การจัดหาเส้นทางแบบมัลติแคสต์ของไอพีทีวีในโครงข่ายเอสดีเอ็น/โอเพนโฟลว์

นางสาวพรณิภา รัตนาวดี

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2557

IPTV MULTICAST ROUTING IN SDN/OPENFLOW

Miss Pornnipa Rattanawadee

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 2014

หัวข้อวิทยานิพนธ์	การจัดหาเส้นทางแบบมัลติแคสต์ของไอพีทีวีในโครงข่ายเอสดีเอ็น/โอเพนโฟลว์
โดย	นางสาวพรณิภา รัตนาวดี
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.ชัยเชษฐ์ สายวิจิตร

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร.บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.สุภาวดี อร่ามวิทย์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.ชัยเชษฐ์ สายวิจิตร)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.เชาวน์ดิศ อัศวกุล)

.....กรรมการภายนอกมหาวิทยาลัย

(รองศาสตราจารย์ ดร.ภูมิพัฒ แสงอุดมเลิศ)

พรณิภา รัตนาวดี : การจัดหาเส้นทางแบบมัลติแคสต์ของไอพีทีวีในโครงข่ายเอสดีเอ็น/โอเพนโฟลว์ (IPTV MULTICAST ROUTING IN SDN/OPENFLOW) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. ดร.ชัยเชษฐ์ สายวิจิตร, 62 หน้า.

วิทยานิพนธ์นี้ได้ศึกษาการคำนวณเส้นทางของโครงข่ายโอเพนโฟลว์โดยการใช้ตัวควบคุมพอกซ์ และมีการใช้ขั้นตอนวิธีต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมในการคำนวณเส้นทาง เนื่องจากการคำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามนี้ไม่มีการพิจารณาน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์ งานวิทยานิพนธ์นี้จึงนำเสนอการพัฒนาการคำนวณเส้นทางภายในตัวควบคุมพอกซ์ให้สามารถคำนวณน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์โดยเสนอขั้นตอนวิธีไดร์คสตราและพริม เส้นทางที่ได้จากการคำนวณของตัวควบคุมพอกซ์จะถูกใช้เพื่อให้บริการไอพีทีวีแบบมัลติแคสต์ ในการทดลองได้ใช้โปรแกรมมินิเน็ทเพื่อเลียนแบบการทำงานภายในโครงข่ายโอเพนโฟลว์ โครงข่ายโอเพนโฟลว์ที่ใช้ในการทดลองประกอบด้วยตัวบริการไอพีทีวี ลูกข่ายไอพีทีวี โอเพนวีสวิตช์ และตัวควบคุมพอกซ์ โดยวิทยานิพนธ์นี้มีการเปรียบเทียบเวลาที่ใช้ในการส่งแพ็กเกตยูดีพีจากตัวบริการไอพีทีวีไปยังลูกข่ายไอพีทีวีทั้งหมดและมีการทดสอบการสตรีมวีดีทัศน์จากตัวบริการไอพีทีวีไปยังลูกข่ายไอพีทีวีด้วย นอกจากนี้เพื่อให้เห็นผลกระทบของการใช้โครงข่ายโอเพนโฟลว์ในการให้บริการไอพีทีวี งานวิจัยนี้จึงมีการทดลองเวลาที่ตัวควบคุมพอกซ์ใช้ในการเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์รวมถึงเวลาที่ใช้ในการคำนวณเส้นทางของแต่ละขั้นตอนวิธี ซึ่งจากผลการทดลองสามารถสรุปได้ว่าการคำนวณเส้นทางด้วยขั้นตอนวิธีไดร์คสตรานั้นเหมาะสมกับการให้บริการไอพีทีวีแบบยูนิแคสต์และขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริมเหมาะสมกับการให้บริการไอพีทีวีแบบมัลติแคสต์ โดยการทำงานของตัวควบคุมพอกซ์ไม่มีผลกระทบกับการให้บริการไอพีทีวีบนโครงข่ายโอเพนโฟลว์

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2557

5570553021 : MAJOR ELECTRICAL ENGINEERING
KEYWORDS: SDN/OPFNFLOW / IPTV MULTICAST

PORNNIPA RATTANAWADEE: IPTV MULTICAST ROUTING IN SDN/OPENFLOW. ADVISOR: ASST. PROF.CHAIYACHET SAIVICHIT, Ph.D., 62 pp.

This thesis has studied the route calculation of OpenFlow network by using POX controller and spanning tree algorithm in POX controller has been used to calculate routes. Since this route calculation by using spanning tree algorithm did not consider edge weight between Open vSwitches. This thesis proposed the development of route calculation within POX controller that can calculate edge weight between Open vSwitches by implementing Dijkstra's and Prim's algorithms. The route calculation of POX controller has been used to IPTV multicast service. Emulated-OpenFlow network has been implemented by using Mininet emulator in order to deploy OpenFlow IPTV multicast service. The OpenFlow network is composed of IPTV services, IPTV clients, Open vSwitches and POX controller. This thesis compares the time of sending UDP packets from IPTV server to all IPTV clients and demonstrates video streaming from IPTV server to IPTV clients. In addition to consider the impact of OpenFlow network usability of IPTV service . This thesis demonstrates the time of POX controller uses to learn the connection between Open vSwitch including the time of POX controller uses to calculate route of three algorithms. The experiment results shown that the route calculation of Dijkstra's algorithm is suitable for IPTV unicast service and Spanning tree's as well as Prim's algorithms are suitable for IPTV multicast service. The function of POX controller do not have impact for IPTV service on OpenFlow network.

Department:	Electrical Engineering	Student's Signature .....
Field of Study:	Electrical Engineering	Advisor's Signature .....
Academic Year:	2014	

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงได้ด้วยดี ต้องขอกราบขอบพระคุณ ผศ. ดร.ชัยเชษฐ์ สายวิจิตร อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งได้ให้คำแนะนำ อันเป็นประโยชน์อย่างยิ่งต่อการทำวิจัย ขอขอบพระคุณ ผศ. ดร.สุภาวดี อร่ามวิทย์ ประธานกรรมการสอบวิทยานิพนธ์ ผศ. ดร. เชาวน์ดิศ อัศวกุลและ รศ. ดร.ภูมิพัฒ แสงอุดมเลิศ กรรมการสอบวิทยานิพนธ์ ที่ได้สละเวลาตรวจสอบและให้คำแนะนำเพื่อให้วิทยานิพนธ์ฉบับนี้สมบูรณ์ยิ่งขึ้น และขอขอบพระคุณคณาจารย์ทุกท่านที่ได้ประสิทธิประสาทความรู้อันเป็นประโยชน์ต่อการทำวิทยานิพนธ์ฉบับนี้

งานวิจัยนี้ได้รับทุนสนับสนุนจากโครงการขับเคลื่อนการวิจัย กองทุนรัชดาภิเษกสมโภช (Special Task Force for Activating Research (STAR)) ภายใต้กลุ่มวิจัยโครงข่ายไร้สายและอินเทอร์เน็ตอนาคต (Wireless Network and Future Internet Research Group) จุฬาลงกรณ์มหาวิทยาลัย

ขอขอบคุณกลุ่มวิจัยโครงข่าย (Network Research Group) ซึ่งดูแลโดย ผศ. ดร. เชาวน์ดิศ อัศวกุลและ ผศ. ดร.ชัยเชษฐ์ สายวิจิตร ที่จัดกิจกรรมเพื่อส่งเสริมการเรียนรู้และการทำงานของผู้วิจัยให้มีประสิทธิภาพที่ดียิ่งขึ้น รวมถึงให้ความอนุเคราะห์อุปกรณ์เครื่องมือในการทำวิจัยแก่ผู้วิจัย ทำให้งานวิจัยนี้สำเร็จได้อย่างสะดวกราบรื่น

ขอบคุณเพื่อนพี่น้องนักวิจัยทุกคน รวมถึงเจ้าหน้าที่ บุคลากรของภาควิชาวิศวกรรมไฟฟ้า สาขาโทรคมนาคม จุฬาลงกรณ์มหาวิทยาลัย ที่ให้ความช่วยเหลือ และเป็นกำลังใจที่ดียิ่งต่อผู้วิจัยตลอดมา

สุดท้ายนี้ขอขอบพระคุณครอบครัวของผู้วิจัย ซึ่งได้ให้การสนับสนุนและเป็นกำลังใจให้แก่ผู้วิจัยตั้งแต่เริ่มการทำวิจัย จนสำเร็จการศึกษา

สารบัญ

หน้า

หน้า

หน้า

สารบัญรูปภาพ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

โครงข่ายการสื่อสารในปัจจุบันมีอุปกรณ์โครงข่ายที่เป็นหลักคือสวิตช์ (switch) ซึ่งเป็นอุปกรณ์ที่ใช้ส่งผ่านข้อมูลภายในโครงข่าย ในหนึ่งโครงข่ายประกอบด้วยอุปกรณ์สวิตช์ที่เชื่อมต่อกันอยู่เป็นจำนวนมาก โดยสวิตช์ที่ใช้อยู่ในปัจจุบันต้องเรียนรู้การเชื่อมต่อระหว่างสวิตช์ภายในโครงข่ายด้วยการทำงานของสวิตช์เอง ดังนั้นหากต้องการจัดการภายในโครงข่ายเพื่อให้ได้การสื่อสารที่มีประสิทธิภาพสูงอาจเกิดความยุ่งยากซับซ้อน จึงได้มีงานวิจัยที่เสนอแนวคิดการสร้างตัวควบคุม (controller) เพื่อเป็นศูนย์กลางในการจัดการภายในโครงข่ายการสื่อสาร โดยใช้ชื่อว่า เอสดีเอ็น (Software-Defined Networking : SDN) [1]

เอสดีเอ็น คือ สถาปัตยกรรมโครงข่ายการสื่อสารที่อนุญาตให้ระนาบควบคุม (Control Plane) แยกออกจากระนาบข้อมูล (Data Plane) ซึ่งระหว่างระนาบควบคุมและระนาบข้อมูลสื่อสารกันได้โดยผ่านโพรโทคอลโอเพนโฟลว์ (OpenFlow Protocol) ในส่วนของระนาบควบคุมคือการใช้ตัวควบคุมโอเพนโฟลว์ (OpenFlow controller) เป็นศูนย์กลางการจัดการภายในโครงข่ายการสื่อสาร และส่วนของระนาบข้อมูลคือการใช้สวิตช์โอเพนโฟลว์ (OpenFlow switch) เพื่อส่งผ่านข้อมูลจากอุปกรณ์ต้นทางไปยังอุปกรณ์ปลายทาง ดังนั้นจึงเรียกโครงข่ายการสื่อสารนี้ว่า โครงข่ายโอเพนโฟลว์ ลักษณะเด่นของสถาปัตยกรรมเอสดีเอ็นคือการมีระนาบควบคุมเป็นศูนย์กลางการจัดการภายในโครงข่ายการสื่อสาร ซึ่งง่ายต่อการควบคุมการส่งแพ็กเกตภายในโครงข่ายให้เป็นไปตามที่ผู้ออกแบบโครงข่ายต้องการ งานวิจัยที่เกี่ยวข้องกับโครงข่ายโอเพนโฟลว์ ได้มีงานวิจัยเพื่อคำนวณเส้นทางของการส่งข้อมูลภายในโครงข่ายโอเพนโฟลว์โดยใช้โพรโทคอลต้นไม้แบบทอดข้าม (spanning tree protocol) เช่น การนำเสนอตัวควบคุมต้นไม้แบบทอดข้ามมูลค่ารวมต่ำสุด (green minimum spanning tree : GreenMST) [2] ซึ่งสามารถลดการใช้พลังงานไฟฟ้าในโครงข่ายได้ด้วย โดยในตัวควบคุมได้ใช้ขั้นตอนวิธีของครูสกาล (Kruskal's algorithm) เพื่อคำนวณเส้นทางต้นไม้แบบทอดข้ามน้อยที่สุด

การให้บริการไอพีทีวี (Internet Protocol Television : IPTV) เป็นการให้บริการแพร่สัญญาณโทรทัศน์ผ่านโครงข่ายอินเทอร์เน็ตความเร็วสูงตั้งแต่ตัวบริการไอพีทีวี (IPTV server) จนถึงลูกข่ายไอพีทีวี (IPTV client) โดยลูกข่ายไอพีทีวีต้องมีการลงทะเบียนกับตัวบริการไอพีทีวีเพื่อขอเข้าถึงเนื้อหาไอพีทีวี โดยการรับชมเนื้อหาไอพีทีวีสามารถรับชมผ่านทางคอมพิวเตอร์ หรือสมาร์ตโฟนได้โดยตรง หรือถ้าต้องการรับชมผ่านทางโทรทัศน์ ลูกข่ายไอพีทีวีต้องติดตั้งกล่องแปลงสัญญาณอินเทอร์เน็ตเป็น

สัญญาณโทรทัศน์ หรือ กล่องเซตทอปบอกซ์ (Set-Top-Box : STP) เพื่อให้สามารถรับชมเนื้อหาไอพีทีวีผ่านโทรทัศน์ได้ การให้บริการไอพีทีวีแบ่งออกเป็น 2 ประเภทคือการให้บริการแบบยูนิแคสต์ (IPTV unicast) และการให้บริการแบบมัลติแคสต์ (IPTV multicast) ในงานวิจัยนี้ให้ความสนใจเฉพาะการให้บริการแบบมัลติแคสต์หรือเป็นการให้บริการรายการที่ออกอากาศตามเวลาปกติ โดยตัวบริการไอพีทีวีจะสร้างเลขที่อยู่ไอพี (IP address) เฉพาะของตัวบริการไอพีทีวีนั้น ๆ เรียกว่า เลขที่อยู่ไอพีของกลุ่มมัลติแคสต์ (IP Multicast address) และเมื่อมีผู้สนใจรับชมรายการของตัวบริการไอพีทีวีนั้น ผู้สนใจต้องส่งคำร้องขอโดยการระบุเลขที่อยู่ไอพีของกลุ่มมัลติแคสต์ของตัวบริการนั้นถึงจะสามารถรับชมรายการที่สนใจได้

งานวิจัยที่เกี่ยวกับการให้บริการไอพีทีวีด้วยโพรโทคอลมัลติแคสต์แบบดั้งเดิม [3] ได้นำเสนอเทคโนโลยีใหม่โดยอาศัยการคำนวณเส้นทางแบบต้นไม้ (tree) ในโครงข่ายแบบมัลติแคสต์ ซึ่งเป็นการบูรณาการจุดรวมข้อมูล (Rendezvous Point) ของการคำนวณเส้นทางแบบใช้ต้นไม้ร่วมกัน (shared tree) กับการอ้างอิงผู้ให้บริการ (specific source) ของเส้นทางที่สั้นที่สุด (shortest path tree) ซึ่งเป็นโพรโทคอลพื้นฐานของการส่งข้อมูลแบบมัลติแคสต์โดยอาศัยไอพี (IP multicast) ซึ่งเรียกว่า เอชที อีอาร์เอ็ม (Hybrid Tree Based Explicit Routed Multicast : HT-ERM) ผลการทดลองแสดงให้เห็นว่างานวิจัยดังกล่าวนี้สามารถปรับปรุงการควบคุมการเข้าใช้งาน (admission control) โดยสามารถลดอัตราส่วนของสมาชิกที่ไม่สามารถเข้าใช้บริการ (blocking ratio) และสามารถลดเวลาหน่วง (delay time) ของการเปลี่ยนช่องสัญญาณ (channel switching) ได้ และยังมีงานวิจัยที่นำโพรโทคอลพิม เอสเอ็ม (PIM-SM) [4] มาลดขั้นตอนที่ไม่จำเป็นของช่วงการเปลี่ยนการใช้งานจุดรวมข้อมูลไปเป็นการใช้งานเส้นทางที่สั้นที่สุดทำให้สามารถส่งแพ็กเกตข้อมูลได้ทันทีผ่านเส้นทางที่สั้นที่สุดโดยไม่ต้องใช้จุดรวมข้อมูล ซึ่งผลที่ออกมาสามารถลดการใช้จุดรวมข้อมูลหรือลดปริมาณแบนด์วิดท์ (bandwidth) ที่ใช้งานได้

เนื่องจากโพรโทคอลแบบดั้งเดิมของการให้บริการมัลติแคสต์ส่วนใหญ่ต้องอาศัยจุดรวมข้อมูลเพื่อเป็นศูนย์กลางในการคำนวณเส้นทางและการส่งข้อมูลจากตัวบริการไปยังลูกข่าย ซึ่งจุดรวมข้อมูลนี้ก็คือสวิตช์ตัวหนึ่งในโครงข่าย ข้อเสียของสวิตช์ดังที่กล่าวไปตอนต้น สวิตช์แบบดั้งเดิมมีการทำงานด้วยตัวเอง จึงต้องใช้เวลาในการรวบรวมข้อมูลของสวิตช์ทั้งโครงข่ายเพื่อประมวลผลเส้นทางที่เหมาะสมกับการส่งข้อมูล ดังนั้นจึงได้เกิดงานวิจัยเพื่อปรับปรุงข้อเสียดังกล่าวโดยการเสนอให้มีการใช้โครงข่ายโอเพนโฟลว์เพื่อให้บริการมัลติแคสต์ ยกตัวอย่างงานวิจัยที่เกี่ยวกับการใช้โครงข่ายโอเพนโฟลว์เพื่อการให้บริการไอพีทีวี เช่น การนำเสนอตัวควบคุมโอเพนโฟลว์เพื่อคำนวณเส้นทางใหม่เมื่อเส้นทางเก่ามีความคับคั้ง (congestion) ของข้อมูล [5] โดยสวิตช์ระบุความคับคั่งได้จากสถิติที่ถูกเก็บไว้เมื่อมีการส่งแพ็กเกตข้อมูล ซึ่งผลลัพธ์ที่ได้ทำให้แพ็กเกตที่ถูกทิ้งเนื่องจากความคับคั่งของข้อมูลมีปริมาณลดลง

นอกจากนี้ได้มีงานวิจัยที่เสนอวิธีการใหม่เพื่อคำนวณเส้นทางของการส่งมัลติแคสต์ภายในโครงข่ายโอเพนโฟลว์ โดยการนำเสนอมัลติโฟลว์ (Multiflow) [6] หรือตัวควบคุมโอเพนโฟลว์ที่ใช้ขั้นตอนวิธีของไดร์คสตรา (Dijkstra's algorithm) ในการคำนวณเส้นทางที่สั้นที่สุดของการส่งมัลติแคสต์ ซึ่งข้อดีของการใช้ตัวควบคุมโอเพนโฟลว์คือทำให้สามารถคำนวณเส้นทางจากสถานีเชื่อมโยงต้นทางไปยังสถานีเชื่อมโยงปลายทางได้ก่อนล่วงหน้า แต่อย่างไรก็ตามข้อเสียของงานวิจัยดังกล่าวคือการใช้ขั้นตอนวิธีของไดร์คสตราในการคำนวณเส้นทางของการส่งมัลติแคสต์เนื่องจากขั้นตอนวิธีของไดร์คสตราเป็นขั้นตอนวิธีที่เหมาะสมกับการส่งแบบยูนิแคสต์ โดยเส้นทางที่ได้เป็นเส้นทางที่สั้นที่สุดจากตัวบริการไปยังแต่ละลูกข่าย ถ้านำมาใช้กับการส่งแบบมัลติแคสต์จะทำให้ฝั่งตัวบริการสิ้นเปลืองทรัพยากรภายในโครงข่าย นอกจากขั้นตอนวิธีของไดร์คสตราที่ถูกนำเสนอในตัวควบคุมโอเพนโฟลว์แล้ว ยังมีขั้นตอนวิธีพริม (Prim's algorithm) ได้ถูกกล่าวไว้ในงานวิจัย [7] ว่าเป็นขั้นตอนวิธีที่ดีในการคำนวณเส้นทางของการให้บริการมัลติแคสต์ เนื่องจากขั้นตอนวิธีพริมมีการทำงานที่ไม่ซับซ้อนมากเมื่อเทียบกับขั้นตอนวิธีอื่น งานวิจัยนี้จึงเสนอการนำโครงข่ายโอเพนโฟลว์มาพัฒนาปรับใช้กับการคำนวณเส้นทางของการให้บริการไอพีทีวีแบบมัลติแคสต์ โดยผู้วิจัยได้สนใจศึกษาและพัฒนาการคำนวณเส้นทางต้นไม้แบบทอดข้ามในตัวควบคุมโอเพนโฟลว์ให้มีประสิทธิภาพสูงขึ้นโดยการประยุกต์ใช้ขั้นตอนวิธีไดร์คสตราและพริม ตัวควบคุมโอเพนโฟลว์ที่ถูกใช้ในงานวิจัยนี้คือ ตัวควบคุมพอกซ์ (POX controller) ซึ่งมีการโปรแกรมการทำงานด้วยภาษาไพทอน (python script)

1.2 วัตถุประสงค์ของวิทยานิพนธ์

1. เพื่อศึกษาการทำงานของการคำนวณเส้นทางของต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์
2. เพื่อพัฒนาการคำนวณเส้นทางของต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์โดยการเลือกใช้ขั้นตอนวิธีของไดร์คสตราและพริม
3. เพื่อเปรียบเทียบสมรรถนะของการให้บริการไอพีทีวีแบบมัลติแคสต์บนโครงข่ายโอเพนโฟลว์ระหว่างการคำนวณเส้นทางของต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์ ขั้นตอนวิธีไดร์คสตราและพริม

1.3 ขอบเขตวิทยานิพนธ์

1. พัฒนาโปรแกรมต้นไม้แบบทอดข้ามในตัวควบคุมพอกซ์ โดยเลือกใช้ขั้นตอนวิธีไดร์คสตราและพริม

2. ทดลองการส่งแพ็กเกตยูดีพีเพื่อวัดค่าเวลาของการขอเข้าร่วมกลุ่มมัลติแคสต์จนกระทั่งลูกข่ายไอพีทีวีได้รับแพ็กเกตยูดีพีแรกของการคำนวณเส้นทางของต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์ ขั้นตอนวิธีของไดร์คสตราและพริม
3. ทดสอบคุณภาพของการสตรีมวีดีทัศน์ด้วยการวัดค่าพีเอสเอ็นอาร์ (PSNR) ของการสตรีมวีดีทัศน์ของการรับส่งระหว่างตัวบริการไอพีทีวีและลูกข่ายไอพีทีวีด้วยเส้นทางที่ได้จาก การคำนวณทั้งสามขั้นตอนวิธี

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถปรับปรุงสมรรถนะของการคำนวณเส้นทางของต้นไม้แบบทอดข้ามในตัวควบคุมพอกซ์ได้
2. สามารถนำผลการศึกษาจากงานวิจัยมาประยุกต์ใช้กับการให้บริการไอพีทีวีในปัจจุบันได้

1.5 ประมวลวิทยานิพนธ์

บทที่ 1 บทนำ: กล่าวถึงความเป็นมาของโครงข่ายโอเพนโฟลว์และการให้บริการไอพีทีวี งานวิจัยที่เกี่ยวข้องรวมถึงแรงจูงใจที่ทำให้เกิดงานวิจัยนี้

บทที่ 2 เอสดีเอ็นและไอพีทีวี: กล่าวถึงการทำงานของเอสดีเอ็น โพรโทคอลโอเพนโฟลว์ ความรู้พื้นฐานของการให้บริการไอพีทีวีรวมถึงการส่งข้อมูลแบบมัลติแคสต์

บทที่ 3 การทำงานภายในโครงข่ายโอเพนโฟลว์จำลองการให้บริการไอพีทีวีแบบมัลติแคสต์: กล่าวถึงวิธีการและขั้นตอนการทดลองการให้บริการไอพีทีวีบนโครงข่ายโอเพนโฟลว์

บทที่ 4 การทดลองการให้บริการไอพีทีวีบนโครงข่ายโอเพนโฟลว์: กล่าวถึงการทดลองส่งแพ็กเกตยูดีพีและการสตรีมวีดีทัศน์ระหว่างตัวบริการไอพีทีวีและลูกข่ายไอพีทีวี

บทที่ 5 บทสรุปและข้อเสนอแนะ: กล่าวถึงการสรุปผลการทดลองและข้อเสนอแนะที่ควรปรับปรุงในอนาคต

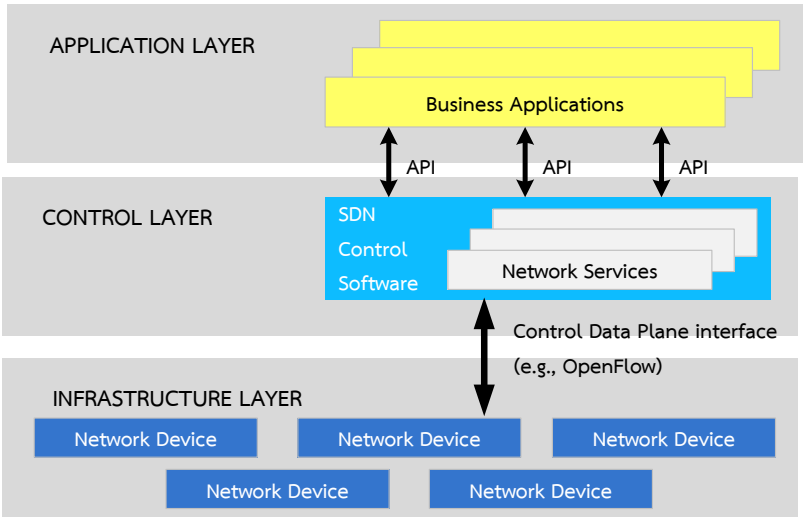
บทที่ 2
เอสดีเอ็นแลไอพีทีวี

2.1 เอสดีเอ็น (Software-Defined Networking : SDN)

การสื่อสารในปัจจุบันมีจำนวนอุปกรณ์ที่ต่อเข้ากับโครงข่ายการสื่อสารเพิ่มขึ้นจำนวนมากอีกทั้งยังมีการพัฒนาโปรแกรมประยุกต์ (application) เพื่อใช้งานร่วมกับอุปกรณ์ที่เพิ่มขึ้นดังกล่าว ซึ่งทำให้เกิดความซับซ้อนในการใช้งานโครงข่ายการสื่อสารมากขึ้น จึงนำไปสู่การวิจัยโครงข่ายการสื่อสารด้วยสถาปัตยกรรมเอสดีเอ็น

แนวคิดของเอสดีเอ็น [1] คือการแยกระนาบข้อมูลออกจากระนาบควบคุมซึ่งเป็นการเปลี่ยนแปลงแนวคิดแบบเก่าของโครงข่ายการสื่อสารที่รวมส่วนระนาบข้อมูลและระนาบควบคุมไว้ด้วยกันทำให้เกิดความซับซ้อนในการจัดเส้นทางระหว่างอุปกรณ์ภายในโครงข่ายการสื่อสาร การแยกระนาบควบคุมเพื่อให้เป็นศูนย์กลางการจัดการของโครงข่ายการสื่อสาร ทำให้การออกแบบและการเปลี่ยนแปลงการจัดเส้นทางมีความง่ายต่อผู้ออกแบบยิ่งขึ้น

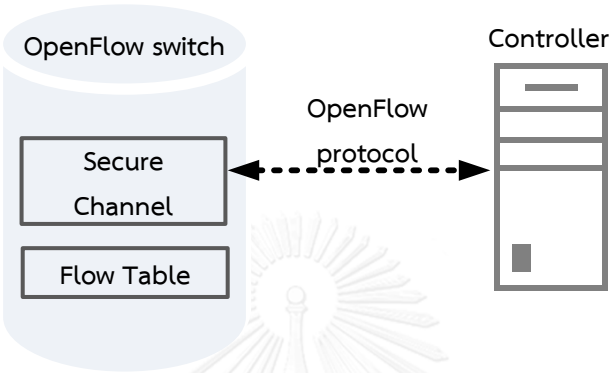
รูปที่ 2.1 แสดงให้เห็นถึงสถาปัตยกรรมเอสดีเอ็น [1] ซึ่งประกอบด้วย ชั้นของโปรแกรมประยุกต์ (application layer) ชั้นของการควบคุม (control layer) และ ชั้นของโครงสร้างพื้นฐาน (infrastructure layer) โดยชั้นของโครงสร้างพื้นฐานประกอบด้วยอุปกรณ์ภายในโครงข่ายการสื่อสาร การสื่อสารระหว่างชั้นของการควบคุมและชั้นของโครงสร้างพื้นฐานอาศัยโพรโทคอลการสื่อสาร เช่น โพรโทคอลโอเพนโฟลว์ และชั้นของโปรแกรมประยุกต์จะสื่อสารกับชั้นของการควบคุมด้วยเอพีไอ (Application Programming Interface : API)



รูปที่ 2.1: สถาปัตยกรรมเอสดีเอ็น

2.2 โครงข่ายโอเพนโฟลว์

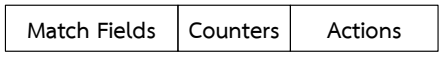
การสื่อสารด้วยโครงข่ายโอเพนโฟลว์เป็นการสื่อสารที่อาศัยตัวควบคุมโอเพนโฟลว์ในการจัดการภายในโครงข่าย โดยโครงข่ายโอเพนโฟลว์ประกอบด้วยสวิตช์โอเพนโฟลว์ที่ถูกกำหนดการส่งผ่านข้อมูลด้วยตัวควบคุมโอเพนโฟลว์โดยใช้โพรโทคอลโอเพนโฟลว์เป็นโพรโทคอลสื่อสารภายในโครงข่าย รูปที่ 2.2 แสดงการสื่อสารระหว่างอุปกรณ์พื้นฐานภายในโครงข่ายโอเพนโฟลว์ [8]



รูปที่ 2.2: การสื่อสารระหว่างสวิตช์โอเพนโฟลว์และตัวควบคุม

2.2.1 สวิตช์โอเพนโฟลว์

ความแตกต่างของสวิตช์โอเพนโฟลว์กับสวิตช์แบบดั้งเดิมคือภายในสวิตช์แบบดั้งเดิมมีทั้งระนาบควบคุมและระนาบข้อมูล แต่ภายในสวิตช์โอเพนโฟลว์มีเพียงระนาบข้อมูล ทำให้การทำงานของสวิตช์แบบดั้งเดิมมีความซับซ้อนกว่าสวิตช์โอเพนโฟลว์เนื่องจากต้องมีการประมวลผลเส้นทางก่อนการส่งผ่านข้อมูล ซึ่งสวิตช์โอเพนโฟลว์สามารถส่งผ่านข้อมูลตามที่ตัวควบคุมกำหนดไว้ได้เลย งานวิจัยนี้ได้ใช้สวิตช์โอเพนโฟลว์ชื่อว่า โอเพนวีสวิตช์ (Open vSwitch) ภายในสวิตช์โอเพนโฟลว์ ประกอบด้วย [8] ตารางโฟลว์ (flow table) และ ช่องสัญญาณแบบปลอดภัย (secure channel)



รูปที่ 2.3: องค์ประกอบหลักของโฟลว์เอนทรี

2.2.1.1 ตารางโฟลว์

ภายในตารางโฟลว์ประกอบด้วยโฟลว์เอนทรี (flow entry) เพื่อบอกให้รู้ว่าสวิตช์โอเพนโฟลว์ควรกระทำ (action) อย่างไรกับแต่ละแพ็กเกตที่รับเข้ามา องค์ประกอบหลักของโฟลว์เอนทรีแสดงดังรูปที่ 2.3 ซึ่งประกอบด้วย

- เขตข้อมูลการจับคู่ (Match field)

ใช้เพื่อจับคู่ระหว่างแพ็กเกตที่เข้ามา (incoming packet) กับเงื่อนไขที่กำหนดเพื่อให้ส่งแพ็กเกตนั้นออกไปตามช่องทางขาออก (output port) ที่กำหนดไว้ ซึ่งสามารถระบุข้อมูลเฉพาะที่ต้องการจับคู่ได้ดังรูปที่ 2.4 โดยทั้งนี้ความสามารถของการระบุข้อมูลเฉพาะขึ้นอยู่กับรุ่น (version) ของโพรโทคอลโอเพนโฟลว์ที่เลือกใช้ด้วย

Ingress Port	Metadata	Ether src	Ether dst	Ether type	VLAN id	VLAN priority	MPLS label	MPLS traffic class	IPv4 src	IPv4 dst	IPv4 proto/ARP opcode	IPv4 ToS bits	TCP/UDP/SCTP src port ICMP type	TCP/UDP/SCTP dst port ICMP code

รูปที่ 2.4: เขตข้อมูลที่สามารถใช้เพื่อการจับคู่ของแพ็กเกต

- ตัวนับ (counter)

ใช้เป็นตัวนับจำนวนแพ็กเกตที่เข้ามาในสวิตช์โอเพนโฟลว์เพื่อนำไปกำหนดเงื่อนไขการจับคู่ให้แต่ละแพ็กเกตที่เข้ามาได้

- การกระทำ (actions)

ใช้ระบุการกระทำของแต่ละแพ็กเกตที่สวิตช์โอเพนโฟลว์รับเข้ามา เช่นการส่งต่อแพ็กเกตไปยังสวิตช์โอเพนโฟลว์ถัดไปหรือละทิ้งแพ็กเกตนั้นเป็นต้น

2.2.1.2 ช่องสัญญาณแบบปลอดภัย

ช่องสัญญาณแบบปลอดภัยถูกใช้เพื่อเป็นตัวกลางการสื่อสารระหว่างสวิตช์โอเพนโฟลว์และตัวควบคุมโอเพนโฟลว์ โดยทำหน้าที่เป็นตัวรับและส่งแพ็กเกตระหว่างตัวควบคุมโอเพนโฟลว์และสวิตช์โอเพนโฟลว์เพื่อให้การส่งแพ็กเกตระหว่างสวิตช์โอเพนโฟลว์เป็นไปตามที่ตัวควบคุมโอเพนโฟลว์กำหนด

2.2.2 ตัวควบคุมโอเพนโฟลว์

การจัดการภายในโครงข่ายโอเพนโฟลว์กระทำได้ด้วยตัวควบคุมโอเพนโฟลว์ ในงานวิจัยนี้ตัวควบคุมโอเพนโฟลว์ที่ถูกใช้คือตัวควบคุมพอกซ์ที่มีการเขียนโปรแกรมด้วยภาษาไพทอน โดยตัวควบคุมพอกซ์มีหน้าที่เรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์ภายในโครงข่ายเพื่อนำมาใช้ในการคำนวณเส้นทางและกำหนดโฟลว์เอนทรีให้กับโอเพนวีสวิตช์แต่ละโนด

2.2.3 โพรโทคอลโอเพนโฟลว์

โพรโทคอลโอเพนโฟลว์ถูกใช้กำหนดการสื่อสารระหว่างระนาบควบคุมและระนาบข้อมูลของสถาปัตยกรรมเอสดีเอ็น โดยตัวควบคุมโอเพนโฟลว์เรียกใช้งานโพรโทคอลโอเพนโฟลว์ เพื่อกำหนดโฟลว์เอนทรีให้กับสวิตช์โอเพนโฟลว์แต่ละโนด

2.3 ไอพีทีวี

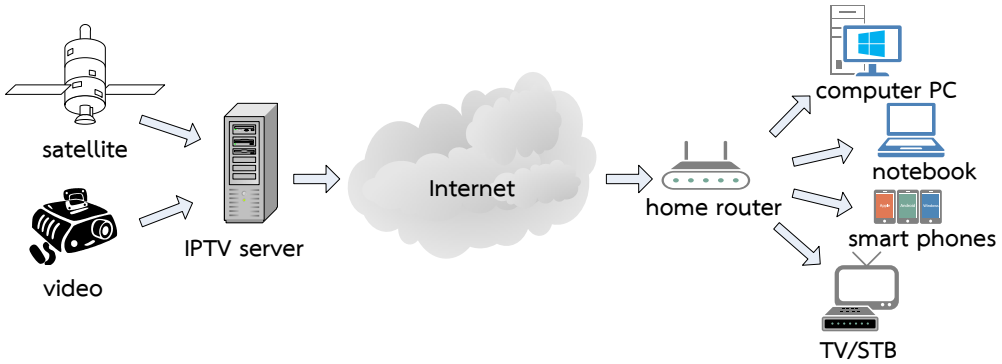
ไอพีทีวี คือ การแพร่สัญญาณโทรทัศน์ผ่านโครงข่ายอินเทอร์เน็ตความเร็วสูง ทำให้สามารถรับชมรายการโทรทัศน์ผ่านอุปกรณ์รับสัญญาณอินเทอร์เน็ตได้หลากหลายชนิด เช่น สมาร์ทโฟน สมาร์ททีวี คอมพิวเตอร์ แล็ปท็อปคอมพิวเตอร์ หรือ โทรทัศน์ เป็นต้น ในเรื่องของความคมชัดและความต่อเนื่องของสัญญาณภาพรับประกันได้เนื่องจากโครงข่ายใยแก้วนำแสง (fiber optic) ถูกใช้เป็นสื่อกลางของการแพร่สัญญาณโทรทัศน์ผ่านโครงข่ายอินเทอร์เน็ตความเร็วสูง ซึ่งใยแก้วนำแสงนี้เกิดสัญญาณรบกวนได้ยากกว่าการแพร่สัญญาณโทรทัศน์ผ่านดาวเทียม นอกจากนี้ผู้ใช้บริการสามารถเลือกค้นหาข้อมูลของรายการที่ชื่นชอบพร้อมกับการรับชมโดยผ่านบริการเชิงโต้ตอบ (interactive) ได้อีกด้วย [9] ซึ่งการให้บริการไอพีทีวีแบ่งได้ 2 ประเภทหลัก ดังนี้

- การรับชมถ่ายทอดสด (live television)

การรับชมถ่ายทอดสดหรือการให้บริการไอพีทีวีแบบมัลติแคสต์ คือ การรับชมรายการโทรทัศน์ที่ออกอากาศตามเวลาปกติ ซึ่งผู้ใช้บริการสามารถรับชมรายการโทรทัศน์ที่สนใจพร้อมกับโต้ตอบกับผู้ให้บริการได้ผ่านทางการพิมพ์ข้อความถึงผู้ให้บริการพร้อมกันได้

- การรับชมวีดิทัศน์ตามคำขอ (video on demand : VOD)

การรับชมวีดิทัศน์ตามคำขอหรือการให้บริการไอพีทีวีแบบยูนิแคสต์ คือ ผู้ใช้บริการส่งคำร้องขอเพื่อขอเข้าชมรายการที่สนใจ ซึ่งเป็นการรับชมเฉพาะผู้ที่ร้องขอเท่านั้น ผู้ใช้บริการสามารถกดหยุด (stop) หรือ พัก (pause) ได้ตามความต้องการ

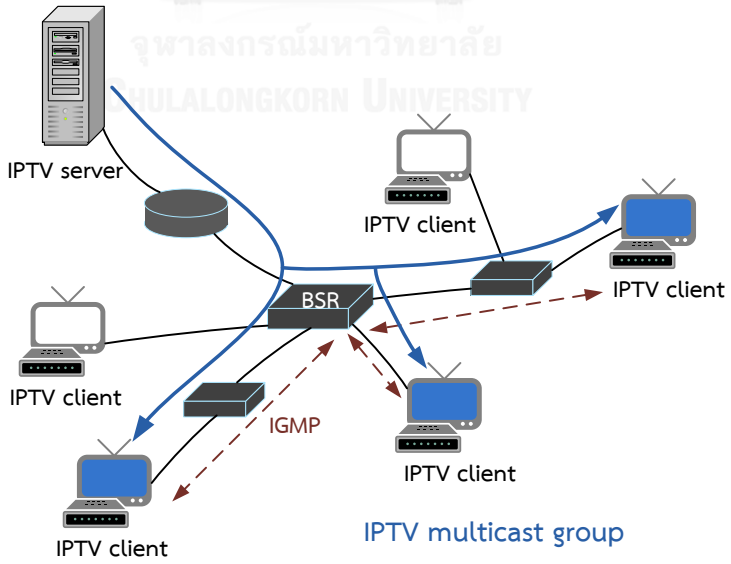


รูปที่ 2.5: ภาพรวมของการให้บริการไอพีทีวี

จากรูปที่ 2.5 แสดงภาพรวมของการให้บริการไอพีทีวี [9] ตัวบริการไอพีทีวีสามารถรับเนื้อหา (content) ที่เป็นรายการโทรทัศน์ถ่ายทอดสดจากดาวเทียมหรือเนื้อหาที่เป็นวีดีทัศน์ทั่วไปได้จากผู้ให้บริการสื่อต่างๆ เมื่อรับมาแล้วจะทำการห่อหุ้ม (encapsulate) เนื้อหาให้อยู่ในรูปแพ็กเกตของไอพี (IP packet) แล้วจึงส่งแพ็กเกตของไอพีผ่านโครงข่ายอินเทอร์เน็ตไปถึงฝั่งลูกข่ายไอพีทีวี ซึ่งฝั่งลูกข่ายไอพีทีวีมีอุปกรณ์หลักคืออุปกรณ์จัดเส้นทางภายในบ้าน (home router) เพื่อส่งแพ็กเกตของเนื้อหาไอพีทีวีไปยังอุปกรณ์รับชมต่างๆ ภายในบ้าน เช่น แล็ปท็อปคอมพิวเตอร์ สมาร์ทโฟน เป็นต้น

2.3.1 การให้บริการมัลติแคสต์ (Multicasting)

ไอพีเป็นพื้นฐานของการส่งข้อมูลแบบยูนิแคสต์ ซึ่งการส่งข้อมูลแบบยูนิแคสต์คือการส่งข้อมูลจากตัวบริการไปยังลูกข่ายแบบหนึ่งต่อหนึ่งเท่านั้น ถ้าผู้ให้บริการต้องการส่งข้อมูลชนิดเดียวกันไปยังลูกข่ายมากกว่าหนึ่งราย การส่งแบบยูนิแคสต์จะทำให้สิ้นเปลืองทรัพยากรในโครงข่าย จึงได้มีการส่งแบบมัลติแคสต์ที่เป็นการส่งข้อมูลชุดเดียวกันไปยังลูกข่ายมากกว่า 1 ราย โดยเส้นทางของการส่งจะถูกกำหนดด้วยโพรโทคอลของการส่งแบบมัลติแคสต์ โดยโพรโทคอลส่วนใหญ่มีการจัดสรรเส้นทางแบบต้นไม้ ในการให้บริการมัลติแคสต์ที่แสดงดังรูปที่ 2.6 [10] ตัวบริการไอพีทีวีจะมีไอพีของกลุ่มมัลติแคสต์ เมื่อลูกข่ายไอพีทีวีต้องการรับชมรายการของตัวบริการไอพีทีวี ลูกข่ายไอพีทีวีต้องส่งแพ็กเกตไอจีเอ็มพี (IGMP) เพื่อขอเข้าร่วมกลุ่มของตัวบริการไอพีทีวีนั้น ซึ่งสามารถเข้าร่วมกลุ่มได้พร้อมกันมากกว่าหนึ่งลูกข่ายไอพีทีวี [10] ดังนั้นการส่งข้อมูลชุดเดียวกันจากตัวบริการไอพีทีวีไปยังลูกข่ายไอพีทีวีด้วยการส่งแบบมัลติแคสต์ทำให้ประหยัดทรัพยากรภายในโครงข่ายได้ดีกว่าแบบยูนิแคสต์ [10]



รูปที่ 2.6: ภาพรวมของการให้บริการไอพีทีวีแบบมัลติแคสต์

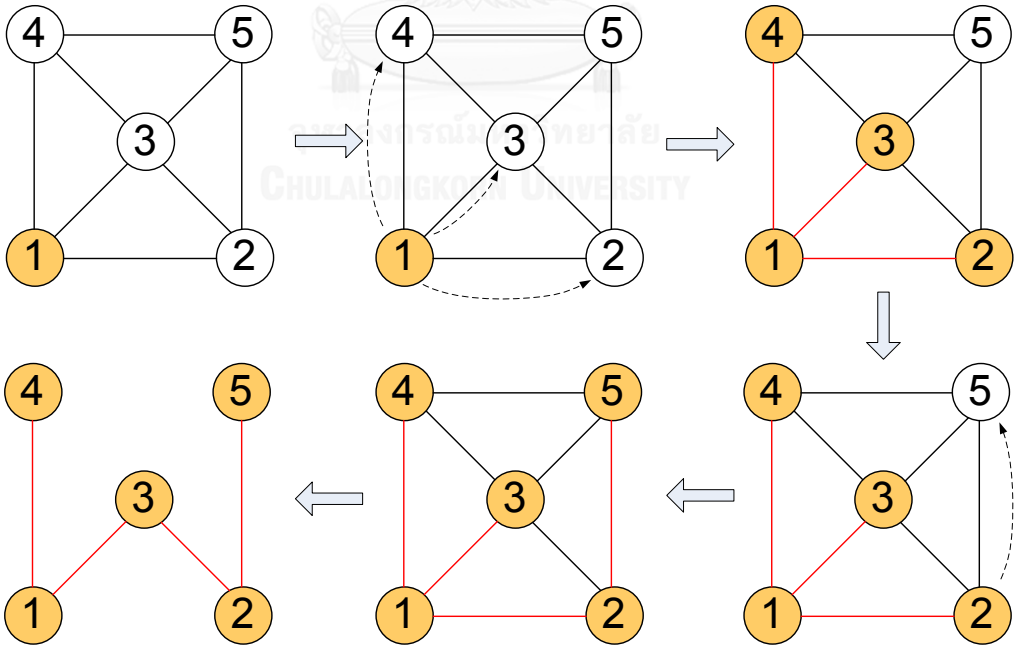
2.3.2 โพรโทคอลไอจีเอ็มพี (Internet Group Management Protocol : IGMP)

โพรโทคอลไอจีเอ็มพี [11] ใช้รายงานสถานะการเป็นสมาชิกของกลุ่มมัลติแคสต์ไปยังอุปกรณ์จัดเส้นทางแบบมัลติแคสต์ (multicast router) ที่อยู่ติดกับกลุ่มมัลติแคสต์นั้น ๆ โดยโพรโทคอลไอจีเอ็มพีรุ่นสอง (IGMP version 2 : IGMPv2) ประกอบด้วยแพ็กเกตไอจีเอ็มพี 3 ประเภท

- ไอจีเอ็มพีประเภทสอบถาม (IGMP query) ถูกส่งโดยตัวบริการไอพีทีวีเพื่อบอกให้ลูกข่ายไอพีทีวีรับรู้ว่ามีการให้บริการของตัวบริการไอพีทีวีดังกล่าว
- ไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่ม (IGMP join) ถูกส่งโดยลูกข่ายไอพีทีวีเพื่อขอเข้าร่วมกลุ่มมัลติแคสต์ของตัวบริการไอพีทีวีที่ต้องการ
- ไอจีเอ็มพีประเภทขอออกจากกลุ่ม (IGMP leave) ถูกส่งโดยลูกข่ายไอพีทีวีเพื่อขอยกเลิกการเข้าร่วมกลุ่มมัลติแคสต์

2.4 การจัดเส้นทาง (Routing)

การจัดเส้นทางคือการประมวลผลเพื่อเลือกเส้นทางที่เหมาะสมสำหรับโครงข่ายการสื่อสาร [12] ซึ่งขั้นตอนวิธีที่ใช้ในการคำนวณเส้นทางมีอยู่จำนวนมาก แต่ที่จะกล่าวถึงในงานวิจัยนี้ประกอบด้วย 3 ขั้นตอนวิธี ดังนี้



รูปที่ 2.7: การคำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้าม

2.4.1 ขั้นตอนวิธีต้นไม้แบบทอดข้าม (Spanning tree's algortihm)

ภายในตัวควบคุมพอกซ์มีโปรแกรมที่ใช้สำหรับการคำนวณเส้นทางที่ไม่ทำให้เกิดการวนซ้ำ (loop) ที่เรียกว่าต้นไม้แบบทอดข้าม ซึ่งหลักการคำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามในตัวควบคุมพอกซ์คือการพิจารณาเส้นทางจากลำดับหมายเลขของโอเพนวีสวิตช์จากโอเพนวีสวิตช์หมายเลขน้อยไปยังโอเพนวีสวิตช์หมายเลขมาก โดยมีการทำงานดังรูปที่ 2.7

2.4.1.1 เรียงลำดับหมายเลขของโอเพนวีสวิตช์จากหมายเลขน้อยไปยังหมายเลขมากและเลือกโอเพนวีสวิตช์ที่มีหมายเลขน้อยที่สุดเป็นสถานีเชื่อมโยงปัจจุบันดังนั้นสถานีเชื่อมโยงเริ่มต้นของรูปที่ 2.7 คือสถานีเชื่อมโยง 1

2.4.1.2 จากสถานีเชื่อมโยงปัจจุบันเลือกเส้นเชื่อมที่เชื่อมระหว่างสถานีเชื่อมโยงปัจจุบันไปยังสถานีเชื่อมโยงใด ๆ

2.4.1.3 พิจารณาสถานีเชื่อมโยงถัดไป โดยเลือกสถานีเชื่อมโยงที่เชื่อมกับสถานีเชื่อมโยงที่พิจารณาปัจจุบัน และมีหมายเลขน้อยสุดเป็นสถานีเชื่อมโยงถัดไป ดังนั้นสถานีเชื่อมโยงถัดไปของรูปที่ 2.7 คือสถานีเชื่อมโยง 2 แล้วทำการเลือกเส้นทางของทุกสถานีเชื่อมโยงที่มีเส้นเชื่อมกับสถานีเชื่อมโยงปัจจุบัน แต่ต้องไม่ซ้ำกับเส้นทางที่เลือกไปแล้ว

2.4.1.4 พิจารณาสถานีเชื่อมโยงถัดไปตามข้อ 2.4.1.3 จนครบทุกสถานีเชื่อมโยง ถึงได้เส้นทางที่คำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้าม

2.4.2 ขั้นตอนวิธีไดร์คสตรา (Dijkstra's algorithm)

ขั้นตอนวิธีไดร์คสตราถูกใช้เพื่อคำนวณเส้นทางที่สั้นที่สุดจากสถานีเชื่อมโยงต้นทาง (source node) ไปยังแต่ละสถานีเชื่อมโยงปลายทาง (destination node) ในโครงข่ายการสื่อสาร ในงานวิจัยนี้ขั้นตอนวิธีไดร์คสตราถูกนำมาใช้เพื่อคำนวณเส้นทางของการให้บริการไอพีทีวีแบบมัลติแคสต์ ดังนั้นสถานีเชื่อมโยงที่ถูกกำหนดให้เป็นสถานีเชื่อมโยงต้นทางจะอยู่ติดกับตัวบริการไอพีทีวี การคำนวณเส้นทางด้วยขั้นตอนวิธีไดร์คสตราแสดงได้ดังรูปที่ 2.8 อธิบายการทำงาน [13] ได้ดังนี้

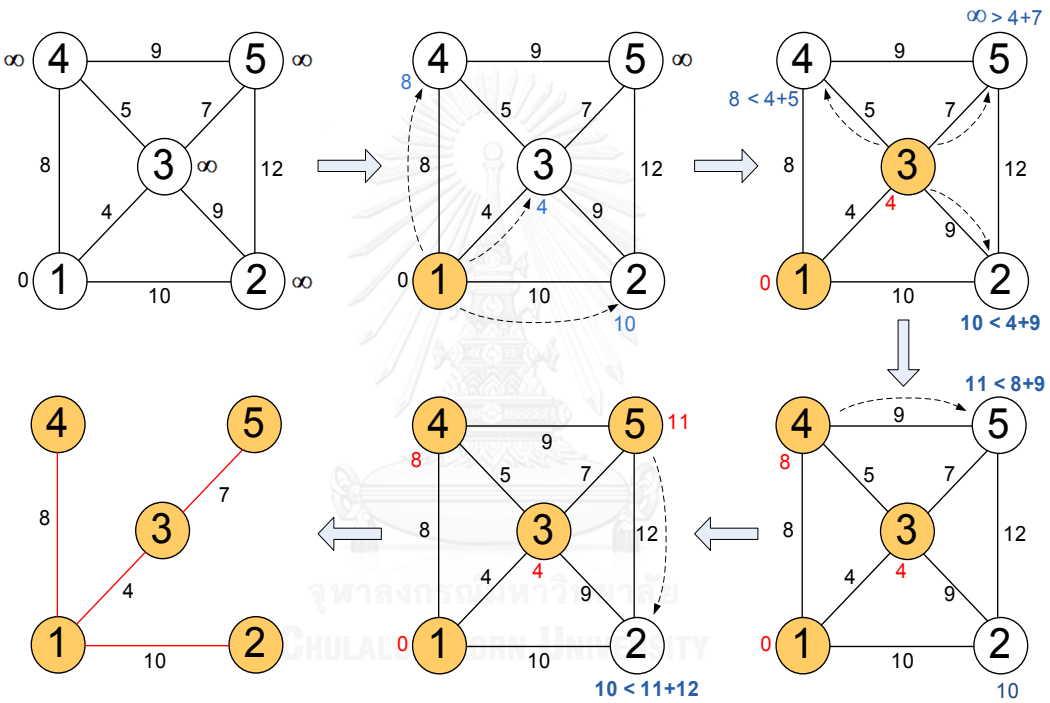
2.4.2.1 กำหนดสถานีเชื่อมโยงต้นทางเป็นสถานีเชื่อมโยงเริ่มต้น (initial node) และสถานีเชื่อมโยงอื่น ๆ เป็นสถานีเชื่อมโยงใด ๆ โดยที่กำหนดให้ทุกสถานีเชื่อมโยงมีค่าระยะทางตามค่าเวลาของแต่ละเส้นเชื่อมหรือเรียกว่า น้ำหนักของเส้นเชื่อม (edge weight) และการเริ่มต้นกำหนดให้สถานีเชื่อมโยงเริ่มต้นมีค่าระยะทางเป็นศูนย์ สถานีเชื่อมโยงอื่น ๆ มีค่าเป็นอนันต์

2.4.2.2 จากสถานีเชื่อมโยงเริ่มต้น พิจารณาสถานีเชื่อมโยงข้างเคียงตามเส้นเชื่อมทุกสถานีเชื่อมโยงที่ยังไม่ไปเยือนและคำนวณน้ำหนักของเส้นเชื่อม เช่น สถานีเชื่อมโยงเริ่มต้นปัจจุบันคือ 1 มีระยะทางของสถานีเชื่อมโยงเป็น 0 และเส้นเชื่อมที่ต่อจากสถานีเชื่อมโยง 1 ไปยังสถานีเชื่อมโยงข้างเคียง 2 3 และ 4 มีระยะทางเป็น 10 4 และ 8 ตามลำดับ ดังนั้นระยะทางของสถานีเชื่อมโยง 2

เท่ากับ 0+10=10 ระยะทางของสถานีเชื่อมโยง 3 เท่ากับ 0+4 = 4 และระยะทางของสถานีเชื่อมโยง 4 เท่ากับ 0+8 = 8 ถ้าน้ำหนักของเส้นเชื่อมที่คำนวณได้มีค่าน้อยกว่าค่าน้ำหนักของเส้นเชื่อมที่บันทึกอยู่ของสถานีเชื่อมโยงนั้น ให้เขียนทับค่าน้ำหนักของเส้นเชื่อมของสถานีเชื่อมโยงดังกล่าว

2.4.2.3 เมื่อพิจารณาสถานีเชื่อมโยงข้างเคียงจากสถานีเชื่อมโยงเริ่มต้นครบทุกสถานีเชื่อมโยงแล้ว เปลี่ยนสถานีเชื่อมโยงที่พิจารณาโดยสถานีเชื่อมโยงที่ถูกพิจารณาถัดมาจะเป็นสถานีเชื่อมโยงที่มีค่าน้ำหนักของเส้นเชื่อมน้อยที่สุด แล้วทำการพิจารณาตามขั้นตอนที่ 2.4.2.2

2.4.2.4 เมื่อพิจารณาครบทุกสถานีเชื่อมโยงแล้ว ถึงได้เส้นทางของการคำนวณด้วยขั้นตอนวิธีไดร์คสตรา



รูปที่ 2.8: การคำนวณเส้นทางด้วยขั้นตอนวิธีไดร์คสตรา

2.4.3 ขั้นตอนวิธีพริม (Prim's algorithm)

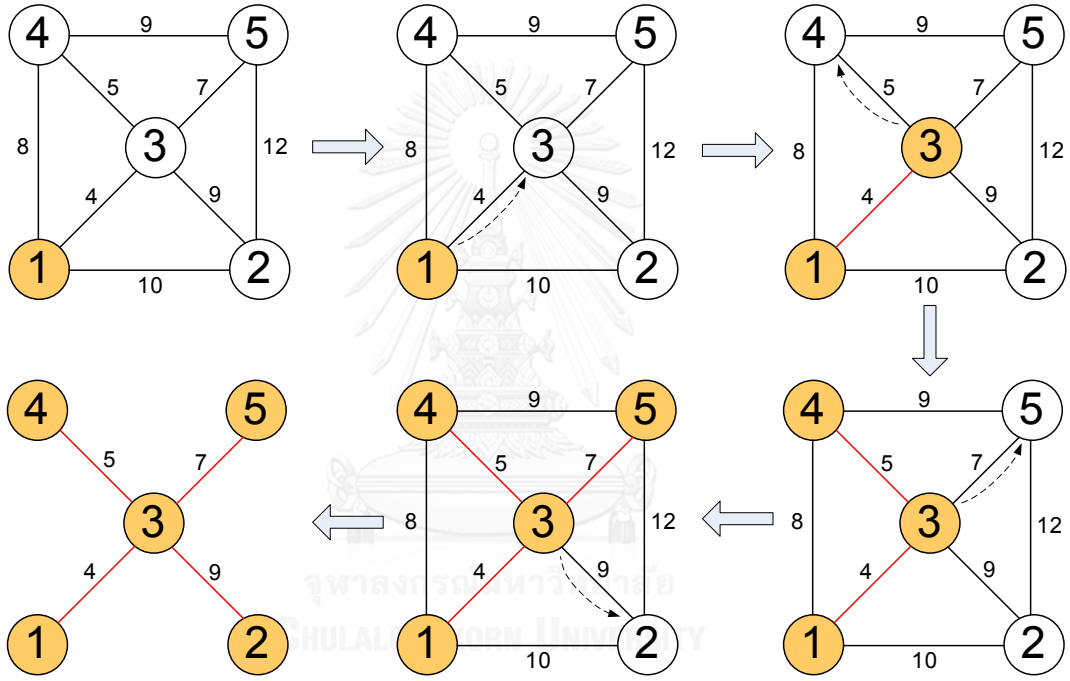
กราฟที่มีลักษณะเป็นต้นไม้แบบทอดข้ามมูลค่ารวมต่ำสุด คือ กราฟที่มีการหาเส้นเชื่อมระหว่างสถานีเชื่อมโยงต่าง ๆ ได้ทุกสถานีเชื่อมโยงและได้ระยะทางรวมน้อยที่สุด ซึ่งระยะทางคือน้ำหนักของเส้นเชื่อม ขั้นตอนวิธีพริมถูกคิดค้นเพื่อใช้คำนวณเส้นทางของต้นไม้แบบทอดข้ามมูลค่ารวมต่ำสุด การคำนวณเส้นทางด้วยขั้นตอนวิธีพริมแสดงได้ดังรูปที่ 2.9 อธิบายการทำงาน [13] ได้ดังนี้

2.4.3.1 เลือกสถานีเชื่อมโยงที่อยู่ติดกับตัวบริการไอพีทีวีเป็นสถานีเชื่อมโยงต้นทาง ซึ่งจะเรียกว่าสถานีเชื่อมโยงเริ่มต้นของการทำงาน สถานีเชื่อมโยงอื่น ๆ เรียกว่าเป็นสถานีเชื่อมโยงใด ๆ

2.4.3.2 เลือกเส้นเชื่อมที่เชื่อมระหว่างสถานีเชื่อมโยงเริ่มต้นกับสถานีเชื่อมโยงใด ๆ บนกราฟมา 1 เส้น โดยต้องมีน้ำหนักของเส้นเชื่อมน้อยที่สุด ถ้ามีเส้นเชื่อมที่น้ำหนักเท่ากันให้เลือกเส้นใดเส้นหนึ่งเท่านั้น ดังนั้นสถานีเชื่อมโยงที่ถูกเลือกลำดับแรกคือสถานีเชื่อมโยง 3 มีค่าน้ำหนักของเส้นเชื่อมเท่ากับ 4

2.4.3.3 ต่อมาพิจารณาเส้นเชื่อมที่เชื่อมระหว่างสถานีเชื่อมโยงที่ 1 ไปยังสถานีเชื่อมโยงใด ๆ และสถานีเชื่อมโยงที่ 3 ไปยังสถานีเชื่อมโยงใดๆ โดยเลือกเส้นเชื่อมที่ยังไม่ถูกพิจารณาและค่าน้ำหนักของเส้นเชื่อมน้อยที่สุด และไม่เลือกเส้นเชื่อมที่ทำให้เกิดการวนซ้ำ

2.4.3.4 เมื่อพิจารณาเส้นเชื่อมครบทุกเส้น ถึงได้เส้นทางที่คำนวณด้วยขั้นตอนวิธีพริม



รูปที่ 2.9: การคำนวณเส้นทางด้วยขั้นตอนวิธีพริม

บทที่ 3
การทำงานภายในโครงข่ายโอเพนโฟลว์จำลองการให้บริการไอพีทีวีแบบมัลติแคสต์

งานวิจัยนี้เสนอการนำโครงข่ายโอเพนโฟลว์มาพัฒนาปรับใช้กับการคำนวณเส้นทางของการให้บริการไอพีทีวีแบบมัลติแคสต์ โดยผู้วิจัยได้สนใจศึกษาและพัฒนาการคำนวณเส้นทางต้นไม้แบบทอดข้ามในตัวควบคุมพอกซ์และได้พัฒนาการคำนวณเส้นทางในตัวควบคุมพอกซ์โดยการเลือกใช้ขั้นตอนวิธีไดร์คสตราและพริม ซึ่งมีรายละเอียดที่เกี่ยวข้องกับการทดลองดังนี้

3.1 โครงข่ายโอเพนโฟลว์จำลองการให้บริการไอพีทีวี

งานวิจัยนี้ได้ทดลองการส่งแพ็กเกตและการสตรีมวีดีทัศน์บนโครงข่ายโอเพนโฟลว์ โดยโครงข่ายโอเพนโฟลว์ที่ใช้สำหรับการทดลองแสดงได้ดังรูปที่ 3.1 ประกอบด้วย

3.1.1 ตัวบริการไอพีทีวี

ในโครงข่ายจำลองนี้เป็นการทดลองเพื่อรับส่งแพ็กเกตยูดีพีและการสตรีมวีดีทัศน์ดังนั้นในการทดลองจึงต้องใช้ตัวบริการไอพีทีวีเพื่อส่งแพ็กเกตยูดีพีด้วยโปรแกรมเนเมซิส (Nemesis) [14] และสตรีมวีดีทัศน์ด้วยโปรแกรมวีแอลซี (VLC media player) [15]

3.1.2 โอเพนวีสวิตช์

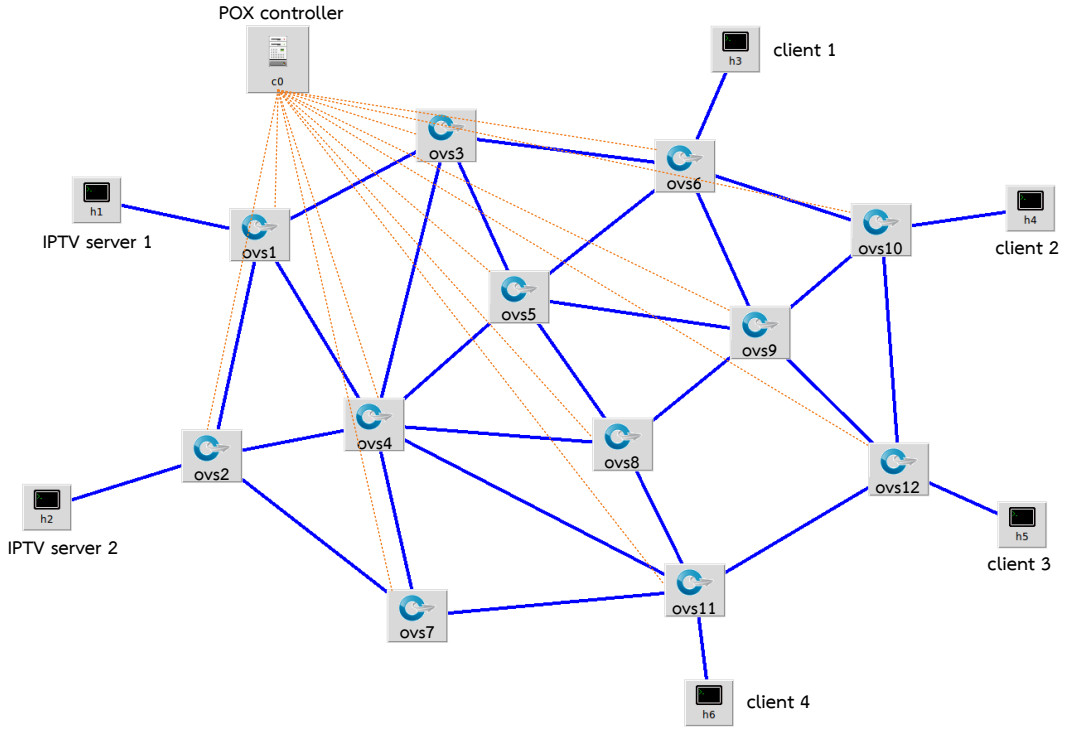
สวิตช์โอเพนโฟลว์ที่ถูกใช้ในโครงข่ายโอเพนโฟลว์จำลองนี้เรียกว่าโอเพนวีสวิตช์ ซึ่งโอเพนวีสวิตช์ถูกใช้ส่งผ่านแพ็กเกตระหว่างตัวบริการไอพีทีวีและลูกข่ายไอพีทีวี โดยกฎการส่งต่อแพ็กเกตจะถูกกำหนดโดยตัวควบคุมพอกซ์

3.1.3 ลูกข่ายไอพีทีวี

เมื่อตัวบริการไอพีทีวีทำหน้าที่ส่งแพ็กเกตยูดีพีและสตรีมวีดีทัศน์ในโครงข่ายจำลอง ลูกข่ายไอพีทีวีมีหน้าที่รับแพ็กเกตและรับวีดีทัศน์ที่ถูกสตรีมจากตัวบริการไอพีทีวี

3.1.4 ตัวควบคุมพอกซ์

ในโครงข่ายโอเพนโฟลว์จำลองนี้ ตัวควบคุมโอเพนโฟลว์ที่ถูกใช้ในการจัดการภายในโครงข่ายเรียกว่า ตัวควบคุมพอกซ์ ซึ่งทำหน้าที่คำนวณเส้นทางและระบุกฎการส่งแพ็กเกตให้กับโอเพนวีสวิตช์เพื่อการให้บริการไอพีทีวี



รูปที่ 3.1: โครงข่ายโอเพนโฟลว์จำลองการให้บริการไอพีทีวี

3.2 แผนภาพวิธีการทำงานของไอพีทีวีแบบมัลติแคสต์

งานวิจัยนี้ได้ประยุกต์ใช้แบบจำลองมัลติโฟลว์ซึ่งถูกเสนอไว้ในงานวิจัยที่ [6] โดยผู้วิจัยได้สนใจในส่วนของการคำนวณเส้นทางของแบบจำลองดังกล่าว โดยงานวิจัยดังกล่าวนี้นำเสนอเพียงขั้นตอนวิธีแบบไดร์คสตรา ในงานวิจัยนี้จึงนำเสนอการคำนวณเส้นทางด้วยขั้นตอนวิธีแบบไดร์คสตราและพริมโดยประยุกต์ใช้กับโปรแกรมต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์ แผนภาพจำลองการทำงานของไอพีทีวีแบบมัลติแคสต์แสดงดังรูปที่ 3.2 [6] แบ่งการทำงานเป็น 3 ส่วน ดังนี้

3.2.1 การสอบถาม

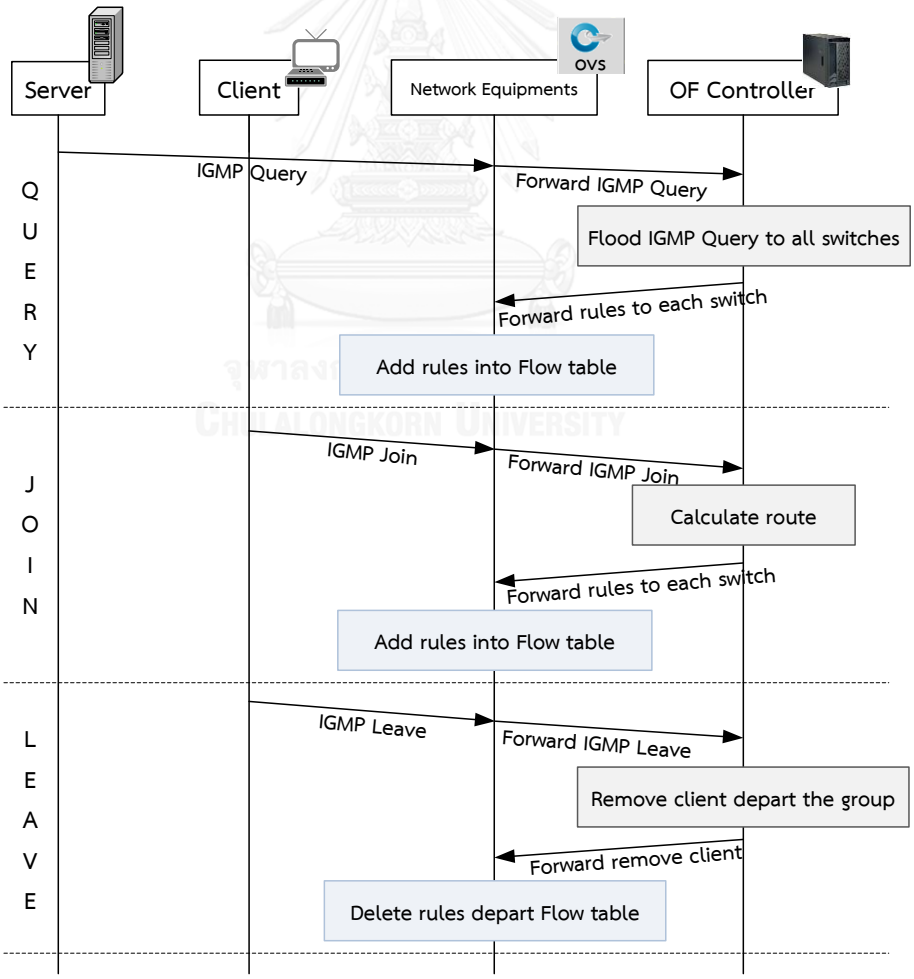
ตัวบริการไอพีทีวีส่งแพ็กเกตไอจีเอ็มพีประเภทสอบถามเพื่อบ่งบอกให้ลูกข่ายไอพีทีวีรับรู้การให้บริการของตัวบริการนั้น ซึ่งการส่งแพ็กเกตไอจีเอ็มพีประเภทสอบถามเป็นการประกาศเลขที่อยู่ไอพีของตัวบริการไอพีทีวีให้ลูกข่ายไอพีทีวีได้รับรู้ โดยเลขที่อยู่ไอพีจะแตกต่างกันตามกลุ่มมัลติแคสต์ แพ็กเกตไอจีเอ็มพีประเภทสอบถามจะถูกส่งไปยังอุปกรณ์ในโครงข่ายหรือโอเพนวีสวิตช์ ซึ่งในการส่งแพ็กเกตครั้งแรก ตารางโฟลว์ของโอเพนวีสวิตช์จะยังไม่มีโฟลว์เอนทรีของแพ็กเกตนั้น โอเพนวีสวิตช์จึงต้องส่งต่อแพ็กเกตไปยังตัวควบคุมพอกซ์เพื่อให้ตัวควบคุมพอกซ์คำนวณเส้นทางของแพ็กเกตนั้น โดยเส้นทางของแพ็กเกตไอจีเอ็มพีประเภทสอบถามที่ได้จะเป็นการส่งแพ็กเกตออกไปทุกช่องทางของโอเพนวีสวิตช์ แต่ยกเว้นช่องทางขาเข้า (input port) เพื่อให้ถึงลูกข่ายไอพีทีวีทุก ๆ ตัวในโครงข่าย

3.2.2 การเข้าร่วมกลุ่ม

เมื่อลูกข่ายไอพีทีวีได้รับแพ็กเกตไอจีเอ็มพีประเภทสอบถาม และสนใจเข้าร่วมกลุ่มมัลติแคสต์ของตัวบริการไอพีทีวี ลูกข่ายไอพีทีวีจะส่งแพ็กเกตไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่มไปยังตัวบริการนั้น โดยส่งแพ็กเกตผ่านโอเพนวีสวิตช์ตัวที่อยู่ติดกับลูกข่ายไอพีทีวี เพื่อให้ส่งต่อแพ็กเกตไปยังตัวควบคุมพอกซ์ เมื่อตัวควบคุมพอกซ์ได้รับแพ็กเกตไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่ม ตัวควบคุมพอกซ์จะทำการคำนวณเส้นทางด้วยโปรแกรมของแต่ละขั้นตอนวิธีให้กับแพ็กเกตยูดีพีที่จะถูกส่งมาจากตัวบริการไอพีทีวี

3.2.3 การออกจากกลุ่ม

เมื่อลูกข่ายไอพีทีวีต้องการยกเลิกการเข้าร่วมกลุ่มมัลติแคสต์ ลูกข่ายไอพีทีวีต้องทำการส่งแพ็กเกตไอจีเอ็มพีประเภทขอออกจากกลุ่มไปยังโอเพนวีสวิตช์ เมื่อตัวควบคุมพอกซ์ได้รับแพ็กเกตจะทำการลบโฟลว์เอนทรีของลูกข่ายไอพีทีวีในโอเพนวีสวิตช์ออกทั้งหมด



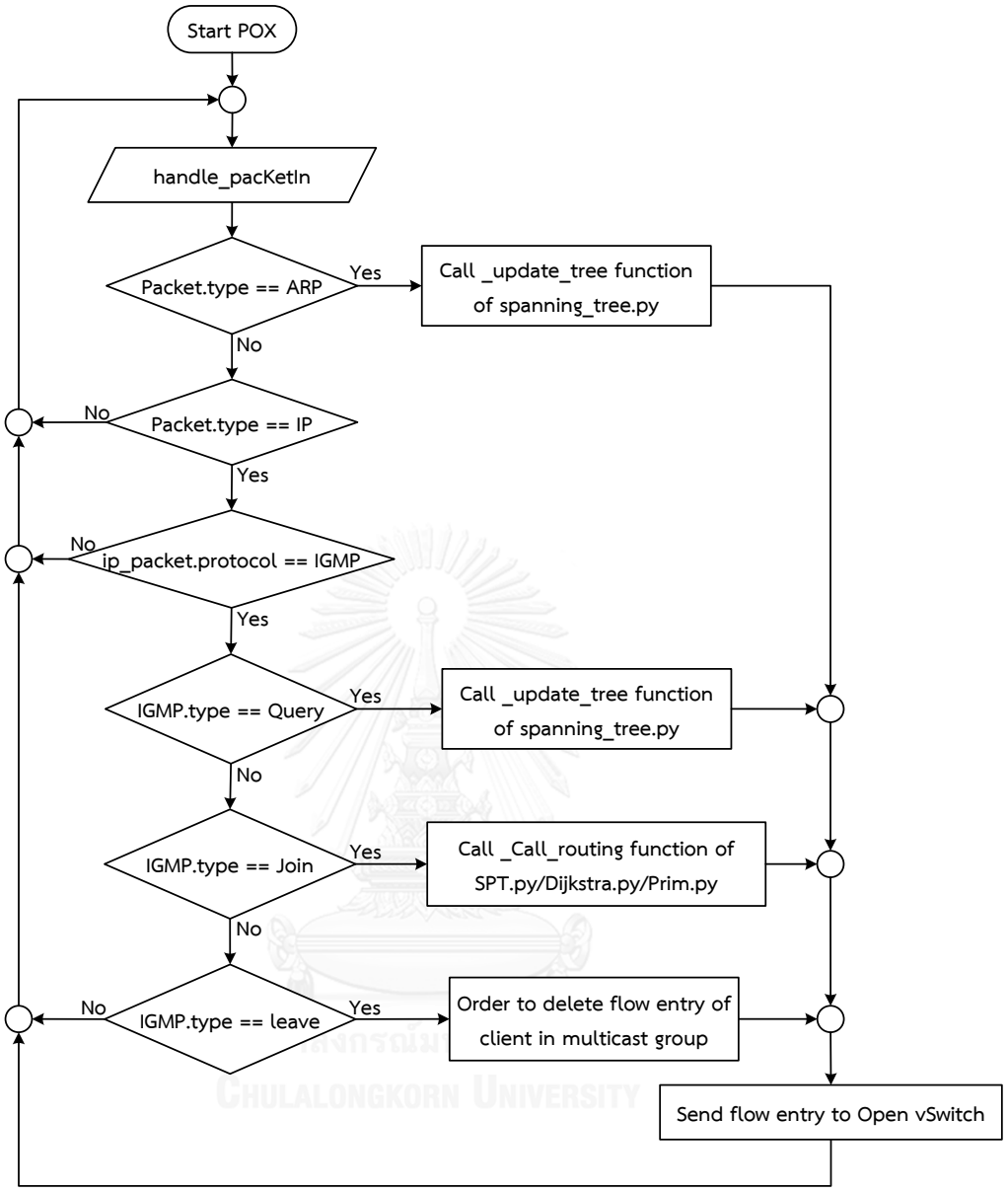
รูปที่ 3.2: แผนภาพวิธีการทำงานของไอพีทีวีแบบมัลติแคสต์

3.3 หลักการทำงานของโปรแกรมในตัวควบคุมพอกซ์

งานวิจัยนี้ได้ศึกษาการคำนวณเส้นทางของโปรแกรมต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์และได้พัฒนาการคำนวณเส้นทางของต้นไม้แบบทอดข้ามโดยการเลือกใช้ขั้นตอนวิธีไดร์คสตราและพริม เพื่อให้ได้การคำนวณเส้นทางที่มีการพิจารณาน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์ โปรแกรมที่เขียนในตัวควบคุมพอกซ์แบ่งออกเป็น 2 โปรแกรมคือ โปรแกรมการจัดการและโปรแกรมการคำนวณ ซึ่งโปรแกรมการจัดการมีหน้าที่เรียกใช้โปรแกรมการคำนวณและจัดการเพิ่มโฟลว์เอนทรีให้กับโอเพนวีสวิตช์ โปรแกรมการคำนวณมีหน้าที่คำนวณเส้นทางของทั้งสามขั้นตอนวิธี โดยโปรแกรมการจัดการของทั้งสามขั้นตอนวิธีมีชื่อว่า POX.manage.py โดยมีการเรียกใช้โปรแกรมการคำนวณของทั้งสามขั้นตอนวิธีแสดงได้ดังโปรแกรมที่ ค.3 บรรทัดที่ 94-96 สามารถเขียนเป็นผังงาน (flow chart) การทำงานของโปรแกรมการจัดการได้ดังรูปที่ 3.3เมื่อมีการดำเนินการ (run) โปรแกรมจะเริ่มทำงานที่ฟังก์ชัน (function) launch() ซึ่งฟังก์ชัน launch() จะดำเนินการเรียกใช้งานฟังก์ชัน handle_PacketIn ในฟังก์ชันนี้จะทำงานเมื่อได้รับแพ็กเกตมาจากโอเพนวีสวิตช์ โดยเริ่มจากการพิจารณาประเภทของแพ็กเกต คำสั่งที่ใช้ในการตรวจสอบประเภทของแพ็กเกตนั้นแตกต่างกัน ถ้าเป็นแพ็กเกตประเภทเออาร์พี (ARP packet) จะใช้คำสั่ง packet.type==packet.ARP_TYPE ซึ่งแพ็กเกตเออาร์พีถูกใช้เพื่อค้นหาโฮสต์ (host) ที่มีเลขที่อยู่ไอพีตามที่ระบุไว้ในแพ็กเกตเออาร์พี ดังนั้นเมื่อแพ็กเกตเออาร์พีถูกส่งมาที่ตัวควบคุมพอกซ์ ตัวควบคุมพอกซ์จะทำการเรียกใช้งานฟังก์ชัน update_tree ของโปรแกรม spanning_tree.py [16] เพื่อให้คำนวณเส้นทางของแพ็กเกตเออาร์พี โปรแกรม spanning_tree.py เป็นโปรแกรมต้นแบบที่มีอยู่ในตัวควบคุมพอกซ์ ซึ่งคำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามที่พิจารณาเส้นทางจากลำดับของหมายเลขโอเพนวีสวิตช์จากหมายเลขน้อยไปยังโอเพนวีสวิตช์หมายเลขมาก หรือถ้าเป็นแพ็กเกตไอจีเอ็มพีซึ่งอยู่ในประเภทแพ็กเกตไอพีต้องใช้คำสั่ง packet.type==packet.IP_TYPE ตรวจสอบก่อนแล้วจึงใช้คำสั่ง ip_packet.protocol==ipv4.IGMP_PROTOCOL เพื่อพิจารณาว่าเป็นแพ็กเกตไอจีเอ็มพี จากนั้นจะทำการตรวจสอบชนิดของแพ็กเกตไอจีเอ็มพีด้วยคำสั่ง igmp_packet.ver_and_type เพื่อตรวจสอบประเภทของแพ็กเกตจากหมายเลขของแพ็กเกต โดยการจัดการแพ็กเกตไอจีเอ็มพีแต่ละประเภทอธิบายได้ดังนี้

3.3.1 แพ็กเกตไอจีเอ็มพีประเภทสอบถาม

เมื่อตัวควบคุมพอกซ์ตรวจสอบว่าเป็นแพ็กเกตไอจีเอ็มพีหมายเลข 17 ซึ่งเป็นแพ็กเกตไอจีเอ็มพีประเภทสอบถามที่ถูกส่งมาจากโฮสต์ที่เป็นตัวบริการไอพีทีวี ตัวควบคุมพอกซ์จะทำการเรียกใช้งานฟังก์ชัน update_tree ของโปรแกรม spanning_tree.py และใช้เส้นทางที่คำนวณได้เพื่อส่งแพ็กเกตไอจีเอ็มพีประเภทสอบถามไปยังโฮสต์ที่เป็นลูกข่ายไอพีทีวีทุกลูกข่ายในโครงข่าย



รูปที่ 3.3: ผังงานการทำงานของตัวควบคุมพอกซ์

3.3.2 แพ็กเกตไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่ม

เมื่อตัวควบคุมพอกซ์ตรวจสอบว่าเป็นแพ็กเกตไอจีเอ็มพีหมายเลข 18 ซึ่งเป็นแพ็กเกตไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่มที่ถูกส่งมาจากโฮสต์ที่เป็นลูกข่ายไอพีทีวีเพื่อขอเข้าร่วมกลุ่มมัลติแคสต์ของตัวบริการไอพีทีวี ตัวควบคุมพอกซ์จะทำการคำนวณเส้นทางให้กับแพ็กเกตยูดีพีที่จะถูกส่งมาจากตัวบริการไอพีทีวีด้วยโปรแกรมของสามขั้นตอนวิธีที่ได้นำเสนอ โดยตัวควบคุมพอกซ์เข้าใช้งานฟังก์ชัน Call_routing เพื่อเรียกใช้งานฟังก์ชัน update_tree ของโปรแกรมคำนวณเส้นทางทั้งสามโปรแกรม เมื่อการคำนวณเส้นทางเสร็จสิ้นจะได้เป็นเซตของหมายเลขช่องทางของโอเพนวีสวิตช์ที่เชื่อมต่อกัน

ออกมา ซึ่งจะนำมาใช้ในการเพิ่มโฟลว์เอนทรีให้กับตารางโฟลว์ของโอเพนวีสวิตช์แต่ละโนด ในการเพิ่มโฟลว์เอนทรีได้เลือกใช้คำสั่ง of.ofp_flow_mod เพื่อให้มีการพิจารณาช่องทางขาเข้า (in_port) และประเภทของแพ็กเกตซึ่งแพ็กเกตยูดีพีนี้ใช้คำสั่ง dl_type=0x0800 และ nw_proto=17 ถ้าแพ็กเกตยูดีพีที่ถูกส่งมาจากตัวบริการไอพีทีวีเข้ามาตรงตามเงื่อนไขที่กำหนด แพ็กเกตยูดีพีนั้นจะถูกส่งออกไปยังช่องทางขาออกที่กำหนดไว้ด้วยคำสั่ง action=of.ofp_action_output(port=[ ]) ถ้าแพ็กเกตยูดีพีหรือแพ็กเกตใดๆที่เข้ามาไม่ตรงตามเงื่อนไขแพ็กเกตนั้นจะถูกทิ้ง (drop) จากตัวควบคุมพอกซ์โดยอัตโนมัติ

3.3.3 แพ็กเกตไอจีเอ็มพีประเภทขอออกจากกลุ่ม

เมื่อตัวควบคุมพอกซ์ตรวจสอบว่าเป็นแพ็กเกตไอจีเอ็มพีหมายเลข 23 ซึ่งเป็นแพ็กเกตไอจีเอ็มพีประเภทขอออกจากกลุ่มที่ถูกส่งมาจากโฮสต์ที่เป็นลูกข่ายไอพีทีวีเพื่อขอยกเลิกการเข้าร่วมกลุ่มมัลติแคสต์กับตัวบริการไอพีทีวี ตัวควบคุมพอกซ์จะดำเนินการลบโฟลว์เอนทรีของลูกข่ายไอพีทีวีตัวนั้นในโอเพนวีสวิตช์ เพื่อเป็นการยกเลิกเส้นทางของการรับส่งแพ็กเกตระหว่างตัวบริการไอพีทีวีและลูกข่ายไอพีทีวีดังกล่าว

เมื่อตัวควบคุมพอกซ์พิจารณาประเภทของแพ็กเกตที่เข้ามาเสร็จสิ้น ตัวควบคุมพอกซ์จะกระทำการเพิ่มโฟลว์เอนทรีให้กับโอเพนวีสวิตช์แต่ละโนด จากนั้นตัวควบคุมพอกซ์จะรอดำเนินการกับแพ็กเกตใหม่ที่เข้ามา

3.4 ขั้นตอนวิธีของการจัดเส้นทาง (Routing algorithm)

ในหัวข้อนี้อธิบายหลักการทำงานของโปรแกรมการคำนวณ ซึ่งเป็นการคำนวณเส้นทางของทั้งสามขั้นตอนวิธีที่ถูกโปรแกรมไว้ในตัวควบคุมพอกซ์ โดยต่อเนื่องมาจากโปรแกรมการจัดการ (POX_manage.py) เมื่อโปรแกรมการจัดการเรียกใช้งานฟังก์ชัน Call_routing ซึ่งจะมีการเรียกใช้งานฟังก์ชัน update_tree ของโปรแกรมคำนวณเส้นทางแต่ละขั้นตอนวิธี ข้อมูลรับเข้าของฟังก์ชัน update_tree คือสถานีเชื่อมโยงต้นทาง การทำงานของฟังก์ชัน update_tree ในโปรแกรมต้นไม้แบบทอดข้ามจะเป็นการเรียนใช้งานฟังก์ชัน calc_spanning_tree เพื่อเข้าใช้งานฟังก์ชัน flip ส่วนโปรแกรมไดร์คสตราจะเป็นการเรียกใช้งานฟังก์ชัน calc_topology เพื่อเข้าใช้งานฟังก์ชัน flip แต่ในโปรแกรมพริมจะเป็นการเรียกใช้งานฟังก์ชัน prim เพื่อเข้าใช้งานฟังก์ชัน flip ซึ่งฟังก์ชัน flip มีหน้าที่เรียกใช้งานฟังก์ชัน Discovery.Link ให้ส่งแพ็กเกตแอลแอลดีพี (Link Layer Discovery Protocol : LLDP) เพื่อเก็บข้อมูลของการเชื่อมต่อระหว่างโอเพนวีสวิตช์ในโครงข่าย ซึ่งจะได้ข้อมูลขาออก (output data) คือ adj เซตของคู่การเชื่อมต่อระหว่างโอเพนวีสวิตช์ และ switches เซตของ

โอเพนวีสวิตช์ในโครงข่ายทั้งหมด จากนั้นฟังก์ชันจะคำนวณเส้นทางของทั้งสามขั้นตอนวิธี ซึ่งอธิบายการทำงานของแต่ละขั้นตอนวิธีได้ดังนี้

ตารางที่ 3.1: ขั้นตอนวิธีของต้นไม้แบบทอดข้าม

Algorithm 1 Spanning tree algorithm

Input: adj (set of Open vSwitches and port numbers), switches (set of Open vSwitches)

Output: tree (Spanning tree path)

```
1:  let q be an empty dictionary
2:  let more be a set of switches
3:  let done be an empty set
4:  let tree be an empty dictionary
5:  while True :
6:      q = sorted(list(more)) + q
7:      remove all items from more
8:          if len(q) == 0 : break
9:          remove the item of position 0 in q and return it to v
10:         if v is in done : continue
11:         add v to done
12:         for w,p is the value of position v in adj :
13:             if w is in tree : continue
14:             add w to more
15:             add (w,p) be the value of v in tree
16:             add (v,adj[w][v]) be the value of w in tree
17: return tree
```

3.4.1 ขั้นตอนวิธีของต้นไม้แบบทอดข้าม

โปรแกรมต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์มีชื่อว่า spanning_tree.py เมื่อนำมาใช้ในงานวิจัยนี้จึงมีการเลือกเฉพาะส่วนของฟังก์ชันที่ถูกใช้งานนั่นคือฟังก์ชัน update_tree เปลี่ยนชื่อฟังก์ชันเป็น update_spt_tree และตั้งชื่อโปรแกรมใหม่ว่า SPT.py การคำนวณเส้นทางถูกเขียนอยู่ภายในฟังก์ชัน calc_spanning_tree โดยเริ่มจากการเรียกใช้งานฟังก์ชัน flip เพื่อให้ได้ข้อมูลขาออก adj และ swtiches แล้วนำข้อมูลทั้งสองใช้เป็นข้อมูลขาเข้าของการคำนวณต้นไม้แบบทอดข้าม ซึ่งสามารถเขียนขั้นตอนวิธีของการคำนวณเส้นทางของต้นไม้แบบทอดข้ามได้ดังตารางที่ 3.1 การคำนวณเริ่มด้วยการเรียงลำดับหมายเลขของโอเพนวีสวิตช์จากหมายเลขน้อยไปหมายเลขมาก ต่อมาจะเลือกคู่ของโอเพนวีสวิตช์ที่เชื่อมต่อกันจากโอเพนวีสวิตช์หมายเลขน้อยเรียงไปจนถึงโอเพนวีสวิตช์

ตัวสุดท้าย ซึ่งก่อนทำการเลือกคู่จะมีการพิจารณาถ้าโอเพนวีสวิตช์คู่ไหนถูกเลือกไปก่อนแล้วจะข้ามโอเพนวีสวิตช์โนดนั้นไปและทำการเลือกคู่ของโอเพนวีสวิตช์ตัวถัดไป เมื่อการคำนวณเส้นทางเสร็จสิ้นจะได้เป็นเซตของช่องทางของโอเพนวีสวิตช์ที่เชื่อมต่อกันออกมาในตัวแปร tree และคืนค่าตัวแปร tree ให้กับฟังก์ชัน Call_routing ของโปรแกรม POX.manage.py จากหลักการทำงานของโปรแกรม SPT.py จะเห็นได้ว่าการคำนวณเส้นทางของโปรแกรมต้นไม้แบบทอดข้ามเป็นเพียงการเลือกเส้นทางตามลำดับของหมายเลขโอเพนวีสวิตช์ ซึ่งไม่ได้มีการพิจารณาน้ำหนักเส้นเชื่อมระหว่างโอเพนวีสวิตช์

ตารางที่ 3.2: ขั้นตอนวิธีของโปรแกรมฟังก์ชัน Dijkstra

Algorithm 1 Dijkstra function

Input: R_G (set of Open vSwitches and port numbers)
S (initial Open vSwitch), SW (set of Open vSwitches)

Output: TREE (Dijkstra path)

```
let TREE be an empty dictionary
let Q be an empty list
for END is in R_G.keys():
    if S is END : continue
    else :
        call Shortest function and return value to DIJ and ROT
        let PART be an empty list
        while 1 :
            add END to the end of PATH
            if END = S : break
            add value of END in ROT to END
            reverse the elements of PATH
            add the number of elements in PATH to LEN
            for L in range (LEN-1) :
                add the value of position L in PATH to SW1
                add the value of position L+1 in PATH to SW2
                if SW2 is None : continue
                else :
                    add the value of G[SW1][SW2] to PORT
                    add set of SW1,SW2,PORT to TREE
return TREE
```

ตารางที่ 3.3: ขั้นตอนวิธีของโปรแกรมฟังก์ชัน Shortest

Algorithm 2 Shortest function

Input: S (initial Open vSwitch), E (destination Open vSwitch)

Output: DIJ,ROT

let G be the graph of proposed topology that composes of V, pair of nodes(Open vSwitch), and E, edge weight between 2 nodes
let N_W be the initial edge weight of node S, equal to [0]
let DIJ and ROT be an empty dictionary
for node v is in N_W **:**
    let value of DIJ equal to N_W
    if node v is node E **:**
        break:
for node w is the value of node v in G **:**
    find weight of (v_i,w_i) = DIJ[v] + weight of (v,w) in G
    if node w_i is already in DIJ **:**
        if weight of (v_i,w_i) < the value of node w in DIJ **:**
            show Value Error
    elif node w_i is not already in N_W **or** weight of (v_i,w_i) < value of node w in N_W **:**
            add set of node w_i and weight of (v_i,w_i) to N_W
            add set of node v_i and node w_i to ROT
return (DIJ,ROT)

3.4.2 ขั้นตอนวิธีของไดร์คสตรา

ขั้นตอนวิธีของโปรแกรมไดร์คสตาถูกแบ่งออกเป็น 2 ฟังก์ชัน คือ ฟังก์ชัน Dijkstra และ ฟังก์ชัน Shortest ซึ่งอธิบายการทำงานได้จากขั้นตอนวิธีดังตารางที่ 3.2 และ ตารางที่ 3.3 ตามลำดับ โดยฟังก์ชัน Dijkstra เมื่อถูกเรียกใช้งานจะมีข้อมูลรับเข้าคือ R_G เซตของหมายเลขช่องทาง (port number) ที่เชื่อมต่อระหว่างโอเพนวีสวิตช์ นอกจาก R_G ยังมี S โอเพนวีสวิตช์ที่เป็นสถานีเชื่อมโยงต้นทาง และ SW เซตของโอเพนวีสวิตช์ทั้งหมดในโครงข่าย ฟังก์ชัน Dijkstra มีหน้าที่กำหนดคู่ของสถานีเชื่อมโยงต้นทางและสถานีเชื่อมโยงปลายทางเพื่อส่งให้กับฟังก์ชัน Shortest ทำการคำนวณเส้นทางที่สั้นที่สุดระหว่างสถานีเชื่อมโยงต้นทางและสถานีเชื่อมโยงปลายทางที่ได้รับมาทีละคู่ โดยข้อมูลรับเข้าของฟังก์ชัน Shortest คือ S สถานีเชื่อมโยงต้นทางและ E สถานีเชื่อมโยงปลายทาง รวมถึงข้อมูลที่กำหนดให้อยู่แล้วคือ G=(V,E) เซตของน้ำหนักเส้นเชื่อมระหว่างโอเพนวีสวิตช์ หลักการทำงานของฟังก์ชัน Shortest คือการเปรียบ เทียบน้ำหนักเส้นเชื่อมของแต่ละคู่สถานีเชื่อมโยง เพื่อให้

ได้น้ำหนักเส้นเชื่อมที่น้อยที่สุดของแต่ละคู่สถานีเชื่อมโยง จากนั้นฟังก์ชัน Dijkstra จะรวบรวมคู่ของสถานีเชื่อมโยงที่ได้จากฟังก์ชัน Shortest นำมาสร้างเป็นเส้นทางของการส่งจากสถานีเชื่อมโยงต้นทางไปยังปลายทางทั้งหมด

ตารางที่ 3.4: ขั้นตอนวิธีของโปรแกรมพริม

Algorithm 3 Prim's algorithm

Input: S (initial Open vSwitch)

Output: MST

```
let G be the graph of proposed topology that composes of V,
pair of nodes(Open vSwitch), and E, edge weight between 2 nodes
let VER be the initial Open vSwitch
let ADJ be an empty dictionary
for node n1, node n2, edge weight c in G :
    add c, n1, n2 be the value of n1 to ADJ
    add c, n2, n1 be the value of n2 to ADJ
let MST be an empty list
add the value of VER[0] to USED
add the value of ADJ[S] to EDG_U
convert EDG_U to heap function
while EDG_U :
    return the smallest value of EDG_U to COT, N1, N2 and remove it
    if N2 is not already in USED :
        add N2 to USED
        add COT, N1, N2 to the end of MST
        for E in ADJ[2] :
            if value of E[2] is not already in USED :
                push E onto EDG_U
return MST
```

3.4.3 ขั้นตอนวิธีของพริม

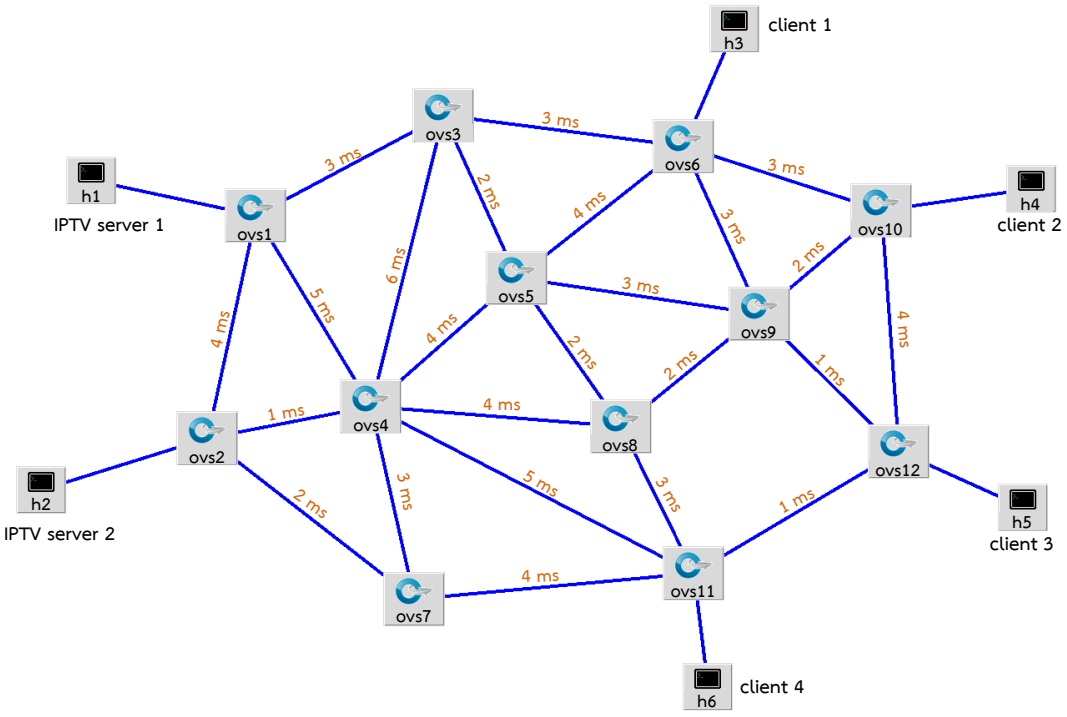
ขั้นตอนวิธีของโปรแกรมพริมใช้ชื่อโปรแกรมว่า Prim.py เมื่อโปรแกรมการจัดการเรียกใช้งานฟังก์ชัน update_tree ของโปรแกรม Prim.py ฟังก์ชัน update_tree จะเรียกใช้งานฟังก์ชัน Prim เพื่อใช้งานฟังก์ชัน flip และเมื่อฟังก์ชัน flip ได้ข้อมูลขาออกทั้ง adj และ swtiches แล้วการคำนวณเส้นทางจึงจะเริ่มต้นขึ้น สามารถเขียนขั้นตอนวิธีการทำงานของโปรแกรม Prim.py ได้ดังตารางที่ 3.4

ซึ่งในการคำนวณได้มีการกำหนดข้อมูลขาเข้าให้อยู่แล้วคือ G=(V,E) เป็นเซตของน้ำหนักเส้นเชื่อมระหว่างโอเพนวีสวิตช์ และ VER เซตของโอเพนวีสวิตช์ทั้งหมดในโครงข่ายจำลอง หลักการทำงานของโปรแกรมพริมคือการใช้ฟังก์ชัน heap เพื่อเลือกแต่ละคู่ของโอเพนวีสวิตช์ที่ทำให้ค่าน้ำหนักเส้นเชื่อมรวมทั้งโครงข่ายมีค่าน้อยที่สุด และนำโอเพนวีสวิตช์แต่ละคู่มารวมเป็นเส้นทางของการส่งจากสถานีเชื่อมโยงต้นทางไปยังสถานีเชื่อมโยงปลายทางทั้งหมดซึ่งถูกเก็บในตัวแปร MST จากนั้นฟังก์ชัน Prim จะคืนค่า MST และ adj ให้กับฟังก์ชัน update_tree ซึ่งฟังก์ชัน update_tree จะทำการเพิ่มหมายเลขช่องทางระหว่างโอเพนวีสวิตช์ที่ได้มาจาก adj ให้กับ MST โดยเก็บในตัวแปร tree และคืนค่าตัวแปร tree ให้กับฟังก์ชัน Call_routing ของโปรแกรม POX.manage.py

บทที่ 4
การทดลองการให้บริการไอพีทีวีบนโครงข่ายโอเพนโฟลว์

การคำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์ไม่ได้พิจารณาน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์ ในงานวิจัยนี้จึงนำการคำนวณเส้นทางที่มีการพิจารณาน้ำหนักของเส้นเชื่อมมาปรับใช้กับโปรแกรมต้นไม้แบบทอดข้ามที่มีในตัวควบคุมพอกซ์โดยเลือกใช้การคำนวณเส้นทางด้วยขั้นตอนวิธีไดร์คสตราและพริมเพื่อคำนวณเส้นทางที่สั้นที่สุดและเพื่อเปรียบเทียบประสิทธิภาพของการคำนวณเส้นทางระหว่างสามขั้นตอนวิธี งานวิจัยนี้จึงเสนอการคำนวณเส้นทางของสามขั้นตอนวิธีเพื่อการให้บริการไอพีทีวีบนโครงข่ายโอเพนโฟลว์

งานวิจัยนี้ได้ทำการทดลองด้วยโปรแกรมมินิเน็ท (Mininet) [17] ซึ่งเป็นโปรแกรมที่ใช้เลียนแบบ (emulate) การทำงานของโครงข่ายการสื่อสาร โดยงานวิจัยนี้ได้จำลองการรับส่งแพ็กเกตยูดีพีและการสตรีมวีดีทัศน์ผ่านโอเพนวีสวิตช์ โดยโอเพนวีสวิตช์สื่อสารกันด้วยโพรโทคอลโอเพนโฟลว์ ที่ถูกควบคุมด้วยตัวควบคุมพอกซ์ รุ่นของซอฟต์แวร์ที่ใช้ในการทดลองแสดงได้ดังตารางที่ 4.1 รวมถึงทอพอโลยี (topology) ของโครงข่ายโอเพนโฟลว์จำลองการให้บริการไอพีทีวีแสดงการเชื่อมต่อระหว่างโอเพนวีสวิตช์ดังรูปที่ 4.1



รูปที่ 4.1: ทอพอโลยีของโครงข่ายโอเพนโฟลว์จำลองประกอบด้วยโอเพนวีสวิตช์ 12 โนด

ตารางที่ 4.1: รุ่นของซอฟต์แวร์ที่ใช้ในงานวิจัย

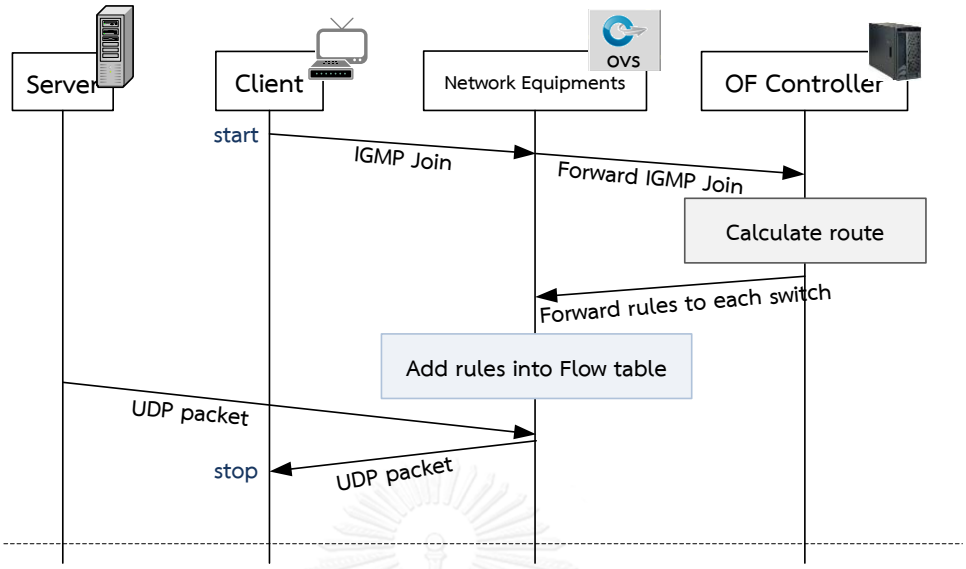
Software	Version
Mininet	2.2
OpenFlow	1.0
Open vSwitch	1.4.6
POX	3.0

4.1 การทดลองเปรียบเทียบเวลาที่ใช้ในการส่งแพ็กเกตยูดีพี

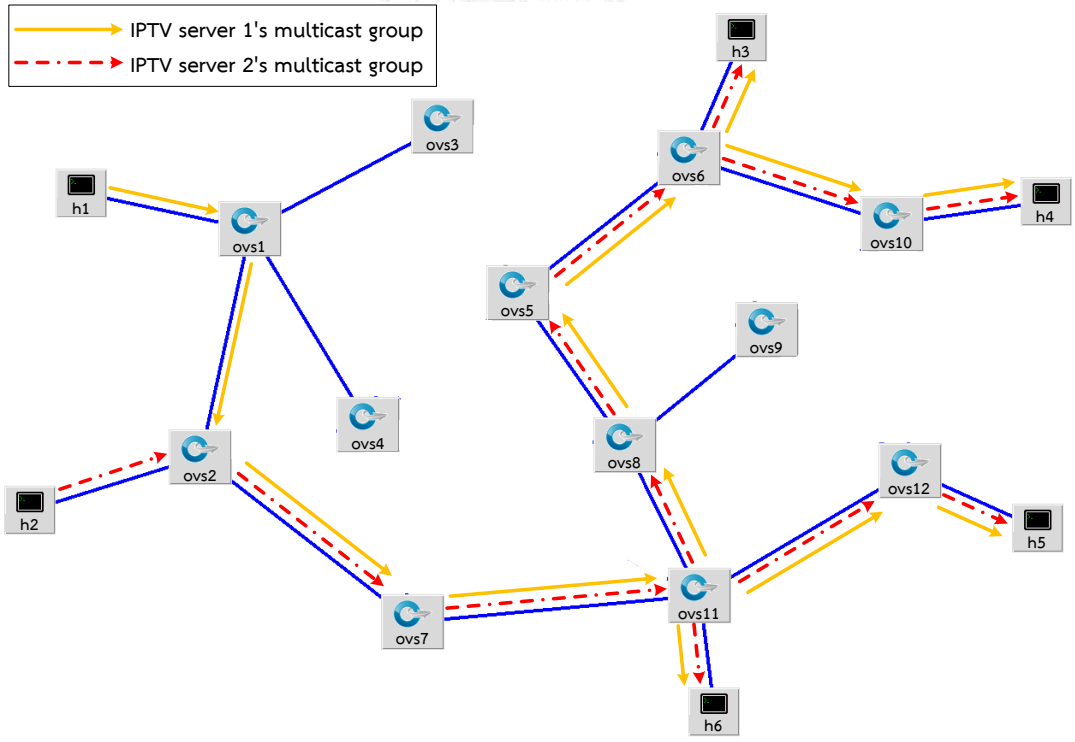
ประสิทธิภาพที่แสดงให้เห็นถึงความแตกต่างของการคำนวณเส้นทางระหว่างสามขั้นตอนวิธีที่ได้นำเสนอนั่นคือการเปรียบเทียบเวลาที่ใช้ในการส่งแพ็กเกตจากตัวบริการไอพีทีวีไปยังลูกข่ายไอพีทีวีบนเส้นทางที่ได้มาจากการคำนวณของทั้งสามขั้นตอนวิธี การทดลองนี้จึงเป็นการทดลองเพื่อเปรียบเทียบเวลาที่ลูกข่ายไอพีทีวีขอเข้าร่วมกลุ่มมัลติแคสต์จนกระทั่งลูกข่ายไอพีทีวีได้รับแพ็กเกตยูดีพีแรกจากตัวบริการไอพีทีวี การทดลองใช้ทอพอโลยีรูปที่ 4.1 ที่แสดงค่าน้ำหนักเส้นเชื่อมระหว่างโอเพนวีสวิตช์ซึ่งได้จากการสุ่ม (random) ด้วยคำสั่ง randi ( ) เป็นการสุ่มเลขจำนวนเต็มแบบเอกรูป (Uniformly distributed) [18] เพื่อให้เห็นถึงความแตกต่างของการคำนวณเส้นทางทั้งสามขั้นตอนวิธี ในการทดลองตัวบริการไอพีทีวี 1 และ 2 ถูกแทนด้วยโฮสต์ 1 และ 2 (h1, h2) ตามลำดับ ลูกข่ายไอพีทีวี 1,2,3 และ 4 ถูกแทนด้วยโฮสต์ 3,4,5 และ 6 (h3-h6) ตามลำดับ

ขั้นตอนการทดลองเวลาของการรับส่งแพ็กเกตยูดีพีแสดงได้ดังรูปที่ 4.2 เริ่มจากลูกข่ายไอพีทีวีส่งแพ็กเกตไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่มโดยส่งผ่านโอเพนวีสวิตช์โนดที่อยู่ติดกับลูกข่ายไอพีทีวีตัวนั้น เพื่อให้ส่งต่อแพ็กเกตไปยังตัวควบคุมพอกซ์ เมื่อตัวควบคุมพอกซ์ได้รับแพ็กเกตไอจีเอ็มพีประเภทขอเข้าร่วมกลุ่ม ตัวควบคุมพอกซ์จะทำการคำนวณเส้นทางให้กับแพ็กเกตยูดีพีที่จะถูกส่งมาจากตัวบริการไอพีทีวีนั้นด้วยโปรแกรมของแต่ละขั้นตอนวิธี จากนั้นตัวควบคุมพอกซ์ทำการเพิ่มโฟลว์เอนทรีให้กับโอเพนวีสวิตช์ที่เกี่ยวข้องทุกโนด เมื่อตัวบริการไอพีทีวีส่งแพ็กเกตยูดีพีแรก โอเพนวีสวิตช์ที่ได้รับแพ็กเกตยูดีพีและมีโฟลว์เอนทรีของแพ็กเกตยูดีพีอยู่แล้วจะทำการส่งต่อแพ็กเกตยูดีพีไปยังโอเพนวีสวิตช์ถัดไปเพื่อให้แพ็กเกตยูดีพีไปถึงลูกข่ายไอพีทีวีและเวลาจะสิ้นสุดลงเมื่อลูกข่ายไอพีทีวีได้รับแพ็กเกตยูดีพีแรก

เริ่มต้นการทดลองด้วยการสร้างทอพอโลยีรูปที่ 4.1 ในโปรแกรมมินิเน็ทด้วยการใช้โปรแกรมภาคผนวก ก และในตัวควบคุมพอกซ์ใช้โปรแกรมภาคผนวก ข, ค.1 และ ค.2 เพื่อคำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้าม แบบไดร์คสตรา และแบบพริมตามลำดับ แพ็กเกตไอจีเอ็มพีและยูดีพีถูกส่งด้วยโปรแกรมเนเมซิส เส้นทางที่ได้จากการคำนวณของแต่ละขั้นตอนวิธีอธิบายได้ดังนี้



รูปที่ 4.2: ขั้นตอนการทดลองเวลาที่ลูกข่ายไอพีทีวีส่งแพ็กเกตไอจีเอ็มพีและได้รับแพ็กเกตยูดีพีแรก



รูปที่ 4.3: เส้นทางของการคำนวณเส้นทางด้วยโปรแกรม SPT.py

4.1.1 เส้นทางการส่งแพ็กเกตยูดีพีด้วยขั้นตอนวิธีต้นไม้แบบทอดข้าม

เส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามในตัวควบคุมพอกซ์แสดงได้ดังรูปที่ 4.3 การคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์เป็นการพิจารณาจาก

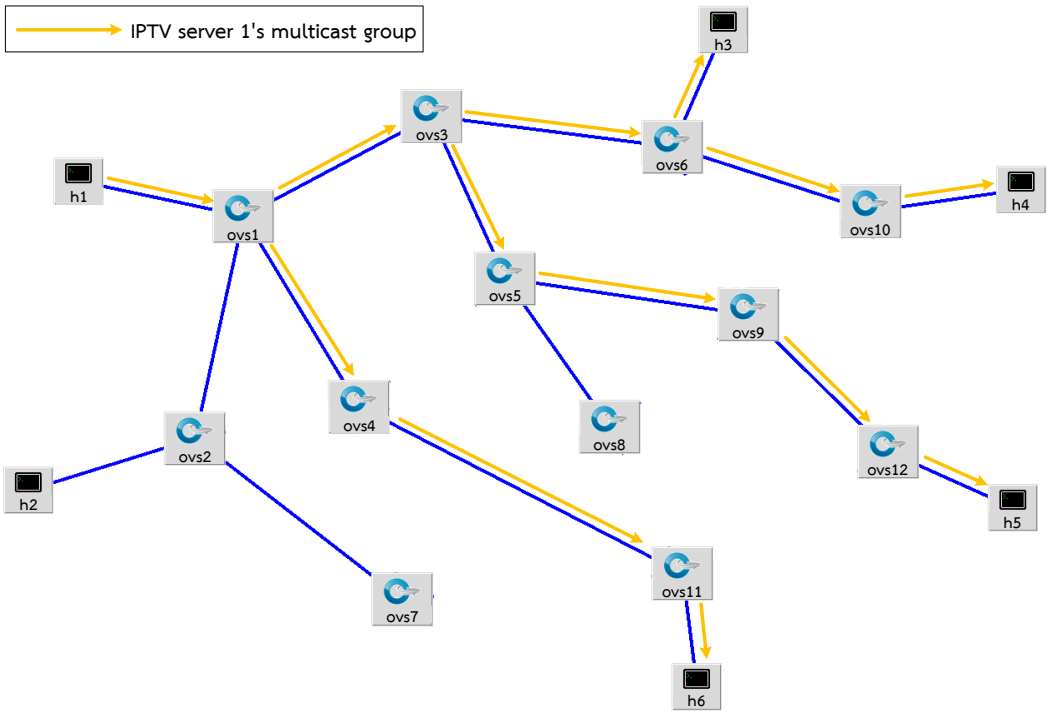
ลำดับหมายเลขของโอเพนวีสวิตช์ โดยการเรียงลำดับโอเพนวีสวิตช์หมายเลขน้อยไปยังโอเพนวีสวิตช์หมายเลขมากสุด แล้วจึงเริ่มการพิจารณาจากโอเพนวีสวิตช์หมายเลขน้อยสุด ดังนั้นจากทอพอโลยีรูปที่ 4.1 โอเพนวีสวิตช์หมายเลข 1 จะเป็นโอเพนวีสวิตช์เริ่มต้นเสมอ ทำให้การคำนวณเส้นทางของตัวบริการไอพีทีวี 1 และ 2 นั้นเหมือนกัน จากโอเพนวีสวิตช์เริ่มต้นโปรแกรมจะเลือกเส้นเชื่อมไปยังโอเพนวีสวิตช์ที่เชื่อมกับโอเพนวีสวิตช์เริ่มต้นทุกโนด แล้วโปรแกรมจึงพิจารณาโอเพนวีสวิตช์ถัดไปตามลำดับหมายเลขโอเพนวีสวิตช์ โดยเลือกโอเพนวีสวิตช์หมายเลขน้อยที่สุดเป็นโอเพนวีสวิตช์ถัดไป ดังนั้นโอเพนวีสวิตช์ถัดไปจึงเป็นโอเพนวีสวิตช์หมายเลข 2 และโปรแกรมทำการเลือกเส้นเชื่อมที่เชื่อมกับโอเพนวีสวิตช์หมายเลข 2 โดยจะไม่เลือกโอเพนวีสวิตช์ที่ถูกพิจารณาไปแล้ว และทำการเลือกเส้นเชื่อมของโอเพนวีสวิตช์ถัดไปจนครบทุกโนด จึงได้เส้นทางของการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามดังรูปที่ 4.3

4.1.2 เส้นทางการส่งแพ็กเกตยูดีพีด้วยขั้นตอนวิธีไดร์คสตรา

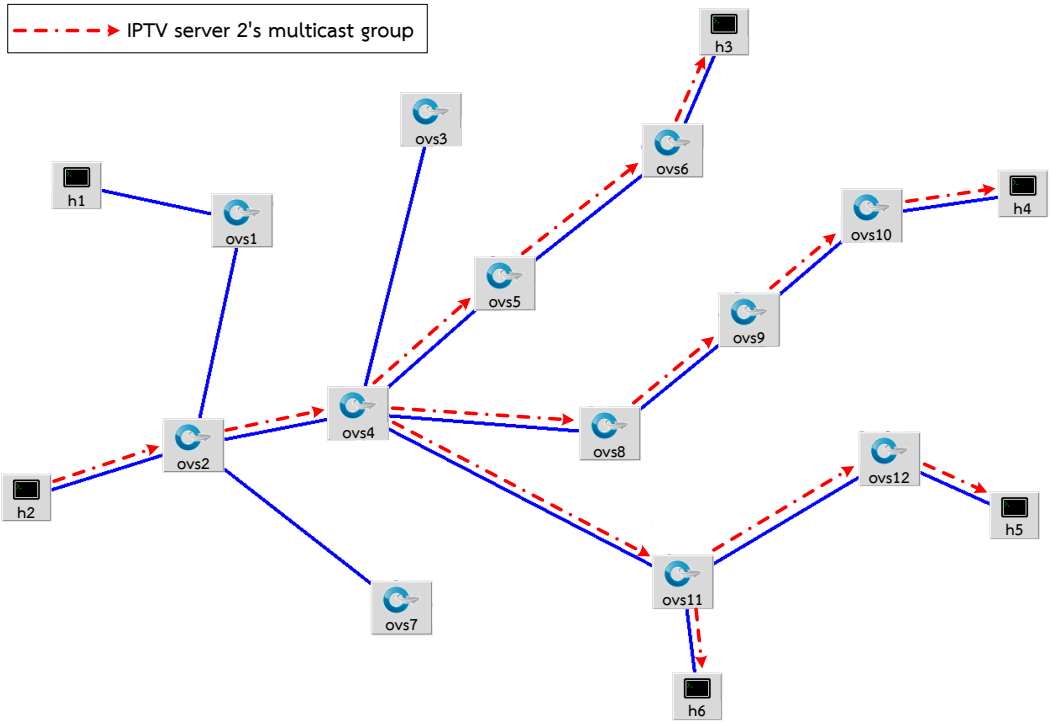
เส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีไดร์คสตราในตัวควบคุมพอกซ์แสดงได้ดังรูปที่ 4.4 ซึ่งรูปที่ 4.4 (ก) และ (ข) เป็นเส้นทางของการส่งแพ็กเกตยูดีพีจากตัวบริการไอพีทีวี 1 และตัวบริการไอพีทีวี 2 ไปยังลูกข่ายไอพีทีวีทุกลูกข่ายในทอพอโลยีตามลำดับ โปรแกรมการคำนวณเส้นทางด้วยขั้นตอนวิธีไดร์คสตราในตัวควบคุมพอกซ์เป็นการคำนวณเส้นทางทีละคู่ของโอเพนวีสวิตช์ต้นทางและโอเพนวีสวิตช์ปลายทางและพิจารณาเส้นทางจากน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์ ซึ่งโอเพนวีสวิตช์ต้นทางหมายถึงโอเพนวีสวิตช์ที่อยู่ติดกับตัวบริการไอพีทีวี และโอเพนวีสวิตช์ปลายทางหมายถึงโอเพนวีสวิตช์ที่อยู่ติดกับลูกข่ายไอพีทีวี ดังนั้นการคำนวณเส้นทางจึงเริ่มจากโอเพนวีสวิตช์ที่อยู่ติดกับตัวบริการไอพีทีวีและพิจาณาเลือกโอเพนวีสวิตช์ถัดไปตามลำดับน้ำหนักของเส้นเชื่อมซึ่งจะพิจาณาเลือกเส้นทางที่ให้ค่าน้ำหนักของเส้นเชื่อมน้อยที่สุด การพิจารณาทีละคู่ของโอเพนวีสวิตช์ต้นทางและโอเพนวีสวิตช์ปลายทางทำให้เส้นทางที่ได้เป็นเส้นทางที่แตกกิ่งก้านตามจำนวนของลูกข่ายไอพีทีวีดังรูปที่ 4.4

4.1.3 เส้นทางการส่งแพ็กเกตยูดีพีด้วยขั้นตอนวิธีพริม

เส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีพริมในตัวควบคุมพอกซ์แสดงได้ดังรูปที่ 4.5 ซึ่งเส้นทางที่ได้มีลักษณะเป็นต้นไม้แบบทอดข้ามเหมือนการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้าม แต่เส้นทางแตกต่างกันเนื่องจากขั้นตอนวิธีพริมมีการพิจารณาน้ำหนักเส้นเชื่อม เพื่อให้การส่งแพ็กเกตได้น้ำหนักเส้นเชื่อมรวมระหว่างตัวบริการไอพีทีวีและลูกข่ายไอพีทีวีภายในโครงข่ายทั้งหมดน้อยที่สุด โดยการคำนวณเริ่มต้นจากโอเพนวีสวิตช์ที่อยู่ติดกับตัวบริการไอพีทีวีแล้วจึงคำนวณเส้นทางตามค่าน้ำหนักของเส้นเชื่อมไปยังลูกข่ายไอพีทีวีทุกลูกข่าย

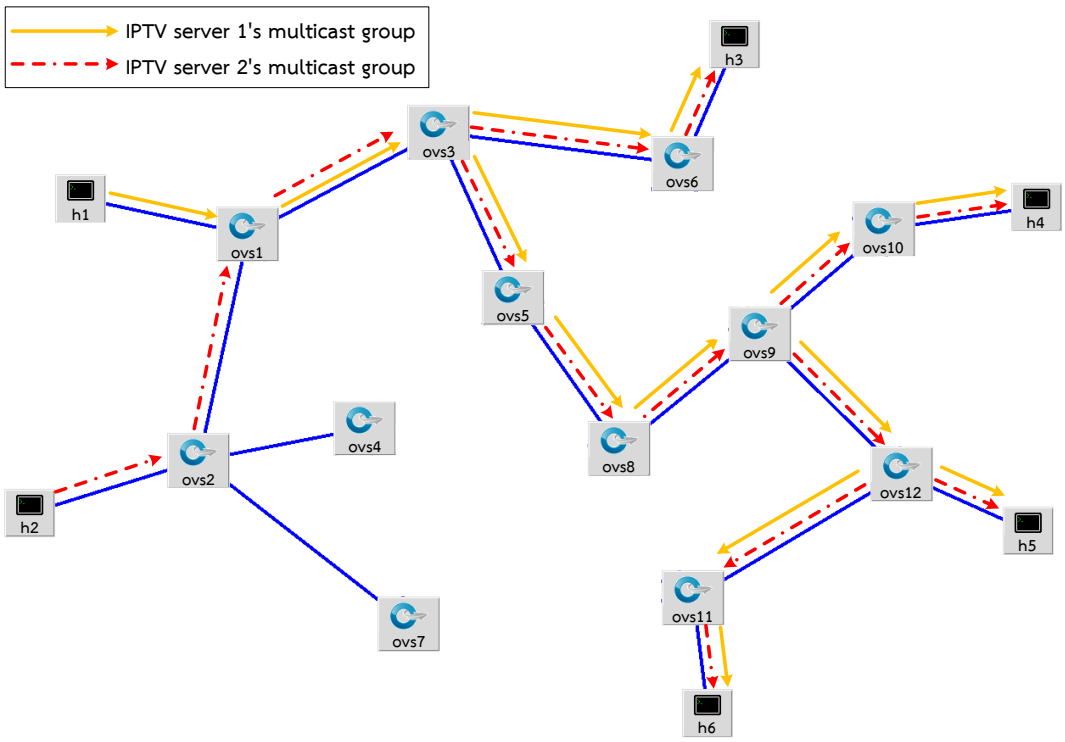


(ก) เส้นทางของการคำนวณเส้นทางของตัวบริการไอพีทีวี 1



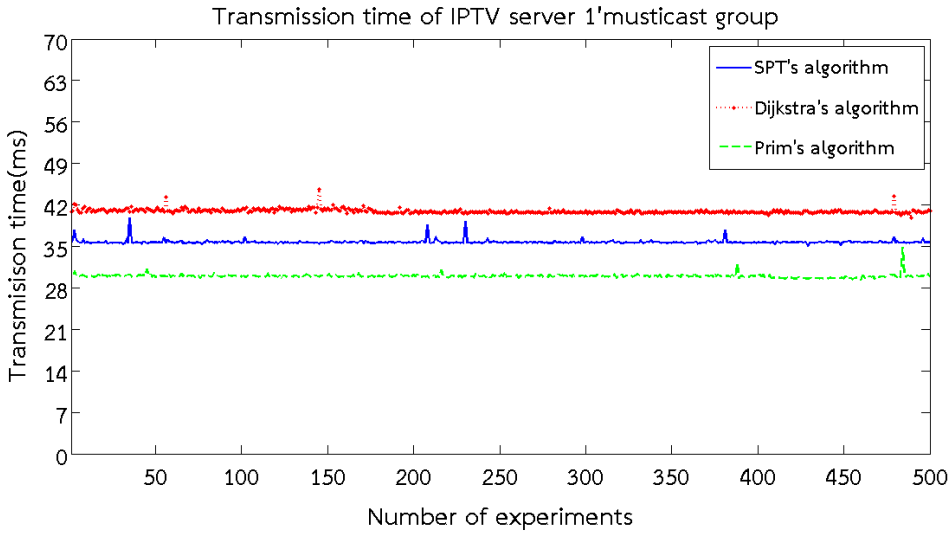
(ข) เส้นทางของการคำนวณเส้นทางของตัวบริการไอพีทีวี 2

รูปที่ 4.4: เส้นทางของการคำนวณเส้นทางด้วยโปรแกรม Dijkstra.py

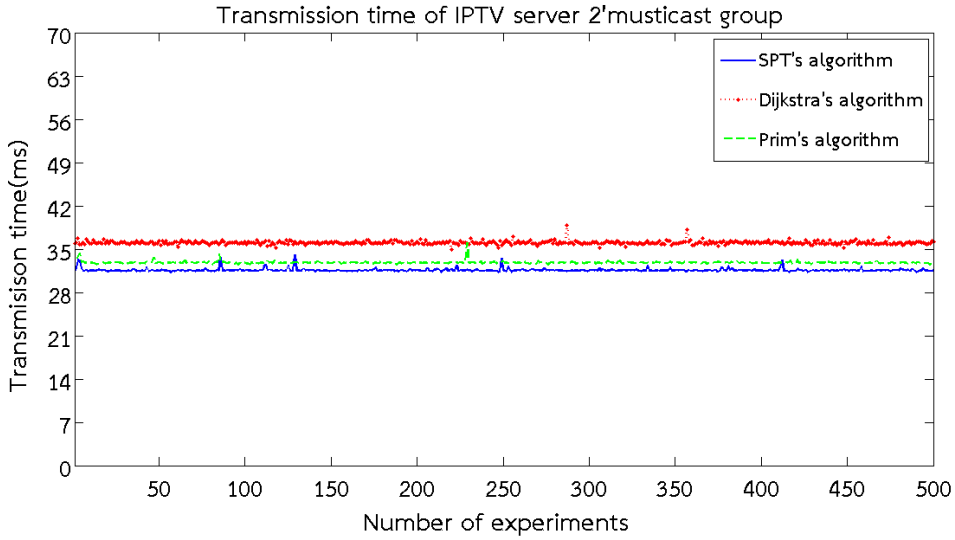


รูปที่ 4.5: เส้นทางของการคำนวณเส้นทางด้วยโปรแกรม Prim.py

ผลการทดลองเวลาของการส่งคำร้องขอเข้าร่วมกลุ่มของลูกข่ายไอพีทีวี 1, 2, 3 และ 4 เพื่อขอเข้าร่วมกลุ่มมัลติแคสต์ของตัวบริการไอพีทีวี 1 จนกระทั่งได้รับแพ็กเกตยูดีพีแรกพร้อมกันแสดงได้ดังรูปที่ 4.6 โดยแกนแนวนอนเป็นจำนวนการทดลองรับส่งแพ็กเกต 500 ครั้ง แกนแนวตั้งเป็นเวลาที่ลูกข่ายไอพีทีวีขอเข้าร่วมกลุ่มจนกระทั่งได้รับแพ็กเกตยูดีพีแรกของทั้งสามขั้นตอนวิธี เวลาของการรับส่งแพ็กเกตด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามมีค่าเฉลี่ย 35.69 มิลลิวินาที และเวลาของการรับส่งแพ็กเกตด้วยขั้นตอนวิธีพริมมีค่าน้อยที่สุดเฉลี่ย 29.96 มิลลิวินาที ผลการทดลองรับส่งแพ็กเกตของตัวบริการไอพีทีวี 2 แสดงดังรูปที่ 4.7 เวลาของการรับส่งแพ็กเกตด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามมีค่าเฉลี่ย 31.56 มิลลิวินาที เป็นค่าเวลาน้อยที่สุดจากสามขั้นตอนวิธี จากผลการทดลองการรับส่งแพ็กเกตยูดีพีของทั้งตัวบริการไอพีทีวี 1 และ 2 แสดงให้เห็นว่า เส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริมที่ส่งแพ็กเกตยูดีพีให้ทุกลูกข่ายไอพีทีวีพร้อมกัน ใช้เวลารวมน้อยกว่าขั้นตอนวิธีของไดร์คสตรา ซึ่งจากรูปที่ 4.4 แสดงเส้นทางของการส่งแพ็กเกตด้วยขั้นตอนวิธีไดร์คสตราแสดงให้เห็นว่าเส้นทางของขั้นตอนวิธีไดร์คสตราเหมาะสมกับการส่งแพ็กเกตแบบยูนิแคสต์จากตัวบริการไอพีทีวีไปยังแต่ละลูกข่ายไอพีทีวีจึงทำให้เวลาของการส่งแพ็กเกตรวมด้วยเส้นทางของไดร์คสตรามีเวลารวมมากที่สุด ซึ่งไม่เหมาะสมกับการให้บริการไอพีทีวีแบบมัลติแคสต์



รูปที่ 4.6: เวลารวมของการส่งแพ็กเกตยูดีพีของกลุ่มมัลติแคสต์ตัวบริการไอพีทีวี 1

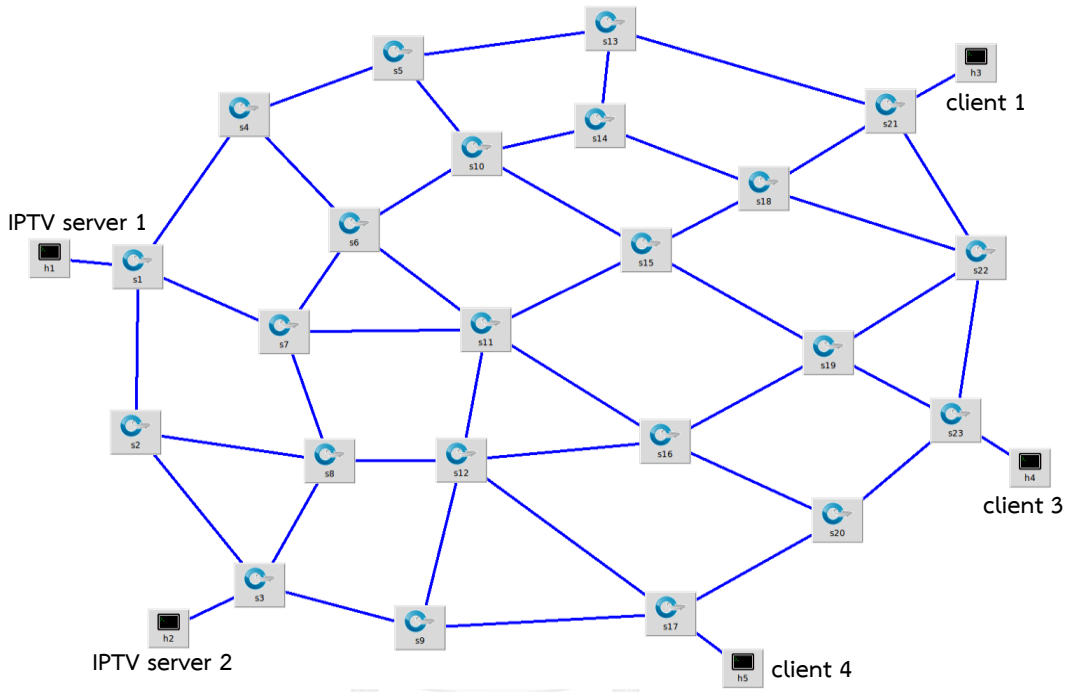


รูปที่ 4.7: เวลารวมของการส่งแพ็กเกตยูดีพีของกลุ่มมัลติแคสต์ตัวบริการไอพีทีวี 2

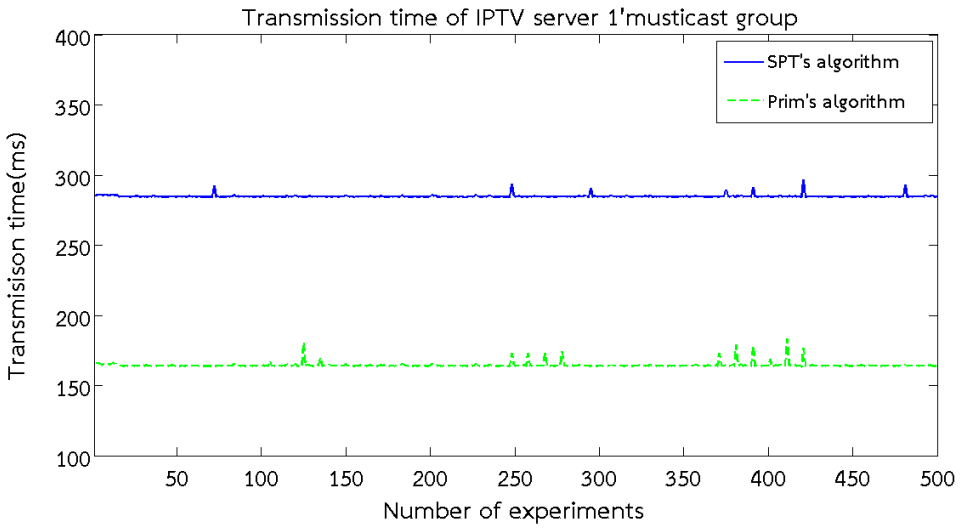
การทดลองต่อมาเพื่อให้เห็นความแตกต่างของเส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริม จึงได้มีการเพิ่มจำนวนโอเพนวีสวิตช์จาก 12 โนด เป็น 23 โนด ซึ่งทำให้จำนวนเส้นเชื่อมระหว่างโอเพนวีสวิตช์เพิ่มขึ้นด้วยแสดงดังทอพอโลยีรูปที่ 4.8

จากผลการทดลองรูปที่ 4.9 แสดงเวลาของการส่งแพ็กเกตรวมของตัวบริการไอพีทีวี 1 ของทอพอโลยีรูปที่ 4.8 แสดงให้เห็นความแตกต่างของเวลาระหว่างเวลาของการส่งแพ็กเกตยูดีพีด้วยเส้นทางต้นไม้แบบทอดข้ามและพริม ซึ่งเวลาของเส้นทางพริมมีค่าน้อยกว่าเส้นทางต้นไม้แบบทอดข้ามอยู่ประมาณ 120 มิลลิวินาที และเวลาของการส่งแพ็กเกตรวมของตัวบริการไอพีทีวี 2 แสดงดังรูปที่ 4.10 ซึ่งเวลาของเส้นทางพริมมีค่าน้อยกว่าเส้นทางต้นไม้แบบทอดข้ามอยู่ประมาณ 42

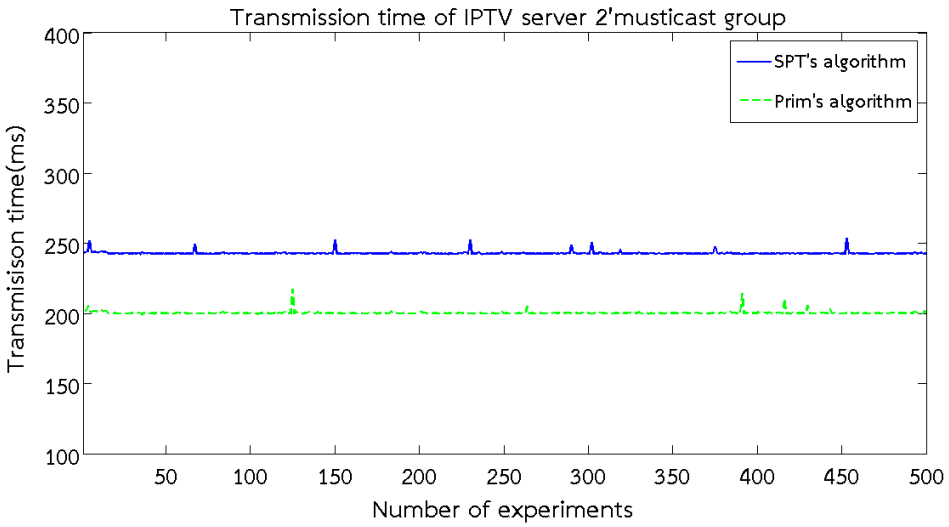
มิลลิวินาที จากผลการทดลองแสดงให้เห็นว่าการที่ขั้นตอนวิธีต้นไม้แบบทอดข้ามมีการคำนวณเส้นทางจากการเรียงลำดับหมายเลขของโอเพนวีสวิตช์นั้นทำให้ได้เวลาที่ใช้ในการส่งแพ็กเกตยูดีพีมากกว่าการคำนวณเส้นทางที่พิจารณาน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์ด้วยขั้นตอนวิธีพริม ดังนั้นเส้นทางที่ได้จากการคำนวณของขั้นตอนวิธีพริมจะทำให้การบริการไอพีทีวีแบบมัลติแคสต์ด้วยโครงข่ายโอเพนโฟลว์จำลองใช้เวลาในการรับส่งแพ็กเกตน้อยที่สุด



รูปที่ 4.8: ทอพอโลยีของโครงข่ายโอเพนโฟลว์จำลองประกอบด้วยโอเพนวีสวิตช์ 23 โนด



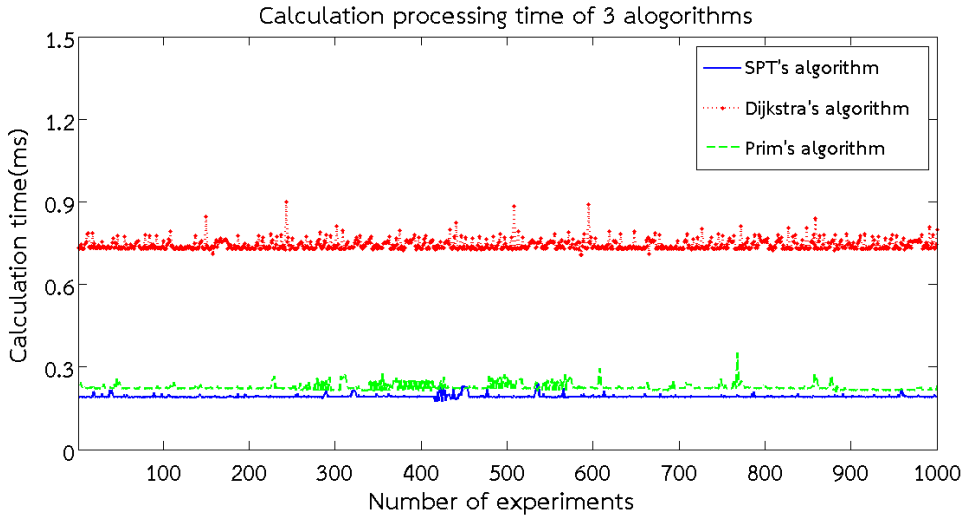
รูปที่ 4.9: เวลารวมของการส่งแพ็กเกตยูดีพีของกลุ่มมัลติแคสต์ตัวบริการไอพีทีวี 1



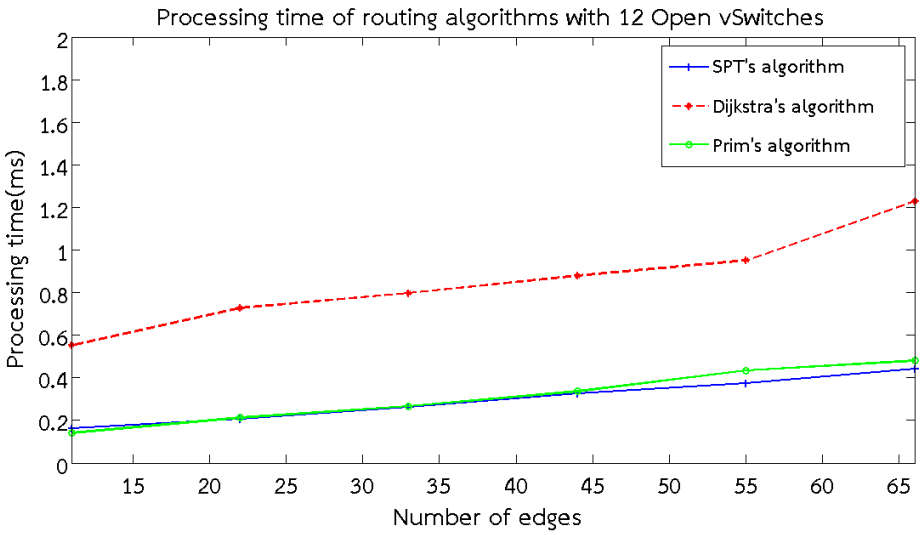
รูปที่ 4.10: เวลารวมของการส่งแพ็กเกตยูดีพีของกลุ่มมัลติแคสต์ตัวบริการไอพีทีวี 2

4.2 การทดลองเปรียบเทียบเวลาของการคำนวณเส้นทาง

การทดลองนี้เพื่อดูความแตกต่างของเวลาที่ตัวควบคุมพอกซ์ใช้ในการคำนวณเส้นทางของแต่ละขั้นตอนวิธี โดยทดลองเปรียบเทียบเวลาของการคำนวณเส้นทางของทอโพลอยีรูปที่ 4.1 ซึ่งเปรียบเทียบเวลาตั้งแต่ตัวควบคุมพอกซ์เริ่มเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์จนกระทั่งตัวควบคุมพอกซ์คำนวณเส้นทางของทั้งสามขั้นตอนวิธีเรียบร้อย



รูปที่ 4.11: เวลาที่ใช้ในการคำนวณเส้นทางของทั้งสามขั้นตอนวิธี



รูปที่ 4.12: การทดลองเปรียบเทียบเวลาที่ใช้ในการคำนวณเส้นทาง
เมื่อกำหนดให้จำนวนโอเพนวีสวิตช์คงทีที่ 12 โนด

ผลการทดลองรูปที่ 4.11 แสดงการเปรียบเทียบเวลาที่ใช้ในการคำนวณเส้นทางของทั้งสามขั้นตอนวิธี โดยทำการทดลองจำนวน 1000 รอบแสดงในแกนแนวนอน และแกนแนวตั้งแสดงเวลาที่ใช้ในการคำนวณเส้นทาง ผลการทดลองแสดงให้เห็นว่าการคำนวณเส้นทางของขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริมมีเวลาที่แตกต่างกันเล็กน้อย ขั้นตอนวิธีต้นไม้แบบทอดข้ามมีเวลาการคำนวณเฉลี่ย 0.19 มิลลิวินาที และพริมมีเวลาคำนวณเฉลี่ย 0.22 มิลลิวินาที ในส่วนของขั้นตอนวิธีไดร์คสตรามีเวลาการคำนวณที่มากที่สุด มีค่าเฉลี่ย 0.74 มิลลิวินาที การทดลองต่อมาเป็นการทดลองเปรียบเทียบความเปลี่ยนแปลงของเวลาที่ใช้ในการคำนวณเส้นทางเมื่อกำหนดให้โอเพนวีสวิตช์เท่ากันที่ 12 โนด และทำการเพิ่มจำนวนเส้นเชื่อมระหว่างโอเพนวีสวิตช์ ผลการทดลองแสดงได้ดังรูปที่ 4.12

$$\text{Dijkstra's Time complexity} = O(|E| + |V| \log|V|) \quad (4.1)$$

$$\text{Prim's Time complexity} = O(|E| + \log|V|) \quad (4.2)$$

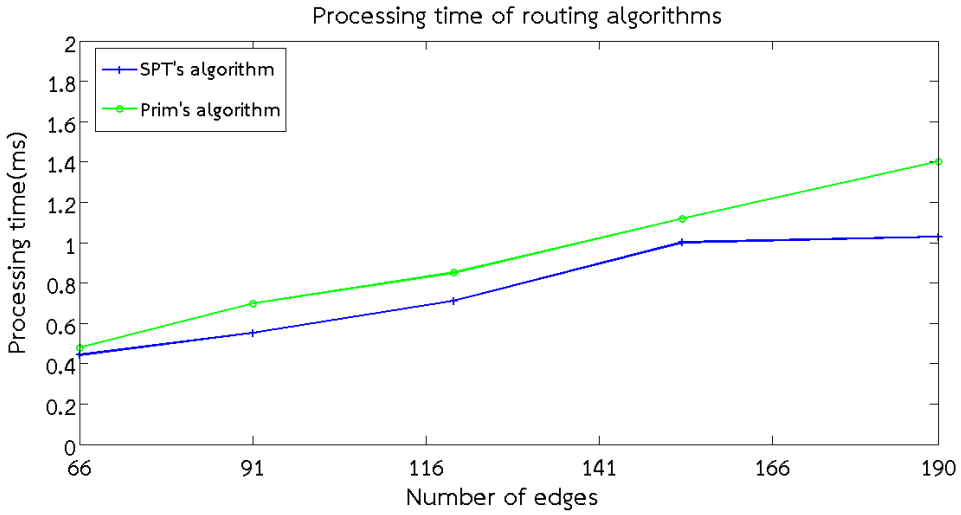
โดย E คือ จำนวนเส้นเชื่อมระหว่างโอเพนวีสวิตช์ทั้งหมดในโครงข่าย

V คือ จำนวนโอเพนวีสวิตช์ทั้งหมดในโครงข่าย

จากผลการทดลองเปรียบเทียบเวลาที่ใช้ในการคำนวณเส้นทางของทั้งสามขั้นตอนวิธีเมื่อมีจำนวนโอเพนวีสวิตช์ 12 โนดเท่ากัน เวลาที่ใช้ในการคำนวณเส้นทางของทั้งสามขั้นตอนวิธีมีแนวโน้มเพิ่มขึ้นเมื่อจำนวนเส้นเชื่อมเพิ่มขึ้น โดยเฉพาะการคำนวณด้วยขั้นตอนวิธีไดร์คสตราที่ใช้เวลาในการคำนวณสูงสุดเป็นเพราะการคำนวณต้องพิจารณาน้ำหนักของเส้นเชื่อมทีละคู่ของโอเพนวีสวิตช์ต้นทางและปลายทางทำให้การคำนวณมีความซับซ้อน ซึ่งสามารถแสดงความซับซ้อนของเวลา [13] ที่ใช้ใน

การคำนวณได้ดังสมการที่ 4.1 และความซับซ้อนของเวลาที่ใช้ในการคำนวณด้วยขั้นตอนวิธีพริมแสดงได้ดังสมการที่ 4.2

จากสมการความซับซ้อนของการคำนวณด้วยขั้นตอนวิธีไดร์คสตราและพริมมีความแตกต่างกันที่ขั้นตอนวิธีไดร์คสตราต้องนำจำนวนโอเพนวีสวิตช์คูณกับค่า log|V| แล้วจึงนำไปรวมกับจำนวนเส้นเชื่อม จึงทำให้การคำนวณเส้นทางของขั้นตอนวิธีไดร์คสตราใช้เวลามากกว่าการคำนวณของขั้นตอนวิธีพริม



รูปที่ 4.13: การทดลองเปรียบเทียบเวลาที่ใช้ในการคำนวณเส้นทาง
เมื่อเพิ่มจำนวนโอเพนวีสวิตช์และจำนวนเส้นเชื่อม

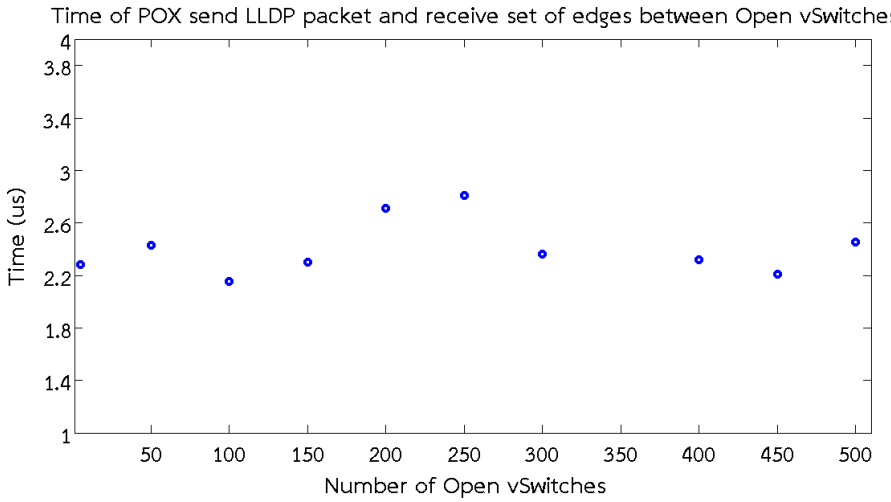
จากผลการทดลองรูปที่ 4.12 เวลาที่ใช้ในการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริมมีค่าใกล้เคียงกันมาก การทดลองต่อมาจึงทำการเพิ่มจำนวนโอเพนวีสวิตช์และจำนวนเส้นเชื่อมเพื่อดูความเปลี่ยนแปลงของเวลาที่ใช้คำนวณเส้นทางด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริม จากผลการทดลองรูปที่ 4.13 ที่โอเพนวีสวิตช์จำนวน 12 โนดและมีจำนวนเส้นเชื่อม 66 เส้น เวลาของการคำนวณเส้นทางด้วยขั้นตอนวิธีพริมมีมากกว่าขั้นตอนวิธีต้นไม้แบบทอดข้ามประมาณ 0.036 มิลลิวินาที และที่โอเพนวีสวิตช์จำนวน 20 โนดและมีจำนวน 190 เส้นเชื่อม เวลาของการคำนวณเส้นทางด้วยขั้นตอนวิธีพริมมีมากกว่าขั้นตอนวิธีต้นไม้แบบทอดข้ามประมาณ 0.37 มิลลิวินาที ซึ่งดังที่กล่าวไว้แล้วว่าขั้นตอนวิธีพริมมีการพิจารณาน้ำหนักของแต่ละเส้นเชื่อมทำให้ใช้เวลาในการคำนวณเส้นทางมากกว่าขั้นตอนวิธีต้นไม้แบบทอดข้ามที่พิจารณาเพียงหมายเลขของโอเพนวีสวิตช์

4.3 การทดลองวัดผลกระทบของการให้บริการไอพีทีวีด้วยโครงข่ายโอเพนโฟลว์

การให้บริการไอพีทีวีแบบดั้งเดิมนั้น โพรโทคอลของการคำนวณเส้นทางอาศัยการสร้างเส้นทางด้วยสวิตช์ที่เป็นจุดรวมข้อมูล จึงทำให้จุดรวมข้อมูลเปรียบเสมือนเป็นศูนย์กลางของการจัดการภายในโครงข่าย ซึ่งการทำงานของจุดรวมข้อมูลของการให้บริการไอพีทีวีแบบดั้งเดิมนั้นตรงกับแนวคิดของโครงข่ายเอสดีเอ็นที่มีตัวควบคุมเป็นศูนย์กลางการจัดการภายในโครงข่ายเช่นกัน ในงานวิจัยนี้จึงได้เสนอการใช้งานโครงข่ายเอสดีเอ็น/โอเพนโฟลว์เพื่อทดสอบการให้บริการไอพีทีวี ในการทดลองได้ทดสอบผลกระทบของตัวควบคุมพอกซ์กับการให้บริการไอพีทีวี โดยหน้าที่หลักของตัวควบคุมพอกซ์ที่คล้ายกับจุดรวมข้อมูลคือการเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์แต่ละโนดภายในโครงข่าย

ในงานวิจัยนี้ตัวควบคุมพอกซ์เริ่มทำงานด้วยการเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์ การทดลองนี้จึงเป็นการศึกษาเวลาที่ตัวควบคุมพอกซ์ใช้เรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์ โดยการใช้คำสั่ง core.openflow_discovery.adjacency ซึ่งข้อมูลขาออกที่ได้คือหมายเลขโอเพนวีสวิตช์และหมายเลขช่องทางที่มีเส้นเชื่อมระหว่างโอเพนวีสวิตช์ จากนั้นนำหมายเลขโอเพนวีสวิตช์และช่องทางมาสร้างเป็นเซตของการเชื่อมต่อระหว่างโอเพนวีสวิตช์ การทดลองเริ่มที่โอเพนวีสวิตช์จำนวน 5 โนดและเพิ่มขึ้นทีละ 50 โนด จนถึงจำนวน 500 โนด โดยทอพอโลยีของโอเพนวีสวิตช์ที่ใช้ในการทดลองแสดงดังรูปที่ 4.14

รูปที่ 4.14: ทอพอโลยีของการทดลองเวลาการเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์



รูปที่ 4.15: เวลาที่ตัวควบคุมพอกซ์ใช้เรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์ในโครงข่ายจำลอง

จากผลการทดลองรูปที่ 4.15 แกนแนวนอนแสดงจำนวนโอเพนวีสวิตช์ที่ถูกใช้ในการทดลองและแกนแนวตั้งแสดงเวลาที่ได้ของแต่ละจำนวนโอเพนวีสวิตช์ ซึ่งแสดงให้เห็นว่าแม้จำนวนโอเพนวีสวิตช์จะเพิ่มขึ้นแต่ค่าเวลาที่ได้มีทั้งเพิ่มขึ้นและลดลงซึ่งค่าต่ำสุดมีค่า 2.15 ไมโครวินาที และค่าสูงสุดมีค่า 2.80 ไมโครวินาที จากผลการทดลองการเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์ของตัวควบคุมพอกซ์มีหน่วยเป็นไมโครวินาทีซึ่งเมื่อเปรียบเทียบกับเวลาที่ใช้ในการส่งแพ็กเกตยูดีพีของการทดลองที่ 4.1 อยู่ในหน่วยมิลลิวินาที ดังนั้นเวลาที่ตัวควบคุมพอกซ์ใช้เรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์จึงไม่มีผลกระทบกับเวลาที่ใช้ในการให้บริการไอพีทีวี

4.4 การทดลองวัดคุณภาพของการสตรีมวีดีทัศน์

การให้บริการไอพีทีวีคือการส่งสัญญาณภาพและเสียงผ่านโครงข่ายอินเทอร์เน็ต ดังนั้นคุณภาพของภาพและเสียงจึงเป็นตัวแปรสำคัญของการให้บริการไอพีทีวี ในงานวิจัยนี้จึงมีการทดสอบคุณภาพของการสตรีมวีดีทัศน์บนโครงข่ายโอเพนโฟลว์

ตารางที่ 4.2: การตั้งค่าเนื้อหาไอพีทีวี

Output	UDP
Encapsulation	MPEG-TS
Video codec bitrate	1024 kb/s
Audio codec bitrate	192 kb/s

โดยการสตรีมวีดีทัศน์บนโครงข่ายรูปที่ 4.1 ที่มีการคำนวณเส้นทางด้วยสามขั้นตอนวิธี ซึ่งตัวบริการไอพีทีวีทำการสตรีมวีดีทัศน์ผ่านโปรแกรมวีแอลซีและลูกข่ายไอพีทีวีรับชมวีดีทัศน์ด้วยโปรแกรมวีแอลซีเช่นกัน การกำหนดค่าการสตรีมวีดีทัศน์ [19] แสดงดังตารางที่ 4.2

พารามิเตอร์ (parameter) ที่ถูกใช้ในการแสดงคุณภาพของวีดีทัศน์คือค่าอัตราส่วนของสัญญาณรบกวนสูงสุด [18] หรือพีเอสเอ็นอาร์ (PSNR: Peak signal-to-noise ratio) ซึ่งสามารถคำนวณได้ดังสมการที่ 4.3

$$\mathrm{PSNR} = 10\log_{10}\left[\frac{(2^{n}-1)^{2}}{\mathrm{RMSE}}\right] \tag{4.3}$$

โดย n คือ จำนวนบิตที่ใช้แทนค่าสีในแต่ละจุดภาพ

RMSE คือ ค่าผิดพลาดของค่ากำลังสองเฉลี่ยของรากที่สอง หรือ Root Mean Square Error หาได้จากสมการที่ 4.4

$$\mathrm{RMSE} = \sqrt{\sum \frac{(I-T)^{2}}{N}} \tag{4.4}$$

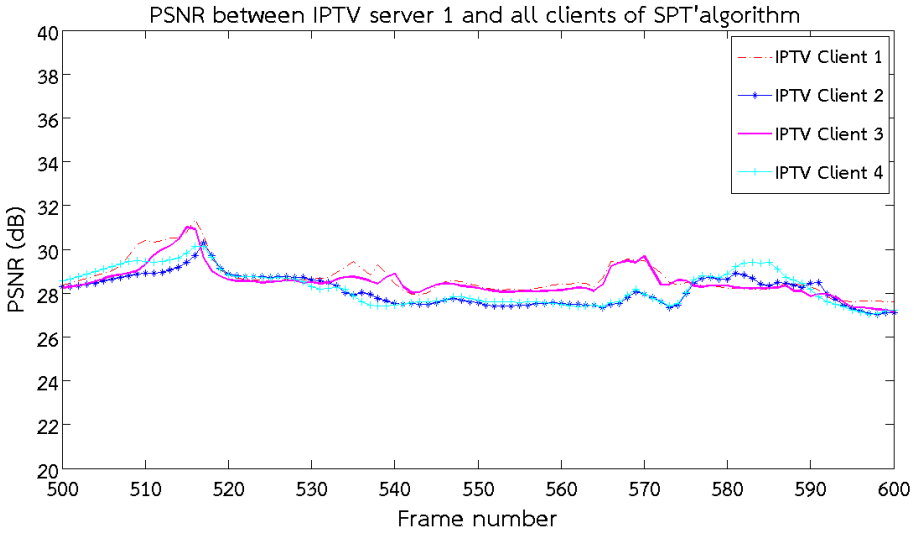
โดย I คือ ภาพจากตัวบริการไอพีทีวี

T คือ ภาพจากลูกข่ายไอพีทีวี

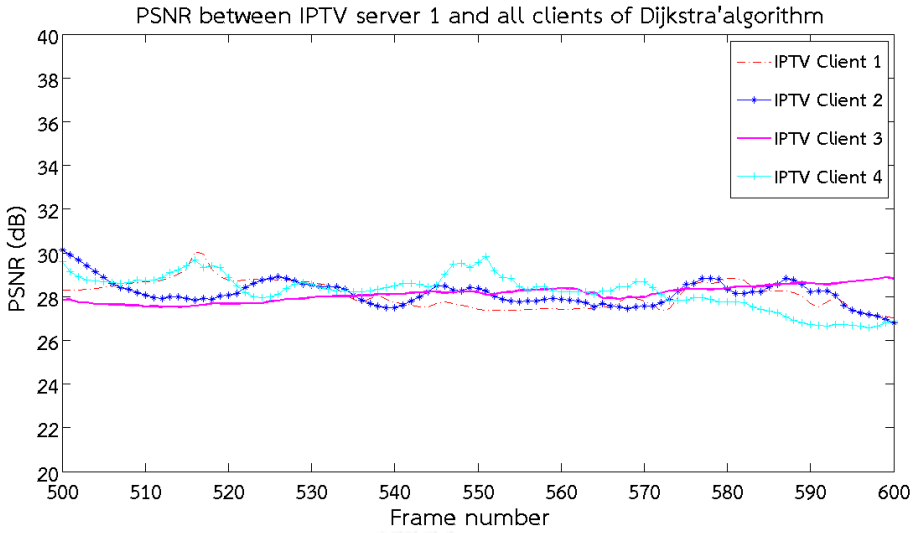
N คือ ขนาดของภาพ

ในการทดลองตัวบริการไอพีทีวี 1 ทำการสตรีมวีดีทัศน์ชื่อ Frozen.mp4 [20] ส่วนลูกข่ายไอพีทีวีทั้ง 4 ลูกข่ายรับวีดีทัศน์และบันทึกด้วยนามสกุล mp4 จากนั้นนำวีดีทัศน์ที่ตัวบริการไอพีทีวีใช้ในการสตรีมและที่ลูกข่ายไอพีทีวีรับมาเปลี่ยนเป็นนามสกุล yuv เพื่อใช้ในการคำนวณค่าพีเอสเอ็นอาร์ โดยวีดีทัศน์นามสกุล yuv มีขนาด 1280 x 572 จุดภาพ (pixel) และทำการคำนวณภาพที่เฟรม (frame) หมายเลข 500 ถึงเฟรมหมายเลข 600 ซึ่งเป็นช่วงที่มีค่าพีเอสเอ็นอาร์สูงที่สุด โดยรูปที่ 4.16, 4.17 และรูปที่ 4.18 แสดงค่าพีเอสเอ็นอาร์ของการสตรีมวีดีทัศน์ด้วยตัวบริการไอพีทีวี 1 บนเส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้าม ขั้นตอนวิธีไดร์คสตรา และขั้นตอนวิธีพริม ตามลำดับ แกนแนวนอนคือเฟรมหมายเลข 500 ถึงเฟรมหมายเลข 600 และแกนแนวตั้งคือค่าพีเอสเอ็นอาร์ของแต่ละเฟรม

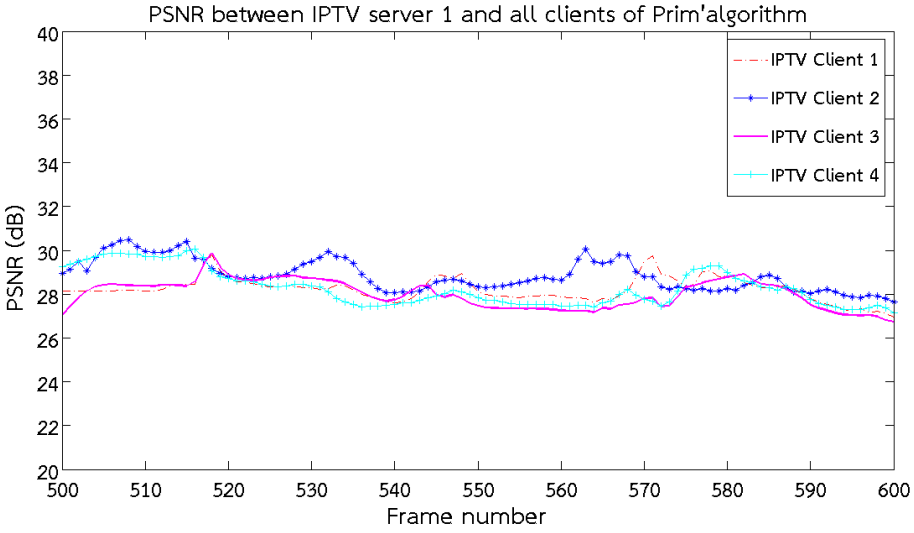
จากผลการทดลองวัดประสิทธิภาพของการสตรีมวีดีทัศน์บนโครงข่ายโอเพนโฟลว์ ที่มีการสตรีมวีดีทัศน์จากตัวบริการไอพีทีวี 1 ไปยังลูกข่ายไอพีทีวีทั้ง 4 ลูกข่าย แสดงให้เห็นว่าค่าพีเอสเอ็นอาร์กับคุณภาพของภาพรูปที่ 4.19 และรูปที่ 4.20 มีความสอดคล้องกัน ดังนั้นการสตรีมวีดีทัศน์ภายในโครงข่ายโอเพนโฟลว์ไม่เกิดการสูญเสียแพ็กเกตจนทำให้คุณภาพของวีดีทัศน์ลดลง



รูปที่ 4.16: ค่าพีเอสเอ็นอาร์ของการสตรีมวีดีทัศน์ระหว่างตัวบริการไอพีทีวี 1 และลูกข่ายไอพีทีวีทั้งหมดของขั้นตอนวิธีต้นไม้แบบทอดข้าม



รูปที่ 4.17: ค่าพีเอสเอ็นอาร์ของการสตรีมวีดีทัศน์ระหว่างตัวบริการไอพีทีวี 1 และลูกข่ายไอพีทีวีทั้งหมดของขั้นตอนวิธีไดร์คสตรา



รูปที่ 4.18: ค่าพีเอสเอ็นอาร์ของการสตรีมวีดีทัศน์ระหว่างตัวบริการไอพีทีวี 1 และลูกข่ายไอพีทีวีทั้งหมดของขั้นตอนวิธีพริม

(ก) เฟรมหมายเลข 1 ของตัวบริการไอพีทีวี 1

	
ลูกข่ายไอพีทีวี 1	ลูกข่ายไอพีทีวี 2
	
ลูกข่ายไอพีทีวี 3	ลูกข่ายไอพีทีวี 4

(ข) เฟรมหมายเลข 1 ของลูกข่ายไอพีทีวี

รูปที่ 4.19: คุณภาพวีดีทัศน์ของเฟรมหมายเลข 1 ของการส่งด้วยขั้นตอนวิธีไดร์คสตรา

(ก) เฟรมหมายเลข 500 ของตัวบริการไอพีทีวี 1

	
ลูกข่ายไอพีทีวี 1	ลูกข่ายไอพีทีวี 2
	
ลูกข่ายไอพีทีวี 3	ลูกข่ายไอพีทีวี 4

(ข) เฟรมหมายเลข 500 ของลูกข่ายไอพีทีวี

รูปที่ 4.20: คุณภาพวีดีทัศน์ของเฟรมหมายเลข 500 ของการส่งด้วยขั้นตอนวิธีไดร์คสตรา

บทที่ 5
บทสรุปและข้อเสนอแนะ

5.1 บทสรุป

งานวิจัยนี้นำเสนอการนำโครงข่ายโอเพนโฟลว์มาพัฒนาปรับใช้กับการคำนวณเส้นทางของการให้บริการไอพีทีวีแบบมัลติแคสต์ โดยผู้วิจัยได้สนใจศึกษาและพัฒนาการคำนวณเส้นทางต้นไม้แบบทอดข้ามในตัวควบคุมพอกซ์ซึ่งเป็นการคำนวณเส้นทางเพื่อให้ได้โครงข่ายต้นไม้แบบทอดข้ามโดยที่ไม่มีการพิจารณาน้ำหนักของเส้นเชื่อม ผู้วิจัยจึงได้เสนอการประยุกต์ใช้ขั้นตอนวิธี ไดร์คสตราและพริมเพื่อให้ตัวควบคุมพอกซ์มีการคำนวณเส้นทางโดยพิจารณาน้ำหนักของเส้นเชื่อมระหว่างโอเพนวีสวิตช์ด้วย จากผลการทดลองแสดงให้เห็นว่าเส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีไดร์คสตรามีความเหมาะสมกับการให้บริการไอพีทีวีแบบยูนิแคสต์ แต่เส้นทางที่ได้จากการคำนวณด้วยขั้นตอนวิธีต้นไม้แบบทอดข้ามและพริมมีความเหมาะสมกับการให้บริการไอพีทีวีแบบมัลติแคสต์ซึ่งเวลาของการส่งแพ็กเกตแบบมัลติแคสต์ของเส้นทางที่คำนวณด้วยขั้นตอนพริมใช้เวลารวมน้อยกว่าขั้นตอนวิธีต้นไม้แบบทอดข้ามนั้นเป็นเพราะขั้นตอนวิธีพริมมีการคำนวณเส้นทางที่พิจารณาน้ำหนักของเส้นเชื่อมรวมน้อยที่สุด แต่ขั้นตอนวิธีต้นไม้แบบทอดข้ามเป็นการพิจารณาเพียงลำดับหมายเลขของโอเพนวีสวิตช์ซึ่งแม้เวลาที่ใช้ในการคำนวณเส้นทางของขั้นตอนวิธีพริมจะมากกว่าขั้นตอนวิธีต้นไม้แบบทอดข้ามแต่ผลการทดลองรูปที่ 4.9 และรูปที่ 4.10 แสดงให้เห็นแล้วว่า ในโครงข่ายที่มีขนาดใหญ่เวลาที่ใช้ในการส่งแพ็กเกตรวมของขั้นตอนวิธีพริมก็ได้ค่าเวลาน้อยกว่าขั้นตอนวิธีต้นไม้แบบทอดข้าม ในส่วนของผลกระทบของตัวควบคุมพอกซ์กับการให้บริการไอพีทีวีแสดงให้เห็นแล้วว่าการเรียนรู้การเชื่อมต่อระหว่างโอเพนวีสวิตช์มีหน่วยเป็นไมโครวินาทีซึ่งไม่มีผลกับเวลาของการส่งแพ็กเกตที่มีหน่วยเป็นมิลลิวินาทีและผลของการสตรีมวีดีทัศน์ด้วยโครงข่ายโอเพนโฟลว์ก็แสดงให้เห็นถึงคุณภาพของวีดีทัศน์ที่ตัวบริการไอพีทีวีสตรีมไปยังลูกข่ายไอพีทีวี โดยค่าพีเอสเอ็นอาร์แสดงให้เห็นว่าคุณภาพของวีดีทัศน์อยู่ในระดับปกติไม่เกิดการสูญเสียแพ็กเกตจนทำให้คุณภาพของวีดีทัศน์นั้นลดลง

5.2 ข้อเสนอแนะ

5.2.1 การพัฒนาโปรแกรม

ในงานวิจัยนี้เป็นการพิจารณาค่าน้ำหนักเส้นเชื่อมระหว่างโอเพนวีสวิตช์ ซึ่งเป็นค่าน้ำหนักเส้นเชื่อมที่ได้มาจากการสุ่ม ในอนาคตจึงควรศึกษาวิธีการรับค่าน้ำหนักเส้นเชื่อมจากทอพอโลยีที่มีการทดลองจริง เพื่อนำมาคำนวณเส้นทางของการส่งแพ็กเกตระหว่างตัวบริการไอพีทีวีและลูกข่ายไอพีทีวี

รูปที่ 5.1: ระบบทดสอบโอเพนโฟลว์ขนาดเล็ก

5.2.2 การทดสอบบนระบบทดสอบโอเพนโฟลว์ขนาดเล็ก (testbed)

งานวิจัยนี้ประสบความสำเร็จในการพัฒนาโปรแกรมการคำนวณเส้นทางของการให้บริการไอพีทีวีบนโครงข่ายโอเพนโฟลว์ด้วยโปรแกรมมินิเน็ทซึ่งเป็นโปรแกรมสำหรับการจำลองการทำงานภายในโครงข่าย ดังนั้นเพื่อให้ได้ผลการทดลองที่เสมือนการใช้งานระบบจริง ในอนาคตจึงควรมีการทดลองบนระบบทดสอบโอเพนโฟลว์ขนาดเล็กเพื่อให้ได้ผลการทดลองที่เสมือนการทำงานจริงเนื่องจากระบบทดสอบโอเพนโฟลว์ขนาดเล็กแสดงดังรูปที่ 5.1 ประกอบด้วยคอมพิวเตอร์ที่ถูกจำลองให้เป็นโฮสต์ โอเพนวีสวิตช์ และตัวควบคุมพอกซ์ซึ่งจะมีสภาพแวดล้อมในการทดสอบที่เสมือนระบบจริง

รายการอ้างอิง

[1] *SDN architecture: The new norm for networks, ONF White Paper*, 2014.

[2] L. Prete, F. Farina, M. Campanella, and A. Biancini, "Energy efficient minimum spanning tree in OpenFlow networks," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, 2012, pp. 36-41.

[3] C.-C. Wen, C.-S. Wu, and M.-T. Yang, "Hybrid tree based explicit routed multicast for QoS supported IPTV service," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, 2009, pp. 1-6.

[4] J. Ko, S. Park, and E. Lee, "An extended PIM-SM for efficient data transmission in IPTV services," in *Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on*, 2010, pp. 115-119.

[5] P. McDonagh, C. Olariu, A. Hava, and C. Thorpe, "Enabling IPTV Service Assurance using OpenFlow," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, 2013, pp. 1456-1460.

[6] L. Bondan, L. F. Müller, and M. Kist, "Multiflow: Multicast clean-slate with anticipated route calculation on OpenFlow programmable networks," *Journal of Applied Computing Research,* vol. 2, pp. 68-74, 2013.

[7] S. K. Sahana, A. Jain, and A. Mustafi, "A Comparative Study on Multicast Routing using Dijkstra's, Prims and Ant Colony Systems," *International Journal of Computer Engineering & Technology (IJCET),* vol. 2, pp. 301-310, 2010.

[8] "OpenFlow Switch Specification," vol. 1.1.0, ed, 2011.

[9] "The Evolving IPTV Service Architecture ", ed. USA, 2007, pp. 1-12.

[10] F. J. Hens and J. M. Caballero, *Triple Play: Building the converged network for IP, VoIP and IPTV* vol. 3: John Wiley & Sons, 2008.

[11] T. Yonghui and R. Hu, "A resolution for IGMP v3 protocol using finite state machine," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, 2012, pp. 517-520.

[12] D. Medhi, *Network routing: algorithms, protocols, and architectures*: Morgan Kaufmann, 2010.

[13] *Graph Algorithms: Dijkstra's Algorithm, Travelling Salesman Problem, Kruskal's Algorithm, Prim's Algorithm, Shortest Path Problem, Nearest Neighbour Algorithm, Ford-Fulkerson Algorithm, Knight's Tour, Minimax, A* Search Algorithm, Depth-first Search*: General Books, 2011.

[14] M. Grimes and J. Nathan, "The Nemesis Project Documentation," ed, 2011.

[15] VLC media player [Online]. Available: http://www.videolan.org/vlc

[16] POX program [Online]. Available: https://github.com/noxrepo/pox

[17] D. Marschke, J. Doyle, and P. Moyer, *Software Defined Networking (SDN): Anatomy of OpenFlow Volume I*: You Lulu Incorporated, 2015.

[18] S. Qureshi, *Embedded Image Processing on the TMS320C6000TM DSP: Examples in Code Composer StudioTM and MATLAB*: Springer, 2005.

[19] IPTV setting [Online]. Available: http://wiki.openpli.org/GenericIPTV

[20] Frozen movie [Online]. Available: http//www. http://disney.wikia.com

ภาคผนวก ก

การจำลองโครงข่ายโอเพนโฟลว์ด้วยโปรแกรมมินิเน็ท

ในการจำลองโครงข่ายโอเพนโฟลว์ซึ่งประกอบด้วยโฮสต์ โอเพนวีสวิตช์และตัวควบคุมพอกซ์ที่ถูกใช้ในการทดลอง สามารถเขียนได้ดังโปรแกรมภาคผนวก ก

```
1.  #topo-thesis.py by Pornnipa Rattanawadee
2.  #To create thesis topology
3.  #compose of 6 hosts and 12 Open vSwitches
4.
5.  from mininet.topo import Topo
6.  class MyTopo( Topo ):
7.      "Simple topology example."
8.
9.      def __init__( self ):
10.         "Create custom topo."
11.         # Initialize topology
12.         Topo.__init__( self )
13.         # Add hosts and switches
14.         host1 = self.addHost( 'h1' )       #IPTV server 1
15.         host2 = self.addHost( 'h2' )       #IPTV server 2
16.         host3 = self.addHost( 'h3' )       #IPTV client 1
17.         host4 = self.addHost( 'h4' )       #IPTV client 2
18.         host5 = self.addHost( 'h5' )       #IPTV client 3
19.         host6 = self.addHost( 'h6' )       #IPTV client 4
20.         Switch1 = self.addSwitch( 's1' )    #Open vSwitch 1
21.         Switch2 = self.addSwitch( 's2' )    #Open vSwitch 2
22.         Switch3 = self.addSwitch( 's3' )    #Open vSwitch 3
23.         Switch4 = self.addSwitch( 's4' )    #Open vSwitch 4
24.         Switch5 = self.addSwitch( 's5' )    #Open vSwitch 5
25.         Switch6 = self.addSwitch( 's6' )    #Open vSwitch 6
26.         Switch7 = self.addSwitch( 's7' )    #Open vSwitch 7
27.         Switch8 = self.addSwitch( 's8' )    #Open vSwitch 8
28.         Switch9 = self.addSwitch( 's9' )    #Open vSwitch 9
29.         Switch10 = self.addSwitch( 's10' )  #Open vSwitch 10
```

```
30.         Switch11 = self.addSwitch( 's11' )  #Open vSwitch 11
31.         Switch12 = self.addSwitch( 's12' )  #Open vSwitch 12
32.         linkopts1 = dict(bw=10) #defined bandwidth = 10 Mbps of all edges
33.         self.addLink(host1, Switch1, delay = '5ms', **linkopts1 )
34.         self.addLink(host2, Switch2, delay = '5ms', **linkopts1 )
35.         self.addLink(host3, Switch6, delay = '5ms', **linkopts1 )
36.         self.addLink(Switch10, host4, delay = '5ms', **linkopts1 )
37.         self.addLink(Switch12, host5, delay = '5ms', **linkopts1 )
38.         self.addLink(Switch11, host6, delay = '5ms', **linkopts1 )
39.         self.addLink(Switch2, Switch1, delay = '4ms', **linkopts1 )
40.         self.addLink(Switch1, Switch3, delay = '3ms', **linkopts1 )
41.         self.addLink(Switch3, Switch6, delay = '3ms', **linkopts1 )
42.         self.addLink(Switch6, Switch5, delay = '4ms', **linkopts1 )
43.         self.addLink(Switch5, Switch4, delay = '4ms', **linkopts1 )
44.         self.addLink(Switch2, Switch4, delay = '1ms', **linkopts1 )
45.         self.addLink(Switch4, Switch7, delay = '3ms', **linkopts1 )
46.         self.addLink(Switch7, Switch2, delay = '2ms', **linkopts1 )
47.         self.addLink(Switch7, Switch11, delay = '4ms', **linkopts1 )
48.         self.addLink(Switch4, Switch8, delay = '4ms', **linkopts1 )
49.         self.addLink(Switch8, Switch11, delay = '3ms', **linkopts1 )
50.         self.addLink(Switch4, Switch11, delay = '5ms', **linkopts1 )
51.         self.addLink(Switch5, Switch8, delay = '2ms', **linkopts1 )
52.         self.addLink(Switch5, Switch9, delay = '3ms', **linkopts1 )
53.         self.addLink(Switch9, Switch12, delay = '1ms', **linkopts1 )
54.         self.addLink(Switch11, Switch12, delay = '1ms', **linkopts1 )
55.         self.addLink(Switch10, Switch12, delay = '4ms', **linkopts1 )
56.         self.addLink(Switch6, Switch10, delay = '3ms', **linkopts1 )
57.         self.addLink(Switch6, Switch9, delay = '3ms', **linkopts1 )
58.         self.addLink(Switch9, Switch10, delay = '2ms', **linkopts1 )
59.         self.addLink(Switch1, Switch4, delay = '5ms', **linkopts1 )
60.         self.addLink(Switch3, Switch4, delay = '6ms', **linkopts1 )
61.         self.addLink(Switch3, Switch5, delay = '2ms', **linkopts1 )
62.         self.addLink(Switch8, Switch9, delay = '2ms', **linkopts1 )
63.     topos = { 'mytopo': ( lambda: MyTopo() ) }
```

โปรแกรมที่ ก: การจำลองโครงข่ายในโปรแกรมมินิเน็ท

ภาคผนวก ข

โปรแกรมการจัดการเส้นทางที่มีในตัวควบคุมพอกซ์

โปรแกรมการจัดเส้นทางด้วยต้นไม้แบบทอดข้ามที่มีอยู่ในตัวควบคุมพอกซ์อยู่แล้วนั้นสามารถแสดงได้ดังโปรแกรมภาคผนวก ข

```
1.  #SPT.py by POX controller
2.  #To calculate spanning tree
3.  #Run with POX_manage.py
4.
5.  from pox.core import core
6.  import pox.openflow.libopenflow_01 as of
7.  from pox.lib.revent import *
8.  from collections import defaultdict
9.  from pox.openflow.discovery import Discovery
10. from pox.lib.util import dpidToStr
11. from pox.lib.recoco import Timer
12. from pox.lib.packet import *
13. import time
14.
15. log = core.getLogger()
16. def _calc_spanning_tree ():
17.   def flip (link):
18.     return Discovery.Link(link[2],link[3], link[0],link[1])
19.   adj = defaultdict(lambda:defaultdict(lambda:[]))
20.   switches = set()
21.   # Add all links and switches
22.   for l in core.openflow_discovery.adjacency:
23.     adj[l.dpid1][l.dpid2].append(l)
24.     switches.add(l.dpid1)
25.     switches.add(l.dpid2)
26.   # Cull links -- we want a single symmetric link connecting nodes
27.   for s1 in switches:
28.     for s2 in switches:
29.       if s2 not in adj[s1]:
30.         continue
```

```
31.       if not isinstance(adj[s1][s2], list):
32.         continue
33.       assert s1 is not s2
34.       good = False
35.       for l in adj[s1][s2]:
36.         if flip(l) in core.openflow_discovery.adjacency:
37.           # This is a good one
38.           adj[s1][s2] = l.port1
39.           adj[s2][s1] = l.port2
40.           good = True
41.           break
42.       if not good:
43.         del adj[s1][s2]
44.         if s1 in adj[s2]:
45.           # Delete the other way too
46.           del adj[s2][s1]
47.   q = []
48.   more = set(switches)
49.   done = set()
50.   tree = defaultdict(set)
51.   while True:
52.     q = sorted(list(more)) + q
53.     more.clear()
54.     if len(q) == 0: break
55.     v = q.pop(False)
56.     if v in done: continue
57.     done.add(v)
58.     for w,p in adj[v].iteritems():
59.       if w in tree: continue
60.       more.add(w)
61.       tree[v].add((w,p))
62.       tree[w].add((v,adj[w][v]))
63.   if False:
64.     log.debug("*** SPANNING TREE ***")
65.     for sw,ports in tree.iteritems():
```

```
66.       #print " ", dpidToStr(sw), ":", sorted(list(ports))
67.       #print " ", sw, ":", [l[0] for l in sorted(list(ports))]
68.       log.debug((" %i : " % sw) + " ".join([str(l[0]) for l in
69.                                             sorted(list(ports))]))
70.     log.debug("**********************")
71.   return tree
72.
73. def _update_spt_tree (force_dpid = None):
74.   tree = _calc_spanning_tree()
75.   log.debug("Spanning tree updated")
76.   return tree
```

โปรแกรมที่ ข: การจัดการเส้นทางของต้นไม้แบบทอดข้ามที่มีในตัวควบคุมพอกซ์ [21]

ภาคผนวก ค

โปรแกรมการจัดการเส้นทางที่นำมาใช้

โปรแกรมในตัวควบคุมพอกซ์ถูกแบ่งออกเป็นสองหน้าที่คือโปรแกรมเพื่อการคำนวณเส้นทางและโปรแกรมการจัดการเพิ่มโฟลว์เอนทรีให้กับโอเพนวีสวิตช์ ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

```
1.  #Dijkstra.py is applied from SPT.py
2.  #To calculate Dijkstra algorithm
3.  #Run with POX_manage.py
4.
5.  from pox.core import core
6.  import pox.openflow.libopenflow_01 as of
7.  from pox.lib.packet import ethernet, ipv4, ICMP
8.  from pox.lib.revent import *
9.  from collections import defaultdict
10. from pox.openflow.discovery import Discovery
11. from pox.lib.util import dpidToStr
12. from pox.lib.recoco import Timer
13. from priodict.priodict import priorityDictionary
14. import bisect
15. import time
16.
17. log = core.getLogger()
18. def _calc_topology (force_dpid = None):
19. #To send LLDP to create set of topology
20.   def flip (link):
21.     return Discovery.Link(link[2],link[3], link[0],link[1])
22.   adj = defaultdict(lambda:defaultdict(lambda:[ ]))
23.   switches = set()
24.   # To create set of topology elements in "adj"
25.   # and set of switches in "switches"
26.   for l in core.openflow_discovery.adjacency:
27.     adj[l.dpid1][l.dpid2].append(l)
28.     switches.add(l.dpid1)
29.     switches.add(l.dpid2)
```

```
30.     for s1 in switches:
31.       for s2 in switches:
32.         if s2 not in adj[s1]:
33.           continue
34.         if not isinstance(adj[s1][s2], list):
35.          continue
36.         assert s1 is not s2
37.         good = False
38.         for l in adj[s1][s2]:
39.           if flip(l) in core.openflow_discovery.adjacency:
40.             # This is a good one
41.             adj[s1][s2] = l.port1
42.             adj[s2][s1] = l.port2
43.             good = True
44.             break
45.         if not good:
46.           del adj[s1][s2]
47.           if s1 in adj[s2]:
48.             # Delete the other way too
49.             del adj[s2][s1]
50.     return adj,switches
51.
52.   def Shortest(S,E=None):
53.   # To calculate dijkstra's algorithm
54.       DIJ = { }
55.       ROT = { }
56.       N_W = priorityDictionary()
57.       N_W[S] = 0
58.       # G is set of edge weight between OpenvSwitches
59.       G = {1: {3:5}, 2: {4:5}, 3: {1:5, 5:7, 6:6}, 4: {2:5, 6:9, 7:3},
60.            5: {3:7, 8:7, 9:4}, 6: {3:6, 4:9, 8:16, 9:13, 10:9},
61.            7: {4:3, 9:5, 10:5}, 8: {5:7, 6:16, 11:8},
62.            9: {5:4, 6:13, 7:5, 11:18, 12:4}, 10: {6:9, 7:5, 12:12},
63.            11: {8:8, 9:18}, 12: {9:4, 10:12}}
64.       for v in N_W:
```

```
65.             DIJ[v] = N_W[v]
66.             if v == E: break
67.             for w in G[v]:
68.                 vwLength = DIJ[v] + G[v][w]
69.                 if w in DIJ:
70.                     if vwLength < DIJ[w]:
71.                         raise ValueError, \
72.       " hai messo metriche negative"
73.                 elif w not in N_W or vwLength < N_W[w]:
74.                     N_W[w] = vwLength
75.                     ROT[w] = v
76.         return (DIJ,ROT)
77.
78.     def Dijkstra(R_G,S,SW):
79.     # To create dijkstra graph in "TREE"
80.             Dij_path = defaultdict(set)
81.             TREE = defaultdict(set)
82.             more = set(SW)
83.             Q = [ ]
84.             done = set()
85.             for END in R_G.keys():
86.              if S is END:
87.                continue
88.              else:
89.              D,P = Shortest(S,END)
90.              PART = [ ]
91.              while 1:
92.               PART.append(END)
93.               if END == S: break
94.               END = P[END]
95.              PART.reverse()
96.              length = len(PART)
97.              for l in range(length-1):
98.                sw1 = PART[l]
99.                sw2 = PART[l+1]
```

```
100.            if sw2 is None:
101.              continue
102.            else:
103.              PORT = R_G[sw1][sw2]
104.              Dij_path[sw1].add((sw2,PORT))
105.        while True:
106.          Q = sorted(list(more)) + Q
107.          more.clear()
108.          if len(Q) == 0: break
109.          v = Q.pop(False)
110.          if v in done: continue
111.          done.add(v)
112.          for w,p in Dij_path[v]:
113.            more.add(w)
114.            TREE[v].add((w,p))
115.            TREE[w].add((v,R_G[w][v]))
116.        return TREE
117.
118.  def _update_Dijkstra_tree (force_dpid):
119.  #To call _calc_topology function and return "dijk_tree" to POX_dijkstra.py
120.    adj,switches = _calc_topology(force_dpid)
121.    log.debug("Graph updated")
122.    dijk_tree = Dijkstra(adj,force_dpid,switches)
123.    return dijk_tree
```

โปรแกรมที่ ค .1: ขั้นตอนวิธีของไดร์คสตรา (Dijkstra.py)

```
1.  #Prim.py is applied from SPT.py
2.  #To calculate Prim algorithm
3.  #Run with POX_manage.py
4.
5.  from pox.core import core
6.  import pox.openflow.libopenflow_01 as of
7.  from pox.lib.packet import ethernet, ipv4, ICMP
8.  from pox.lib.revent import *
9.  from collections import defaultdict
```

```
10.  from pox.openflow.discovery import Discovery
11.  from pox.lib.util import dpidToStr
12.  from pox.lib.recoco import Timer
13.  from heapq import *
14.  import time
15.
16.  log = core.getLogger()
17.  def Prim (S):
18.  #To send LLDP to create set of topology and calculate Prim's algorithm
19.    def flip (link):
20.      return Discovery.Link(link[2],link[3], link[0],link[1])
21.
22.    ad = defaultdict(lambda:defaultdict(lambda:[ ]))
23.    switches = set()
24.
25.    # To create set of topology elements in "adj"
26.    # and set of switches in "switches"
27.    for l in core.openflow_discovery.adjacency:
28.      ad[l.dpid1][l.dpid2].append(l)
29.      switches.add(l.dpid1)
30.      switches.add(l.dpid2)
31.    for s1 in switches:
32.      for s2 in switches:
33.        if s2 not in ad[s1]:
34.          continue
35.        if not isinstance(ad[s1][s2], list):
36.          continue
37.        assert s1 is not s2
38.        good = False
39.        for l in ad[s1][s2]:
40.          if flip(l) in core.openflow_discovery.adjacency:
41.            # This is a good one
42.            ad[s1][s2] = l.port1
43.            ad[s2][s1] = l.port2
44.            good = True
```

```
45.             break
46.         if not good:
47.           del ad[s1][s2]
48.           if s1 in ad[s2]:
49.             # Delete the other way too
50.             del ad[s2][s1]
51.     # G is set of edge weight between OpenvSwitches
52.     VER = [S]
53.     G = [ (1, 3, 5), (2, 4, 5), (3, 5, 7), (3, 6, 6), (4, 6, 9), (4, 7, 3),
54.           (5, 8, 7), (5, 9, 4), (6, 8, 16), (6, 9, 13), (6, 10, 9), (7, 9, 5),
55.           (7, 10, 5), (8, 11, 8), (9, 11, 18), (9, 12, 4), (10, 12, 12)]
56.     # Prim's algorithm calculation
57.     ADJ = defaultdict( list )
58.     for n1,n2,c in G:
59.           ADJ[ n1 ].append( (c, n1, n2) )
60.           ADJ[ n2 ].append( (c, n2, n1) )
61.     MST = [ ]
62.     USED = [ VER[ 0 ] ]
63.     EDG_U = ADJ[S][:]
64.     heapify( EDG_U )
65.     while EDG_U:
66.           COT, N1, N2 = heappop( EDG_U )
67.           if N2 not in USED:
68.               USED.append( N2 )
69.               MST.append( ( N1, N2, COT ) )
70.               for E in ADJ[ N2 ]:
71.                   if E[ 2 ] not in USED:
72.                       heappush( EDG_U, E )
73.     if False:
74.       log.debug("*** SPANNING TREE ***")
75.       for sw,ports in tree.iteritems():
76.         #print " ", dpidToStr(sw), ":", sorted(list(ports))
77.         #print " ", sw, ":", [l[0] for l in sorted(list(ports))]
78.         log.debug((" %i : " % sw) + " ".join([str(l[0]) for l in
79.                                               sorted(list(ports))]))
```

```
80.       log.debug("**********************")
81.     return MST,ad
82.
83.   def _convert_graph (force_dpid):
84.   # To create prim graph in "TREE"
85.       prim_tree = defaultdict(set)
86.       mst,adj = Prim(force_dpid)
87.       for i,j,k in mst:
88.         if j in adj[i].keys():
89.           prim_tree[i].add((j,adj[i][j]))
90.           prim_tree[j].add((i,adj[j][i]))
91.       return prim_tree
92.
93.   def _update_Prim_tree (force_dpid):
94.   #To call _calc_topology function and return "tree" to POX_manage.py
95.       tree = _convert_graph(force_dpid)
96.       log.debug("Prim algorithm tree updated")
97.       return tree
```

โปรแกรมที่ ค .2: ขั้นตอนวิธีของพริม (Prim.py)

```
1.   #POX_manage.py is applied from SPT.py
2.   #To manage calculation in POX controller
3.   #./pox.py forwarding.l2_learning openflow.discovery openflow.POX_manage
4.   # --no-flood --hold-down
5.
6.   from pox.core import core
7.   import pox.openflow.libopenflow_01 as of
8.   from pox.lib.util import dpid_to_str
9.   from pox.lib.util import str_to_bool
10.  from pox.lib.addresses import IPAddr, EthAddr
11.  from pox.lib.revent import *        #
12.  from pox.lib.packet import * #ethernet, ipv4, ICMP, UDP, IGMP
13.  from pox.openflow.discovery import Discovery        #
14.  from threading import Timer
15.  from collections import defaultdict
```

```
16.  import time
17.  from time import time
18.  import datetime
19.  from pox.openflow.spanning_tree import _update_tree  #to call spanning_tree program
20.  from pox.openflow.SPT import _update_spt_tree          #to call update_tree of SPT.py
21.  from pox.openflow.dijkstra import _update_Dijkstra_tree # call udpate_tree of Dijkstra.py
22.  from pox.openflow.prim import _update_Prim_tree        #to call update_tree of Prim.py

23.  log = core.getLogger()
24.  def _handle_PacketIn (event):
25.  # To consider packet type and call _Call_routing function
26.          log.debug("Switch %s has come up.", dpid_to_str(event.dpid))
27.          packet = event.parsed #parse pkt
28.          if packet.type == packet.ARP_TYPE:
29.              ip_packet = packet.payload
30.              _update_tree(event.dpid)
31.          elif packet.type == packet.IP_TYPE:
32.              ip_packet = packet.payload
33.              if ip_packet.protocol == ipv4.IGMP_PROTOCOL:
34.                  igmp_packet = ip_packet.payload
35.              ##### IGMP Query  #####
36.                  if igmp_packet.ver_and_type == 17:
37.                      _update_tree(event.dpid)
38.              ##### IGMP Report  #####
39.                  elif igmp_packet.ver_and_type == 18:
40.                      if ip_packet.srcip == IPAddr("10.0.0.3"):
41.  #                          print "*****Client 1 joint group*****"
42.                              des = core.openflow.getConnection(6)
43.                              des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
44.                                      match=of.ofp_match(dl_type=0x800,nw_proto=17,
45.                                      nw_dst=IPAddr("10.0.0.3")) ))
46.                      elif ip_packet.srcip == IPAddr("10.0.0.4"):
47.  #                          print "*****Client 2 joint group*****"
48.                              des = core.openflow.getConnection(10)
49.                              des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
```

```
50.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17,
51.                                 nw_dst=IPAddr("10.0.0.4")) ))
52.                         elif ip_packet.srcip == IPAddr("10.0.0.5"):
53.     #                       print "*****Client 3 joint group*****"
54.                             des = core.openflow.getConnection(12)
55.                             des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
56.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17,
57.                                 nw_dst=IPAddr("10.0.0.5")) ))
58.                         elif ip_packet.srcip == IPAddr("10.0.0.6"):
59.     #                       print "*****Client 4 joint group*****"
60.                             des = core.openflow.getConnection(11)
61.                             des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
62.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17,
63.                                 nw_dst=IPAddr("10.0.0.6")) ))
64.                         if ip_packet.dstip == IPAddr("10.0.0.1"):       #IPTV server1
65.                             _Call_routing(1)
66.                         elif ip_packet.dstip == IPAddr("10.0.0.2"):     #IPTV server2
67.                             _Call_routing(2)
68.                     ##### IGMP Prune  #####
69.                     elif igmp_packet.ver_and_type == 23:
70.                         if ip_packet.srcip == IPAddr("10.0.0.3"):
71.                             des = core.openflow.getConnection(6)
72.                             des.send( of.ofp_flow_mod( action=[of.ofp_action_output(port=1),
73.                                 of.ofp_action_output(port=4)],
74.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17,
75.                                 nw_dst=IPAddr("10.0.0.3")), command=of.OFPFC_DELETE ))
76.                         elif ip_packet.srcip == IPAddr("10.0.0.4"):
77.                             des = core.openflow.getConnection(10)
78.                             des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
79.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17,
80.                                 nw_dst=IPAddr("10.0.0.4")), command=of.OFPFC_DELETE ))
81.                         elif ip_packet.srcip == IPAddr("10.0.0.5"):
82.                             des = core.openflow.getConnection(12)
83.                             des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
84.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17,
```

```
85.                                 nw_dst=IPAddr("10.0.0.5")), command=of.OFPFC_DELETE ))
86.                     elif ip_packet.srcip == IPAddr("10.0.0.6"):
87.                             des = core.openflow.getConnection(11)
88.                             des.send( of.ofp_flow_mod( action=of.ofp_action_output(port=1),
89.                                     match=of.ofp_match(dl_type=0x800,nw_proto=17,
90.                                     nw_dst=IPAddr("10.0.0.6")), command=of.OFPFC_DELETE ))
91.
92.     def _Call_routing(force_dpid):
93.     # To manage route calculation and add forwarding rules to OpenvSwitches
94.                     tree = _update_SPT_tree(force_dpid)         #SPT calculation
95.                     tree = _update_Dijkstra_tree(force_dpid)    #Dijkstra calculation
96.                     tree = _update_Prim_tree(force_dpid)        #Prim calculation
97.                     sw_use = []
98.                     for sw, ports in tree.iteritems():
99.                         port_out = []
100.                        for port in ports:
101.                            port_out.append(port[1])
102.                        son = core.openflow.getConnection(sw)
103.                        if len(port_out) == 1:
104.                         son.send( of.ofp_flow_mod( action=of.ofp_action_output(port=port_out[0]),
105.                                 match=of.ofp_match(dl_type=0x800,nw_proto=17)) )
106.                        elif len(port_out) == 2:
107.                         son.send( of.ofp_flow_mod( action=of.ofp_action_output(port=port_out[1]),
108.                             match=of.ofp_match(in_port=port_out[0],dl_type=0x800,nw_proto=17)) )
109.                         son.send( of.ofp_flow_mod( action=of.ofp_action_output(port=port_out[0]),
110.                             match=of.ofp_match(in_port=port_out[1],dl_type=0x800,nw_proto=17)) )
111.                        elif len(port_out) == 3:
112.                         son.send( of.ofp_flow_mod( action=[of.ofp_action_output(port=port_out[1]),
113.                             of.ofp_action_output(port=port_out[2])],
114.                             match=of.ofp_match(in_port=port_out[0],dl_type=0x800,nw_proto=17)) )
115.                         son.send( of.ofp_flow_mod( action=[of.ofp_action_output(port=port_out[0]),
116.                             of.ofp_action_output(port=port_out[2])],
117.                             match=of.ofp_match(in_port=port_out[1],dl_type=0x800,nw_proto=17)) )
118.                         son.send( of.ofp_flow_mod( action=[of.ofp_action_output(port=port_out[0]),
119.                             of.ofp_action_output(port=port_out[1])],
```

```
120.                         match=of.ofp_match(in_port=port_out[2],dl_type=0x800,nw_proto=17)) )
121.
122. def launch (no_flood = False, hold_down = False):
123. #This funcion start working when POX_manage run
124.   def start_multi_network ():
125.     core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
126.     log.debug("Multicast network start")
127.   core.call_when_ready(start_multi_network, "openflow_discovery")
```

โปรแกรมที่ ค .3: การจัดการเส้นทางและการเพิ่มโฟลว์เอนทรีให้กับโอเพนวีสวิตช์

ประวัติผู้เขียนวิทยานิพนธ์

พรณิภา รัตนาวดี เกิดเมื่อวันที 19 กรกฎาคม พ.ศ. 2531 ที่กรุงเทพมหานคร สำเร็จการศึกษาในระดับประกาศนียบัตรวิชาชีพ สาขาวิศวกรรมไฟฟ้าจากสถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ ในปีการศึกษา 2549 จากนั้นเข้าศึกษาในระดับปริญญาตรีคณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมไฟฟ้า มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ และสำเร็จการศึกษาในปีการศึกษา 2553 จากนั้นเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ณ จุฬาลงกรณ์มหาวิทยาลัย และสำเร็จการศึกษาในปีการศึกษา 2557

บทความทางวิชาการจากวิทยานิพนธ์

[1] Rattanawadee, P., C. Saivichit, and N. Ruengsakulrach, “IPTV Multicast Routing in SDN/OpenFlow Environments,” in Internet Architecture Workshops, 2014 IEICE, 2014, pp.45-50.

[2] Rattanawadee, P., C. Saivichit, and N. Ruengsakulrach. The Transmission Time Analysis of IPTV Multicast Service in SDN/OpenFlow Environments. in Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2015 12th International Conference on, 2015.