IoT devices with different requirements than those of current UE like smart phones. Different IoT device types could have different operational requirements based on the type of services they provide and the environments they operate in. Depending on the services a device provides it could require high data rate or low data rate connection. And depending on the environment it is operating in, it could be required to operate with very minimum power consumption so that the battery could last for years.

To accommodate IoT devices in 3GPP networks, 3GPP has tried to identify, see [24], the requirements and use cases of these devices. It has analyzed the operational aspects, connectivity aspects and resource efficiency aspects of these devices. In terms of resource efficiency, it is required among other things; to minimize the signaling required for configuration of these devices and transmission of user data, to optimize battery consumption of these devices and to support efficient service discovery mechanisms.

There are radio technology standards, namely Long-Term Evolution, Category M1 (LTE-M), Narrow Band IoT (NB-IoT) and Extended Coverage Global System for Mobile communications (GSM) IoT (EC-GSM-IoT), that are developed by 3GPP to meet some of these requirements [25]. Since the radio technology standards are not enough, 3GPP has developed enhancements in the core network elements to support transmission of small data efficiently and to support transmission of non-IP data. These enhancements are called Control Plane (CP) Cellular IoT (CIoT) EPS optimization, User Plane (UP) CIoT optimization, attachment without PDN and support of non-IP data [25]. To support these enhancements, Mobility Management Entity (MME) should have the capability to transfer small user data using signaling. And a new network element called Service Capability Exposure Function (SCEF) is introduced to support non-IP data transfer, see Figure 11.

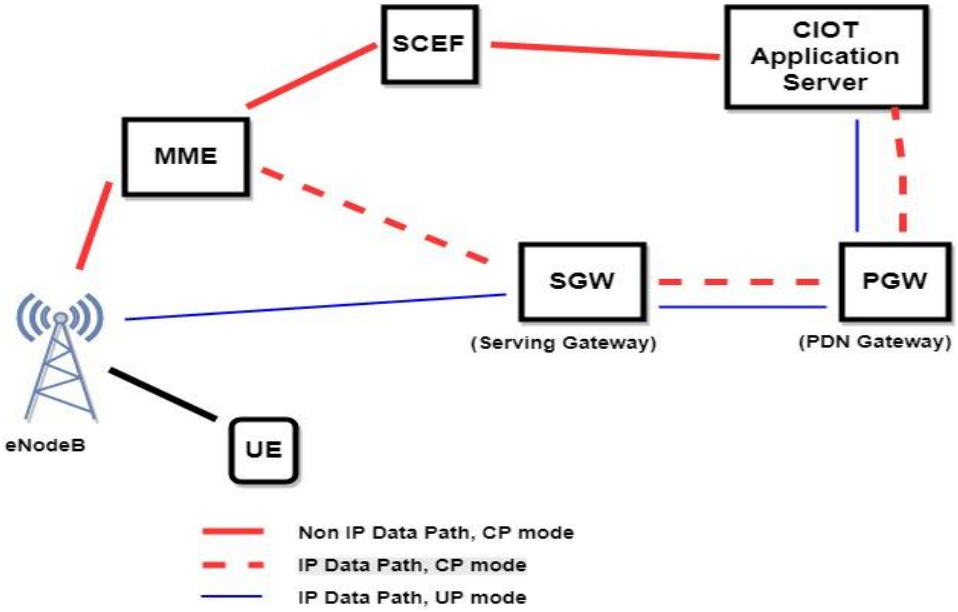


Figure 11. CIoT optimization

2.3.2 Challenges related to IoT devices and solutions

Due to limited or lack of user interface and the substantial number of IoT devices, it creates a challenge for owners to manage (discover, configure and monitor) their IoT devices. This challenge has made the IoT a hot research subject for researchers and a business's opportunity for big and start-up companies. In this section we try to present some of the solutions proposed and developed by researchers, organizations and companies.

To discover and configure sensor devices autonomously a framework that utilizes the current mobile devices is proposed in [26]. This framework is called Context-aware Dynamic Discovery of Things (CADDOT). Before developing this framework, the researchers have identified among other challenges the challenge of configuring devices manually when there are many sensor devices. This framework relies on a nearby mobile device that runs an application which can communicate with the sensor devices. The application running on the mobile device will get the configuration data of the sensors from the cloud. The framework is designed under the assumption the sensor devices will try to connect to a Wi-Fi network automatically and with no authentication.

Another approach which uses mobile devices as a hub for dynamic configuration of sensors is proposed in [27]. In this approach the sensors communicate with an application running on the mobile using wireless technologies like Wi-Fi and Bluetooth. And a plug and play mechanism is used to configure the sensors.

A different approach, which uses a client server architecture for discovery of IoT devices is being developed by XMPP Standards Foundation in one of its XEP series

specifications [28]. In this approach a client (an IoT device) can discover an XMPP server using DHCP, Multicast DNS or SSDP/UPnP. Once finding an XMPP server, it uses Extensible Messaging and Presence Protocol (XMPP) to communicate with the server and register itself. It defines a way for the IoT device to be discovered only by the owner of the device.

Beyond the solutions proposed on paper above as a specification and a research topic, there are real commercial IoT management solution which are already deployed. One of them is Amazon's 'AWS IoT Device Management' system [29]. It claims to provide services like bulk registration of IoT devices, device discovery in real-time, remote device monitoring and remote device management including software updates. Other commercial solutions that claim to provide similar solutions are 'Microsoft Azure IoT Hub' [30].

3. User Data Convergence Design

This chapter describes the design of the User Data Repository. The chapter describes the information model which is defined in 3GPP for storing User Data. Currently 3GPP has defined Baseline Information Models for EPS and IMS, which are explained in this chapter.

3.1 UDR information model

An information model is used in the UDR to model a managed objects at a conceptual level [31]. The information model for the UDR is based on the Common Baseline Information Model, see Figure 12 and Figure 13, defined in 3GPP TS 32.182[32]. As stated in 3GPP TR 22.985[4], UDR information model should start with Common Baseline Information Model.

The Common Base line information model defines the relation between subscription and services, services and End Users and so on as shown in Figure 12. It also defines the relation between EPS, General Packet Radio Service (GPRS), CS and IMS services and some of the information object classes that are related to these services, see Figure 13.

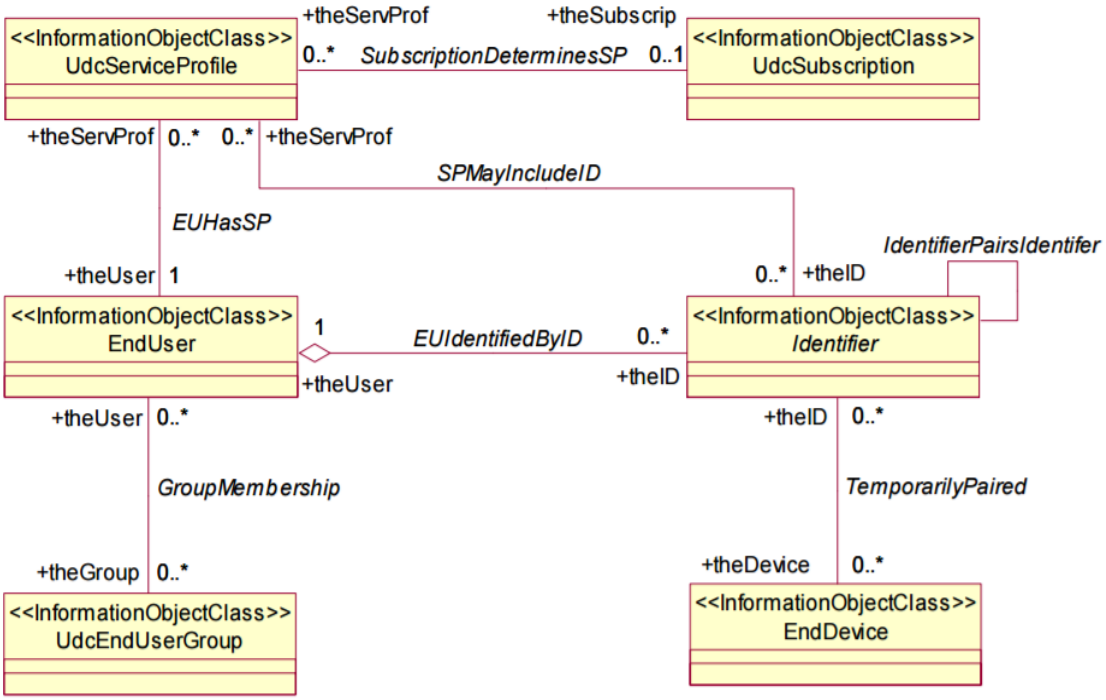


Figure 12. UDC Core Common Baseline Information Model [32]

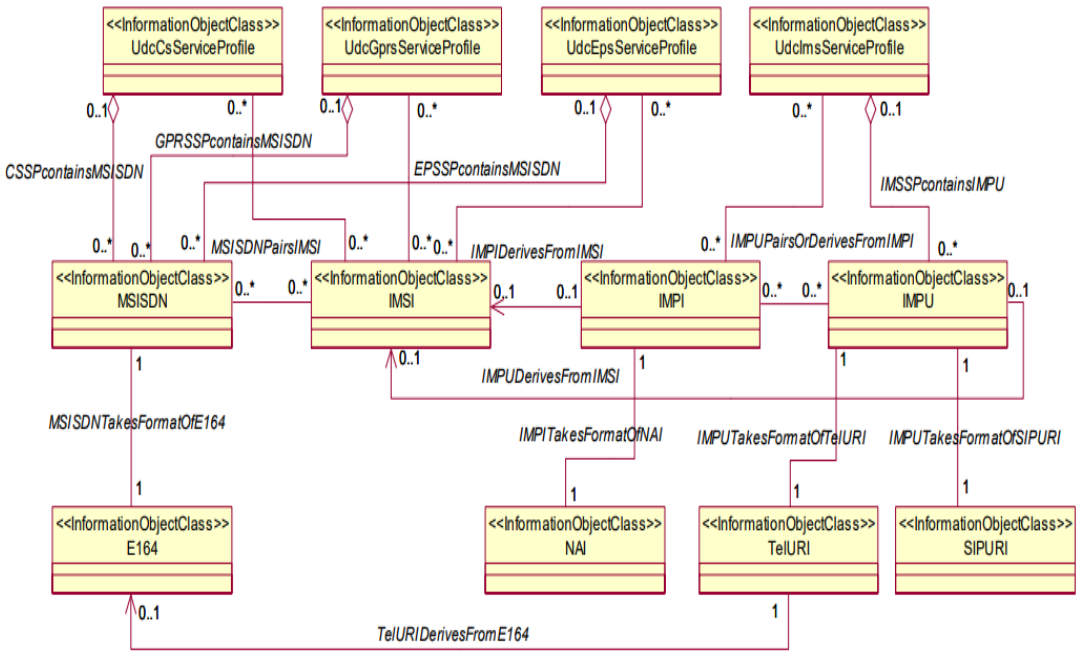


Figure 13. UDC Identifiers Common Baseline Information Model [32]

Based on the Common Baseline Information Model, a Specialized Information Model for EPS and IMS shall be designed. “A Specialized Information Model describes the specific relationships between the information in a given particular case. The Specialized Information Model takes into account the specific applications, the functionality included and the relevant business information” [4]. An EPS Specialized Information Model is designed for EPS data that is normally stored in EPS HSS. And an IMS Specialized Information Model is designed for IMS data that is normally stored in IMS HSS.

For the purpose of analyzing the proposed solution in this thesis, a Specialized Information Model for IMS and EPS is designed to store user data. The design is mainly based on the principle that the design shall be flexible enough to easily add a new information model for a new application or to add new data in the existing information model.

Base on the EPS and IMS Specialized Information Models designed, one converged information model is created for the UDR. Since the UDR is user centric, i.e. almost all the data stored in the UDR is about the user, ‘EndUser’ information object class is used as a point of convergence for different Specialized Information Models as shown in Figure 14.

Adding new data or new information model in the existing converged information model for an application FE shall only affect the operation of the application FE that the data is added for. To achieve this, each Information object class in Specialized Information Model of an application FE is designed to hold data that is used only by that application FE, see Figure 14. Any data that could be used commonly by different application FEs shall have its own information object class.

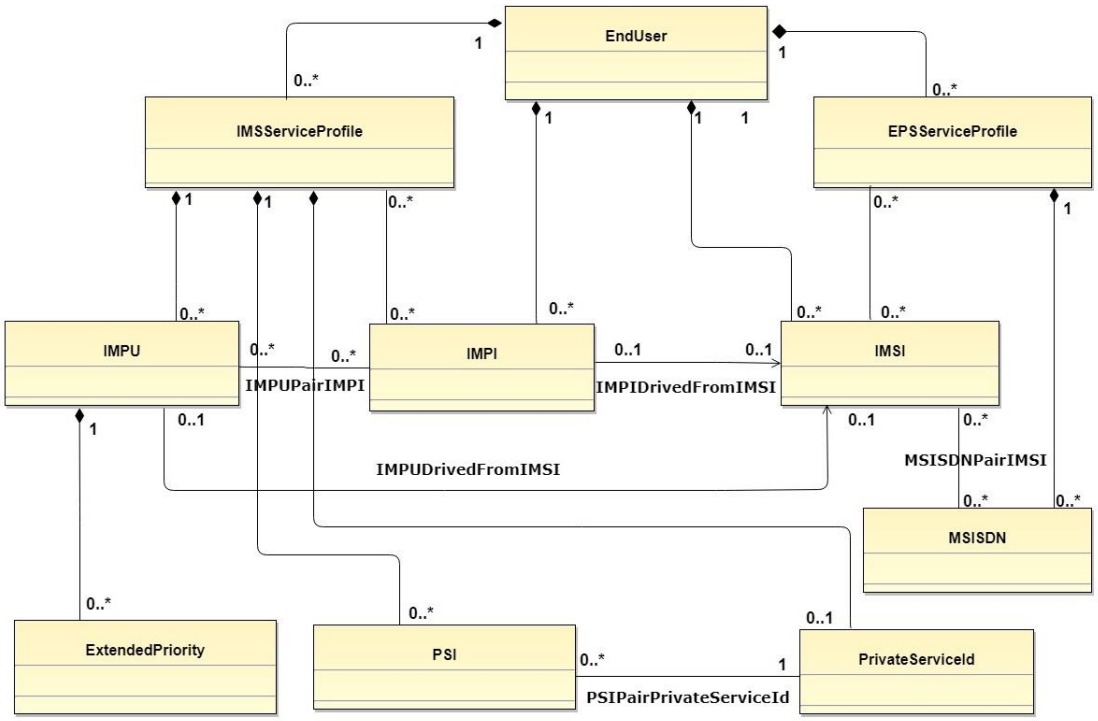


Figure 14. Converged Information Model

Figure 15 shows a Specialized Information model for storing EPS service profile data. Each service profile is identified by a service profile id which is unique in the UDR. It contains a permanent subscription related data for an end user. It also holds dynamic information (like which MME is currently serving the UE(s)) of the UE(s) that use the service profile data.

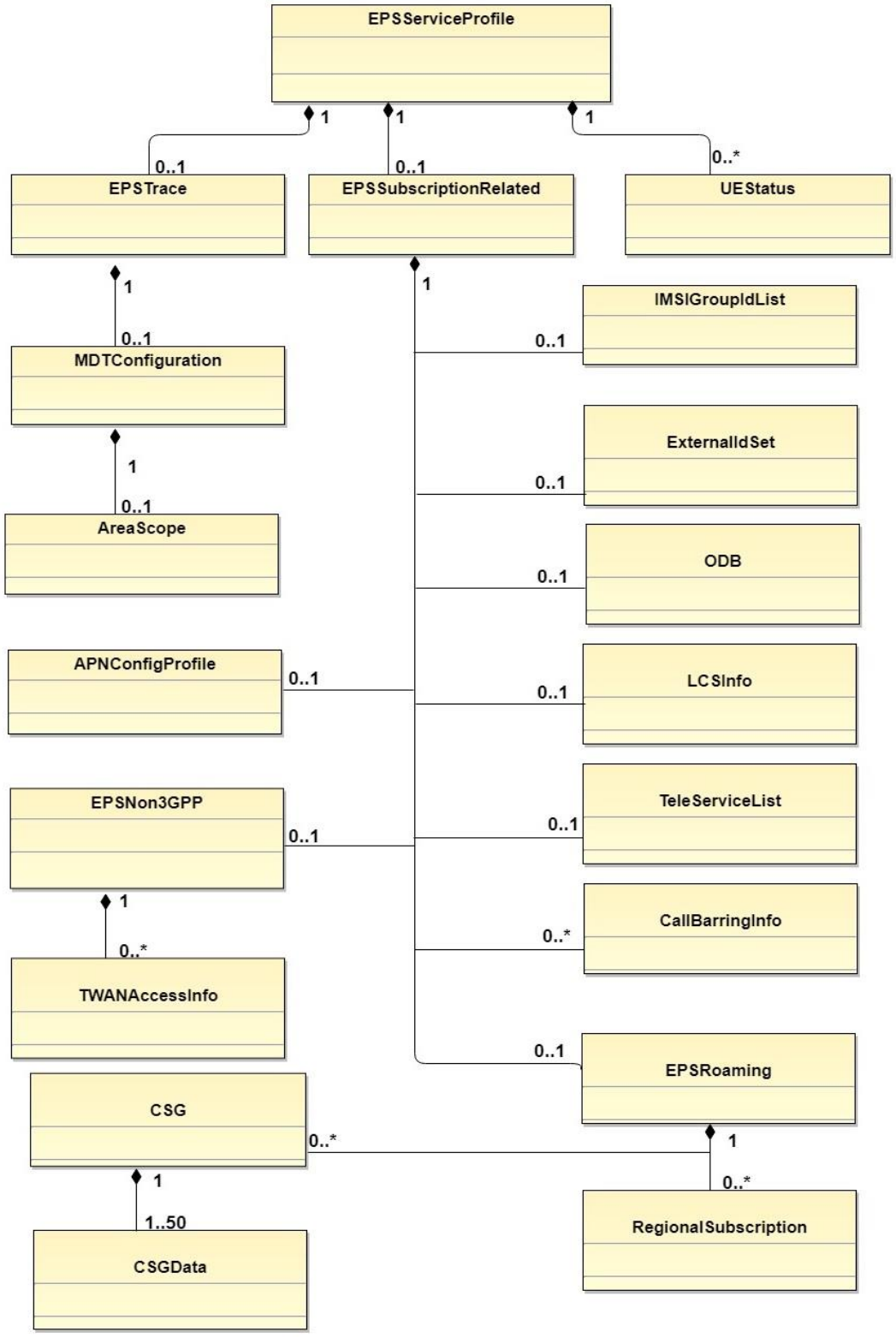


Figure 15. Information Model for EPS Service Profile

3.2 UDR data model

After designing the information model for the UDR, the next step is designing a data model that could be used to create a schema for Directory Server(s) that will store the information. The data model used in the Directory Server for organizing data has hierarchical structure, so a Tree-like data modeling is used.

As discussed in section 2.3.3, there can be more than one data model derived from a single information model based on implementation requirements. For this data model, the requirements that are taken into consideration were the number of operators, scalability, flexibility and the kind of system that stores the data.

The data model is designed for a single operator. So, on the top of the Tree-like data model is the operator, as shown in Figure 16. To make the system scalable, each system (i.e. IMS and EPS) data is put in a separate tree. For example, EPS subscription data will be stored under one tree and IMS subscription data will be stored under another tree as depicted in Figure 17. This enables easy deployment of EPS data in one directory server and IMS data in another directory server. To maintain the logical convergence of data, a referral(s) shall be configured in each server. The definition of referrals is given in Section 2.2.4.

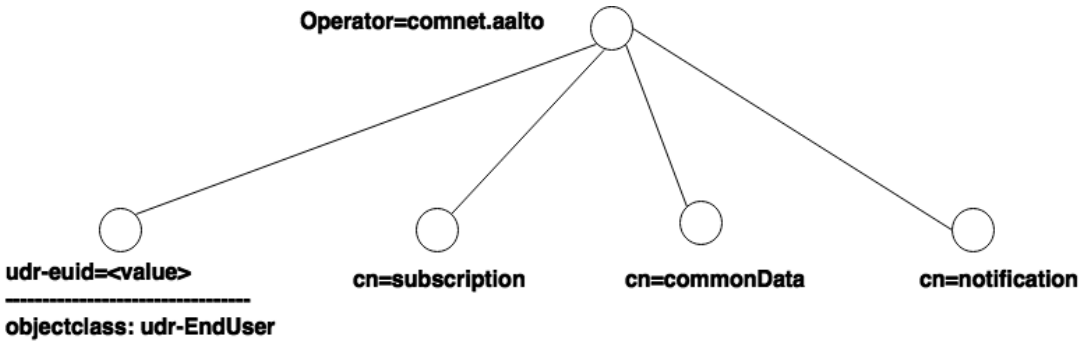


Figure 16. Root of the Tree-like UDR Data Model

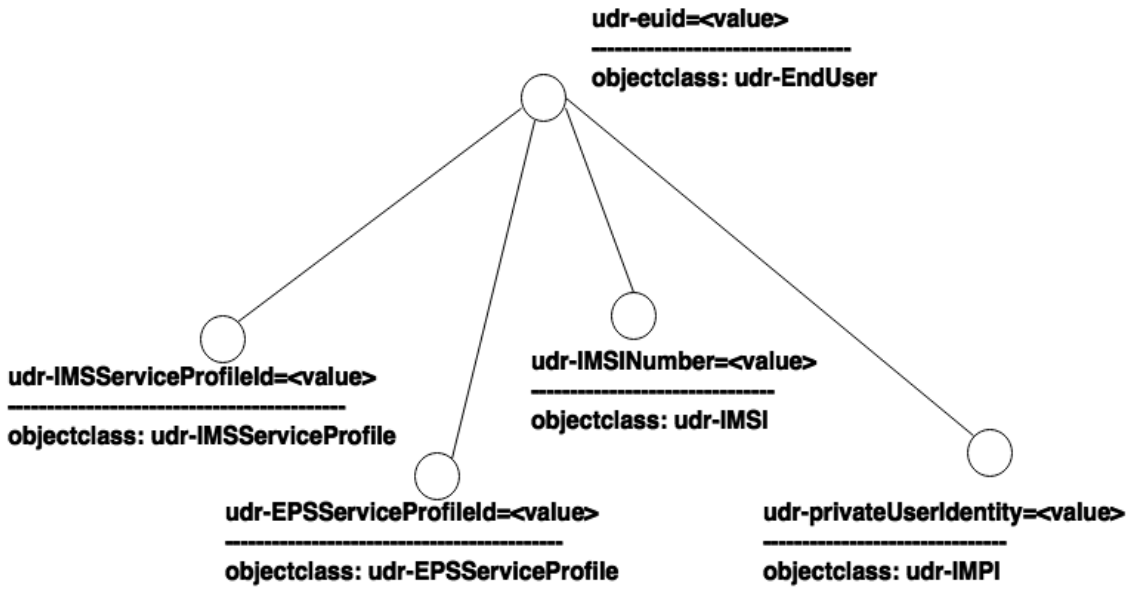


Figure 17. Data model for End-User data

3.3 EPS HSS FE software design

As part of this thesis, I have implemented EPS HSS Application Front-End. This EPS HSS Application FE module is implemented in C and is divided in three modules as shown in Figure 18. These modules are called FE Core, FE LDAP Interface and FE S6a Interface. The implementation is divided based on the functionality that each module should provide. Hence, a change in functionality only affects a single module. For example, if the library used for LDAP client does not support some feature, it can be replaced by a new library that supports the missing feature. Therefore, changing the LDAP client only requires a modification in the 'FE LDAP Interface' module, where the new LDAP client library is used.

"FE LDAP Interface" and "FE S6a Interface" modules communicate with the core module with a well-defined interface. The interface is defined in a C header file that contains the methods and structures used for the communication. There is one C header file defining the interface between the 'FE Core' and the 'FE LDAP Interface' module. There is another C header file defining the interface between the 'FE Core' and the 'FE S6a Interface' module. There is no interface between the "FE LDAP Interface" and "FE S6a Interface" module. This design allows replacing the modules with other modules that implement the same functionality. The only requirement is that the new module should support the interface used to communicate with other modules as defined in those C header files.

S6a and Ud are the only standard communication interfaces defined in the EPS HSS FE implemented. Hence, it only processes messages received from the Mobility

Management Entity (MME). The EPS HSS Application FE uses its "FE LDAP Interface" module to get/update user data from a Directory Server. This data is required during processing of messages received from the MME. This data is not permanently stored, it is only temporarily stored in memory during processing of the message. After the message is processed, the data is discarded. The only data that is permanently stored in memory is the configuration data, like the MME Names and IP addresses.

In practice this means that the EPS HSS FE Application functions are stateless, and all the messages received are processed independently irrespective of the message sent or received previously.

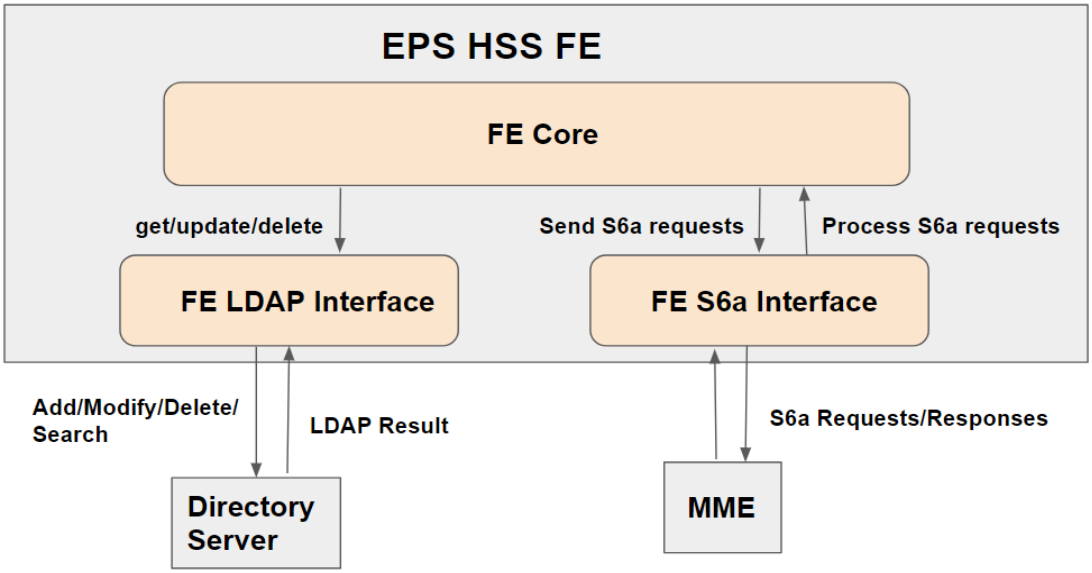


Figure 18. EPS HSS FE software architecture

3.3.1 FE Core module

This module handles the main logic of EPS HSS FE application. It processes Update Location Request, Authentication Information Request, Cancel Location Request, Purge UE Request and Notify Request messages received from an MME. In the current implementation the Reset-Request, Delete Subscriber Data and Insert Subscriber data messages are not supported.

This module receives the messages to be processed from the "FE S6a Interface" module. The messages are passed in C structures define in a header file used to interface the "FE Core" module and "FE S6a Interface" module. There is one C structure defined for each message. For example, for Purge-UE Request message, a C structure called '`pur_msg`' is defined to hold the message, see below.

```
struct pur_msg{

    struct utf8string mme_name;
    struct utf8string imsi;
    unsigned32 pur_flags;
    struct supported_features *supported_features;
    struct eps_location_information * eps_location_information;
};
```

This module checks if the received request message is for a valid user. To validate the user, the module uses “FE LDAP Interface” to get the service profile of the user from the LDAP Server. The IMSI received in the request message is used to identify the user. If there is no data for the given user, the processing is interrupted, and a response message is passed to the “FE S6a Interface” module. This response follows the standard specifications defined in 3GPP [33]. However, if the “FE LDAP Interface” can successfully retrieve the service profile data for the user, the module continues processing the message. There could be more than one interaction (add, delete, update) with the Directory Server during processing of a single request message.

3.3.2 FE S6a Interface module

This module handles the communications with the MME based on S6a specifications defined in 3GPP [33]. This module parses the S6a messages received from the MME and checks if the received message is valid. If it is valid, it passes the message to the ‘FE Core’ module for further processing. The “FE S6a Interface” module only passes part of the message that the ‘FE Core’ module needs for processing the message.

This module uses an open source Diameter protocol implementation called ‘freeDiameter’[1]. ‘freeDiameter’ handles sending and receiving of Diameter Application messages. It also has methods used to register call back functions that are used when a specific Diameter application message is received. But since it only implements the basic diameter protocol, an additional S6a library is used for processing the S6a Diameter messages. I have implemented this library, but not as part of this thesis.

The S6a library provides methods to parse the content of the S6a interface messages and Attribute Value Pair (AVP). The S6a library also provides methods to register callback functions that are called when S6a messages are received. The S6a library implements the S6a interface messages and AVP structures that are defined in 3GPP specifications [33]. The RFC [34] on Diameter Protocol standard defines the basic structure of APVs and Diameter commands (messages). As an example, a structure of

[1] freeDiameter website: http://www.freediameter.net/

User-Id AVP is provided below.

```
/* User-Id */
{
   struct dict_avp_data data = {
        1644, /* Code For User-Id AVP*/
        10415,   /* Vendor */
        "User-Id",  /* Name */
        AVP_FLAG_VENDOR |AVP_FLAG_MANDATORY,  /* Fixed flags */
        AVP_FLAG_VENDOR,   /* Fixed flag values */
        AVP_TYPE_OCTETSTRING  /* base type of data ( format of
the data ) */
   };
        CHECK_dict_new(DICT_AVP, &data, UTF8String_type, NULL);
};
```

The S6a library uses the methods defined in ‘freeDiameter’ for parsing contents and registering callback functions. ‘freeDiameter’ checks validity of S6a messages and if the message is not valid, it responds with the appropriate Diameter error message. The S6a message could be invalid if the message contains AVP(s) that do not belong to the message, or if a mandatory AVP that should be present in the message is missing.

“FE S6a Interface” module uses the methods defined in the S6a library for parsing contents of messages and registering call back functions. The callback functions are defined in this module. When a valid S6a message is received, the callback function that is registered for the message is called. And the message is passed to the callback function as an argument into the callback function’s parameter.

For example, ‘`ss_reg_cb_ulr`’ method defined in S6a method registers a callback function called ‘`fe_s6a_ulr_cp`’ for handling the Update Location Request message. When an Update Location Request message is received, the message is passed to ‘`fe_s6a_ulr_cb`’ function as argument to the function’s parameter called ‘`msg`’, as shown below.

```
ss_reg_cb_ulr( fe_s6a_ulr_cb ); /*Register fe_s6a_cb_ulr*/

int fe_s6a_ulr_cb(struct msg ** msg, struct avp * av, struct
    session * sess, void * opaq, enum disp_action * act);
```

The callback function ‘`fe_s6a_ulr_cb`’ defined in the “FE S6a Interface” module parses the content of the message using methods defined in the S6a library. Then it copies the content in to a structure called ‘`struct ulr_msg`’. Then it passes the structure to a function called ‘`fe_ulr`’ which is defined in ‘FE Core’ module. The ‘`fe_ulr`’ function processes the Update Location Request messages. After

processing the Update Location Request message, it copies the content of the Update Location Answer message in the structure called ‘`struct ulr_response`’. Then it returns this structure to ‘`fe_s6a_ulr_cb`’ method. The ‘`fe_s6a_ulr_cb`’ method uses the data stored in this structure to fill the Update Location Answer message and then send Update Location Answer message to the MME.

```
struct ulr_msg{
   struct utf8string mme_name;
   struct utf8string * imsi;
   struct supported_features * supported_features;
   struct terminal_info terminal_info;
   enum rat_type rat_type;
   unsigned32 ulr_flg;
   enum ue_srvcc_capability ue_srvcc_capability;
   struct octetstring * visited_plmn_id;
   struct octetstring * sgsn_number;
   enum homogeneous_support_of_ims_voice_over_ps_sessions *
        homogeneous_ims_vop;
   struct address * gmlc_address;
   struct active_apn * active_apn;
   struct equivalent_plmn_list * equivalent_plmn_list;
   struct octetstring * mme_number_mtsmsm;
   enum sms_register_request  sms_register_request;
   struct diameterid * coupled_node_diameter_id;
};

struct ulr_response * fe_ulr( struct ulr_msg msg );

struct ulr_response{

     struct supported_features * supported_features;
     unsigned32 ula_flags;
     struct subscription_data * subscription_data;
     struct reset_id * reset_id;
     enum error_diagn *error_diagnostic;
     S6A_RESULT          *result;          /*DIAMETER_SUCCESS,
DIAMETER_ERROR_USER_UNKNOWN …*/
};
```

3.3.3 FE LDAP Interface module

This module handles the Ud interface with an LDAP Server. The ‘FE LDAP Interface’ module interacts with the ‘FE Core’ module to fetch, delete and update user data stored in a Directory Server (DS). The ‘FE LDAP Interface’ module and the ‘FE Core’

module are aware of the schema of the data stored in the DS.

This module acts as an LDAP client to access the data stored in an LDAP server. There are opensource LDAP client libraries implemented in C language, such as OpenLDAP's[2] and NetIQ[3]. NetIQ client library is based on OpenLDAP and has the most recent updates from 2016 [35]. But OpenLDAP client library is recent, so it is chosen to be used in the implementation of the LDAP client in 'FE LDAP Interface' module.

OpenLDAP client library provides a lot of methods that could be used to interact with an LDAP Server. But the 'FE LDAP Interface' module only uses '`ldap_initialize`', '`ldap_search_ext_s`', '`ldap_modify_s`', '`ldap_add_s`' and '`ldap_delete_s`' methods. It uses these methods to establish connection with DS and then to add, delete, update and retrieve user data.

As an example, when a method defined in 'FE Core' module wants to get Access Point Name (APN) data of a user, it calls the method '`fe_ldap_fetch_apn_config_profile`', see below.

```
struct entry * fe_ldap_fetch_apn_config_profile( char
       *service_profile_distinguished_name );
```

When it calls this function, it passes the search-base[4] as an argument in the function's parameter '`service_profile_distinguished_name`'. '`fe_ldap_fetch_apn_config_profile`' is defined in 'FE S6a Interface' module. This method uses '*ldap_search_ext_s*' method to search the APN data from the LDAP Server where the APN data for the user is stored. If the LDAP Sever returns the requested APN data, the module copies the data into a structure called '*struct entry*'. And it returns the structure to the method that has called this method. If there is no APN data, it returns NULL. '*struct entry*' is defined in a C header file that is used to interface 'FE S6a Interface' module with 'FE Core' module. Its definition is given below.

```
struct entry{

   char *dn;  /*Distinguished Name of the entry*/
   struct attr_val_pair *attr_val_pairs;  /* Attribute types
                                  and corresponding values*/
   struct entry *next;
};
```

[2] http://www.openldap.org/

[3] https://www.novell.com/developer/ndk/ldap_libraries_for_c.html

[4] A Search base is the place where a search for a data stored in a directory server starts. It is the distinguished name of the entry within which or under which the data can be found.

3.4 SOAP Subscription/Notification design

One of the functionalities expected from the UDR is a subscription and notification functionality. These functionalities are used to receive data when changes occur in the data stored in the UDR. After checking available Open Source implementations of Directory Servers, none was found with a support for Subscription/Notification using SOAP protocol. Thus, I have implemented a new plug-in software module that handles the Subscribe/Notify functionalities.

To implement Subscription/Notification functionality using SOAP, the Directory Server chosen shall provide a means to include a new functionality. 389 Directory Server[5] and openLDAP Directory Server both provide Application Programing Interface (API). Both DS's API is in C language. And these APIs can be used to integrate Subscription/Notification functionality in the Directory Servers. The 389 DS is chosen to be used because it has a better documentation [36] on how to use the APIs.

The documentation of Red Hat Directory Server can be used for 389 Directory Server.[6] It is possible to integrate a new program into the Directory Server as a plug-in. There are mainly two Plug-in types in 389 DS, namely Pre-Operation and Post-Operation Plug-in [37]. Pre-Operation Plug-in is called to process a request before the backend database is accessed. And Post-Operation Plug-in is called after backend database activities (i.e. adding/deleting/modifying data in the backend database). There is also a Plug-in type that is called when the Directory Server is starting. The Subscribe/Notify Plug-in software module implemented submodules that get registered as Post-Operation Plug-in type and a Plugin-in type that is called when the DS starts.

In this implementation, SOAP uses HTTP to transport the SOAP messages. So, I have implemented an HTTP server as a submodule to be integrated into the Directory server. This submodule is registered as a Plug-in type that is called when the DS starts. This submodule will accept SOAP Subscribe/Unsubscribe messages sent by an Application Front End. It passes this messages to methods that can process them. The submodule uses 'GNU Libmicrohttpd'[7] library to implement the HTTP server.

The Subscribe/Notify module also handles the SOAP Notify Request message. This message is sent when there is a subscription for notification when there is a change on data stored in the UDR. If the change (add/modify/delete) made on the data meets the notification conditions, a Notify Request message is sent to the Application FE which subscribed for the notification. To implement this functionality, there are two options.

---

[5] http://directory.fedoraproject.org/

[6] http://directory.fedoraproject.org/docs/389ds/documentation.html

[7] https://www.gnu.org/software/libmicrohttpd/

One is to implement a submodule as Pre-Operation Plug-in type and the other is to implement a submodule as Post-Operation Plug-in type. If the submodule is implemented as a Pre-Operation Plug-in, Notify Request message will be sent before data is added, deleted, modified in the backend database. That means if the LDAP operation fails on the backend database, the Notify Request message sent will contains incorrect information. So, we chose to implement it as a Post-Operation Plug-in. This way it is possible to know if the LDAP operation is successfully performed on the backend database.

How a SOAP Subscribe request message is handled is shown in Figure 19. When an Application FE 'FE-2' wants to subscribe for notification on change made on some data, it sends a SOAP Subscribe Request message. The HTTP server running in the Subscribe/Notify Plug-in will receive the message and pass it to the method that handles it. This method after it processes the Subscribe message, stores the subscribe information in the backend database.

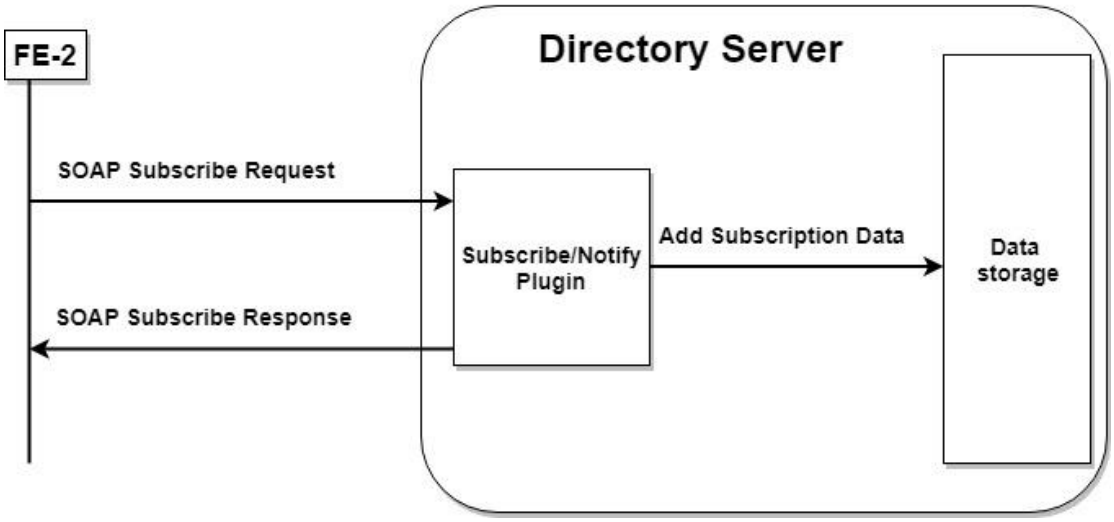


Figure 19. SOAP Subscribe request

The process of sending a SOAP Notify message is shown in Figure 20. Let's assume 'FE-2' has already subscribed to get a notification on change made on some data. And 'FE-1' is requesting an LDAP operation (Add/Delete/Modify) on this data. The Directory Server's own Pre-Operation Plug-in will process the request. Then it performs the LDAP operation on the backend database. The result of the operation is then processed by the Post-Operation Plug-in. Since the submodule that handles notification is registered as a Post-Operation Plug-in type, the result of the operation, along the data, is passed to it. If the result of the operation is successful, the submodule checks if the change made on the data meets the notification conditions. To check this, it gets the subscription information stored in the backend database. Since 'FE-2' has already subscribed for notification on a change on this data, the

submodule sends a SOAP Notify Request to ‘FE-2’.

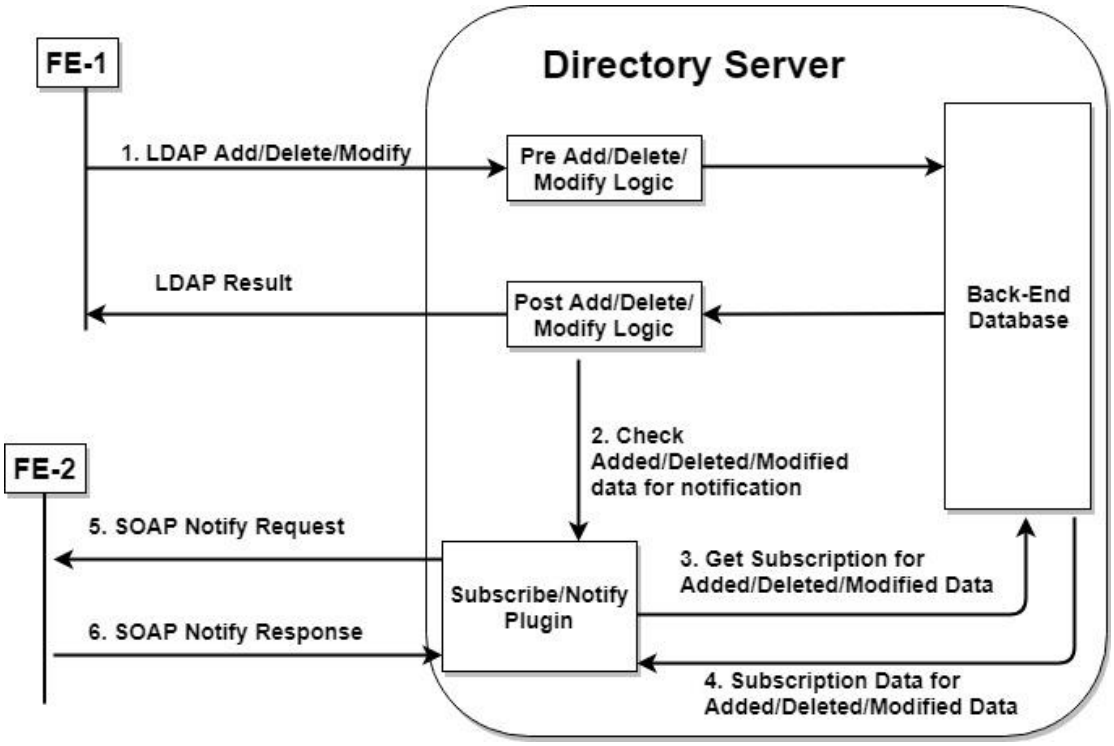


Figure 20. SOAP Notify Request

3.5 Summary of opensource libraries used

Table 3 summarized the open source libraries used in the implementation of EPS HSS FE Application and Subscribe/Notify Plugin software.

Table 3. Summary of opensource libraries used

Name	**Description**
freeDiameter	Diameter protocol implementation library. API provided by this library is used to send and receive messages on the S6a interface
openLdap Client Library	LDAP client implementation. API provided by this library is used to connect to an LDAP server and then fetch and update user data.
GNU Libmicrohttpd	Provides API to implement HTTP server as part of another application. API provided by this library is used build an HTTP server that listens for incoming Subscription Requests.

4. Extended Information Model

This Chapter describes how the information model explained in Chapter 3 is extended to hold IoT device data. We propose extending the initial 3GPP design of UDC for future mobile network to manage IoT devices besides traditional HSS and IMS subscriptions. This chapter explains in detail how the extended information model enables bulk subscription and management of IoT devices. In order to validate this extension a testbed is deployed, and performance results are presented in the next chapter.

4.1 UDR information model for IoT device data

This section describes the extensions made on the information model to manage IoT devices. Before modifying the current information model, the additional information required for storing IoT devices was identified. 3GPP [38] has identified some features required for Machine Type Communication (MTC).

The parameters identified by 3GPP [38] are Low Mobility, Time Controlled, Small Data Transmissions, Infrequent Mobile Terminated, MTC Monitoring, Secure Connection and Grouped Based MTC Features. These features can be considered as permanent IoT device data as they do not change frequently. Therefore, this static data can be stored as a service profile data for IoT devices in the UDR.

After identifying the IoT device specific data, the current information model was extended to hold the IoT specific data. The extended information model is proposed to allow bulk subscription of IoT devices and facilitate management of these devices. Even though IoT devices could use both EPS and IMS services. We consider that IoT devices are using EPS services, so the extended information model assumes that each IoT device will have an IMSI number.

4.1.1 Extension for bulk subscription

To achieve bulk subscription of many IoT devices, a new information model is designed to allow sharing of one EPS Service Profile information object among multiple IMSI information objects. The information contained in one IMSI information object uniquely identifies one IoT device. During bulk subscription of IoT devices which use the same EPS service, a new EPS Service Profile information object is created to hold the information about the EPS services. IMSI information objects are created for each IoT device, and each IMSI information object will hold an information that points to the EPS Service Profile information object, see Figure 21. In case of having 100 IoT devices where each IoT device has a unique IMSI number,

there will be 100 IMSI information objects. Therefore, if these 100 IoT devices use the same EPS service, one EPS Service Profile information object will be created for each of them. These 100 IMSI information objects will hold an information element that points to this EPS Service Profile information object.

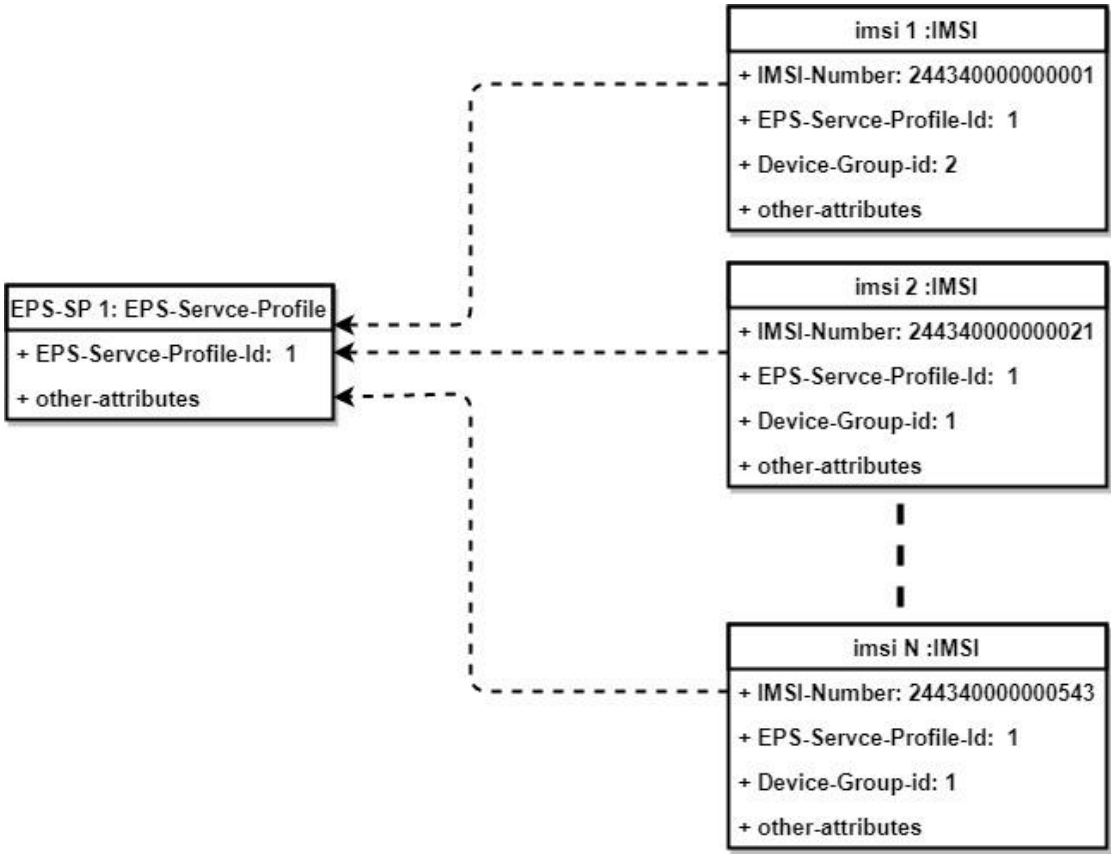


Figure 21. Object diagram for bulk subscription

The proposed bulk subscription solution is intended for bulk subscription of IoT devices owned by a single user. However, this solution cannot be used for bulk subscription of devices owned by different users in case one of the users wants to modify the EPS service. The fact that the IoT devices are associated to the EPS service, any change on each subscriber' EPS service will require the modification of the EPS Service Profile information object. Thus, since this EPS Service Profile information object is shared by other users, modifying it could affect the EPS service used by IoT devices of these users.

Figure 22 shows how the current EPS information model, shown in Figure 15, is extended to hold IoT device specific data. The 'MTCServiceProfile' class is added into the information model.

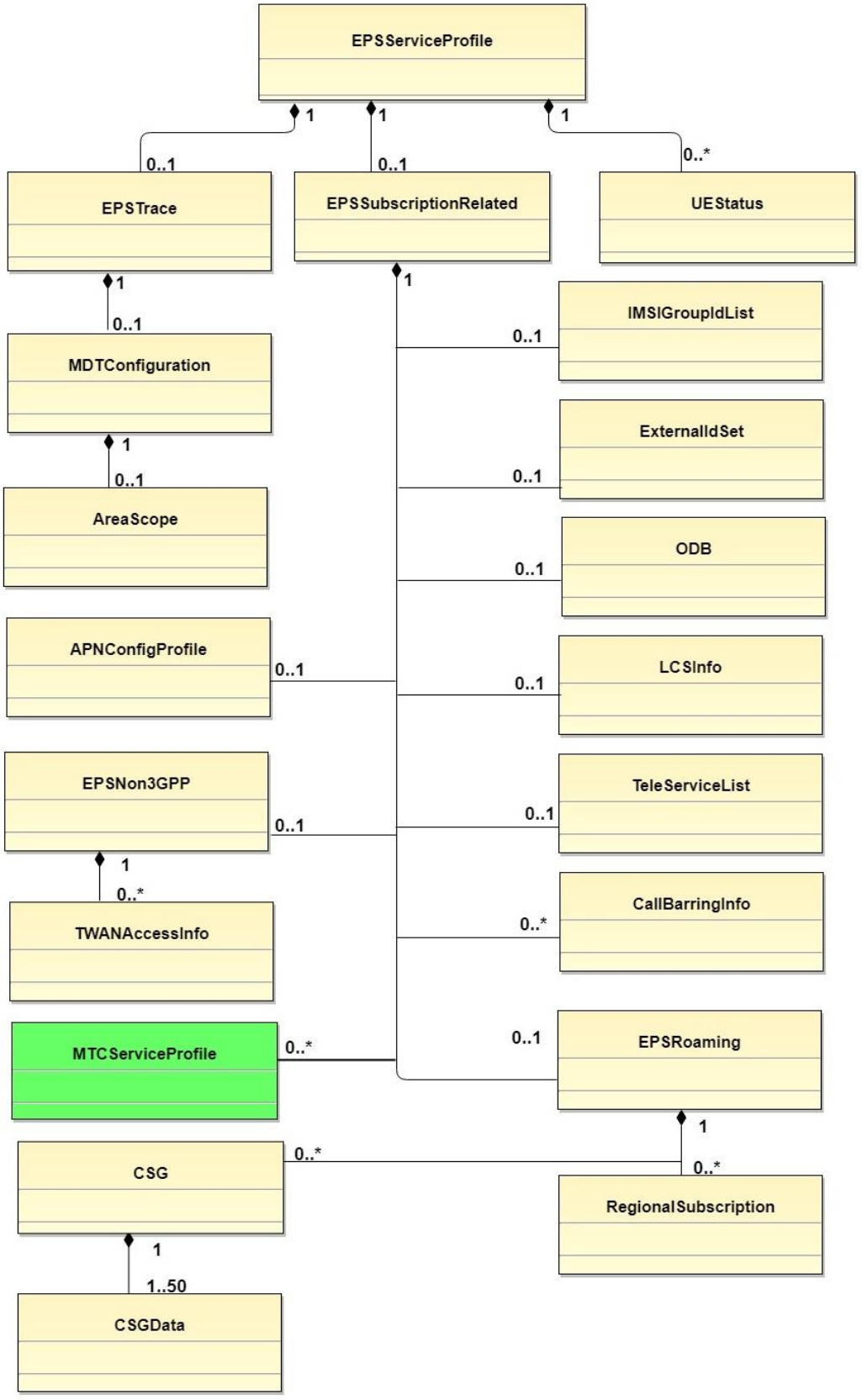


Figure 22. Information Model for EPS Service Profile including IoT devices data

The instances of this MTC Service Profile information class will hold IoT device specific EPS service data for a group of IoT devices. The grouping of IoT devices is based on the combination of IoT device specific EPS services that they commonly utilize. For example, in a set of IoT devices owned by a user, let's assume only a certain group of IoT devices use Small Data and Low Mobility services. An MTC Service Profile object, that holds Small Data and Low Mobility services, will be created for this group.

There could be more than one MTC Service Profile object under one EPS Service Profile information object. This is due to the fact that one user could have more than one group of IoT devices. Each MTC Service Profile object will be identified uniquely using a Device Group Id. This Id is unique within one EPS Service Profile information object. The IMSI object of each IoT device will hold the Device Group Id of the group the IoT device belongs to.

4.1.2 Network supported management of IoT devices

For the network to support management of IoT devices, it is necessary to identify the kind of data that is helpful for this purpose. Management of IoT devices in this context means automatic configuration and software update of these devices. The data identified as necessary are device id, IP address of the device, device server id, IP address of a device server, device location, status of device, software version of device and type of device.

Considering how the management process is performed was a key factor in identifying the necessary data. Two scenarios were considered, one is when the management process is initiated by the IoT device and the other is when it is initiated by some management server. When an IoT device initiates the process, it needs to know the IP address of the server that manages it. When the management server initiates the process, it could require device id, device IP address, device status, device type, device location and software version of device.

When an IoT device initiates the management processes, the proposed solution requires the presence of a device server. A device server in this context is any server whose IP address should be configured in the IoT device over the air. The device server could be a management server or a server that holds information about the management server. In addition to device management, the device server could also be used for storage of data collected by the IoT devices. The device server is identified by a device server id which is unique for a single user. The device server id will be set as a part of EPS service profile in the MTC service profile object. So IoT devices that belong to the same group will have the same device server as they share the same MTC service profile object. But as the same server id could be set in multiple MTC service profile objects, IoT devices that belong to different groups

could have the same device server. And if a user changes a device server, there is no need to change the device server id. Changing the IP address associated to the device server id is sufficient. As such the device server id is a permanent user data.

When the management server initiates the management process, the server needs a way to get IoT device information like IP address of the device from the UDR. This can be achieved through an Application Server FE or by a direct communication between the management server and the UDR. An Application Server FE can subscribe for the notification of these data on behalf of the management server. Or it can directly retrieve the data from the UDR when requested by the management server. For this to work, an Application Server FE shall be present in the network. This thesis does not explore the details about how this Application Server FE and the management server communicate. But the Application Server FE can communicate with the UDR using Ud interface as discussed in Chapter 2.

If the IP address of the IoT device is to be stored in the UDR, the management server shall be able to communicate with the device using this IP address. If IP address is not stored in the UDR, the management server can get the IP address of the device through other means like Domain Name server (DNS). As the IP address can vary, the management server does not use this IP to identify the device. It uses the device id or IMSI of the device for identification.

If the device id is stored in the UDR, the management server can use it to uniquely identify a single IoT device. This id shall be unique at least across the IoT devices managed by the management server. The EPS network shall get this id from the IoT device. The EPS network will not use this id for identification of the device, it only stores it in the UDR. If the device id is not stored in UDR, the management server identifies IoT devices using the IMSIs' of the devices. But this approach is laborious and inflexible. It will require configuration in the management server to map IoT devices with the Universal Subscriber Identity Module (USIM), which stores the IMSI, they are using. And Each USIM shall be inserted carefully to each IoT device as per the configuration. And in situation where a software Subscriber Identity Module (SIM) is used, the software SIM should be configured on each device carefully.

The configuration data of the IoT devices could be different for different type of IoT devices. The type of an IoT device can be determined based on the functionality the IoT device provides. A temperature sensor IoT device could have a type called 'temperature sensor'. Temperature sensor IoT devices and humidity sensor IoT devices owned by a single user may send the collected data to different servers and/or at different intervals. So, the configuration data will be different for these two types of IoT devices. The management server uses the device type information to determine which configuration data to send to the IoT device.

The configuration data could also vary depending on the location of the IoT device. As an example, the configuration data used for the IoT device could contain an IP address of the nearest (or appropriate) data collection server based on the location of the IoT device. Or the configuration data used could enable the IoT device display information in a certain language based on location. So, the management server could use the location information for such purposes.

The software version stored in the UDR can be used in two ways. In one case the management server can use the software version stored by an IoT device to determine whether to update the software of the device. In another case the management server can store in the UDR the latest software version the IoT device shall use. And the IoT device can check the software version it is currently using with the one that is stored in the UDR. If the current software version is old, the IoT device could initiates a software update procedure.

Finally, it should be noted that the proposed solution in this section is to help existing IoT device management systems, like the ones mentioned in section 2.3.2, in discovering and monitoring IoT devices. The proposed solution is not an IoT management system.

4.1.3 IoT device specific data information model

There is some difference between the identified data mentioned in the previous section and the data normally stored in the EPS service profile class. One difference is that the identified data is not relevant in determining the EPS services provided to the device. The other difference is that some of the identified data like IP address and Location could change more frequently. So, a separate Information class is defined to hold the identified data, see Figure 23.

Those IoT device specific data mentioned in section 4.1.1 affect the EPS service provided to the device. So, these data are stored in an Information class under EPS service profile information class, see Figure 23.

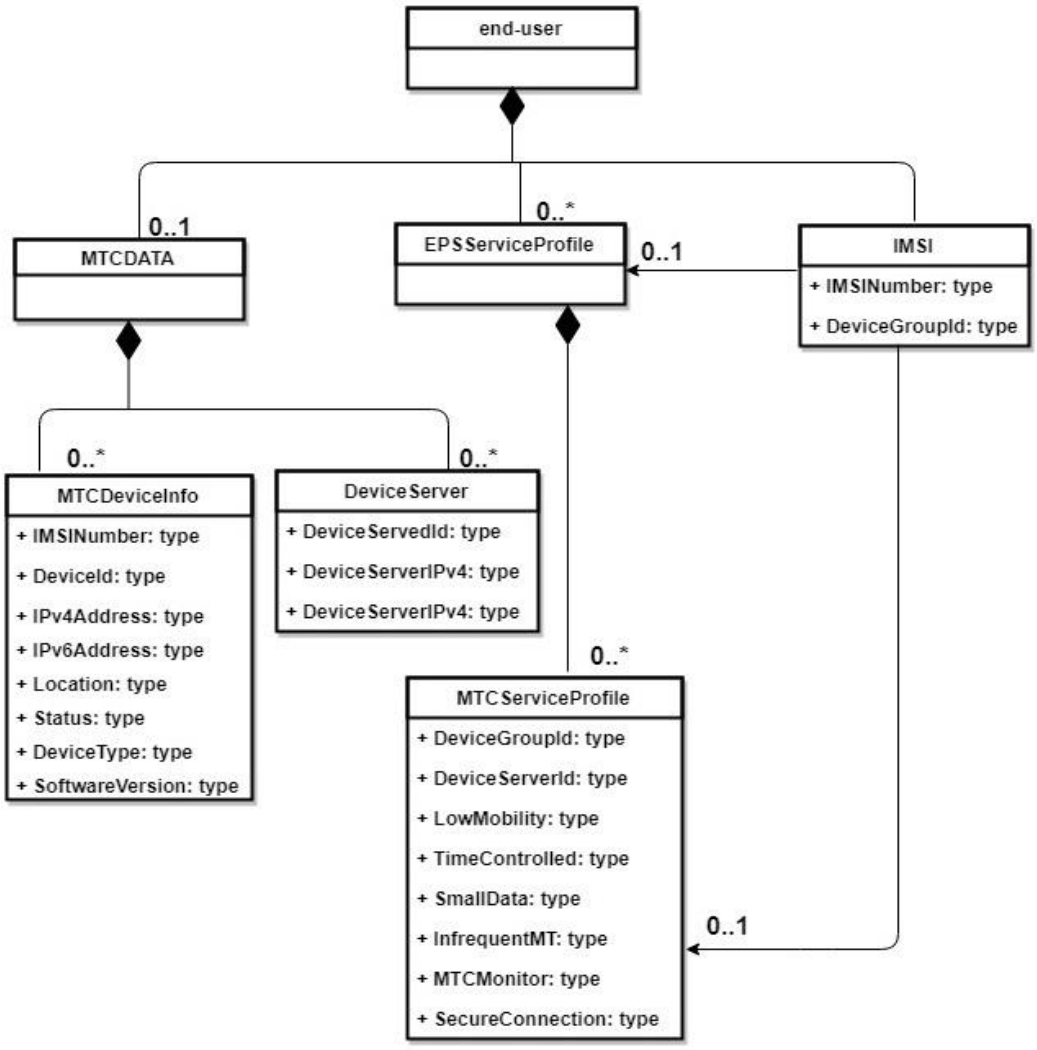


Figure 23. Information Model for IoT device specific data

4.2 UDR data model for IoT device data

The data model discussed in section 3.2 is extended based on the extended information model for IoT device specific data.

4.3 Limitations of the model

The information model is based on the UDC concept and it could be implemented in a UDR. As discussed in Chapter 2, the UDR has a standardized interface that enables access to the data in it. But the usefulness of the information model implemented in a UDR depends on the requirements which are not standardized. The model could only be useful in a customized EPS network that enables IoT devices to store and retrieve IoT device specific data.

5. Performance Results

This chapter provides an analysis of the performance of current UDC when used for managing end user UE and other devices (e.g. sensors/actuators) connected to mobile networks. This section will describe the setup to perform the performance testing and identify limitations with the current design of UDC.

5.1 UDC testbed

In this chapter we set up a testbed to measure the performance of the UDC extensions for IoT devices. The test bed is set up as shown in Figure 24 and consists of one laptop and three servers interconnected through a switch. The prototype UDR and the EPS HSS FE are running in the same laptop depicted in the upper part of Figure 24. In order to emulate the IoT device subscription two MME emulators are running in two different servers depicted in the lower part of Figure 24.

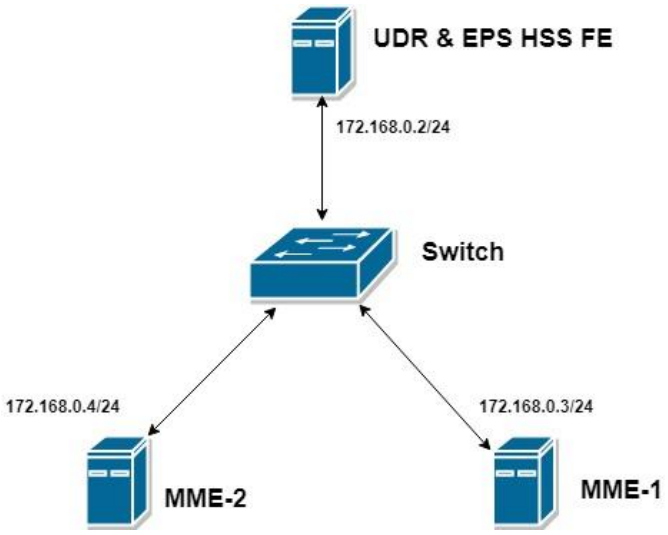


Figure 24. Testbed setup

The UDR and the EPS HSS FE are running on a ‘Lenovo Legion Y520’ laptop computer, which uses Centos 7 operating system. The reason for using Centos 7 is because it is the default operating system recommended to set up 389 DS that the UDR uses to store data.

In order to measure large setup of device subscription without real IoT devices, a test software which emulates an MME is developed. This emulator only generates and sends S6a commands to the EPS HSS FE which consist of the device subscription or attach. It does not provide any other functionality like processing other requests from

an eNodeB. The two MME emulators, ‘MME-1’ and ‘MME-2’, are deployed on two separate servers as shown in Figure 24 running Ubuntu 16.04 operating system.
The two ubuntu machines and the centos machine are connected using a network Switch. Each machine connects to the Hub using one physical Ethernet interface. The interfaces that the machines used to connect to the hub are configured to be in the same subnetwork.

5.2 Customizations required on standard EPS network

As mentioned in section 4.3, the model that supports IoT devices requires a customized EPS network. One of the customization required is on the S6a interface where 3GPP has no standardized commands and AVPs necessary to carry the IoT device specific data. The S6a has been extended to include the proposed information model for IoT devices. The current S6a commands i.e. are sufficient but the AVPs have to be extended to include the new information model in these commands. The AVP extensions for including the IoT device specific data are defined in section 5.2.1.

3GPP defines vendor id that is used in the prototype together with new AVP codes we define for IoT information model which are not standardized yet. The prototype will use experimental codes that 3GPP [39] has reserved for future use. 3GPP has defined the codes and AVP in [34] and [33].

5.2.1. AVPs extensions for IoT

Device-Data: This AVP is type Grouped. It holds IoT device data that are used for the management of IoT devices. Its AVP code is 1800

Device-Data ::= <AVP header: 1800 10415>
[Device-Id]
[Device-IPV4-Address]
[Device-IPV6-Address]
[Device-Type]
[Device-Software-Version]
[Device-Location]
**[Device-Status]*
**[AVP]*

Device-IPV4-Address: This AVP is type Address. It holds the IPv4 address of the IoT device assigned by the EPS network. Its AVP code is 1801

Device-IPV6-Address: This AVP is type Address. It holds the IPv6 address of the IoT device assigned by the EPS network. Its AVP code is 1802

Device-Type: This AVP is type UTF8String. It holds the device type value of the IoT device. Its AVP code is 1803

Device-Software-Version: This AVP is type UTF8String. It holds the software version the IoT device. Its AVP code is 1804

Device-Location: This AVP is type UTF8String. It holds the last known location of the IoT device. Its AVP code is 1805
Device-Status: This AVP is type UTF8String. It holds the status of the IoT device. The value in this AVP depends on the MTC-Monitor service the IoT device uses. If for example the IoT device uses MTC-Monitory service 'Loss Of Connectivity', the value in this AVP will be 'CONNECTIVITY_LOST' when connectivity with the IoT device is lost. Its AVP code is 1806

MTC-Subscription-Data: This AVP is type Grouped. It holds IoT device specific subscription data. Its AVP code is 1807

MTC-Subscription-Data ::= <AVP header: 1807 10415>
[Server-IPV4-Address]
[Server-IPV6-Address]
[Low-Mobility]
[Time-Controlled]
[Small-Data]
[Infrequent-MT]
**[MTC-Monitor]*
[Secure-Connection]
**[AVP]*

Server-IPV4-Address: This AVP is type Address. It holds the IPv4 address of the device server. Its AVP code is 1808

Server-IPV6-Address: This AVP is type Address. It holds the IPv6 address of the device server. Its AVP code is 1809

Low-Mobility: This AVP is type Enumerated. It is present when the IoT device has a subscription for Low Mobility. Its AVP code is 1810

Time-Controlled: This AVP is type Unsigned32. It holds a configuration identifier number. The configuration identified by this number holds the time control information of the IoT device. This identifier number shall be unique within a home network. Its AVP code is 1811

Small-Data: This AVP is type Enumerated. It is present when the IoT device has a subscription for small data. Its AVP code is 1812

Infrequent-MT: This AVP is type Enumerated. It is present when the IoT device has a subscription for Infrequent mobile terminated. Its AVP code is 1813

MTC-Monitor: This AVP is type Enumerated. It holds the type of MTC-Monitor service the IoT device has subscribed to. Its AVP code is 1814

Secure-Connection: This AVP is type Enumerated. It is present when the IoT device has a subscription for secure connection. Its AVP code is 1815

Device-Id: This AVP is type UTF8String. It holds id of the IoT device. Its AVP code is 1816

5.2.2. Modified existing S6a commands and AVPs

Notify Request Command: This command is modified to additionally hold 'Device-data'.

< Notify-Request> ::= < Diameter Header: 323, REQ, PXY, 6777251 >
[Device-Data]

Subscription-Data AVP: This AVP is modified to additionally hold 'MTC-Subscription-Data'.

Subscription-Data ::= <AVP header: 1400 10415>
[MTC-Subscription-Data] 1807

5.3 Performance test setup

The testing scenario is depicted in the Figure 25 where we emulate the sequence flow used when deploying NB-IoT by mobile operator. Firstly, the mobile operator will register the IMSI of the NB-IoT devices in the HSS which is now part of the UDC as represented with transaction 1 in Figure 25. When the NB-IoT is deployed in the field and register or attach to the network for the first time as represented in transaction 2. In this first attach request, the MME has to request all the information from the UDC as show in transaction 3. After this first attach the device information is stored locally in the MME database for subsequence authentication/authorization requests. After the initial attach the NB-IoT sensor is available for accessing data from the sensor operator, which can subscribe to events associated to the sensor with transaction 4 or receive the data from the sensor represented with transaction 5.

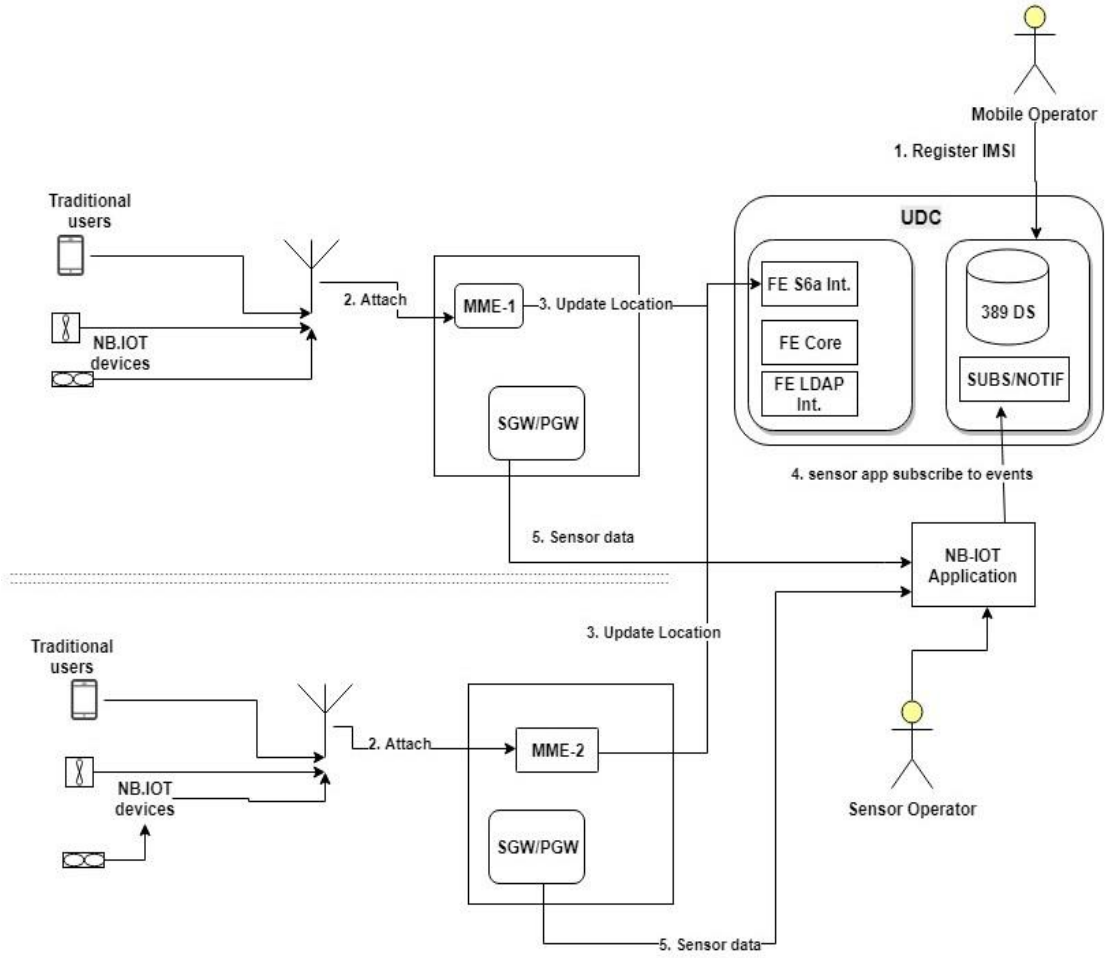


Figure 25. Testbed flow diagram

In this performance test two types of devices are considered to show the effect of the information model added to UDC that supports IoT devices. The first type of devices are normal UE that use existing 3GPP defined AVPs for traditional user subscription. There is different EPS Service Profile entry in the DS for each UE that registers to the network.

For the second type of devices the MME will emulate S6a commands coming from NB-IoT including the newly defined AVPs for sensor subscriptions.

In addition to the difference in AVPs we assume that all the IoT devices had bulk subscription for the same EPS service. Thus, the EPS Service Profile data fetched form the DS during processing of each S6a command for each IoT device is the same EPS Service Profile entry.

The performance test will measure the delay in accessing device information during the attach process represented with transaction 3 in Figure 25. This transaction consists of S6a command named Update Location Request (ULR) that MME sends to the HSS Application FE when a new device registers or attaches to the network. This

command is chosen because the AVP in the message is modified to support IoT devices compared to the command used by normal subscriber's UE. Moreover, the ULR is delay sensitive since the UE should receive an attach response within 15seconds according to the timer T3410 defined in 3GPP specifications [40]. The transport protocol used on the S6a interface is Stream Control Transmission Protocol (SCTP) [41].

Table 4. The performance test includes the following test cases with different number of NB-IoT devices and traditional UE.

Test Case	Traditional UE pre-registered in UDR (1)	NB-IoT devices pre-registered in UDR (2)	Total ULR commands sent to UDR (3)	ULR commands/sec per MME for Traditional UE (4)	ULR commands/sec per MME for NB-IoT dev. (4)
#1	1000	1000	10000	50	0
#2	1000	1000	10000	0	50
#3	2000	2000	10000	50	0
#4	2000	2000	10000	0	50
#5	5000	5000	10000	50	0
#6	5000	5000	10000	0	50
#7	1000	1000	10000	500	0
#8	1000	1000	10000	0	500

(1) Each traditional UE is registered in UDR with own separate EPS service profile.
(2) Each NB-IoT device is registered in UDR sharing the same EPS service profile.
(3) The ULR commands sent to the UDR are split between the two MME emulators.
(4) Each MME emulator sends N ULR commands/ second which is equivalent to have N devices attach/second.

For IoT devices, 24 entries (i.e. EPS service profile entries) that are shared by all IoT devices are inserted in directory server database. Additionally, one entry (i.e. IMSI information entry) is inserted for each IoT device. Whereas for traditional users, a total of 24 entries are inserted in to directory server database for each traditional user. Additionally, one entry (i.e. IMSI information entry) is inserted for each traditional user.

5.3.1 Performance test results

In the performance the transmission delay and the propagation delay are insignificant. This is because the distance between the machines is less than 1m and the link capacity interconnecting them is 1Gbs which is very high compared to the Update Location Request/Answer command packet sizes.

The delay is analyzed for the three test cases described in section 5.3. For analyzing the delay, ULR and Update Location Answer (ULA) command packets are captured using tcpdump [42] on the machines running the HSS FE and the 389 DS. Then the captured packets were analyzed using network protocol analyzer tool Wireshark [43].

Table 5. ULR processing delay test result for test scenarios #1 and #2

TEST CASE	Delay of 90% of commands is less than (ms)	Minimum delay(ms)	Maximum Delay(ms)	Average Delay(ms)
# 1	10	1	590	10
# 2	10	1	585	11

Table 6. ULR processing delay test result for test scenarios #3 and #4

TEST CASE	Delay of 90% of commands is less than (ms)	Minimum delay(ms)	Maximum Delay(ms)	Average Delay(ms)
# 3	12	1	504	11
# 4	16	1	310	11

Table 7. ULR processing delay test result for test scenarios #5 and #6

TEST CASE	Delay of 90% of commands is less than (ms)	Minimum delay(ms)	Maximum Delay(ms)	Average Delay(ms)
#5	13	1	427	11
#6	2215266	959	2436008	1285809

Table 8. ULR processing delay test result for test scenarios #7 and #8

TEST CASE	Minimum delay(ms)	Maximum Delay(ms)	Average Delay(ms)
#7	4	8469	3959
#8	4	13243	6189

The information in the above tables only gives some key performance measurement parameters. To have a greater visualization of the ULR processing delay, the tool R [44] is used to plot the distribution of the delay and the trend of the delay based on the amount of ULR commands received.

Figure 26 shows the distribution of ULR command processing delay for Test Cases #1 to #6. It helps to understand the range of the delay when processing most of the ULR commands.

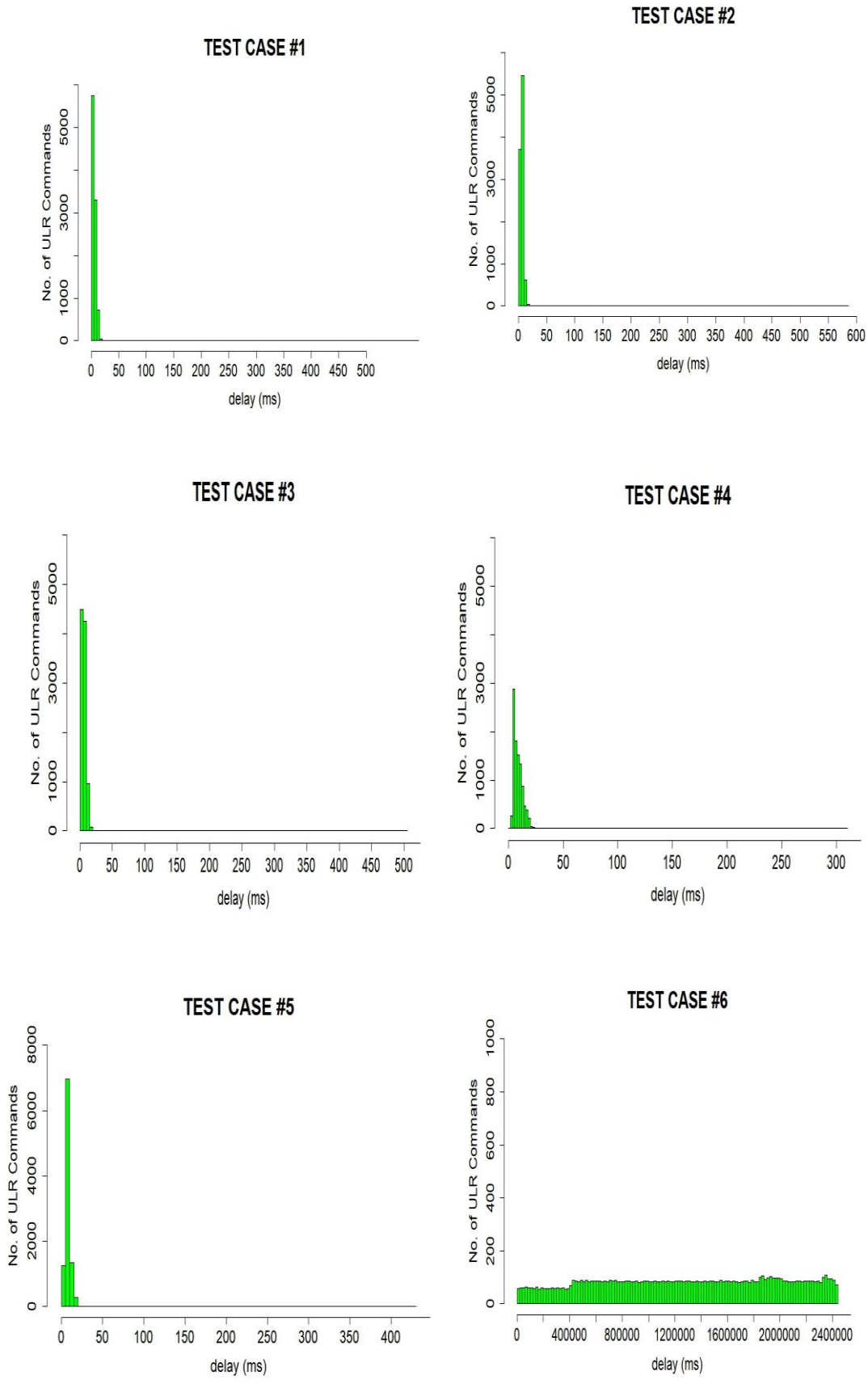


Figure 26. ULR command processing Delay distribution for Test Case #1 - #6

Figure 27 shows how the ULR command processing delay is affected as more and more ULR commands arrive for Test Cases #1 to #6. This helps in understanding the trend in ULR processing delay of the system overtime under similar load conditions.

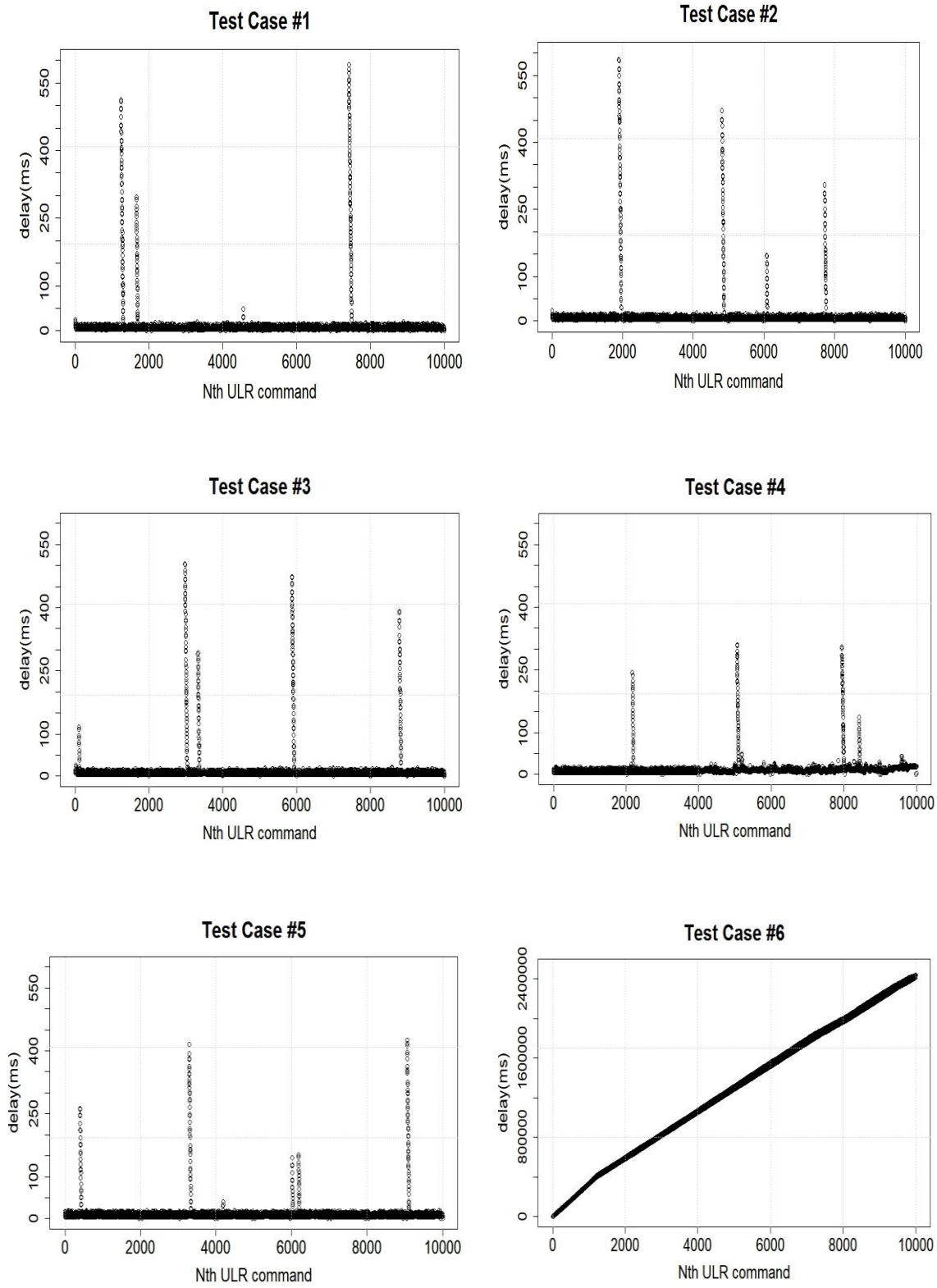


Figure 27. ULR command processing delay for Test Case #1 to #6

5.3.2 Analysis

The results show the difference in delay when processing ULR command for IoT and traditional users. The results in Table 5,6 and 7 indicate the increase in number of entries in the LDAP server affects the processing delay for IoT devices more severely than for traditional users. The other difference observed, see Figure 26, is that the delay for IoT is more equally distributed than the delay for traditional devices.

In Figure 27, very interesting results are observed for Test scenario 6 where the processing delay for IoT device subscription increases proportionally to the amount of ULR commands received. This test case shows that the prototype UDR has severe limitations for managing IoT devices where there is a high amount of subscriptions (i.e. 5000 traditional users and 5000 IoT devices). Whereas as can be observed in the same figure, the delay is more or less the same for Test cases 1,2,3,4 and 5 as more and more ULR commands are received. This shows when the system is stable under load, so it can be used for both traditional and IoT users.

The results presented in Table 8 show the prototype system performs badly under a higher load condition. The results show the system performs badly and it can affect the attach request timer requirement mentioned in section 5.3.

After observing the delay in the different test cases, the next step was determining the main cause of the delay difference between IoT and traditional users. To do this, further analysis was done on the LDAP search and modify request and response messages. The HSS Application FE sends multiple LDAP requests towards the LDAP server during a processing of a single ULR command. So, analyzing the delay on processing these messages was important.

For this analysis packets were captured on the loopback interface of the machine where the HSS FE and the 389 DS are running on. These packets were captured while performing the tests mentioned in the previous section. Then the captured packets were analyzed using network protocol analyzer tool Wireshark. The result of the analysis shows the main cause of the delay was the delay caused by 389 DS to processes LDAP requests. As a sample, the LDAP delay for Test case #5 and Test case #6 is presented in table 9 and 10 respectively.

As can be seen from the tables there are some 8000 more search requests in Test case #6. The reason for this is because in Test case #6, which is for IoT devices, the HSS FE tries to fetch device service profile. There is no device service profile for traditional UE. On top of the additional Search requests the average delay for a Search request in case of Test case #6 is almost 100 times more than that is in Test case #5.

Table 9. LDAP request processing delay for test scenario #5

LDAP Request Type	Number of Requests	Minimum delay(ms)	Maximum Delay(ms)	Average Delay(ms)
Modify	19937	0.253	255.021	0.469
Search	419225	0.050	420.963	0.278

Table 10. LDAP request processing delay for test scenario #6

LDAP Request Type	Number of Requests	Minimum delay(ms)	Maximum Delay(ms)	Average Delay(ms)
Modify	19962	0.283	92.830	0.558
Search	443275	0.044	2216.631	28.999

In Figure 27, it is observed that the jitter for Test Cases #1 to #5 very high at some points. To find the source of this jitter pattern, LDAP modify request delays for Test Case #1 and Test Case #2 were analyzed. The result shows modify requests delay exhibits a similar jitter pattern as the ULR delay in Test Case #1 and Test Case #2, see Figure 27 and Figure 28. This shows the jitter could be caused by the LDAP server.

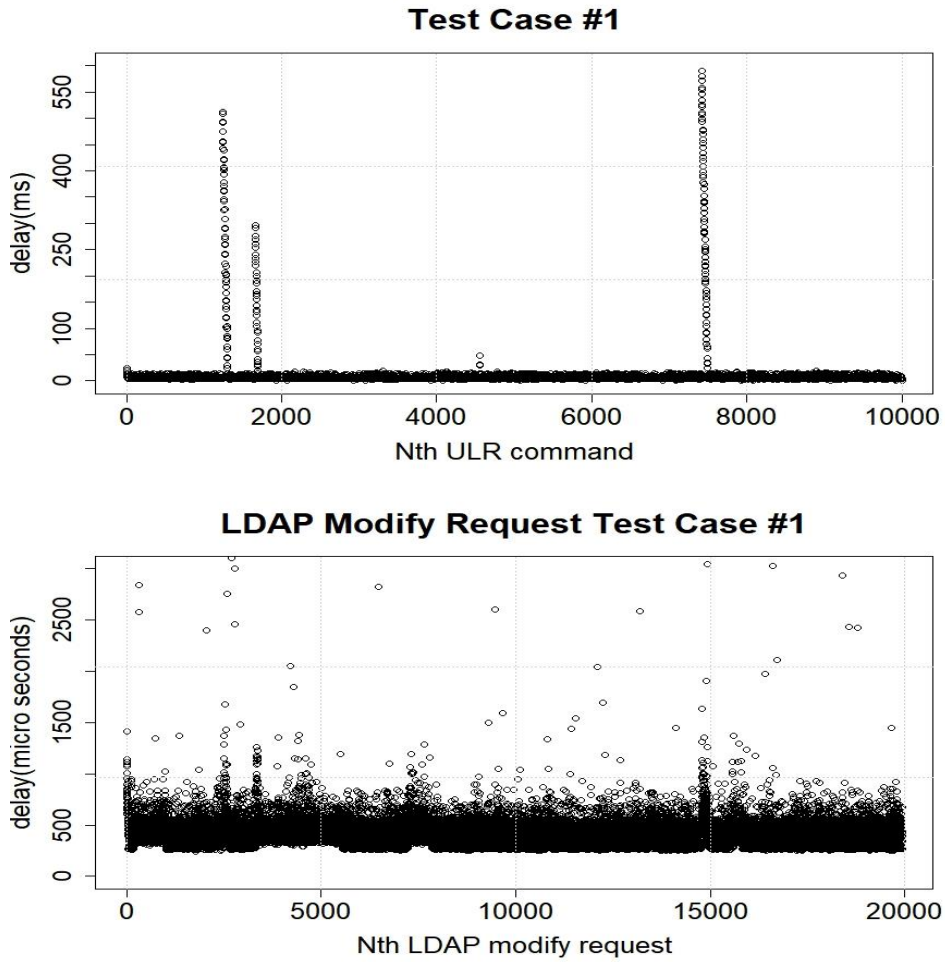


Figure 28. LDAP modify request and ULR jitter comparison for Test Case #1

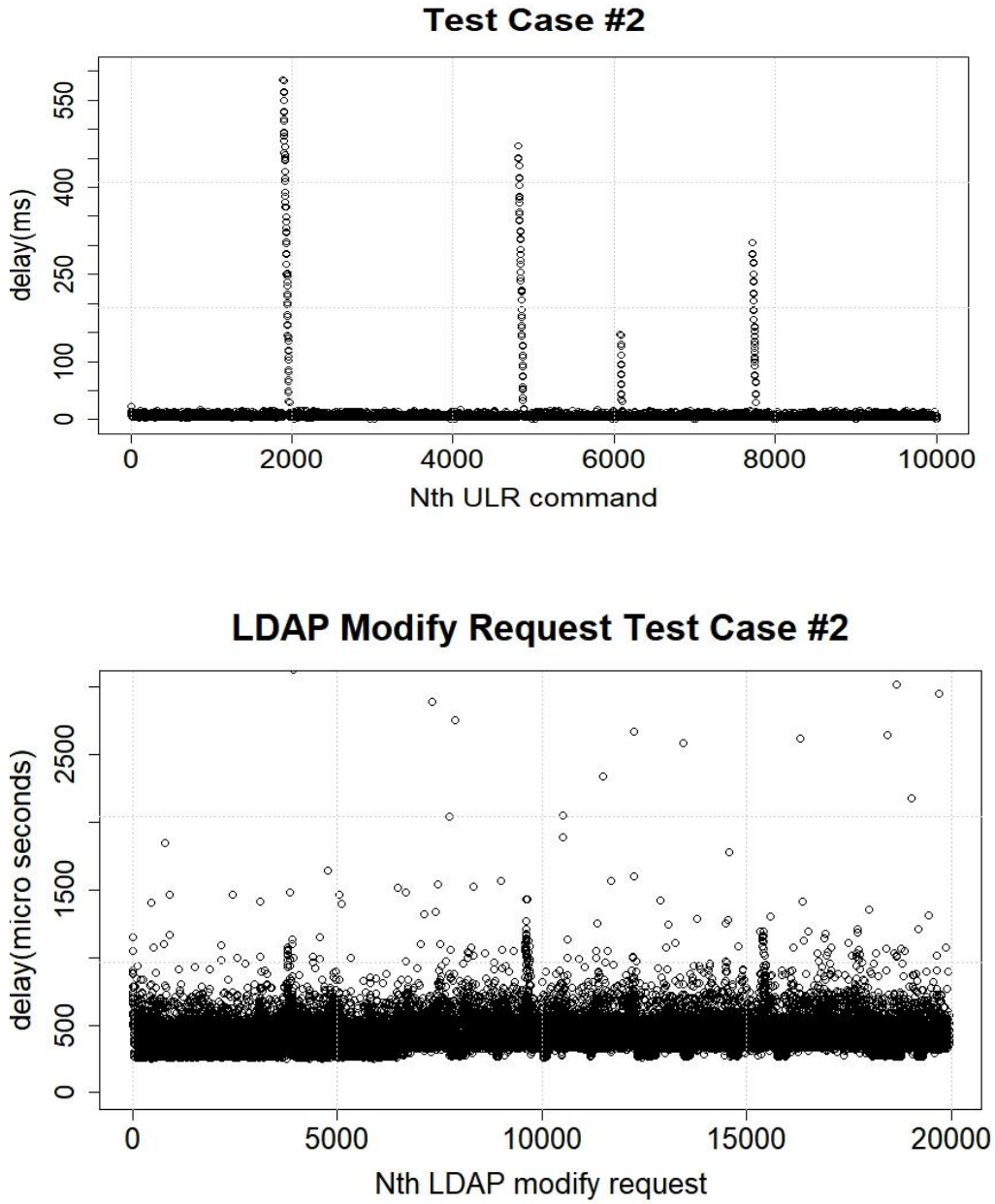


Figure 29. LDAP modify request and ULR jitter comparison for Test Case #2

After this finding, a further analysis was done on the overall system by using a linux system monitoring tool vmstat[8]. The tool vmstat was used to gather current system information in two cases. In the first case neither the LDAP server nor the EPS HSS FE were running. In the second case both the LDAP server and the EPS HSS FE were running, and Test Case #1 was performed. The first case was done to differentiate the effect, on the system information, of running Test Case #1. The result of the analysis shows the block sent to a block device increased significantly every time those high jitters occur. This shows the LDAP server is writing significantly more data into the

[8] https://linux.die.net/man/8/vmstat

disk. And generally, writing to or reading data from disk degrades LDAP server performance [45]. This indicates those jitters could be caused by the LDAP server.

Another thing observed was that the delay increased when the number of entries in the 389 DS is increased. Also, it increased when the frequency of the ULR command received is increased. This delay can be explained by an LDAP performance analysis done in [45]. It shows how the number of entries in an LDAP server and the frequency of LDAP request affect an LDAP server performance. It shows an increase in the number of entries and/or frequency of LDAP requests negatively affects the performance. This can explain why there is an increase in delay observed as the number of entries increase and as the ULR command sent increases from 5 per second to 50 per second.

5.4 Conclusion

The result of the test on the prototype shows the difference in delay for IoT devices in comparison to traditional users increases as the number of IoT device which share the same EPS service profile increases. The result shows the way the LDAP server operates becomes a bottleneck on the system. This could be improved by deploying the LDAP server on high performance system as indicated in [45]. And also, by optimizing the prototype HSS Application FE code for higher performance.

Overall, the result shows the feasibility of designing a user data storage system for future mobile networks that besides managing the traditional HSS data takes into consideration the needs of IoT devices. The prototype used for testing can manage at least 2000 traditional users and 2000 IoT devices.

6. Conclusions and Future Work

6.1 Summary

The main goal of the thesis, which is designing a user data storage system taking into account IoT devices, is achieved. The designed storage system allows bulk subscription of IoT devices and additionally facilitates management of IoT devices. As there is no existing standard EPS system that can utilize the designed system, an EPS HSS Application is developed to test the system.

To test the design a prototype was developed using an LDAP server called 389 Directory Server to implement the user data storage. EPS service profile was defined for traditional and IoT devices. These entries were inserted in the LDAP server and the system was tested by sending ULR commands to the EPS HSS Application. The ULR commands are sent to the LDAP server by the EPS HSS Application. The delay between the ULR commands received by the Application and the corresponding ULA message sent by the EPS HSS Application is measured. This delay is analyzed to test the performance of the prototype system.

The result of the test shows the feasibility of the design. One issue observed during the test is that the LDAP server becomes the bottleneck on the performance of the designed system. It is advised that the LDAP server is properly optimized and there is sufficient processing power and memory for the server to work optimally.

6.2 Future works

In the proposed design some IoT device data like IP address, Location and device type can be stored in the user data storage. The EPS network can facilitate a secure automatic discovery of IoT devices that belong to the same user by providing these data to the devices based on their subscription. The feasibility and effect of this service can be studied further.

There could be many IoT device management servers. These servers need to subscribe for notification to get IoT device data stored in the UDR. As these servers are only allowed to get certain device data of certain IoT devices, there should be a system that manages the access rights of these servers. For this purpose, an Application Front End can be developed. The access right can be managed by this application and the application can subscribe to notification on behalf of the management servers based on their access right. This application can be placed in the operator network. The feasibility and details of such an application can be studied further.

Reference

[1] O. Teyeb, G. Wikström, M. Stattin, T. Cheng, S. Faxér, and H. Do, "Evolving LTE to fit the 5G future," Ericson Technology Review, Jan. 31, 2017. [Online]. Available: https://www.ericsson.com/en/publications/ericsson-technology-review/archive/2017/evolving-lte-to-fit-the-5g-future. [Accessed Apr 09, 2018].

[2] N. Narang and S. Kasera, *2G Mobile Networks GSM and HSCSD*. New Delhi, India: Tata Mac-GrowHill, 2007.

[3] 3GPP, "Network architecture," TS 23.002 V12.5.0, Sep. 2014.

[4] 3GPP, "Service requirements for the User Data Convergence (UDC)," TR 22.985 V14.0.0, Mar. 2017.

[5] 3GPP, "User Data Convergence (UDC); Technical realization and information flows; Stage 2," TS 23.335 V14.0.0, May 2017.

[6] 3GPP, "User Data Convergence (UDC); User data repository access protocol over the Ud interface; Stage 3," TS 29.335 V14.0.0, Apr. 2017.

[7] J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol," RFC 4511, June 2006.

[8] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. "Simple Object Access Protocol (SOAP) 1.1," W3C Note, 08 May 2000. [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508. [Accessed Apr. 7, 2018].

[9] ITU-T, "Information Technology – Open Systems Interconnection – The Directory: Models," X-501 Recommendation, 2005.

[10] RACLE, "Sun Directory Server Enterprise Edition 7.0 Administration Guide," [Online]. Available: https://docs.oracle.com/cd/E19424-01/820-4809/bcadz/index.html. [Accessed Feb 22, 2018]

[11] ZyTrax, "Chapter 7 Referrals, Aliases & Proxies (Chaining)," [Online]. Available: http://www.zytrax.com/books/ldap/ch7/referrals.html. [Accessed Feb 22, 2018]

[12] ZyTrax, "LDAP Concepts & Overview," [Online]. Available:

http://www.zytrax.com/books/ldap/ch2/index.html#database.
[Accessed Feb 16, 2018]

[13] Microsoft, “Understanding the Role of Directory Services Versus Relational Databases,” Sep. 2009. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2000/bb727070(v=technet.10).
[Accessed Feb 16, 2018]

[14] UnboundID, “Why Use LDAP,” [Online]. Available: https://www.ldap.com/why-use-ldap. [Accessed Feb 16, 2018]

[15] E. Grafström, “Resilient and optimized LDAP database implementation for a large scale HLR/HSS,” M.S. thesis, Uppsala University, Uppsala, Sweden, 2012.

[16] O. F. Sogunle, “A Unified Data Repository for Rich Communication Services,” M.S. thesis, Rhodes University, South Africa, 2016.

[17] D. Chappell, *Introducing OData: Data access for the web, the cloud, mobile devices, and more*. Microsoft Whitepaper, May 2011.

[18] J. T. Hackos, “WHAT IS AN INFORMATION MODEL & WHY DO YOU NEED ONE?,” *Content, Computing, and Commerce – Technology & Trends*, vol. 10, No. 1, February 2002. [Online]. Available: https://gilbane.com/artpdf/GR10.1.pdf. [Accessed Feb 21, 2018]

[19] UML specification website: http://www.omg.org/spec/UML

[20] L. D. Xu, W. He and S. Li, "Internet of Things in Industries: A Survey," in *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233-2243, Nov. 2014.

[21] R. Minerva, A. Biru and D. Rotondi, *Towards a definition of the Internet of Things (IoT)*. IEEE Internet Initiative, May 2015.

[22] M. Chen, Y. Miao, Y. Hao and K. Hwang, "Narrow Band Internet of Things," in *IEEE Access*, vol. 5, pp. 20557-20577, 2017.

[23] G. A. Akpakwu, B. J. Silva, G. P. Hancke and A. M. Abu-Mahfouz, "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges," in *IEEE Access*, vol. 6, pp. 3619-3647, 2018.

[24] 3GPP, “FS_SMARTER – massive Internet of Things,” TR 22.861 V14.1.0, Sep. 2016.

[25] *3GPP LOW POWER WIDE AREA TECHNOLOGIES*, GSMA WHITE PAPER

[26] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos and P. Christen, "Sensor discovery and configuration framework for the Internet of Things paradigm," *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Seoul, 2014, pp. 94-99.

[27] C. Perera, P. Jayaraman, A. Zaslavsky, P. Christen and D. Georgakopoulos, "Dynamic configuration of sensors using mobile sensor hub in internet of things paradigm," *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, Melbourne, VIC, 2013, pp. 473-478.

[28] P. Waher, and R. Klauck, “Internet of Things-Discovery,” XEP-0347, Sep. 2017.

[29] Amazon, “Amazon AWS IoT Device Management Features,” [online]. Available: https://aws.amazon.com/iot-device-management/features/. [Accessed Mar 22, 2018]

[30] Microsoft, “Microsoft Azure IoT Hub,” [online]. Available: https://azure.microsoft.com/en-us/services/iot-hub/. [Accessed Mar 22, 2018]

[31] A. Pras, and J. Schoenwaelder, “On the Difference between Information Models and Data Models,” RFC 3444, January 2003.

[32] 3GPP, “Telecommunication management; User Data Convergence (UDC); Common baseline information model (CBIM),” TS 32.182 V14.0.0, Apr. 2017.

[33] 3GPP, “Evolved Packet System (EPS); Mobility Management Entity (MME) and Serving GPRS Support Node (SGSN) related interfaces based on Diameter protocol, ” TS 29.272 V13.8.0, Jan. 2017.

[34] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, “Diameter Base Protocol”. RFC 6733, Oct. 2012.

[35] NetIQ, “NetIQ OpenLDAP – Based SDK and Extended Library,” [Online]. Available: ftp://sdk.provo.novell.com/ndk/cldap/builds/2016/NetIQ-OpenLDAP-based-SDK.pdf. [Accessed Feb 26, 2018]

[36] M. Muehlfeld, P, Bokoč, T. Čapek, and Ella Deon Ballard, “RED HAT DIRECTORY SERVER 10, PLUG-IN GUIDE,” [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_directory_server/10/html/plug-in_guide/index. [Accessed Feb 27, 2018]

[37] 389 Directory Server, “Plugin Design,” [Online]. Available: http://directory.fedoraproject.org/docs/389ds/design/plugins.html. [Accessed Feb 27, 2018]

[38] 3GPP, “Service requirements for Machine-Type Communications (MTC); Stage 1,” TS 22.368 V14.0.1, Oct. 2017.

[39] 3GPP, “Diameter applications; 3GPP specific codes and identifiers,” TS 29.230 V14.7.0, Jan. 2018.

[40] 3GPP, 2017. “Non-Access-Stratum(NAS) protocol for Evolved Packet System(EPS); Stage 3”. TS 24.301, 3rd Generation Partnership Project (3GPP).

[41] R. Stewart, “Stream Control Transmission Protocol,” RFC 4960, Sep. 2007.

[42] TCPDUMP website: http://www.tcpdump.org/

[43] Wireshark website: https://www.wireshark.org/

[44] R website: https://www.r-project.org/

[45] X. Wang, H. Schulzrinne, D. Kandlur and D. Verma, "Measurement and Analysis of LDAP Performance," in *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 232-243, Feb. 2008.