การสตรีมวีดิทัศน์เป็นกลุ่มก้อนบนเอสดีเอ็นคลาวด์เพลย์กราวด์ OF@TEIN ซึ่งมีข่ายเชื่อมโยงแบบมีสายและแบบไร้สาย

นางสาวโฝว เม เต็ท

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2558
ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

CHUNKED VIDEO STREAMING OVER OF@TEIN SDN CLOUD PLAYGROUND WITH WIRED AND WIRELESS LINKS

Miss Phyo May Thet

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2015

Thesis Title	CHUNKED VIDEO STREAMING OVER OF@TEIN SDN CLOUD PLAYGROUND WITH WIRED AND WIRELESS LINKS
By	Miss Phyo May Thet
Field of Study	Electrical Engineering
Thesis Advisor	Associate Professor Chaodit Aswakul
Thesis Co-Advisor	Professor JongWon Kim

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

.....Dean of the Faculty of Engineering
(Associate Professor Supot Teachavorasinskun, Ph.D)

THESIS COMMITTEE

.....Chairman
(Assistant Professor Supavadee Aramvith, Ph.D)

.....Thesis Advisor
(Associate Professor Chaodit Aswakul, Ph.D)

.....Thesis Co-Advisor
(Professor JongWon Kim, Ph.D)

.....Examiner
(Assistant Professor Chaiyachet Saivichit, Ph.D)

.....External Examiner
(Associate Professor Poompat Saengudomlert, Ph.D)

โฝว เม เต็ท : การสตรีมวีดิทัศน์เป็นกลุ่มก้อนบนเอสดีเอ็นคลาวด์เพลย์กราวด์ OF@TEIN ซึ่งมีข่ายเชื่อมโยงแบบมีสายและแบบไร้สาย (CHUNKED VIDEO STREAMING OVER OF@TEIN SDN CLOUD PLAYGROUND WITH WIRED AND WIRELESS LINKS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ. ดร. เชาวน์ดิศ อัศวกุล, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม : ศ. ดร. จองวอน คิม, 141 หน้า.

วิทยานิพนธ์ฉบับนี้ได้ออกแบบและพัฒนามิดเดิลบอกซ์ และฟังก์ชันการแยกสำหรับการสตรีมกลุ่มก้อนวีดิทัศน์ประเภทอาร์ทีพี ตลอดจนกลไกการถ่ายโอนแฟ้มข้อมูลวีดิทัศน์ ผ่านพหุวิถีพร้อมกันบนเอสดีเอ็น (โครงข่ายกำหนดโดยซอฟท์แวร์) คลาวด์เพลย์กราวด์ OF@TEIN ซึ่งมีข่ายเชื่อมโยงแบบมีสายและแบบไร้สาย ระบบทดสอบต่าง ๆ ทั้งหมดได้ถูกสร้างขึ้นจริงบนโครงข่ายเฉพาะที่และโครงข่ายระดับนานาชาติใน 3 ประเทศ ได้แก่ เกาหลีใต้, มาเลเซีย และไทย ผลการทดลองครอบคลุมถึงการประเมินคุณภาพเชิงจิตวิสัยของวีดิทัศน์, อัตราส่วนการสูญเสียแพ็กเกต และค่าเวลาประวิงแพ็กเกต โดยสรุปวิทยานิพนธ์ฉบับนี้มีผลงานหลักสามส่วนดังต่อไปนี้

ส่วนที่หนึ่ง ระบบการสตรีมกลุ่มก้อนวีดิทัศน์ที่นำเสนอได้ถูกทดสอบในระบบทดสอบโครงข่ายเสมือนมินิ-เน็ตที่ติดตั้งโอเพนวีสวิตช์ และในระบบโครงข่ายจริงเอสดีเอ็นคลาวด์เพลย์กราวด์ OF@TIEN ซึ่งจะประกอบด้วยโอเพนสแต็ก, โอเพนวีสวิตช์และสมาร์ตเอ็กซ์บอกซ์ ผลลัพธ์การทดลองที่ได้แสดงให้เห็นว่าวิธีการสตรีมวีดิทัศน์ผ่านพหุวิถีสามารถมีประสิทธิผลเมื่อความจุของวิถีหลักเพียงวิถีเดียวไม่เพียงพอในการส่งแพ็กเกตของวีดิทัศน์ที่เข้ามาทั้งหมดได้ และอัตราส่วนการแบ่งกลุ่มก้อนของแพ็กเกตวีดิทัศน์สามารถกระจายอัตราแพ็กเกตที่เข้ามาให้ตรงกับความจุของวิถีต่าง ๆ ที่สามารถใช้งานได้ แต่อย่างไรก็ตามวิธีการสตรีมวีดิทัศน์ผ่านพหุวิถีนี้ไม่สมควรนำมาใช้งานเมื่อความจุของวิถีหลักนั้นเพียงพอสำหรับการส่งแพ็กเกตที่เข้ามาของสตรีมวีดิทัศน์แล้ว เหตุผลนั้นคือมิดเดิลบอกซ์ซึ่งต้องรวมแพ็กเกตที่เข้ามาจากวิถีต่าง ๆ ในเวลาจริงจะทำให้เกิดการประวิงเวลาแพ็กเกตเพิ่มขึ้นเนื่องจากข้อจำกัดของฮาร์ดแวร์ที่ใช้

ส่วนที่สอง เพื่อลดกระบวนการที่ต้องทำในเวลาจริงของมิดเดิลบอกซ์และเพื่อสามารถใช้ประโยชน์จากความจุของพหุวิถีได้ วิทยานิพนธ์นี้เสนอให้รวมฟังก์ชันการถ่ายโอนแฟ้มข้อมูลผ่านพหุวิถี และโพรโทคอลซูนามิ หลักการคือการส่งแฟ้มข้อมูลวีดิทัศน์แบบ 4k ขนาดใหญ่ล่วงหน้าจากตัวให้บริการส่งผ่านแฟ้มข้อมูลแบบซูนามิในประเทศเกาหลีใต้มายังสมาร์ทเอ็กซ์บอกซ์ของประเทศไทย โดยใช้การส่งไฟล์วีดิทัศน์เป็นกลุ่มก้อนผ่านวิถีต่าง ๆ และภายในสมาร์ทเอ็กซ์บอกซ์ของประเทศไทยมีตัวให้บริการวีดิทัศน์เฉพาะที่ทำหน้าที่สตรีมวีดิทัศน์แบบ 4k ไปยังเครื่องผู้รับบริการซึ่งอยู่ในไทยต่อไป ผลลัพธ์จากการทดสอบแสดงว่ากลไกที่นำเสนอสามารถใช้เวลาต่ำระดับ 23.51-72.20 วินาทีในการถ่ายโอนแฟ้มข้อมูลวีดิทัศน์แบบ 4k ความยาว 10 นาที และสามารถใช้ความจุของพหุวิถีได้อย่างมีประสิทธิผลในการให้บริการกับเครื่องผู้รับบริการสตรีมวิดีโอ

ส่วนสุดท้าย วิทยานิพนธ์นี้ได้สร้างระบบทดสอบเพื่อทดลองการสตรีมวีดิทัศน์เป็นกลุ่มก้อนผ่านวิถีซึ่งปรับตัวได้ในโครงข่ายมีสายระดับนานาชาติ OF@TEIN และโครงข่ายไร้สายระดับห้องปฏิบัติการ OF@Chula-EE การปรับตัวได้ของวิถีเป็นสิ่งจำเป็นเนื่องจากกระบวนการแฮนด์โอเวอร์ของผู้ใช้ไวไฟในประเทศไทยจากแอคเซส-พอยท์ไวไฟอันเก่าไปสู่อันใหม่ แอคเซสพอยท์เหล่านี้ได้รับการติดตั้งโอเพนวีสวิทช์บนพื้นฐานของโอเพนดับเบิลยูอาร์ที ซึ่งสื่อสารกับตัวควบคุมโอเพนโฟลว์ที่อยู่ห่างไกลออกไปในเกาหลีใต้ กระบวนการแฮนด์โอเวอร์ดังกล่าวถูกสาธิตในงานวิจัยนี้โดยใช้การวัดคุณภาพช่องสัญญาณไวไฟอย่างต่อเนื่อง, กลไกการส่งวีดิทัศน์เป็นกลุ่มก้อนล่วงหน้าซึ่งมีการซ้ำกันบางส่วนเพื่อลดการเสื่อมคุณภาพของวีดิทัศน์ที่รับได้ระหว่างการแฮนด์โอเวอร์ และกลไกการเปลี่ยนวิถีไร้สายซึ่งต้องทำงานได้ข้ามโดเมนบริหารจัดการระหว่างเกาหลีใต้และไทย การทดลองสาธิตนี้จะเป็นพื้นฐานสำหรับงานวิจัยคลาวด์สตรีมวีดิทัศน์ด้วยเอสดีเอ็นไร้สายต่อไปในอนาคต

ภาควิชา วิศวกรรมไฟฟ้า	**ลายมือชื่อนิสิต** .....
สาขาวิชา วิศวกรรมไฟฟ้า	**ลายมือชื่อ อ.ที่ปรึกษาหลัก** .....
ปีการศึกษา 2558	**ลายมือชื่อ อ.ที่ปรึกษาร่วม** .....

5770524021 : MAJOR ELECTRICAL ENGINEERING
KEYWORDS: OPENFLOW/ SOFTWARE-DEFINED NETWORK/ CHUNKED VIDEO STREAMING/ FILE TRANSFER/ MIDDLE-BOX/ WI-FI.

PHYO MAY THET : CHUNKED VIDEO STREAMING OVER OF@TEIN SDN CLOUD PLAYGROUND WITH WIRED AND WIRELESS LINKS .
ADVISOR: ASSOC. PROF. CHAODIT ASWAKUL, Ph.D., CO-ADVISOR: PROF. JONGWON KIM, Ph.D., 141 pp.

This thesis has designed and developed the middle-box and splitting functionalities for chunked RTP video streaming as well as video file pre-transferring mechanism over multiple concurrent paths over OF@TEIN SDN (software-defined network) cloud playground with wired and wireless links. All testbed scenarios have been constructed in actual over local and international networks in three countries: South Korea, Malaysia and Thailand. Experimental results include the evaluation of subjective video quality, packet loss ratio and packet delay. In summary, this thesis has three main contributions as follows.

Firstly, the proposed chunked video streaming system has been tested in both Open vSwitch-enabled Mininet-emulated network testbed and actual OF@TEIN SDN cloud playground testbed which includes OpenStack, Open vSwitch and SmartX box. The obtained experimental results show that the multi-path video streaming method is effective when the capacity of the main path alone is not enough to carry the whole incoming packets of video stream and the employed video packet chunk splitting ratio decomposes the incoming packet rate to match the capacity of available paths. However, this multi-path video streaming method should not be used when the main path capacity already suffices for transmitting the incoming packets of video stream. This is because the middle-box combining in real-time the arriving packets over multiple paths would incur additional packet delay due to its hardware limitation.

Secondly, to reduce real-time middle-box processing and to enable multi-path capacity usage, this thesis has proposed to combine the multi-path file transferring function and Tsunami protocol. The principle is in transferring beforehands the large 4k video file from the Tsunami file transfer server in South Korea to Thailand's SmartX box by using chunked video file transferring via multiple paths. And within the Thailand's SmartX box, a local video server is responsible for streaming out in real-time the 4k video to the client in Thailand. The test results show that the proposed mechanism can take as low as 23.51-72.20 seconds in transferring the 10-minute 4k video file and can utilize the multi-path capacity effectively in serving the video streaming client.

Finally, this thesis has constructed a testbed to experiment on the adaptive-path chunked video streaming over OF@TEIN international-scaled wired network and OF@Chula-EE laboratory-scaled wireless network. Adaptivity of utilized path is necessary due to the handover process of Wi-Fi user in Thailand from the old to new Wi-Fi access points. These access points have been installed the OpenWRT-based Open vSwitch, which communicates with the remote OpenFlow controller in South Korea. Such handover process has been demonstrated in this research with the continuous measurement of Wi-Fi channel quality, a partially redundant chunked video pre-transferring mechanism to reduce the received video quality degradation during the handover and a wireless-path changing mechanism that must work across administrative domains between South Korea and Thailand. This demonstrating experiment will serve as the basis for future wireless SDN video-streaming cloud research.

Department : Electrical Engineering	**Student's Signature** .....
Field of Study : Electrical Engineering	**Advisor's Signature** .....
Academic Year : 2015	**Co-advisor's Signature** .....

Acknowledgements

First of all, I owe my debt of gratitude to my advisor, Assoc. Prof. Dr. Chaodit Aswakul, for giving me a chance to be one of his advisees and for his great and valuable guidance, caring and patience on me thus far starting from Chula scholarship application process to the finalization of this thesis. I cannot think of my study life at Chula without his supports and guidance. All the questions and advises raised by him during our research discussion make me to improve my logical thinking skills and problem solving skills. I would like to thank to my co-advisor, Prof. JongWon Kim from Gwangju Institute of Science and Technology, Gwangju, South Korea for his valuable suggestions on my research experiments and papers writing.

I would also like to express my gratitude to my thesis exam committee members, Asst. Prof. Dr. Supavadee Aramvith, Asst. Prof. Dr. Chaiyachet Saivichit, Assoc. Prof. Dr. Poompat Saengudomlert, Assoc. Prof. Dr. Chaodit Aswakul and Prof. Dr. JongWon Kim, who provide valuable suggestions for my thesis. I would also like to thank to Asst. Prof. Dr. Chaiyachet Saivichit who provides suggestions on my thesis during our LaLa meetings and NRG conference.

In addition, I would like to express my special gratitude to Graduate School of Chulalongkorn University for granting International Graduate Students in ASEAN Countries Scholarship to study this master degree program. This research has been financially supported by the Special Task Force for Activating Research (STAR) Funding in Wireless Network and Future Internet Research Group, Chulalongkorn University. Special thanks to Prof. JongWon Kim and Prof. Chaodit Aswakul for giving me a chance to participate in Summer Global Internship Program for one and half months at Gwangju Institute of Science and Technology, South Korea.

I appreciate the help of my group members from Wireless Network and Future Internet Research Group (WiFUN) and OpenFlow Chula during my study life at Chula. I also would like to thank to Aris and Usman (GIST OF@TEIN Administrators), Chula IT, Uninet IT, TEIN NOC and KOREN NOC teams for their great support during my experiments.

Finally, I would like to express my deepest gratitude to my parents, brothers and sister, who always support and encourage me with their best wishes.

Contents

List of Tables

Page

List of figures

Page

Chapter 1

Introduction

1.1 Research Motivation

In today's information and communication technology (ICT) environment, the dynamic innovations and controllability of underlying network are required towards Future internet infrastructure in order to meet the variety of requirements of users. From the viewpoint of user needs for emerging ICT society, the wireless communication environment is essential. Mobile operators, vendors and internet service providers need to consider how to conquer the high volumes of traffic and to support increasingly sophisticated services such as providing subscribers to be able to access online video streaming, file transferring and any other internet applications. Moreover, the surge demand of mobile users becomes a major challenge for network providers. Many mobile and network operators are planning to overcome their mobile traffic congestion by using Wi-Fi offloading technique. When Wi-Fi offloading takes place in a mobile network, the advanced management controllability of Wi-Fi network will become essential. However, the current Wi-Fi management technology has the limited control of seamless handover and maintaining the multiple networks. As a result, the network administrators resort to manual, time-consuming tasks due to lacking the full controllability of their large network systems. These situations force the network administrator to be a human middleware.

In order to overcome these networking issues, software-defined networking (SDN) paradigm has brought flexible controllability and sufficient programmability to the network operators by separating the control and data planes with an open and standardized interface [1]. In this regard, OpenFlow interface is the first and has become one of the popular protocols widely accepted. The OpenFlow standard enables the direct communication between SDN controllers and networking devices so that network managements become easier than the traditional network managements with variety of proprietary issues [2]. SDN can lead various promising benefits to network operators, vendors and users such as centralized network management, lower capital expenses, ability to control of multi-vendor networking devices, increasing rate of reliability and security of

networks via programmability with open interfaces [1]. When taking into account for network congestion and security, SDN enables middle-boxes play an important role in the networking environment for addressing network allocation and security management.

SDN comes up with network function virtualization and network orchestration which are very popular technologies for implementing future internet infrastructures. Since SDN becomes a new paradigm for leveraging the future internet technology, several SDN-based testbeds have been implemented in both large-scaled and small-scaled networks. For example, Open-Access Research Testbed for Next-Generation Wireless Networks testbed (ORBIT) is a two-tier laboratory emulator network testbed for investigating research in cognitive radio networks, ad hoc network routing and delay tolerant networks and wireless security [3]. Network Implementation Testbed using Open Source (NTOS) is another wireless SDN-based research testbed for implementing the wireless network and studying the network performance in real environments [4]. Moreover, various SDN testbeds have been deployed under the SmartFire project [5] in Europe and Korea. Among several SDN-based testbeds, we select OpenFlow at Trans-Eurasia Information Network (OF@TEIN) SDN-based virtual cloud playground [6, 7] which is currently connected to 11 sites in 9 countries (Korea, Indonesia, Malaysia, Thailand, Vietnam, Philippines, Pakistan, India and Taiwan) in order to investigate the functionalities of network virtualization and network orchestration in a large-scaled network.

Moreover, the vast majority of end-user demand for application services e.g. video streaming, file transferring, voice over internet protocol (VoIP) calls and online gaming are originating on subscriber devices connected to the network via Wi-Fi and wired networks. The growth demands of Wi-Fi users drive the software-defined wireless networking (SDWN) to become a potential new technology for future wireless network. Although many SDN researches have focused on wired and core networks, SDN over wireless network remains largely unexplored and necessary solutions are required before SDN becomes ready for the wireless platform. The management of wireless networks is more complex than that of wired networks. By leveraging the SDN into wireless network, troubleshooting, traffic optimization and capacity planning can be done easily thanks to the end-to-end control of wired and wireless networks.

In order to overcome the recent requirements of video streaming users, we will evaluate the functionalities of SDN on both real wired/wireless network and OpenFlow emulated network with the investigation scope of SDN functions which include *chunked video streaming, chunked video file transferring* with splitting and middle-box functionalities and *chunked video streaming over OF@TEIN wired and wireless links with chunked video pre-transferring mechanism.*

1.2 Problem Statement

The dynamically increasing rates of video streaming users become the major concern for network service providers. Video streaming is nowadays responsible for the major consumption and traffic congestion on the internet network. To overcome these bandwidth limitations, delay time and packet loss problems in the network, many researchers are trying to investigate the solutions by using several approaches. Video buffering for achieving high performance video becomes an essential technique during network congestion.

In addition, many subscribers are using wired and Wi-Fi networks for video streaming, file transferring, video conferencing and VoIP calls. All these applications can create the traffic congestion and need to have the simultaneous connectivity of internet network. The more video streaming users require seamless network connectivity, the more advanced techniques are needed for managing Wi-Fi networks. Many researchers have proposed several approaches to solve the problem of network congestion due to video streaming over wired and wireless network. To overcome the challenges of internet such as bandwidth constraints, delays and packet losses, the mechanisms of static and dynamic multi-path TCP streaming over homogeneous and heterogeneous paths have been proposed in multi-path live TCP video streaming [8]. Single-path TCP and UDP video streaming over emulated and real OpenFlow PC cluster networks have been demonstrated in [9] in order to investigate the packet delays and performance of video streaming. In addition, the authors of [9] extended their experiments with multi-path chunked video streaming over locally OpenFlow enabled PC-cluster testbed to find the effect of chunked size and buffering on video performance [10]. As for video streaming over wireless network, SDN

based Wi-Fi handover management with real-time video streaming over wireless local area network (WLAN) has been proposed in [11]. However, there are no researches that investigate chunked video streaming over SDN-cloud network in the international gateway environment together with wired and wireless links. To meet various requirements of users, the so-called middle-box becomes a new SDN enabler for addressing e.g. security management and resource allocation function in the network [12]. Especially, requirements need to be considered thoroughly when the video streaming experiments are conducted over international SDN testbed such as OF@TEIN testbed [13]. In that kind of international testbed spanning different time zones, the network congestion is geospatially time varying and international link bandwidths are limited by each nation.

When considering the experiments over international collaborative research testbeds such as OF@TEIN SDN cloud playground, fast file transferring is important since not all data are located in one site. In order to support high speed data transfer, UDP-based data transfer protocol (UDT) for high-speed wide area networks and Tsunami file transfer protocol via UDP and TCP has been implemented in [14] and [15]. As for file transferring over TCP with SDN technology, a multi-path controller for GridFTP transfer over SDN has been introduced in [16] by testing in both virtual and real global-scale networks. However, file transferring over TCP can lead to congested networks and delay due to retransmission and multiple acknowledgement requests. This challenge has led us to conduct research on UDP file transferring with SDN multi-path technologies over an international large-scaled network.

The above mentioned research challenges become our motivation for studying on buffering effect in the middle-box via multi-path video streaming to help aggregate for larger path bandwidths than those achievable by relying only on a single restrictive-bandwidth path. Moreover, the requirements of file transferring in the international collaborative research testbeds lead us to investigate SDN multi-path functions over a large-scaled international testbed. In addtion, the continuing growth in video streaming demand from subscribers over stochastic Wi-Fi network and restrictive-bandwidth links has encouraged us to look ahead at how our SDN-based chunked streaming scenarios can be made ready to meet future extreme capacity and

performance demands. Our following proposed approaches of chunked video streaming over restricted emulated OpenFlow network and chunked video file transferring and streaming over international OF@TEIN SDN cloud playground with real wired and wireless links at OpenFlow at Chulalongkorn Electrical Engineering (OF@Chula-EE) would facilitate us to investigate more about the effects of real-time chunked video streaming over stochastic bandwidth links.

1.3 Objective

The objective of this thesis is to design, implement and evaluate three chunked video streaming scenarios over OpenFlow-enabled emulated testbed, real international OF@TEIN cloud testbed and OF@Chula-EE lab-scaled wireless testbed with wired and wireless links with details as follows:

1. Multi-path chunked video streaming over emulated OpenFlow network and international OF@TEIN SDN cloud playground with middle-box and splitting functionalities.

2. Multi-path chunked video file transferring and streaming over international OF@TEIN SDN cloud playground.

3. Adaptive-path chunked video streaming over OF@TEIN international wired and wireless links with chunked video pre-transferring mechanism.

We have evaluated the performance of chunked video streaming in terms of subjective video quality, jitter, delay time, packet loss ratio and buffering effects for each scenario.

Five international nodes which have been used in this thesis's video streaming experiments over OF@TEIN testbed are as follows:

1. CHULA Node at Chulalongkorn University (CU), Thailand.

2. MYREN Node at Malaysian Research and Education Network (MYREN), Malaysia.

3. MY Node at University of Malaya, Malaysia.

4. GIST-A Node at Gwangju Institute of Science and Technologies (GIST), South Korea.

5. GIST-B Node at Gwangju Institute of Science and Technologies (GIST), South Korea.

1.4 Scope of Thesis

This research focuses on the measurements of jitter performance, packet delay and video quality for three chunked video streaming scenarios over emulated OpenFlow network and international OF@TEIN SDN cloud playground with wired and wireless links. The following mechanisms have been included in this research:

1. Designing and implementing the middle-box and splitting functionalities for chunked video streaming over multiple concurrent paths and constructing a prototype system to evaluate these functionalities with various scenarios of splitting ratio and initial buffering time over emulated OpenFlow network and international OF@TEIN SDN cloud playground by using Mininet emulator, OpenStack, SmartX Boxes and POX controller.

2. Designing and implementing multi-path chunked video file transferring and streaming with splitting and middle-box functionalities over international OF@TEIN SDN cloud playground (connected with three countries: Thailand, Malaysia and Korea) by using OpenStack, SmartX Boxes, POX controller.

3. Designing and implementing adaptive-path chunked video streaming over OF@TEIN international wired and wireless links to stream chunked video between two wireless access points (APs) with chunked video pre-transferring mechanism.

   For the above mentioned scenarios, we have evaluated the received video quality based on transmission parameters (at network layer) such as packet loss ratio, packet delay or jitter with various parameters (chunk size ratio, initial buffering time and video file transfer rate). The evaluated results in this thesis focus on the subjective video quality, i.e., no reports on video quality such as peak signal-to-noise ratio (PSNR) are included.

1.5 Expected Outcome and Contribution

After completing this research, we expect to achieve following benefits:

1. We can design and implement the multi-path chunked video streaming over both emulated OpenFlow network and actual experiments over the international OF@TEIN SDN cloud playground and multi-path video file transferring and streaming over international OF@TEIN SDN cloud playground with splitting and middle-box functionalities.

2. We can implement the adaptive-path chunked video streaming over OF@TEIN international wired and wireless links to stream chunked video between two wireless access points (APs) with chunked video pre-transferring mechanism.

3. We can introduce the solution to address bandwidth limitation issues by using load balancing, splitting functionalities of SDN and solution for reducing the cost of resource consumption for wireless network.

4. We can evaluate the video transmission performance in terms of packet loss ratio, jitter, buffering time effects for chunked video streaming over OF@TEIN SDN cloud playground with wired and wireless links.

1.6 Organization of Thesis

The arrangements of chapters in this thesis are as follows. Chapter 2 describes the background of software-defined networking, concepts of OpenFlow and literature review about video streaming over wired and wireless SDN, as well as file transferring over SDN. Chapter 3 expresses the implementations and evaluations of three experiment methods as described in the objective. The first experiment method in Chapter 3 explains and discusses about the implementations and evaluations of multi-path chunked video streaming over emulated network and OF@TEIN SDN cloud playground. The second method discusses about multi-path video file transferring and streaming experiments over OF@TEIN SDN cloud playground. The last method presents the implementations and evaluations of adaptive-path chunked video streaming over OF@TEIN SDN cloud

playground with wired and wireless links. The conclusions of the whole research and future suggestive research direction are described in Chapter 4.

Chapter 2

Background and Literature Review

2.1 Background

2.1.1 Software-Defined Networking

As networking plays a key role in emerging ICT society for the purpose of doing business, education, and others, the innovation and controllability of underlying networks have become crucial for Future Internet infrastructure. To help people easily access internet from their mobile devices, mobile operators, vendors and Internet service providers consider how to conquer the high volumes of traffic and to support increasingly sophisticated services. Moreover, the traditional network architecture has a limited controllability for large scaled network. In order to overcome these issues, software-defined networking (SDN) paradigm has brought flexible controllability and sufficient programmability to the network operators by separating the control and data planes with an open and standardized interface. In this regard, OpenFlow interface is the first and has become one popular protocol widely accepted. The OpenFlow standard enables the direct communication between SDN controllers and networking devices so that network managements become easier than the traditional network managements with variety of proprietary issues [2]. In originally proposed SDN architecture, the control plane is usually implemented as a centralized control plane while in traditional network architecture is implemented with distributed control planes. Figure 2.1 depicts the comparison between traditional network architecture and software-defined network architecture.

SDN consists of three layers: application layer, control layer and infrastructure layer. The architecture of SDN is shown in Figure 2.2.

2.1.2 OpenFlow

OpenFlow is a standard open protocol for direct communicating between SDN controllers and networking devices. There are many functions in each OpenFlow version.

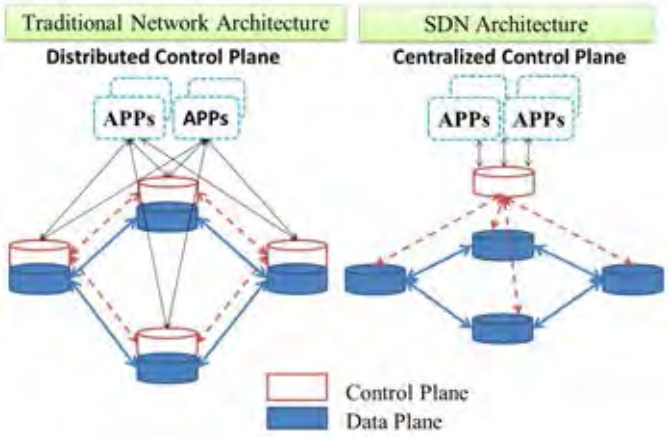


Figure 2.1: Comparison of traditional network and SDN architectures [18].

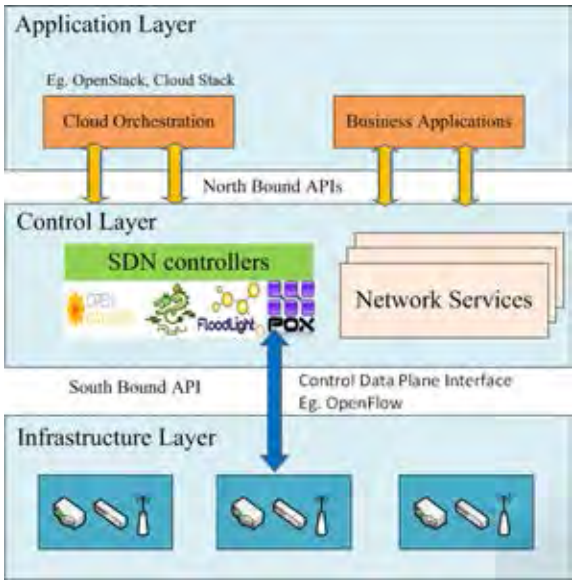


Figure 2.2: Architecture of software-defined networking [1].

In this thesis, we have been used OpenFlow 1.0 version for testing our experiments. The function of OpenFlow 1.0 [17] is shown in Figure 2.3.

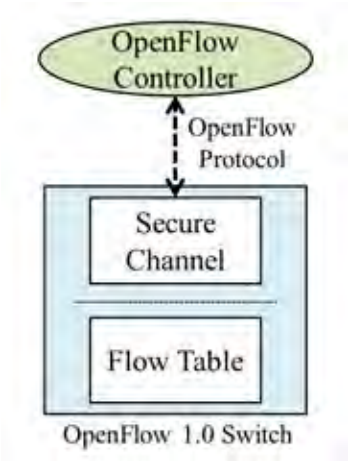


Figure 2.3: Components of OpenFlow 1.0 switch [17].

OpenFlow 1.0 switch consists of a single flow table for performing packet lookups, forwarding and communicating to OpenFlow controller via secure channel [17]. OpenFlow controller is responsible for matching openflow rules and adding/removing flow entries from OpenFlow switches. Due to various functions of OpenFlow rules, we can design new network architectures by taking benefits of OpenFlow rules such as load balancing and dynamic routing.

2.2 Literature Review

There are potential benefits which SDN can bring to network operators, software vendors and users such as virtualized network management, ability to control of multivendor networking devices, increasing rate of reliability and security of networks via programmability with open interfaces [1]. In the literature, several researchers have investigated how to renovate the traditional networks with the functions of SDN programmability and controllability. In order to conduct these researches, Mininet [19] is a widely popular experimental platform. Mininet, as a network emulator, can create virtual OpenFlow networks within a single computer, allowing rapid prototyping and testing. For example, for a video conferencing experiment, OpenSession [20] has proposed a new management protocol for multi-stream 3D tele-immersion by using OpenFlow-based

architecture. The Rent-a-Super Peer (RASP) architecture is designed for peer-to-peer streaming over OpenFlow networks [21]. CastFlow [22] has proposed an IPTV multicast prototype by emulating over a Mininet testbed. In addition to the emulated testbeds, SDN is commercially deployed at Googles large-scale data centers [23].

Amongst various SDN-enabled applications, this thesis focuses on chunked video streaming and chunked video file transferring over multiple concurrent paths. The motivation is inherited from the majority of Internet traffics becoming video streams and file transfers nowadays. Existing legacy traffic engineering mechanisms need refurbished by the SDN paradigm to mitigate potential congestions of such bandwidth-hungry workloads that can diminish the performance of video reception at clients and can increase the transmission delay. In this scope, an earlier attempt is to verify the usages of a Mininet-emulated video streaming testbed with the simplest scenario comprising a pair of video server and client [9]. However, the tests conducted so far in [9] are limited to the video streaming over a single path, which therefore can run out of bandwidth easily. And the usage of SDN for video streaming over a single path and multi-path local OpenFlow testbed has been demonstrated in an earlier research trial [10]. However, in [10], no investigations have been carried out video streaming via stochastic international links where available bandwidth capacity is time varying and limited by each nation. Such concern will be amplified when the tests are extended towards the video streaming scenarios over OF@TEIN international testbed [13], where available paths must traverse internal and expensive links with limited and shared capacity. In order to deal with issues on bandwidth hunger, stringent delay time and packet loss requirements, multi-path live video streaming via TCP has been investigated in [8].

For solving the requirements of video streaming users upon network congestion, the so-called middle-box becomes a new SDN enabler for addressing e.g. security management and resource allocation function in the network [12]. Especially, these requirements need to be considered thoroughly when the tests are going to be extended to international SDN testbed such as OF@TEIN testbed [13]. In that kind of international testbed spanning different time zones, the network congestion is geospatially time varying and international link bandwidths are limited by each nation. Setting a proper initial buffering time of the

middle-box is essential for streaming out packets towards the client steadily to facilitate the quality of video service during the playback time at the receiving client. To improve the smoothness of video playback, the varying of delay and jitter need to be considered [24]. These limitations become our motivation for studying on buffering effects in the middle-box via multi-path video streaming to help aggregate higher path bandwidths than those achievable by relying only on a single restrictive-bandwidth path.

File transferring over international networking environments is also becoming important issue since recent networking approaches are interested to run over international collaborative network testbeds. In that kind of situation, experiments data are not located only in one country node instead located on different country nodes. Network administrators and users are facing long duration of data transferring. In order to overcome file transferring issues, UDP-based data transfer protocol (UDT) for high-speed wide area networks and Tsunami file transfer protocol via UDP and TCP have been implemented in [14] and [15]. As for file transferring over TCP with SDN technology, a multi-path controller for GridFTP transfer over SDN has been introduced in [16] by testing in both virtual and real global-scale networks. However, file transferring over TCP can lead congested network conditions and can occur transmission delay due to retransmission and multiple acknowledgement requests. Moreover, although multi-path TCP file transferring method over large-scaled network has been introduced in [16], no investigations have been made for how much transmission delay can decrease due to their proposed multi-path TCP file transfer over international large-scaled network. This challenge has motivated us to conduct research on UDP file transferring with SDN multi-path technologies over an international large-scaled network.

In addition to the requirements of network management and network congestion over wired networks, this thesis also focuses on the requirements of network management and network congestion over wireless networks. In our today's society, the number of mobile subscribers who connect to internet network is tremendously increasing. Especially, the surge traffics in wireless networks come from popular applications such as video streaming, file transferring, video conferencing and VoIP calls. The growth of mobile traffic leads the mobile and network operators to renovate their current architecture with flexible

controllability and programmability. Moreover, the increasing deployment rate of unplanned Wi-Fi network with 2.4 GHz (IEEE 802.11b/g/n) unlicensed spectrum is also a major challenge for internet service providers since the non-overlapping channel of 2.4 GHz band is limited with three channels (Channels: 1, 6 and 11). The radio interference rate is also increasing in 2.4 GHz band Wi-Fi network due to its commonly shared frequency usage with daily equipments such as microwave ovens and Bluetooth sensors. In addition, channel allocation, large scale network monitoring, control and policy management and seamless handover management of Wi-Fi network are also needed. The limited controllability of access points and routers in today's wireless local area network (WLAN) environment has motivated researchers to investigate how to improve over the traditional wireless infrastructure.

OpenFlow-based wireless software-defined network [25] has introduced the flexible and cost-effective communication platform for addressing the above mentioned limitation in today's WLAN networks. There are many benefits that wireless software-defined networking can bring to the mobile and wireless network such as resource allocation and optimization, multi-network planning, security, open control management, interference and traffic management [26]. Since mobile offloading has become an important mechanism for mitigating network congestion, SDWN becomes a major part for innovating future wireless network infrastructure. Slicing and channel isolation, monitoring and status report, handoffs are major challenges when leveraging software-defined networking into wireless networks. However, load balancing, handover, security and access control management are still in the stage of most challenging technology for implementing the future SDWN infrastructures. Telecommunication and networking companies such as Ericsson, Nokia, Cisco, Meru, Anyfi networks and others are trying to introduce the benefits of using SDN technology [27, 28, 29, 30, 31].

Researchers have proposed approaches for video streaming and handover management over wireless software-defined networking. For example, OpenRoads testbed [32] has been successfully deployed for wireless network with demonstration of n-casting using OpenFlow [33]. However, in contrast to our study, the tests conducted in n-casting [33] streams out the video locally and explored the handoffs mechanism between Wi-Fi and

WiMAX. Our study focuses on cloud video streaming over international link to stream out the adaptive-path chunked video streaming with chunked video pre-transferring mechanism during the handover process between wireless access points. Moreover, the usages of OpenFlow for wireless mesh network have been demonstrated in [34]. Meru network's SDN for Wi-Fi [30] is also the example of commercial SDN deployment in wireless networks. The SDN in Wi-Fi handover management has been investigated in CloudMAC [35] with network initiated handovers. Real-time video streaming over WLAN for handover management with SDN has been introduced in [11] by evaluating the performance of handover mechanism with video freeze events and play out buffer rate. Light Virtual Access points (LVAPs) for addressing load-balancing, mobility management, automatic channel detection and guest policy enforcement has been investigated in Odin [36]. OpenSDWN [37] has also introduced the per-client virtual access points and per-client virtual middle-boxes for mobility and seamless migration. Flow-based mobility management has been investigated in [38] by modifying the mobile nodes to be controlled with SDN flow-based mechanisms.

Although many researchers have proposed video streaming over wired and wireless networks, no investigations have been conducted on chunked video streaming over a virtual cloud network with a combination of international wired and wireless links. The projects described in [39] also become our motivation to investigate the performance of adaptive-path chunked video streaming over International OF@TEIN testbed [13] and OF@Chula-EE lab-scaled software-defined wireless network. OF@TEIN is the SDN-based virtual cloud playground which has been launched in July 2012. Currently, OF@TEIN playground is connected to 11 sites in 9 countries (Korea, Indonesia , Malaysia, Thailand, Vietnam, Philippines, Pakistan, India and Taiwan) over TEIN4 [6, 7]. The architecture of OF@TEIN is depicted in Figure 2.4.

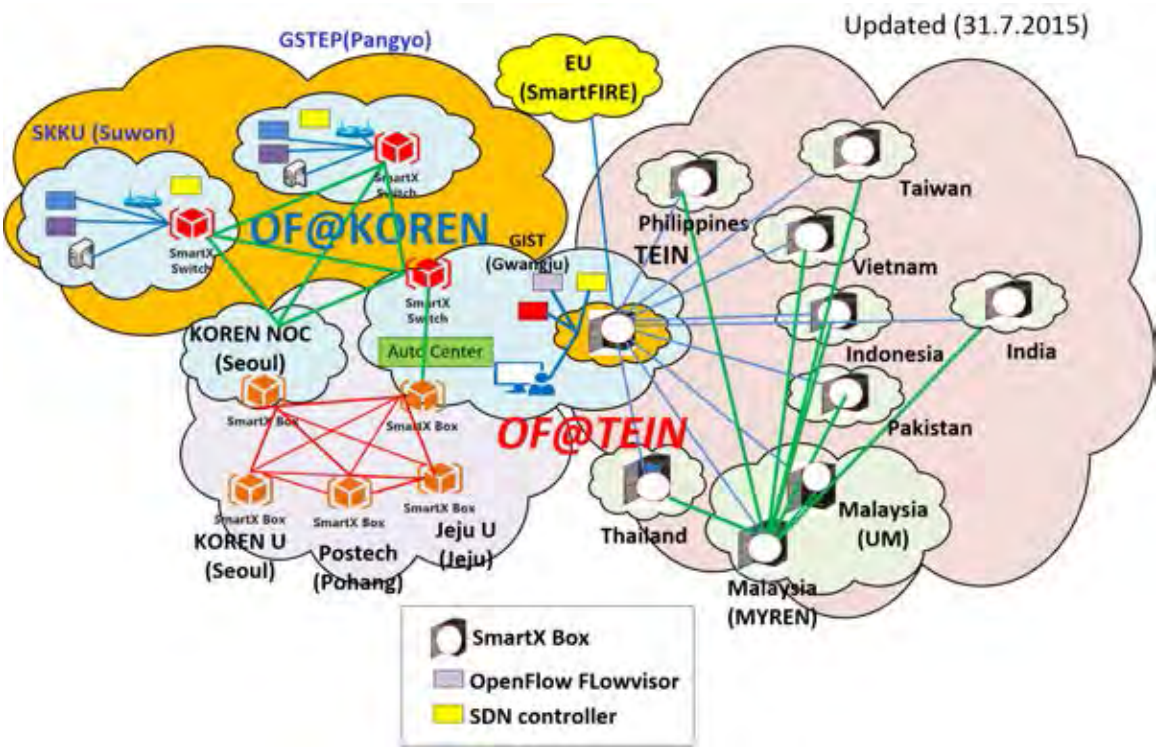


Figure 2.4: OF@TEIN infrastructure [6, 7].

Chapter 3

Research Methodology

As we described in our objectives and scope of work, the goal of this thesis is to investigate the performance of chunked video streaming over emulated OpenFlow and OF@TEIN SDN cloud playground. Moreover, we will investigate chunked video file transferring and streaming over OF@TEIN SDN cloud playground and chunked video streaming over OF@TEIN SDN cloud playground with wired and wireless links where path capacities are possibly uncontrollably time-varying. Since the video streaming over stochastic international wired and wireless links is one of the challenging approaches, the plan is to implement chunked video streaming mechanisms on emulated OpenFlow testbed and then on the actual international OF@TEIN testbed and OF@Chula-EE lab-scaled testbed with wired and wireless links. The procedures of our research methodology are as follows:

1. Implement middle-box and splitting functionalities for chunked video streaming over multi-path emulated OpenFlow network and real OF@TEIN SDN cloud playground (GIST-A, MYREN, CHULA) and evaluate these functionalities with various scenarios of splitting ratio [40] and video buffering effects [41].

2. Implement multi-path chunked video file transferring and streaming with splitting and middle-box functionalities over OF@TEIN SDN cloud playground (connected with three SmartX sites located in Chulalongkorn University (CU), Thailand, University of Malaya, Malaysia and Gwangju Institute of Science and Technology (GIST), South Korea).

3. Implement adaptive-path chunked video streaming over OF@TEIN international wired and wireless links with chunked video pre-transferring mechanism.

3.1 Implementation of Emulated Multi-path Video Streaming and SDN-Based Middle-box Functionality [40, 41]

Before we implement our multi-path chunked video streaming over international OF@TEIN testbed, we have designed and tested necessary SDN-based middle-box functionalities to enable multi-path chunked video streaming over emulated OpenFlow network. Two experiments namely experiment 1 and experiment 2 have carried out with two different purposes. The purpose of experiment 1 is investigating the effects of chunk size ratio, i.e., the proportion of time the originating video packet stream is switched alternately on two parallel paths. For the experiment 2, the purpose is to investigate the effects of initial buffering in the middle-box, i.e., the time period of storing first initial packets of the video stream in the middle-box before they are streamed out towards the client. These two experiments use automated video chunk splitting, multi-path forwarding, and load-combining for the same video streaming session in a transparent manner to both video server and client applications. In the following, subsection 3.1.1 presents the design of SDN-based middle-box functionalities: splitter at an ingress Open vSwitch (OVS) and load-combining at an egress Open vSwitch. Subsection 3.1.2 specifies the OpenFlow configuration for packet forwarding on multiple paths in the Mininet-emulated framework. Experimental scenarios are detailed in Subsection 3.1.3 and a summary is in Subsection 3.1.5.

3.1.1 Design of Emulated SDN-Based Middle-box Functionalities

As a consequence of allowing multiple paths to serve concurrently the same video streaming session, there are additional modification necessities.

Firstly, load balancing mechanism must be implemented into the SDN testbed. In that regard, this thesis has assigned a traffic ingress Open vSwitch on the video streaming server side to function like a video chunk splitter. Our design choice is based on a periodic time-based splitting whose chunk size ratios can be controlled by a remote OpenFlow controller program developed under the POX [43] framework. Decision on the proper location and

chunk size ratio for the splitter affects the network congestion profiles and is a contemporary traffic engineering problem.

Secondly, one must make sure that the conventional server and client applications should be normally operable without additionally required changes in their codes. That is, the multi-path streaming at the intermediary OpenFlow-enabled network should be programmatically transparent to the client-server pair. At the egress of Open vSwitch receiving video streaming packets from all the ongoing multiple paths, a middle-box has been introduced. Due to the requirements of security, policies, and load balancing of today's enterprise networks, such middle-boxes (e.g., firewalls and gateways) which can support for these functions have become an extra feature of SDN [12]. In this research, the middle-box performs as a packet scheduler and classifier with two parallel buffers (buffer 1 and buffer 2). The middle-box is designed to collect all incoming packets from whatever path, reorder them according to the correct video packet sequence integrity, then pushes the sorted video packet stream out towards the client as smoothly as possible.

Thirdly, unlike [9] with the employed transports of TCP vs UDP, in this research, RTP has instead been used. RTP time stamp field, identifying the timing of packet generation at the server, can then be used for packet order identification in the middle-box. Figure 3.1 depicts a topology of OpenFlow network exemplifying both the splitter functionality at Open vSwitch 1 and the middle-box functionality at Open vSwitch 3 which is used for both experiments. Two paths are available to connect the server with the client, namely, the upper path via Open vSwitch 2 and the lower path via Open vSwitch 4.

3.1.2 Mininet-Emulated Testbed Implementation

For testing video streaming experiment 1 and 2 in the emulated OpenFlow multi-path network, the following implementation details have been carried out. Mininet version 2.1.0+ has been installed in the personal computer with Intel Core i5-375S CPU @ 2.9GHz x 4 running the Ubuntu OS 12.04 LTS for experiment 1 and in Ubuntu 12.04 LTS with Intel Core 2 Duo 2.8 GHz x 2 personal for experiment 2. This Mininet version can support the middle-box module integration. This research has chosen to continue using the network topology settings as earlier used in [9] and [42]. The network interface architecture in the

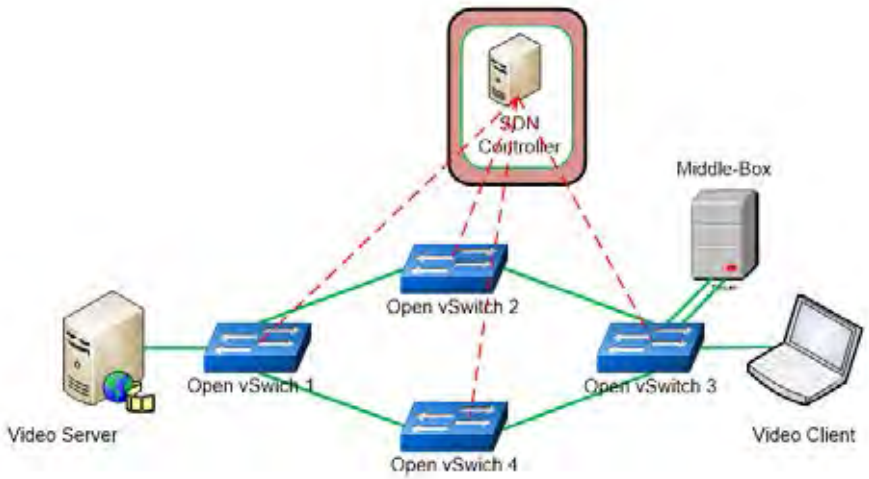


Figure 3.1: Multi-path video streaming over OpenFlow-enabled network [10].

Mininet emulation tool for both experiments is shown in Figure 3.2. Two emulated hosts h1 and h2 serve as the video server and video client. Four Open vSwitches (OVS1, OVS2, OVS3, OVS4) have been instantiated with one controller c0 and one middle-box m1.

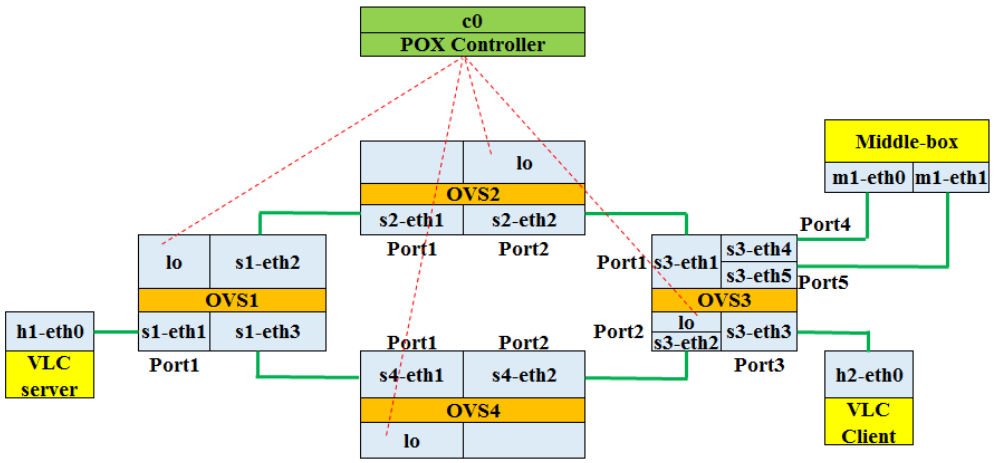


Figure 3.2: Mininet configuration of two-path OpenFlow network [10].

Necessary Python scripts developed in [10] have been used to assign link capacity values. Particularly, for experiment 1, the links between h1-s1, h2-s3, m1-s3, s1-s2, s2-s3 are configured in Mininet with 0.3 Mbits/s and links between s1-s4 and s4-s3 are set to 0.15 Mbits/s for link capacity while for experiment 2 is set to 0.25 Mbits/s (upper path) and 0.15 Mbits/s (lower path). After the network topology inside the Mininet has been

instantiated, the Python script has been executed to add flow entries to all the switches via the POX controller. For adding flow entries to all switches, there are two special actions for OVS1 and OVS3.

OVS1 is responsible for splitting the incoming video stream from the server into two smaller streams. The packets are chunked at this switch periodically by specifying m s to parameter hard_timeout of the flow entry for packet transmission via path 1 and n s to parameter hard_timeout for that via path 2. The ratio m:n then determines the chunk size ratio in this splitting. The controller periodically feeds the new flow entries in turn to OVS1 such that, upon an expired flow entry, a renewal of the other flow entry will be issued immediately. Figure 3.3 depicts the state machine diagram of our implemented OpenFlow controller. Tables 3.1 and 3.2 give the header matching criteria as well as corresponding actions for transmission of packets via data path 1 and data path 2, respectively.

Table 3.1: Flow entry of OVS1 for transmission via path 1 [10]

Header field	Action	Timeout
in_port = 1	output:2	hard_timeout = m
in_port = 2	output:1	hard_timeout = m

Table 3.2: Flow entry of OVS1 for transmission via path 2 [10]

Header field	Action	Timeout
in_port = 1	output:3	hard_timeout = n
in_port = 3	output:1	hard_timeout = n

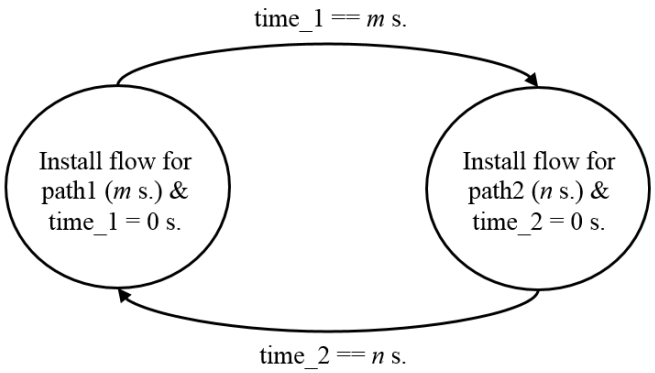


Figure 3.3: OpenFlow controller state machine at OVS1 [10].

OVS3 is responsible for combining the packets via path 1 and path 2. Before OVS3

forwards arriving packets for further processing in the middle-box, OVS3 marks an arriving packet according to the path being traversed by the packet. For convenience, the layer-2 address is used for such marking. The middle-box can then use the layer-2 address to implement packet classification once it can capture packets at interface m1-eth0. After the classification, packets arriving at OVS3 via path 1 (h1-s1-s2-s3-m1) and via path 2 (h1-s1-s4-s3-m1) are enqueued at two parallel dedicated packet buffers. A packet scheduling algorithm is then invoked to choose to send the waiting packet in the two buffers with the lowest time stamp value in RTP header. This ensures that the packets forwarded out of middle-box can maintain the correct sequence of packet delivery as close as possible to the original timing sequence generated from the video streaming server. The outgoing packets from the middle-box is forwarded by interfaces m1-eth1 to s3-eth5 then s3-eth3 of OVS3, and finally towards the video client. As for the middle-box implementation, this research has used a Python script developed in [10] based on packages, pcapy, dpkt and scapy for capturing, parsing and sending out RTP packets, respectively.

Figure 3.4 summarizes the overall mechanism being implemented in our testbed by a conceptual queuing model diagram, where we define the following notations

λ: incoming packet rate from video server (packets/s),

λ_1: packet rate on path 1 (packets/s),

λ_2: packet rate on path 2 (packets/s),

μ_1: bandwidth of path 1 (packets/s),

μ_2: bandwidth of path 2 (packets/s),

μ: packet scheduling rate of middle-box (packets/s).

In Figure 3.4, the timeout parameters of flow entries in OVS1 is directly proportional to the resultant packet rate on each path i.e. $\lambda_1 = m\lambda/(m+n)$ and $\lambda_2 = n\lambda/(m+n)$. If both paths can maintain lossless packet transmission, then the same values of λ_1 and λ_2 will continue towards the classifier. However, if the paths are lossy, then only a reduced fraction of packet rates will be present. If the packet backlogs in both buffers 1 and 2 are not depleted throughout the video playback period, then the scheduling packet rates for the two buffers will be proportional to the incoming demand rates with the proportional constant being the total packet scheduling rate. For such a scenario, the rates at different parts of

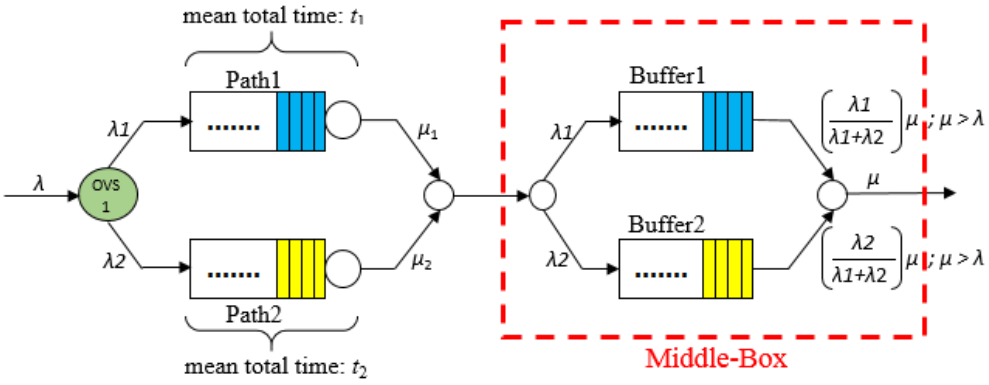


Figure 3.4: Packet queuing model example conceptualizing OpenFlow network testbed in case of two lossless paths [10].

the overall packet queuing model will be exactly as depicted in Figure 3.4. In practice, the chunk splitting ratio and instantaneous path delays will affect the queue length variation in buffers of the middle-box. However, in our current settings of first-trial experiments, the focus is to verify if the load splitting and combining functionalities can work properly in the testbed with no background traffics on both paths 1 and 2 and to try to check the effects of varying the chunk size ratio upon the performance of streaming video.

3.1.3 Video Streaming Experiment 1 Scenario and Results

The Big Buck Bunny [44], an animation video with the resolution 432 x 242 and the duration of 10 minutes, is used for streaming video experiment 1. The H.264 video codec with video and audio mean bit rate of total 296 kbits/s is streamed from the video server to the video clients, both of which are running VLC player software [45]. The VLC player from the video server is configured to stream out video by using RTP mode.

Three scenarios in Table 3.3 have been tested by varying chunk size ratio upon the same outgoing packet rate and no initial buffering in the middle-box before first starting the scheduler operation. To evaluate these results, the Wireshark software [46] has been used to capture all the incoming and outgoing packets on both Ethernet interfaces of video server and clients. And a Matlab script is written based on the main program developed in [10] for off-line packet header processing to produce all result outcomes by adding the function to generate mean/standard deviation of packet loss and packet delay. Table 3.4 shows the trends of resultant packet loss ratio, mean/standard deviation of packet delay for each chunk size ratio. For each of the testing scenarios, we have repeated the test three times to check the variation of outcomes. Furthermore, in order to compare the results of packet loss and delay between cases using multi-path scenarios and without using multi-path scenarios, we have also evaluated the extremal values at the chunk size ratio of 800:0 (using path 1 only) and 0:800 (using path 2 only). The results are depicted in Table 3.5.

Table 3.3: Testing experimental scenario

μ (packets/s)	Chunk size ratio
40	30:10(Multi-path,30s via path1:10s via path2) 10:10 (Multi-path,10s via path1:10s via path2) 10:30 (Multi-path,10s via path1:30s via path2)

Table 3.4: Experimental results for μ=40

Chunk size ratio	Packet loss ratio (%)	Packet delay (s)	
		mean	standard deviation
30:10	0	4.3	2.2
10:10	0	4.3	3.1
10:30	19.2	43	30.5

Table 3.5: Experimental results without using multi-path scenario

Chunk size ratio	Packet loss ratio (%)	Packet delay (s)	
		mean	standard deviation
800:0 (Using path 1 only)	0	0.2	0.1
0:800 (Using path 2 only)	35.7	60.6	17.6

From Table 3.4, the average packet loss is 19.2% for chunk size ratio 10:30 since path 2 has only 0.15 Mbits/s capacity while the incoming video/audio packet rate is 0.296 Mbits/s. In this case, the extra 0.146 Mbits/s will be prone to losses if the storing buffers get overflown due to the limited bandwidth of path 2. The packet loss ratio of video streaming with chunk size ratio 30:10 and 10:10 is zero since most of the duration video streaming forwards packets on path 1 and sufficiently small duration of the time on path 2. Moreover, the case of 30:10 outperforms the case of 10:10 in terms of standard deviation of packet delay or jitter because the case of 30:10 better matches with the available path capacity. The resultant video packet rates can thus be properly fitted into the available capacity of both individual paths. Therefore, this reported packet loss and delay result confirms that the multi-path video streaming method can be beneficially applied in the network when the capacity of the main path alone is not enough to carry the whole incoming packets of video stream and the employed chunk splitting ratio decomposes the incoming packet rate to match the capacity of available paths.

However, when comparing the cases of 30:10 and 800:0 (using path 1 only), the latter gives a better packet jitter performance. This is due to that, without need to pass packets through both paths, one can save the increased processing complexities at both the splitting and combining functionalities by avoiding the possible large deviation of path delays. Thus, the multi-path video streaming method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video stream due to the increased implementation complexities. And in case that chunk size ratio is equally using path 1 and path 2 (10:10), the packet jitter is higher than that of 30:10 but lower than that of 10:30. This case of 10:10 signifies the equal load balancing case that treats both paths with the same balancing weight. As seen in Table 3.4, such equality does not necessarily work for its performance depends a great deal on the available path capacity at the time.

For detailed dynamics of packet generation arriving at the video server and the received packet stream departing at the video client, we include here the arrival and departure curves in Figures 3.5, 3.6, 3.7 and 3.8. The packet number shown in graphs is the RTP sequence number with offsetting to start the packet number at zero. These curves confirm the packet loss and delay results as consistently summarized earlier in Tables 3.4 and 3.5.

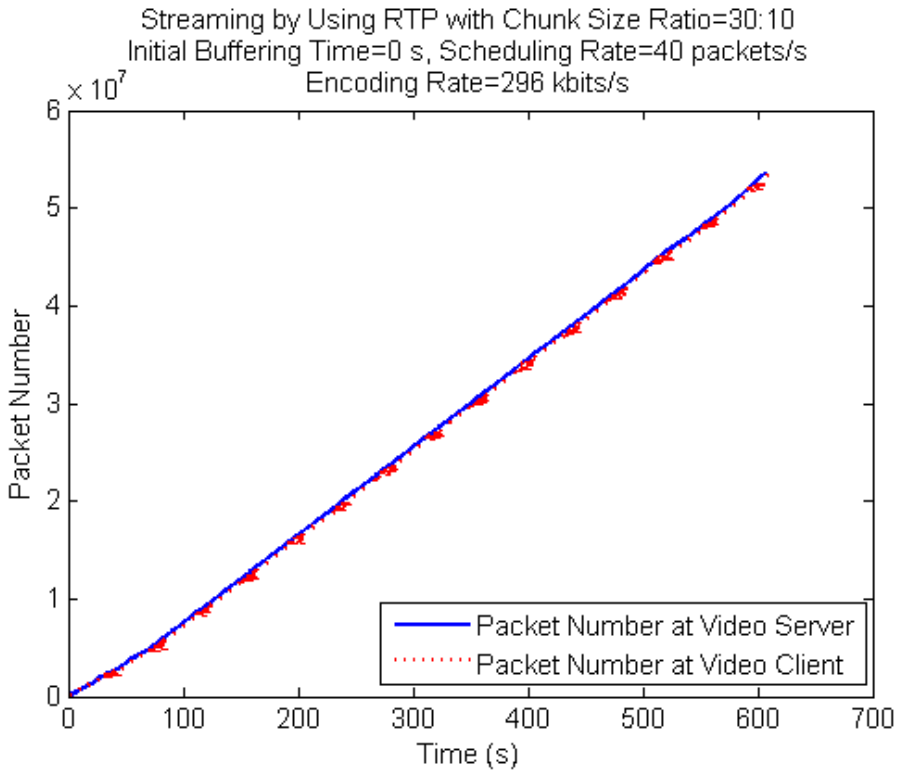


Figure 3.5: Experiment result with μ=40: chunk size ratio 30:10.

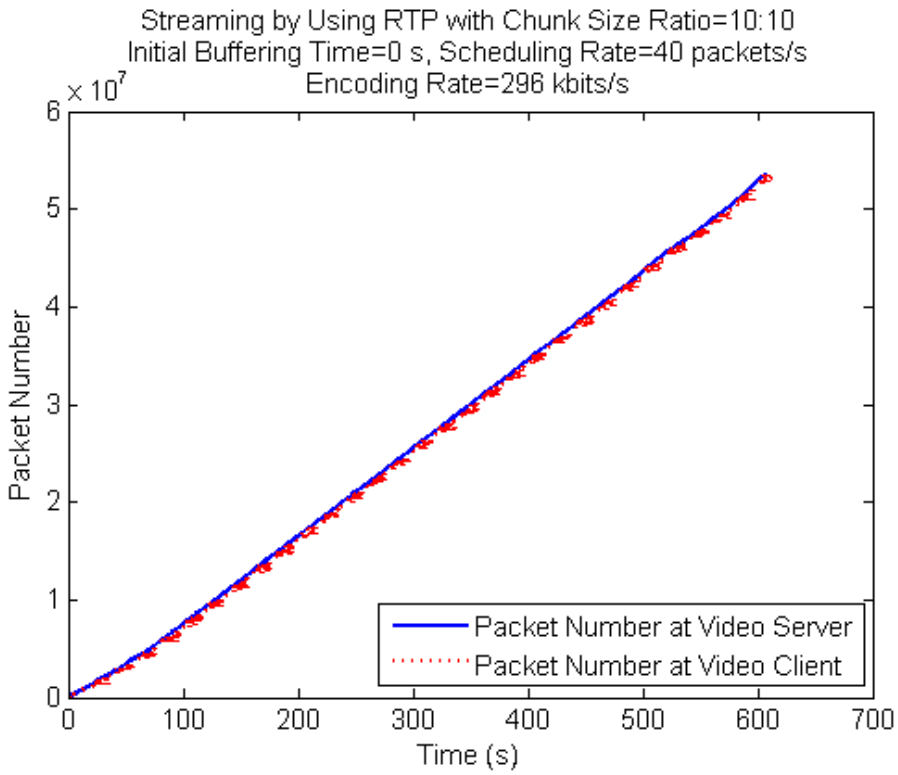


Figure 3.6: Experiment result with μ=40: chunk size ratio 10:10.

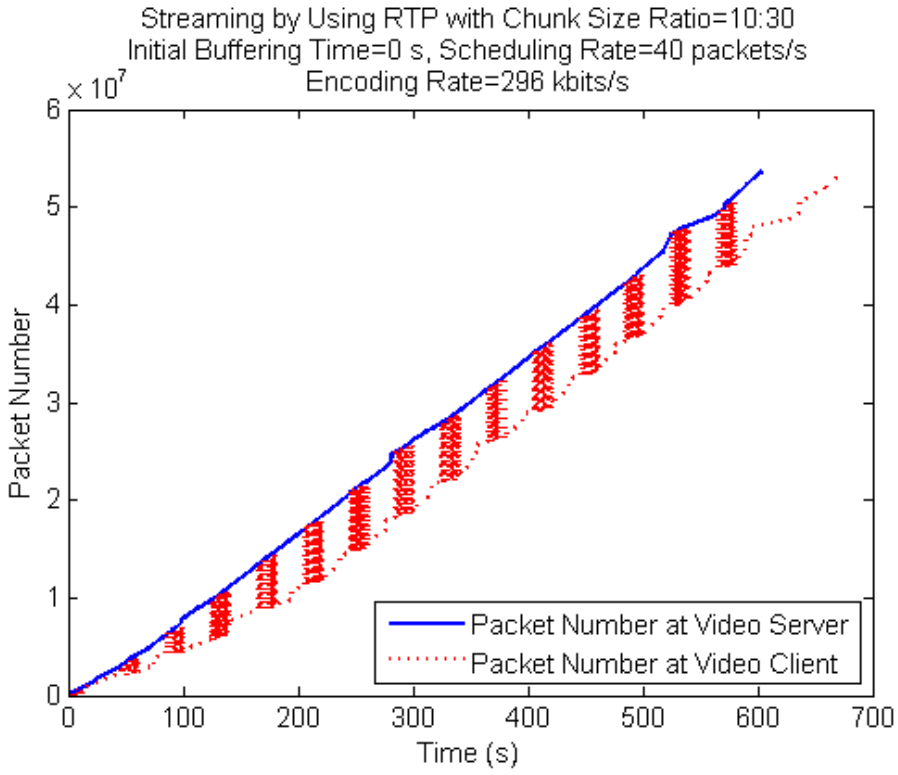
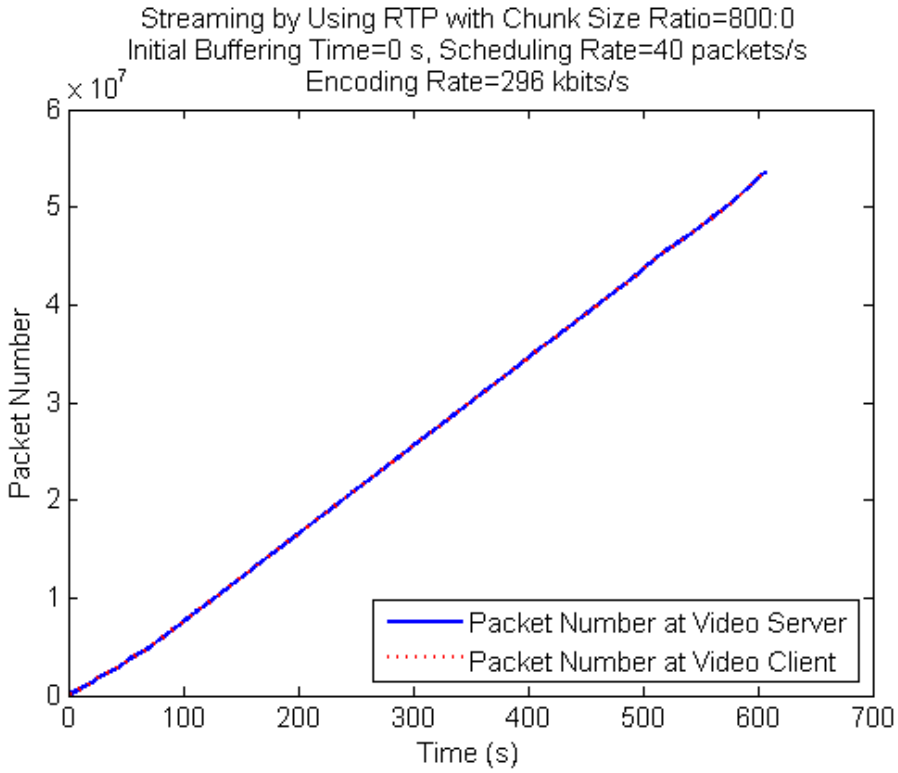
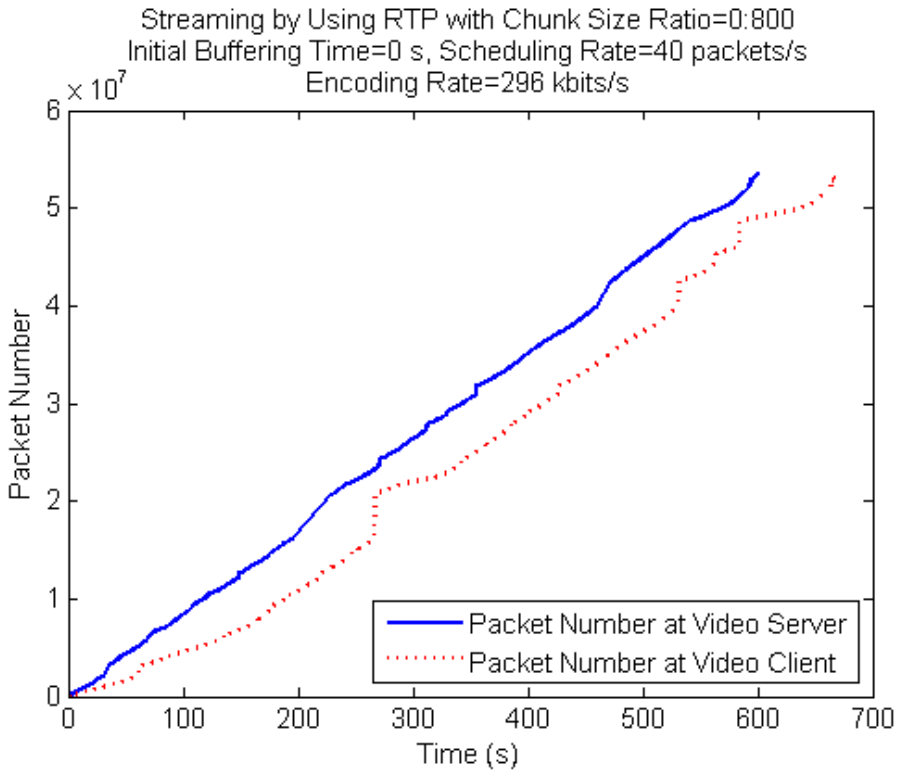


Figure 3.7: Experiment result with μ=40: chunk size ratio 10:30.



(a)



(b)

Figure 3.8: Experiment results without using multi-path scenarios: (a) chunk size ratio 800:0 (using only path 1) (b) chunk size ratio 0:800 (using only path 2).

3.1.4 Video Streaming Experiment 2 Scenario and Results

Unlike experiment 1 in Subsection 3.1.3, a Frozen video [47] clip with H.264 video codec, 320x144 resolution, total video/audio bit rate 246 kbits/s and 2-minute playback duration has been used in this experiment 2. These settings have been chosen to allow the dynamic range of testable input parameters with available performance of computer hardware specification. VLC server program has been used to stream out the video by RTP mode. The initial buffering time varies from 0, 5, 10, 15 to 20 s. The chunk size ratios of 20:10 and 10:20 have been used. The packet scheduling rate of middle-box is set to 25 packets/s to match with the server video transmission rate.

As in the experiment 1, wireshark software is used to capture all the transmitting and receiving packets on both Ethernet interfaces of server and client. To investigate the effects of initial buffering time and chunk size ratio on packet delay, we have tested three times with the same parameter settings and computed the mean/standard deviation of packet delay. Figures 3.9 and 3.10 depict the mean/standard deviation of packet delay in our experiments. These are the average results from three experiments with chunk size ratio (20:10 and 10:20) and initial buffering times (0, 5, 10, 15, 20 s).

From Figure 3.9, when initial buffering time increases for both chunk size ratio settings, the mean packet delay increases with its upper bound at the initial buffering time. The mean packet delay of chunk size ratio 20:10 case is lower than that of 10:20 case. In case of 20:10, most of the packets are streamed out through path 1 and use only a short period of time on path 2. The former path has enough capacity to carry the whole incoming traffic than the latter path. When compared to the packet delay results between 20:10 and 10:20 cases, it is clear that proper chunk size setting is important to reduce packet delay due to the higher delay of 10:20 in which using the lower bandwidth path is more than the higher bandwidth path.

Moreover, we have studied the jitter performance in terms of the standard deviation of packet delay. The standard deviation of packet delay results from Figure 3.10 demonstrates that jitter performance of 20:10 case is better than that of 10:20 case. In order to obtain the best jitter performance, the proper initial buffering time and chunk size ratio settings need to be considered carefully. The reason that 20:10 case is better than 10:20 is because path 1

has enough 0.25 Mbits/s capacity to carry the whole video traffic 0.246 Mbits/s while path 2 has only 0.15 Mbits/s. Finally, it is noted that, since the middle-box can handle all the video packets in this two-minute video streaming experiment, the packet loss ratio is zero in most experiments. Although the packet loss ratio results show zero in this experiment, there may have performance degradation when we calculate the PSNR values upon the received video quality at the client side due to the capability of computer specification for middle-box processing. And the high performance machine is recommended for operating the middle-box function in real network since the middle-box has to process a number of functions on fast-coming packets.

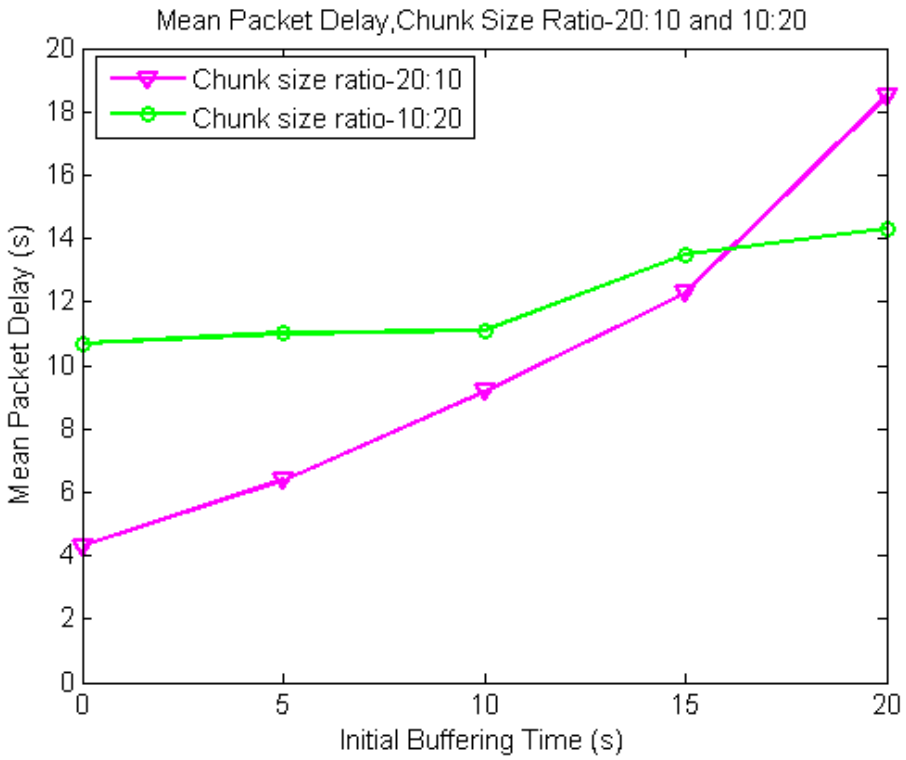


Figure 3.9: Mean packet delay vs initial buffering time for chunk size ratio 20:10 and 10:20.

3.1.5 Summary of Emulated Multi-path Video Streaming

In these experiments, we have reported the design and functionality test of middle-box and load splitting for chunked video streaming over the OpenFlow-enabled multi-path Mininet network. The reported packet delay, jitter and packet loss results in experiment 1 demonstrate that multi-path video streaming method can be beneficially applied in the

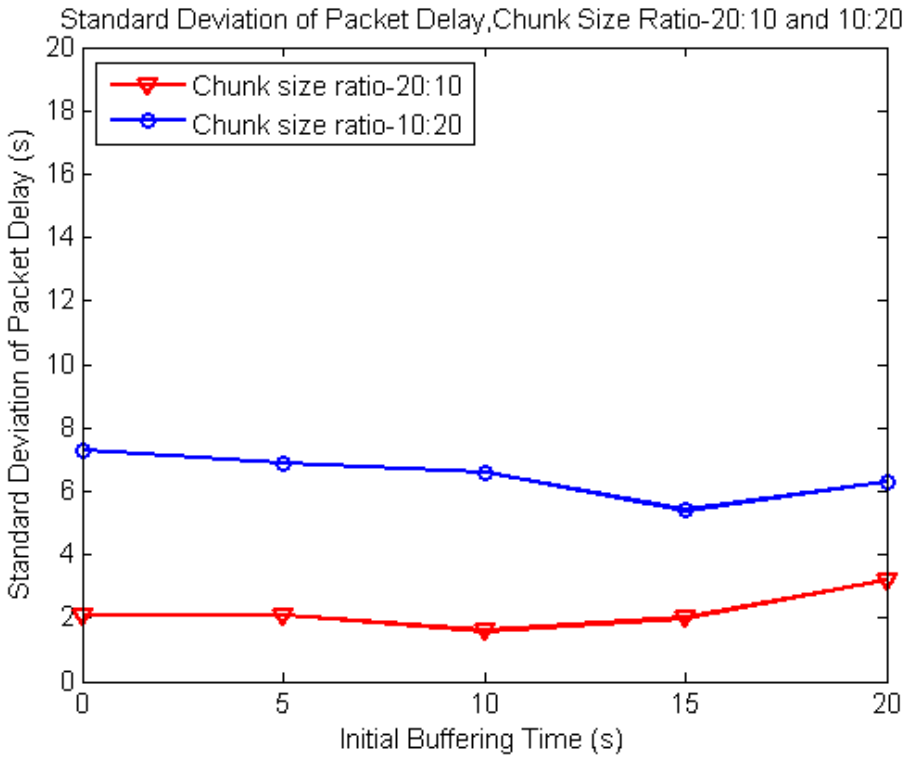


Figure 3.10: Standard deviation of packet delay vs initial buffering time for chunk size ratio 20:10 and 10:20.

network when the capacity of the main path alone (using path 2 alone) is not enough to carry the whole incoming packets of video stream and the employed chunk splitting ratio decomposes the incoming packet rate to match the capacity of available paths (both paths 1 and 2). However, this multi-path video streaming method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video stream due to the results of experiment 1. Since by introducing splitting and combining functionalities, relevant complexities, e.g. mismatching of proper chunk splitting ratio, could worsen the received video quality. Particularly, we can see this from the case in experiment 1 "using path1 only" vs "using paths 1 and 2", where mean and standard deviation of packet delay of the former is always lower than those of the latter. Due to the results described in experiment 1, we realize that matching of chunk splitting ratio to the capacity of available paths can affect greatly the resultant received subjective video quality in terms of packet loss ratio and mean/standard deviation of packet delay. This poses a new practical challenge when path capacity is time varying and stochastic in nature. According to the results of experiment 2, we observe that the proper chunk size ratio and initial

buffering time need to be considered carefully to improve the mean/standard deviation of packet delay. Moreover, it is recommended to use the high performance machine for middle-box processing since the maximum possible packet scheduling rate of the middle-box depends directly on computer hardware specifications. All these investigate results have provided a firm foundation when we plan to implement the multi-path chunked video streaming experiment over the OF@TEIN multi-national Openflow testbed [13] and include the wireless link for conducting research on wireless SDN experiments.

3.2 Design of Middle-box and Multi-path Chunked Video Streaming over OF@TEIN SDN Cloud Playground

As in our objective, the goal of this thesis is to investigate the middle-box and splitting functionalities of chunked video streaming over OF@TEIN SDN cloud playground between Thailand, Malaysia and Korea. OF@TEIN SDN cloud playground is connected with SmartX Boxes from 9 countries including Thailand, Malaysia and Korea. OF@TEIN [6, 7] is one of the large-scaled SDN-Cloud testbeds with 9 research and educational networks as shown in Figure 3.11.

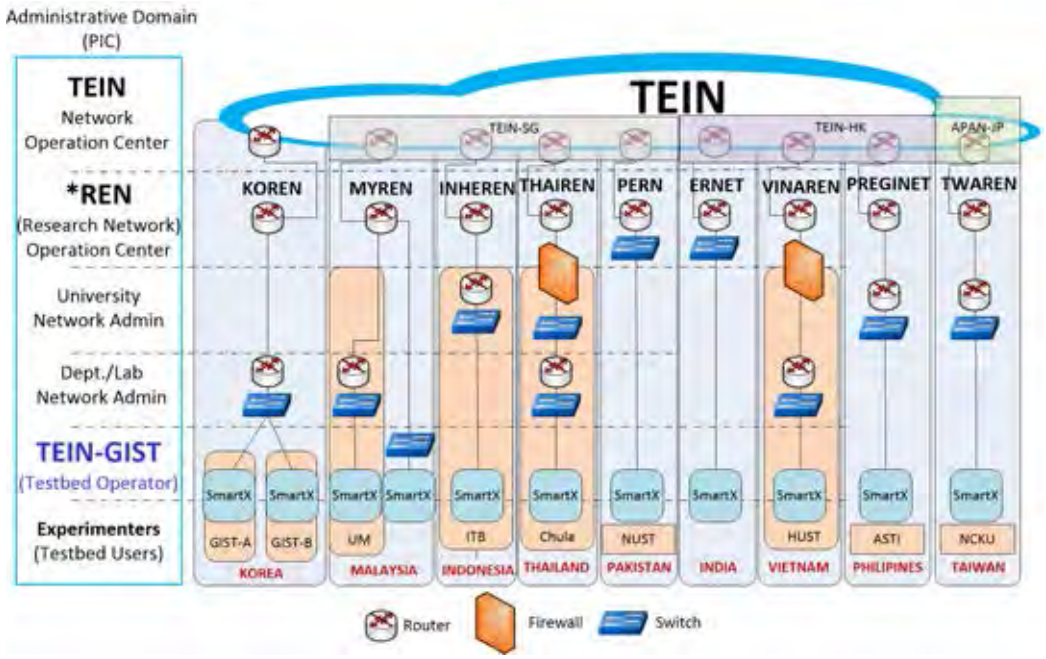


Figure 3.11: OF@TEIN multi-domain network infrastructure [6, 7].

3.2.1 Implementation of Multi-path Chunked Video Streaming Sessions over OF@TEIN SDN Cloud Playground

Currently, SmartX Boxes in OF@TEIN SDN cloud playground are upgraded to Type B* with OpenStack and Open vSwitches except the domestic Korea sites are installed with Type C [6]. OpenStack [48] is the most current popular open source software to create public and private cloud computing networks. Open vSwitch [49] is a multilayer virtual switch to create the OpenFlow rules in virtualized network environment. The SmartX Type B*

in Thailand and Malaysia has single CPU and SATA storage with tunneling-based overlay network connection whereas Type C used in Korea sites has dual CPU and SSD storage with VLAN-based network configuration.

In order to run the experiments over OF@TEIN SDN cloud playground, we first need to request the network slices from OF@TEIN network administrator. After that we design our multi-path chunked video streaming infrastructure on top of the OF@TEIN SDN cloud playground. Moreover, iPerf [50] testing between CHULA, MYREN and GIST-A needs to perform in order to check the available bandwidth capacity of each international link. The network architecture of this experiment is extended from the main architecture of OF@TEIN SDN cloud playground [6].

The architecture of this multi-path video streaming experiment consists of three international nodes located in CHULA (Thailand), MYREN (Malaysia), and GIST-A (Korea). Thailand's SmartX Box is located at Chulalongkorn University Gateway room, Chamchuri 9 Building. The physical SmartX boxes locations of GIST-A (Korea) and CHULA (Thailand) are shown in Figure 3.12.

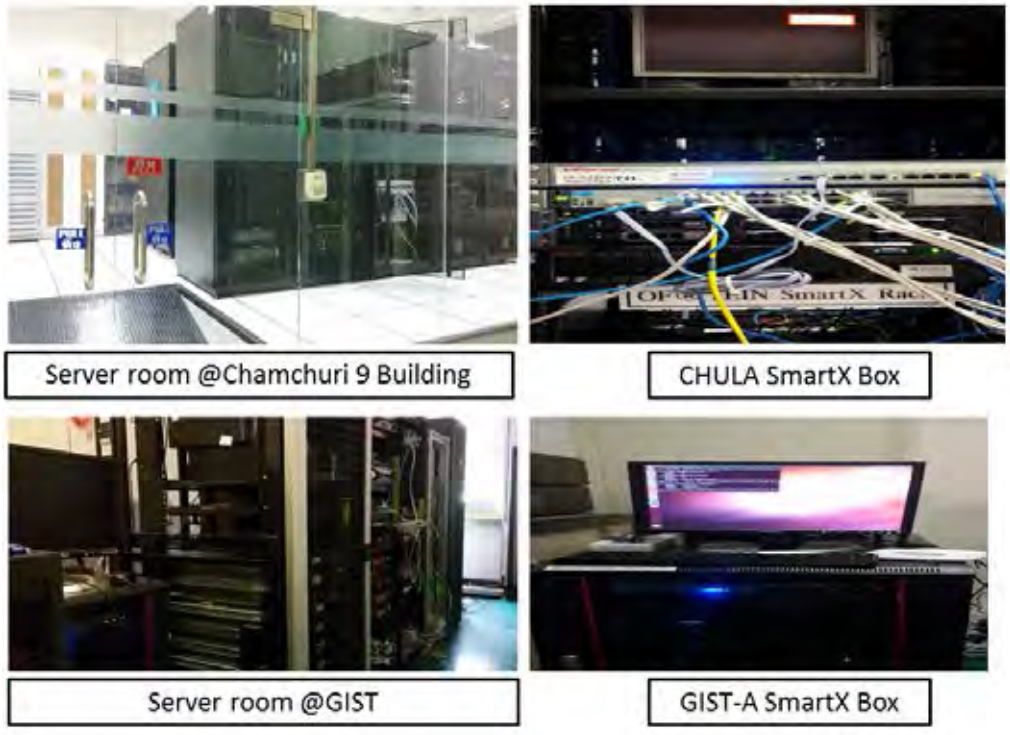


Figure 3.12: Physical SmartX boxes locations of GIST and CHULA.

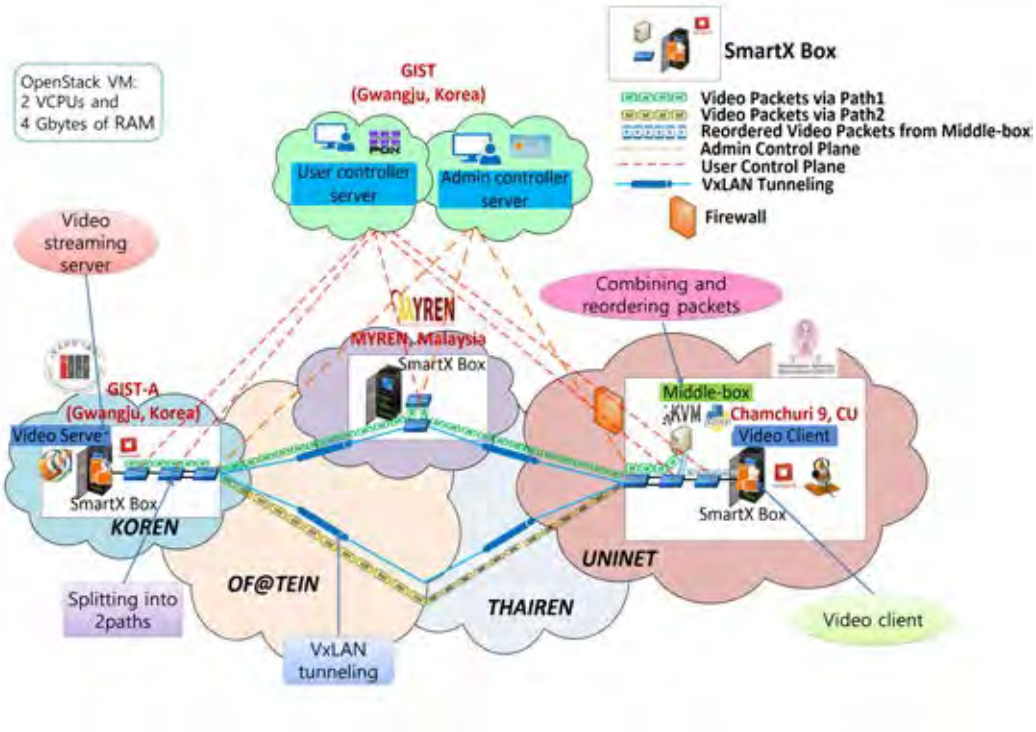


Figure 3.13: Overview of multi-path chunked video streaming over OF@TEIN SDN cloud playground.

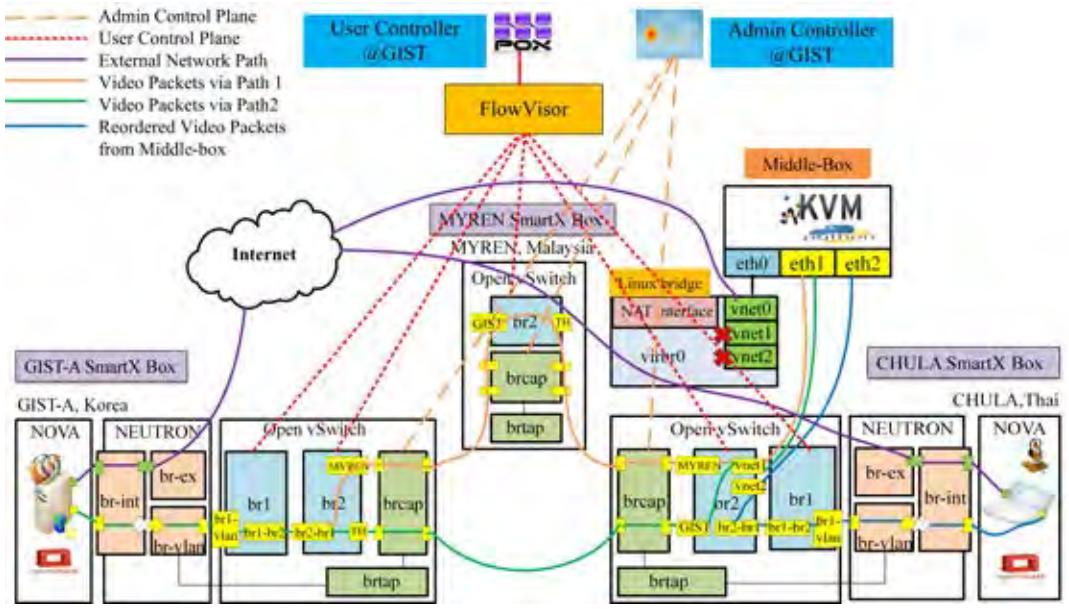


Figure 3.14: Architecture of multi-path chunked video streaming over OF@TEIN SDN cloud playground.

The overall and detail architectures of multi-path chunked video streaming over OF@TEIN SDN cloud playground are depicted in Figures 3.13 and 3.14. As described in Figures 3.13 and 3.14, we have created OpenStack virtual machines (VMs) with 2 VCPUs and 4 GBytes of RAM in both GIST-A and CHULA OpenStack compute nodes for serving as a video streaming server and a video client respectively. Both OpenStack VMs are running Ubuntu 14.04 OS. According to OF@TEIN architecture, there are two networks (control plane: eth0 and data plane: eth1) for each OpenStack VM. The control plane: eth0 is used for accessing VM via ssh and data plane: eth1 is used for transmitting the experimental data such as iPerf traffic and video traffic. The Maximum Transmission Unit (MTU) of eth1 has been set to be 1410 bytes in order to be able to transmit iPerf traffic since OF@TEIN playground VMs cannot send iPerf traffic with default MTU size 1500 bytes. FlowVisor [51] has been used to create the slices of network resources and delegate control of each slice to different controllers. In addition, we are not responsible for FlowVisor management since it is controlled by OF@TEIN admin team. Each developer in OF@TEIN has its own VLAN ID in order to identify and assign the developers to run experiments without inferencing each other. Nova [52] and Neutron [53] are OpenStack compute and networking projects in which Nova is responsible for initializing the OpenStack instances while Neutron is for providing network services between interface devices (e.g. vNICs) managed by other OpenStack services (e.g. Nova). Virtual extensible LAN (VxLAN) [54], a network virtualization technology for large cloud computing network, is used to connect the nodes between Korea, Malaysia and Thailand. The Neturon in this architecture consists of three interfaces (br-int, br-ex and br-vlan). The interface br-int is to communicate between logical interfaces of OpenStack instances and physical interfaces of local host while br-ex is to connect with external networks from OpenStack instances. The interface br-vlan is to communicate between VMs across the networks through VxLAN tunneling and Open vSwitches.

There are 4 Open vSwitch bridges in each site of SmartX Boxes: br1, b2, brcap and brtap. The OVS bridges (br1 and br2) are controlled by OF@TEIN user's controller (i.e. POX controller) while brcap and brtap are controlled by OF@TEIN operator's controller (i.e. OpenDayLight controller [55]). The purpose of each bridge is as follows:

br1: to connect between OVSs and OpenStack Neutron interface (br-vlan)

br2: to connect between OVS (br1) and OVS (brcap)

brcap: to connect between OvSs in different countries via VxLAN tunneling

brtap: to monitor the traffic of tap interfaces of VMs

In this multi-path chunked video streaming experiment, OpenStack VM (VLAN ID: 111) at GIST-A serves as a video streaming server and OpenStack VM (VLAN ID: 111) at CHULA serves as a video client. In this experiment, the total 5 OVS bridges are used in SmartX Boxes of GIST-A, MYREN and CHULA. To introduce load balancing mechanism with chunked video splitting functionality, we have chosen two paths namely Path 1 and Path 2 as in our previous emulated experiments network. The route of Path1 is via GIST-A $\gg$ MYREN $\gg$ CHULA while Path 2 is directly connected between GIST-A and CHULA. SmartX Boxes of GIST-A, MYREN and CHULA are connected to each other by using Open vSwitch VxLAN tunneling. POX controller is used for adding the flow entries into OVS bridges (br1 and br2) at GIST-A, MYREN and CHULA.

Firstly, the video server at GIST-A, Korea streams out the chunked video streaming packets into multiple concurrent paths via Path 1 (via GIST-A $\gg$ MYREN $\gg$ CHULA) and Path 2 (via GIST-A $\gg$ CHULA). GIST-A OVS (br2) is responsible for splitting the chunked video streaming into two paths. CHULA OVS (br2) serves as a packet combiner from two paths towards the middle-box (vnet1). Virtual middle-box has been implemented in CHULA SmartX Box by using Kernel Virtual Machine (KVM) with 1 VCPU and 2 Gbytes of RAM running Ubuntu 12.04 LTS OS for performing a packet scheduler and classifier with two parallel buffers (buffer1 and buffer2). Virt-manager has been used to create the middle-box KVM. Once initiate the KVM, three virtual interfaces namely vnet0, vnet1 and vnet2 are attached to SmartX box's linux bridge (virbr0). After that only vnet0 is attached to virbr0 and vnet1 and vnet2 are removed from that linux bridge. vnet0 is attached to linux bridge (virbr0) and then used for internet access and VM access via ssh. vnet1 and vnet2 are attached to OVS (br2) for capturing and processing the incoming packets and generating the packets accordingly. Those network interfaces are appeared inside the middle-box namely as eth0, eth1 and eth2. When implementing the middle-box in CHULA SmartX box, we have faced the looping issue due to wrong configuration.

However, we have found the solution to avoid looping issue. The middle-box has been used to combine and reorder all the incoming video packets from two paths by using RTP video time stamps and then forward the video packet stream via (vnet2) towards OVS (br1) and destination to be the video client as smooth as possible. When implementing the middle-box KVM in CHULA SmartX box with virtual-machine manager (virt-manager), one important thing is for configuring to avoid looping issue. The implementation set up for middle-box configuration will be described in the Appendix D.

The same middle-box functionalities Python script used in the previous emulated multi-path video streaming experiments has also used in this experiments by modifying the codes to generate packets with VLAN header and changing the MAC addresses of two concurrent paths accordingly. The splitting and middle-box functionalities are the same as in the emulated multi-path video streaming experiments. The port information and flow entries of each SmartX boxes have been used in this experiments are shown in Tables 3.6-3.12.

Table 3.6: Port information for SmartX boxes (GIST-A,MYREN and CHULA)

SmartX boxes	br1 ports	br2 ports
GIST-A(103.22.221.170)	1(br1_br2) 2(br1_vlan)	1(MYREN) 2(TH) 4(br2_br1)
MYREN(103.26.47.228)		3(TH) 4(GIST)
CHULA(161.200.25.99)	1(br1_vlan) 2(br1_br2)	1(br2_br1) 2(GIST) 3(MYREN) 10(vnet1) 11(vnet2)

Table 3.7: Flow entry of GIST-A SmartX box (OVS: br1)

Header field	Action
in_port=1(br1_br2)	output:2(br1_vlan)
in_port=2(br1_vlan)	output:1(br1_br2)

The flow entries of OVSs (br1) at GIST-A and CHULA and OVS (br2) at MYREN as shown in Tables 3.7, 3.11 and 3.10 are responsible for forwarding the incoming packets from the ingress port towards egress port and vice versa.

Table 3.8: Flow entry of GIST-A SmartX box (OVS: br2) for transmission via path 1 (GIST-A ≫ MYREN ≫ CHULA) [10]

Header field	Action	Timeout
in_port = 1(MYREN)	output:4(br2_br1)	hard_timeout = m
in_port = 4(br2_br1)	output:1(MYREN)	hard_timeout = m

Table 3.9: Flow entry of GIST-A SmartX box (OVS: br2) for transmission via path 2 (GIST-A ≫ CHULA) [10]

Header field	Action	Timeout
in_port = 2(TH)	output:4(br2_br1)	hard_timeout = n
in_port = 4(br2_br1)	output:2(TH)	hard_timeout = n

As shown in Tables 3.8 and 3.9, the video packets are periodically chunked into smaller chunked video packets in this br2 at GIST-A by specifying the m s to parameter hard_timeout of the flow entry for packet transmission via Path 1 and n s to parameter hard_timeout for that via Path 2. The ratio m:n then determines the chunk size ratio in this splitting function as in the emulated multi-path video streaming experiments. The detail of splitting function is the same as in the emulated multi-path video streaming experiments.

Table 3.10: Flow entry of MYREN SmartX box (OVS: br2)

Header field	Action
in_port=3(TH)	output:4(MYREN)
in_port=4(MYREN)	output:3(TH)

Table 3.11: Flow entry of CHULA SmartX box (OVS: br1)

Header field	Action
in_port=1(br1_vlan)	output:2(br1_br2)
in_port=2(br1_br2)	output:1(br1_vlan)

The flow entry of OVS (br2) in CHULA SmartX box is depicted in Table 3.12. This br2 in CHULA SmartX box is responsible for combining the packets from two paths and forwarding the reordered video packets from the middle-box to the video client in CHULA OpenStack VM. When the packets arrive from the ingress MYREN and GIST-A ports, the packet headers are set to be e2:9c:e6:30:bf:06 for packets coming from MYREN and

Table 3.12: Flow entry of CHULA SmartX box (OVS: br2) [10]

Header field	Action
in_port = 3(MYREN)	mod_dl_src:e2:9c:e6:30:bf:06, output:10(vnet1)
in_port = 2(GIST)	mod_dl_src:66:c2:a5:35:d2:d6, output:10(vnet1)
in_port = 1(br2_br1)	ALL
in_port = 11(vnet2)	output:1(br2_br1)
in_port = 10(vnet1)	output:drop

66:c2:a5:35:d2:d6 for packets coming from GIST-A and then forwarded to the vnet1 for further processing in the middle-box. The packets arrive from br2_br1 ingress port from CHULA OpenStack VM will be forwarded to all available ports at br2 except the ingress port br2_br1. The reordered packets generated from the middle-box will be forwarded via vnet2 to br2_br1 and then finally towards br1 and the video client. All the packets coming from vnet1 is dropped in order to process the packets reordering in the middle-box.

3.2.2 Results and Discussion of Multi-path Chunked Video Streaming over OF@TEIN SDN Cloud Playground

As in the emulated multi-path video streaming experiment 2, a Frozen video [47] clip with H.264 video codec, 640x286 resolution, total video/audio bit rate 628 kbits/s and 2-minute playback duration has been used in this multi-path video streaming experiments. These setting have been chosen to allow the dynamic range of testable input parameters with available performance of SmartX box hardware specification since CPU and RAM of SmartX boxes have limitation due to running multiple users and multiple software applications. As in the previous experiments, VLC server program has been used to stream out the video by RTP mode. At the time of running experiments, OpenStack VMs over OF@TEIN playground have no function for using GUI applications with remote desktop. In order to open GUI applications, only X 11 display is available and it has limitation to open GUI applications from a distant node with fast access. Therefore, in this experiment, VLC command line has been used for GIST-A video server and VLC with GUI interface has been used for CHULA video client. GUI applications to gain an access of OpenStack VMs still need to be improved. The demonstrating environment of multi-path chunked video streaming over OF@TEIN SDN cloud playground is shown in Figure: 3.15.

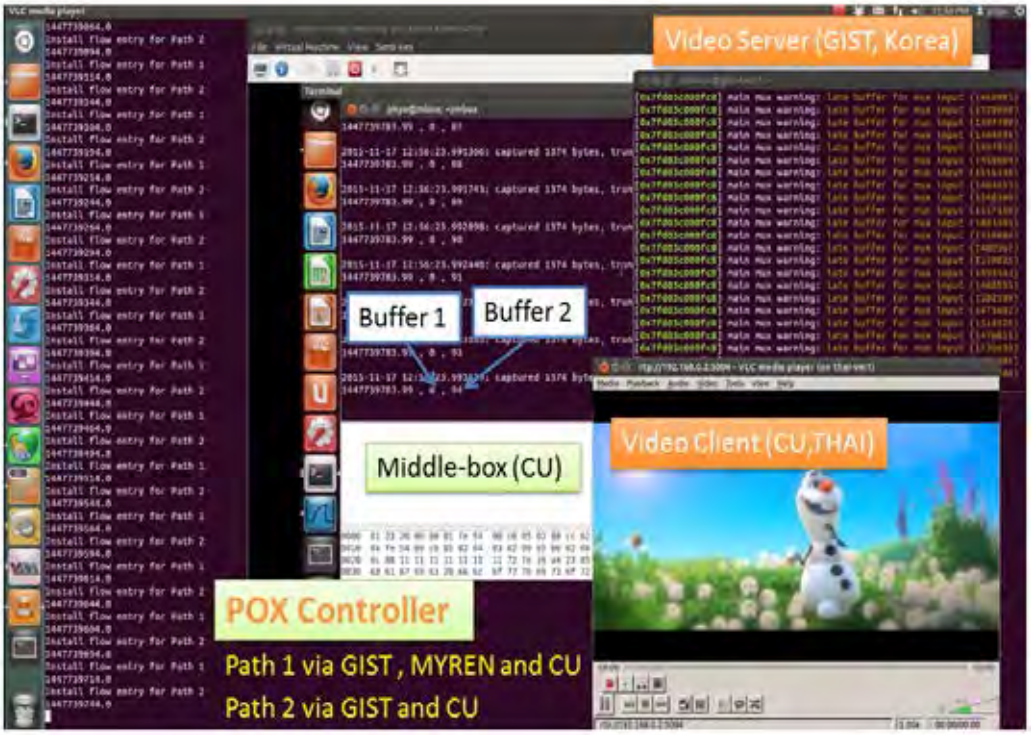


Figure 3.15: Demonstrating environment of multi-path chunked video streaming over OF@TEIN SDN cloud playground.

iPerf application has been used in order to know the available bandwidth capacity between GIST-A, MYREN and CHULA nodes. According to the iPerf UDP test results, the bandwidth capacities of around 82 Mbps (between OpenStack VMs via GIST-A, MYREN and CHULA link) via Path 1 and around 450 Mbps (between OpenStack VMs via GIST-A $\gg$ CHULA) via Path 2 have been obtained from the test. The iPerf test results indicate that there are plenty of bandwidth capacities between three selected nodes (GIST-A, MYREN, CHULA) in order to run our multi-path video streaming experiments. The initial buffering time in the middle-box varies from 0, 10 and 20 s. The chunk size ratios of 20:10 and 10:20 have been used as in the emulated video streaming experiments 2. The packet scheduling rate (μ) of the middle-box is set to 150 packets/s to match with server video transmission rate.

Unlike in the emulated video streaming experiments, tcpdump [57], a command line packet analyzer, has been used to capture all the transmitting and receiving packets on both Ethernet interfaces (eth1) of server and client due to the limited access convenience of GUI applications over OF@TEIN SDN cloud playground. As in the emulated multi-path

video streaming experiments, we have tested three times for both multi-path and single-path video streaming experiments over OF@TEIN SDN cloud playground with the same parameter settings and computed the mean/standard deviation of packet delay in order to investigate the effects of chunk size ratio on packet delay. The average results of packet delay from three experiments with chunk size ratios (20:10 and 10:20), single-path with and without the middle-box (mbox) and initial buffering time (0 s) are shown in Table 3.13.

Table 3.13: Chunked video streaming experimental results with initial buffer (0 s)

Scenario	Packet delay (s)	
	mean	standard deviation
20:10 (Multi-path)	3.32	1.89
10:20 (Multi-path)	1.76	1.44
200:0 GIST-A $\gg$ MYREN $\gg$ TH (Single-path with mbox)	1.75	1.12
0:200 GIST-A $\gg$ TH (Single-path with mbox)	1.76	1.07
GIST-A $\gg$ MYREN $\gg$ TH (Single-path without mbox)	0.01	0.01
GIST-A $\gg$ TH (Single-path without mbox)	0.01	0.01

From Table 3.13, the mean and standard deviation of packet delay using single-path without the middle-box outperforms the mean and standard deviation of packet delay using single-path with the middle-box because of packet processing delay in the middle-box. Unlike the results that have been achieved from emulated multi-path and single-path video streaming experiments, both transmission via Path 1 and Path 2 over OF@TEIN SDN cloud playground have plenty of bandwidth capacity except bandwidth capacity are stochastically time varying and also video streaming traffic are smaller amount than available bandwidth capacity. However, the emulated multi-path network in Section 3.1 has implemented with limited path capacity that is one path cannot carry the whole incoming video streams. The standard deviation of packet delay or jitter in using chunk size ratio 20:10 case is higher than that in using 10:20 case. This is because in the case of 20:10, the packets are transmitted for 20 seconds via the Malaysia route which has round trip time (RTT):$\simeq$ 125 ms. However, in 10:20 case, the packets are transmitted for 20 seconds from Korea to Thailand via the direct route which has approximately 0.2% shorter path delay (RTT:$\simeq$ 105 ms) than that via the Malaysia route. When compared to using the single-path scenario with and without the middle-box cases, the packet jitter of using the single-path transmission without the middle-box is approximately zero in most of the experiments while the packet jitter is around 1.12 seconds in the case with the

middle-box. The reason is that, with the middle-box, the time required for packet processing and reordering causes additional delay in delivering packets towards the client playback session. From the results of 20:10 and 10:20 cases, we have noticed that the RTT delay of each path needs to be considered carefully even when there is enough bandwidth capacity for both links. The longer time for packets transmission on the high RTT delay path, the larger the packet jitter at the client side. The packet loss ratio in all experiment scenarios are approximately zero since the link capacity is abundant to carry the selected video streaming traffic. Although the packet loss ratio results show zero in this experiment, there may have performance degradation when we calculate the PSNR values upon the received video quality at the client side due to the performance of SmartX box, middle-box and real-time network condition. In addition, for the comparison of video streaming cases via single-path vs. multi-path transmissions, the results of packet delay over OF@TEIN SDN cloud playground confirm the results of packet delay over the emulated network in Section 3.1. That is, using multi-path video streaming method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video stream due to the increased implementation complexities at both the splitting and combining functionalities. Figures 3.16-3.21 depict the number of delayed packets and the packet delay of single-path and multi-path video streaming without any initial buffering time in the middle-box. These graphs confirm that the delay results are consistently summarized in Table 3.13.

Moreover, we have investigated the effect of initial buffering time in the middle-box by varying initial buffering times to be 0, 10 and 20 seconds for multi-path video streaming of chunk size ratios (20:10 and 10:20). Figure 3.22 depicts the mean/standard deviation of packet delay vs initial buffering times (0, 10 and 20 s) for multi-path video streaming over OF@TEIN SDN cloud playground with chunk size ratios (20:10 and 10:20). According to the Figure 3.22, mean packet delay is increasing with the rate of initial buffering time in both chunk size ratios (20:10 and 10:20) cases because of storing and packet processing delay in the middle-box. In addition, the packet delay, jitter is increasing with directly proportional to the initial buffering time. Unlike the emulated experiments in Section 3.1, the increasing initial buffering time does not help to decrease the packet delay since the bandwidth capacity

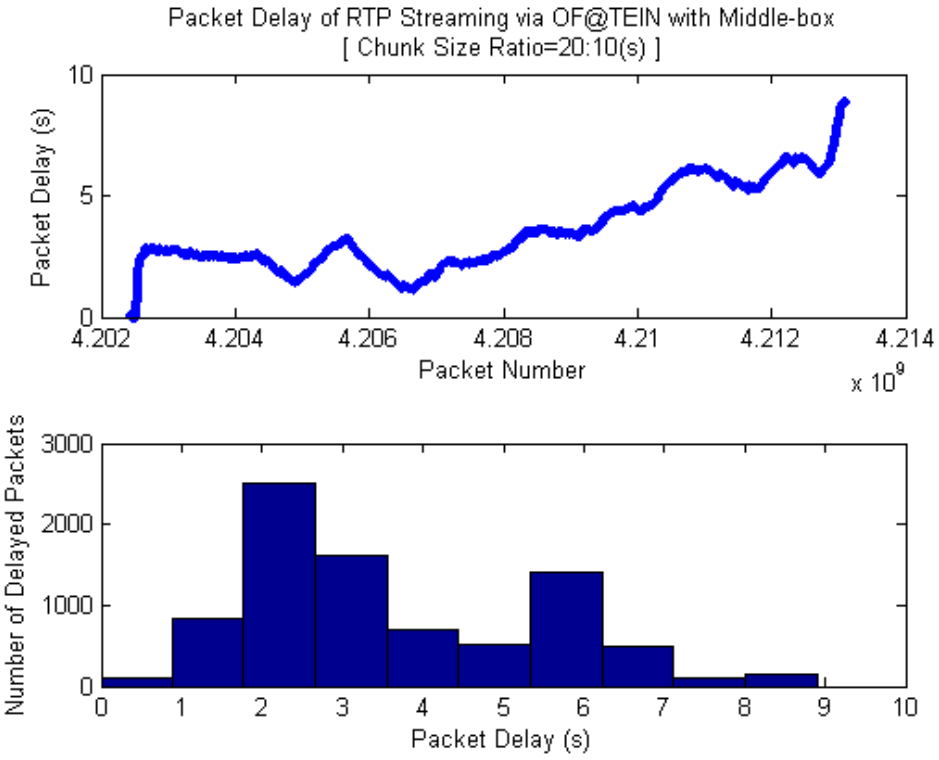


Figure 3.16: Packet delay of RTP streaming via OF@TEIN with middle-box (Chunk size ratio=20:10 (s).

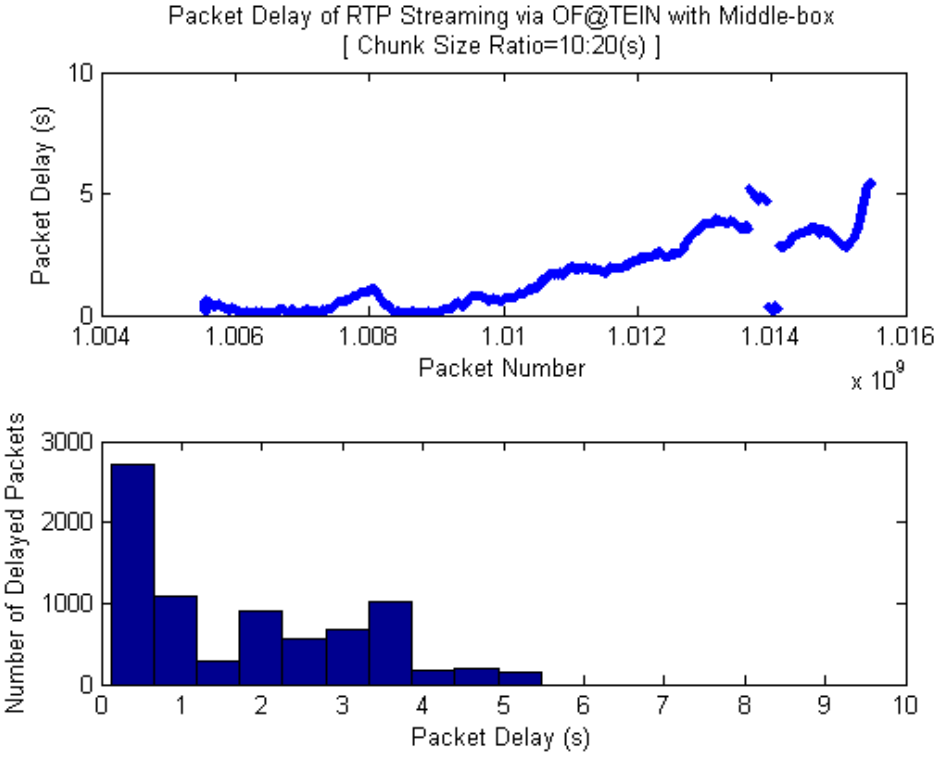


Figure 3.17: Packet delay of RTP streaming via OF@TEIN with middle-box (Chunk size ratio=10:20 (s).

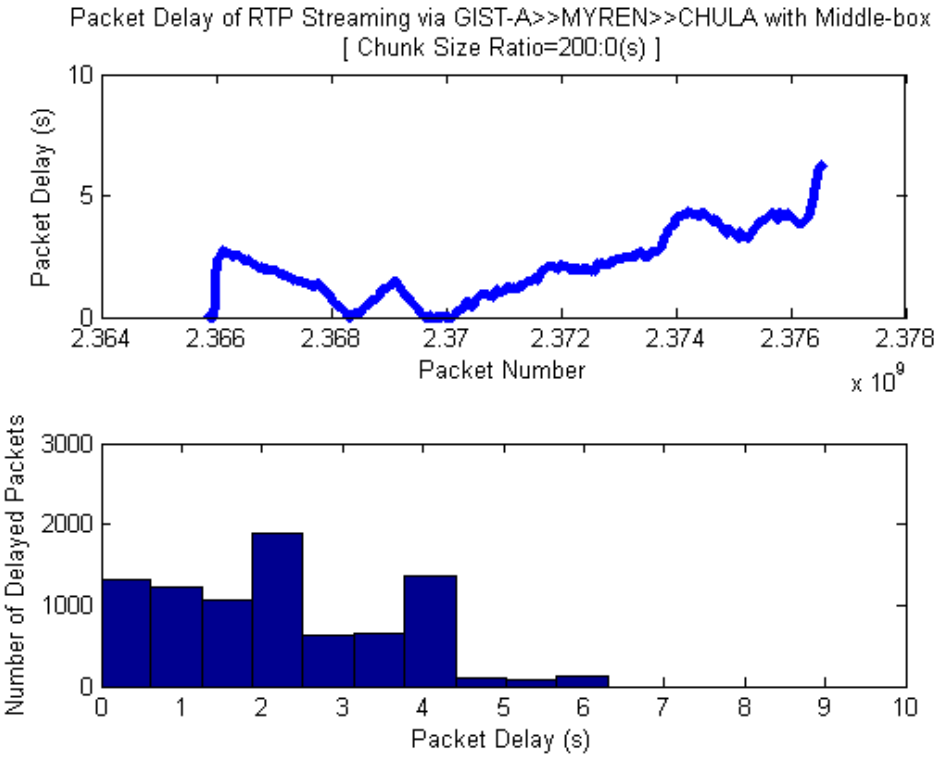


Figure 3.18: Packet delay of RTP streaming via GIST-A ≫ MYREN ≫ CHULA with middle-box (Chunk size ratio=200:0 (s)).

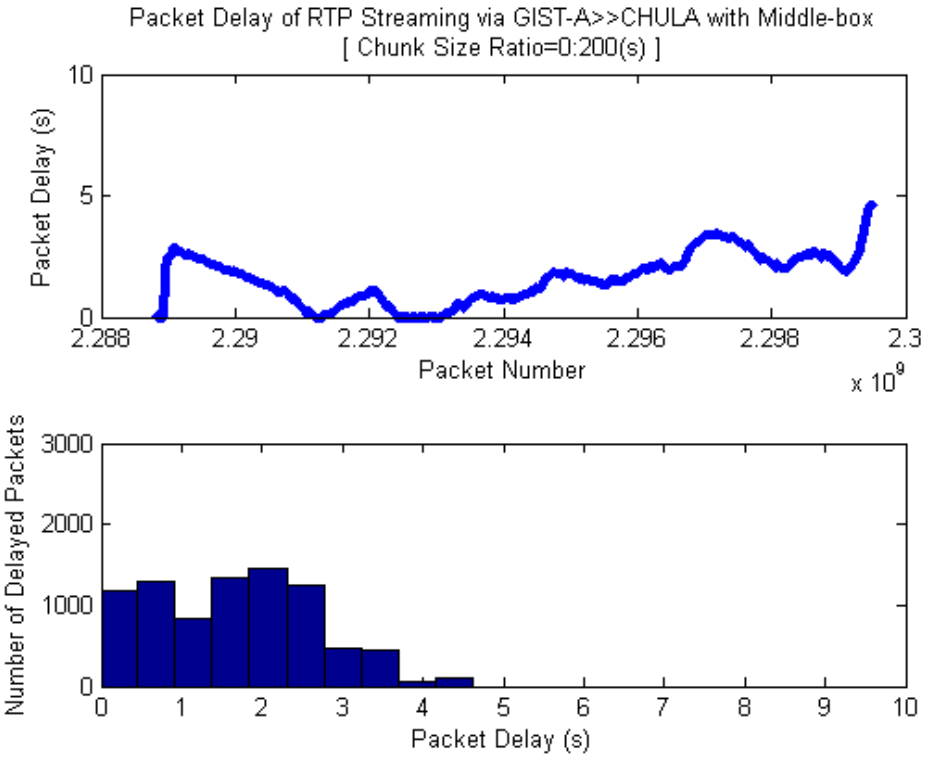


Figure 3.19: Packet delay of RTP streaming via GIST-A ≫ CHULA with middle-box (Chunk size ratio=0:200 (s)).

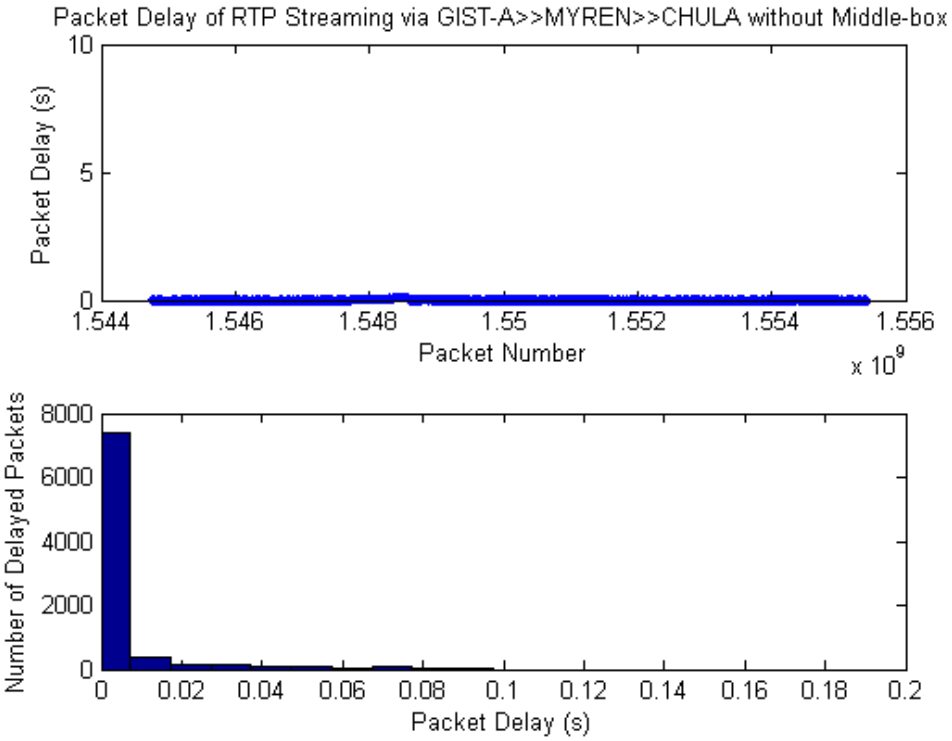


Figure 3.20: Packet delay of RTP streaming via GIST-A ≫ MYREN ≫ CHULA without middle-box.

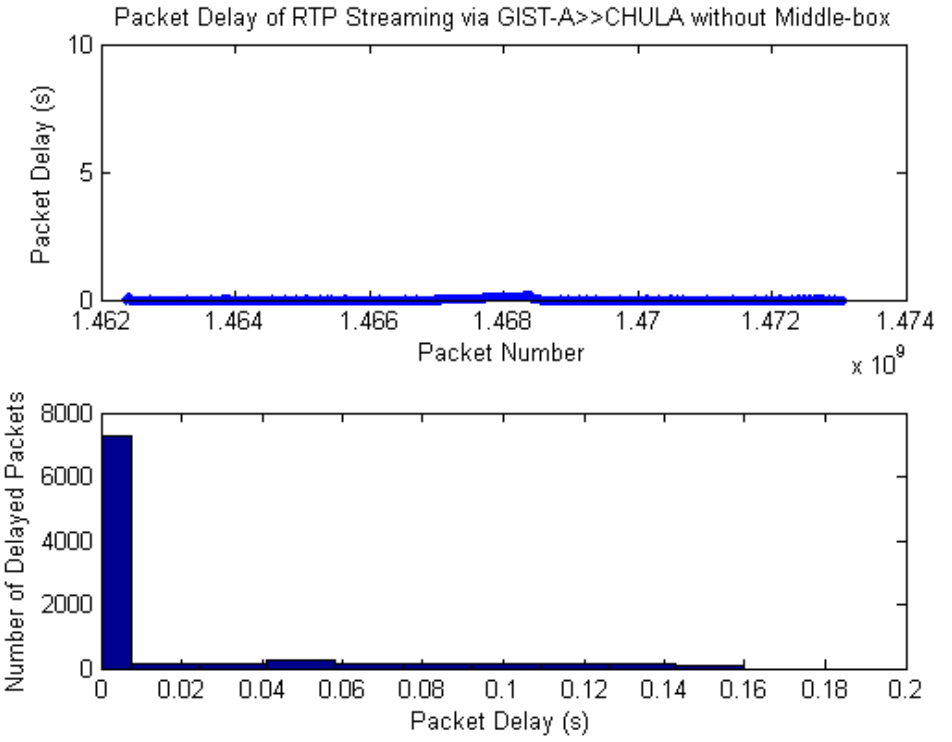


Figure 3.21: Packet delay of RTP streaming via GIST-A ≫ CHULA without middle-box.

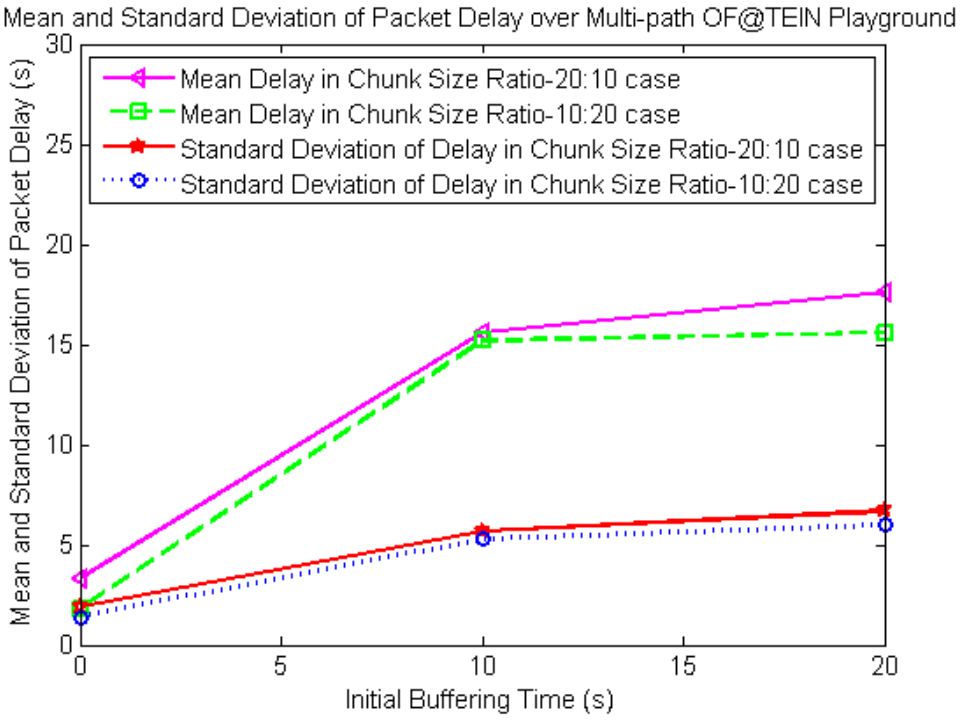


Figure 3.22: Mean and standard deviation of packet delay vs initial buffering time for multi-path streaming over OF@TEIN playground.

is abundant to transmit the selected rate of video packet transmission. Therefore, the results confirm that using initial buffering time and multi-path streaming method in the high-speed bandwidth capacity links make higher packet delay than single-path streaming because the middle-box requires the fast processing speed in order to generate the packets as fast as possible. When considering the implementation of the middle-box in CHULA SmartX box, CPU utilization and RAM are the most important facts to consider in order for not overloading the SmartX box since it is running multiple processing such as OpenStack.

3.2.3 Summary of Multi-path Streaming over OF@TEIN Playground

In this multi-path video streaming experiments, we have implemented the splitting and middle-box functionalities over international OF@TEIN large-scaled network (GIST-A, MYREN and CHULA nodes). The reported packet delay and jitter results in this multi-path streaming experiments over OF@TEIN have confirmed the recommendation from our emulated multi-path video streaming that is 'using multi-path video streaming method is not recommended when the main path capacity already suffices for carrying out the

incoming packets of video stream'. In OF@TEIN SDN cloud playground, both of the selected paths via Malaysia and direct link between GIST-A and CHULA have much more enough bandwidth to carry the video streaming traffics. The one important notice from the reported delay results is that the RTT delay of transmission paths is necessary to consider in order for transmitting packets with low delay. These reported results and information will be used for selecting low delay paths and chunk size ratio when we investigate multi-path file transferring and streaming. The time responsiveness or interactivity of GUI applications in accessing remote OpenStack VMs still needs to be improved. We will try to solve this problem in the later experiments. During experiments over OF@TEIN playground, the longer time has been taken for transferring the experiments data from one country node to another. This becomes our motivation for testing multi-path file transferring experiments over OF@TEIN SDN cloud playground that we will describe in the next section. The implemented middle-box can be used as a processing machine for further experimental setups. However, careful allocation of memory and configurations are required in order to avoid looping and full memory issues of SmartX box. Apart from this consideration, our implemented middle-box can be used with full functionalities in the further experimentation.

3.3 Design of Multi-path Chunked Video File Transferring and Streaming over OF@TEIN SDN Cloud Playground

In this section, the solutions for addressing file transferring delay over international gateway environments will be described. According to the multi-path video streaming experimental results over OF@TEIN SDN cloud playground, we have learnt the best case scenarios of chunk size ratio and requirements for investigating experiments in this video file transferring and streaming. The objective is to find the applicability ranges of splitting functionalities over OF@TEIN SDN cloud playground which has plenty of available bandwidth. TCP file transferring over international links is limited in terms of transfer duration and bandwidth capacity due to the firewall in each node and the distance between nodes. Moreover, in transferring via TCP, all packet losses are counted as congestion and multiple retransmission requests decrease the bandwidth capacity. In order to solve out that kind of issues with TCP, we have selected Tsunami [15] file transfer protocol in these multi-path file transferring and streaming experiments. Tsunami [15] is a combination of UDP and TCP file transfer protocols in which bulk data are transferred via UDP and control data are transferred via TCP. In this section, we will describe the benefits of combining proposed multi-path file transferring method and normal Tsunami file transfer protocol over OF@TEIN SDN cloud playground. In addition, after the completion of file transferring experiments, we would investigate how downloaded video file can be streamed out smoothly within the local area network.

3.3.1 Implementation of Multi-path Chunked Video File Transferring and Streaming Sessions over OF@TEIN SDN Cloud Playground

As in the previous multi-path video streaming experiments over OF@TEIN SDN cloud playground, we have selected three countries: Thailand, Malaysia and Korea. However, different SmartX boxes have been used except Thailand node due to re-designed architecture of OF@TEIN SDN cloud playground. The following SmartX boxes have been used in this multi-path chunked video file transferring and video streaming experiments: GIST-B, MY and CHULA. The overall and detail architectures of multi-path chunked

video file transferring and streaming over OF@TEIN SDN cloud playground are depicted in Figures 3.23 and 3.24.

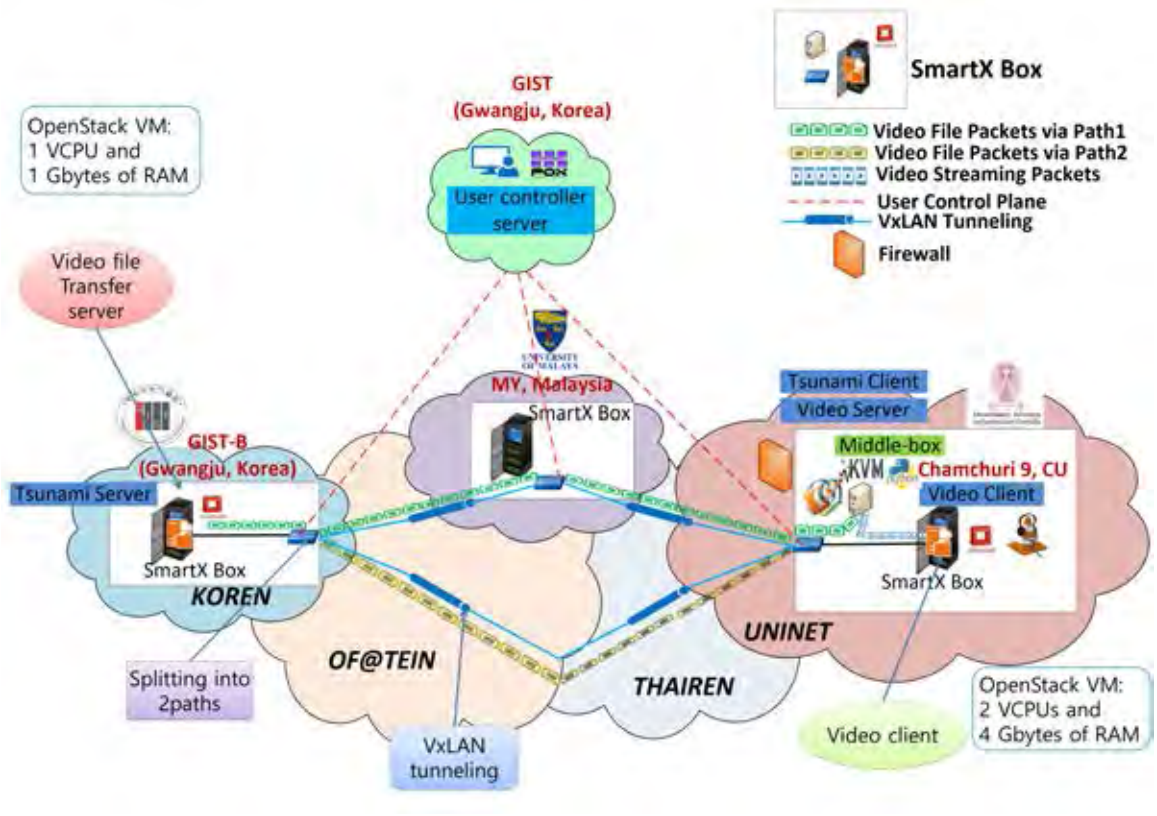


Figure 3.23: Overview of multi-path chunked video file transferring and streaming over OF@TEIN SDN cloud playground.

As described in Figures 3.23 and 3.24, we have created OpenStack VM running Ubuntu 14.04 OS with 1 VCPU and 1 GBytes of RAM in GIST-B OpenStack compute node and OpenStack VM running Ubuntu 14.04 OS with 2 VCPUs and 4 GBytes of RAM in Thailand OpenStack compute node for serving as a Tsunami video file transfer server and a video client respectively. The middle-box with 1 VCPU and 2 GBytes of RAM running Ubuntu 12.04 OS which has been implemented in the previous multi-path video streaming experiments is used for a Tsunami video file transfer client and a VLC video streaming server. OpenStack Nova and Neutron functions are the same as in the Section 3.2 experiments . However, the different design of OVS bridges of each SmartX box has been used in this multi-path file transferring and streaming experiments in order to support OpenFlow 1.0 and OpenFlow 1.3 in OF@TEIN SDN cloud playground. Unlike in the old OF@TEIN SDN cloud playground architecture, FlowVisor has not used in this

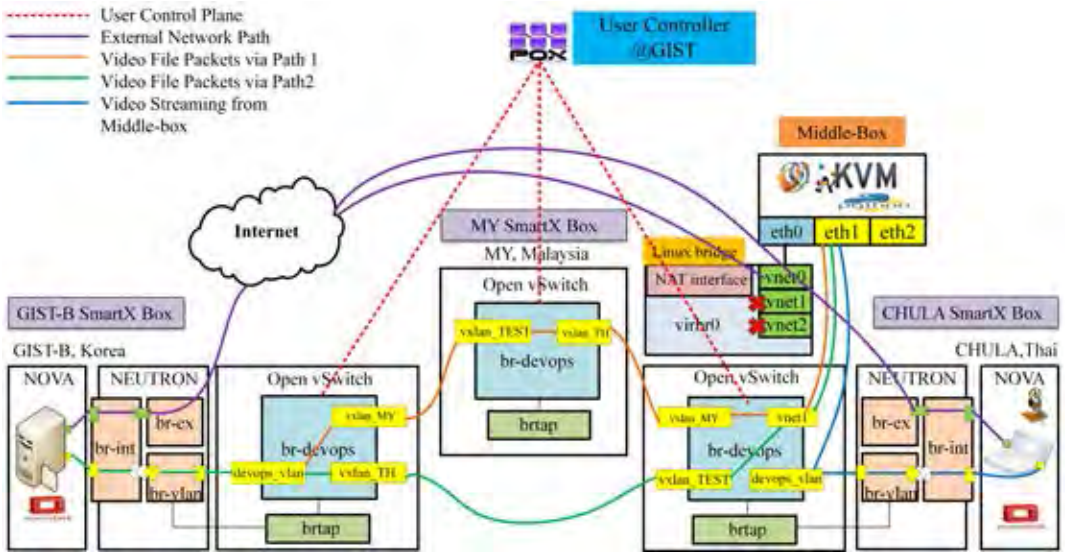


Figure 3.24: Architecture of multi-path chunked video file transferring and streaming over OF@TEIN SDN cloud playground.

re-designed OF@TEIN SDN cloud playground architecture since FlowVisor does not support for OpenFlow 1.3 functions. Therefore, only one user has been allowed to run experiments over new OF@TEIN architecture during the requested time schedule. Therefore, developers need to reserve the timing for running experiments over OF@TEIN. The OVS bridge (br-devops) has been created instead of br1, br2 and brcap which have used in the old OF@TEIN SDN cloud playground architecture. VxLAN tunneling has been used to connect between three br-devops in three SmartX boxes (GIST-B, MY and CHULA) from different countries. There has no separation of user and admin controllers in this new OF@TEIN SDN cloud playground architecture. The POX controller has been used for adding flow entries into OVS bridges (br-devops) at GIST-B, MY and CHULA.

Firstly, the Tsunami video file transfer server sends out the chunked video files into multiple concurrent paths via Path 1 (via GIST-B $\gg$ MY $\gg$ CHULA) and Path 2 (via GIST-B $\gg$ CHULA). The OVS (br-devops) at GIST-B is responsible for splitting chunked video files into Path 1 and Path 2. The OVS (br-devops) at CHULA is responsible for combining the video file packets from two paths and then forwarding video streaming packets from the middle-box to the video client in CHULA OpenStack VM. The middle-box serves as a Tsunami video file transfer client and a VLC video streaming server. The IP address of eth1 in the middle-box have been configured to be the same network and the same VLAN

ID (111) as in OpenStack VMs. VLAN (802.1q) configuration program (vconfig) [58] have been used for adding VLAN ID into eth1 of the middle-box. The MTU size of all eth1 interfaces for GIST-B VM, CHULA VM and the middle-box have been configured as 1410 bytes in order to be able to transmit the iPerf traffic. The port information and flow entries of each SmartX boxes have been used in these experiments are shown in Tables 3.14-3.18.

Table 3.14: Port information for SmartX boxes (GIST-B, MY and CHULA)

SmartX boxes	br-devops ports
GIST-B(103.22.221.31)	1(devops_vlan) 2(vxlan_MY) 3(vxlan_TH)
MY(203.80.21.4)	2(vxlan_TEST) 3(vxlan_TH)
CHULA(161.200.25.99)	1(devops_vlan) 2(vxlan_TEST) 3(vxlan_MY) 5(vnet1) 7(vnet2)

Table 3.15: Flow entry of GIST-B SmartX box (OVS: br-devops) for transmission via path 1 (GIST-B $\gg$ MY $\gg$ CHULA) [10]

Header field	Action	Timeout
in_port = 1(devops_vlan)	output:2(vxlan_MY)	hard_timeout = m
in_port = 2(vxlan_MY)	output:1(devops_vlan)	hard_timeout = m

Table 3.16: Flow entry of GIST-B SmartX box (OVS: br-devops) for transmission via path 2 (GIST-B $\gg$ CHULA) [10]

Header field	Action	Timeout
in_port = 1(devops_vlan)	output:3(vxlan_TH)	hard_timeout = n
in_port = 3(vxlan_TH)	output:1(devops_vlan)	hard_timeout = n

As shown in Tables 3.15 and 3.16, the video file packets are periodically chunked into smaller files at GIST-B (br-devops) by specifying the m s to parameter hard_timeout of the flow entry for packet transmission via Path 1 and n s to parameter hard_timeout for that via Path 2. The ratio m:n then determines the chunk size ratio in this splitting function as in the emulated and OF@TEIN multi-path video streaming experiments.

Table 3.17: Flow entry of MY SmartX box (OVS: br-devops)

Header field	Action
in_port=2(vxlan_TEST)	output:3(vxlan_TH)
in_port=3(vxlan_TH)	output:2(vxlan_TEST)

The flow tables at MY OVS (br-devops) as shown in 3.17 are simply that when the packets arrive from vxlan_TEST (GIST-B), those packets will be forwarded to vxlan_TH. When the packets arrive from vxlan_TH, those packets will be forwarded to vxlan_TEST.

Table 3.18: Flow entry of CHULA SmartX box (OVS: br-devops)

Header field	Action
in_port = 3(vxlan_MY)	mod_dl_src:c2:ff:3e:21:81:d4, output:5(vnet1)
in_port = 2(vxlan_TEST)	mod_dl_src:f2:a8:b7:ea:85:63, output:5(vnet1)
in_port = 5(vnet1)	ALL
in_port = 1(devops_vlan)	output:5(vnet1)

The flow entries of br-devops in CHULA SmartX box which is shown in Table 3.18 are similar to the flow entries of br2 as described in the previous multi-path video streaming over OF@TEIN SDN cloud playground experiments. The br-devops in CHULA SmartX box is responsible for combining the video file packets from two paths and forwarding the video streaming packets from the middle-box to the video client in CHULA OpenStack VM. When the packets arrive from the ingress vxlan_MY and vxlan_TEST ports, the packet headers are set to be c2:ff:3e:21:81:d4 for packets coming from vxlan_MY and f2:a8:b7:ea:85:63 for packets coming from vxlan_TEST and then forwarded to the vnet1 for receiving Tsunami video file client in the middle-box. After completely receiving the video file packets in the middle-box, the middle-box serves as a VLC streaming server. For the packets arrive from the vnet1 ingress port, all packets will be forwarded to all available ports at br-devops except the ingress port vnet1. Therefore, the network of the middle-box is reachable to both GIST-B and CHULA OpenStack VMs. The packets arrive from devops_vlan will be forwarded only to vnet1 so that CHULA OpenStack VM data path network is reachable to the middle-box. The reason is that we do not need to transmit any traffic between GIST-B OpenStack VM and CHULA OpenStack VM in this experiment. Therefore, we cut off the routing between GIST-B OpenStack VM and CHULA OpenStack VM in order to save the unnecessary bandwidth usages.

Another implementation to solve the fast time responsiveness of GUI applications in accessing remote OpenStack VMs over OF@TEIN SDN cloud playground have been introduced in this section. As mentioned in the previous section about limited access of GUI applications with X11 display over OF@TEIN OpenStack VMs is limited for streaming video with large video resolution size. In order to solve out that issues, we have implemented the X11 Desktop Environment in OpenStack VMs by using light weight X11 desktop environment (LXDE) [59] which supports fast desktop performance for easy access GUI applications in the cloud. Moreover, in order to remote access to the implemented Desktop Environments, we have installed an open source remote desktop protocol(rdp) server called xrdp [60], a free remote control software called tightvnc [61] and an open source implementation of the X Window system called xorg [62]. There are three access methods for OpenStack VMs: (1) access via xrdp without requiring port information, (2) access via tightvnc with requiring 5901 port access and (3) access via ssh as in the old method with X11 display. For accessing method 1 and 2, we can use either remote desktop connection with Window OSs or rdesktop with Linux/Ubuntu OSs. The light weight X11 desktop environments (LXDE) of GIST-B and CHULA OpenStack VMs by accessing rdesktop with Ubuntu OS are depicted in Figure 3.25.

Figure 3.25: Light weight X11 desktop environments of GIST-B and CHULA OpenStack VMs.

3.3.2 Results and Discussion of Multi-path Chunked Video File Transferring and Streaming over OF@TEIN SDN Cloud Playground

The Big Buck Bunny [63], a 4k animation video with a total file size of 843 Mbytes, resolution 3840 x 2610 and the duration of 10 minutes has been used in this multi-path chunked video file transferring and streaming experiments. Tsunami file transfer application [15] has been installed on both GIST-B OpenStack VM and middle-box VM in CHULA SmartX box. Since the purpose of this experiment is to investigate the effects of combining our proposed multi-path splitting function and traditional Tsunami file transfer protocol, we have not modified the default codes of Tsunami application.

The Tsunami server in GIST-B OpenStack VM starts the Tsunami server by using the command "tsunamid" as shown in Figure 3.26. The middle-box VM in CHULA SmartX box receives the video file by using the command "tsunami" as shown in Figure 3.27. The parameters for Tsunami file transfer protocol are the default block size: 1024 bytes (how large UDP blocks to use) and buffer size: 20 Mbytes (size of ring buffer in RAM) in this experiment. Tsunami protocol allows a client to choose many parameters such as block size, buffer size, target file transfer rate, error threshold, and inter-packet delay. However, in this experiment, we vary only target file transfer rates in order to investigate the effects of varying target file transfer rates on transfer duration. After completely received the video file in the middle-box, Tsunami protocol generates the analytic results of file transferring. The sample analytic results at Tsunami client is depicted in Figure 3.28. Among various output results from Tsunami analytic results, transfer duration, file data, throughput and final fie rate have been used for analysing our multi-path chunked file transferring experiments. After completely transferring a video file in the middle-box, the middle-box serves as a VLC streaming server to stream out the downloaded video towards CHULA OpenStack VM in order to investigate the performance of the downloaded video stream.

As in the previous experiments, iPerf UDP test has been performed to check the available bandwidth between GIST-B, MY and CHULA nodes. Unlike in the previous OF@TEIN playground environment (GIST-A, MYREN and CHULA), the bandwidth capacities of around 422 Mbps between OpenStack VM at GIST-B and middle-box VM at CHULA

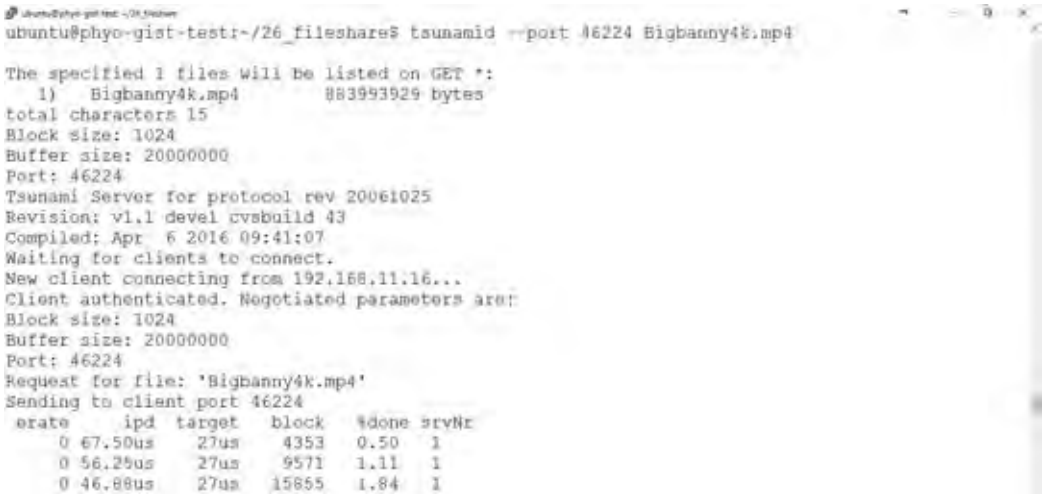


Figure 3.26: Tsunami file transfer server in GIST-B OpenStack VM.

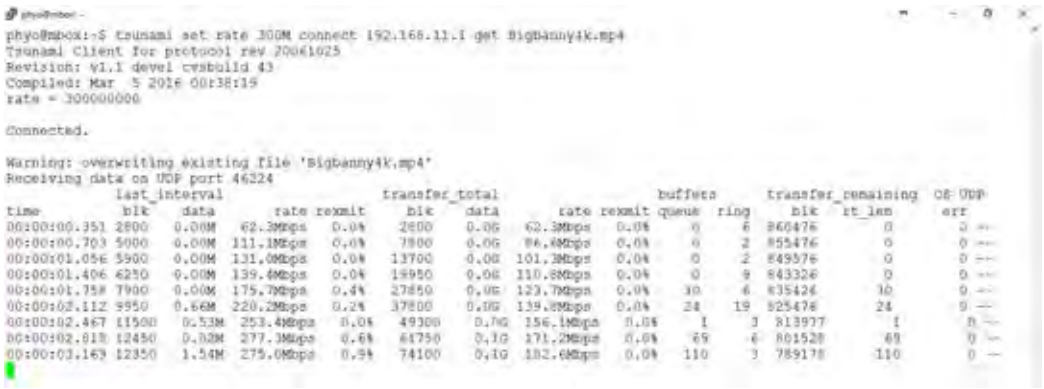


Figure 3.27: Tsunami file transfer client in middle-box VM.

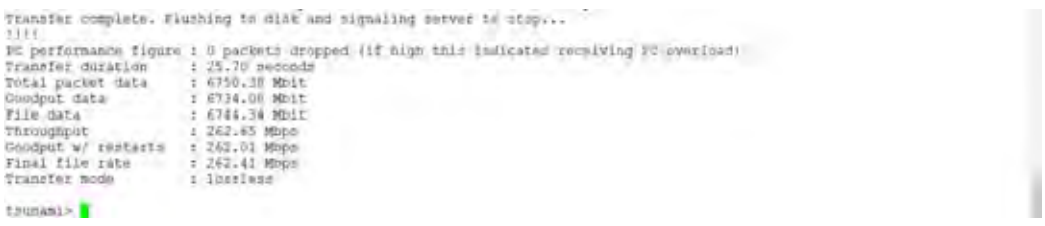


Figure 3.28: Sample analytic output results by Tsunami client.

via Path 1 (GIST-B ≫ MY ≫ CHULA) and around 456 Mbps via Path 2 (GIST-B ≫ CHULA) have been obtained from the test. As investigated in the previous multi-path video streaming experiments over OF@TEIN playground, round trip time delay (RTT) via Path 1 and Path 2 are the same. The RTT via the MY route is approximately 125 ms and the RTT delay from GIST-B to CHULA via the direct route is about 105 ms. Therefore, the best case scenario (transmitting long period via Korea and Thailand direct link) of chunk size ratio obtains from the previous multi-path video streaming experiments has been used for this multi-path file transferring and streaming experiments.

Three scenarios have been tested for this multi-path file transferring experiments. They are (1) multi-path with chunk size ratio (1:2 sec) which is periodically splitting by transmitting 1 second via Path 1 and transmitting 2 seconds via Path 2 (2) using Path 1 alone (GIST-B ≫ MY ≫ CHULA) (3) using Path 2 alone (GIST-B ≫ CHULA). In order to investigate the effects of target file transfer rate on the transfer duration, we have varied the target file transfer rates of Tsunami protocol to be 100, 200, 300, 400 and 500 Mbps and have tested three times with the same parameter settings for all three selected scenarios. The file transfer duration can be computed from the following formula.

$$Transfer\ Duration = \frac{Total\ File\ size}{Actual\ File\ Transfer\ Rate}$$

where: $Transfer\ Duration$ = File transfer duration between server and client in seconds (s),

$Total\ File\ size$ = File size in bits (bits) and

$Actual\ File\ Transfer\ Rate$ = Rate of file transmission in bits per second (bps).

Transfer duration results of transferring 843 Mbytes (6744Mbits) video file by using three scenarios: multi-path (1:2sec), using Path 1 alone (GIST-B ≫ MY ≫ CHULA) and using Path 2 alone (GIST-B ≫ CHULA) are shown in Figure 3.29. According to the tested results, file transfer duration decreases with upper bound to the available link bandwidth capacity when the target file transfer rate of Tsunami protocol increases for all three scenarios. That result trend can be seen when comparing the cases of target file transfer rates (100, 200, 300 and 400 Mbps). The required transfer duration of using Path 2 alone (GIST-B ≫ CHULA)

and multi-path (1:2sec) are similar until the target file transfer rate is up to 300 Mbps when the link bandwidth capacity is enough to carry the whole video file traffic. As for using Path 1 alone (GIST-B $\gg$ MY $\gg$ CHULA), the required file transfer duration is higher than the other two scenarios, although there has enough link capacity to carry the whole traffic. The reason is because the longer round trip time (RTT):$\simeq$ 125 ms require for transmitting via Path 1 (via MY route) while the RTT delay via Path 2 (via a direct link to CHULA) require $\simeq$ 105 ms. Moreover, the RTT delay of multi-path (1:2sec) is periodically switching between 125 ms and 105 ms. Therefore, the RTT delay is important to consider in order to obtain the lower transmission delay. In the case of target file rate 400 and 500 Mbps where the link becomes congested, the file transfer duration results of our proposed multi-path (1:2sec) case outperform the using Path 1 alone and Path 2 alone cases. However, the target file transfer rate of 500 Mbps is not recommended to use with our multi-path function because it requires a longer delay than those of 400 Mbps rate. In that case, UDP packet losses increase due to the overloaded links. The reason that the file transfer duration of our proposed multi-path (1:2sec) case cannot outperform in the cases of target file transfer rate 100, 200, 300 Mbps is because using Tsunami file transfer protocol limits the maximum transfer rate. So that the transfer rate cannot be more than the specified target rate when using our multi-path function. The tested results of file transfer duration confirm that using multi-path splitting function achieves the lowest file transfer duration time when the links are congested and not enough to carry the whole traffic by using single path.

Figures 3.30 and 3.31 depict the actual file transfer rate and file transfer throughput for three scenarios with various target file transfer rates. Those throughput results confirm the results of file transfer duration as discussed in the above. The file transfer throughput is the available throughput during the file transferring period. According to the file transfer throughput results in the case of 400 and 500 Mbps target file transfer rates, the throughput of using multi-path function is higher than that in using single-path alone. However, those file transfer throughput results have not been used for calculating file transfer duration results. In order to calculate the file transfer duration results, the actual file transfer rate results as shown in Figure 3.30 have been used. According to the actual file transfer results, the highest actual file transfer rate achieves in the case of using 400 Mbps target file rate

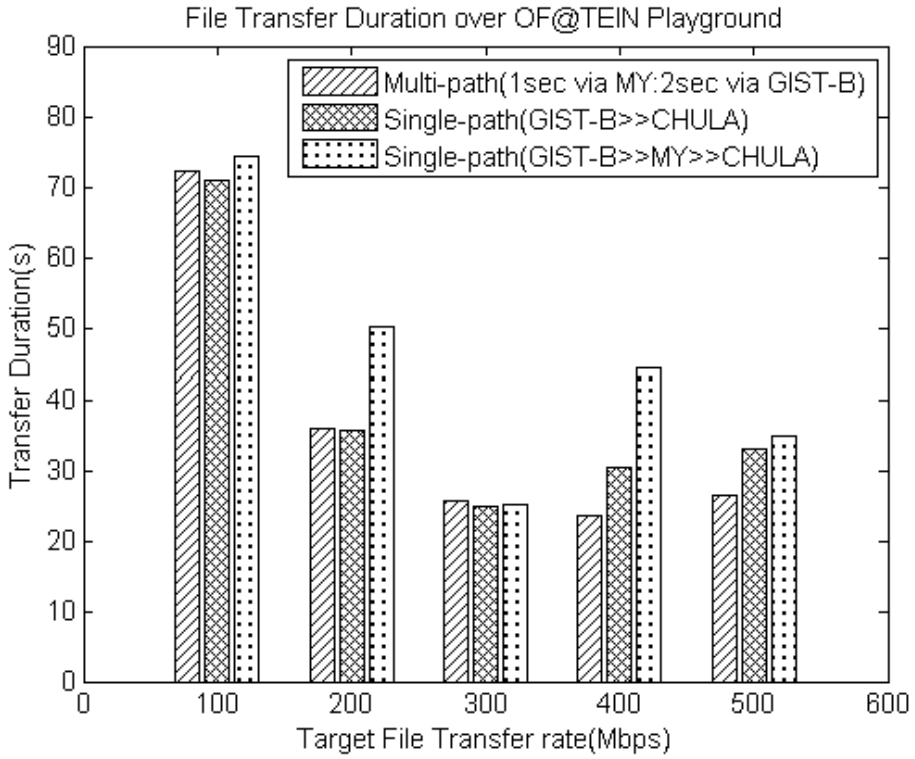


Figure 3.29: File transfer duration over OF@TEIN SDN cloud playground.

with multi-path scenario which combines the bandwidth of two paths periodically. In the cases of 100, 200, 300 Mbps, we recommend to use only Path 2 (GIST-B $\gg$ CHULA) in order to avoid configuration complexity. Moreover, using the multiple network links can lead to higher operating expenses (OPEX). However, multi-path function is recommended to use in order to obtain the highest file transfer rate when the single-path alone is not enough to carry out the whole video file traffic. We can see that trend from the case of target file transfer rate 400 Mbps.

After completely received the video file in the middle-box VM, the middle-box VM serves as a VLC streaming server for CHULA OpenStack VM. This experiment is in order to evaluate the performance of downloaded video file and to stream out smoothly within CHULA SmartX box network. The Big Buck Bunny H. 264 video codec with video and audio mean bit rate of total 11133kbits/sec with resolution 3840 x 2610 and duration of 10 minutes, has been used for streaming from the middle-box VLC server to the video client in CHULA OpenStack VM. The VLC player from the video server has been configured to stream out video by using RTP mode. The VLC GUI interface has been used in order to

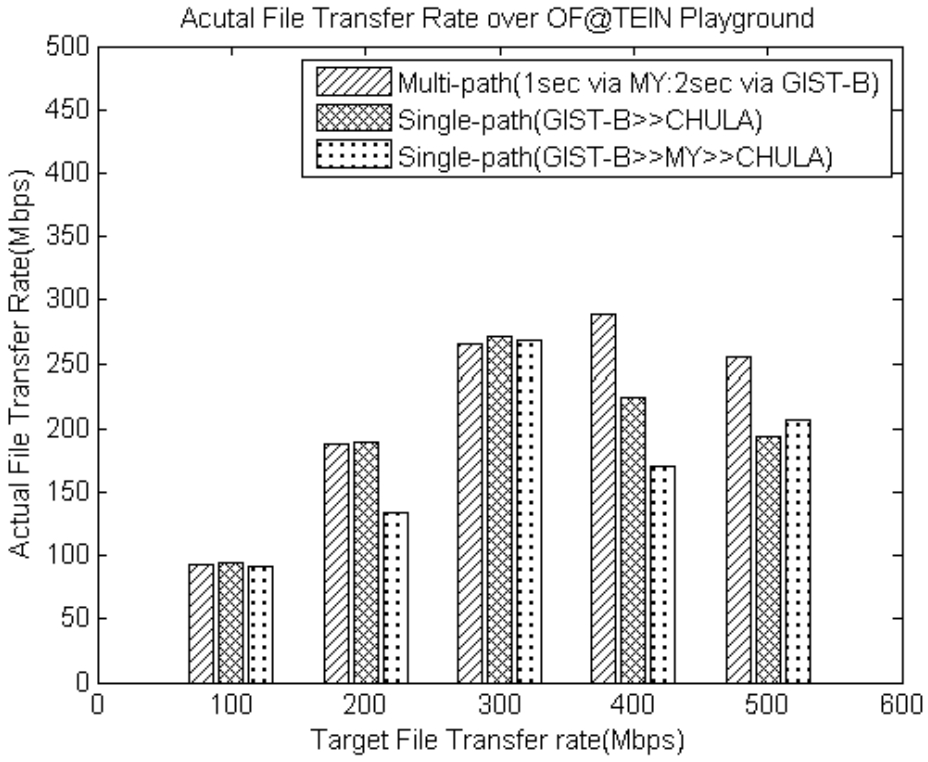


Figure 3.30: Actual file transfer rate over OF@TEIN SDN cloud playground.

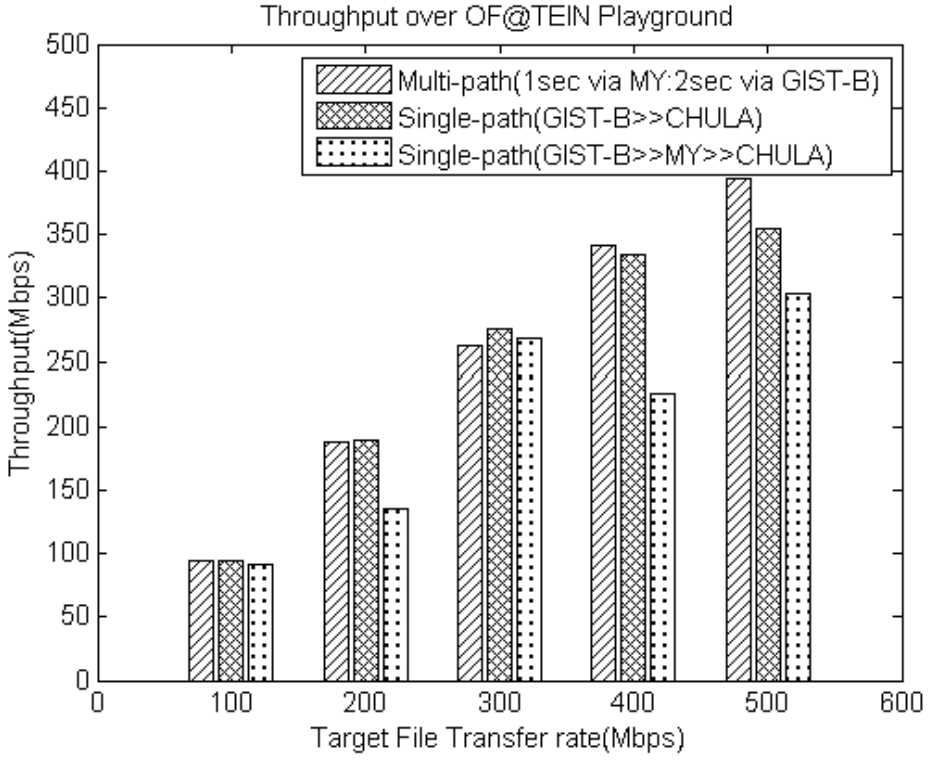


Figure 3.31: File transfer throughput over OF@TEIN SDN cloud playground.

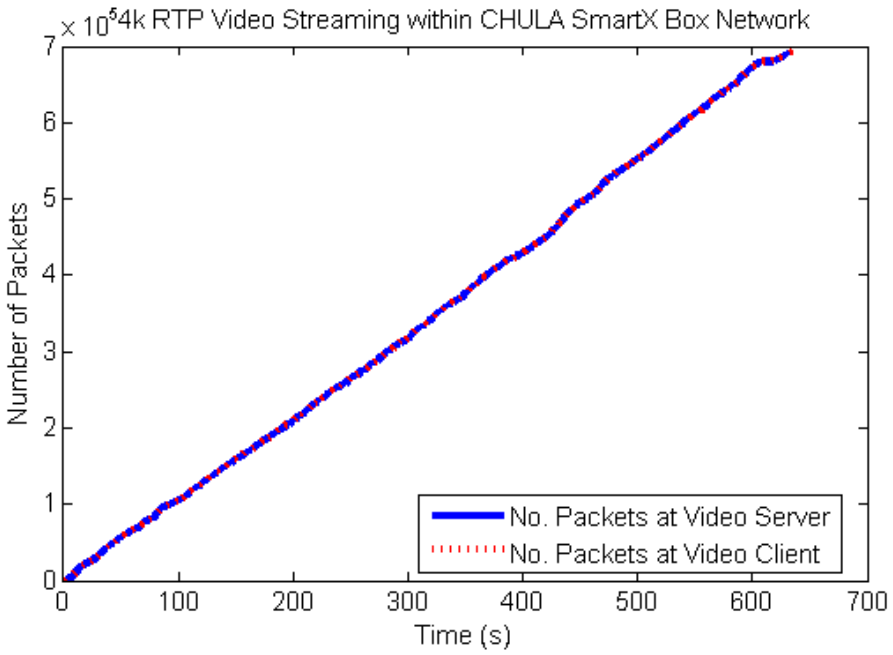


Figure 3.32: 4k RTP video streaming within CHULA SmartX box network.

stream out with original video tracks. The video packets are captured at eth1.111 of the middle-box VM and eth1 of CHULA OpenStack VM by using tcpdump packet analyzer. To investigate the performance of video streaming, we have run three times for the same video streaming session. For the detailed dynamics of packet generation arriving at the video server and the received packet stream departing at the video client, we include here the arrival and departure curve in Figure 3.32. The number of packets shown in the graph is the number of packets transmitted during video streaming. The number of packet losses is zero in all tests. This curve confirms that 4k RTP video streaming within CHULA SmartX box network can stream out without the extra delay.

3.3.3 Summary of Multi-path Chunked File Transferring and Streaming over OF@TEIN SDN Cloud Playground

In this multi-path video file transferring and streaming experiments, we have implemented the X 11 desktop environment and access method for OpenStack VMs in order to use the GUI applications with fast access. In addition, we have tested the combination of traditional Tsunami protocol and proposed multi-path file transferring function. According to the tested results with three scenarios: multi-path (1:2 sec), using

Path 1 alone (GIST-B ≫ MY ≫ CHULA) and using Path 2 alone (GIST-B ≫ CHULA), the proposed multi-path file transferring method can be transferred with the minimum transmission delay when the links are congested and not enough to carry the whole video file traffic. Moreover, using multi-path function can be achieved the maximum actual file transfer rate and throughput when the selected target file transfer rate of Tsunami protocol lead to be congested on the available network links. However, this multi-path file transferring method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video file traffic and the main path already having a lower RTT delay than other available paths due to the results of target file transfer rates (100, 200, and 300 Mbps) by using Path 2 alone. As for the links with higher RTT delay, our multi-path file transferring method can be beneficially applied in order to transfer with lower transmission delay, higher file transfer rate and throughput. Therefore, the RTT delay is important to consider in order to transmit with the low transmission delay.

Moreover, the tested local video streaming results confirm that our implemented testbed can stream out the 4k resolution video within CHULA SmartX box network by using X11 desktop environment. However, we observe one fact that the VLC application itself has some limitations to play back video with very high resolution, even in normal playing back without a streaming session. Therefore, when streaming out 4k resolution video with cloud playground, it is recommended to adjust the video resolution scale to be around 50% (eg. resolution:1630x937) lower than the normal 4k resolution scale.

3.4 Design of Adaptive-path Chunked Video Streaming over OF@TEIN Wired and Wireless Links

Video streaming over Wi-Fi network is one of the most popular applications in today's internet network and its need the seamless network connectivity. However, traditional Wi-Fi network is still having the limitation on mobility management. Wi-Fi handover delay time can be increased the video freezing period so that performance of video can be decreased. Even video streaming over the local wired network and wireless networks are having many challenging issues, video streaming between two different countries through wired and wireless networks require more advanced techniques. To investigate these challenges, we have introduced the adaptive-path chunked video streaming with chunked video pre-transferring mechanism over OF@TEIN international wired and wireless links in this section. The objective of this experiment is to introduce the combination of cloud video streaming service over the international OF@TEIN wired links and software-defined wireless links at OpenFlow@Chula-EE lab-scaled SDN wireless testbed. The main functionality of chunked video pre-transferring mechanism during the handover process for resulting less video freezes and reducing resource consumption of wireless network have been explained in the following section.

3.4.1 Implementation of Adaptive-path Chunked Video Streaming over OF@TEIN Wired and Wireless Links

In this experiment, the video server and video client are located in different countries nodes: GIST-B (Korea) and Chula (Thailand), respectively. The SSIDs of two OpenFlow-enabled access points (APs) are namely SDN-AP1 and SDN-AP2. sFlow-RT [64] for monitoring real-time traffic visibility tool has been used in order to monitor the real-time traffic of two OpenFlow-enabled APs. Two PCs with running Ubuntu OS at Telecommunication System Research Laboratory (TSRL), 13th Floor, Engineering Building 4, Chula have been used as sFlow monitoring servers. The physical APs location, station (STA) machine, remote and monitoring machine are shown in Figure 3.33

Figure 3.33: Physical location of APs and machines for adaptive-path chunked video streaming over OF@TEIN wired and wireless links.

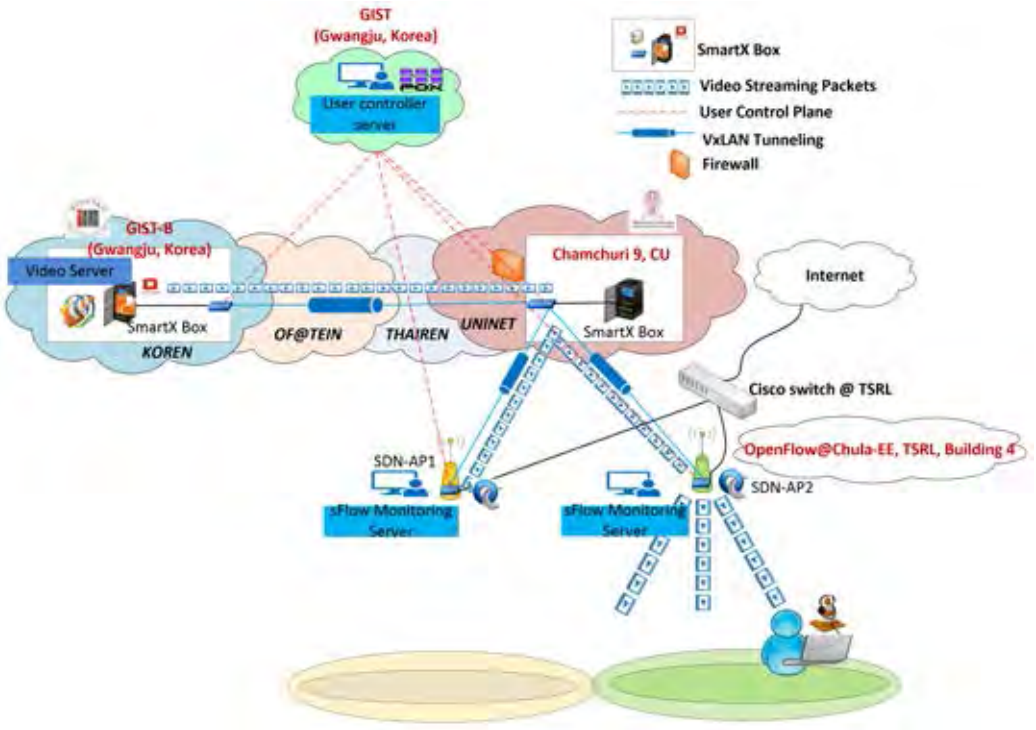


Figure 3.34: Overview of adaptive-path chunked video streaming over OF@TEIN cloud playground with wired and wireless links.

The overview of adaptive-path chunked video streaming over OF@TEIN wired and wireless links with chunked video pre-transferring mechanism is shown in Figure 3.34.

The architecture of this adaptive-path chunked video streaming over OF@TEIN wired and wireless links includes 4 OVS bridges which are GIST-B (br-devops), CHULA (br-devops) and 2 OpenFlow-enabled access points (ovs-ap1 and ovs-ap2). The implementation of wireless testbed consists of two TPLINK-TL 1043ND V2.1 (2.4 GHz, 450 Mbps, IEEE 802.11 b/g/n) routers which are running OpenWrt Chaos Calmer 15.05 with Linux operation system. OpenWrt [56] is one of the most popular Open Source Third party firmwares among wireless SDN researchers. Both access points (APs) operate on Channel 11, IEEE 802.11 n and 20 dbm (100mW) transmit power. As a trial of the combination of OF@TEIN international wired testbed and OpenFlow@Chula-EE lab-scaled SDN wireless testbed, security and authentication systems have not been considered in this implementation and only one user has been allowed to use OpenFlow@Chula-EE lab-scaled SDN wireless network. All APs have been operated on Open Access without the encryption key. We have installed Open vSwitch software (version 2.3.90) to enable OpenFlow rules into our routers. These two APs are connected with Open vSwitch in CHULA SmartX box via VxLAN tunneling. The automatic configuration shell script files for configuring OVSs in APs have been developed. In order to configure wireless network to be OpenFlow-enabled network, the modification of network, wireless and firewall files inside OpenWrt system have been required. WAN ports (eth0) of APs have been used for management, VxLAN tunneling and monitoring the real-time traffic via sFlow-RT. In order to keep alive the packets forwarding function in OVSs when controller fails, OVSs have been configured in 'fail-safe-mode (standalone)'. The installation and configuration steps for OpenFlow-enabled routers and VxLAN configuration will be described in the Appendix G. Both APs have been placed at Telecommunication System Research Laboratory (TSRL), 13th Floor, Engineering Building 4 and the distances between two APs are about 27 m (90ft). Since two APs are located very closed, we have removed two antennas from APs and have used only one antenna in order to perform the handover process. A POX controller Python script for controlling wired and wireless networks and on-request dynamic routing functions have

been developed. The details architecture of the combining OF@TEIN wired network and OpenFlow@Chula-EE lab-scaled SDN wireless testbed adaptive-path chunked video streaming is shown in Figure 3.35.

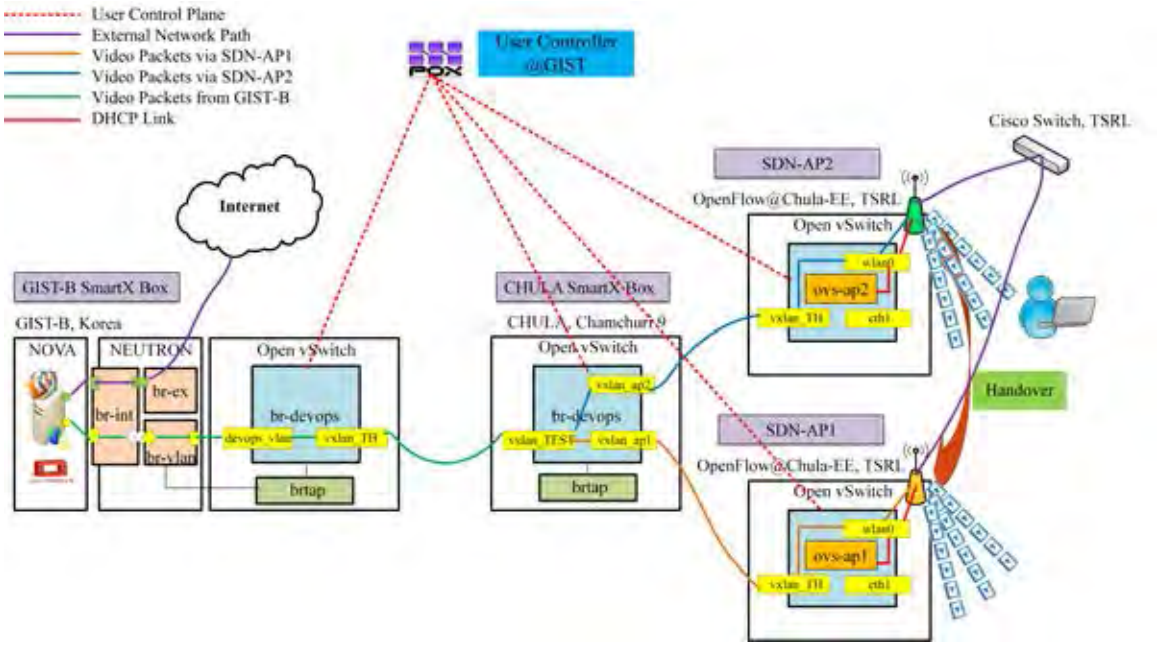


Figure 3.35: Architecture of adaptive-path chunked video streaming over OF@TEIN cloud playground with wired and wireless links.

As shown in Figure 3.35, 4 OVSs including GIST-B (br-devops), CHULA (br-devops) and 2 OpenFlow-enabled APs (ovs-ap1 and ovs-ap2) are controlled by POX controller in which the controller server is located at GIST (Korea). The video streams out from the OpenStack video server (Virtual Machine) at GIST-B later on will be referred as SVR (server) to the video client (Laptop with Ubuntu OS) namely STA (station). The STA Laptop has been used with two WLAN cards (Intel Ultimate N WiFi Link 5300 and Ralink 802.11 n USB WLAN) at TSRL to connect OF@TEIN wired networks and lab-scaled Openflow@Chula-EE wireless networks. Two APs have been configured to be the same network and sub-netmask as in GIST-B OpenStack VM which is the IP address 192.168.11.1/24 for GIST-B OpenStack VM with VLAN ID (111), SDN-AP1 (192.168.11.2/24) and SDN-AP2 (192.168.11.3/24) in which OVSs (ovs-ap1 and ovs-ap2) of APs have been configured as DHCP servers.

Four scenarios have been evaluated for this adaptive-path chunked video streaming over OF@TEIN wired and wireless links experiments. They are (1)adaptive-path chunked

video streaming using single WLAN with 100% duplication (2) adaptive-path chunked video streaming using dual WLANs with 100% duplication (3) adaptive-path chunked video streaming using dual WLANs with chunked video pre-transferring mechanism and (4) adaptive-path chunked video streaming using dual WLANs without chunked video pre-transferring mechanism. The timing diagrams for adaptive-path chunked video streaming over OF@TEIN wired and wireless links experiments are as shown in the following Figures.

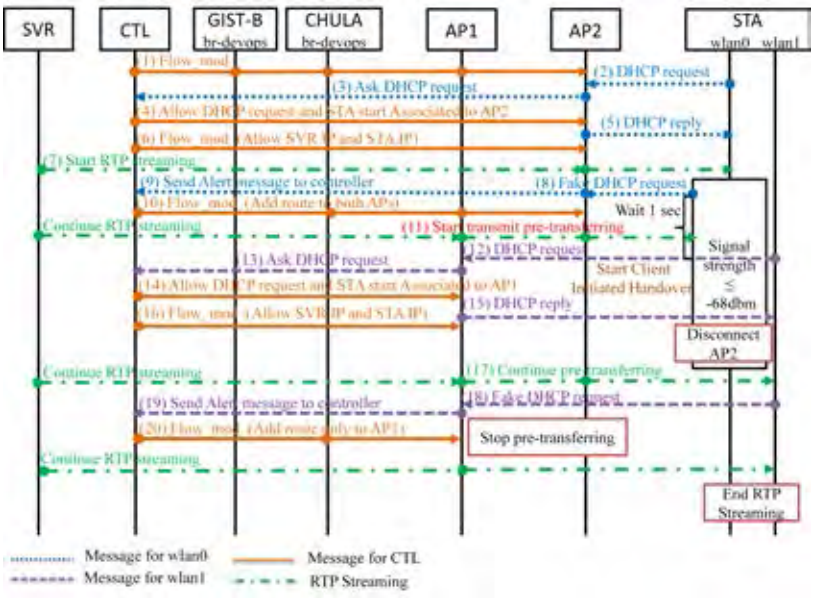


Figure 3.36: Timing diagram of adaptive-path chunked video streaming using dual WLANs with chunked video pre-transferring mechanism.

The timing diagram for proposed adaptive-path chunked video streaming over OF@TEIN wired and wireless links with chunked video pre-transferring mechanism is shown in Figure 3.36. There are 7 components in this diagram which are SVR (GIST-B OpenStack VM: Video Server), CTL (POX controller), GIST-B SmartX box, CHULA SmartX box, two OpenFlow-enabled APs and STA (Station for video client). A total 20 signalling processes need to be done for this adaptive-path chunked video streaming with chunked video pre-transferring mechanism. Firstly, the POX controller at GIST starts adding the flow entries to GIST-B (br-devops) for forwarding packets to CHULA (br-devops). As for CHULA (br-devops), the controller starts add the flow entries for forwarding packets only to AP2 since client always starts associating with AP2 in every test scenario in order to simplify testing. For AP1 and AP2, before STA does not start

associating with them, the controller adds flow entries only for LOCAL port to wlan0 port. (1) Since STA has two WLAN interfaces (wlan0 and wlan1), we have used wlan0 interface to connect to AP2 and wlan1 for connecting with AP1. Both wlan0 and wlan1 have been configured to have the same MAC addresses in order to obtain the same IP address assignment from both AP1 and AP2. Once STA starts associating with AP2, it will send the DHCP request to AP2 and then AP2 forwards this DHCP request to controller for allowing requested IP address (2). After reception of the DHCP request from AP2 at the controller via PacketIn message, the controller adds the flow entries for allowing STA IP address and GIST-B OpenStack IP address to be reachable each other. In this stage, VLAN ID 111 header has been added to the packets arrive from STA using mod_vlan_id action and strip_vlan action for the packets arrive from GIST-B OpenStack VM via vxlan_ap2. (3-6)

After STA has been completely associated with AP2, SVR starts streaming out the video towards STA via AP2 (7) and also STA starts run the client initiated handover shell script. In this experiment, the client initiated handover and on-request dynamic routing functions have been used in order to perform handover testing with less video freezes. To perform the client initiated handover and on-request dynamic routing functions, a shell script file for STA to scan the wireless signal level in every second, sending request signal, namely fake DHCP request (8) for changing the route and automatic handover to another AP once current AP wireless signal strength is less than or equal to the threshold level (Signal strength $\leq$ -68dbm) have been developed. According to the signalling messages send by STA, the controller takes the action for on-request dynamic routing function. The OVS (br-devops) in CHULA SmartX Box is responsible for chunked video pre-transferring before the handover events occur between APs. *Chunked video pre-transferring mechanism* which is transmitting the chunked video packets between each OpenFlow-enabled AP during STA is staying in one AP coverage area or before STA is arriving to another AP coverage area. When AP2's signal strength is less than or equal to -68dbm, STA starts sending the first fake DHCP request by using dhcping tool [65] to AP2 and then the controller notices that event via PacketIn message (8-9). Once the controller receives the message from a STA, it will instruct CHULA (br-devops) to forward the duplicate chunked video packets

to both current AP station and another AP station once before the STA reaches to the next AP station (10). Therefore, the duplicate video packet transmission called chunked video pre-transferring starts until the new signalling message comes from STA (11). So the need for the perfect synchronization of chunk assignment to the two APs would be relaxed. Therefore, the client can stream the video with less video freeze time during the handover process. Before handover to AP1, STA waits for 1 second to make sure that the controller has already added the duplicate flow entries to CHULA (br-devops). After that the STA starts client initiated handover via wlan1 to AP1 and the controller assigns the flow entries to AP1 as in the AP2 (12-16). Followed by this process, wlan0 in STA disconnects from AP2 and the duplicated streaming still remain(17). To release the unnecessary traffic usage in AP2, STA sends another fake DHCP request via AP1 and the controller takes this action for changing the route only to AP1 and stop duplicating packets (18-20). Finally, the SVR streams out the video to STA via only AP1 until the video end. The state machine diagram for client initiated handover processes is depicted in Figure 3.37 and event driven program of STA for client initiated handover processes with chunked video pre-transferring mechanism is shown in Figure 3.38.

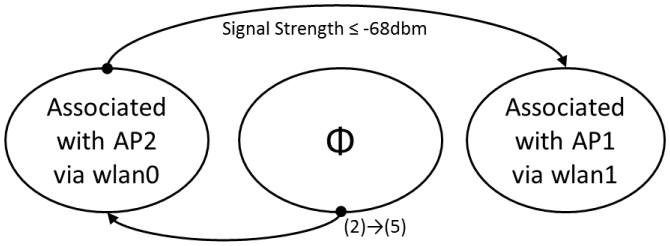


Figure 3.37: State machine diagram of STA for client initiated handover processes.

As for the timing diagram for adaptive-path chunked video streaming using Dual WLANs without chunked video pre-transferring mechanism is shown in Figure 3.39. The signalling processes (8-11) from the timing diagram of using chunked video pre-transferring do not include in this without using chunked video pre-transferring mechanism. Other processes are the same as in the timing diagram of using chunked video pre-transferring mechanism. Event driven program of STA for client initiated handover processes without chunked video pre-transferring mechanism is the same as in the

Figure 3.38: Event driven program of STA for client initiated handover processes with chunked video pre-transferring mechanism.

program of using chunked video pre-transferring mechanism except the processes :'send fake DHCP to AP2' ,'start pre-transferring' and 'stop pre-transferring' are not included in this event driven program as shown in Figure 3.40.

The timing diagram for adaptive-path chunked video streaming by using single WLAN with 100% duplication and using dual WLANs with 100% duplication are shown in Figures 3.41 and 3.42. Moreover, the event driven program of STA for client initiated handover processes by using single WLAN and dual WLANs with 100% duplication cases are shown in Figures 3.43 and 3.44.

The port information of GIST-B (br-devops), CHULA (br-devops), AP1 (ovs-ap1) and AP2 (ovs-ap2) are shown in Table 3.19 and flow entries are shown in Figures 3.45, 3.46, 3.51, 3.52. As shown in Figure 3.45 for the flow entries of br-devops in GIST-B SmartX box, when the packets arrive from the GIST-B OpenStack VM via devops_vlan (1), it will be forwarded to vxlan_TH (3) towards CHULA SmartX box. When the packets arrive from CHULA SmartX box via vxlan_TH (3), it will be forwarded to GIST-B OpenStack VM via devops_vlan (1).

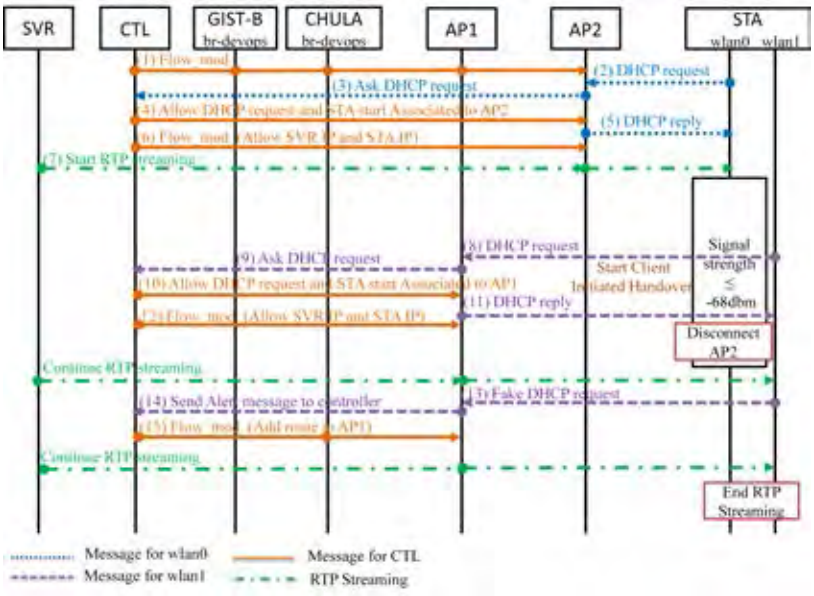


Figure 3.39: Timing diagram of of adaptive-path chunked video streaming using dual WLANs without chunked video pre-transferring mechanism.

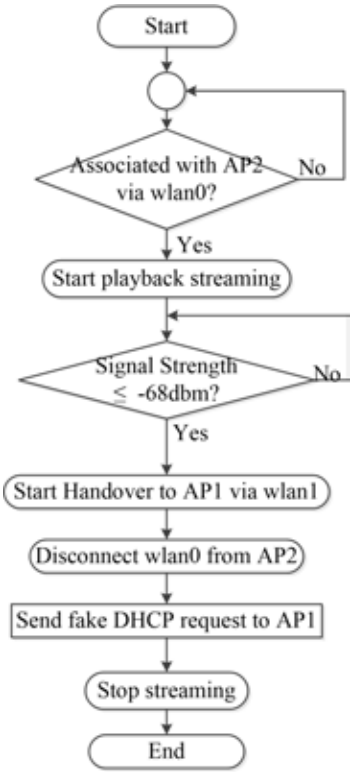


Figure 3.40: Event driven program of STA for client initiated handover processes without chunked video pre-transferring mechanism.

The flow entries of CHULA (br-devops) are shown in Figure 3.46 in which br-devops is responsible for on-request dynamic routing function. As described in above, STA always

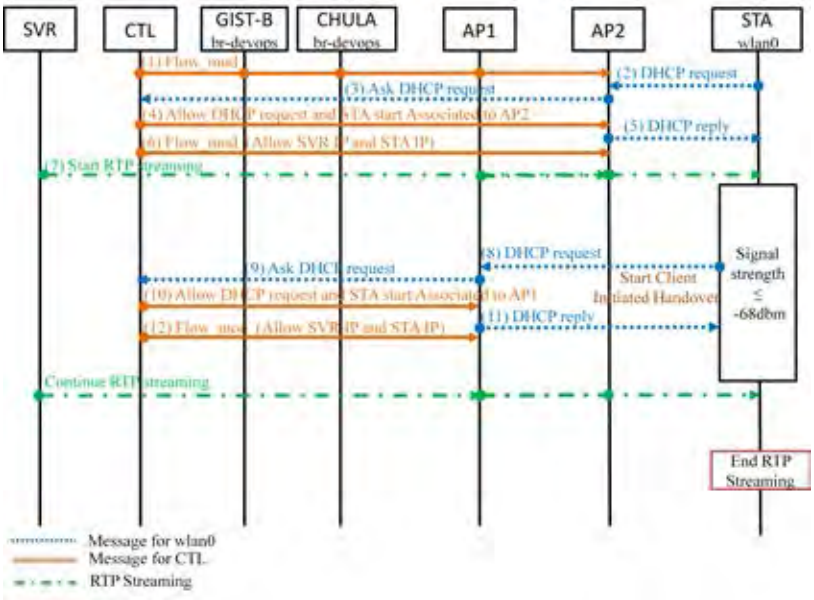


Figure 3.41: Timing diagram of of adaptive-path chunked video streaming using single WLAN with 100% duplication.

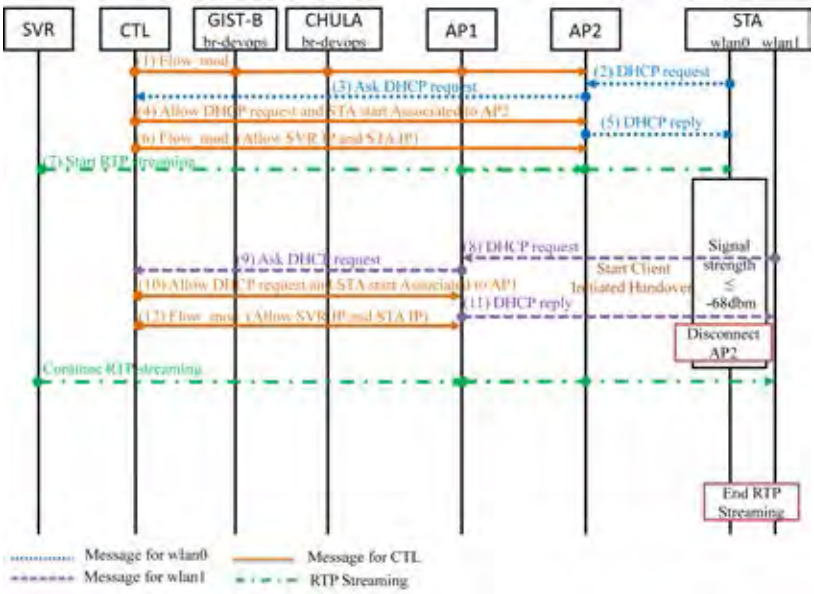


Figure 3.42: Timing diagram of of adaptive-path chunked streaming using dual WLAN with 100% duplication.

connects to SDN-AP2 so that the route to SDN-AP2 is enabled once the controller starts execute, it forwards packets via vxlan_TEST (2) to vxlan_ap2 (6) and then also add the flow for the reverse direction. The actions: ALL has been used for chunked video pre-transferring mechanism when the STA sends the signal for signal strength notification before handover to SDN-AP1. Once STA completely handover to SDN-AP1, STA will send another signalling

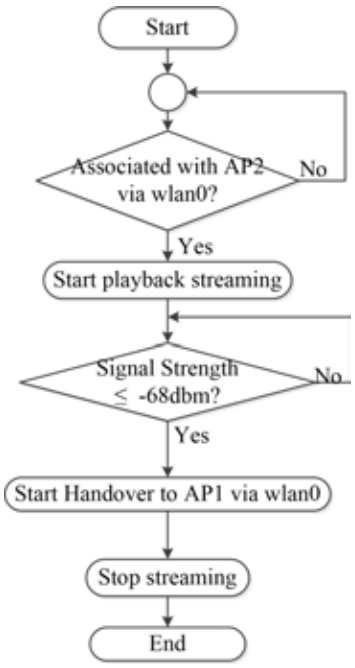


Figure 3.43: Event driven program of STA for client initiated handover processes using single WLAN with 100% duplication.

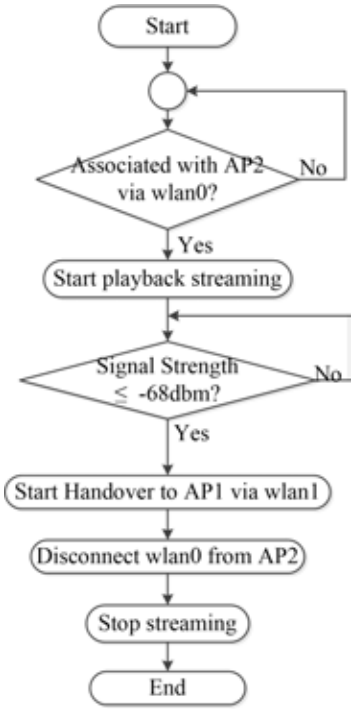


Figure 3.44: Event driven program of STA for client initiated handover processes using dual WLAN with 100% duplication.

message to change the route only to SDN-AP1 which is for forwarding packets only via vxlan_ap1(8).

Table 3.19: Port information for SmartX boxes (GIST-B,CHULA) and access points (SDN-AP1 and SDN-AP2)

SmartX boxes	br-devops ports
GIST-B(103.22.221.31)	1(devops_vlan) 3(vxlan_TH)
CHULA(161.200.25.99)	2(vxlan_TEST) 6(vxlan-ap2) 8(vxlan-ap1)
SDN-AP1(161.200.90.120)	1(eth1) 2(wlan0) 3(vxlan_TH) LOCAL(ovs-ap1)
SDN-AP1(161.200.90.103)	1(eth1) 2(wlan0) 3(vxlan_TH) LOCAL(ovs-ap2)

```
tein@SmartX-BPlus-TEST:~$ sudo ovs-ofctl dump-flows br-devops
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=247.677s, table=0, n_packets=0, n_bytes=0, idle_age=247, in_port=3 actions=output:1
 cookie=0x0, duration=247.639s, table=0, n_packets=2, n_bytes=193, idle_age=34, in_port=1 actions=output:3
tein@SmartX-BPlus-TEST:~$
```

Figure 3.45: Flow entries of GIST-B (br-devops) for wireless streaming.

The signalling messages (fake DHCP request) sent by STA to SDN-AP1 and SDN-AP2 for dynamic routing are described in Figure 3.47 for changing routes to both APs and Figure 3.48 for changing route only to SDN-AP1. Moreover, STA associated messages with SDN-AP1 and SDN-AP2 are shown in Figures 3.49 and 3.50. However, those associated messages appear only in the first time when STA associates with APs. After installing flow entries for those associated requests, no message appears in the controller terminal and STA can associate with APs within a short period of time. All messages are shown in

```
tein@SmartX-BPlus-TH:~$ sudo ovs-ofctl dump-flows br-devops
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=99.771s, table=0, n_packets=0, n_bytes=0, idle_age=99, in_port=8 actions=output:2
 cookie=0x0, duration=99.916s, table=0, n_packets=0, n_bytes=0, idle_age=99, in_port=6 actions=output:2
 cookie=0x0, duration=99.771s, table=0, n_packets=0, n_bytes=0, idle_age=99, in_port=2 actions=output:6
tein@SmartX-BPlus-TH:~$ sudo ovs-ofctl dump-flows br-devops
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=650.508s, table=0, n_packets=0, n_bytes=0, idle_age=650, in_port=8 actions=output:2
 cookie=0x0, duration=650.653s, table=0, n_packets=294, n_bytes=13516, idle_age=0, in_port=6 actions=output:2
 cookie=0x0, duration=650.508s, table=0, n_packets=5, n_bytes=474, idle_age=170, hard_age=84, in_port=2 actions=ALL
tein@SmartX-BPlus-TH:~$ sudo ovs-ofctl dump-flows br-devops
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=74.793s, table=0, n_packets=57, n_bytes=2610, idle_age=0, in_port=8 actions=output:2
 cookie=0x0, duration=75.084s, table=0, n_packets=0, n_bytes=0, idle_age=75, in_port=6 actions=output:2
 cookie=0x0, duration=74.793s, table=0, n_packets=0, n_bytes=0, idle_age=74, hard_age=23, in_port=2 actions=output:6
tein@SmartX-BPlus-TH:~$
```

Figure 3.46: Flow entries of CHULA (br-devops) for wireless streaming.

Figures 3.47-3.50 are messages appeared in the POX controller terminal.

```
IPv4 parsed packet [IP+UDP 192.168.11.3>255.255.255.255 (cs:9f60 v:4 hl:5 l:328 t:64)]
TCP parsed packet None
[openwrt_sdn_v3_new     ]  Packet_in from: 22-22-22-22-22-02|8738
[openwrt_sdn_v3_new     ]  src_ip: 192.168.11.3
[openwrt_sdn_v3_new     ]  dst_ip: 255.255.255.255
start finding new client!: 2016-05-30 10:17:14.922312
------- Found Fake DHCP request!!! --------
```

Figure 3.47: 1st Signalling message (fake DHCP request via SDN-AP2) for routing to both SDN-AP1 and SDN-AP2.

```
IPv4 parsed packet [IP+UDP 192.168.11.2>255.255.255.255 (cs:e1a2 v:4 hl:5 l:328 t:64)]
TCP parsed packet None
[openwrt_sdn_v3_new     ]  Packet_in from: 22-22-22-22-22-01|8738
[openwrt_sdn_v3_new     ]  src_ip: 192.168.11.2
[openwrt_sdn_v3_new     ]  dst_ip: 255.255.255.255
start finding new client!: 2016-05-30 10:32:39.183638
------- Found Fake DHCP request and change route only to AP1!!! --------
```

Figure 3.48: 2nd Signalling message (fake DHCP request via SDN-AP1) for routing only to SDN-AP1.

```
IPv4 parsed packet [IP+UDP 192.168.11.2>192.168.11.142 (cs:48f9 v:4 hl:5 l:331 t:64)]
TCP parsed packet None
[openwrt_sdn_v3_new     ]  Packet_in from: 22-22-22-22-22-01|8738
[openwrt_sdn_v3_new     ]  src_ip: 192.168.11.2
[openwrt_sdn_v3_new     ]  dst_ip: 192.168.11.142
start finding new client!: 2016-05-30 10:31:56.361061
------- My LAB DELL PC connected to AP1 is detected! --------
 ----- Install Flow Entry for my LAB DELL PC to AP1 ---------
2016-05-30 10:31:56.362541 Start Associated LAB PC on AP1
```

Figure 3.49: STA associated message to SDN-AP1.

```
IPv4 parsed packet [IP+UDP 192.168.11.3>192.168.11.142 (cs:8519 v:4 hl:5 l:331 t:64)]
TCP parsed packet None
[openwrt_sdn_v3_new     ]  Packet_in from: 22-22-22-22-22-02|8738
[openwrt_sdn_v3_new     ]  src_ip: 192.168.11.3
[openwrt_sdn_v3_new     ]  dst_ip: 192.168.11.142
start finding new client!: 2016-05-30 10:24:26.283399
------- My LAB DELL PC connected to AP2 is detected! --------
 ----- Install Flow Entry for my LAB DELL PC to AP2 ---------
2016-05-30 10:24:26.284844 Start Associated LAB PC on AP2
```

Figure 3.50: STA associated message to SDN-AP2.

The flow entries of SDN-AP1 and SDN-AP2 are shown in Figures 3.51 and 3.52. The flow entries for SDN-AP1 and SDN-AP2 are similar, except the LOCAL OVS IP addresses (192.168.11.2 and 192.168.11.3) are different. Firstly, the controller installs flow entries to forward packets arrive from an ingress port 2 (wlan0) to LOCAL port without the limitation of destination IP address for both SDN-AP1 and SDN-AP2. In this

implementation, only one user IP address has been allowed in order to simplify experiments. To allow the DHCP request, the flow entries with the specified network destination IP address of the STA (192.168.11.142) install in SDN-AP1 and SDN-AP2 upon STA request. Moreover, to be able to transmit packets from GIST-B OpenStack VM to STA which is connected to OpenFlow@Chula-EE wireless networks, the flow entries which include strip_vlan action and mod_vlan_vid:111 have been installed. The reason is that, OpenStack VM network have been configured with VLAN ID 111 while Wi-Fi network is not. In both AP1 and AP2, for the packets arrive from an ingress port wlan0 (2), the header VLAN ID 111 has been added with the specified destination IP address of GIST-B OpenStack VM (192.168.11.1) and forwarded via an egress port vxlan_TH (3) towards CHULA SmartX box. The processes of POX controller for all four scenarios are shown in Figures 3.53 and 3.54. The DPID of each Open vSwitch and strings are shown in Table 3.20.

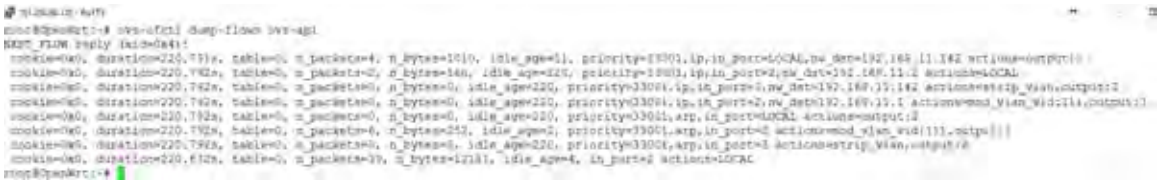


Figure 3.51: Flow entries for SDN-AP1.

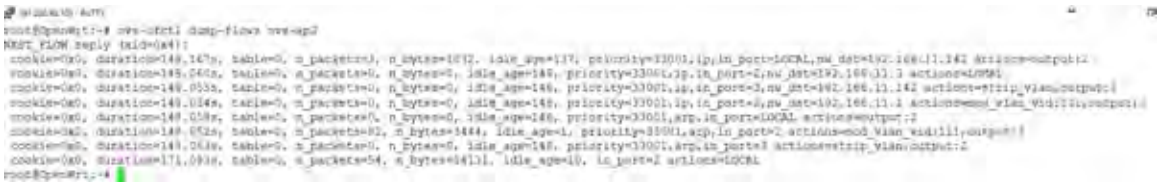


Figure 3.52: Flow entries for SDN-AP2.

Table 3.20: DPIDs and DPID strings of Open vSwitches in wireless streaming.

DPIDs of Open vSwitches	DPID string
SDN-AP1:ovs_ap1 (2222222222222201)	2459565876494606849
SDN-AP2:ovs_ap2 (2222222222222202)	2459565876494606850
GIST-B: br-devops (4444444444444401)	4919131752989213697
CHULA: br-devops (4444444444444403)	4919131752989213699

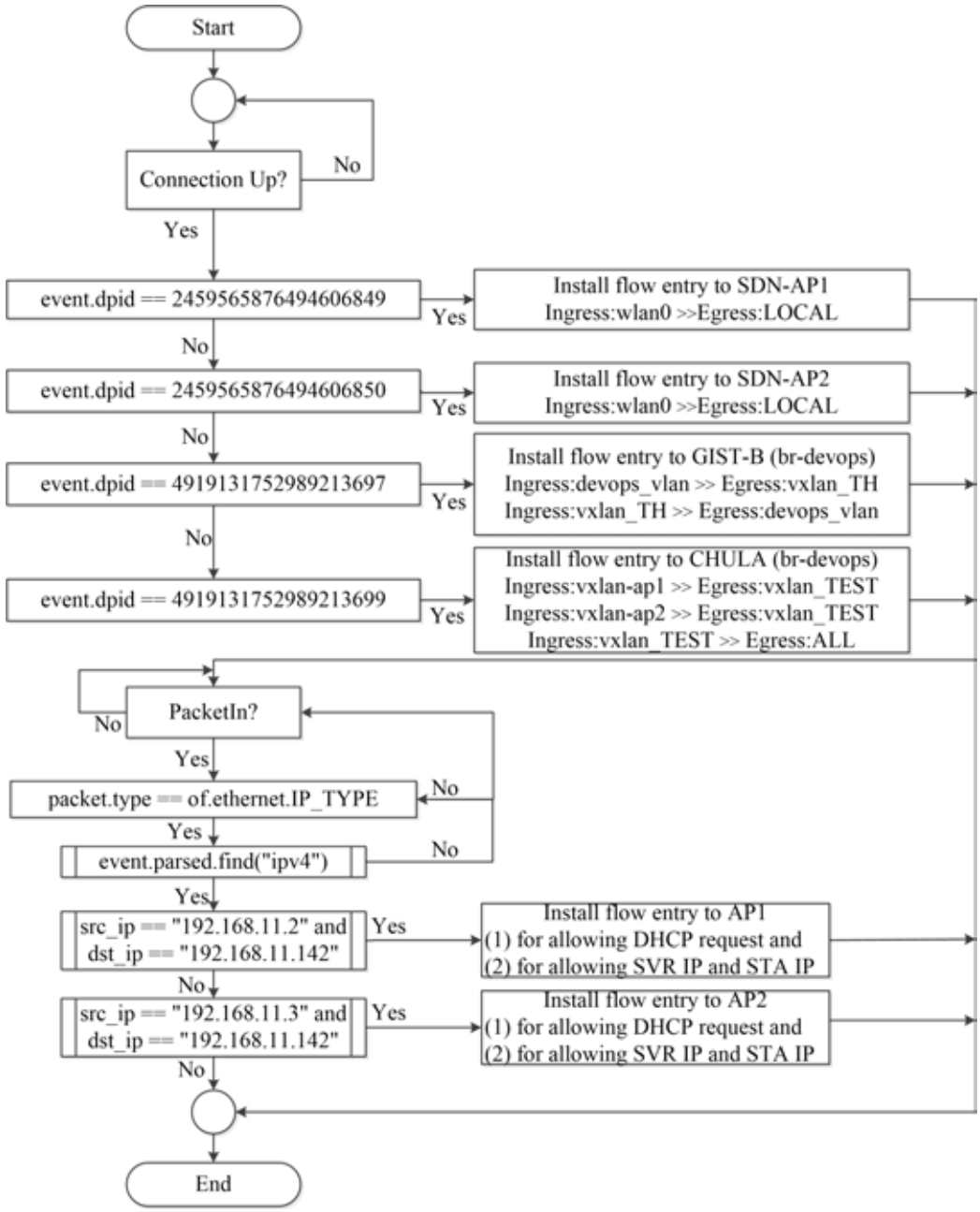


Figure 3.53: POX controller processes for adaptive-path chunked video streaming using single WLAN and dual WLANs with 100% duplication.

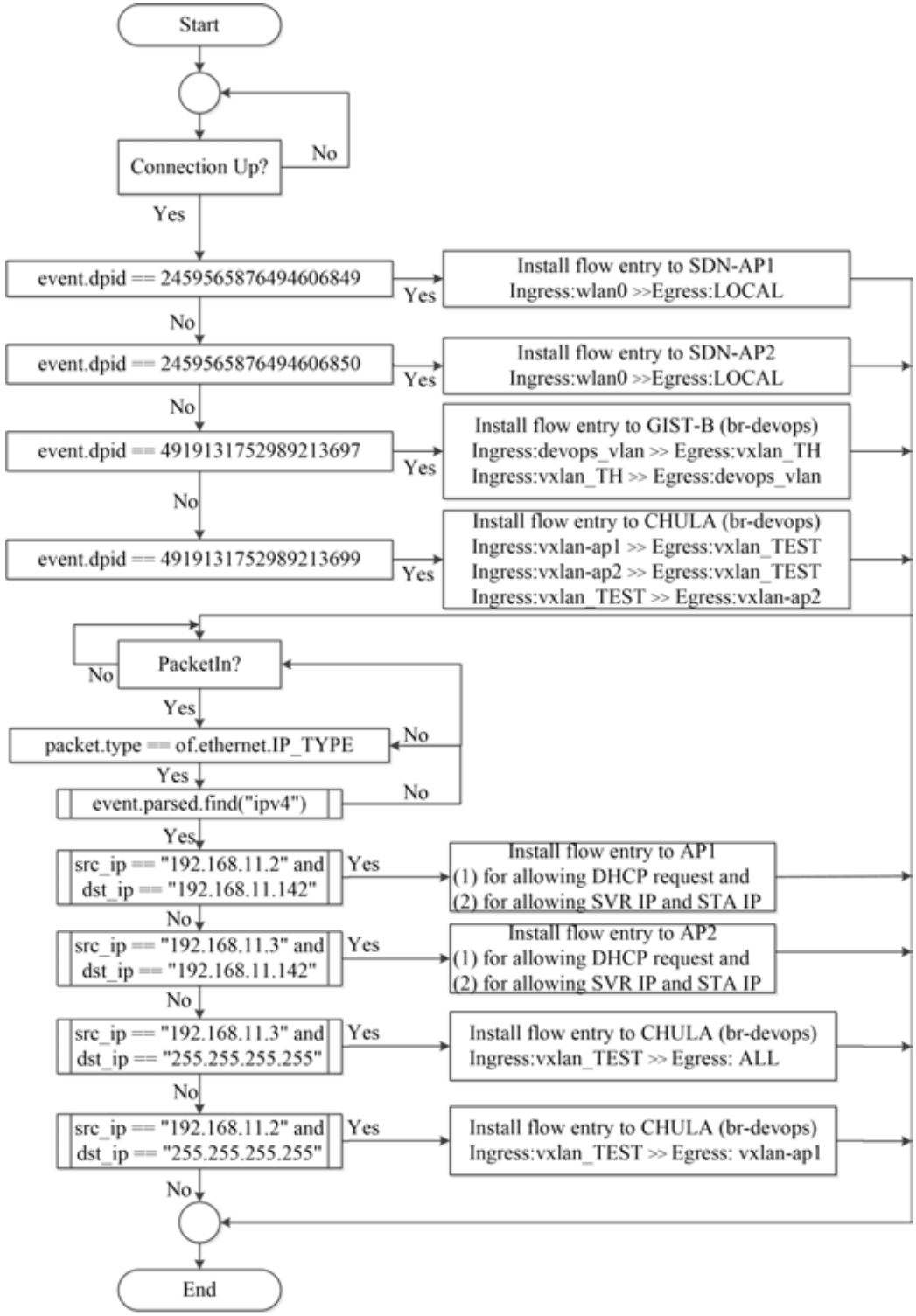


Figure 3.54: POX controller processes for adaptive-path chunked video streaming with and without using chunked video pre-transferring mechanism.

3.4.2 Results and Discussion of adaptive-path Chunked Video Streaming over OF@TEIN Wired and Wireless Links

A Frozen video [47] clip with H. 264 video codec, 1280x572 resolution, total video/audio bit rate 1128kbits/s and 2-minutes playback duration has been used in this chunked video streaming over OF@TEIN wired and wireless links. A VLC server in GIST-B OpenStack VM streams out the video with RTP mode by using command line. Ubuntu 12.04 LTS, Intel Core 2 Duo 2.8 GHz x 2 personal with 2 WLAN cards is used for a Wi-Fi video streaming client. UDP iPerf testing has been performed in order to check the available bandwidth capacity between GIST-B OpenStack VM and WiFi client (STA). According to the iPerf tests, around 8 Mbps has been obtained from GIST-B to SDN-AP1 link and around 28 Mbps via SDN-AP2. The stochastic bandwidth capacity of wireless networks and international links is the challenging issues in this experiment.

Four scenarios have been tested for performing handover testing with adaptive-path chunked video streaming as follows: (1) single WLAN with 100% duplication (2) dual WLANs with 100% duplication (3) dual WLANs with chunked video pre-transferring mechanism and (4) dual WLANs without chunked video pre-transferring mechanism. Among those four scenarios, only scenario 3 sends the signalling messages for on-request dynamic routing function. In all scenarios, STA always starts connecting to SDN-AP2 and then moves towards the SDN-AP1 coverage area. Once STA starts associating with SDN-AP2, we start executing the client initiated handover shell script file in STA for checking signal strength in every second and for automatic handover once specified threshold is over.

As in the previous video streaming experiments, tcpdump [57], a command line packet analyzer, has been used for capturing video packets at 'eth1' interface of GIST-B OpenStack cloud video server and 'any' interface of Wi-Fi STA at TSRL lab. sFlow-RT monitoring tool has been used for measuring the real-time resource consumption of access points. To evaluate the resource consumption of each access point and performance of video during the handover process with four scenarios, we have tested for each scenario with the same parameter settings and computed the packet loss ratio. The packet rate results of real-time traffic monitoring from sFlow-RT [64] use to calculate the resource consumption results for

each AP. The rate of UDP traffic (bps) obtain from the sFlow-RT monitoring results and then convert to packets per second by using UDP traffic(bps)/(1370*8) in which 1370 is the packet size of UDP packet. We sum all the packets per second values to get the total resource consumption rate for each access point. Since the purpose of this experiment is to investigate the effects of the handover process on video quality in terms of PSNR, the packet delay or jitter has not been considered. The comparison of resource consumption and packet loss ratio using four scenarios for adaptive-path chunked video streaming over OF@TEIN wired and wireless links are shown in Table 3.21.

From Table 3.21, the packet loss ratio is 5.7% for single WLAN with 100% duplication case. The reason is because only one WLAN card has been used and video packets cannot playback during the handover process due to IP address has released from wlan0 of STA and it takes time for waiting the IP address assignment from the new AP. Therefore, the packet drop occurs during the handover process when using single WLAN with 100% duplication scenario. The packets arrival and departure curve and associated events of APs by using single WLAN with 100% duplication scenario can be seen in Figure 3.55. In order to investigate the effects of the handover process between a server and a client, we have offset the packets/s at server to start the same period with the client in the resultant graph so that the occurrence of events in each time slot can be seen clearly. In order to filter out only RTP packets, the position of associated events and handover events occurrence, we have used wireshark [46] filters 'rtp' and 'tcp.port == 67 || udp.port == 67' (DHCP events) in IO graph statistic.

Table 3.21: Resource consumption of access points vs video performance at STA

Scenario	Resource Consumption (packets/s)			Video Performance at STA
	Total packets/s via AP1	Total packets/s via AP2	Total packets/s via AP1+AP2	Packet loss ratio (%)
Single WLAN with 100% duplication	14580.4	14686.0	29266.3	5.7
Dual WLANs with 100% duplication	15030.3	13054.8	28085.2	0.1
Dual WLANs with chunked video pre-transferring	9355.4	7648.5	17003.9	0.9
Dual WLANs without chunked video pre-transferring	4584.3	8333.1	12917.3	1.4

When compared to the cases of using dual WLANs with 100% duplication and with chunked video pre-transferring scenarios, packet loss ratio of using 100% duplication outperforms the case of using chunked video pre-transferring mechanism. This is because

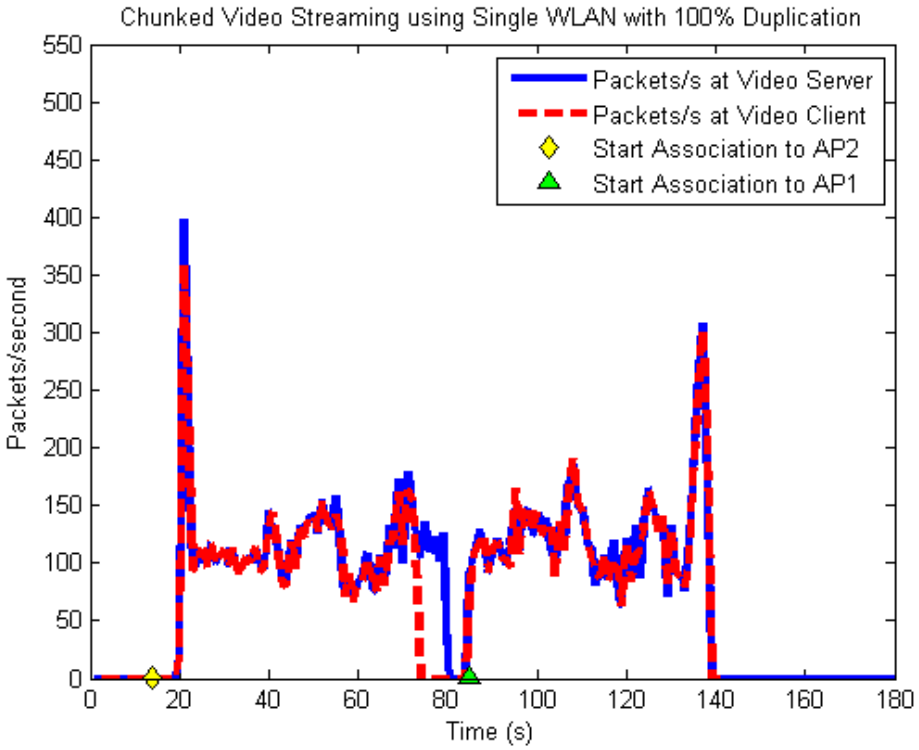


Figure 3.55: Chunked video streaming using single WLAN with 100% duplication.

of using 100% duplication does not require to add the route flow again when STA moves to the new AP coverage area while using chunked video pre-transferring requires for updating flow entry to the new AP. However, both cases have some video burst errors during the handover process due to updating the IP address of WLAN interface in STA. Although using 100% duplication case is less packet loss ratio than using chunked video pre-transferring case, the former case consumes the resource consumption of AP for the whole period of video streaming even though there has no any client associate with it while the latter is not. When considering about reducing the cost of resource consumption, we recommend the proposed chunked video pre-transferring mechanism so that non-associated AP can be relaxed when it does not require. Moreover, every case has pros and cons because using chunked video pre-transferring raises the challenges of sending signal strength status alert messages to AP and the controller. According to the packet loss results without chunked video pre-transferring case, the result of packet loss ratio confirms that the proposed chunked video pre-transferring scenario requires for achieving the lowest video freezes. The small video burst errors during the handover

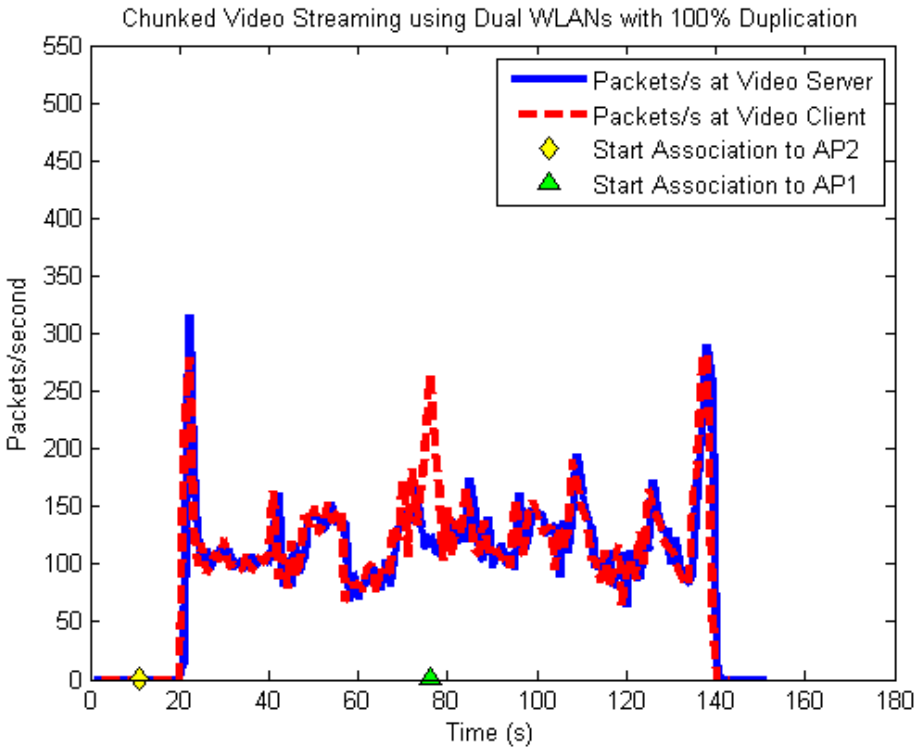


Figure 3.56: Chunked video streaming using dual WLANs with 100% duplication.

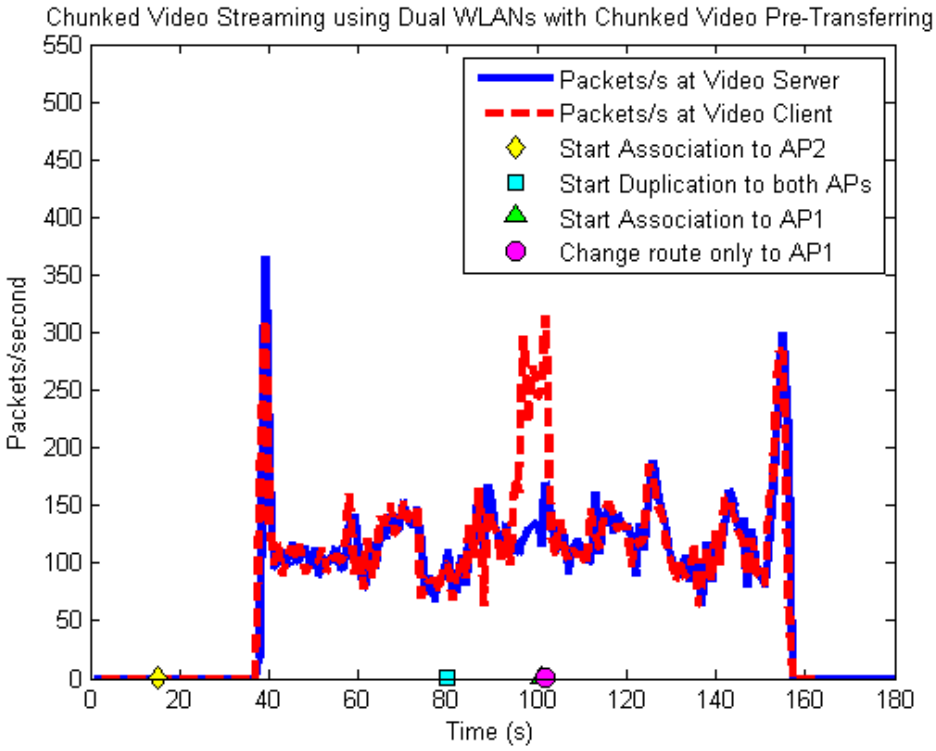


Figure 3.57: Chunked video streaming using dual WLANs with chunked video pre-transferring.

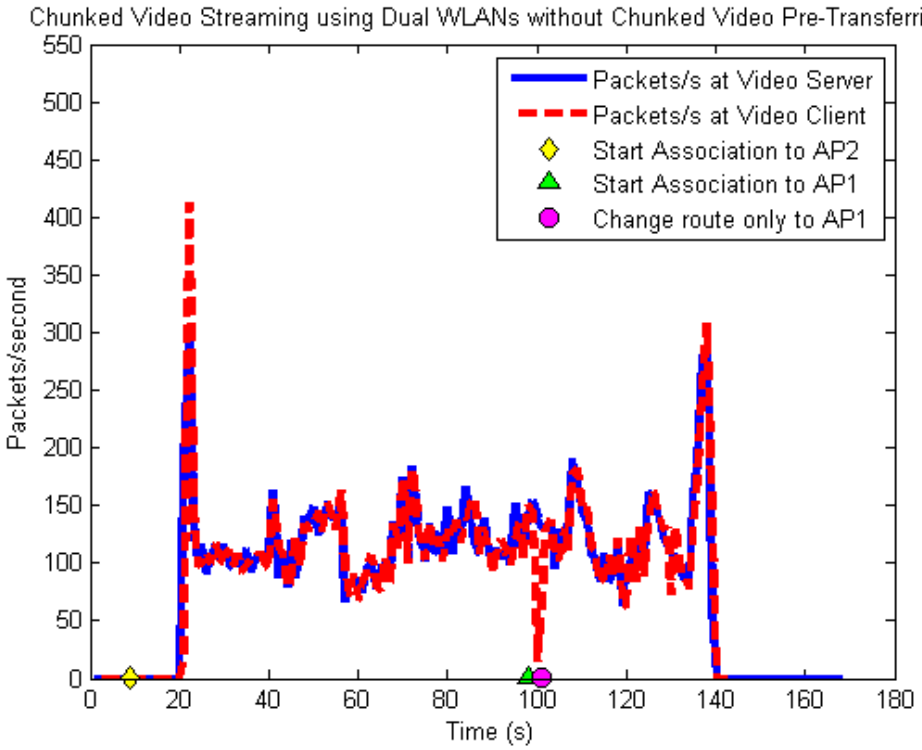


Figure 3.58: Chunked video streaming using dual WLANs without chunked video pre-transferring.

process needs to be improved in the future experiments. According to the results of resource consumption using 100% duplication and proposed pre-transferring mechanisms, using 100% duplication trade-off more resources for better video quality. When considering the lots of path condition cases, the proposed pre-transferring mechanism may consume less resources for similar received video quality. An adaptive controls on the amount of resource consumption based on the path monitoring results (for real-time traffic on each router) needs to be considered in the future. Finally, we have implemented and tested the combination of international cloud video streaming via OpenFlow@Chula-EE lab-scaled Wi-Fi links in this section. The results confirm that the implemented adaptive-path chunked video streaming with cloud service can stream out with less video freezes by using the proposed chunked video pre-transferring scenario. The packet loss period, signal strength alert message period and handover period of scenarios (2-4) are shown in Figures: 3.56-3.58. As in the scenario 1, the packets/s of server has been offset to start the same time slot with the packets/s on client in each graph.

The screen captures of tested four scenarios with real-time monitoring sFlow-RT traffic

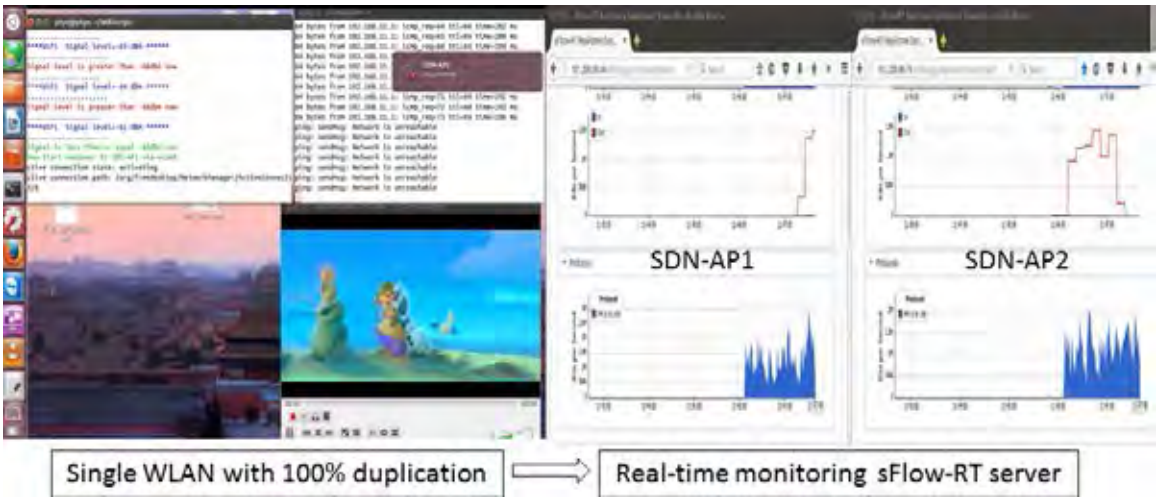


Figure 3.59: Video freeze events at STA and sFlow-RT real-time monitoring graph for single WLAN with 100% duplication case.

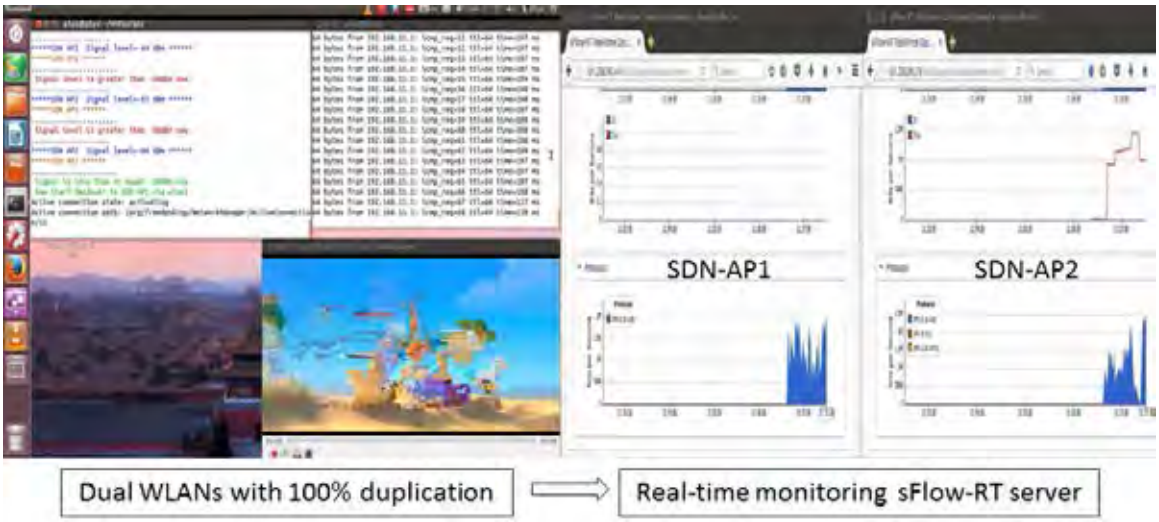


Figure 3.60: Video burst errors events at STA and sFlow-RT real-time monitoring graph for dual WLANs with 100% duplication.

Figure 3.61: Video burst errors events at STA and sFlow-RT real-time monitoring graph for dual WLANs with chunked video pre-transferring.

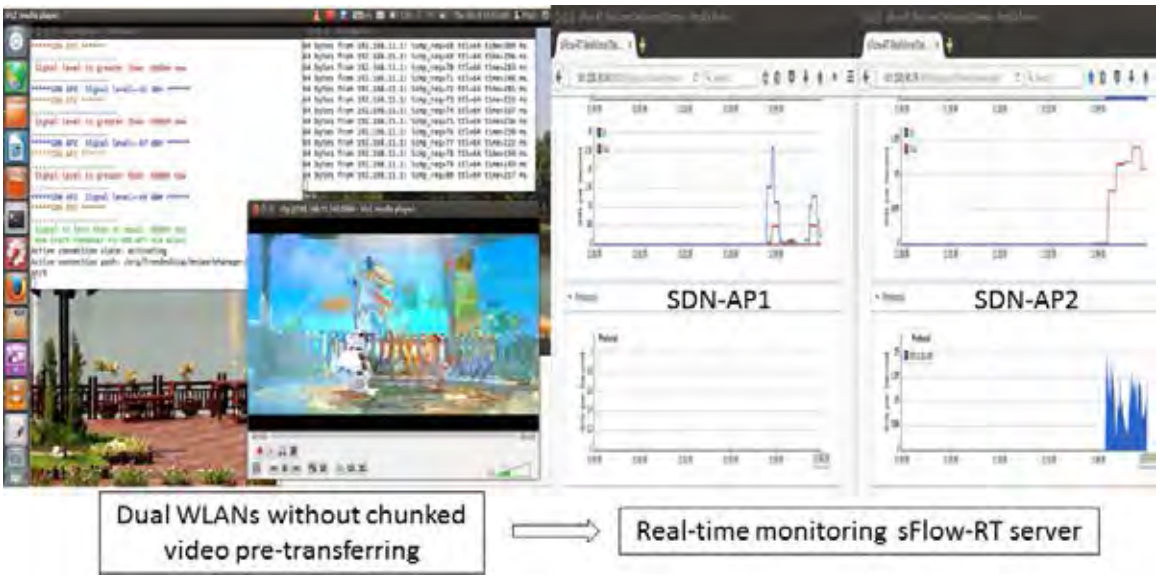


Figure 3.62: Video burst errors events at STA and sFlow-RT real-time monitoring graph for dual WLANs without chunked video pre-transferring.

graphs are shown in Figures 3.59- 3.62. The screen capture at STA side for each scenario show the long video freezes events for single WLAN with 100% duplication case and video burst errors events for other three scenarios during the handover process. The reported video capture burst errors and video freezes events confirm the reported results of packet loss ratio. Moreover, the real-time traffic monitoring at sFlow-RT server shows the resource consumption rate for each access point.

3.4.3 Summary of Adaptive-path Chunked Video Streaming over OF@TEIN Wired and Wireless Links

In this adaptive-path chunked video streaming over OF@TEIN wired and wireless links experiments, we have implemented and tested the combination of OF@TEIN cloud video streaming service and OpenFlow@Chula-EE lab-scaled wireless links for a single user. Moreover, we have developed the automatic shell script for client initiated handover for Ubuntu Wi-Fi STA and the POX controller Python script for on-request dynamic routing function. We have tested adaptive-path chunked video streaming with four scenarios: (1)single WLAN with 100% duplication (2) dual WLANs with 100% duplication (3)dual WLANs with chunked video pre-transferring mechanism and (4) dual WLANs without chunked video pre-transferring mechanism. Among them, using single WLAN with 100% duplication is not recommended due to the highest video freezes during handover process and using dual WLANs with 100% duplication is also not recommended when considering the cost of resource consumption. Using dual WLANs with chunked video pre-transferring is recommended for streaming with less video freezes and reducing the cost of resource consumption for non-associated APs, but sending signalling alert message challenge still remains to address in the future. Moreover, using dual WLANs without chunked video pre-transferring case is recommended when not considering the packet loss rate since it also can reduce the cost of resource consumption for APs. According to the results of resource consumption using 100% duplication and proposed pre-transferring mechanisms, using 100% duplication trade-off more resources for better video quality. When considering the lots of path condition cases, the proposed pre-transferring mechanism may consume less resources for similar received video quality.

Moreover, when implementing wireless SDN network with Open vSwitch over OpenWrt firmware, it is recommended to configure as 'fail-safe-mode (standalone)' in order to keep alive flow entries when the controller fails. Moreover, it notices that software switch on OpenWrt can be shut down frequently. Therefore, 'openflow.keepalive' command requires to use when the controller starts. As for the further experiments, multi-users supporting, signalling alert system and security rules need to be considered for future implementation. An adaptive controls on the amount of resource consumption based on the path monitoring results (for real-time traffic on each router) needs to be considered in the future.

Chapter 4

Conclusion

In this thesis, we have studied the performance of chunked video streaming and file transferring over emulated OpenFlow network and real experiments over the international OF@TEIN SDN cloud playground testbed with wired and wireless links. As a first method, we have designed and evaluated the functionalities of middle-box and load splitting for chunked video streaming over the OpenFlow-enabled multi-path Mininet network and real international OF@TEIN SDN cloud playground testbed which includes three countries: Korea, Malaysia and Thailand. Secondly, we have also evaluated the combination of proposed multi-path file transferring function and traditional Tsunami UDP file transferring protocol on actual. Finally, we have implemented OpenFlow@Chula-EE lab-scaled SDN wireless networks for combining with OF@TEIN cloud video streaming service and introducing the proposed chunked video pre-transferring mechanism during Wi-Fi handover process. Various scenarios and parameters have been used for evaluating the performance of subjective video quality, file transferring in the listed experiment scenarios.

Firstly, RTP video streaming experiments with the total encoding bit rate 296 kbits/sec (experiment1) and 246 kbits/sec (experiment 2) have been tested for investigating the middle-box and splitting functionalities and buffering effects over emulated multi-path OpenFlow network. The emulated network has been implemented with congested and non-congested links for carrying the whole video streaming. According to the reported packet delay, jitter and packet loss results in emulated experiment1, the multi-path video streaming method can be beneficially applied in the network when the capacity of the main path alone is not enough to carry the whole incoming packets of the video stream. However, this multi-path video streaming method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video stream due to the results of emulated experiment1. Since by introducing splitting and combining functionalities, relevant complexities, e.g. mismatching of proper chunk splitting ratio, could worsen the received subjective video quality in terms of packet loss

ratio and mean/standard deviation of packet delay. According to the results of emulated experiment2, we observe that the proper chunk size ratio and initial buffering time need to be considered carefully to improve the mean/standard deviation of packet delay. In addition, it is recommended to use the high performance machine for the middle-box processing since the maximum possible packet scheduling rate of the middle-box depends directly on computer hardware specifications.

Secondly, RTP video streaming experiments with encoding bit rate 628 kbits/sec over multi-path international OF@TEIN SDN cloud playground have been tested and reported. Unlike in the emulated network, the selected paths via Malaysia and direct link between GIST-A and CHULA are abundant bandwidth to carry the video streaming traffics and the link capacities are stochastically time varying. The reported packet delay, jitter results confirm the recommendation from our emulated multi-path video streaming that is 'using multi-path video streaming method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video stream'. The RTT delay of transmission paths are necessary to consider in order for transmitting packets with low delay according to the tested results. When implementing the middle-box over OF@TEIN SmartX boxes, careful allocation of memory and configuration are required in order to avoid looping issue and full memory issue of SmartX box. During experiments over OF@TEIN playground, the longest time has been taken for transferring the experimental data from one country node to another.

Thirdly, the combination of proposed multi-path file transferring function and traditional Tsunami UDP file transferring protocol has been evaluated with various scenarios in order to solve the file transferring delay between far distance nodes. According to the tested results, the proposed multi-path file transferring method can be transferred with minimum transmission delay when the links are congested and not enough to carry the whole video file traffic. Moreover, using multi-path function can be achieved the maximum actual file transfer rate and throughput when the selected target file transfer rate of Tsunami protocol lead to be congested on the available network links. However, this multi-path file transferring method is not recommended when the main path capacity already suffices for carrying out the incoming packets of video file traffic and

the main path already having a lower RTT delay than other available paths. The time responsiveness or interactivity of GUI applications in accessing remote OpenStack VMs have been solved out by implementing X11 Desktop Environment. Therefore, 4k video streaming with encoding bit rate 11133kbits/sec within CHULA SmartX box network has been performed by using X11 remote desktop environment. However, we observe one fact that the VLC application itself has some limitations to play back video with very high resolution, even in normal playing back without a streaming session. Therefore, when streaming out 4k resolution video with cloud playground, it is recommended to adjust the video resolution scale to be around 50% (eg. resolution:1630x937) lower than the normal 4k resolution scale.

Finally, we have also implemented OpenFlow@Chula-EE lab-scaled wireless SDN testbed for combining OF@TEIN cloud video streaming service and introducing chunked video pre-transferring mechanism during the handover process for a single user. Moreover, we have developed the automatic shell script for client initiated handover for Ubuntu Wi-Fi STA and the POX controller Python script for on-request dynamic routing function. According to the reported packet loss results, the proposed chunked video pre-transferring mechanism can stream out the video with less video freezes during the handover process. Moreover, the proposed method can be reduced the cost resource consumption for non-associated APs. According to the results of resource consumption using 100% duplication and proposed pre-transferring mechanisms, using 100% duplication trade-off more resources for better video quality. When considering the lots of path condition cases, the proposed pre-transferring mechanism may consume less resources for similar received video quality. Using dual WLAN cards of STA is recommended to perform for seamless handovers. However, sending signal alert message challenge still remains to address in the future. Moreover, when implementing wireless SDN network with Open vSwitch over OpenWrt firmware, it is recommended to configure as ‘fail-safe-mode (standalone)’ in order to keep alive flow entries when controller fails. Moreover, it notices that software switch on OpenWrt can be shut down frequently. Therefore, ‘openflow.keepalive’ command requires to use when the controller starts.

For the future work, the performance of video streaming with multi-path TCP (mTCP)

over OF@TEIN playground is worth to study. OpenFlow 1.3 group function for load-balancing over international links should be used instead of periodic splitting function. So that OpenFlow overhead messages would be reduced. Moreover, supporting multi-users with multi-path cloud video streaming is also an additional study to be considered. As for the wireless SDN streaming, network initiated handover, multi-users supporting, signalling alert system, centralized authentication management and security rules should be considered for future implementation. Moreover, deploying the SDN Wi-Fi network for mobile users is also one of the interesting approaches for future wireless mobile network. An adaptive controls on the amount of resource consumption based on the path monitoring results (for real-time traffic on each router) needs to be considered in the future. In addition, it is recommended to measure the video quality in terms of PSNR to investigate more about the video quality since the evaluation results of this thesis focus only on the subjective video quality in terms of transmission parameters at network layer such as packet loss ratio and packet delay.

References

[1] ONF Market Education Committee, and others. Software-defined networking: The new norm for networks. ONF White Paper. Palo Alto, US: Open Networking Foundation (April, 2012).

[2] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review 38 (April, 2008): 69-74.

[3] ORBIT [Online]. Available from : https://www.orbit-lab.org/ [2016, May]

[4] NITOS [Online]. Available from : http://nitlab.inf.uth.gr/NITlab/ [2016, May]

[5] SmartFire project [Online]. Available from : http://eukorea-fire.eu/ [2016, May]

[6] Risdianto, A. C., Kim, N. L., Shin, J., Bae, J., Usman, M., Ling, T. C., Panwaree, P., Thet, P. M., Aswakul, C., Thanh, N. H., Iqbal A., Javed, U., Ilyas, M. U., and Kim, J. OF@TEIN: A community efforts towards open/shared SDN-Cloud virtual playground. in Proc. of the Asia-Pacific Advanced Network 40 (August 10-14, 2015): 22-28.

[7] Risdianto, A. C., Kim, N. L., Shin, J., Bae, J., Usman, M., Ling, T. C., Panwaree, P., Thet, P. M., Aswakul, C., Thanh, N. H., Iqbal A., Javed, U., Ilyas, M. U., and Kim, J.Presentation of OF@TEIN: A community efforts towards open/shared SDN-Cloud virtual playground. [Online]. Available from : https://www-lk.apan.net/meetings/KualaLumpur2015/Sessions/11/SDN-4.pdf [2016, May]

[8] Wang, B., Wei, W., Guo, Z., and Towsley, D. Multipath live streaming via TCP: scheme, performance and benefits. in Prof. of ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) (2009): 25.

[9] Panwaree, P., Kim, J., and Aswakul, C. Packet delay and loss performance of streaming video over emulated and real openflow networks. in Proc. of 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC), (July 1-4, 2014): 777-779.

[10] Panwaree, P., Kim, J., and Aswakul, C. SDN-Coordinated multi-path chunked video streaming. Master's Thesis, Department of Electrical Engineering, Chulalongkorn University, Thailand (2014).

[11] Dely, P., Kassler, A., Chow, L., Bambos, N., Bayer, N., Einsiedler, H., Peylo, C., Mellado, D., and Sanchez, M. A software-defined networking approach for handover management with real-time video in WLANs. Journal of Modern Transportation, vol. 21, (June 10, 2013): 58-65.

[12] Qazi, Z. A., Tu, C. C., Chiang, L., Miao, R., Sekar, V., Yu, M. SIMPLE-fying middlebox policy enforcement using SDN. in Proc. of ACM SIGCOMM Computer Communication Review (August, 2013): 27-38.

[13] Kim, J., Cha, B., Kim, J., Kim, N. L., Noh, G., Jang, Y., and Kang, S. M. OF@TEIN: An OpenFlow-enabled SDN testbed over international SmartX rack sites. in Proc. of the Asia-Pacific Advanced Network 36 (August 19-23, 2013): 17-22.

[14] Gu, Y., and Grossman, RL. UDT: UDP-based data transfer for high-speed wide area networks. in Proc. of Computer Networks 51 (May 16, 2007): 1777-1799.

[15] Meiss, M. R. Tsunami: A high-speed rate-controlled protocol for file transfer. Indiana University 2004.

[16] Huang, C., Nakasan, C., Ichikawa, K., and Iida, H. A multipath controller for accelerating GridFTP transfer over SDN. in Proc. of 2015 IEEE 11th International Conference on IEEE (August 31, 2015): 439-447.

[17] OpenFlow 1.0 OpenFlow 1.0 Switch Specification, Version 1.0 (December 31, 2009).

[18] Cisco IT Blog [Online]. Available from : http://blogs.cisco.com/ciscoit/sdn-101-what-it-is-why-it-matters-and-how-to-do-it [2016, May]

[19] Lantz, B., Heller, B., and McKeown, N. A network in a laptop: Rapid prototyping for software-defined networks. in Proc. of ACM SIGCOMM Workshop on Hot Topics in Networks (October, 2010): 19-24.

[20] Arefin, A., Rivas, R., Tabassum, R., and Nahrstedt, K. OpenSession: SDN-based cross-layer multi-stream management protocol for 3D teleimmersion. in Proc. of 21st IEEE International Conference on Network Protocols (October 7-10, 2013): 1-10.

[21] Ruckert, J., Blendi, J., and Hausheer, D. RASP: Using OpenFlow to push overlay streams into the underlay. in Proc. of IEEE Thirteenth International Conference on Peer-to-Peer Computing (P2P) (September, 2013): 1-2.

[22] Marcondes, C. A. C., Santos, T. P. C., Godoy, A. P., Viel, C. C., and Teixeira, C. A. C. CastFlow: Clean-slate multicast approach using in-advance path processing in programmable networks. in Proc. of IEEE Symposium on Computers and Communications (ISCC) (July, 2012): 94-101.

[23] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hlzle, U., Stuart, S., and Vahdat, A. B4: Experience with a globally-deployed software defined WAN. in Proc. of ACM SIGCOMM Computer Communication Review vol. 21 , no. 4 (August 12-16, 2013): 3-14.

[24] Pereira, R., and Pereira, E. G. Client buffering considerations for video streaming. in Proc. of Advanced Information Networking and Applications Workshops (WAINA), 27th International Conference on IEEE (2013): 595-600.

[25] Brief. OpenFlow-enabled mobile and wireless networks. ONF Solution,(2013).

[26] Claude, C., and Haddad, Y. Wireless software defined networks: Challenges and opportunities. in Proc. of Microwaves, Communications, Antennas and Electronics Systems (COMCAS), 2013 IEEE International Conference (October 21- 23, 2013): 1-5.

[27] Ericsson SDN projects [Online]. Available from : http://www.ericsson.com/spotlight/cloud/network-modernization/sdn [2016, May]

[28] Nokia SDN projects [Online]. Available from : https://networks.nokia.com/ [2016, May]

[29] Cisco SDN projects [Online]. Available from : http://www.cisco.com/web/solutions/trends/sdn/index.html [2016, May]

[30] Meru Networks Committee. SDN for Wi-Fi. Meru Networks White Paper (2014) [Online]. Available from : www.completenetworks.co.uk/uploads/meru-sdn-wifi.pdf [2016, May]

[31] Anyfi networks [Online]. Available from : http://www.anyfinetworks.com/ [2016, May]

[32] Yap, K. K., Kobayashi, M., Underhill, D., Seetharaman, S., Kazemian, P., and McKeown, N. The Stanford Openroads deployment. in Proc. of the 4th ACM international workshop on Experimental evaluation and characterization (September 20-25, 2009): 59-66.

[33] Yap, K. K., Huang, T. Y., Kobayashi, M., Chan, M., Sherwood, R., Parulkar, G., and Mckeown, N. Lossless handover with n-casting between WiFi-WiMAX on OpenRoad. in Proc. of ACM Mobicom (Demo), vol. 12 (September 20-25, 2009): 40-52.

[34] Dely, P., Kassler, A., and Bayer, N. Openflow for wireless mesh networks. in Proc. of 20th International Conference on Computer Communications and Networks (ICCCN) (July 31- August 4, 2011): 1-6.

[35] Vestin, J., Dely, P., Kassler, A., Bayer, N., Einsiedler, H., and Peylo, C. CloudMAC: towards software defined WLANs. in Proc. of ACM SIGMOBILE Mobile Computing and Communications Review 16, no. 4 (2013): 42-45.

[36] Zander, J. S., Suresh, L., Sarrar, N., Feldmann, A., Hhn, T., and Merz, R. Programmatic orchestration of wifi networks. in Proc. of 2014 USENIX Annual Technical Conference (USENIX ATC 14) (2014): 347-358.

[37] Zander, J. S., Mayer, C., Ciobotaru, B., Schmid, S., and Feldmann, A. OpenSDWN: Programmatic control over home and enterprise WiFi. in Proc. of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (2015): 16.

[38] Meneses, F., Corujo, D., Guimaraes, C. and Aguiar, R. L. Multiple flow in extended SDN wireless mobility. in Proc. of 2015 Fourth European Workshop on Software Defined Networks (EWSDN), IEEE (September 30 - October 2, 2015): 1-6.

[39] You, T., Baron, L., Fdida, S., Kim, J. Enabling SDN Experimentation with wired and wireless resources: The SmartFIRE facility. [Online]. Available from : http://eukorea-fire.eu/wp-content/uploads/2015/09/Enabling_SDN_Experimentation_with_Wired_and_Wireless_Resources-The_SmartFIRE_facility.pdf [2016, May]

[40] Thet, P. M., Panwaree, P., Kim, J., and Aswakul, C. Design and functionality test of chunked video streaming over emulated multi-path OpenFlow network. in Proc. of Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 12th International Conference, IEEE (June 24-26, 2015): 1-6.

[41] Thet, P. M., Tientrakul, N., Kim, J., and Aswakul, C. Emulated OpenFlow based experimental study on middle-box buffering effect for multi-path chunked video streaming. in Proc. of 30th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC) (June 29 - July 2, 2015): 184-187.

[42] Trinh, T., Esaki, H., and Aswakul, C. Dynamic virtual network allocation for OpenFlow based cloud resident data center," in Proc. of IEICE Transactions on Communications, IEICE (January, 2013): 56-64.

[43] POX Wiki [Online]. Available from : https://openflow.stanford.edu/display/ONL/POX+Wiki [2016, May]

[44] Big Buck Bunny Movie [Online]. Available from : http://www.bigbuckbunny.org [2016, May]

[45] The VideoLAN project [Online]. Available from : http://www.videolan.org/vlc [2016, May]

[46] Wireshark [Online]. Available from : https://www.wireshark.org/ [2016, May]

[47] Frozen Movie [Online]. Available from : http://disney.wikia.com/wiki/Frozen [2016, May]

[48] Openstack [Online]. Available from : https://www.openstack.org/ [2016, May]

[49] Open vSwitch [Online]. Available from : http://openvswitch.org/ [2016, May]

[50] iPerf [Online]. Available from : https://iperf.fr/ [2016, May]

[51] Sherwood, R., Gibb, G., Yap, K. K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G. FlowVisor: A network virtualization layer [Online], 2009 October. Available from : http://openflowswitch.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf [2016, May]

[52] OpenStack Compute, Nova [Online]. Available from : https://wiki.openstack.org/wiki/Nova [2016, May]

[53] OpenStack Networking, Neutron [Online]. Available from : https://wiki.openstack.org/wiki/Neutron [2016, May]

[54] Virtual extensilbe LAN, VxLAN [Online]. Available from : https://en.wikipedia.org/wiki/Virtual_Extensible_LAN [2016, May]

[55] OpenDayLight [Online]. Available from : https://www.opendaylight.org/ [2016, May]

[56] OpenWrt [Online]. Available from : https://openwrt.org/ [2016, May]

[57] Tcpdump [Online]. Available from : http://www.tcpdump.org/ [2016, May]

[58] VLAN (802.1q) configuration program (vconfig) [Online]. Available from : http://linux.die.net/man/8/vconfig [2016, May]

[59] LXDE [Online]. Available from : http://lxde.org/ [2016, May]

[60] Xrdp [Online]. Available from : http://www.xrdp.org/ [2016, May]

[61] Tightvnc [Online]. Available from : http://www.tightvnc.com/ [2016, May]

[62] Xorg [Online]. Available from : https://www.x.org/wiki/ [2016, May]

[63] Big Buck Bunny 4k Video [Online]. Available from : https://www.youtube.com/watch?v=aqz-KE-bpKQ [2016, May]

[64] sFlow-RT [Online]. Available from : http://www.inmon.com/products/sFlow-RT.php [2016,May]

[65] dhcping [Online]. Available from : http://linux.die.net/man/8/dhcping [2016,May]

Appendices

Appendix A

Mininet Script File for Emulated Experiment 1, 2

```
"""Custom topology exapmle
Four switches plus two hosts+one middlebox
     host1--- switch1 --- switch2 --- switch3---host3
                      |           |         ||
                     --- switch4 ---        mb

This file is for creating topology with 2 path & setting link bandwidth and also create middle-box.
This file is used for running experiments for emulated experiment 1 (ECTICON) and experiment 2 (ITC_CSCC)
Reference: Mininet with middle-box: https://github.com/yeasy/mininet/tree/devel/mb"""

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class MiddleBoxTopo(Topo):
    """
    Middlebox test topology.
    """

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        host1 = self.addHost( 'h1' )
        host2 = self.addHost( 'h2' )
        #mb = self.addMiddleBox( 'm1' )  # To use Built-in middlebox function
        mb = self.addHost( 'm1' )        # To run our created middlebox script
        switch1= self.addSwitch( 's1' )
        switch2= self.addSwitch( 's2' )
        switch3= self.addSwitch( 's3' )
        switch4= self.addSwitch( 's4' )

        # Add links
        linkopts1 = dict(bw=0.3)   # Link Bandwidth setting used in ECTICON 2015
        linkopts2 = dict(bw=0.15)  # Link Bandwidth setting used in ECTICON 2015
        #linkopts1 = dict(bw=0.25) # Remove comment to run ITC_CSCC 2015
        #linkopts2 = dict(bw=0.15) # Remove comment to run ITC_CSCC 2015
        self.addLink( host1, switch1, **linkopts1 )
        self.addLink( switch1, switch2, **linkopts1 )
        self.addLink( switch2, switch3, **linkopts1 )
        self.addLink( switch1, switch4, **linkopts2 )
        self.addLink( switch4, switch3, **linkopts2 )
        self.addLink( switch3, host2, **linkopts1 )
```

```
50          self.addLink( switch3, mb, **linkopts1 )
51          self.addLink( switch3, mb, **linkopts1 )
52
53  topos = { 'mbtopo': ( lambda: MiddleBoxTopo() ) }
```

Listing A.1: Run Mininet Script

```
1  sudo mn --custom ~/mininet/custom/ecti_itccscc.py --topo mbtopo --controller remote
       --link tc --mac
```

Appendix B

POX Controller Python Script for Emulated Experiment 1, 2

```
"""
pox-mbox1.py
This file is for adding flow entries to all switches.
And use to split chunk video periodically at ovs1.
Phyo V1 modified to run in mininet
hard_timeout is used for varying chunk size ratio
Reference:Panwaree, P., Kim, J., and Aswakul, C. SDN-Coordinated multi-path chunked
    video streaming. Master's Thesis, Department of Electrical Engineering,
    Chulalongkorn University, Thailand (2014).
"""
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.packet as pkt
from threading import Timer
import time
import math

log = core.getLogger()

class MyComponent (object):

    def __init__ (self):
        core.openflow.addListeners(self)

    def _handle_ConnectionUp (self, event):
        log.debug("Switch %s has come up.", dpid_to_str(event.dpid))

        if event.dpid == 2: ## 2 is the dpid of s2
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        match=of.ofp_match(in_port=1)))
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        match=of.ofp_match(in_port=2)))

        elif event.dpid == 3:  #s3
            event.connection.send(of.ofp_flow_mod(
                        action=(of.ofp_action_dl_addr.set_src(
                        EthAddr("00:00:00:00:00:05")), #s3-eth1
                        of.ofp_action_output(port=4)),
                        match=of.ofp_match(in_port=1)))
            event.connection.send(of.ofp_flow_mod(
                        action=(of.ofp_action_dl_addr.set_src(
                        EthAddr("00:00:00:00:00:06")), #s3-eth2
                        of.ofp_action_output(port=4)),
                        match=of.ofp_match(in_port=2)))
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=of.OFPP_ALL),
```

```
                        match=of.ofp_match(in_port=3)))
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=3),
                        match=of.ofp_match(in_port=5)))
            event.connection.send(of.ofp_flow_mod(
                        match=of.ofp_match(in_port=4)))

        elif event.dpid == 4:  #s4
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        match=of.ofp_match(in_port=1)))
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        match=of.ofp_match(in_port=2)))

        elif event.dpid == 1:  #s1
            def install_path1():
                print "Install flow entry for Path 1"
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        hard_timeout=10, match=of.ofp_match(in_port=1)))
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        hard_timeout=10, match=of.ofp_match(in_port=2)))
            def install_path2():
                print "Install flow entry for Path 2"
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=3),
                        hard_timeout=20, match=of.ofp_match(in_port=1)))
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        hard_timeout=20, match=of.ofp_match(in_port=3)))

            def install_flow():

                install_path1()
                time_int_path1 = 10
                time_int_path2 = 20
                prev_time = math.floor(time.time())
                current_path_installed = 1

                while True:

                    if current_path_installed == 1:
                        if math.floor(time.time())-prev_time==time_int_path1:
                            install_path2()
                            print math.floor(time.time())
                            current_path_installed = 2
                            prev_time = math.floor(time.time())

                    elif current_path_installed == 2:
                        if math.floor(time.time())-prev_time==time_int_path2:
                            install_path1()
                            print math.floor(time.time())
                            current_path_installed = 1
                            prev_time = math.floor(time.time())


            t = Timer(10,install_flow)
```

```
109             t.start()
110
111 def launch ():
112     core.registerNew(MyComponent)
```

Listing B.1: Run controller Script

```
1 sudo ./pox.py pox-mbox1
```

Appendix C

Middle-box Python Script for Emulated Experiment 1, 2

```
#!/usr/bin/env python
"""
mininet_mbox.py
Phyo modified to run in mininet
This file is for middlebox to capture, parse, store and send packets in emulation experiments.
Reference:Panwaree, P., Kim, J., and Aswakul, C. SDN-Coordinated multi-path chunked video streaming. Master's Thesis, Department of Electrical Engineering, Chulalongkorn University, Thailand (2014).
"""
import socket
from struct import *
import datetime
import time
import math
import pcapy
import dpkt
import sys
from scapy.all import *
import threading

global count, buffer1, buffer2, ts1, ts2, arp, pkt, fwd, rtp1, flag
count = 0
arp = 0
flag = 0
buffer1 = []
buffer2 = []
ts1 = []
ts2 = []
pkt = []
fwd = []
sys.setrecursionlimit(5000)

def main(argv) :

    #list all devices
    devices = pcapy.findalldevs()
    #print devices

    '''
    open device
    # Arguments here are:
    #   device
    #   snaplen (maximum number of bytes to capture _per_packet_)
    #   promiscious mode (1 for true)
    #   timeout (in milliseconds)
    '''
    cap = pcapy.open_live("m1-eth0" , 65536 , 1 , 0)

    #start sniffing packets
    while(1) :
```

```
49          (header, packet) = cap.next()
50          #print ('%s: captured %d bytes, truncated to %d bytes'
        %(datetime.datetime.now(), header.getlen(), header.getcaplen()))
51          parse_packet(packet)
52
53  #function to parse a packet
54  def parse_packet(packet) :
55
56      global count, buffer1, buffer2, ts1, ts2, arp, pkt, fwd, rtp1
57
58      print time.time(),',',(len(buffer1)),',',(len(buffer2))
59
60      eth = dpkt.ethernet.Ethernet(packet)
61
62      #Parse eth packets,
63      if eth.type == dpkt.ethernet.ETH_TYPE_IP:
64
65          ip = eth.data
66          udp = ip.data
67          rtp1 = udp.data
68
69          #UDP protocol
70          if ip.p == 17:
71              #RTP packet
72              rtp = dpkt.rtp.RTP(str(rtp1))
73
74              if eth.src == '\x00\x00\x00\x00\xd3\xe1' : #eth1 s3 (Path1)
75                  count += 1
76                  buffer1.append(packet)
77                  time_stamp = rtp.ts
78                  ts1.append(time_stamp)
79
80              elif eth.src == '\x00\x00\x00\x03\xd3\xe2' : #eth2 s3 (Path2)
81                  count += 1
82                  buffer2.append(packet)
83                  time_stamp = rtp.ts
84                  ts2.append(time_stamp)
85
86          #some other UDP packet like IGMP
87          else :
88              fwd.append(packet)
89              s_fwd = fwd.pop(0)
90              h_fwd = '\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x01\x08\x00'
91              f = Ether(h_fwd+s_fwd[14:])
92              sendp(f, iface = 'm1-eth1', verbose=0)
93
94      #ARP packet :
95      elif eth.type == dpkt.ethernet.ETH_TYPE_ARP:
96          arp += 1
97          if arp == 1:
98              pkt.append(packet)
99              s_pkt = pkt[0]
100             #Phyo to change header h2-eth0 and h1-eth0
101             head_pkt = '\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x01\x08\x06'
102             y = Ether(head_pkt+s_pkt[14:]) #14 bytes for ARP
103             sendp(y, iface = 'm1-eth1', verbose=0)
104
105     print
106
```

```
107 def sendpkt():
108
109     global rate, buffering_time, flag
110
111     prev_time = math.floor(time.time()*rate)
112     start_time = math.floor(time.time())
113
114     while True:
115
116         #Phyo h2-eth0 and h1-eth0
117         head = '\x00\x00\x00\x00\x00\x02\x00\x00\x00\x00\x00\x01\x08\x00'
118         if count == 1:
119             start_time = math.floor(time.time())
120         else:
121             pass
122
123         if (math.floor(time.time()) - start_time >= buffering_time) and
    (math.floor(time.time()*rate) != prev_time):
124
125             if (len(buffer1) != 0) and (len(buffer2) != 0):
126                 a = ts1[0]
127                 b = ts2[0]
128
129                 if a > b:       # send packet in buffer2
130                     bb = buffer2.pop(0)
131                     ts2.pop(0)
132                     x = Ether(head+bb[14:])
133                     try :
134                         sendp(x, iface = 'm1-eth1', verbose=0)
135                         prev_time = math.floor(time.time()*rate)
136                     except socket.error :
137                         continue
138
139                 elif b > a :    # send packet in buffer1
140                     aa = buffer1.pop(0)
141                     ts1.pop(0)
142                     z = Ether(head+aa[14:])
143                     try :
144                         sendp(z, iface = 'm1-eth1', verbose=0)
145                         prev_time = math.floor(time.time()*rate)
146                     except socket.error :
147                         continue
148
149                 elif a == b :    # send packet in buffer1
150                     aa = buffer1.pop(0)
151                     ts1.pop(0)
152                     buffer2.pop(0)  # delete pkt in buffer2
153                     ts2.pop(0)
154                     z = Ether(head+aa[14:])
155                     try :
156                         sendp(z, iface = 'm1-eth1', verbose=0)
157                         prev_time = math.floor(time.time()*rate)
158                     except socket.error :
159                         continue
160
161             elif len(buffer1) == 0 and len(buffer2) != 0:
162                 bb = buffer2.pop(0)
163                 ts2.pop(0)
164                 x = Ether(head+bb[14:])
```

```
                try :
                    sendp(x, iface = 'm1-eth1', verbose=0)
                    prev_time = math.floor(time.time()*rate)
                except socket.error :
                    continue

            elif len(buffer2) == 0 and len(buffer1) != 0:
                aa = buffer1.pop(0)
                ts1.pop(0)
                z = Ether(head+aa[14:])
                try :
                    sendp(z, iface = 'm1-eth1', verbose=0)
                    prev_time = math.floor(time.time()*rate)
                except socket.error :
                    continue

            else :
                pass

        else :
            pass


if __name__ == "__main__":

    global rate, buffering_time
    rate = 40    #Vary mbox packets generation rate
    buffering_time = 0 #Vary initial buffering time

    t1 = threading.Thread(target=main, args=(sys.argv))
    t2 = threading.Thread(target=sendpkt)

    t1.start()
    t2.start()

    t1.join()
    t2.join()
```

Listing C.1: Middle-box set up

```
Middle-box set up on Ubuntu 12.04 LTS
sudo apt-get install python-dpkt  #To install dpkt for parsing
sudo apt-get install python-pcapy #To install pcapy for capturing
sudo apt-get install scapy  #To install scapy for generating
sudo apt-get install libpcap-dev  #To install libpcap
sudo python mininet_mbox.py
```

Appendix D

Middle-box Set Up in CHULA SmartX Box

```
#!/usr/bin/env python
"""
mbox_oftein.py
This file is for middlebox to capture, parse, store and send packets.
This file is modified to run over OF@TEIN testbed
Reference:Panwaree, P., Kim, J., and Aswakul, C. SDN-Coordinated multi-path chunked
    video streaming. Master's Thesis, Department of Electrical Engineering,
    Chulalongkorn University, Thailand (2014).
"""

import socket
from struct import *
import datetime
import time
import math
import pcapy
import dpkt
import sys
from scapy.all import *
import threading

global count, buffer1, buffer2, ts1, ts2, arp, pkt, fwd, rtp1, flag
count = 0
arp = 0
arp1 = 0 # To solve ping
flag = 0
buffer1 = []
buffer2 = []
ts1 = []
ts2 = []
pkt = []
fwd = []
sys.setrecursionlimit(5000)

def main(argv) :

    #list all devices
    devices = pcapy.findalldevs()
    #print devices

    '''
    open device
    # Arguments here are:
    #   device
    #   snaplen (maximum number of bytes to capture _per_packet_)
    #   promiscious mode (1 for true)
    #   timeout (in milliseconds)
    '''
    #cap = pcapy.open_live("eth1" , 65536 , 1 , 0)
    cap = pcapy.open_live("eth1" , 65536 , 1 , 0)

    #start sniffing packets
    while(1) :
```

```
52          (header, packet) = cap.next()
53          print ('%s: captured %d bytes, truncated to %d bytes'
    %(datetime.datetime.now(), header.getlen(), header.getcaplen()))
54          parse_packet(packet)
55
56  #function to parse a packet
57  def parse_packet(packet) :
58
59      global count, buffer1, buffer2, ts1, ts2, arp, pkt, fwd, rtp1
60
61      print time.time(),',',(len(buffer1)),',',(len(buffer2))
62
63      eth = dpkt.ethernet.Ethernet(packet)
64
65      #Parse eth packets,
66      if eth.type == dpkt.ethernet.ETH_TYPE_IP:
67
68          ip = eth.data
69          udp = ip.data
70          rtp1 = udp.data
71
72          #UDP protocol
73          if ip.p == 17:
74              #RTP packet
75              rtp = dpkt.rtp.RTP(str(rtp1))
76
77              if eth.src == '\xe2\x9c\xe6\x30\xbf\x06' :  #port3 MYREN at TH br2
    (Path1) e2:9c:e6:30:bf:06
78                  count += 1
79                  buffer1.append(packet)
80                  time_stamp = rtp.ts
81                  ts1.append(time_stamp)
82
83              elif eth.src == '\x66\xc2\xa5\x35\xd2\xd6' : #port2 GIST at TH br2
    (Path2) 66:c2:a5:35:d2:d6
84                  count += 1
85                  buffer2.append(packet)
86                  time_stamp = rtp.ts
87                  ts2.append(time_stamp)
88
89          #some other UDP packet like IGMP
90          else :
91              fwd.append(packet)
92              s_fwd = fwd.pop(0)
93              #fa:16:3e:72:3a:99 MAC address of GIST VM -eth1  fa:16:3e:7f:d6:75 MAC
    addr of TH VM -eth1
94              h_fwd = '\xfa\x16\x3e\x7f\xd6\x75\xfa\x16\x3e\x72\x3a\x99\x81\x00'
95              f = Ether(h_fwd+s_fwd[14:])
96              sendp(f, iface = 'eth2', verbose=0)
97              print "Send from IGMP"
98              #sendp(f, iface = 'eth0', verbose=0)
99
100     #ARP packet :
101     elif eth.type == dpkt.ethernet.ETH_TYPE_ARP:
102         arp += 1
103         if arp == 1:
104             pkt.append(packet)
105             s_pkt = pkt[0]
```

```
106             #Change to 0x8100 instead of 0x0806, actual arp header is 0x0806,.To be
    ping work
107             head_pkt = '\xfa\x16\x3e\x7f\xd6\x75\xfa\x16\x3e\x72\x3a\x99\x81\x00'
108             y = Ether(head_pkt+s_pkt[14:]) #14 bytes for ARP
109             sendp(y, iface = 'eth2', verbose=0)
110             print "Send from ARP"
111             # sendp(y, iface = 'eth0', verbose=0)
112
113     print
114
115 def sendpkt():
116
117     global rate, buffering_time, flag
118
119     prev_time = math.floor(time.time()*rate)
120     start_time = math.floor(time.time())
121
122     while True:
123         #Changed to x81\x00 bcos of VLAN ID header from OpenStack VMs
124         head = '\xfa\x16\x3e\x7f\xd6\x75\xfa\x16\x3e\x72\x3a\x99\x81\x00'
125         if count == 1:
126             start_time = math.floor(time.time())
127         else:
128             pass
129
130         if (math.floor(time.time()) - start_time >= buffering_time) and
    (math.floor(time.time()*rate) != prev_time):
131
132             if (len(buffer1) != 0) and (len(buffer2) != 0):
133                 a = ts1[0]
134                 b = ts2[0]
135
136                 if a > b:       # send packet in buffer2
137                     bb = buffer2.pop(0)
138                     ts2.pop(0)
139                     x = Ether(head+bb[14:])
140                     try :
141                         sendp(x, iface = 'eth2', verbose=0)
142                         #print "Send from Buffer2"
143                         prev_time = math.floor(time.time()*rate)
144                         #print math.floor(time.time() - start_time), " seconds have
    passed"
145                     except socket.error :
146                         continue
147
148                 elif b > a :    # send packet in buffer1
149                     aa = buffer1.pop(0)
150                     ts1.pop(0)
151                     z = Ether(head+aa[14:])
152                     try :
153                         sendp(z, iface = 'eth2', verbose=0)
154                         #print "Send from Buffer1"
155                         prev_time = math.floor(time.time()*rate)
156                         #print math.floor(time.time() - start_time), " seconds have
    passed"
157                     except socket.error :
158                         continue
159
160                 elif a == b :    # send packet in buffer1
```

```
                    aa = buffer1.pop(0)
                    ts1.pop(0)
                    buffer2.pop(0)  # delete pkt in buffer2
                    ts2.pop(0)
                    z = Ether(head+aa[14:])
                    try :
                        sendp(z, iface = 'eth2', verbose=0)
                        #print "Send from Buffer1 and discard pkts from Buffer2"
                        prev_time = math.floor(time.time()*rate)
                        #print math.floor(time.time() - start_time), " seconds have passed"
                    except socket.error :
                        continue

            elif len(buffer1) == 0 and len(buffer2) != 0:
                bb = buffer2.pop(0)
                ts2.pop(0)
                x = Ether(head+bb[14:])
                try :
                    sendp(x, iface = 'eth2', verbose=0)
                    #print "Send out from Buffer1 when Buffer are empty"
                    prev_time = math.floor(time.time()*rate)
                    #print math.floor(time.time() - start_time), " seconds have passed"
                except socket.error :
                    continue

            elif len(buffer2) == 0 and len(buffer1) != 0:
                aa = buffer1.pop(0)
                ts1.pop(0)
                z = Ether(head+aa[14:])
                try :
                    sendp(z, iface = 'eth2', verbose=0)
                    #print "Send out from Buffer2 when Buffer are empty"
                    prev_time = math.floor(time.time()*rate)
                    #print math.floor(time.time() - start_time), " seconds have passed"
                except socket.error :
                    continue

            else :
                pass

        else :
            pass


if __name__ == "__main__":

    global rate, buffering_time
    rate = 150
    buffering_time = 0

    t1 = threading.Thread(target=main, args=(sys.argv))
    t2 = threading.Thread(target=sendpkt)

    t1.start()
    t2.start()

```

```
    t1.join()
    t2.join()
```

Listing D.1: Middle-box configuration

```
Middle-box set up on CHULA SmartX box
Reference:
http://www.howtogeek.com/117635/how-to-install-kvm-and-create-virtual-machines
-on-ubuntu/
sudo apt-get install qemu-kvm libvirt-bin bridge-utils virt-manager
.....................................................................
Create KVM by using virt-manager GUI
sudo virt-manager
Create three Virtual Network (NAT)interfaces (vnet0,vnet1,vnet2)
sudo brctl show #To check linux bridge
sudo brctl delif virbr0 vnet1 vent2
.....................................................................
This configuration is for multi-path video streaming over OF@TEIN
sudo ovs-vsctl add-port br2 vnet1
sudo ovs-vsctl add-port br2 vent2
.....................................................................
This configuration is for multi-path file transferring over OF@TEIN
sudo ovs-vsctl add-port br-devops vnet1
sudo ovs-vsctl add-port br-devops vnet2
.....................................................................
Inside created middle-box KVM
sudo apt-get install openssh-server #To open ssh connection for Mbox
sudo ufw status #To check firewall
ps -ef | grep sshd #To check ssh demo is running or not
sudo netstat -nlp | grep :22
sudo ufw allow 22
.....................................................................
Before running middle-box processing script
sudo apt-get install python-dpkt  #To install dpkt for parsing
sudo apt-get install python-pcapy #To install pcapy for capturing
sudo apt-get install scapy  #To install scapy for generating
sudo apt-get install libpcap-dev  #To install libpcap

sudo vi /etc/config/network #Change network interface eth1 and eth2
auto eth1
iface eth1 inet static
address 0.0.0.0
netmask 0.0.0.0

#Write eth2 also same configuration as eth1
Then save the network file
sudo service network-manager restart
.....................................................................
Run Middle-box processing script for multi-path video streaming
sudo python mbox_oftein.py
.....................................................................
For testing file transferring experiments
Installing Tsunami udp
Reference:
http://bluepiit.com/blog/2015/10/13/tsunami-udp-protocol-installation-setup
-and-limitations/
sudo apt-get install cvs
cvs -z3 -d:pserver:anonymous@tsunami-udp.cvs.sourceforge.net:
/cvsroot/tsunami-udp co -P tsunami-udp #continue from upper line
```

```
sudo apt-get install git gcc
gcc installation error
** solution **
sudo apt-get upgrade  and sudo apt-get update and then sudo apt-get install gcc
sudo apt-get install automake autoconf
sudo apt-get install libtool make lib32z1
cd tsunami-udp
./recompile.sh
sudo make install
Server
tsunamid --port 46224 filename

Client
tsunami connect serverip get * or filename
tsunami set rate 100M connect [server] get *
.....................................................................................
Before running file transferring experiments
Configuration in the middle-box
sudo apt-get install vlan
lsmod | grep 8021q #To check kernel module
sudo modprobe 8021q
#Add VLAN ID 111 to eth1 to be reachable network between GIST-B VM and middle-box
sudo vconfig add eth1 111
#(IP can be change according to the same network IP as GIST-B VM IP)
sudo ifconfig eth1.111 192.168.11.16 netmask 255.255.255.0
#To delete VLAN interface
sudo ifconfig eth1.111 down
sudo vconfig rem eth1.111
```

Appendix E

POX Controller Python Script for Multi-path Video Streaming over OF@TEIN

```
"""
pox_multi_video.py
This file is used for multi-path video streaming over OF@TEIN
This file is for adding flow entries to all switches. Topology connected with GIST>>MYREN>>TH
And use to split chunk video by time at GIST br2.
And use to stream multi-path and connect to GIST>>MYREN>>TH.
Phyo V1 revised to run in OF@TEIN
Reference:Panwaree, P., Kim, J., and Aswakul, C. SDN-Coordinated multi-path chunked video streaming. Master's Thesis, Department of Electrical Engineering, Chulalongkorn University, Thailand (2014).
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.packet as pkt
from threading import Timer
import time
import math

log = core.getLogger()
#fm = of.ofp_flow_mod()

class MyComponent (object):

    def __init__ (self):
        core.openflow.addListeners(self)

    def _handle_ConnectionUp (self, event):
        log.debug("Switch %s has come up.", dpid_to_str(event.dpid))
        print "Switch",event.dpid,"has come up.", dpid_to_str(event.dpid)

        if event.dpid == 1229782938247303473:  #MYREN br1 DPID 1111111111111131
            print "Install flow entry to MYREN br1",event.dpid
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        match=of.ofp_match(in_port=1)))
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        match=of.ofp_match(in_port=2)))

        elif event.dpid == 1229782938247303474: #MYREN br2 DPID 1111111111111132
              print "Install flow entry to MYREN br2",event.dpid
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=4),
                        match=of.ofp_match(in_port=3)))
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=3),
```

```
                        match=of.ofp_match(in_port=4)))

        elif event.dpid == 1229782938247303537:  #TH br1 DPID 1111111111111171
              print "Install flow entry to TH br1",event.dpid
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        match=of.ofp_match(in_port=2)))
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        match=of.ofp_match(in_port=1)))

        elif event.dpid == 1229782938247303538:  #TH br2 DPID:1111111111111172
              print "Install flow entry to TH br2",event.dpid

              event.connection.send(of.ofp_flow_mod(
                        action=(of.ofp_action_dl_addr.set_src(
                        EthAddr("e2:9c:e6:30:bf:06")),
                        of.ofp_action_output(port=10)),
                        match=of.ofp_match(in_port=3)))
              event.connection.send(of.ofp_flow_mod(
                        action=(of.ofp_action_dl_addr.set_src(
                        EthAddr("66:c2:a5:35:d2:d6")),
                        of.ofp_action_output(port=10)),
                        match=of.ofp_match(in_port=2)))
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=of.OFPP_ALL),
                        match=of.ofp_match(in_port=1)))
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        match=of.ofp_match(in_port=11)))
              event.connection.send(of.ofp_flow_mod(
                        match=of.ofp_match(in_port=10)))

        elif event.dpid == 1229782938247303441:  #GIST br1 DPID 1111111111111111
              print "Install flow entry to GIST br1",event.dpid
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        match=of.ofp_match(in_port=2)))
              event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        match=of.ofp_match(in_port=1)))

        elif event.dpid == 1229782938247303442:  #GIST-A br2 DPID:1111111111111112
            def install_path1():
                print "Install flow entry for Path 1:20sec"
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=4),
                        hard_timeout=20, match=of.ofp_match(in_port=1)))
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=1),
                        hard_timeout=20, match=of.ofp_match(in_port=4)))
            def install_path2():
                print "Install flow entry for Path 2:10sec"
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2),
                        hard_timeout=10, match=of.ofp_match(in_port=4)))
                event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=4),
                        hard_timeout=10, match=of.ofp_match(in_port=2)))
```

```
108
109             def install_flow():
110
111                 install_path1()
112                 time_int_path1 = 20
113                 time_int_path2 = 10
114                 prev_time = math.floor(time.time())
115                 current_path_installed = 1
116
117                 while True:
118
119                     if current_path_installed == 1:
120                         if math.floor(time.time())-prev_time==time_int_path1:
121                             install_path2()
122                             print math.floor(time.time())
123                             current_path_installed = 2
124                             prev_time = math.floor(time.time())
125
126                     elif current_path_installed == 2:
127                         if math.floor(time.time())-prev_time==time_int_path2:
128                             install_path1()
129                             print math.floor(time.time())
130                             current_path_installed = 1
131                             prev_time = math.floor(time.time())
132
133
134             t = Timer(10,install_flow)
135             t.start()
136
137 def launch ():
138     core.registerNew(MyComponent)
```

Listing E.1: Run controller Script

```
1 sudo ./pox.py pox-multi-video
```

Appendix F

POX Controller Python Script for Multi-path File Transferring over OF@TEIN

```
'''
pox-multi-file.py
This file is used for multi-path file transferring experiments
This file is for adding flow entries to all switches. Topology connected with GIST>>MY>>TH
And use to split chunk video file periodically  at GIST br-devops
And use to transfer  multi-path and connect to GIST>>MY>>TH.
#Phyo New revised to run in OF@TEIN New Archi on 19.4.2016
Reference:Panwaree, P., Kim, J., and Aswakul, C. SDN-Coordinated multi-path chunked video streaming. Master's Thesis, Department of Electrical Engineering, Chulalongkorn University, Thailand (2014).
'''

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.packet as pkt
from threading import Timer
import time
import math

log = core.getLogger()
#fm = of.ofp_flow_mod()

class MyComponent (object):

    def __init__ (self):
        core.openflow.addListeners(self)

    def _handle_ConnectionUp (self, event):
        log.debug("Switch %s has come up.", dpid_to_str(event.dpid))
        print "Switch",event.dpid,"has come up.", dpid_to_str(event.dpid)

        if event.dpid == 4919131752989213698:  #MY br-devops DPID 4444444444444402
            print "Install flow entry to MY br-devops",event.dpid
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=2), #Port2: vxlan_TH and Port3: vxlan-TEST
                        match=of.ofp_match(in_port=3)))
            event.connection.send(of.ofp_flow_mod(
                        action=of.ofp_action_output(port=3),
                        match=of.ofp_match(in_port=2)))

        elif event.dpid == 4919131752989213699:  #TH br2 DPID:4444444444444403
              print "Install flow entry to TH br-devops",event.dpid

              event.connection.send(of.ofp_flow_mod(
                        action=(of.ofp_action_dl_addr.set_src(
```

```
47                          EthAddr("c2:ff:3e:21:81:d4")), #TH br-devops interface of MY
    port 3, GIST-B port 2
48                          of.ofp_action_output(port=5)), #Port 5 is for mbox.vnet1
    port 7 is for mbox.vnet2
49                          match=of.ofp_match(in_port=3)))
50              event.connection.send(of.ofp_flow_mod(
51                          action=(of.ofp_action_dl_addr.set_src(
52                          EthAddr("f2:a8:b7:ea:85:63")),#TH br-devops interface of
    GIST-B port 2
53                          of.ofp_action_output(port=5)),
54                          match=of.ofp_match(in_port=2)))
55              event.connection.send(of.ofp_flow_mod(
56                          action=of.ofp_action_output(port=of.OFPP_ALL),
57                          match=of.ofp_match(in_port=5)))
58              event.connection.send(of.ofp_flow_mod(
59                          action=of.ofp_action_output(port=5),
60                          match=of.ofp_match(in_port=1)))
61  #GIST-B br-devops DPID:4444444444444401  Port2: Path 1 via MY and Port 3 Path 2
    direct to TH
62          elif event.dpid == 4919131752989213697:
63              def install_path1():
64                  print "Install flow entry for Path 1:1sec"
65                  event.connection.send(of.ofp_flow_mod(
66                          action=of.ofp_action_output(port=2),
67                          hard_timeout=1, match=of.ofp_match(in_port=1)))
68                  event.connection.send(of.ofp_flow_mod(
69                          action=of.ofp_action_output(port=1),
70                          hard_timeout=1, match=of.ofp_match(in_port=2)))
71              def install_path2():
72                  print "Install flow entry for Path 2:2sec"
73                  event.connection.send(of.ofp_flow_mod(
74                          action=of.ofp_action_output(port=1),
75                          hard_timeout=2, match=of.ofp_match(in_port=3)))
76                  event.connection.send(of.ofp_flow_mod(
77                          action=of.ofp_action_output(port=3),
78                          hard_timeout=2, match=of.ofp_match(in_port=1)))
79
80              def install_flow():
81
82                  install_path1()
83                  time_int_path1 = 1
84                  time_int_path2 = 2
85                  prev_time = math.floor(time.time())
86                  current_path_installed = 1
87
88                  while True:
89
90                      if current_path_installed == 1:
91                          if math.floor(time.time())-prev_time==time_int_path1:
92                              install_path2()
93                              print math.floor(time.time())
94                              current_path_installed = 2
95                              prev_time = math.floor(time.time())
96
97                      elif current_path_installed == 2:
98                          if math.floor(time.time())-prev_time==time_int_path2:
99                              install_path1()
100                             print math.floor(time.time())
101                             current_path_installed = 1
```

```
102                             prev_time = math.floor(time.time())
103
104
105             t = Timer(10,install_flow)
106             t.start()
107
108 def launch ():
109     core.registerNew(MyComponent)
```

Listing F.1: Run controller Script

```
1 sudo ./pox.py pox-multi-file
```

Appendix G

Openwrt Configuration for Wireless SDN

```
#!/bin/sh
#ovs-auto.sh
#This file is to configure the Open vSwitch with wlan0 in TPLINK_TL 1043 ND V2.1
#Before running this script make sure network file, firewall files are already
    changed.
#Written by Phyo 25.1.2016

OVS_LAN="ovs-ap1"  #Change to ovs-ap2 for SDN-AP2
LAN_PORT="eth1"
OVS_PORT="wlan0"
LINUX_BRIDGE="br-lan"
DPID=2222222222222201 # Change DPID 2222222222222202 for SDN-AP2
CTLIP=103.22.221.142   #Controller IP address @GIST

# Create Open vSwitch
echo "Create Open vSwitch Bridge "
sleep 2

ovs-vsctl --may-exist add-br $OVS_LAN

#Remove LAN port from Linux bridge (Network breakdown)
echo "Remove the LAN port from Linux Bridge (Network will breakdown)"
sleep 2

brctl delif $LINUX_BRIDGE $LAN_PORT

# Add LAN port to Open vSwitch (Network breakdown)
echo "Add LAN port to Open vSwitch (Network breakdown)"
sleep 2
ovs-vsctl add-port $OVS_LAN $LAN_PORT

#Make sure the switch has the correct datapath ID.
echo "Setting switch Data Path ID"
sleep 2

ovs-vsctl set bridge $OVS_LAN other-config:datapath-id=$DPID

#Configure the switch to have an OpenFlow controller. This will connect to the
    controller.
echo "Configure the switch to connect OpenFlow controller"
sleep 2

ovs-vsctl set-controller $OVS_LAN tcp:$CTLIP:6633

#If the router cannot connect to the controller, then it works as a normal switch
echo "Change OVS setting to be standalone"
sleep 2

ovs-vsctl set-fail-mode $OVS_LAN standalone

echo "Setiing OVS to be OpenFlow10 version"
ovs-vsctl set bridge $OVS_LAN protocols=OpenFlow10

```

```
# Restart network
echo "Network restart"
sleep 2

/etc/init.d/network restart
sleep 2
exit 0
```

```
#!/bin/sh
#ovs-wlan.sh
#This file is to add the wlan0 to OVS in TPLINK-TL 1043 ND V2.1.
#Written by Phyo 26.1.2016
OVS_LAN="ovs-ap1"
LAN_PORT="wlan0"
echo "Adding wlan0 to OVS"
sleep 2
for i in $LAN_PORT ; do
    PORT=$i
       ifconfig $PORT up
   ovs-vsctl add-port $OVS_LAN $PORT
done

exit 0
```

```
#!/bin/sh
#ovs-vxlan.sh
#This file is for adding vxlan tunneling to connect with CHULA SmartX box
#161.200.25.89 is the datapath IP of CHULA SmartX box
ovs-vsctl add-port ovs-ap1 vxlan_TH -- set Interface vxlan_TH type=vxlan
    options:remote_ip=161.200.25.89
```

```
#!/bin/sh
#sflow.sh
#This file is to configure sflow rule in Open vSwitch @OpenWrt routers
COLLECTOR_IP=161.200.90.79 #sflow-RT server IP address
COLLECTOR_PORT=6343
AGENT=eth0
HEADER=128
SAMPLING=64
POLLING=10

ovs-vsctl -- --id=@sflow create sFlow agent=${AGENT}
    target=\"${COLLECTOR_IP}:${COLLECTOR_PORT}\" \header=${HEADER}
    \sampling=${SAMPLING} polling=${POLLING} -- set bridge ovs-ap1 sflow=@sflow
```

Listing G.1: Open vSwitch configuration

```
Download OpenWrt firmware for TPLINK TL 1043 v2 from this link:
https://downloads.openwrt.org/chaos_calmer/15.05-rc3/ar71xx/generic/
Change the original TPLINK firmware to OpenWrt firmware
.....................................................................
After successfully install new firmware, connect to TPLINK Router via WiFi or LAN
    cable
Type 192.168.1.1 in any browser for login to Openwrt luci page to change WAN IP
    address from GUI interface
Change Password so that can use ssh for login to Openwrt router later on
(OR)
```

```
Type telnet 192.168.1.1 in the host terminal for first time login to Openwrt linux terminal
passwd yourpassword
First change only WAN IP address inside network file
Once internet is reachable, install necessary packages
opkg update
opkg install ipset # To be able to configure firewall
opkg install openvswitch
opkg install tcpdump
Change firewall rule as in the following file:
save and /etc/init.d/firewall restart
Change uhttpd rule, save and /etc/init.d/uhttpd restart
After changed firewall rule, try to login to OpenWrt via ssh root@your_WAN_IP
After that change lan interface inside network file as follows and save.
.....................................................................
Sample network configuration of Openwrt for SDN-AP1
vi /etc/config/network
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config globals 'globals'
        option ula_prefix 'fdd0:1976:a4b4::/48'

#Add new configuration to enable Open vSwitch
config interface 'lan'
        option ifname 'ovs-ap1' # Change to ovs-ap2 for SDN-AP2
        option force_link '1'
        option proto 'static'
        option ipaddr '192.168.11.2' #Change to 192.168.11.3 for SDN-AP2
        option netmask '255.255.255.0'

#Remove out default configuartion
#config interface 'lan'
#       option ifname 'eth1'
#       option force_link '1'
#       option type 'bridge'
#       option proto 'static'
#       option ipaddr '192.168.1.1'
#       option netmask '255.255.255.0'
#       option ip6assign '60'

#WAN configuration
config interface 'wan'
        option ifname 'eth0'
        option _orig_ifname 'eth0'
        option _orig_bridge 'false'
        option proto 'static'
        option ipaddr '161.200.90.120' #Change to 161.200.90.103 for SDN-AP2
        option netmask '255.255.255.128'
        option gateway '161.200.90.126'
        option dns '161.200.80.1'

config interface 'wan6'
        option ifname 'eth0'
        option proto 'dhcpv6'

```

```
config switch
        option name 'switch0'
        option reset '1'
        option enable_vlan '1'

config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '0 1 2 3 4'

config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '5 6'
.....................................................................
vi /etc/config/firewall
#Add this rule for ssh login from WAN IP address
config rule
       option src wan
       option dest_port 22
       option target ACCEPT
       option proto tcp
#Allow all ports: If not allowed, cannot connect via vxlan tunneling
config rule
       option src wan
       option proto tcpudp
       option dest_port 1024:65535
       option family ipv4
       option target ACCEPT
.....................................................................
vi /etc/config/uhttpd
Comment out to be able to login OpenWrt luci GUI webpage with WAN IP address
#option rfc1918_filter '1'
.....................................................................
Now it is time to configure Open vSwitch in Openwrt.
Run the above automatic shell scripts one by one.
sh ovs-auto.sh
sh ovs-wlan.sh
sh ovs-vxlan.sh
sh sflow.sh
```

Appendix H

THAI SmartX box Architecture, Routing and Required Ports for OpenStack

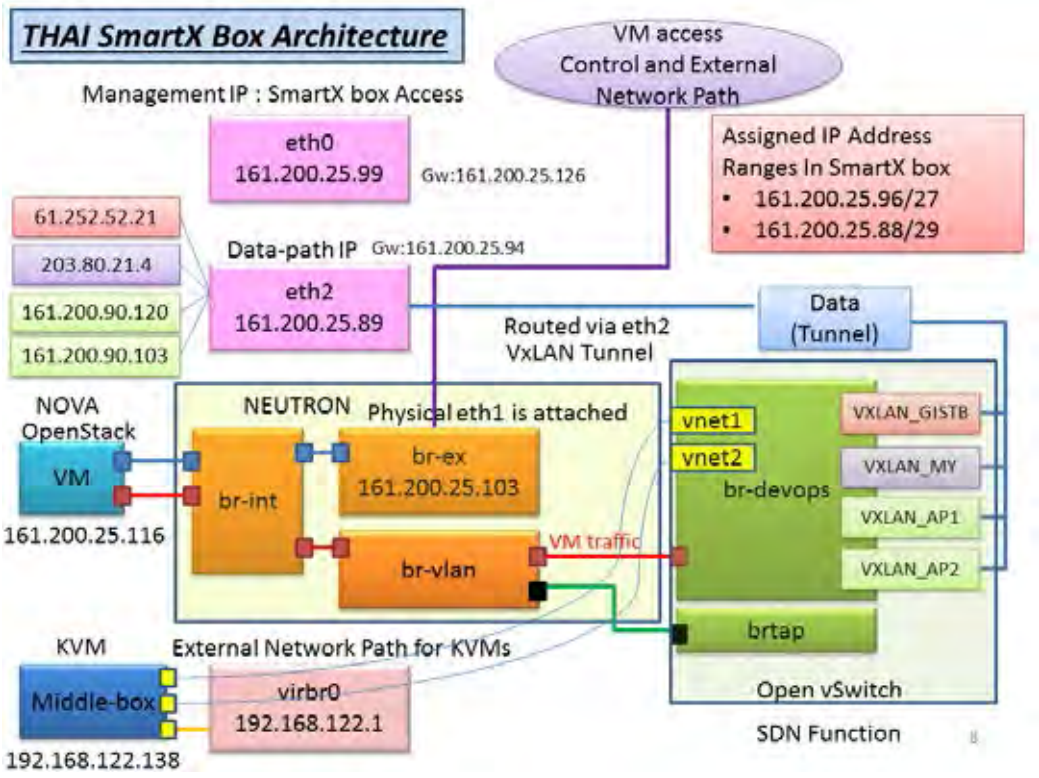


Figure H.1: THAI SmartX box architecture.

```
Adding routes inside CHULA SmartX box to connect with OpenFlow@Chula-EE SDN networks
    via tunneling
Add route for vxlan tunneling between APs and CHULA SmartX box
sudo route add -host 161.200.90.120/32 gw 161.200.25.94 dev eth2 #SDN-AP1
sudo route add -host 161.200.90.103/32 gw 161.200.25.94 dev eth2 #SDN-AP2
```

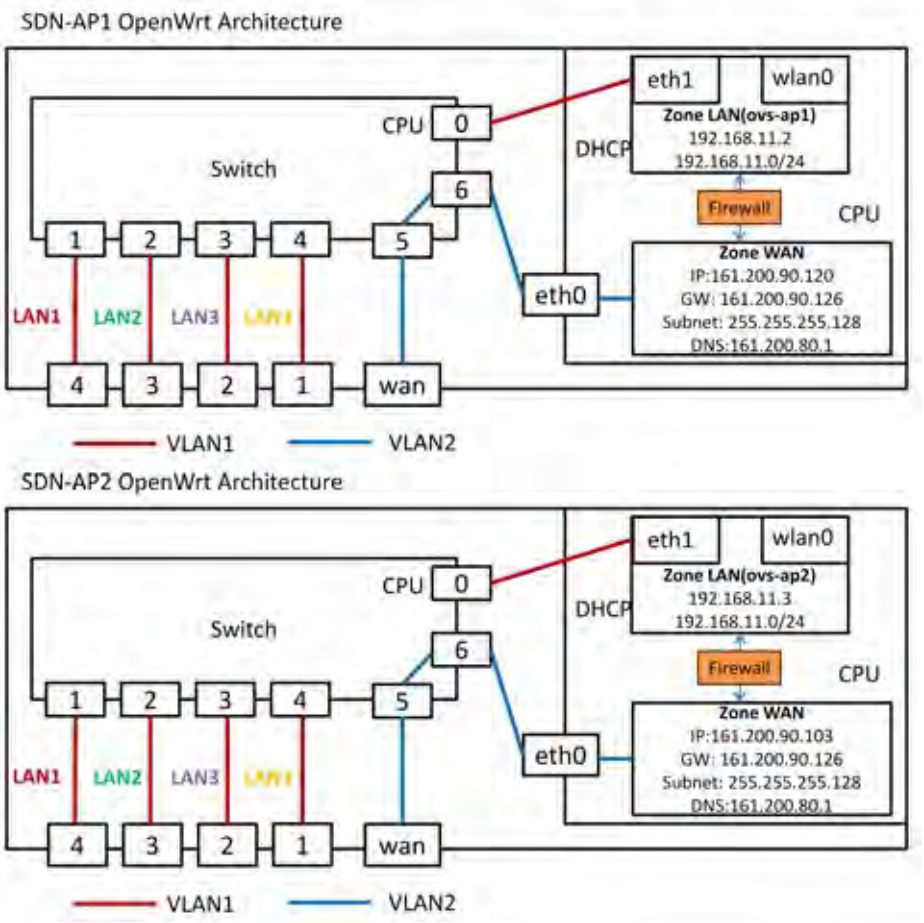


Figure H.2: OpenWrt detail architecture for SDN-AP1 and SDN-AP2.

OpenStack Service	Default ports	Port Type
Block storage (cinder)	8776	publicurl and adminurl
Compute (nova) endpoints	8774	publicurl and adminurl
Compute API (nova-api)	8773, 8775	
Compute ports for access to virtual machine consoles	5900-5999	
Compute VNC proxy for browsers (openstack-nova-novncproxy)	6080	
Compute VNC proxy for traditional VNC clients (openstack-nova-novncproxy)	6081	
Proxy port for HTML5 console used by compute service	6082	
Identity service (keystone) administrative endpoint	35357	adminurl
Identity service public endpoint	5000	publicurl
Image service (glance) API	9292	publicurl and adminurl
Image service registry	9191	
Networking (neutron)	9696	publicurl and adminurl
Object storage (swift)	6000, 6001, 6002	
Orchestration (heat) endpoint	8004	publicurl and adminurl
Orchestration AWS CloudFormation-compatible API (openstack-heat-api-cfn)	8000	
Orchestration AWS CloudWatch-compatible API (openstack-heat-api-cloudwatch)	8003	
Telemetry (ceilometer)	8777	publicurl and adminurl

Figure H.3: Required OpenStack port numbers to open firewall rules at CHULA.

Appendix I

POX Controller Python Script for Wireless SDN with and without Chunked Video Pre-transferring

```
'''
#Filename:openwrt_sdn.py by Phyo May Thet
#This file is for testing Dual WLANs with and without chunked video pre-transferring
#This file is for parsing signalling msg and on-request dynamic routing function
#Add flows for GIST-B:br-devops and CHULA:br-devops, SDN-AP1:ovs_ap1 and SDN-AP2:ovs_ap2
#Phyo V1 21.1.2016 modified on 9.4.2016
#Reference from http://sdnhub.org/tutorials/pox/
#Used:modified version of POX from this reference http://atnog.github.io/of_mobilenode/started.html
#Reference from http://sdnhub.org/tutorials/pox/
#Solution for DHCP parsed error http://ikimi.net/?p=131
#INFO:packet:(dhcp parse) warning DHCP packet data too short to parse header: data len 86
#Solution:Change the value of miss_send_len in OFPT_SET_CONFIG message from 128 bytes to be 345 inside libopenflow_01.py. In POX, it is assigned to #Macro OFP_DEFAULT_MISS_SEND_LEN=128.
#Install colorama to see color print
#Usage: sudo ./pox.py log.level --DEBUG samples.pretty_log openflow.keepalive openwrt_sdn
'''
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.packet as pkt
from threading import Timer
import time
import math
from pox.lib.revent import *
from datetime import datetime
from colorama import Fore, Back, Style

log = core.getLogger()
gist_dpid=0
th_dpid=4919131752989213699
ap1_dpid=0
ap2_dpid=0
print th_dpid
class PacketIn(Event):
    def __init__(self,connection,ofp):
        Event.__init__(self)
        self.connection = connection
        self.dpid = connection.dpid

class ConnectionDown(Event):
    def __init__(self,connection,ofp):
        Event.__init__(self)
        self.connection = connection
        self.dpid = connection.dpid
```

```
46
47 class MyComponent (object):
48
49     def __init__ (self):
50
51         core.openflow.addListeners(self)
52
53     def _handle_ConnectionUp (self, event):
54         log.debug("Switch %s has come up.", dpid_to_str(event.dpid))
55         ## dpid of ovs-ap1 2222222222222201 SDN-AP1
56         if event.dpid == 2459565876494606849:
57             event.connection.send(of.ofp_flow_mod(
58                         action=of.ofp_action_output(port=of.OFPP_LOCAL),
59                         match=of.ofp_match(in_port=2)))
60             print (Fore.YELLOW +'Install Flow entry to wlan0>>ovs-ap1')
61
62         ## dpid of ovs-ap1 2222222222222202 SDN-AP2
63         elif event.dpid == 2459565876494606850:
64             event.connection.send(of.ofp_flow_mod(
65                         action=of.ofp_action_output(port=of.OFPP_LOCAL),
66                         match=of.ofp_match(in_port=2)))
67             print (Fore.GREEN +'Install Flow entry to wlan0>>ovs-ap2')
68
69         ## dpid 4444444444444401 of SmartX GIST-B to CHULA
70         elif event.dpid == 4919131752989213697:
71             event.connection.send(of.ofp_flow_mod(
72                         action=of.ofp_action_output(port=1),
73                         match=of.ofp_match(in_port=3)))
74             event.connection.send(of.ofp_flow_mod(
75                         action=of.ofp_action_output(port=3),
76                         match=of.ofp_match(in_port=1)))
77
78         ## dpid 4444444444444403 of SmartX GIST-B br-devops to SDN-AP2 is port 6 and
    SDN-AP1 is port 8
79         elif event.dpid == 4919131752989213699:
80             event.connection.send(of.ofp_flow_mod(
81                         action=of.ofp_action_output(port=2),
82                         match=of.ofp_match(in_port=6)))
83             event.connection.send(of.ofp_flow_mod(
84                         action=of.ofp_action_output(port=2),
85                         match=of.ofp_match(in_port=8)))
86             event.connection.send(of.ofp_flow_mod(
87                         action=of.ofp_action_output(port=6),
88                         match=of.ofp_match(in_port=2)))
89
90     def _handle_ConnectionDown(self,event):
91         ConnectionDown(event.connection,event.dpid)
92         log.info("Switch %s DOWN.",dpid_to_str(event.dpid))
93
94 #This class is to scan the connected devices and instruct to install flow for that
    device in each AP.
95 class CustomFlow(object):
96     def __init__(self):
97         core.openflow.addListeners(self)
98     def _handle_PacketIn(self,event):
99         global src_ip,dst_ip,ipv4_packet,th_dpid
100
101         src_ip = []
102         dst_ip = []
```

```
103
104   PacketIn(event.connection,event.ofp)
105   packet_in = event.ofp
106   packet = event.parsed
107   src_mac = packet.src
108   dst_mac = packet.dst
109   if packet.type == of.ethernet.IP_TYPE:
110       ipv4_packet = event.parsed.find("ipv4")
111             #tcp_packet = event.parsed.find("tcp")
112             print "IPv4 parsed packet",ipv4_packet
113             #print "TCP parsed packet",tcp_packet
114       # IPv4 Packet processing
115       src_ip = ipv4_packet.srcip
116       dst_ip = ipv4_packet.dstip
117
118   match = of.ofp_match.from_packet(packet)
119         log.info(" Packet_in from: %s ",dpid_to_str(event.dpid))
120         log.info(" src_ip: %s ",src_ip)
121         log.info(" dst_ip: %s ",dst_ip)
122
123     print  (Fore.BLUE + 'start finding new client!:'), str(datetime.now())
124         if src_ip == "192.168.11.2" and dst_ip == "192.168.11.142":
125     print (Fore.YELLOW +"------- My LAB DELL PC connected to AP1 is detected!
      --------")
126                 My_flow1 (event)
127                 My_flow11 (event)
128                 print (Fore.YELLOW +" ----- Install Flow Entry for my LAB DELL PC to
      AP1 ---------")
129                 start_time = str(datetime.now())
130                 print start_time, (Fore.YELLOW +"Start Associated LAB PC on AP1")
131         elif src_ip == "192.168.11.3" and dst_ip == "192.168.11.142":
132                 print (Fore.YELLOW +"------- My LAB DELL PC connected to AP2 is
      detected! --------")
133                 My_flow2 (event)
134                 My_flow22 (event)
135                 print (Fore.YELLOW +" ----- Install Flow Entry for my LAB DELL PC to
      AP2 ---------")
136                 start_time = str(datetime.now())
137                 print start_time, (Fore.YELLOW +"Start Associated LAB PC on AP2")
138         # To change on-request dynamic routing to both APs
139         elif src_ip == "192.168.11.3" and dst_ip == "255.255.255.255":
140                 print (Fore.GREEN +"------- Found Fake DHCP request!!! --------")
141                 f = of.ofp_flow_mod()
142                 f.command=of.OFPFC_MODIFY_STRICT
143                 f.match.in_port = 2
144                 f.actions.append(of.ofp_action_output(port=of.OFPP_ALL))
145                 core.openflow.sendToDPID(th_dpid,f)
146         #To change route only to AP1
147         elif src_ip == "192.168.11.2" and dst_ip == "255.255.255.255":
148                 print (Fore.RED +"------- Found Fake DHCP request and change route
      only to AP1!!! --------")
149                 m = of.ofp_flow_mod()
150                 m.command=of.OFPFC_MODIFY_STRICT
151                 m.match.in_port = 2
152                 m.actions.append(of.ofp_action_output(port=8))
153                 core.openflow.sendToDPID(th_dpid,m)
154
155 #To add flow for allow DHCP request and to reachable b/w GIST-B IP and STA IP
156 #For allowing DHCP request (AP1)
```

```
157 def My_flow1 (event):
158   f = of.ofp_flow_mod()
159   f.match.in_port = of.OFPP_LOCAL
160   f.priority = 33001
161   f.match.dl_type = 0x0800
162   f.match.nw_dst = IPAddr("192.168.11.142")
163   f.actions.append(of.ofp_action_output( port = 2) )
164   event.connection.send( f )
165 
166   f = of.ofp_flow_mod()
167   f.match.in_port = 2
168   f.priority = 33001
169   f.match.dl_type = 0x0800
170   f.match.nw_dst = IPAddr("192.168.11.2")
171   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
172   event.connection.send( f )
173 
174   f = of.ofp_flow_mod()
175   f.match.in_port = of.OFPP_LOCAL
176   f.priority = 33001
177   f.match.dl_type = 0x0806
178   f.actions.append(of.ofp_action_output( port = 2 ) )
179   event.connection.send( f )
180 
181   f = of.ofp_flow_mod()
182   f.match.in_port = 2
183   f.priority = 33001
184   f.match.dl_type = 0x0806
185   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
186   event.connection.send( f )
187 
188 #VxLAN connection with wlan0 AP1 and SmartX Chula br-devops and specified src IP and
      dst IP
189 def My_flow11 (event):
190   f = of.ofp_flow_mod()
191   f.match.in_port = 3
192   f.priority = 33001
193   f.match.dl_type = 0x0800
194   f.match.nw_dst = IPAddr("192.168.11.142")
195   f.actions.append(of.ofp_action_strip_vlan()) #Remove VLAN header
196   f.actions.append(of.ofp_action_output( port = 2) )
197   event.connection.send( f )
198 
199   f = of.ofp_flow_mod()
200   f.match.in_port = 2
201   f.priority = 33001
202   f.match.dl_type = 0x0800
203   f.match.nw_dst = IPAddr("192.168.11.1")
204   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111)) #Add VLAN header
205   f.actions.append(of.ofp_action_output( port = 3) )
206   event.connection.send( f )
207 
208   f = of.ofp_flow_mod()
209   f.match.in_port = 3
210   f.priority = 33001
211   f.match.dl_type = 0x0806
212   f.actions.append(of.ofp_action_strip_vlan())
213   f.actions.append(of.ofp_action_output( port = 2 ) )
214   event.connection.send( f )
```

```
215
216   f = of.ofp_flow_mod()
217   f.match.in_port = 2
218   f.priority = 33001
219   f.match.dl_type = 0x0806
220   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111))
221   f.actions.append(of.ofp_action_output( port = 3 ) )
222   event.connection.send( f )
223
224 #For allowing DHCP request (AP2)
225 def My_flow2 (event):
226   f = of.ofp_flow_mod()
227   f.match.in_port = of.OFPP_LOCAL
228   f.priority = 33001
229   f.match.dl_type = 0x0800
230   f.match.nw_dst = IPAddr("192.168.11.142")
231   f.actions.append(of.ofp_action_output( port = 2) )
232   event.connection.send( f )
233
234   f = of.ofp_flow_mod()
235   f.match.in_port = 2
236   f.priority = 33001
237   f.match.dl_type = 0x0800
238   f.match.nw_dst = IPAddr("192.168.11.3")
239   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
240   event.connection.send( f )
241
242   f = of.ofp_flow_mod()
243   f.match.in_port = of.OFPP_LOCAL
244   f.priority = 33001
245   f.match.dl_type = 0x0806
246   f.actions.append(of.ofp_action_output( port = 2 ) )
247   event.connection.send( f )
248   f = of.ofp_flow_mod()
249   f.match.in_port = 2
250   f.priority = 33001
251   f.match.dl_type = 0x0806
252   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
253   event.connection.send( f )
254
255 #VxLAN connection with wlan0 AP2 and SmartX Chula br-devops and specified src IP and
      dst IP
256 def My_flow22 (event):
257   f = of.ofp_flow_mod()
258   f.match.in_port = 3
259   f.priority = 33001
260   f.match.dl_type = 0x0800
261   f.match.nw_dst = IPAddr("192.168.11.142")
262   f.actions.append(of.ofp_action_strip_vlan())
263   f.actions.append(of.ofp_action_output( port = 2) )
264   event.connection.send( f )
265
266   f = of.ofp_flow_mod()
267   f.match.in_port = 2
268   f.priority = 33001
269   f.match.dl_type = 0x0800
270   f.match.nw_dst = IPAddr("192.168.11.1")
271   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111))
272   f.actions.append(of.ofp_action_output( port = 3) )
```

```
273   event.connection.send( f )
274
275   f = of.ofp_flow_mod()
276   f.match.in_port = 3
277   f.priority = 33001
278   f.match.dl_type = 0x0806
279   f.actions.append(of.ofp_action_strip_vlan())
280   f.actions.append(of.ofp_action_output( port = 2 ) )
281   event.connection.send( f )
282
283   f = of.ofp_flow_mod()
284   f.match.in_port = 2
285   f.priority = 33001
286   f.match.dl_type = 0x0806
287   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111))
288   f.actions.append(of.ofp_action_output( port = 3 ) )
289   event.connection.send( f )
290
291 def launch ():
292   core.registerNew(CustomFlow)
293   core.registerNew(MyComponent)
```

Listing I.1: Run POX Controller for Wireless Streaming with and without Chunked Video Pre-transferring

```
1 sudo ./pox.py log.level --DEBUG samples.pretty_log openflow.keepalive openwrt_sdn
```

Appendix J

POX Controller Python Script for Wireless SDN with 100% Duplication

```
'''
#Filename:openwrt_sdn_100.py by Phyo May Thet
#This file is for testing Single WLAN with 100% duplication and Dual WLANs with 100% duplication
#This file is for parsing signalling msg and on-request dynamic routing function
#Add flows for GIST-B:br-devops and CHULA:br-devops, SDN-AP1:ovs_ap1 and SDN-AP2:ovs_ap2
#Phyo V1 21.1.2016 modified on 9.4.2016
#Reference from http://sdnhub.org/tutorials/pox/
#Used:modified version of POX from this reference http://atnog.github.io/of_mobilenode/started.html
#Reference from http://sdnhub.org/tutorials/pox/
#Solution for DHCP parsed error http://ikimi.net/?p=131
#INFO:packet:(dhcp parse) warning DHCP packet data too short to parse header: data len 86
#Solution:Change the value of miss_send_len in OFPT_SET_CONFIG message from 128 bytes to be 345 inside libopenflow_01.py. In POX, it is assigned to #Macro OFP_DEFAULT_MISS_SEND_LEN=128.
#Install colorama to see color print
#Usage: sudo ./pox.py log.level --DEBUG samples.pretty_log openflow.keepalive openwrt_sdn_100
'''
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.packet as pkt
from threading import Timer
import time
import math
from pox.lib.revent import *
from datetime import datetime
from colorama import Fore, Back, Style

log = core.getLogger()
gist_dpid=0
th_dpid=4919131752989213699
ap1_dpid=0
ap2_dpid=0
print th_dpid
class PacketIn(Event):
    def __init__(self,connection,ofp):
        Event.__init__(self)
        self.connection = connection
        self.dpid = connection.dpid

class ConnectionDown(Event):
    def __init__(self,connection,ofp):
        Event.__init__(self)
        self.connection = connection
```

```
45          self.dpid = connection.dpid
46
47  class MyComponent (object):
48
49      def __init__ (self):
50
51          core.openflow.addListeners(self)
52
53      def _handle_ConnectionUp (self, event):
54          log.debug("Switch %s has come up.", dpid_to_str(event.dpid))
55          # dpid of ovs-ap1 2222222222222201 SDN_AP1 router
56          if event.dpid == 2459565876494606849:
57              event.connection.send(of.ofp_flow_mod(
58                          action=of.ofp_action_output(port=of.OFPP_LOCAL),
59                          match=of.ofp_match(in_port=2)))
60              print (Fore.YELLOW +'Install Flow entry to wlan0>>ovs-ap1')
61
62          # dpid of ovs-lan 2222222222222202 SDN_AP2 router
63          elif event.dpid == 2459565876494606850:
64              event.connection.send(of.ofp_flow_mod(
65                          action=of.ofp_action_output(port=of.OFPP_LOCAL),
66                          match=of.ofp_match(in_port=2)))
67              print (Fore.GREEN +'Install Flow entry to wlan0>>ovs-ap2')
68
69          ## dpid 4444444444444401 of SmartX GIST-B to CHULA
70          elif event.dpid == 4919131752989213697:
71              event.connection.send(of.ofp_flow_mod(
72                          action=of.ofp_action_output(port=1),
73                          match=of.ofp_match(in_port=3)))
74              event.connection.send(of.ofp_flow_mod(
75                          action=of.ofp_action_output(port=3),
76                          match=of.ofp_match(in_port=1)))
77          ## dpid 4444444444444403 of SmartX CHULA: SDN AP2 is port 6 and SDN AP1 is
        port 8
78          elif event.dpid == 4919131752989213699:
79              event.connection.send(of.ofp_flow_mod(
80                          action=of.ofp_action_output(port=2),
81                          match=of.ofp_match(in_port=6)))
82              event.connection.send(of.ofp_flow_mod(
83                          action=of.ofp_action_output(port=2),
84                          match=of.ofp_match(in_port=8)))
85              event.connection.send(of.ofp_flow_mod(
86                          action=of.ofp_action_output(port=of.OFPP_ALL),
87                          match=of.ofp_match(in_port=2)))
88
89      def _handle_ConnectionDown(self,event):
90          ConnectionDown(event.connection,event.dpid)
91          log.info("Switch %s DOWN.",dpid_to_str(event.dpid))
92
93  #This class is to scan the connected devices and instruct to install flow for that
        device in each AP.
94  class CustomFlow(object):
95      def __init__(self):
96          core.openflow.addListeners(self)
97      def _handle_PacketIn(self,event):
98          global src_ip,dst_ip,ipv4_packet,th_dpid
99          src_ip = []
100         dst_ip = []
101
```

```
  PacketIn(event.connection,event.ofp)
  packet_in = event.ofp
  packet = event.parsed
  src_mac = packet.src
  dst_mac = packet.dst
  if packet.type == of.ethernet.IP_TYPE:
      ipv4_packet = event.parsed.find("ipv4")
            tcp_packet = event.parsed.find("tcp")
            print "IPv4 parsed packet",ipv4_packet
            print "TCP parsed packet",tcp_packet
      # IPv4 Packet processing
      src_ip = ipv4_packet.srcip
      dst_ip = ipv4_packet.dstip

  match = of.ofp_match.from_packet(packet)
        log.info(" Packet_in from: %s ",dpid_to_str(event.dpid))
        log.info(" src_ip: %s ",src_ip)
        log.info(" dst_ip: %s ",dst_ip)

    print  (Fore.BLUE + 'start finding new client!:'), str(datetime.now())
        if src_ip == "192.168.11.2" and dst_ip == "192.168.11.142":
    print (Fore.YELLOW +"------- My LAB DELL PC connected to AP1 is detected!
    --------")
                My_flow1 (event)
                My_flow11 (event)
                print (Fore.YELLOW +" ----- Install Flow Entry for my LAB DELL PC to
    AP1 ---------")
                start_time = str(datetime.now())
                print start_time, (Fore.YELLOW +"Start Associated LAB PC on AP1")

        elif src_ip == "192.168.11.3" and dst_ip == "192.168.11.142":
                print (Fore.YELLOW +"------- My LAB DELL PC connected to AP2 is
    detected! --------")
                My_flow2 (event)
                My_flow22 (event)
                print (Fore.YELLOW +" ----- Install Flow Entry for my LAB DELL PC to
    AP2 ---------")
                start_time = str(datetime.now())
                print start_time, (Fore.YELLOW +"Start Associated LAB PC on AP2")

#To add flow for allow DHCP request and to reachable b/w GIST-B IP and STA IP
#For allowing DHCP request (AP1)
def My_flow1 (event):
  f = of.ofp_flow_mod()
  f.match.in_port = of.OFPP_LOCAL
  f.priority = 33001
  f.match.dl_type = 0x0800
  f.match.nw_dst = IPAddr("192.168.11.142")
  f.actions.append(of.ofp_action_output( port = 2) )
  event.connection.send( f )

  f = of.ofp_flow_mod()
  f.match.in_port = 2
  f.priority = 33001
  f.match.dl_type = 0x0800
  f.match.nw_dst = IPAddr("192.168.11.2")
  f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
  event.connection.send( f )

```

```
157   f = of.ofp_flow_mod()
158   f.match.in_port = of.OFPP_LOCAL
159   f.priority = 33001
160   f.match.dl_type = 0x0806
161   f.actions.append(of.ofp_action_output( port = 2 ) )
162   event.connection.send( f )
163
164   f = of.ofp_flow_mod()
165   f.match.in_port = 2
166   f.priority = 33001
167   f.match.dl_type = 0x0806
168   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
169   event.connection.send( f )
170
171 #VxLAN connection with wlan0 AP1 and SmartX Chula br-devops and specified src IP and
      dst IP
172 def My_flow11 (event):
173   f = of.ofp_flow_mod()
174   f.match.in_port = 3
175   f.priority = 33001
176   f.match.dl_type = 0x0800
177   f.match.nw_dst = IPAddr("192.168.11.142")
178   f.actions.append(of.ofp_action_strip_vlan()) #Remove VLAN header
179   f.actions.append(of.ofp_action_output( port = 2) )
180   event.connection.send( f )
181
182   f = of.ofp_flow_mod()
183   f.match.in_port = 2
184   f.priority = 33001
185   f.match.dl_type = 0x0800
186   f.match.nw_dst = IPAddr("192.168.11.1")
187   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111)) #Add VLAN header
188   f.actions.append(of.ofp_action_output( port = 3) )
189   event.connection.send( f )
190
191   f = of.ofp_flow_mod()
192   f.match.in_port = 3
193   f.priority = 33001
194   f.match.dl_type = 0x0806
195   f.actions.append(of.ofp_action_strip_vlan())
196   f.actions.append(of.ofp_action_output( port = 2 ) )
197   event.connection.send( f )
198
199   f = of.ofp_flow_mod()
200   f.match.in_port = 2
201   f.priority = 33001
202   f.match.dl_type = 0x0806
203   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111))
204   f.actions.append(of.ofp_action_output( port = 3 ) )
205   event.connection.send( f )
206
207 #For allowing DHCP request (AP2)
208 def My_flow2 (event):
209   f = of.ofp_flow_mod()
210   f.match.in_port = of.OFPP_LOCAL
211   f.priority = 33001
212   f.match.dl_type = 0x0800
213   f.match.nw_dst = IPAddr("192.168.11.142")
214   f.actions.append(of.ofp_action_output( port = 2) )
```

```
215   event.connection.send( f )
216
217   f = of.ofp_flow_mod()
218   f.match.in_port = 2
219   f.priority = 33001
220   f.match.dl_type = 0x0800
221   f.match.nw_dst = IPAddr("192.168.11.3")
222   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
223   event.connection.send( f )
224
225   f = of.ofp_flow_mod()
226   f.match.in_port = of.OFPP_LOCAL
227   f.priority = 33001
228   f.match.dl_type = 0x0806
229   f.actions.append(of.ofp_action_output( port = 2 ) )
230   event.connection.send( f )
231   f = of.ofp_flow_mod()
232   f.match.in_port = 2
233   f.priority = 33001
234   f.match.dl_type = 0x0806
235   f.actions.append(of.ofp_action_output( port = of.OFPP_LOCAL ) )
236   event.connection.send( f )
237
238 #VxLAN connection with wlan0 AP2 and SmartX Chula br-devops and specified src IP and
      dst IP
239 def My_flow22 (event):
240   f = of.ofp_flow_mod()
241   f.match.in_port = 3
242   f.priority = 33001
243   f.match.dl_type = 0x0800
244   f.match.nw_dst = IPAddr("192.168.11.142")
245   f.actions.append(of.ofp_action_strip_vlan())
246   f.actions.append(of.ofp_action_output( port = 2) )
247   event.connection.send( f )
248
249   f = of.ofp_flow_mod()
250   f.match.in_port = 2
251   f.priority = 33001
252   f.match.dl_type = 0x0800
253   f.match.nw_dst = IPAddr("192.168.11.1")
254   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111))
255   f.actions.append(of.ofp_action_output( port = 3) )
256   event.connection.send( f )
257
258   f = of.ofp_flow_mod()
259   f.match.in_port = 3
260   f.priority = 33001
261   f.match.dl_type = 0x0806
262   f.actions.append(of.ofp_action_strip_vlan())
263   f.actions.append(of.ofp_action_output( port = 2 ) )
264   event.connection.send( f )
265
266   f = of.ofp_flow_mod()
267   f.match.in_port = 2
268   f.priority = 33001
269   f.match.dl_type = 0x0806
270   f.actions.append(of.ofp_action_vlan_vid(vlan_vid=111))
271   f.actions.append(of.ofp_action_output( port = 3 ) )
272   event.connection.send( f )
```

```
273
274 def launch ():
275   core.registerNew(CustomFlow)
276   core.registerNew(MyComponent)
```

Listing J.1: Run POX Controller for Wireless Streaming with 100% Duplication

```
1 sudo ./pox.py log.level --DEBUG samples.pretty_log openflow.keepalive openwrt_sdn_100
```

Appendix K

Wi-Fi STA Shell Script for Scenario 1

```
#!/bin/bash
#Written by Phyo May Thet
#sudo chmod +x monitor_onelan.sh
#To check uuid of AP: useage: nmcli con
#sudo ./monitor_onelan.sh filename
#This file is to scan the WiFi signal strenght of AP1 and AP2 and to perform client initiated handover
#Once Signal level of AP2 reach to less than or equal -68dbm, it will handover to AP1
#This file is used for testing (1)WiFi streaming using single WLAN with 100% duplication
red=‘tput setaf 1‘
green=‘tput setaf 2‘
yellow=‘tput setaf 3‘
blue=‘tput setaf 4‘
magenta=‘tput setaf 5‘
reset=‘tput sgr0‘

echo "${blue} ####Start Monitoring WiFi Signal Strength####${reset}"
#forever loop
while :;do
a=wlan0

#Grep Signal level from iwconfig of wlan0
c=$(iwconfig $a | grep 'Signal level' | cut -b 30-51)
e=$(iwconfig $a | grep 'Signal level' | cut -b 45-47)
i+=$(iwconfig $a | grep 'Signal level' | cut -b 45-47)

#Save Signal Level of AP1 and AP2 to .csv file
echo "AP2 $i+" > $1.csv

#Check AP2 signal level is less than or equal -68dbm?
if [[ "$e" -ge 68 ]];then
   echo "${green} Signal is less than or equal -68dbm now${reset}"
   echo "${green} Now Start Handover to SDN-AP1 via wlan0${reset}"
   sudo nmcli c up uuid 78ed6b02-2216-4750-a01d-cd34a9728144 iface wlan0
   echo "....................."

else
   echo "${red} Signal level is greater than -68dbm now${reset}"
   echo "....................."

fi
echo "${blue}*****WiFi $c******${reset}"

echo "......................"
sleep 1
done
```

Listing K.1: Run Wi-Fi STA Script for Wireless Streaming Scenario 1

```
sudo ./monitor_v1.sh filenameAP2 filenameAP1
```

Appendix L

Wi-Fi STA Shell Script for Scenario 2-4

```
#!/bin/bash
#Written by Phyo May Thet
#sudo chmod +x monitor_v1.sh
#To check uuid of AP: useage: nmcli con
#sudo ./monitor_v1.sh filenameAP2 filenameAP1
#This file is to scan the WiFi signal strenght of AP1 and AP2 and to perform client initiated handover
#Once Signal level of AP2 reach to less than or equal -68dbm, it will handover to AP1 first and then disconnect AP2
#This file is used for testing
#(2)WiFi streaming using dual WLANs with 100% duplication
#(3)WiFi streaming using dual WLANs with chunked video pre-transferring mechanism
#(4)WiFi streaming using dual WLANs without chunked video pre-transferring mechanism.
#In scenario 2 and 3, it sends the singal alert for on-request dynamic routing

red=`tput setaf 1`
green=`tput setaf 2`
yellow=`tput setaf 3`
blue=`tput setaf 4`
magenta=`tput setaf 5`
reset=`tput sgr0`

echo "${blue} ####Start Monitoring WiFi Signal Strength####${reset}"
#forever loop
while :;do
a=wlan0
b=wlan1

#Grep Signal level from iwconfig of wlan0 wlan1
c=$(iwconfig $a | grep 'Signal level' | cut -b 30-51)
d=$(iwconfig $b | grep 'Signal level' | cut -b 30-51)
e=$(iwconfig $a | grep 'Signal level' | cut -b 45-47)
f=$(iwconfig $b | grep 'Signal level' | cut -b 45-47)
i+=$(iwconfig $a | grep 'Signal level' | cut -b 45-47)
j+=$(iwconfig $b | grep 'Signal level' | cut -b 45-47)

#Save Signal Level of AP1 and AP2 to .csv file
echo "AP2 $i+" > $1.csv
echo "AP1 $j+" > $2.csv

#Check AP2 signal level is less than or equal -68dbm?
if [[ "$e" -ge 68 ]];then
   sudo dhcping -s 255.255.255.255 -r -v -h 00:00:00:00:00:01 # Remove this line for without chunked video pre-transferring and dual WLANs with 100% duplication
   sleep 1 # Remove this line for without chunked video pre-transferring and dual WLANs with 100% duplication
   echo "${green} Signal is less than or equal -68dbm now${reset}"
   echo "${green} Now Start Handover to SDN-AP1 via wlan1${reset}"
   sudo nmcli c up uuid f9a9afaf-0517-48ce-90b8-05a954b26f08 iface wlan1
   echo "${magenta} Disconnect wlan0 from SDN-AP2${reset}"
   nmcli dev disconnect iface wlan0   #Remove for dual WLANs with 100% duplication
   sudo dhcping -s 255.255.255.255 -r -v #Remove for dual WLANs with 100% duplication
```

```
49    echo "..................."
50
51 else
52    echo "${red} Signal level is greater than -68dbm now${reset}"
53    echo "..................."
54
55 fi
56 echo "${blue}*****SDN AP2 $c******${reset}"
57 echo "${yellow}*****SDN AP1 $d******${reset}"
58
59 echo "....................."
60 sleep 1
61 done
```

Listing L.1: Run Wi-Fi STA Script for Wireless Streaming Scenario 2-4

```
1 sudo ./monitor_onelan.sh filename
```

Appendix M

Setting Up X11 Desktop Environment for OpenStack VMs

```
Reference:
http://vandorp.biz/2012/01/installing-a-lightweight-lxdevnc-desktop
-environment-on-your-ubuntudebian-vps/#.V1Rjg74bog4

# Make sure Debian is the latest and greatest
apt-get update
apt-get upgrade
apt-get dist-upgrade

# Install X, LXDE, VPN programs

apt-get install xorg lxde-core tightvncserver xrdp

# Start VNC to create config file

tightvncserver :1

# Then stop VNC

tightvncserver -kill :1

# Edit config file to start session with LXDE:

nano ~/.vnc/xstartup

# Add this at the bottom of the file:
lxterminal &
/usr/bin/lxsession -s LXDE &

# Restart VNC

tightvncserver :1
```

Biography

Phyo May Thet was born in 1990 in Yangon, Myanmar. She received B.Eng degree in Electronic Engineering from Technological University (Thanlyin), Myanmar, in 2011. From 2012 to 2014, she worked as a telecommunication engineer in Thailand and Myanmar. She is a Master's degree student in the field of Wireless Network and Future Internet (STAR) Research Group at Department of Electrical Engineering, Chulalongkorn University, Thailand. From 2014 to present, she is a recipient of scholarship for International Graduate Students in ASEAN countries, Chulalongkorn University, Thailand. During 29 June to 7 August 2015, she has joined Networked Computing Systems (NetCS) Laboratory as a summer internship student at Gwangju Institute of Science and Technology (GIST), Korea. Her research interests include Cloud Computing, Future Internet Technology and Software Defined Networking.

List of Publications

[1] Thet, P. M., Panwaree, P., Kim, J., and Aswakul, C. Design and functionality test of chunked video streaming over emulated multi-path OpenFlow network. in Proc. of Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 12th International Conference, IEEE (June 24-26, 2015): 1-6.

[2] Thet, P. M., Tientrakul, N., Kim, J., and Aswakul, C. Emulated OpenFlow based experimental study on middle-box buffering effect for multi-path chunked video streaming. in Proc. of 30th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC) (June 29-July 2, 2015): 184-187.

[3] Risdianto, A. C., Kim, N. L., Shin, J., Bae, J., Usman, M., Ling, T. C., Panwaree, P., Thet, P. M., Aswakul, C., Thanh, N. H., Iqbal A., Javed, U., Ilyas, M. U., and Kim, J. OF@TEIN: A community efforts towards open/shared SDN-Cloud virtual playground. in Proc. of the Asia-Pacific Advanced Network 40 (August 10-14, 2015): 22-28.

[4] Risdianto, A. C., Thet, P. M., Iqbal, A., Shaari, N. A. B. M., Atluri, H. K., Nurkahfi, G. N., Wantamanee, A., Hakimi, R., Aswakul, C., Ilyas M. U., Ling, T. C., Paventhan, A., Mulyana, E., and Kim, J. Deploying and evaluating access center and its feasibility for access federation. in Prof. of the Asia-Pacific Advanced Network 41 (July 31-August 5, 2016).