Software Defined Networking and Tunneling for Mobile Networks

BINGHAN LIU

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Software Defined Networking and Tunneling for Mobile Networks

Binghan Liu

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

February 17, 2012

Examiner: Professor Gerald Q. Maguire Jr.

Abstract

With the deployment of Long Term Evolution (LTE) networks, mobile networks will become an important infrastructure component in the cloud ecosystem. However, in the cloud computing era, traditional routing and switching platforms do not meet the requirements of this new trend, especially in a mobile network environment. With the recent advances in software switches and efficient virtualization using commodity servers, Software Defined Networking (SDN) has emerged as a powerful technology to meet the new requirements for supporting a new generation of cloud service.

This thesis describers an experimental investigation of cloud computing, SDN, and a mobile network's packet core. The design of a mobile network exploiting the evolution of SDN is also presented. The actual implementation consists of a GTP enabled Open vSwitch together with the transparent mode of mobile network SDN evolution. Open vSwitch is a SDN product designed for computer networks. The implementation extends Open vSwitch with an implementation of the GTP protocol. This extension enables Open vSwitch to be an excellent SDN component for mobile networks. In transparent mode, a cloud data center is deployed *without* making any modification to the existing mobile networks. In the practical evaluation of the GTP-U tunnel protocol implementation, the measured metrics are UDP and TCP throughput, end-to-end latency and jitter. Two experiments have been conducted and described in the evaluation chapter.

Cloud computing has become one of the hottest Internet topics. It is attractive for the mobile network to adopt cloud computing technology in order to enjoy the benefits of cloud computing. For example, to reduce network construction cost, make the network deployment more flexible, etc. This thesis presents an potential direction for mobile network cloud computing. Since this thesis relies on open source projects, readers may use the results to explore a feasible direction for mobile network cloud computing evolution.

Sammanfattning

Med utbyggnaden av långa (LTE) Term Evolution nätverk, mobila nätverk kommer blivit en viktig infrastruktur komponent i molnet ekosystemet. Men i cloud computing eran, uppfyller traditionella routing och switching plattformar inte kraven i denna nya trend, särskilt i ett mobilnät miljö. Med de senaste framstegen i programvara växlar och effektiv virtualisering påråvaror servrar, programvarustyrd Nätverk (SDN) har utvecklats till en kraftfull teknik för att möta de nya kraven för att stödja en ny generation av molntjänst.

Denna avhandling beskrivarna en försöksverksamhet inriktad undersökning av cloud computing, SDN och ett mobilnät är Packet Core. Utformningen av ett mobilnät utnyttja SDN utveckling presenteras också. Det faktiska genomförandet består av en GTP aktiverad Open Vswitch tillsammans med transparent läge av mobilnätet SDN evolution. Öppna Vswitch är en SDN-produkt avsedd för datornätverk. Genomförandet utökar Open Vswitch med en implementering av GTP-protokollet. Denna uppgradering gör Open Vswitch vara som en utmärkt SDN komponent för mobila nätverk. I transparent läge är ett moln datacenter utplacerade *utan* göra eventuella ändringar till befintliga mobilnät. I den praktiska utvärderingen av GTP-U tunnel protokollimplementering, de uppmätta mått är UDP och TCP genomströmning, end-to-end-latens, jitter och paketförluster. Tvåexperiment har utförts i utvärderingen kapitlet.

Cloud computing har blivit en av de hetaste av Internet. Således kan framtiden för det mobila nätet ocksåanta teknik cloud computing och dra nytta av cloud computing. Till exempel minska kostnaderna nätbyggnad, gör nätverket distribuera mer flexibla, etc. .. Denna avhandling presenterar en möjlig inriktning för mobilnät cloud computing. Eftersom denna avhandling bygger påopen source-projekt, läsarna använda resultatet av den att utforska möjliga riktning mobilnät cloud computing utveckling.

Acknowledgements

The completion of this thesis would not have been possible without the support of my supervisor Professor Gerald Q. Maguire Jr.. I wish to give my utmost thanks to him for his patient guidance. His guidance helped me all the time in researching and writing this thesis. His rigorous scholarship showed me how to do my research in a proper way.

My sincere thanks also goes to my industrial supervisors, Azimeh Sefidcon, Bob Melander, and Enrique Fernandez. I thank them for providing the initial idea for this thesis. Without them, this thesis was almost impossible. It was a really good learning experience working under them.

I have been fortunate enough to have the help of my friend Sriniva Vinay Yadhav. I wish to thank him for supporting me.

My deepest thanks goes to my family, for their unconditional support and understanding throughout my study abroad.

Contents

List of Figures

List of Tables

List of listings

List of Acronyms and Abbreviations

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
ARP	Address Resolution Protocol
CAPWAP	Control And Provisioning of Wireless Access Points
CDN	Content Distributed Network
DHCP	Dynamic Host Conguration Protocol
DPIF	Datapath Interface
EGP	Exterior Gateway Protocol
EPC	Evolved Packet Core
ePDG	Evolved Packet Data Gateway
GPRS	General packet radio service
GRE	Generic Routing Encapsulation
GTP	GPRS tunneling protocol
GTP-C	GPRS tunneling protocol control plane
GTP-U	GPRS tunneling protocol user plane
HSS	Home Subscriber Server
IGP	Interior Gateway Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4

IPv6	Internet Protocol version 6
LTE	Long Term Evolution
MAC	Media Access Control
MME	Mobility Management Entity
N-PDU	Network-Protocol Data Unit
NIC	Network Interface Card
PCRF	Policy and Charging Rules Function
PDN	Packet Data Network
PGW	PDN Gateway
QoS	Quality of Service
RTT	Round Trip Time
SAE	System Architecture Evolution
SDN	Software Defined Networking
SGSN	Serving GPRS Support Node
SGW	Serving Gateway
STP	Spanning Tree Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TEID	Tunnel Endpoint ID
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network

Chapter 1

Introduction

1.1 Motivation

Cloud computing has had a large effect on the entire Internet. Nowadays, there exist services including data storage, audio and video entertainment, and even programming platforms, which are provided in the cloud. Subscribers access these services from a heterogeneous set of devices and networks. However, with respect to the mobile network itself, the cloud evolution seems to move forward only very slowly. The latest mobile network concepts, such as packet core evolution, System Architecture Evolution (SAE), and Evolved Packet Core (EPC), mainly focus on integrating the various access networks[1]. The evolution of cloud computing is not within the scope of 3GPP's discussions. With the large-scale deployment of Long Term Evolution (LTE)[2] networks, more and more services, such as voice over IP and video streaming, are available via mobile networks. Mobile phones have became one of the most popular devices for accessing the Internet. Due to the advantages of cloud computing[3], the mobile network should itself adopt the cloud computing paradigm. For instance, we can deploy computational nodes in the cloud, rather than placing them within the operator's core network. In fact the Mobility Management Entity (MME), Serving Gateway (SGW), and Packet Data Network Gateway (PGW), ..., and mobile network control entities, can be virtualized and migrated to the cloud. This transitions would make the mobile network more flexible, while reducing the cost of evolving the mobile network and accelerating innovation of mobile networks. Migrating most of these services and network entities into the cloud can make the mobile network more reliable, scalable, and efficient.

We need to address some significant challenges to move increasingly large parts of the mobile network into the cloud. Software Defined Networking (SDN), also called a programmable networks, is a promising and powerful technology.

Today this technology is revolutionizing the cloud networking industry. SDN decouples the control plane from the packet forwarding plane using new software that can be run external to the packet forwarding hardware. This software-driven control platform can automatically create entries in the forwarding table and hence can control network services. SDN reduces network complexity and costs, while enabling virtualization in order to meet cloud computing requirements and in this process accelerating network innovation. All of these advantages position SDN as the technology to enable moving the mobile packet core into the cloud. Another issue that needs to be considered is tunneling in mobile network. Tunneling of IP packets plays an important role in a mobile networks, as mobility is dependent on tunneling between mobile networks entities. It is essential to investigate an appropriate tunneling mechanism in order to introduce cloud computing concepts into mobile networks.

1.2 Related Work

The combination of cloud computing and mobile communication is discussed in several papers. However, most of the work focuses on migrating mobile terminal and radio base station to cloud. Giurgiu, et al.[4] and Chun, et al.[5] provide some practical ideas about implementing cloud computing in mobile terminals. Zhu, et al.[6] proposes a concept "wireless network cloud" which considers all the radio base stations as cloud. The cloud base stations are connected to a "Virtual Base Station Pool" for traffic processing. Spyridon Vassilaras and Gregory S. Yovanof provides four views of wireless communication cloud in [7] and proposes a novel idea called "Wireless Networking Functionality as a Service". The recent master's thesis "Telecommunication Services' Migration to the Cloud"[8] by Isaac Albarrán and Manuel Parras implements a telecommunication service with cloud computing resource and evaluates the service performance based on an Ericsson prototype mobile switching center server. This thesis provides a good evaluation of pros and cons of migrating telecommunication service to cloud.

1.3 Contributions

The project has resulted in two main contributions. First, GPRS Tunneling Protocol-User Plane (GTP-U)[9] has been implemented in the data center's gateway. This implementation is based upon a virtual switch. Implementing GPRS tunneling protocol (GTP)[10] removes the main barrier to moving the mobile packet core network into the cloud as the eNodeB, MME, SGW, and PGW in a LTE network communicate with each other via GTP. The GTP protocol can be divided into a tunneling protocol for control plane traffic: GPRS Tunneling Protocol-User Plane(GTP-C)[11] and a tunneling protocol for user plane traffic: GTP-U. In the implementation of transparent mode, where only user plane traffic is processed within the cloud, GTP-U is required. In the experimental portion of this project, enabling GTP-U in an existing SDN product (called Open vSwitch) was done in order to introduce SDN technology into a mobile network. In the evaluation chapter, the performance of this GTP-U tunnel protocol implementation was investigated in details.

The second contribution is the design of a new SDN oriented mobile network evolution framework. Due to the complexity of mobile networks, it requires a major effort to manage a mobile network cloud if we simply move the mobile network's packet core to the cloud. SDN offers us an appropriate technology for cloud management, hence we introduce a SDN solution for the cloud in order to support mobile networks. A rational and incremental mobile network SDN evolution scheme can blow the "cloud" from the Internet to the mobile network. This evolutionary scheme is based on the existing mobile network architecture, but ultimately leads to a mobile cloud computing ecosystem. The experimental implementation is the first step in this mobile network SDN evolution toward introducing a cloud in transparent mode.

1.4 Structure of this thesis

Chapter 2 introduces the required background knowledge. The relevant knowledge areas include cloud computing, SDN, the evolved packet core, Open VSwitch, and the Linux network subsystem.

In chapter 3, a design for a mobile network SDN evolution framework is presented. The primary idea is to begin by transparently introducing a data center into the mobile network. After this we can incrementally move the mobile network control entities to the data center.

In chapter 4, the implementation of this transparent mode is introduced in detail. This initial implementation was realized in a demonstration. This demonstration includes an implementation of GTP-U in Open vSwitch and utilizes an OpenFlow controller to realize a transparent cloud service in a mobile network. The evaluation results for the performance of GTP-U protocol implementation are given in Chapter 5. Chapter 6 presents the conclusions and suggests some future work.

Chapter 2

Background

This chapter introduces the essential background knowledge for understanding this thesis. This chapter is divided into five sections. The first section is about cloud computing. The basic concept and benefits of cloud computing are introduced. Then, the topics of this thesis, SDN and mobile network architecture-EPC are depicted in the second and third sections, respectively. In the fourth section, the basic concept underlying Open vSwitch and the reason why Open vSwitch was selected as the implementation platform are presented. In the last section, the Netlink technique for communication between userspace and kernel space is introduced.

2.1 Cloud computing

"A lot of people are jumping on the cloud bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of the cloud. "

—Andy Isherwood, HP's vice president for software services in Europe, quoted in ZDnet News, December 11, 2008[12].

Everyone has his or her own definition(s) of cloud computing. As the technology evolves, new definitions emerge for it. However, generally speaking, cloud computing is an Internet-based distributed computing model[13]. In a data center tens of thousands of servers are connected together to form a cloud. Subscribers access this data center via PCs, mobile phones, or tablet computers to utilize computational, storage, and networking resources based on their needs.

Cloud computing utilizes a (generally remote) data center, rather than the local servers. This allows computing resources to be allocated based on the needs of applications, thus companies no longer need to make large capital investments in both hardware and human resources to support these applications. Cloud computing has introduced a revolution in how companies approach computing. Consider the case of a man who used to cook himself his own dinner, but who now can have dinner in a restaurant. This person does not need to buy the raw materials and waste time and energy to cook dinner. Furthermore, the restaurant provides all the food he wants and exactly what he wants, all that he needs to do is pay the bill. With cloud computing, computing becomes a commodity. As someone else is providing the resources for the cloud, companies do not risk over-investing or under-investing in hardware. Nor do these companies need to worry about wasting costly computing resources for a service with an unpredictable future. As the company's demands evolve, they simply rent the resources that they need.

2.1.1 Service Models of Cloud Computing

Normally, cloud computing can be divided into three major service models. They are as follows:

Software as a Service (SaaS) is a new way of accessing software services. It does not require the users to install the software on their own computers or servers, but satisfies their requirements via appropriate cloud service provider[3]. SaaS is a major trend in the development of the software industry. SaaS has attracted the participation of many software giant likes leading to offerings such as Google Apps, Salesforce.com, and Zoho Office[14].

Platform as a Service (PaaS) combines computer platforms, application design, application development, application testing, and application hosting as a service to provide to customers [3]. In this service model, customers do not need to purchase hardware and software. Instead they create, test, and deploy applications and services with the help of PaaS. Two typical examples of the PaaS model are Google App Engine, and Salesforce.com's Force.com platform[14].

Infrastructure as a Service (IaaS) enables subscribers to use cloud computing technology to remotely access computing resources, including computing, networking, and storage[3]. End-users, SaaS providers, and PaaS providers can obtain computing resource via IaaS. Amazon, IBM, and Google are three major players in the IaaS market[14].

A cloud as IaaS can be used to move a mobile packet network's core network into the cloud. This is possible because the mobile network's packet core network is simply a large computing infrastructure. However, migrating this core network into the cloud makes the mobile network's core infrastructure more flexible and reduces the costs of mobile network evolution while accelerating innovation of the mobile network. These benefits will be described in the following subsection.

2.1.2 Benefits of Cloud Computing

According to Zhang, et al. the major benefits of cloud computing are[15]:

Reduced cost: Cost reduction is a obvious benefit of cloud computing. Cloud users reduce their capital investment in infrastructure. They do not need to invest in expensive hardware and software, but can instead share the costs of infrastructure installation and system management with other cloud users. They just need to pay for the resources that they use.

Flexibility: Cloud computing provides greater flexibility for enterprises. Enterprises can determine their needs for additional resources. They can start with something small, with a minimum of investment to meet their requirements. Enterprises can selectively increase computing resources, according to their own business model. Additionally, a cloud computing system is (logically) centralized, in this architecture upgrades and patches can be applied easily[3]. Mobile networks can really benefit from this feature, because there is always a need to evolve the services of the mobile network.

Reliability: Cloud computing's ability for load balancing and failover makes it very reliable[3]. Because of the (logcially) centralized architecture, a cloud service provider can provide most professional solutions to ensure data security. Since, a user needs to access to the data center remotely, the data also needs to be secured while it is being transferred between the cloud and the user.

Sustainability: Some of the existing devices (computers, power supplies, routers, etc.) have poor efficiency, which is environmentally and economically undesirable. Through allocating the resources more reasonably and utilizing more efficient equipment, cloud computing can reduce energy consumption, compared with a traditional server-based architecture[15].

These benefits explain why companies cannot wait to move into the cloud computing world. Shifting the mobile network core network to the cloud paradigm is an obvious means to improve the core network of today's mobile networks in the aspect of cost reduction and network management. It should not be surprising that there have already been efforts to do this, such as reported in the recent master's thesis by Isaac Albarrán and Manuel Parras[8].

2.2 Software Defined Networking

Software Defined Networking (SDN), also called a programmable network, makes the network control plane a independent software platform[16]. SDN is a network solution for the network connection within the data center network. Traditionally, the network consists of switches and routers. This network equipment is developed by manufacturers who developed both the hardware and the software. Because the firmware (and other software) is developed by each vendor for their own hardware, innovation of traditional network equipment has progressed slowly. SDN decouples the control plane and data plane, which can accelerate innovation. With this architecture, software can be developed independently of the hardware. SDN can reduce network complexity, hence it is suitable for sophisticated environments such as a mobile network. SDN is a revolution in comparison with the traditional Internet architecture.

In an ideal SDN environment, the distributed control units can be centralized one. In this approach, the central control node pushes out switching or routing information to the data plane. The switching and routing nodes of the data plane update their forwarding tables based on the received control information.

2.2.1 Why Do We Need SDN?

SDN can efficiently separate the control logic from packet forwarding. This allows users to utilize very complex forwarding rules. According to Goth, SDN provides the following benefits[17]:

- Simplistic, central control can be achieved. It can simplify very complex protocol processing.
- Rapid deployment and maintenance.
- Flexible expansion, SDN can be applied for both small network and large networks.
- Easy upgrade. The user can upgrade their SDN network based on their requirements at any time.

SDN has the potential to accelerate network innovation. The control plane of SDN is centralized and sends the control information to the data plane. This architecture provides a view of the entire network enabling the user to optimize the entire network.

2.2.2 OpenFlow

OpenFlow is the first standard designed for SDN[16]. The OpenFlow concept was first proposed in the March 2008 in the paper "OpenFlow:enabling innovation in campus networks" published by Nick McKeown, a professor at Stanford University[18]. OpenFlow was initially designed for innovative development for a campus network. In a traditional campus network, introducing innovative ideas or new designs for the network is very hard to do, because researchers can not modify the underlying network hardware. For this reason, OpenFlow separates the control plane from forwarding plane, hence it extracts the control logic from the network equipment. This innovation enables researchers to perform experiments via a programmable interface to the underlying network hardware. This makes it possible to verify new network protocols or topologies without modifying the underlying network equipment. OpenFlow architecture is shown in Figure 2.1.

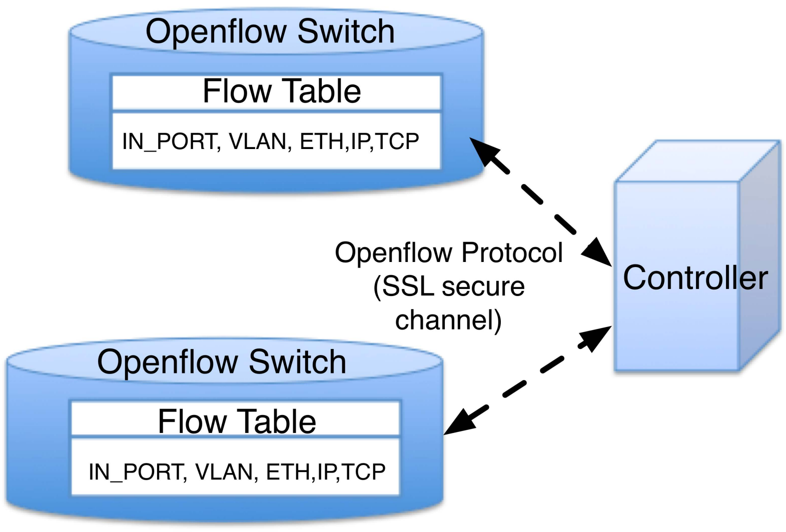
Openflow Switch
Flow Table
IN_PORT, VLAN, ETH,IP,TCP
Openflow Protocol (SSL secure channel)
Controller
Openflow Switch
Flow Table
IN_PORT, VLAN, ETH,IP,TCP

Figure 2.1: OpenFlow architecture, adapted from [18]

Each piece of network equipment maintains a flow table. All the traffic received by the equipments is forwarded based on this flow table. The flow table's configuration is fully controlled by an OpenFlow controller. The flow table is not only concerned with IP, but also utilizes other information (for example for higher layer protocols or for non-IP network protocols). The OpenFlow 1.0 specification defines twelve key fields including ingress port number, VLAN, Layer 2, Layer 3, and Layer 4 information and each field can be wildcard[19]. Network operators can create forwarding rules based on any field. For instance, operators only needing routing based on the destination IP address, can configure the controller to only validating utilize the destination IP field.

In the OpenFlow architecture, Layer 2 features such as MAC address learning, VLAN, and Spanning Tree Protocol (STP) or Layer 3 routing protocols such as Interior Gateway Protocol (IGP) and Exterior Gateway Protocol (EGP) are implemented remotely in the controller, rather than locally in the switch. The controller generates the flow table for each switch based on these protocols and distributes them. Based on Pettit, et al.[19], this control information can be distributed in an active or passive mode. In an active mode, the controller distributes all the flow tables it knows to all equipment. Subsequently the network equipment will forward traffic according to the flow table it has received. In passive mode, the controller does not initially distribute the flow table, but rather when the network equipment receives a packet that does not match any flow in its flow tables, the network equipment forwards the packet to the controller. The controller will forward the packet based on its global knowledge of all flow tables and the controller will then distribute the flow table relevant to this packet to the network equipment that had forwarded to this packet. As a result subsequent packets of the same type are forwarded by network equipment itself. The passive mode's benefit is that network equipment does not need to maintain the entire flow table, but only updates its local flow table when an actual flow arrives. Each flow entry in the flow table is associated with a timer. When this timer expires, the flow entry is flushed from the flow table. This design can greatly reduce the memory required in the network equipment. From a traditional telecommunications point of view, the controller is a logical switch and each item of network equipment is simply a line card.

2.3 Evolved Packet Core

The design of this project is based on an EPC network. Thus, this section introduces the necessary background knowledge about EPC and the tunneling protocol which enables mobility in LTE network, i.e., GTP.

2.3.1 Network architecture

This project is concerned with the core network used in the LTE network architecture. An LTE network is supported by three main EPC entities (see Figure

2.2): the Mobility Management Entity (MME), the Serving Gateway (SGW), and the Packet Data Network Gateway (PGW)[1]. However, the EPC is designed not only to support LTE. In order to handle access to the EPC via any legacy 3GPP radio access network, the EPC also support signalling between the Serving GPRS Support Node (SGSN) and the MME. The Home Subscriber Server (HSS) stores the subscription data and security related parameters relevant also for the SGSN accessed. The SGSN also connects to the SGW. The PGW is the gateway between the EPC and external Packet Data Networks. For security reasons an evolved Packet Data Gateway (ePDG) is sometimes needed for certain types of non-3GPP IP access. The Policy and Charging Rules Function (PCRF) node is required for IP Multimedia Subsystem (IMS) controlled Quality of Service (QoS) and charging mechanisms.

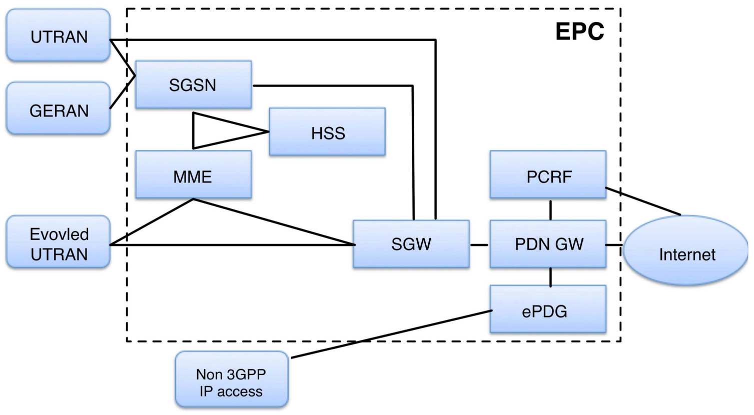

Figure 2.2: Architecture

2.3.1.1 Mobility Management Entity (MME)

The MME is the key control entity. The MME is responsible for User Equipment (UE) registration, security, and bearer management[1]. When a UE initializes a connection, the MME needs to select an appropriate SGW for this UE.

2.3.1.2 Serving Gateway (SGW)

The SGW is responsible for handovers with neighboring eNodeBs. Data transfer of all the user's data packets occurs via the user data plane. The SGW acts as a mobility interface or anchors to other 3GPP systems. The SGW also maintains the UE's context information and generates paging requests for the UE[1].

2.3.1.3 Packet Data Network Gateway (PGW)

The PGW is the gateway between the EPC and external Packet Data Networks (PDNs). It provides connectivity to external packet data networks and serves as the main mobility anchor point. The UE may connect via multiple PGWs, hence UE IP address allocation is performed by each PGW that is being utilized. The PDN GW is also an anchor for mobility between 3GPP and non-3GPP technologies[1].

2.3.2 GPRS Tunneling Protocol (GTP)

GTP is a very central protocol in the EPC. It is used by many different interfaces and for many different purposes. Unfortunately, GTP is also the main difficulty preventing the use of some Internet equipments in the mobile network's core network. Figure 2.3 illustrates the protocol stack relating to GTP in EPC network. GTP is the central protocol used for communication between the major nodes, for instance, SGW, MME, and PGW. In Figure 2.3, the protocol used between eNodeB and MME is called S1 Application Protocol (S1AP). S1AP is used for communication between radio access network and packet core network.

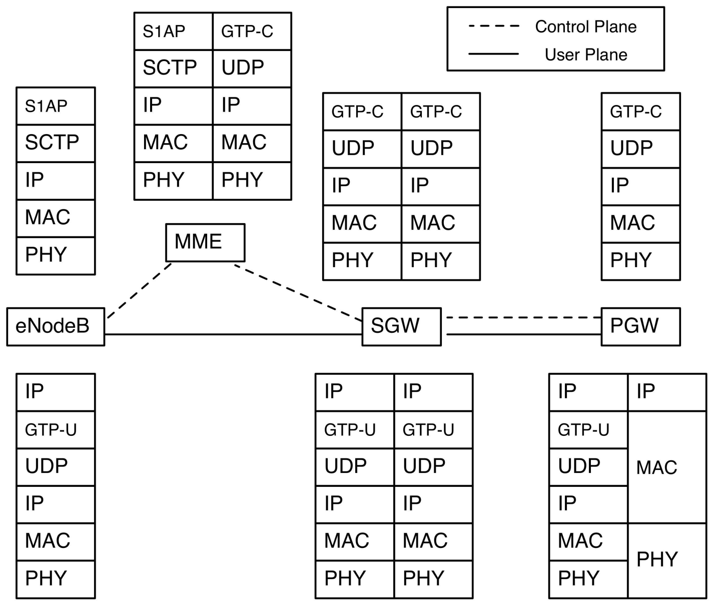

Figure 2.3: GTP overview in EPC[9][11]

GTP has three main functions [10]: tunnel management, mobility management, and user data transfer.

Tunnel management takes care of the establishment, modification, and release of default and dedicated bearers. The signalling parameters exchanged, for a given bearer, mostly relate to QoS and node addressing. A GTP tunnel needs to be established for each bearer.

The mobility management of GTP takes care of a large number of possible mobility scenarios, for UEs in both idle mode and connected mode.
The above functions concern the GTP signaling or control plane, hence they belong to the GTP-C protocol. For more details see GTP-C version 2, introduced in the 3GPP specifications for release R8 [11]. For interworking purposes a given EPC node may have to fallback to the older GTP-C version 1, introduced in release R99 [10]. Because this thesis focuses on GTP-U implementation, the GTP-U protocol stack with header and extension header are shown in Figure 2.4.

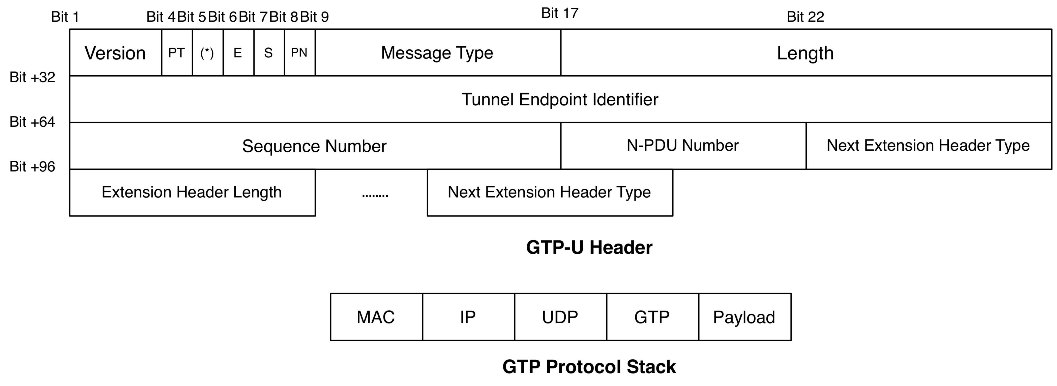

Figure 2.4: GTP-U header and protocol stack[10]

The user data transfer function serves the user plane (GTP-U). The primary function of GTP-U is the encapsulation and tunneling of the user's IP packets. EPC uses the R99 [10] GTP-U version 1 protocol for this purpose.

A GTP tunnel is uniquely identified within a given node by the combination of IP address, UDP port number, and the allocated Tunnel Endpoint ID (TEID). In GTPv2 signaling messages, the node's IP address and the selected TEID are sent as a single parameter called the Fully Qualified TEID (F-TEID).

Since a tunnel is bi-directional it will be associated with two F-TEIDs, one in each tunnel end point. When a tunnel is to be established, the initiating GTP node selects the F-TEID for its end of the tunnel and sends it to the other GTP node. In the same manner, the other GTP node selects an F- TEID for its end of the tunnel and sends it to the initiating node in a response message. It is always the receiving end point of a GTP tunnel that assigns the F-TEID value which the transmitting side has to use.

In the user plane there is a one-to-one relationship between a tunnel and a bearer. Thus, there may be multiple user plane tunnels between two GTP nodes associated with a given UE. In the control plane there is only one tunnel per PDN connection associated with a given UE between two GTP nodes.

2.4 Open vSwitch

Open vSwitch is an advanced edge switch designed under Apache 2.0 license and it has the following features: port bonding, GRE and IPsec tunneling, and per-virtual machine traffic policing, etc[20]. Open vSwitch has been part of the Linux kernel, since Linux kernel version 3.3[21].

2.4.1 How does Open vSwitch work?

The Open vSwitch implementation can be divided into userspace and kernel space. In kernel space, the data forwarding is called a datapath. The most important feature of Open vSwitch is that it separates the control plane from data plane. This is the same approach to separation of these two places as used in OpenFlow. The network device (netdev) layer is used for configuring Open vSwitch. The implementation has a user interface via both a unix socket and via secure sockets layer. The datapath interface (dpif) is a interface for the datapath to communicate with the control layer. The top layer is the OpenFlow controller, it realizes the control layer for a whole Open vSwitch. The relation between the modules used to realize Open vSwitch are shown in Figure 2.5 and will be described in detail in sections 4.1.1.1 and 4.1.1.2.

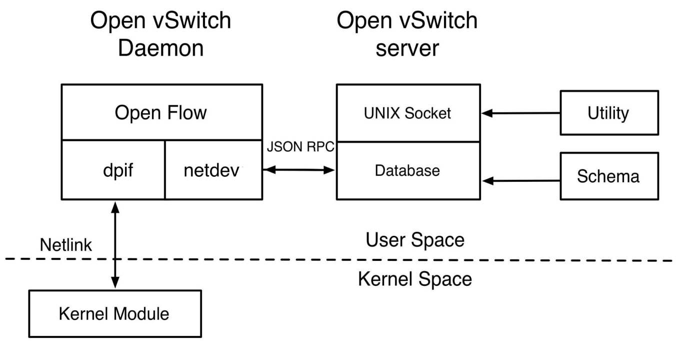

Figure 2.5: Open vSwitch protocol stack

Most of the packets are only processed in kernel space. When Open vSwitch receives a packet from one of its virtual ports, the Open vSwitch kernel module first delivers the packet to the "datapath". The datapath generates a "flow key". According to Horman this flow key could contain[22]:

"Tunnel ID, IPv6 target, IPv4 or IPv6 source address, IPv4 or IPv6 destination address, input port ethernet frame type, VLAN ID, TCP/UDP source port, TCP/UDP destination port, ethernet source address ethernet destination address, IP Protocol or lower 8 bits of ARP opcode, IP ToS, ARP source hardware address, ARP destination hardware address."

The "datapath" performs a lookup in the kernel's flow table. The flow table associates every flow key with actions. These actions may include forwarding the packet to a specific port or ports, encapsulating and forwarding the packet to controller, dropping the packet, etc. The packet is processed according to the actions retrieved from the flow table. If the flow key does not exist in the kernel table, then the Open vSwitch kernel module passes this packet to user space via Netlink (Netlink will be introduced in next section). If the user space has already configured the OpenFlow protocol, then the packet is delivered to the OpenFlow controller. The OpenFlow controller performs a lookup in its flow table and processes the packet in user space. Additionally, the entry for this packet is also delivered to the kernel's flow table. Subsequently, all the packets with this same flow key are processed in the kernel space. If there is no OpenFlow controller, then the packet is delivered to the netdev module in the userspace. This netdev layer is configured by the Open vSwitch database. Netdev follows the same procedure to process the packets as the OpenFlow layer.

For tunnel functions, the control mechanism is slightly different from the above. All tunnel functions are implemented in the kernel space. Open vSwitch has already implemented four kinds of tunnel protocols, Generic Routing Encapsulation (GRE), IPSec, GRE over IPSec, and Control And Provisioning of Wireless Access Points (CAPWAP)[23]. When a tunnel packet arrives the kernel module performs a lookup in the tunnel virtual port table to find the destination virtual port based on IP address, tunnel type, etc. After that it is processed by a virtual port in the kernel space.

2.4.2 Why Open vSwitch?

Linux has a built-in layer 2 switch (the bridge code), which is fast and reliable. However, we selected Open vSwitch as our implementation platform because Open vSwitch is target at enabling the OpenFlow protocol and a cloud computing implementation. Open vSwitch is maintained by a startup called Nicira, established by Nick McKeown.

Because Open vSwitch is OpenFlow based, it inherits several characteristics which are perfect for the project's design. Open vSwitch can respond to network changes dynamically. Open vSwitch supports remote monitoring via Secure Sockets Layer (SSL), thus, we can utilize sFlow to monitor the network's status[24]. As noted earlier, Open vSwitch has already implemented GRE, IPSec, GRE over IPSec, and CAPWAP tunnel protocol, thus, it might be possible to exploit these features to implement GTP for our project.

2.5 User Space to Kernel Space

Open vSwitch utilizes existing Linux network subsystems whenever possible in order to keep the kernel size small. The implementation in this project focused primarily on Open vSwitch kernel module development, thus, it is necessary to introduce the reader how processes running in user space communicate with kernel space in the Linux network subsystem.

According to Benvenut there exist several mechanisms to communicate with the kernel[25]:

The procfs (/proc filesystem) and sysfs (/sys filesystem) mechanisms are virtual filesystems that allows the kernel to export internal information to user space as if this information were stored in a file.

The sysctl (/proc/sys directory) mechanism is used to read and modify the value of kernel variables.

The system call mechanism is the most well-known interface for user space and kernel space communication. This interface can be used to send a command to the kernel or to read configuration information stored in the kernel.

However, this project uses none of the mechanisms above, because currently the preferred interface for kernel network subsystem programming is **Netlink**. The structure of the Netlink header is shown in Figure 2.6. Netlink link messages use the same Netlink header[26]. The content of each field needs to be filled in before using Netlink to communicate. The user space and kernel space need to negotiate the parameters of the Netlink header.

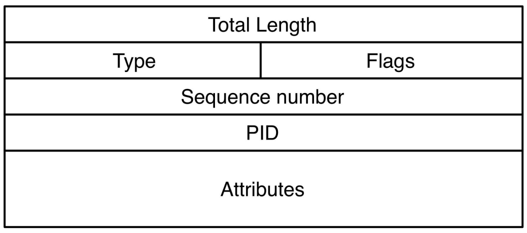

Figure 2.6: Netlink header, adapt from [26]

Netlink offers a datagram oriented system and enables applicaitons in user space to exchange information with kernel with the socket API (for further details see Stevens, et al. [27]). In the kernel of linux 2.4 or later, many applications utilize Netlink to communicate between user space and kernel[25]. For instance, the "iproute2"[28] network management tool performs all of its interactions with the kernel space by utilizing Netlink. The kernel packet filtering tool "Netfilter"[29] also utilizes Netlink for kernel-user space communication. Netlink is also the interface used by Open vSwitch.

A Netlink socket is a special inter-process communication mechanism used for transferring information between kernel and user space processes. It provides standard full-duplex socket communication[30]. Users can programme with a Netlink socket just as with any other socket family. According to Kevin Kaichuan He, there are several reasons why software developers utilize Netlink rather than a system call for Linux network subsystem programming[30]:

- To use Netlink, programmers only need to create a new type of Netlink definition and include it into netlink.h. For instance, we can define NETLINK_DEFINITION 1, then the kernel and user space applications can immediately exchange data between kernel space and user space via this type of Netlink. However, for a system call or ioctl, the programmer needs to instantiate a new device or file, which makes the code more complex. For the /proc file system, we need to add a new file or directory in /proc, this will make /proc more confusing.

- Netlink is an asynchronous communication mechanism. Messages between kernel space and user space are enqueued in a buffer. In contrast, system calls and ioctl are synchronous communication mechanisms. If the exchange of data is too long, the scheduling granularity may be damaged.

- The kernel part of Netlink can be implemented as a kernel module. The user space and kernel space parts of Netlink do not have direct a dependency when compiling. In contrast a system call needs to be linked into the kernel statically and it can not be implemented as a kernel module.

- Netlink supports multicast. As result a kernel module or user space applications can multicast messages to a netlink multicast group. As long as the application belongs to this group, it will receive the messages.

- Netlink running in the kernel can initiate a session, but a system call can only be invoked by users actively.

- Netlink uses the standard socket API, so it is easy to use.

Chapter 3

Design

This chapter presents the design of a mobile network and how I expect this network architecture to evolve in the face of the cloud revolution. The mobile network's architecture and functional entities are described in details in the following sections.

3.1 Design Considerations

To design mobile networks in the face of the cloud revolution, we first need to consider the services that could be deployed in the cloud. Figure 3.1 illustrates the services can we envision could be deployed. This figure shows a logical representation of a mobile network's cloud ecosystem, rather than the physical network architecture.

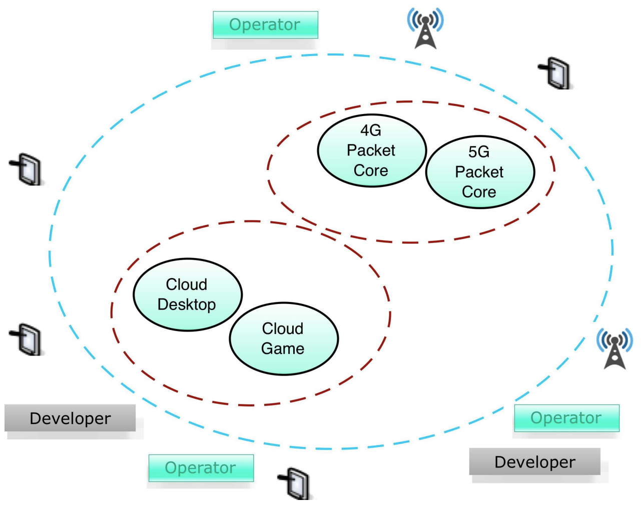

Figure 3.1: The envisioned mobile network's cloud ecosystem

As shown in the figure, in future mobile networks the subscribers will benefit from services in distributed cloud; service providers will benefit from the flexible and low risk of introducing new services; and network operators will benefit from the operator's management and low cost of such a network. To optimize service to the subscribers, the cloud ecosystems could be deployed close to the service area. The cloud ecosystem can be operated as the distributed server in Content Distributed Network (CDN). According to Johnson, et al.[31], a CDN network can reduce the delay dramatically. Admittedly, the distributed cloud ecosystem is different from the servers utilized in a CDN network. However, the success of CDN indicates that deploying cloud ecosystems in the close vicinity of a service area is a promising solution to reduce the delay between a user and a server. This is especially attractive for delay-sensitive services. Since, subscribers are attracted by the cloud ecosystem's low-delay performance, service providers or application developers should be willing to deploy their delay-sensitive services, such as video streaming, live streaming, and internet games in such an ecosystem. As described in section 2.1, cloud computing reduces the risk of deploying new applications which maybe not as popular as the developer predicts. Whenever services are unprofitable, service providers can remove the services from cloud. A mobile network operator does not need to be bothered by the rapid evolution of their mobile network. In the cloud ecosystem, the control plane and data plane are decoupled. For a mobile network's packet core evolution, deploying new services in the cloud means replacing the old network control entities with new ones. Because all the control entities are (logically) centralized in a cloud ecosystem network management is easy to perform. The error-prone network configuration problem is solved by cloud style network management based upon measurements and software that computes a new configuration.

The network entities and services must be deployed according to the existing EPC architecture. The design should facilitate a smooth evolution which transforms the EPC achitecture according to the cloud paradigm. Few operators want to deploy a completely new network infrastructure, especially one with an unpredictable future. In order to avoid a forklift system upgrade a progressive evolution is a better choice.

3.2 Design Approach

The mobile network's cloud evolution is mainly designed based upon the following two fundamental philosophies: (1) the network is a service. (2) the control and data planes should be decoupled.

3.2.1 Network as a service

Due to the recent trend of cloud evolution, it is expected that the all kinds of computing components can be deployed in the cloud[32]. Thus, a mobile network's cloud evolution should consider how the mobile network's packet core could be realized as a service provided by the cloud. This approach would be similar to utilizing the cloud for other services and should result in a postive feedback cycle as shown in Figure 3.2. The players in the figure are described as below,

Network Provider The entity provides mobile cloud computing network. (Network Provider can be also a mobile operator)

Mobile Operator The entity provides mobile communication services with mobile cloud computing network.

Application Developer The entity provides application based on mobile cloud computing.

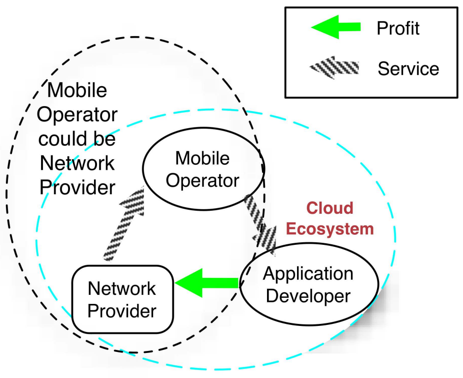

Figure 3.2: Cloud ecosystem positive feedback cycle

The network provider make profits by offering their network infrastructure to deliver application services to clients. The mobile operator provides application users for the application developer. Application developer makes their profit from application users and pays for using the network provider's network infrastructure. If there is an on-going need to improve the network's performance, then the mobile operator can re-invest some of their profits into the network infrastructure to improve the network's performance. The mobile operator can improve the loyalty of the application developer with by providing a better network, then the resulting value-chain forms a positive feedback cycle. The major idea behind it is that if the mobile operator serves all the applications users, it will also attract the application developer by acting as network owner (Note: mobile operator and network provider cloud be the same enterprise). However, this approach could violate telecommunication regulation. Thus, it deserves further study in the future.

Today's operators confront a crisis caused by flat rate mobile broadband subscriptions[33]. This problem occurs because data-consuming services are delivered via the mobile network, but most of the income belongs to service providers. However, cloud computing could change this trend. If the mobile operators also act as a cloud platform provider, rather than only acting as a network provider, then they could charge the service providers for using their cloud computing infrastructure. The reason behind is that the service provider could get lower delay from this mobile cloud as introduced in section 3.1. If they want to deploy some delay sensitive services for mobile users, the mobile cloud infrastructure would be the best choice. This is just a proposal for resolve the revenue gap, there could be more conditions need to be considered.

3.2.2 Control Plane and Data Plane Decoupling

To simplify cloud management, the network control architecture should also evolve. Since SDN is a promising direction for cloud management, the decoupling of the control plane and data plan needs to be considered. Unlike computer networks, mobile networks' control entities are more complicated, because of their support for mobility. Fortunately, an isolated and centralized control plane is easier to manage than distributed control entities. The reduced risk of configuration error can be a driver for this decoupling. The decoupling of the control plane and data plane eases network management. Since the control entities are isolated, it is easily to replace them with new network control technology.

3.3 Network Architecture

Due to needs of delay sensitive content services, the cloud ecosystem should be deployed close to the eNodeBs in order to deploy a CDN like network which can reduce the delay as described in section 3.1. By deploying the OpenFlow protocol and with a careful design, the original EPC network protocol stack can be modified at a very low level. This design facilitates the mobile network's cloud evolution and demonstrates the platform characteristics of a cloud ecosystem. Generally, the mobile network's cloud evolution design can be divided to two phases. The first phase is a transparent phase based upon deploying a data center between the eNodeB and SGW. In this phase, no EPC control entities are involved, rather the data center contains only content servers. In the decoupling phase, the EPC control plane and data plane are decoupled, hence all the EPC control entities are moved to the cloud. The result is that the mobile network is a service which is provided by a mobile cloud ecosystem.

3.3.1 Transparent Phase

The functional reference model of the proposed transparent phase is shown is Figure 3.3. The Mobile Network Gateway duplicates all GTP-U packets and delivers them to the cloud ecosystem for processing and the original packets are forwarding as normal. In this way, if we switch the service to the cloud in the middle of the service, the switch process is transparent. If the data center decides to provide the service to the subscribers in a specific tunnel, then the two gateways create a new tunnel and substitute themselves for the original tunnel in the mobile network. However, in transparent phase, the traffic may need to be blocked from the from the external network via the PDG to SGW (For more details about this behavior, please refer to section 4.2). It violates the telecommunication regulation in Europe and north America. Thus, this design also needs the collaboration with telecommunication regulator. It requires the modification of current regulation.

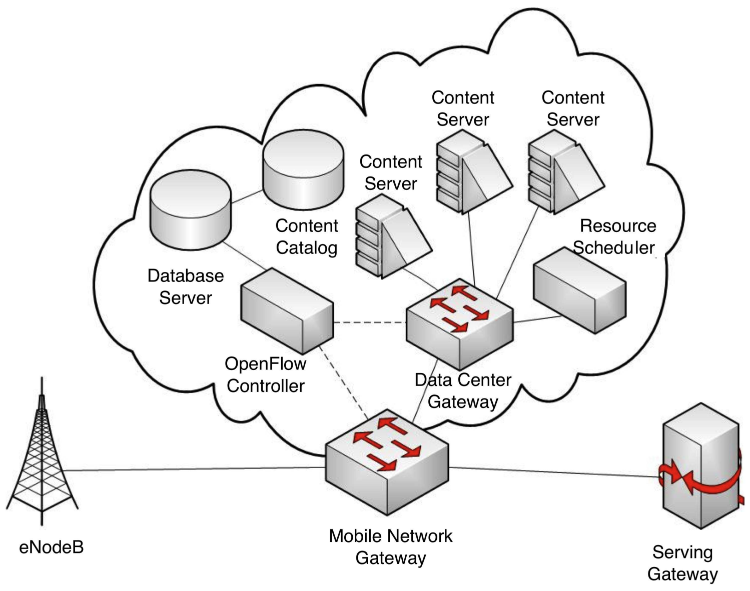

Figure 3.3: Transparent phase network architecture

The primary consideration is that the cloud ecosystem should be deployed transparently in this phase. However, the GTP network is simply providing a set of tunnels, so all of these tunnels could pass via the cloud and the different servers could be migrated into the cloud. While this increases the complexity of the mobile network's packet core this approach avoids the need for the data center to be directly deployed within the EPC. The cloud ecosystem needs some control entities, but in a transparent mode implementation we do not involve any control entities (such as the PGW-C). Since the PGW-C allocates the UE's IP address, as described in section 2.3, the data center must reuse the same IP allocation. The control mechanism is some what more complex in the proposed cloud ecosystem. As the OpenFlow protocol is proposed for cloud management, we can implement these complex control mechanisms in the OpenFlow controller. As mentioned in section 2.2.2, the OpenFlow protocol is very flexible and is not based on any routing protocol, rather the entire network is simply a switched network. An OpenFlow node can be programmed to match any specific type of packet in order to process them as desired. The OpenFlow controller is the main controller in the mobile cloud ecosystem.

In this design, two gateways are set up, one for the mobile network and another one for data center. This configuration separates the data center from switch that is connected to the eNodeB or SGW. The mobile gateway focuses on forwarding packets, hence it works as a normal switch. The data center gateway should be a OpenFlow switch which can also process GTP-U packets. In the data plane, the GTP-U encapsulation is the only difference between a mobile network. A computer network with respect to the packet data format. A database server is used to store existing tunnel attributes, a content catalog is used for storage and retrieval of the content information for data stored in the data center. The resource scheduler is an entity which enhances the utility of the cloud's resources. In section 3.4, the functions of each of these entities are described in detail.

3.3.2 Decoupling Phase

Figure 3.4 shows the functional model of the decoupling phase. Obviously, the basic idea for this phase is that the entire EPC network and configuration of all the control functions of EPC are inputs to the OpenFlow controller. The eNodeB is simply a means to access the mobile network's cloud service. In the mobile network's cloud evolution, the main modification is the addition and evolution of the OpenFlow controller. This controller becomes more and more powerful as the cloud ecosystem evolves. Because the data center and Mobile Network Gateway simply deal with the data plane, they do not require any modifications with respect the configuration of the rest of the system. Note that the data center gateway and mobile network gateway must be controlled by the OpenFlow controller and support a tunnel between them.

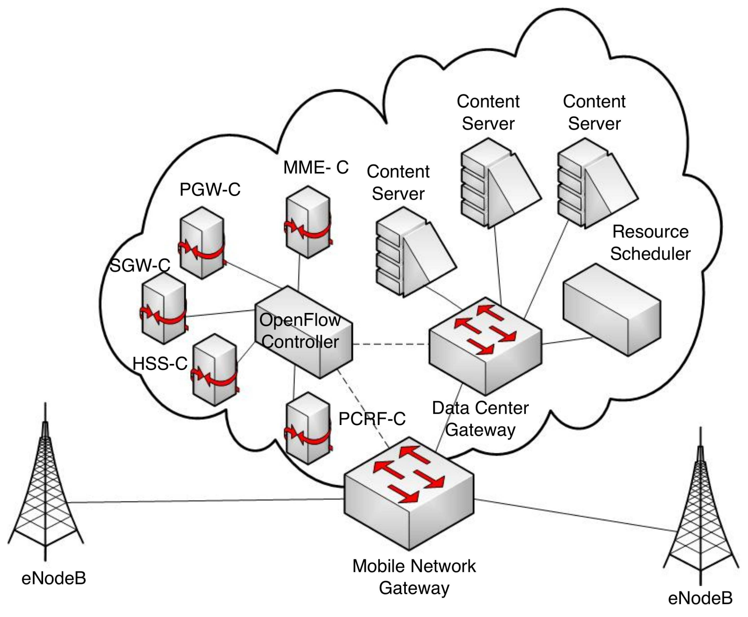

Figure 3.4: Decouple phase network architecture

3.4 Functional Entities

The mobile network cloud evolution initially reuses the mobile network's functional entities, but separates the control plane and data plane. Additionally, there are some new entities that are used to extend the mobile network for the cloud ecosystem. These nodes are logical entities, they can be grouped into a physical node. These entities will be introduced in this section.

3.4.1 Mobile Network Gateway

The Mobile Network Gateway is responsible for forwarding packets among data center, eNodeB, OpenFlow controller, and the network outside the cloud. Here the Mobile Network Gateway is an OpenFlow based switch, thus it has only a data plane; therefore, it does not actually switch any packets to/from the OpenFlow controller! It is controlled by an OpenFlow controller via the OpenFlow protocol.

3.4.2 Data Center Gateway

Data Center Gateway is a gateway with a GTP-U extension. It can decapsulate and encapsulate packets according to the GTP-U protocol stack. The Data Center Gateway is also an OpenFlow based switch.

3.4.3 OpenFlow Controller

The OpenFlow Controller is the brain of the mobile network's cloud ecosystem. It realizes the middle layer between the control plane and data plane. It processes control plane packets with the support of other mobile network control entities, then distributes the resulting flow table information to the Data Center Gateway and Mobile Network Gateway.

3.4.4 Resource Scheduler

The resource scheduler OpenFlow controller is the scheduler for cloud resources. There are many research results related to on this topic, see for example: [35] and [36]. The function of the resource scheduler is to enhance the utility of cloud computing resources.

3.4.5 Network Database

The network database only exists in the transparent phase and it is used to record the tunnel information. Because the mobile network's cloud ecosystem cannot control tunnel establishment and release, we need to store the exsiting tunnel information in a database in order to provide this data at the appropriate time(s).

3.4.6 Data Center Content Catalog

The Data Center Content Catalog only appears in the transparent phase. Because of the transparency requirement, we do not know that whether we have the content or not. This catalog is a very complex functional entity. It stores imformation about the content stored by the data center, this information is necessary so that the OpenFlow can decide whether to switch the tunnel to the data center.

Chapter 4

Implementation

In the previous chapter, an overview of the design of an SDN framework for the evolution for a mobile network into the cloud was presented. In this chapter, the details of this design and the experimental implementation of such a SDN is given. In section 4.1, the full details of the different parts of the implementation concerning the implement of GTP are given. Following this in section 4.2, the full details of the transparent mode implementation are given.

4.1 GPRS Tunneling Protocol

As described earlier the GPRS tunneling protocol (GTP) is a very central protocol for carrying subscriber's IP packets through the mobile network's core network. For more details of this protocol, please refer to section 2.3.2. It is essential to implement GTP in the SDN switch that we will use, specifically in the Open vSwitch which has been adopted in this project. We will described the details of this implementation before introducing any of the SDN concepts into the proposed mobile network.

4.1.1 The User Space Control Daemon

In this section, an overview of of an implementation of an Open vSwitch control daemon is described. This userspace daemon is implemented in C. An overview of the different modules is shown in Figure 4.1. In this figure netdev represents a network device and dpif represents a datapath interface.

The modules depicted in the figure are described in detail in the following subsection. For now we note that the user space control daemon is the bridge between the user interface and the kernel module. Once Open vSwitch is installed, it extracts the network device framework from a schema file. The framework only creates a database for the network device as the Open vSwitch will not communicate with any other functional entities. The virtual switch and virtual port should be configured via the user interface. The GTP protocol implementation must ensure that it can be configured via this user space control daemon.

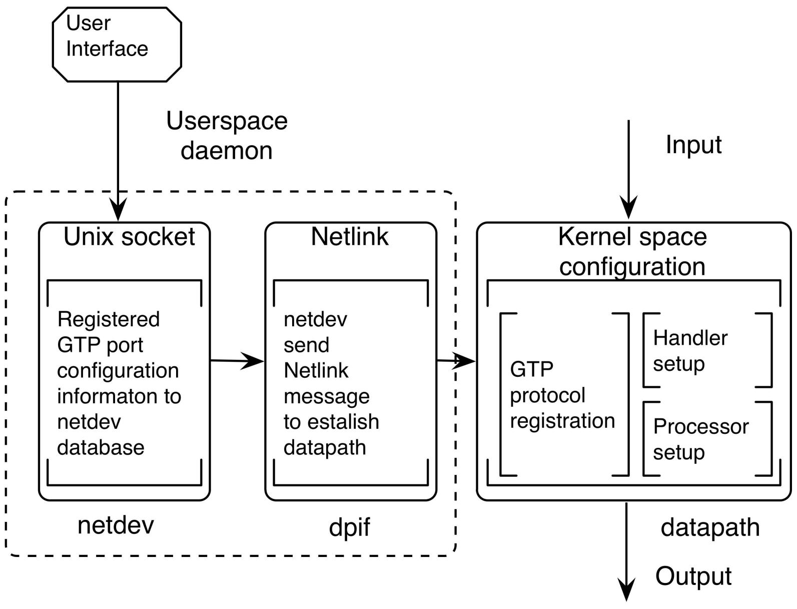
User Interface
Userspace daemon
Input
Unix socket
Registered GTP port configuraton informaton to netdev database
Netlink
netdev send Netlink message to estalish datapath
Kernel space configuration
GTP protocol registration
Handler setup
Processor setup
netdev
dpif
datapath
Output

Figure 4.1: User space control daemon

4.1.1.1 Network Device (netdev)

As GTP is a new protocol for Open vSwitch, we must register this protocol in the netdev database. In order to do this, the existing `port` data structure is reused. We create a new port type called "`gtp`". This port contains all the GTP tunnel's attributes, i.e., port name, TEID_IN, TEID_OUT, remote IP address, local IP address, and ARP reply sproofing. The port name, remote IP address, and local IP address are three attributes of any tunnel port type. The remaining attributes are newly added for use with a GTP tunnel. As, all the outcomes of this thesis work are Ericsson intellectual property, the source code has not been merged into the open source Open vSwitch code. These new attributes have to be added to the source code of Open vSwitch. An example for configuring a GTP port is:

```
ovs-vsctl add-port br1 gtp0 -- set interface gtp0 type=gtp
options:remote_ip=192.168.1.1, local_ip=192.168.1.2,
teid_in=1, teid_out=2, arp_reply=true.
```

Listing 1: GTP-U port configuration

The netdev refreshes itself after a very short interval. Whenever netdev detects a modification in the database, it will trigger the datapath interface to generate a Netlink message in order to configure the datapath protocol handler and setup the processor to correctly forward packets.

4.1.1.2 Datapath Interface (dpif)

The datapath interface is the middleware between the user space control daemon and the datapath. It utilizes Netlink to deliver a message from user space to kernel space and vice versa. The dpif module (which implements DPIF) provides APIs for configuring the datapath as shown in Figure 4.2. The configration steps are as described below.

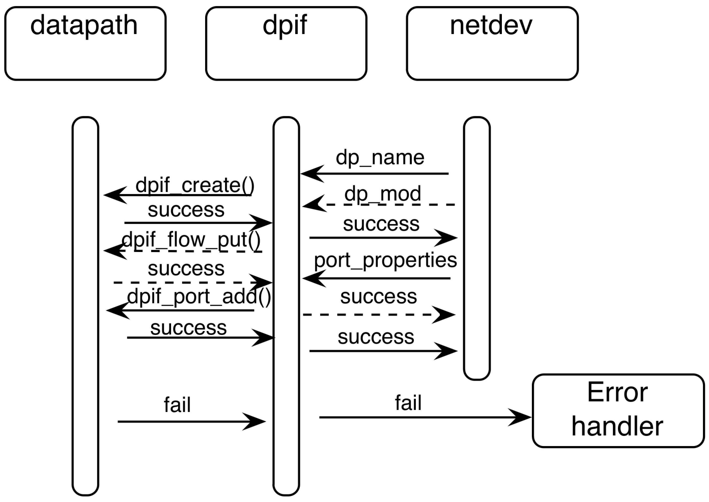

Figure 4.2: The netdev utilizes dpif to configure a GTP datapath in kernel space

First, netdev sends the datapath name `gtp` and datapath type `gtp` to `dpif`. The dpif module creates a empty datapath and returns the execution to user space. The API to create a datapath using the dpif module is:

```
int dpif_create(const char *name, const char *type,
                struct dpif **dpifp)
```

Listing 2: dpif_create

Second, netdev sends a netdev configuration which contains GTP port properties to `dpif`. The datapath assigns a port number to the port and adds the port information to the datapath. To do this the following function is called:

```
int dpif_port_add(struct dpif *dpif, struct netdev *netdev,
                  uint16_t *port_nop)
```

Listing 3: dpif_port_add

Finally, if the port information is modified, the function dpif_flow_put is called to modify the datapathÕs configuration. The prototype for this functions is:

```
int dpif_flow_put(struct dpif *dpif,
                  enum dpif_flow_put_flags flags,
                  const struct nlattr *key, size_t key_len,
                  const struct nlattr *actions,
                  size_t actions_len,
                  struct dpif_flow_stats *stats)
```

Listing 4: dpif_flow_put

A failure may occur because a datapath already exists or because of resource exhaustion.

4.1.2 The Kernel Module

In this section, we describe the implementation of the components making up the Linux kernel module as depicted in Figure 4.3. The GTP module is one part of the Open vSwitch kernel module. The kernel module is the only component responsible for processing tunnelled packets. It contains three main functions: protocol registration, GTP protocol encapsulation and decapsulation, and ARP reply sproofing. For GTP protocol encapsulation and decapsulation, I will mainly focus on the the case of an ethernet header, as this is most likely to be the physical interface to which tunnelled packets are sent and received.

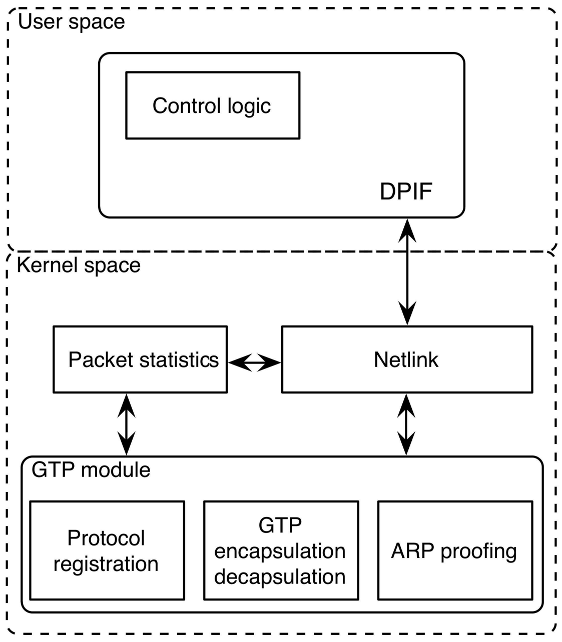

Figure 4.3: The implementation of the GTP kernel module is made up of three parts.

4.1.2.1 Netlink Communication

As described previously in section 2.5.2, Netlink sockets are used to transmit messages between user space and the Linux kernel module. In the following paragraphs, the technical details of the netlink sockets' implementation in the kernel module is discussed. For the GTP module, the netlink messages only deliver error messages and packet statistics to user space. The message group for sending a flow key to user space is not necessary for tunnel packets. In order to keeping the code neat, there are two portions of the code that we need to focus on. One portion of the code concerns only unicast communication, another portion of the code asynchronously receives messages from user space. We have left out multicast and synchronous communication netlink communication because they can introducing subtle thread related bugs. Theoretically, Netlink messages should be sent and received asynchronously. Additionally, the OpenvSwitch kernel module process messages in an asynchronous way. The code of GTP-U module also follows the programming recommendations for Netlink in the "generic_netlink_howto " at[38].

4.1.2.2 Protocol Registration

The approach to register the GTP protocol is different from the normal Internet domain protocol registration. This difference occurs because the Linux kernel does not assign a protocol number to the GTP protocol and because GTP encapsulation is UDP based. These two differences result in the GTP protocol registration procedure not being very straightforward. There does not exist an API to perform this registration, but rather the registration is based on the kernel UDP socket binding procedure. The UDP socket struct in the Linux kernel is:

```
struct udp_sock {
        /* inet_sock has to be the first member */
        struct inet_sock inet;
#define udp_port_hash inet.sk.__sk_common.skc_u16hashes[0]
#define udp_portaddr_hash inet.sk.__sk_common.skc_u16hashes[1]
#define udp_portaddr_node inet.sk.__sk_common.skc_portaddr_node
        int pending; /* Any pending frames ? */
        unsigned int corkflag; /* Cork is required */
          __u16 encap_type; /* Is this an Encapsulation socket? */
        /*
           * Following member retains the information to
           * create a UDP header when the socket is uncorked.
           */
        __u16 len; /* total length of pending frames */
        /*
           * Fields specific to UDP-Lite.
           */
        __u16 pcslen;
        __u16 pcrlen;
/* indicator bits used by pcflag: */
#define UDPLITE_BIT 0x1 /* set by udplite proto init function */
#define UDPLITE_SEND_CC 0x2 /* set via udplite setsockopt*/
#define UDPLITE_RECV_CC 0x4 /* set via udplite setsocktopt*/
        __u8 pcflag; /* marks socket as UDP-Lite if > 0*/
        __u8 unused[3];
        /*
           * For encapsulation sockets.
           */
        int (*encap_rcv)(struct sock *sk, struct sk_buff *skb);
};
```

Listing 5: UDP socket struct

We can find two members relating to encapusulation: `encap_type` is the registration number for the tunneling protocol (this number can be a random number, but it should not conflict with any other UDP based tunneling protocol). If Open vSwitch attempts to run two UDP based tunneling protocols with conflicting type numbers, it will cause a kernel panic.

The registration procedure is shown in Figure 4.4. The first part of the GTP registration is the same as the UDP socket binding procedure. Since UDP port 2152 has been assigned by the Internet Assigned Numbers Authority for GTP-U protocol, thus, we register the `GTP_DST_PORT` as 2152. The error handing processes for `sock_create` and `kernel_bind` are different. If the socket create process fails, the kernel only needs to print a warning to the `syslog`. However, if the kernel bind fails, then the code has to release the socket first (as shown in listing 6).

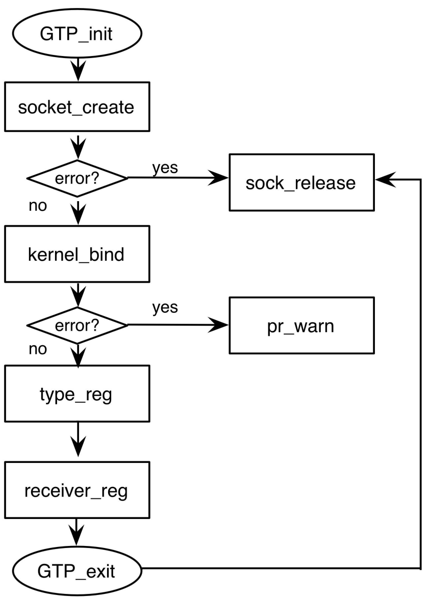

Figure 4.4: Open a UDP socket and register the GTP protocol

```
int err;
    struct sockaddr_in sin;
    err = sock_create(AF_INET, SOCK_DGRAM, 0, &gtp_rcv_socket);
    if (err)
            goto error;
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = htonl(INADDR_ANY);
    sin.sin_port = htons(GTP_DST_PORT);
    err = kernel_bind(gtp_rcv_socket, (struct sockaddr *)&sin,
       sizeof(struct sockaddr_in));
    if (err)
            goto error_sock;
```

Listing 6: UDP socket kernel bind

As the protocol number field in the IP header is limited to a maximum of 255, while the `encap_type` field is a 16 bit unsigned number, we define `UDP_ENCAP_GTP` as 65535 for `encap_type`. Setting these values in the socket is shown in the code listing below. For the details of `gtp_rcv`, please continuing to read next section. At this point, the entire GTP registration process is finished.

```
udp_sk(gtp_rcv_socket->sk)->encap_type = UDP_ENCAP_GTP;
udp_sk(gtp_rcv_socket->sk)->encap_rcv = gtp_rcv;
```

Listing 7: Setting encapsulation type and receive fuction

4.1.2.3 Ethernet Header Processings

Before discussing the details of Ethernet header processing, it is necessary to clarify the reason why this project needs this processing. Since we choose an OpenFlow architecture for our implementation, an OpenFlow based switch is the platform used for processing all the traffic. Since Open vSwitch is basically a switch, the packets being switched need to have layer 2 headers, even thought the system will ignore the layer 2 headers in its forwarding process.

As described in section 2.3.2, there is no ethernet header after the UDP header in the GTP protocol stack. This means that if we simply deal with the GTP protocol as another type of tunneling processing, after decapsulation,the packet will not have any layer 2 information, hence in order to implement the GTP protocol in Open vSwitch, we need to include layer 2 information. The simplest means of doing this is to create a dummy ethernet header, but this is not sufficient. The packet has to be forwarded based on layer 3 information, since the actual layer 2 information does not exist. One option would be to simply route the packet. However, since Open vSwitch is OpenFlow based, we utilize the OpenFlow protocol for layer 3 forwarding. The reason behind this that OpenFlow is open for future development and in this way we can also implement all the functionality in Open vSwitch.

With OpenFlow, we can also have two different configurations for processing packets. The first configuration only specifies the input port and output port in Open vSwitch with the OpenFlow Protocol, but enables the promiscuous mode of the virtual machine's Network Interface Card (NIC). Another configuration is to enable layer 3 mapping and Ethernet header modification as we will do in the following section. Based on experiment 2 in evaluation chapter, the first configuration's performance is slightly better than the second configuration. However, if we enable promiscuous mode, this will add an extra burden for processing all the packets in the network. It is not worth enabling the promiscuous mode simply to improve the performance slightly (as this increases the total load on the CPU which will have to process all packets received by the interface). For this reason, we choose the second OpenFlow configuration as our solution.

Figure 4.5 illustrates how Open vSwitch processes arriving GTP packets. As described in the previous section, gtp_rcv() listens on UDP port 2152. First, gtp_rcv() decapsulates the packet that was encapsulated by GTP. The resulting packet is simply a IP packet. It does not have an ethernet header. To insert such a header gtp_rcv() creates a dummy ethernet header with special source and destination MAC addresses. The destination MAC address is 00:00:00:00:00:00 and the source MAC address is 11:11:11:11:11:11. The purpose of these special MAC addresses is to enable the OpenFlow controller to identify the GTP packet easily and to avoid disturbing other forwarding behaviors. After this, the task of delivering the packet to the correct virtual machine is handled by the OpenFlow controller.

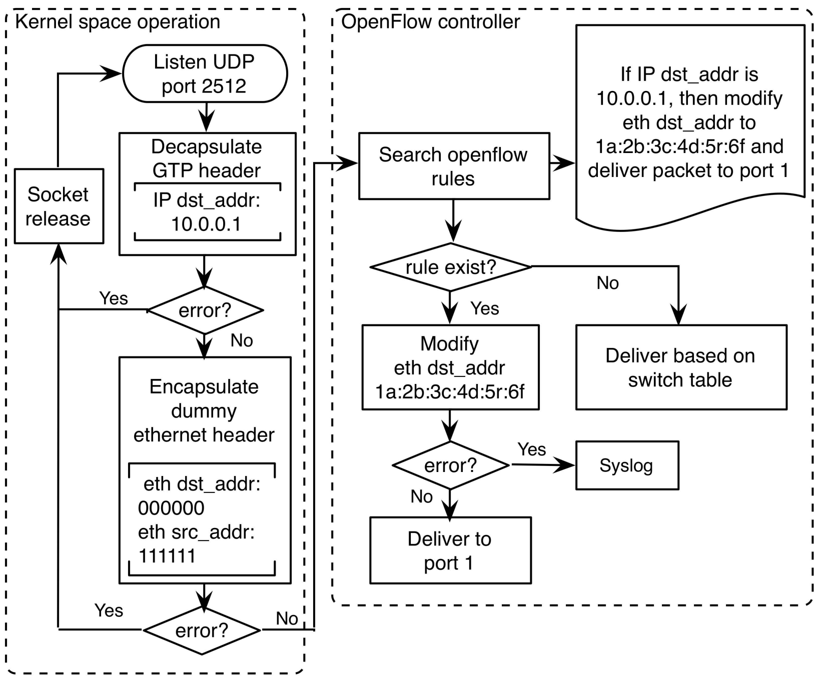

Figure 4.5: Ethernet header processing when receiving a packet

The sending process is shown in Figure 4.6. When sending, the process is reversed from receiving part. However, we need a new ARP reply spoofing mechanism to deal with the ARP request packets from the the computer network. This ARP reply spoofing mechanism will be described in next section. When a virtual machine receives the spoofed ARP reply, it will believe the destination MAC address is 11:11:11:11:11:11 for any IP destination address. When the Openflow controller finds packets with this special destination MAC address, it delivers them to a suitable port based on the OpenFlow rules. For kernel space operation, sending is simpler than reception. Open vSwitch simply needs to remove the ethernet header and encapsulate the packet by adding a GTP header. The resulting packet will be forwarded through the GTP tunnel.

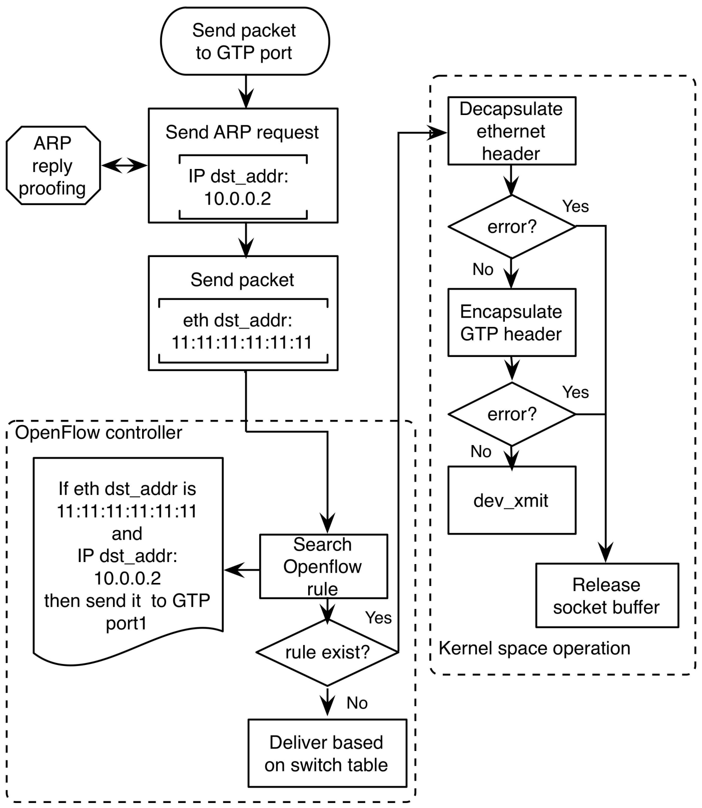

Figure 4.6: Ethernet header processing when sending a packet in a GTP tunnel

4.1.2.4 ARP reply spoofing

In a local area network, there is an address resolution protocol (ARP) that is used to learn what MAC address is associated with a destination IP (or other layer 3 protocol) address. This MAC address is used by a layer 2 switch to properly forward the packet. When a host wants to send the packet to a destination IP address, it utilizes ARP to learn the MAC address to which it should forward this packet. However, there does not exist ethernet information for the GTP tunnel. In order to use Open vSwitch we have to create a ARP spoofing mechanism to cause the host to forward the packet to Open vSwitch. Then, OpenFlow controller tells the Open vSwitch to ignore the layer 2 address and to forward the packet based on its IP address. This process is shown in Figure 4.7.

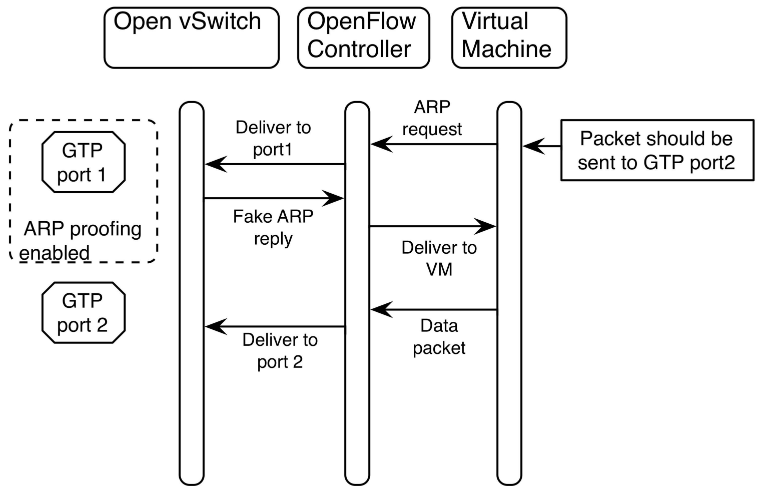

Figure 4.7: ARP reply spoofing mechanism

As mentioned in section 4.1.1.1, there is a field called `arp_reply=true` in the the GTP port configuration. This attribute enables the ARP reply spoofing for the GTP port. When the GTP port receives an ARP request packet, it will reply with a fake ARP reply to inform the virtual machine that the destination MAC address is 11:11:11:11:11:11 (as described in the previous section). Since, an ARP cache is not implemented for the GTP port, it is better to only enable ARP reply spoofing for one GTP port. Otherwise, more than one ARP reply packets will be sent to the virtual machine. Although, it does not disturb the normal processing, processing these additional reply packets would waste resources. After the virtual machine receives the ARP reply, it sends the data packet with a dummy MAC destination address. Now, the OpenFlow controller delivers the packet to the correct port based on OpenFlow rules.

4.2 Transparent Mode

In the following, I present the implementation details of transparent mode as described in section 3.1. In the transparent case, the data center's deployment does not disturb the normal packet forwarding in a mobile network. However, this processing can not be accomplished by traditional network entities such as switches and routers. In the implementation, I utilize SDN technique-OpenFlow to achieve the complex packet forwarding required in mobile networks.

4.2.1 Transparent Mode Overview

The transparent mode is implemented by adding a switch as the data center gateway. This switch is located at the data center. The switch consists of an Open vSwitch, a MySQL database server, and a OpenFlow controller (in the case of this project we have used Floodlight version 0.85[39]). The data center uses the Open vSwitch as a gateway and the data storage function is implemented in a virtual machine (in the case of this project we have used VirtualBox 4.2.4 for Linux Hosts[40]). An overview of the configuration of the demonstration network is shown in Figure 4.8.

As stated above the Open vSwitch is a gateway for the data center. The gateway for the mobile network is an Open vSwitch with our GTP extension so that it can handle GTP-U packets. However, the Open vSwitch that acts as the gateway for the data center can be a standard version of the Open vSwitch as it does not need to have any GTP functions. The virtual machine was implemented with VirtualBox 4.2.4.

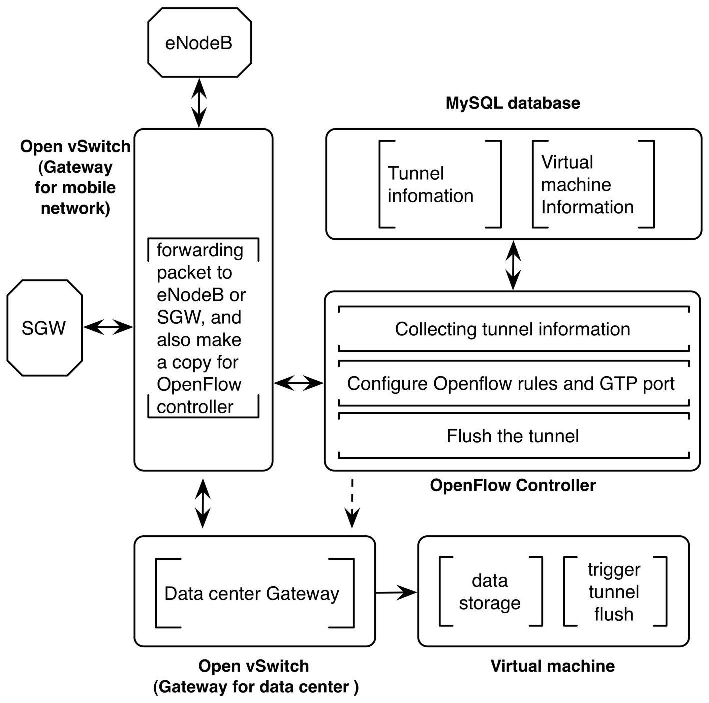
eNodeB
MySQL database
Open vSwitch (Gateway for mobile network)
Tunnel infomation
Virtual machine Information
forwarding packet to eNodeB or SGW, and also make a copy for OpenFlow controller
SGW
Collecting tunnel information
Configure Openflow rules and GTP port
Flush the tunnel
OpenFlow Controller
Data center Gateway
data storage
trigger tunnel flush
Open vSwitch (Gateway for data center)
Virtual machine

Figure 4.8: The overview of transparent mode

Open vSwitch acts as a gateway to the mobile network. The pair of gateways (the mobile network gateway and the data center gateway) are deployed between all of the elements in the mobile network, such as the eNodeB and SGW. This

Open vSwitch provides the data plane for the entire logical mobile network. For our demonstration we have configured the Open vSwitch with four network interfaces. Two interfaces to connect to the single eNodeB and SGW that are used in the demonstration network. One interface is the control channel for the OpenFlow controller and another interface connects the switch to the rest of the data center. The OpenFlow controller implements the control plane for the switch. This controller implements three functions, collecting tunnel information, Open vSwitch configuration, and tunnel flush. The OpenFlow controller makes use of a database to record the tunnel and virtual machine information. This database component is implemented by a MySQL database server.

4.2.2 GTP-U Packet Processing

The data packets between eNodeB and SGW are transported by the GTP-U protocol. The GTP-U packet processing consists of three parts. The first step is collecting the tunnel information from the packets. The second step is for the switch function which delivers the packet to the cloud and if necessary to create a new tunnel. The third step is to remove the tunnel, if the tunnel has been not in used for a while. In the paragraphs below we will examine each of these components of the GTP-U packet processing.

4.2.2.1 Collecting Tunnel Information

Collecting tunnel information should not interrupt or slow down the normal forwarding processes of packets between the eNodeB and SGW. Naturally, the solution for this is to simply duplicating the GTP-U packet and send a copy of it to the Openflow controller for processing. As mentioned in section 4.2.1, a MySQL database server stores the tunnel information. As illustrated in Figure 4.8 the MySQL database maintains two tables. One table contains the tunnel information, while the other tables contains the virtual machine information (virtual machine's IP address, MAC address, and the port number used to connect to the Open vSwitch). Table 4.1 shows the tunnel attributes and table 4.2 shows the virtual machine attributes.

Table 4.1: Tunnel Information Table

Attribute	**Type**	**Description**
e_ip_src_addr	varchar	The tunnel start point IP address
e_ip_dst_addr	varchar	The tunnel end point IP address
i_ip_src_addr	varchar	The internal source IP address
i_ip_dst_addr	varchar	The internal destination IP address
teid_enode	varchar	The tunnel endpoint ID for eNodeB
teid_sgw	varchar	The tunnel endpoint ID for SGW

Table 4.2: Virtual machine information table

Attribute	**Type**	**Description**
ip_addr	varchar	The virtual machine IP address
mac_addr	varchar	The virtual machine MAC address
port	varchar	The Open vSwitch port that virtual machine connect to

The tunnel information table stores six attributes. These attributes identify a given tunnel. For details, see section 2.3.2. The first five attributes were extracted from the packets sent from the eNodeB to the SGW. The sixth attribute is extracted from the packet sent from the SGW to the eNodeB.

The table of virtual machine information (whose attributes were shown in Table 4.2) contains the server content catalog that was described in section 3.1. We assume that if the virtual machine's IP address is the same as the packet's inner IP destination address, then the switch unit should create a new tunnel through which to forward the packet to this virtual machine in the cloud for processing.

Figure 4.9 illustrates the detailed signaling process for collecting tunnel information. Before we explain the details of the signaling process between the five nodes in the figure, we note that there are two tunnels in transparent mode. These two tunnels have the same tunnel attributes, however, the tunnel between eNode and SGW is the original actual tunnel, while the tunnel between the eNodeB and data center will be substituted for the original tunnel. Because of the way that we instantiate the new tunnel, the tunnel's substitution is transparent to the eNodeB. From the eNodeB's point of view, the two tunnels are the same (i.e., there is no change in IP addresses or tunnel endpoint IDs). However, it is only applicable for UDP traffic. For TCP traffic, the content server and original server may generate different sequence numbers in TCP handshake stage. Thus, when the tunnel is switched, the TCP connection would be restablished. Then, it may affect end-user's service. Therefore, we need some processing mechanism to deal with this issue. However, due to the time limitation, this issue will be considered as further work.

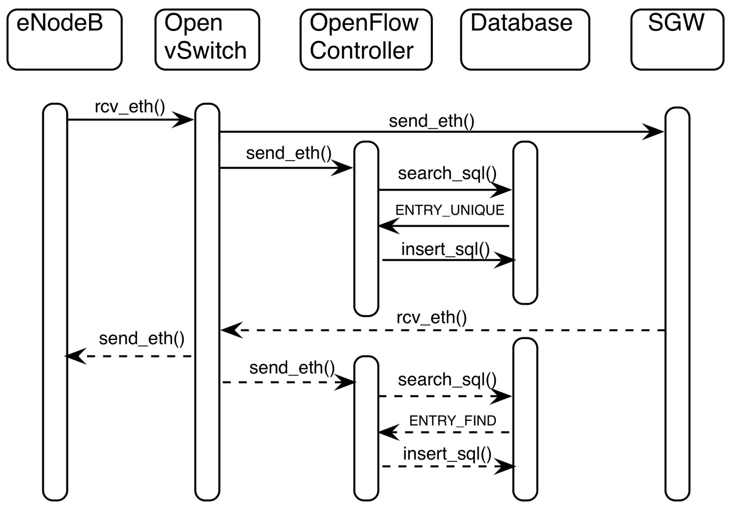

Figure 4.9: Openflow controller collects tunnel information from the packets - the upper half of the figure shows the packets being sent by the eNodeB, while the lower half of the figure shows the packets being sent by the SGW

The signaling procedure is not very complex. Initially, the eNodeB sends a packet to the SGW. This packet is switched by the Open vSwitch that is acting as the gateway for the mobile network. This Open vSwitch duplicates the packet and sends a copy of this packet to the OpenFlow Controller. The controller will extract the first five tunnel attributes shown in table 4.1. Given this information the Openflow controller will search in the database to check if these five tunnel attributes uniquely identify a tunnel. If so, then it will insert an entry for this tunnel into the database. If it is not unique, then it will process the next packet without modifying the database. When a packet is sent from the SGW to the eNodeB, the same basic procedure will be followed. The only difference is when searching in the database, the OpenFlow controller only uses the first four attributes. If an entry is found, then the controller will simply insert the TEID for the eNodeB into the entry. If the entry is not found, then the next packet will be processed without modifying the database .

4.2.2.2 Switching to the Cloud

Above, we presented the procedure to collect tunnel information. This section illustrates how the OpenFlow controller configures the mobile network gateway and data center gateway to switch the tunnel to the cloud in order to transparently replace the original GTP-U tunnel. In this way, virtual machines only receive the service request for the users that they can serve. The rest of the service requests are sent to the Internet. Thus, the tunnel switching process is completely transparent for users. The signaling process to realize this is complex. Figure 4.10 shows the entire procedure. The solid line arrows indicate that the tunnel switching switch procedure is in process and the actual communication is still between the eNodeB and the SGW. The dotted line arrows indicate the packet flow when the process of setting up the new tunnel is over and the eNodeB directly communicates with the process running in the virtual machine.

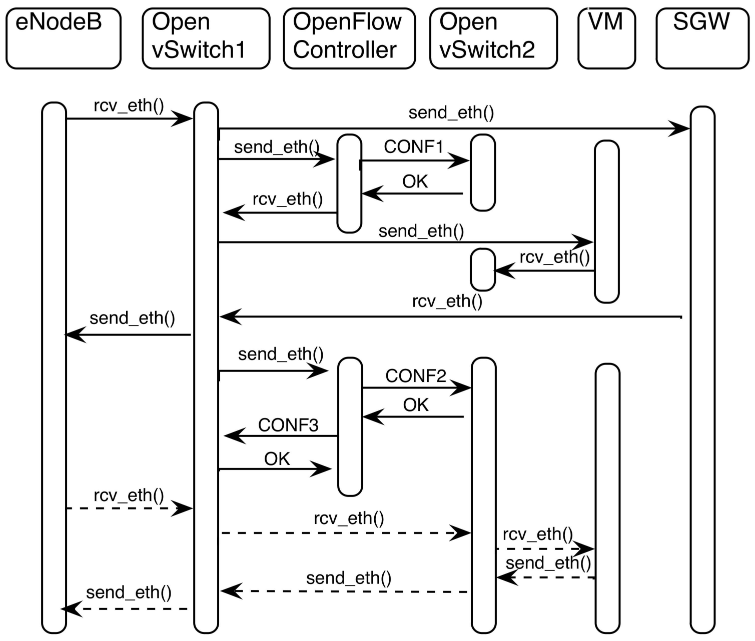

Figure 4.10: Switch signaling procedure

Before we introduce the tunnel switching signaling procedure, it is necessary to explain how the OpenFlow controller differentiates packets from different nodes. We pre-configure the port number for each network node in the mobile gateway. In the demonstration implementation, ports 1 and 2 are for the eNodeB and the SGW, respectively. The data center is connected via port 3. Port 4 is used for a control channel to/from the OpenFlow controller. In the code below, the packet from port1 invokes the function `processRBSPacket()` , while the function `processGWPacket()` is used to process a packet from SGW.

```
public Command receive(IOFSwitch sw, OFMessage msg,
                            FloodlightContext cntx) {
   OFPacketIn packetInMessage=(OFPacketIn) msg;
   byte[] payload=packetInMessage.getPacketData();

   switch (packetInMessage.getInPort()) {
       case 1:
           return this.processRBSPacket(sw,
                           packetInMessage, cntx, payload);
       case 2:
          return this.processGWPacket(sw,
                           packetInMessage, cntx, payload);
       default:
          return Command.CONTINUE;
            }
}
```

Listing 8: Main Function for processing GTP-U packet

In Figure 4.10, the Openflow controller configured the Open vSwitch2(the data center gateway) by sending it CONF1, after processing the packet received from the eNodeB. CONF1's content is:

```
ovs-vsctl add-port br0 gtp "teid_enode" -- set interface gtp
"teid_enode" type=gtp options:remote_ip="e_ip_dst_addr"

ovs-vsctl -- set interface gtp "X" type=gtp
options:in_key="teid_enode"

ovs-vsctl -- set interface gtp "teid_enode" type=gtp
options:arp_reply=true

ovs-ofctl add-flow br0 "in_port='teid_enode'
actions=mod_dl_dst:'mac_addr', output:'port'"
```

Listing 9: Tunnel Switch CONF1

The primary task of the controller is to configure the GTP port in Open vSwitch2 according to the tunnel information in the database. The tunnel ID should be "teid_node" in order to make the tunnel ID unique. If this is the first GTP port in the gateway, then the arp_reply spoofing option is also enabled. After this, an OpenFlow rule needs to be configured to ensure the delivery of packets from Open vSwitch2 to the correct virtual machine. However, we have not yet configured a rule for a packet being sent from the virtual machine to Open vSwitch2. This is also the reason why the packet from the virtual machine is blocked at Open vSwitch2 (as was shown in Figure 4.10).

In order to make the tunnel switching process completely transparent, we must ensure that the virtual machine and original server receive exactly the same packets. Thus, the Openflow controller also needs to send the same packets to virtual machine. This is done by the source code shown below. As we mentioned above, the port number for the data center is 3. However, in the Floodlight API, if you intend to send a the packet, the buffer ID should set as -1. Otherwise, the packet is never sent.

```
short egressPort=3;
OFPacketOut packetOutMessage = (OFPacketOut)floodlightProvider
.getOFMessageFactory().getMessage(OFType.PACKET_OUT);
// set initial length
short packetOutLength = (short)OFPacketOut.MINIMUM_LENGTH;
packetOutMessage.setBufferId(-1);
packetOutMessage.setInPort(pi.getInPort());
packetOutMessage.setActionsLength((short)OFActionOutput
.MINIMUM_LENGTH);
packetOutLength += OFActionOutput.MINIMUM_LENGTH;
```

Listing 10: Packet_out setting

After the OpenFlow controller receives a packet from the SGW, it should set up the appropriate configurations in both Open vSwitch1 and Open vSwitch2. This is done via CONF2 and CONF3. CONF2 is simple, it is simply:

```
ovs-vsctl -- set interface gtp "teid_enode" type=gtp
options:out_key="teid_sgw"
ovs-ofctl add-flow br0 "in_port='port',
action=output:'teid_enode'"
```

Listing 11: Tunnel Switch CONF2

Because the database already has the “teid_sgw”, the CONF2 need to configure the out_key as “teid_sgw”. The Openflow rule for traffic going the direction from the virtual machine to Open vSwitch2 also needs to be setup.

At this point, we already have a substitute tunnel. However, if we stop here, all the packets need to be processed by the OpenFlow controller. This will be slow and waste resources. Therefore, we configure the mobile gateway by sending it CONF3 . CONF3 contains the following rules:

```
ovs-ofctl add-flow br0 "priority=2 in_port=1
ip="e_ip_src_addr"
actions=mod_dl_dst:mac_addr_data_center,
mod_nw_dst:ip_addr_data_center,output:3

ovs-ofctl add-flow br0 "priority=2 in_port=3
actions=mod_dl_src:"mac_addr",
mod_nw_src:"i_ip_dst_addr",output:1

ovs-ofctl add-flow br0 "priority=2 in_port=2
ip="e_ip_src_addr" actions=drop
```

Listing 12: Tunnel Switch CONF3

The first rule will cause the mobile network gateway to deliver packets with the IP source address e_ip_src_addr to the data center. Thus, these packets are not longer sent to the OpenFlow controller. At this point we should mention the issue of packet matching. The TEID in GTP header field should be matched against the specific tunnel end point ID for the tunnel we are creating. However, the OpenFlow protocol does not support GTP matching in the present version of OpenFLow. Thus, GTP matching is left for future work. The Open vSwitch will only check if the packet's source address matches the source IP address that we have configured in this demonstration.

The second rule establishes a path from the data center to the eNodeB. Because the data center's IP address and MAC address are different from the SGW's IP and MAC addresses, the Open Switch should modify the source MAC address and source IP address. Finally, the packets being sent from the SGW in this same tunnel should be blocked, since we can not control the tunnel release in transparent mode.

4.2.2.3 Tunnel Flush

The eNodeB or SGW could release a GTP tunnel at any time. Therefore, our system need a mechanism to flush the tunnel when it is released. However, if we invoke any mobile networking signaling function, then the system is no longer transparent. The solution is to utilize the Open vSwitch traffic statistics and triggering the tunnel flush process from the Open vSwitch kernel module. If a tunnel does not have any traffic on it for a while, for example 30 seconds, then the tunnel can be flushed. However, implementation of this is left for future work.

Chapter 5

Evaluation

This chapter describes the network performance of the GTP-U tunnel. The GTP-U tunnel is compared with two other tunneling techniques (GRE and CAPWAP) with regard to four aspects: throughput, average latency, and jitter. Although the main theme of this chapter is the evaluation of the GTP-U tunnel technology, GTP-U has also been compared with different configurations of the OpenFlow protocol. The reason is that there are two solutions to complete a GTP-U tunnel. One solution uses the OpenFlow protocol for layer 3 switching. Another solution simply enables the promiscuous mode of the virtual machine, so that the virtual machine will ignore the ethernet header. Therefore, this chapter describes two different experiments to evaluate these alternative solutions. For each experiment we will give a detailed description of how the experiment was performed. The analysis of these experiments' results is also presented.

5.1 Experimental Set Ups

In this thesis project, two experiments were set up with two Linux machines. Each Linux machine has a pre-installed Open vSwitch and a virtual machine. The first experiment is a tunnel performance test in which three different tunnel technologies, GRE, CAPWAP, and GTP-U, have been compared for a range of different packet size, different loads, and different Maximum Segment Sizes. The second experiment focuses on the performance of the OpenFlow protocol. Two different OpenFlow configurations are compared. The metric of the second experiment is same as the first experiment. In these experiments, we use five criterion to evaluate the network's performance, these are UDP throughput, TCP throughput, latency, and UDP Jitter. For UDP and TCP throughput tests, sender A offers different UDP or TCP packet loads to receiver B. The received data rate is considered as the UDP or TCP throughput. The latency is measured with a ping program (since ping is the easy way to perform latency test). Latency is the average round trip time of a ping test. Jitter are measured for UDP packets.

As illustrated in Figure 5.1, these experiments use two Linux machines connected with a 1GB NIC. Two Linux machines are connected with a 1 Gbps unshielded twisted pair Ethernet cable with a length of 5m. For each Linux machine should has a pre-installed VirtualBox version 4.2.4, Open vSwitch, and Iperf version 2.0.5 (Details of this software can be found at [41] and [42]). The Iperf network traffic generator is used to generate UDP and TCP traffic. One Linux machine runs the Iperf server and the other linux machine runs the Iperf client. For Open vSwitch version 1.4.0 was used with the GTP-U tunnel extension (as implemented in this project). The detailed hardware and software configuration is presented in Appendix A.

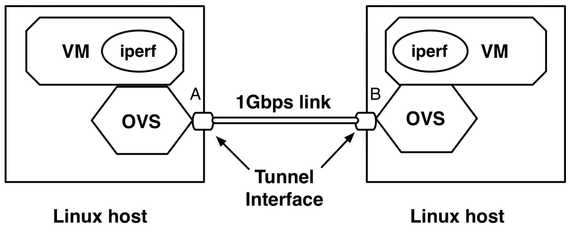

Figure 5.1: Evaluation Testbed

As described above, the performance experiments were conducted using two nodes. In each test, the client node sends UDP packets or TCP packets at a constant rate to the server node. For both the TCP and UDP tests, the experiment was repeated with various packet sizes, maximum segment sizes, and loads.

Before describing the evaluation results, two points need to be mentioned. Firstly, in order to increase the accuracy, all the test processes were repeated 5 times to enhance accuracy and the average of these five tests is reported. Secondly, in the throughtput test, we only present the received throughput.

5.2 Experiment 1: Performance of different tunnel technologies as a function of packet length and load

This experiment examines the different tunneling methods' behaviours in terms of throughput, latency, and jitter, when the GRE, CAPWAP, or GTP-U tunnel technology is adopted. In experiment one, we utilized first OpenFlow solution , i.e., we only specifying the input port and output port in Open vSwitch with

OpenFlow Protocol, but enable the promiscuous mode for the virtual machine's NIC. Please refer to section 4.1.2.3 for the details of this OpenFlow solution to accomplish GTP-U tunneling. The experimental results are shown in Figures 5.2 – Figure 5.10.

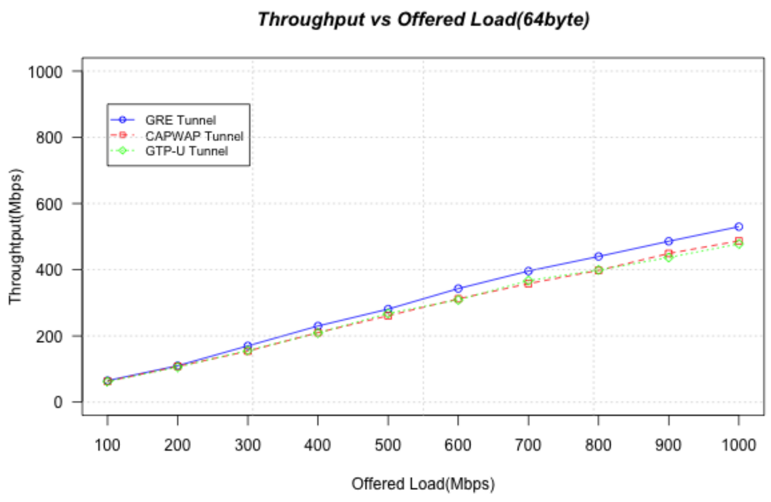

Figure 5.2: Throughput vs. Offered Load (UDP, Packet Size 64byte)-Various tunnel types

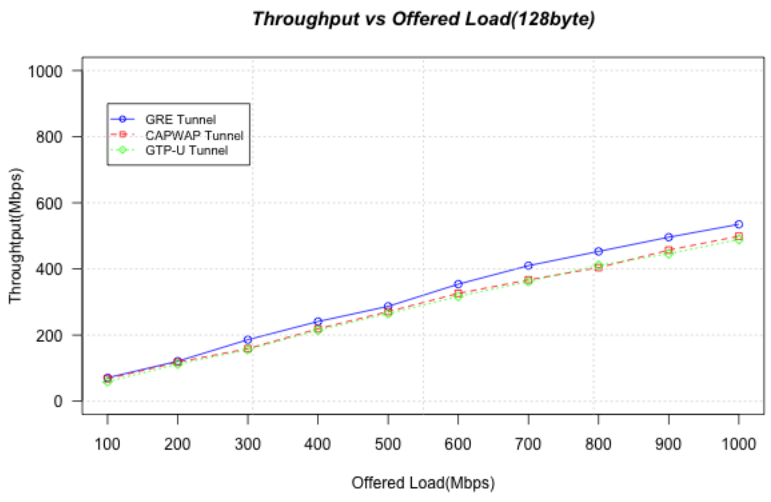

Figure 5.3: Throughput vs. Offered Load (UDP, Packet Size 128byte)-Various tunnel types

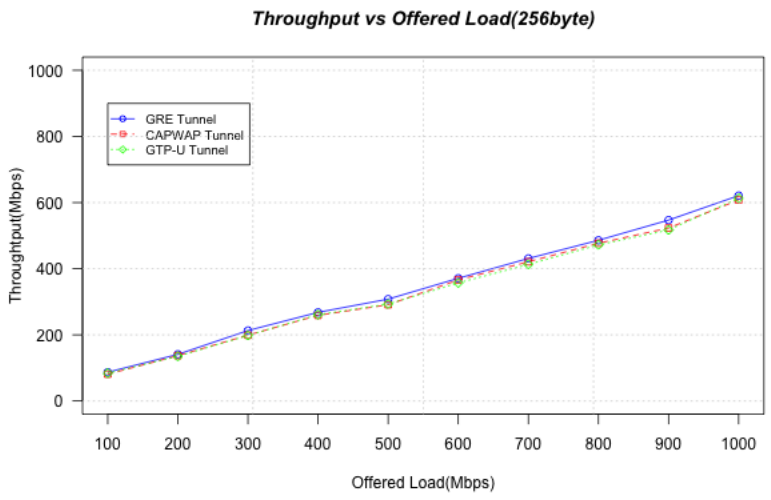

Figure 5.4: Throughput vs. Offered Load (UDP, Packet Size 256byte)-Various tunnel types

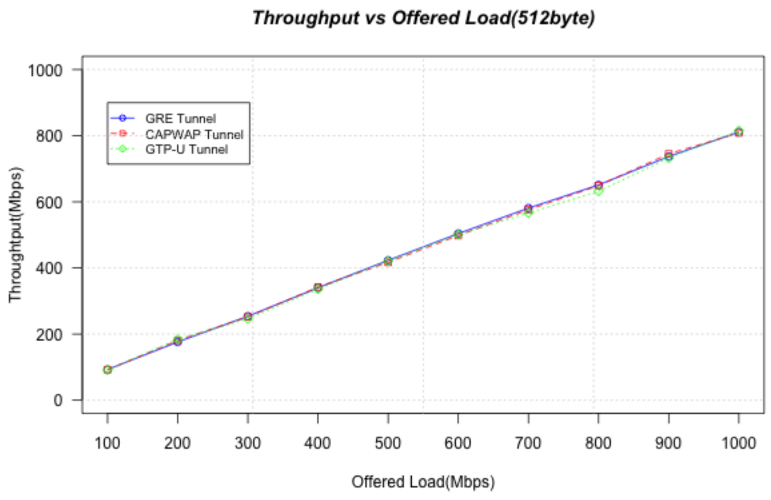

Figure 5.5: Throughput vs. Offered Load (UDP, Packet Size 512byte)-Various tunnel types

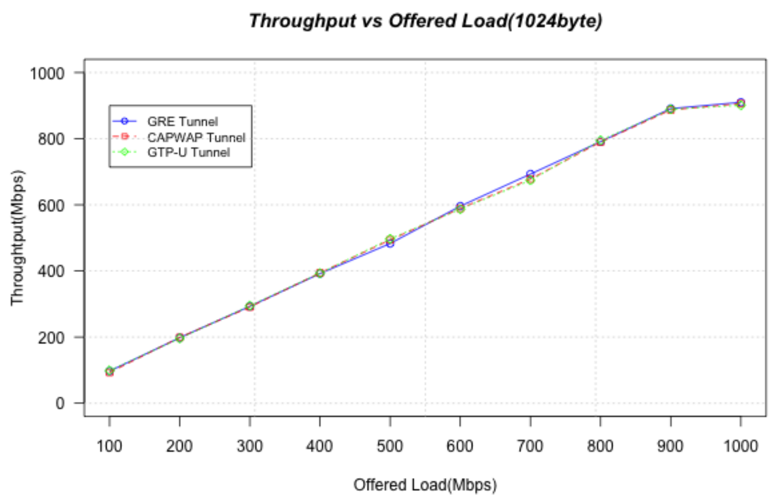

Figure 5.6: Throughput vs. Offered Load (UDP, Packet Size 1024byte)-Various tunnel types

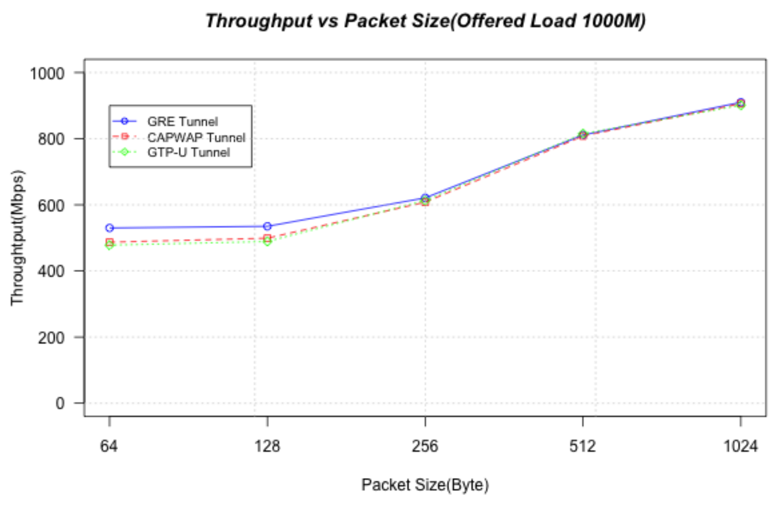

Figure 5.7: Throughput vs. Packet Size (UDP, Offered Load 1000Mbps)-Various tunnel types

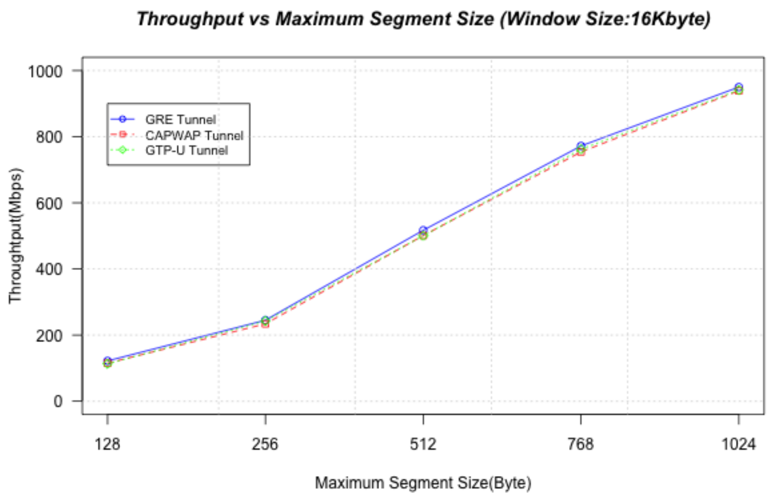

Figure 5.8: Throughput vs. Maximum Segment Size (TCP, Window Size 16Kbyte)-Various tunnel types

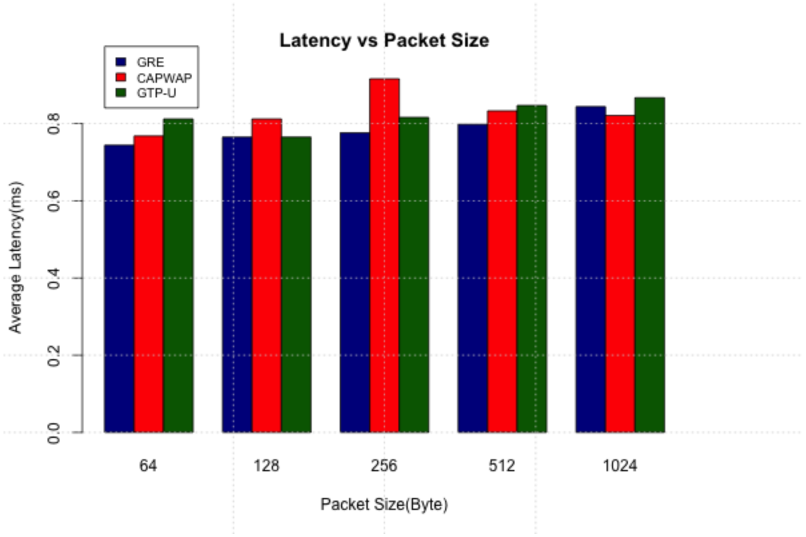

Figure 5.9: Latency vs. Packet Size (ICMP)-Various tunnel types

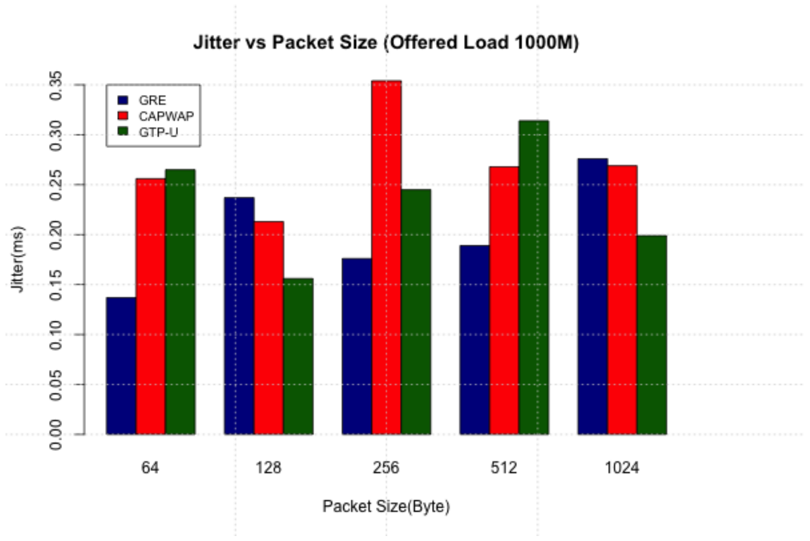

Figure 5.10: Jitter vs. Packet Size (UDP, Offered Load 1000Mbps)-Various tunnel types

5.2.1 Analysis

For experiment 1, we have graphs for different loads and with different UDP packet sizes and TCP maximum segment sizes for GRE, CAPWAP, and GTP-U tunnel technologies. In Figures 5.2, 5.3, 5.4, 5.5, 5,6, 5.7, and 5.8, we can see that the performance of the three tunnel technologies are same basically the except for the GRE tunnel when the packet size is smaller than 256K bytes. In this later case GRE has a somewhat better performance in terms of throughput. This phenomenon occurs because GRE is an IP layer encapsulation and CAPWAP and GTP-U are transport layer encapsulation. When CAPWAP and GTP-U process small packets, the tunnel overhead is higher that for GRE. However, there exsits a interesting phenomenon, the offered load not equal to the throughput in these figures. The reason behind it is when packet size is small, the CPU could be overload in this test bed. The TCP test results shown in Figure 5.8 shows a similar result. Figure 5.9 and 5.10 illustrate latency and jitter, respectively, for these three tunneling protocols. We can see that the performance of three tunnel technology are similar. Overall, the implemention of GTP-U with the first openflow solution was successful as the implementation shows a quite good performance (i.e., it is comparable to the performance of CAPWAP and GRE).

5.3 Experiment 2: Performance of GTP-U tunnel with different OpenFlow configurations

This second experiment shows the different behaviour of the three tunneling protocols in terms of throughput, latency, and jitter, when two different OpenFlow configurations are selected. CONF1 only specifies the input port and output port in Open vSwitch with the OpenFlow Protocol, but enables the promiscuous mode of the virtual machine's NIC. CONF2, enables layer 3 mapping and ethernet header modification. In this later case the NIC works as normal. Please refer to section 4.1.2.3 for the details of these OpenFlow solutions to accomplish GTP-U tunneling. These results are shown in Figure 5.11 to 5.19.

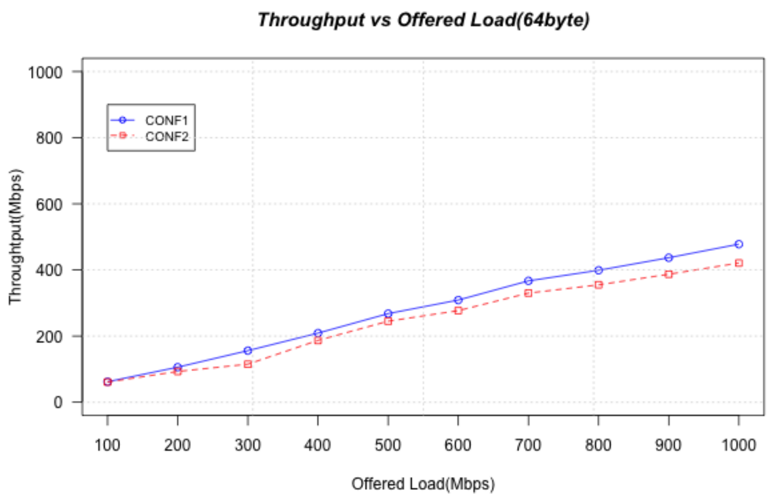

Figure 5.11: Throughput vs. Offered Load (UDP, Packet Size 64byte) with two OpenFlow configurations

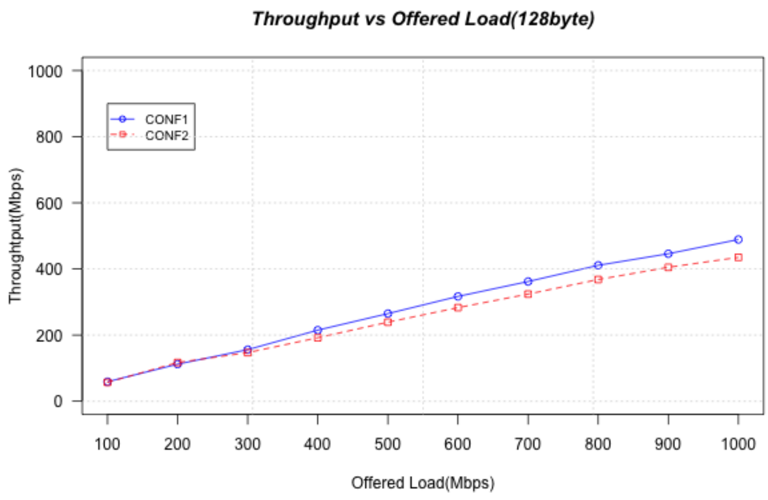

Figure 5.12: Throughput vs. Offered Load (UDP, Packet Size 128byte) with two OpenFlow configurations

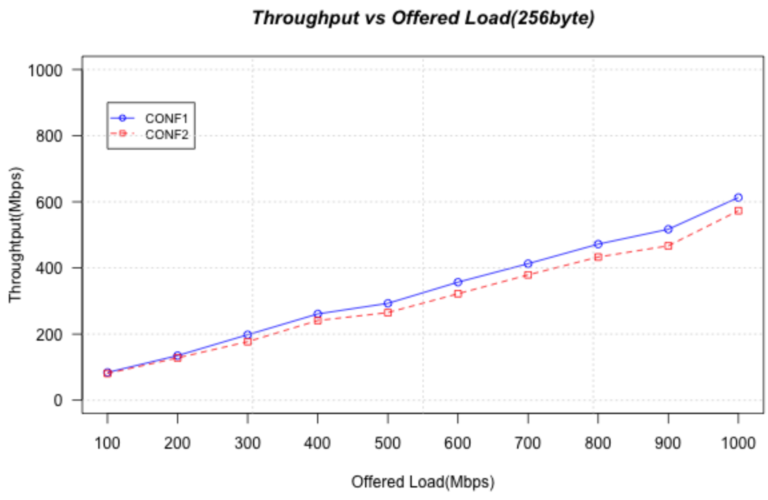

Figure 5.13: Throughput vs. Offered Load (UDP, Packet Size 256byte) with two OpenFlow configurations

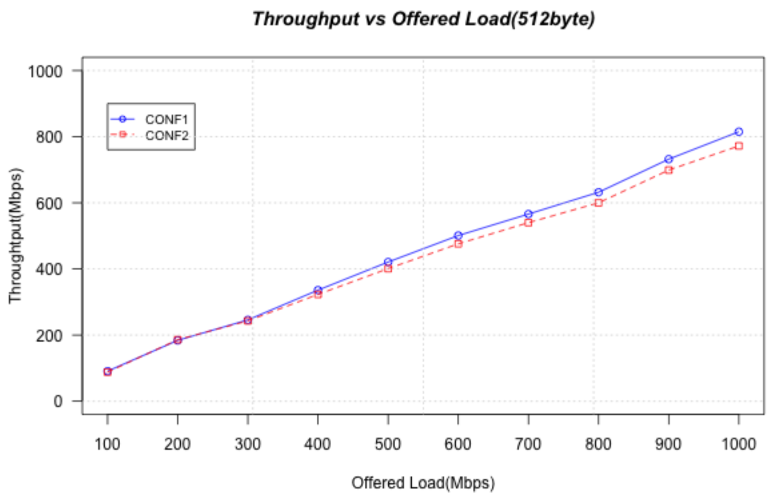

Figure 5.14: Throughput vs. Offered Load (UDP, Packet Size 512byte) with two OpenFlow configurations

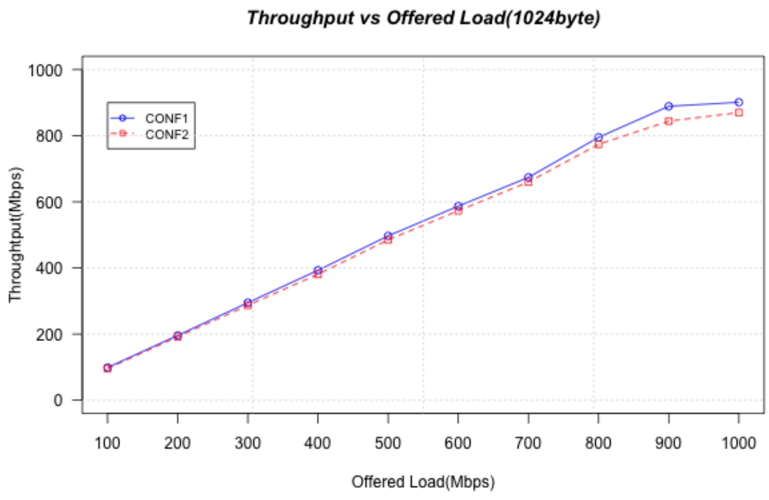

Figure 5.15: Throughput vs. Offered Load (UDP, Packet Size 1024byte) with two OpenFlow configurations

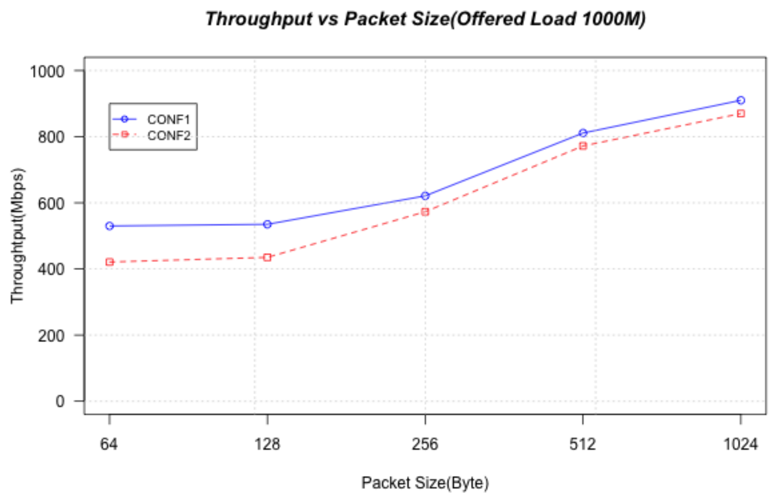

Figure 5.16: Throughput vs. Packet Size (UDP, Offered Load 1000MB) with two OpenFlow configurations

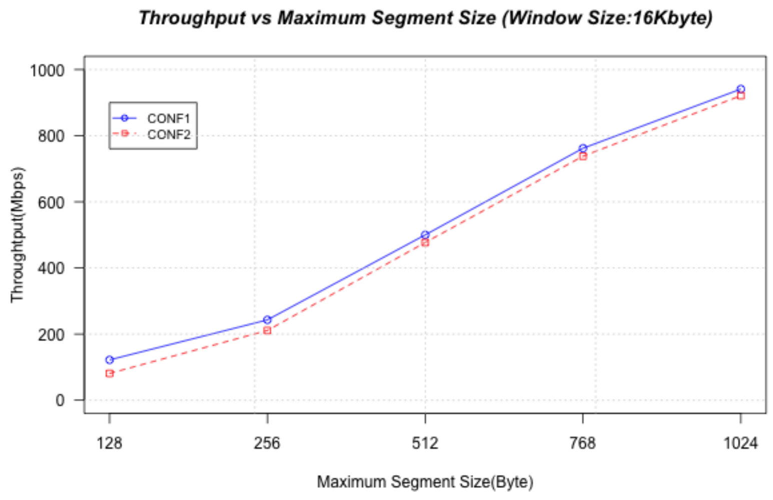

Figure 5.17: Throughput vs. Maximum Segment Size (TCP, Window Size 16Kbyte) with two OpenFlow configurations

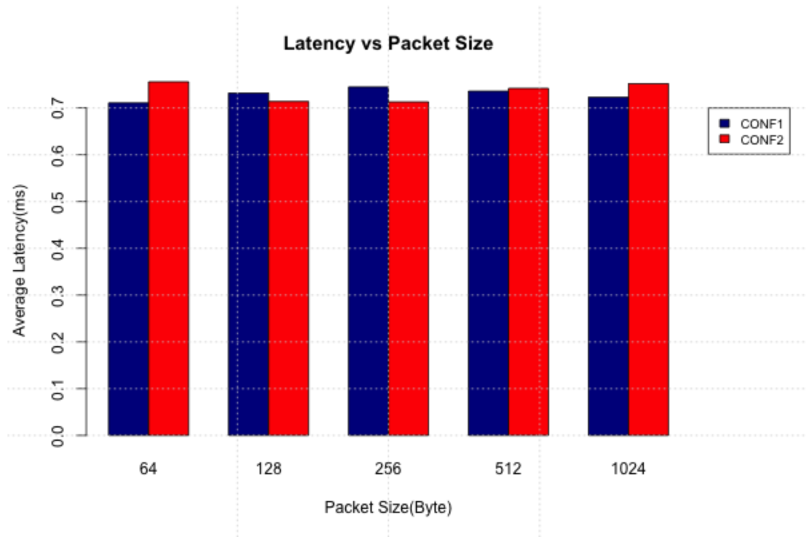

Figure 5.18: Latency vs. Packet Size (ICMP) with two OpenFlow configurations

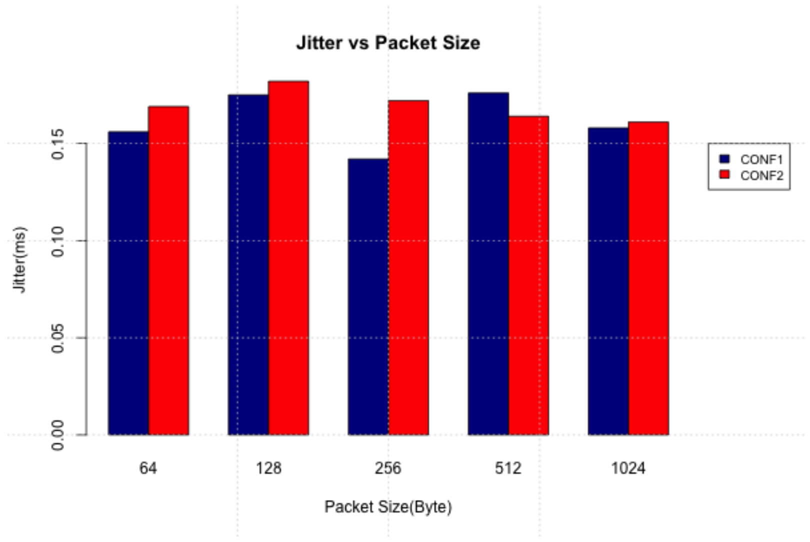

Figure 5.19: Jitter vs. Packet Size (UDP, Offered Load 1000Mbps) with two OpenFlow configurations

5.3.1 Analysis

The results of experiment 2 for different loads and with different UDP packet sizes and TCP maximum segment size for different OpenFlow Solutions have been shown above. In Figure 5.11, 5.12, 5.13, 5.14, and 5.15, we can see that the performance of the first solution is better than the second solution when the offered load is greater than 500 Mbps. However, as shown in Figure 5.16, the throughput difference decreases when the packet size increases. Figure 5.17 illustrates that the TCP performances of the two solutions are mostly the same. The layer 3 mapping and Ethernet header modification increase the overhead of the GTP-U tunnel slightly. As for latency and jitter, the results are similar for these two solutions as shown in Figure 5.18 and 5.19. The performance of the first solution is better. However, there is one drawback we have to enable promiscuous mode for this solution and this increases the resources used by the virtual machine to process all of those packet which are not destined to it. The second solution decreases the performance slightly, but retains the standard Internet five-layer model.

Chapter 6

Conclusions and Future work

In this section we will state our conclusions and insights gained as result of this thesis project. Some future work is suggested. The chapter ends with some reflections on the social, economic, ethical, and other aspects of this thesis project.

6.1 Conclusions

In this thesis we have described a potential evolutionary path for the mobile network to a cloud ecosystem. Based upon some primary design principles for SDN and mobile networks, the project designed an architecture for a mobile network cloud and presented a set of cloud entities that could be utilized to realize the proposed mobile network cloud architecture. Mobile network cloud evolution can be viewed as a revolutionary approach in the packet core network for future mobile networks. The experimental implementation of the transparent phase of this evolution was also demonstrated and evaluated. By exploring the implementation possibilities and the design alternatives, we gained knowledge of SDN and the flexibility and power of the SDN based protocol OpenFlow. As a part of the implementation process, the key enabler for mobile network cloud evolution is the mobile network tunneling protocol-GTP. The user plane of the GTP protocol was implemented in a SDN based switch-Open vSwitch. Exploiting the control plan and data plane decoupling, only the GTP-U needed to be implemented. GTP-C could be a part of any mobile network control entity, thus it can be easily implemented as a control backend in the OpenFlow controller as described in section 3.3.2. As shown in Chapter 5, the performance of our GTP-U tunnel implementation was excellent, as its performance was comparable to the existing GRE and CAPWAP tunneling protocols in Open vSwitch.

It is expected that the research for SDN and GTP tunneling mechanism will be considered as building blocks for the overall design of future mobile network's cloud evolution. A lot of works remains to be done from an engineering perspective to realized the proposed architecture and control entities. However, as stated above, cloud computing and SDN are significant in defining the direction of the evolution of mobile networks.

6.2 Future Work

In this section, we suggest some future work with regards to our transparent mode implementation and decoupling phases. We first describe which of the parts of the existing implementation have to be improved in order to make the proposed solution work in a practical environment. We then describe some issues regarding the design of and decoupling phase.

6.2.1 Transparent Mode

The future work regarding the transparent mode can be split into the following topics: Server Content Catalog, Tunnel Flush, OpenFlow GTP header matching, and IPv6 support. The each of these areas is described in more detail below.

Server Content Catalog
The implementation for the entity is a table in the MySQL database was described in Section 4.2.2.1. However, this implementation could be more complicated in a practical environment. OpenFlow controller may need to analyze application layer information. A good solution is to use the Publish and Subscribe Internet Routing Paradigm[37]. This is a information centric communication model. The main idea is the server publishes the content that it has, then the clients subscribe to the desired content. Because clients would subscribe to the server in the data center, it is easier for the OpenFlow controller to decide whether it can server these clients.

Tunnel Flush
As discussed in section 4.2.2.3, the Open vSwitch statistics include all the packet information that has been collected about the packets flowing through the switch. We could use this information to trigger the tunnel flush process from the Open vSwitch kernel module, whenever a tunnel has not been in use for a while.

TCP Sequence Number Process
As described in section 4.2.2.1, in order to be completely transparent, we need to ensure the sequence numbers generate from content server and original server in ACK of TCP handshake phase could be exactly the same. Otherwise, it may

trigger the TCP connection re-establishment. Then, the transparent phase is not transparent anymore.

OpenFlow GTP Header Match
In section 4.2.2.2, configuration CONF3 sets up a rule based on an IP address. However, this is not a good choice. Even for the same tunnel end points, there may exists several tunnels. The optimal solution is to match the TEID directly, but the OpenFlow protocol does not yet support this matching. We should extend this protocol to support this matching. The GTP header matching could be implemented in the Open vSwitch or Floodlight OpenFlow controller. In our scenario, all of the packets are processed by the Mobile Network Gateway. Thus, implementing GTP header matching in Open vSwitch is preferable.

IPv6 support
For experimental purposes, the implementation described in this thesis was based on IPv4. However, IPv6 is inevitable , especially in mobile networks. Naturally, a future implementations should include support for IPv6.

Latency Evaluation
In the evaluation section, the latency evaluation is simply performed with ping program. However, it may not be accurate enough. Another better solution is using UTP synchronize clock with wireshark to do latency measurements.

Mobile Network Gateway
In the design section, the mobile network gateway is deployed between eNodeB and SGW to achieve CDN like network architecture. However, it would make the traffic processing more complex. If we deploy mobile network gateway between SGW and PGW, then all the control mechanisms related to MME can be avoid. This is a good idea that deserving further study.

6.2.2 Decoupling Phase Design

The design in chapter 3 only suggests the framework for Decoupling phases. To realize this phase, additional issues need to be considered. Some of these issues are listed below.

Dimensioning the Access Network
SDN is a centralized control mechanism. However, the access network should be distributed because the mobile subscribers are scattered over the service area. The issue of correctly dimensioning our cloud ecosystem is very important to be considered. A high-density deployment may waste resources, because the

subscribers in the service area may not be distributed with high density. However, low-density deployment may also induce bad performance.

Consider of Heterogeneous Wireless Access Networks
The design is presented in this thesis was on a modern LTE network. However, LTE is not the only option for a broadband wireless access network. It is expected that future access network will consists of a variety of heterogeneous network environments. These access networks may have different protocol stacks. How to best communicate via these heterogeneous networks is an issue that needs to be considered in future research.

6.3 Required Reflections

Our society is experiencing network evolution which has changed our life in many aspects. Broadband networks, mobile communication, and cloud computing are three significant parts of this evolution. It is expected that, in the next few years, technological progresses in network area will bring new opportunities to society for innovation, and thus change the world. Our communication will be more convenient and efficient.

The combination of broadband, mobile communication, and cloud computing is a promising way of evolution. This thesis notices this trend and provides a experimental implementation for applying cloud computing to broadband mobile networks (LTE). This project is conducted in the cloud computing department of Ericsson. Due to the lack of LTE network equipment, we can only perform the implementation in some linux hosts and introduce the LTE tunneling mechanism (GTP-U). It may cause the results of this thesis are impractical in many aspects. However, it is expected that the research for SDN and GTP tunneling mechanism will be considered as building blocks for the overall design of future mobile network's cloud evolution. If this evolution becomes reality in the near future, it will reduce mobile operators' operating costs and provide better services for subscribers.

In this thesis project, I deeply feel that I need to learn more knowledge of this area, such as cloud computing's resource allocation, as well as detailed knowledge of mobile broadband. I hope that I can continue working in this field and make it to be reality.

Bibliography

[1] "3GPP TS 23.002 V11.3.0: Network architecture(Release 11) " *Technical Specification Group Services and System Aspects, June 2012*. 3rd Generation Partnership Project, pp. 28–29

[2] A. Kumar, J. Sengupta and Y. Liu " 3GPP LTE: The Future of Mobile Broadband" *Wireless Personal Communications,vol:62 iss:3 February 2012*, Springer,Netherlands , pp. 671–686.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia "Above the Clouds: A Berkeley View of Cloud Computing," *Communications of the ACM Volume 53 Issue 4, April 2010*, ACM, New York, NY, USA , pp. 50–58.

[4] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso "Calling the cloud: Enabling mobile phones as interfaces to cloud applications" *Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware, 2009*, Springer-Verlag,Berlin, Heidelberg, pp. 83–102. http://dl.acm.org/citation.cfm?id=1813355.1813362, Last access date: 2013-02-17

[5] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti "CloneCloud: elastic execution between mobile device and cloud" *Proceedings of the sixth conference on Computer systems, 2011*, ACM, New York, NY, USA , pp. 301–314. DOI 10.1145/1966445.1966473 http://doi.acm.org/10.1145/1966445.1966473, Last access date: 2013-02-17

[6] Z. Zhu, P. Gupta, Q. Wang, S. Kalyanaraman, Y. Lin, H. Franke, and S. Sarangi "Virtual Base Station Pool: Towards A Wireless Network Cloud for Radio Access Networks" *Proceedings of the 8th ACM International Conference on Computing Frontiers, 2011*, ACM, New York, NY, USA , pp. 34:1–34:10. DOI 10.1145/2016604.2016646 http://doi.acm.org/10.1145/2016604.2016646, Last access date: 2013-02-17

[7] S. Vassilaras and G. Yovanof "Wireless Going in the Cloud: A Promising Concept or Just Marketing Hype?" *Wireless Pers Communication, Volume*

58, Issue 1, May 2011, Published online: 9 April 2011 Springer Science and Business Mediapp,pp. 5–16. DOI 10.1007/s11277-011-0284-9 http://link.springer.com/article/10.1007/s11277-011-0284-9, Last access date: 2013-02-17

[8] I. Albarrán Munoz and M. Parras Ruiz De Azúa, “Telecommunication Services’ Migration to the Cloud: Network Performance analysis” *Royal Institute of Technology(KTH), School of Information and Communication Technology (ICT), Communication Systems (CoS), May 2012*, http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-93841, Last access date: 2013-01-20

[9] “ 3GPP TS 29.281 V8.0.0: General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U) (Release 8)” *Technical Specification Group Core Network and Terminals, December, 2008*. 3rd Generation Partnership Project.

[10] “3GPP TS 29.060 V3.19.0 : General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface (Release 1999) ” *Technical Specification Group Core Network, March 2004*. 3rd Generation Partnership Project, pp. 65–71

[11] “3GPP TS 29.274 V8.11.0: 3GPP Evolved Packet System (EPS); Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C); Stage 3 (Release 8)” *Technical Specification Group Core Network and Terminals, 2011-2012*. 3rd Generation Partnership Project.

[12] C. Barker, “HP dismisses cloud ‘hype’ ”, Web resource, http://www.zdnet.com/news/hp-dismisses-cloud-hype/255222. Last access date: 2013-01-20

[13] S. Barrie, “Chapter 1. Defining Cloud Computing” *Cloud Computing Bible*, John Wiley and Sons, 2011, pp. 24

[14] E. Dudin and Y. Smetanin, “ A review of cloud computing” *Scientific and Technical Information Processing,Volume 38 Issue 4*, Allerton Press, Inc., October 2011, pp. 280–284

[15] S. Zhang, H. Yan, and X. Chen, “Research on Key Technologies of Cloud Computing” *Physics Procedia, Volume 33*, ICMPBE2012, 2012, pp. 1791–1797.

[16] “Software Defined Networking: The New Norm for Networks” *ONF White Paper April 13, 2012*,

[17] G. Goth, “Software-Defined Networking Could Shake Up More than Packets” *Internet Computing,vol.15, no.4,July-Aug.2011*, IEEE, pp. 6–9.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks” *SIGCOMM Comput. Commun. Rev. 38, 2, March 2008*, ACM, New York, NY, USA , pp. 42–47.

[19] H. Brandon, “OpenFlow Switch Specification, Version 1.0.0” *Open Networking Foundation(ONF), December 2009*, `http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf`, Last access date: 2013-01-20

[20] J. Pettit, J. Gross, B. Pfaff, M. Casado and S. Crosby, “ Virtual Switching in an Era of Advanced Edges” `http://openvswitch.org/papers/dccaves2010.pdf`, Last access date: 2013-01-20

[21] OpenvSwitch.org, “Why Open vSwitch?” `http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=WHY-OVS`, Last access date: 2013-01-20

[22] S. Horman, “An Introduction to Open vSwitch” *LinuxCon Japan, Yokohama, June 2nd, 2011*, `http://openvswitch.org/slides/openvswitch.en-2.pdf`, Last access date: 2013-01-20

[23] OpenvSwitch.org, “Open vSwitch” `http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=README;hb=HEAD`, Last access date: 2013-01-20

[24] OpenvSwitch.org, “sFlow” `http://openvswitch.org/support/config-cookbooks/sflow/`, Last access date: 2013-01-20

[25] C. Benvenuti, “Understanding Linux Network Internals” *December 2005*, O’Reilly, Sebastopol, USA, Ebook ISBN: 978-0-596-10367-5, ISBN 10: 0-596-10367- pp. 58–71.

[26] P. N. Ayuso, R. M. Gasca, and L. Lefévre, “Communicating between the kernel and user-space in Linux using Netlink sockets” *Software: Practice and Experience, August 2010*, pp. 797–810.

[27] W. Stevens, B. Fenner, and A. Rudoff, “ Network Programming Volume 1, Third Edition: The Sockets Networking API” *volume 1,UNIX Network Programming*, Addison Wesley Professional, third edition, 2003.

[28] LinuxFoundation.org "iproute2" *Web Resource*, http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2, Last access date: 2013-01-20

[29] netfilter.org "The netfilter.org project", *Web Resource, Number 19,2009*, http://www.netfilter.org/, Last access date: 2013-01-20

[30] Kevin Kaichuan He, "Kernel Korner - Why and How to Use Netlink Socket" *Linux Journal, Janunary 05, 2005*, http://www.linuxjournal.com/article/7356, Last access date: 2013-01-20

[31] K. Johnson, J. Carr, M. Day, and M. F. Kaashoek " The Measured Performance of Content Distribution Networks" *Computer Communications, 2001*,pp. 202–206 http://www.cs.rutgers.edu/~rmartin/teaching/spring02/cs553/readings/johnson00.pdf, Last access date: 2013-01-20

[32] P. Costa, M. Migliavacca, P. Pietzuch, and A. Wolf " NaaS: Network-as-a-Service in the Cloud" *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE'12),April 2012*, USENIX,San Jose, CA, USA

[33] Bengt. Möolleryd, J. Märkendahl, and Ö. Makitalo " Analysis of operator options to reduce the impact of the revenue gap caused by flat rate mobile broadband subscriptions" *8th Conf. on Telecom, Media and Internet Tele-Economics, June 2009*

[34] "3GPP TS 29.303 version 8.1.0: Universal Mobile Telecommunications System (UMTS); LTE; Domain Name System Procedures; Stage 3 (Release 8)" *Technical Specification, April 2009* 3rd Generation Partnership Project.

[35] V. Vinothina, R. Sridaran, and PadmavathiGanapathi, " A Survey on Resource Allocation Strategies in Cloud Computing" *International journal of advanced computer science and applications volume 3 issue 6 2012*, pp. 97

[36] G. Wei, A. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services" *The Journal of Supercomputing, Volume 54, Number 2,2010*, pp. 252–269

[37] D. Lagutin, K. Visala, and S. Tarkoma, "Publish/Subscribe for Internet: PSIRP Perspective" http://www.booksonline.iospress.nl/Content/View.aspx?piid=16473, Last access date: 2013-01-20

[38] LinuxFoundation.org, “generic_netlink_howto”, *Web Resource, November 19,2009* `http://www.linuxfoundation.org/collaborate/workgroups/networking/generic_netlink_howto`, Last access date: 2013-01-20

[39] OpenFlowHub.org, “Floodlight OpenFlow Controller”, *Web Resource*, `http://floodlight.openflowhub.org/`, Last access date: 2013-01-20

[40] VirtualBox.org, “Oracle VM virtualbox”, *Web Resource*, `https://www.virtualbox.org/`, Last access date: 2013-01-20

[41] SourceForge.net, “Iperf”, *Web Resource*, `http://sourceforge.net/projects/iperf/`, Last access date: 2013-02-03

[42] OpenManiak.com, “IPERF - the easy tutorial”, *Web Resource*, `http://sourceforge.net/projects/iperf/`, Last access date: 2013-02-03

Appendix A

Testbed Hardware and Software Configuration

LInux Host 1
CPU: Intel - SLG9K - Intel 6Core E7450 2.4ghz-12mb
Memory: 4GB
Disk: 500GB
NIC: Intel A92165-004 PRO/1000 Mt 10/100/1000
Operating system: Ubuntu 12.04
Software installed: VirtualBox 4.2.6 for Linux hosts and Iperf 2.0.3 released

Linux Host 2
CPU: Intel - SLG9K - Intel 6Core E7450 2.4ghz-12mb
Memory: 4GB
Disk: 500GB
NIC1: Intel A92165-004 PRO/1000 Mt 10/100/1000
Operating system: Ubuntu 12.04
Software installed: VirtualBox 4.2.6 for Linux hosts and Iperf 2.0.3 released

VirtualBox Hardware Configuration
Memory: 2GB
Disk: 30GB
NIC: Intel PRO/1000 MT Desktop Adapter Network adapter

GRE tunnel Configuration
“X" can be adapt to your own network parameter.

```
ovs-vsctl add-port br0 gre0 – set interface gre0 type=gre options:remote_ip=X.X.X.X
ovs-vsctl – set interface gtp0 type=gtp options:in_key=X
ovs-vsctl – set interface gtp0 type=gtp options:out_key=X
```

CAPWAP tunnel Configuration

```
ovs-vsctl add-port br0 capwap0 – set interface capwap0 type=capwap
options:remote_ip=X.X.X.X
ovs-vsctl – set interface gtp0 type=gtp options:in_key=X
ovs-vsctl – set interface gtp0 type=gtp options:out_key=X
```

GTP-U tunnel Configuration

```
ovs-vsctl add-port br0 gtp0 – set interface gtp0 type=gtp options:remote_ip=X.X.X.X
ovs-vsctl – set interface gtp0 type=gtp options:in_key=X
ovs-vsctl – set interface gtp0 type=gtp options:out_key=X
ovs-vsctl – set interface gtp0 type=gtp options:arp_reply=true
```

Openflow Solution 1

Prerequisite: enabling promiscuous mode for VM's NIC.

```
ovs-ofctl add-flow br0 "priority=65535 in_port=1 actions=output:2"
ovs-ofctl add-flow br0 "priority=65535 in_port=2 actions=output:1"
```

Openflow Solution 2

```
ovs-ofctl add-flow br0 "priority=65535 in_port=1 dl_dst=XX:XX:XX:XX:XX:XX
nw_dst=X.X.X.X,actions=output:2"
ovs-ofctl add-flow br0 "priority=65535 in_port=2 dl_dst=XX:XX:XX:XX:XX:XX
nw_dst=X.X.X.X,actions=mod_dl_dst:08:00:27:d8:05:1c,output:1"
```

Appendix B

Software Installation Tutorial

Open vSwitch
Please referring to official guide "How to Install Open vSwitch on Linux and FreeBSD". `http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=INSTALL;hb=HEAD`

Floodlight
Please referring to official guide "Get started". `http://floodlight.openflowhub.org/getting-started/`

VirtualBox
sudo apt-get install virtualbox (Valid for Linux Ubuntu distribution)

MySQL
Please referring to official guide "Installing MySQL on Linux". `http://dev.mysql.com/doc/refman/5.1/en/linux-installation.html`

Iperf
sudo apt-get install iperf (Valid for Linux Ubuntu distribution)

TRITA-ICT-EX-2013:25