การศึกษาเชิงปริมาณของกระบวนการเลือกเส้นทางของทีซีพีแบบหลายเส้นทางในเครือข่ายที่กำหนดโดยซอฟต์แวร์

Quantitative Study of Path Selection Algorithm for Multipath TCP in Software-Defined Networking

จิรศักดิ์ จุลวัจน์ (Jirasak Jullawat)* เกียรติคุณ กาวิละ (Kiattikun Kawila)*
กุลิสร์ ณ นคร (Kulit Na Nakorn)* และ กุลธิดา โรจน์วิบูลย์ชัย (Kultida Rojviboonchai)*

Received : October 20, 2017
Revised : April 28, 2018
Accepted : May 8, 2018

บทคัดย่อ

ในปัจจุบันการส่งข้อมูลภายในระบบเครือข่ายมีความซับซ้อนและมีเส้นทางมากมายที่สามารถใช้ส่งข้อมูลได้ เพื่อเพิ่มค่า Throughput ของข้อมูลนำส่ง และเพิ่มความคงทนต่อความเสียหาย (Fault Tolerance) สามารถทำได้โดยใช้ความสามารถของโปรโตคอล Multipath TCP (MPTCP) อย่างไรก็ตามการส่งข้อมูลยังคงเกิดการใช้เส้นทางร่วมกันและก่อให้เกิดปรากฏการณ์คอขวดเนื่องจากระบบเครือข่ายใช้วิธีเลือกเส้นทางแบบสั้นที่สุดในการขนส่งเท่านั้น ดังนั้นงานวิจัยชิ้นนี้จึงได้ศึกษาเพิ่มเติมจากงานวิจัยของเราก่อนหน้านี้ ซึ่งใช้การนำส่งข้อมูลภายใต้เครือข่ายที่กำหนดโดยซอฟต์แวร์ในเครือข่ายที่มีขนาดใหญ่และซับซ้อนมากขึ้นเพื่อยืนยันแนวคิดการเลือกเส้นทางส่งข้อมูลที่เหมาะสมโดยใช้กระบวนการระบุ Subflow ใน SDN และใช้ค่าแบนด์วิดท์ที่พร้อมใช้งาน (Available Bandwidth) ในขณะทำการส่งข้อมูลกับค่าความหน่วงเป็นปัจจัยสำหรับการเลือกเส้นทาง ซึ่งผลลัพธ์ค่า Throughput จากการวัดประสิทธิภาพในการทดลองนี้พบว่าสามารถส่งข้อมูลได้สูงกว่าการเลือกเส้นทางแบบที่สั้นที่สุดร้อยละ 66.6

คำสำคัญ: ทีซีพีแบบหลายเส้นทาง โอเพนโฟลว เครือข่ายที่กำหนดโดยซอฟต์แวร์ แบนด์วิดท์ ค่าหน่วง

Abstract

Nowadays, the complicated network has multiple paths to transport data and to improve throughput and fault tolerance can do by using Multipath TCP (MPTCP) features, but the problem is the data transmission can still transport in the same path and cause bottleneck effect because the factor that use to select paths is using shortest path algorithm. Therefore, the aim of the present study was to extend our previous work that used MPTCP in Software-Defined Networking but study in the bigger and more complicated network to confirm in selecting the suitable paths by using Subflow identification algorithm and path selection algorithm in SDN by using available bandwidth and delay as a factor. The outcomes of this study show the throughput was 66.6% improved than using shortest path algorithm as a factor in selecting paths.

Keywords: Multipath-TCP, OpenFlow, Software-Defined Networking, Bandwidth, Delay.

1. บทนำ

สังคมโลกยุคโลกาภิวัตน์ เป็นสังคมที่ก้าวข้ามข้อจำกัดในการติดต่อสื่อสาร ทุกส่วนของโลกสามารถแลกเปลี่ยนข้อมูลข่าวสารอย่างไร้พรมแดนโดยอาศัยเทคโนโลยีสารสนเทศเป็นเครื่องมือ ซึ่งการส่งข้อมูลจากอุปกรณ์หนึ่งไปยังอีกอุปกรณ์หนึ่งผ่านระบบเครือข่ายสามารถส่งข้อมูลพร้อมกันได้หลายเส้นทาง และการส่งต่อข้อมูลผ่านอุปกรณ์เครือข่ายและโครงสร้างพื้นฐานของระบบเครือข่ายโดยมาก ยังคงใช้ Transmission Control Protocol (TCP) [1] หากแต่ยังเป็นวิธีการที่มีข้อจำกัดเนื่องจากสามารถส่งข้อมูลได้เพียงเส้น

* ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
* Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University.

** ส่วนหนึ่งของบทความวิจัยนี้ได้นำเสนอในงานประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 13 และได้รับรางวัล Best Paper Award

ทางเดียวเท่านั้น ด้วยเหตุนี้จึงมีผู้พัฒนา Multipath TCP ขึ้นเพื่อใช้เส้นทางการส่งข้อมูลที่มากขึ้นในโลกยุคปัจจุบันให้เกิดประโยชน์สูงสุด

Multipath TCP (MPTCP) [2] คือส่วนต่อขยาย (Extension) ของ TCP โดยทั่วไป ที่ถูกพัฒนาขึ้นเพื่อแก้ไขข้อจำกัดของ TCP โดยสามารถควบคุมการส่งข้อมูลผ่านหลายเส้นทาง ซึ่งแม้ว่าในปัจจุบัน MPTCP ยังไม่ได้มีการใช้งานอย่างแพร่หลาย แต่ได้มีการเริ่มใช้ในระบบปฏิบัติการ iOS แล้ว [3] เนื่องจากหากการเชื่อมต่อของ TCP ถูกขัดขวางจนสูญเสียการเชื่อมต่อ จะต้องใช้เวลาในการทำกระบวนการ Three-way Handshake อีกครั้งเพื่อเริ่มกระบวนการส่งข้อมูลใหม่ ยิ่งไปกว่านั้นหากสูญเสียการเชื่อมต่อ ในขณะที่กำลังส่งข้อมูลที่มีขนาดใหญ่ อาจต้องเริ่มต้นส่งข้อมูลใหม่ภายหลังการเชื่อมต่อ ซึ่งต่างจากการใช้ MPTCP ที่ยังคงสามารถใช้เส้นทางอื่นในการส่งข้อมูลหากเส้นทางปกติถูกขัดขวาง และสามารถกลับมาใช้เส้นทางเดิมได้อีกครั้งหากสิ่งที่ขัดขวางได้รับการแก้ไขแล้ว เนื่องจาก MPTCP หนึ่งการเชื่อมต่อจะสามารถสร้าง Subflow ย่อยได้ และแต่ละ Subflow สามารถใช้ในการส่งข้อมูลผ่านเส้นทางหรือส่วนต่อขยายที่ต่างกันได้ อย่างไรก็ตามการส่งข้อมูลด้วยวิธีการ MPTCP อาจมีการใช้เส้นทางร่วมกันและสามารถก่อให้เกิดปรากฏการณ์คอขวด (Bottleneck) ขึ้นภายในเครือข่ายเนื่องจากความซับซ้อนของระบบเครือข่ายและกระบวนการเลือกเส้นทางแบบสั้นที่สุด (Shortest Path) แต่ปรากฏการณ์นี้สามารถแก้ไขได้โดยการเขียนโปรแกรมเพื่อใช้เลือกเส้นทางที่เหมาะสมให้กับ MPTCP ก่อนหน้านี้เราจึงได้เสนอ Path Selection Algorithm หรือ PSA [4] ขึ้นมาพัฒนากระบวนการเลือกเส้นทางของ MPTCP ในเครือข่ายที่กำหนดโดยซอฟต์แวร์ หรือ Software-Defined Networking (SDN) [5] เพื่อแก้ไขปัญหาดังกล่าว

ในงานวิจัยนี้เราได้นำกระบวนการเลือกเส้นทางที่ได้เสนอมาทดลองในเครือข่ายที่ใหญ่ขึ้นโดยเป็นเครือข่ายที่ออกแบบมาจากเครือข่ายจริง ซึ่งเราได้ทำการศึกษาบน Mininet Network Emulator [6] และ Floodlight Controller [7] และใช้โปรแกรม Iperf ในการทดสอบส่งข้อมูล และวัดผลจากค่า Throughput ที่ได้การเขียนโปรแกรมดึงผลการส่งข้อมูลที่ถูกบันทึกจากโปรแกรม Wireshark โดยระบบเครือข่ายถูกควบคุมการทำงานโดยใช้อุปกรณ์ควบคุม (Controller) ผ่านโพรโทคอล OpenFlow [8] ซึ่งกระบวนการทำงานของ Controller นี้ จะศึกษาคุณลักษณ์ของแพ็คเก็ต MPTCP Header พร้อมทั้งใช้ข้อมูลจากค่าแบนด์วิดท์ที่พร้อมใช้งาน (Available Bandwidth) ในขณะทำการส่งข้อมูล และค่าความหน่วง (Delay) ระหว่างต้นทางและปลายทาง เพื่อเลือกเส้นทางที่ดีที่สุดให้แต่ละ Subflow

2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 MPTCP

MPTCP เป็นโปรโตคอลทำงานอยู่ในชั้นทรานสปอร์ต (Transport Layer) เช่นเดียวกับ TCP ซึ่งจะเป็นตัวกลางและผสานการทำงานระหว่างชั้นแอปพลิเคชัน (Application Layer) และชั้นของ Internet Protocol โดย MPTCP จะทำการแบ่ง TCP ปกติออกเป็นหลาย TCP ย่อย หรือที่เรียกว่า Subflow ด้วยเหตุนี้ MPTCP จึงสามารถใช้เส้นทางได้หลายเส้นทางภายใต้การเชื่อมต่อเดียว (Single connection) และยังสามารถใช้งานได้กับแอปพลิเคชันเดิมได้โดยไม่ต้องทำการเปลี่ยนแปลง

MPTCP มีส่วนที่ใช้ในการควบคุมการส่งข้อมูลในแต่ละ Subflow ที่เรียกว่า MPTCP Scheduler เพื่อแบ่งและควบคุมการส่งไปในแต่ละ Subflow ให้มีประสิทธิภาพ MPTCP Scheduler ที่เป็นค่าเริ่มต้น (Default) มีชื่อเรียกว่า Lowest RTT First จะมีลักษณะการทำงานคือส่งข้อมูลใน Subflow ที่มีค่า Round Trip Time (RTT) ต่ำสุดซึ่งจะส่งข้อมูลไปจนกว่าจะเต็ม Congestion Window (CWND) หลังจากนั้นจึงจะส่งข้อมูลใน Subflow อื่นที่มีค่า RTT ต่ำรองลงมา

ใน TCP ปกติ จะมี Sequence numbers ที่ระบุลำดับของข้อมูลแต่ละแพ็คเก็ตที่ถูกส่งมาให้ฝั่งผู้รับได้จัดเรียงลำดับก่อนส่งไปหา Application Layer ได้ถูกต้อง แต่เมื่อข้อมูลถูกแบ่งเป็น Subflow ย่อย ซึ่งแต่ละ Subflow จะมี Sequence Numbers เป็นของตัวเองนั้น เพื่อให้ฝั่งผู้รับยังคงสามารถจัดเรียงลำดับข้อมูลได้ถูกต้องนั้น ในการส่ง ข้อมูลที่เป็น MPTCP จะส่งข้อมูลในส่วนของ Data Sequence Signal หรือ DSS ลงไปในส่วน TCP Option Header ด้วย ซึ่งแสดงในภาพที่ 1

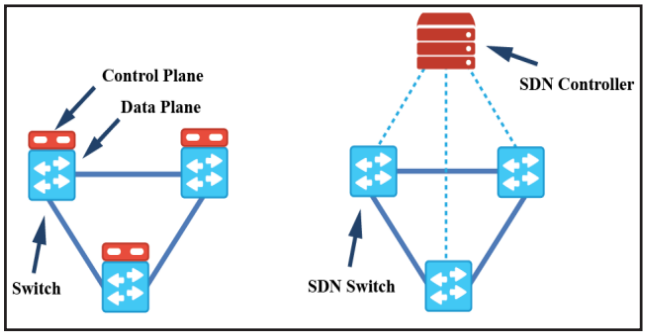
Bit offset	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
0	Kind	Length	Subtype	(reserved) F m M a A
32	Data ACK			
64	Data sequence number			
96	Subflow Sequence Number			
128	Data-Level Length		Checksum	

ภาพที่ 1 *ตัวอย่าง Option Header ที่เป็นแบบ DSS*

โดย DSS จะประกอบด้วย Sequence Numbers ของข้อมูลก่อนที่จะถูกแยกเป็น Subflow ย่อย

2.2 เครือข่ายที่กำหนดโดยซอฟต์แวร์

เครือข่ายที่กำหนดโดยซอฟต์แวร์ (Software-Defined Networking) หรือ SDN เป็นเครือข่ายแบบใหม่ที่สามารถควบคุมการทำงานของเครือข่ายได้อย่างมีประสิทธิภาพ โดยใช้อุปกรณ์ควบคุมหรือ Controller เป็นส่วนควบคุมและทำงานของอุปกรณ์ในระบบเครือข่ายซึ่งแสดงในภาพที่ 2 โดยรูปซ้ายมือเป็นส่วนของโครงสร้างเครือข่ายแบบเดิมที่มีส่วนควบคุมอยู่ภายในอุปกรณ์แต่ละตัว ส่วนรูปทางขวามือเป็นโครงสร้างตามแบบ SDN ซึ่งจะรวมส่วนควบคุมไว้ภายนอกแล้วสามารถควบคุมทุกอุปกรณ์ได้

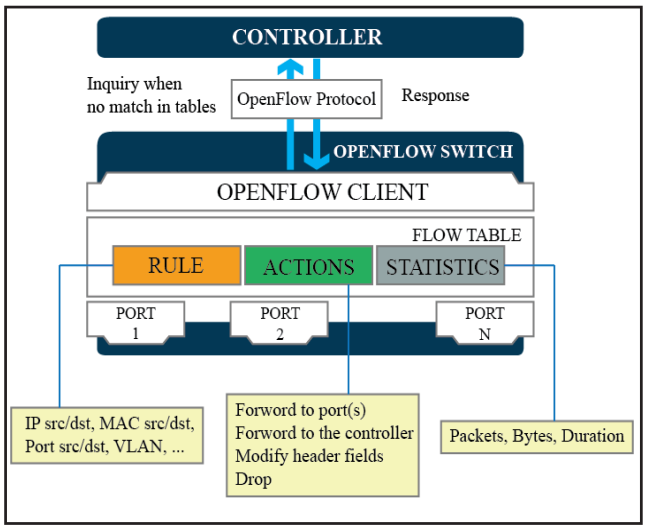


ภาพที่ 2 *ภาพโครงสร้างเครือข่ายแบบเดิมเปรียบเทียบกับเครือข่ายแบบ SDN*

ในปัจจุบันโพรโทคอลที่นิยมใช้ในเครือข่าย SDN คือ OpenFlow Protocol ซึ่งถูกออกแบบมาจากหลักการของ 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 3 จะแสดงโครงสร้างการทำงานของ OpenFlow Protocol

2.3 Floodlight Controller

ปัจจุบันมี Controller ให้เลือกใช้งานอยู่หลากหลาย ซึ่งในงานวิจัยนี้เลือกใช้ Controller ที่มีชื่อว่า Floodlight ซึ่งเป็น Controller ที่สร้างขึ้นมาจากภาษา Java การเขียนโปรแกรมควบคุมการทำงานของ Controller ทำได้สองวิธีคือ



ภาพที่ 3 *การสื่อสารระหว่าง Controller และ OpenFlow Switch ผ่าน OpenFlow Protocol*

เขียน Java Module และ REST API โดยงานวิจัยนี้ต้องเขียนโปรแกรมด้วย Java Module เพื่อจัดการกับ PACKET_IN 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

2.4 งานวิจัยที่เกี่ยวข้อง

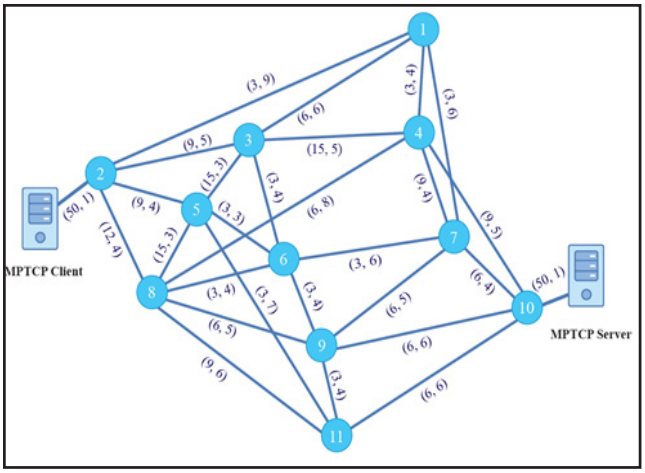
หลังจาก Paasch และคณะ [9] ได้ทำการพัฒนา MPTCP เพื่อใช้งานจริงใน Linux Kernel ซึ่งได้รับความสนใจอย่างมาก ต่อมา Pol และคณะ [10] จึงได้ศึกษาวิธีการเพิ่ม Throughput ในการจัดส่งข้อมูล MPTCP โดยกำหนดหมายเลข VLAN เพื่อเลือกเส้นทางในการลำเลียงข้อมูลผ่านทาง Subflow ที่มีความสามารถในการส่งต่อข้อมูลได้เยอะที่สุดระหว่างเครื่องต้นทางและปลายทาง และประยุกต์ใช้เครือข่าย SDN เพื่อควบคุมการลำเลียงข้อมูลภายในเครือข่ายให้เป็นไปอย่างถูกต้องอีกทั้งยังเป็นการป้องกันการทับซ้อนกันของเส้นทางที่ใช้ขณะลำเลียงข้อมูล อย่างไรก็ตามข้อจำกัดของกระบวนการนี้ คือต้องทำการกำหนดหมายเลข VLAN ก่อนส่งข้อมูลจึงจะสามารถใช้งานได้

ในภายหลัง Sandri และคณะ [11] และ Zannettou และคณะ [12] จึงได้ศึกษาวิธีการใหม่เพื่อลดข้อจำกัดของการตั้งหมายเลข VLAN โดยเปลี่ยนเป็นใช้ Controller เพื่อแยกแยะและระบุ Subflow แทน ซึ่งวิธีการนี้ก็สามารถเพิ่ม Throughput ในการจัดส่งข้อมูลได้เช่นกัน เนื่องจากแพ็คเก็ต MPTCP Header สามารถจำแนก Subflow แต่ละ Subflow ได้ ทำให้ Controller สามารถเลือกส่งแพ็คเก็ตไปได้ในเส้นทางที่แตกต่างกัน อย่างไรก็ตามงานวิจัยทั้งสองนี้ยังมีข้อจำกัด คือ Controller ไม่สามารถจำแนก Subflow จากหลายการเชื่อมต่อที่ส่งมาต้นกำเนิดเดียวกันในระยะเวลาใกล้เคียงกัน อีกทั้งการเลือกเส้นทางในการลำเลียงข้อมูลยังคงเป็นการเลือกเส้นทางแบบสั้นที่สุดเท่านั้น แม้ว่าจะมีเส้นทางอื่นที่มีประสิทธิภาพมากกว่าก็ตาม

ก่อนหน้านี้เราได้เสนอ Path Selection Algorithm หรือ PSA [4] ขึ้นมาที่มีกระบวนการระบุ Subflow ที่ถูกต้องแม่นยำ และกระบวนการเลือกเส้นทางที่ใช้ข้อมูลแบนด์วิดท์ที่พร้อมใช้งานและค่าความหน่วง เพื่อประมาณค่า Throughput ในการเลือกคู่เส้นทางที่จะได้ค่า Throughput สูงสุดจากเส้นทางทั้งหมดในเครือข่ายและได้ทดลองในเครือข่ายแบบอย่างง่ายที่มีลักษณะคล้ายท่อนเหล็กยกน้ำหนัก (Simple Dumbbell Topology)

3. วิธีดำเนินการวิจัย

งานวิจัยนี้ได้ใช้ Mininet Network Emulator 2.2.1 [6], Ubuntu 14.04.4 LTS, MPTCP: Stable release v0.91.2 [9] และ Iperf 2.0.8 [13] การทดลองสามารถใช้ 1 ส่วนต่อขยายของเครื่องส่งและเครื่องรับได้นั้นอันเนื่องด้วย MPTCP



ภาพที่ 4 *โครงสร้างเครือข่ายที่ใช้ในการทดลอง*

เวอร์ชั่น 0.90 ขึ้นไปสามารถกำหนดให้สามารถส่งได้มากกว่า 1 Subflow ใน 1 คู่ของ IP Address

การทดลองทำบนโครงสร้างเครือข่ายตามภาพที่ 4 ซึ่งเป็นโครงสร้างเครือข่ายที่ได้นำโครงสร้าง ค่าแบนด์วิดท์ ค่าความหน่วง มาจากเครือข่าย COST 239 [14] ซึ่งเป็นเครือข่ายจริงจากความร่วมมือของกลุ่มประเทศในทวีปยุโรป โดยตารางที่ 1 จะระบุชื่อเมืองของที่ตั้งในแต่ละ Node ตัวเลขในวงเล็บของภาพที่ 4 คือค่าแบนด์วิดท์ในหน่วย เมกะบิตต่อวินาที (Mbps) และค่าความหน่วงในหน่วยมิลลิวินาที (ms)

ตารางที่ 1 *ที่ตั้งของแต่ละ Node*

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

โดยการทดลองจะออกแบบเป็นสองรูปแบบคือ รูปแบบแรก Controller ทำการเลือกเส้นทางแบบสั้นที่สุด (Shortest Path) โดยแต่ละเส้นทางในหนึ่งการเชื่อมต่อจะถูกเลือกไม่ให้ใช้เส้นทางซ้ำกัน อันเนื่องมาจากค่าเริ่มต้นของ Controller ที่เลือกเส้นทางแบบเส้นที่สุดนั้น และแบบที่สองคือ Controller ทำการเลือกเส้นทางโดยใช้ Path Selection Algorithm [4] โดยมีการแก้ไขกระบวนการเลือกเส้นทางเพิ่มเติมตามภาพที่ 5 ซึ่งจะเพิ่มเติมในส่วนของค่าคงที่ k (บรรทัดที่ 1) ซึ่งจะใช้เป็นค่าสูงสุดของจำนวนเส้นทางที่จะเอามาเปรียบเทียบ เนื่องจากกระบวนการเลือกเส้นทางที่เราจะทำการเปรียบเทียบเส้นทางทั้งหมดที่เป็นไปได้ เมื่อเครือข่ายมีขนาดใหญ่ทำให้เส้นทางที่เป็นไปได้มีจำนวนมาก หากไม่มีการจำกัดจำนวนเส้นทางจะมีผลทำให้สิ้นเปลืองการทำงานที่อาจ ไม่จำเป็น ในการทดลองนี้ได้กำหนดค่า k = 30 (ได้จากการทดลองปรับเปลี่ยนค่าให้เหมาะสมกับการทดลองนี้) ในการกำหนดค่า k ที่มีค่าน้อยอาจจะมีผลทำให้เส้นทางที่ดีที่สุดไม่ถูกพิจารณา เพราะอาจจะเป็นเส้นทางที่ Dijkstra's algorithm ยังไม่พบ

```
Input:   Source and Destination switches
         First path or Null
Output:  OutputPath
1:  k ← Maximum number of paths from Routing module.
2:  availablePaths ← Dijkstra's algorithm
                     from Routing module function
                     (Maximum number of paths = k)
3:  set i = 1
4:  While i<= k and i <= number of availablePaths
5:     tempPath ← Next path from availablePaths list
6:     if First path is Null or tempPath not using same switch
          with First path then
7:        estBW ← estimated Bandwidth from tempPath
                  using Multiple Linear Regression Equation
8:        if estBW > BestEstBW then
9:           BestPath ← tempPath
10:          BestEstBW ← estBW
11:       end if
12:  end if
13: end
14: OutputPath ← BestPath
```

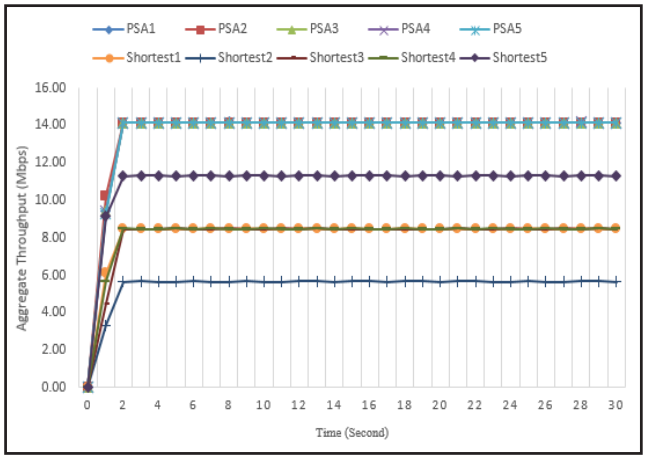
ภาพที่ 5 *โค้ดเทียมของอัลกอริทึมการเลือกเส้นทาง*

เส้นทางนั้น หากทำการกำหนดค่า k ที่มีค่าสูงเกินไปจะทำให้ Dijkstra's algorithm ใช้เวลาเพิ่มขึ้นในการหาเส้นทาง และกระบวนการเลือกเส้นทางต้องคำนวณค่าแบนด์วิดท์และความหน่วงของจำนวนเส้นทางที่มากมาเปรียบเทียบกันด้วย แต่ละรูปแบบการทดลองจะทดลอง 5 ครั้ง ในแต่ละครั้งจะใช้โปรแกรม Wireshark ดักจับข้อมูลที่ Network Card ของเครื่อง MPTCP Server และใช้โปรแกรม Iperf สร้างข้อมูลที่เป็น TCP ส่งข้อมูลจากเครื่อง MPTCP Client ไปยังฝั่ง MPTCP Server ซึ่งแต่ละการทดลองสามารถบันทึกค่า Throughput ได้การเขียนโปรแกรมอ่านข้อมูลที่ส่งที่บันทึกไว้ด้วยโปรแกรม Wireshark

4. ผลการดำเนินงาน

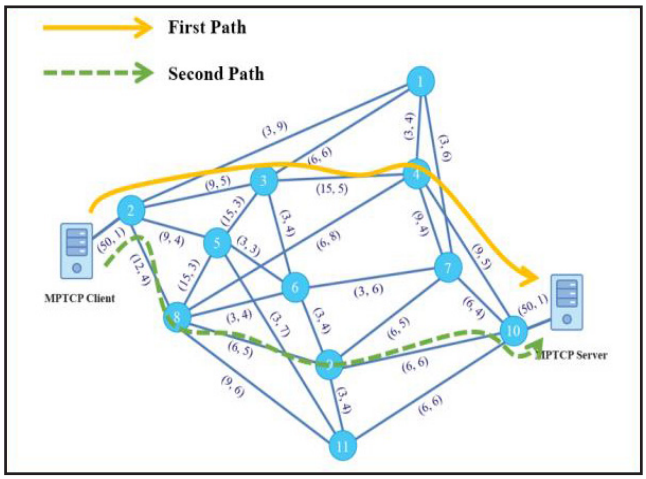
ผลจากการทดลองสามารถแสดงในภาพที่ 6 ซึ่งแสดงผล Throughput รวมจากแต่ละเส้นทาง ที่ได้จากการคำนวณเส้นทางโดยใช้ Path Selection Algorithm หรือ PSA จำนวน 5 การทดลอง และผล Throughput ที่ได้จากการคำนวณเส้นทางโดยใช้ Shortest Path Algorithm หรือ Shortest จำนวน 5 การทดลอง

จากผลพบว่าผล Throughput ที่ได้จากการคำนวณเส้นทางโดยใช้ Path Selection Algorithm จะสามารถ เลือกเส้นทางที่ทำให้ได้ค่า Throughput ที่สูงที่สุด และเลือกเส้นทางเดียวกันทั้ง 5 การทดลอง ตามภาพที่ 7 (ก) ทำให้ผล Throughput ในภาพที่ 6 เกาะกลุ่มอยู่ที่ 14 Mbps ส่วนผล

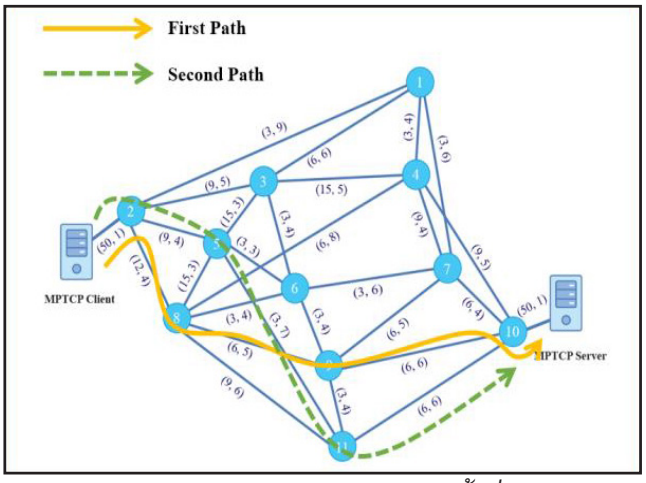


ภาพที่ 6 *ค่า Throughput จากการทดลอง*

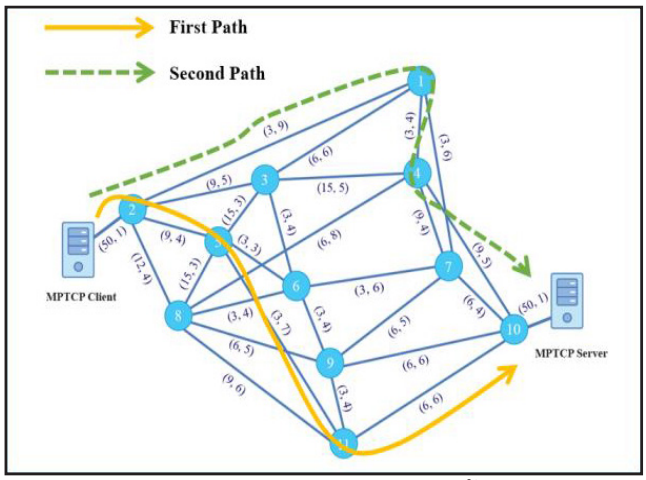
Throughput ที่ได้จากการใช้ Shortest Path Algorithm จะไม่เกาะกลุ่มกันตามภาพที่ 7 (ก) ถึง ภาพที่ 7 (ฉ) และมีค่าเฉลี่ยจาก 5 การทดลองอยู่ที่ 8.4 Mbps อันเนื่องมาจากการเลือกเส้นทางที่สั้นที่สุดที่มีหลายเส้นทางทำให้มีลักษณะคล้ายกับการสุ่มเลือกเส้นทาง



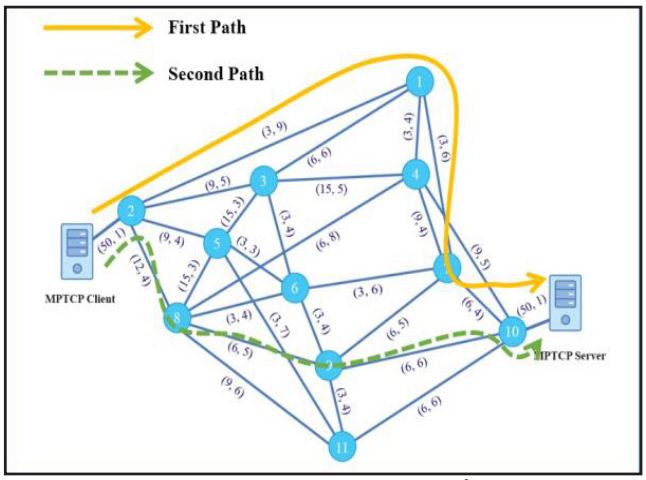
(ก) การทดลองแบบ PSA



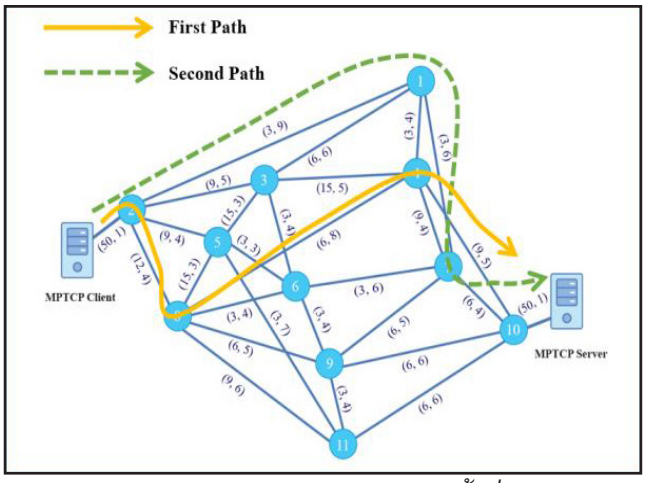
(ข) การทดลองแบบ Shortest ครั้งที่ 1



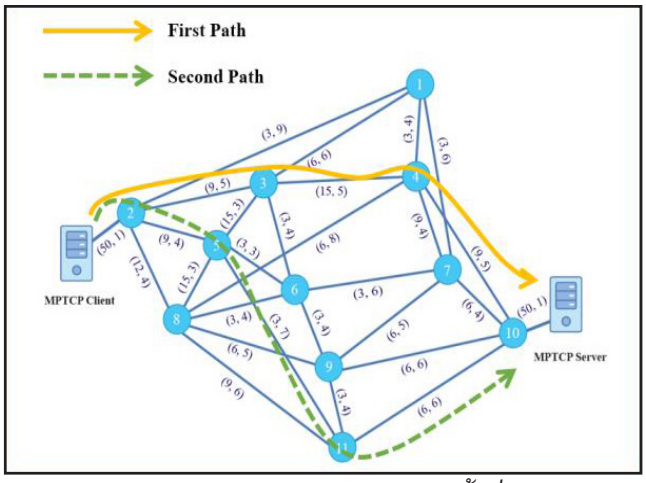
(ค) การทดลองแบบ Shortest ครั้งที่ 2



(ง) การทดลองแบบ Shortest ครั้งที่ 3



(จ) การทดลองแบบ Shortest ครั้งที่ 4



(ฉ) การทดลองแบบ Shortest ครั้งที่ 5

ภาพที่ 7 *เส้นทางในการส่งข้อมูล*

5. สรุป

งานวิจัยนี้แสดงให้เห็นว่ากระบวนการเลือกเส้นทางที่คำนวณค่า Throughput ไว้ก่อนที่จะส่งข้อมูล โดยอัลกอริทึมที่สามารถระบุ Subflow แยกออกกันได้อย่างถูกต้องจากข้อมูลในแพ็คเก็ต และการพิจารณาเลือกเส้นทางจากการพิจารณาค่าตัวแปรที่มีผลกับค่า Throughput ของการส่งข้อมูลแบบ MPTCP ซึ่งค่าความสำคัญของแต่ละตัวแปรถูกคำนวณจากกระบวนการทำ Multiple Linear Regression นั้นสามารถนำมาใช้กับเครือข่ายที่มีขนาดใหญ่และมีความซับซ้อนขึ้นได้ โดยสามารถทำให้ค่าเฉลี่ยของ Throughput ดีขึ้นร้อยละ 66.6 เมื่อเปรียบเทียบกับการเลือกเส้นทางแบบสั้นที่สุด

6. เอกสารอ้างอิง

[1] J. Postel. *Transmission Control Protocol.* Available Online at https://tools.ietf.org/html/rfc793, accessed on 1981.

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

[3] *Use Multipath TCP to create backup connections for iOS*. Available Online at https://support.apple.com/en-hk/HT201373, accessed on 2016.

[4] J. Jullawat, K. Kawila, K. N. Nakorn, and K. Rojviboonchai. “Path Selection Algorithm for

Multipath TCP in Software-Defined Networking (NCCIT2017 Best Paper Awards)." *In The 13th National Conference on Computing and Information Technology (NCCIT)*, pp. 100-105, 2017.

[5] 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.

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

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

[8] N. McKeown et al., "OpenFlow: enabling innovation in campus networks." *SIGCOMM Computer Communication Review*, Vol. 38, No. 2, pp. 69-74, 2008.

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

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

[11] 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*, pp. 9-16, 2015.

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

[13] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. *Iperf*. Available Online at https://sourceforge.net/projects/iperf/, accessed on 2010.

[14] 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*, Vol. 4, pp. 5, 2005.