การสร้างส่วนต่อประสานผู้ใช้โดยใช้เว็บเพื่อป้องกันความล้มเหลวในระบบเอสดีเอน

IMPLEMENTATION OF WEB-BASED USER INTERFACE FOR FAILURE PROTECTION IN SDN NETWORK

อุกฤษฎ์ สีสมาน

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรบัณฑิต (ฟิสิกส์ประยุกต์)
ภาควิชาฟิสิกส์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2559

การสร้างส่วนต่อประสานผู้ใช้โดยใช้เว็บเพื่อป้องกันความล้มเหลวในระบบเอสดีเอน

IMPLEMENTATION OF WEB-BASED USER INTERFACE FOR FAILURE PROTECTION IN SDN NETWORK

อุกฤษฎ์ สีสมาน

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรบัณฑิต (ฟิสิกส์ประยุกต์)
ภาควิชาฟิสิกส์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2559

IMPLEMENTATION OF WEB-BASED USER INTERFACE FOR FAILURE PROTECTION IN SDN NETWORK

UKRIST SRISAMARN

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF SCIENCE (APPLIED PHYSICS)

DEPARTMENT OF PHYSICS, FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2016

หัวข้อโครงงานพิเศษ	การสร้างส่วนต่อประสานผู้ใช้โดยใช้เว็บเพื่อป้องกันความล้มเหลวในระบบเอสดีเอน Implementation of Web-based User Interface for Failure Protection in SDN Network
ชื่อนักศึกษา	นายอุกฤษฎ์ สีสมาน รหัสนักศึกษา 56051253
ปริญญา	วิทยาศาสตรบัณฑิต (ฟิสิกส์ประยุกต์)
ภาควิชา	ฟิสิกส์
ปีการศึกษา	2559
อาจารย์ที่ปรึกษา	ดร.วิฑูรย์ ยินดีสุข
อาจารย์ที่ปรึกษาร่วม	ผศ.ดร.ณัฐพงศ์ กิจสุวรรณ

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.) อนุมัติให้โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต(ฟิสิกส์ประยุกต์) ประจำปีการศึกษา 2559

คณะกรรมการสอบ	ลายมือชื่อ
ผศ.ดร.ประธาน บุรณศิริ ประธานกรรมการ	
อ.สุรชาติ กมลดิลก กรรมการ	
อ.ธรรมรัตน์ แต่งตั้ง กรรมการ	
ดร.วิฑูรย์ ยินดีสุข กรรมการและอาจารย์ที่ปรึกษา	

หัวข้อโครงงานพิเศษ การสร้างส่วนต่อประสานผู้ใช้โดยใช้เว็บเพื่อป้องกันความล้มเหลวในระบบเอสดีเอน

ชื่อนักศึกษา นายอุกฤษฏ์ สีสมาน รหัสนักศึกษา 56051253

ปริญญา วิทยาศาสตรบัณฑิต (ฟิสิกส์ประยุกต์)

ภาควิชา ฟิสิกส์

คณะ วิทยาศาสตร์

มหาวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.)

ปีการศึกษา 2559

อาจารย์ที่ปรึกษา ดร.วิฑูรย์ ยินดีสุข

อาจารย์ที่ปรึกษาร่วม ผศ. ดร.ณัฐพงศ์ กิจสุวรรณ

บทคัดย่อ

ระบบ Software defined network เป็นระบบที่ใช้ OpenFlow controller ที่อยู่บน Control plane ควบคุม OpenFlow switches ที่อยู่บน data forwarding plane ในการส่ง Packets โดย OpenFlow controller ติดต่อกับ OpenFlow switch ผ่าน OpenFlow protocols ในปัจจุบันมีผู้ใช้ระบบเน็ตเวิร์คที่มากขึ้นเรื่อยๆ ปัญหาความล้มเหลวที่เกิดขึ้นบนระบบเน็ตเวิร์คจึงมีความสำคัญมาก งานวิจัยนี้ได้ใช้วิธี Protection design เพื่อแก้ไขปัญหาความล้มเหลวของระบบเน็ตเวิร์ค เส้นทางสำรองได้ถูกเตรียมเอาไว้ เมื่อเกิดความล้มเหลวบนเส้นทางหลักที่ใช้ส่ง Packets จากต้นทางไปยังปลายทาง OpenFlow switch ที่ติดกับลิงค์ที่เกิดความล้มเหลวทำการส่งสถานะของ port ไปยัง OpenFlow controller เพื่อทำการประมวลผลและส่งคำสั่งไปยัง switch ต้นทางเพื่อส่ง Packets บนเส้นทางสำรอง โดย Switch-over time น้อยกว่า 50 มิลลิวินาที เมื่อเทียบกับระบบทั่วไป จะต้องใช้เวลาในการแก้ปัญหาที่นาน และซับซ้อนกว่า ในงานวิจัยนี้ได้ทำการทดลองบนระบบจำลอง Mininet

คำสำคัญ : Software defined network, OpenFlow controller, OpenFlow switch, OpenFlow protocols, Protection design, Switch-over time

Title	Implementation of Web-Based User Interface for Failure Protection in SDN Network
Students	Mr.Ukrist Srisamarn Student ID 56051253
Degree	Bachelor of Science (Applied Physics)
Department	Physics
Faculty	Science
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)
Academic Year	2016
Advisor	Dr.Witoon Yindeesuk
Co-advisor	Asst. Prof. Dr.Nattapong Kitsuwan

Abstract

Software defined netwok (SDN) uses OpenFlow (OF) controller on the control plane to control the OF swtiches on the data forwarding plane which forward the packets. The OF controller contact with OF switch via the OF protocols. Nowadays, serveral people use the network and there is a growing tendency. The failure that occur on the network problem is extreamly important. In this paper uses the protection design for solve failure on the network. The backup path is prepared when failure occurs on the primary path which uses to forward the packets from source to destination. OF switch which close up the link failure sends the port status to the OF controller for processing. The OF controller get the command back to the source OF switch for forwarding the packets on the backup path with switch-over time least than 50 ms. If we compare this process with general process, It will use a lot of time to solve this problem and very complex. In this paper, experiments were conducted on the Mininet simulation system.

Keywords : Software defined network, OpenFlow controller, OpenFlow switch, OpenFlow protocols, Protection design, Switch-over time

กิตติกรรมประกาศ

โครงงานพิเศษฉบับนี้สำเร็จลุลวงได้ด้วยดีเนื่องจากได้รับความกรุณาอย่างสูงจาก ดร.วิฑูรย์ ยินดีสุข อาจารย์ที่ปรึกษาโครงงานพิเศษและ Asst. Prof. Dr.Nattapong Kitsuwan ที่ได้ให้ความช่วยเหลือ ให้คำแนะนำและตรวจสอบแก้ไขการดำเนินการจัดโครงงานพิเศษ ผู้วิจัยมีความซาบซึ้งและเป็นพระคุณเป็นอย่างสูง ณ โอกาสนี้

ขอบพระคุณอาจารย์ในสาขาฟิสิกส์ทุกท่าน ที่ได้กรุณาให้ความรู้ ให้คำแนะนำ และให้ความคิดเห็นเป็นอย่างดีเสมอมา ทำให้ได้ข้อมูลครบถ้วนในงานวิจัยครั้งนี้

คุณงามความดีอันพึงมีจากโครงงานพิเศษฉบับนี้ ผู้วิจัยขอมอบแด่บิดา มารดา อันเป็นที่เคารพยิ่งและคณาอาจารย์ผู้ประสาทวิชาความรู้ตลอดจนทุกๆท่านที่ให้กำลังใจช่วยเหลือจนกระทั่งโครงงานพิเศษฉบับนี้จบลงด้วยดี

อุกฤษฏ์ สีสมาน

สารบัญ

สารบัญ (ต่อ)

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

คำย่อและสัญลักษณ์

คำย่อ/สัญลักษณ์	คำอธิบาย
OF controller	OpenFlow controller
OF switch	OpenFlow switch
SDN	Software Defined Network

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันได้มีการพัฒนาในหลากหลายด้าน ซึ่งในด้านการสื่อสารนั้นก็เป็นสิ่งที่โดดเด่นอย่างมาก และยังก้าวต่อไปโดยไม่มีที่สิ้นสุด ในเรื่องของการสื่อสารสิ่งที่สำคัญคือผู้สื่อสาร กับผู้รับสารจะต้องได้รับข้อมูลที่ครบถ้วนและรวดเร็ว

การส่งข้อมูลบนระบบเน็ตเวิร์คนั้นถ้าหากเกิดความล้มเหลวขึ้นบนเส้นทางที่ใช้ส่งข้อมูลอยู่จะทำให้เกิดการชะงักหรืออาจทำให้ไม่สามารถส่งข้อมูลต่อไปได้จนกว่าจะมีการแก้ไข ดังนั้นในงานวิจัยนี้ได้ทำการสร้างการป้องกันการเกิดความล้มเหลวขึ้นบนระบบเน็ตเวิร์ค โดยใช้วิธีการ Protection Design ซึ่งมีการสร้างเส้นทางหลักและทำการสร้างเส้นทางสำรองเพื่อใช้ป้องกันความล้มเหลวที่จะเกิดขึ้น โดย Protection Design ที่ใช้ในงานวิจัยนี้เป็นการสร้างแบบ Disjoint path กล่าวคือ เป็นการสร้างเส้นทางหลักและเส้นทางสำรองที่ไม่มีเส้นทางเกี่ยวข้องกัน

ในปัจจุบันมีการนำระบบซอฟแวร์เข้ามาผนึกรวมในระบบการสื่อสารที่มีชื่อว่า Software-defined network (SDN) อุปกรณ์ทางด้านเน็ตเวิร์คในปัจจุบันมี control plane ในตัวเองมีการตัดสินใจโดยประมวลผลจากสถานะที่ได้รับมาจากโหนดข้างเคียงการตัดสินใจนี้เป็นการตัดสินใจที่ดีในระดับหนึ่งแต่ไม่ดีที่สุดเนื่องจากไม่ทราบข้อมูลของทั้งเน็ตเวิร์ค อีกจุดประสงค์หนึ่งของ SDN คือเพื่อความไม่ขึ้นกับผู้ผลิตไม่ว่าจะใช้อุปกรณ์ของผู้ผลิตใดก็ใช้ซอฟแวร์เดียวกันในการควบคุมได้

งานวิจัยนี้ได้นำการป้องกันความล้มเหลวที่เกิดขึ้นบนระบบเน็ตเวิร์คซึ่งใช้ในระบบ SDN มี controller ส่วนกลางที่คอยควบคุมการไหลของข้อมูล ในงานวิจัยนี้ผู้วิจัยได้สร้างระบบจากการเขียนโปรแกรมคำสั่งใน controller เพื่อให้สร้างเส้นทางหลักและสร้างเส้นทางสำรองเอาไว้เมื่อมีความล้มเหลวเกิดขึ้นบนระบบเน็ตเวิร์คมีการคำนวณให้เปลี่ยนเส้นทางทันทีโดยอัตโนมัติและเมื่อเส้นทางหลักพร้อมใช้งานระบบจะทำการเปลี่ยนเส้นทางกลับมาใช้เส้นทางหลักทันทีโดยอัตโนมัติ อีกทั้งยังเอื้ออำนวยความสะดวกต่อผู้ควบคุมระบบเน็ตเวิร์ค เนื่องจากในงานวิจัยนี้ได้สร้างเว็บเพจขึ้นมาเพื่อเชื่อมต่อระหว่างผู้ควบคุมระบบกับระบบเน็ตเวิร์ค ทำให้มีความสะดวกและรวดเร็วในการร้องขอเส้นทางเพื่อใช้งานได้ง่าย

1.2 วัตถุประสงค์ของงานวิจัย

โครงงานพิเศษนี้มีวัตถุประสงค์เพื่อการแก้ปัญหาของระบบเน็ตเวิร์คเมื่อเกิดความล้มเหลวขึ้นบนเส้นทางหลัก และเพื่อให้มีการเข้าถึงการใช้งานได้ง่าย

1) เมื่อเกิดความล้มเหลวของเส้นทางในการส่งข้อมูลสามารถแก้ไขปัญหาได้ทันทีโดยอัตโนมัติ

2) ทำให้กระบวนการสร้างการติดต่อทางเน็ตเวิร์คให้สะดวกที่สุด

1.3 ขอบเขตของงานวิจัย

1) สร้างระบบป้องกันความล้มเหลวในการสื่อสารที่ใช้เวลาน้อยกว่า 50 มิลลิวินาที
2) ระบบมีความง่ายต่อการใช้งาน เอื้อประโยชน์ต่อผู้ควบคุมระบบ
3) เอาระบบ Software-defined network มาใช้เพื่อความสะดวกในการแก้ไขความล้มเหลวบนระบบเน็ตเวิร์ค

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) อำนวยความสะดวกต่อผู้ควบคุมระบบในการกำหนดค่าเส้นทางเพื่อป้องกันความล้มเหลวที่จะเกิดขึ้นบนระบบเน็ตเวิร์ค
2) มีการแก้ปัญหาเมื่อเกิดความล้มเหลวขึ้นบนระบบเน็ตเวิร์คทันทีโดยอัตโนมัติ

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในระบบเน็ตเวิร์คการป้องกันความล้มเหลวที่เกิดขึ้นบนระบบเป็นสิ่งที่สำคัญอย่างมากและในปัจจุบันมีระบบเน็ตเวิร์คที่เรียกว่า Software-defined network (SDN) ซึ่งระบบนี้จะมีส่วนสำคัญอยู่ 3 ส่วน คือ

1. Controller ทำหน้าที่ควบคุมสวิตช์บนระบบเน็ตเวิร์ค
2. สวิตช์ ทำหน้าที่ส่ง Packet บนระบบเน็ตเวิร์ค
3. OpenFlow Protocol ทำหน้าที่สร้างการติดต่อระหว่าง Controller กับสวิตช์

โดยระบบนี้มีข้อดีก็คือสามารถทำการกำหนดค่าหรือคำสั่งบนสวิตช์โดยใช้ Controller แทนที่จะใช้คนได้ และมีประสิทธิภาพสูง รวดเร็วต่อการสร้างการติดต่อ จากความโดดเด่นดังกล่าวนี้ในงานวิจัยนี้จึงนำมาประยุกต์ใช้เพื่อป้องกันความล้มเหลวบนระบบเน็ตเวิร์คโดยใช้วิธีการแบบ protection design กล่าวคือเมื่อเกิดความล้มเหลวบนเส้นทางหลักระบบ SDN จะต้องทำการเปลี่ยนเส้นทางที่ใช้ส่งข้อมูลไปเป็นเส้นทางสำรองเพื่อส่งข้อมูลในทันที แล้วเมื่อเส้นทางหลักพร้อมใช้งาน ระบบจะต้องทำการเปลี่ยนมาใช้เส้นทางหลักดังเดิม โดยระยะทางของเส้นทางหลักจะต้องเป็นระยะทางที่ดีที่สุดบนระบบเน็ตเวิร์คโดยใช้วิธีการคำนวณเส้นทางที่ดีที่สุดโดยงานวิจัยนี้ใช้ Dijkstra's Algorithm ในการคำนวณเส้นทางที่ดีที่สุด

2.1 ระบบ Software-defined network (SDN)

1. Software-defined network (SDN) เป็นสิ่งที่น่าสนใจมากเนื่องจากมันยอมให้ผู้ควบคุมระบบเน็ตเวิร์คควบคุมความต้องการของระบบเน็ตเวิร์คโดย SDN แยกระบบเน็ตเวิร์คออกเป็น Control Plane และ Data forwarding plane ออกจากกันสิ่งนี้มันยอมให้ ผู้ควบคุมระบบเน็ตเวิร์คและวิศวกรระบบเน็ตเวิร์ค ให้คำสั่งได้อย่างรวดเร็วเพื่อทำการเปลี่ยนแปลงสิ่งที่ระบบเน็ตเวิร์คต้องการจากใจกลาง Controller ผู้ควบคุมระบบเน็ตเวิร์คสามารถใช้ประโยชน์ของความสามารถทางด้านโปรแกรมของระบบเน็ตเวิร์คที่ถูกพัฒนาที่มีการควบคุมสวิตช์ทางกายภาพจาก Controller ได้อย่างยืดหยุ่น ซึ่ง Controller ทำการควบคุมอย่างชาญฉลาดและมีการจัดการทางด้านซอฟแวร์อีกด้วย [1]

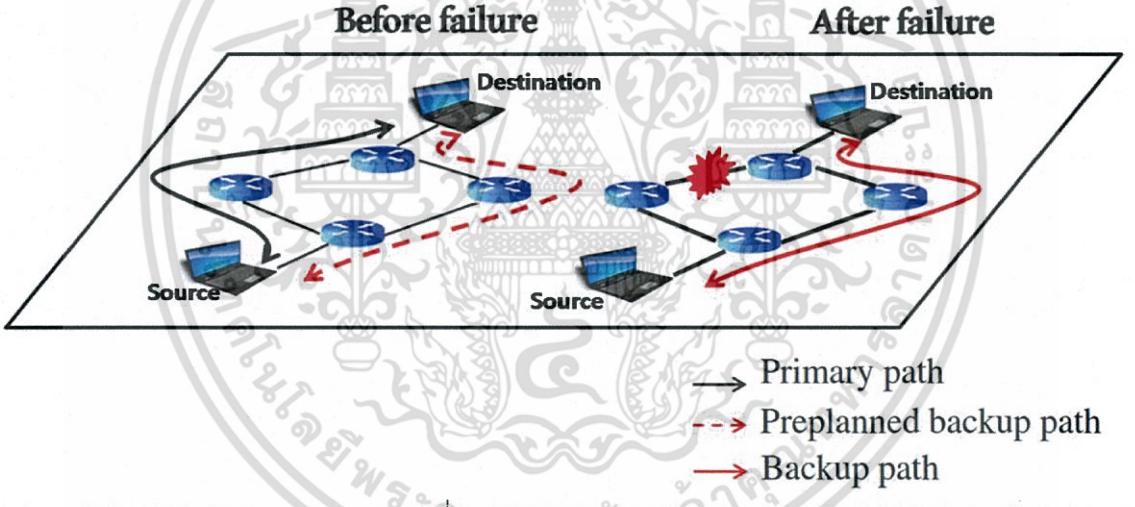
2. OpenFlow คือ โปรโตคอลที่เป็นที่รู้จักกันอย่างกว้างขวาง (South-bound interface) สำหรับ SDN networks ที่ยอมให้ Controller กระทำกับ forwarding plane ของสวิตช์และทำการปรับเปลี่ยนระบบเน็ตเวิร์ค โดยฮาร์ดแวร์สวิตช์สามารถใช้ OpenFlow messages แจ้งให้ Controller ทราบอีกด้วยเมื่อเกิดความล้มเหลวที่เส้นทาง หรือเมื่อ Packet มาแบบไม่มีคำสั่งเจาะจงคำสั่งในการส่งขึ้นอยู่กับ Flow entry คือสิ่งที่ถูกกำหนดโดยชุดของตัวแปรที่มีความเฉพาะตัว

เช่น Source, Destination, Ethernet/IP address, The switch, Input port, and VLAN tag, etc. [1]

3. Controller กำหนดความเฉพาะชุดของตัวแปรและให้ Packet นั้นเข้ากันได้กับ Flow ซึ่ง Packet ถูกทำให้เข้ากันกับ Flow entry ขึ้นกับการจัดลำดับความสำคัญ Flow entry ที่มีความสำคัญสูงถูกใช้ก่อนอันที่มีความสำคัญต่ำกว่า [1]

2.2 การออกแบบสำหรับป้องกันความล้มเหลวบนระบบเน็ตเวิร์ค

Protection Design ใช้วิธีการพึ่งพาการวางแผนล่วงหน้าของ เส้นทางสำรอง ที่ได้ Packet มาจาก เส้นทางหลัก เพื่อส่งไปยังเส้นทางอื่นๆแสดงไว้ที่รูป 2.1 แต่ละสวิตช์มีการกำหนด เส้นทางสำรอง และ พอร์ทหลัก Packet จะเคลื่อนที่ผ่านทาง เส้นทางหลักโดยทางพอร์ทหลักของแต่ละสวิตช์ไปยังปลายทางในช่วงเวลาปกติถ้าสวิตช์จับได้ว่าพอร์ทหลักไม่พร้อมใช้งาน Packet จะถูกปัดจากเส้นทางหลักไปยังสวิตช์ตัวใกล้ๆจากนั้นส่ง Packet ไปยังปลายทางผ่านพอร์ทหลัก



รูปที่ 2.1 Protection design

การสร้าง Segment protection แสดงใน OpenFlow-based network เนื่องด้วยการสร้าง segment protection ที่แสดงใน OpenFlow มีค่าความสำคัญของ Flow entry ที่เป็นชุดสำหรับเส้นทางสำรองที่ต่ำและมีค่าความสำคัญของเส้นทางหลักในแต่ละสวิตช์ที่สูง ซึ่งหมายความว่า เส้นทางหลักจะถูกใช้เพื่อส่ง Packet เป็นอันดับแรกเนื่องจากมีค่าความสำคัญที่สูงกว่าเส้นทางสำรอง

Fast switch-over time หรือก็คือเวลาที่ใช้ในการกู้คืนสถานะของเส้นทางเมื่อเกิดความล้มเหลวขึ้นบนเส้นทางที่ใช้ส่ง Packet อยู่ในขณะนั้น ซึ่งการกู้คืนสถานะของเส้นทางถูกดำเนินการโดยสวิตช์ที่เชื่อมต่อกับเส้นทางที่เกิดความล้มเหลว รวมถึงกลไกต่างๆที่ถูกร้องขอในทุกๆสวิตช์เพื่อเอา Flow entry ออกจากเส้นทางหลักที่มีความสัมพันธ์กับเส้นทางที่เกิดความล้มเหลว หลังจากสวิตช์ส่ง

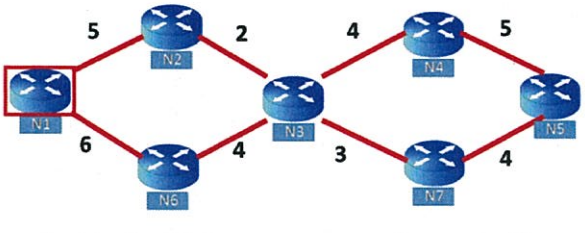
สถานะของพอร์ทของพวกมันไปยัง Controller เพื่อปรับปรุง Topology เพื่อให้สามารถส่ง Packet จากต้นทางไปยังปลายทางได้อีกครั้ง

2.3 การคำนวณเส้นทางที่ดีที่สุด

การคำนวณเส้นทางที่ดีที่สุดบนระบบเน็ตเวิร์คในกราฟที่มีค่าน้ำหนัก หากเอาไปแทนระยะทางระหว่างจังหวัดต่างๆ น้ำหนักหมายถึงระยะทาง เวลาเดินทาง ค่าใช้จ่ายและอื่นๆ รวมกัน ปัญหาลักษณะนี้จึงเป็นเรื่องที่น่าศึกษาเพื่อหาเส้นทางที่มีน้ำหนักน้อยที่สุดจากโหนดที่กำหนดไปยังโหนดอื่นๆ ทุกโหนดในกราฟที่มีเส้นทางไปถึง หรือเรียกว่า เส้นทางที่ดีที่สุดจากจุดเดียว Single Source Shortest Path อัลกอริทึมที่ใช้เพื่อแก้ปัญหานี้รู้จักในชื่ออัลกอริทึมของ Dijkstra' Algorithm ซึ่งเป็นชื่อของนักคณิตศาสตร์ ชาวดัทช์ (Dutch) ผู้เสนอวิธีการแก้ปัญหานี้ในปี ค.ศ. 1959 ข้อจำกัดประการหนึ่งของอัลกอริทึมข้างต้น คือ ค่าน้ำหนักของเส้นทาง(เส้นทางที่เชื่อมต่อกันระหว่างโหนด)ต้องมีค่าไม่น้อยกว่า 0 (Non Negative) ส่วนกราฟจะเป็นแบบมีทิศทางหรือไม่ก็ได้ แต่ต้องเป็นกราฟที่ต่อถึงกัน (Connected) กล่าวคือ จะต้องมีเส้นทาง (Path) ระหว่างทุก ๆ โหนดของกราฟ Dijkstra's Algorithm คล้ายกับอัลกอริทึมของการค้นหาในแนวกว้าง แต่จะต้องเพิ่มส่วนของน้ำหนักในแต่ละโหนด เพื่อแสดงน้ำหนักจากโหนดตั้งต้นจนถึงโหนดนั้นๆ เราสามารถกำหนดค่าเริ่มต้น (Initialize) ให้กับโหนดตั้งต้นให้มีน้ำหนักเป็น 0 ส่วนน้ำหนักที่โหนดใดๆ สามารถหาได้จากการหาผลบวกของน้ำหนักของเส้นทางจากโหนดตั้งต้นถึงโหนดนั้นๆ

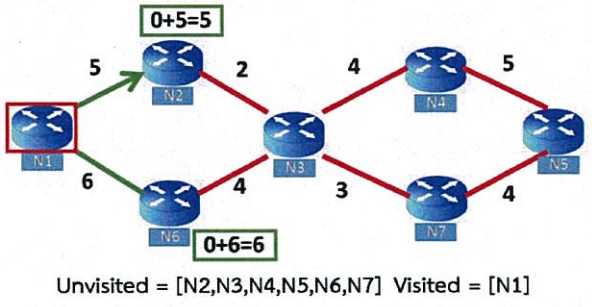
2.3.1 ตัวอย่างการคำนวณ Dijkstra's Algorithm

กำหนดให้แต่ละโหนดมีชื่อดังแสดงในรูป ในตอนเริ่ม ทุกโหนดจะมีค่า น้ำหนักเท่ากับอนันต์น้ำหนัก ในกรณีนี้น้ำหนักคือระยะทางระหว่างโหนดนึงถึงโหนดนึงเราจะแบ่งกลุ่มของโหนดเป็น 2 โหนดที่วงสีแดง และโหนดที่ไม่ได้วงสีแดงโดยโหนดที่เราวงสีแดงคือโหนดที่เรากำลังพิจารณาอยู่เรากำหนดโหนดเริ่มต้นเป็น N1 และปลายทางคือ N5 จากนั้นพิจารณาโหนดข้างเคียงที่มีเส้นทางเชื่อมต่อกัน



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	∞	
N3	∞	
N4	∞	
N5	∞	
N6	∞	
N7	∞	

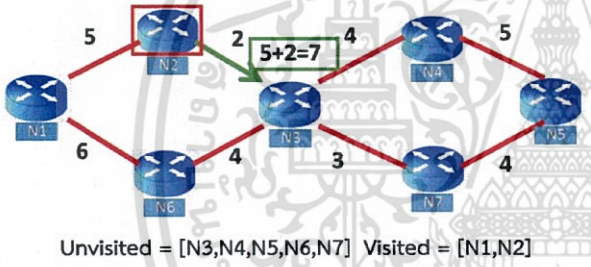
รูปที่ 2.2 กำหนดโหนดเริ่มต้น คือ N1 และปลายทางคือ N5



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	5	N1
N3	∞	
N4	∞	
N5	∞	
N6	6	N1
N7	∞	

รูปที่ 2.3 พิจารณาที่โหนด N1

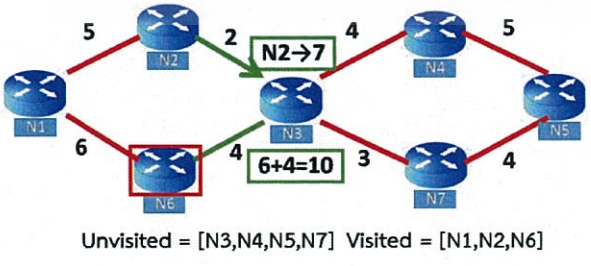
จากรูปที่ 2.3 ทำการพิจารณาที่โหนดใกล้เคียงที่มีเส้นทางเชื่อมต่อกันกับโหนดที่พิจารณา พบว่าค่าน้ำหนักจากโหนด N1→N2 มีค่าเท่ากับ 5 ซึ่งน้อยกว่า N1→N6 ที่มีค่าเท่ากับ 6 ดังนั้นเราจึงเลือกที่จะพิจารณาที่โหนด N2 เป็นโหนดต่อไป



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	5	N1
N3	7	N2
N4	∞	
N5	∞	
N6	6	N1
N7	∞	

รูปที่ 2.4 พิจารณาที่โหนด N2

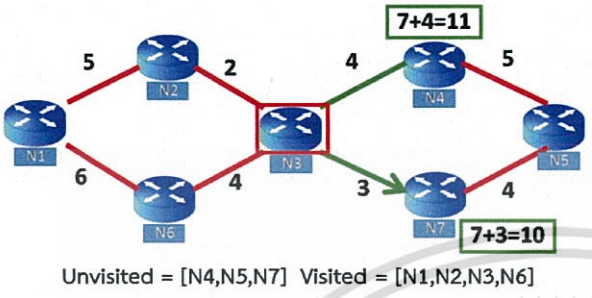
จากรูปที่ 2.4 พบว่า มีเส้นทางที่เชื่อมต่อกับโหนดใกล้เคียงเพียงหนึ่งโหนดคือ N3 ซึ่งมีค่าน้ำหนักเท่ากับ 2 โดยเส้นทางจากโหนด N1→N2→N3 มีค่าน้ำหนักเท่ากับ 5+2 = 7



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	5	N1
N3	7	N2
N4	∞	
N5	∞	
N6	6	N1
N7	∞	

รูปที่ 2.5 พิจารณาที่โหนด N6

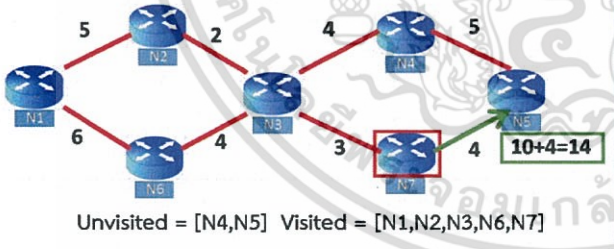
จากรูปที่ 2.5 เมื่อเราพิจารณาที่โหนด N6 ซึ่งมีเส้นทางต่อไปยังโหนด N3 ได้ผลรวมของค่าน้ำหนักคือ 6+4 = 10 ซึ่งมีค่ามากกว่า เส้นทางจากโหนด N1→N2→N3 มีค่าน้ำหนักเท่ากับ 5+2 = 7 ดังนั้นเราจึงเลือกเส้นทางจากโหนด N1→N2→N3 แล้วจึงไปพิจารณาที่โหนด N3 ต่อไป



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	5	N1
N3	7	N2
N4	11	N3
N5	∞	
N6	6	N1
N7	10	N3

รูปที่ 2.6 พิจารณาที่โหนด N3

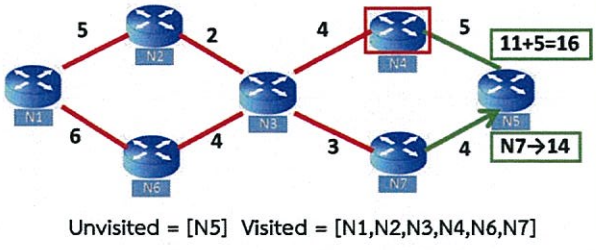
จากรูปที่ 2.6 มีเส้นทางจากที่เชื่อมต่อกับโหนดใกล้เคียงกับโหนดที่พิจารณาคือ N4 และ N5 เมื่อพิจารณาที่ค่าน้ำหนักรวม N1→N2→N3→N4 มีค่าเท่ากับ 7+4 = 11 แต่น้ำหนักรวมของ N1→N2→N3→N7 มีค่าเท่ากับ 7+3 = 10 ซึ่งมีค่าที่น้อยกว่าดังนั้นเราจึงเลือกที่จะพิจารณาที่โหนด N7 ต่อไป



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	5	N1
N3	7	N2
N4	11	N3
N5	14	N7
N6	6	N1
N7	10	N3

รูปที่ 2.7 พิจารณาที่โหนด N7

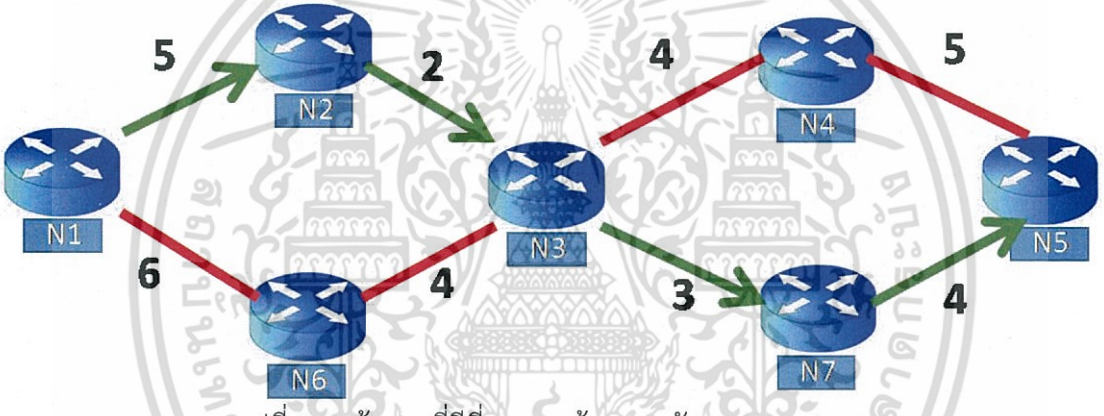
จากรูปที่ 2.7 พบว่ามีเส้นทางที่เชื่อมต่อกับโหนดที่พิจารณาเพียงเส้นทางเดียวคือ N7→N5 ดังนั้นสรุปได้ว่า N1→N2→N3→N7→N5 เป็นเส้นทางที่ดีที่สุดจากระหว่าง N1 ถึง N5 ซึ่งมีค่าน้ำหนักรวมเท่ากับ 14



โหนด	เส้นทางที่สั้นที่สุดจาก N1	โหนดก่อนหน้า
N1	0	
N2	5	N1
N3	7	N2
N4	11	N3
N5	14	N7
N6	6	N1
N7	10	N3

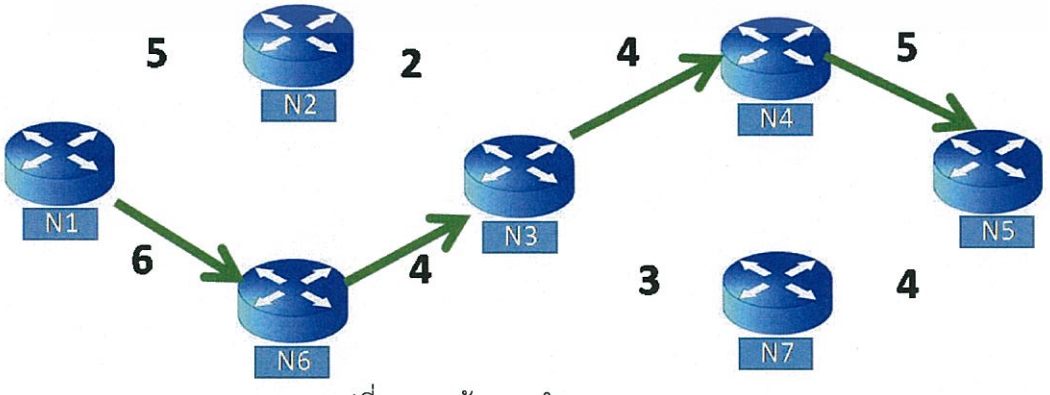
รูปที่ 2.8 พิจารณาที่โหนด N4

จากรูปที่ 2.8 พบว่าเส้นทางจากโหนด N1→N2→N3→N4→N5 มีค่าเท่ากับ 11+5 = 16 ซึ่งมีค่ามากกว่า เส้นทางจากโหนด N1→N2→N3→N7→N5 ที่มีค่าน้ำหนักรวมเท่ากับ 14 เราจึงเลือกเส้นทางจากโหนดว่า N1→N2→N3→N7→N5 เนื่องจากมีเส้นทางที่ดีที่สุด



รูปที่ 2.9 เส้นทางที่ดีที่สุดของเส้นทางหลักบน Topology

เราจะได้เส้นทางหลักที่ดีที่สุดในระบบเน็ตเวิร์คนี้คือ N1→N2→N3→N7→N5 ดังแสดงใน รูปที่ 2.9 ดังนั้นเมื่อนำเส้นทางที่เหลือมาคำนวณระยะทางที่ดีที่สุดอีกครั้ง หากว่าเราไม่คิดเส้นทางที่เป็นเส้นทางหลักจะได้



รูปที่ 2.10 เส้นทางสำรองของ Topology

ดังนั้นโหนด N2 และ N7 ไม่เป็นไปตามเงื่อนไขของ Dijkstra's Algorithm เนื่องจากไม่มีเส้นทางเชื่อมต่อถึงกัน ทำให้พบว่าเส้นทางจาก N1→N6→N3→N4→N5 คือเส้นทางสำรองสำหรับระบบ Topology นี้ซึ่งมีน้ำหนักรวมคือ 19

2.4 งานวิจัยที่เกี่ยวข้อง

2.4.1 Independent Transient Plane Design for Protection in OpenFlow-Based Network

Nattapong Kitsuwan,et.al ได้กล่าวถึงวิธีการป้องกัน Link failure โดยการออกแบบ Flow entry ให้เหมาะสม

2.4.2 OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown,et.al ได้กล่าวถึงระบบ Software-Defined Network และส่วนสำคัญต่างๆในระบบ

2.4.3 Measuring of Failure Switch-Over Time in Software-Defined Network

Zhang Hongbo and Nattapong Kitsuwan ได้กล่าวถึงกระบวนการจับ Switch-Over time

บทที่ 3
วิธีการดำเนินงานวิจัย

ในโครงงานวิจัยนี้ได้ทำการจำลองระบบเน็ตเวิร์คขึ้นมาโดยใช้ตัวจำลองระบบเน็ตเวิร์คที่มีชื่อว่า Mininet และเพื่อจำลองระบบ SDN ซึ่งถูกใช้เพื่อลดกระบวนการต่างๆ เมื่อผู้ใช้ต้องการร้องขอให้สร้างการติดต่อขึ้นบนระบบเน็ตเวิร์ค โดย Controller ในระบบ SDN ถูกออกแบบโปรแกรมให้ใช้ขั้นตอนวิธีการแบบ Dijkstra's Algorithm ซึ่งจะทำการคำนวณเส้นทางที่ดีที่สุดที่สามารถส่งข้อมูลไปได้ โดยวิธีการดำเนินงานวิจัยนี้ได้ทำการทดลองระบบในตัวจำลองระบบโดยใช้ Mininet สามารถแบ่งออกเป็น 2 ขั้นตอน คือ

1.การเตรียมพร้อมใช้งานระบบ

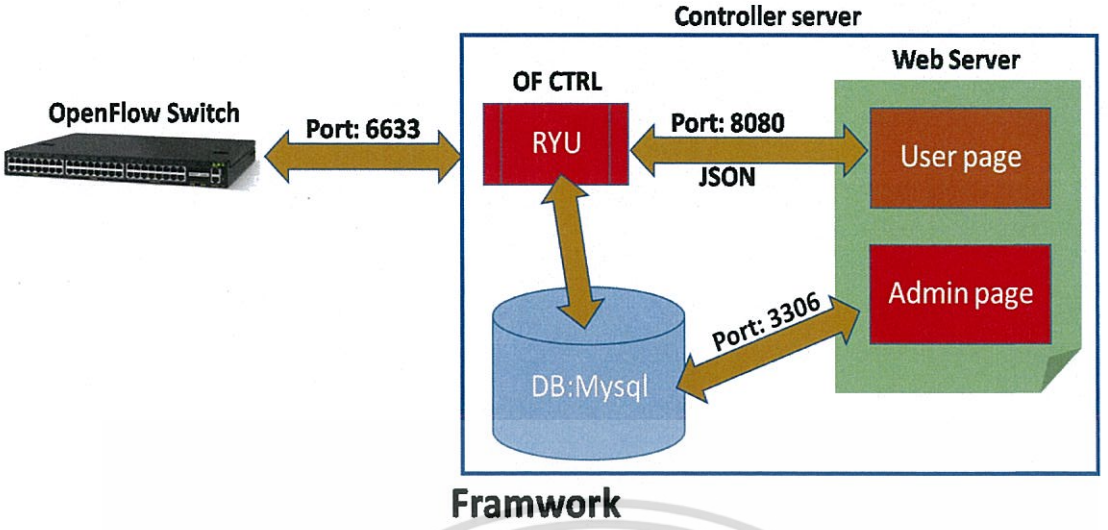
2.เริ่มใช้งานระบบ

3.1 โปรแกรมตัวจำลองระบบเน็ตเวิร์ค

3.1.1 Mininet

Mininet คือ ตัวจำลองระบบเน็ตเวิร์คซึ่งสร้างระบบเน็ตเวิร์คที่มีการจำลอง Hosts, Switches, Controllers, และ Links โดย Mininet ทำงานบนระบบเน็ตเวิร์คของซอฟแวร์ลีนุคและมันสนับสนุน OpenFlow Switch ซึ่งมีความยืนหยุ่นสูงในการสร้างเส้นทางและระบบ SDN

Mininet สนับสนุนในด้านการทดลอง การพัฒนา การเรียนรู้ การวิจัย การแก้ไขบัค และในหัวข้ออื่นๆ ซึ่งสามารถใช้ประโยชน์ของมันอย่างเต็มที่ในการทำการทดลองระบบเน็ตเวิร์คบนแลปทอป หรือ พีซีอื่น



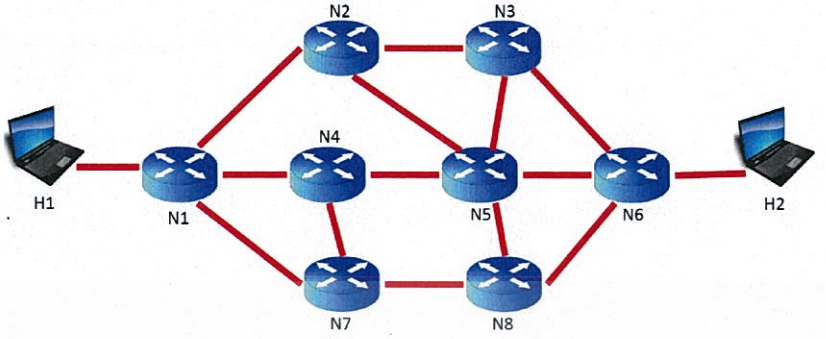
รูปที่ 3.1 Framwork

3.1.2 ส่วนต่างๆภายใน Framwork

1. OpenFlow switch โดยในการทดลองนี้ได้จำลอง OpenFlow switch โดยใช้โปรแกรม Mininet ซึ่ง OpenFlow switch ติดต่อกับ Controller โดยใช้พอร์ต 6633
2. OpenFlow Controller เป็นซอฟแวร์ภายในคอมพิวเตอร์ทำหน้าที่เป็น Controller ของระบบนี้
3. Database ซึ่งการทดลองนี้ใช้ดาต้าเบสที่มีชื่อว่า มายเอสคิวแอล(Mysql) ทำหน้าที่จัดเก็บข้อมูลที่ถูกเพิ่มโดยผู้ควบคุมระบบ
4. เว็บเซิฟเวอร์ โดยภายในมีอยู่สองส่วนคือ 1.User page ทำหน้าที่รับข้อมูลต้นทางและปลายทางจากผู้ใช้ระบบ จากนั้นเว็บเซิฟเวอร์จะทำหน้าที่ส่งข้อความเจสัน (JSON Message) ซึ่งก็คือไวยากรณ์สำหรับการจัดเก็บและการแลกเปลี่ยนข้อมูลไปยัง Controller โดยผ่านพอร์ต 8080 ส่วนที่ 2 คือ Admin page ทำหน้าที่ติดต่อกับดาต้าเบสโดยใช้พอร์ต 3306 เพื่ออัพเดทข้อมูลโดยผู้ควบคุมระบบ หน้าต่างนี้ผู้ควบคุมระบบเท่านั้นที่มีสิทธิใช้ได้

3.2 การเตรียมพร้อมใช้งานระบบ

1.ทำการสร้าง Topology บนระบบเน็ตเวิร์ค โดยใช้โปรแกรมMininet



รูปที่ 3.2 รูปตัวอย่าง Topology สำหรับการทดลอง

2.ทำการเพิ่มข้อมูลเกี่ยวกับระบบเน็ตเวิร์คเช่น Topology, ชื่อสวิตช์, พอร์ต, ต้นทาง, ปลายทาง ไว้ที่ Admin page จากนั้นข้อมูลต่างๆจะถูกเก็บไว้ที่ฐานข้อมูล (Data base)

Table Name = paths

src host	dst host	src node	dst node	src_IP	dst_IP	primary_path	backup_path	wk_no	used		
H1	H2	N1	N6	10.0.0.1	10.0.0.2			0	0	DELETE	EDIT

Src host name:
Dst host name:
Src node name:
Dst node name:
Src_IP:
Dst_IP:
Specified switches on primary path:
Specified switches on backup path:
wk_no:
used:
Add
Edit
Cancle

Table Name = ports				
sw_id	outport	neighbor_sw_id		
N1	1	H1	DELETE	EDIT
N1	2	N2	DELETE	EDIT
N1	3	N4	DELETE	EDIT
N1	4	N7	DELETE	EDIT
N2	1	N1	DELETE	EDIT
N2	2	N3	DELETE	EDIT
N2	3	N5	DELETE	EDIT
N3	1	N2	DELETE	EDIT
N3	2	N5	DELETE	EDIT
N3	3	N6	DELETE	EDIT

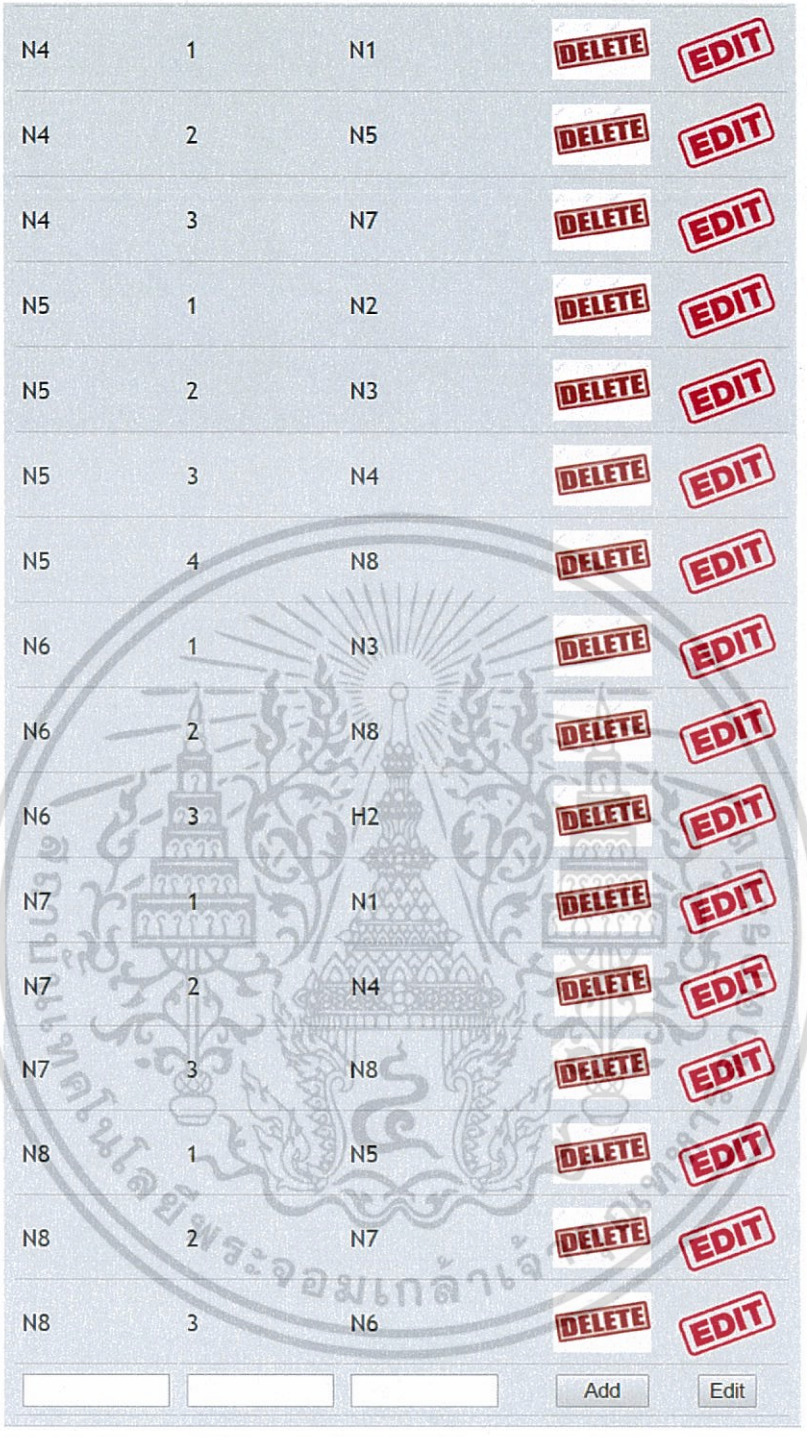
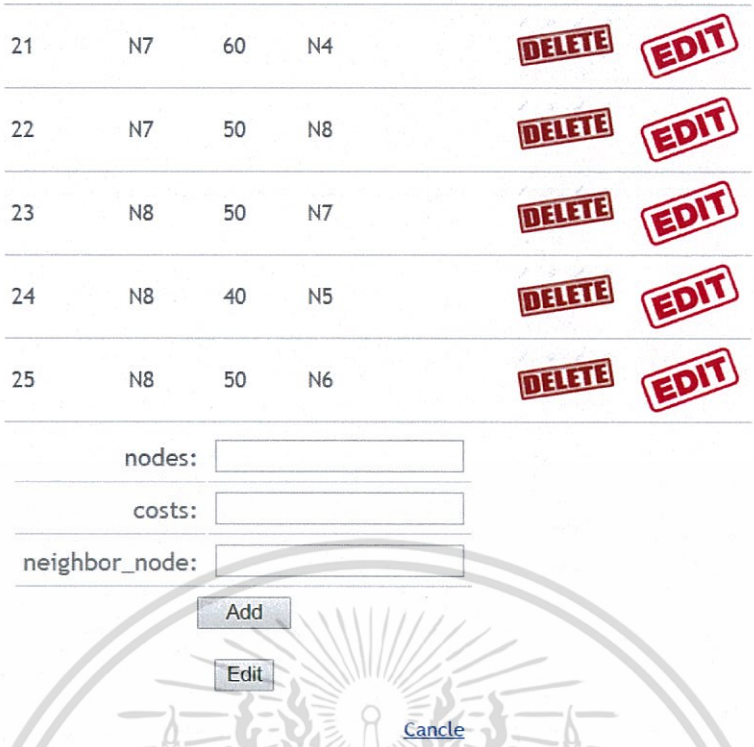
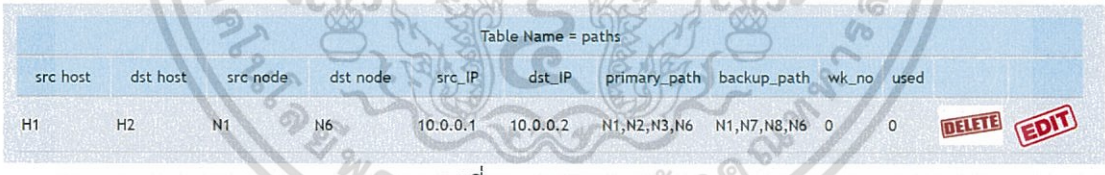


Table Name = Topology				
number	nodes	costs	neighbor_node	
1	N1	50	N2	DELETE EDIT
2	N1	50	N4	DELETE EDIT
3	N1	60	N7	DELETE EDIT
4	N2	50	N1	DELETE EDIT
5	N2	60	N5	DELETE EDIT
6	N2	50	N3	DELETE EDIT
7	N3	50	N2	DELETE EDIT
8	N3	50	N6	DELETE EDIT
9	N3	60	N5	DELETE EDIT
10	N4	50	N1	DELETE EDIT
11	N4	50	N5	DELETE EDIT
12	N4	60	N7	DELETE EDIT
13	N5	50	N4	DELETE EDIT
14	N5	60	N2	DELETE EDIT
15	N5	60	N3	DELETE EDIT
16	N5	40	N8	DELETE EDIT
17	N6	50	N3	DELETE EDIT
18	N6	50	N8	DELETE EDIT
20	N7	60	N1	DELETE EDIT



รูปที่ 3.3 Admin page

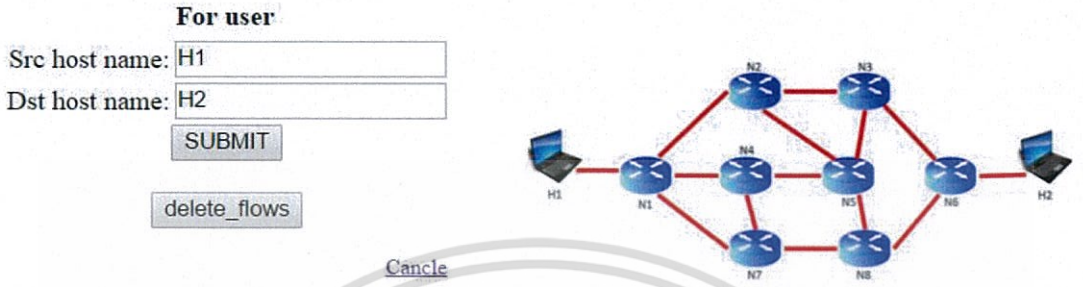
3.ทำการรัน Controller เพื่อทำการกำหนดเส้นทางจากต้นทางไปยังปลายทางที่ดีที่สุดโดยใช้ Dijkstra's Algorithm Controller ทำการเพิ่มข้อมูลเส้นทางหลักและเส้นทางสำรองไปที่ฐานข้อมูล (Data base) เพื่อเก็บข้อมูล ซึ่งข้อมูลถูกแสดงอยู่บน Admin page



รูปที่ 3.4 Table path

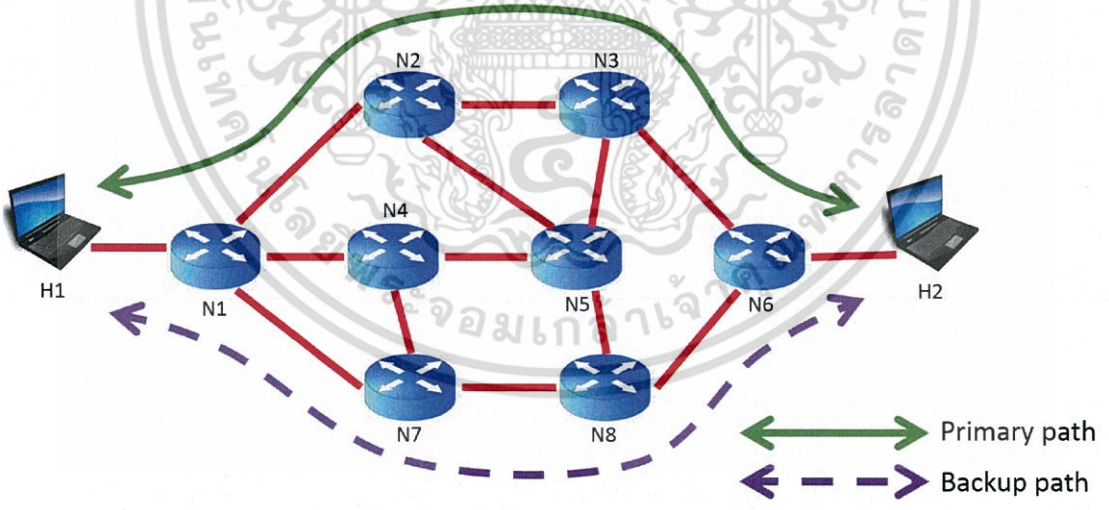
3.3 เริ่มใช้งานระบบ

1.เมื่อผู้ใช้ร้องขอเส้นทางระหว่าง ต้นทาง ไปยังปลายทาง โดยผ่านทาง User page โดยเว็บเซิฟเวอร์จะทำการส่งข้อความเจสัน (JSON Message) ไปหา Controller



รูปที่ 3.5 User page

2.Controller ได้รับข้อความเจสัน (JSON Message) จากนั้น Controller ทำการดึงข้อมูลจากฐานข้อมูล (Data base) และทำการเพิ่ม Flow entry บนสวิตช์แต่ละตัวที่เกี่ยวข้องเพื่อสร้างการติดต่อบนระบบเน็ตวิร์ค



รูปที่ 3.6 Primary path และ Backup path

3.เมื่อเกิดความล้มเหลวบนเส้นทางหลักขึ้น OpenFlow Switch ที่มีเส้นทางติดกับเส้นทางที่เกิดความล้มเหลว ทำการแจ้งสถานะของพอร์ตไปยัง Controller

4.Controller ทำการเปลี่ยนเส้นทางโดยใช้เส้นทางสำรองในการส่งข้อมูลบนระบบเน็ตเวิร์ค

5.หากเส้นทางหลักพร้อมใช้งาน OpenFlow Switch แจ้งสถานะพอร์ตไปยัง Controller

6.Controller ทำการเปลี่ยนเส้นทางการส่งข้อมูลมาเป็นเส้นทางหลักดังเดิม

7.เมื่อผู้ใช้ยกเลิกการร้องขอใช้งานเว็บเซิฟเวอร์ส่งข้อความเจสัน (JSON Message) ไปยัง Controller

8.Controller ทำการเอา Flow entry ออกจากสวิตช์ที่เกี่ยวข้อง ทำให้ผู้ใช้ไม่สามารถส่งข้อมูลจากต้นทางไปยังปลายทางได้

3.4 ขั้นตอนการทดสอบระบบ

หลังจากทำการติดตั้งระบบทั้งหมดเรียบร้อย จึงต้องทำการทดสอบระบบการทำงานโดยมีการแบ่งการทดสอบออกเป็นสองตอน คือ

1.การทดลองการเปลี่ยนเส้นทางเมื่อเกิด Link failure

2.การจับเวลาในการเปลี่ยนเส้นทาง (Switch-over time)

3.4.1 การทดลองการเปลี่ยนเส้นทางเมื่อเกิด Link failure

1.ทำการรัน Controller

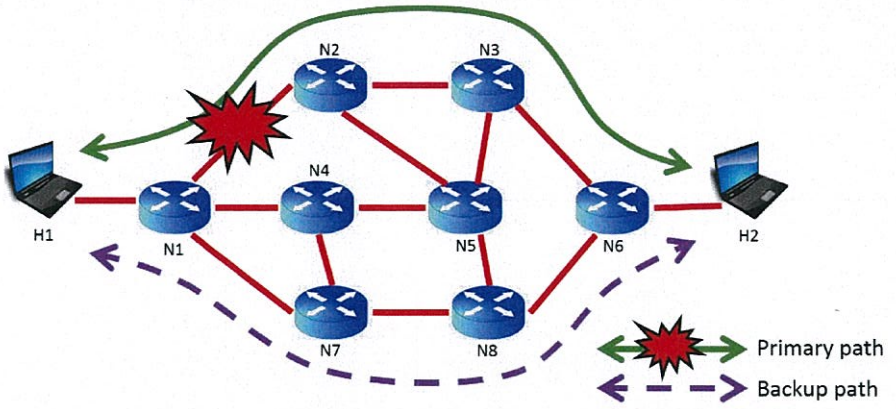
2.ทำการร้องขอโดยการกำหนดต้นทางและปลายทางโดยกำหนดต้นทางเป็นคอมพิวเตอร์ H1 และปลายทางคือคอมพิวเตอร์ H2 ผ่าน User page

3.ทำการรันโปรแกรม Wireshark ซึ่งคือโปรแกรมที่ใช้งานสำหรับคอมพิวเตอร์ของเราที่ต้องการจะทราบ Packet ต่างๆที่ส่งออกไปยังเครือข่ายอินเตอร์เน็ต และการตอบกลับ Response จากเซิฟเวอร์ที่ได้รับข้อมูลที่ส่งออกจากคอมพิวเตอร์ของเรา โดยโปรแกรมนี้เป็นโปรแกรมสำหรับวิเคราะห์การรับส่งข้อมูล Packet ต่างๆได้เป็นอย่างดีอีกทั้งยังสามารถบอกได้ว่าการรับส่งข้อมูลเป็นไปตามลำดับอย่างไร

4.จาก Topology ที่เราสร้างดังแสดงในรูปที่ 3.1 หลังจากทำการรันโปรแกรม Wireshark บนเครื่องคอมพิวเตอร์ทุกเครื่องแล้ว กำหนดไอพีของคอมพิวเตอร์ทุกพอร์ต และทำการส่ง Packets จากคอมพิวเตอร์ H1⟶H2 จากนั้นสังเกตุความเปลี่ยนแปลงบนโปรแกรม Wireshark

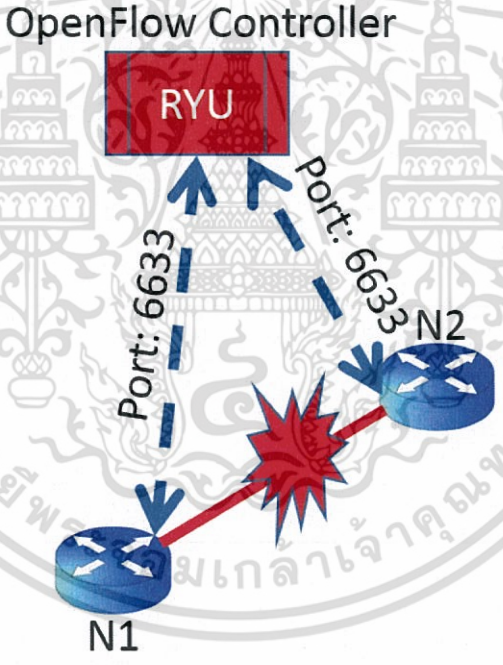
5.ถ้าหากตรวจสอบแล้ว มีข้อมูลไหลผ่านบนคอมพิวเตอร์ที่อยู่บนเส้นทางหลักจริง แสดงว่าถูกต้องแต่ถ้าไม่ ให้ดำเนินการแก้ไขปัญหาด้วยวิธีต่างๆ ดังนี้ ทำการตรวจสอบว่ามีการจำลอง Topology ถูกต้องหรือมีเส้นทางไหนที่อยู่บนเส้นทางหลักเสียหายหรือไม่ ทำการตรวจสอบการกำหนดค่าบนสวิตช์แต่ละตัวทำการตรวจสอบว่าโค้ดของโปรแกรมมีความถูกต้องหรือไม่ หรืออื่นๆ

6.จากนั้นทำการทดลองให้เกิดความล้มเหลวบนเส้นทางหลัก



รูปที่ 3.7 Topology เมื่อเกิดความเสียหายขึ้นที่เส้นทางหลัก

เมื่อเกิดความล้มเหลวขึ้นที่เส้นทางหลักทำให้ไม่สามารถใช้เส้นทางหลักในการส่งข้อมูลได้ OpenFlow switch ทำการส่งสถานะของพอร์ตไปยัง OpenFlow controller



รูปที่ 3.8 แสดงการส่งสถานะของพอร์ตไปยัง Controller

7.ทำการส่ง Packets จากคอมพิวเตอร์ H1→H2 อีกครั้ง สังเกตผลจากโปรแกรม wireshark ถ้าหากข้อมูลไหลผ่านเส้นทางสำรองแสดงว่าถูกต้องแต่ถ้าไม่ ให้ดำเนินการแก้ไขปัญหาด้วยวิธีต่างๆ ดังนี้ ทำการตรวจสอบว่ามีเส้นทางไหนที่อยู่บนเส้นทางสำรองเสียหายหรือไม่ ทำการตรวจสอบการกำหนดค่าบนสวิตช์แต่ละตัว ทำการตรวจสอบว่าโค้ดของโปรแกรมมีความถูกต้องหรือไม่ หรืออื่นๆ

8.ทำการซ่อมแซมเส้นทางหลักให้พร้อมใช้งาน หลังจากนั้น OpenFlow switch ทำการอัพเดตสถานะพอร์ตและส่งข้อความไปยัง OpenFlow controller

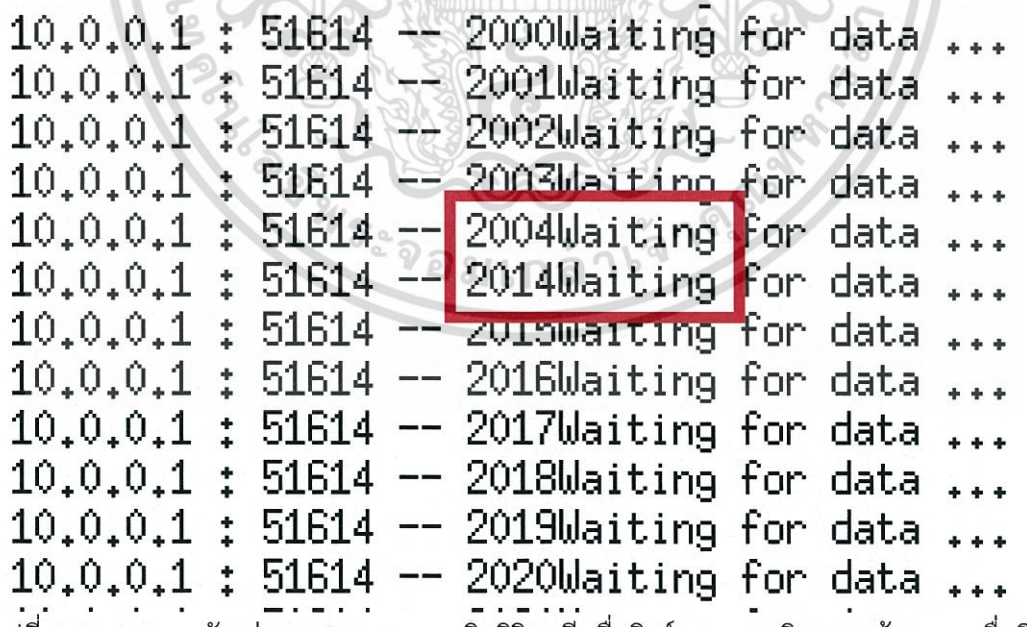
9.ทำการส่ง Packets จากคอมพิวเตอร์ H1→H2 อีกครั้ง สังเกตผลจากโปรแกรม wireshark ถ้าหากตรวจสอบแล้ว มีข้อมูลไหลผ่านบนคอมพิวเตอร์ที่อยู่บนเส้นทางหลักจริง แสดงว่าถูกต้อง แต่ถ้าไม่ ให้ดำเนินการแก้ไขปัญหาด้วยวิธีต่างๆ ดังนี้ ทำการตรวจสอบว่ามีการจำลอง Topology ถูกต้องหรือมีเส้นทางไหนที่อยู่บนเส้นทางหลักเสียหายหรือไม่ ทำการตรวจสอบการกำหนดค่าบนสวิตช์แต่ละตัวทำการตรวจสอบว่าโค้ดของโปรแกรมมีความถูกต้องหรือไม่ หรืออื่นๆ

10.ทดลองทำการร้องขอเพื่อยกเลิกการใช้เส้นทางบนระบบเน็ตเวิร์คเพื่อส่งข้อมูล

11.ทำส่ง Packets จากคอมพิวเตอร์ H1→H2 อีกครั้ง สังเกตผลจากโปรแกรม wireshark ถ้าหากตรวจสอบแล้วว่ายังมีข้อมูลไหลอยู่บนระบบเน็ตเวิร์ค แสดงว่าผิดพลาดแต่ถ้าไม่ แสดงว่าผลการทดลองประสบความสำเร็จ

3.4.2 การจับเวลาในการเปลี่ยนเส้นทาง (Switch-over time)

การจับเวลาในขณะที่ทำการส่ง Packet แล้วเกิด Link failure เพื่อตรวจวัดเวลาในการแก้ไขปัญหา Link failure ทำการจับเวลาโดยใช้วิธีการให้ ต้นทางส่ง Packet ทุกๆเวลา 1 มิลลิวินาที ไปหาปลายทาง ดังนั้นแต่ละ Packet ที่ปลายทางรับได้จะมีเวลาห่างกัน Packet ละ 1 มิลลิวินาที



รูปที่ 3.9 แสดงการรับ-ส่ง Packet ทุกๆ 1 มิลลิวินาที เมื่อลิงค์ N1 N2 เกิดความล้มเหลว เมื่อลิงค์ N1 N2 เกิดความล้มเหลวขึ้น มีการแก้ไขปัญหาโดยใช้เวลา 2014 – 2004 = 10 มิลลิวินาที

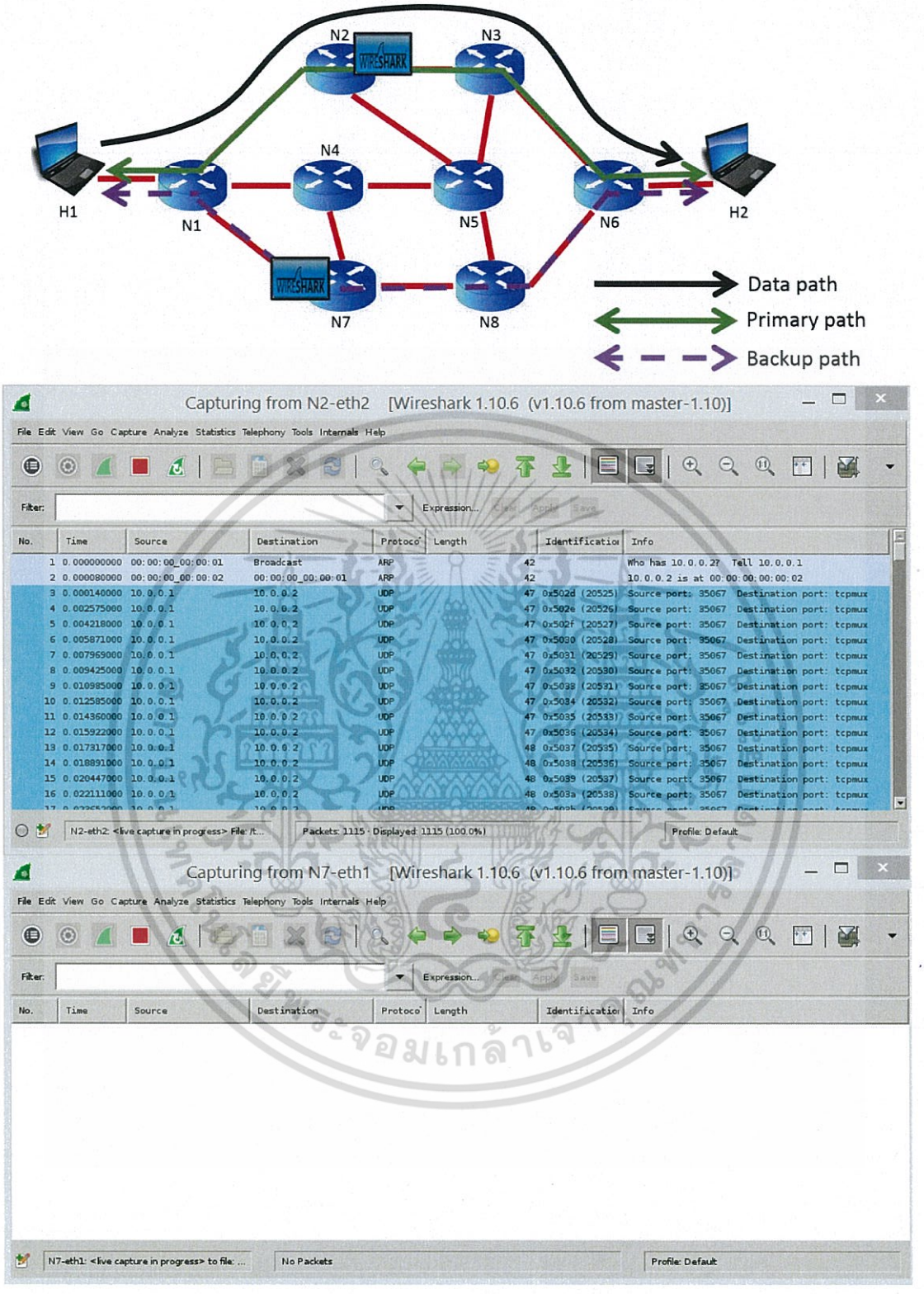
บทที่ 4
ผลการวิจัยและการอภิปรายผล

จากการทดลองให้เกิดความล้มเหลวขึ้นบนเส้นทางที่ใช้ส่ง Packet เพื่อแก้ไขปัญหาเนื่องจากเส้นทางที่ใช้เกิดเสียทำให้ไม่สามารถส่ง Packet ได้ Controller ได้ทำการเปลี่ยนเส้นทางแบบอัตโนมัติ เราสามารถทราบว่า Packet ที่ถูกส่งอยู่บนเส้นทางใด โดยการใช้โปรแกรม Wireshark เพื่อทำการตรวจว่ามี Packet เคลื่อนผ่านเส้นทางไหนบ้าง ซึ่งการเปลี่ยนแปลงเส้นทางนี้ เราได้ทำการตรวจวัดเวลาที่ใช้ในการแก้ไขปัญหา Link failure โดยผลที่ได้คือใช้เวลาน้อยกว่า 50 มิลลิวินาที เนื่องด้วยระบบทั่วไปคือ เมื่อเกิดความล้มเหลวบนเส้นทางที่ใช้ส่ง Packet ทางผู้ดูแลจะทำการแจ้งไปยังฝ่ายช่างเทคนิคเพื่อเข้าไปแก้ปัญหาที่พื้นที่นั้นๆ ซึ่งมีความล่าช้าเป็นอย่างมาก หากเทียบกับระบบ SDN แล้วทำให้เห็นความแตกต่างของเวลาที่ใช้ในการแก้ปัญหาอย่างชัดเจน ผลการทดลองได้ถูกแบ่งออกเป็นสองส่วนคือ

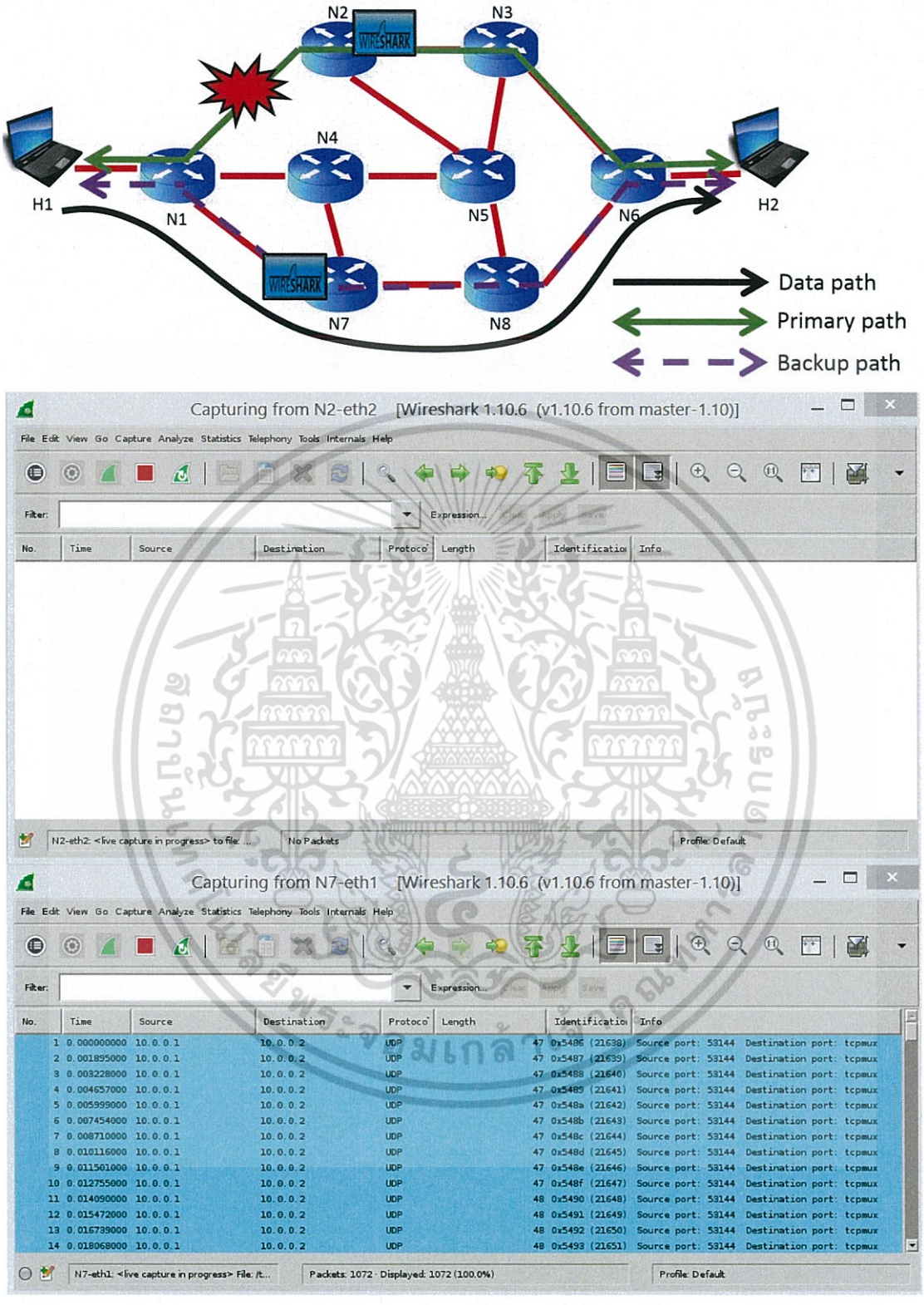
1. ผลการทดสอบการเปลี่ยนเส้นทางเมื่อเกิด Link failure
2. ผลการจับเวลาในการเปลี่ยนเส้นทางเมื่อเกิดความล้มเหลวบนเส้นทางหลัก

4.1 ผลการทดสอบการเปลี่ยนเส้นทางเมื่อเกิด Link failure

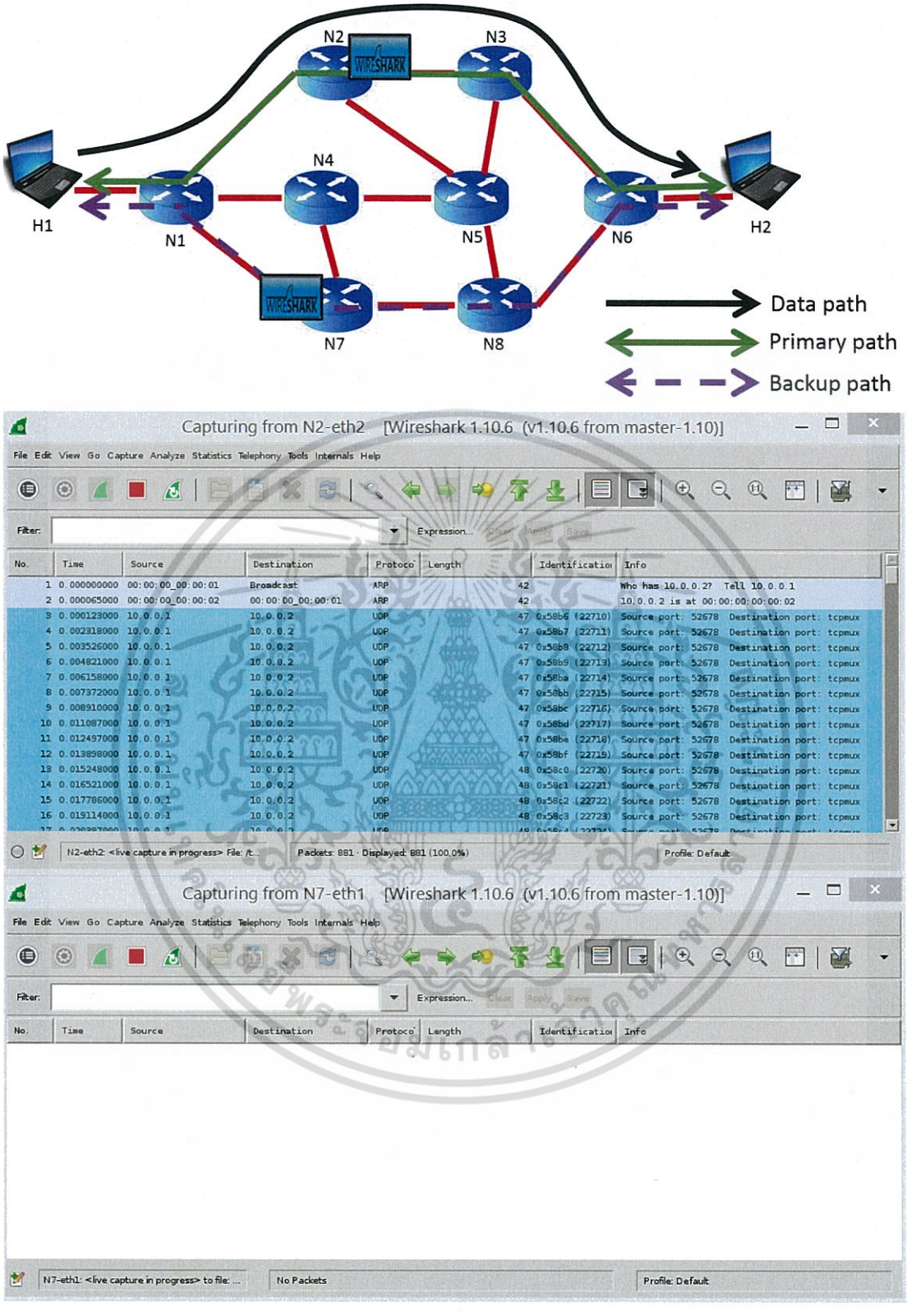
จากผลการผลการทดสอบการเปลี่ยบนเส้นทางเมื่อเกิด Link failure ปรากฏว่า มีผลเป็นไปตามสมมติฐาน คือถ้ายังไม่เกิด Link failure บนเส้นทางหลักโปรแกรม wireshark จะสามารถจับ Packet ที่วิ่งผ่านบนเส้นทางหลักได้ แต่จะไม่พบ Packet บนเส้นทางสำรอง แต่เมื่อเส้นทางหลักเกิด Link failure ขึ้น พบว่าโปรแกรม wireshark ตรวจพบ Packet ที่วิ่งผ่านบนเส้นทางสำรองแต่ไม่พบ Packet บนเส้นทางหลัก ในกรณีที่ User ยกเลิกการใช้งานผ่าน User page เมื่อทำการใช้โปรแกรม wireshark ทดสอบปรากฏว่าไม่พบ Packet วิ่งผ่านบนระบบเน็ตเวิร์คเลยทำให้ไม่สามารถส่งข้อมูลหรือติดต่อกันได้



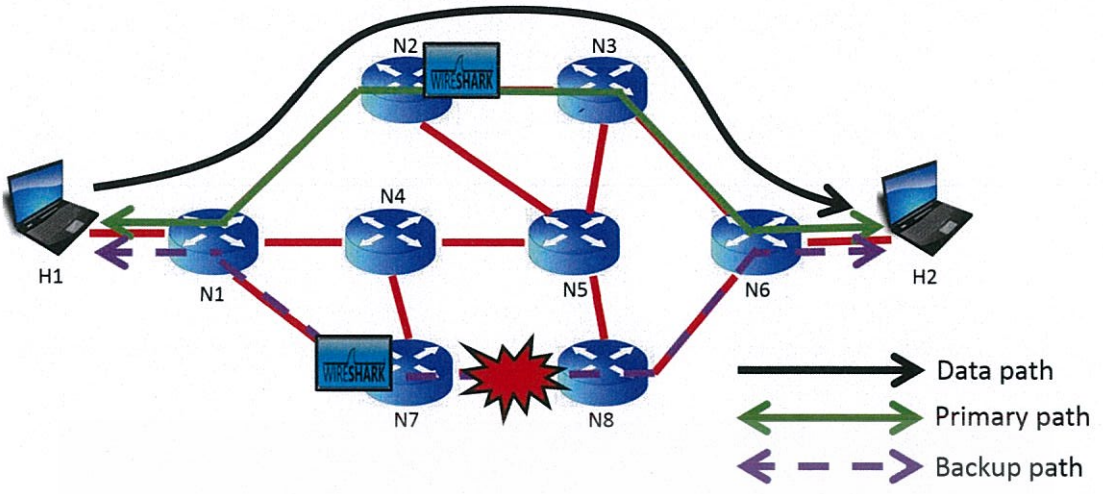
รูปที่ 4.1 เมื่อทำการใช้โปรแกรม wireshark ในขณะที่ไม่เกิด Link failure บนเส้นทางหลัก

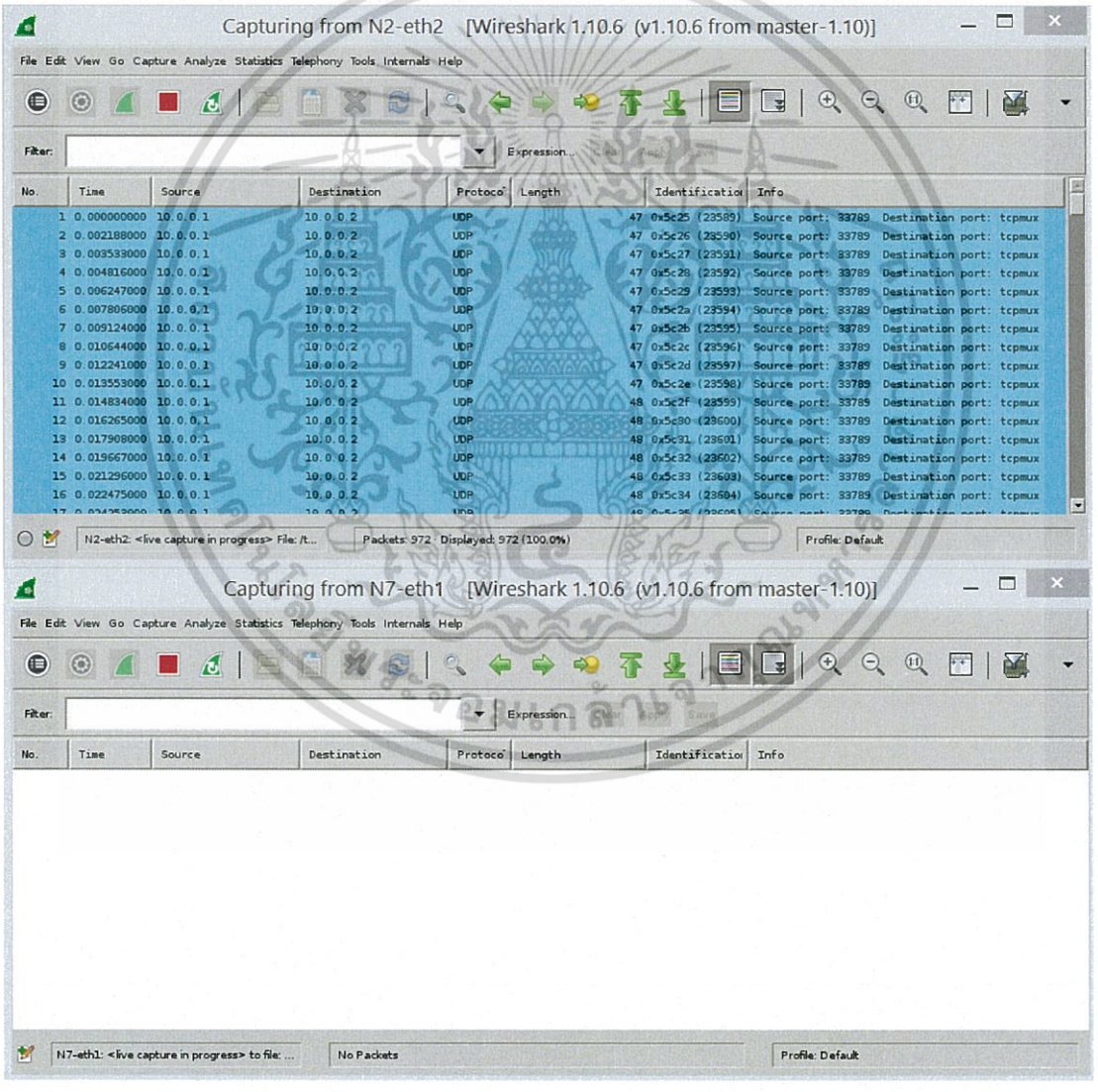


รูปที่ 4.2 เมื่อทำการใช้โปรแกรม wireshark ในขณะที่เกิด Link failure บนเส้นทางหลัก

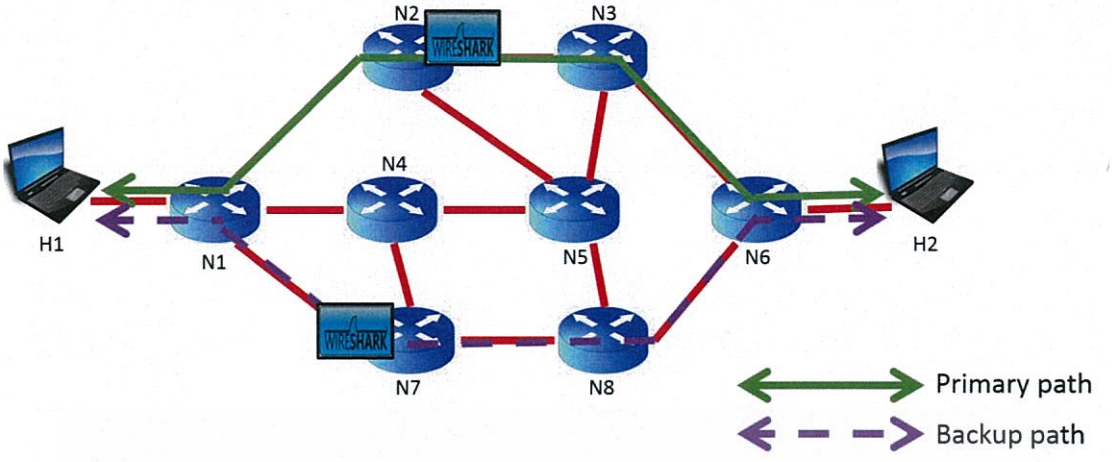


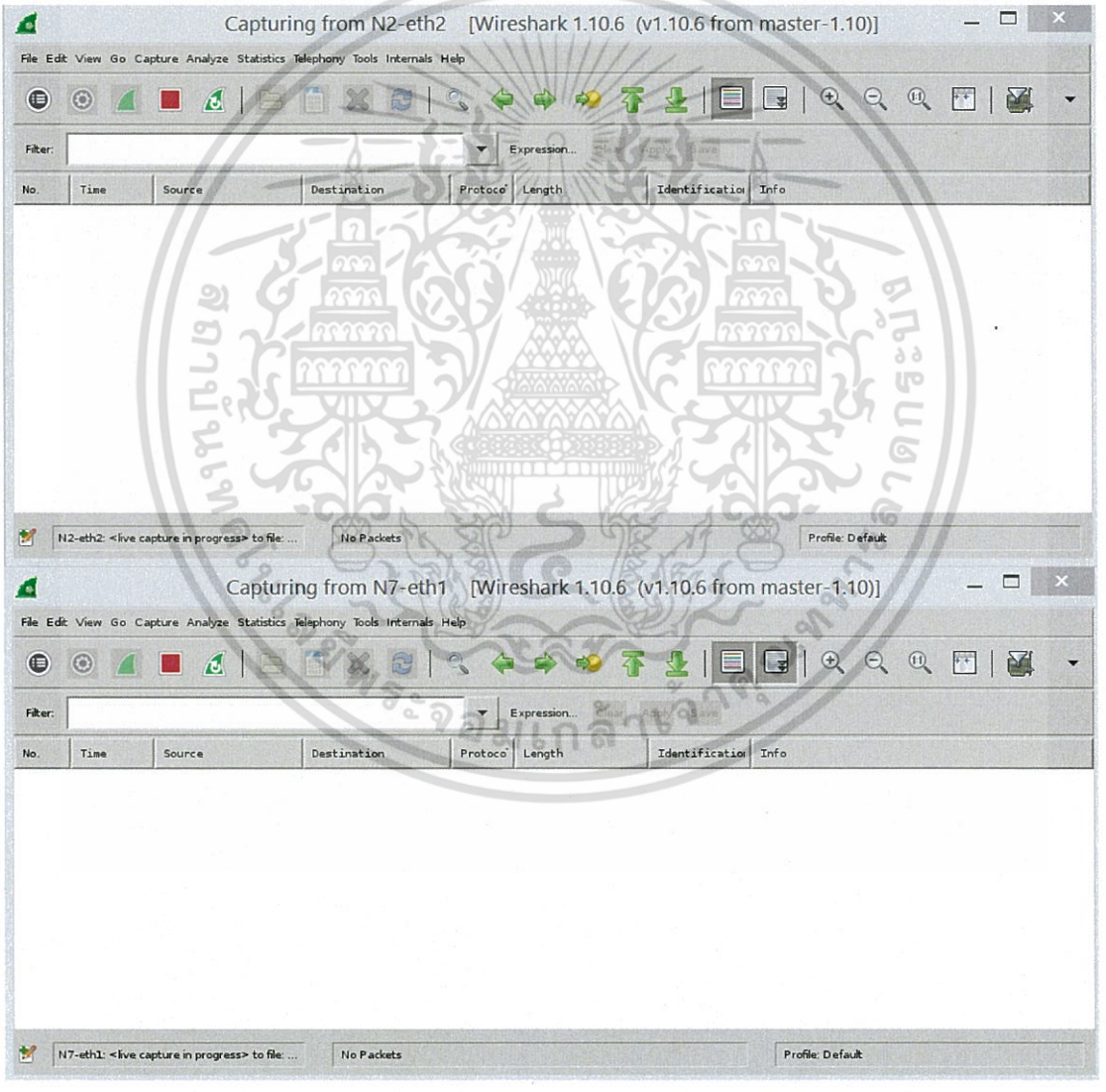
รูปที่ 4.3 เมื่อทำการใช้โปรแกรม wireshark ในขณะที่เส้นทางหลักกลับมาพร้อมใช้งาน





รูปที่ 4.4 เมื่อทำการใช้โปรแกรม wireshark ในขณะที่เกิด Link failure บนเส้นทางสำรอง





รูปที่ 4.5 ทำการใช้โปรแกรม wireshark เมื่อ User ยกเลิกการใช้งาน

4.2 ผลการจับเวลาในการเปลี่ยนเส้นทางเมื่อเกิดความล้มเหลวบนเส้นทางหลัก

ตารางที่ 4.1 ตารางแสดงผลของการจับเวลาในการเปลี่ยนเส้นทาง เมื่อเกิดความล้มเหลวบนเส้นทางหลัก

ลำดับ Packet ก่อนหน้า	ลำดับ Packet ถัดมา	ผลต่างของลำดับ Packet (มิลลิวินาที)
1409	1400	9
1502	1484	18
1559	1545	14
2014	2004	10
1708	1700	8
1193	1182	11
1452	1441	11
1682	1670	12
1728	1696	32
1594	1582	12
1509	1498	11
1619	1608	11
2021	2006	15
1614	1604	10
1631	1620	11
1076	1065	11
1858	1833	25
1577	1565	12
2488	2476	12
1682	1672	10
1443	1432	11
1837	1824	13
2284	2274	10
1385	1373	12
1857	1822	35

ตารางที่ 4.1 ตารงแสดงผลของการจับเวลาในการเปลี่ยนเส้นทาง
เมื่อเกิดความล้มเหลวบนเส้นทางหลัก (ต่อ)

ลำดับ Packet ก่อนหน้า	ลำดับ Packet ถัดมา	ผลต่างของลำดับ Packet (มิลลิวินาที)
1529	1490	39
2162	2148	14
1437	1423	14
1579	1565	14
2549	2539	10

ผลลัพธ์ของค่าเฉลี่ยผลต่างของลำดับ Packet มีค่าประมาณ 14.5667 มิลลิวินาที

ทำการหาระดับนัยสำคัญ (Level of significance) เพื่อหาโอกาสที่จะเกิดความคลาดเคลื่อนในการสรุปผลตามผลการทดสอบสมมุติฐานซึ่งจะสะท้อนถึงความเชื่อมั่นในการสรุปตามผลการทดสอบ หรือเป็นการแสดงว่าข้อสรุปนั้นเชื่อถือได้มากน้อยเพียงใด โดยการทดลองนี้ กำหนดระดับนัยสำคัญที่ 0.05

ขั้นที่ 1 ตั้งสมมุติฐาน

สมมุติฐานทางสถิติ $H0 : \mu \leq 30, H1 : \mu > 30$

ขั้นที่ 2 กำหนดระดับนัยสำคัญ

$\alpha = 0.05$

ขั้นที่ 3 เลือกสถิติที่ใช้ในการทดสอบสมมุติฐาน

$$S = \sqrt{\frac{\sum(X-\bar{X})}{N-1}} = 8.7671$$

$$t = \frac{\bar{X}}{S/\sqrt{n}} \; ; \; df = n - 1$$

ขั้นที่ 4 กำหนดขอบเขตวิกฤติ

กำหนด $\alpha = 0.05$ และเป็นการตั้งสมมุติฐานแบบทางเดียว $df = 30-1 = 29$

เปิดตาราง ที่ $\alpha = 0.05$ จะได้ค่าวิกฤต $t = 1.6991$ (t ตาราง $= 1.6991$)

ขั้นที่ 5 คำนวณค่าสถิติตามสูตร

$$t = \frac{\bar{X}}{S / \sqrt{n}}; \quad df = 30 - 1 = 29$$

$$t = \frac{14.5667 - 30}{8.7671/\sqrt{30}} = -9.6419$$

ขั้นที่ 6 สรุปตัดสินใจ

เมื่อ t คำนวณ > t ตาราง จะ ปฏิเสธ H0 และ ยอมรับ H1

เมื่อ t คำนวณ < t ตาราง จะยอมรับ H0

เนื่องจาก t คำนวณ = -9.6419 น้อยกว่า t ตาราง =1.6991 ดังนั้น ยอมรับ H0 นั้นคือ การทดลองมีค่าของ Switch-over time ต่ำกว่าเกณฑ์ที่กำหนดไว้ คือ 30 ms อย่างมีนัยสำคัญทางสถิติที่ระดับ 0.05

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

ในงานวิจัยนี้ได้ทำการสร้างเว็บเพื่อตอบสนองความสะดวกของผู้ควบคุมระบบและผู้ใช้ระบบ โดยระบบ SDN กำหนดเส้นทางโดยใช้ Dijkstra's Algorithm เพื่อหาเส้นทางที่ดีที่สุดในการส่ง Packet เมื่อเกิดความล้มเหลวบนเส้นทางหลัก Controller ทำการเปลี่ยนเส้นทางในการส่ง Packet เป็นเส้นทางสำรองโดยมีความเร็วที่ใช้ในการเปลี่ยนเส้นทางน้อยกว่า 50 มิลลิวินาที

5.2 ข้อเสนอแนะ

จากการทดลองนั้นเป็นการทดลองโดยใช้ระบบจำลองบนเครื่องคอมพิวเตอร์ ทำให้ผลลัพธ์ของค่าต่างๆที่ทำการวัด เช่น ค่าเวลาที่ใช้ในการแก้ปัญหาความล้มเหลวบนเส้นทางหลัก มีค่าที่ไม่แน่นอนสูงจึงต้องทำการทดลองซ้ำหลายรอบแล้วจึงหาค่าเฉลี่ย ถ้าหากได้ทดลองบนระบบจริงจะทำให้ผลลัพธ์ต่างๆมีความเที่ยงตรงมากขึ้น และเนื่องด้วยเครื่องคอมพิวเตอร์อาจมีการทำงานของโปรแกรมอยู่เบื้องหลังจึงทำให้ประสิทธิภาพของคอมพิวเตอร์ถูกใช้ในการทดลองไม่ครบหนึ่งร้อยเปอร์เซ็น

เอกสารอ้างอิง

Nattapong Kitsuwan,et.al (2015) . Independent Transient Plane Design for Protection in OpenFlow-Based Networks . Journal of Optical Communications and Networking , 264-275

Zhang Hongbo and Nattapong Kitsuwan (2016) . Measuring of Failure Switch-Over Time in Software-Defined Network . 2016 IEEE 17th International Conference on High Performance Switching and Routing

Nick McKeown,et.al (2008) . OpenFlow: Enabling Innovation in Campus Networks . ACM SIGCOMM Computer Communication Review

ภาคผนวก

ภาคผนวก ก

ในหมวดของภาคผนวก ก จะกล่าวถึงเครื่องมือที่ใช้ในการทดลอง

เครื่องมือที่ใช้ในการทดลอง

คอมพิวเตอร์โน๊ตบุ๊ค ยี่ห้อ MSI รุ่น GE62 2QC Apache, Latest 5th Gen. Intel® Core™ i7 5950HQ / 5700HQ processor Ram 8 GB

ภาคผนวก ข

ในหมวดของภาคผนวก ข จะกล่าวถึงขั้นตอนการติดตั้งโปรแกรมที่จำเป็นสำหรับการทดลอง

การติดตั้งโปรแกรมที่จำเป็นสำหรับการทดลอง

1. Oracle VM Virtual Box

2. Putty

3. WinSCP

4.Mininet

ในส่วนของ Mininet จะต้องติดตั้งโปรแกรมโดยใช้คำสั่งดังต่อไปนี้

sudo apt-get update

sudo apt-get install python-pip python-dev -y

ขั้นที่ 1

sudo apt-get install python-eventlet python-routes python-webob

python-paramiko -y

ขั้นที่ 2

sudo pip install ryu

sudo apt-get install git -y

git clone git://github.com/osrg/ryu.git

cd ryu; sudo python ./setup.py install

ขั้นที่ 3

sudo pip install oslo.config

sudo pip install msgpack-python

sudo pip install eventlet –upgrade

sudo pip install six –upgrade

ขั้นตอนการติดตั้ง LAMP (Linux, Apache, MySQL, PHP)

ขั้นที่ 1

sudo apt-get update

ขั้นที่ 2

sudo apt-get -y install apache2 mysql-server php5-mysql php5 libapache2-mod-php5

php5-mcrypt

ขั้นที่ 3

sudo mysql_install_db

ขั้นที่ 4

sudo mysql-secure_installation

ขั้นที่ 5

sudo nano /etc/apache2/mods-enabled/dir.conf

DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm

save and exit

sudo service apache2 restart

ขั้นที่ 6

sudo apt-get install python-mysqldb

sudo apt-get install build-essential python-dev libmysqlclient-dev

pip install yaml

sudo apt-get install php-curl

sudo apt-get install python-pycurl

ภาคผนวก ค

ในหมวดของภาคผนวก ค จะแสดงถึงบทความวิจัยที่เกี่ยวข้อง

Independent Transient Plane Design for Protection in OpenFlow-Based Networks

Nattapong Kitsuwan, Séamas McGettrick, Frank Slyne, David B. Payne, and Marco Ruffini

***Abstract*—Network protection against link failure is required to allow packets to reach the destination with minimal packet loss when a failure occurs. When a link fails, traffic that attempts to use the failed link is interrupted. Typically, routers in the network discover the failure and find a new route to bypass the failed link. Alternatively, well-known segment protection schemes can also be used to speed up the link recovery time by rerouting packets locally through precalculated protection paths. However, several backup paths have to be prepared for each primary path, making path configuration rather complex and poorly scalable. This paper proposes a design for fast rerouting in an OpenFlow-based network. This new design reduces the number of flow entries and the number of configuration messages needed for network rerouting, which in turn reduces the memory size needed in each switch and the CPU load at the controller. We show empirically and using simulations that our design can reduce the number of flow entries and configuration messages needed by about 60% and 75%, respectively, when compared with an existing OpenFlow-based segment protection design. Furthermore, we implement the proposed design on a pan-European network and show that our design can recover from a link failure in as little as 25 ms.**

***Index Terms*—Mininet; Network protection; OpenFlow; Routing; Software-defined network.**

I. Introduction

Failure protection is an important issue in network survivability. When connectivity is lost due to a failure, packets that attempt to pass the failed link cannot be delivered to their destination. An open shortest path first (OSPF) network, in which a link-state-based routing protocol is used to learn network topology, can detect the link failure and converge to a new topology. However, recovery time in the OSPF network takes more than a second [1]. Recovery times of this order are not acceptable in many networks, as target protection times of 50 ms or 100 ms are common, respectively, for leased line traffic and video/audio services [2]. Multiprotocol label switched (MPLS) networks provide fast rerouting using an alternative label switched path (LSP) to reroute packets from a node connected to the failed link to another node or to the destination. In a typical MPLS network, operating a distributed control plane [3,4], forwarding decisions are made locally by each switch, based on its predefined configurations. Although this allows for dynamic and automatic forwarding decisions to be made in the local switch, there is no guarantee that the chosen route is still optimal when the failure occurs.

Segment protection approaches rely on preplanned backup paths that deflect packets from a primary path to other paths [5–7], as shown in Fig. 1. At each switch, primary and backup ports are defined. Packets travel through the primary path via the primary port of each switch to the destination during normal operations. If the switch detects that the primary port is not available, the packets are deflected from the primary path to a neighboring switch via the backup port of that switch. The neighboring switch then forwards the packets to the destination via its primary port.

Software-defined networking (SDN) [8,9] is an attractive solution that enables a network administrator to control the network requirements, including how to deal with failure protection [10,11]. SDN separates the network control and data forwarding planes. This allows network engineers and administrators to respond quickly to changing requirements of the network from a centralized controller. The network administrator can avail of improved network programmability to flexibly control the physical switches from the controller, which runs all the intelligent control and management software, regardless of the vendor or the model of the switches used.

OpenFlow [12,13] is a widely known protocol (southbound interface) for SDN networks that enables the controller to interact with the forwarding plane of the switches and thus make adjustments to the network. The hardware switches can also use OpenFlow messages to inform the controller when links go down or when a packet arrives with no specified forwarding instructions. The forwarding instructions are based on a flow entry, which is defined by a set of specific parameters, such as the source and destination Ethernet/IP addresses, the switch input port and VLAN tag, etc. [13]. The controller specifies the set of parameters and how packets that match the flow entry should be processed. Packets are matched against flow entries based on prioritization. Higher priority flow entries are used, when available, over lower priority ones.

Manuscript received October 9, 2014; revised February 4, 2015; accepted February 4, 2015; published March 23, 2015 (Doc. ID 224583).

The authors are with CTVR, Trinity College Dublin, Dublin 2, Ireland (e-mail: kitsuwan@tcd.ie).

http://dx.doi.org/10.1364/JOCN.7.000264

1943-0620/15/040264-12$15.00/0

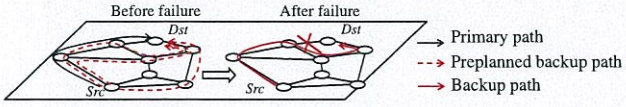


Fig. 1. Segment protection design.

In [14] Sgambelluri *et al.* implemented a segment protection scheme in an OpenFlow-based network. In order to implement the segment protection scheme in OpenFlow, a low priority value of flow entry is set for the backup path, and a high priority value is set for the primary path in each switch. A fast switchover time is guaranteed by the fact that path recovery is performed locally by the switch connected to the failed link. An additional mechanism is required in every switch to remove flow entries from the primary path that relate to the failed link. The switches then send their port status to the controller to update the topology.

This design has some disadvantages. First, the primary and backup paths are correlated, as the latter is reassigned every time the primary path is changed. In addition, several backup paths have to be prepared for the primary path. The number of backup paths (N_{BP}) for each primary path depends on the number of intermediate switches (N_S), where $N_{BP} = N_S + 1$. This makes the provisioning of backup paths more complex if there are several intermediate nodes between the source and destination pair. This will be explained in more detail using an example in Section II. In a large network, a large number of flow entries for source–destination pairs have to be stored in each switch, which can lead to larger loads on the forwarding tables of the switch [thus requiring larger amounts of ternary content-addressable memories (TCAMs)]. TCAM is a type of high-speed memory capable of searching its entire content in one clock cycle. This makes them very fast and highly appropriate for use in switching tables. However, their complexity also makes them very expensive to produce. Second, this segment protection design requires the controller to issue a large number of flow commands when the data path is changed, since backup ports on each switch have to be recalculated. This leads to an increased load on the CPU and in turn to increased power consumption in the controller.

In this paper we propose a failure protection design, originally introduced in [15,16], which addresses many of the disadvantages of segment protection. The proposed design reduces the forwarding table size by reducing the number of flow entries, compared to the segment protection design reported in [14], while maintaining fast protection times. In the proposed design, two uncorrelated, working, and transient planes are adopted for the data forwarding plane. A working path on the working plane controls routing between source and destination when no failure occurs. The transient plane, which is permanently stored in each switch, provides routing for any case of failure to take care of on-the-fly packets that are already on route to the destination at the time of failure. Packets are then routed through a backup path on the working plane after the configuration is done. Results from the numerical analysis and simulations show a reduction in the number of flow entries of about 60% and in the number of configuration messages of about 75%. Furthermore, we have implemented this design on a pan-European OpenFlow network achieving protection times of less than 25 ms.

II. OpenFlow-Based Segment Protection

Segment protection schemes are designed to provide bypass backup paths for any possible link failure between adjacent nodes in advance of any failure occurring. This has the advantage that segment protection designs have greatly reduced switchover times compared to other protection schemes since the predefined backup path is already available when a failure occurs.

In a segment protection design all switches on the network should contain a working and backup data port for all links. The working port is used during normal operation, and the backup port is only used in the event of a link failure. An interesting OpenFlow-based segment protection scheme was developed in [14], which maintains two flows at different priority levels for each working and backup path needed in the network. The priority levels ensure that the working path is used by default and that the backup path is only used in the event of a link failure on the primary port. When a link failure occurs, an *auto-reject* mechanism on switches connected to the failed link removes the corresponding flow entries. This leaves the lower priority backup port flow, provided that it too has not been affected by the failure, to divert the traffic to the backup path. The switches then send their port status to the controller to update the network topology. Thus, the backup flow entries guide packets from the switch connected to the failed link to a destination via a predefined backup path that corresponds to the failure.

It should be noted that an OpenFlow switch forwards a packet based on a match field that can contain a large number of parameters, including fields in headers at layers 2, 3, and 4 of the OSI stack, in addition to physical port matching. While a detailed match field (e.g., considering IP addresses and TCP ports) is useful for certain applications, segment protection makes use of port matching, as this allows aggregating a large number of flows into one table entry, while using only port-matching information to bypass the failed link. Thus OpenFlow allows applying a "wildcard" to files in the header of layer 2, 3, and 4 protocols.

Figure 2 shows an example of the segment protection design described in [14]. The working path between $H1$ and $H2$ in Fig. 2(a) is A, B, C, F. Three backup paths are provided for each link failure along the working path. A backup path BP1 is provided for link failure between A and B. A backup path BP2 is provided for link failure between B and C. A backup path BP3 is provided for link failure between C and F. If the working path is changed to A, G, H, F, as shown in Fig. 2(b), the backup paths have to be recalculated. A backup path BP4 is provided for link failure between A and G. A backup path BP5 is provided for link failure between G and H. A backup path BP6 is provided for link failure between H and F. The controller reconfigures flows by removing 13 flows and adding 14 flows overall

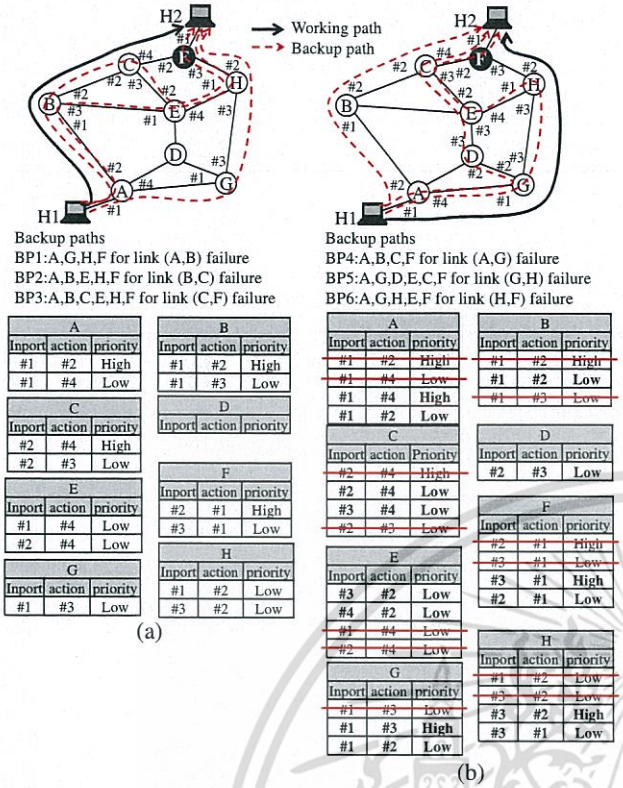


Fig. 2. Example of the configuration for each switch in the segment protection design. (a) Working path is A,B,C,F; (b) working path is A,G,H,F.

in the switches, bringing the total number of required flow commands to 27.

III. Proposed Independent Transient Plane Design

The proposed independent transient plane (ITP) design provides segment protection together with ease of path configuration. It consists of working and transient planes, which are used at different stages of the failure recovery action, as shown in Fig. 3.

The working plane is used as a data path to route packets from source to destination and contains information about both the primary paths and disjointed backup paths, one backup path for each primary path. The primary path is used for normal network operations, while the backup path replaces the corresponding primary path when it is brought down by a failure.

The transient plane, where the routing is statically predefined to support all failure cases, is only temporarily used while the backup path is being configured. A shortest path is considered as a routing policy. Without the transient plane, on-the-fly packets have to wait for forwarding instruction at the switch connected to the failed link until exceeding a *hard timeout* timer. The transient plane is only temporarily used to route packets that are already en route to their destination. As soon as the controller is informed of the failure the backup path is installed in the switches so that all new packets entering the network will follow the new backup route.

The operation of the ITP design is as follows. During normal operation, packets are forwarded to the destination using configurations of the primary path on the working plane, shown in Fig. 3(a). When a link failure occurs, switches connected to the failed link detect the failure and send a port status to the controller. The controller adds a flow entry to the switch as a means to move packets to the transient plane, as shown in Fig. 3(b). Such a connection deflects incoming packets that attempt to pass the failed link to the transient plane via a backup port of the local switches. The flow entry of that connection replaces a tag (either a VLAN or a MPLS can be used) in the packet header with one representing a new link ID. The flow entry of the connection to the transient plane also identifies the backup output port of the switch. Figure 4 shows an example: suppose the link between switch-B (SW_B) and SW_C fails. The tag of packets with the destination SW_C is replaced with an ID (say 101), and the packets are sent to backup port #2 at SW_B. After the packet arrives at SW_A, it is forwarded to port #2 via the transient plane. It should be noted that the link ID is used by the transient plane to avoid sending packets to the failed link.

As described above the ITP uses OpenFlow as is, without any additional extensions. Therefore, the flow message from the controller to move packets from the working plane to the transient plane is required. However, an extension mechanism such as *auto-rejection* in [14] can be used to add

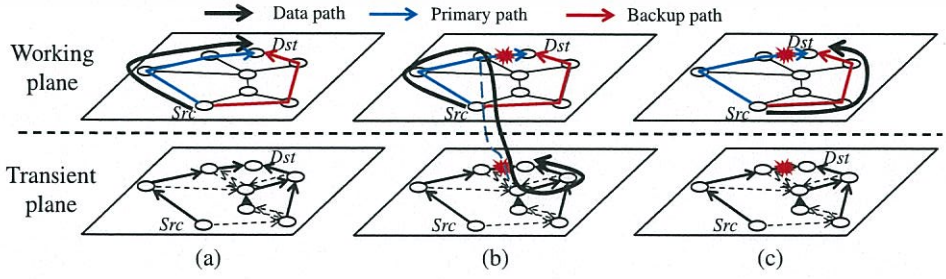


Fig. 3. Actions when link failure occurs in ITP design. (a) Packets travel on the primary path before failure; (b) controller adds a connection to the transient plane to switch, and packets travel via the transient plane before configuration of the backup path is done; (c) packets travel on the backup path after the configuration is done.

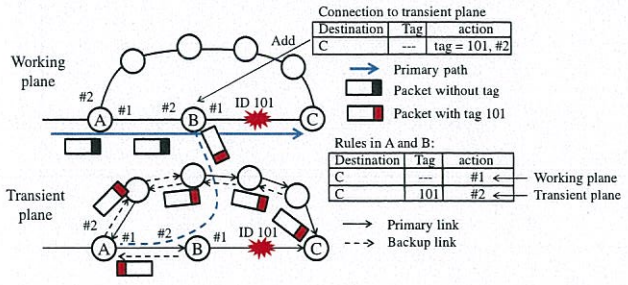


Fig. 4. Example of using a tag.

the flow, instead of the controller, before reporting the port status to the controller.

The transient plane guides on-the-fly packets to their destination. The switches then generate an OFPPR_DELETE port status message and send it to the controller to update the topology. After that, the controller changes the data path from the primary path to a predefined backup path, by sending OFPFC_ADD messages to set up the backup path and OFPFC_DELETE_STRICT messages to remove the primary path.

The ITP addresses the three main disadvantages of segment protection mentioned above. First, The ITP backup path is completely independent of the working path. A single backup path exists for every working path. In segment protection, multiple backup paths were needed depending on where the failure occurred. The transient plane replaces the need for multiple backup paths. This means that changes can be made to the working path without making changes to the backup path. Second, the transient plane can be designed to avoid congestion on the network. Since a single transient plane routing is used for any failure it is easier to avoid congestion than it is with single flow backup paths used in segment protection. Finally, thanks to the fact that a single transient plane is used for all failures there is a decrease in the number of flows in each switch in a large network with multiple source and destination pairs. In addition, the number of controller messages needed to recover the network from failure also decreases. We will quantify this in the results section.

IV. Configuration for the Transient Plane

There are three configurations needed in each switch: the working plane, the transient plane, and the connection between those two planes. The configuration for the working plane is needed for the switches along the primary path when no failure has occurred and along the backup path when a failure has occurred. The configuration for the transient plane is needed for every switch in the network. The configuration for the connection between those two planes is needed only in the switches connected to the failed link. All configurations are categorized based on OpenFlow prioritization. The configurations are as follows:

1. *Configuration on the working plane*: Source and destination IP addresses and priority value W are considered as matching parameters for a flow entry. Figure 5(c) shows flow entries in each switch of the primary path (highlighted in black), which are A, B, C, and F, as preassigned in Fig. 5(a). After a link failure occurs between C and F, the primary path is removed and the preassigned backup path, as shown in Fig. 5(a), is configured by the controller.
2. *Configuration on the transient plane*: The transient plane is based on a tree topology, which we refer to as the transient tree. The transient tree is used to guide packets from any switch in the network to a specific destination switch, called the root, when a failure occurs. The transient tree is actually created by overlapping a number of simpler trees, which we refer to as the common tree and multiple link-failure trees.

The common tree represents the network with no failures and uses the shortest path from each switch in the network to the root destination node. As shown in Fig. 6, switch F is assumed to be the root. A common tree for this topology is shown in Fig. 6(b). Each link on the common tree is called a primary link.

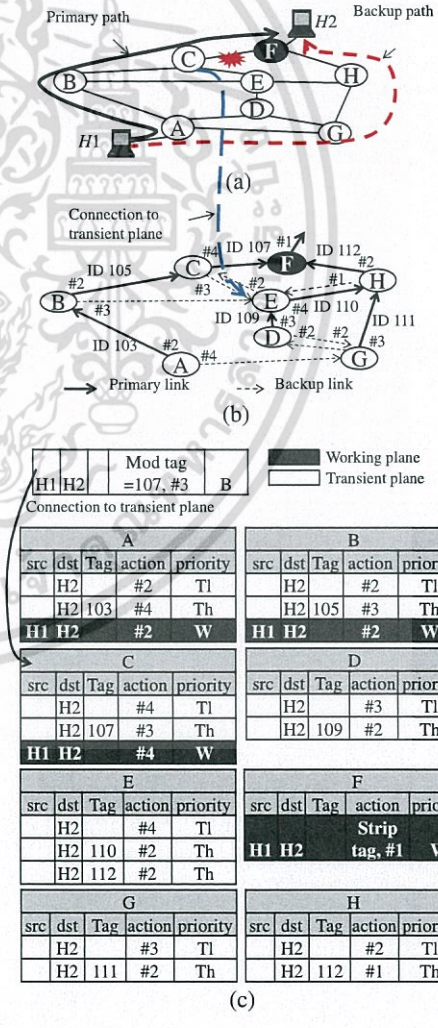


Fig. 5. Example of configuration for each switch in the ITP design. (a) Routing on working planes; (b) routing on transient planes; (c) flow tables in each switch.

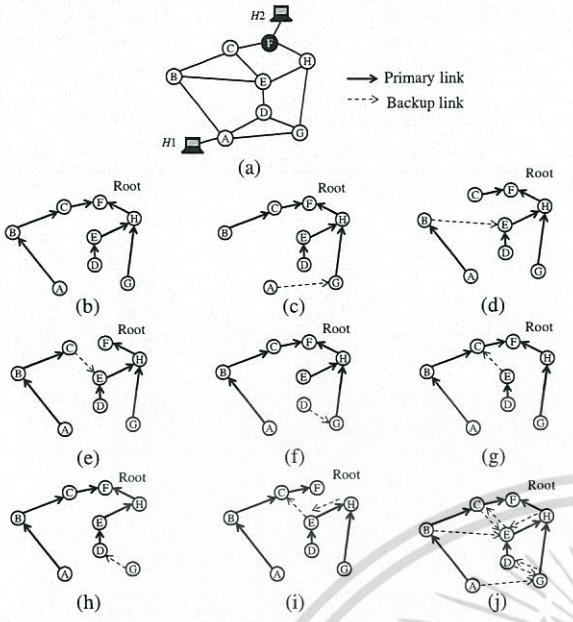


Fig. 6. Creating the transient tree for root F. (a) Network topology; (b) common tree; (c) link-failure tree for (A,B); (d) link-failure tree for (B,C); (e) link-failure tree for (C,F); (f) link-failure tree for (D,E); (g) link-failure tree for (E,H); (h) link-failure tree for (G,H); (i) link-failure tree for (H,F); (j) transient tree for root F.

A link-failure tree takes possible link failures into account and must be calculated for every primary link. The link-failure tree is calculated from the common tree by removing a primary link and finding a shortest path from a switch connected to the failed link to the root. The calculation is repeated until every primary link is considered as the failure. For example, let (A,B) be a link between switches A and B. It should be noted that if the link (A,B) fails, packets at the switch A cannot be received from or sent to switch B via (A,B). As shown in Fig. 6(c), when link (A,B) fails, switch A deflects packets to switch G, and primary links from switch G to the root are used as routing. In Fig. 6(d), when link (B,C) fails, switch B deflects packets to switch E, and primary links from switch E to the root are used as routing. The transient tree, Fig. 6(j), merges the common tree and all the link-failure trees so that routing is available for every possible failure in the network. The process of creating transient trees is repeated until every switch is considered as the root.

In the transient plane, destination IP addresses, a tag, and priority are used as matching parameters for each flow entry. Two priorities, T_h and T_l, are used for the transient tree, where $T_h > W > T_l$. T_h reroutes the packet to avoid the failed link via the backup link. Otherwise, T_l guides packets to the root via primary links. Figure 5(c) shows the configuration of the transient plane (highlighted with white) in each switch.

3. *Configuration of the connection to the transient plane*: This connection is inserted by the controller to a local switch when the controller receives port status after the failure from the switch to modify a tag and to deflect the packets to the transient plane via a backup output port. Source and destination IP addresses and a priority B, where $B > T_h > W > T_l$, are considered as patching parameters for the flow entry of the connection. Figure 5(c) shows the flow entry of the connection when link (C,F) fails.

V. Number of Flow Entries Analysis

Since the objective of this work is to reduce the number of flow entries required for protection and the number of configuration messages when link failure occurs, we report a comparison of those numbers in both the segment protection and the ITP designs. We based our analysis on a $n \times n$ grid topology due to its suitability to carry out analytical studies.

A. Analysis of Segment Protection Design

The total number of flow entries needed in the network is the sum of the flow entries needed by every source–destination pair. Some source–destination pairs will only affect the flow tables of a subsection of the grid topology, and so it can be useful to break the grid into submatrices for the calculation.

For a generalized case, let us assume an $n \times n$ matrix defining the network topology, and a subset of this topology identified by the submatrix $i \times j$. Each cell of the matrix identifies a network node where a switch SW is located. Let $\mathrm{SW}_{i,j}$ be an index for a switch in the submatrix, where $i,j \le n$ and $j \le i$. Figure 7 shows an example of the configuration of each switch for sending packets from $H1$ to $H2$. $H1$ connects to a source switch $\mathrm{SW}_{i,1}$, which is shown in gray. $H2$ connects to a destination switch $\mathrm{SW}_{1,j}$, which is shown in black. The number in each switch represents the number of flow entries needed for source $H1$ and destination $H2$.

Let $F_{i,j}$ be the number of flow entries, including primary and backup paths, needed in the network to route packets from source $\mathrm{SW}_{i,1}$ to destination $\mathrm{SW}_{1,j}$. It is assumed that the primary path travels along the border of the submatrix, and the backup path is designed as in Fig. 8. Figure 8 shows examples of $F_{i,j}$ when i is 2, 3, and 4 and j is 2, 3, and 4. Let $P_{i,j}$ be the number of source–destination pairs on submatrix $i \times j$. $F_{i,j}$ and $P_{i,j}$ are expressed by

$$F_{i,j} = \begin{cases} 6 + 4(i-2); & j = 1 \\ 8 + 4(i-2); & j = 2, \\ 16 + 4(i-3) + 4(j-3); & j \ge 3 \end{cases} \quad (1)$$

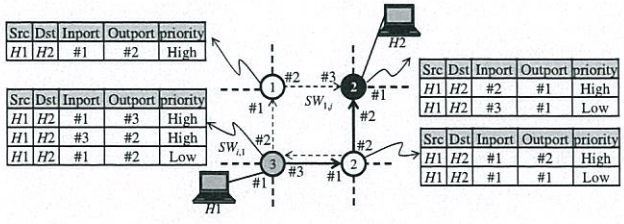


Fig. 7. Flow entries in each switch for segment protection.

$$P(i,j) = \begin{cases} (n-i+1)\times(n-j+1)\times 4; & j=1 \text{ or } i=j \\ (n-i+1)\times(n-j+1)\times 8; & \text{otherwise} \end{cases}. \quad (2)$$

The constants 6 and 8 in Eq. (1) represent the number of flow entries in the switches at the edge when $j=1$ and $j=2$, respectively. For example, in the 2×2 submatrix in Fig. 8 the sum of the flows in the switches is 6 where $j=1$ and 8 where $j=2$. The variable term given by $4(i-2)$ in Eq. (1) represents the number of extra flow entries needed for each new row you add to the submatrix; i.e., for each new row added to the submatrix, four new flow entries must be written to switches in the network. When $j\geq 3$, the number of flows at edge and corner switches is 16. This time two variable terms are added to the equation $4(i-3)$ and $4(j-3)$. These represent the number of flow entries at intermediate switches on horizontal and vertical planes, respectively.

In Eq. (2), $(n-i+1)$ and $(n-j+1)$ represent the total number of pairs from (1,1) to (i,j) in submatrix $n\times j$ and $i\times n$, respectively. The constant multiplier is caused by the fact that each submatrix can be mirrored and transposed. In addition, bidirectional communication is required so each destination is also a source and *vice versa*, which gives $2\times 2\times 2=8$; see Fig. 9, where $P_{2,1}$. In the case of $i=j$, we only consider mirroring and transposing of submatrices, since link bidirectionality can be represented as a matrix transposition. Therefore, ×4 is used in cases of $j=1$ or $i=j$; see Fig. 9, where $P_{2,2}$.

The total number of flow entries in the network with matrix $n\times n$, $T_{\mathrm{SEG}}(n)$, is obtained by

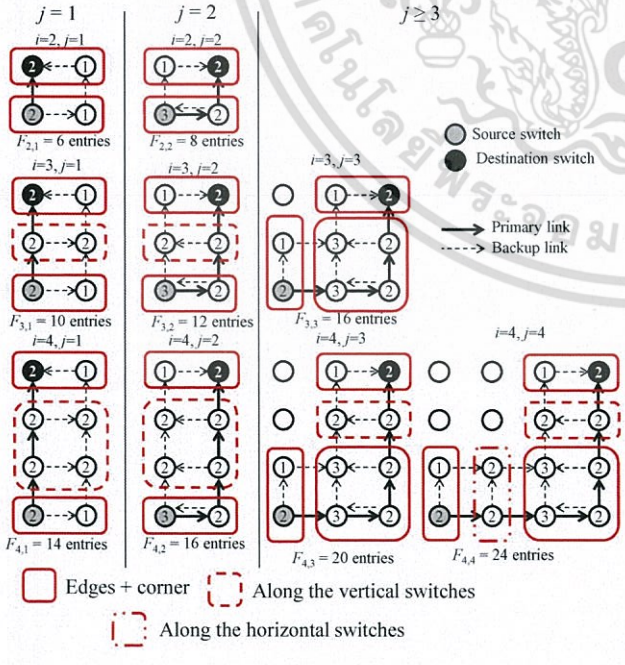


Fig. 8. Examples of $F_{i,j}$ in segment protection. Numbers in the switches represent the number of flow entries stored at that location.

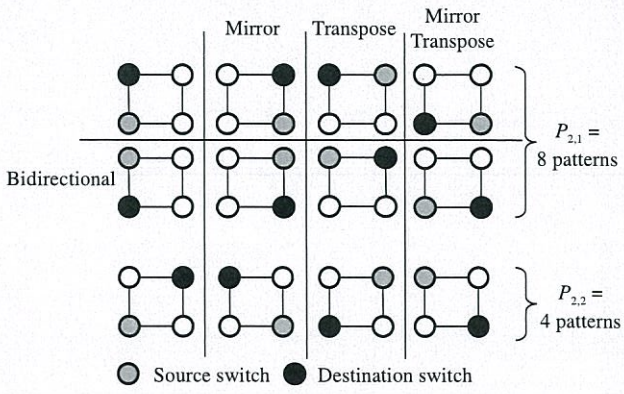


Fig. 9. Examples of $P_{i,j}$, when $i=2$.

$$T_{\mathrm{SEG}}(n)=\sum_{i=2}^{n}\sum_{j=1}^{i}(F_{i,j}\times P_{i,j}). \quad (3)$$

To clarify the analysis, the matrix $n=2$ is used as an example. From Fig. 9 we know the submatrix 2×1 has eight patterns. Each pattern has six flow entries, as calculated in Fig. 8, where $j=1$. Therefore the 2×1 submatrix has 48 flow entries. In the submatrix 2×2, there are four patterns each with eight flow entries giving a total of 32 flow entries. Thus the total number of required flow entries in the matrix where $n=2$ is 80.

B. Analysis of ITP Design

The common and link-failure trees for the transient tree in the transient plane are shown in Fig. 10. The total number of flow entries in the ITP design in the network with matrix $n\times n$, $T_{\mathrm{ITP}}(n)$, is obtained by

$$T_{\mathrm{ITP}}(n)=T_T(n)+T_W(n), \quad (4)$$

where $T_T(n)$ and $T_W(n)$ represent the total number of flow entries for the transient plane and the working plane in the matrix $n\times n$, respectively. $T_T(n)$ and $T_W(n)$ are defined as follows:

$$T_T(n)=n^2\times[1+2n(n-1)+3(n-1)]. \quad (5)$$

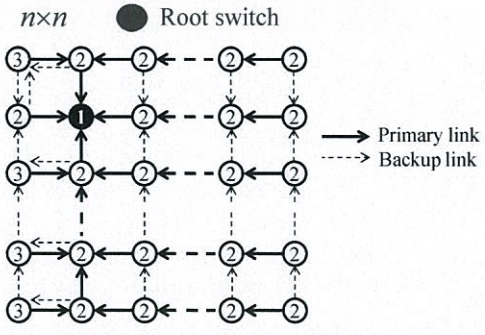


Fig. 10. Number of flow entries in each switch for transient plane in ITP.

For example, Fig. 10 shows the number of flow entries needed by each of the switches when switch (2,2) is considered as the root. From Fig. 10 we see that the root node itself needs only one flow entry, $(n-1)$ switches require three flow entries, and all other switches, i.e., $n(n-1)$, require two flow entries each. This process is then repeated, making each switch, in turn, the root node of the network: this is accounted for by the n^2 multiplier in Eq. (5).

$$T_W(n) = \sum_{i=2}^{n} \sum_{j=1}^{i} (E(i,j) \times P(i,j)), \tag{6}$$

where $P_{i,j}$ is obtained from Eq. (2), and $E_{i,j}$ represents the number of flow entries needed for the working plane to make a connection from $SW_{i,1}$ to $SW_{1,j}$ in submatrix $i \times j$, which is defined by

$$E_{i,j} = i + j - 1. \tag{7}$$

VI. Performance Analysis

In this section we compare the performance of our ITP design with that of the segment protection design presented in [14]. The comparison is carried out over a grid network topology with $2 \le n \le 8$.

First, the performance of both designs is evaluated in terms of total number of flow entries in the network, which is counted after the configurations of the primary path and the transient tree are completed. Figure 11 shows that for all grid topologies tested the ITP design requires fewer flow entries than the segment protection scheme. From Fig. 11 we can see that ITP requires 25% fewer flow entries than segment protection in a 2×2 grid. The benefits of ITP are even greater in larger networks with a 60% reduction in flow entries needed in the network when ITP is used instead of segment protection in an 8×8 grid. Figure 12 shows the percentage of flow entries reduction of the ITP design compared to that of segment protection, which is above 50% for topologies counting as few as 30 switches. This shows that our ITP mechanism can significantly reduce the occupation of flow table resources in the switch for most practical scenarios.

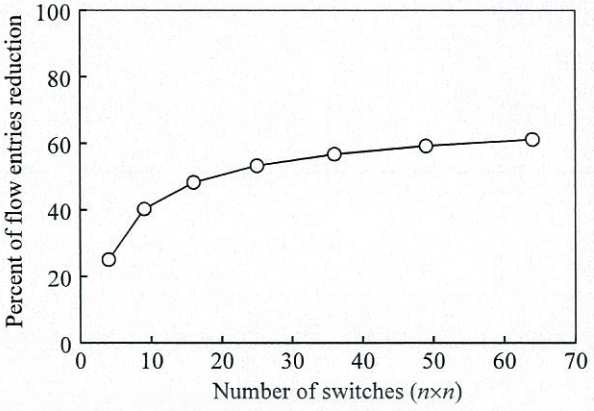


Fig. 12. Percentage of flow entries' reduction.

Finally, the number of configuration messages is evaluated when the primary path is changed to the backup path. Figure 13 shows that the number of configuration messages in the ITP design is 75% smaller than that of the segment protection design. For example, when $n \times n = 64$, only 28 configuration messages are needed in the ITP design, while the segment protection design requires 112 configuration messages. The reduction of configuration messages required directly translates into lower computational resources consumed.

The analysis of the number of flow entries considers the case in which each OpenFlow switch uses TCAMs to store all of the flow entries. However, some switch vendors might implement different memory architectures. For example, some strategies are implemented for which only active flow entries lie in TCAM and other inactive flow entries lie in a non-TCAM memory. In this case, the number of active flow entries in both segment protection and the ITP design are the same, since the working paths of both designs are the same, and only one flow entry is needed in each switch for each active flow. Therefore, the only improvement brought by ITP on switching table size would affect mainly non-TCAM memory. However, ITP still has an advantage in reducing the message processing overhead. The analysis of the number of flows for non-TCAM in this case is almost

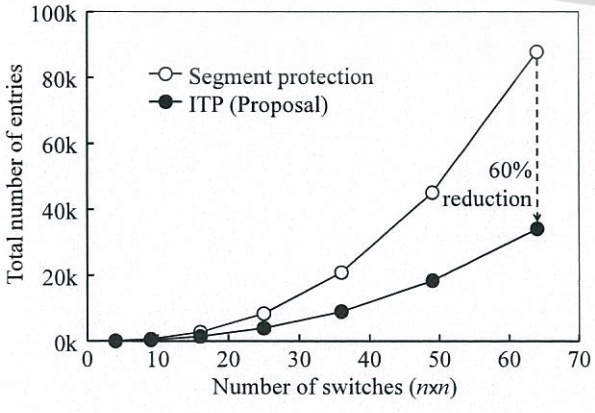


Fig. 11. Total number of flow entries in the network.

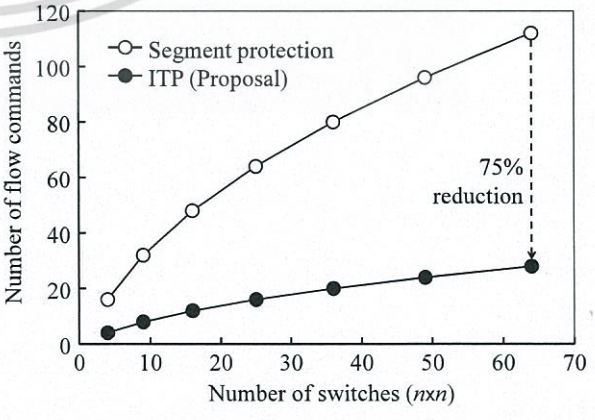


Fig. 13. Number of configuration messages when primary path is changed.

the same as the case in which TCAM is used to store all flows.

Moreover, we evaluate the performance of the ITP design with nonsymmetrical topologies in Fig. 14, which consists of 7 and 14 nodes, and considering every possible source–destination pair. We evaluate the number of flow entries in both the segment protection and the ITP design by simulation. A shortest path policy is used as routing for each source and destination pair. With the topology in Fig. 14(a), the total number of flow entries needed in the network is 447 flows in the segment protection. In the ITP design, the number of flow entries for the transient plane and the working plane is 124 and 149 flows, respectively. The total number of flow entries in the ITP design is 273 flows, which is 39% reduced compared to the segment protection. With the topology in Fig. 14(b), the total number of flow entries needed in the network is 2271 flows in the segment protection. In the ITP design, the number of flow entries for the transient plane and the working plane is 509 and 593 flows, respectively. The total number of flow entries in the ITP design is 1102 flows, which is 51.5% reduced compared to the segment protection. This shows that the advantages of ITP still hold for irregular, more realistic, topologies.

VII. Mininet Emulation Results

In the previous section we have shown that the ITP design requires fewer flow entries and configuration messages than segment protection designs. In this section we use a Mininet emulator [17] to confirm functionality of the ITP design. In this implementation, the working plane, ports, and transient plane information are stored in the controller database in three tables: *Paths*, *Port_map*, and *Transient*, respectively, as shown in Fig. 15. The *Paths table* stores the primary and backup paths for source and destination IP address pairs by identifying switch IDs along the paths from the source to the destination. The *Port_map table* keeps a record of backup ports and link ID. The *Transient table* stores routing information for all the switches to each destination including alternate routes to avoid failed links. In this example we implement the network shown in Fig. 5. The IP addresses of $H1$ and $H2$ are set to 10.0.0.1 and 10.0.0.2, respectively. Priorities used in the experiment are $B = 20{,}000$, $T_h = 15{,}000$, $W = 10{,}000$, and $T_l = 1000$. POX is used as a controller.

Our protection scenario operates as follows. UDP packets are sent from $H1$ to $H2$. A link failure occurs between switches C and F. When the switches detect the link

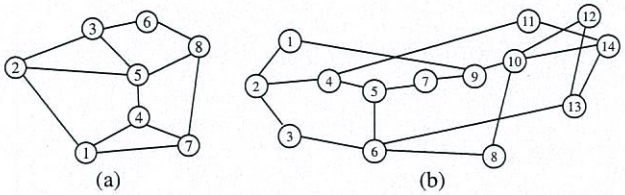


Fig. 14. Network topologies. (a) Sample topology; (b) NSF network topology.

Table Name = *Paths*

src_IP	dst_IP	pri_path	bk_path	
10.0.0.1	10.0.0.2	H1,A,B,C,F,H2	H1,A,G,H,F,H2	×

Source IP:
Destination IP:
Specified switch on working path:
Specified switch on backup path:
Add

Table Name = *Transient*

sw_id	prio	dst_ip	vlan	outport	output_option	
A	15000	10.0.0.2	103	4		×
A	1000	10.0.0.2		2		×
B	15000	10.0.0.2	105	3		×
B	1000	10.0.0.2		2		×
C	15000	10.0.0.2	107	3		×
C	1000	10.0.0.2		4		×
D	15000	10.0.0.2	109	2		×
D	1000	10.0.0.2		3		×
E	15000	10.0.0.2	110	2		×
E	15000	10.0.0.2	112	2		×
E	1000	10.0.0.2		4		×
G	15000	10.0.0.2	111	2		×
G	1000	10.0.0.2		3		×
H	15000	10.0.0.2	112	1		×
H	1000	10.0.0.2		2		×
					strip_vlan	Add

Table Name = *Port_map*

src_IP	dst_IP	sw_src	port	bk_port	vlan_ID	
10.0.0.1	10.0.0.2	A	2	4	103	×
10.0.0.1	10.0.0.2	B	2	3	105	×
10.0.0.1	10.0.0.2	C	4	3	107	×
10.0.0.1	10.0.0.2	D	3	2	109	×
10.0.0.1	10.0.0.2	E	4	2	110	×
10.0.0.1	10.0.0.2	G	3	2	111	×
10.0.0.1	10.0.0.2	H	2	1	112	×
						Add

Fig. 15. Tables in the database.

failure, they generate an OFPPR_DELETE port status message and send it to the controller. The controller retrieves the failed link ID and backup port number from the *Port_map table*. The controller then reroutes packets that will be affected by the failure from the working plane to the transient plane by sending an OFPFC_ADD message (which adds a flow entry as a connection to the transient plane) to switches C and F. This flow entry deflects incoming packets directed toward the failed link to the transient plane via the backup ports on the switches. This message also adds a flow tag to the packet header with the failed link ID that is used to identify the failed link and route packets on the transient plane. The controller then sets up the permanent backup path by selecting a backup path from its *Paths table* and configuring the corresponding switches. It also removes the old primary path.

When the controller starts, the working path and the transient plane are read from the *Path table* and *Transient table*, respectively. The controller configures each switch using command "*ovs-ofctl dump-flows xx*," where *xx* is the switch_id. Flows for the transient plane are added into each switch except switch F, which is the root. The working

plane is added to switches A, B, C, and F, which are the working path. During normal operations, 575 packets travel along the working path, as shown in Fig. 16.

The link between C and F is then broken by the command "link C F down" on Mininet. After the link failure, switch C adds the connection to the transient plane, which is the flow with priority 20,000. It should be noted that the connection to the transient plane should also be added at switch F. However, we omit this step since this simulation sends data in a single direction. The connection to the transient plane is added to switch C. Flows for the working path are removed from switches A, B, and C. Flows for the backup path are added to switches A, G, and H. At switch C, we observe 48 packets traveling to the transient plane via backup port #3. These packets travel to $H2$ via C, E, H, F, thus proving the functionality of the transient plane. After the backup path is activated, 2100 packets travel on the backup path via A, G, H, F to $H2$, as shown in Fig. 17, showing that the network has switched over to the backup path. We also confirm that there is no packet loss during the experiment by verifying the packet sequence number at $H2$. All packets are received at $H2$. However, some packets arrive at the destination out of sequence during the switchover. This is caused by the configuration delay and propagation delay that occur when the path is changed.

Moreover, the performance of topologies as in Fig. 14 with the number of source–destination pairs is investigated in both functionality and switchover time. Eight and 14 source–destination pairs are generated for the sample and NSF topologies, respectively. In the simulation, computers with Intel Core i7-3517U CPUs at 1.90 GHz with 8 GB of RAM are used to run Mininet. In the sample topology, link failure between nodes 3 and 6 is simulated. The ITP function performs correctly after the failure. The switchover time is measured, which is 47.2 ms as in Fig. 18(a), on the source–destination pair between 1 and 6. In the NSF network topology, link failure between nodes 9 and 10 is simulated. The ITP function also performs correctly after the failure. The switchover time is measured, which is 66.4 ms as in Fig. 18(b), on the source–destination pair between 1 and 14. Furthermore, we confirm that the switchover time does not vary appreciably when increasing the number of source–destination pairs. When testing with 1, 14, and 28 of the source–destination pairs in the NSF network topology we obtained switchover times of 66.11, 66.40, and 67.97 ms, respectively.

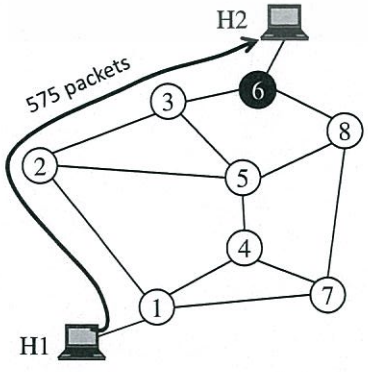


Fig. 16. Packet analysis before failure.

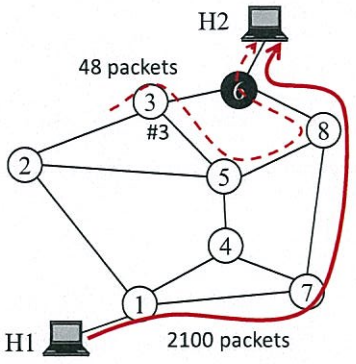
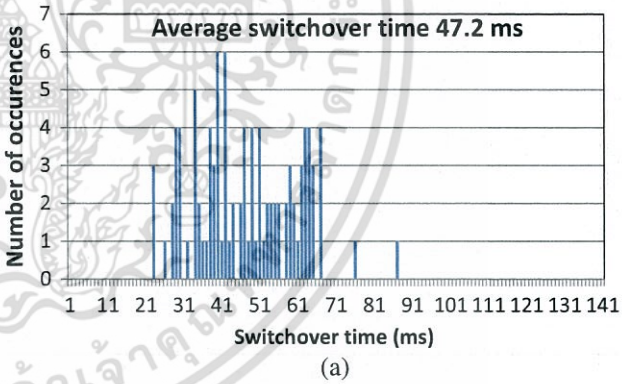


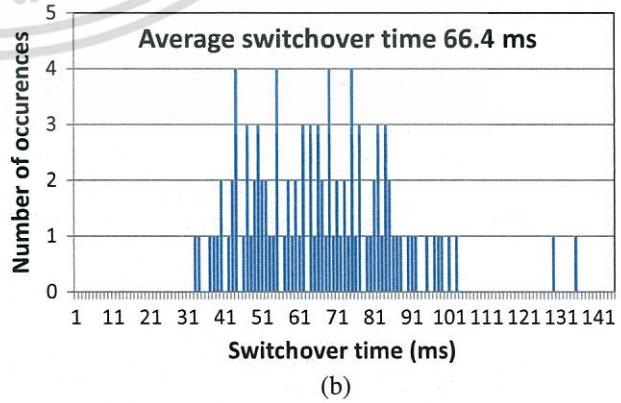
Fig. 17. Packet analysis after failure.

VIII. Testbed Experimental Results

Having proven the functionality of the ITP design with Mininet, we devised an experiment to measure the switchover time of the ITP on a real network. In fact, although Mininet can emulate link latency by configuring link propagation delay parameters, a realistic switchover time may not be obtained since there are other parameters that are not configurable, such as the delay between the



(a)



(b)

Fig. 18. Switchover time measured from Mininet. (a) Switchover time for a sample topology; (b) switchover time for NSF topology.

controller and each switch. Moreover, the computer running Mininet may also be running background processes, which could skew the results. We have thus run an experiment on the pan-European research and education (GÉANT) [18] OpenFlow facility network.

A. Experimental Setup

The GÉANT network nodes used in this experiment are deployed at five different locations around Europe, namely Amsterdam (NL–SW_1), Frankfurt (DE–SW_2), London (UK–SW_3), Vienna (AT–SW_4), and Zagreb (HR–SW_5). The topology we have created is a full mesh and is shown at the top of Fig. 19(a). The SDN controller, implemented in POX, is located at a server in the NL node.

In the scenario we have created for this experiment, as shown in Fig. 19(a), a working path UK → SW_3 → SW_1 → SW_5 → HR will be replaced by a backup path UK → SW_3 → SW_4 → SW_5 → HR when the former fails. Both paths are stored in the *Paths* table. The transient plane is designed as Fig. 19(b). We assume the link failure occurs between SW_1 and SW_5. Port #1 of SW_1 is set as a backup port. After the link failure, the controller configures the backup path. On-the-fly packets travel from the working plane to the transient plane via the connection to the transient plane during the configuration. We capture such packets at the DE node, while packets traveling on the backup route are captured at the AT node.

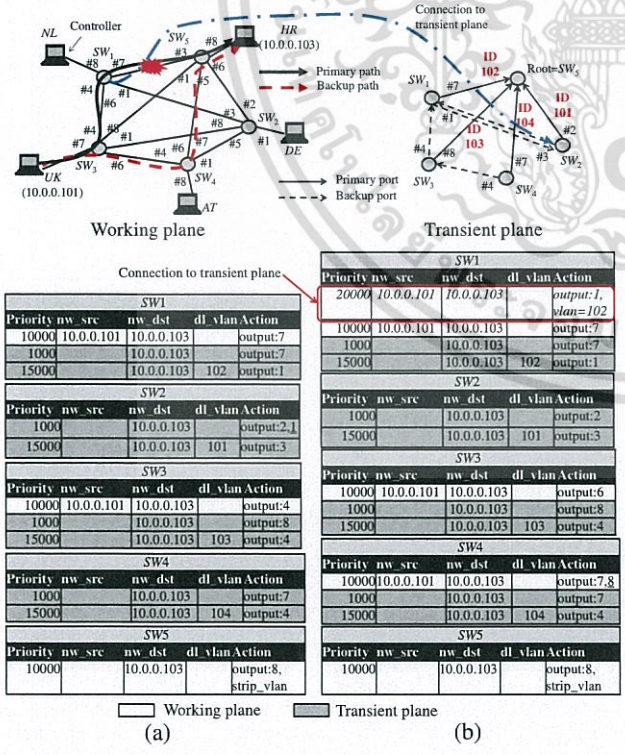


Fig. 19. GÉANT network setup showing the number of configuration message when the primary path is changed. (a) Flow tables before link failure; (b) flow tables after link failure.

In the figure we show for simplicity only a number of parameters, such as priority, IP address (nw_src), destination IP address (nw_dst), VLAN (dl_vlan), and actions, although in reality other parameters of the MAC, MPLS, IP, and TCP protocols could also be used for flow matching.

The initial flow table in every switch is set as in Fig. 19(a). Flow entries for the transient plane are set in every switch with priorities of 1000 and 15,000. A flow entry for the working plane is injected into the switches along the working path only, which in this example is SW_1, SW_3, and SW_5, with priority 10,000.

We simulate a failure by triggering a ping message from the AT node to the controller (we use this artifice because we are not allowed to configure any switch on the GÉANT network by command line). After the controller receives the trigger, a connection to the transient plane is injected into SW_1, as in Fig. 19(b). The backup path is then configured. At SW_3, the output on the flow entry with priority 10,000 is changed from #4 to #6. At SW_4, a new flow entry with priority 10,000 is added. It should be noted that output #1 at SW_2 and output #8 at SW_4 are used for the purpose of packet monitoring at the DE and AT nodes, respectively. The other flow entries remain unchanged.

B. Results and Evaluation

We initially measure the round trip time between the UK and HR nodes, which is approximately 45 and 50 ms, respectively, for the working and backup paths.

Next, we confirm that on-the-fly packets travel to the transient plane when a failure occurs. This is done using a client and server socket program that generates a packet every millisecond at UK with a destination of HR. During normal operation, no packets are captured on the transient plane at DE or the backup path at AT; thus packets are traveling on the working path. After the link failure, 95 packets appear on the transient plane at DE. This confirms that the packets are sent out at port #1 of SW_1 to the transient plane due to the connection to the transient plane. It should be noted that these 95 packets would be lost if the transient plane was disabled. After that, the transient plane at DE receives no more packets, and instead the packets begin to arrive on the backup path at AT. This implies that the packets travel on the transient plane, via SW_2, before the configuration of the backup path and via SW_4 when the configuration is completed. We also check the network for packet loss and find that no packet loss occurs during the protection process.

From this analysis, one and two new flow entries are required for switches on the working and transient planes, respectively. Two flow messages, which change the output from port #4 to port #6 on SW_3 and add an output to port #7 to SW_4, are required to change from the working to backup paths. One flow message is required for the connection to the transient plane on SW_1.

Finally, the switchover time is measured by capturing packets at the HR node. The switchover time is measured

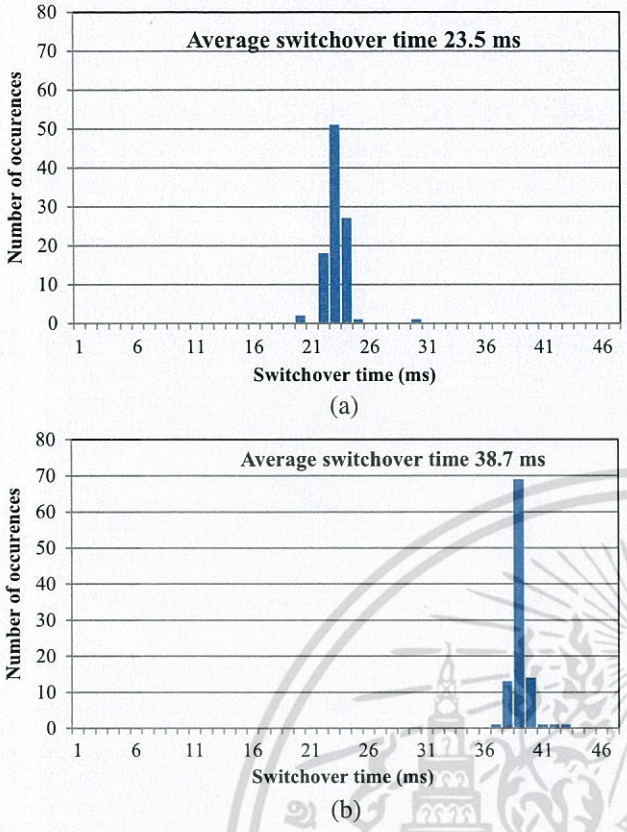


Fig. 20. Switchover time. (a) Switchover time with ITP; (b) switchover time without ITP.

by timing the difference of arrival time between the last received packet before the failure and the first received packet after the failure. The failure between SW_1 and SW_5 is repeated 100 times. Figure 20(a) shows the switchover time distribution obtained using ITP. The distribution is concentrated in the range of 22–24 ms, with an average of 23.5 ms. Figure 21 shows a breakdown of the switchover time. The switchover time includes the time during which packets are queued because of the failure and the difference in propagation delay between $SW_1 \rightarrow SW_5$ and $SW_1 \rightarrow SW_2 \rightarrow SW_5$. Packet queuing times include the controller processing time, the time required to send the configuration of the connection to the transient plane from the controller to SW_1, and the flow installation time at SW_1. This takes approximately 17 ms. The remaining time is the difference in propagation delay between the primary and backup routes, which in this instance is approximately 6.5 ms. Figure 20(b) shows the results of the same experiment carried out using a basic version of segment protection without ITP for comparison. From these results we can see that ITP improves the switchover time by approximately 40% for this network. Furthermore, we have tested the same scenario with a larger number of flows (using five source–destination pairs), obtaining the same results.

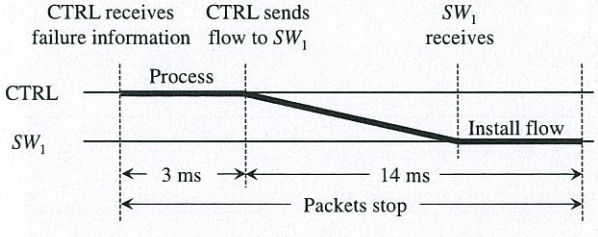


Fig. 21. Time chart for failure processing.

IX. Conclusion

This work presented a novel network protection mechanism, called ITP, implemented through OpenFlow. The advantages of the design are multifold: first it reduces the number of flow entries required in the switches' forwarding tables and the number of configuration messages from the controller, which impacts the size of the required TCAM memory in each switch and CPU process in the controller. Second, it reduces packet loss, as a temporary fast reroute path is made available while the network controller informs all switches about the new backup route to be used. Our analysis shows that our ITP design can reduce the number of flow entries by 60% and the number of configuration messages by 75% when compared to the existing segment protection designs. Moreover, the design can achieve a switchover time of approximately 25 ms.

Acknowledgments

This work is supported in part by the Science Foundation of Ireland through the CTVR CSET grant 10/CE/I1853 and in part by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318137 (Collaborative project DISCUS) [19,20].

References

[1] M. Goyal, K. K. Ramakrishnan, and W.-c. Feng, "Achieving faster failure detection in OSPF networks," in *Proc. IEEE Int. Conf. on Communications (IEEE ICC)*, 2003, pp. 296–300.

[2] Metro Ethernet Forum (MEF), "Requirements and framework for Ethernet service protection in metro Ethernet networks," Tech. Spec. MEF 2, 2004.

[3] J. Zhang, J. Zhou, J. Ren, and B. Wang, "A LDP fast protection switching scheme for concurrent multiple failures in MPLS network," in *Proc. Int. Conf. on Multimedia Information Networking and Security (MINES)*, 2009, pp. 259–262.

[4] L. Hundessa and J. Domingo-Pascual, "Reliable and fast rerouting mechanism for a protected label switched path," in *Proc. IEEE Global Telecommunications Conf. (IEEE GLOBECOM)*, 2002, pp. 1608–1612.

[5] B. Jaumard, N. Nahar Bhuiyan, S. Sebbah, F. Huc, and D. Coudert, "A new framework for efficient shared segment protection scheme for WDM networks," in *Proc. Int. Conf. on High Performance Switching and Routing (HPSR)*, 2010, pp. 189–196.

[6] B. Kantarci, H. T. Mouftah, and S. Oktug, "Availability analysis and connection provisioning in overlapping shared segment protection for optical networks," in *Proc. 23rd Int. Symp. on Computer and Information Sciences (ISCIS)*, Istanbul, Turkey, 2008.

[7] J. Tapolcai, P.-H. Ho, D. Verchere, T. Cinkler, and A. Haque, "A new shared segment protection method for survivable networks with guaranteed recovery time," *IEEE Trans. Reliab.*, vol. 57, no. 2, pp. 272–282, 2008.

[8] D. McDysan, "Software defined networking opportunities for transport," *IEEE Commun. Mag.*, vol. 51, no. 3, pp. 28–31, 2013.

[9] "Software-defined networking (SDN): The new norm for networks," ONF White Paper, Apr. 2012 [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[10] R. Kanagavelu, L. Bu Sung, R. Felipe Miguel, L. N. T. Dat, and L. N. Mingjie, "Software defined network based adaptive routing for data replication in data centers," in *Proc. 19th IEEE Int. Conf. on Networks (IEEE ICON)*, 2013.

[11] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band OpenFlow networks," in *Proc. 9th Int. Conf. on the Design of Reliable Communication Networks (DRCN)*, 2013, pp. 52–59.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," Mar. 2008 [Online]. Available: http://archive.openflow.org/documents/openflow-wp-latest.pdf.

[13] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using OpenFlow: A survey," *IEEE Commun. Surv. Tutorials.*, vol. 16, no. 1, pp. 493–512, 2014.

[14] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow-based segment protection in Ethernet networks," *J. Opt. Commun. Netw.*, vol. 5, no. 9, pp. 1066–1075, 2013.

[15] N. Kitsuwan, F. Slyne, S. McGettrick, D. B. Payne, and M. Ruffini, "A Europe-wide demonstration of fast network restoration with OpenFlow," *IEICE Commun. Express*, vol. 3, no. 9, pp. 275–280, 2014.

[16] N. Kitsuwan, D. B. Payne, and M. Ruffini, "A novel protection design for OpenFlow-based networks," in *Proc. 16th Int. Conf. on Transparent Optical Networks (ICTON)*, Graz, Austria, 2014, paper We.A4.3.

[17] Mininet, http://mininet.org/.

[18] GÉANT network, http://geant3.archive.geant.net/.

[19] M. Ruffini, N. Doran, M. Achouche, N. Parsons, T. Pfeiffer, X. Yin, H. Rohde, M. Schiano, P. Ossieur, B. O'Sullivan, R. Wessaly, L. Wosinska, J. Montalvo, and D. B. Payne, "DISCUS: End-to-end network design for ubiquitous high speed broadband services (Invited)," in *Proc. 16th Int. Conf. on Transparent Optical Networks (ICTON)*, Cartagena, Spain, 2013, paper We.B3.1.

[20] M. Ruffini, L. Wosinska, M. Achouche, J. Chen, N. J. Doran, F. Farjady, J. Montalvo, P. Ossieur, B. O'Sullivan, N. Parsons, T. Pfeiffer, X.-Z. Qiu, C. Raack, H. Rohde, M. Schiano, P. Townsend, R. Wessaly, X. Yin, and D. B. Payne, "DISCUS: An end-to-end solution for ubiquitous broadband optical access," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. S24–S32, Feb. 2014.

Nattapong Kitsuwan received B.E. and M.E. degrees in electrical engineering (telecommunication) from Mahanakorn University of Technology, King Mongkut's Institute of Technology, Ladkrabang, Thailand, and a Ph.D. in information and communication engineering from the University of Electro-Communications, Japan, in 2000, 2004, and 2011, respectively. From 2002 to 2003, he was an exchange student at the University of Electro-Communications, Tokyo, Japan, where he performed research regarding optical packet switching. From 2003 to 2005, he was working for ROHM Integrated Semiconductor, Thailand, as an Information System Expert. He was a post-doctoral researcher at the University of Electro-Communications. He currently works for the Telecommunications Research Centre (CTVR), Trinity College Dublin, Ireland. His research focuses on optical networks, optical burst switching, optical packet switching, scheduling algorithms, and software-defined networks.

Séamas McGettrick (mcgettrs@tcd.ie) is a research fellow at CTVR, the Telecommunications Research Centre at Trinty College Dublin. He is interested in reconfigurable hardware and how it can be used to solve complex real-world problems. He is currently researching and prototyping protocols for next-generation passive optical networks.

Frank Slyne is completing his Ph.D. at Trinity College Dublin in the research area of metro nodes as implemented in a flat-core and long-reach passive optical network (LR-PON) architecture. His interests are in the application of software-defined networks for the optimization of energy, capacity, and performance. He has worked for a number of years at Eircom, where he was responsible for ISP platforms and systems. He has a B.E. (Elect) from UCC (Cork) and an M.Eng. from DCU (Dublin).

David B. Payne spent most of his career with BT, where in his latter years he was responsible for broadband and optical networks research. After leaving BT in 2007, he did consultancy work before joining TCD as a professor in the Department of Optical Networks, and more recently he also joined Aston University. He now coordinates the DISCUS EU project.

Marco Ruffini (marco.ruffini@tcd.ie) received his M.Eng. in telecommunications in 2002, and after working as a research scientist for Philips in Germany he joined Trinity College Dublin (TCD), where he received his Ph.D. in 2007. He is now an assistant professor at TCD and co-coordinator of the DISCUS project. He is a co-author of more than 50 publications and 8 patents. His research focuses on flexible high-capacity fiber broadband architectures.

Measuring of Failure Switch-Over Time in Software-Defined Network

Zhang Hongbo and Nattapong Kitsuwan

Department of Communication Engineering and Informatics,
The University of Electro-Communications, Tokyo, Japan.

***Abstract*—This paper evaluates switch-over time of a failure protection in a software-defined network (SDN). In the area of network protection, it is very necessary to overcome the link failure. This failure may cause the traffic interrupted and the packets are lost in the network. In this case the time of recovery of the failure should be as short as possible. In case of open shortest path first (OSPF) network, it may need a quite long time to detect the topology of network and exchange the network information of routers. Therefore we use software-defined network to make it shorter. We measure the switch-over time in the SDN network to confirm that it is faster than that of the OSPF network. This work is implemented on an OpenFlow emulator, called Mininet. The switch-over time in the SDN network is 35 ms, approximately, as the target protection times is less than 50 ms.**

I. Introduction

Communication is interrupted when at least one link on an active path fails. This situation is called a link failure. In a network the requirement of a switch-over time is 50 ms which is common for leased line traffic and video/audio services [1]. However, the open shortest path first (OSPF) mechanism, which uses a link state to exchange the information between consecutive routers, takes more than 1 second to recover the link failure[2].

Software-defined network (SDN) separates the network control and data planes, as shown in Fig. 1. The control plane moves to a centralize controller and determines how to proceed packets. Meanwhile, data plane forwards packets as decision from the controller. So network engineering/administrator can easily control the network by a software from the controller, regardless of vendor and model of the switch.

The link failure can be quickly recovered in the SDN network. The reason is that the switches that connect to the failed link directly inform the controller. The controller recalculates a new route, and modifies the flow entries in the switches to configure a new route. Unlike the OSPF network, the routers that connect to the failed link exchange the information of with their neighbor routers. The neighbor routers recalculate the route and repeat exchanging the information until all routers in the network receive the information. This process takes very long time, especially in a large scale network.

This paper evaluates the switch-over time when a link failure occurs in the SDN network. A scenario is setup. In the scenario, primary and backup paths are prepared in advance. Once a link on the primary path fails, the communication is switched to the backup path.The switch-over time is measured from a network emulator.

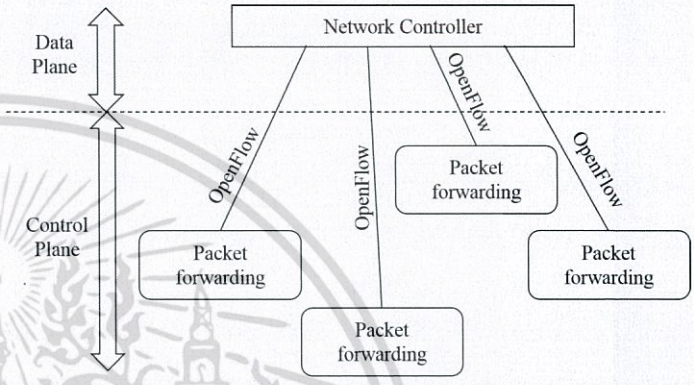


Fig. 1. Software-defined networking (SDN)

II. OpenFlow

OpenFlow is a protocol to communicate between the controller and OpenFlow switches in the SDN network. It is the southbound interface of SDN. The OpenFlow switches use an Asynchronous Message to inform switch status to the controller. Then the controller uses to Controller-to-Switch Messages to send instructions to the switches. The instruction is written into a flow table. The flow table consists of a set of flow entries. Each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets, as shown in Fig. 2.There are some important components of a flow entry.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Fig. 2. Components of flow entry.

Match field: to match against packets. These consist of the ingress port and packet headers, and optionally metadata specified by a previous table.

Priority: matching precedence of the flow entry.

Counter: to update for matching packets.

Instruction: to modify the action set or pipeline processing.

Timeouts: maximum amount of time or idle time before flow is expired by the switch.

978-1-4799-8950-8/16/$31.00 ©2016 IEEE

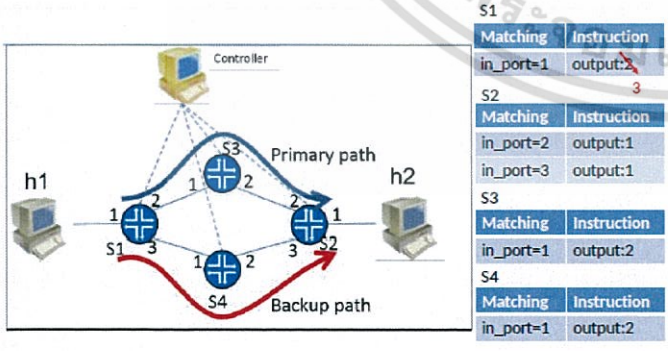
Cookie: opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion, not used when processing packets.[4]

III. Implementation

In this work, a failure protection is implemented in the SDN network. In the implementation, a network emulator, called Mininet[3], is used to build the SDN network. POX is used as a controller. The switches can communicate with the controller. They inform the status of link state to the controller using OpenFlow messages when the link status changes. The controller determines how to proceed the packets when they arrive at switches connected to the the failed link. The decision of the controller is converted into OpenFlow command, and sent to corresponding switches.

In the experimental setup, there are two hosts and four switches. In a normal situation, the communication between the two hosts is via a primary path. Once a link failure occurs, the communication is changed to a backup path. A socket program is used to send user datagram protocol (UDP) packets from the source host to the destination host. When the link of the primary path is failed, the controller sends messages to the switches of the backup path and packets then travel along the backup path.

Figure 3 shows a scenario of the experiment and flow tables. A primary path is via S1, S3, and S2. A backup path is via S1, S4, and S2. At initial configuration, S2, S3, and S4 forward the packets that direct to h2 to their output ports. S1 forwards the packets from port 1 to port 2, S3 forwards from port 1 to port 2, and S2 forwards from port 2 to port 1, via the primary path. So that h1 can send the packets to h2 via the primary path. During the communication, the link between S1 and S3 is manually failed. After the failure, S1 and S3 inform their link status to the controller. The controller changes the path from the primary to the backup path by modifying output port of the flow entry in S1 from output=3 to output=2. The later packets are forwarded through the backup path to h2.



S1

Matching	Instruction
in_port=1	output:2 (3)

S2

Matching	Instruction
in_port=2	output:1
in_port=3	output:1

S3

Matching	Instruction
in_port=1	output:2

S4

Matching	Instruction
in_port=1	output:2

Fig. 3. Scenario and flow entries in each switch.

IV. Performance Evaluation

The packets are captured at h2. A time gap between two consecutive packets is measured. The speed of sending packets from h1 is 1000 packets per second. So before the failure occurs the time gaps are all about 1ms, except several ones are about 5ms. When the failure occurs, 23 packets that already depart from port 2 of S1 are lost (packet number from 4503 to 4525) . After the flow entry in S1 is modified, incoming packets from port 1 of S1 is forwarded to port 3 and then go via the backup path. After the flow entries in backup path are installed the time gaps change to about 1 ms again. The recovery time (35.7 ms) is measured between the last packet before loss and the first packet after the flow entry is modified.

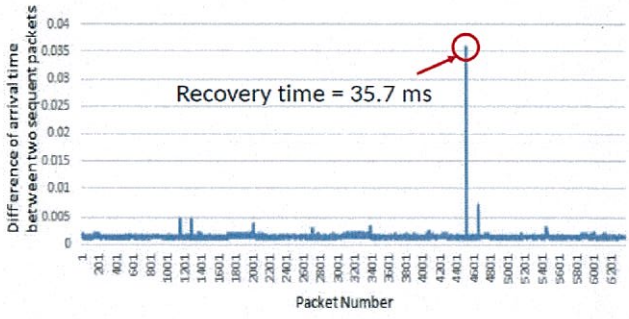


Fig. 4. Time gap between two consecutive packets.

V. Conclusion

Failure protection in SDN was implemented. The result shows that the recovery time is 35.7 ms, which satisfies the requirement of 50 ms.

References

[1] N. Kitsuwan, S. McGettrick, F. Slyne, D.B. Payne, and M. Ruffini, "Independent transient plane design for protection in OpenFlow-based networks," IEEE/OSA Journal of Optical Communications and Networking, vol. 7, iss. 4, pp. 264-275, 2015.
[2] J. Moy, "OSPF Version 2," RFC2328, 1998.
[3] OpenFlow Switch Specification Version 1.3.0, June 2012 [Online]. Available: https://www.opennetworking.org/.
[4] Mininet, [Online], Available: http://mininet.org/.

OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown
Stanford University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Guru Parulkar
Stanford University

Larry Peterson
Princeton University

Jennifer Rexford
Princeton University

Scott Shenker
University of California, Berkeley

Jonathan Turner
Washington University in St. Louis

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
Authors take full responsibility for this article's technical content.
Comments can be posted through CCR Online.

ABSTRACT

This whitepaper proposes OpenFlow: a way for researchers to run experimental protocols in the networks they use every day. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. Our goal is to encourage networking vendors to add OpenFlow to their switch products for deployment in college campus backbones and wiring closets. We believe that OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI. Two buildings at Stanford University will soon run OpenFlow networks, using commercial Ethernet switches and routers. We will work to encourage deployment at other schools; and We encourage you to consider deploying OpenFlow in your university network too.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Routers

General Terms

Experimentation, Design

Keywords

Ethernet switch, virtualization, flow-based

1. THE NEED FOR PROGRAMMABLE NETWORKS

Networks have become part of the critical infrastructure of our businesses, homes and schools. This success has been both a blessing and a curse for networking researchers; their work is more relevant, but their chance of making an impact is more remote. The reduction in real-world impact of any given network innovation is because the enormous installed base of equipment and protocols, and the reluctance to experiment with production traffic, which have created an exceedingly high barrier to entry for new ideas. Today, there is almost no practical way to experiment with new network protocols (e.g., new routing protocols, or alternatives to IP) in sufficiently realistic settings (e.g., at scale carrying real traffic) to gain the confidence needed for their widespread deployment. The result is that most new ideas from the networking research community go untried and untested; hence the commonly held belief that the network infrastructure has "ossified".

Having recognized the problem, the networking community is hard at work developing programmable networks, such as GENI [1] a proposed nationwide research facility for experimenting with new network architectures and distributed systems. These programmable networks call for programmable switches and routers that (using *virtualization*) can process packets for multiple isolated experimental networks simultaneously. For example, in GENI it is envisaged that a researcher will be allocated a *slice* of resources across the whole network, consisting of a portion of network links, packet processing elements (e.g. routers) and end-hosts; researchers program their slices to behave as they wish. A slice could extend across the backbone, into access networks, into college campuses, industrial research labs, and include wiring closets, wireless networks, and sensor networks.

Virtualized programmable networks could lower the barrier to entry for new ideas, increasing the rate of innovation in the network infrastructure. But the plans for nationwide facilities are ambitious (and costly), and it will take years for them to be deployed.

This whitepaper focuses on a shorter-term question closer to home: *As researchers, how can we run experiments in our campus networks?* If we can figure out how, we can start soon and extend the technique to other campuses to benefit the whole community.

To meet this challenge, several questions need answering, including: In the early days, how will college network administrators get comfortable putting experimental equipment (switches, routers, access points, etc.) into their network? How will researchers control a portion of their local network in a way that does not disrupt others who depend on it? And exactly what functionality is needed in network

switches to enable experiments? Our goal here is to propose a new switch feature that can help extend programmability into the wiring closet of college campuses.

One approach - that we do not take - is to persuade commercial "name-brand" equipment vendors to provide an open, programmable, virtualized platform on their switches and routers so that researchers can deploy new protocols, while network administrators can take comfort that the equipment is well supported. This outcome is very unlikely in the short-term. Commercial switches and routers do not typically provide an open software platform, let alone provide a means to virtualize either their hardware or software. The practice of commercial networking is that the standardized external interfaces are narrow (i.e., just packet forwarding), and all of the switch's internal flexibility is hidden. The internals differ from vendor to vendor, with no standard platform for researchers to experiment with new ideas. Further, network equipment vendors are understandably nervous about opening up interfaces inside their boxes: they have spent years deploying and tuning fragile distributed protocols and algorithms, and they fear that new experiments will bring networks crashing down. And, of course, open platforms lower the barrier-to-entry for new competitors.

A few open software platforms already exist, but do not have the performance or port-density we need. The simplest example is a PC with several network interfaces and an operating system. All well-known operating systems support routing of packets between interfaces, and open-source implementations of routing protocols exist (e.g., as part of the Linux distribution, or from XORP [2]); and in most cases it is possible to modify the operating system to process packets in almost any manner (e.g., using Click [3]). The problem, of course, is performance: A PC can neither support the number of ports needed for a college wiring closet (a fanout of 100+ ports is needed per box), nor the packet-processing performance (wiring closet switches process over 100Gbits/s of data, whereas a typical PC struggles to exceed 1Gbit/s; and the gap between the two is widening).

Existing platforms with specialized hardware for line-rate processing are not quite suitable for college wiring closets either. For example, an ATCA-based virtualized programmable router called the Supercharged PlanetLab Platform [4] is under development at Washington University, and can use network processors to process packets from many interfaces simultaneously at line-rate. This approach is promising in the long-term, but for the time being is targeted at large switching centers and is too expensive for widespread deployment in college wiring closets. At the other extreme is NetFPGA [5] targeted for use in teaching and research labs. NetFPGA is a low-cost PCI card with a user-programmable FPGA for processing packets, and 4-ports of Gigabit Ethernet. NetFPGA is limited to just four network interfaces—insufficient for use in a wiring closet.

Thus, the commercial solutions are too closed and inflexible, and the research solutions either have insufficient performance or fanout, or are too expensive. It seems unlikely that the research solutions, with their complete generality, can overcome their performance or cost limitations. A more promising approach is to compromise on generality and to seek a degree of switch flexibility that is:

- Amenable to high-performance and low-cost implementations.
- Capable of supporting a broad range of research.
- Assured to isolate experimental traffic from production traffic.
- Consistent with vendors' need for closed platforms.

This paper describes the OpenFlow Switch—a specification that is an initial attempt to meet these four goals.

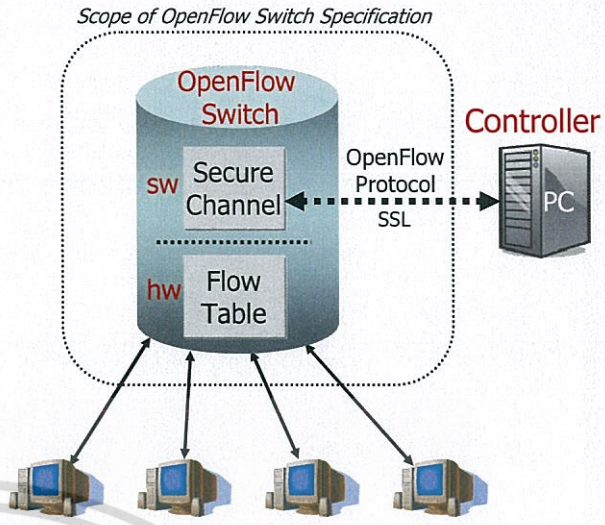


Figure 1: Idealized OpenFlow Switch. The Flow Table is controlled by a remote controller via the Secure Channel.

2. THE OPENFLOW SWITCH

The basic idea is simple: we exploit the fact that most modern Ethernet switches and routers contain flow-tables (typically built from TCAMs) that run at line-rate to implement firewalls, NAT, QoS, and to collect statistics. While each vendor's flow-table is different, we've identified an interesting common set of functions that run in many switches and routers. OpenFlow exploits this common set of functions.

OpenFlow provides an open protocol to program the flow-table in different switches and routers. A network administrator can partition traffic into production and research flows. Researchers can control their own flows - by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP. On the same network, the production traffic is isolated and processed in the same way as today.

The datapath of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry. The set of actions supported by an OpenFlow Switch is extensible, but below we describe a minimum requirement for all switches. For high-performance and low-cost the datapath must have a carefully prescribed degree of flexibility. This means forgoing the ability to specify arbitrary handling of each packet and seeking a more limited, but still useful, range of actions. Therefore, later in the paper, define a basic required set of actions for all OpenFlow switches.

An OpenFlow Switch consists of at least three parts: (1) A *Flow Table*, with an action associated with each flow entry, to tell the switch how to process the flow, (2) A *Secure Channel* that connects the switch to a remote control process (called the *controller*), allowing commands and packets to be sent between a controller and the switch using (3) The *OpenFlow Protocol*, which provides an open and standard way for a controller to communicate with a switch. By specifying a standard interface (the OpenFlow Protocol) through which entries in the Flow Table can be defined externally, the OpenFlow Switch avoids the need for researchers to program the switch.

It is useful to categorize switches into dedicated OpenFlow switches that do not support normal Layer 2 and Layer 3 processing, and OpenFlow-enabled general purpose commercial Ethernet switches and routers, to which the OpenFlow Protocol and interfaces have been added as a new feature.

Dedicated OpenFlow switches. A dedicated OpenFlow Switch is a dumb datapath element that forwards packets between ports, as defined by a remote control process. Figure 1 shows an example of an OpenFlow Switch.

In this context, flows are broadly defined, and are limited only by the capabilities of the particular implementation of the Flow Table. For example, a flow could be a TCP connection, or all packets from a particular MAC address or IP address, or all packets with the same VLAN tag, or all packets from the same switch port. For experiments involving non-IPv4 packets, a flow could be defined as all packets matching a specific (but non-standard) header.

Each flow-entry has a simple action associated with it; the three basic ones (that all dedicated OpenFlow switches must support) are:

1. Forward this flow's packets to a given port (or ports). This allows packets to be routed through the network. In most switches this is expected to take place at line-rate.

2. Encapsulate and forward this flow's packets to a controller. Packet is delivered to Secure Channel, where it is encapsulated and sent to a controller. Typically used for the first packet in a new flow, so a controller can decide if the flow should be added to the Flow Table. Or in some experiments, it could be used to forward all packets to a controller for processing.

3. Drop this flow's packets. Can be used for security, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from end-hosts.

An entry in the Flow-Table has three fields: (1) A packet header that defines the flow, (2) The action, which defines how the packets should be processed, and (3) Statistics, which keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow (to help with the removal of inactive flows).

In the first generation "Type 0" switches, the flow header is a 10-tuple shown in Table 1. A TCP flow could be specified by all ten fields, whereas an IP flow might not include the transport ports in its definition. Each header field can be a wildcard to allow for aggregation of flows, such as flows in which only the VLAN ID is defined would apply to all traffic on a particular VLAN.

In	VLAN	Ethernet			IP			TCP	
Port	ID	SA	DA	Type	SA	DA	Proto	Src	Dst

Table 1: The header fields matched in a "Type 0" OpenFlow switch.

The detailed requirements of an OpenFlow Switch are defined by the *OpenFlow Switch Specification* [6].

OpenFlow-enabled switches. Some commercial switches, routers and access points will be enhanced with the OpenFlow feature by adding the Flow Table, Secure Channel and OpenFlow Protocol (we list some examples in Section 5). Typically, the Flow Table will re-use existing hardware, such as a TCAM; the Secure Channel and Protocol will be ported to run on the switch's operating system. Figure 2 shows a network of OpenFlow-enabled commercial switches and access points. In this example, all the Flow Tables are managed by the same controller; the OpenFlow Protocol allows a switch to be controlled by two or more controllers for increased performance or robustness.

Our goal is to enable experiments to take place in an existing production network alongside regular traffic and applications. Therefore, to win the confidence of network administrators, OpenFlow-enabled switches must isolate experimental traffic (processed by the Flow Table) from production traffic that is to be processed by the normal Layer 2 and Layer 3 pipeline of the switch. There are two ways to achieve this separation. One is to add a *fourth action*:

4. Forward this flow's packets through the switch's normal processing pipeline.

The other is to define separate sets of VLANs for experimental and production traffic. Both approaches allow normal production traffic that isn't part of an experiment to be processed in the usual way by the switch. All OpenFlow-enabled switches are required to support one approach or the other; some will support both.

Additional features. If a switch supports the header formats and the four basic actions mentioned above (and detailed in the *OpenFlow Switch Specification*), then we call it a "Type 0" switch. We expect that many switches will support additional actions, for example to rewrite portions of the packet header (e.g., for NAT, or to obfuscate addresses on intermediate links), and to map packets to a priority class. Likewise, some Flow Tables will be able to match on arbitrary fields in the packet header, enabling experiments with new non-IP protocols. As a particular set of features emerges, we will define a "Type 1" switch.

Controllers. A controller adds and removes flow-entries from the Flow Table on behalf of experiments. For example, a static controller might be a simple application running on a PC to statically establish flows to interconnect a set of test computers for the duration of an experiment. In this case the flows resemble VLANs in current networks—providing a simple mechanism to isolate experimental traffic from the production network. Viewed this way, OpenFlow is a generalization of VLANs.

One can also imagine more sophisticated controllers that dynamically add/remove flows as an experiment progresses. In one usage model, a researcher might control the complete

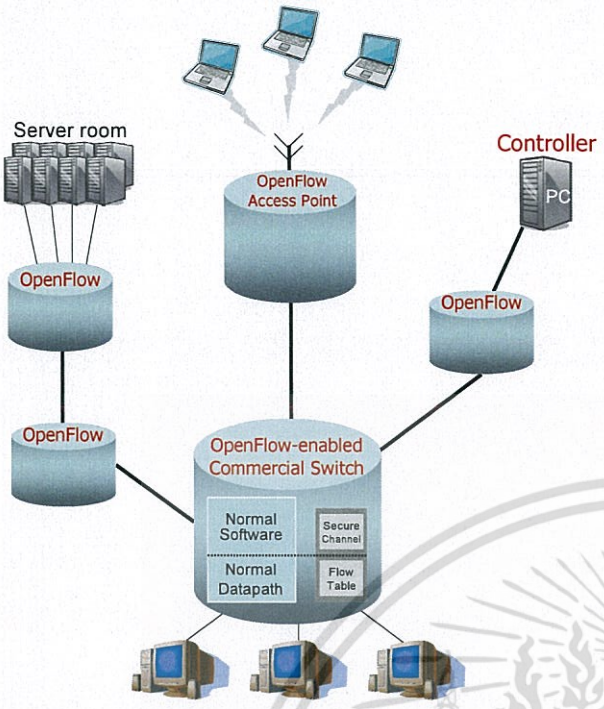


Figure 2: Example of a network of OpenFlow-enabled commercial switches and routers.

network of OpenFlow Switches and be free to decide how all flows are processed. A more sophisticated controller might support multiple researchers, each with different accounts and permissions, enabling them to run multiple independent experiments on different sets of flows. Flows identified as under the control of a particular researcher (e.g., by a policy table running in a controller) could be delivered to a researcher's user-level control program which then decides if a new flow-entry should be added to the network of switches.

3. USING OPENFLOW

As a simple example of how an OpenFlow Switch might be used imagine that Amy (a researcher) invented Amy-OSPF as a new routing protocol to replace OSPF. She wants to try her protocol in a network of OpenFlow Switches, without changing any end-host software. Amy-OSPF will run in a controller; each time a new application flow starts Amy-OSPF picks a route through a series of OpenFlow Switches, and adds a flow- entry in each switch along the path. In her experiment, Amy decides to use Amy-OSPF for the traffic entering the OpenFlow network from her own desktop PC—so she doesn't disrupt the network for others. To do this, she defines one flow to be all the traffic entering the OpenFlow switch through the switch port her PC is connected to, and adds a flow-entry with the action "Encapsulate and forward all packets to a controller". When her packets reach a controller, her new protocol chooses a route and adds a new flow-entry (for the application flow) to every switch along the chosen path. When subsequent packets arrive at a switch, they are processed quickly (and at line-rate) by the Flow Table.

There are legitimate questions to ask about the performance, reliability and scalability of a controller that dynamically adds and removes flows as an experiment progresses: Can such a centralized controller be fast enough to process new flows and program the Flow Switches? What happens when a controller fails? To some extent these questions were addressed in the context of the Ethane prototype, which used simple flow switches and a central controller [7]. Preliminary results suggested that an Ethane controller based on a low-cost desktop PC could process over 10,000 new flows per second — enough for a large college campus. Of course, the rate at which new flows can be processed will depend on the complexity of the processing required by the researcher's experiment. But it gives us confidence that meaningful experiments can be run. Scalability and redundancy are possible by making a controller (and the experiments) stateless, allowing simple load-balancing over multiple separate devices.

3.1 Experiments in a Production Network

Chances are, Amy is testing her new protocol in a network used by lots of other people. We therefore want the network to have two additional properties:

1. Packets belonging to users other than Amy should be routed using a standard and tested routing protocol running in the switch or router from a "name-brand" vendor.
2. Amy should only be able to add flow entries for her traffic, or for any traffic her network administrator has allowed her to control.

Property 1 is achieved by OpenFlow-enabled switches. In Amy's experiment, the default action for all packets that don't come from Amy's PC could be to forward them through the normal processing pipeline. Amy's own packets would be forwarded directly to the outgoing port, without being processed by the normal pipeline.

Property 2 depends on the controller. The controller should be seen as a platform that enables researchers to implement various experiments, and the restrictions of Property 2 can be achieved with the appropriate use of permissions or other ways to limit the powers of individual researchers to control flow entries. The exact nature of these permission-like mechanisms will depend on how the controller is implemented. We expect that a variety of controllers will emerge. As an example of a concrete realization of a controller, some of the authors are working on a controller called NOX as a follow-on to the Ethane work [8]. A quite different controller might emerge by extending the GENI management software to OpenFlow networks.

3.2 More Examples

As with any experimental platform, the set of experiments will exceed those we can think of up-front — most experiments in OpenFlow networks are yet to be thought of. Here, for illustration, we offer some examples of how OpenFlow-enabled networks could be used to experiment with new network applications and architectures.

Example 1: Network Management and Access Control. We'll use Ethane as our first example [7] as it was the research that inspired OpenFlow. In fact, an OpenFlow

Switch can be thought of as a generalization of Ethane's datapath switch. Ethane used a specific implementation of a controller, suited for network management and control, that manages the admittance and routing of flows. The basic idea of Ethane is to allow network managers to define a network-wide policy in the central controller, which is enforced directly by making admission control decisions for each new flow. A controller checks a new flow against a set of rules, such as "Guests can communicate using HTTP, but only via a web proxy" or "VoIP phones are not allowed to communicate with laptops." A controller associates packets with their senders by managing all the bindings between names and addresses — it essentially takes over DNS, DHCP and authenticates all users when they join, keeping track of which switch port (or access point) they are connected to. One could envisage an extension to Ethane in which a policy dictates that particular flows are sent to a user's process in a controller, hence allowing researcher-specific processing to be performed in the network.

Example 2: VLANs. OpenFlow can easily provide users with their own isolated network, just as VLANs do. The simplest approach is to statically declare a set of flows which specify the ports accessible by traffic on a given VLAN ID. Traffic identified as coming from a single user (for example, originating from specific switch ports or MAC addresses) is tagged by the switches (via an action) with the appropriate VLAN ID.

A more dynamic approach might use a controller to manage authentication of users and use the knowledge of the users' locations for tagging traffic at runtime.

Example 3: Mobile wireless VOIP clients. For this example consider an experiment of a new call- handoff mechanism for WiFi-enabled phones. In the experiment VOIP clients establish a new connection over the OpenFlow-enabled network. A controller is implemented to track the location of clients, re-routing connections — by reprogramming the Flow Tables — as users move through the network, allowing seamless handoff from one access point to another.

Example 4: A non-IP network. So far, our examples have assumed an IP network, but OpenFlow doesn't require packets to be of any one format — so long as the Flow Table is able to match on the packet header. This would allow experiments using new naming, addressing and routing schemes. There are several ways an OpenFlow-enabled switch can support non-IP traffic. For example, flows could be identified using their Ethernet header (MAC src and dst addresses), a new EtherType value, or at the IP level, by a new IP Version number. More generally, we hope that future switches will allow a controller to create a generic mask (offset + value + mask), allowing packets to be processed in a researcher-specified way.

Example 5: Processing packets rather than flows. The examples above are for experiments involving flows — where a controller makes decisions when the flow starts. There are, of course, interesting experiments to be performed that require every packet to be processed. For example, an intrusion detection system that inspects every packet, an explicit congestion control mechanism, or when modifying the contents of packets, such as when converting packets from one protocol format to another.

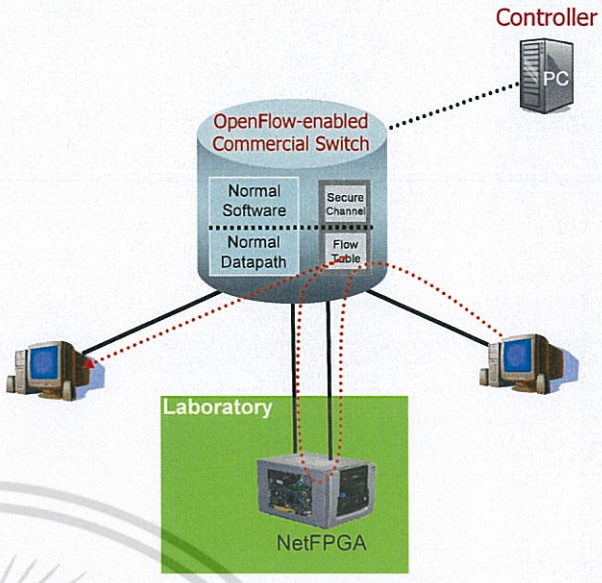


Figure 3: Example of processing packets through an external line-rate packet-processing device, such as a programmable NetFPGA router.

There are two basic ways to process packets in an OpenFlow-enabled network. First, and simplest, is to force all of a flow's packets to pass through a controller. To do this, a controller doesn't add a new flow entry into the Flow Switch — it just allows the switch to default to forwarding every packet to a controller. This has the advantage of flexibility, at the cost of performance. It might provide a useful way to test the functionality of a new protocol, but is unlikely to be of much interest for deployment in a large network.

The second way to process packets is to route them to a programmable switch that does packet processing — for example, a NetFPGA-based programmable router. The advantage is that the packets can be processed at line-rate in a user-definable way; Figure 3 shows an example of how this could be done, in which the OpenFlow-enabled switch operates essentially as a patch-panel to allow the packets to reach the NetFPGA. In some cases, the NetFPGA board (a PCI board that plugs into a Linux PC) might be placed in the wiring closet alongside the OpenFlow-enabled switch, or (more likely) in a laboratory.

4. THE OPENFLOW CONSORTIUM

The OpenFlow Consortium aims to popularize OpenFlow and maintain the *OpenFlow Switch Specification*. The Consortium is a group of researchers and network administrators at universities and colleges who believe their research mission will be enhanced if OpenFlow-enabled switches are installed in their network.

Membership is open and free for anyone at a school, college, university, or government agency worldwide. The OpenFlow Consortium welcomes individual members who are not employed by companies that manufacture or sell Ethernet switches, routers or wireless access points (because we want to keep the consortium free of vendor influence). To join, send email to *join@OpenFlowSwitch.org*.

The Consortium web-site [1] contains the OpenFlow Switch Specification, a list of consortium members, and reference implementations of OpenFlow switches.

Licensing Model: The OpenFlow Switch Specification is free for all commercial and non-commercial use. (The exact wording is on the web-site.) Commercial switches and routers claiming to be "OpenFlow-enabled" must conform to the requirements of an OpenFlow Type 0 Switch, as defined in the OpenFlow Switch Specification. OpenFlow is a trademark of Stanford University, and will be protected on behalf of the Consortium.

5. DEPLOYING OPENFLOW SWITCHES

We believe there is an interesting market opportunity for network equipment vendors to sell OpenFlow-enabled switches to the research community. Every building in thousands of colleges and universities contains wiring closets with Ethernet switches and routers, and with wireless access points spread across campus.

We are actively working with several switch and router manufacturers who are adding the OpenFlow feature to their products by implementing a Flow Table in existing hardware; i.e. no hardware change is needed. The switches run the Secure Channel software on their existing processor.

We have found network equipment vendors to be very open to the idea of adding the OpenFlow feature. Most vendors would like to support the research community without having to expose the internal workings of their products.

We are deploying large OpenFlow networks in the Computer Science and Electrical Engineering departments at Stanford University. The networks in two buildings will be replaced by switches running OpenFlow. Eventually, all traffic will run over the OpenFlow network, with production traffic and experimental traffic being isolated on different VLANs under the control of network administrators. Researchers will control their own traffic, and be able to add/remove flow-entries.

We also expect many different OpenFlow Switches to be developed by the research community. The OpenFlow website contains "Type 0" reference designs for several different platforms: Linux (software), OpenWRT (software, for access points), and NetFPGA (hardware, 4-ports of 1GE). As more reference designs are created by the community we will post them. We encourage developers to test their switches against the reference designs.

All reference implementations of OpenFlow switches posted on the web site will be open-source and free for commercial and non-commercial use.[2]

6. CONCLUSION

We believe that OpenFlow is a pragmatic compromise that allows researchers to run experiments on heterogeneous switches and routers in a uniform way, without the need for vendors to expose the internal workings of their products, or researchers to write vendor-specific control software.

If we are successful in deploying OpenFlow networks in our campusses, we hope that OpenFlow will gradually catch-on in other universities, increasing the number of networks that support experiments. We hope that a new generation of control software emerges, allowing researchers to re-use controllers and experiments, and build on the work of others. And over time, we hope that the islands of OpenFlow networks at different universities will be interconnected by tunnels and overlay networks, and perhaps by new OpenFlow networks running in the backbone networks that connect universities to each other.

7. REFERENCES

[1] Global Environment for Network Innovations. Web site http://geni.net.

[2] Mark Handley Orion Hodson Eddie Kohler. "XORP: An Open Platform for Network Research," *ACM SIGCOMM Hot Topics in Networking, 2002.*

[3] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. "The Click modular router," *ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297.*

[4] J. Turner, P. Crowley, J. Dehart, A. Freestone, B. Heller, F. Kuhms, S. Kumar, J. Lockwood, J. Lu, M.Wilson, C. Wiseman, D. Zar. "Supercharging PlanetLab - High Performance, Multi-Application, Overlay Network Platform," *ACM SIGCOMM '07, August 2007, Kyoto, Japan.*

[5] NetFPGA: Programmable Networking Hardware. Web site http://netfpga.org.

[6] The OpenFlow Switch Specification. Available at http://OpenFlowSwitch.org.

[7] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, Scott Shenker. "Ethane: Taking Control of the Enterprise," *ACM SIGCOMM '07, August 2007, Kyoto, Japan.*

[8] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casadao, Nick McKeown, Scott Shenker, "NOX: Towards an Operating System for Networks," *In submission. Also: http://nicira.com/docs/nox-nodis.pdf.*

[1] http://www.OpenFlowSwitch.org

[2] Some platforms may limit the license terms of software running on them. For example, a reference implementation on Linux may be limited by the Linux GPL.


สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง