MOSA: A Mobile Offloading Architecture using Software-defined network*

Junguk Cho - Binh Nguyen[1] and Jacobus Van Der Merwe[1]

[1]School of Computing, University of Utah

Abstract

As delay-sensitive traffic such as online real time gaming requires a stringent delay budget, moving services close to users is beneficial. The current cellular architecture, however, hinders the efforts of reducing end-to-end delay due to issues such as hierarchical routing and the speed of light constraint. We propose an offloading architecture for cellular network using Software-defined network (SDN) to snoop and transparently reroute traffic to servers located inside the cellular core. The approach shows significant network performance improvement by eliminating both hierarchical routing and Internet delay issues while leaving the current architecture untouched.

Keyword

service offloading, Software-defined network (SDN)

1 Introduction

As the cellular network is moving toward all-IP, accessing a variety of services using smart phones or tablets has become more and more popular. While the popular web browsing does not require a strict Quality of Service (QoS) requirement, services such as gaming and cloud services require a certain delay and bandwidth budget in order to operate appropriately. These QoS realizations are deeply integrated into the current cellular network design as multiple QCIs (QoS class of identifier) are specified in 3GPP specifications. For example, as in [2], online real time gaming services require 50ms delay budget with a guaranteed bit-rate (GBR), interactive gaming services require 100ms of delay budget.

The current cellular network architecture, however, has some limitations that hinder the efforts to deeply reduce network delay. As mentioned in [4], the number of gateways in 3G networks is small, and since these gateways are the Internet access points, traffic in cellular network have to pass through these geographically distributed nodes and this adds additional delay to the path as opposed to direct routing (hierarchical routing issue [6]). Moreover, despite the fact that the recent network design (e.i., 4G/Longterm Evolution and Evolved Packet Core or 4G/LTE-EPC) is flat and the delay caused by the cellular core network is reduced (the average end-to-end RTT is about 70ms in 4G/LTE [9]), end-to-end delay still suffers from and is gradually dominated by the Internet delay (the delay between cellular gateways and Internet servers). This delay could not be eliminated by any mean due to the constraint of speed of light.

There are approaches that try to reduce the end-to-end delay in cellular networks by leveraging Content Delivery Network (CDN) and knowledge about the geographical distribution of cellular gateways. For example, cloud providers move their CDN servers close to the cellular core network to

*Potential CFP: ICCCN 2014

reduce the Internet delay portion; knowledge about the geographical distribution of IP addresses of mobile devices aids the CDN servers placement [13]; or grouping P2P (peer-to-peer) devices that are geographically closed could reduce the delay between them [3]. All of the above works, however, concentrate on the Internet portion of the end-to-end delay yet can not further reduce the total delay due to the hierarchical routing issue remained inside the core. Moreover, the fact that the number of cellular gateways is small and these gateways are geographically distributed adds another degree of complexity to the above approaches.

Our work is strongly inspired by MOCA [5]. MOCA proposed an offloading architecture for cellular networks in which services can be placed inside the core network to eliminate both the hierarchical routing issue and Internet delay. MOCA leverages SDN (Software-defined Network) to offload traffic to a local SPGW (service and packet gateway) instance and an offloading server inside the core. In MOCA architecture, although modifications in the current network were intentionally minimized, the MME (Mobility Management Entity) still played a centralized role and need to be modified. Here, we raise the question of whether SDN could realize the offloading architecture proposed in MOCA without touching the components of the current cellular architecture.

The rest of the paper is presented as follow: section 2 provides a background information of 4G/LTE-EPC networks, section 3 describes our approach and the high-level system design, section 4 presents our implementation, evaluation and results are presented in section 5.

2 Background

In this section, we briefly introduce the cellular network infrastructure, especially LTE and EPC architecture and protocol. Figure 1 shows LTE and EPC architecture which is made up of two components: Radio Access Network(RAN) and the Core Network(CN).

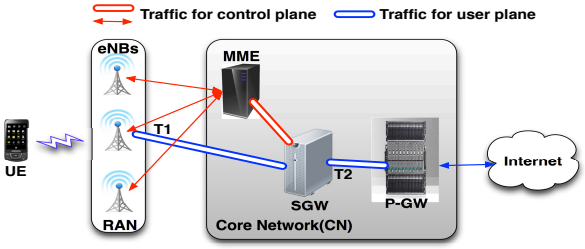

Figure 1: LTE/EPC Architecture

The RAN consists of eNodeBs (enhanced NodeBs) which are connected to User Equipment (UE) like cellphone through radio link and sends the packet to Serving Gateway(SGW). In addition, it manages radio resource and supports the intra-LTE mobility. The EPC is a framework which consists of Mobility Management Entity(MME), Serving Gateway(SGW), and Packek Data Network Gateway(PGW). The MME is a key component for performing UE registration, authentication, and mobility management. Also, it is responsible for the tracking and the paging of UE in idle-mode. It does not participate in packet forwarding. The SGW and the PGW are mainly responsible for routing/forwarding the data packets from all UE to the external network and vice versa. Besides packet forwarding, the SGW works as an anchor point in case of handover between eNodeBs and reserves buffers for paging functionality. The PGW performs the most functions among components in EPC such as IP address allocation to UE, charging, packet filtering, and supporting Quality of Service(QoS) according to the users data plan.

When the user attaches the cellular network, the components exchange many control messages, which are shown as red line in Figure 1 and called control plane. After finishing exchanging control message, the connection between UE and cellular network is established with two separate tunnels. After establishing the connection, the UE can send and receive the packet via tunnels which called user plane and shown as blue lines in Figure 1.

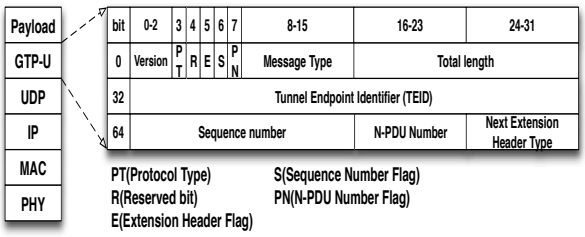

Figure 2: Protocol stack and GTP-U packet

The tunnel between two componets is represented by GPRS Tunneling Protocol(GTP). The cellular network uses GTP for data transfer as well as for supporting mobility, quality of service per user. The Figure 2 shows the protocol stack and GTP-U packet header. Each component in LTE and EPC architecture communicates with each other by using this protocol stack. For example, the enodeb encapsulates the packet received by UE before sending the packet to SGW and forwards encapsulated packet to SGW. After SGW receives the encapsulated packet from enodeb, it decapsulates, re-encapsulates and sends it to PGW. The encapsulation and decapsulation process is to replace the protocol layers depicted by gray boxes in Figure 2. However, the IP and payload layer above GTP-U layer does not change in cellular network and PGW finally decapsulates the packet and forwards it to external network. The packet from external network also work same way. Each tunnel can be distinguished by the unique Tunnel EndPoint Identifier(TEID) and when each component encapsulates the packet, TEID field is set by the destination TEID in GTP-U packet header shown in Figure 2.

3 Architecture

3.1 Overview

Our work adds to the current cellular network two main components: a SDN substrate, and a registration and control server (cloud manager in figure 3). The SDN substrate locates in between the Radio Access Network (RAN) and the cellular core network that intercepts and reroutes offloading traffic to the offloading server(s) inside the core. This substrate consists of Openflow switches (OFSes) that talk to eNodeBs via GTP tunnels and to offloading servers via regular IP connections. The substrate is controlled by controller(s) to realize traffic rerouting. The cloud manager maintains a database of registered offloading services and is responsible for allocating offloading servers inside the core. It also exposes to users an interface to register for offloading services and performs other functions such as charging and security.

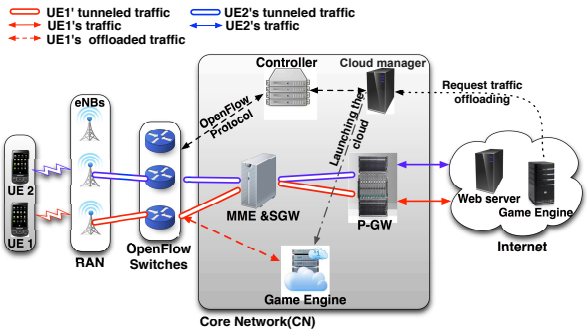

Figure 3: System architecture

A basic work-flow of the system is described as follow: users (e.g, game providers) contact the cloud manager to register offloading functionality for their services. The cloud manager allocates offloading server(s) and notifies the SDN controller(s) about the offloading server(s) and the registered traffic. The SDN controller then translates these information into SDN rules (e.i., match:actions) and installs these rules to the SDN substrate. Once these rules are deployed, the SDN substrate is able to reroute registered offloading traffic from/to RAN to/from appropriate offloading servers while leaving other unregistered traffic untouched.

3.2 Realization

3.2.1 Cloud manager

Cloud manager maintains information about offloading traffic and provides an interface that allows users

to register for offloading functionality. Users can specify a traffic that will enjoy offloading beforehand or during operation (similar to traffic balancing concept when a server becomes overwhelmed or when the network condition is not favorable causing an increasing of delay and therefore offloading is needed). Ideally, users can further specify a certain QoS associated with a traffic. For example, a game provider registers IP(s) of game engine(s) that it wants to do offloading due to the delay requirement of the game or a bad network condition. This game provider can also specify guaranteed bit rate and delay budget for the engine(s). The game provider interacts with the cloud manager via an interface (e.g, web front-end) to register/modify these services.

3.2.2 SDN substrate

SDN substrate is a set of Openflow switches (OFSes) controlled by controller(s) that realizes the offloading (as in figure 3) Traffic from RAN are intercepted by this substrate and applied offloading rules if needed. An OFS acts like an access switch (AS) that maintains GTP tunnels with eNodeBs and performs GTP encapsulation/de-capsulation and contacts its controller to query for offloading rules. The AS also performs packet classification to map and translate IP packets to GTP tunneling.

After going through an AS, packets are pure IPs and therefore could be routed using regular IP routing. Current Openflow switches do not support packet payload matching and therefore can not forward packets based on GTP tunneling information. Therefore, the pipeline of an OFS should be modified to be able to de-capsulate incoming GTP packets before handing them to the regular OFS's pipeline (IP packet matching and action).

3.2.3 Packet offloading

When attaches to the network for the first time, an UE follows a normal attaching process (e.g, authentication, bearers establishment, and IP address acquiring from the PGW). After resources are allocated, the UE can talk to an Internet server using the server's IP address. Traffic from UE, however, are not entirely routed to the Internet as usual but intercepted by the AS and offloading traffic are rerouted to the appropriate local offloading server. (dotted arrow in Figure 3).

An example of packet processing in an uplink direction could be described as follow: packets from an UE are sent by eNodeB to the AS in a GTP encapsulated form. These are UDP packets that have eNodeB's IP as the source IP address and SGW's IP as the destination IP address. At the beginning of a connection, AS forwards the first GTP packet to its controller for processing. The controller strips the outer GTP header of the original UDP packet to reveal the inner IP packet (the source and destination IP addresses of the inner IP packet are UE's IP and the remote server's IP respectively). It then compares the destination IP of this packet with its offloading database. A match means offloading service is applicable for this flow and therefore a flow entry will be installed into the AS to process future incoming packets. The flow entry in the AS replaces the source IP address of the incoming packets with the AS's IP and the destination IP address with the offloading server's IP. At the offloading server's perspective, it is communicating with the AS but not the eNodeB. Since the AS acts similarly to an network address translator (NAT), the controller has to store GTP's header information such as GTP message type and TEID (tunnel ID) for encapsulating returning packets in the downlink direction.

For downlink traffic, the AS encapsulates returning IP packets to create UDP packets with appropriate GTP headers to send to eNodeBs. Similarly, a flow entry is installed into the AS to reform the returning IP packets to UDP/GTP packets that match the original uplink tunnel (as described in the last paragraph). These actions include changing the source IP and destination IP addresses of the returning IP packets to SGW's IP and eNodeB's IP and placing correct GTP TEID (as what received in uplink direction). The mapping between returning packets and

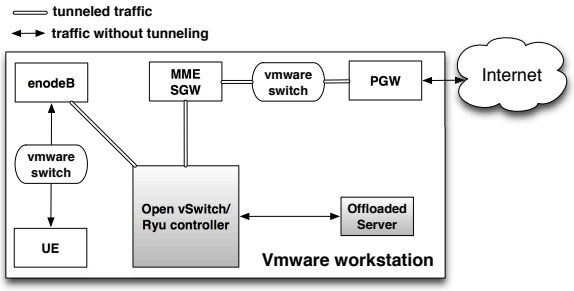

Figure 4: Openepc testbed

GTP TEID could be done using port number (AS could be seen as a NAT).

4 Implementation

4.1 Openepc

We used openepc testbed for mobile network. It includes all the components and a major part of the functionality of the EPC as well as supports a variety of radio access network (e.g. 2G, 3G, LTE, WiFi, and WiMax) and UE by an emulation. We deployed the openepc testbed in Emulab with vmware workstation and Figure 4 shows the deployed system. Each box is one of the vm instance and two vms which are depicted as gray boxes are added for our prototype. The lines with an arrow mean the connection without GTP tunnel and the lines without arrow mean connection with GTP tunnel.

4.2 Prototype Implementation

To implement prototype, we used Open vSwitch(OVS) version 2.0 and Ryu controller which support OpenFlow 1.3 standard. The OVS has the three ports which are connected to enodeB, MME&SGW, and Offloaded Server shown in Figure 4.

When the controller first runs, it sets up three matching rules in OVS tables. First matching rule is for forwarding GTP-U packet to a controller to a controller by using *OXM_OF_UDP_DST* field because all of the GTP-U packets have the same UDP destination port number(2152). Second matching rule is for forwarding the packets from a offloaded server by using *OXM_OF_TCP_SRC* filed. Whenever new offloaded servers are launched in core network(CN), the controller sets up the matching rules with their IP address. The last matching rules is to forward the rest of packets like control plane data which does not match first and second rules without contacting a controller. The rules is set up with *OXM_OF_IN_PORT* and should have lower priority than first and second rules.

However, current OVS does not support GTP encapsulation and decapsulation and OpenFlow standard does not define the GTP. So, we implemented the python library with *struct* [1] module in ryu controller for the encapsulation and decapsulation of GTP-U packets in a controller.

Whenever UE sends packets to a server, the first matching rule is matched and the packet is forwarded to a controller. The controller decapsulates the GTP-U packet and checks if the destination IP address is the same as the offloaded server or not. If so, it removes the MAC, IP, UDP, and GTP-U layers from a packet and then forwards it to a offloaded server. The controller keeps the mapping table which stores mac and IP address of enodeb and SGW for GTP-U encapsulation of returning packet with UEs IP address as key. Otherwise, the packet without a decapsulation is sent to the port which is connected to SGW.

When the offloaded server sends the packet to UE, the second rule is matched and the packet is forwarded to a controller. The controller checks the mapping tables with destination ip address from a packet and encapsulates it with stored data. Then, it sends the encapsulated packet to port which is connected to enodeb.

5 Evaluation

Using the settings in the implementation section, we measure end-to-end delay between an UE and different Internet servers. The delay inside the core network is the delay between VMs in a same physical machine. Although this delay is overly small (less

than 10ms) and unrealistic, we argue that delay gain shown in this evaluation section is the worst case scenario (in other words, with real networks, we would expect a higher gain compared to what showing here).

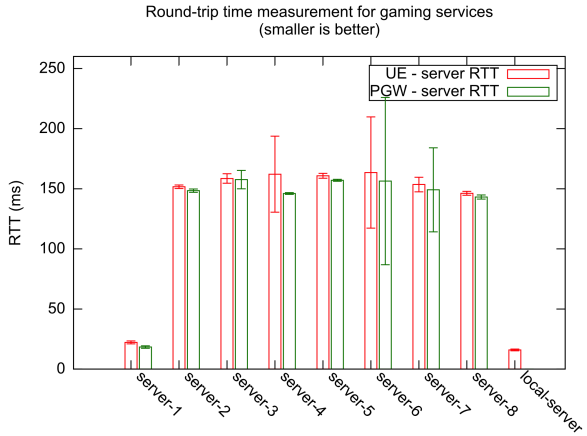

Figure 5: Game service RTT

We pick 5 most popular streaming services based on [8] and top 10 Minecraft [12] online game servers based on [7] to measure end-to-end round-trip times (RTTs) between an UE/PGW and the servers. A round-trip time is measured using 50 ICMP pings and the average delay is shown. In 5 streaming services, one does not allow ICMP ping. Two listed game servers are also dead.

Compared to contacting servers in the Internet, using the offloading server significantly reduces the RTT as in figure 5 and 6 (offloading RTT is about 18ms while accessing Internet servers would result in about 4 times higher delay for streaming services and about 9 times higher delay for gaming services). Green boxes show RTTs between the PGW and Internet servers (these are essentially Internet delays), the red boxes show the RTTs between the UE and the server (total end-to-end delay). Surprisingly, offloading does not defeat youtube.com. This could be

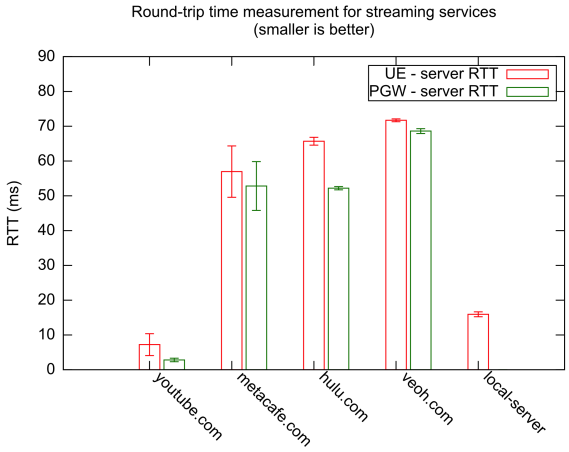

Figure 6: Stream services RTT

speculated as: (1) heavy processing at the controller (since we intercept *every* packet) adds extra delay to the path, (2) youtube.com might deploy CDN servers very closed to the PGW so that the Internet delay is small, (3) since the core delay is so small, this eliminates the gain of offloading approach over hierarchical routing issue and in the mean time this eliminates the delay that youtube.com should have in reality.

6 Related Work

Local IP Access (LIAP) and Selected IP Traffic Offload (SITOP) have been proposed in 3GPP standard for data offloading. While LIAP is more suitable for home environments, SITOP is proposed to route data out of the core network as soon as possible but not to the CDN servers inside the core. LIAP and SITOP also require specific hardware to be deploy while the goal of our work is not to touch the current components.

There are works [10, 11] that address scalability and inflexibility problems of PGW using SDN. These

works suggest the possibility of using SDN to enable flexible routing inside the core. MOCA relates to the works in the sense that it leverages SDN to share the pressure of data delivering on the PGW.

7 Conclusion

We leveraged SDN to introduce an offloading architecture for cellular networks. The architecture allows traffic to be intercepted and rerouted to offloading servers located inside the cellular core. This approach not only eliminates the hierarchical routing issue, but also removes the Internet delay and showes a promising low end-to-end delay. The SDN approach allows us to realize the offloading with minimum modifications on the current network architecture.

References

[1] . struct - Interpret strings as packed binary data. `docs.python.org/2/library/struct.html`.

[2] 3GPP. 3gpp ts 23.203 policy and charging control architecture. Tech. rep., 3GPP, 03 2012.

[3] Agarwal, S., and Lorch, J. R. Matchmaking for online games and other latency-sensitive p2p systems. In *ACM SIGCOMM Computer Communication Review* (2009), vol. 39, ACM, pp. 315–326.

[4] Balakrishnan, M., Mohomed, I., and Ramasubramanian, V. Where's that phone?: geolocating ip addresses on 3g networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 294–300.

[5] Banerjee, A., Chen, X., Erman, J., Gopalakrishnan, V., Lee, S., and Van Der Merwe, J. Moca: a lightweight mobile cloud offloading architecture. In *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture* (2013), ACM, pp. 11–16.

[6] Dong, W., Ge, Z., and Lee, S. 3g meets the internet: Understanding the performance of hierarchical routing in 3g networks. In *Proceedings of the 23rd International Teletraffic Congress* (2011), ITC '11, International Teletraffic Congress, pp. 15–22.

[7] http://www.serverpact.com. Top 10 gaming servers of our complete serverpact database, 11 2013.

[8] http://www.techsupportalert.com. 5 best free video streaming sites, 11 2013.

[9] Huang, J., Qian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., and Spatscheck, O. An in-depth study of lte: Effect of network protocol and application behavior on performance. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (2013), ACM, pp. 363–374.

[10] Jin, X., Li, L. E., Vanbever, L., and Rexford, J. Softcell: Scalable and flexible cellular core network architecture.

[11] Li, L. E., Mao, Z. M., and Rexford, J. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on* (2012), IEEE, pp. 7–12.

[12] Minecraft.net. Minecraft, 11 2013.

[13] Xu, Q., Huang, J., Wang, Z., Qian, F., Gerber, A., and Mao, Z. M. Cellular data network infrastructure characterization and implication on mobile content placement. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (2011), ACM, pp. 317–328.