กระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทาง ในเครือข่ายที่กำหนดโดยซอฟต์แวร์

นายจิรศักดิ์ จุลวัจน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2560

Path Selection Algorithm for Multipath TCP in Software-Defined Networking

Mr. Jirasak Jullawat

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2017

หัวข้อวิทยานิพนธ์	กระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทาง ในเครือข่ายที่กำหนดโดยซอฟต์แวร์
โดย	นายจิรศักดิ์ จุลวัจน์
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.กุลธิดา โรจน์วิบูลย์ชัย

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(รองศาสตราจารย์ ดร.กุลธิดา โรจน์วิบูลย์ชัย)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร.อภิรักษ์ จันทร์สร้าง)

จิรศักดิ์ จุลวัจน์ : กระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทาง ในเครือข่ายที่กำหนดโดยซอฟต์แวร์ (Path Selection Algorithm for Multipath TCP in Software-Defined Networking) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร.กุลธิดา โรจน์วิบูลย์ชัย, 47 หน้า.

Multipath TCP (MPTCP) มีความสามารถในการเพิ่ม Throughput และเป็น Load balance ซึ่งทำให้คงทนต่อความเสียหาย (Fault Tolerance) จากเส้นทางเชื่อมต่อที่มีปัญหา แต่ด้วยเครือข่ายที่มีความซับซ้อนในปัจจุบัน ทำให้การที่ส่งข้อมูลโดยใช้ MPTCP เกิดการใช้เส้นทางที่ทับซ้อนกัน อย่างไรก็ตาม ในการส่งข้อมูลภายใต้เครือข่ายที่กำหนดโดยซอฟต์แวร์ (Software-Defined Network) ที่ทราบถึงลักษณะของข้อมูลที่ถูกส่งภายในระบบ และสามารถเขียนโปรแกรมควบคุมการทำงานได้ทั้งระบบนั้น จะสามารถช่วยเลือกเส้นทางที่เหมาะสมได้ งานวิจัยนี้จึงเสนอกระบวนการระบุ Subflow ที่ถูกต้องแม่นยำและกระบวนการเลือกเส้นทางที่ใช้ข้อมูลแบนด์วิดท์ที่พร้อมใช้งานแบบเรียลไทม์และค่าความหน่วง เพื่อประมาณค่า Throughput ในการเลือกคู่เส้นทางที่จะได้ค่า Throughput สูงสุดจากเส้นทางทั้งหมดในเครือข่าย ผลลัพธ์จากการวัดประสิทธิภาพในการทดลองนี้พบว่าสามารถส่งข้อมูลที่ได้ค่า Throughput ที่สูงกว่าการเลือกเส้นทางแบบที่สั้นที่สุดร้อยละ 76 ในเครือข่ายแบบดัมเบลอย่างง่าย และร้อยละ 66.6 ในเครือข่าย COST 239

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2560

5870914221 : MAJOR COMPUTER SCIENCE
KEYWORDS: MULTIPATH-TCP / OPENFLOW / SOFTWARE-DEFINED NETWORKING / BANDWIDTH / DELAY

JIRASAK JULLAWAT: Path Selection Algorithm for Multipath TCP in Software-Defined Networking. ADVISOR: ASSC. PROF. KULTIDA ROJVIBOONCHAI, Ph.D., 47 pp.

The features of Multipath TCP (MPTCP) are throughput improvement, load balancing and fault tolerance when used link is down. However, the current network is complicated. Therefore, data transmission from the same source and destination usually shares the same path. However, if we use MPTCP in Software-Defined Networking in which the characteristics of packets and paths can be known at the controller, we can use those information to perform the path selection. This study proposed the accurate subflow identification algorithm and path selection algorithm which utilizes information of real-time traffic and delay for estimating throughput. The performance evaluation results show that, comparing to the traditional shortest path selection algorithm, the proposed path selection algorithm achieves 76% throughput improvement in the simple dumbbell topology, and 66.6% throughput improvement in COST 239 topology.

Department: Computer Engineering Student's Signature .....
Field of Study: Computer Science Advisor's Signature .....
Academic Year: 2017

กิตติกรรมประกาศ

ตลอดการทำวิทยานิพนธ์ฉบับนี้ได้ผ่านอุปสรรคต่าง ๆ ที่เกิดขึ้นนานัปการ ซึ่งเป็นบทเรียนและประสบการณ์อันมีค่าแก่ผู้จัดทำ ซึ่งสามารถได้เรียนรู้ ฝึกฝนและเพิ่มพูนทักษะในการทำการวิจัย รวมถึงการแก้ปัญหาต่าง ๆ วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความสนับสนุนจากบุคคลหลายฝ่าย ซึ่งผู้จัดทำได้ซาบซึ้งในความกรุณาอย่างล้นพ้น

งานวิจัยนี้ได้รับการสนับสนุนจากหน่วยปฏิบัติการวิจัยโครงข่ายไร้สาย และอินเตอร์เน็ตอนาคต (Wireless Network and Future Internet Research Unit) กองทุนรัชดาภิเษกสมโภช จุฬาลงกรณ์มหาวิทยาลัย

ขอขอบพระคุณอาจารย์ที่ปรึกษาวิทยานิพนธ์ รองศาสตราจารย์ ดร.กุลธิดา โรจน์วิบูลย์ชัย ที่ช่วยแนะนำ พร้อมทั้งให้ข้อคิดในการทำการวิจัย

ขอขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ ผู้ช่วยศาสตราจารย์ ดร.ณัฐวุฒิ หนูไพโรจน์ ผู้ช่วยศาสตราจารย์ ดร.เกริก ภิรมย์โสภา และผู้ช่วยศาสตราจารย์ ดร.อภิรักษ์ จันทร์สร้าง ที่ช่วยให้คำวิจารณ์และข้อเสนอแนะเพื่อนำไปพัฒนางานวิจัยให้ดีขึ้น

ขอขอบพระคุณนายเกียรติคุณ กาวิละ และเพื่อน ๆ พี่ๆ น้อง ที่สนับสนุนและให้ความช่วยเหลือเรื่องต่าง ๆ

สารบัญ

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

ในปัจจุบันนี้ คอมพิวเตอร์ มือถือ แท็บเล็ต รวมไปถึงเครื่องเซิร์ฟเวอร์มักจะมีส่วนต่อขยาย (Network Interface) จำนวน 2 ส่วนต่อขยายหรือมากกว่านั้น อุปกรณ์เครือข่ายและโครงสร้างพื้นฐานของระบบเครือข่ายสามารถรองรับการส่งข้อมูลพร้อมกันหลายเส้นทางจากอุปกรณ์หนึ่งไปยังอุปกรณ์หนึ่งได้ การส่งข้อมูลโดยมากจะใช้โพรโทคอลที่มีชื่อว่า Transmission Control Protocol (TCP)[1] แต่อย่างไรก็ตาม TCP สามารถทำได้เพียงแค่ส่งข้อมูลในเส้นทางเดียวเท่านั้น ดังนั้น TCP ในปัจจุบันจึงไม่ได้ใช้ประโยชน์จากการที่มีเส้นทางการส่งข้อมูลที่มากขึ้น และด้วยเหตุนี้จึงมีผู้พัฒนา Multipath TCP ขึ้น

Multipath TCP (MPTCP) [2] คือ ส่วนต่อขยาย (Extension) ของ TCP ปกติ ที่จัดการควบคุมการส่งข้อมูลในหลายเส้นทาง ใน MPTCP หนึ่งการเชื่อมต่อจะสามารถสร้าง Subflow ย่อยได้ และแต่ละ Subflow สามารถใช้ในการส่งข้อมูลผ่านเส้นทางหรือส่วนต่อขยายที่ต่างกันได้ จะเห็นว่า MPTCP สามารถใช้ประโยชน์ได้มากกว่า TCP หากการเชื่อมต่อของ TCP ถูกขัดขวางและสูญเสียการเชื่อมต่อ เมื่อต้องการเชื่อมต่ออีกครั้งการเชื่อมต่อแบบ TCP ต้องทำกระบวนการ Three-way Handshake ยิ่งไปกว่านั้น หากการเชื่อมต่อถูกขัดขวางในขณะที่กำลังส่งข้อมูลที่มีขนาดใหญ่ภายหลังการเชื่อมต่ออีกครั้งข้อมูลนี้อาจจะต้องเริ่มต้นส่งใหม่อีกครั้งหนึ่ง ดังนั้น MPTCP สามารถช่วยแก้ปัญหานี้ได้ โดยใช้เส้นทางอื่นที่ยังสามารถใช้งานได้และยังเชื่อมต่ออยู่ ภายหลังหากเส้นทางที่ถูกขัดขวางได้รับการแก้ไขแล้ว จะสามารถส่งข้อมูลในเส้นทางนี้ได้อีกครั้ง ในปัจจุบัน MPTCP ยังไม่ได้ถูกใช้อย่างแพร่หลาย แต่อย่างไรก็ตามระบบปฏิบัติการ iOS ได้มีการใช้ MPTCP และยินยอมให้ iPhone และ iPad สร้างการเชื่อมต่อสำรองผ่านเครือข่ายสัญญาณโทรศัพท์ [3]

ในทางทฤษฎีแล้ว MPTCP มีประสิทธิภาพที่ดี แต่เนื่องจากความซับซ้อนของระบบเครือข่ายและการเลือกเส้นทางแบบสั้นที่สุด (Shortest Path) ทำให้การเชื่อมต่อจากเครื่องต้นทางไปยังเครื่องปลายทางด้วยโพรโทคอล MPTCP อาจจะมีการใช้เส้นทางร่วมกันและเกิดคอขวด (Bottleneck) ในเครือข่ายขึ้นได้ แต่หากใช้ MPTCP ภายใต้เครือข่ายที่กำหนดโดยซอฟต์แวร์ ซึ่งทราบคุณลักษณ์แพ็ค

เก็ตของ MPTCP และยอมให้สามารถเขียนโปรแกรมเพื่อใช้ในการเลือกเส้นทาง น่าจะสามารถใช้เลือกเส้นทางที่เหมาะสมให้กับ MPTCP ได้

งานวิจัยนี้มีวัตถุประสงค์เพื่อพัฒนากระบวนการเลือกเส้นทางของ MPTCP ภายใต้เครือข่ายที่กำหนดโดยซอฟต์แวร์ หรือ Software-Defined Networking (SDN) ซึ่งเป็นเครือข่ายแห่งอนาคต [4] โดยใช้อุปกรณ์ควบคุมหรือ Controller เป็นส่วนควบคุมและทำงานผ่านโพรโทคอล OpenFlow [5] กระบวนการทำงานของ Controller จะศึกษาคุณลักษณ์ของแพ็คเกต MPTCP Header และจะจำแนกแต่ละการเชื่อมต่อของ MPTCP นอกจากนี้แล้วในการเลือกเส้นทางของแต่ละ Subflow จะใช้ข้อมูลจากค่าแบนด์วิดท์ที่พร้อมใช้งาน (Available Bandwidth) และความหน่วง (Delay) ระหว่างต้นทางและปลายทางสำหรับการเลือกเส้นทางที่ดีที่สุด งานวิจัยนี้จะทำการศึกษาบน Mininet Network Emulator [6] และ Floodlight Controller [7] ทำการทดลองบนโครงสร้างเครือข่ายที่มีการส่งข้อมูลแบบ UDP [8] ผสมด้วย (Background UDP Traffic) และได้ทำการวัดผลจากค่า Throughput โดยใช้โปรแกรม Iperf [9]

1.2 วัตถุประสงค์ของงานวิจัย

วัตถุประสงค์ของงานวิจัยชิ้นนี้สามารถแบ่งออกเป็น 3 ส่วนหลัก ๆ ดังนี้

1. เพื่อพัฒนากระบวนการเลือกเส้นทางของ MPTCP ภายใต้เครือข่ายที่กำหนดโดยซอฟต์แวร์
2. เพื่อศึกษาโครงสร้าง และหลักการทำงานของ MPTCP
3. เพื่อศึกษากระบวนการเขียนโปรแกรม และส่วนประกอบของ SDN

1.3 ขอบเขตของการวิจัย

งานวิจัยนี้ได้ดำเนินงานภายใต้ขอบเขต 3 ประการดังต่อไปนี้

1. MPTCP ในการทดลองใช้ MPTCP Linux Kernel implementation [10]
2. เป็นการทดลองที่กำหนดจำนวน Subflow ต่อหนึ่งการเชื่อมต่อที่ 2 Subflows เนื่องจากเครื่องผู้ส่งที่เป็นผู้กำหนดจำนวนของ Subflow ไม่ทราบจำนวน Subflow ที่ควรสร้าง และการส่งจำนวน Subflow ที่มากเกินไปจะเป็น Overhead ให้กับเครือข่าย

โดยไม่เกิดผลประโยชน์ และมาจากเป็นกรณีที่สามารถเกิดขึ้นได้เยอะในปัจจุบัน ดังสังเกตจากคอมพิวเตอร์ มือถือ แท็บเล็ต รวมไปถึงเครื่องเซิร์ฟเวอร์มักจะมีส่วนต่อขยายอย่างต่ำจำนวน 2 ส่วนต่อขยาย และเครือข่ายที่นิยมทำ Redundant เพื่อเพิ่มเสถียรภาพ

3. งานวิจัยนี้ทดสอบโดยการใช้ระบบเครือข่ายจำลองเท่านั้นโดยยังไม่ได้ทดสอบในระบบเครือข่ายจริง

1.4 ประโยชน์ที่ได้รับ

ประโยชน์ที่ได้รับจากงานวิจัยชิ้นนี้สามารถแบ่งออกเป็น 2 ส่วนหลัก ๆ ดังนี้

1. ทำให้ได้มาซึ่งกระบวนการเลือกเส้นทางของ TCP แบบหลายเส้นทางในเครือข่ายที่กำหนดโดยซอฟต์แวร์
2. ผลการทดลองและการเปรียบเทียบประสิทธิภาพจะเป็นประโยชน์ในการวิจัยและพัฒนาต่อไปได้

1.5 วิธีดำเนินการวิจัย

1. รวบรวมความรู้ที่เกี่ยวข้อง
   - ศึกษาการทำงาน MPTCP, Mininet, SDN ศึกษาเครื่องมือที่ใช้ในงานวิจัย
   - ทดลองติดตั้ง MPTCP และทดสอบการทำงานใน Mininet
   - ศึกษาวิธีเขียนโปรแกรมโดยใช้ Floodlight Controller รวมถึงการใช้งาน Monitoring module
   - ศึกษางานวิจัยที่เกี่ยวข้อง
2. ออกแบบอัลกอริทึม
   - ออกแบบกระบวนการทำงานออกแบบอัลกอริทึม
   - พัฒนาระบบ
3. ทดสอบประสิทธิภาพการทำงานของกระบวนการที่นำเสนอ
4. วิเคราะห์และสรุปผลการทดลอง
5. สรุป เรียบเรียง และจัดทำวิทยานิพนธ์

1.6 ผลงานตีพิมพ์จากการวิจัย

บทความทางวิชาการเรื่อง “**กระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทาง ในเครือข่ายที่กำหนดโดยซอฟต์แวร์ (Path Selection Algorithm for Multipath TCP in Software-Defined Networking)**” โดย จิรศักดิ์ จุลวัจน์ เกียรติคุณ กาวิละ กุลิสร์ ณ นคร และกุลธิดา โรจน์วิบูลย์ชัย ในการประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 13 (The 13th National Conference on Computing and Information Technology) ซึ่งจัดขึ้น ณ โรงแรม อโนมา แกรนด์ กรุงเทพฯ ประเทศไทย ระหว่างวันที่ 6 - 7 กรกฎาคม 2560 (ได้รับรางวัล Best Paper Award)

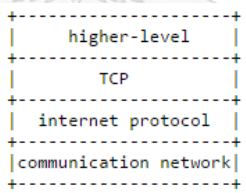
บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

งานวิจัยนี้มีจุดมุ่งหมายเพื่อจะพัฒนาการส่งข้อมูลของ MPTCP โดยมีทฤษฎีที่เกี่ยวข้องดังต่อไปนี้

2.1.1 TCP

Transmission Control Protocol หรือ TCP [1] เป็นโปรโตคอลที่ใช้ในการขนส่งข้อมูลที่ได้รับมาจากระดับชั้นที่สูงกว่า (Higher-level layer) ซึ่งสามารถดูลำดับการเรียงตัวของแต่ละลำดับชั้น (Layer) ได้ในภาพที่ 2.1 โดย TCP นั้นจะมีลักษณะเป็นการกำหนดการเชื่อมต่อ (Connection-Oriented) กล่าวคือก่อนที่จะมีการส่งข้อมูลได้นั้นจะต้องมีกระบวนการสร้างการเชื่อมต่อ (Connection) และเมื่อจะปิดการเชื่อมต่อก็จะต้องมีกระบวนการที่ใช้ในการปิด นอกจากนี้เมื่อ TCP อยู่ในสถานะเชื่อมต่อแล้วยังสามารถส่งข้อมูลระหว่างกันได้สองทางพร้อมกัน (Full duplex) อีกด้วย

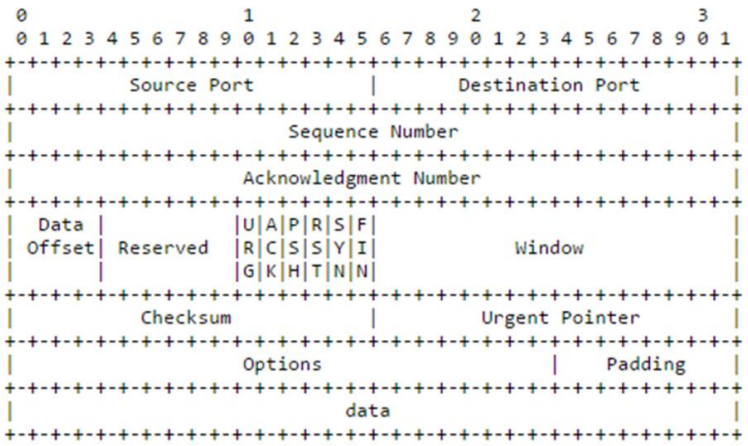


ภาพที่ 2.1 ลำดับชั้นของโปรโตคอล [1]

TCP จะมีการแก้ไขความผิดพลาดของการส่งข้อมูลเช่น เสียหาย สูญหาย มีความซ้ำซ้อน หรือมีการส่งที่ไม่เป็นลำดับหรือที่เรียกว่าความน่าเชื่อถือ (Reliability) ของการส่งข้อมูล ด้วยเหตุนี้ TCP จึงมีการกำหนดลำดับตัวเลข (Sequence number) ไปในแต่ละแพ็คเก็ตที่ส่ง เมื่อผู้ส่งทำการส่งแพ็คเก็ตไปแล้วจะต้องได้รับแพ็คเก็ตการตอบรับหรือแอ็ค (Acknowledgement: ACK) จากผู้รับภายในเวลาที่กำหนด นั่นหมายความว่าแพ็คเก็ตนั้นได้ถูกส่งเรียบร้อยแล้ว ส่วนในฝั่งของผู้รับนั้นลำดับตัวเลขจะถูกใช้ในการเรียงลำดับแพ็คเก็ตให้ถูกต้อง หากผู้ส่งไม่ได้รับแอ็คในเวลาที่กำหนดจะมี

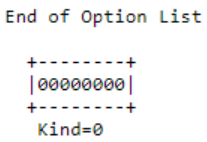
กระบวนการส่งข้อมูลนั้นใหม่อีกครั้ง นอกจากนี้ยังมีกระบวนการทำเช็คซัม (Checksum) ข้อมูลเพื่อเป็นการตรวจสอบความถูกต้องของข้อมูลอีกด้วย

ในการส่งข้อมูลของ TCP จะถูกส่งข้อมูลผ่าน Internet Protocol หรือ IP ซึ่ง Internet Protocol Header จะประกอบไปด้วยข้อมูลที่จำเป็นเช่น IP Address ของเครื่องต้นทางและปลายทาง [11] ในส่วนของ TCP header จะมีโครงสร้างตาม Internet Protocol Header โดยจะมีข้อมูลที่เฉพาะเจาะจงกับ TCP Protocol ดังแสดงตามภาพที่ 2.2



ภาพที่ 2.2 โครงสร้างของ TCP Header [1]

ภายในส่วนของ TCP header จะมีในส่วนของ Options ซึ่งสามารถประกอบได้หลาย TCP option เมื่อหมดในส่วนของ Options จะปิดท้ายด้วย Padding ซึ่งมีโครงสร้างดังภาพที่ 2.3

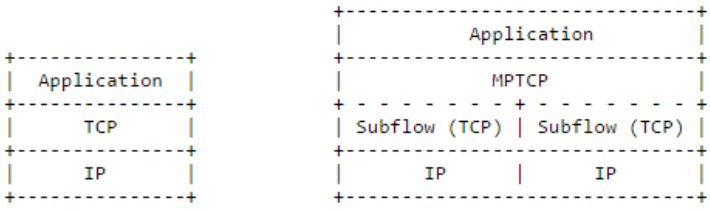


ภาพที่ 2.3 โครงสร้าง Padding [1]

2.1.2 MPTCP

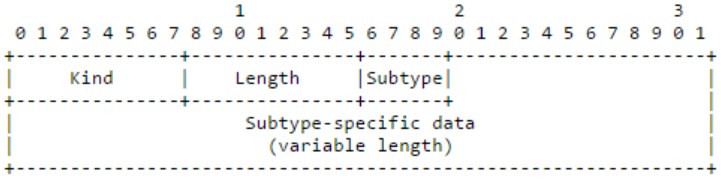
Multipath TCP หรือ MPTCP [2] คือส่วนเพิ่มต่อขยาย (Extension) ของ TCP [1] ที่รองรับการส่งข้อมูลแบบหลายเส้นทาง เมื่อเปิดการเชื่อมต่อในการส่งข้อมูลแล้วจะสามารถส่งข้อมูลไปทุก ๆ เส้นทางได้พร้อม ๆ กัน ซึ่งจะสามารถเพิ่มปริมาณข้อมูลที่ส่งได้ในหนึ่งช่วงเวลา (Throughput) และเป็นการลดความเสียหายในกรณีที่เกิดความขัดข้องของเครือข่าย

MPTCP เป็นโปรโตคอลทำงานอยู่ในชั้นทรานสปอร์ต (Transport layer) เช่นเดียวกับ TCP ซึ่งจะเป็นตัวกลางและผสานการทำงานระหว่างชั้นแอพพลิเคชั่น (Application layer) และชั้นของ Internet Protocol โดย MPTCP จะทำการแบ่ง TCP ปกติออกเป็นหลาย TCP หรือที่เรียกว่า Subflow ดังแสดงในภาพที่ 2.4 ด้วยเหตุนี้ MPTCP จึงสามารถใช้เส้นทางได้หลายเส้นทางภายใต้การเชื่อมต่อเดียว (Single connection) และยังสามารถใช้งานได้กับแอพพลิเคชั่นเดิมโดยไม่ต้องทำการเปลี่ยนแปลง



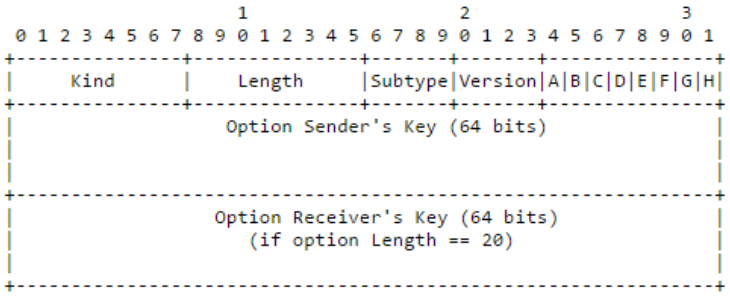
ภาพที่ 2.4 ลำดับชั้นของโปรโตคอลที่เปรียบเทียบระหว่าง TCP และ MPTCP [2]

เมื่อตัดแบ่ง TCP Header มาเฉพาะส่วนของ Option ที่เป็นส่วนของ MPTCP จะมีลักษณะดังภาพที่ 2.5 โดยส่วนของ Option นี้จะมีลักษณะที่แตกต่างกันแล้วแต่ Subtype ของ MPTCP



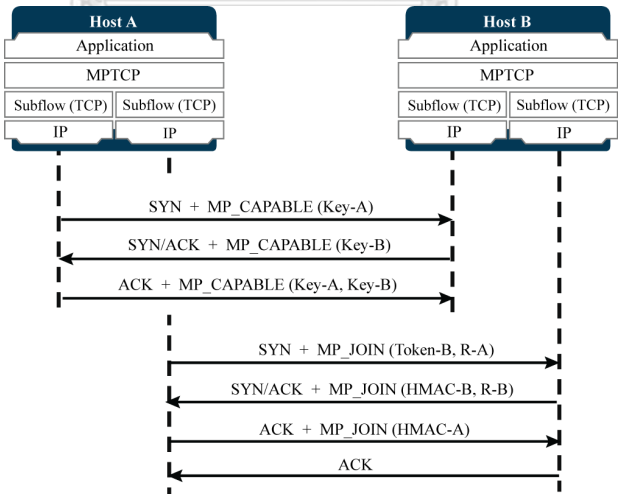
ภาพที่ 2.5 แสดงรูปแบบของ MPTCP Option [2]

และภาพที่ 2.6 แสดงตัวอย่างของ Option Header ที่เป็นแบบ MP_CAPABLE ที่ไว้สร้างการเชื่อมต่อ



ภาพที่ 2.6 ตัวอย่างรูปแบบของ MPTCP Option [2]

ตามภาพที่ 2.7 เมื่อเริ่มการเชื่อมต่อ MPTCP จะใช้กระบวนการ Three-way Handshake เหมือน TCP ปกติ ทั้งกระบวนการเริ่มต้น Flow หลัก และ Subflow รอง โดยที่ส่วนของ TCP Option จะมีการเพิ่มส่วนของ MP_CAPABLE ใน Flow หลัก MP_JOIN ใน Subflow อื่น ๆ ซึ่งหนึ่งในพารามิเตอร์ที่จำเป็นคือ Token ซึ่ง Token คือ Cryptographic Hash ของค่า Key ที่ถูกสร้างขึ้นและส่งไปให้เครื่องอีกฝั่งทราบ โดยงานวิจัยนี้เลือกใช้ค่า Token ที่เกิดจาก Key ของฝั่ง MPTCP Server (Host B) จุดเด่นของ Token คือในหนึ่งการเชื่อมต่อค่า Token จะเป็นค่าเดียวกันเสมอ ดังนั้นเราจึงใช้ประโยชน์ตรงจุดนี้เพื่อมาใช้ในการแบ่งแยก Subflow ที่อยู่ในการเชื่อมต่อ MPTCP เดียวกันได้



ภาพที่ 2.7 โครงสร้างข้อมูลที่ส่งผ่านโปรโตคอล MPTCP

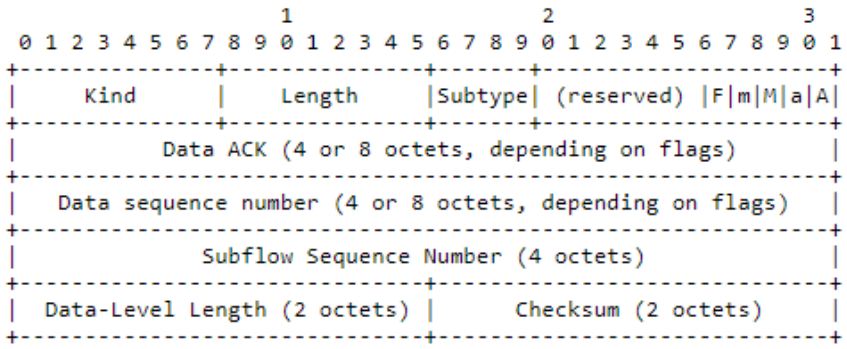
รายละเอียดของการคำนวณค่า Token ซึ่งได้กำหนดไว้ใน [2] ว่า Token จะถูกสร้างโดย SHA-1 [12] อัลกอริทึม และเลือกใช้แค่ 32 bits บนของผลลัพธ์โดยสามารถเขียนได้ดังสมการที่ (1)

$$Token_{Sender} = Upper_{32}(SHA\text{-}1(Key_{Sender})) \quad (1)$$

การจัดลำดับในการส่งข้อมูล (Packet Scheduling) เพื่อเป็นการเลือกว่าจะส่งข้อมูลที่ได้รับมาจากแอปพลิเคชันไปยัง Subflow ใดและเมื่อไร ซึ่งการจัดลำดับในการส่งข้อมูลของ MPTCP ในเวอร์ชัน 9.1.x จะมีด้วยกัน 3 แบบคือ

- แบบค่าเริ่มต้น (default) ซึ่งการส่งข้อมูลแบบนี้จะทำการส่งข้อมูลไปใน Subflow ที่มีค่าเวลาที่ใช้ในการส่งข้อมูลไปยังเครื่องปลายทาง จนกระทั่งได้ข้อมูลตอบกลับหรือ Round trip time (RTT) มีค่าน้อยที่สุดก่อน โดยจะส่งไปจนเต็ม Congestion-window หลังจากนั้นจะไปส่งใน Subflow ที่มีค่า RTT น้อยที่สุดเป็นอันดับถัดไป
- การส่งแบบเวียนเทียนหรือ Round-Robin จะทำการส่งวนไปในทุก Subflow ที่ได้ทำการสร้างขึ้น โดยในแต่ละรอบ จะส่งไปตามจำนวน segment ที่ถูกตั้งค่าไว้ ซึ่งค่าจำนวน segment เริ่มต้นจะมีค่าเท่ากับ 1
- การส่งแบบทำซ้ำซ้อนหรือ Redundant ที่จะพยายามส่งข้อมูลไปในทุก Subflow ซึ่งวิธีนี้จะเหมาะกับแอปพลิเคชันที่ต้องการให้เวลาที่ข้อมูลเดินทาง (latency) น้อยที่สุด แต่วิธีนี้จะต้องแลกมาด้วยการเสียค่าแบนด์วิดท์ในการส่งข้อมูลซ้ำซ้อน

การจัดเรียงลำดับในการรับข้อมูล (Data Sequence Mapping) ข้อมูลที่ถูกส่งมาแต่ละ Subflow จะสามารถจัดเรียงลำดับให้ถูกต้องได้โดยใช้ข้อมูลที่ถูกเก็บไว้ใน Data Sequence Signal (DSS) Option ซึ่งแสดงโครงสร้างตามภาพที่ 2.8

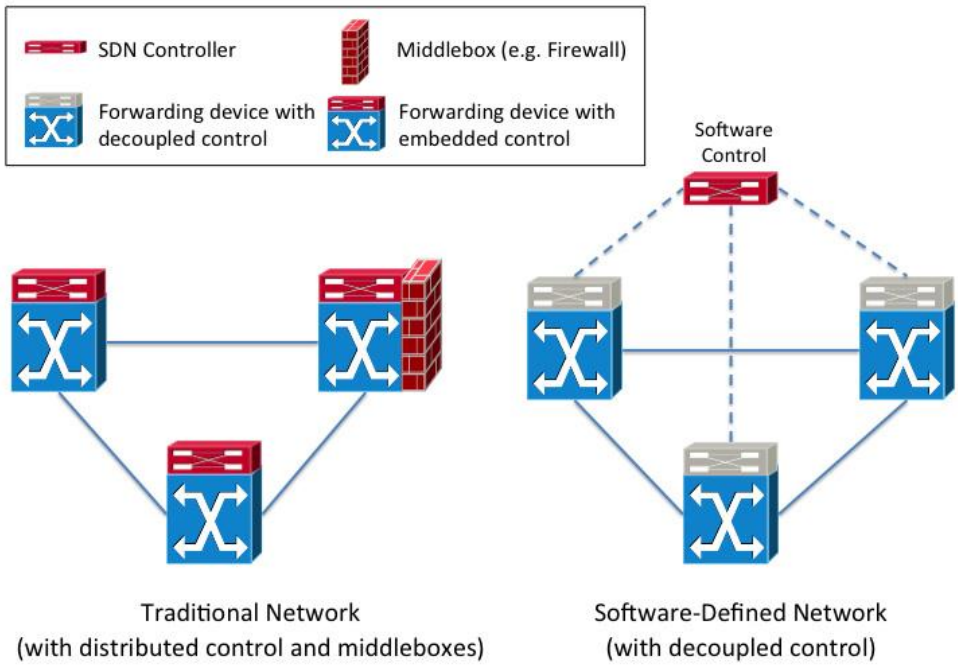


ภาพที่ 2.8 โครงสร้างของ Data Sequence Signal (DSS) Option [2]

ภายใน DSS Option จะประกอบด้วย Data sequence number ซึ่งเป็นเลขลำดับของข้อมูลต้นฉบับก่อนที่จะถูกแบ่งส่งไปในแต่ละ Subflow

2.1.3 เครือข่ายที่กำหนดโดยซอฟต์แวร์

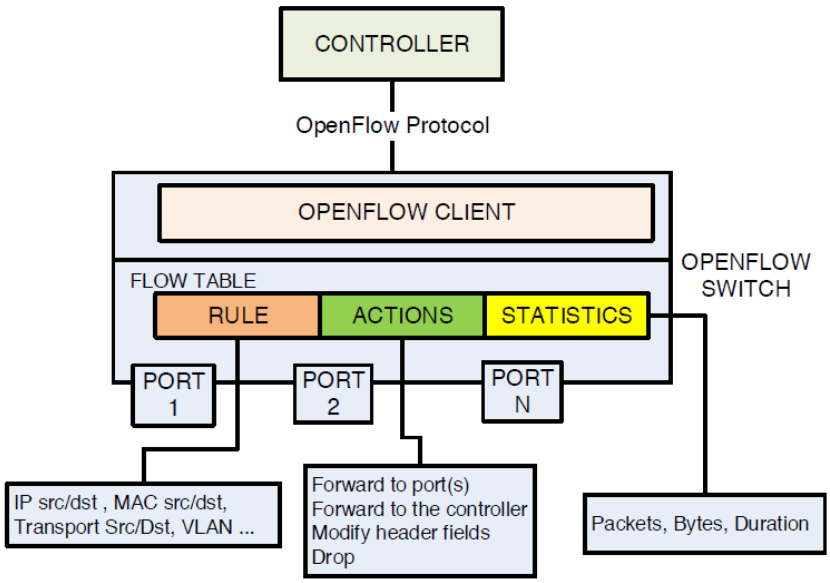
เครือข่ายที่กำหนดโดยซอฟต์แวร์ (Software-Defined Networking) หรือ SDN เป็นเครือข่ายแห่งอนาคต [4] โดยใช้อุปกรณ์ควบคุมหรือ Controller เป็นส่วนควบคุมและทำงานของอุปกรณ์ในระบบเครือข่ายซึ่งแสดงในภาพที่ 2.9 โดยรูปซ้ายมือเป็นส่วนของโครงสร้างเครือข่ายแบบเดิมที่มีส่วนควบคุมอยู่ภายในอุปกรณ์แต่ละตัว ส่วนรูปทางขวามือเป็นโครงสร้างตามแบบ SDN ซึ่งจะรวมส่วนควบคุมไว้ภายนอกแล้วสามารถควบคุมทุกอุปกรณ์ได้



ภาพที่ 2.9 โครงสร้างของ SDN เปรียบเทียบกับเครือข่ายแบบเดิม [4]

OpenFlow Protocol ถูกออกแบบมาจากหลักการของ SDN ซึ่ง OpenFlow ที่แยกส่วนการส่งต่อแพ็คเก็ต (Data Plane) และส่วนการตัดสินใจ (Control Plane) ที่เกิดขึ้นใน OpenFlow Switch ตัวเดียวกัน โดย Control Plane นี้จะถูกควบคุมด้วย Controller เมื่อมี Flow ของข้อมูลใหม่เข้ามาที่ OpenFlow Switch จากนั้น Switch จะส่งข้อมูลนั้น (PACKET_IN) ไปหา Controller หลังจากนั้น Controller จะจัดการกับ PACKET_IN โดยจะตรวจสอบข้อมูล Header ของแพ็คเก็ต

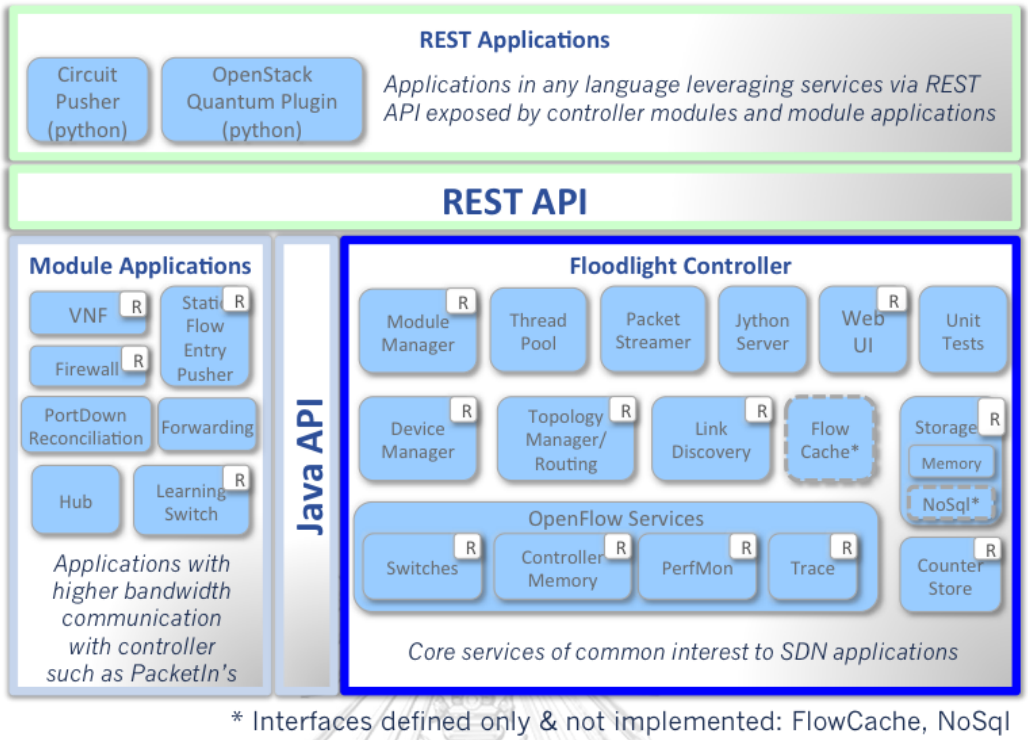
และสั่งให้ OpenFlow Switch สร้าง Rule ตามที่กำหนด โดยภาพที่ 2.10 จะแสดงโครงสร้างการทำงานของ OpenFlow Protocol



ภาพที่ 2.10 การสื่อสารระหว่าง Controller และ switch ผ่าน OpenFlow Protocol [4]

2.1.4 Floodlight Controller

ปัจจุบันมี Controller ให้เลือกใช้งานอยู่หลากหลาย ซึ่งในงานวิจัยนี้เลือกใช้ Controller ที่มีชื่อว่า Floodlight [7] ซึ่งเป็น Controller ที่สร้างขึ้นมาจากภาษา Java ภาพที่ 2.11 แสดง Architecture Diagram ของ Floodlight Controller การเขียนโปรแกรมควบคุมการทำงานของ Controller ทำได้สองวิธีคือ เขียน Java Module และ REST API โดยงานวิจัยนี้ต้องเขียนโปรแกรมด้วย Java Module เพื่อจัดการกับ PACKET_IN



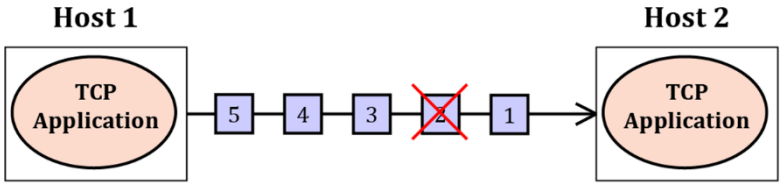
ภาพที่ 2.11 Architecture Diagram ของ Floodlight Controller [7]

ส่วนประกอบหลักที่จำเป็นต้องใช้ใน Controller ได้แก่

- Topology Manager ซึ่งจะเก็บภาพรวม (Global View) ของ Topology จากการส่งข้อมูลระหว่าง SDN Controller และ Switches โดยเราสามารถดึงข้อมูลได้ว่ามีเส้นทางไหนบ้างที่เชื่อมต่อ Node ที่เราต้องการส่งข้อมูลถึงกัน
- Monitoring Module ซึ่งสามารถดึงข้อมูลสถิติมาใช้งานได้
- Forwarding Module ซึ่งเป็นส่วนหลักในการรับค่า PACKET_IN มาวิเคราะห์ โดยดึงข้อมูลจาก Module อื่นๆ และทำการเพิ่ม OpenFlow Rules ใน OpenFlow Switch

2.1.5 เฮดออฟลายบล็อกกิ้ง (Head-of-line blocking)

ปัญหาเฮดออฟลายบล็อกกิ้ง (Head-of-line blocking) เป็นปัญหาที่เกิดจากบางส่วนของข้อมูลที่ถูกส่งผ่านระบบเครือข่ายสูญหายไปทำให้ผู้รับไม่สามารถนำเอาข้อมูลที่ได้รับมาไปใช้ได้



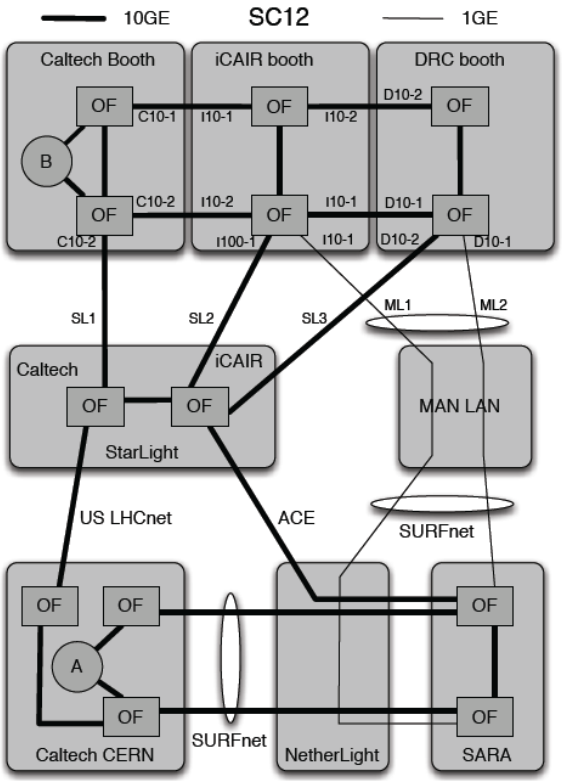
ภาพที่ 2.12 ปัญหาเฮดออฟลายบล็อกกิ้ง

การทำงานของกระบวนการจัดการการส่งข้อมูลที่แบ่งข้อมูลออกมาส่งหลายเส้นทาง โดยแต่ละเส้นทางมีการหน่วงที่ไม่เท่ากัน ทำให้ข้อมูลที่ถูกส่งไปถึงผู้รับปลายทางไม่เป็นตามลำดับที่ถูกต้อง ยกตัวอย่างเช่น ตามภาพที่ 2.12 สมมติว่าแพ็คเก็ตที่ 2 ถูกส่งมาใน Subflow ที่มีการหน่วงที่สูงจะทำให้แพ็คเก็ตที่ถูกส่งเป็นลำดับถัดมาจะยังไม่สามารถนำไปใช้งานได้

โดยงานวิจัยนี้ได้เสนอวิธีที่จะพิจารณาเลือกความหน่วงของเส้นทางเพื่อลดการเกิดปัญหาเฮดออฟลายบล็อกกิ้ง

2.2 งานวิจัยที่เกี่ยวข้อง

หลังจากเครือข่าย SDN เริ่มได้รับความสนใจ และ Paasch และคณะ [10] ได้พัฒนา MPTCP ใน Linux Kernel ขึ้น หลังจากนั้น Pol และคณะ [13] ได้ใช้เครือข่าย SDN ที่เป็นเครือข่าย SDN ระดับนานาชาติในการจัดการส่งข้อมูล MPTCP โดยได้ความร่วมมือจากหลายหน่วยงานได้แก่ SARA, Caltech, SURFnet, StarLight, และ iCAIR และภาพที่ 2.13 แสดง Topology ที่ใช้ในการทำวิจัย



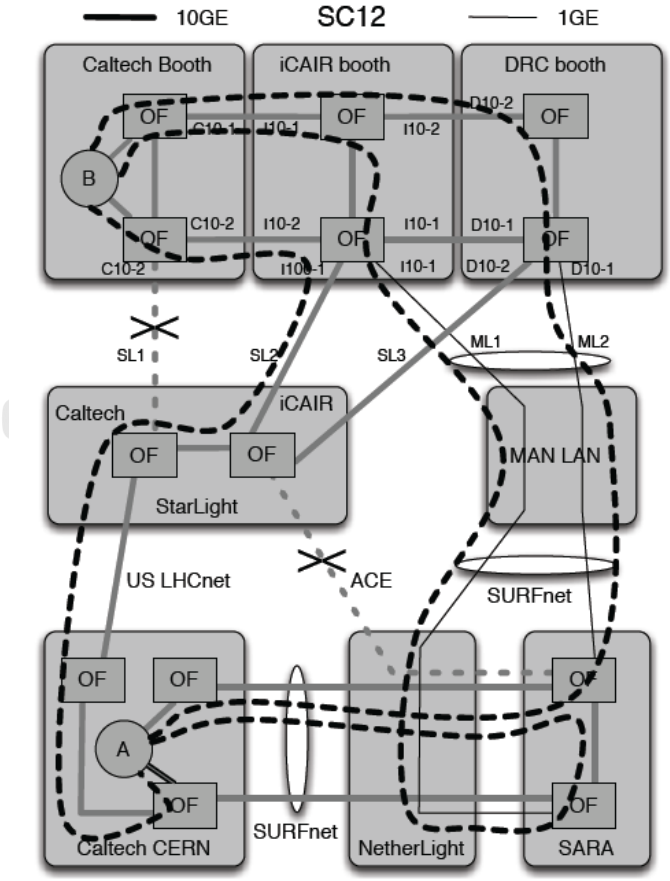
ภาพที่ 2.13 โครงสร้างเครือข่ายในการทำวิจัย [13]

โดยทำการกำหนดค่า VLAN สำหรับแต่ละส่วนต่อขยายของทั้งเครื่องต้นทางและเครื่องปลายทางก่อนที่จะส่งข้อมูลตามภาพที่ 2.14 โดยจำนวนของ VLAN จะขึ้นกับจำนวน Subflow ของ MPTCP ซึ่ง Controller สามารถระบุ Subflow ได้จากหมายเลข VLAN ที่ถูกกำหนดไว้ใน Header ของแพ็คเก็ต

	server B	
	eth1	eth2
server A eth1	500, 501	502, 503
server A eth2	504, 505	506, 507

ภาพที่ 2.14 ค่า VLAN ที่กำหนดให้กับ Server Interfaces [13]

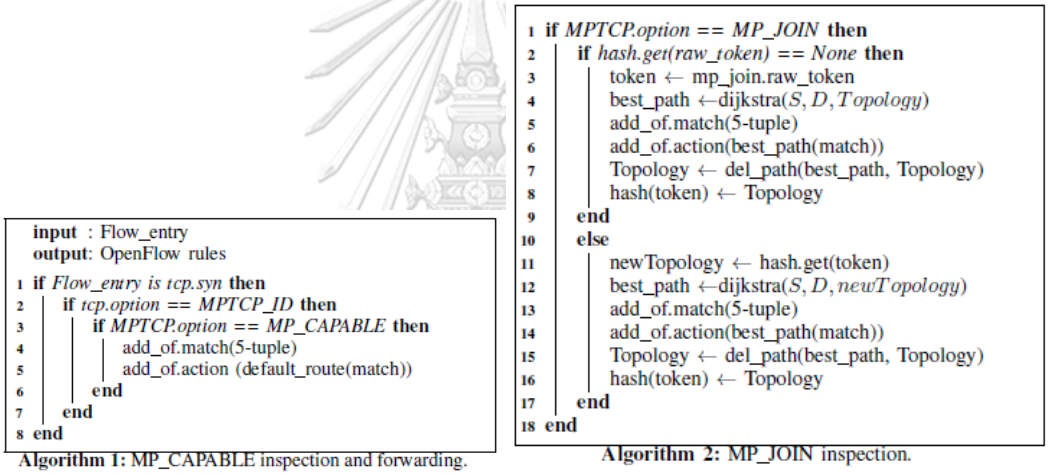
การกำหนดเส้นทางจะเลือกจากเส้นทางที่มีความสามารถส่งข้อมูลได้เยอะที่สุดตลอดเส้นทางระหว่างเครื่องต้นทางและเครื่องปลายทาง จากงานวิจัยนี้จะได้ 3 เส้นทางที่เป็นเส้นประดังแสดงตามภาพที่ 2.15 ซึ่งกระบวนการนี้ช่วยเพิ่ม Throughput ในการส่งข้อมูล อย่างไรก็ตามข้อจำกัดของกระบวนการนี้คือต้องทำการกำหนด VLAN และเส้นทางระหว่างเครื่องต้นทาง (A) และเครื่องปลาย (B) ทางก่อนจึงจะสามารถใช้งานได้



ภาพที่ 2.15 เส้นทางที่ใช้ในการส่งข้อมูลได้ [13]

ดังนั้นเพื่อลดความยุ่งยากหรือซับซ้อนในการตั้งค่า VLAN การศึกษาของ Sandri และคณะ [14] และ Zannettou และคณะ [15] ได้ศึกษาลักษณะของแพ็คเก็ต MPTCP Header โดยพบว่าสามารถแบ่งแยก Subflow แต่ละ Subflow ได้ และช่วยให้ Controller เลือกจัดการกับแพ็คเก็ตแต่ละ Subflow ซึ่งสามารถเพิ่ม Throughput ให้ข้อมูลที่ส่งอันเนื่องมาจากการเลือกส่งแพ็คเก็ตไปคนละเส้นทาง

ภาพที่ 2.16 แสดงให้เห็นวิธีการที่ Sandri และคณะ [14] ใช้ในการจำแนกข้อมูลแต่ละ Subflow โดยขั้นตอนแรกจะทำการแยกแพ็คเก็ตที่เข้ามาออกเป็น MP_CAPABLE และ MP_JOIN อันเนื่องมาจากเพียงแค่ MP_JOIN ที่จะสามารถใช้ค่า Token ในการระบบการเชื่อมต่อได้ ดังนั้นเมื่อมีหลายการเชื่อมต่อ MPTCP เข้ามาจาก host เดียวกัน ทุกแพ็คเก็ตที่เป็น MP_CAPABLE จะถูกส่งไปในเส้นทางเดียวกัน



```
input : Flow_entry
output: OpenFlow rules
if Flow_entry is tcp.syn then
    if tcp.option == MPTCP_ID then
        if MPTCP.option == MP_CAPABLE then
            add_of.match(5-tuple)
            add_of.action (default_route(match))
        end
    end
end
```

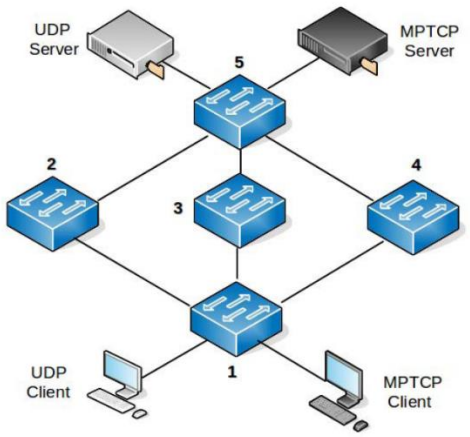
Algorithm 1: MP_CAPABLE inspection and forwarding.

```
if MPTCP.option == MP_JOIN then
    if hash.get(raw_token) == None then
        token ← mp_join.raw_token
        best_path ←dijkstra(S, D, Topology)
        add_of.match(5-tuple)
        add_of.action(best_path(match))
        Topology ← del_path(best_path, Topology)
        hash(token) ← Topology
    end
    else
        newTopology ← hash.get(token)
        best_path ←dijkstra(S, D, newTopology)
        add_of.match(5-tuple)
        add_of.action(best_path(match))
        Topology ← del_path(best_path, Topology)
        hash(token) ← Topology
    end
end
```

Algorithm 2: MP_JOIN inspection.

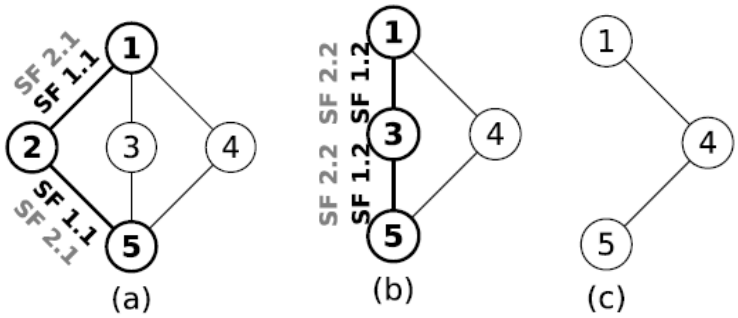
ภาพที่ 2.16 อัลกอริทึมในการแยก Subflow [14]

ในการทำการทดลองได้ออกแบบโครงสร้างเครือข่ายตามภาพที่ 2.17 ซึ่งประกอบด้วย 3 เส้นทางที่เป็นไปได้จากการเชื่อมต่อของ OpenFlow Switch จำนวน 5 ตัว มี UDP traffic ใช้เส้นทางที่ 1-3-5 ด้วย



ภาพที่ 2.17 โครงสร้างเครือข่ายที่ใช้ในการทำวิจัย [14]

การทำงานในส่วนการเลือกเส้นทางที่ส่ง จะอธิบายเป็นตัวอย่างตามภาพที่ 2.18 ตัวอย่างจะส่ง MPTCP จำนวน 2 การเชื่อมต่อคือ MPTCP1 (SF 1.1, SF 1.2) และ MPTCP2 (SF2.1 SF 2.2) แต่ละการเชื่อมต่อจะมี 2 Subflows โดยใช้ชื่อว่า SF 1.1, SF 1.2, SF2.1, และ SF 2.2 ลักษณะ Topology ตัวอย่างจะเป็นดังรูป (a) เมื่อ Subflow ที่ 1.1 เข้ามาในระบบ Controller จะสามารถทราบว่าเป็น Subflow ที่ 1 จาก Header ของแพ็คเก็ตและจะเลือกเส้นทางแบบสั้นที่สุด ซึ่งในที่นี้ได้เส้นทาง 1-2-5 หลังจากนั้นก็ทำการเพิ่ม OpenFlow Rules ให้กับ Switch หลังจากนั้น ก็ทำการลบเส้นทางแบบสั้นที่สุดที่ใช้ไปออกจาก Topology และเก็บ Topology ไว้สำหรับการเชื่อมต่อ MPTCP นี้ รูป Topology ที่เก็บไว้จะเป็นไปตามรูป (b) เมื่อมี Subflow ที่ 2 หรือ 1.2 เข้ามา ก็ทำการหาเส้นทางแบบสั้นที่สุดจาก Topology ที่ได้เก็บไว้ ซึ่งในที่นี้ได้เส้นทาง 1-3-5 หลังจากนั้น ก็ทำการลบเส้นทางแบบสั้นที่สุดที่ใช้ไปออกจาก Topology ที่เก็บไว้อีกครั้ง และเก็บ Topology ที่ถูกลบไว้แทน รูป Topology ที่เก็บไว้จะเป็นไปตามรูป (c) หากมีการเชื่อมต่ออื่น เช่น MPTCP2 ก็ทำเหมือนกันตั้งแต่ต้น



ภาพที่ 2.18 ตัวอย่างการส่งข้อมูล [14]

Zannettou และคณะ [15] ได้มีแนวคิดที่คล้ายกับ Sandri และคณะ [14] โดยการเปลี่ยนจากวิธีการเก็บ Topology แต่ละรูปแบบมาเป็นเก็บเส้นทางที่เคยใช้ไปแล้วแทน และหาเส้นทางใหม่ที่เป็นเส้นทางแบบสั้นที่สุดและไม่ซ้อนทับกับที่เก็บไว้แทน ซึ่งทำให้เหมาะกับ Topology ที่มีขนาดใหญ่กว่า ภาพที่ 2.19 ได้แสดงวิธีการทำงานของงานวิจัยนี้ จะพบว่ากระบวนการแยก Subflow มีการแยก MP_CAPABLE และ MP_JOIN ออกจากกันเหมือนงานวิจัยของ Sandri และคณะ [14]

Algorithm 1 Forwarding Module pseudocode

Input: p is the subflow setup packet received at the controller
1: $IPs \leftarrow$ Extract source/destination IP addresses from p
2: $ports \leftarrow$ Extract source/destination port numbers from p
3: $type \leftarrow$ Extract MPTCP option from p
4: **if** $type == MP\_CAPABLE$ **then**
5: Store IPs in $primaryIPs$
6: Query $pathCache$/TM to find the shortest path for IPs
7: If TM queried, update $pathCache$
8: $path \leftarrow$ shortest path
9: **if** $type == MP\_JOIN$ **then**
10: $token \leftarrow$ Extract MPTCP token from p
11: **if** $token$ does not exist in $flows$ **then**
12: Create a new $entry$ in $flows$ using the $token$
13: Query $pathCache$/TM to get set of paths for IPs
14: If TM queried, update $pathCache$
15: Create a new $IPentry$ in $entry$ using IPs
16: $path \leftarrow$ Get next path from $IPentry$
17: Update $IPentry$
18: **else**
19: $entry \leftarrow flows[token]$
20: **if** IPs exists in $entry$ **then**
21: $IPentry \leftarrow entry[IPs]$
22: $path \leftarrow$ Get next path from $IPentry$
23: Update $IPentry$
24: **else**
25: Query $pathCache$/TM to get set of paths for IPs
26: If TM queried, update $pathCache$
27: Create a new $IPentry$ in $entry$ using IPs
28: $path \leftarrow$ Get next path from $IPentry$
29: Update $IPentry$
30: Install rules for the $path$ to the switches using IPs and $ports$

ภาพที่ 2.19 Algorithm ในการทำงาน [15]

อย่างไรก็ตามงานวิจัยทั้งสองนี้ยังไม่สามารถจำแนก Subflow จากหลายการเชื่อมต่อที่มาจากต้นกำเนิดเดียวกัน และส่งในเวลาใกล้เคียงกันได้ เพราะไม่มีการระบุค่า Token ใน subflow หลัก ซึ่งไม่เหมาะในกรณีที่ไม่ต้องการใช้เส้นทางหลักเป็นเส้นทางเดิม อีกทั้งกระบวนการเลือกเส้นทางยังเป็นแบบการเลือกเส้นทางที่สั้นที่สุดเท่านั้น ซึ่งอาจจะมีเส้นทางอื่นที่มีประสิทธิภาพในการส่งที่ดีกว่า

บทที่ 3
กระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทาง

เพื่อให้ Controller สามารถควบคุมการส่งข้อมูลที่เป็น MPTCP ผ่านเครือข่าย SDN ได้อย่างถูกต้องนั้น ในการศึกษานี้จะแบ่งการทำงานออกเป็น 2 ส่วนย่อย ส่วนแรกเป็นการระบุลำดับของ Subflow ส่วนย่อยที่สองคือ เลือกเส้นทางที่เหมาะสมสำหรับส่งแพ็คเก็ตนั้น โดยงานวิจัยนี้จะนำเสนอกระบวนการทำงานของ Controller เป็น 2 ส่วน ดังนี้

3.1 กระบวนการระบุ Subflow ใน SDN

ในส่วนนี้จะอธิบายกระบวนการระบุ Subflow ใน SDN โดยจะเริ่มจากการอธิบายการทำงานของ MPTCP หลังจากนั้น จะอธิบายวิธีการที่จะสามารถใช้ MPTCP ในเครือข่าย SDN

อัลกอริทึมที่เราเสนอนั้นใช้แนวคิดของ OpenFlow ที่แยกส่วนการส่งต่อแพ็คเก็ต (Data Plane) และส่วนการตัดสินใจ (Control Plane) ซึ่งเกิดขึ้นใน OpenFlow Switch ตัวเดียวกัน โดย Control Plane นี้จะถูกควบคุมด้วย Controller เมื่อมี Flow ของข้อมูลใหม่เข้ามาที่ OpenFlow Switch จากนั้น Switch จะส่งข้อมูลนั้น (PACKET_IN) ไปหา Controller หลังจากนั้น Controller จะจัดการกับ PACKET_IN โดยจะตรวจสอบข้อมูล Header ของแพ็คเก็ต

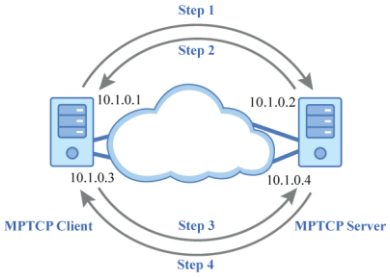
ความคิดพื้นฐานนี้ใช้จำแนกแต่ละ Subflow ของ MPTCP ซึ่ง MPTCP จะประกอบไปด้วยการเชื่อมต่อ TCP ปกติหลายการเชื่อมต่อ การศึกษาชิ้นนี้ได้สร้าง Mapping Table สำหรับเก็บข้อมูล โดยในแต่ละแถวคือแต่ละการเชื่อมต่อ TCP ค่า Token จะเป็นตัวเลขที่ถูกใส่ไว้ในแต่ละแถวและจะถูกใช้เพื่อจำแนกการเชื่อมต่อ TCP ที่มาจากการเชื่อมต่อ MPTCP เดียวกัน นอกจากนี้แต่ละแถวยังมีค่าตัวแปรที่ชื่อว่า Subflow ซึ่งไว้เก็บลำดับของแต่ละ Subflow

ภาพที่ 3.1 จะแสดงกระบวนการทำงานของอัลกอริทึมในแต่ละขั้นตอน และภาพที่ 3.2 แสดงตัวอย่างเมื่อมีการส่งข้อมูล MPTCP จำนวน 2 การเชื่อมต่อมาในเวลาเดียวกัน แต่ละการเชื่อมต่อมี 2 Subflows ในแผนภาพส่วนบนแสดงทิศทางของแพ็คเก็ตแต่ละขั้นตอน และตารางส่วนล่างแสดงถึงการเปลี่ยนแปลงของ Mapping Table ในแต่ละขั้นตอน การทำงานเริ่มจากขั้นตอนที่ 1 เมื่อแพ็คเก็ตแรกที่มาจากเครื่องต้นทาง (MPTCP Client) ไปยังเครื่องปลายทาง (MPTCP Server) เมื่อ Controller ได้รับแพ็คเก็ตใหม่ และตรวจพบว่าเป็น MPTCP (บรรทัดที่ 2) และพบว่าเป็น MP_CAPABLE (บรรทัดที่ 12) และแพ็คเก็ตนี้เป็น TCP SYN (บรรทัดที่ 13) Controller จะค้นหาเส้นทางให้แพ็คเก็ตนี้ (บรรทัดที่ 14) ไว้สำหรับเพิ่มไปใน Flow Table ของ OpenFlow switch หลังจากนั้น Controller จะใส่ต้นทางปลายทางและตั้งค่า Flow เป็น Subflow ที่ 1 ใน Mapping

Table (บรรทัดที่ 15) แต่อย่างไรก็ตามในขั้นตอนนี้จะยังไม่สามารถระบุตัวเลข Token เนื่องจากยังไม่มีข้อมูลของ Token ในแพ็คเก็ตที่เข้ามา โดยจะกลับมาใส่ในขั้นตอนที่ 2 ในส่วนของขั้นตอนที่ 2 3 และ 4 นั้นจะมีส่วนที่คล้ายคลึงกับขั้นตอนที่ 1 ดังแสดงตามภาพที่ 3.1 และภาพที่ 3.2 แต่จะต่างกันตรงกระบวนการระบุค่า Token ซึ่งขั้นตอนที่ 2 จะได้ค่า Token โดยการคำนวณจากสมการที่ (1) (บรรทัดที่ 18) ขั้นตอนที่ 3 จะสามารถได้ค่า Token มาโดยตรงจากแพ็คเก็ต ขั้นตอนที่ 4 ต้องใช้ค่า Token จากขั้นตอนที่ 3 โดยจะค้นหาแถวของขั้นตอนที่ 3 โดยหาค่า Source ที่เท่ากับ Destination ของแพ็คเก็ตนี้ (บรรทัดที่ 30)

```
Input: Flow entry
Output: OpenFlow rules
1:  p ⟵ Packet header received by the controller
2:  if p is MPTCP packet then
3:      ip_src ⟵ Extract source IP from p
4:      ip_dest ⟵ Extract destination IP from p
5:      port_src ⟵ Extract source port from p
6:      port_dest ⟵ Extract destination port from p
7:      match ⟵ Create match from IP and Port
8:      if match in PathCache then
9:          path ⟵ get path from PathCache
10:     else
11:         mptcp_type ⟵ Extract MPTCP type from TCP option in p
12:         if mptcp_type == MP_CAPABLE then
13:             if SYN packet then
14:                 path ⟵ Get path from Path Selection Algorithm
15:                 Store match and path in PathCache
16:             else if SYN-ACK packet then
17:                 senderKey ⟵ Extract Sender key from p
18:                 token ⟵ Hash a senderKey
19:                 Map token to SYN packet path in PathCache
20:                 path ⟵ Get revert of path from PathCache with IP and Port information
21:                 Store match and path in path PathCache
22:             end if
23:         end if
24:         if mptcp_type == MP_JOIN then
25:             if SYN packet then
26:                 token ⟵ Extract Receiver Token from TCP option
27:                 path ⟵ Get another path
28:                 Store a new match and path to PathCache
29:             else if SYN-ACK packet then
30:                 token ⟵ Lock up from PathCache with IP and Port
                      information
31:                 path ⟵ Get revert of path from PathCache with IP and Port information
32:                 Store match and path in PathCache
33:             end if
34:         end if
35:     end if
36:     Add rules to switches with path information.
37: end if
```

ภาพที่ 3.1 โค้ดเทียมของอัลกอริทึมการระบุ Subflow



Step 1 :

Source	Destination	Token	Subflow
10.1.0.1:55604	10.1.0.2:5001		1
10.1.0.1:34211	10.1.0.2:5001		1

Step 2 :

Source	Destination	Token	Subflow
10.1.0.1:55604	10.1.0.2:5001	1234	1
10.1.0.1:34211	10.1.0.2:5001	5678	1
10.1.0.2:5001	10.1.0.1:55604	1234	1 (reply)
10.1.0.2:5001	10.1.0.1:34211	5678	1 (reply)

Step 3 :

Source	Destination	Token	Subflow
10.1.0.1:55604	10.1.0.2:5001	1234	1
10.1.0.1:34211	10.1.0.2:5001	5678	1
10.1.0.2:5001	10.1.0.1:55604	1234	1 (reply)
10.1.0.2:5001	10.1.0.1:34211	5678	1 (reply)
10.1.0.3:50812	10.1.0.4:5001	1234	2
10.1.0.3:37222	10.1.0.4:5001	5678	2

Step 4 :

Source	Destination	Token	Subflow
10.1.0.1:55604	10.1.0.2:5001	1234	1
10.1.0.1:34211	10.1.0.2:5001	5678	1
10.1.0.2:5001	10.1.0.1:55604	1234	1 (reply)
10.1.0.2:5001	10.1.0.1:34211	5678	1 (reply)
10.1.0.3:50812	10.1.0.4:5001	1234	2
10.1.0.3:37222	10.1.0.4:5001	5678	2
10.1.0.4:5001	10.1.0.3:50812	1234	2 (reply)
10.1.0.4:5001	10.1.0.3:37222	5678	2 (reply)

ภาพที่ 3.2 ตัวอย่างการเริ่มต้นส่งข้อมูล

3.2 ขั้นตอนการเลือกเส้นที่ใช้ส่งข้อมูล

เนื่องจากเครื่องส่งที่เป็นผู้กำหนดจำนวนของ Subflow ไม่ทราบจำนวน Subflow ที่ควรสร้าง และการส่งจำนวน Subflow ที่มากเกินไปจะเป็น Overhead ให้กับเครือข่ายโดยไม่เกิดผลประโยชน์ ดังนั้นงานวิจัยนี้จึงเสนอการเลือกเส้นทางที่ใช้สำหรับ 2 Subflows เนื่องจากเป็นกรณีที่สามารถเกิดขึ้นได้เยอะที่สุดในปัจจุบัน ดังสังเกตจากคอมพิวเตอร์ มือถือ แท็บเล็ต รวมไปถึงเครื่องเซิร์ฟเวอร์มักจะมีส่วนต่อขยายอย่างต่ำจำนวน 2 ส่วนต่อขยาย และเครือข่ายที่นิยมทำ Redundant เพื่อเพิ่มเสถียรภาพ

การเลือกเส้นทางควรจะได้เส้นทางที่ส่งข้อมูลแล้วจะได้ค่า Throughput ที่สูง ดังนั้น Throughput จึงเป็นค่าสำคัญที่เราต้องการทราบเพื่อเปรียบเทียบในการเลือกเส้นทางแต่ละรูปแบบที่สามารถส่งข้อมูลได้ ข้อมูลที่ Controller หาได้และมีผลกับ Throughput ในการส่งข้อมูลของ MPTCP คือแบนด์วิดท์ และค่าความหน่วงของแต่ละเส้นทาง ซึ่งค่าแบนด์วิดท์ที่เราจะใช้คือค่าแบนด์วิดท์ทั้งหมดลบออกจากแบนด์วิดท์ที่ถูกใช้อยู่ในขณะนั้น ส่วนค่าความหน่วงจะมีผลทำให้การทำ

Three-Way Handshake ทำได้ช้าและยังมีผลกับการส่งข้อมูลแบบหลายเส้นทางเพราะค่าความหน่วงของ 2 เส้นทางที่ต่างกันจะทำให้เกิดปัญหา Head-of line blocking และจะส่งผลกับ Throughput ด้วย แต่ค่าความสำคัญของตัวแปรเหล่านี้ที่ไม่เท่ากันและขึ้นกับลำดับของเส้นทางที่เริ่มส่งด้วย จึงทำให้ประมาณค่า Throughput จากข้อมูลเหล่านี้ได้ยาก

งานวิจัยชิ้นนี้จึงเสนอการหาค่าความสัมพันธ์ของแบนด์วิดท์เส้นทางที่ 1 แบนด์วิดท์เส้นทางที่ 2 ค่าความหน่วงเส้นทางที่ 1 และค่าความหน่วงเส้นทางที่ 2 เพื่อประมาณหาค่า Throughput โดยใช้สมการที่ได้จากกระบวนการทำ Multiple Linear Regression

การหาข้อมูลเริ่มต้นเพื่อใช้ในกระบวนการทำ Multiple Linear Regression เริ่มต้นจากกำหนดค่าข้างต้นประจำแต่เส้นทางตามตารางที่ 3.1 ซึ่งเป็นค่าที่กำหนดมาให้ครอบคลุมค่าในโดเมนที่เราจะเอาไปใช้ จากนั้นทดลองส่งข้อมูลแบบ MPTCP ไปทุกรูปแบบการจับคู่เส้นทาง เพื่อทดสอบหาค่า Throughput ซึ่งการทดลองนี้ในแต่ละรูปแบบการทดลองจะทำการทดลอง 3 ครั้งและหาค่าเฉลี่ย ซึ่งผลการทดลองจะแสดงตามภาพที่ 3.3 และนำผลการทดลองมาสร้างสมการที่ (2) ซึ่งได้จากการคำนวณสมการ Multiple Linear Regression โดยใช้ภาษา R ผ่านโปรแกรม RStudio เพื่อประมาณหาค่า Throughput โดยกำหนดให้

BW1 คือแบนด์วิดท์เส้นทางที่ 1

BW2 คือแบนด์วิดท์เส้นทางที่ 2

D1 คือค่าความหน่วงเส้นทางที่ 1

D2 คือค่าความหน่วงเส้นทางที่ 2

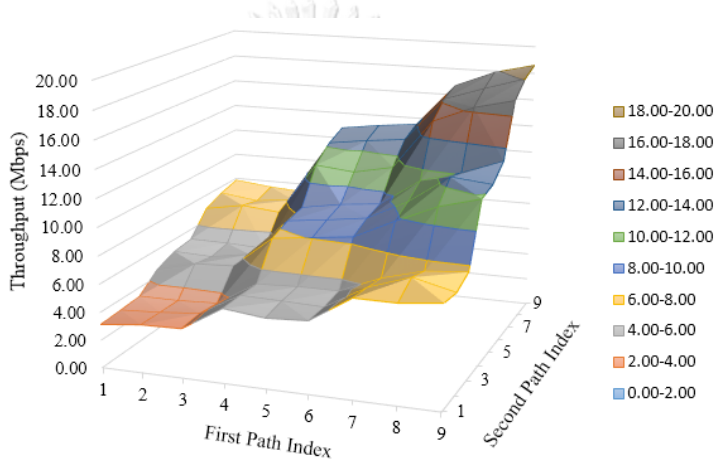
TP คือค่า Throughput

ตารางที่ 3.1 คุณสมบัติแต่ละเส้นทาง จากต้นทางถึงปลายทาง

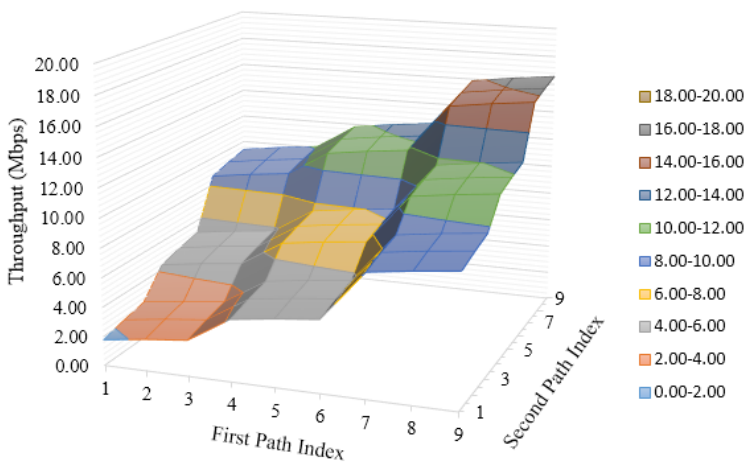
Path Index	Bandwidth (Mbps)	Delay (ms)
1	2	200
2	2	100
3	2	20
4	5	200
5	5	100
6	5	20
7	10	200
8	10	100
9	10	20

$$TP = 0.021327 + (0.857795 \times BW1) + (0.811956 \times BW2) - (0.004140 \times D1) - (0.003819 \times D2) \quad (2)$$

ภาพที่ 3.4 แสดงกราฟผลของ Throughput ที่ได้จากการแทนค่าสมการที่ (2) พบว่าเมื่อเปรียบเทียบกราฟในภาพที่ 3.3 และภาพที่ 3.4 แล้วมีลักษณะที่ใกล้เคียงกัน งานวิจัยนี้จึงนำสมการนี้ไปใช้ใน Controller เพื่อการประมาณค่า Throughput ของแต่ละคู่เส้นทางที่เป็นไปได้ และทำการเลือกเส้นทางที่มีการประมาณค่า Throughput ได้สูงสุด



ภาพที่ 3.3 ผล Throughput จากการทดลองในแต่ละคู่เส้นทาง



ภาพที่ 3.4 ผล Throughput จากการแทนค่าสมการในแต่ละคู่เส้นทาง

กระบวนการเลือกเส้นทางจะทำตามกระบวนการตามภาพที่ 3.5 ซึ่งจะมีการระบุค่าคงที่ k (บรรทัดที่ 1) ซึ่งจะใช้เป็นจำนวนสูงสุดของจำนวนเส้นทางที่จะเอามาพิจารณา (บรรทัดที่ 2) เนื่องจากกระบวนการเลือกเส้นทางที่เราจะทำการเปรียบเทียบเส้นทางทั้งหมดที่เป็นไปได้ เมื่อเครือข่ายมีขนาดใหญ่ทำให้เส้นทางที่เป็นไปได้มีจำนวนมาก หากไม่มีการจำกัดจำนวนเส้นทางจะมีผลทำให้สิ้นเปลืองการทำงานที่อาจไม่จำเป็น ในการทดลองนี้ได้กำหนดค่า k = 30 (ได้จากการทดลองปรับเปลี่ยนค่าให้เหมาะสมกับการทดลองนี้) ในการกำหนดค่า k ที่มีค่าน้อยอาจจะมีผลทำให้เส้นทางที่ดีที่สุดไม่ถูกพิจารณาเพราะอาจจะเป็นเส้นทางที่อัลกอริทึมของ Dijkstra ยังไม่พบเส้นทางนั้น หากทำการกำหนดค่า k ที่มีค่าสูงเกินไปจะทำให้อัลกอริทึมของ Dijkstra ใช้เวลาเพิ่มขึ้นในการหาเส้นทาง และกระบวนการเลือกเส้นทางต้องคำนวณค่าแบนด์วิดท์และความหน่วงของจำนวนเส้นทางที่มากมาเปรียบเทียบกันด้วย

หลังจากได้เส้นทางตามจำนวนที่ต้องการจะทำการดึงเส้นทางมาทีละคู่ (บรรทัดที่ 5 และบรรทัดที่ 8) เพื่อไปคำนวณประมาณ Throughput จาก Multiple Linear Regression Equation (บรรทัดที่ 10) โดยกระบวนการนี้จะทำทุกคู่เส้นทางที่ไม่ได้ใช้เส้นทางร่วมกัน (บรรทัดที่ 9) เพื่อหาคู่ของเส้นทางที่ทำให้ค่า Throughput รวมมีค่าสูงสุด

```
Input: Source and Destination switches
Output: OutputPathSet
1:  k ⟵ Maximum number of paths from Routing module.
2:  availablePathList ⟵ Dijkstra's algorithm from Routing module function
3:  i ⟵ 1
4:  While i<= k and i <= number of paths in availablePathList do
5:      tempFirstPath ⟵ Next path from availablePathList
6:      j ⟵ 1
7:      While j<= k and j <= number of paths in availablePathList do
8:          tempSecondPath ⟵ Next path from availablePathList
9:          if tempSecondPath not using same sub path with tempSecondPath then
10:             estTP ⟵ estimated aggregate throughput of tempFirstPath and tempSecondPath
                        using Multiple Linear Regression Equation
11:             if estTP > BestEstTP then
12:                 BestPathSet ⟵ tempFirstPath and tempSecondPath
13:                 BestEstTP ⟵ estTP
14:             end if
15:         end if
16:         j ⟵ j+1
17:     end while
18:     i ⟵ i+1
19: end while
20: OutputPathSet ⟵ BestPathSet
```

ภาพที่ 3.5 แสดงโค้ดเทียมของอัลกอริทึมการเลือกเส้นทาง

บทที่ 4
การทดลองและวิเคราะห์ผลการทดลอง

การทดลองในงานวิจัยนี้จะทำการจำลองเครือข่ายในเครื่อง Mininet Server เพื่อทดสอบส่งข้อมูลในเครือข่ายที่ได้จำลองขึ้น และทำการเขียนโปรแกรมควบคุมการทำงานของ Controller บนเครื่อง Controller

สภาพแวดล้อมและเครื่องมือที่ใช้ในการพัฒนามีดังนี้

1. Mininet Server
    - VMware ESXi 6.5
    - 4C4T vCPUs Intel(R) Core(TM) i7-6770HQ CPU @ 2.60GHz
    - 4 GB RAM
    - 48 GB SSD Drive
    - Ubuntu Server 14.04.4 LTS
    - MPTCP: Stable release v0.91.2 [10]
    - Iperf 2.0.8 [9]
    - Python 2.7.6
    - Mininet Network Emulator 2.2.1 [6]
2. Controller
    - VMware ESXi 6.5
    - 4C4T vCPUs Intel(R) Core(TM) i7-6770HQ CPU @ 2.60GHz
    - 4 GB RAM
    - 32 GB SSD Drive
    - Ubuntu Server 14.04.4 LTS
    - OpenJDK 1.8.0_151
    - Floodlight 1.2

งานวิจัยนี้ใช้ 1 ส่วนต่อขยายของเครื่องส่งและเครื่องรับในการทดลอง เนื่องด้วย MPTCP เวอร์ชั่น 0.90 ขึ้นไปสามารถกำหนดให้ส่งได้มากกว่า 1 Subflow ใน 1 คู่ของ IP Address

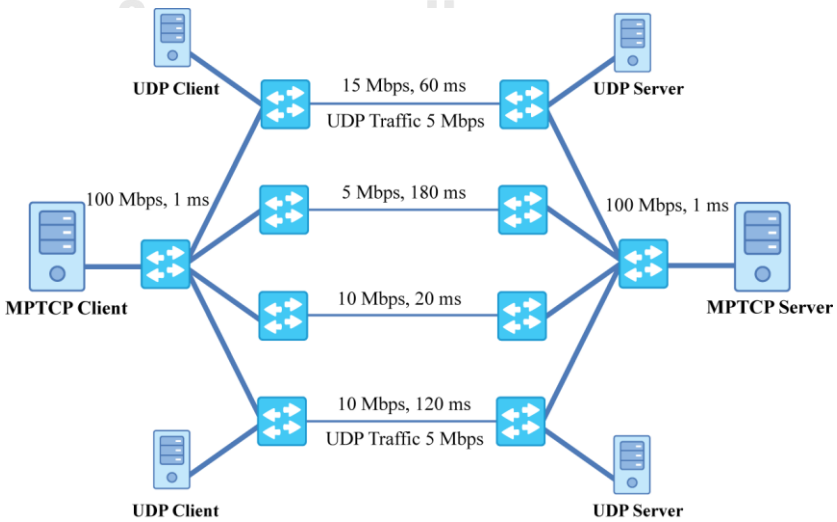
งานวิจัยนี้จะทำการทดลอง 2 ชุด ในแต่ละชุดออกแบบเป็นสองรูปแบบเพื่อเปรียบเทียบประสิทธิภาพ โดยที่รูปแบบแรก Controller ทำการเลือกเส้นทางเป็นแบบไม่ซ้ำกันของ Shortest

Path จากอัลกอริทึมของ Dijkstra ซึ่งเป็นค่าเริ่มต้นของ Controller และแบบที่สองคือ Controller ทำกระบวนการระบุ Subflow ตามหัวข้อ 3.1 และกระบวนการเลือกเส้นทางตามหัวข้อ 3.2 แต่ละรูปแบบของการทดลองจะทดลอง 5 ครั้ง ครั้งละ 30 วินาที จากการส่งข้อมูลด้วยโปรแกรม Iperf

4.1 การทดลองที่ 1

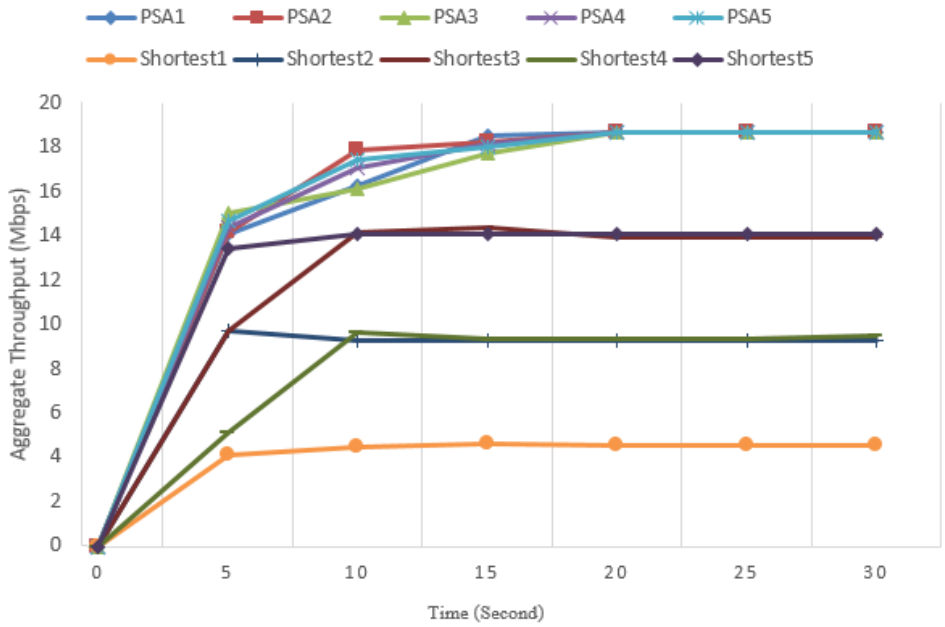
ทดลองบนโครงสร้างเครือข่ายตามภาพที่ 4.1 ซึ่งเป็นเครือข่ายอย่างง่ายที่มีลักษณะคล้ายท่อนเหล็กยกน้ำหนัก (Simple Dumbbell Topology) การทดลองจะตั้งค่าแบนด์วิดท์ ค่าความหน่วง ซึ่งตลอดการทดลองจะมีข้อมูล UDP ผสมตามเส้นทางที่ระบุในภาพ โดยโครงสร้างเครือข่ายถูกออกแบบมาให้ดูง่าย ไม่ซับซ้อน สะดวกในการ Implement สะดวกวิเคราะห์และตรวจสอบความถูกต้องของผลการทดลอง โดยทุกเส้นทางเป็น Shortest Path อันเนื่องมาจากเราต้องการเปรียบเทียบกับ Algorithms การเลือกเส้นทางแบบ Shortest Path กับ Algorithms ที่ได้เสนอไป

หากใช้โครงสร้างเครือข่ายแบบอื่นที่มีเส้นทางที่ไม่ใช่ Shortest Path ด้วย เส้นทางนั้นจะไม่ถูกใช้งานด้วย Algorithms แบบ Shortest Path ซึ่ง Throughput ของการเลือกเส้นทางแบบ Shortest Path จะถูกกำหนดโดยการกำหนดค่าแบนด์วิดท์และความหน่วงของเส้นทางที่เป็น Shortest Path หากกำหนดให้เส้นทางที่เป็นเส้นทางที่ Shortest Path มีค่าแบนด์วิดท์สูง ค่าความหน่วงต่ำ จะทำให้การเลือกเส้นทางแบบ Shortest Path ได้ผลการทดลองออกมาดี แต่ในกรณีที่ทุกเส้นทางในโครงสร้างเครือข่ายเป็น Shortest Path ทั้งหมดนั้น ทำให้การเลือกแบบ Shortest Path นั้นทำงานแบบสุ่ม มีผลให้การเลือกเส้นทางแบบ Shortest Path มีโอกาสใช้งานทุกเส้นทางเท่า ๆ กัน



ภาพที่ 4.1 โครงสร้างเครือข่ายที่ใช้ในการทดลองที่ 1

ผลการทดลองจากค่า Throughput ที่แสดงจากโปรแกรม Iperf ทุก ๆ 5 วินาที สามารถแสดงได้ตามภาพที่ 4.2 ซึ่งแสดงผล Throughput รวมจากแต่ละเส้นทาง ที่ได้จากการคำนวณเส้นทางโดยใช้ Path Selection Algorithm หรือ PSA จำนวน 5 การทดลอง และผล Throughput ที่ได้จากการคำนวณเส้นทางโดยใช้ Shortest Path Algorithm หรือ Shortest จำนวน 5 การทดลอง



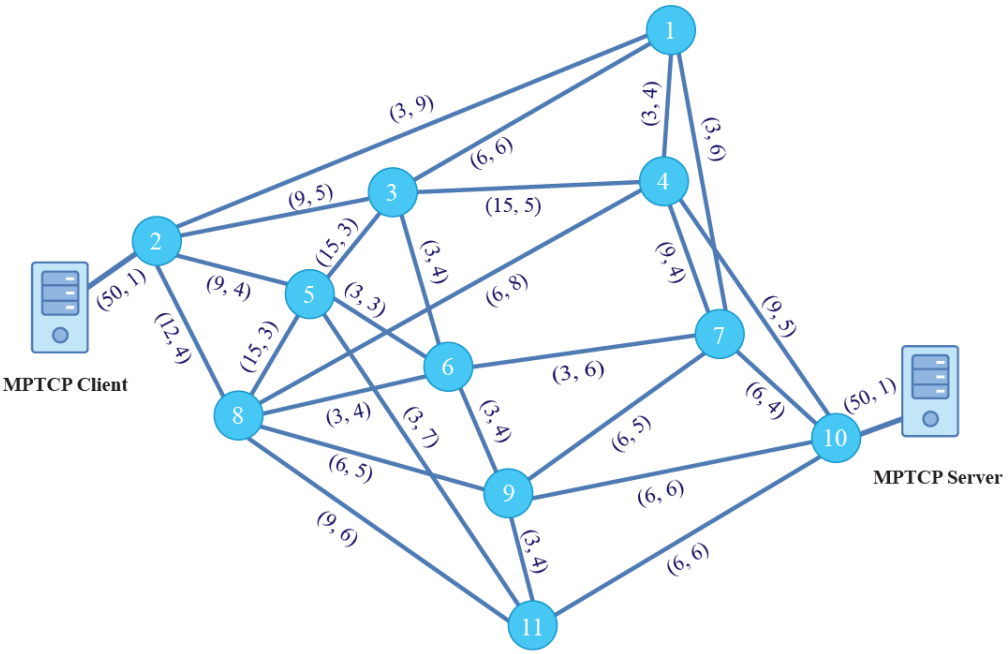
ภาพที่ 4.2 ผลที่ได้จากการทดลองที่ 1

จากผลการทดลองพบว่าค่า Throughput ที่ได้จากการคำนวณเส้นทางโดยใช้ Path Selection Algorithm จะสามารถเลือกเส้นทางที่ทำให้ได้ค่า Throughput ที่สูงที่สุด และเลือกเส้นทางเดียวกันทั้ง 5 การทดลอง ทำให้ผล Throughput ในภาพที่ 4.2 เกาะกลุ่มอยู่ที่ประมาณ 17 Mbps ส่วนผล Throughput ที่ได้จากการใช้ Shortest Path Algorithm จะไม่เกาะกลุ่มกัน และมีค่าเฉลี่ยจาก 5 การทดลองอยู่ที่ 10.01 Mbps อันเนื่องมาจากการเลือกเส้นทางที่สั้นที่สุดที่มีหลายเส้นทางทำให้มีลักษณะคล้ายกับการสุ่มเลือกเส้นทาง

4.2 การทดลองที่ 2

ทำการทดลองบนโครงสร้างเครือข่ายตามภาพที่ 4.3 ซึ่งเป็นโครงสร้างเครือข่ายจำลองของ COST 239 [16] ซึ่งเป็นเครือข่ายจริงของความร่วมมือของกลุ่มประเทศในทวีปยุโรป โดยตารางที่ 4.1

จะระบุชื่อเมืองของที่ตั้งในแต่ละ Node ตัวเลขในวงเล็บของภาพที่ 4.3 คือค่าแบนด์วิดท์ในหน่วย Mbps และค่าความหน่วงในหน่วย ms



ภาพที่ 4.3 โครงสร้างเครือข่ายที่ใช้ในการทดลองที่ 2

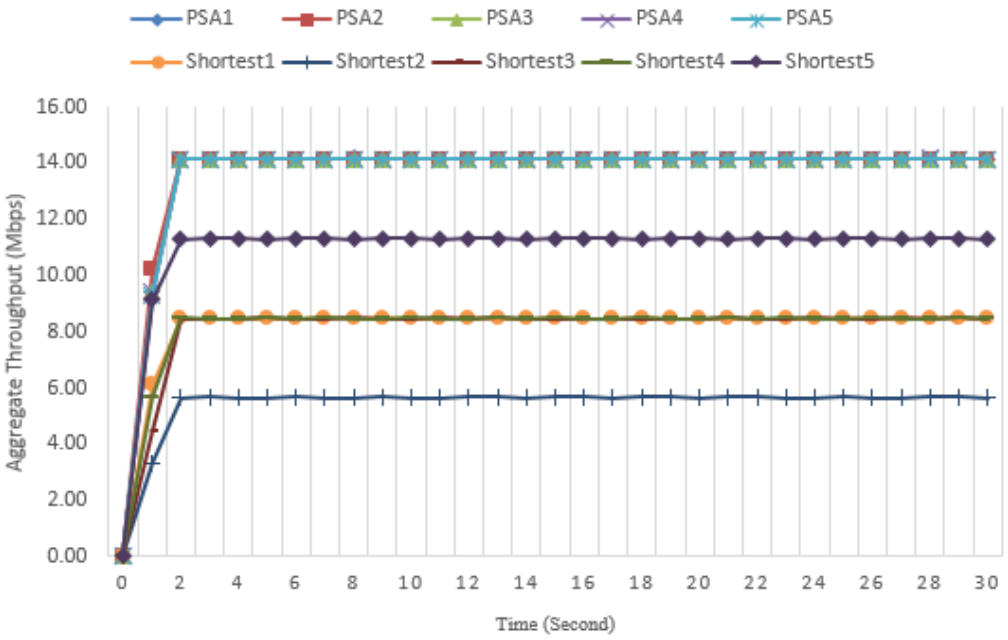
ตารางที่ 4.1 ตารางแสดงที่ตั้งของแต่ละ Node

Node ที่	ที่ตั้ง
1	Copenhagen
2	London
3	Amsterdam
4	Berlin
5	Brussels
6	Luxembourg
7	Prague
8	Paris
9	Zurich
10	Vienna
11	Milan

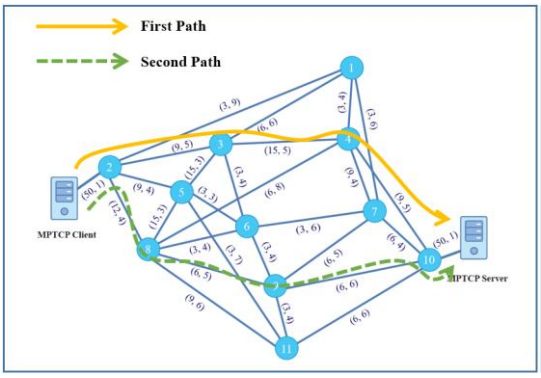
แต่ละรูปแบบของการทดลองจะทดลอง 5 ครั้ง ในแต่ละครั้งจะ ใช้โปรแกรม Wireshark ดักจับข้อมูลที่ Network Card ของเครื่อง MPTCP Server และใช้โปรแกรม Iperf สร้างข้อมูลที่เป็น TCP ส่งข้อมูลจากเครื่อง MPTCP Client ไปยังฝั่ง MPTCP Server ซึ่งแต่ละการทดลองสามารถบันทึกค่า Throughput ได้การเขียนโปรแกรมอ่านข้อมูลที่ส่งที่บันทึกไว้ด้วยโปรแกรม Wireshark

ผลการทดลองดังแสดงในภาพที่ 4.4 ซึ่งแสดงค่า Throughput รวมจากแต่ละเส้นทาง ที่ได้จากการคำนวณเส้นทางโดยใช้ Path Selection Algorithm หรือ PSA จำนวน 5 การทดลอง และผล Throughput ที่ได้จากการคำนวณเส้นทางโดยใช้ Shortest Path Algorithm หรือ Shortest จำนวน 5 การทดลอง

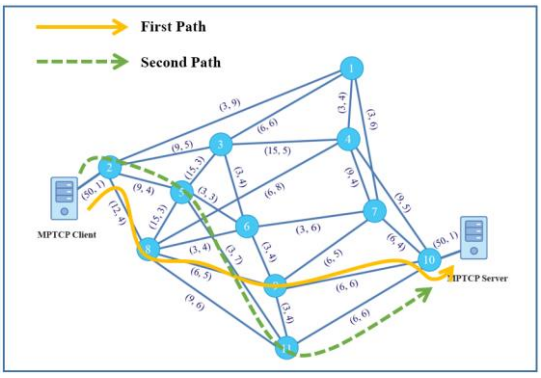
จากผลพบว่าผล Throughput ที่ได้จากการคำนวณเส้นทางโดยใช้ Path Selection Algorithm จะสามารถเลือกเส้นทางที่ทำให้ได้ค่า Throughput ที่สูงที่สุด และเลือกเส้นทางเดียวกันทั้ง 5 การทดลองตามภาพที่ 4.5 (ก) ทำให้ผล Throughput ในภาพที่ 4.4 เกาะกลุ่มอยู่ที่ 14 Mbps ส่วนผล Throughput ที่ได้จากการใช้ Shortest Path Algorithm จะไม่เกาะกลุ่มกันตามภาพที่ 4.5 (ข) ถึง ภาพที่ 4.5 (ฉ) และมีค่าเฉลี่ยจาก 5 การทดลองอยู่ที่ 8.4 Mbps อันเนื่องมาจากการเลือกเส้นทางที่สั้นที่สุดที่มีหลายเส้นทางทำให้มีลักษณะคล้ายกับการสุ่มเลือกเส้นทาง



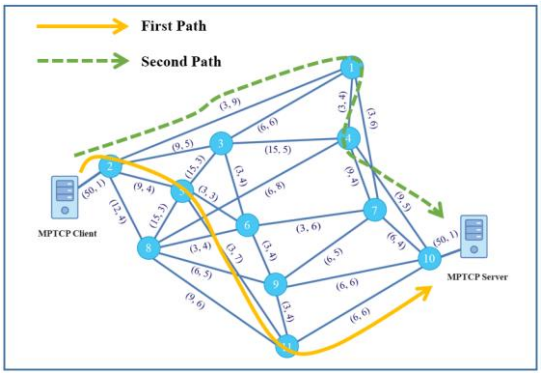
ภาพที่ 4.4 ผลที่ได้จากการทดลองที่ 2



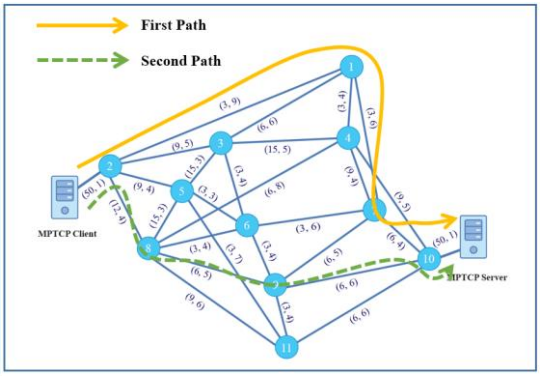
(ก) การทดลองแบบ PSA



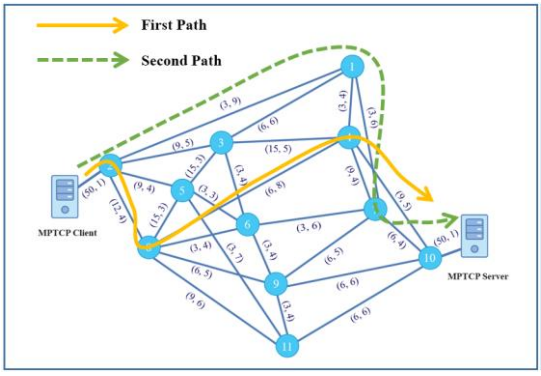
(ข) การทดลองแบบ Shortest ครั้งที่ 1



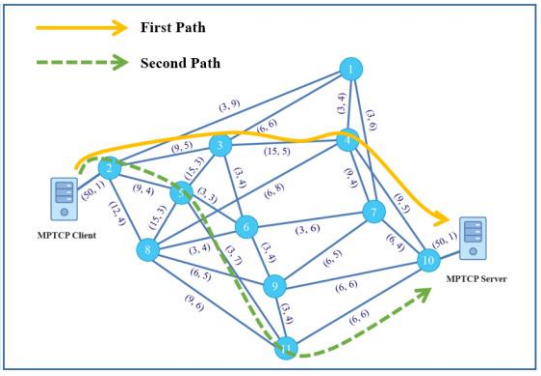
(ค) การทดลองแบบ Shortest ครั้งที่ 2



(ง) การทดลองแบบ Shortest ครั้งที่ 3



(จ) การทดลองแบบ Shortest ครั้งที่ 4



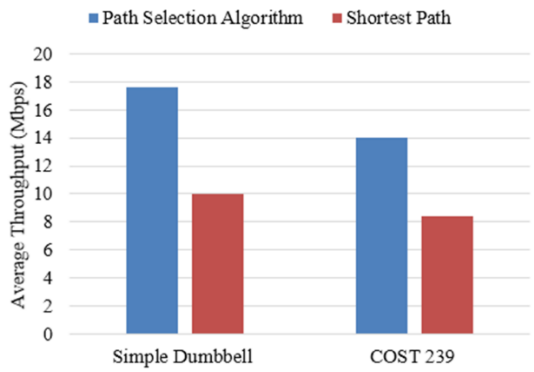
(ฉ) การทดลองแบบ Shortest ครั้งที่ 5

ภาพที่ 4.5 แสดงเส้นทางในการส่งข้อมูล

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

งานวิจัยนี้แสดงให้เห็นว่าสามารถนำ SDN เข้ามาช่วยควบคุมการส่งข้อมูลที่ใช้โพรโทคอล MPTCP จากผลการทดลองที่ทดลองใน Simple Dumbbell Topology และ COST 239 Topology พบว่ากระบวนการเลือกเส้นทางที่ได้นำเสนอไปนี้สามารถทำให้ค่าเฉลี่ยของ Throughput จากการทดลอง 2 การทดลองนี้ดีขึ้นร้อยละ 76 และดีขึ้นร้อยละ 66.6 เมื่อเปรียบเทียบกับการเลือกเส้นทางแบบสั้นที่สุด ตามภาพที่ 5.1 อันเนื่องมาจากกระบวนการนี้เป็นการเลือกเส้นทางที่คำนวณและประมาณค่า Throughput ไว้ก่อนที่จะส่งข้อมูล โดยอัลกอริทึมที่นำเสนอนี้สามารถระบุแต่ Subflow แยกออกกันได้อย่างถูกต้องจากข้อมูลในแพ็คเก็ต และการพิจารณาเลือกเส้นทางจากค่าตัวแปรที่มีผลกับค่า Throughput ของการส่งข้อมูลแบบ MPTCP ซึ่งค่าความสำคัญของแต่ละตัวแปรถูกคำนวณจากกระบวนการทำ Multiple Linear Regression เพื่อให้ได้เส้นทางที่สามารถส่งข้อมูลได้อย่างมีประสิทธิภาพ

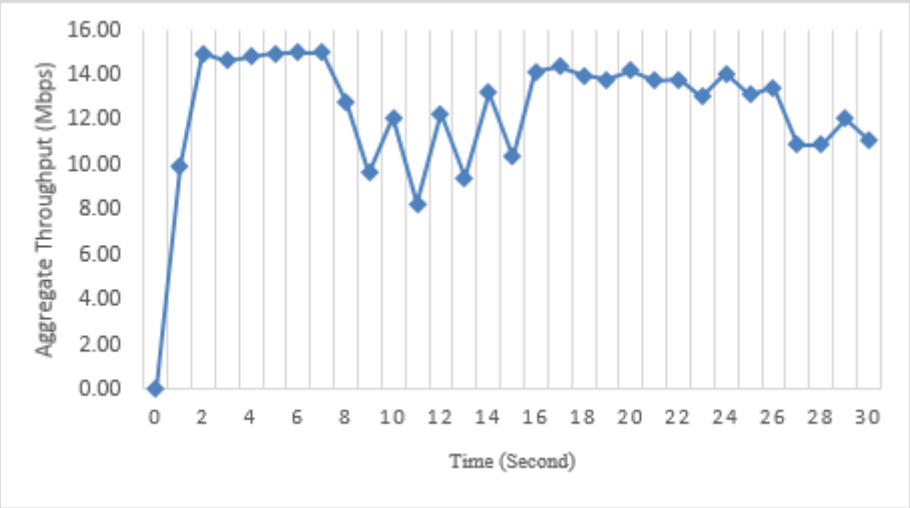


ภาพที่ 5.1 แสดงค่า Throughput โดยเฉลี่ยของแต่ละการทดลอง

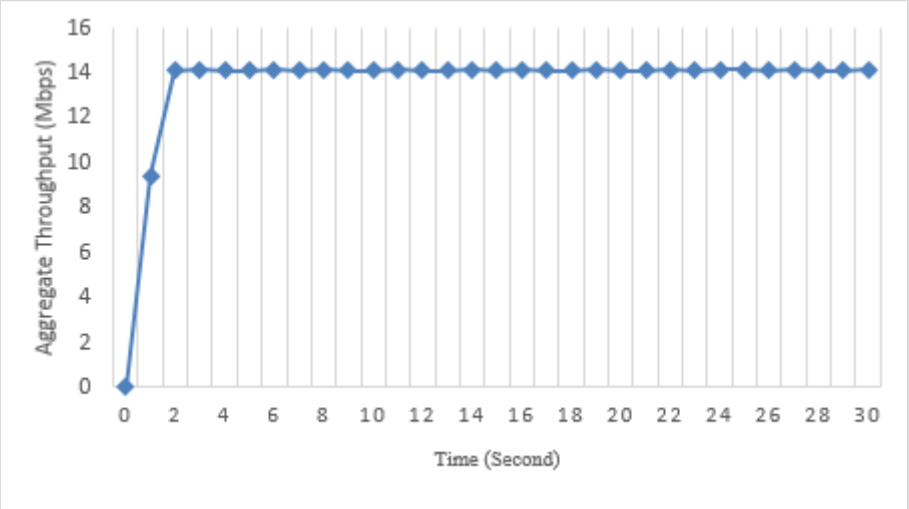
5.2 ข้อเสนอแนะ

การวัดค่า Throughput ที่ได้จากการทดลองใน Mininet อาจจะได้ค่าที่ไม่ถูกต้องเนื่องจากปัจจัยหลักเช่น การใช้จำนวน switch หรือตั้งค่าค่าแบนด์วิดท์ที่มากเกินไป การใช้โปรแกรม Wireshark ดักจับแพ็คเก็ตจำนวนมากเพราะโปรแกรม Wireshark จะเก็บข้อมูลทั้งหมดที่ส่ง กราฟ

Throughput ที่ไม่เสถียรแสดงในภาพที่ 5.2 และกราฟ Throughput ที่เสถียรแสดงในภาพที่ 5.3 โดยงานวิจัยนี้ได้แก้ปัญหาโดยการเลือกจำนวน switch และค่าแบนด์วิดท์ที่เหมาะสมกับประสิทธิภาพของเครื่องคอมพิวเตอร์ที่จะนำมาใช้ทดลอง และตั้งค่าโปรแกรม Wireshark ให้เก็บค่าเฉพาะส่วนของ Header



ภาพที่ 5.2 แสดงค่า Throughput ตัวอย่างที่ไม่เสถียร



ภาพที่ 5.3 แสดงค่า Throughput ตัวอย่างที่เสถียร

กระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทางในเครือข่ายที่กำหนดโดยซอฟต์แวร์ที่ได้นำเสนอนี้จะเหมาะกับแอปพลิเคชันที่สร้างการเชื่อมต่อ TCP เป็นเวลานาน (long-lived connections) เพราะหากเป็นการเชื่อมต่อระยะสั้นจะไม่เกิดประโยชน์ในการคำนวณเส้นทาง 2 เส้นทาง เนื่องจากข้อมูลจะถูกส่งใน Subflow แรกเสร็จก่อนที่จะสร้าง Subflow ที่สองเสร็จ

ในงานวิจัยนี้เป็นการทดลองที่กำหนดจำนวน Subflow ต่อหนึ่งการเชื่อมต่อที่ 2 Subflows เนื่องจากเครื่องผู้ส่งที่เป็นผู้กำหนดจำนวนของ Subflow ไม่ทราบจำนวน Subflow ที่ควรสร้าง และการส่งจำนวน Subflow ที่มากเกินไปจะเป็น Overhead ให้กับเครือข่ายโดยไม่เกิดผลประโยชน์ และมาจากเป็นกรณีที่สามารถเกิดขึ้นได้เยอะในปัจจุบัน ดังสังเกตจากคอมพิวเตอร์ มือถือ แท็บเล็ต รวมไปถึงเครื่องเซิร์ฟเวอร์มักจะมีส่วนต่อขยายอย่างต่ำจำนวน 2 ส่วนต่อขยาย และเครือข่ายที่นิยมทำ Redundant เพื่อเพิ่มเสถียรภาพ อย่างไรก็ตามหากต้องการเพิ่มจำนวน Subflow สามารถทำได้ 2 วิธีคือสร้างสมการ Multiple Linear Regression ที่รับตัวแปรของ Subflow ที่เพิ่มขึ้น อีกหนึ่งวิธีที่สามารถเพิ่มจำนวน Subflow ได้คือ Subflow ที่ 3 เป็นต้นไปอาจจะพิจารณาการเลือกเส้นทางโดยใช้เฉพาะค่าแบนด์วิดท์เพียงอย่างเดียว เป็นต้น

สิ่งที่ควรวัดผลเพิ่มเติมจากงานวิจัยนี้นอกจากค่า Throughput ได้แก่ ค่าความหน่วง การทำงานที่เพิ่มมากขึ้นของ Controller

รายการอ้างอิง

[1] J. Postel. (1981). *TRANSMISSION CONTROL PROTOCOL*. Available: https://tools.ietf.org/html/rfc793

[2] A. Ford, Raiciu, C., Handley, M., and O. Bonaventure. (2013). *TCP Extensions for Multipath Operation with Multiple Addresses*. Available: https://tools.ietf.org/html/rfc6824

[3] (2016). *Use Multipath TCP to create backup connections for iOS*. Available: https://support.apple.com/en-hk/HT201373

[4] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 3, pp. 1617-1634, 2014.

[5] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.,* vol. 38, no. 2, pp. 69-74, 2008.

[6] B. Lantz and B. Heller. *Mininet*. Available: http://mininet.org

[7] R. Izard. *Floodlight OpenFlow Controller*. Available: http://www.projectfloodlight.org

[8] J. Postel. (1980). *User Datagram Protocol*. Available: https://tools.ietf.org/html/rfc768

[9] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. (2010). *Iperf*. Available: https://sourceforge.net/projects/iperf/

[10] *MultiPath TCP - Linux Kernel implementation*. Available: http://multipath-tcp.org/

[11] J. Postel. (1981). *Internet Protocol*. Available: http://www.rfc-editor.org/info/rfc791

[12] D. Eastlake 3rd and P. Jones. (2001). *US Secure Hash Algorithm 1 (SHA1).* Available: https://tools.ietf.org/html/rfc3174

[13] R. v. d. Pol *et al.*, "Multipathing with MPTCP and OpenFlow," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 1617-1624.

[14] M. Sandri, A. Silva, L. A. Rocha, and F. L. Verdi, "On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015, pp. 9-16.

[15] S. Zannettou, M. Sirivianos, and F. Papadopoulos, "Exploiting path diversity in datacenters using MPTCP-aware SDN," in *Proceedings - IEEE Symposium on Computers and Communications*, 2016, vol. 2016-August, pp. 539-546.

[16] H. Wensheng, F. Jing, and A. K. Somani, "A p-cycle based survivable design for dynamic traffic in WDM networks," in *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, 2005, vol. 4, p. 5 pp.

ประวัติผู้เขียนวิทยานิพนธ์

นายจิรศักดิ์ จุลวัจน์ จบการศึกษาในระดับมัธยมศึกษาตอนปลายจากโรงเรียนพรมคีรีพิทยาคม จังหวัดนครศรีธรรมราช จากนั้นเข้าศึกษาต่อในปีการศึกษา 2548 ที่ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยธรรมศาสตร์ และสำเร็จการศึกษาในหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต ปีการศึกษา 2551 และได้เข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต ที่ภาควิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2558