การศึกษาการสร้างโมเดลและการจำลองเครือข่าย ด้วย NETWORK SIMULATOR 2

NETWORK MODELING AND SIMULATION USING NETWORK SIMULATOR 2 (NS-2)

โดย

กวิน สื่อจูงใจรักษ์

นฤดล รุ่งวีรกุลอนันต์

สันต์สลิล สุนทรนันท์

อาจารย์ที่ปรึกษา

รศ.ดร. โชติพัชร์ ภรณวลัย

อาจารย์ที่ปรึกษาร่วม

อาจารย์ สุเมธ ประภาวัต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคเรียนที่ 2 ปีการศึกษา 2549

NETWORK MODELING AND SIMULATION
USING NETWORK SIMULATOR 2 (NS-2)

KWIN SURJUNGJAIRAX

NARUDOL RUNGVEERAKULANAN

SUNSALIN SOONTORANUNT

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF SCIENCE PROGRAM IN INFORMATION TECHNOLOGY

FACULTY OF INFORMATION TECNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2/ 2006

ใบรับรองปริญญานิพนธ์ ประจำปีการศึกษา 2549

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การศึกษาการสร้างโมเดลและการจำลองเครือข่ายด้วย Network Simulator 2

Network Modeling and Simulation using Network Simulator 2 (NS-2)

ผู้จัดทำ

1. นาย กวิน สื่อจูงใจรักษ์ รหัสประจำตัว 46060004
2. นาย นฤดล รุ่งวีรกุลอนันต์ รหัสประจำตัว 46060019
3. นางสาว สันต์สลิล สุนทรนันท์ รหัสประจำตัว 46060094

.....อาจารย์ที่ปรึกษา

(รศ.ดร. โชติพันธ์ ภรณวลัย)

.....อาจารย์ที่ปรึกษาร่วม

(อาจารย์ สุเมธ ประภาวัต)

หัวข้อ	การศึกษาการสร้างโมเดลและการจำลองเครือข่ายด้วย Network Simulator 2
นักศึกษา	นาย กวิน สื่อจูงใจรักษ์ 46060004
	นาย นฤดล รุ่งวีรกุลอนันต์ 46060019
	นางสาวสันต์สลิล สุนทรนันท์ 46060094
ปริญญา	วิทยาศาสตรบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
ปีการศึกษา	2549
อาจารย์ที่ปรึกษา	รศ.ดร. โชติพัชร์ ภรณวลัย
อาจารย์ที่ปรึกษาร่วม	อาจารย์ สุเมธ ประภาวัต

บทคัดย่อ

โครงงานฉบับนี้นำเสนอการศึกษาประสิทธิภาพของระบบเครือข่ายและการจำลองเครือข่ายด้วย Network Simulator 2 หรือ NS-2 ภายใต้สภาพแวดล้อมต่างๆและนำผลลัพธ์ที่ได้จากการจำลองมากระทำการประมวลผลและวิเคราะห์เพื่อหารูปแบบและทฤษฎีของการได้มาหรือสูญเสียประสิทธิภาพรวมถึงการวัดประสิทธิภาพของเครือข่ายในรูปแบบต่างๆ

Title	Network Modeling and Simulation using Network Simulator 2 (NS-2)
Student	Mr. Kwin Surjungjairax 46060004
	Mr. Narudol Rungveerakulanan 46060019
	Miss. Sunsalin Soontoranunt 46060094
Degree	Bachelor of Science
Programme	Information Technology
Academic Year	2006
Advisor	Assoc.Prof. Dr.Chotipat Pornavalai
Co-Advisor	Sumet Prabhavat

ABSTRACT

This project has represented the performance measurement of network and simulation with Network Simulator 2 (NS-2) within variety of environment. And get simulation result into process and analyze for find pattern and theory about obtaining and losing of network performance include network performance measurement.

กิตติกรรมประกาศ

ปริญญานิพนธ์เล่มนี้สำเร็จได้ด้วยความกรุณาจากอาจารย์ที่ปรึกษา รศ.ดร. โชติพัชร์ ภรณวลัย และอาจารย์ที่ปรึกษาร่วม อ.สุเมธ ประภาวัต ที่ให้ความช่วยเหลือ ให้คำชี้แนะ ช่วยแก้ปัญหาตลอดจนให้ความรู้และประสบการที่ดีแก่ข้าพเจ้า

ขอขอบพระคุณ ผศ. อัครินทร์ คุณกิตติ และอาจารย์ลภัส ประดิษฐ์ทัศนีย์ กรรมการสอบ ที่กรุณาให้คำแนะนำ ตลอดจนข้อชี้แนะ จนในที่สุดทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้

ขอขอบพระคุณพี่ นวลพรรณ ศรีฟ้า ที่ให้คำแนะนำในเรื่องของทฤษฎีการสูญเสียพลังงานในระบบเครือข่ายเฉพาะกิจ

กวิน สื่อจูงใจรักษ์
นฤดล รุ่งวีรกุลอนันต์
สันต์สลิล สุนทรนันท์

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญตาราง(ต่อ)

สารบัญตาราง(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันนั้นการติดต่อสื่อสารเป็นสิ่งที่จำเป็นขึ้นอย่างมาก และด้วยเทคโนโลยีที่มีการพัฒนาอย่างไม่หยุดยั้ง จึงทำให้เครือข่ายการสื่อสารเดิมเริ่มมีประสิทธิภาพลดลง การที่จะได้มาซึ่งระบบเครือข่ายที่มีประสิทธิภาพนั้นจะต้องมีการพัฒนาโปรโตคอลที่ใช้ในการสื่อสารเดิมให้สามารถรองรับต่อการใช้งานได้โดยต้องคำนึงถึงประสิทธิภาพเป็นหลักนั้น จำเป็นต้องมีการออกแบบระบบเครือข่าย และการเลือกใช้งานเทคโนโลยีที่ดีและเหมาะสมกับลักษณะงานที่ใช้ หากมีการพัฒนาและใช้งานอย่างไม่เหมาะสมแล้ว จะทำให้ไม่สามารถใช้งานระบบเครือข่ายได้เต็มประสิทธิภาพ โดยจะไม่มีความคุ้มค่าในการที่ลงทุนเพื่อพัฒนาเครือข่าย และที่สำคัญการออกแบบระบบที่ดีนั้นเป็นเรื่องที่ทำได้ยาก เพียงแค่ทฤษฎีไม่อาจทำให้เราสามารถทำการวิเคราะห์และออกแบบระบบเครือข่ายที่ดีเพียงพอได้ จึงมีการนำเครื่องมือที่ใช้ในการจำลองระบบเครือข่ายมาใช้ เพื่อจำลองการใช้งานของระบบที่ออกแบบมา ก่อนที่จะนำไปพัฒนาจริง เพื่อให้สามารถคาดเดาเหตุการณ์ที่จะเกิดขึ้นก่อนที่จะพัฒนาจริง โดยนำไปเปรียบเทียบประสิทธิภาพกับโปรโตคอลที่ใช้ในการสื่อสารเดิมเพื่อดูว่าดีกว่าหรือด้อยกว่าอย่างไร เพื่อทดสอบความสามารถของระบบที่ออกแบบมาภายใต้เงื่อนไขต่างๆ เพื่อให้ได้ระบบเครือข่ายและการใช้เทคโนโลยีอย่างมีประสิทธิภาพสูงสุดตามที่ต้องการ โดยเฉพาะอย่างยิ่งเทคโนโลยีที่ยังไม่มีการศึกษาและใช้อย่างแพร่หลายนัก โดยในโครงงานได้เลือก Multi Protocol Label Switching (MPLS) และ Mobile Ad-hoc Network (MANET)

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ทำการวิจัยและทดลองแบบจำลอง (Model) ทางด้านเน็ทเวิร์คชนิดต่างๆ โดยใช้เครื่องมือในการจำลองระบบเครือข่ายที่ชื่อ Network Simulator 2 (NS-2) เพื่อนำผลลัพธ์ที่ได้จากการจำลองมาทำการวิเคราะห์และสรุปผล เพื่อพิสูจน์และศึกษาหาทฤษฎีหรือรูปแบบของการนำมาซึ่งประสิทธิภาพของระบบในรูปแบบต่างๆ โดยเป็นการศึกษาถึงคำสั่งที่จำเป็น ภาษาที่ใช้ รวมถึงโครงสร้างที่ควรรู้เกี่ยวกับ NS-2 เพื่อให้สามารถจำลองระบบเครือข่ายได้ตามที่ต้องการ และจะกล่าวถึงการจำลองแบบจำลองที่เลือกมาเพื่อหาประสิทธิภาพและข้อเท็จจริงบางประการที่เกิดขึ้นกับระบบ ภายใต้สภาพแวดล้อมและเงื่อนไขต่างๆ ที่เปลี่ยนไป เพื่อหาข้อสรุปเกี่ยวกับรูปแบบของผลลัพธ์ที่เกิดขึ้น ซึ่งใน MANET จะเป็นการวัดประสิทธิภาพของระบบที่แตกต่างจากระบบเครือข่ายอื่นๆ ซึ่งจะเป็นประสิทธิภาพในด้านการใช้พลังงานของอุปกรณ์ในระบบ โดยจะศึกษาถึงรูปแบบรวมถึงพฤติกรรม

ของการใช้พลังงานรวมถึงลักษณะการกระจายของความยุติธรรมในการใช้พลังงานของแต่ละอุปกรร์ในเครือข่าย

1.3 ทฤษฎีหรือแนวคิดที่ใช้ในการวิจัย

Network Simulator 2 หรือ NS-2 จะมีแบบจำลองรวมอยู่ในตัวโปรแกรมด้วยจำนวนหนึ่ง ซึ่งแบบจำลองทางระบบเครือข่ายที่มีอยู่แล้วใน NS-2 มีอยู่หลากหลาย โดยจะอิงตามทฤษฎีที่มีอยู่จริง เพื่อให้ได้แบบจำลองที่ใช้แทนการติดตั้งจริงได้อย่างมีประสิทธิภาพ ซึ่งแบบจำลองแต่ละชนิดจำเป็นต้องมีความรู้ความเข้าใจเกี่ยวกับทฤษฎีของที่เกี่ยวข้องกับแบบจำลอง และเข้าใจในโครงสร้างของแบบจำลองทำงานอยู่บนเครื่องมือจำลองนี้ เพื่อที่จะสามารถทำการจำลองได้ตามที่ต้องการและมีประสิทธิภาพ โดยในการจำลองจะกระทำผ่าน Tcl สคริปท์ เพื่อกำหนดคุณลักษณะ (Parameter) ในการจำลอง ผลลัพธ์ที่ได้จากการจำลองจะอยู่ในรูปแบบของเทรซไฟล์ ซึ่งเปรียบเสมือนไฟล์ที่บันทึกเหตุการณ์ต่างๆที่เกิดขึ้นในการจำลอง ซึ่งเป็นรูปแบบของข้อมูลดิบ จึงจำเป็นต้องใช้เครื่องมือในการกรองข้อมูลเพื่อให้ได้เฉพาะข้อมูลที่ต้องการและนำมาทำการวิเคราะห์ด้วย PERL และทำการแสดงผลให้ออกมาในรูปแบบที่มนุษย์สามารถเข้าใจได้ง่ายด้วยแผนภูมิลักษณะต่างๆ (Graph) โดยรายงานฉบับนี้จะทำการนำเสนอเนื้อหาและทฤษฎีที่จำเป็นในเรียนรู้เพื่อการจำลอง และนำเสนอถึงอัลกอริธึมและคำสั่งที่ใช้ในการวิเคราะห์หาประสิทธิภาพรวมถึงทำการสรุปผลที่ได้จากการทดลอง

1.4 ขอบเขตการวิจัย

เป็นการศึกษาทฤษฎีที่เกี่ยวข้องเกี่ยวกับแบบจำลองที่เลือกมา รวมถึงทำความเข้าใจในทฤษฎีและโครงสร้างของเครื่องมือในการจำลอง NS-2 ที่จำเป็นต้องศึกษาในการจำลองแบบจำลองที่เลือกมา และทำการจำลองภายใต้เงื่อนไขต่างๆ เพื่อให้ครอบคลุมทุกความเป็นไปได้ให้มากที่สุด โดยใช้ Tcl สคริปท์ในการจำลอง และนำผลลัพธ์มาวิเคราะห์เพื่อหาข้อเท็จจริงที่ซ่อนอยู่ หรือในอีกแง่มุมหนึ่งด้วย PERL สคริปท์และแสดงผลในรูปแบบของแผนภูมิ เพื่อศึกษา วิเคราะห์หาถึงรูปแบบของประสิทธิภาพต่างๆที่เกิดขึ้น ภายใต้เงื่อนไขที่เปลี่ยนแปลงไป และทำการสรุปผลการทดลอง

1.5 ขั้นตอนของการศึกษา

- ทำการศึกษาการใช้งานของ NS-2 และการเขียนโปรแกรมภาษา TCL และ PERL เพื่อใช้ในการทดลอง
- ทดลองเขียนสคริปเพื่อทดสอบประสิทธิภาพของโมเดลทางระบบเครือข่ายในรูปแบบต่างๆ เช่น ทรูพุต (Throughput) , ความหน่วง (delay) ,จิตเตอร์ (jitter) เป็นต้น

- ศึกษาโมเดลทางระบบเครือข่ายที่นอกเหนือจากโมเดลพื้นฐาน ซึ่งในที่นี้คือ MPLS
- เขียนสคริปเพื่อทดสอบประสิทธิภาพของ MPLS
- ทำการศึกษาทฤษฎีที่จำเป็นและเกี่ยวข้องกับ MANET
- ศึกษาทฤษฎีเกี่ยวกับประสิทธิภาพในด้านลักษณะและพฤติกรรมการใช้พลังงานของอุปกรณ์ใน MANET
- ทำการศึกษาโครงสร้างและการทำงานของ NS-2 เกี่ยวกับการจำลองของ MANET
- ทำการศึกษาวิธีการและโครงสร้างของ PERL ซึ่งเป็นเครื่องมือใช้ในการวิเคราะห์ผล รวมถึงออกแบบอัลกอริธึมในการวิเคราะห์
- ทำการจำลองแบบจำลองภายใต้สภาพแวดล้อมต่างๆ
- นำผลการทดลองทำการวิเคราะห์เพื่อหาข้อเท็จจริง และข้อมูลต่างๆที่ต้องการด้วย Perl
- ทำการสรุปผลการทดลอง

บทที่ 6
การทดลองเพื่อหารูปแบบการใช้พลังงาน ในระบบเครือข่ายเฉพาะกิจ

6.1 บทนำ

ในบทที่ 5 จะนำเสนอถึงการทดลองรวมถึงอธิบายสภาพแวดล้อม, ปัจจัย และเงื่อนไขต่างๆในการจำลอง โดยวัตถุประสงค์ของการทดลองคือ การทดลองเพื่อหารูปแบบของการสูญเสียพลังงานของอุปกรณ์เครือข่ายไร้สายเฉพาะกิจ โดยจะอธิบายถึงลักษณะการสูญเสียพลังงานในแต่ละกิจกรรม รวมถึงจำแนกจุดประสงค์ของการสูญเสียพลังงานนั้น เพื่อให้ได้มาซึ่งรูปแบบของความคุ้มค่าและความยุติธรรมของการสูญเสียพลังงานที่เกิดขึ้นในแต่ละโหนด

โดยการทดลองจะใช้เครื่องมือในการจำลองแทนการจำลองจริง โดยการจำลองนั้นจะกำหนดตัวแปรที่จะใช้เป็นเงื่อนไขในการทดลอง ซึ่งจะทำการทดลองภายใต้ขนาดพื้นที่ที่เปลี่ยนแปลงไป โดยจะตั้งสมติฐานเกี่ยวกับการที่พื้นที่ที่มีผลต่อประสิทธิภาพของเครือข่ายและอุปกรณ์ที่เปลี่ยนแปลงไป และในการจำลองของแต่ละขนาดพื้นที่ จะทำการจำลองหลายๆครั้งเพื่อที่จะครอบคลุมความเป็นไปได้ให้มากที่สุด และนำผลลัพธ์ทั้งหมดมาหาค่าเฉลี่ยเพื่อให้ได้ผลลัพธ์ที่ใช้แทนผลลัพธ์ของการทดลองของขนาดพื้นที่นั้นๆ โดยเมื่อได้ผลการทดลองของทุกพื้นที่จะนำผลลัพธ์มาทำการวิเคราะห์เพื่อหาข้อสรุปต่อไป

6.2 สภาพแวดล้อมของการทดลอง

6.2.1 เครื่องมือในการจำลอง

ในการทดลองนั้น จะไม่สามารถทดลองกับอุปกรณ์และระบบจริงได้ เนื่องจากเป็นการสิ้นเปลืองค่าใช้จ่ายและระยะเวลา ดังนั้นจึงใช้เครื่องมือในการจำลองที่ชื่อว่า NS-2 (Network Simulator 2) ซึ่งเป็นเครื่องมือที่ใช้ในการจำลองระบบเครือข่ายเพื่อวัดประสิทธิภาพของระบบที่สามารถใช้ได้ฟรีและสามารถจำลองได้อย่างมีประสิทธิภาพ และยังได้รับการยอมรับในเรื่องของความน่าเชื่อถือ โดยการจำลองนั้นจะกระทำการจำลองโดยใช้ Tcl สคริปท์ ในการกำหนดลักษณะการจำลอง, รูปแบบ และเงื่อนไขต่างๆ รวมถึงสภาพแวดล้อมของการจำลอง เมื่อทำการจำลองจะได้ผลลัพธ์จาการจำลองออกมาในรูปของเทรซไฟล์ ซึ่งเป็นไฟล์ที่เก็บบันทึกเหตุการณ์ต่างๆที่เกิดขึ้นในการจำลอง รวมถึงรานละเอียดเกี่ยวกับเหตุการณ์นั้นๆ

6.2.2 ขอบเขตพื้นที่ในการเคลื่อนที่

ในการจำลองเราจะต้องระบุขนาดพื้นที่ของการจำลอง โดยสมมติให้เป็นพื้นที่ที่แต่ละโหนดในเครือข่ายเฉพาะกิจสามารถเคลื่อนที่ไปได้ โดยจะกำหนดความกว้างและความยาวในหน่วยเมตร โดยในการทดลองจะกำหนดให้ขนาดพื้นที่เป็นตัวแปรต้น ในการนำมาเป็นปัจจัยในการจำลอง ซึ่งจะกำหนดขนาดความกว้างคูณยาวให้เป็น 500*500, 750*750, 1000*1000, 1250*1250 และ 1500*1500 โดยมีหน่วยเป็นเมตร ซึ่งเมื่อพื้นที่มีขนาดมากขึ้นก็จะมีความจำเป็นที่ต้องมีการส่งผ่านอุปกรณ์ตัวกลางมากขึ้น ในการส่งจากต้นทางไปยังปลายทาง ขณะที่พื้นที่ยิ่งเล็กแต่ละอุปกรณ์ก็จะมีโอกาสเสียพลังงานในการรับฟังมากยิ่งขึ้น เนื่องจากในเครือข่ายไร้สายจะส่งข้อมูลเป็นคลื่นวิทยุทำให้ทุกอุปกรณ์มีโอกาสในการรับข้อมูล

6.2.3 โหนด

โหนดในการทดลองคืออุปกรณ์ไร้สาย จึงมีความสามารถในการเคลื่อนที่ได้ เนื่องจากไม่มีข้อจำกัดเรื่องสาย โดยแต่ละโหนดจะมีตำแหน่งเริ่มต้นของแต่ละโหนด และมีรูปแบบและทิศทางการเคลื่อนที่ที่แตกต่างกัน ซึ่งจะกล่าวถึงโดยละเอียดในหัวข้อ "รูปแบบการเคลื่อนที่" ซึ่งในการทดลองได้กำหนดจำนวนโหนดให้คงที่คือ 10 โหนด ซึ่งแต่ละโหนดจะมีความสามารถในการรับส่งข้อมูล รวมถึงพลังงานและคุณสมบัติเหมือนกัน ซึ่งจะกำหนดให้แต่ละโหนดมีพลังงานเริ่มต้นเท่ากันคือ 100 จูล

6.2.4 รูปแบบการเคลื่อนที่

ในการทดลองจะมีการกำหนดรูปแบบการเคลื่อนที่ ซึ่งจะเป็นการกำหนดตำแหน่งเริ่มต้นของแต่ละโหนดในลักษณะเป็นพิกัด และกำหนดทิศทางการเคลื่อนที่ โดยจะระบุพิกัดปลายทางและช่วงเวลา ซึ่งจะเป็นผลต่อความเร็วในการเคลื่อนที่ ซึ่งการกำหนดรูปแบบการเคลื่อนที่มีหลายวิธีการ เช่น Random Waypoint, Reference Point Group Mobility Model หรือ Manhattan Grid

โดยในการทดลองจะเลือกใช้ Random Waypoint ซึ่งจะมีพารามิเตอร์ที่สำคัญอยู่ 3 ตัว คือ เวลาหยุด, ความเร็วสูงสุด และความเร็วต่ำสุด โดยเมื่อเริ่มจะมีการกำหนดตำแหน่งเริ่มต้นของแต่ละโหนดโดยการสุ่ม จากนั้นแต่ละโหนดจะอยู่ในตำแหน่งนั้นตามเวลาหยุดที่กำหนดไว้ เมื่อหมดเวลาก็จะทำการเปลี่ยนตำแหน่ง โดยจะไปยังตำแหน่งที่ได้จากการสุ่มอีกเช่นกัน โดยจะเคลื่อนที่ย้ายตำแหน่งด้วยความเร็ว ที่เกิดจากการสุ่มค่าระหว่างความเร็วสูงสุดและความเร็วต่ำสุด และก็จะหยุดอยู่ในตำแหน่งใหม่ตามเวลาหยุดที่กำหนดและเมื่อหมดก็จะเริ่มย้ายตำแหน่งอีกครั้ง และเป็นในลักษณะนี้ไปเรื่อยๆ

โดยในการทดลองนั้น ในการจำลองที่ขนาดพื้นที่เดียวกันในแต่ละครั้งนั้น จะมีรูปแบบการเคลื่อนที่เหมือนกัน กล่าวคือ หากในขาดพื้นที่เท่ากันแล้วทุกโหนดจะมีรูปแบบการเคลื่อนที่เหมือนเดิมทุกครั้ง

6.2.5 รูปแบบการเชื่อมต่อ

ในการทดลองจะกำหนดรูปแบบการเชื่อมต่อโดยจะเป็นการกำหนดปริมาณข้อมูล ต้นทาง และปลายทางของข้อมูลที่จะมีการส่ง รวมถึงกำหนดเวลาที่เริ่มการส่งนั้นด้วย โดยจะมีพารามิเตอร์ที่กำหนดให้คงที่อยู่ 4 ตัว คือ ประเภทของทราฟฟิกเป็น CBR (Constatnt Bit Rate), ขนาดของแพ็คเกตเป็น 512 ไบท์, จำนวนแพ็คเกตเป็น 10,000 แพ็คเกต และระยะเวลาระหว่างแพ็คเกตเป็น 0.25 วินาที ซึ่งค่า 4 ค่านี้จะเหมือนกันในทุกๆการเชื่อมต่อ ซึ่งในแต่ละการจำลองจะกำหนดให้มีการเชื่อมต่อจำนวน 5 ครั้ง โดยจะมีต้นทาง, ปลายทาง และเวลาที่เริ่มส่งแตกต่างกัน โดยเกิดจากค่าสุ่ม โดยใน 1 การทดลองจะกำหนดให้มีการเชื่อมต่อจำนวน 5 การเชื่อมต่อ ซึ่งในการทดลองนั้น ในแต่ละขนาดพื้นที่จะมีการจำลองทั้งหมด 10 ครั้ง โดยแต่ละครั้งจะมีรูปแบบการเชื่อมต่อแตกต่างกัน โดยเมื่อต่างขนาดพื้นที่ และครั้งที่ต่างกันในการทดลองจะมีรูปแบบการเชื่อมต่อที่แตกต่างกัน ซึ่งได้จากการสุ่ม โดยกำหนดให้มีทั้งหมด 5 การเชื่อมต่อทั้งการจำลอง

6.2.6 เราติ้งโปรโตคอล

ในการทดลองนั้นได้เลือกโปรโตคอล Dynamic Source Routing (DSR) เป็นเราติ้งโปรโตคอลในเครือข่ายเฉพาะกิจนี้ ซึ่งเป็นเราติ้งโปรโตคอลประเภท Reactive หรือ on-demand ที่จะมีการค้นหาและสร้างเส้นทางก็ต่อเมื่อต้นทางมีความต้องการที่จะส่งข้อมูล หรือมีการร้องขอเท่านั้น และมีกลไกการทำงานที่ไม่ซับซ้อนนัก

6.2.7 รูปแบบการใช้พลังงาน

ในการจำลองได้ใช้ Network Simulator 2 (NS-2) เป็นเครื่องมือในการจำลอง ซึ่งใน NS-2 กำหนดรูปแบบการใช้พลังงานของแต่ละอุปกรณ์ไว้ โดยได้กำหนดให้เกิดการสูญเสียพลังงานเมื่อมีการทำงานในระดับ MAC ตามมาตรฐาน 802.11 เช่น การรับหรือส่งข้อมูล รวมถึงการรับฟังหรือดักจับข้อมูลที่ไม่ใช่ของตนเองด้วย โดยจะมีรูปแบบการสูญเสียพลังงานดังนี้

Energy =m * length + b

โดย Energy คือพลังงานที่สูญเสียสำหรับกิจกรรมนั้นๆ

m คืออัตราการสูญเสียที่เพิ่มตามขนาดของข้อมูล โดยจะแตกต่างกันตามกิจกรรม

length คือปริมาณข้อมูลที่รับหรือส่ง

b คือค่าคงที่ที่จะสูญเสียทุกครั้งเมื่อมีการทำกิจกรรม โดยแตกต่างกันตามกิจกรรม ซึ่งค่าต่างๆจะเป็นไปดังตารางต่อไปนี้

ตารางที่ 6.1 แสดงตัวแปรในการคำนวณพลังงาน

สัญลักษณ์	ค่า	หน่วย
mในการส่ง	1.89	ไมโครวัตต์ต่อวินาที/ไบท์
b ในการส่ง	246	ไมโครวัตต์ต่อวินาที
m ในการรับ	0.49	ไมโครวัตต์ต่อวินาที/ไบท์
b ในการรับ	56.1	ไมโครวัตต์ต่อวินาที
b ในการส่งข้อความควบคุม	120	ไมโครวัตต์ต่อวินาที
b ในการรับข้อความควบคุม	29	ไมโครวัตต์ต่อวินาที

ยกตัวอย่างเช่นเมื่อมีการส่งข้อมูล RTS ซึ่งเป็นข้อความควบคุมใน 802.11 จะนำเอา เวลาที่ใช้ในการส่งข้อมูลออกจากโหนดโดยจะแปรตามขนาดข้อมูล คูณด้วย m ในการส่ง และบวกด้วย b ในการส่งข้อความควบคุม เป็นต้น

โดยในการทดลองจะกำหนดค่าพลังงานเริ่มต้นของแต่ละโหนดคือ 100 จูล ซึ่งก็จะลดลงไปเรื่อยๆ ตามแต่กิจกรรมและขนาดของข้อมูล ซึ่งคำนวณได้จากค่าในตาราง

โดยในการจำลองนั้นจะกำหนดระยะ Receive Threshold และ Carrier Sense Threshold ให้เป็น 250 เมตร และ 550 เมตร ตามลำดับ วัดจากจุดศูนย์กลาง ซึ่งใน NS-2 จะมีการแปลงกลับไปอยู่ในรูปของพลังงานแล้ว คือเมื่อระยะทางจากต้นทางยิ่งไกล กำลังจะยิ่งอ่อนลง นั่นหมายถึงกำลังสัญญาณที่ระดับ Receive Threshold ต้องมากกว่า Carrier Sense Threshold แน่นอน

6.3 การทดลองวิเคราะห์หาความเหลื่อมล้ำและความยุติธรรมในการสูญเสียพลังงาน

การทดลองนี้จะทำการวิเคราะห์หาเพื่อดูระดับความแตกต่างระหว่างพลังงานที่ใช้ในกิจกรรมเพื่อกิจกรรมของตนเอง (Itself), พลังงานที่ใช้ในกิจกรรมเพื่อผู้อื่น (Give) และพลังงานรวมที่ผู้อื่นใช้ในการทำกิจกรรมให้กับตนเอง (Take) ซึ่งทั้ง 3 ส่วนนี้ สามารถเอามาเป็นปัจจัยในการพิจารณาถึงความยุติธรรมในการสูญเสียพลังงานในเครือข่ายเฉพาะกิจได้

โดยพลังงานที่ใช้เพื่อกิจกรรมของตนเอง (Itself) นั้น จะสามารถดูได้จากการรับและส่งข้อมูล ที่มีต้นทางเป็นของตน หรือปลายทางเป็นของตนตามลำดับ ซึ่งในกรณีที่ส่งข้อมูลที่ต้นทางคือตนเองนั่นหมายถึงว่าเป็นการส่งข้อมูลที่ตนเองมีความต้องการที่จะส่ง ขณะที่การรับข้อมูลที่ปลายทางเป็นของตนนั่นหมายความว่าเป็นการรับข้อมูลของตนเองที่ตนเองต้องการ ดังนั้นพลังงานที่ใช้เพื่อกิจกรรมของตนเองนั้นจะต้องเพียงพอกับความต้องการใช้งาน

พลังงานที่ใช้ในกิจกรรมเพื่อผู้อื่น (Give) จะดูจากการที่มีการรับและส่งข้อมูลที่ตนเองไม่ใช่ต้นทางหรือปลายทางนั้นจริงๆ นั่นหมายความว่า อุปกรณ์ที่ทำกิจกรรมในลักษณะนี้ จะทำหน้าที่เป็นตัวกลางในการส่งผ่านข้อมูลของอุปกรณ์อื่น ซึ่งก็แปลว่าเป็นการทำกิจกรรมเพื่อผู้อื่นนั่นเอง โดยการใช้พลังงานเพื่อผู้อื่นนั้น จะเป็นการใช้พลังงานเพื่ออุปกรณ์ที่เป็นต้นทางและปลายทางของข้อมูลสำหรับกิจกรรมนั้นๆ เช่น อุปกรณ์ ค. ทำการส่งข้อมูลโดยต้นทางและปลายทางจริงๆของข้อมูลคืออุปกรณ์ ก. และ ข. ตามลำดับ นั่นหมายถึง อุปกรณ์ ค. ได้มีการสูญเสียพลังงานเพื่ออุปกรณ์ ก. และ ข. นั่นเอง เนื่องจากทำหน้าที่เป็นตัวกลางในการส่งข้อมูลระหว่างอุปกรณ์ทั้งสอง โดยแต่ละอุปกรณ์ไม่ควรเสียพลังงานในส่วนนี้เกินกว่าการสูญเสียพลังงานเพื่อกิจกรรมของตนเอง ซึ่งไม่ควรจะสูญเสียพลังงานเพื่อผู้อื่นจนไม่สามารถใช้พลังงานในการทำกิจกรรมเพื่อตนเองได้

พลังงานรวมที่ผู้อื่นใช้ในการทำกิจกรรมให้กับตนเอง (Take) นั้น จะมองได้จากมีอุปกรณ์อื่นสูญเสียพลังงานเพื่อกิจกรรมของตนเองเท่าไหร่ โดยเมื่อมีอุปกรณ์ใดๆก็ตามสูญเสียพลังงานเพื่ออุปกรณ์อื่นแล้ว จะต้องมีอุปกรณ์ที่ได้รับผลประโยชน์จากการสูญเสียพลังงานในครั้งนั้น ซึ่งก็คืออุปกรณ์ต้นทางและปลายทางของกิจกรรมนั้นนั่นเอง จากกิจกรรมเมื่อตัวอย่างที่แล้ว อุปกรณ์ ก. และ ข. จะได้รับผลประโยชน์จากการสูญเสียพลังงานจากอุปกรณ์ ค. ทั้งคู่ โดยสามารถสรุปได้ว่าเมื่อมีอุปกรณ์ใดอุปกรณ์หนึ่งสูญเสียพลังงานเพื่อผู้อื่นนั้น จะมีอุปกรณ์ที่ได้รับผลประโยชน์จากการสูญเสียครั้งนั้นอย่างน้อย 2 อุปกรณ์เสมอ

โดยในการทดลองจะทำการเปลี่ยนแปลงตัวแปรคือขนาดขอบเขตพื้นที่ของเครือข่ายเฉพาะกิจ ซึ่งจะกำหนดให้เป็น 500*500, 750*750, 1000*1000, 1250*1250 และ 1500*1500 โดยมีหน่วยเป็นเมตร ซึ่งในแต่ละขนาดจะทำการทดลองทั้งหมด 10 ครั้งโดยในแต่ละครั้งจะมีการสุ่มเพื่อสร้างรูปแบบของการเชื่อมต่อขึ้นมา 5 เชื่อมต่อ ซึ่งทดลอง 10 ครั้งก็จะได้ 10 รูปแบบการเชื่อมต่อ และ 50 การเชื่อมต่อ ซึ่งแต่ละการเชื่อมต่อจะแตกต่างกันด้วยต้นทาง, ปลายทาง และเวลาในการส่งข้อมูล

ซึ่งได้จากค่าสุ่ม นอกจากที่ได้กล่าวนี้ สภาพแวดล้อมการทดลองจะเป็นไปดังที่ได้กล่าวไว้ในข้างต้นทั้งสิ้น

และในการทดลองในแต่ละขนาดพื้นที่จะนำผลการทดลองทั้ง 10 ครั้งมาพิจารณาเพื่อหาค่าเฉลี่ย ซึ่งจะถือว่าเป็นตัวแทนผลลัพธ์จากการทดลองของขนาดพื้นที่นั้นๆ

ด้วยสภาพแวดล้อมและเงื่อนไขต่างๆดังที่ได้กล่าวไป จะนำมาทำการสร้าง TCL สคริปท์ ซึ่งนำไปใช้ในการจำลองด้วย NS-2 ซึ่งจะจำลองเป็นเวลา 1,000 วินาที โดยเป็นเวลาที่มากพอที่จะให้เห็นความเปลี่ยนแปลงของปัจจัยบางอย่าง และไม่ผู้จัดทำไม่อาจทำการทดลองให้เวลามากกว่านี้ได้ เนื่องจากด้วยประสิทธิภาพของอุปกรณ์และความจุของหน่วยความจำ รวมถึงระยะเวลาที่ใช้ในการประมวลผลค่อนข้างเป็นปัญหามาก โดยเมื่อทำการจำลองจะได้ผลลัพธ์ออกมาอยู่ในรูปของเทรซไฟล์ ซึ่งจะนำมาวิเคราะห์และประมวลผลด้วย PERL ต่อไป

โดยเราทำการทดลองที่ขนาดพื้นที่ 5 ขนาดที่แตกต่างกัน และทดลองขนาดละ 10 ครั้ง ซึ่งจะได้ผลลัพธ์ที่เป็นเทรซไฟล์ออกมาทั้งหมดจำนวน 50 ไฟล์ จากนั้นจึงจะนำเทรซไฟล์เหล่านี้ไปประมวลผลด้วย PERL เพื่อกรองและดึงข้อมูลที่สำคัญออกมาและทำการประมวลผลเพื่อให้ได้ข้อมลที่ต้องการ ดังที่กล่าวไปแล้วในบทที่ 5

6.4 ผลการทดลอง

โดยการทดลองจะนำเสนอมุมมองในการพิจารณาการใช้พลังงานในส่วนต่างๆ โดยจะแยกเป็น

- จำนวนแพ็คเกตและพลังงานที่สูญเสียในการทำกิจกรรมเพื่อตนเอง
- จำนวนแพ็คเกตและพลังงานที่สูญเสียในการทำกิจกรรมเพื่อผู้อื่น
- จำนวนแพ็คเกตและพลังงานที่เป็นผลประโยชน์ที่ได้รับจากการสูญเสียของผู้อื่น
- จำนวนแพ็คเกตและพลังงานที่สูญเสียในการทำกิจกรรมเกี่ยวกับโปรโตคอล ARP
- จำนวนแพ็คเกตและพลังงานที่สูญเสียในการทำกิจกรรมเกี่ยวกับโปรโตคอล DSR
- จำนวนแพ็คเกตและพลังงานที่สูญเสียในการทำกิจกรรมเกี่ยวกับข้อความควบคุม
- จำนวนแพ็คเกตและพลังงานที่สูญเสียในการทำกิจกรรมเกี่ยวกับข้อมูลที่ไม่ทราบชนิด
- พลังงานคงเหลือของแต่ละอุปกรณ์

ซึ่ง 3 ค่าแรกคือ Give, Take และ Itself นั้นจะเกิดขึ้นได้ก็ต่อเมื่อเป็นการทำกิจกรรมเกี่ยวกับกับการรับส่งข้อมูลที่เป็น CBR เท่านั้น เนื่องจากจะสามารถทราบต้นทางและปลายของข้อมูลจริงๆได้จากหมายเลขประจำการไหลของข้อมูลจากเทรซไฟล์ ซึ่งจะมีเฉพาะที่กิจกรรมเกี่ยวกับ CBR เท่านั้น ขณะที่พลังงานที่สูญเสียในการทำกิจกรรมรับฟัง (L) โดยไม่มีการรับหรือลบทิ้งต่อนั้น จะไม่สามารถทราบชนิดของข้อมูลได้ ขณะที่การส่ง, รับ และลบทิ้งนั้นสามารถทราบชนิดของแพ็คเกตในการทำกิจกรรมนั้นๆได้ทั้งสิ้น

โดยจะกำหนดสัญลักษณ์และความหมายดังนี้

Take	พลังงานที่ผู้อื่นใช้ในการทำกิจกรรมให้ตนเอง
Remain	พลังงานคงเหลือ
Unknown	คือพลังงานที่ไม่สามารถจำแนกได้ว่าเป็นกิจกรรมของข้อมูลประเภทใด
Ctrl	พลังงานที่ใช้ในกิจกรรมของข้อความควบคุม
DSR	พลังงานที่ใช้ในกิจกรรมของโปรโตคอล DSR
ARP	พลังงานที่ใช้ในกิจกรรมของโปรโตคอล ARP
Itself	พลังงานที่ใช้เพื่อกิจกรรมของตนเอง
Give	พลังงานที่ใช้เพื่อกิจกรรมของผู้อื่น

โดยผลการทดลองทั้งหมดจะอยู่ในรูปเทรซไฟล์ และเมื่อทำการประมวลผลด้วย PERL เพื่อแยกจำนวนแพ็คเกตและปริมาณพลังงานที่ใช้ในแต่ละกิจกรรมแยกตามจุดประสงค์

โดยจากการผลจากการประมวลผลสามารถนำมานำเสนอในรูปของตารางซึ่งแสดงรายละเอียดการสูญเสียพลังงานของแต่ละโหนดในแต่ละการทดลองในแต่ละขนาดพื้นที่ โดยจะแสดงปริมาณพลังงานที่ใช้ในแต่ละกิจกรรมแยกตามจุดประสงค์

ซึ่งสามารถนำเสนอผลสรุปโดยแผนภูมิแท่งแสดงค่าเฉลี่ยของการสูญเสียพลังงานของแต่ละโหนดในแต่ละขนาดพื้นที่ ซึ่งคำนวณได้จากสมการดังนี้

พลังงานที่ใช้ในกิจกรรม x ของโหนด y = ผลรวมพลังงานพลังงานที่ใช้ในกิจกรรม x ของโหนด y ของ t หารทดลอง/t

และนำเสนอผลสรุปโดยแสดงแผนภูมิวงกลมที่แสดงพลังงานที่เสียไปในแต่ละกิจกรรมของแต่ละขนาดพื้นที่ โดยเป็นค่าเฉลี่ยของทุกอุปกรณ์ในการทดลองของพื้นที่นั้นๆ ตามสมการดังนี้

พลังงานเฉลี่ยที่ใช้ในกิจกรรม x = (ผลรวมพลังงงานที่ใช้ในกิจกรรม x จำนวน n โหนด ของ t การทดลอง)/(t*n)

โดยกำหนดให้ x คือชนิดของกิจกรรม, y คือลำดับของโหนด, n คือจำนวนโหนดทั้งหมดในการทดลอง ซึ่งคือ 10 โหนดและ t คือจำนวนครั้งในการทดลอง ซึ่งในการทดลองคือ 10 ครั้ง โดยพลังงานที่แสดงในทุกตารางและทุกแผนภูมิจะมีหน่วยเป็นจูล

พื้นที่ 500x500 ตร.ม.

ตารางที่ 6.2 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t0 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	8.844578	3.450853	8.788542	0.006039	0.107254	4.078791	39.74315	43.76934
1	13.1363	8.004071	17.59899	0.004791	0.066567	6.653887	30.51773	41.61665
2	4.504003	2.47447	8.810448	0.00508	0.037392	3.047037	40.98064	48.95138
3	19.04975	3.127296	10.53387	0.00835	0.115821	7.097559	32.89024	37.71098
4	4.939062	1.438396	6.441563	0.006426	0.045614	2.335779	41.73639	49.49833
5	5.32284	3.990996	4.09231	0.006111	0.09222	3.257861	39.00217	48.3278
6	0	5.907758	0	0.008116	0.092603	2.135569	43.74169	48.11426
7	9.554776	2.649646	10.20705	0.008539	0.06685	4.005614	40.0739	43.64068
8	5.341441	1.906205	10.20705	0.006897	0.044201	3.407685	39.78334	49.51023
9	0	5.390227	0	0.005756	0.055924	1.595967	44.4817	48.47043

ตารางที่ 6.3 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t1 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	18.03658	3.319519	18.95697	0.006913	0.148013	6.338994	36.91848	35.2315
1	4.736865	4.600043	8.261643	0.006116	0.063278	3.410652	42.2349	44.94815
2	21.75279	2.055044	32.90369	0.008696	0.11618	8.5369	32.21907	35.31132
3	14.37218	6.936577	17.97423	0.008627	0.110964	6.986723	33.65725	37.92768
4	10.19256	4.53166	16.8027	0.010268	0.089321	5.126401	35.74639	44.30341
5	0	6.481981	0	0.006236	0.103846	2.341147	46.38294	44.68385
6	0	7.244546	0	0.008819	0.105499	2.625766	45.68495	44.33042
7	0	2.048806	0	0.007104	0.068435	0.746499	50.43063	46.69852
8	0	2.792204	0	0.00933	0.073539	1.042882	49.06177	47.02027
9	0	7.439235	0	0.007488	0.090122	2.219197	45.84373	44.40023

ตารางที่ 6.4 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t2 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	5.44252	0	0.006319	0.049747	1.717144	42.86127	49.92301
1	19.4091	1.240991	18.11571	0.006902	0.103822	5.464869	33.51417	40.26014
2	5.116441	2.327415	9.372502	0.006043	0.041051	2.836255	39.11249	50.5603
3	14.27198	3.501609	15.11916	0.00576	0.057969	6.145176	31.42852	44.58899
4	13.72597	1.860945	12.8886	0.004892	0.062124	4.674384	34.02767	45.64401
5	14.4314	5.607835	11.1724	0.008264	0.072052	6.031384	30.4179	43.43116
6	5.098787	4.330158	4.659756	0.006518	0.068019	3.514916	37.30897	49.67263
7	0	1.386911	0	0.005759	0.055208	0.474702	46.39013	51.68729
8	0	4.296248	0	0.0051	0.043314	2.011189	43.17148	50.47267
9	0	5.669431	0	0.005183	0.054431	1.63745	42.76391	49.86959

ตารางที่ 6.5 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t3 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	19.00463	4.664608	19.76354	0.00576	0.157188	7.407369	35.72052	33.03992
1	4.88297	3.619473	8.521209	0.007095	0.063546	3.309341	43.6581	44.45948
2	22.47503	1.716782	34.50198	0.008328	0.111786	8.375503	32.86882	34.44376
3	12.96048	7.161945	18.46636	0.007186	0.110456	6.632644	35.31755	37.80974
4	9.44478	4.097761	17.75493	0.007676	0.093358	4.872238	37.80256	43.68163
5	0	6.504652	0	0.007487	0.10308	2.362366	47.22967	43.79275
6	0	6.692434	0	0.00835	0.108895	2.482158	47.01909	43.68908
7	0	5.083305	0	0.007871	0.074293	1.641455	48.68254	44.51054
8	0	2.840958	0	0.007197	0.079816	1.119314	49.84195	46.11076
9	0	7.122094	0	0.007872	0.076308	2.027099	47.0683	43.69833

ตารางที่ 6.6 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t4 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	5.136703	0	0.008712	0.040828	1.378895	40.5976	52.83726
1	0	4.624033	0	0.009682	0.032188	1.97447	40.61668	52.74295
2	0	2.175189	0	0.010139	0.033302	0.897863	42.74319	54.14032
3	8.332919	3.765228	6.284544	0.008438	0.06076	3.405867	35.90815	48.51864
4	14.44218	3.407609	13.2882	0.008336	0.058608	5.377988	29.76467	46.94061
5	14.28324	5.422039	11.05886	0.010449	0.083208	6.228485	27.59938	46.37319
6	13.76413	2.863825	9.289885	0.008356	0.058231	5.584034	30.20834	47.51309
7	4.628617	1.255188	5.234683	0.007194	0.021151	2.48741	37.86227	53.73817
8	9.279199	1.723697	9.569483	0.009397	0.038452	2.913191	37.1221	48.91397
9	5.093272	1.77406	9.569483	0.008134	0.024244	2.203288	37.41342	53.48359

ตารางที่ 6.7 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t5 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	7.911432	0	0.005378	0.010602	2.186124	37.31761	52.56885
1	17.19399	1.862347	16.14002	0.008143	0.061246	5.808342	30.40572	44.66022
2	12.46566	0.355217	12.72606	0.006416	0.019953	4.508145	32.27009	50.37452
3	18.37654	1.812571	18.75509	0.00576	0.027677	6.613761	24.67995	48.48374
4	12.38759	1.225552	15.27776	0.007762	0.025879	5.795271	31.09463	49.46331
5	4.643741	4.492174	4.72781	0.006617	0.024838	3.268516	34.23476	53.32936
6	0	5.377522	0	0.006803	0.034072	1.533698	39.41995	53.62796
7	0	4.156251	0	0.007179	0.027261	1.059958	40.55812	54.19123
8	0	4.807334	0	0.005089	0.022362	1.48853	39.84444	53.83224
9	0	1.812975	0	0.003354	0.015847	0.658455	42.22199	55.28738

ตารางที่ 6.8 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t6 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	9.883714	9.401621	9.538191	0.009606	0.038851	4.661004	36.67258	39.33262
1	14.61605	2.55665	20.59317	0.01206	0.054827	5.52813	35.71272	41.51957
2	23.31402	1.366082	29.35889	0.009019	0.052211	7.933128	30.70388	36.62166
3	4.832973	4.359299	2.69779	0.008554	0.037754	3.910583	40.6254	46.22544
4	4.806244	1.247489	15.60612	0.00941	0.021713	2.397541	43.75855	47.75905
5	0	6.44543	0	0.01147	0.048376	2.003741	45.34905	46.14194
6	0	7.565166	0	0.008906	0.037847	2.350066	44.37608	45.66194
7	9.498199	2.346068	4.63309	0.010549	0.047544	3.519621	42.4169	42.16112
8	5.009702	6.319749	4.63309	0.009408	0.036536	4.080148	39.06282	45.48164
9	0	1.92262	0	0.007672	0.015243	0.616493	49.08556	48.35241

ตารางที่ 6.9 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t7 ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	17.08871	4.177392	12.48177	0.005467	0.035962	6.552911	29.68674	42.45282
1	23.14312	0.600325	26.43325	0.006996	0.054112	7.696324	25.91392	42.5852
2	9.567511	0.22131	14.1056	0.004416	0.018803	3.962838	32.6522	53.57292
3	4.887227	3.077852	5.832262	0.005277	0.020595	3.324697	35.99889	52.68546
4	0	0.564432	0	0.00746	0.024384	0.302872	44.57584	54.52501
5	0	4.450127	0	0.006904	0.026619	1.397745	41.25653	52.86207
6	9.88263	4.734014	1.817653	0.006044	0.039611	4.451511	34.18318	46.70301
7	5.19726	2.707812	1.817653	0.005751	0.01694	2.919895	36.20691	52.94543
8	0	7.292954	0	0.005659	0.030552	2.476525	38.63972	51.55459
9	0	3.417873	0	0.006913	0.015075	0.848739	41.95676	53.75464

ตารางที่ 6.10 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t8 ณ.พื้นที่ 500x500 ตร.ม.

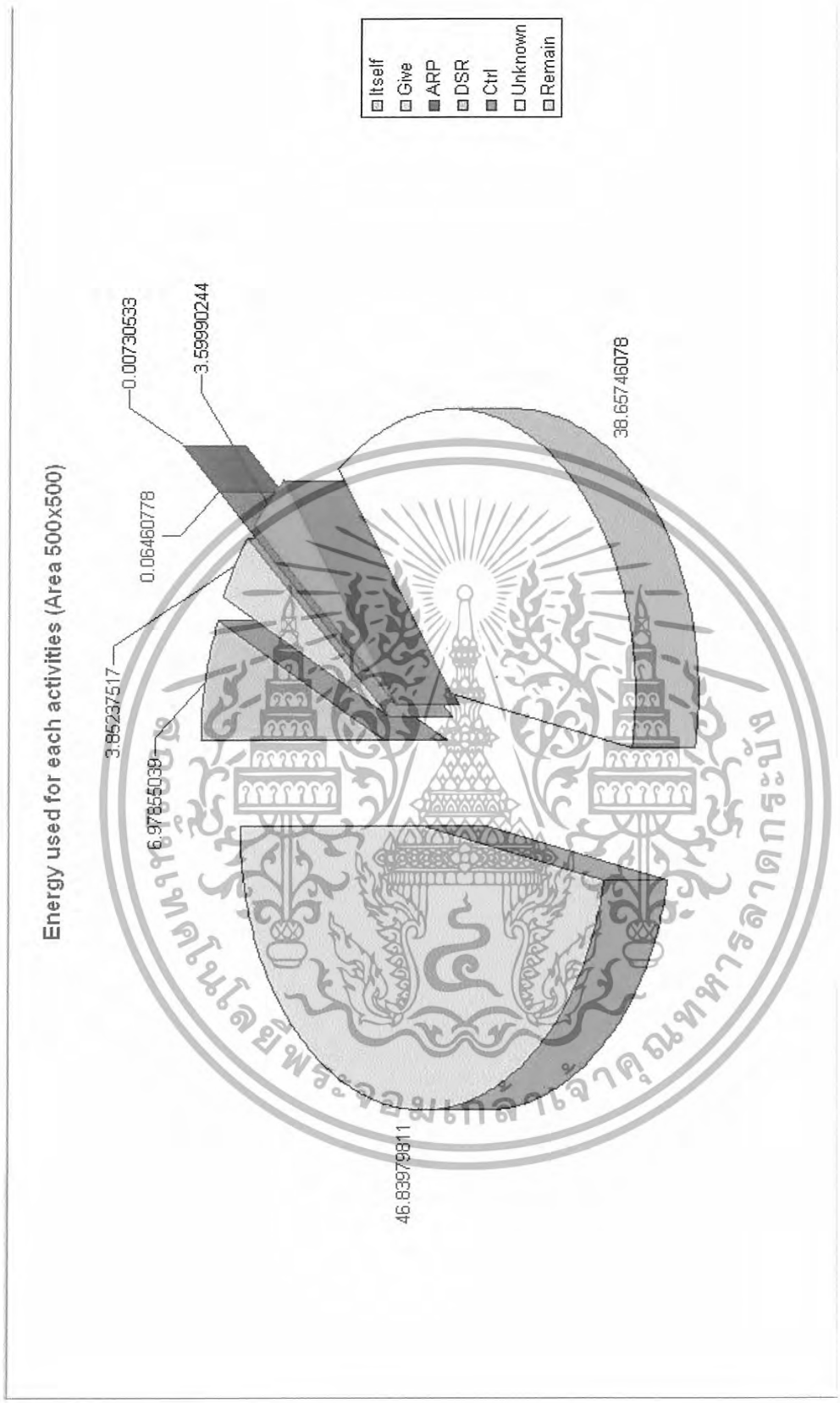
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	17.82734	5.578451	19.55194	0.006357	0.163562	7.228694	30.46328	38.73231
1	4.645969	1.735643	8.590068	0.005096	0.061298	2.564687	40.28908	50.69822
2	4.849326	3.37779	10.96188	0.007401	0.072245	3.395521	38.4561	49.84162
3	9.021853	6.958879	6.310318	0.005572	0.131203	5.157229	35.20921	43.51605
4	22.75146	2.059271	18.67289	0.006902	0.112725	7.201144	27.99583	39.87267
5	4.915513	6.488682	7.222549	0.004719	0.109357	3.881821	35.97419	48.62572
6	4.610504	3.239375	5.140025	0.005183	0.114392	2.78337	39.17417	50.073
7	0	1.962287	0	0.005869	0.075088	0.620418	46.06819	51.26815
8	0	2.253953	0	0.004982	0.066939	1.251666	45.09707	51.32539
9	0	4.570504	0	0.005872	0.087931	1.531153	43.71855	50.08599

ตารางที่ 6.11 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t9 ณ.พื้นที่ 500x500 ตร.ม.

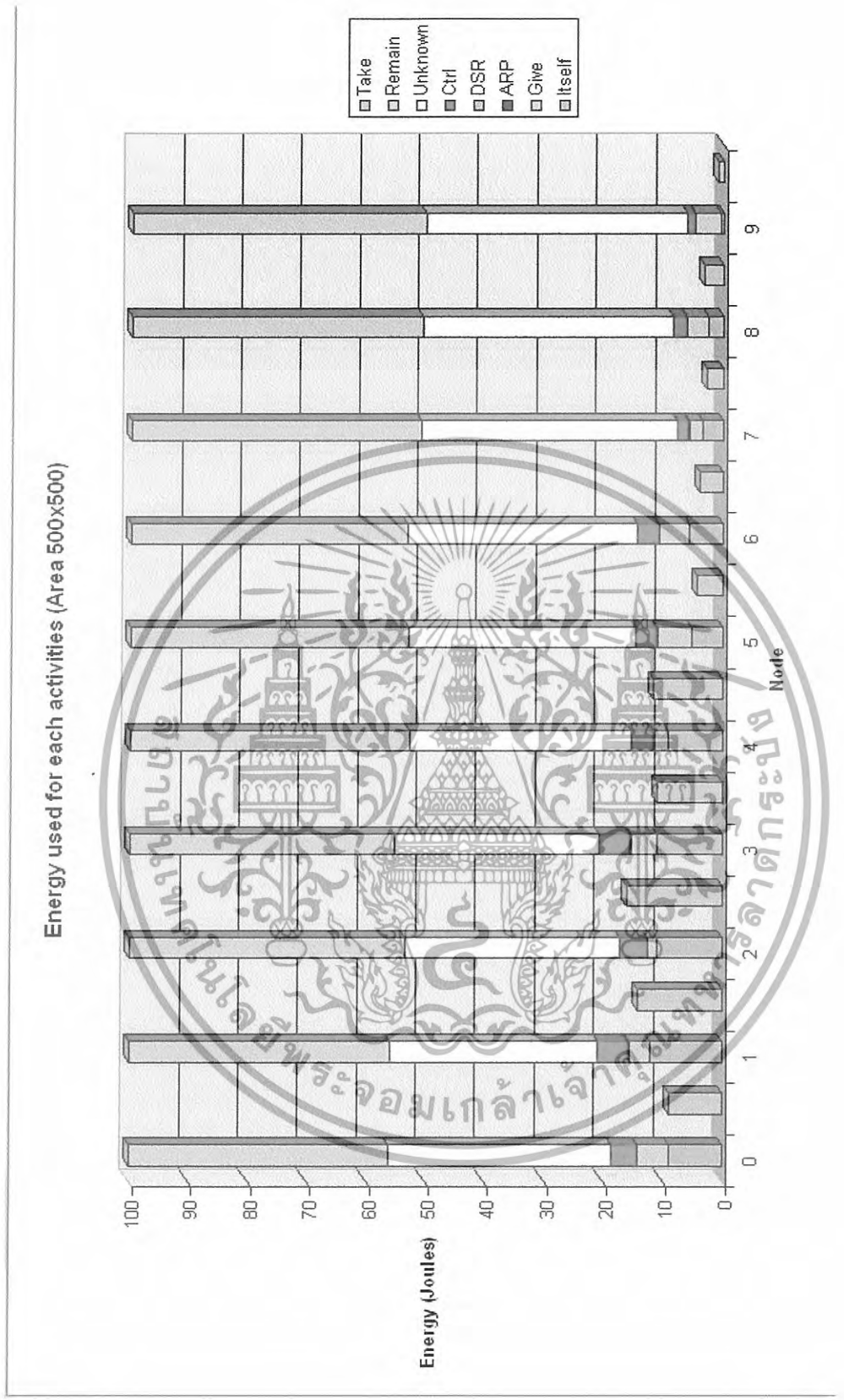
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	2.977117	0	0.008795	0.070833	1.247667	45.30819	50.3874
1	19.33094	7.441044	18.16612	0.007096	0.12542	7.872008	28.30776	36.91574
2	5.278942	1.225392	8.897888	0.005079	0.042302	2.387924	40.49532	50.56504
3	4.714455	2.980168	9.268236	0.007383	0.06597	3.694204	38.70917	49.82865
4	0	2.476815	0	0.01112	0.079141	0.819156	45.93275	50.68102
5	10.19481	7.675877	4.70255	0.013155	0.128067	4.851752	34.69794	42.4384
6	22.90645	3.90332	17.15374	0.006039	0.112003	9.06811	25.64603	38.35805
7	4.733569	1.159066	5.361927	0.008894	0.062499	2.43655	41.01085	50.58857
8	4.85018	2.690089	7.089262	0.007873	0.051498	3.229203	39.33353	49.83762
9	0	2.790975	0	0.007003	0.053935	0.733298	45.80489	50.6099

ตารางที่ 6.12 แสดงค่าพลังงานเฉลี่ยของแต่ละกิจกรรม ณ.พื้นที่ 500x500 ตร.ม.

node no	Itself	Give	ARP	DSR	Ctrl	Unknown	Remain	Take
0	9.068555	5.206022	0.006935	0.082284	4.279759	37.52894	43.8275	8.908096
1	12.10953	3.628462	0.007398	0.06863	5.028271	35.11708	44.04063	14.24202
2	10.93237	1.729469	0.007062	0.054523	4.588111	36.25018	46.43828	16.16389
3	11.08204	4.368142	0.007091	0.073917	5.296844	34.44243	44.72953	11.12419
4	9.268984	2.290993	0.008025	0.061287	3.890277	37.24353	47.2369	11.67328
5	5.379155	5.755979	0.008141	0.079166	3.562482	38.21445	47.00062	4.297648
6	5.62625	5.185812	0.007313	0.077117	3.65292	38.67624	46.77434	3.806106
7	3.361242	2.475534	0.007471	0.051527	1.991212	42.97004	49.14297	2.725441
8	2.448052	3.692339	0.007093	0.048721	2.302033	42.09582	49.40594	3.149889
9	0.509327	4.190999	0.006525	0.048906	1.407114	44.03588	49.80125	0.956948



รูปที่ 6.1 กลางแท่งที่ช้อนแต่ละชิ้นรวม (พื้นที่ 500x500)



รูปที่ 6.2 พลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 500x500)

ที่พื้นที่ 750x750

ตารางที่ 6.13 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t0 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	8.316259	2.662601	9.777026	0.007397	0.196756	3.868931	38.47171	46.48725
1	11.79578	3.185122	15.56547	0.006454	0.182768	6.077957	32.70783	46.05428
2	4.126121	5.170023	5.788444	0.00787	0.198604	4.539262	36.25948	49.71259
3	14.26237	2.742099	20.79691	0.01651	0.239985	6.42823	33.15559	43.15908
4	4.289738	0.971082	12.1858	0.00616	0.102363	2.089872	39.2346	53.31637
5	3.492051	0.871421	8.611108	0.024566	0.075924	1.891119	37.57872	56.07638
6	0	3.565598	0	0.005743	0.145203	2.285055	40.80445	53.20414
7	9.078122	8.594475	2.873788	0.010238	0.196226	6.779958	31.89531	43.46075
8	5.073429	6.764919	2.873788	0.010728	0.17588	5.410573	33.04248	49.53309
9	0	4.708828	0	0.029559	0.110971	1.789569	38.85166	54.51961

ตารางที่ 6.14 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t1 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	16.58867	5.185744	9.65475	0.010473	0.251035	8.683983	37.62565	31.65444
1	4.455793	5.477925	9.071774	0.008346	0.143298	3.477741	44.35584	42.08106
2	20.61775	2.851781	32.75399	0.008153	0.283727	7.574745	35.2027	33.46115
3	12.15314	4.428938	23.41798	0.010016	0.15797	7.191079	39.13299	36.92587
4	8.657883	1.70697	34.20186	0.006419	0.047763	4.453207	41.01651	44.11125
5	0	3.86452	0	0.030744	0.171551	1.458229	44.94316	49.5318
6	0	2.784729	0	0.007769	0.095136	2.021892	49.83151	45.25897
7	0	11.22633	0	0.007659	0.244933	4.43207	44.13887	39.95014
8	0	9.392784	0	0.007387	0.214061	5.157572	44.38367	40.84453
9	0	7.630459	0	0.029314	0.165059	2.385759	42.64086	47.14855

ตารางที่ 6.15 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t2 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	5.329023	0	0.00876	0.25336	2.024053	50.15113	42.23367
1	15.45717	4.814926	15.98597	0.009886	0.244881	8.56163	38.04064	32.87087
2	4.5289	6.230001	7.921873	0.008232	0.268058	4.005008	43.97457	40.98523
3	11.75674	7.439245	20.37967	0.008819	0.232346	7.197325	37.22377	36.14175
4	9.295325	2.320445	30.37864	0.004902	0.161281	5.050378	41.01008	42.15759
5	8.461187	1.567267	38.24222	0.006911	0.244867	3.946216	39.81873	45.95483
6	2.791572	3.259395	20.17914	0.007481	0.098886	2.901685	44.51448	46.42651
7	0	10.25245	0	0.011962	0.247361	4.18746	44.85701	40.44376
8	0	13.78269	0	0.01213	0.245056	5.312331	41.61911	39.02868
9	0	11.54832	0	0.008396	0.21307	3.766564	40.31701	44.14664

ตารางที่ 6.16 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t3 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	16.97512	3.568982	9.34345	0.008808	0.223623	8.456928	37.98012	32.78642
1	4.520896	5.826547	8.742299	0.007945	0.138761	3.494298	43.22827	42.78329
2	20.75788	4.00394	32.74807	0.005458	0.29293	8.194071	33.09651	33.64921
3	11.23121	4.121433	23.82901	0.011035	0.181878	6.31942	39.42443	38.71059
4	7.770699	1.698959	34.15133	0.005841	0.046723	4.139517	41.33918	44.99908
5	0	2.935629	0	0.009397	0.163801	1.169838	45.0478	50.67354
6	0	2.67069	0	0.00805	0.078981	1.897005	49.24929	46.09598
7	0	13.2831	0	0.008441	0.242768	5.012826	41.53878	39.91409
8	0	8.835116	0	0.007495	0.199248	4.999794	44.4719	41.48644
9	0	7.46268	0	0.015138	0.145245	2.222139	42.25667	47.89812

ตารางที่ 6.17 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t4 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	2.808351	0	0.013626	0.235044	0.925487	49.61643	46.40106
1	0	8.983006	0	0.007968	0.207639	3.453297	42.47319	44.8749
2	0	5.496851	0	0.011175	0.25473	2.984421	46.60264	44.65018
3	6.956217	8.884221	13.00528	0.034263	0.260762	5.852054	37.64892	40.36357
4	9.372523	2.168438	32.22141	0.014984	0.197849	5.30648	38.89544	44.04429
5	8.050928	0.594104	36.84097	0.007295	0.21081	3.306042	36.11516	51.71566
6	9.453139	2.659008	21.78732	0.015832	0.128223	5.017039	39.04909	43.67767
7	3.715214	8.342928	4.16248	0.020549	0.247505	5.176518	38.85621	43.64107
8	7.688202	12.46162	9.437351	0.033873	0.258048	7.740262	34.60309	37.2149
9	4.382604	11.04756	9.437351	0.013267	0.164995	4.715323	32.90673	46.76775

ตารางที่ 6.18 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t5 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	3.702771	0	0.016271	0.111465	1.9665	40.33189	53.8711
1	13.36407	6.664742	24.18335	0.033466	0.201316	7.090418	34.99784	37.64816
2	11.1876	6.756504	21.76686	0.005747	0.130958	6.223114	32.36127	43.3348
3	15.21778	1.643389	41.35562	0.014687	0.152604	6.129918	33.21418	43.62744
4	12.51388	6.547511	21.95558	0.025456	0.158559	6.976073	33.62661	40.15191
5	4.521859	1.432475	4.748294	0.005175	0.121506	2.466278	41.12251	50.3302
6	0	1.476465	0	0.004698	0.125939	0.551202	46.6145	51.2272
7	0	5.733287	0	0.015414	0.164132	2.485492	46.24503	45.35664
8	0	12.71782	0	0.016284	0.180052	4.412788	39.85937	42.81369
9	0	10.32988	0	0.005948	0.16306	3.665038	39.68453	46.15154

ตารางที่ 6.19 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t6 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	6.097989	1.841522	17.83128	0.006711	0.184589	3.127595	39.68861	49.05299
1	11.20555	4.594565	31.22287	0.005658	0.180558	5.825143	37.45719	40.73134
2	20.62582	5.312586	31.31471	0.009103	0.250282	9.617308	28.02401	36.16089
3	4.597922	5.379526	10.5039	0.010258	0.197102	3.499809	40.10869	46.2067
4	4.511047	9.482214	7.419222	0.007767	0.228043	6.075022	37.73754	41.95836
5	0	2.03793	0	0.017362	0.175993	0.937063	46.16062	50.67103
6	0	1.557864	0	0.009185	0.177463	0.76519	44.91157	52.57873
7	7.918847	7.404471	6.879665	0.01244	0.218568	5.717598	40.67698	38.0511
8	4.44219	9.291188	6.879665	0.0113	0.208083	5.911518	38.30291	41.83282
9	0	9.123791	0	0.011018	0.272936	3.627574	42.24805	44.71663

ตารางที่ 6.20 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t7 ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	11.31906	1.138235	28.02928	0.029378	0.282631	4.852913	38.21824	44.15955
1	16.86438	5.740313	39.43625	0.007192	0.271875	8.305573	34.59484	34.21582
2	6.948709	6.795	24.95363	0.024328	0.217694	5.339194	36.51488	44.1602
3	3.729943	6.298405	13.56971	0.01506	0.190298	3.884302	42.72776	43.15423
4	0	13.53644	0	0.012338	0.273085	5.776223	42.88883	37.51308
5	0	2.501046	0	0.005547	0.170262	0.927229	45.74594	50.64997
6	9.544893	1.014837	17.25646	0.008056	0.207192	3.073115	40.06733	46.08458
7	4.837822	9.774355	17.25646	0.014579	0.239594	5.542973	40.87953	38.71115
8	0	13.05938	0	0.013623	0.254852	5.34286	43.33615	37.99313
9	0	10.39288	0	0.008921	0.23857	4.379547	43.74792	41.23216

ตารางที่ 6.21 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t8 ณ.พื้นที่ 750x750 ตร.ม.

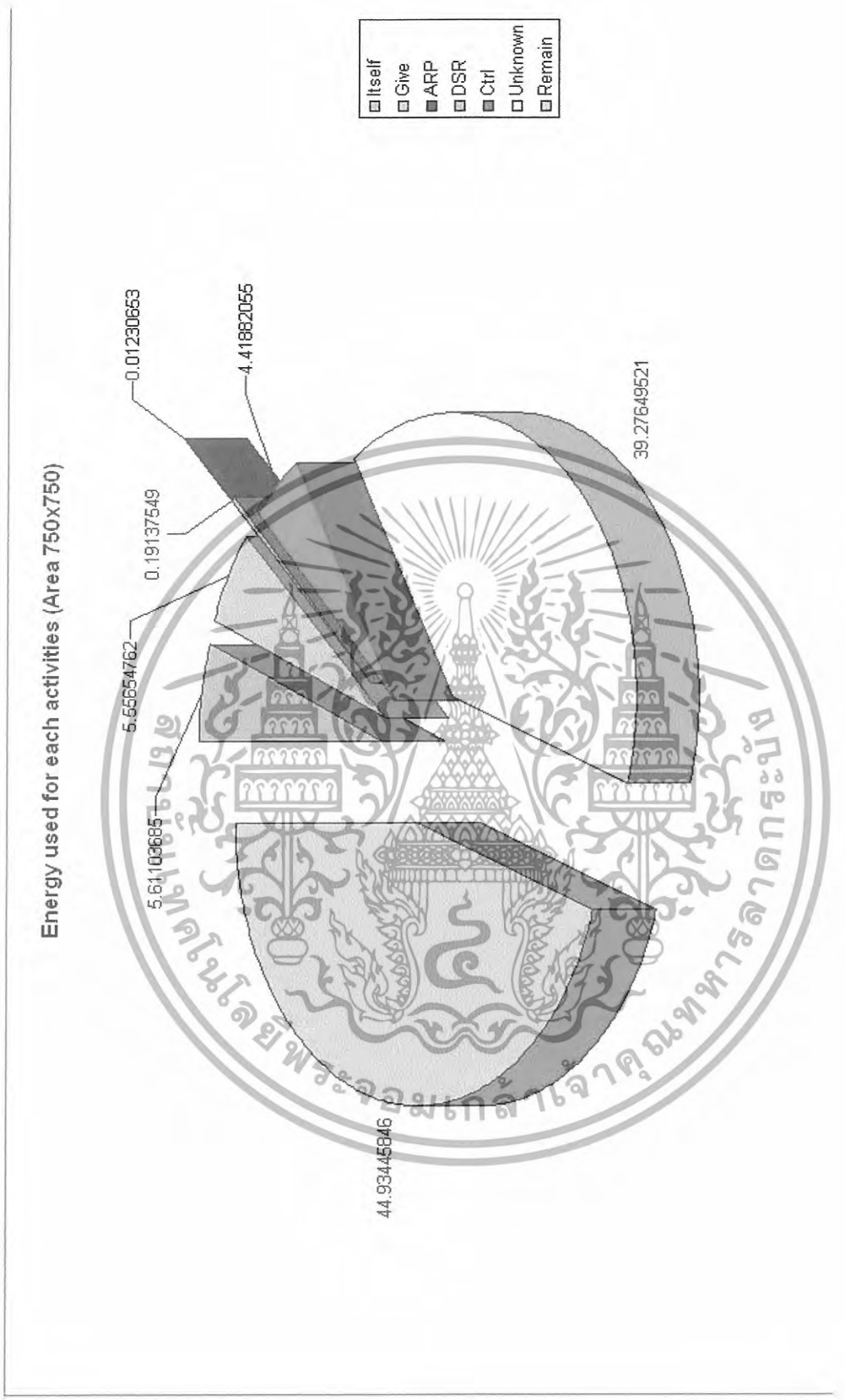
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	16.24148	1.499214	9.757319	0.018347	0.233248	7.010882	34.96847	40.02837
1	4.436363	4.068571	9.147253	0.007769	0.172562	3.920126	37.54514	49.84947
2	4.866956	4.36786	0.610066	0.012658	0.222612	4.085736	38.422	48.02218
3	7.36234	5.582562	12.72991	0.011348	0.24734	4.703232	37.69246	44.40072
4	14.32495	0	39.91383	0.004023	0.189748	6.776017	34.42555	44.27971
5	2.750826	1.18666	17.5486	0.034874	0.119371	1.524464	37.71273	56.67108
6	3.313095	1.48522	9.635328	0.003639	0.097022	2.541359	42.19049	50.36918
7	0	8.987318	0	0.012072	0.247437	4.150497	39.77262	46.83006
8	0	14.87835	0	0.011996	0.226962	5.896295	34.76942	44.21698
9	0	7.615397	0	0.020829	0.140311	2.461985	37.16658	52.5949

ตารางที่ 6.22 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t9 ณ.พื้นที่ 750x750 ตร.ม.

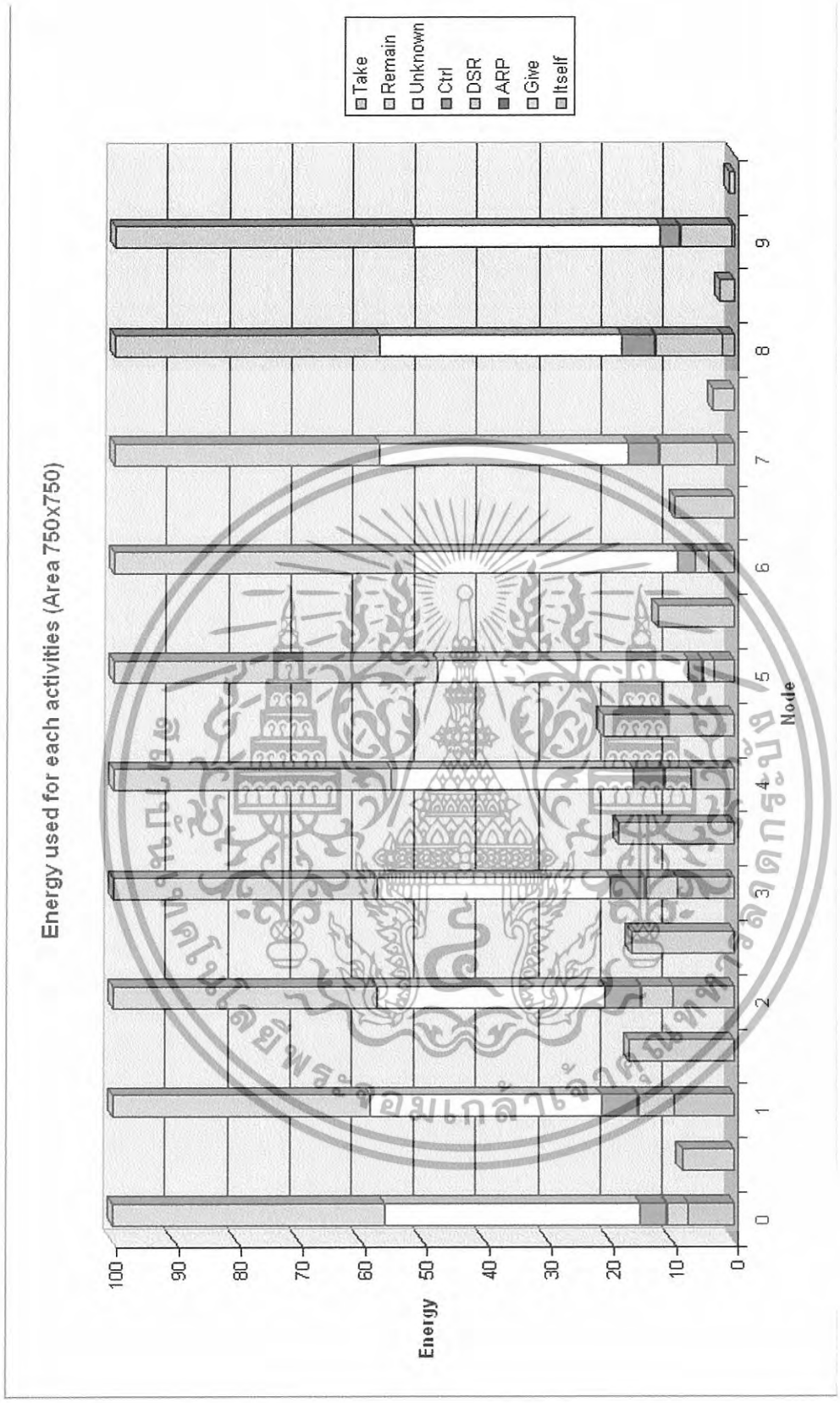
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	3.839957	0	0.007524	0.189295	1.950004	41.50908	52.50414
1	15.24323	6.539661	16.573	0.007029	0.240477	9.005633	25.90626	43.05771
2	4.561794	4.820317	8.197419	0.009187	0.206336	3.392932	35.31152	51.69791
3	4.124533	4.916109	8.375582	0.005853	0.152495	3.321371	34.42807	53.05157
4	0	3.722675	0	0.022216	0.105392	1.529195	39.37639	55.24413
5	5.66566	1.591113	18.32107	0.006718	0.216899	2.920874	30.54478	59.05396
6	15.64804	1.589041	27.94594	0.006425	0.150596	6.805852	27.26873	48.53131
7	3.602148	7.685298	4.028417	0.021597	0.173618	4.80396	32.89694	50.81644
8	3.440269	5.050537	5.596455	0.015431	0.189597	3.428947	35.65486	52.22036
9	0	4.76423	0	0.008724	0.179028	1.825912	36.79213	56.42998

ตารางที่ 6.23 แสดงค่าพลังงานเฉลี่ยของแต่ละกิจกรรม ณ.พื้นที่ 750x750 ตร.ม.

node no	Itself	Give	ARP	DSR	Ctrl	Unknown	Remain	Take
0	7.553857	3.15764	0.01273	0.216105	4.286728	40.85613	43.9179	8.439311
1	9.734323	5.589538	0.010171	0.198414	5.921182	37.1307	41.41669	16.99282
2	9.822153	5.180486	0.010191	0.232593	5.595579	36.57696	42.58343	16.60551
3	9.139221	5.143593	0.013785	0.201278	5.452674	37.47569	42.57415	18.79636
4	7.073605	4.215474	0.011011	0.151081	4.817198	38.95507	44.77758	21.24277
5	3.294251	1.858217	0.014859	0.167098	2.054735	40.47902	52.13284	12.43123
6	4.075074	2.206285	0.007688	0.130464	2.785939	42.45014	48.34543	9.680419
7	2.915215	9.128401	0.013495	0.222214	4.828935	40.17573	42.71752	3.520081
8	2.064409	10.62344	0.014025	0.215184	5.361294	39.0043	42.71846	2.478726
9	0.43826	8.462402	0.015111	0.179325	3.083941	39.66122	48.16059	0.943735



รูปที่ 6.3 กลางแก้ไขของแต่ละกิจกรรม (พื้นที่ 750x750)



รูปที่ 6.4 กราฟแสดงพลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 750x750)

ที่พื้นที่ 1000x1000

ตารางที่ 6.24 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t0 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	3.105656	1.781569	7.889403	0.004133	0.212569	1.746291	15.09764	78.05214
1	3.325227	0.039918	13.90326	0.002974	0.167714	1.52791	19.38545	75.55081
2	0.649249	0.180677	6.013861	0.012858	0.090603	0.418509	18.01817	80.62993
3	13.31748	3.701473	5.779393	0.005295	0.214012	5.430788	13.48255	63.84841
4	3.057183	3.144242	3.660301	0.006447	0.129824	2.565907	15.13192	75.96448
5	3.760118	5.972201	2.119092	0.005609	0.179056	5.657446	14.30035	70.12522
6	0	2.309771	0	0.028356	0.145117	0.931546	22.87097	73.71424
7	4.838421	1.255452	4.768025	0.003768	0.22683	1.755302	17.65537	74.26486
8	2.15112	4.455464	4.768025	0.005045	0.144067	3.473099	18.73137	71.03983
9	0	1.609915	0	0.004701	0.165655	0.597465	21.54662	76.07565

ตารางที่ 6.25 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t1 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	5.4092	1.491152	11.66377	0.019767	0.260695	2.899452	14.23823	75.6815
1	1.44902	0.0632	7.353248	0.011333	0.076763	0.598249	18.77353	79.02791
2	7.073081	0.113637	18.51968	0.006042	0.200325	4.385943	15.24691	72.97406
3	7.664168	5.169518	10.26322	0.007004	0.158583	5.047349	14.26827	67.68511
4	5.335009	2.192303	11.42477	0.004747	0.101729	3.19575	14.0471	75.12336
5	0	6.593402	0	0.018056	0.146076	3.958146	18.35096	70.93336
6	0	2.770429	0	0.027881	0.099016	1.544659	22.40127	73.15675
7	0	4.72232	0	0.040546	0.190591	1.496471	16.90501	76.64506
8	0	4.603297	0	0.011066	0.093575	3.340995	19.86454	72.08653
9	0	1.893089	0	0.032103	0.153596	0.578815	22.90895	74.43345

ตารางที่ 6.26 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t2 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0.571627	0	0.007466	0.144318	0.421178	15.92233	82.93308
1	7.356262	0.377877	11.26307	0.006142	0.240499	3.661398	21.76499	66.59283
2	1.03634	0.314991	5.994712	0.00869	0.065339	0.607353	23.86563	74.10165
3	8.498201	5.587146	9.067369	0.016118	0.267437	5.710591	17.1268	62.79371
4	8.770199	1.988521	6.993271	0.005481	0.218135	4.048435	17.77581	67.19342
5	8.09277	7.150501	12.48242	0.008162	0.28148	6.123126	14.84936	63.4946
6	2.407404	1.869186	9.288157	0.00975	0.102779	1.730871	21.50519	72.37482
7	0	2.504665	0	0.011127	0.1987	1.054221	19.33645	76.89484
8	0	4.829274	0	0.00669	0.214821	3.30981	22.74801	68.89139
9	0	2.350712	0	0.00851	0.138752	0.967359	25.73915	70.79552

ตารางที่ 6.27 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t3 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	6.868281	1.263699	14.04901	0.00607	0.243164	3.171945	15.49198	72.95487
1	1.622672	1.255764	7.888234	0.015682	0.089264	1.251332	19.57518	76.1901
2	7.354502	0.183717	21.05012	0.008836	0.198515	4.753228	16.43146	71.06974
3	7.509027	5.463517	10.6874	0.005752	0.127959	4.931437	16.15854	65.80377
4	5.192926	2.692987	12.24642	0.005087	0.09972	3.157054	16.27341	72.57882
5	0	6.938799	0	0.020377	0.136002	3.90474	19.96286	69.03722
6	0	3.356377	0	0.020008	0.09819	2.022889	24.00498	70.49755
7	0	5.76615	0	0.007777	0.188182	1.800933	17.85902	74.37794
8	0	4.138189	0	0.006566	0.096597	3.169301	20.79139	71.79796
9	0	1.901395	0	0.016512	0.144335	0.64013	25.29457	72.00306

ตารางที่ 6.28 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t4 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	2.843364	0	0.029256	0.119035	1.480239	19.49799	76.03012
1	0	0.487219	0	0.025499	0.138964	0.221649	27.83337	71.2933
2	0	1.137496	0	0.013757	0.106426	0.773081	25.5342	72.43505
3	5.929654	10.77318	4.168807	0.008631	0.311452	6.076023	18.08442	58.81665
4	9.353871	1.6316	7.368884	0.021291	0.224977	3.830038	18.04111	66.89711
5	8.306123	4.468433	9.749647	0.014438	0.316279	5.666115	18.52625	62.70236
6	6.086506	1.208337	11.64352	0.005693	0.178701	2.63891	24.10922	65.77263
7	1.266773	1.822943	5.093947	0.018699	0.151122	1.472951	22.59072	72.67679
8	3.779703	5.03794	12.19251	0.00594	0.270617	5.604969	22.63083	62.67001
9	2.571122	1.794399	12.19251	0.015967	0.139593	1.366092	24.97075	69.14208

ตารางที่ 6.29 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t5 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	3.764096	0	0.007268	0.222228	2.014945	29.66521	64.32626
1	11.63838	3.004639	15.39318	0.018903	0.328638	7.77815	23.70541	53.52588
2	7.535345	1.897308	18.40062	0.006774	0.204606	3.61515	25.22185	61.51897
3	11.16835	1.599202	23.22214	0.00634	0.174004	5.280427	22.13415	59.63753
4	6.945761	2.099279	9.801939	0.010625	0.217106	3.770908	24.46958	62.48675
5	1.796064	2.594695	6.53721	0.006618	0.175035	2.428097	27.17162	65.82787
6	0	6.200648	0	0.021325	0.272582	2.790936	29.55477	61.15974
7	0	2.942638	0	0.011283	0.207367	1.297331	29.30614	66.23524
8	0	0.75931	0	0.004018	0.088854	0.238817	30.06934	68.83966
9	0	11.81573	0	0.007871	0.255558	6.313416	23.27738	58.33005

ตารางที่ 6.30 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t6 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	6.636973	5.931291	9.332498	0.006897	0.287905	5.054597	31.67949	50.40285
1	10.98767	4.20892	19.46188	0.010047	0.351059	7.124017	31.62026	45.69803
2	14.24087	2.385418	34.542	0.006872	0.275255	6.492411	30.07139	46.52779
3	2.984639	1.246483	14.63879	0.004555	0.148062	1.946271	37.23427	56.43572
4	2.248962	4.567524	9.773826	0.009192	0.14052	3.000508	29.95518	60.07812
5	0	6.043685	0	0.011778	0.292504	3.075359	33.69764	56.87904
6	0	10.37045	0	0.009216	0.383333	4.309169	36.14739	48.78044
7	5.134813	3.111968	10.9638	0.008701	0.353567	2.839711	35.85901	52.69223
8	2.541518	0.851573	10.9638	0.004218	0.121445	1.458035	36.98804	58.03517
9	0	16.12098	0	0.008331	0.321947	6.917314	31.38459	45.24684

ตารางที่ 6.31 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t7 ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	10.76481	1.845242	12.52493	0.006938	0.284761	5.324897	25.79596	55.97739
1	15.44156	2.864547	23.39026	0.005193	0.249321	8.592009	23.23919	49.60818
2	6.129062	3.882015	12.1643	0.004943	0.117636	4.062634	26.64491	59.1588
3	3.951192	1.579339	9.399174	0.005665	0.099656	2.099275	30.58806	61.67681
4	0	3.67197	0	0.008173	0.197761	1.465592	30.3799	64.27661
5	0	2.453194	0	0.004973	0.173635	1.4491	31.2712	64.64789
6	5.828428	3.390035	7.998014	0.00748	0.25204	5.315079	31.29205	53.91488
7	4.048906	6.103814	7.998014	0.007824	0.194262	3.669705	26.11191	59.86358
8	0	1.306288	0	0.004082	0.080116	0.756555	31.11129	66.74167
9	0	9.640905	0	0.008318	0.247681	5.291732	29.77516	55.0362

ตารางที่ 6.32 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t8 ณ.พื้นที่ 1000x1000 ตร.ม.

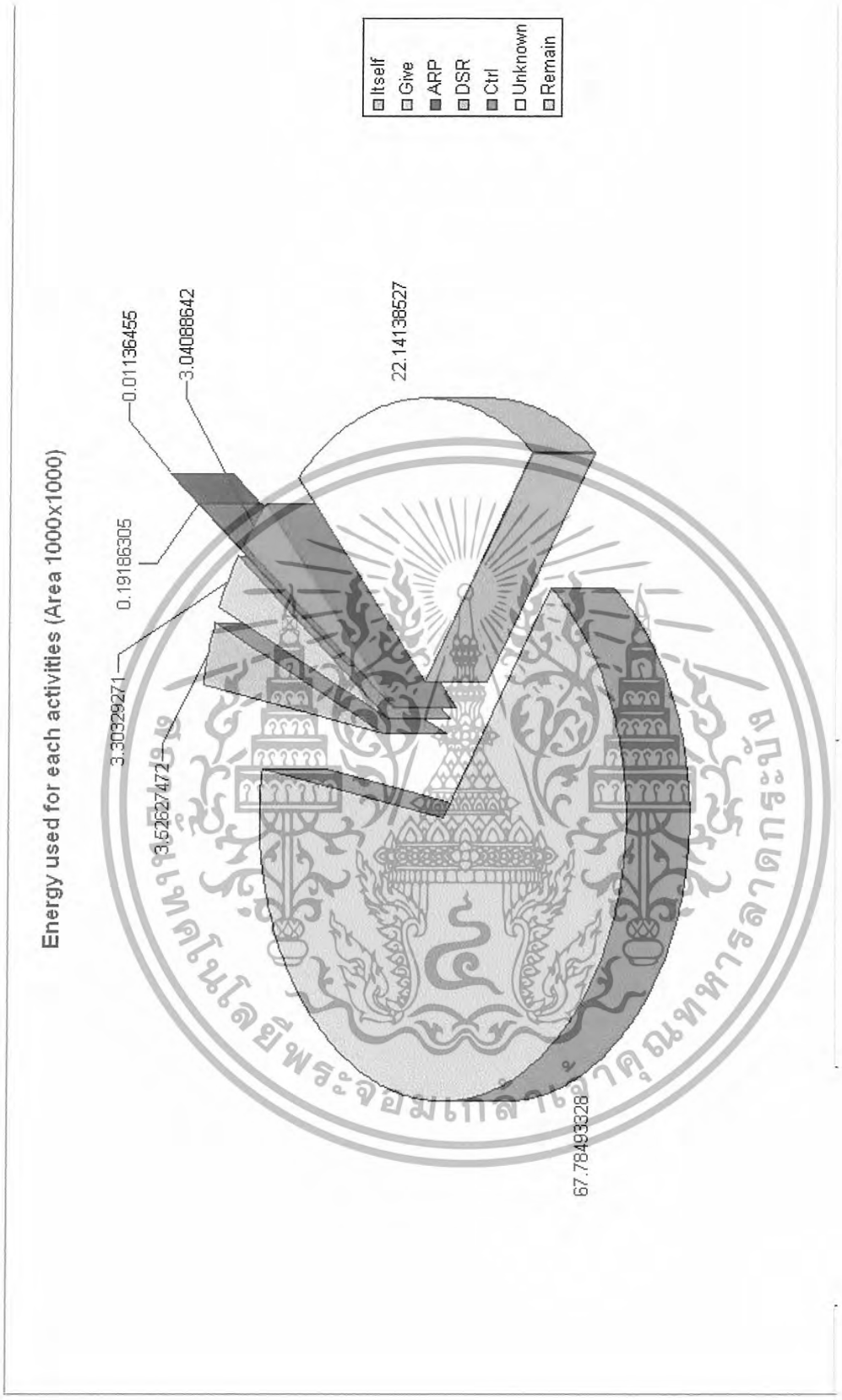
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	5.967185	0.722589	12.6288	0.004588	0.309393	2.979646	14.86814	75.14846
1	1.362701	0.562828	7.45426	0.00931	0.111266	0.80676	22.10604	75.0411
2	1.084703	0.187366	5.174538	0.004218	0.099823	0.566507	20.25916	77.79823
3	6.025725	9.473982	4.230013	0.00662	0.215271	5.418609	14.66077	64.19902
4	12.66186	1.52907	16.42007	0.018843	0.227246	5.064473	13.12989	67.36862
5	3.066938	5.292187	3.04834	0.013899	0.188782	4.86767	17.49641	69.07411
6	1.894657	1.609181	9.141713	0.004911	0.065752	1.668557	24.11161	70.64533
7	0	3.040255	0	0.008987	0.25525	1.15071	20.29967	75.24513
8	0	4.668385	0	0.004516	0.147491	3.441855	21.61696	70.12079
9	0	1.963021	0	0.027902	0.180003	0.660385	22.88458	74.28411

ตารางที่ 6.33 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t9 ณ.พื้นที่ 1000x1000 ตร.ม.

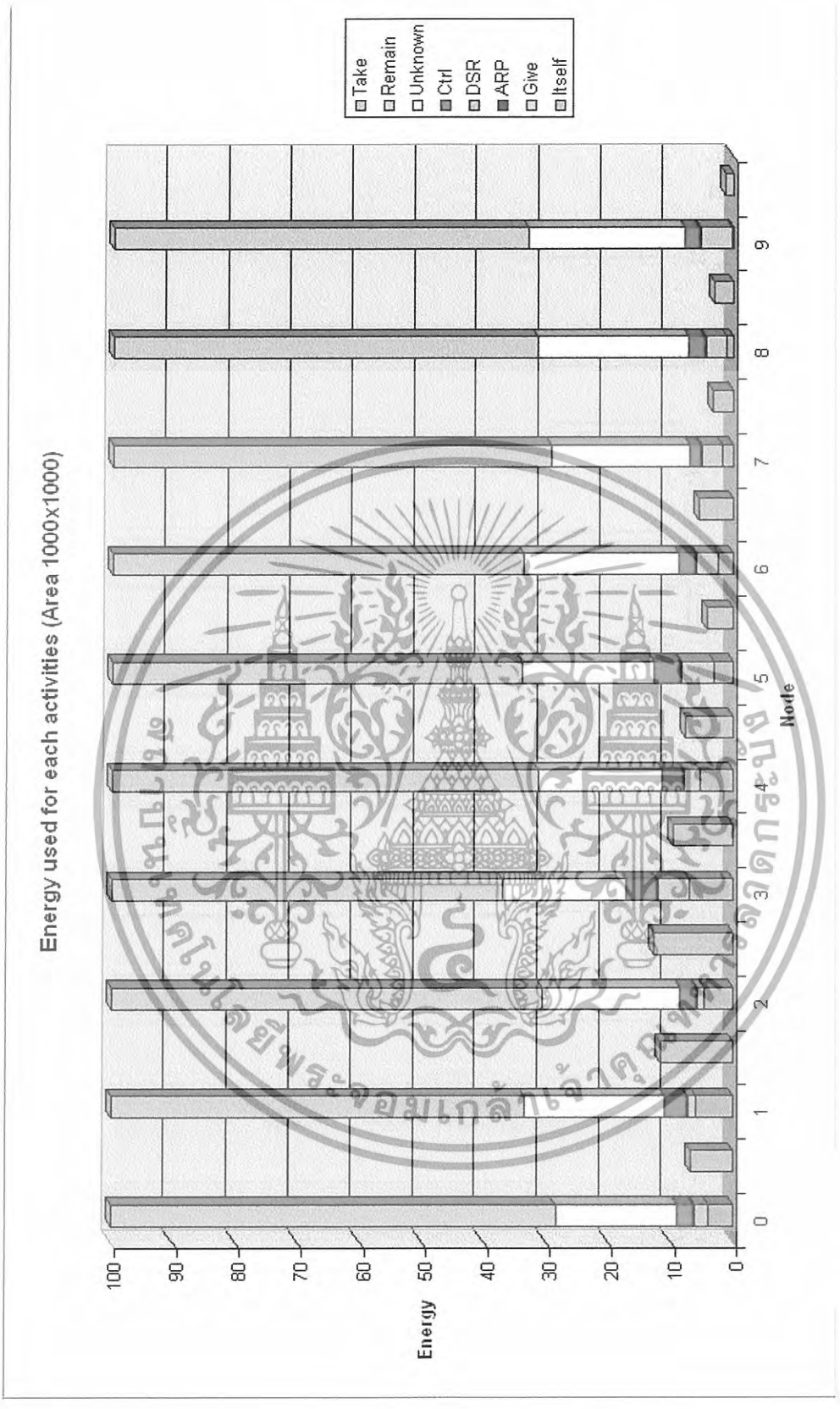
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	2.489994	0	0.00623	0.148445	1.354284	12.57533	83.42571
1	6.930089	0.108125	10.87623	0.005292	0.272399	3.540809	17.04016	72.10313
2	1.17517	0.462435	6.578774	0.01056	0.089011	0.675966	18.89722	78.68964
3	2.451583	7.861775	4.297454	0.048295	0.324374	4.586226	15.25169	69.47606
4	0	2.865332	0	0.006145	0.256643	1.763848	19.20509	75.90294
5	5.333293	6.276589	8.358259	0.022705	0.346688	4.279212	14.62856	69.11295
6	8.361212	1.362448	16.37797	0.005093	0.253739	4.499934	13.25857	72.259
7	1.11422	1.824454	4.491088	0.031331	0.151937	1.18873	16.79618	78.89315
8	2.037569	2.422339	3.528622	0.022426	0.221861	2.34831	18.89678	74.05072
9	0	1.580706	0	0.012458	0.194737	0.736514	23.21222	74.26336

ตารางที่ 6.34 แสดงค่าพลังงานเฉลี่ยของแต่ละกิจกรรม ณ.พื้นที่ 1000x1000 ตร.ม.

node no	Itself	Give	ARP	DSR	Ctrl	Unknown	Remain	Take
0	3.87521	2.270462	0.009861	0.223251	2.644747	19.48323	71.49324	6.808841
1	6.011358	1.297304	0.011038	0.202589	3.510228	22.50436	66.46313	11.69836
2	4.627832	1.074506	0.008355	0.144754	2.635078	22.01909	69.49039	12.84386
3	6.950002	5.245561	0.011427	0.204081	4.6527	19.89895	63.03728	9.575376
4	5.356577	2.638283	0.009603	0.181366	3.186251	19.8409	68.78702	7.768948
5	3.035531	5.378369	0.012661	0.223554	4.140901	21.02552	66.18346	4.229497
6	2.457821	3.444686	0.013971	0.185125	2.745255	24.9256	66.22754	5.444937
7	1.640313	3.309466	0.015004	0.211781	1.772607	22.27195	70.77888	3.331487
8	1.050991	3.307206	0.007457	0.147944	2.714175	24.34486	68.42737	3.145295
9	0.257112	5.067085	0.014267	0.194186	2.406922	25.0994	66.96103	1.219251



รูปที่ 6.5 กราฟแท่งแสดงพลังงานที่ใช้ของแต่ละอินดิวิดวม (พื้นที่ 1000x1000)



รูปที่ 6.6 กราฟแสดงพลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 1000x1000)

ที่พื้นที่ 1250x1250

ตารางที่ 6.35 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t0 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0.434836	0	0.281836	0.004981	0.11716	0.39754	0.515185	99.01396
1	5.753171	3.008605	9.157622	0.010866	0.341004	5.228676	18.94148	76.07668
2	2.970132	2.39918	8.875786	0.008132	0.134574	2.317706	20.90866	79.68797
3	12.79233	2.648388	6.737583	0.011036	0.396335	9.155112	14.3487	73.82084
4	3.376889	3.822635	2.392187	0.008831	0.15238	3.88197	12.63693	85.72924
5	4.08943	7.87217	4.345396	0.014301	0.206818	5.442408	16.76781	75.87226
6	0	2.328178	0	0.027135	0.290688	1.42292	22.74634	81.22421
7	5.504156	1.35222	10.61332	0.013062	0.365203	2.892544	22.13242	77.55833
8	2.652099	0.284542	10.61332	0.008624	0.080534	1.309466	21.60871	81.75189
9	0	2.792604	0	0.013014	0.32176	1.551004	15.75031	88.74311

ตารางที่ 6.36 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t1 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0.864496	0.025523	0.314984	0.004478	0.12069	0.822678	0.358317	97.80382
1	0	0.187452	0.080028	0.002871	0.133329	0.143831	5.975488	93.55703
2	6.057189	0.043915	4.204073	0.005433	0.222923	2.298928	4.301044	87.07057
3	4.403326	1.27926	3.325785	0.00383	0.096598	3.278847	2.724651	88.21349
4	2.855242	0.985699	2.657996	0.004215	0.016382	1.589164	3.221794	91.3275
5	0	0.683115	0	0.005842	0.106399	0.346658	8.179888	90.6781
6	0	1.206483	0	0.003626	0.082776	0.737968	6.38616	91.58299
7	0	0.492697	0	0.012285	0.076757	0.268226	6.633968	92.51607
8	0	0.090425	0	0.002777	0.058863	0.087877	3.941444	95.81861
9	0	0.296864	0	0.020697	0.0657	0.233025	6.358614	93.0251

ตารางที่ 6.37 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t2 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0	0	0	0.004326	0	0.174407	99.82127
1	7.033517	0.167307	10.14335	0.023272	0.29582	3.76021	11.77986	76.94002
2	2.651827	1.539943	6.226466	0.013276	0.15234	2.276669	10.97965	82.3863
3	4.454948	2.499401	5.037745	0.03456	0.230856	3.246969	11.82335	77.70992
4	4.871961	0.857207	3.07434	0.058081	0.181789	2.753059	7.402218	83.87569
5	3.267249	4.621792	4.85539	0.013397	0.262956	3.838628	9.107228	78.88875
6	0.903549	2.007686	2.901913	0.021121	0.170403	1.705096	12.13469	83.05746
7	0	1.342306	0	0.030632	0.196954	0.917038	16.22422	81.28885
8	0	0.525607	0	0.004505	0.134528	0.410489	15.16762	83.75725
9	0	2.558352	0	0.041247	0.206134	1.157104	7.813548	88.22362

ตารางที่ 6.38 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t3 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0.869052	0.022969	0.339615	0.00533	0.11383	0.850083	0.440826	97.69791
1	0	0.444186	0.086526	0.004306	0.144578	0.319593	6.235133	92.8522
2	6.266848	0.048637	4.771599	0.0054	0.22172	2.292447	4.488729	86.67622
3	4.443341	1.294583	3.744993	0.004402	0.098431	3.248805	3.08003	87.83041
4	2.9337	1.000632	3.264659	0.005791	0.02292	1.594924	3.564985	90.87705
5	0	0.856408	0	0.006888	0.116985	0.410574	8.57863	90.03052
6	0	1.214528	0	0.003282	0.0827	0.731737	6.701284	91.26647
7	0	0.536446	0	0.007975	0.067648	0.290931	7.085473	92.01153
8	0	0.270668	0	0.003355	0.074784	0.250759	4.236456	95.16398
9	0	0.414639	0	0.015748	0.062036	0.335912	6.66352	92.50815

ตารางที่ 6.39 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t4 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0	0	0	0.009649	0	1.13541	98.85494
1	0	1.257369	0	0.00491	0.161136	0.757916	13.43356	84.3851
2	0	2.890643	0	0.003391	0.17926	1.945694	9.984151	84.99686
3	3.039243	2.038795	1.94331	0.0044	0.242021	3.057582	11.00985	80.60811
4	4.855833	1.559305	3.686756	0.028893	0.173188	3.313225	6.282931	83.78663
5	3.534767	1.271337	5.247729	0.030772	0.26112	2.468315	10.77911	81.65458
6	5.1496	1.151705	6.478322	0.029402	0.211324	3.17682	8.284329	81.99682
7	2.168876	1.047603	2.974039	0.025213	0.118554	1.528383	12.50679	82.60458
8	1.814678	0.757237	3.023832	0.031463	0.167659	1.560722	10.21223	85.45601
9	0.824204	1.214916	3.023832	0.020086	0.12717	1.184544	7.77717	88.85191

ตารางที่ 6.40 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t5 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	2.073389	0	0.036443	0.170338	1.070911	12.1962	84.45272
1	6.063984	1.275671	4.594495	0.002846	0.178733	3.839871	8.213066	80.42583
2	3.829067	2.026426	7.118161	0.015441	0.226072	2.731388	9.007489	82.16412
3	5.247956	0.004349	10.28687	0.004242	0.183258	2.008654	8.7151	83.83644
4	2.380954	0.615963	5.94831	0.011803	0.156391	1.308578	5.873424	89.65289
5	0.823206	2.352358	4.24956	0.003734	0.091173	1.66078	9.492763	85.57599
6	0	0	0	0	0.024919	0	7.333797	92.64128
7	0	2.667794	0	0.005077	0.109828	1.641514	11.39646	84.17933
8	0	5.035641	0	0.004415	0.142544	2.394735	9.629459	82.79321
9	0	0.047109	0	0.012254	0.092113	0.019142	7.538311	92.29107

ตารางที่ 6.41 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t6 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	3.306634	1.19756	4.791379	0.036186	0.275051	2.383821	11.79492	81.00583
1	4.331751	0.916041	6.714964	0.005273	0.208308	2.104387	9.115977	83.31826
2	6.659152	1.471399	7.404966	0.021533	0.291944	4.487828	8.690036	78.37811
3	1.404265	0.770908	5.209643	0.004929	0.18338	1.316471	10.23377	86.08628
4	1.642706	0.586948	0.271738	0.003877	0.11738	1.302382	7.152988	89.19372
5	0	2.311884	0	0.006418	0.194229	1.423399	9.867912	86.19616
6	0	0	0	0.000576	0.057354	0	8.549321	91.39275
7	2.822974	2.196161	1.243067	0.029157	0.239746	2.281765	10.3436	82.08659
8	1.136675	3.606707	1.243067	0.036315	0.208373	2.448604	9.928307	82.63502
9	0	0.381804	0	0.002397	0.115109	0.285134	9.102487	90.11307

ตารางที่ 6.42 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t7 ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	6.200853	0.446268	6.265969	0.007195	0.227582	3.635256	10.41813	79.06471
1	7.568713	0.299543	8.648078	0.020739	0.152316	3.777976	8.517672	79.66304
2	3.393881	1.715056	3.808701	0.005989	0.089724	2.464072	8.053751	84.27753
3	2.652999	0.068603	3.330452	0.006623	0.095974	0.543348	10.9744	85.65805
4	0	0.588558	0	0.001101	0.039405	0.286849	6.934648	92.14944
5	0	2.452459	0	0.004678	0.059952	1.75882	8.830178	86.89391
6	3.969558	0	0.942234	0.004004	0.112795	1.413495	6.754642	87.74551
7	1.918348	2.370194	0.942234	0.007277	0.106895	2.093713	11.45945	82.04412
8	0	3.88009	0	0.016011	0.108351	2.229652	9.870782	83.89511
9	0	0.148063	0	0.002368	0.091435	0.065351	8.275637	91.41715

ตารางที่ 6.43 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t8 ณ.พื้นที่ 1250x1250 ตร.ม.

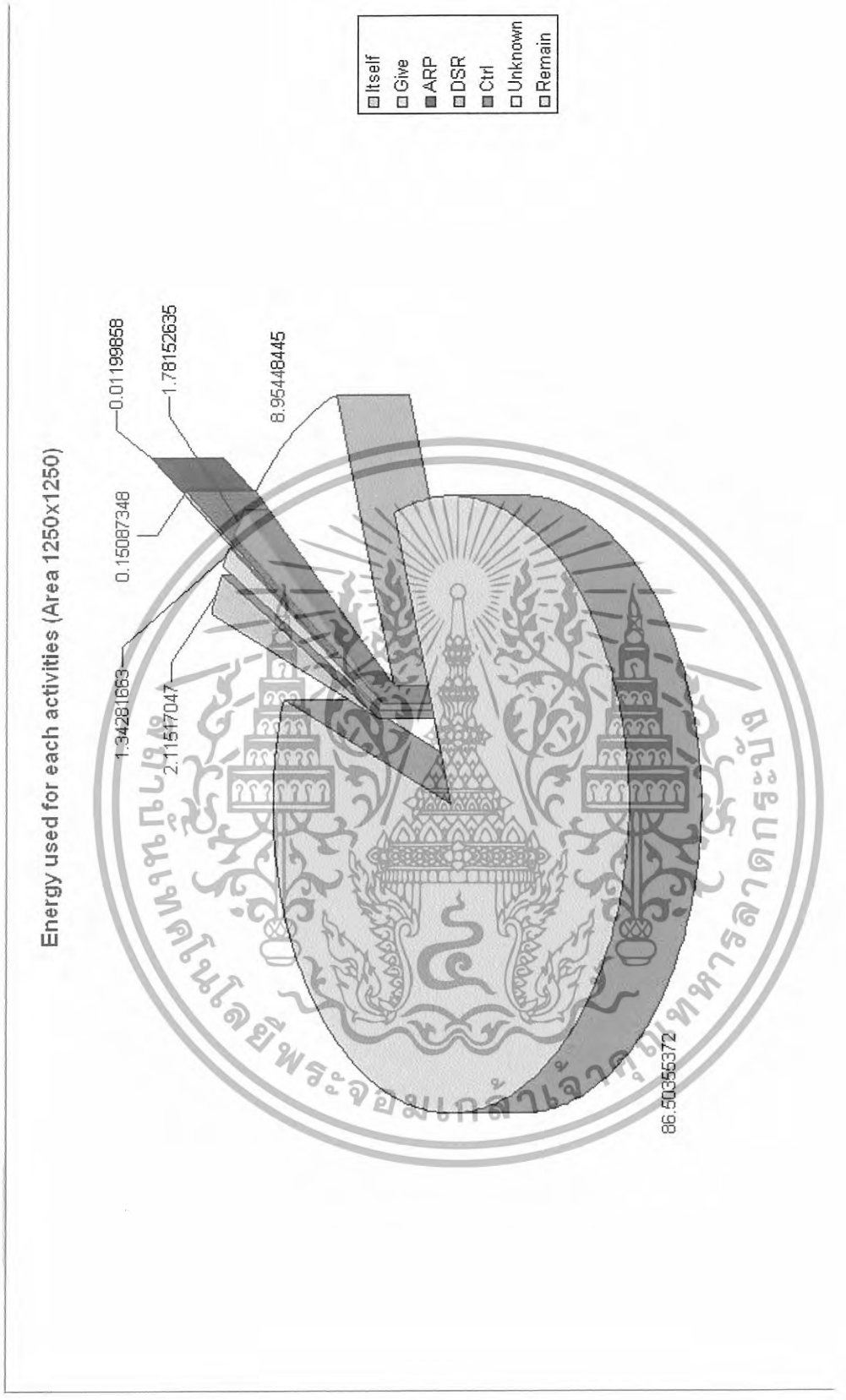
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0.810842	0	0.253449	0.002838	0.082513	0.708208	0.907679	97.48792
1	0	0.160336	0	0.006908	0.050854	0.135888	6.500428	93.14559
2	0.551658	0.151399	0.253449	0.001561	0.059373	0.303068	6.150515	92.78243
3	2.79417	0.553118	0.949787	0.006234	0.16084	2.100464	4.476343	89.90883
4	6.062307	0.045335	4.028319	0.005987	0.147641	3.455389	2.143392	88.13995
5	1.403967	0.434627	0.981089	0.004601	0.077054	0.760945	4.452662	92.86614
6	1.400134	0.703651	2.097443	0.007297	0.070658	1.073085	5.093661	91.65151
7	0	0.612653	0	0.011148	0.037894	0.337669	6.727882	92.27275
8	0	0.116014	0	0.005847	0.026992	0.077281	4.957797	94.81607
9	0	1.504635	0	0.013554	0.061841	0.73036	4.753238	92.93637

ตารางที่ 6.44 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t9 ณ.พื้นที่ 1250x1250 ตร.ม.

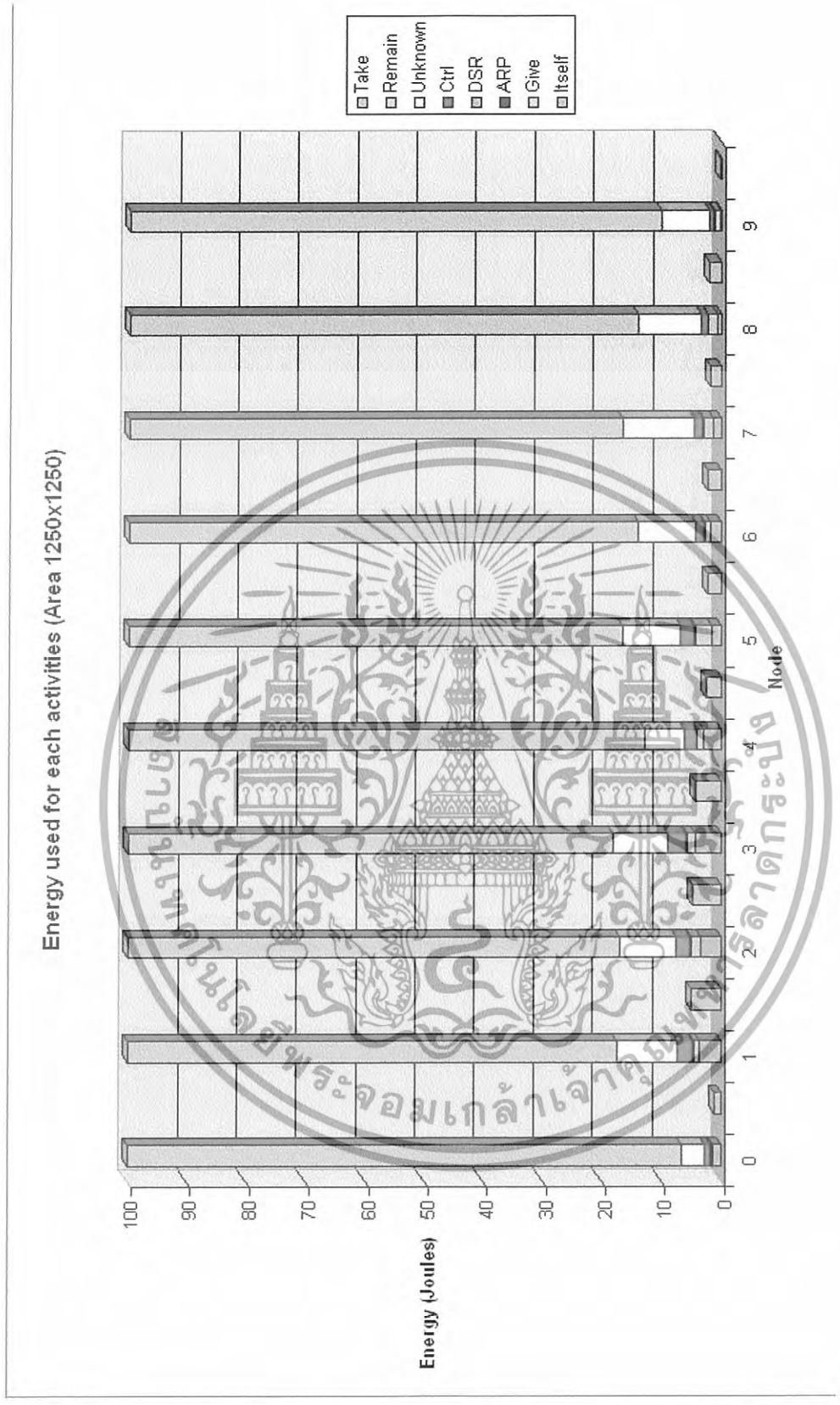
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0	0	0	0.010464	0	1.058424	98.93111
1	6.742922	1.594338	10.36904	0.006755	0.32431	4.886495	13.16625	73.27893
2	2.717124	3.293395	6.408385	0.011685	0.231804	3.071339	11.96774	78.70691
3	1.228407	3.929951	3.960657	0.014467	0.176197	2.453964	15.16958	77.02743
4	0	1.05526	0	0.023428	0.164043	0.580423	11.3205	86.85634
5	1.933403	5.567148	3.467853	0.039331	0.293695	4.400196	11.15832	76.60791
6	6.634256	0.995636	10.91175	0.035024	0.30479	3.872161	11.80881	76.34932
7	1.894261	2.603333	2.742029	0.007161	0.225155	2.037983	15.51489	77.71722
8	1.153433	0.33882	4.701863	0.004438	0.129743	0.743829	16.8349	80.79484
9	0	1.902906	0	0.005559	0.221123	0.655216	7.49544	89.71976

ตารางที่ 6.45 แสดงค่าพลังงานเฉลี่ยของแต่ละกิจกรรม ณ.พื้นที่ 1250x1250 ตร.ม.

node no	Itself	Give	ARP	DSR	Ctrl	Unknown	Remain	Take
0	1.248671	0.376571	0.009745	0.11316	0.98685	3.89995	93.41342	1.224723
1	3.749406	0.931085	0.008875	0.199039	2.495484	10.18789	83.36427	4.97941
2	3.509688	1.557999	0.009184	0.180973	2.418914	9.453176	83.7127	4.907159
3	4.246098	1.508736	0.009472	0.186389	3.041022	9.255578	83.06998	4.452683
4	2.897959	1.111754	0.015201	0.117152	2.006596	6.653381	88.15884	2.53243
5	1.505202	2.84233	0.012996	0.167038	2.251072	9.72145	84.52643	2.314702
6	1.80571	0.960787	0.013147	0.140841	1.413328	9.579303	86.89083	2.333166
7	1.430861	1.522141	0.014899	0.154463	1.428977	12.00252	84.42794	1.851469
8	0.675688	1.490575	0.011775	0.113237	1.151341	10.63877	86.6882	1.958208
9	0.08242	1.126189	0.014692	0.136442	0.621679	8.152827	90.78293	0.302383



รูปที่ 6.7 พลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 1250x1250)



รูปที่ 6.8 พลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 1250x1250)

ที่พื้นที่ 1500x1500

ตารางที่ 6.46 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t0 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	1.071472	0	0	0.000586	0.051208	0.91744	0.55932	97.39997
1	1.109439	0.110541	0.590474	0.003653	0.11368	0.605528	1.80833	96.24883
2	0	0.513743	0.590474	0.001435	0.042689	0.318197	0.500787	98.62315
3	2.079001	0.415482	1.230861	0.003542	0.206442	2.773529	1.919629	92.60238
4	1.559083	0.650121	0.927794	0.001986	0.035503	0.034253	1.580721	96.13833
5	0.421328	0.170588	0.303067	0.002778	0.046308	0.338493	1.387941	97.63256
6	0	0	0	0	0.017262	0	1.639861	98.34288
7	1.562944	0	0.932133	0.002132	0.085094	0.966787	1.897806	95.48524
8	0.933222	0.193561	0.932133	0.002103	0.04476	0.425252	1.487115	96.91399
9	0	0.699432	0	0.004353	0.090673	0.477485	2.86593	95.86213

ตารางที่ 6.47 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t1 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	1.116386	0	0.042217	0.001853	0.09703	1.054686	0.844974	96.88507
1	0.814528	0	0	0.000475	0.026818	0.300813	0.342726	98.51464
2	4.879498	0.67337	1.522518	0.002682	0.145571	4.603508	1.221977	88.47339
3	3.09812	0.045188	2.623988	0.002962	0.114136	2.836777	3.124865	90.77795
4	4.319248	0.089093	2.187355	0.003435	0.048528	0.771646	3.875197	90.89285
5	0	0.177757	0	0.0012	0.032157	0.18832	3.07975	96.52082
6	0	0.360978	0	0.001243	0.037687	0.297985	5.495757	93.80635
7	0	0	0	0.000384	0	0	2.064451	97.93517
8	0	0.454675	0	0.001485	0.01719	0.204775	1.349869	97.97201
9	0	1.386978	0	0.002201	0.050614	1.076636	2.673747	94.80982

ตารางที่ 6.48 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t2 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0	0	0	0.041222	0	2.552621	97.40616
1	1.243778	0.092203	0.542892	0.002915	0.223086	0.81093	0.756237	96.87085
2	0	1.144623	0.218742	0.002532	0.079643	0.690521	2.483751	95.59893
3	1.887793	0.283992	1.726013	0.0025	0.150082	2.964662	2.593494	92.11748
4	2.330406	0	2.279669	0.002881	0.101739	0.931892	2.229609	94.40347
5	2.814468	0.03976	1.138895	0.00221	0.061825	2.341985	2.26141	92.47834
6	1.930739	0.67076	0.261089	0.002202	0.062585	1.30959	1.546546	94.47758
7	0	0.046953	0	0.011977	0.017997	0.02674	2.95526	96.94107
8	0	0.335918	0	0.007864	0.065117	0.193395	1.171344	98.22636
9	0	0.469441	0	0.002779	0.11352	0.31179	2.232907	96.86956

ตารางที่ 6.49 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t3 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	1.279318	0	0.043712	0.002382	0.088936	1.371251	0.638595	96.61952
1	1.112283	0	0	0.000475	0.033887	0.330579	0.338568	98.18421
2	4.946205	0.754994	1.491801	0.002092	0.149323	4.64769	1.243446	88.25625
3	2.281394	0.045253	2.313797	0.002364	0.09707	2.239088	3.035846	92.29899
4	3.771141	0.08702	1.829116	-0.00803	0.048894	0.626318	3.787638	91.68702
5	0	0.182743	0	0.0012	0.03147	0.180913	3.111231	96.49244
6	0	0.350696	0	0.001804	0.038786	0.294526	5.88894	93.42525
7	0	0	0	0	0	0	1.260814	98.73919
8	0	0	0	0	0.003466	0	1.157558	98.83898
9	0	1.418507	0	0.002111	0.049752	1.109326	2.32411	95.09619

ตารางที่ 6.50 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t4 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0	0	0	0.009273	0	4.818444	76.03012
1	0	1.239349	0	0.00182	0.048278	0.887181	2.127259	71.2933
2	0	1.151702	0	0.001912	0.095842	0.673958	2.821026	72.43505
3	0.777975	0.275711	1.292441	0.003487	0.148509	2.356832	2.782591	58.81665
4	2.008815	0	2.126386	0.00309	0.12828	0.861811	2.677949	66.89711
5	2.916176	0.077107	1.655695	0.01186	0.103439	2.718341	2.403281	62.70236
6	2.704267	0.587127	0.82175	0.002001	0.119597	2.062293	1.520044	65.77263
7	0.642887	0.086141	0	0.001052	0.02783	0.296218	2.936198	72.67679
8	2.012741	0	1.068998	0.001637	0.070473	0.932815	1.289201	62.67001
9	1.071867	0.599997	1.068998	0.012153	0.066679	0.80259	2.508735	69.14208

ตารางที่ 6.51 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t5 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0.047904	0	0.012525	0.050942	0.019769	0.726783	99.14208
1	0.854859	0	0.100757	0.008679	0.128853	0.498398	0.77341	97.7358 .
2	0.841546	0.564703	0.161422	0.002687	0.118629.	0.810655	0.453364	97.20842
3	0.571744	0	0.242516	0.001861	0.0716	0.287205	0.208233	98.85936
4	0.316706	0	0.975015	0.000959	0.096163	0.090385	0.542753	98.95303
5	0.149651	0.147485	0.793164	0.001434	0.042155	0.277779	1.600718	97.78078
6	0	0.039784	0	0.000768	0.02848	0.060542	0.27426	99.59617
7	0	0.33359	0	0.002413	0.049499	0.284194	0.98257	98.34773
8	0	0	0	0	0.013949	0	0.873245	99.11281
9	0	0.002971	0	0.000859	0.055509	0.006537	0.876153	99.05797

ตารางที่ 6.52 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t6 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	1.274061	0.854231	1.526554	0.003559	0.17161	1.124632	4.19855	92.37336
1	2.377844	0.577589	1.894106	0.004003	0.139662	1.543354	3.530932	91.82662
2	3.132576	0.302961	0.875806	0.004674	0.1659	2.511689	2.134852	91.74735
3	0.20531	0	0	0.000858	0.014097	0.069191	1.25719	98.45335
4	1.400219	0.476347	0.508254	0.00172	0.077808	0.960858	2.774601	94.30845
5	0	0.587146	0	0.00259	0.077593	0.300839	6.44074	92.59109
6	0	0.663246	0	0.00182	0.048161	0.566346	3.649835	95.07059
7	2.734521	0.994323	2.335667	0.005779	0.163043	2.286327	3.830368	89.98564
8	1.703576	0.185984	2.335667	0.006328	0.046058	0.837374	4.305144	92.91554
9	0	0.0962	0	0.009306	0.070865	0.035244	7.728199	92.06019

ตารางที่ 6.53 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t7 ณ.พื้นที่ 1500x1500 ตร.ม.

node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	1.704116	0	1.835206	0.018725	0.194047	1.123976	2.15846	94.80068
1	1.712022	0.00884	1.080399	0.003949	0.133859	0.967548	2.675076	94.49871
2	1.505977	0.356893	1.084617	0.004598	0.040884	0.881422	1.515783	95.69444
3	0	0	0	0.004608	0.017749	0	0.474357	99.50329
4	0	1.015869	0	0.001957	0.049089	0.920163	1.767098	96.24582
5	0	0.450253	0	0.012735	0.060511	0.151762	3.837033	95.48771
6	1.053519	0.087907	0.733962	0.006056	0.090722	1.139213	1.70436	95.91822
7	0.839428	0.630678	0.733962	0.016061	0.095865	0.630966	2.293294	95.49371
8	0	0.12419	0	0.018123	0.064603	0.112933	2.671981	97.00817
9	0	0.059443	0	0.014942	0.076274	0.035613	3.925797	95.88793

ตารางที่ 6.54 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t8 ณ.พื้นที่ 1500x1500 ตร.ม.

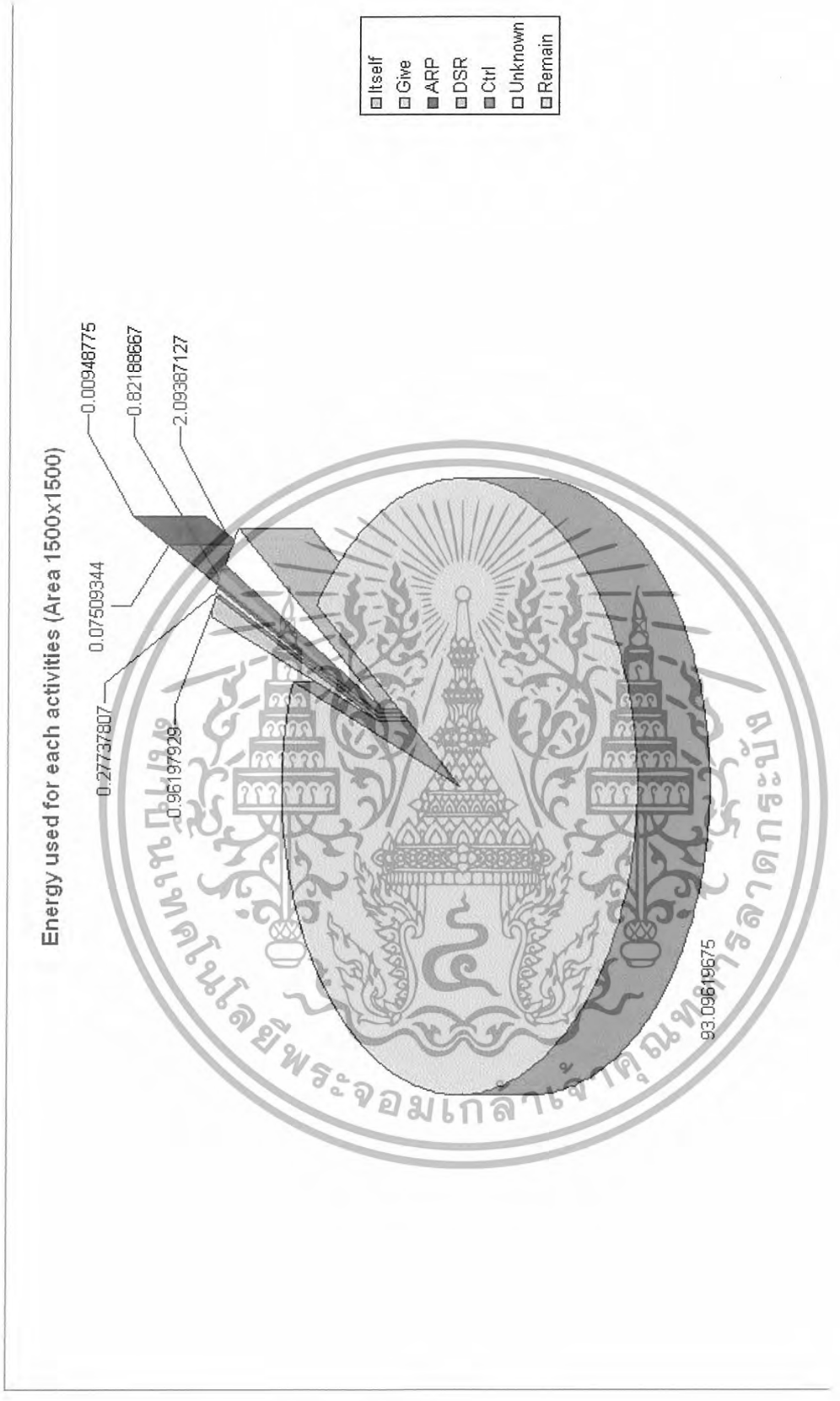
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	1.08783	0	0.04366	0.001977	0.081147	1.040107	0.511576	97.27736
1	0.77272	0	0	0.000475	0.030086	0.288599	0.223819	98.6843
2	0	1.393972	0.04366	0.002343	0.105623	0.777956	2.497392	95.22271
3	1.159954	0.04366	1.29744	0.003834	0.156781	2.558109	2.665364	93.4123
4	3.498353	0	2.383683	0.002111	0.153429	2.237363	1.245192	92.86355
5	0.258464	0.079425	0.841354	0.001344	0.024829	0.227846	2.219945	97.18815
6	1.124415	0.59683	0.244889	0.001239	0.038619	1.037995	2.406684	94.79422
7	0	0	0	0.000192	0.012347	0	1.207518	98.77994
8	0	0	0	0	0.006644	0	1.128676	98.86468
9	0	0.313456	0	0.001152	0.032853	0.251631	1.214828	98.18608

ตารางที่ 6.55 แสดงพลังงานของแต่ละกิจกรรมครั้งที่ t9 ณ.พื้นที่ 1500x1500 ตร.ม.

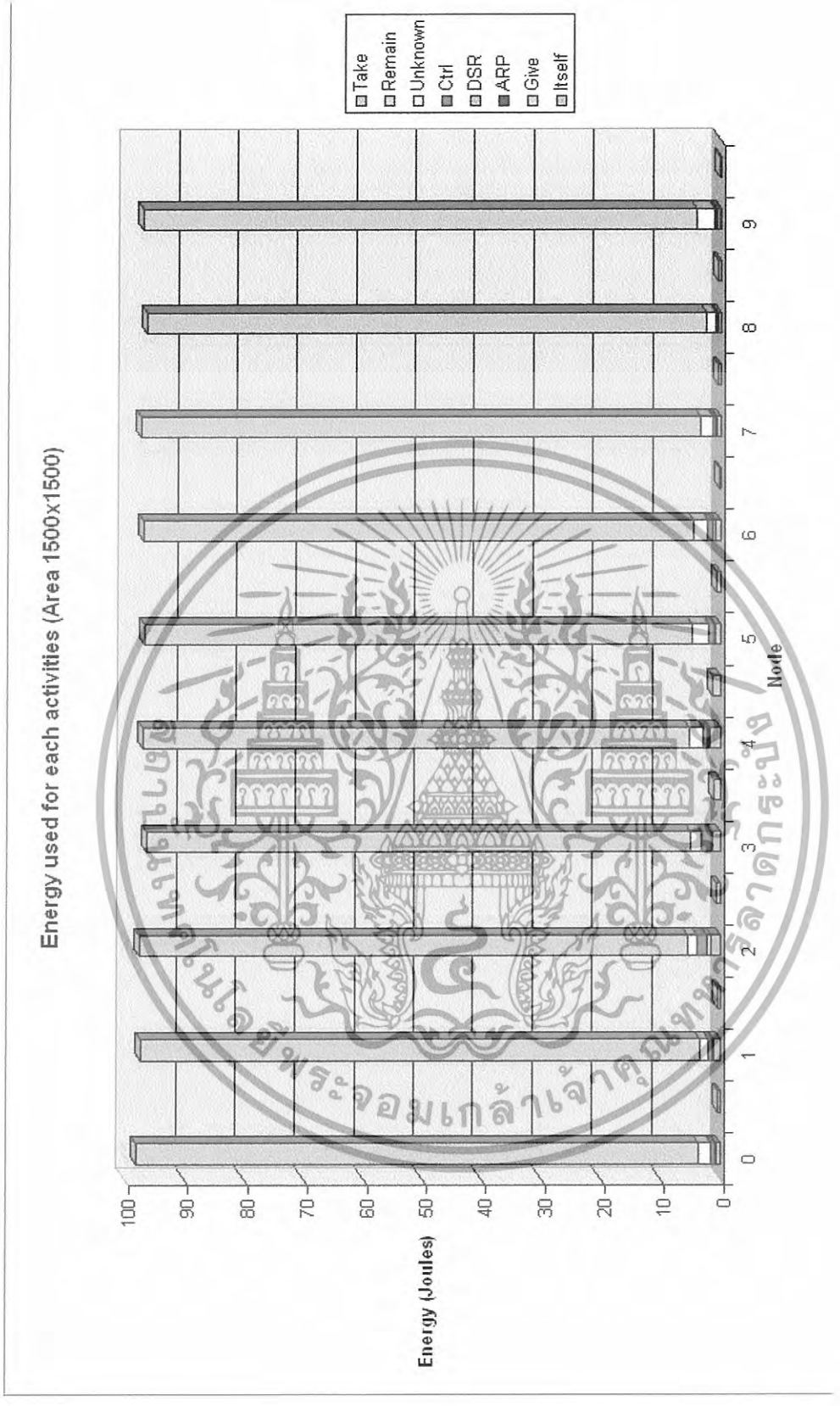
node no	Itself	Give	Take	ARP	DSR	Ctrl	Unknown	Remain
0	0	0	0	0.000192	0.037747	0	3.256346	96.70572
1	0.986541	0.261254	0	0.001162	0.199337	0.456565	0.94853	97.14661
2	0	0	0	0	0.040991	0	1.582494	98.37652
3	0.49135	0.180231	0	0.001396	0.031876	0.325921	1.033476	97.93575
4	0	0	0	0	0.049064	0	1.549499	98.40144
5	2.67502	0	0.820423	0.010627	0.1196	2.408076	1.628357	93.15832
6	2.566181	0	0.820423	0.00104	0.185676	1.364187	0.077387	95.80553
7	0.502904	0.090268	0	0.001052	0.024093	0.257751	2.107168	97.01676
8	0	0	0	0	0.040001	0	0.825257	99.13474
9	0	0.28867	0	0.02581	0.104472	0.256302	1.697074	97.69857

ตารางที่ 6.56 แสดงค่าพลังงานเฉลี่ยของแต่ละกิจกรรม ณ.พื้นที่ 1500x1500 ตร.ม.

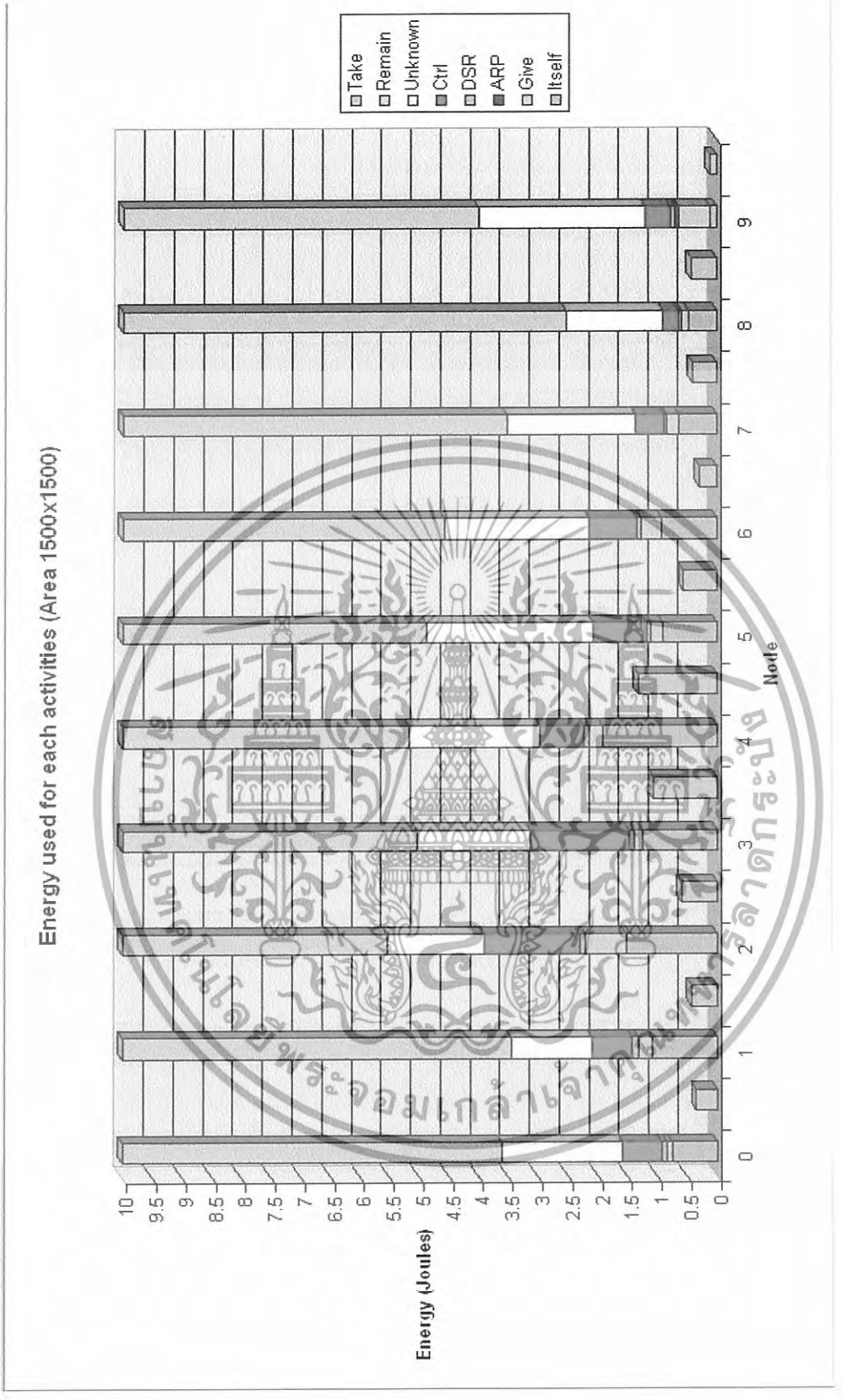
node no	Itself	Give	ARP	DSR	Ctrl	Unknown	Remain	Take
0	0.753318	0.090213	0.00418	0.082316	0.665186	2.026567	94.464	0.349135
1	1.098401	0.228978	0.002761	0.107755	0.66895	1.352489	94.10039	0.420863
2	1.53058	0.685696	0.002496	0.09851	1.59156	1.645487	92.16362	0.598904
3	1.255264	0.128952	0.002741	0.100834	1.641131	1.909504	91.47775	1.072706
4	1.920397	0.231845	0.001011	0.07885	0.743469	2.203026	92.07911	1.321727
5	0.923511	0.191226	0.004798	0.059989	0.913435	2.797041	92.20326	0.55526
6	0.937912	0.335733	0.001817	0.066758	0.813268	2.420367	92.70094	0.288211
7	0.628268	0.218195	0.004104	0.047577	0.474898	2.153545	94.14012	0.400176
8	0.464954	0.129433	0.003754	0.037226	0.270654	1.625939	94.16573	0.43368
9	0.107187	0.533509	0.06235	0.071121	0.436315	2.804748	93.46705	0.1069



รูปที่ 6.9 กลางแก้ไขของแต่ละกิจกรรม (พื้นที่ 1500x1500)



รูปที่ 6.10 กราฟแท่งแสดงพลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 1500x1500)



รูปที่ 6.11 ตัวอย่างพลังงานที่ใช้ของแต่ละกิจกรรม (พื้นที่ 1500x1500)

โดยตารางจะเป็นการนำเสนอข้อมูลดิบซึ่งได้จากการทดลองและวิเคราะห์ จากนั้นจะนำมาแสดงผลเป็นแผนภูมิเพื่อให้ง่ายต่อการศึกษาและทำความเข้าใจ โดยจากข้อมูลดังกล่าวจะสามารถสรุปข้อเท็จจริงต่างๆได้

ซึ่งจากแผนภูมิจะเห็นถึงสัดส่วนและปริมาณพลังงานที่โหนดตัวอย่างได้สูญเสียไปในแต่ละกิจกรรม ซึ่งจะเห็นได้ว่า พลังงานโดยมากจะหมดไปกับการใช้เพื่อตนเอง (Itself) และการใช้เพื่อดักจับข้อมูลที่ไม่สมารถจำแนกประเภทได้ (Unknown) โดยในกรณีที่โหนดนั้นมีการส่งและรับข้อมูลของตนเองจะใช้พลังงานในส่วนของ itself มาก และสาเหตุที่สูญเสียไปกับการดักฟังข้อมูลที่ไม่สามารถจำแนกประเภทไปเป็นปริมาฯมากนั้น เนื่องจากในเครือข่ายไร้สายนั้น จะใช้คลื่นในการส่งสัญญาณในอากาศ ดังนั้นทุกโหนดจะต้องพยายามดักจับข้อมูลทุกอย่างที่วิ่งไปมา เพราะไม่อาจเลือกดักจับได้ ทำให้สามารถเกิดกรณีที่มีการดักจับข้อมูลที่ปลายทางไม่ใช่ตนเอง หรือดักจับแล้วพบว่ากำลังสัญญาณไม่เพียงพอที่จะใส่ใจ เนื่องจากกำลังสัญญาณน้อยกว่า receive threshold ซึ่งทั้ง 2 กรณีสามารถเกิดได้มากที่สุดในเครือข่าย

ขณะที่จะใช้พลังงานเล็กน้อยในการใช้เพื่อในกิจกรรมสำหรับโปรโตคอล ARP และ DSR เนื่องจากจะมีการรับส่ง 2 โปรโตคอลนี้ก็ต่อเมื่อมีความต้องการที่จะส่งข้อมูลเท่านั้นหรือมีการร้องขอเท่านั้น และค่าที่ได้จากโปรโตคอลจะถูกบันทึกลงตาราง จากนั้นจะไม่มีการรับส่งโปรโตคอลดังกล่าวอีกหากค่าที่ต้องการยังอยู่ในตาราง และจะถูกแลกเปลี่ยนใหม่เมื่อไม่พบค่าในตารางที่ต้องการ

. การแลกเปลี่ยนข้อความควบคุมเป็นการแลกเปลี่ยนที่จำเป็นต้องมีตามมาตรฐาน 802.11 เพื่อให้การรับส่งข้อมูลสามารถดำเนินไปได้ และเป็นการหลีกเลี่ยงการเกิด Collision ซึ่งประกอบด้วย RTS, CTS และ ACK ซึ่งจะต้องส่งทุกครั้งเมื่อจะมีการรับส่งข้อมูล แต่ที่เห็นว่าปริมาณพลังงานที่ใช้ในส่วนนี้ค่อนข้างน้อย เนื่องจากข้อมูลประเภทนี้มีขนาดค่อนข้างเล็กนั่นเอง

เมื่อโหนดมีการใช้พลังงานเพื่อตนเอง (Itself) มากเท่าไหร่แต่ละโหนดก็จะมีโอกาสได้รับผลประโยชน์จากการสูญเสียพลังงานของผู้อื่น (Take) มากเท่านั้นซึ่งหากมีการส่งหรือรับข้อมูลของตนเอง แล้วคู่ของการติดต่อไม่สามารถสื่อสารกันได้โดยตรง จึงจะมีโหนดตัวกลางในการทำหน้าที่ส่งผ่านข้อมูล ซึ่งโหนดต้นทางและปลายทางจะได้ผลประโยชน์จากการทำกิจกรรมของโหนดตัวกลางนี้ โดยยิ่งมีการทำกิจกรรมเพื่อตนเองมาก ก็ยิ่งมีโอกาสที่จะได้รับผลประโยชน์จากการทำกิจกรรมของผู้อื่นมากขึ้นตามไปด้วย แต่ในการทดลองก็พบกรณีที่ไม่มีการสูญเสียพลังงานในส่วนของ Itself แต่กลับได้รับพลังงานในส่วนของ Take ซึ่งคือ โหนด 1 ของการทดลองขนาดพื้นที่ 1250*1250 ครั้งที่ 1 และ 3 และของโหนด 2 ของการทดลองขนาดพื้นที่ 1500*1500 ครั้งที่ 2 และ 8 โดยเมื่อตรวจสอบที่รูปแบบการเชื่อมต่อของการทดลองดังกล่าวพบว่า โหนดดังกล่าวเป็นปลายทางของการเชื่อมต่อเท่านั้น ไม่ได้เป็นผู้ส่งแต่อย่างใด ซึ่งพบว่าแพ็คเกตดังกล่าวถูกส่งออกจากต้นทางเพื่อจะส่งไปยังปลายทางและมีการส่งผ่านตัวกลาง โหนดต้นทางและปลายทางจึงได้รับ

Take จากการส่งผ่านครั้งนั้น แต่เนื่องจากขนาดพื้นที่ที่มาก แต่ละโหนดมีโอกาสที่จะอยู่ไกลเกินกว่าจะรับส่งกันได้ จึงทำให้ตัวกลางไม่สามารถส่งไปถึงปลายทางได้ ปลายทางจึงไม่เสียพลังงานในส่วนของ Itself ที่เกิดจากการรับข้อมูล

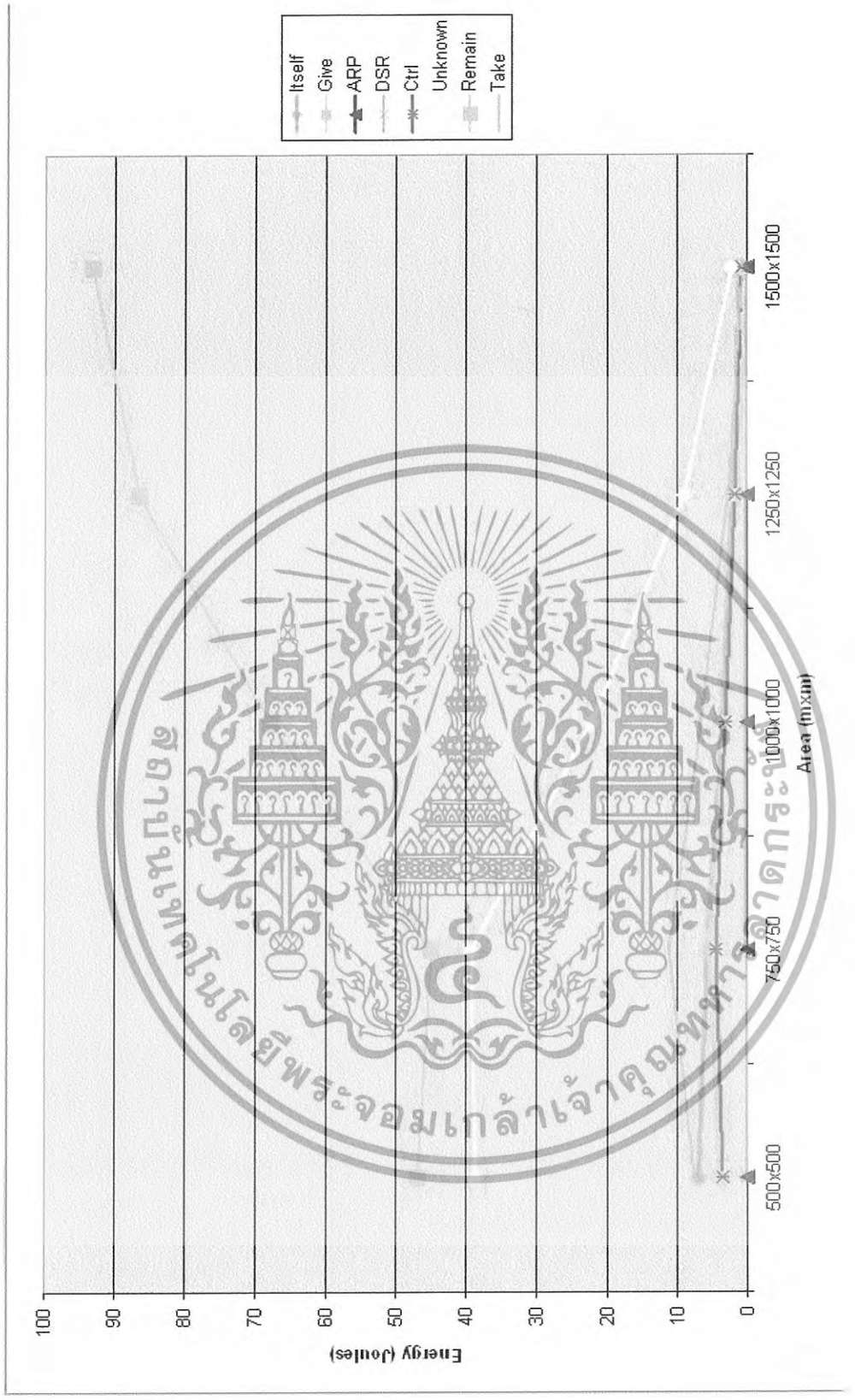
พบว่าโหนดที่มีการใช้พลังงานมากที่สุดในแต่ละขนาดพื้นที่ คือโหนดที่เป็นต้นทางและปลายทางของการเชื่อมต่อมากที่สุด โดยโหนดดังกล่าวจะต้องสูญเสียพลังงานในกิจกรรมที่ตนเองใช้ ซึ่งต้นทางและปลายทางจะถูกกำหนดโดยรูปแบบการเชื่อมต่อ ซึ่งเป็นหนึ่งในตัวแปรของการจำลองนี้

จากแผนภูมิวงกลมที่แสดงการสูญเสียพลังงานในแต่ละกิจกรรมนั้น เมื่อนำแต่ละส่วนมาบวกเข้าด้วยกัน จะได้เท่ากับ 100 ซึ่งก็คือพลังงานเริ่มต้นที่กำหนดให้แต่ละโหนดนั่นเอง

โดยหากนำข้อมูลที่ได้จากแผนภูมิวงกลมซึ่งเป็นพลังงานเฉลี่ยในแต่ละกิจกรรมของทุกโหนดในแต่ละขนาดพื้นที่มาทำการเปรียบเทียบกันจะได้ผลลัพธ์ดังนี้

ตารางที่ 6.57 แสดงค่าพลังงานเฉลี่ยของแต่ละกิจกรรม ณ. ทุกพื้นที่

Area/Activity	Itself	Give	ARP	DSR	Ctrl	Unknown	Remain	Take
500x500	6.97855	3.852375	0.007305	0.064608	3.599902	38.65746	46.8398	7.70475
750x750	5.611037	5.556548	0.012307	0.191375	4.418821	39.2765	44.93446	11.1131
1000x1000	3.526275	3.303293	0.011365	0.191863	3.040886	22.14139	67.78493	6.606585
1250x1250	2.11517	1.342817	0.011999	0.150873	1.781526	8.954484	86.50355	2.685633
1500x1500	0.961979	0.277378	0.00236	0.075093	0.821887	2.093871	93.0962	0.554756



รูปที่ 6.12 เปรียบเทียบพลังงานแต่ละชนิดและรวมของทุกโหนดในแต่ละขนาดพื้นที่

ซึ่งจากแผนภูมิจะพบว่าขนาดพื้นที่ (area) ยิ่งมาก จะส่งผลให้พลังงานที่เหลืออยู่ (Remain) มากตามไปด้วย เนื่องจากว่าขนาดที่กว้างขึ้นทำให้แต่ละโหนดมีขอบเขตในการเคลื่อนที่ได้มากขึ้น แต่ขณะที่พื้นที่ที่กำลังสัญญาณไปถึงจะยังเท่าเดิม ทำให้มีโอกาสที่ผู้รับจะสามารถดักจับข้อมูลได้ก็จะลดลงตามไปด้วย ซึ่งถึงแม้ว่าจะมีการส่งข้อมูลแต่ละชนิดเท่าเดิมก็ตาม

ขนาดพื้นที่ (area) ยิ่งมาก จะส่งผลให้การใช้พลังงานในจุดประสงค์ต่างๆ ลดลง เนื่องจากพลังงานที่เสียไปจะเกิดจากกิจกรรมส่งและดักจับ แต่เมื่อพื้นที่ที่มากเกินระดับหนึ่ง แต่ละโหนดจะมีโอกาสอยู่นอกพื้นที่สัญญาณทั้ง Receive threshold และ carrier sense threshold ของกันและกันมากขึ้น ทำให้มีโอกาสที่ผู้รับจะสามารถดักจับข้อมูลได้ก็จะลดลงตามไปด้วย

ขนาดพื้นที่ที่มากขึ้นจะทำให้แต่ละโหนดมีการสูญเสียพลังงานในส่วน Itself น้อยลง เนื่องจากว่า Itself เกิดจากการรับและส่งข้อมูลของตน แต่พื้นที่ที่เปลี่ยนแปลงไป การส่งข้อมูลอาจจะเท่าเดิม หรืออาจจะลดลงเนื่องจากไม่สามารถหาเส้นทางไปถึงปลายทางได้ และ itself ในส่วนของการรับก็จะลดลง เพราะข้อมูลไม่สามารถไปถึงปลายทางได้ หรือในอีกกรณีคือต้นทางไม่สามารถหาเส้นทางไปยังปลายทางได้ ก็จะไม่ทำการส่งออกไป

โดยสังเกตได้ว่าที่ขนาดพื้นที่ 1500*1500 นั้น จะไม่มีโหนดใดที่มี Give มากกว่า Itself เลย ทั้งนี้เนื่องจากขนาดพื้นที่ดังกล่าวแต่ละโหนดได้มีการใช้พลังงานเพื่อผู้อื่นน้อยมาก เพราะว่าไม่สามารถเป็นตัวกลางในการส่งข้อมูลไปยังปลายทางได้ เนื่องด้วยระยะทางระหว่างโหนดมากนั่นเอง

โดยเราอาจจะสรุปได้ว่าพื้นที่ยิ่งมากขึ้น จะยิ่งทำให้แต่ละโหนดสามารถติดต่อกันได้น้อยลง เนื่องจากระยะทางระหว่างต้นทางและปลายางจะไกลขึ้น และด้วยระยะทางที่มากขึ้นนี้เองทำให้ในการส่งข้อมูลจำเป็นต้องมีการใช้ตัวกลางมากยิ่งขึ้น ซึ่งจากการผลการทดลอง สามารถนำมาวเคราหีเพื่อหาจำนวนโหนดที่ใช้ตลอดเส้นทางจากต้นทางไปยังปลายทางได้ โดยหากติดต่อกันได้โดยตรงจะถือว่าใช้ 1 โหนดในการไปถึงปลายทาง แต่สมมติให้มีตัวกลาง 1 ตัวจะต้องใช้ 2 โหนดในการไปถึงปลายทาง

ในไอพีแพ็คเกตได้มีการกำหนดค่า Time To live หรือ TTL ซึ่งจะเป็นค่าที่ถูกกำหนดจากต้นทางและจะลดลง 1 เมื่อมีการผ่านโหนด 1 โหนด ซึ่งเราสามารถนำค่า TTL มาพิจารณาจากเทรซไฟล์ได้ ซึ่งใน NS-2 จะกำหนดค่า TTL ไว้คือ 32 นั่นหมายถึงหากแพ็คเกตผ่านโหนด 32 โหนดค่า TTL จะเหลือ 0 ซึ่งเราสามารถนำค่า TTL จากแพ็คเกตที่ปลายทางได้รับมาใช้ในการวิเคราะห์หาจำนวนโหนดได้

ตารางที่ 6.58 ตารางแสดงถึงจำนวนแพ็คเกตเฉลี่ยสำหรับแต่ละจำนวนโหนดที่ใช้ในการไปถึงปลายทาง

Hop/Area	500x500	750x750	1000x1000	1250x1250	1500x1500
1	15145	10007.9	7183.5	5343.2	3413.8
2	7538.6	7275.9	3798.5	1892.5	579.1
3	1702.4	3903.6	2226.8	780.4	181.8
4	412.7	1511.3	980.5	264.6	16.3
5	140.2	393.7	293.6	116.2	0.2
6	31.7	52.1	36.1	8.4	0.2
7	14.1	28.2	12.4	0.9	0.2
8	8.8	16.7	7.8	1	0.1
9	6.3	6.3	4	0.4	0
10	6.1	5.8	1.1	0.4	0
11	4.8	1.5	0.6	0.3	0
12	3.2	2.2	0.3	0.1	0
13	2.6	0.9	0	0.1	0
14	3.1	2.4	0.2	0.1	0
15	1.4	0.2	0.1	0.1	0
16	2.2	0.2	0	0	0
17	0.5	0	0	0	0
18	0.8	0.1	0.1	0	0
19	0.5	0	0	0	0
20	0.2	0	0	0	0
21	0.1	0	0	0	0
22	0.1	0	0	0	0
23	0.1	0	0	0	0
24	0	0	0	0	0
25	0.1	0	0	0	0
26	0	0	0	0	0
27	0.1	0	0	0	0
28	0	0	0	0	0

ตารางที่ 6.58 ตารางแสดงถึงจำนวนแพ็คเกตเฉลี่ยสำหรับแต่ละจำนวนโหนดที่ใช้ในการไปถึงปลายทาง (ต่อ)

29	0	0	0	0	0
30	0	0	0	0	0
31	0	0	0	0	0
32	0	0	0	0	0

รูปที่ 6.13 กราฟแสดงจำนวนแพ็คเกตเฉลี่ยสำหรับแต่ละจำนวนโหนดที่ใช้ในการวิเคราะห์

ซึ่งจากแผนภูมิจะสังเกตได้ว่าในพื้นที่ที่เล็กที่สุดในการทดลอง (500*500) นั้น จะมีการรับแพ็คเกตมากที่สุด เนื่องจากพื้นที่ที่เล็ก ทำให้มีโอกาสที่ข้อมูลจะถูกส่งไปถึงปลายทางมากขึ้น ขณะเมื่อพื้นที่ที่ใหญ่ขึ้นจะสามารถรับแพ็คเกตได้ลดลง และจำเป็นต้องใช้จำนวนโหนดมากขึ้นในการไปให้ถึงปลายทาง โดยสังเกตจากจะมีปริมาณแพ็คเกตที่ใช้จำนวนโหนดมาก มากขึ้น โดยพื้นที่ที่มีการใช้ตัวกลางแล้วยังสามารถไปถึงปลายทางได้ดีคือ 750*750 และเมื่อพื้นที่มากกว่านี้จะเริ่มรับแพ็คเกตได้น้อยลงถึงแม้จะใช้จำนวนโหนดในการไปถึงปลายทาง (Hop) มากขึ้นก็ตาม

ซึ่งสามารถสรุปได้ว่า พื้นที่ที่เล็กจะทำให้แต่ละโหนดสามารถรับส่งข้อมูลกันได้มากกว่าพื้นที่ที่มาก พื้นที่ยิ่งเล็กก็จะสามารถส่งกันโดยตรงได้มากขึ้น คือใช้ 1 โหนด และเมื่อพื้นที่มากขึ้น ก็จะมีความจำเป็นจะต้องใช้โหนดระหว่างทางมากขึ้นตามไปด้วย

แต่เมื่อพื้นที่มากเกินไปข้อมูลจะไม่สามารถส่งไปถึงปลายทางได้มากนัก เนื่องจากระยะทางระหว่างโหนดค่อนข้างไกล และไม่สามารถหาเส้นทางไปถึงได้

ซึ่งจากผลการทดลองดังกล่าว เราสามารถนำมาวิเคราะห์ความคุ้มค่าหรือหาความยุติธรรมในการใช้พลังงานได้ โดยพิจารณาปริมาณการสูญเสียพลังงานในกิจกรรมที่ทำเพื่อผู้อื่น เปรียบเทียบกับพลังงานที่สูญเสียจากการทำกิจกรรมเพื่อตนเอง โดยแต่ละโหนดมีจุดประสงค์หลักคือต้องการใช้พลังงานเพื่อกิจกรรมของตนเอง โดยหากเสียพลังงานให้ผู้อื่นโดยมากเกินกว่าที่จะใช้พลังงานเพื่อกิจกรรมของตนเองก็ถือว่าเป็นเรื่องที่ไม่ควร

ในการทดลองจะเปรียบเทียบจำนวนโหนดที่มีปริมาณพลังงานที่สูญเสียการทำกิจกรรมเพื่อผู้อื่น (Give) มากกว่าที่สูญเสียจากการทำกิจกรรมเพื่อตนเอง (Itself) สำหรับแต่ละขนาดพื้นที่ โดยจะพิจารณาเฉพาะโหนดที่มีค่า Itself ไม่เป็น 0 เท่านั้น โดยจะเปรียบเทียบในแต่ละขนาดพื้นที่

โดยแต่ละจุดของแผนภูมิจะแทนจำนวนโหนดที่มีการสูญเสียพลังงานในส่วนของ Give มากกว่า Itself และ Itself ไม่เท่ากับ 0 ซึ่งคำนวณจากสมการต่อไปนี้

จำนวนโหนดเฉลี่ย = จำนวนโหนดที่มี Give มากกว่า Itself และ Itself ไม่เท่ากับ 0 / จำนวนโหนดที่ Itself ไม่เท่ากับ0*100

จากนั้นจะทำการทดลองเพื่อหาปริมาณโหนดที่มีการใช้พลังงานเพื่อผู้อื่น (Give) มากกว่าที่ได้รับจากผู้อื่น (Take) เนื่องจากมองว่าการที่โหนดใดๆยอมสูญเสียพลังงานของตนเองเพื่อผู้อื่นนั้นย่อมต้องการให้ผู้อื่นเสียพลังงานเพื่อตนเองด้วยเช่นเดียวกัน แต่ในการทดลองจะพิจารณาเฉพาะโหนดที่มีการเสียพลังงานเพื่อตนเองด้วย (Itself) เนื่องจากให้ไม่มีการรับหรือส่งเพื่อตนเองนั้น จะไม่มีทางได้รับผลประโยชน์ที่ผู้อื่นทำให้เลย โดยโหนดที่มีการส่งข้อมูลและมีการใช้พลังงานของตนเพื่อผู้อื่น ก็ย่อมต้องการความยุติธรรมในการที่จะได้รับการใช้พลังงานของผู้อื่นเพื่อตนเองเช่นเดียวกัน

โดยโหนดที่ใช้พลังงานในการทำกิจกรรมเพื่อผู้อื่นมากกว่าที่ตนเองได้รับจากผู้อื่นนั้น ถือว่าเป็นความไม่ยุติธรรมที่เกิดขึ้น หรือจะมองได้ว่าโหนดดังกล่าวมีโอกาสที่จะพลังงานหมดก่อนที่จะทำกิจกรรมเพื่อตนเองตามที่ต้องการได้ เนื่องจากหมดไปกับการสูญเสียเพื่อผู้อื่น

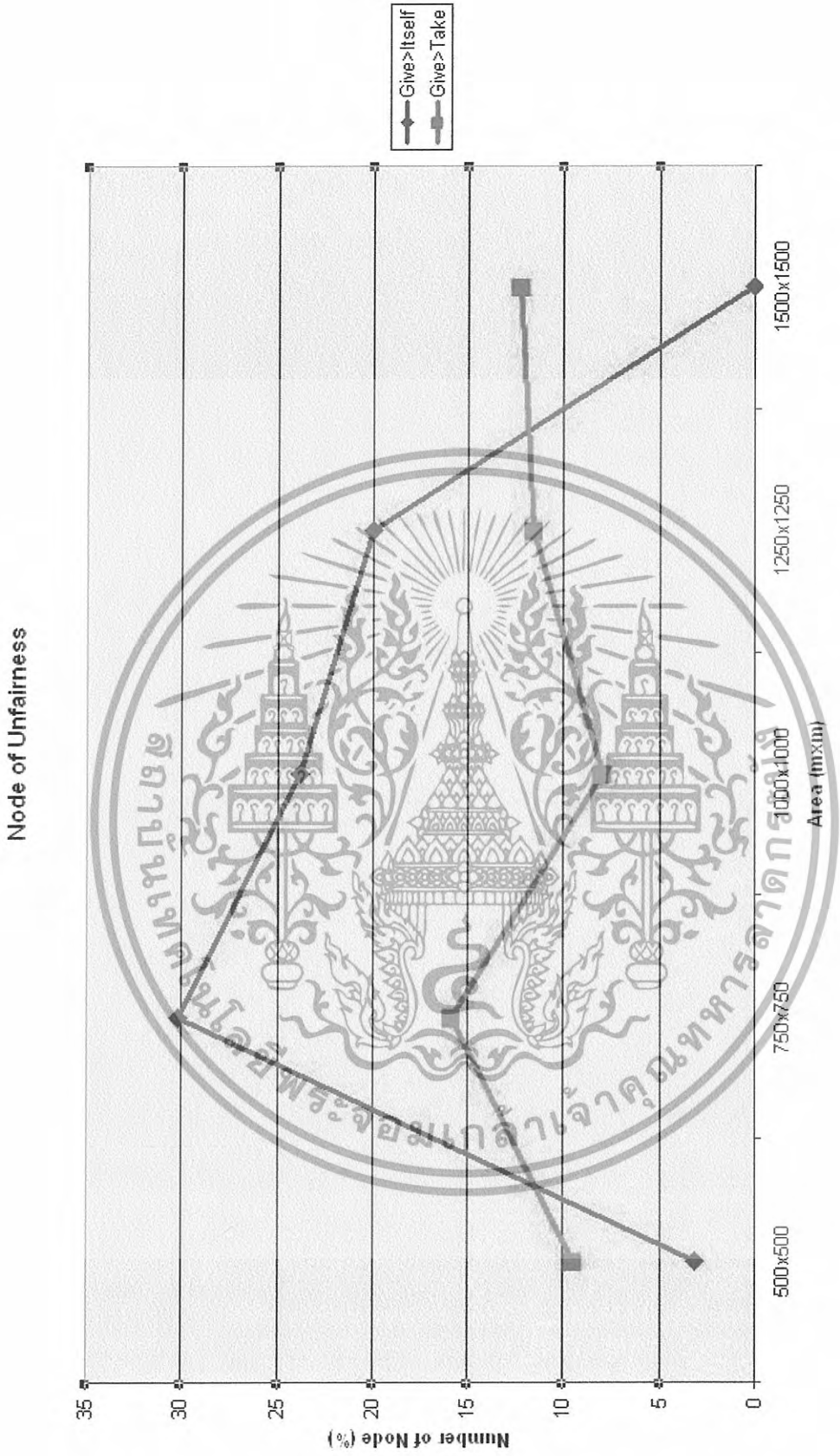
โดยแต่ละจุดบนแผนภูมิสามารถหาได้จากสมการ

จำนวนโหนดเฉลี่ย = จำนวนโหนดที่มี Give มากกว่า Take และ Itself ไม่เท่ากับ 0 / จำนวนโหนดที่ Itself ไม่เท่ากับ0*100

ซึ่งสามารถนำเสนอในรูปของแผนภูมิเส้นได้ดังนี้

ตารางที่ 6.59 ตารางแสดงความไม่ยุติธรรมในการส่งข้อมูล

Number of Node/Area	500x500	750x750	1000x1000	1250x1250	1500x1500
Give > Itself	3.1746	30.1587	23.809	20	0
Give > Take	9.5238	15.873	7.9365	11.5942	12.2807



รูปที่ 6.14 กราฟแสดงความไม่ยุติธรรมในการส่งข้อมูล

จากแผนภูมินั้น เส้นจำนวนโหนดที่ Give มากกว่า Itself จะแสดงให้เห็นถึงอัตราการเกิดโหนดที่มีการสูญเสียพลังงานเพื่อผู้อื่นมากกว่าที่สูญเสียเพื่อกิจกรรมของตนเอง ซึ่งจะอยู่ในรูปของร้อยละของจำนวนโหนดทั้งหมด ซึ่งจะเห็นได้ว่า ในช่วงแรกปริมาณโหนดที่ใช้พลังงานในการให้ผู้อื่นมากกว่าเพื่อตนเองนั้นจะเพิ่มขึ้นตามขนาดพื้นที่ที่เพิ่มขึ้น แต่เมื่อถึงจุดหนึ่งพลังงานที่ใช้ในการทำเพื่อผู้อื่นจะลดลงผกผันกับขนาดพื้นที่ที่เพิ่มขึ้น

เนื่องจากช่วงแรกนั้น ขนาดพื้นที่ที่เล็กมีความเป็นไปได้ว่าโหนดต้นทางและปลายทาง จะมีโอกาสที่จะอยู่ในพื้นที่การให้สัญญาณของกันและกันสูง ทำให้สามารถส่งข้อมูลได้โดยไม่มีความจำเป็นที่จะต้องผ่านโหนดตัวกลาง เมื่อพื้นที่เพิ่มขึ้นโอกาสที่โหนดต้นทางและปลายทางจะอยู่ในพื้นที่การให้สัญญาณของกันและกันก็จะลดน้อยลงไปด้วย ก็จะมีโอกาสที่จำเป็นต้องใช้โหนดตัวกลางเพื่อทำหน้าที่ส่งผ่านข้อมูลไปยังปลายทางมากขึ้นตามไปด้วย

และเมื่อเลยจุดหนึ่งปริมาณโหนดที่ใช้พลังงานเพื่อผู้อื่นมากกว่าใช้เพื่อตัวเองจะลดลง เนื่องจากเป็นไปได้ว่า ขนาดพื้นที่ที่เพิ่มมากขึ้น ทำให้แต่ละโหนดอยู่นอกพื้นที่กันและกันมากขึ้น และไม่สามารถหาเส้นทางใดๆไปหาโหนดปลายทางได้เลย จึงไม่สามารถทำการส่งข้อมูลได้ จึงทำให้แต่ละโหนดใช้พลังงานทำงานเพื่อผู้อื่นน้อยลง เนื่องจากรับมาแล้วไม่สามารถหาเส้นทางส่งต่อไปยังปลายทางได้นั่นเอง

ซึ่งจุดที่เกิดการเปลี่ยนแปลงคือที่ขนาดพื้นที่ 750*750 โดยจะเป็นจุดที่มีการใช้พลังงานในการ Give มากที่สุด เนื่องจากระยะทางมากพอที่จะต้องใช้โหนดตัวกลาง และไม่มากเกินที่จะทำให้ข้อมูลไปไม่ถึงปลายทาง

และจากแผนภูมิ นั้นเส้นจำนวนโหนดที่ Give มากกว่า Take ที่แสดงจะพบว่าทุกขนาดพื้นที่ จะมีโหนดที่เสียเปรียบในเรื่องความยุติธรรมคือโหนดที่ใช้พลังงานเพื่อผู้อื่นมากกว่าที่ได้รับ อยู่ในระดับร้อยละ 12 ซึ่งจะอยู่ในระดับนี้ทุกพื้นที่ ซึ่งจะต่างไม่เกินร้อยละ 5 ซึ่งเป็นอัตราที่มากพอที่เราควรจะนำพิจารณา เนื่องจาก โหนดเหล่านี้มีโอกาสที่จะพลังงานหมดก่อนที่จะใช้งานเพื่อความต้องการของตนเองได้สูง เนื่องจากทำให้ผู้อื่นมากเกินกว่าที่ได้รับ ดังนั้นเป็นไปได้ว่าโหนดเหล่านี้อาจจะไม่สามารถทำงานต่อได้ นั่นหมายถึงในเครือข่ายไร้สายเฉพาะกิจนั้นจะมีโหนดที่เกิดความไม่ยุติธรรมถึงประมาณร้อยละ 12

ซึ่งอัตราส่วนดังกล่าวเป็นปริมาณที่มากพอที่เราจะบอกได้ว่า ควรนำรูปแบบและลักษณะการใช้พลังงานมาเป็นปัจจัยในการออกแบบหรือศึกษาการทำงานของเครือข่ายเฉพาะกิจด้วย

และเมื่อเปรียบเทียบกันทั้ง 2 เส้น จะพบว่าในขนาดพื้นที่ 750*750 นั้น จะมีการใช้โหนดตัวกลางมากที่สุดและมีประสิทธิภาพมากที่สุด เนื่องจากมีส่วนต่างระหว่าง 2 เส้นมากหมายถึง มี Itself น้อยกว่า Take มาก หมายความว่า ในการที่เราจะส่งไปถึงปลายทางจะมีผู้ที่สูญเสียพลังงานเพื่อตัวเองมาก ซึ่งถือว่าคุ้มค่าในการที่จะส่งหรือยอมเสียพลังงานเพื่อผู้อื่น เนื่องจากขนาดใหญ่เกินกว่าที่จะส่งโดยตรงและเล็กเกินกว่าที่จะเกินระยะสัญญาณ

ซึ่งในส่วนของพื้นที่ขนาดใหญ่มากๆจะสังเกตได้ว่า Itself จะมีค่ามากกว่ากว่า Take เนื่องจากเส้น Give มากกว่า Itself นั้นอยู่ต่ำกว่าเส้น Give มากกว่า Take เมื่อต้นทางใช้พลังงาน Itself ในการส่งข้อมูลไปแล้ว โหนดระหว่างทางไม่อาจส่งต่อไปให้ถึงปลายทางได้ แต่ Itself ของขนาดพื้นที่ดังกล่าวก็ยังน้อยกว่าของขนาดอื่นอยู่ดี

โดยเมื่อทราบจำนวนโหนดที่ Give มากกว่า itself และจำนวนโหนดที่ Give มากกว่า Take แล้ว ในส่วนนี้จะพิจารณาว่า ที่มากกว่านั้น มากกว่าเท่าไหร่ โดยจะแสดงถึงการกระจายและระดับพลังงานที่ Give มากกว่า Itself และ Give มากกว่า Take ที่ระดับความแตกต่างของพลังงานที่ต่างกัน

ซึ่งจะแบ่งระดับของความแตกต่างเป็น 6 ช่วง 0%-20%, 20%-50%, 50%-100%, 100%-200%, 200%-500% และ 500% ขึ้นไป โดยเป็นอัตราส่วนร้อยละระหว่าง Give กับ Itself หรือ Take

$$Dgi = (G/I)*100 \text{ และ } Dgt = (G/T)*100$$

โดยDgi คือระดับพลังงานนั้นๆ ที่ Give มากกว่า Itself

Dgt คือระดับพลังงานนั้นๆ ที่ Give มากกว่า Take

G คือพลังงาน Give

I คือพลังงาน Itself

T คือพลังงาน Take

โดยแผนภูมิและตารางต่อไปนี้ จะแสดงจำนวนโหนดที่ Give มากกว่า Itself ที่ระดับความแตกต่างที่ต่างกัน โดยจำนวนโหนดที่แสดงจะอยู่ในรูปแบบของร้อยละจากจำนวนโหนดทั้งหมดที่ Give มากกว่า Itself และ Itself มีค่ามากกว่า 0 ซึ่งจะคำนวณได้จากสมการดังนี้

$$Xgi = Ygi/Ngi*100$$

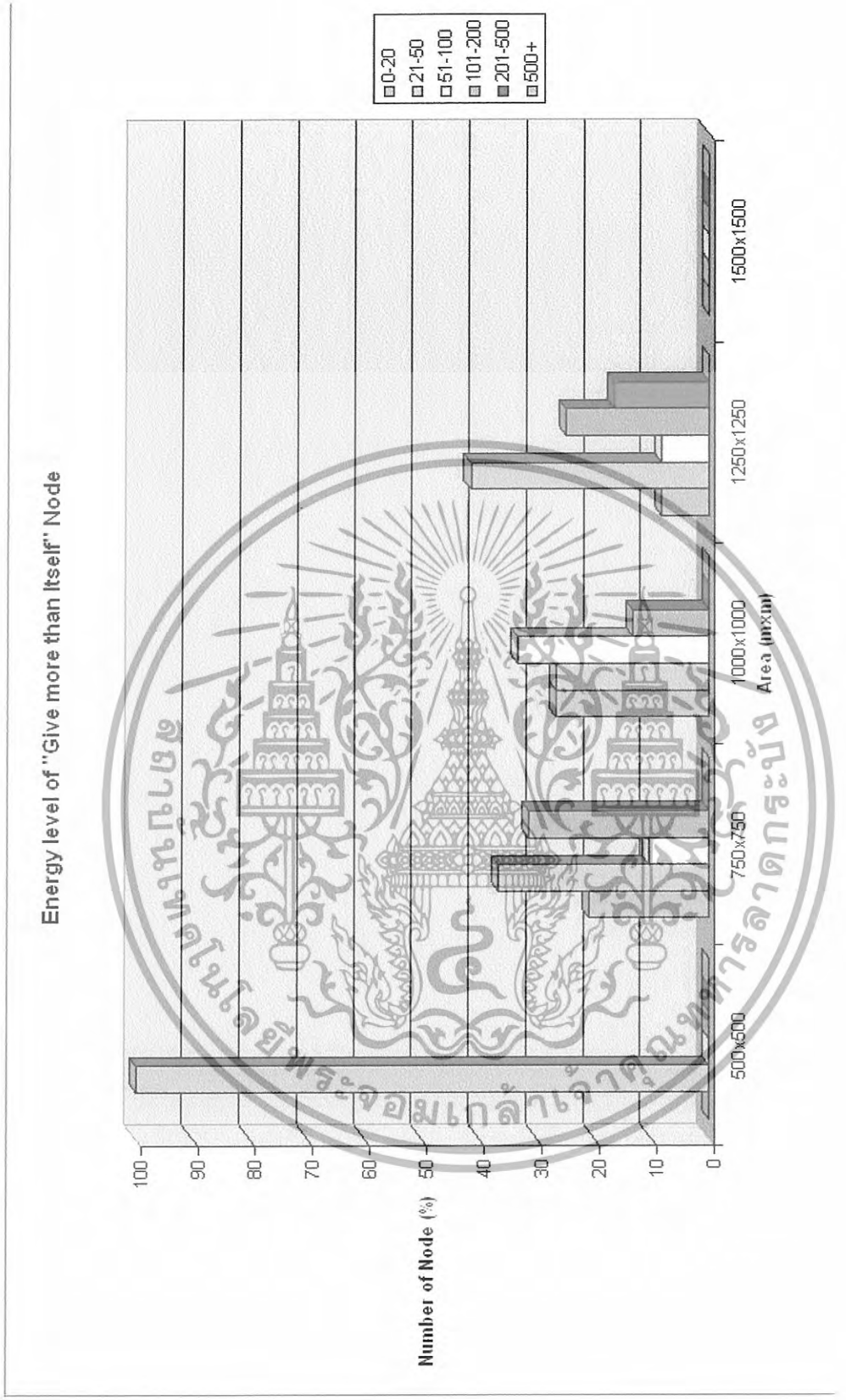
โดย Xgi คืออัตราการเกิดโหนดที่ Give มากกว่า Itself ที่ระดับพลังงานนั้นๆ

Ygi คือจำนวนโหนดที่ Give มากกว่า Itself ที่ระดับนั้นๆ และ Itself มากกว่า 0

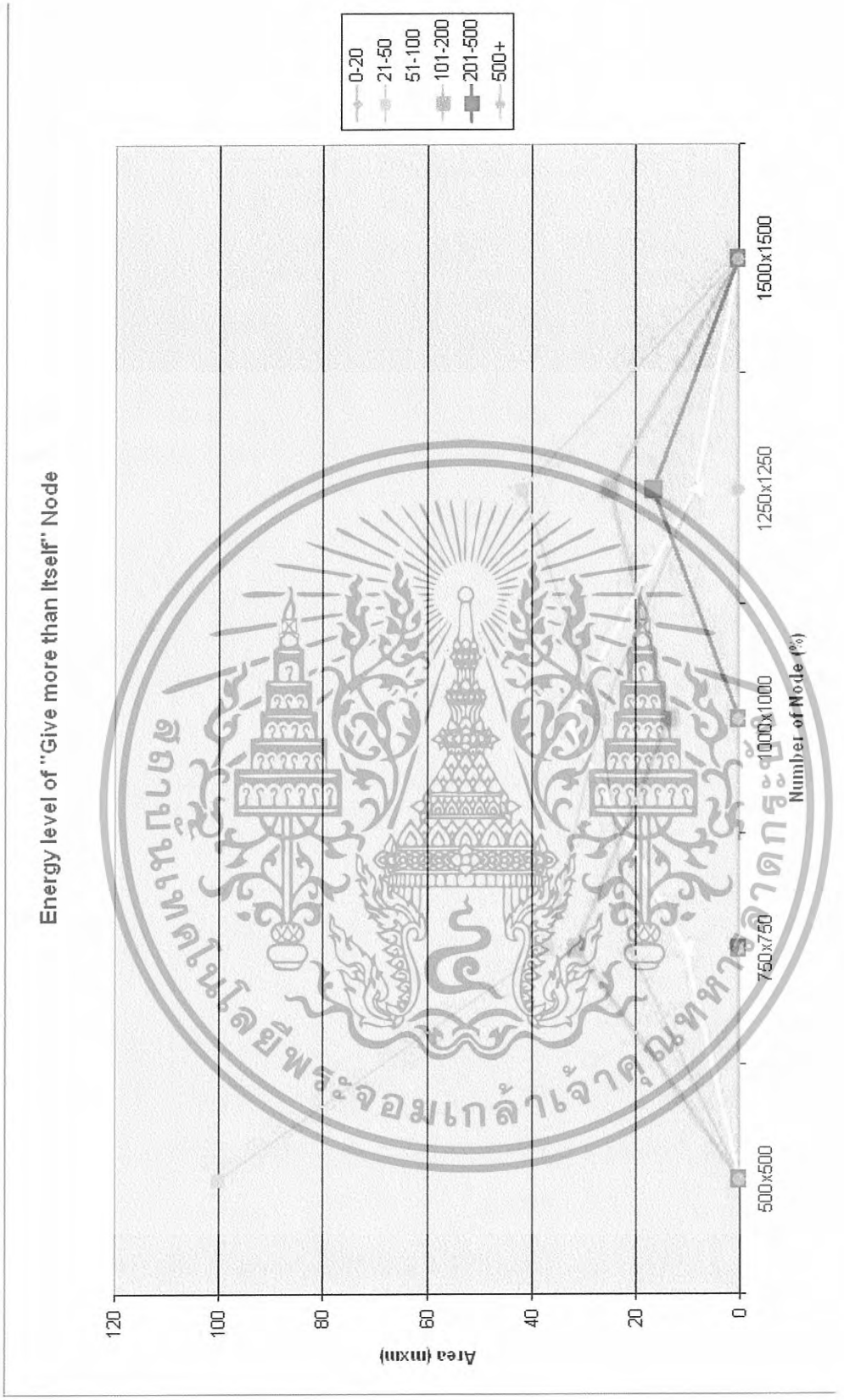
Ngi คือจำนวนโหนดทั้งหมดที่ Give มากกว่า Itself และ itself มีค่ามากกว่า 0

ตารางที่ 6.60 แสดงระดับพลังงานที่แตกต่างกันของ Give และ Itself

ระดับพลังงาน/พื้นที่	500x500	750x750	1000x1000	1250x1250	1500x1500
0-20	0	21.052	26.67	8.33	0
21-50	100	36.842	26.67	41.67	0
51-100	0	10.526	33.33	8.33	0
101-200	0	31.578	13.33	25	0
201-500	0	0	0	16.67	0
500+	0	0	0	0	0



รูปที่ 6.15 แสดงระดับพลังงานที่มากกว่าตัวเอง Give More Itself



รูปที่ 6.16 แสดงระดับพลังงานที่เหลือกรณี Give more than Itself

จากแผนภูมิจะพบว่าในขนาดพื้นที่ที่เล็กกว่าจะมีการใช้พลังงาน Give มากกว่า Itself เพียงเล็กน้อยโดยจะรวมอยู่ที่ระดับเดียว แต่เมื่อขนาดพื้นที่ที่เพิ่มมากขึ้นจะเริ่มมีการกระจายระดับการสูญเสียนี้ เนื่องด้วยขนาดพื้นที่ที่เล็กนั้นจะมีการกระจายกันทำหน้าที่เป็นตัวกลางใกล้เคียงกัน เพราะเคลื่อนที่อย่างไรก็ยังอยู่ในขอบเขตกันและกัน แต่เมื่อพื้นที่มากขึ้น จะเริ่มมีโหนดที่ทำหน้าที่นี้และไม่ทำจะเริ่มเกิดความเหลื่อมล้ำระหว่างกัน ในหมู่โหนดที่อาจจะพูดได้ว่าไม่เกิดความยุติธรรมนี้

ขณะที่ตารางและแผนภูมิต่อไปนี้จะแสดงระดับพลังงานที่ Give มากกว่า Take จะแสดงจำนวนโหนดที่ในรูปแบบของร้อยละจากจำนวนโหนดทั้งหมดที่ Give มากกว่า Take และ Itself มีค่ามากกว่า 0 ซึ่งจะคำนวณได้จากสมการดังนี้

$$Xgt = Ygt/Ngt*100$$

โดย Xgt คืออัตราการเกิดโหนดที่ Give มากกว่า Take ที่ระดับพลังงานนั้นๆ

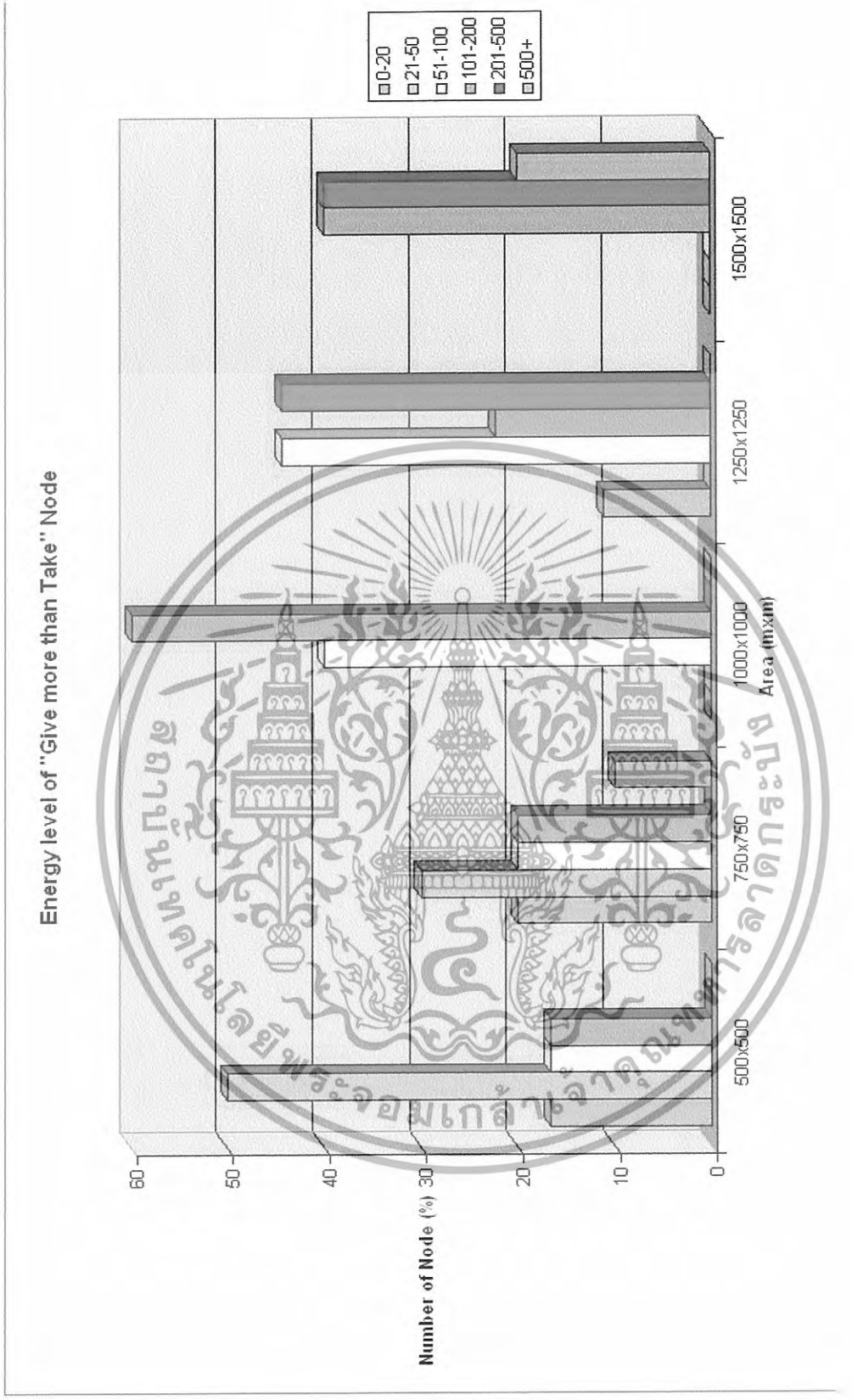
Ygt คือจำนวนโหนดที่มี Give มากกว่า Take ที่ระดับพลังงานนั้นๆ และ Itself มีค่ามากกว่า 0

Ngt คือจำนวนโหนดทั้งหมดที่ Give มากกว่า Take และ Itself มีค่ามากกว่า 0

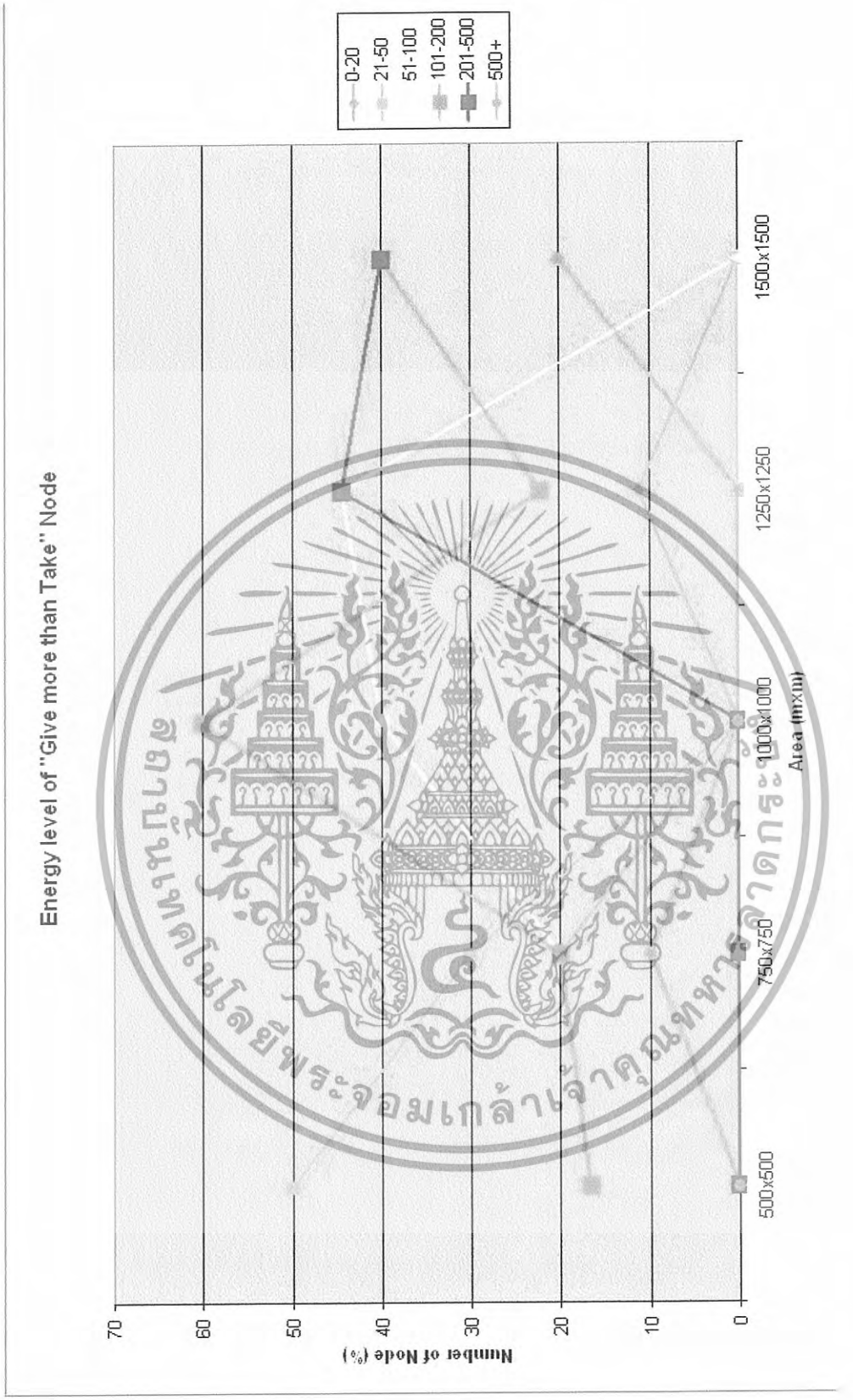
ซึ่งจะจากการวิเคราะห์และคำนวณดังกล่าวสามารถแสดงผลได้ในรูปของตาราง, แผนภูมิแท่ง และแผนภูมิเส้น ในการเปรียบเทียบระดับความแตกต่างที่ระดับต่างๆ ในขนาดพื้นที่ที่แตกต่างกัน

ตารางที่ 6.61 แสดงระดับพลังงานที่แตกต่างกันของ Give และ Take

ระดับพลังงาน/พื้นที่	500x500	750x750	1000x1000	1250x1250	1500x1500
0-20	16.666	20	0	11.11	0
21-50	50	30	0	0	0
51-100	16.666	20	40	44.44	0
101-200	16.666	20	60	22.22	40
201-500	0	0	0	44.44	40
500+	0	10	0	0	20



รูปที่ 6.17 แสดงระดับพลังงานที่มีค่ามากกว่าของ Give และ Take



รูปที่ 6.18 แสดงระดับพลังงานที่เหลือของ Give และ Take

โดยจากแผนภูมิดังกล่าวจะแสดงให้เห็นถึงระดับปริมาณของพลังงานที่ Give มากกว่า Take โดยโหนดที่มี Give มากกว่า Take อาจหมายถึงโหนดที่เกิดความยุตอธรรมขึ้น โดยหากสูญเสียในระดับพลังงานที่สูง ย่อมหมายถึงเกิดความไม่ยุติธรรมมาก โดยจากแผนภูมิจะสังเกตได้ว่าขนาดพื้นที่เล็กแต่ละโหนดจะมีระดับการเสียเปรียบกระจายกันและค่อนข้างน้อย ขณะที่พื้นที่ที่มากขึ้นแต่ละโหนดจะเสียเปรียบมาก เนื่องจากความแตกต่างระหว่าง Give และ Take มีมากนั่นเอง โดยจะสังเกตว่า ในพื้นที่ที่เล็กจะมีการกระจายระดับพลังงานในระดับที่ความแตกต่างน้อย แต่ในขณะพื้นที่มากจะมีการกระจายระดับพลังงานในช่วงที่มีความแตกต่างสูง เนื่องจากขนาดพื้นที่ที่มาก จะมีเพียงบางโหนดที่อยู่ในตำแหน่งที่เหมาะจะเป็นตัวกลาง ซึ่งอาจจะอยู่บริเวณกลางๆของพื้นที่ ซึ่งจะถูกทำหน้าที่เป็นตัวกลางของโหนดอื่นอย่างหนักเนื่องจากมีตัวเลือกไม่มากนั่นเอง

6.5 สรุปผลการทดลอง

จากการทดลองทั้งหมดจะสามารถสรุปผลได้ว่า โหนดแต่ละโหนดใน MANET จะมีการสูญเสียพลังงานในแต่ละกิจกรรมแตกต่างกันไป โดยกิจกรรมในการส่งข้อมูลจากโหนดหนึ่งไปยังโหนดหนึ่ง อาจทำให้ต้องมีการสูญเสียพลังงานจากผู้อื่นร่วมด้วย ในการที่จะทำให้การสื่อสารสามารถดำเนินไปได้ โดยเฉพาะอย่างยิ่งอุปกรณ์ไร้สายจะมีข้อจำกัดในเรื่องของพลังงาน ดังนั้นพลังงานทุกจูลที่สูญเสียควรเป็นไปอย่างมีประโยชน์และคุ้มค่ามากที่สุด และควรมองในเรื่องของความยุติธรรมเนื่องจากมีโหนดที่เสียผลประโยชน์ก็จะเกิดโหนดที่ได้ผลประโยชน์เช่นเดียวกัน ซึ่งจากผลการทดลองพบว่า มีโหนดที่เกิดความไม่ยุติธรรมมากในระดับหนึ่ง มากพอที่เราควรจะนำเรื่องความยุติธรรมในการใช้พลังงานมาเป็นอีกหนึ่งปัจจัยที่ควรตระหนักในการพิจารณาในการศึกษาและออกแบบรวมถึงพัฒนาเครือข่ายไร้สายเฉพาะกิจนี้หรือ MANET ต่อไป

บทที่ 7
สรุปผลการวิจัย และข้อเสนอแนะ

7.1 สรุปผลการวิจัย และข้อเสนอแนะ

ในปัจจุบันซึ่งเป็นยุคแห่งเทคโนโลยีที่การสื่อสารเป็นปัจจัยสำคัญในการที่จะแข่งขันในการเป็นผู้นำทางการตลาด โดยปัจจุบันธุรกิจต่างๆต่างปรับให้มีการติดต่อสื่อสารกันได้อย่างรวดเร็ว ปลอดภัยและมีประสิทธิภาพ ปัจจัยสำคัญอยู่ที่เครือข่ายที่ให้บริการ ด้วยเทคโนโลยีต่างๆที่พัฒนาอยู่ตลอดเวลา และประสิทธิภาพจของอุปกรณ์ที่เพิ่มขึ้นอย่างต่อเนื่อง ทำให้องค์กรต่างๆจำเป็นต้องปรับตัวเพื่อที่จะสามารถใช้เทคโนโลยีที่มีอยู่อย่างมีประสิทธิภาพสูงสุด โดยจำเป็นต้องมีการวิเคราะห์และออกแบบเครือข่ายอย่างเหมาะสม

การจำลองระบบเครือข่ายเป็นอีกวิธีหนึ่งที่สามารถนำมาเป็นเครื่องมือที่ใช้ในการวิเคราะห์ระบบเครือข่าย ซึ่งข้อดีของเครื่องมือนี้มีจุดประสงค์หลักอยู่ที่การที่สามารถทดสอบระบบเพื่อวิเคราะห์เครือข่ายได้โดยไม่ต้องทำการติดตั้งระบบจริงซึ่งเป็นการสิ้นเปลืองและลำบากในการติดตั้ง แต่ทั้งนี้ประสิทธิภาพของการวิเคราะห์ขึ้นอยู่กับความสามารถของแบบจำลอง โดยสมควรที่จะสามารถจำลองแทนเทคโนโลยี อุปกรณ์ และเครือข่ายจริงได้ โดยพยายามให้มีความใกล้เคียงของจริงมากที่สุด ซึ่ง NS-2 เป็นเครื่องมือที่มีความสามารถดังกล่าวค่อนข้างครบถ้วนและเป็นหนึ่งในตัวเลือกที่ดีที่สุดในการจำลองระบบเครือข่าย

ทั้งด้วยข้อดีในเรื่องของเป็นเครื่องมือที่สามารถใช้ได้ฟรี และมีความยืดหยุ่นค่อนข้างมาก อีกทั้งยังมีแบบจำลองหรือโครงร่างที่กำหนดมาให้ค่อนข้างหลากหลายและครบถ้วนตามที่ต้องการอีกทั้งยังทันสมัย ที่สำคัญ NS-2 เป็นเครื่องมือที่นักวิจัยทั่วโลกต่างให้การยอมรับในความน่าเชื่อถือของผลการจำลอง

แต่อย่างไรก็ดีแม้เครื่องมือจะมีประสิทธิภาพเพียงไร หากผู้ใช้ไม่ใช้ให้เกิดประสิทธิภาพย่อมไม่มีประโยชน์ ข้อเสียหนึ่งของ NS-2 คือการใช้งานที่ค่อนข้างยากและซับซ้อน ซึ่งจำเป็นต้องสร้างความรู้ความเข้าใจเกี่ยวกับฌครงสร้างของเครื่องมือรวมถึงวิธีใช้เพื่อให้เกิดประโยชน์สูงสุดในการจำลอง

แบบจำลอง NS-2 นั้นจะถูกควบคุมการทำงานและปรับแต่งการจำลองด้วย TCL สคริป และคลาส C++ ซึ่งจำเป็นต้องมีพื้นฐานในเรื่องของภาษาที่ใช้ในระดับหนึ่ง รวมถึงควรเข้าใจถึงโครงสร้างการทำงาน และโครงสร้างพื้นฐานต่างๆของ NS-2 เพื่อที่จะสามารถใช้แบบจำลองนี้ได้อย่างมีประสิทธิภาพสูงสุด ซึ่งผลจากการจำลองจะอยู่ในรูปของเทรซไฟล์ ซึ่งเป็นเหมือนบันทึกเหตุการณ์ทุกอย่างที่เกิดขึ้นในการจำลอง แต่การวิเคราะห์ผลจากการจำลองอาจจำเป็นต้องใช้ Perl สคริปในการคัดแยกข้อมูลมาต้องการมาวิเคราะห์และประมวลผล เนื่องจากเทรซไฟล์จะบอกถึง

รายละเอียดเหตุการณ์ทุกอย่าง ทั้งที่มีประโยชน์ที่จะนำมาวิเคราะห์ในการทดลอง หรืออาจไม่จำเป็นต้องนำมาพิจารณา จากนั้นเมื่อทำการวิเคราะห์เสร็จเรียบร้อยสามารถใช้เครื่องมือต่างๆในการสร้างแผนภูมิในการแสดงผลการวิเคราะห์ดังกล่าวเพื่อให้ง่ายต่อการเข้าใจ หรือสะดวกในการเปรียบเทียบผลต่างๆ

ในวิทยานิพนธ์ฉบับนี้จะนำเสนอถึงความรู้พื้นฐานที่จำเป็นในการใช้แบบจำลอง NS-2 โดยจะเริ่มตั้งแต่พื้นฐานภาษาโปรแกรมที่ใช้เช่น C++ และ OTcl รวมถึงการนำเสนอทฤษฎีพื้นฐานโครงสร้างการทำงานของ NS-2 ที่จำเป็นต่อการใช้แบบจำลองเบื้องต้น เพื่อทดสอบระบบเครือข่ายในรูปแบบต่างๆ รวมถึงแสดงตัวอย่างการวิเคราะห์ผลจากการจำลองด้วยเครื่องมือต่างๆ ในรูปแบบต่างๆ โดยใช้ทฤษฎีพื้นฐานของระบบเครือข่าย และได้นำเทคโนโลยีใหม่ๆที่ไม่ใช่เทคโนโลยีพื้นฐาน มาทำการทดสอบ ซึ่งจะอยู่นอกเหนือจากความรู้พื้นฐานในการใช้แบบจำลองที่ได้กล่าวข้างต้น โดยจะนำเสนอวิธีการสร้างแบบจำลองเทคโนโลยี Multi Protocol Label Switching (MPLS) และเครือข่ายไร้สายเฉพาะกิจ (MANET) ซึ่งเป็นเทคโนโลยีและรูปแบบเครือข่ายที่ยังค่อนข้างใหม่และยังไม่แพร่หลายนัก การวิจัยและศึกษายังอยู่ในระดับที่น้อยกว่ามาตรฐานหรือเทคโนโลยีอื่นๆ จึงเป็นแรงจูงใจให้เกิดการทดลองและวิเคราะห์ เพื่อวัดประสิทธิภาพในแง่มุมต่างๆ

ในการจำลอง MPLS จะเป็นการพยายามจำลองโมเดลที่นอกเหนือจากพื้นฐาน โดยทั้งนี้เพื่อศึกษาการทำงาน, ขั้นตอนรวมถึงอัลกอริธึมต่างๆในการสร้างเส้นทางและการแลกเปลี่ยนลาเบล ซึ่งเป็นส่วนที่ต่างจากเครือข่าย TCP/IP ทั่วไป โดยจุดมุ่งหมายจะเป็นเพื่อการศึกษาการใช้ NS-2 และทำความเข้าใจใน MPLS และทำการวัดและเปรียบเทียบประสิทธิภาพ

โดยสำหรับ MANET นั้น จะเลือกวัดประสิทธิภาพจากปัจจัยที่เป็นประเด็นหลักในความแตกต่างระหว่าง MANET กับเน็ทเวิร์คทั่วไปเช่นอีเธอร์เน็ท ซึ่งก็คือปัจจัยในด้านการใช้พลังงาน เนื่องจากอุปกรณ์ไร้สายจะมีข้อจำกัดในเรื่องของแหล่งกำเนิดพลังงานนั่นเอง เราจึงทำการวิเคราะห์เพื่อหาจุดประสงค์ รวมถึงลักษณะการสูญเสียพลังงานในรูปแบบต่างๆของอุปกรณ์ไร้สายเพื่อที่จะสามารถเป็นทฤษฎีหรือใช้อ้างอิงในการศึกษาได้

โดยเนื้อหาในวิทยานิพนธ์จะเป็นเพียงส่วนหนึ่งของการใช้แบบจำลอง NS-2 ซึ่งจะเป็นเพียงรูปแบบหนึ่งในการใช้แบบจำลองเท่านั้น โดยแบบจำลองยังสามารถเพิ่มเทคโนโลยีใหม่ๆเข้าไปได้โดยผู้ใช้สามารถกำหนดและสร้างได้เอง หรือแม้แต่การใช้เทคโนโลยีที่มีอยู่เดิมมาทำการทดลองและวิเคราะห์เพื่อหาข้อเท็จจริงหรือข้อมูลที่ซ่อนอยู่ เพื่อเอามาขยายผลและประยุกต์รวมถึงใช้เป็นมาตรวัดประสิทธิภาพในแง่มุมต่างๆของระบบได้ ทั้งนี้ประสิทธิภาพการจำลองจะขึ้นอยู่แนวความคิดและสมมติฐานของผู้ทำการจำลองเป็นสำคัญ โดยจำเป็นต้องพยายามจำลองเพื่อครอบคลุมทุกความเป็นไปได้ และให้สอดคล้องกับสมมติฐานนั้น โดย NS-2 จะเป็นเพียงเครื่องมือที่ก่อให้เกิดผลลัพธ์จากการจำลองเท่านั้น โดยผลลัพธ์ดังกล่าวจะนำมาวิเคราะห์หรือหาข้อเท็จจริงให้เกิดประโยชน์เพียงใดก็ขึ้นอยู่กับตัวผู้ทำการจำลองเอง

บทที่ 2

Network Simulator 2

2.1 การวิเคราะห์ประสิทธิภาพของระบบเครือข่าย

การที่เราต้องการรู้ประสิทธิภาพของระบบเครือข่ายเราสามารถหาได้จากการวิเคราะห์ ซึ่งการวิเคราะห์ประสิทธิภาพของเครือข่ายนั้นทำได้ 3 วิธีคือ การวิเคราะห์ด้วยหลักการทางคณิตศาสตร์ (Analytical Model) การจำลอง (Simulation) และการวัดจากระบบจริง (Measurement) ซึ่งทั้ง 3 วิธีนั้นจะมีข้อดีและข้อเสียแตกต่างกันไป โดยการวิเคราะห์ด้วยหลักการทางคณิตศาสตร์นั้น มีข้อดีคือ สามารถทำ ณ. ขณะใดก็ได้ไม่ว่าจะอยู่ในช่วงของการออกแบบหรือติดตั้งระบบเครือข่ายไปแล้ว และที่สำคัญใช้เวลาค่อนข้างง่ายและเร็วเพราะสามารถแทนค่าลงไปในสมการได้เลย เพียงแต่ผลการวิเคราะห์จากการคำนวณจะเป็นเพียงค่าในแง่ทฤษฎีเท่านั้น อาจไม่แม่นยำเพียงพอ แต่ประเด็นสำคัญอีกประการก็คือใช้ค่าใช้จ่ายต่ำเมื่อเทียบกับวิธีอื่น ขณะที่การจำลองนั้น จะคล้ายกับวิธีการวิเคราะห์ด้วยหลักการทางคณิตศาสตร์ (Analytical Model) เพียงแต่จะมีความแม่นยำที่มากกว่า เนื่องจากการจำลองจะทำให้เห็นภาพได้ง่าย และสามารถจำลองระบบขึ้นมาแทนระบบจริงได้โดยคุณสมบัติใกล้เคียงกัน แต่ทั้งนี้ก็ขึ้นอยู่กับคุณสมบัติของแบบจำลองด้วย โดยการวิเคราะห์จะกระทำผ่านภาษาโปรแกรม ทำให้จำเป็นต้องใช้ทักษะทางด้านนี้ด้วย วิธีสุดท้ายแบบการวัดจากระบบจริง เป็นวิธีที่มีความแม่นยำมากที่สุดเนื่องจากต้องทำหลังจากมีการติดตั้งระบบเครือข่ายแล้วเท่านั้น ซึ่งค่าที่ได้จากการวิเคราะห์คือค่าที่เกิดขึ้นจริงๆ แต่ทั้งนี้วิธีนี้จำเป็นต้องใช้ค่าใช้จ่ายสูงเนื่องจากต้องติดตั้งเสียก่อน ซึ่งเราสามารถสรุปได้ดังนี้

ตารางที่ 2.1 แสดงการเปรียบเทียบของการวิเคราะห์ประสิทธิภาพของเครือข่าย

เกณฑ์	Analytical Model	Simulation	Measurement
สถานะ	Any	Any	Post prototype
เวลาที่ใช้	เล็ก	ปานกลาง	แปรปรวน
เครื่องมือ	นักวิเคราะห์	ภาษาคอมพิวเตอร์	อุปกรณ์
ความถูกต้องแม่นยำ	น้อย	ปานกลาง	แปรปรวน
ค่าใช้จ่าย	น้อย	ปานกลาง	สูง
Salability	น้อย	ปานกลาง	สูง

โดยปกติแล้วเราจะทำการวิเคราะห์จากระบบที่เราออกแบบ ซึ่งเราจะกระทำก่อนที่จะติดตั้งระบบจริงเพื่อไม่ให้เป็นการสิ้นเปลืองค่าใช้จ่าย ดังนั้นวิธีการวิเคราะห์จากระบบจริงจึงไม่

เหมาะสม ขณะที่การวิเคราะห์ด้วยหลักการทางคณิตศาสตร์ไม่อาจให้ข้อมูลที่แม่นยำได้เพียงพอ ทำให้วิธีที่นิยมที่สุดคือการจำลอง เนื่องจากให้ข้อมูลที่ค่อนข้างแม่นยำ รวมถึงมีความยืดหยุ่นสูง สามารถปรับแต่งแก้ไขได้โดยง่ายและรวดเร็ว ทำให้สะดวกในการเปรียบเทียบหลายระบบเข้าด้วยกัน แต่อย่างไรก็ตามวิธีนี้จำเป็นต้องใช้ทักษะทางด้านการเขียนโปรแกรม รวมถึงความรู้และทฤษฎีทางด้านสถิติและเครือข่ายที่เพียงพอ

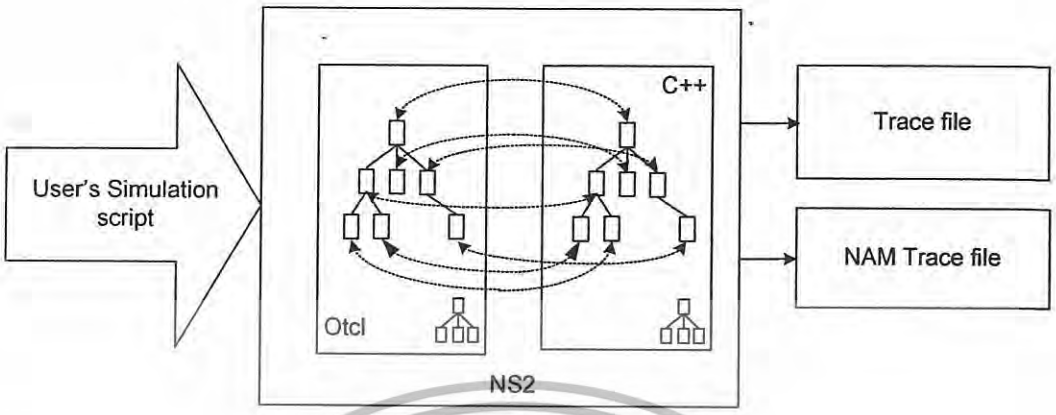
เครื่องมือที่ใช้ในการจำลองมีมากมาย เช่น เช่น OPNET, GlobalSIM และ NS-2 ก็เป็น 1 ในนั้น ซึ่งต่างก็มีคุณสมบัติที่แตกต่างกันไป แต่เหตุผลที่ใช้ NS-2 เนื่องจากได้มีการใช้งานและได้รับการยอมรับในระดับหนึ่งถึงความน่าเชื่อถือและความแม่นยำ และที่สำคัญยังฟรี ซึ่ง NS-2 สามารถทำงานได้บนหลายแพลทฟอร์ม และมีความยืดหยุ่นสูงในแง่ของการวิเคราะห์ผลลัพธ์ ซึ่งเราสามารถใช้ภาษาใดๆก็ได้ และ NS-2 ยังมีฟังก์ชั่นรวมถึงคลาสต่างๆเกี่ยวกับการจำลองระบบเครือข่ายที่จำเป็นเตรียมไว้อยู่แล้ว ที่เราสามารถดึงมาใช้งานได้โดยสะดวกอีกทั้งเรายังสามารถโมเดลใหม่ๆเข้าไปได้ ไม่ว่าจะรูปแบบการส่ง หรือโปรโตคอลใหม่ๆ โดย NS-2 มีการพัฒนาปรับปรุงและเพิ่มเติมให้เหมาะสมกับระบบเครือข่ายในปัจจุบันได้เป็นอย่างดี จึงทำให้ NS-2 เป็นเครื่องมือที่ใช้ในการจำลองกันอย่างแพร่หลายมาก

2.2 ประเภทของการจำลอง

การจำลองนั้นมีหลายประเภทสามารถแบ่งได้ดังนี้

- Emulation เป็นการจำลองที่ใช้อุปกรณ์ทางฮาร์ดแวร์มาเกี่ยวข้อง
- Monte Carlo Simulation เป็นการจำลองที่ไม่เกี่ยวกับเวลาส่วนมากจะเป็นการทดลองหาค่าสถิติ
- Trace-driven Simulation เป็นการจำลองโดยใช้ข้อมูลที่ได้จากการทดลองจริงเป็นอินพุตของระบบที่จำลอง
- Discrete-time Simulation เป็นการจำลองระบบที่ไม่ต่อเนื่องทางเวลา การจำลองแบบนี้สามารถตั้งเวลาจำลองเหตุการณ์ด้วยเวลาที่คงที่(unit time Approach) หรือการจำลองด้วยเวลาที่ควบคุมด้วยเหตุการณ์ที่เกิดขึ้น (event-driven approach)

2.3 โครงสร้างของ NS



รูปที่ 2.1 แสดงโครงสร้างของ NS

การจำลองใน ns นั้นจะเป็นการจำลองเน็ตเวิร์คภายใต้เงื่อนไขต่างๆ โดยจุดประสงค์หลักของการทดลองก็เพื่อทดสอบหรือทำการวิจัย ซึ่งสิ่งที่ผู้ใช้ต้องการได้จากการจำลองก็คือผลการทดลอง โดยผู้ใช้แต่ละคนย่อมต้องการผลลัพธ์ที่แตกต่างกันไปเพื่อนำไปใช้หรือการวิเคราะห์ ผลลัพธ์ที่ได้จากการจำลองก็คือ เทรสไฟล์ (trace file) ซึ่งในการวิจัยนั้นเราอาจไม่ได้นำผลลัพธ์ดังกล่าวไปใช้งานโดยตรง จะต้องนำมาคำนวณหรือผ่านกระบวนการบางอย่างเพื่อที่จะได้ข้อมูลที่ต้องการออกมา ซึ่งแต่ละคนก็ต้องการข้อมูลหลากหลายรูปแบบ เพื่อจะนำมาวิเคราะห์ในจุดประสงค์ที่แตกต่างกันไป

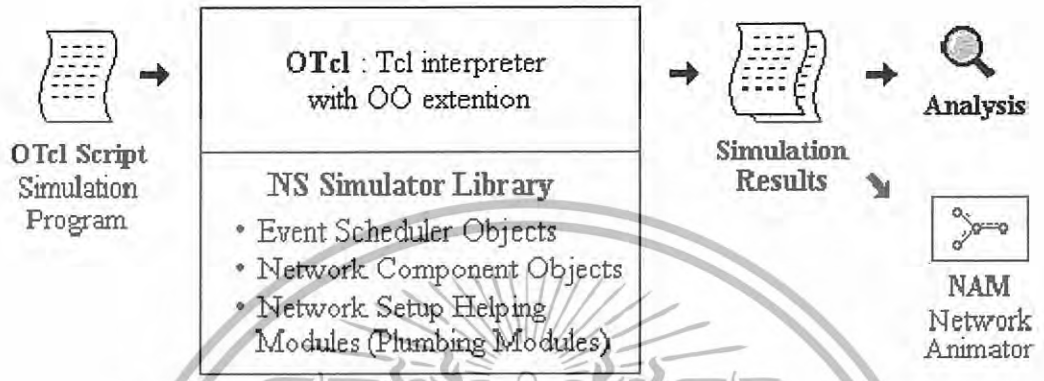
NS2 เป็นการจำลองที่มีลักษณะเป็นการจำลองที่เป็นแบบใช้เวลาเป็นตัวกำหนดเหตุการณ์ต่างที่เกิดขึ้น โดยจะจำลองในระดับแพ็คเกตซึ่งเขียนด้วยโปรแกรมเชิงวัตถุ 2 ภาษา คือ OTcl และ C++ ภายใน NS2 นั้นจะแบ่งเป็นลำดับชั้น (Hierarchy) 2 ชนิด คือ C++ จะทำการสร้างกลุ่มที่เป็นลำดับชั้นของตัวแปรภาษา (compiler hierarchy) และ OTcl จะสร้างกลุ่มที่เป็นลำดับชั้นของตัวแปรภาษา (interpreter hierarchy) โดยทั้งสองโครงสร้างนี้จะทำงานร่วมกัน

ในการใช้งาน เมื่อผู้ใช้ต้องการจำลอง จะสามารถเขียนโปรแกรมเพื่ออธิบายถึงสิ่งที่ต้องการจำลองโดยใช้ภาษา TCL แต่ถ้าหากว่าต้องการแก้ไขเปลี่ยนแปลงการทำงานของ NS2 จะต้องแก้ไขในส่วนของโครงสร้างที่เป็น hierarchy ซึ่งจะต้องทำโดยใช้ภาษา C++ และ OTCL

ในส่วนของผลของการจำลองนั้น NS สามารถให้ข้อมูลจากการจำลองได้ค่อนข้างละเอียดอยู่ในรูปของไฟล์ที่เป็นตัวอักษร (Text-based) หรือว่าจะแสดงผลออกมาอยู่ในรูปของกราฟิกที่เรียกว่า NAM (Network Animator) ก็ได้ ซึ่ง NAM มีส่วนที่ใช้ในการติดต่อกับผู้ใช้ที่เป็นกราฟิก สามารถแสดงข้อมูลที่อยู่ในรูปแบบต่างๆได้หลากหลาย แต่ว่าข้อมูลที่เป็นแบบกราฟิกนั้นก็ไม่ถูกต้องพอที่จะใช้ในการวิเคราะห์การจำลองได้การนำผลการทดลองที่ได้มาแสดงนั้นอาจใช้ Perl และ

Gnuplot โดยอาจดึงค่าผลการทดลองที่ได้มาใช้ในการคำนวณต่อเพื่อหาสิ่งที่ต้องการแล้วแสดงผลออกมาในรูปของกราฟ

2.4 NS ในมุมมองของผู้ใช้



รูปที่ 2.2 แสดงถึง NS ในมุมมองของผู้ใช้

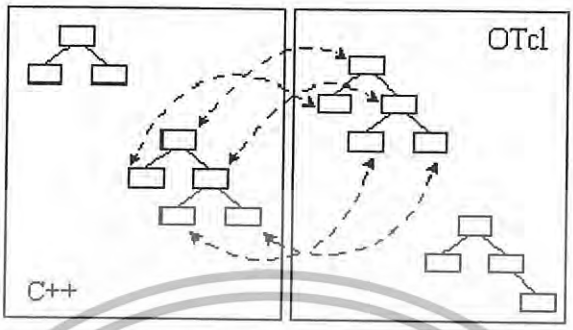
ในมุมมองของผู้ใช้ NS เป็นตัวแปรภาษา Object-oriented Tcl (OTcl) ซึ่งภายในประกอบด้วย

- Simulation event scheduler
- Network component object
- Network setup Helping
- Module libraries

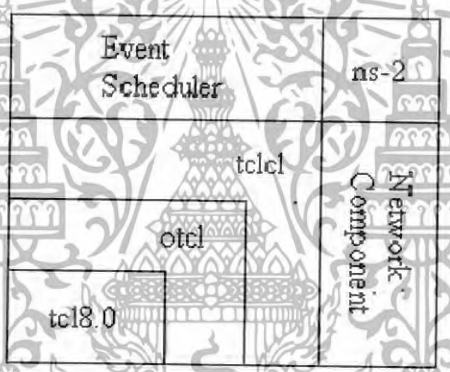
ในการใช้งาน NS นั้น เราจะเขียนภาษา OTcl เพื่อเป็นการตั้งค่าและรันการจำลองระบบเครือข่าย โดยที่ผู้ใช้นั้นจะทำการเขียน OTcl script ซึ่งใช้เริ่ม event scheduler , ทำการกำหนดโครงสร้างของเครือข่าย (network topology)โดยใช้ network object แล้วทำเรียกฟังก์ชันต่างๆ ใน library จากนั้นก็กำหนดแหล่งกำเนิดของ traffic โดยกำหนดว่าเมื่อไรจะเริ่มต้นและสิ้นสุดการส่งแพ็คเกต ผ่านทาง event scheduler ซึ่ง event Scheduler จะเป็นตัวที่บันทึกเวลาในการจำลอง และทำให้เกิดเหตุการณ์ทุกเหตุการณ์ ที่อยู่ในคิว(queue scheduled) โดยจะเรียก Network Component ที่เหมาะสมมาทำงาน

NS นั้นไม่ได้เขียนโดยใช้ OTcl อย่างเดียวแต่ใช้ C++ ด้วย โดย NS จะทำการแยกการ ทำงานของสองส่วนนี้ออกจากกันเพื่อเป็นการลดเวลาในการประมวลผลแพ็คเกตและเหตุการณ์ โดย event scheduler และออปเจค (object) ของ network component พื้นฐาน

รูปที่ 2.3 แสดงลำดับชั้นของ object ที่อยู่ภายใน C++ และ OTcl สำหรับ C++ object จะมีการสร้างลำดับชั้นของ OTcl Linkage ซึ่งจะไปสัมพันธ์กับลำดับชั้นของ OTcl Object เสมือนกับลำดับชั้นที่อยู่ใน C++



รูปที่ 2.3 แสดงถึงการทำงานร่วมกันระหว่าง C++ และ OTcl



รูปที่ 2.4 แสดงมุมมองโครงสร้างของ NS

รูปที่ 2.4 แสดงโครงสร้างของ NS ในมุมมองของผู้ใช้ทั่วไปโดยให้มองจากมุมล่างซ้าย แล้วทำการออกแบบและทำการจำลองภายใน Tcl โดยใช้ simulator object ที่อยู่ใน OTcl Library

Event Scheduler และ network component ส่วนใหญ่นั้นจะเขียนโดยใช้ C++ และมีไว้เพื่อให้ OTcl สามารถเข้ามาเรียกใช้ ได้ โดยผ่านทาง OTcl Linkage ซึ่งเขียนด้วย Tclcl

2.5 การจำลอง NS2 พื้นฐาน

การจำลองภายใน NS2 นั้นจะแบ่งการจำลองออกได้ 2 แบบคือ

1. การจำลองแบบลอจิคอลโมเดลซึ่งการจำลองแบบนี้จะอยู่บนระดับชั้นที่ 3-5 คือ ระดับชั้นเนตเวิร์ก ระดับชั้นทรานสปอร์ตและระดับชั้นแอพพลิเคชั่นในระดับชั้นที่ 1 และ 2 นั้นจะจำลองด้วย queue และ link delay
2. การจำลองแบบเรียลโมเดลจะใช้จำลองในทุกระดับชั้น

2.6 ส่วนประกอบต่างๆของ NS2

2.6.1 Class Simulator

ภาพรวมทั้งหมดของการจำลองนั้นจะถูกอธิบายด้วย Class simulator ซึ่งคลาสนี้จะมีกลุ่มของอินเตอร์เฟสที่ใช้สำหรับการกำหนดค่าต่างๆในการจำลอง และสำหรับการเลือกชนิดของ Event Scheduler ที่ใช้ทำให้เกิดการจำลองขึ้น สคริป (script) ที่ใช้ในการจำลองนั้นมักจะเริ่มด้วยการสร้าง Instance ของคลาสนี้ แล้วเรียกหลายเมธอดในการสร้างโหนด โทโพโลยีและการกำหนดค่าอื่นๆที่ใช้ในการจำลอง

2.6.2 การกำหนดค่าเริ่มต้นในการจำลอง

เมื่อมีการสร้างออบเจกต์ของการจำลองใหม่ใน tc1 Initialization procedure จะมีการทำตามขั้นตอนดังนี้

- ทำการสร้างรูปแบบของ แพ็คเกต
- สร้าง scheduler – ประมวลผลการจำลองเป็นแบบ event-driven และสามารถใช้ Scheduler ได้หลายแบบ
- สร้าง null agent ซึ่งจะเป็นตัวประมวลผล drop แพ็คเกต หรือ แพ็คเกต ที่ไม่มีปลายทาง

2.6.3 ตัวจัดการตารางเวลาและเหตุการณ์

NS เป็นการจำลองที่เป็นแบบ event-driven โดยมีการจัด scheduler ในการจำลองออกเป็น 4 แบบ โดยจะแบ่งตามโครงสร้างของข้อมูล ได้แก่

- List Scheduler เป็น scheduler ที่ใช้โครงสร้างของลิงค์ลิสต์ซึ่งเรียงตามลำดับของเวลาจะทำงาน โดยเริ่มที่หัวของลิสต์เป็นไปตามลักษณะ FIFO
- Heap Scheduler เป็น scheduler ที่ใช้โครงสร้างของ Heap โดยจะเรียงลำดับโครงสร้างของลิสต์ โดยให้เหตุการณ์ที่มีหมายเลขมากกว่าอยู่ข้างบนของลิสต์ และเวลาที่ใช้ในการแทรก หรือลบเป็น O(log n) สำหรับ n เหตุการณ์

- Calendar Queue Scheduler เป็นโครงสร้างข้อมูลที่คล้ายๆเป็น Calendar 1 ปี โดยเหตุการณ์ที่เกิดในวันและเดือนเดียวกันในหลายๆปี จะทำการบันทึกเอาไว้ที่วันเดียวกัน
- Real time Scheduler

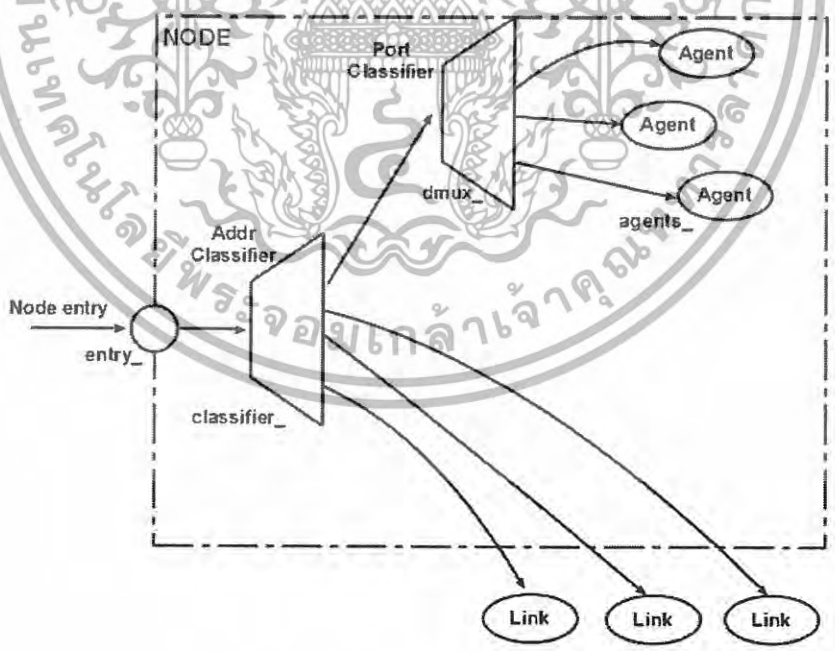
2.6.4 โหนดและการส่งแพ็คเกท

Node ที่อยู่ใน ns2 มี 3 แบบคือ

- โหนดพื้นฐาน
- โหนดแบบมัลติคาส
- โหนดแบบลำดับชั้น

โครงสร้างอย่างง่ายของโหนดนั้นประกอบด้วยส่วนของ OTclObjects 2 ส่วนคือ Address Classifier ซึ่งทำหน้าที่ในการส่งแพ็คเกตโดยจะใช้หมายเลขลอจิคอล ในการกำหนดเส้นทาง และ Port Classifier ซึ่งทำหน้าที่ส่งแพ็คเกตตามหมายเลขพอร์ต

แต่ละโหนดจะต้องประกอบไปด้วยแอดเดรสหรือหมายเลขไอดี ที่ไม่ซ้ำกัน ภายในเนมสเปซโหนดถูกสร้างขึ้นและจะต้องมี ลิสต์ของ neighbors ด้วย

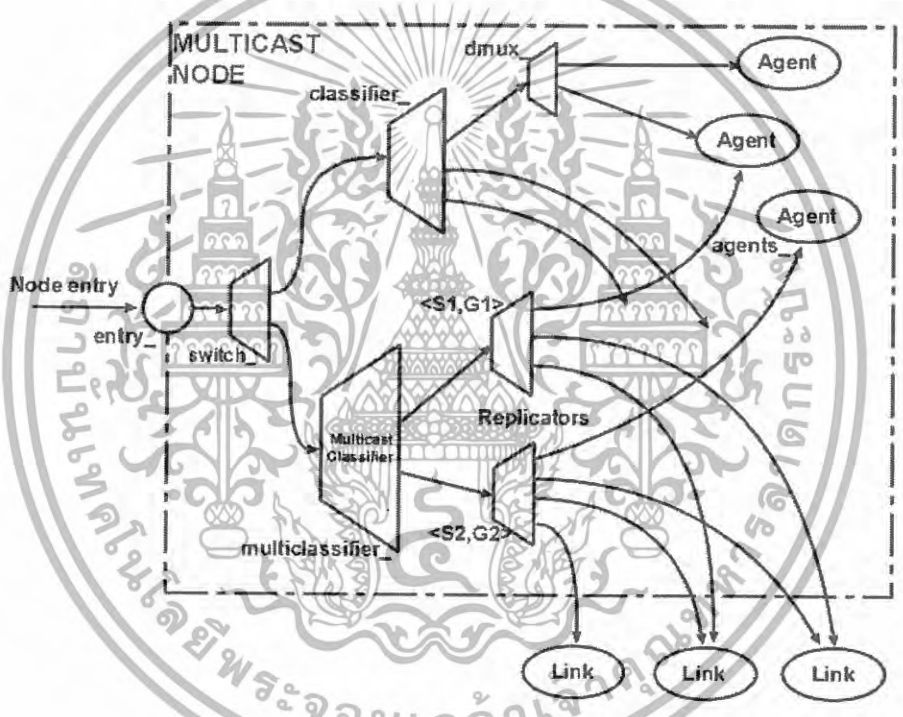


รูปที่ 2.5 แสดงโครงสร้างของโหนดที่เป็นยูนิคาส

โดย ดีฟอลท์แล้วโหนดที่สร้างขึ้นมาใน ns นั้นจะเป็นการจำลองแบบยูนิคาส แต่ว่าถ้าต้องการจำลองในลักษณะที่เป็นมัลติคาสก็สามารถทำได้โดยการใส่ option "-multicast on" เพิ่มเข้าไป เช่น

```
set  ns [new simulator - multicast on]
```

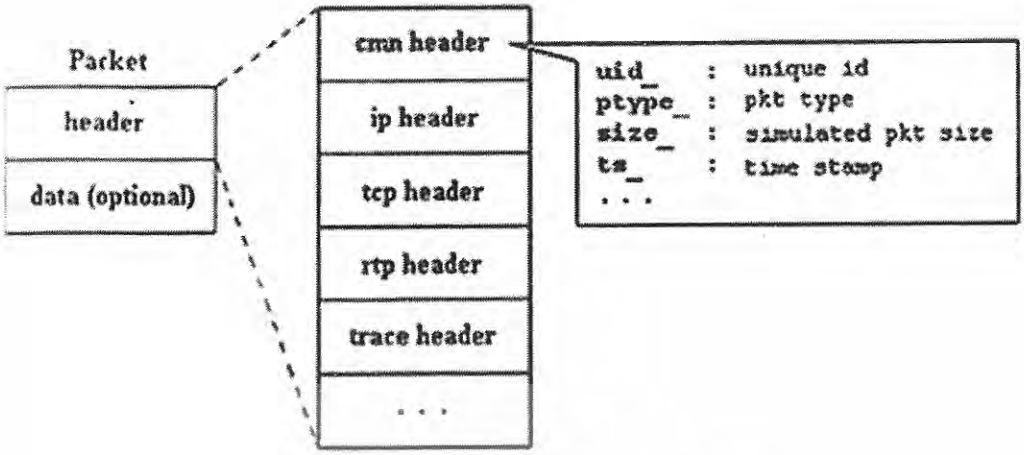
ในกรณีที่ใช้เป็นมัลติคาสเราท์ติ้ง บิตที่สูงที่สุดของแอดเดรสนั้นจะบอกว่าแอดเดรสนั้นมีการส่งแบบใด ซึ่งถ้าเกิดว่าบิตนั้นเป็น 0 จะหมายถึงการส่งที่เป็นยูนิคาสแต่ถ้าหากว่านอกเหนือจากนี้จะถือว่าเป็นการส่งแบบมัลติคาส



รูปที่ 2.6 แสดงโหนดที่เป็นลักษณะมัลติคาส

2.6.5 แพ็คเกตเฮดเดอร์และรูปแบบของแพคเก็ท

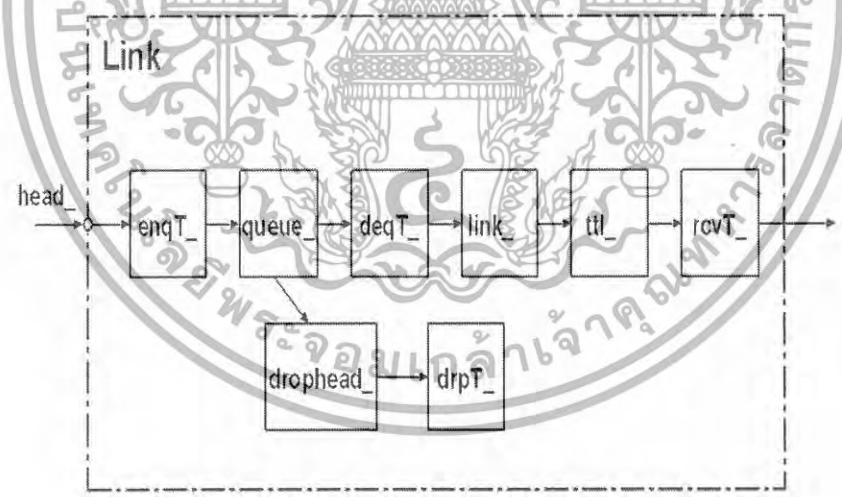
แพ็คเกต ของ NS2 แต่ละแพ็คเกต นั้นจะประกอบด้วยเฮดเดอร์และข้อมูล ซึ่งในส่วนของเฮดเดอร์ ก็มีหลายประเภท แต่ในการจำลองนั้นจะเลือกใช้เฉพาะเฮดเดอร์ที่จำเป็นเพื่อเป็นการประหยัดหน่วยความจำที่ใช้ในการจำลอง



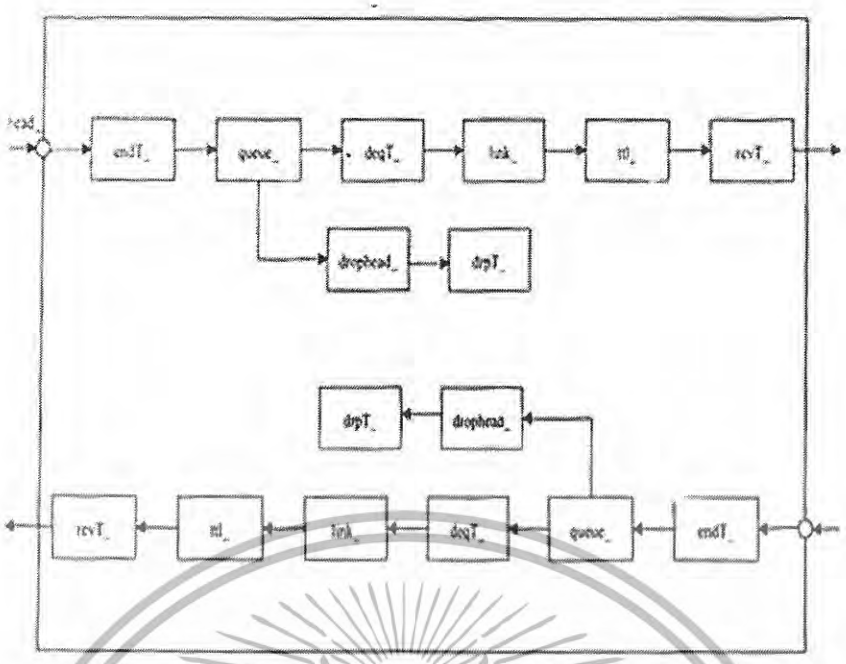
รูปที่ 2.7 แสดงรูปแบบแพ็คเกตของ NS

2.6.6 ลิงค์และดีเลย์

ลิงค์ใน ns นั้นแบ่งออกเป็นสองประเภทคือซิมเพล็กซ์ลิงค์(simplex link) และดูเพล็กซ์ลิงค์ (duplex link) ซึ่งโครงสร้างที่เป็นซิมเพล็กซ์ลิงค์นั้นจะประกอบด้วยคิวและเวลาหน่วงของลิงค์มาเกี่ยวข้องด้วย



รูปที่ 2.8 แสดงซิมเพล็กซ์ลิงค์



รูปที่ 2.9 แสดงดูเพล็กซ์ลิงค์

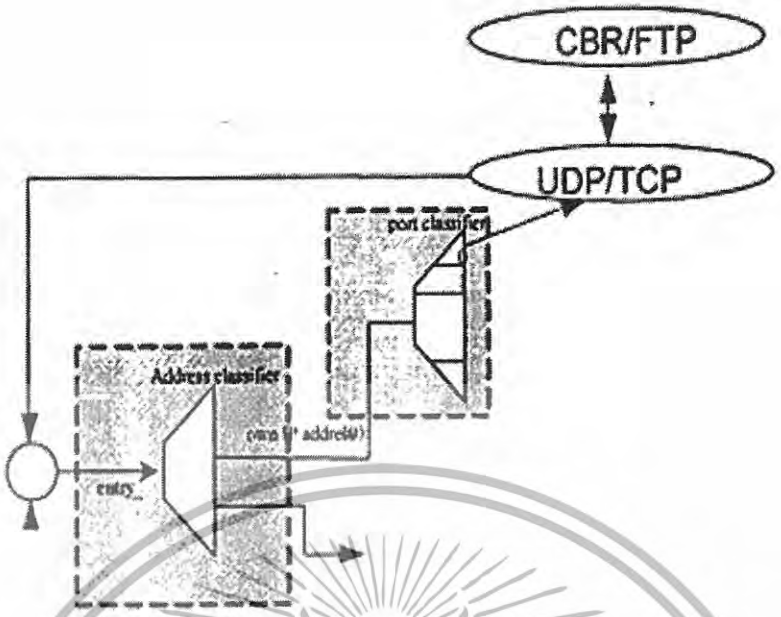
รูปที่ 2.10 แสดงแบบจำลองของดูเพล็กซ์ลิงค์

2.6.7 การจัดการคิวและแพ็คเกต Scheduling

คิวเป็นตำแหน่งที่แพ็คเกตอาจจะถูกเก็บไว้เพื่อส่งต่อหรือดรอปทิ้ง ส่วนแพ็คเกตสเกดูลิ่งจะเป็นกระบวนการที่จะตัดสินใจว่าแพ็คเกตนั้นควรส่งต่อหรือว่าจะทิ้งไป ในการจัดการบัฟเฟอร์ของคิวนั้นมีหลายแบบ เช่น drop-tail ซึ่งเป็นการจัดการคิวที่อิงใช้หลักการ FIFO , การจัดการบัฟเฟอร์ที่เป็นแบบอาร์อีดี , การจัดการที่เป็นแบบซีบีคิว ซึ่งการจัดการแบบนี้จะเกี่ยวกับเรื่องของการจัดลำดับความสำคัญก่อนหลังและการจัดแบบราวด์โรบิน(Round-Robin) เป็นต้น

2.6.8 เอเจนท์

เอเจนท์นั้นเป็นจุดปลายในการเชื่อมต่อของแพ็คเกตในเนตเวิร์กเลเยอร์ซึ่งถูกสร้างขึ้นหรือว่าถูกใช้งาน ในการใช้งานเอเจนท์นั้นจะใช้งานกับโปรโตคอลในหลายระดับชั้น ตัวอย่างของเอเจนท์ที่อยู่ใน ns เช่น TCP UDP RTP และอื่นๆ ซึ่งคลาสเอเจนท์นั้นสร้างขึ้นมาโดยใช้ OTcl และ C++



รูปที่ 2.11 โครงสร้างการใช่งานเอเจนท์และแอพพลิเคชั่น

2.6.9 แอพพลิเคชั่น

โปรแกรมประยุกต์ใน NS นั้นก็มีอยู่หลายชนิดแต่ละชนิดก็ใช้งานไม่เหมือนกัน ตัวอย่างโปรแกรมประยุกต์ใน ns เช่น Application/Traffic/Exponential , Application/Traffic/Pareto , Application/Traffic/CBR , Application/Traffic/Trace , Application/Traffic/FTP เป็นต้น

2.6.10 เทรซไฟล์

เทรซไฟล์ใน ns นั้นจะเป็นผลที่ได้จากการจำลอง ซึ่งภายในจะประกอบไปด้วยรายละเอียดของข้อมูล โดยจะบันทึกทุกแพ็คเกตที่เข้าและออกรวมทั้งแพ็คเกตที่ทิ้งไปจากคิว ตัวอย่างข้อมูลที่อยู่ในเทรซไฟล์เช่น เหตุการณ์ เวลาที่เกิดเหตุการณ์ โหนดที่อยู่ต้นทางและปลายทาง ชนิดและขนาดของแพ็คเกจ หมายเลขไอพีแอดเดรสของต้นทางและปลายทาง เป็นต้น

เหตุการณ์ใน ns นั้นมีทั้งหมด 4 ชนิด คือ

- r แทนโมดูล rcvT_

  ตัวอย่าง

  ```
  r 0.865504 3 4 cbr 210 ------- 0 0.0 9.0 196 540
  ```

- + แทนโมดูล endT_

  ตัวอย่าง

  ```
  + 0.865504 4 8 cbr 210 ------- 0 0.0 9.0 196 540
  ```

- - แทนโมดูล depT_

  ตัวอย่าง

```
- 0.865504 4 8 cbr 210 ------- 0 0.0 9.0 196 540
```

- d แทนโมดูลdrpT_

  ตัวอย่าง

```
d 0.865504 4 8 cbr 210 ------- 0 0.0 9.0 196 540
```

2.7 การสร้างโหนดและการกำหนดคุณสมบัติ

เราสามารถทำการ start NS simulator ได้ด้วยคำสั่ง

```
set ns [new Simulator]
```

ซึ่งคำสั่งนี้เหมือนเป็นการเปิดตัวซิมูเลเตอร์ (Simulator) โดยทำการดึง instance จาก class Simulator มาเก็บไว้ที่ออบเจค (object) ที่ชื่อ ns ซึ่งคำสั่งนี้จะอยู่บรรทัดบนสุดของ tcl สคริปเสมอ

ใน NS นั้นอุปกรณ์เน็ตเวิร์ค (network) ต่างๆที่สามารถทำการส่งและรับข้อมูลได้ จะอยู่ในรูปแบบของโหนดซึ่งการสร้างโหนดจะกระทำได้ด้วยคำสั่ง

```
set n0 [$ns node]
```

จากตัวอย่างเราจะทำการสร้างโหนด ที่มีชื่อว่า n0 ซึ่งหลังจากบรรทัดนี้ หากเราต้องการนำโหนดนี้ไปใช้ เราจะสามารถอ้างอิงถึงโหนดนี้ได้โดยพิมพ์ `$n0` ซึ่งโหนดทุกโหนดจะต้องประกอบไปด้วยอย่างน้อยสองสิ่งนี้

- ที่อยู่หรือไอดี (ID) ของโหนดนั้น
- รายการของโหนดที่อยู่ติดกันของโหนดนั้น

โดยที่อยู่ของโหนดจะมีลักษณะเป็นลำดับชั้น โดยจะมีลำดับอย่างไรจะขึ้นอยู่กับชนิดของโหนดเช่นในเบสิก (Basic) โหนดจะมี 2 ระดับ โดยจะมีรูปแบบเป็น x.x คือตัวจำแนกที่อยู่และตัวจำแนกพอร์ท (port) ตามลำดับโดยคั่นแต่ละระดับด้วย "." โดยที่อยู่จะทำหน้าที่เหมือนเป็นที่อยู่ทางตรรกศาสตร์ขณะที่พอร์ทจะทำหน้าที่เหมือนเป็นพอร์ทของที่อยู่ทางตรรกศาสตร์นั้น

ใน ns นั้น เราไม่สามารถกำหนดได้ ns จะเป็นผู้กำหนดให้เอง โดยจะกำหนดให้ตามลำดับที่โหนดนั้นถูกสร้าง โดยจะมีค่าเริ่มต้นคือ 0 และจะเพิ่มขึ้นทีละ 1 ยกตัวอย่างเช่น

```
set n0 [$ns node]
set n1 [$ns node]
```

ทั้ง 2 โหนดจะได้รับการกำหนดที่อยู่เป็น 0.0 และ 1.0 ตามลำดับ

2.8 การสร้างลิงค์และการกำหนดคุณสมบัติ

เมื่อเราได้มีการสร้างโหนดเกิดขึ้นหลายโหนดย่อมเกิดจากจุดประสงค์ที่ต้องการให้โหนด แต่ละโหนดสามารถติดต่อถึงกันได้ ใน NS จะมีการสร้างการเชื่อมต่อของแต่ละโหนดเรียกว่าลิงค์ โดยเราสามารถสร้างลิงค์ได้ด้วยคำสั่ง

```
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
```

Duplex-link นั้นเป็นฟังก์ชั่นในการสร้างการเชื่อมต่อระหว่าง 2 โหนดแบบดูเพล็กซ์ (duplex) คือส่งข้อมูลได้ 2 ทิศทาง ประกอบด้วย 5 พารามิเตอร์คือ โหนดต้น, โหนดปลาย, แบนวิธ (Bandwidth), ความหน่วงและชนิดของคิว (queue) ซึ่งจากคำสั่งนี้จะหมายความว่า "กำหนดให้ n0 และ n2 เชื่อมต่อกันแบบดูเพล็กโดยให้มีความเร็วในการส่งข้อมูลที่ 5 เมกะบิท (Mb) ต่อวินาทีและมีความหน่วงอยู่ที่ 10 มิลลิวินาทีโดยใช้คิวแบบดรอปเทล (DropTail)"

หากเราต้องการสร้างลิงค์แบบซิมเพล็กซ์คือส่งข้อมูลได้ในทิศทางเดียว ก็ทำได้โดยพิมพ์

```
$ns simplex-link $n0 $n2 10Mb 10ms DropTail
```

โดยจะมีผลให้เกิดลิงค์ ระหว่าง n0 และ n2 แบบซิมเพล็กซ์โดยการส่งข้อมูลจะเป็นไปในทิศทางจาก n0 ไป n2 เท่านั้น

ดรอปเทลจะเป็นคิวที่หากมีการเข้าคิวเกินกว่าที่บัฟเฟอร์ (buffer) จะรองรับได้จะทำการโยนแพ็คเกตที่เข้าคิวล่าสุดทิ้ง นอกจากนี้ชนิดของคิวยังมีอีกหลายชนิดเช่น RED (Random Early Discard), FQ (Fair Queuing) หรือ DRR (Deficit Round-Robin) เป็นต้น

ซึ่งเราสามารถกำหนดขนาดของบัฟเฟอร์ของคิวของแต่ละลิงค์ได้โดยพิมพ์ดังนี้

```
$ns queue-limit $n0 $n2 20
```

คือกำหนดให้ลิงค์ระหว่าง n0 และ n2 คิวมีขนาด 20 แพ็คเกต

ซึ่งอย่างไรก็ตามบางพารามิเตอร์ จะมีค่าดีฟอลท์อยู่ ซึ่งหากเราไม่กำหนดค่าให้ ns จะทำการใช้ค่าดีฟอลท์ซึ่ง queue-limit มีค่าดีฟอลท์คือ 50 ซึ่งจาก code ด้านบน เราจะกำหนดขีดจำกัดของลิงค์ระหว่าง n0 และ n2 เท่านั้น ซึ่งหากมีการสร้างลิงค์ อื่นๆ นั้นจะมีลิมิต ตาม ค่าดีฟอลท์

ใน ns เราสามารถหาค่าดีฟอลท์ ต่างๆได้โดยดูที่ไฟล์ ns-default.tcl จะพบคำสั่ง

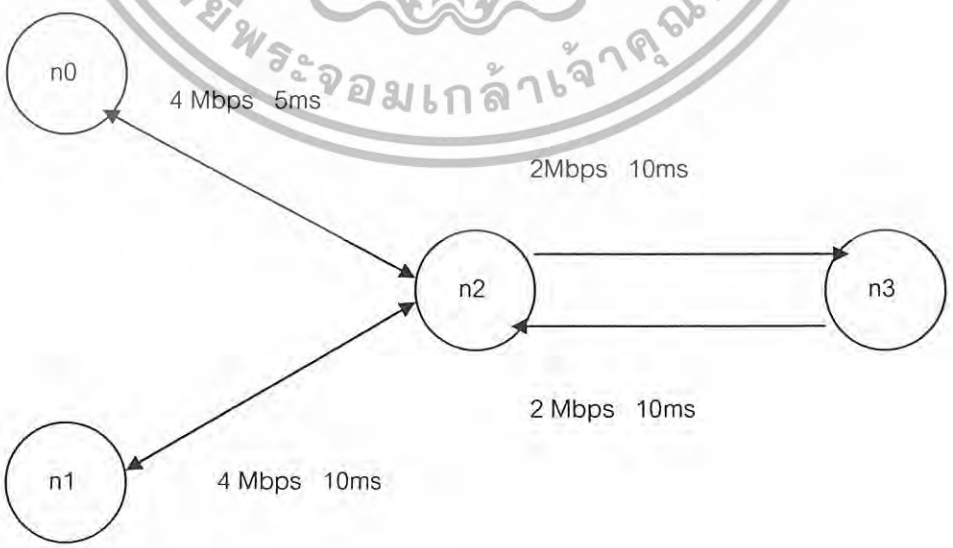
```
Queue set limit_ 50
```

โดยการที่เราจะรู้ว่ามีค่าดีฟอลท์ หรือไม่ และตัวแปรใด เราต้องดูจาก ns-lib.tcl และไปดูที่ฟังก์ชั่น Queue-limit จะสามารถทราบได้ว่าต้องการพารามิเตอร์ใดบ้างและมีค่าดีฟอลท์ หรือไม่

ตัวอย่างที่ 1. แสดงการสร้างโหนดและลิงค์

```
#create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#create link between node
$ns duplex-link $n0 $n2 4Mb 5ms DropTail
$ns duplex-link $n1 $n2 4Mb 10ms DropTail
$ns simplex-link $n2 $n3 2Mb 10ms DropTail
$ns simplex-link $n3 $n2 2Mb 10ms DropTail
# set queue size of link between n2 and n3
$ns queue-limit $n2 $n3 20
```

จากตัวอย่างที่ 1. จะได้ลักษณะของโหนดและลิงค์ดังนี้



รูปที่ 2.12 แสดงตัวอย่างเน็ทเวิร์ค

2.9 การสร้างเอเจนท์และแอพพลิเคชัน

หลักจากที่เราสร้างโหนดและลิงค์ได้แล้ว มันจะไม่มีประโยชน์เลยหากเราไม่มีการสร้าง ทราฟฟิกให้แก่มัน ซึ่งเราจะต้องทำการสร้างเส้นทางของทราฟฟิกโดยการกำหนดต้นทางและ ปลายทางแก่เอเจนท์ และแอพพลิเคชันเพื่อที่จะได้ใช้มัน

2.9.1 เอเจนท์

เอเจนท์จะเป็นเสมือนจุดปลายที่ระดับชั้นเน็ทเวิร์คจะทำการสร้าง หรือใช้งาน หรือจะกล่าวได้ว่าเอเจนท์เปรียบเทียบได้กับระดับชั้นทรานสปอร์ตซึ่งจะมีโปรโตคอลแตกต่างกันไป รวมถึงคุณสมบัติที่แตกต่างกันไป เช่นวิธีการในการจัดการกับขนาดของหน้าต่างเมื่อเกิดความคับคั่ง เป็นต้น โดยเอเจนท์ที่สามารถใช้งานได้ จะต้องอยู่ในคลาสเอเจนท์ ซึ่งอยู่ในไฟล์ agent.cc agent.h และ ns-agent.tcl ซึ่งหากเราจะทำการเพิ่มเอเจนท์ก็ทำได้โดยเขียนโค้ดเพิ่มเข้าไปในไฟล์ดังกล่าว

TCP	a "Tahoe" TCP sender (cwnd = 1 on any loss)
TCP/Reno	a "Reno" TCP sender (with fast recovery)
TCP/Newreno	a modifi ed Reno TCP sender (changes fast recovery)
TCP/Sack1	a SACK TCP sender
TCP/Fack	a "forward" SACK sender TCP
TCP/FullTcp	a more full-functioned TCP with 2-way traffi c
TCP/Vegas	a "Vegas" TCP sender
TCP/Vegas/RBP	a Vegas TCP with "rate based pacing"
TCP/Vegas/RBP	a Reno TCP with "rate based pacing"
TCP/Asym	an experimental Tahoe TCP for asymmetric links
TCP/Reno/Asym	an experimental Reno TCP for asymmetric links
TCP/Newreno/Asym	an experimental Newreno TCP for asymmetric links
TCPSink	a Reno or Tahoe TCP receiver (not used for FullTcp)
TCPSink/DelAck	a TCP delayed-ACK receiver
TCPSink/Asym	an experimental TCP sink for asymmetric links
TCPSink/Sack1	a SACK TCP receiver
TCPSink/Sack1/DelAck	a delayed-ACK SACK TCP receiver
UDP	a basic UDP agent
RTP	an RTP sender and receiver
RTCP	an RTCP sender and receiver
LossMonitor	a packet sink which checks for losses
IVS/Source	an IVS source
IVS/Receiver	an IVS receiver

รูปที่ 2.13 Agent มาตรฐาน ใน ns2

CtrMcast/Encap	a "centralised multicast" encapsulator
CtrMcast/Decap	a "centralised multicast" de-encapsulator
Message	a protocol to carry textual messages
Message/Prune	processes multicast routing prune messages
SRM	an SRM agent with non-adaptive timers
SRM/Adaptive	an SRM agent with adaptive timers
Tap	interfaces the simulator to a live network
Null	a degenerate agent which discards packets
rtProto/DV	distance-vector routing protocol agent

รูปที่ 2.13 Agent มาตรฐาน ใน ns2 (ต่อ)

2.9.2 แอพพลิเคชั่น

หากเอเจนท์ทำหน้าที่เหมือนระดับชั้นทรานสปอร์ตดังนั้นแอพพลิเคชั่นก็จะทำหน้าที่เป็นระดับชั้นแอพพลิเคชั่น คือจะเป็น แอพพลิเคชั่นจะคอยสร้างทราฟฟิกต่างๆ โดยจะใช้บริการผ่านเอเจนท์ซึ่งแอพพลิเคชั่นที่จะคอยทำหน้าที่ในการสร้างทราฟฟิก ใน ns ประกอบไปด้วย 2 ลักษณะคือ

1) เป็นแอพพลิเคชั่นที่จำลองมาจากแอพพลิเคชั่นที่มีอยู่จริง
2) เป็นแอพพลิเคชั่นที่ปรากฏใน ns เพื่อใช้ในการสร้างทราฟฟิก

ในแบบที่ 1 นั้นบางแอพพลิเคชั่นก็ถูกจำลองมาได้เต็มหน้าที่การทำงานรวมถึงรายละเอียดต่างๆ แต่บางแอพพลิเคชั่นก็ถูกออกแบบมาแค่ที่จำเป็นที่ใช้ในการวิจัยและทดลองเท่านั้น ซึ่งการที่ แอพพลิเคชั่นจะอยู่เหนือ (on top of) เอเจนท์ทำให้การที่เราจะใช้แอพพลิเคชั่นใดๆ จะต้องทำการแปะไปบนเอเจนท์ โดยต้องเป็นเอเจนท์ที่เหมาะสมกับแอพพลิเคชั่นนั้นๆด้วย

2.9.3 TCP และแอพพลิเคชั่นที่อยู่เหนือ TCP

ตัวอย่างที่ 2. แสดงการสร้าง FTP แอพพลิเคชั่น ผ่าน TCP เอเจนท์

```
# Setup TCP connection
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
$ns connect $tcp $sink
$tcp set packetSize_ 512
$tcp set fid_ 1
# Setup FTP over TCP
```

ตัวอย่างที่ 2. แสดงการสร้าง FTP แอพพลิเคชั่น ผ่าน TCP เอเจนท์ (ต่อ)

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

ตัวอย่างเช่นในกรณีที่เราต้องการสร้างทราฟฟิกที่เป็น FTP ซึ่ง FTP (File transfer Protocol) เป็นแอพพลิเคชั่นที่ใช้ในการส่งไฟล์ในเน็ทเวิร์คซึ่งใช้โปรโตคอล TCP ใน ระดับชั้นทรานสปอร์ต ซึ่ง TCP (Transport Control Protocol) ในที่นี้ก็คือเอเจนท์โดย TCP โปรโตคอลใน ns นั้นมีความหลากหลายมาก เช่น Tahoe, Reno, NewReno etc. ดังรูปที่ 3.2 โดยเราสามารถสร้างเอเจนท์ได้โดยพิมพ์ดังนี้

```
set tcp [new Agent/TCP]
```

ซึ่งบรรทัดนี้เราจะทำการดึงตัวอย่าง TCP มาจากคลาส Agentเก็บไว้ที่ TCP โดยTCP จะทำหน้าที่เป็นเอเจนท์ สำหรับโหนดที่ส่งข้อมูล ซึ่งเป็นแบบ Tahoe ซึ่งเมื่อ เราสร้างเอเจนท์ แล้วเราต้องนำ เอเจนท์ไปแปะกับโหนดด้วยคำสั่ง

```
$ns attach-agent $n0 $tcp
```

เป็นอันเสร็จสิ้นการสร้างเอเจนท์ ฝั่งผู้ส่ง จากนั้น เราต้องทำการสร้างเอเจนท์ฝั่งผู้รับ โดยหากผู้ส่งเป็น TCP ฝั่งผู้รับก็ต้องเป็น TCP ด้วยจึงจะติดต่อกันได้

```
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $sink
```

ซึ่งจะเป็นการสร้าง TCP เอเจนท์ฝั่งผู้รับ โดยจะให้เอเจนท์ ชื่อ sink โดยจะทำหน้าที่รับ FTP และสร้างการตอบรับ ส่งกลับไปที่ผู้ส่งเพื่อเป็นการรับรองการส่งว่าแพ็คเกตดังกล่าวถึงมือผู้รับ ซึ่งเราต้องนำเอเจนท์ มาแปะกับโหนดที่เป็นผู้รับอีกที จากนั้นเราจะทำการเชื่อมต่อเอเจนท์ ทั้ง 2 เข้าด้วยกัน ด้วยคำสั่ง

```
$ns connect $tcp $sink
```

จากนั้นเราจะสร้าง FTP แอพพลิเคชั่นที่จะทำงานบน TCP เอเจนท์ ด้วยคำสั่ง

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

ซึ่งจะนำ FTP มาวางบน TCP อีกที

TCP เอเจนท์นั้นประกอบไปด้วยหลายพารามิเตอร์ซึ่งมีค่าดีฟอลท์อยู่ เช่นขนาดของแพ็คเกตซึ่งจะมีค่าเท่ากับ 1000 ไบท์ (ตามความจริงแล้วคือขนาดของเซ็กเมนท์ (segment)ในระดับชั้น ทรานสปอร์ต หรือ Maximum Segment Size: MSS) การเราต้องการเปลี่ยนแปลงขนาดแพ็คเกตก็ทำได้โดยใช้คำสั่ง

```
$tcp set packetSize_ 512
```

หากในการจำลอง ประกอบไปด้วยทราฟฟิกหลายสาย เราสามารถจำแนกได้ด้วย flow ID ซึ่งจะเอาไว้ใช้เวลาแยกสีของสาย ใน NAM ซึ่งจะกล่าวถึงในภายหลัง ด้วยคำสั่ง ***$tcp set fid_ 1***

2.9.4 UDP และแอพพลิเคชั่นที่อยู่เหนือ UDP

ตัวอย่างที่ 3. แสดงการสร้าง CBR (Constant Bit Rate) แอพพลิเคชั่น ผ่าน UDP เอเจนท์

```
# Setup UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n2 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$tcp set fid_ 2
# Setup CBR over UDP
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1000
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

เช่นเดียวกันการสร้าง FTP อยู่เหนือ TCP เราสามารถสร้าง UDP เอเจนท์ และ CBR แอพพลิเคชั่น ได้ในลักษณะเดียวกัน จากตัวอย่างที่ 3 เราจะเห็นว่ามีการสร้าง UDP ต้นทาง (Agent/UDP) และ UDP ปลายทาง (Agent/Null) และเรายังเห็นอีกว่าเราสามารถกำหนดอัตราเร็วในการส่งข้อมูล (rate_) และขนาดของแพ็คเกต (packetSize_) ได้เช่นเดียวกัน ซึ่งปกติแล้ว packetSize_ จะมีค่าดีฟอลท์ เป็น 1000 ไบท์อยู่แล้ว

อีกวิธีหนึ่งแทนที่เราจะกำหนดความเร็วในการส่งข้อมูล เราอาจจะกำหนดเป็นระยะเวลาระหว่างการส่งแต่ละแพ็คเกต แทนได้ ด้วยคำสั่ง

```
$cbr set interval_ 0.005
```

อีกพารามิเตอร์ ที่สำคัญของ UDP คือ random_ ซึ่งเป็นตัวกำหนดให้การส่งข้อมูลจะมีผลกระทบจากการรบกวนหรือไม่ซึ่งจะเกิดแบบสุ่มโดยปกติจะมีค่าดีฟอลท์ เป็น Off แต่เราสามารถทำการ on ได้โดยพิมพ์คำสั่ง

```
$cbr set random_ 1
```

นอกจากนี้ UDP ยังรองรับแอพพลิเคชั่นที่เป็นทราฟฟิก แบบอื่นๆ ซึ่งเรามารถกำหนดรูปแบบได้เอง เช่น exponential on-off source, Pareto on-off source และ trace-driver source ซึ่งเราสามารถทำการสร้างทราฟฟิก เหล่านี้ได้ด้วยคำสั่ง

```
set source [new Agent/Trafiic/Pareto]
set source [new Agent/Trafiic/Exponential]
```

ซึ่งเราจะต้องกำหนดพารามิเตอร์ต่างๆ ซึ่งไม่ว่าจะเป็น packetSize_ ซึ่งมีหน่วยเป็น byte, burstTime_ ซึ่งคือเวลาเฉลี่ยที่จะมีการส่งข้อมูล, idleTime_ คือเวลาเฉลี่ยที่จะไม่มีการส่งข้อมูล, rate_ คืออัตราการส่งข้อมูลเมื่อมีการส่งข้อมูล เป็นต้น ซึ่งเรากำหนดได้โดยใช้คำสั่งดังนี้

```
$source packetSize_ 700
$source burstTime_ 100ms
$source idleTime_ 150ms
$source rate_ 100k
```

หรือหากจะใช้แอพพลิเคชั่น แบบเทรซ-ไดรเวน (trace-driven X ซึ่งเราต้องระบุ ไฟล์เทรซซึ่งเราต้องทำการสร้างไฟล์เทรซก่อน ด้วยคำสั่ง

```
set trace [new Tracefile]
$trace filename <FileName>
set source [new Agent/Trafiic/Trace]
$source attach-tracefile $trace
```

ซึ่งเทรซไฟล์ จะมีเป็นเก็บข้อมูลข้อความที่เป็นเลขฐานสองโดยเก็บเวลาระหว่างแพ็คเกตในหน่วยมิลลิวินาที และขนาดแพ็คเกต ในหน่วยไบท์

2.10 การกำหนดช่วงเวลาของเหตุการณ์

NS เป็นการจำลองที่ไม่ต่อเนื่องในแง่ของเหตุการณ์คือเป็นเป็นแบบจำลองที่มีการทำงานในลักษณะไม่ต่อเนื่องของเวลา ซึ่ง TCL สคริปจะเป็นตัวกำหนดเวลาของการเกิดเหตุการณ์ ต่างๆ ซึ่งตั้งแต่บรรทัด

set ns [new Simulator] ก็เท่ากับเป็นการสร้างกำหนดการของเหตุการณ์ ซึ่งเหตุการณ์ ต่างๆ จะถูกจัดการเวลาการทำงาน ด้วยคำสั่งในรูปแบบดังนี้

```
$ns at <time> <event>
```

ซึ่งการจำลองจะเริ่มขึ้นเมื่อมีคาเริ่ม ns ด้วยคำสั่ง $ns run ซึ่งจะเริ่มที่เวลา0.0 ในหน่วยวินาที

ตัวอย่างในการกำหนดเวลาในการเริ่มและสิ้นสุดของการส่งข้อมูล ใน CBR และ FTP application นั้น สามารถทำได้ดังนี้

```
$ns at 0.2 "$cbr start"
$ns at 0.5 "$ftp start"
$ns at 3.0 "$ftp stop"
$ns at 5.0 "$cbr stop"
```

จากตัวอย่างจะกำหนดให้มีการส่ง CBR ทราฟฟิกเมื่อเวลา 0.2 ถึง 5.0 วินาที และส่ง FTP ตั้งแต่เวลา 0.5 ถึง 3.0 วินาที โดยเวลานี้เป็นเวลาของการจำลอง ไม่ใช่ของหน่วยประมวลผลกลาง ลิงค์ ระหว่างโหนด สร้างเอเจนท์ ให้โหนดต้นทางและปลายทางติดต่อกันได้ จากนั้นก็ทำการสร้างทราฟฟิกด้วย แอพพลิเคชันเพื่อจำลองการส่งข้อมูลรูปแบบต่างๆ ถึงจุดนี้จะเท่ากับว่าเรากำหนดทุกอย่างให้พร้อมที่จะทำการจำลองเหลือแต่การกำหนดเหตุการณ์ ในช่วงเวลาต่างๆ

ตัวอย่างที่ 4. Tcl สคริป

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the traffic trace file to record all events
set nd [open out.tr w]
$ns trace-all $nd
```

ตัวอย่างที่ 4. Tcl สคริป (ต่อ)

```
#Define a 'finish' procedure
proc finish {} {
        global ns nf nd
        $ns flush-trace
           #Close the NAM trace file
        close $nf
        close $nd
       #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n0-n2) to 10
$ns queue-limit $n0 $n3 10
#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
```

ตัวอย่างที่ 4. Tcl สคริป (ต่อ)

```
$ns connect $udp $null
$udp set fid_ 2
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

จากตัวอย่างที่ 4. จะเห็นการสร้างแบบจำลองโดยมีกำหนดการให้ มีการส่งข้อมูลของ CBR ตั้งแต่เวลา 0.1 ถึง 4.5 วินาที และให้มีการส่งข้อมูลของ FTP ตั้งแต่เวลา 1.0 ถึง 4.0 วินาที

2.11 การแสดงผลและการเทรซ

ใน ns นั้นที่ผ่านมาเราได้กำหนดเงื่อนไขและข้อกำหนดรวมถึงสิ่งต่างๆที่ควรมีในการจำลอง ซึ่งทำให้เราสามารถรัน ns เพื่อทำการจำลองได้โดยสมบูรณ์ หากแต่เราไม่อาจเห็นผลจากการจำลองได้ ns จึงมีกลไกในการบันทึกผลการจำลองออกมาในรูปแบบของไฟล์ข้อความ เรียกว่า เทรซไฟล์ ซึ่งเราสามารถนำผลจากการ trace นี้มาประมวลผลเพื่อวิเคราะห์หรือวาดกราฟ เพื่อหาข้อมูลที่ต้องการได้ นอกจากนั้นยังสามารถแสดงผลออกมาในรูปแบบของกราฟิกเพื่อให้ง่ายต่อการเข้าใจและเห็นภาพของการจำลองได้ชัดเจนมากยิ่งขึ้น

2.11.1 NAM

NAM หรือ Network Animator เป็นเครื่องมือที่ใช้แสดงผลการจำลองเน็ทเวิร์คในรูปแบบที่เป็นส่วนติดต่อกับผู้ใช้ทางด้านกราฟิก (GUI) โดยจะแสดง โหนด, ลิงค์, และ การไหลของข้อมูลรวมถึงแสดง queue นอกจากนี้ยังสามารถแสดงผลสิ่งอื่นๆในการจำลองได้อีกมากมาย

จากตัวอย่างที่ 4 จะเห็นว่ามีการสร้างโหนด ขึ้นทั้งสิ้น 4 โหนด โดยตามปกติแล้ว ตำแหน่งของ แต่ละโหนดจะถูกจัดวางแบบสุ่มแต่เราก็สามารถจะกำหนดตำแหน่งของโหนดได้เช่นเดียวกัน

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

จากโค้ด ดังกล่าวจะเป็นการกำหนดว่าลิงค์ระหว่าง 2 โหนดใดๆ จะวางอยู่ในลักษณะใดซึ่งต้องทำความเข้าใจว่าส่วนปลายของลิงค์ ทั้ง 2 ด้าน ก็คือโหนดดังนั้นก็คือการวางตำแหน่งโหนดนั่นเอง โดยคำสั่งจะมีรูปแบบคือบอก ลักษณะของลิงค์, โหนดแรก, โหนดที่สอง และตามด้วยตำแหน่งของโหนดที่สองเมื่อเทียบจาก node แรก จากตัวอย่างบรรทัดแรก right-down ก็หมายถึงให้ n2 อยู่ทางขวาล่างของ n0 นั่นเอง

หากเราไม่ทำการกำหนดตำแหน่งลงไป NAM จะกำหนดตำแหน่งให้เองแบบสุ่ม ซึ่งหากเมื่อแสดงผลออกแล้วไม่เป็นที่พอใจ เราสามารถปรับเปลี่ยนได้โดยคลิกที่ปุ่ม re-layout เพื่อให้ NAM ทำการสุ่มตำแหน่งอีกครั้ง หรือหากเราต้องการจัดวางตำแหน่งเองผ่าน GUI ก็สามารถทำได้โดยคลิกที่ปุ่ม Edit/View และทำการลากและวางตำแหน่งแต่ละโหนดไปยังตำแหน่งที่ต้องการ

NAM นั้นสามารถแสดงผลการไหลของแพ็คเกตได้ด้วย ซึ่งหากใน ns มีการสร้างหลายการไหล เราอาจจำแนกความแตกต่างของแต่ละการไหลใน NAM ด้วยสีของแพ็คเกตด้วยคำสั่ง

```
$ns color 1 Blue
$ns color 2 Red
```

ซึ่งเราต้องกำหนด flow ID (fid) เสียก่อน ซึ่งจากตัวอย่างที่ 4 เราได้กำหนดการไหล (flow) ของ tcp เป็น 1 และให้ udp เป็น 2 ซึ่งจากคำสั่งเราจะกำหนดให้ fid 1 สีน้ำเงินและ fid 2 เป็นสีแดง ซึ่งเมื่อเรา run NAM เราจะเห็นว่า แพ็คเกต ที่เป็น FTP จะมีสีน้ำเงิน โดย Acknowledgement (ack) ที่ตอบกลับก็เป็นสีน้ำเงินเช่นแต่จะขนาดสั้นทั้งเพราะด้วย แพ็คเกต Size_ ของ ack น้อยกว่าของ FTP

จากตัวอย่างเราต้องการให้เมื่อทำการรัน ns แล้วให้มีการรัน NAM เลย ซึ่งคำสั่งจะอยู่ในโพรซีเยอร์ finish โดยคำสั่งคือ

```
exec nam out.nam &
```

ซึ่งพารามิเตอร์ "&" ที่กำหนดให้มีการ run NAM ในลักษณะส่วนหน้า (foreground) และ out.nam เป็นชื่อไฟล์ NAM ที่เราต้องมีการสร้างขึ้นมาก่อนด้วยคำสั่ง

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

ซึ่งจะทำการสร้างไฟล์ NAM ให้มีชื่อว่า out.nam โดย "w" เป็นพารามิเตอร์ที่กำหนดให้มีการเปิดไฟล์ดังกล่าวเพื่อเขียนใหม่ หากมีข้อมูลเก่าอยู่จะลบของเดิมทิ้ง ส่วนบรรทัดต่อมาจะเป็นการกำหนดให้เก็บผลการเทรซทั้งหมดลงไฟล์ nam

NAM ยังมีความสามารถที่เรียกว่าจับภาพ (snapshot) โดยเปรียบเหมือนการคัดลอกรูปหน้าจอ (print screen) ของ NAM โดยจะทำการเซฟออกมาเป็นภาพนิ่ง

นอกจากนี้สิ่งต่างๆที่เราสามารถกำหนดได้ใน NAM มีดังนี้

- สีของโหนดในกรณีที่เราต้องการให้ n0 มีสีเหลือง ทำได้โดยพิมพ์

```
$n0 color yellow
```

- สีของลิงค์ในกรณีที่เราต้องการให้ลิงค์ที่ต้องการมีสีเขียว ทำได้โดยพิมพ์

```
$ns duplex-link-op $n0 $n2 color green
```

- รูปร่างของโหนดปกติแล้วจะเป็นวงกลม แต่เรากำหนดได้โดยพิมพ์

```
$n0 shape box
```

โดย box จะมีรูปร่างเป็นสี่เหลี่ยม แต่เราอาจเปลี่ยนเป็น circle หรือ hexagon ก็ได้

- กำหนด mark ที่โหนดโดยเราอาจกำหนดเพื่อเน้นจุดเด่นให้ node ดังกล่าว หรือเพื่อบอกให้สนใจโหนดนั้นโดยต้องระบุช่วงเวลาด้วย ซึ่งทำได้โดยพิมพ์

```
$ns at 1.0 "$n0 add-mark m0 blue box"
$ns at 3.0 "$n0 delete-mark m0"
```

- แปะลาเบล (label) โดยเป็นลักษณะของตัวอักษร (text) ปรากฏขึ้นบนจอในช่วงเวลาที่กำหนด โดยพิมพ์

```
$ns at 1.0 "$n0 label \"Active Node\""
```

ซึ่งจะเป็นการแปะลาเบลไปที่โหนดแต่หากเราต้องแปะลาเบลที่ลิงค์ก็ทำได้โดยพิมพ์

```
$ns duplex-link-op $n0 $n2 label \"Active Input Link\""
```

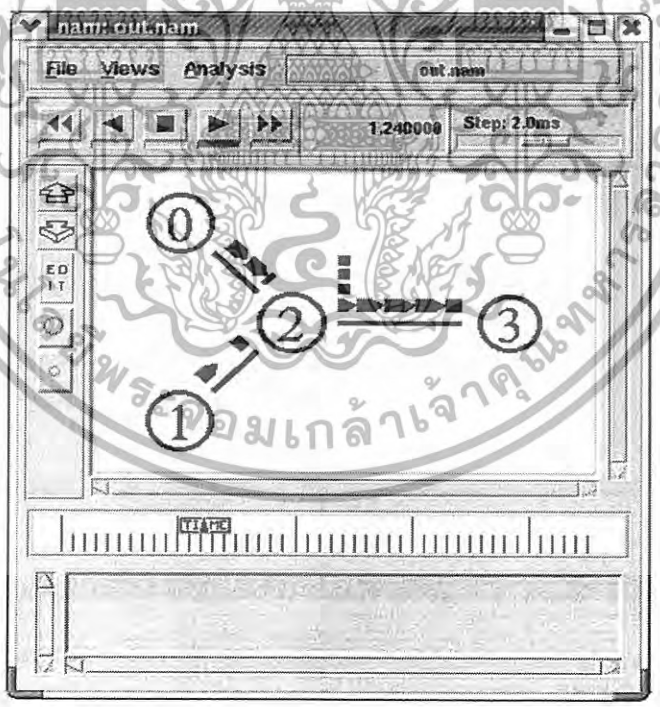
- แปะตัวอักษรโดยจะไปอยู่ในตำแหน่งล่างของ NAM ซึ่งอาจจะใช้ในการอธิบายถึง event ที่เกิด ในช่วงเวลาดังกล่าว ซึ่งทำได้โดยพิมพ์

```
$ns at 1.0"$ns trace-annotate \"packet Drop\""
```

- ทิศทางของคิวเราจะกำหนดทิศทางของการต่อคิวโดย จะเริ่มที่ 0.0 คือ 0 องศา ทางขวา จากนั้นจะเพิ่ม 90 องศา ทุกๆ 0.5 ทิศทางทวนเข็มนาฬิกา ตัวอย่างเช่น

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

ก็จะได้ทิศทางของคิวเป็นแกนตั้งในทิศทางขึ้นบน
ซึ่งจากตัวอย่างที่ 4 เราจะได้การแสดงผลของ NAM ดังนี้

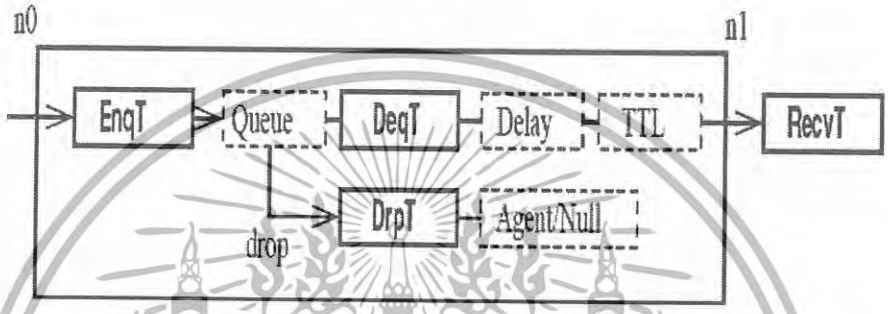


รูปที่ 2.14 ตัวอย่าง GUI ของ NAM

2.11.2 การตรวจสอบ (Tracing)

อย่างที่ได้กล่าวไปในตอนต้นว่า NS นั้น มีการเก็บผลการ simulation ได้ในรูปของเทรซไฟล์ ที่เป็นแอสกี (ASCII) ซึ่งจะเป็นเหมือนการติดตามผลแต่ละแพ็คเกตที่เกิดขึ้นในการจำลอง โดยจะมีรูปแบบการเก็บข้อมูลที่มีโครงสร้างที่แน่นอน

โดยเมื่อเราทำการเทรซตัว NS จะทำการแทรกเทรซออปเจ็ค (tracing object) ไปในลิงค์เพื่อบันทึกผล ซึ่งออปเจ็คที่แทรกเข้าไปประกอบไปด้วย 4 ออปเจ็คคือ EnqT, DeqT, RecvT และ DrpT ดังรูปที่ 2.15



รูปที่ 2.15 แสดงเทรซออปเจ็ค ในซิมเพล็กซ์ลิงค์

ซึ่งออปเจ็คแต่ละตัวจะทำหน้าที่ดักจับแพ็คเกตที่มาถึงออปเจกนั้นๆ ซึ่งจะเก็บรายละเอียดต่างๆของแพ็คเกตนั้น โดยการที่แพ็คเกตจะมาถึงออปเจ็คนั้นได้ ก็สามารถบ่งบอกได้ดังนี้

EnqT คือ แพ็คเกตที่จะถูกปล่อยออกจากตัวผู้ส่ง และทำการเข้าคิวเพื่อรอที่จะส่งออกไปใน link

DeqT คือ แพ็คเกตที่ปล่อยออกจากผู้ส่งไปในลิงค์แล้ว โดยจะเข้าคิวและรอคิวจนกระทั่งถึงคิวที่ตัวเองจะสามารถถูกส่งออกไปได้

DrpT คือ แพ็คเกตที่ถูกดรอป (drop) ทิ้ง คือไม่สามารถถูกส่งออกจากต้นทางไปได้ อาจจะเนื่องจากคิวบัฟเฟอร์ (Queue buffer) ของลิงค์เต็ม (Overflow)

RecvT คือ แพ็คเกตที่ถึงมือผู้รับเรียบร้อยแล้ว

อย่างไรก็ดี NS อนุญาตให้มีการดึงข้อมูลได้มากกว่าการเทรซซึ่งใช้วิธีคิวมอนิเตอร์ รูปแบบของเทรซไฟล์ ที่ได้จากการเทรซจะมีการเข้ารหัสแบบแอสกีซึ่ง มนุษย์สามารถอ่านแล้วเข้าใจได้ โดยจะเก็บเป็น 1 บรรทัดต่อ 1 แพ็คเกตที่ออปเจ็คนั้นๆจับได้ โดยแต่ละบรรทัดจะถูกแบ่งออกเป็น 12 ฟิลด์ (Field) ซึ่งแต่ละฟิลด์จะมีความหมายของตัวเองดังนี้

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id

รูปที่ 2.16 แสดงโครงสร้างของเทรซไฟล์

จากรูปที่ 2.16 สามารถอธิบายโครงสร้างของเทรซไฟล์ได้ดังนี้

1. Event Type จะบอกชนิดของเหตุการณ์ (event) ที่เกิดขึ้น โดยจะขึ้นอยู่กับว่า ถูกดักจับได้ด้วยออปเจกใด โดยจะเป็นไปได้ 4 ชนิดคือ +, - , d และ r ซึ่งก็คือ enqueue, dequeue, drop และ receive ตามลำดับ
2. Time จะเป็นเวลาที่เกิดเหตุการณ์ตามฟิลด์ที่ 1 โดยจะเป็น simulation time
3. From Node จะบอกถึงโหนดต้นทางของลิงค์ที่ทำให้เกิดเหตุการณ์ตามฟิลด์ที่ 1
4. To Node จะบอกถึงโหนดปลายทางของลิงค์ที่ทำให้เกิดเหตุการณ์ตามฟิลด์ที่ 1
5. แพ็คเกต Type บอกชนิดของแพ็คเกตดังกล่าว เช่น CBR จะมีชนิด (type) เป็น CBR หรือ TCP จะมีชนิดเป็น tcp เป็นต้น
6. แพ็คเกต Size จะบอกถึงขนาดของแพ็คเกตดังกล่าว
7. Flag บางแอพพลิเคชั่นจะมีข้อมูลส่วนนี้
8. Flow ID เป็นหมายเลขประจำ flow นั้นๆ โดยจะเป็นไปตามที่เรากำหนด fid
9. Source Address เป็นต้นทางที่ส่งแพ็คเกตซึ่งอยู่ในรูปแบบ node.port
10. Destination Address เป็นปลายทางที่รับแพ็คเกตซึ่งอยู่ในรูปแบบ เดียวกับต้นทาง
11. Sequence Number โดยจะเป็นเลขลำดับแพ็คเกตที่ปรากฏในเน็ตเวิร์คเลเยอร์ (Network Layer) ซึ่งถึงแม้ว่า UDP จะไม่มีการใช้เลขลำดับแพ็คเกตแต่ NS ก็ได้กำหนดให้มีเพื่อใช้ในการวิเคราะห์
12. แพ็คเกต ID เป็นหมายเลขประจำแพ็คเกตนั้นๆ

ซึ่งจากตัวอย่างที่ 4 จะมีการสร้าง trace file โดยให้บันทึกผลทั้งหมดที่เกิดขึ้นด้วยคำสั่ง

```
set nd [open out.tr w]
$ns trace-all $nd
```

ซึ่งเราจะทำการเปิดไฟล์ out.tr เพื่อเก็บผลการเทรซโดย "w" เป็นพารามิเตอร์ที่กำหนดให้มีการเปิดไฟล์ดังกล่าวเพื่อเขียนใหม่ หากมีข้อมูลเก่าอยู่จะลบของเดิมทิ้ง

ตาราง 2.2 แสดงไฟล์ out.tr

out.tr ×

```
+ 0.1 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
- 0.1 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.108 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
- 0.108 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
r 0.114 1 2 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.114 2 3 cbr 1000 ------- 2 1.0 3.1 0 0
- 0.114 2 3 cbr 1000 ------- 2 1.0 3.1 0 0
+ 0.116 1 2 cbr 1000 ------- 2 1.0 3.1 2 2
- 0.116 1 2 cbr 1000 ------- 2 1.0 3.1 2 2
r 0.122 1 2 cbr 1000 ------- 2 1.0 3.1 1 1
+ 0.122 2 3 cbr 1000 ------- 2 1.0 3.1 1 1
- 0.122 2 3 cbr 1000 ------- 2 1.0 3.1 1 1
+ 0.124 1 2 cbr 1000 ------- 2 1.0 3.1 3 3
- 0.124 1 2 cbr 1000 ------- 2 1.0 3.1 3 3
r 0.13 1 2 cbr 1000 ------- 2 1.0 3.1 2 2
+ 0.13 2 3 cbr 1000 ------- 2 1.0 3.1 2 2
- 0.13 2 3 cbr 1000 ------- 2 1.0 3.1 2 2
+ 0.132 1 2 cbr 1000 ------- 2 1.0 3.1 4 4
- 0.132 1 2 cbr 1000 ------- 2 1.0 3.1 4 4
```

จากตัวอย่างเราจะเห็นว่าเราได้ใช้คำสั่งในการบันทึกผลทุกอย่างที่เกิดขึ้นในการจำลอง (Simulate) ซึ่งรูปแบบคำสั่งจะอยู่ในรูปของ $ns trace-all <filename> แต่เราอาจทำการเทรซแค่ส่วนหนึ่งของการจำลอง ได้ โดยคำสั่ง $ns trace-queue เช่น

```
$ns trace-queue $n2 $n3 $file1
```

ซึ่งฟิลด์ที่ 1 ซึ่งเป็นชื่อไฟล์นั้น จะเก็บข้อมูลเฉพาะการเทรซของลิงค์ ระหว่าง $n2 และ $n3 เท่านั้น ซึ่งแน่นอนว่าเราต้องมีการสร้างลิงค์ระหว่าง $n2 และ $n3 ก่อนถึงจะสามารถใช้คำสั่งนี้ได้ ซึ่งเราสามารถใช้วิธีนี้เช่นเดียวกันกับ NAM โดยใช้คำสั่ง namtrace-queue แทน

2.12 การนำไฟล์เทรซมาประมวลผล

การจำลองใน ns นั้นจะเป็นการจำลองเน็ตเวิร์คภายใต้เงื่อนไขต่างๆ โดยจุดประสงค์หลักของการทดลองก็เพื่อทดสอบหรือทำการวิจัยอะไรบางอย่าง ซึ่งสิ่งที่ผู้ใช้ต้องการได้จากการจำลองก็คือผลการทดลอง โดยผู้ใช้แต่ละคนย่อมต้องการผลลัพธ์ที่แจกจ่างกันไปเพื่อนำไปใช้หรือการวิเคราะห์ หาก trace file เปรียบเหมือนผลการทดลอง ก็เป็นเหมือนข้อมูลดิบ ที่หากเราจะนำไปใช้ให้เกิดประโยชน์ต้องผ่านกระบวนการตีความหมาย จำแนก หรือประมวลผลบางอย่างเสียก่อน

2.12.1 การประมวลผล

จากที่ผ่านมาเราจะเห็นได้ว่า NS สามารถให้ข้อมูลจากการจำลองได้ค่อนข้างละเอียด ซึ่งตามจริงแล้วเราให้ความสำคัญหรือสนใจแค่บางค่าเท่านั้น หรือบางค่าอาจไม่ได้จากเทรซไฟล์โดยตรงต้องนำมาคำนวณหรือผ่านกระบวนการบางอย่างเสียก่อนจึงจะได้ข้อมูลที่ต้องการออกมา ซึ่งนักวิจัยต้องการข้อมูลหลากหลายรูปแบบ เพื่อจะนำมาวิเคราะห์เพื่อจุดประสงค์ที่แตกต่างกันไป เราจึงจำเป็นต้องมีวิธีการที่จะดึงเฉพาะข้อมูลที่ต้องการออกมาจากข้อมูลที่ได้จาก NS ที่มากมายนั้น เพื่อนำมาวิเคราะห์หรือประมวลผลต่อไป

ดังนั้นเราควรเขียนโปรแกรมด้วยภาษาอะไรซักอย่าง เพื่อจะจัดการกับไฟล์ข้อมูลเหล่านั้น ซึ่งเราควรหาเครื่องมือ (Tool) ที่มีอยู่แล้ว และใช้งานได้ฟรี และปรับปรุงแก้ไขได้ง่าย และสามารถรันได้บนระบบปฏิบัติการ (Operating System) ที่หลากหลาย เพื่อที่จะใช้เขียนสคริป (script) สั้นๆ เพื่อดึงข้อมูลที่ต้องการออกมาจากไฟล์ ซึ่งสามารถรันได้โดยไม่ต้องผ่านกระบวนการแปลภาษา (compile)

2.12.1.1 การใช้ grep

grep เป็นคำสั่งใน UNIX ที่ใช้ทำการกรอง (filter) ข้อมูลในไฟล์ ซึ่งเราจะทำการสร้างไฟล์ใหม่โดยให้ข้อมูลในไฟล์นั้นมีแต่บรรทัดที่เราต้องการจากไฟล์เดิม ตัวอย่างเช่น จากเทรซไฟล์ซึ่งมีหลากหลายชนิดของแพ็คเกตแต่เราต้องการดูแพ็คเกตแค่บางชนิดเท่านั้น สมมติเราต้องการดูแต่ tcp แพ็คเกตที่ส่งจาก n0 ไป n2 เท่านั้น เราก็ทำได้โดยพิมพ์คำสั่ง

```
grep " 0 2 tcp" out.tr > tcp.tr
```

ซึ่งในไฟล์ tcp.tr จะมีข้อมูลที่ในบรรทัดมี "0" และ "2" และ "tcp" ภายในเท่านั้น ซึ่งจะไปค้นหาบรรทัดที่มีเงื่อนไขดังกล่าวจากไฟล์ out.tr และดึงเอาเฉพาะบรรทัดที่ผ่านเงื่อนไขมาเก็บไว้ที่ไฟล์ tcp.tr หรือหากเราต้องการดึงเฉพาะบรรทัดที่ขึ้นต้นด้วย "r" ก็ทำได้ด้วยคำสั่ง

```
grep "^r" out.tr > recv.tr
```

จะสังเกตว่าเรา "^r" แทนที่จะเป็น "r" เพราะสิ่งที่เราต้องการคือชนิดของเหตุการณ์ (event type) ที่เป็น "r" ซึ่งอยู่ตัวแรกสุด แต่ถ้าเราใช้ "r" ในคำสั่ง grep เราจะได้ cbr แพ็คเกต มาด้วย

เราทราบแล้วว่า grep เป็นคำสั่งที่ใช้ดึงบรรทัดที่ต้องการออกมา เราสามารถนำมาประยุกต์ ใช้ grep ดึงข้อมูลจากการ grep อีกทีได้ เช่น

```
grep "^r" out.tr | grep " 0 2 tcp > recvTcp.tr
```

โดยผลจากการ grep ครั้งแรกจะกลายเป็นกลายเป็นอินพุตให้ในการ grep ครั้งที่สอง

2.12.1.2 การประมวลผลด้วย awk

awk จะมีการทำงาน (operation) ต่างๆเตรียมไว้ให้ในการประมวลผลข้อมูลที่ดึงมา โดยจะใช้ในการดึงข้อมูลในลักษณะของ column โดยต้องระบุ column ที่ต้องการใช้และนำมาทำการประมวลผล เช่นการหาค่าเฉลี่ย หรือการคำนวณทางคณิตศาสตร์ เป็นต้น การรันไฟล์ awk เราต้องพิมพ์คำสั่งบนเทอร์มินอล (terminal) ด้วยคำสั่งในรูปแบบดังนี้

```
awk -f <awk file> <original file>
```

โดย original file ก็คือไฟล์ต้นฉบับที่เราจะให้ awk สคริป ดึงค่าไปประมวลผลนั่นเอง เราสามารถเขียน awk สคริปง่ายๆ ในการหาค่าเฉลี่ยในคอลัมน์ (column) ที่ต้องการด้วย

```
BEGIN {FS = "\t"} {nl++} {s=s+$2} END {print"average:"
s/nl}
```

จากตัวอย่าง FS="\t" หมายถึงคอลัมน์ถูกแบ่งด้วย tab หากแบ่งด้วยช่องว่าง (space) ก็ให้เท่ากับ " " โดย BEGIN และ END เป็นเหมือนการวนลูปตรวจสอบไฟล์ไปทีละบรรทัด ดังนั้นโค้ดที่อยู่ระหว่างนั้น ก็เป็นเหมือนลูปที่จะทำทุกครั้งที่อ่านแต่ละบรรทัด ซึ่ง nl++ หมายถึงเมื่ออ่านบรรทัดหนึ่งจะเพิ่มค่า nl ไป1 นั่นคือ nl จะให้เป็นจำนวนบรรทัดนั่นเอง ซึ่งสมมติ awk สคริป $2 เป็นลำดับคอลัมน์ของไฟล์ต้นฉบับที่เราต้องการจะดึง ซึ่งในที่นี้ก็คือคอลัมน์ที่ 2 ซึ่งเราจะเอาคอลัมน์ที่ 2 ของแต่ละบรรทัดมาบวกกันแล้วหารด้วยจำนวนบรรทัดซึ่งก็คือ ค่าเฉลี่ยนั่นเอง ซึ่งสมมติ awk สคริปนี่ชื่อ avg.awk และต้องการดึงข้อมูลจากไฟล์ out.tr ก็ต้องรัน awk ด้วยคำสั่ง awk –f avg.awk out.tr

2.12.1.3 การประมวลผลด้วย Perl

Perl ย่อมาจาก Practical Extraction and Reporting Language โดย PERL เป็นภาษาโปรแกรมขั้นสูง (High level programming language) ซึ่งเกิดจากการรวมกันของ C-shell, awk และภาษาโปรแกรมอื่นๆ ซึ่งทำให้ PERL มีคุณสมบัติที่ยืดหยุ่นและหลากหลาย และที่สำคัญ Perl นั้นไม่ต้องเสียค่าลิขสิทธิ์ ทำให้สามารถใช้งานได้ฟรี

Perl เป็นตัวช่วยให้สามารถกรองและประมวลผล (process) ข้อมูลที่ต้องการจากแอสกีไฟล์ได้อย่างง่ายๆ โดยภาษานี้คิดค้นขึ้นโดย Larry Wall ซึ่งมีแนวคิดหลักจากการที่ต้องการตัวช่วยให้งานของผู้ดูแลระบบ (System Administrator) นั้นง่ายขึ้น ซึ่งการที่ perl มีข้อดีมากมายดังที่ได้กล่าวบวกกับคุณสมบัติที่มี ทำให้ perl เป็นภาษาที่ใช้ในการจัดการข้อมูลที่มีการใช้กันอย่างแพร่หลายในวงการเว็บและอินเตอร์เน็ต (internet)

Perl เป็นการแปลภาษาทีละบรรทัด (interpreted programming) ซึ่งมีการนำไปใช้อย่างหลากหลาย แต่จุดประสงค์หลักของการใช้ perl จะมุ่งเน้นไปที่การดึงข้อมูล, สืบค้น รวมถึงการรายงาน ซึ่งข้อดีบางประการของ perl มีดังนี้

- สามารถเขียน (implement) ออกมาเป็นโปรแกรมเล็กได้ง่าย ที่จะใช้ในการกรอง และดึงข้อมูลออกมาจากไฟล์ตัวอักษร (text file)
- สามารถทำงานได้บนระบบปฏิบัติการที่หลากหลาย โดยไม่จำเป็นต้องแก้โค้ด
- การบำรุงรักษา (maintain) และการตรวจสอบโค้ด (debug) ใน perl ทำได้ง่ายกว่าโปรแกรมอื่น
- Perl สามารถใช้งานร่วมกับ gnu บนเว็บได้

ในตัวอย่างจะแสดงตัวอย่าง Perl สคริปที่ใช้ในการหาเลขลำดับแพ็คเกตของ CBR แพ็คเกต ที่ปรากฏในเทรซไฟล์

ตัวอย่างที่ 5. Perl สคริป ในการหาเลขลำดับแพ็คเกตจาก trace file

```
open(cmd,">sqs");
open(infile, "out.tr") or die "couldn't open the file";
$line_no=0;
while ($line = <infile>) {
++$line_no;
@entry = split(" ", $line);
if(($entry[0] eq "+")&&($entry[2] eq "0")&&($entry[3] eq
"2")&&($entry[4] eq "cbr"))
{
     print cmd "$entry[1] $entry[10] \n";
}}
close(cmd);
close(infile);
```

จะเห็นว่ามีการเปิดสร้างไฟล์ sqs ขึ้นมาเพื่อเก็บผลลัพธ์จากการกรองข้อมูลจากเทรซไฟล์ที่ชื่อว่า out.tr ซึ่งจะทำการอ่านจากบรรทัดแรกสุดไปเรื่อยๆ และจะแบ่งแยกแต่ละฟิลด์ในบรรทัด นั้นด้วยช่องว่าง " " และเก็บไว้ในเวกเตอร์ (vector) ที่ชื่อว่า entry[] หากเจอบรรทัดใดที่ฟิลด์ที่ 1($entry[0]) มีค่าเป็น "+", ฟิลด์ที่ 3 ($entry[2]) มีค่าเป็น "0", ฟิลด์ที่ 4 ($entry[3]) มีค่าเป็น "2" และ ฟิลด์ที่ 5 ($entry[4]) มีค่าเป็น "cbr" จะทำการดึงฟิลด์ที่ 2 ($entry[1]) และ ฟิลด์ที่ 11 ($entry[10]) ของบรรทัดนั้นมาเก็บไว้ที่ไฟล์ sqs ซึ่งเมื่อเทียบกับโครงสร้างของเทรซไฟล์ที่มีอยู่ 12 ฟิลด์แล้ว ฟิลด์ที่ 1, 3, 4 และ 5 ที่นำมาเทียบกับเงื่อนไขคือฟิลด์ของ Event Type, From Node, To Node และ แพ็คเกต Type ตามลำดับ ซึ่งเมื่อดูจากเงื่อนไขในตัวอย่างที่ 5 จะสามารถแปลความหมายได้ว่า "ต้องการแพ็คเกตที่จะถูกส่งออกไปเพื่อเข้าคิว ที่ถูกส่งออกจาก n0 ไปถึง n2 โดยต้องเป็น CBR แพ็คเกต" โดยจะเอาเวลาและเลขลำดับแพ็คเกตของแพ็คเกตดังกล่าวมาบันทึกลงไฟล์ที่ชื่อว่า sqs เพื่อนำไปใช้ต่อไป โดยผลจากการบันทึกลงไฟล์ sqs ด้วยคำสั่ง

```
print cmd "$entry[1] $entry[10] \n"
```

จะได้ผลออกมาในรูปแบบ <time> <sequence number> โดยจะคั่นระหว่าง 2 ค่า ด้วยช่องว่าง " " และคั่นระหว่างแพ็คเกตด้วย newline "\n"

2.12.2 การ plot graph ด้วย gnuplot

Gnuplot เป็นโปรแกรมตระกูล Gnu ที่ใช้ในการ plot ข้อมูล โดย Gnuplot มีการใช้กันอย่างแพร่หลายเนื่องจากฟรี แบะทำงานได้บนหลาย platform ที่สำคัญมี function เตรียมไว้ให้ค่อนข้างครบถ้วนในการที่จะวาดกราฟ (plot graph)

ตามหลักแล้วเราสามารถสั่งให้ Gnuplot ทำงานด้วยคำสั่ง plot (ชื่อไฟล์) ซึ่งไฟล์ดังกล่าวจะประกอบไปด้วยข้อมูลจำนวน 2 คอลัมน์ แทนข้อมูลในแกน x และแกน y ซึ่งเราจะนำข้อมูล x และ y ดังกล่าวของแต่ละบรรทัดมาทำการกำหนดจุดบนกราฟแต่เพื่อความสะดวกเราจะให้ Gnuplot ทำงานใน Perl สคริปไปเลยเพื่อให้เมื่อประมวลผลเสร็จก็ทำการวาดกราฟได้ทันที โดยปกติเราจะเอาไฟล์ผลลัพธ์จากการประมวลผลด้วย perl สคริปมาใช้เป็นข้อมูลที่จะเอามาใช้ในการวาดกราฟ ดังนั้นโค้ดส่วนที่เป็น Gnuplot เราจะนำไปไว้ใต้บรรทัดของ perl สคริป

โดยเริ่มแรกเราจะต้องทำการเรียก Gnuplot ด้วยคำสั่ง

```
plot_graph();
```

ซึ่งจะเป็นเหมือนการเรียกฟังก์ชั่นในการวาดกราฟขึ้นมา จากนั้นเราก็สามารถใส่โค้ดต่างๆ ของ Gnuplot โดยต้องอยู่ในขอบเขตของ

```
sub plot_graph {.....}
```

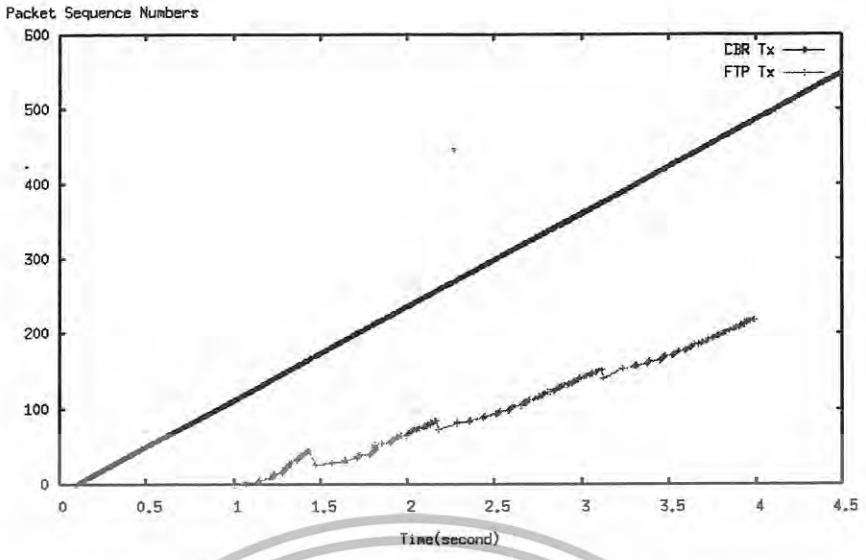
จากนั้นหากเราจะทำการพิมพ์คำสั่งใดๆ ที่ปกติต้องพิมพ์ที่เทอร์มินอลให้พิมพ์ในลักษณะนี้

```
print cmd "plot seq u 1:2 t \"CBR Tx\" w linespoints";
```

ซึ่งคำสั่งนี้หมายถึงให้ทำการวาดกราฟจากไฟล์ชื่อ seq โดยให้เป็นชื่อของจุดคือ CBR TX c และให้ทำการลากเส้นระหว่างจุดต่างๆที่ทำการ plot ด้วย

ตัวอย่างที่ 6. Gnuplot สคริป

```
plot_graph();
sub plot_graph
{
    $output_type=0;
    $outname="data1.eps";
    $x_label="Time(second)";
    $y_label="Packet Sequence Number";
    open(cmd, ">cmd.tmp");
    if($output_type==0){
   print cmd "set term x11 \n";
    }else{
   print cmd "set term post portrait color solid \'Roman\'
11\n";
   print cmd "set output \"$outname\"\n";
   print cmd "set size 1.0 , 0.5\n"
    }
    print cmd "set xlabel \"$x_label\" \n";
    print cmd "set ylabel \"$y_label\"\n";
    $output_files="sqs";
    $output_files1="sqr";
    print cmd "plot \"$output_files\" u 1:2 t \"CBR Tx\" w
linespoints,\"$output_files1\" u 1:2 t \"CBR Rx\" w
linespoints   ";
    print cmd "\n";
    close(cmd);
    system "gnuplot -persist cmd.tmp";
}
```



รูปที่ 2.17 กราฟจากการวาดด้วย Gnuplot

2.12.3 ตัวอย่างการประมวลผล

ในหัวข้อนี้จะแดงถึงตัวอย่าง perl สคริปหรือ awk สคริปที่ใช้ในการนำเทรซไฟล์ไปประมวลผล โดยจะแสดงตัวอย่างการประมวลผลเพื่อให้ได้ผลลัพธ์ออกมาเป็นข้อมูลที่ต้องการสำหรับนักวิจัยที่จะนำข้อมูลไปใช้ในการวิจัยและวิเคราะห์ต่อไป โดยการแสดงผลจะออกมาใน 2 รูปแบบ คือในรูปแบบของข้อมูลเลข หรือในลักษณะของกราฟโดยใช้ Topology จากตัวอย่างที่ 4

2.12.3.1 หาเลขลำดับแพ็คเกต

Sequence number (seq no.) เป็นหมายเลขลำดับของแพ็คเกตที่มีการส่งออกจากผู้ส่ง โดยจะมีเพื่อการซิงค์โคนัส (Synchronous) ระหว่างผู้รับและผู้ส่ง เพื่อแน่ใจว่ารับแพ็คเกตครบถ้วนสมบูรณ์

โดยหากผู้รับสามารถรับข้อมูลได้ทุกแพ็คเกตจากที่ผู้ส่งทำการส่ง ดังนั้นหมายเลขลำดับของแพ็คเกตที่ผู้ส่งทำการส่งออกกับที่ผู้รับทำการรับได้ต้องตรงกัน

โดยจากเทรซไฟล์จะมีฟิลด์ที่เก็บค่าของหมายเลขลำดับของแพ็คเกตอยู่แล้ว ซึ่งคือฟิลด์ที่ 11 ซึ่งเราสามารถใช้ perl ดึงค่าจากฟิลด์นั้นมาใช้ได้เลยโดยไม่ต้องผ่านการคำนวณใดๆ

ตัวอย่างที่ 7. แสดงโค้ดของ plot_cbr_seq.pl

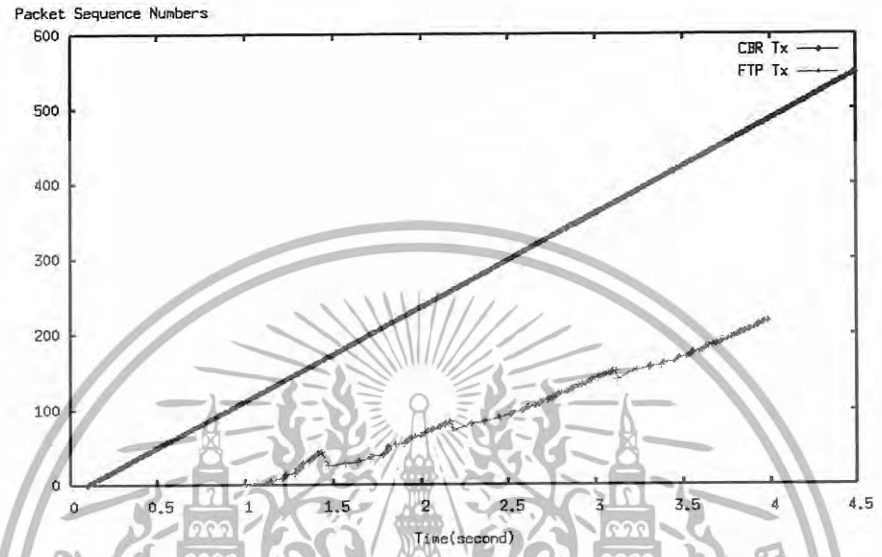
```
open(cmd,">sqs");
open(cmd1,">sqr");
open(infile, "out.tr") or die "couldn't open the file";
$line_no=0;
while ($line = <infile>) {
```

ตัวอย่างที่ 7. แสดงโค้ดของ plot_cbr_seq.pl (ต่อ)

```
        ++$line_no;
          @entry = split(" ", $line);
         if(($entry[0] eq "+")&&($entry[2] eq "0")&&($entry[3]
eq "1")&&($entry[4] eq "cbr")) {
             print cmd "$entry[1] $entry[10] \n";
           }
           if(($entry[0] eq "r")&&($entry[2] eq
"_3_")&&($entry[6] eq "cbr")) {
             @tmp=split(/[\[\]]+/,$entry[1]);
             print cmd1 "$entry[1] $tmp[1]\n";
           }
       }
       close(cmd);
       close(cmd1);
       close(infile);
       # call plot_graph
       plot_graph();
       sub plot_graph
       {
       $output_type=0;
       $outname="data1.eps"
$x_label="Time(second)";
$y_label="Packet Sequence Number";
        open(cmd, ">cmd.tmp");
       if($output_type==0){
             print cmd "set term x11 \n";
        }else{
        print cmd "set term post portrait color solid
\'Roman\' 11\n";
print cmd "set output \"$outname\"\n";
        print cmd "set size 1.0,0.5\n"
     }
     print cmd "set xlabel \"$x_label\" \n";
     print cmd "set ylabel \"$y_label\"\n";
     $output_files="sqs";
     $output_files1="sqr";
     print cmd "plot \"$output_files\" u 1:2 t \"CBR Tx\"
w linespoints,\"$output_files1\" u 1:2 t \"CBR Rx\" w
linespoints   ";
     print cmd "\n";
     close(cmd);
     system "gnuplot -persist cmd.tmp";
}
```

จากตัวอย่างที่ 7. แสดงกราฟของหมายเลขลำดับของแพ็คเกตของทั้งที่ผู้ส่งทำการส่งและที่ผู้รับทำการรับได้ ซึ่งเราสามารถดูกราฟได้หากหมายเลขลำดับของแพ็คเกตฝั่งผู้รับมีครบถ้วนแปลว่ารับได้หมดทุก

แต่หากเป็นแพ็คเกตที่เป็น tcp ซึ่งจะมีการตอบกลับ (acknowledge (ack)) เพื่อการันตีการส่ง หากมีการสูยหาย (loss) เกิดขึ้นทำให้แพ็คเกตไปไม่ถึงมือผู้รับ tcp จะมีกลไกการส่งข้อมูลใหม่ (retransmitted) ซึ่งสามารถดูจากกราฟได้ว่าจะมีการส่งแพ็คเกตที่มี หมายเลขลำดับของแพ็คเกตซ้ำ ซึ่งกลไกดังกล่าวจะไม่มีใน udp



รูปที่ 2.18 แสดงหมายเลขลำดับของแพ็คเกตของ CBR และ FTP

2.12.3.2 หาเวลาหน่วงจากต้นทางถึงปลายทาง

ในส่วนนี้จะกล่าวถึง ระยะเวลาของความหน่วงที่เกิดขึ้นระหว่างการส่งข้อมูลจากต้นทางถึงปลายทาง ซึ่งจะหาได้จาก

เวลาที่แพ็คเกตถึงมือผู้รับ – เวลาที่แพ็คเกตนั้นออกจากผู้ส่ง

ซึ่งเราสามารถดึงค่าฟิลด์ที่ 2 จากเทรซไฟล์มาใช้ได้ โดยต้องเป็นเวลาของแพ็คเกตที่มีหมายเลขลำดับของแพ็คเกตเหมือนกัน

ตัวอย่างที่ 8. แสดง perl สคริป ในการ plot graph ent to end delay

```
open(cmd,">sqs");
open(cmd1,">sqr");
open(infile, "out.tr") or die "couldn't open the file";
$line_no=0;
while ($line = <infile>) {
    ++$line_no;
    @entry = split(" ", $line);
    if(($entry[0] eq "+")&&($entry[2] eq "1")&&($entry[3]
eq "2")&&($entry[4] eq "cbr")) {
     print cmd "$entry[1] $entry[10] \n";
```

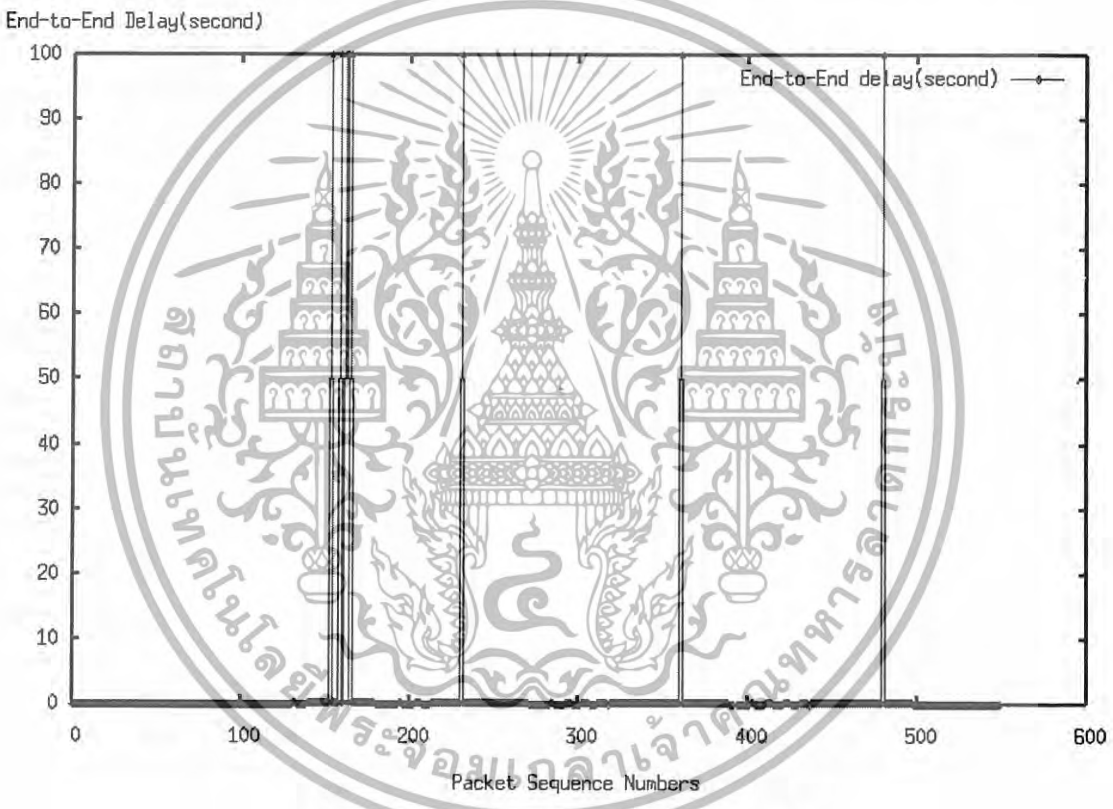
ตัวอย่างที่ 8. แสดง Perl สคริป ในการ plot graph ent to end delay (ต่อ)

```
    }
if(($entry[0] eq "r")&&($entry[3] eq "3")&&($entry[4] eq
"cbr")) {
        print cmd1 "$entry[1]  $entry[10]\n";
    }
}
close(cmd);
close(cmd1);
close(infile);
open(infiles,"sqs");
open(cmd,">delay");
while ($lines = <infiles>) {
    @entrys = split(" ", $lines);
    open(infiler,"sqr");
    $st=0;
    while ($liner = <infiler>) {
      @entryr = split(" ", $liner);
      if($entrys[1] eq $entryr[1]){
          $delay=$entryr[0]-$entrys[0];
          print cmd "$entrys[1] $delay\n";
          $st=1;
          last;
        }
    }
    if($st==0){
      print cmd "$entrys[1]  100\n"
    }
    close(infiler)
}
close(cmd);
close(infiles);
close(infiler);
# call plot_graph
plot_graph();
sub plot_graph{
     $output_type=0;
     $outname="data1.eps";
    $x_label="End-to-End Delay(second)";
    $y_label="Packet Sequence Number";
    open(cmd, ">cmd.tmp");
    if($output_type==0){
      print cmd "set term x11 \n";
    }else{
      print cmd "set term post portrait color solid
\'Roman\' 11\n";
      print cmd "set output \"$outname\"\n";
      print cmd "set size 1.0,0.5\n"
    }
    print cmd "set xlabel \"$x_label\" \n";
    print cmd "set ylabel \"$y_label\"\n";
    $output_files="delay";
    print cmd "plot \"$output_files\" u 1:2 t \"Een-to-
End delay(second)\" w linespoints";
```

ตัวอย่างที่ 8. แสดง Perl สคริป ในการ plot graph ent to end delay (ต่อ)

```
    print cmd "\n";
    close(cmd);
    system "gnuplot -persist cmd.tmp";
}
```

จากตัวอย่างที่ 8. จะมีการดึง seq no. กับเวลาของแพ็คเกตที่สามารถส่งและรับได้ แยกไว้ในไฟล์ชื่อ sqs และ sqr ตามลำดับ ซึ่งจากนั้นจะนำไฟล์ทั้ง 2 ดังกล่าวมาดึงข้อมูลเวลา โดยจะเอาเฉพาะที่ seq no. ตรงกันเท่านั้น แล้วบันทึก seq np. และผลต่างของเวลา ลงในไฟล์ผลลัพธ์ที่ชื่อว่า delay ซึ่งจะใช้ Gnuplot ในการดึงค่าจากไฟล์ไปวาดกราฟต่อไป



รูปที่ 2.19 แสดงผลการรันสคริปจากตัวอย่างที่ 8

2.12.3.3 หา delay jitter

Delay jitter คือค่าความเปลี่ยนแปลงของดีเลย์ระหว่างแพ็คเกต ซึ่งถ้ามีการเปลี่ยนแปลงบ่อยๆและเปลี่ยนแปลงในช่วงที่กว้าง จะหมายถึงเน็ทเวิร์คดังกล่าวไม่มีความเสถียร คือมีการเปลี่ยนแปลงของดีเลย์ อยู่ตลอดเวลา ซึ่งสูตรในการคำนวนหาได้ดังนี้

jitter = ((เวลารับ(j)-เวลาส่ง(j))-(เวลารับ(i)-เวลาส่ง(i)))/(j-i) โดย j > i

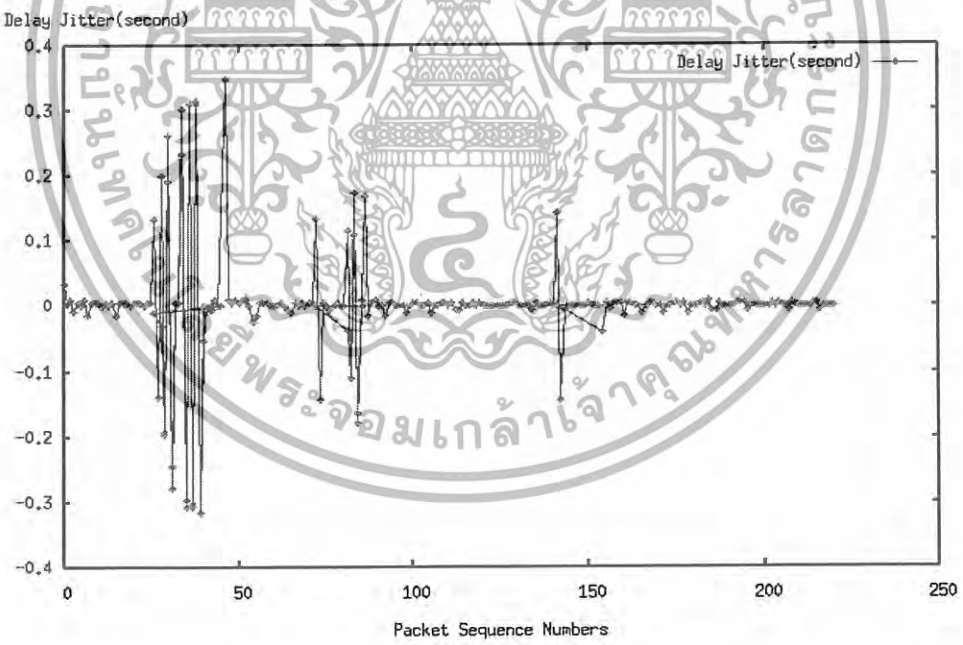
ดังนั้นหมายถึง jitter = delay(j)-delay(i)

ซึ่งเราสามารถทำการเพิ่มเติมโค้ดจากตัวอย่างที่ 8 ได้โดยเพิ่มโค้ดดังนี้

ตัวอย่างที่ 9. แสดงถึงส่วนของ perl สคริป ในการคำนวณ jitter

```
open(infiled,"delay");
$line_no=0;
open(cmdj,">jitter");
$predelay=0;
 while ($lined = <infiled>) {
      ++$line_no;
      @entryd= split(" ", $lined);
          $jitter=$entryd[1]-$predelay;
            $predelay =$entryd[1];
          print cmdj "$entryd[0] $jitter\n";

        }
close(cmdj);
close(infiled);
```

จะเห็นได้ว่าตัวอย่างที่ 9. มีการใช้ไฟล์ delay ซึ่งเป็นผลจากโค้ดในตัวอย่างที่ 8. ทำให้เราสามารถนำโค้ดจากตัวอย่างที่ 9. ไปใส่ในตัวอย่างที่ 8. ได้เลย จากนั้นจึงใช้ Gnuplot ในการดึงข้อมูลไปใช้ต่อไป



รูปที่ 2.20 แสดงกราฟจาก Perl สคริปคำนวณหา jitter

2.12.3.4 หาทรูพุท

ทรูพุท คือปริมาณข้อมูลที่สามารถส่งผ่านไปได้จริงๆณจุดใดจุดหนึ่งในลิงค์ ซึ่งจะมีหน่วยเป็นบิทต่อวินาที โดยจะไม่เท่ากับบิทเรต เนื่องจากบิทเรตเป็นเพียงค่าตามทฤษฎี ที่บอกว่าข้อมูลสามารถส่งไปได้ในปริมาณเท่าใดต่อวินาที แต่ตามความจริงแล้วปริมาณ

ข้อมูลที่ส่งไปได้จริงๆไม่อาจเป็นไปได้ตามบิทเรตเนื่องจากมีสภาพแวดล้อมต่างๆที่มารบกวน หรือมีผลกระทบต่อระบบ ดังนั้นค่าทรูพุทจึงเป็นค่าที่บ่งบอกถึงประสิทธิภาพของลิงค์ตามความจริงจากที่มีการใช้งานจริง ซึ่งเราสามารถหาค่าทรูพุทของระบบได้จากการที่เราทำการวัดข้อมูลที่ผ่านจุดใดจุดหนึ่งของลิงค์ในหนึ่งวินาที โดยปกติจะใช้ปริมาณข้อมูลทั้งหมดหารด้วยเวลาในหน่วยวินาที

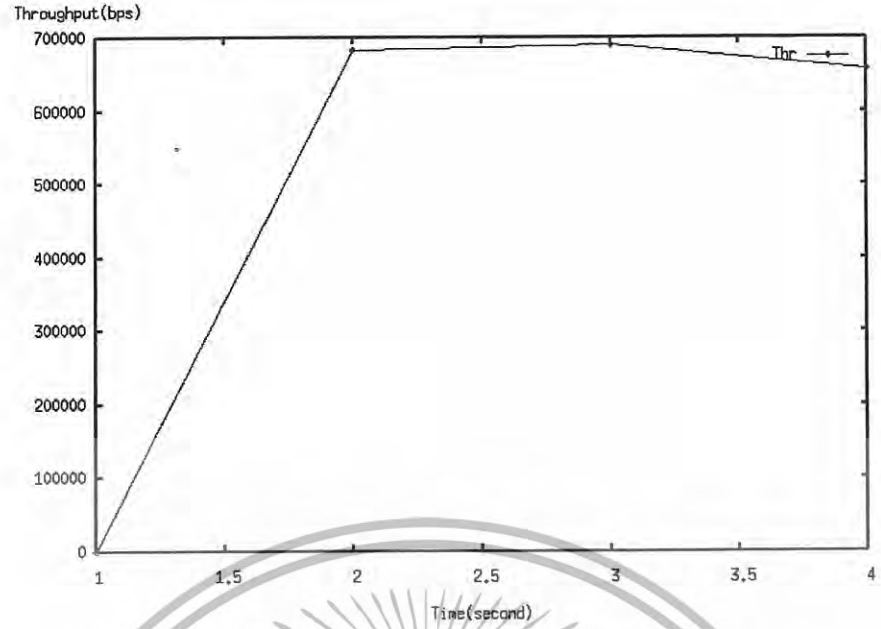
ทรูพุท = จำนวนข้อมูลที่ไหลผ่าน/เวลาทั้งหมด

จากเทรซไฟล์เราสามารถนำขนาดของทุกแพ็คเกตที่ ณ จุดใดจุดหนึ่งที่ทราฟฟิกมีการไหลผ่าน หารด้วยเวลา ก็จะได้ผลลัพธ์ออกมาเป็นค่าทรูพุท

ตัวอย่างที่ 10. Perl สคริปในการหาค่าทรูพุท

```
open(cmd1,">thr");
open(infile, "out.tr") or die "couldn't open the file";
$time=0;
$step=1;
$recv=0;
while ($line = <infile>) {
    @entry = split(" ", $line);
    if($entry[1]>$time+$step){
      $tmp_thr=$recv*8/$step;
      $time=$time+$step;
      print cmd1 "$time $tmp_thr\n";
      $recv=0;      }
    if(($entry[0] eq "r")&&($entry[3] eq "1")&&($entry[4]
eq "tcp")) {
      $recv=$recv+$entry[5];
 }}
close(cmd);
close(cmd1);
close(infile);
```

จากตัวอย่างที่10. เป็นเพิลสคริปที่ใช้ในการคำนวณหาทรูพุทจากเทรซไฟล์ โดยจะเก็บผลลัพธ์ไว้ที่ไฟล์ที่ชื่อว่า thr ซึ่งจะนำไปวาดกราฟด้วย Gnuplot ต่อไป



รูปที่ 2.21 กราฟแสดงค่าทรูพุท

2.12.3.5 หาจำนวนแพ็คเกตที่สูญเสีย

ในการส่งข้อมูลนั้น เรื่องการที่ข้อมูลที่ส่งไปจะถึงมือผู้รับไม่ครบถ้วนสมบูรณ์นั้น เป็นเรื่องที่หลีกเลี่ยงได้ยากมาก ซึ่งอาจจะแก้ปัญหาโดยมีกลไกการทำงานบางอย่างช่วยแจ้งให้ต้นทางส่งแพ็คเกตมาอีกครั้งเรียกการรีทรานสมิท ซึ่ง TCP จะรองรับการทำงานเช่นนี้ แต่จะไม่มีใน UDP

โดยเราสามารถนำเทรซไฟล์มาคำนวณเพื่อหาจำนวนแพ็คเกตที่สูญเสียโดยในที่นี้จะถือว่าแพ็คเกตที่ถูกโยนทิ้งด้วย DrpT จะเป็นแพ็คเกตที่สูญเสีย เนื่องจากจากความรู้ที่เรานำเขียน tcl สคริปนั้น ยังไม่สามารถสร้างผลกระทบอื่นที่ทำให้แพ็คเกตสูญเสียได้ ด้วย awk ดังนี้

ตัวอย่างที่ 11. awk สคริปหาแพ็คเกตที่สูญเสีย

```
BEGIN {
      fsDrops = 0;      numFs = 0;
} {
 if $3==1 && $4==2 && $1 == "+")
            numFs++;
      if ($8==2 && $1 == "d")
            fsDrops++;
}END { printf("number of packet sent:%d lost:%d\n",
numFs, fsDrops);}
```

2.12.3.6 หาแพ็คเกตที่สูญเสียจากดิว

เราสามารถรู้ได้ว่าแพ็คเกตใดบ้างที่ถูกดรอปโดยดูจากเทรซไฟล์ ซึ่งแพ็คเกตที่มีชนิดของเหตุการณ์เป็น "d" แปลว่าถูกโยนทิ้งด้วย DrpT ซึ่งวิธีการหาจำนวนก็สามารถทำได้ด้วยวิธีการเดียวกันกับการหาแพ็คเกตที่สูญเสียในหัวข้อ 2.12.3.5

บทที่ 3
การจำลองเครือข่าย MPLS โดยใช้ NS-2

3.1 บทนำ

เนื่องด้วยในปัจจุบันการเจริญเติบโตของเทคโนโลยีเครือข่ายอินเตอร์เน็ทเพิ่มขึ้นอย่างมากและรวดเร็ว ซึ่งจากเดิมที่อินเตอร์เน็ทใช้ในการส่งข้อมูลข่าวสารและแลกเปลี่ยนบทความวิชาการ แต่ในปัจจุบันมีการให้บริการใหม่ๆเกิดขึ้นตลอดเวลา การบริการแทบทุกลักษณะที่ทำให้อยู่ในรูปแบบของดิจิตอลได้นั้น จะถูกทำให้มาเป็นการบริการบนอินเตอร์เน็ท ทำให้เครือข่ายอินเตอร์เน็ทมีการใช้งานอย่างมากและกว้างขวาง จึงไม่ใช่เรื่องน่าแปลกใจที่เครือข่ายจะเกิดปัญหาความคับคั่งของข้อมูลเป็นอย่างมาก แม้ทางผู้พัฒนาจะมีการพัฒนาเทคโนโลยีต่างๆ ไม่ว่าจะอุปกรณ์ โปรโตคอล หรือตัวกลางเพื่อลดปัญหานี้ แต่ก็ยังไม่สามารถแก้ไขปัญหานี้ได้อย่างที่ควรจะเป็นเนื่องจากมีการใช้งานบางเส้นทางมากเกินไป

ในเครือข่ายปัจจุบันข้อมูลต่างๆจะถูกส่งผ่านตัวกลางที่เรียกว่าเราท์เตอร์ ซึ่งจะมีหน้าที่ส่งข้อมูลข้ามเครือข่าย โดยจะมีอัลกอริธึมในการที่จะตัดสินใจว่าจะนำข้อมูลที่ได้รับมานั้นส่งไปทางใด โดยจะใช้ที่อยู่ปลายทางจากส่วนหัวของแพ็คเกต ซึ่งอยู่ในระดับชั้นเน็ทเวิร์ค มาเป็นตัวตัดสินใจ ซึ่งจะนำมาเทียบกับข้อมูลในตารางเราท์ติ้ง ซึ่งข้อมูลที่ตรงกับที่อยู่ปลายทางมากที่สุด จะถูกนำมาใช้ โดยข้อมูลจะระบุถึงที่อยู่อุปกรณ์ที่ต้องส่งถัดไป ซึ่งจะเห็นได้ว่าเป็นกระบวนการที่ยุ่งยากพอสมควร ซึ่งเวลาหน่วงในขั้นตอนนี้จะเรียกว่า ความหน่วงในการสวิทชิ่ง

การที่เครือข่ายในระดับชั้นเน็ทเวิร์ค หรือระดับที่3 เป็นเครือข่ายไอพี ซึ่งมีอุปกรณ์เราท์เตอร์เป็นตัวหลักในการส่งผ่านข้อมูลไปให้ถึงปลายทาง และโปรโตคอลไอพีมีการทำงานแบบแพ็คเกตสวิทชิ่ง ซึ่งแพ็คเกตแต่ละแพ็คเกตจะเป็นอิสระต่อกัน ไม่สามารถติดตามหาเส้นทางที่แน่นอนได้ จึงเป็นการลำบากที่จะควบคุมและจัดการคุณภาพของการให้บริการ เช่นเวลาหน่วง หรือการจัดสรรแบนด์วิธ รวมถึงยากในการทำวิศวกรรมการจราจร (Traffic Engineering)

จึงเกิดเทคโนโลยีในระดับชั้นที่ 2 หรือดาต้าลิงค์ เช่น Asynchronous Transfer Mode (ATM) หรือเฟรมรีเรย์ ซึ่งจะมีการสร้างวงจรเสมือน (Virtual Circuit) สำหรับแต่ละผู้ใช้ได้ ซึ่งจะสามารถกำหนดเส้นทางจากต้นทางถึงปลายทาง ซึ่งจะสร้างเส้นทางก่อนที่จะมีการส่งข้อมูลซึ่งจะทำให้ไม่เกิดความหน่วงของเวลา และสามารถคาดการณ์อัตราการส่งของข้อมูลได้อย่างถูกต้อง โดยเครือข่าย ATM จะใช้อุปกรณ์ที่เรียกว่า ATM สวิทช์ แต่ปัจจุบันเครือข่าย ATM เติบโตไปมาก และสร้างปัญหาในแง่ของการส่งข้อมูลที่มี Overhead ปนเข้ามาในเครือข่ายมากมาย ซึ่งจากการสำรวจการใช้ ATM ที่เชื่อมต่อแบบ OC-48 จะมี Overhead ปนเข้ามาถึง 498 Mbps

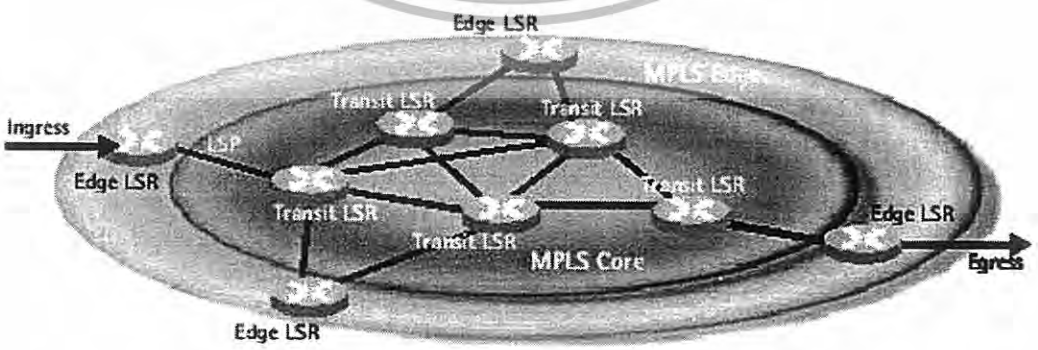
ซึ่งทาง Internet Engineering Task Force (IETF) ได้คิดค้นเทคโนโลยีเพื่อลดปัญหาเหล่านี้ โดยเทคโนโลยีดังกล่าวก็คือ MPLS หรือ Multi Protocol Label Switching

3.2 MPLS Overview

MPLS หรือ Multi Protocol Label Switching เป็นเทคโนโลยีที่ดึงเอาความสามารถของการทำงานของเทคโนโลยีของระดับชั้นที่ 2 และของระดับชั้นที่ 3 มารวมกัน โดยจะใช้การส่งข้อมูลที่เป็นไอพีแพ็คเกต แต่ให้มีการส่งข้อมูลคล้ายกับสวิทช์ในเลเยอร์ที่ 2 ซึ่งจะเป็นการส่งข้อมูลแบบ hop to hop ซึ่งสามารถติดตามและควบคุมการส่งของข้อมูลได้เพราะทราบลิงค์ที่แน่นอน ซึ่งจะเหมือนเป็นการผสมผสานกันระหว่างเครือข่ายที่เป็น connectionless และ connection oriented โดยคำว่า Multi Protocol มาจากการที่ MPLS สามารถใช้งานร่วมกับโปรโตคอลอะไรก็ได้ในระดับชั้นที่ 3 ซึ่งปกติก็คือไอพี

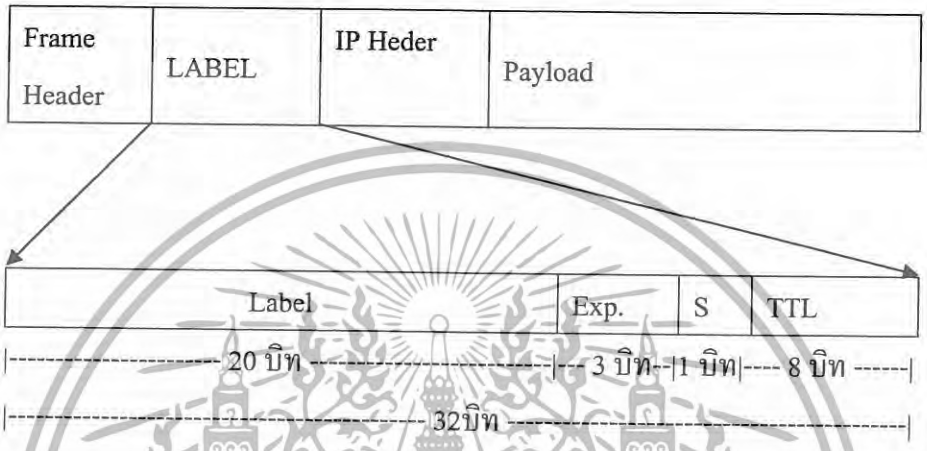
MPLS นั้นเอามาเพิ่มประสิทธิภาพและแก้ปัญหาในการใช้งานเครือข่ายในด้านของความเร็ว, ความสามารถในการเปลี่ยนแปลงขนาด (Scalability), การจัดการเกี่ยวกับคุณภาพของการให้บริการ (QoS) และวิศวกรรมการจราจร (Traffic Engineering) โดย MPLS จะเป็นตัวกลางที่ทำให้โปรโตคอลในระดับชั้นที่ 2 และ 3 เป็นอิสระต่อกัน

เครือข่ายของ MPLS เรียกว่า MPLS โดเมน ซึ่งจะประกอบด้วยกลุ่มของอุปกรณ์ที่ทำหน้าที่สวิทชิ่งเช่น เราท์เตอร์ หรือ ATM สวิทช์ที่รองรับการทำงานของ MPLS เรียกว่า LSR (Label Switch Router) โดย LSR แต่ละตัวในโดเมนจะต้องเชื่อมต่อกัน โดยจะแบ่ง LSR ออกเป็น 2 ชนิดคือ Edge LSR และ Core(transit) LSR โดย Edge LSR จะเป็นตัวกลางระหว่างเครือข่ายอื่นและ MPLS โดเมน โดย Edge LSR จะอยู่ที่ขอบนอกสุดของ domain จะเชื่อมต่อกับเราท์เตอร์ทั่วไป และอีกด้านจะเชื่อมต่อกับ Core LSR ขณะที่ Core LSR คือ LSR ที่อยู่ภายใน MPLS โดเมนซึ่งลิงค์จะเชื่อมต่อกับ LSR ด้วยกันเองเท่านั้น



รูปที่ 3.1 แสดง MPLS โดเมน

ในเราท์เตอร์ ทั่วไปจะใช้ข้อมูลที่อยู่ในส่วนหัวของแพ็คเกตเป็นตัวตัดสินใจว่าจะทำการส่งแพ็คเกตดังกล่าวที่รับมาออกไปทางใด แต่ใน LSR จะใช้ข้อมูลในลาเบล (label) มาพิจารณาและตัดสินใจ ซึ่งลาเบลนั้นมีขนาดที่คงที่และจะถูกแทรกไปอยู่ในทุกๆเฟรมที่มีการส่งใน MPLS โดเมน โดยลาเบลจะแทรกอยู่ระหว่างส่วนหัวของเฟรมที่สร้างจากระดับชั้นดาต้าลิงค์และส่วนหัวของแพ็คเกตที่สร้างจากระดับชั้นเน็ทเวิร์ค



รูปที่ 3.2 แสดงตำแหน่งของลาเบล

- Label จะเป็นค่าของลาเบลจริงที่ LSR แต่ละตัวจะใช้ในการตัดสินใจว่าจะกระทบอย่างไรต่อแพ็คเกตดังกล่าว
- Exp. คือ Experimental Bits เป็นส่วนที่ไว้ใช้แบ่งแยกชั้นของการให้บริการของ DiffServe
- S คือ Stack filed เป็นค่าที่ไว้ใช้กำหนดว่ามีลาเบลระดับต่อไปอีกหรือไม่ในสแต็ก
- TTL คือ Time to Live เป็นค่าที่จะลดลงทุกครั้งที่ผ่านอุปกรณ์เพื่อป้องกันการเกิดลูป

ลาเบลใน MPLS สามารถมีได้หลายระดับโดยจะมีโครงสร้างเป็นแบบสแต็ก (Stack) ซึ่งการเข้าถึงข้อมูลในสแต็กจะต้องกระทำจากข้อมูลที่อยู่เหนือสุดของสแต็กและมีการนำข้อมูลใส่สแต็กเรียกการพุช (Push) ส่วนการดึงข้อมูลออกจากสแต็กเรียกการป๊อป (Pop) ซึ่งจะมีโครงสร้างของคิวเป็นแบบเข้าก่อนออกทีหลัง (LIFO) เช่น

รูปที่ 3.3 แสดงสแต็กของลาเบล

โดยจากรูปจะมีการพุชลาเบล 1 เข้าไปก่อนและตามด้วยลาเบล 2 ซึ่งหากต้องการอ่านข้อมูลในลาเบล 1 ต้องมีการป๊อปลาเบล 2 ออกมาก่อน

ปกติการทำงานของเราท์เตอร์ จะทำก่อนอ่านส่วนหัวของเฟรมก่อนเมื่อพบว่าเป็นเฟรมที่มันต้องนำมาประมวลผลจะถอดส่วนหัวของเฟรมนั้นทิ้งและอ่านส่วนหัวถัดไปซึ่งปกติคือส่วนหัวของไอพี แต่ใน MPLS นั้นเมื่อ LSR ถอดส่วนหัวของเฟรมออกจะพบกับลาเบลแทน ซึ่งข้อมูลใน ลาเบลจะเป็นตัวกำหนดว่าจะให้ LSR ที่รับแพ็คเกตนั้นกระทำอะไรต่อไป และจะจัดการกับแพ็คเกตที่ได้รับมาอย่างไร ซึ่งในส่วนข้อดีตรงนี้ทำให้ MPLS สามารถจัดการเกี่ยวกับคุณภาพของการให้บริการ (QoS) ได้ และสามารถจัดแบ่งชั้นของการให้บริการ (Class of Service) ได้ โดยข้อมูลในลาเบลจะสามารถบอก LSR ได้ทันทีว่าต้องส่งข้อมูลออกทางอินเตอร์เฟซ (interface) ใด และไปที่ใคร โดยไม่ต้องมาทำการเทียบหาค่าที่ตรงมากที่สุดในตารางเราท์ติ้ง จึงทำให้มีความรวดเร็วกว่า แต่อย่างไรก็ดี ด้วยเทคโนโลยีการสวิทชิ่งในปัจจุบันได้พัฒนาไปมากจนความเร็วในส่วนนี้แทบมองไม่เห็นความแตกต่าง ดังนั้น MPLS จึงมุ่งประเด็นไปที่การทำวงจรเสมือนมากกว่า

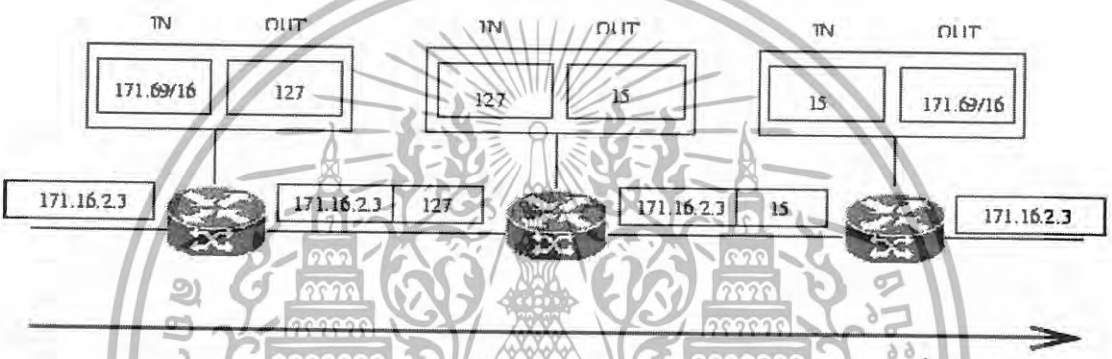
โดยแม้เทคโนโลยีเช่น ATM หรือเฟรมรีเลย์ ก็สามารถทำวงจรเสมือนได้เช่นเดียวกัน แต่ MPLS มีความยืดหยุ่นมากกว่า เพราะสามารถทำงานบนระบบเครือข่ายเดิมได้ทันทีโดยไม่ต้องติดตั้งหรือแก้ไขอุปกรณ์ใหม่ (เราท์เตอร์ ที่ใช้ต้องรองรับ MPLS) จึงประหยัดค่าใช้จ่ายในส่วนนี้ อีกทั้งเราท์เตอร์ หรือ ATM ที่รองรับ MPLS (ต่อไปนี้จะเรียกว่า LSR) ยังสามารถทำหน้าที่เดิมของมันเองได้อย่างเต็มที่ เช่น เราท์เตอร์ ก็ยังสามารถทำการส่งต่อไอพีแพ็คเกตปกติได้อยู่เช่นเดิม เนื่องจากจะมีกลไกตรวจสอบว่าแพ็คเกตที่ส่งมาเป็นไอพีแพ็คเกตหรือ MPLS แพ็คเกต

เส้นทางวงจรเสมือนนั้นเรียกว่า LSP (Label Switching Path) ซึ่งจะเป็นเส้นทางจาก Edge LSR หนึ่งไปยังอีก Edge LSR หนึ่ง ซึ่งอาจผ่านหรือไม่ผ่าน Core LSR ระหว่างทางก็ได้ ซึ่ง LSP นั้นจะมีทิศทางการไหลของข้อมูลแบบทิศทางเดียว (Unidirectional) โดย Edge LSR ที่เป็นคนส่งแพ็คเกตเข้าสู่ MPLS โดเมนเป็น Edge LSR ต้นทางจะเรียกว่า Ingress LSR และ Edge LSR ปลายทางที่จะนำแพ็คเกตออกนอก MPLS โดเมนจะเรียกว่า Egress LSR โดยหนึ่ง LSP จะเป็นเส้นทางจาก Ingress LSR หนึ่งไปสู่ Egress LSR หนึ่งเท่านั้น ซึ่งหากต้องการให้ไปและกลับได้ด้วยก็ต้องมี 2 LSP โดยเส้นทางของ LSP อาจได้จากเราท์ติ้งโปรโตคอลในระดับชั้นที่ 3 เช่น OSPF, IS-IS หรือ BGP

ดังที่ได้กล่าวไปว่า LSR จะใช้ข้อมูลในลาเบลในการตัดสินใจว่าจะส่งต่อข้อมูลออกไปทางใด โดยค่าที่ใช้จะเรียกว่า FEC (Forwarding Equivalence Class) โดย FEC อันหนึ่งจะหมายถึงกลุ่มของไอพีแพ็คเกตที่จะมีการส่งข้อมูลไปในเส้นทางเดียวกันและลักษณะเดียวกัน นั่นหมายถึง ถ้า FEC เหมือนกัน แพ็คเกตนั้นจะถูกส่งผ่านไปใน MPLS โดเมนด้วย LSP เดียวกัน

โดยเมื่อ ingress LSR ได้รับแพ็คเกตที่ไม่มีลาเบลประกอบอยู่ LSR นั้นจะนำที่อยู่ปลายทางที่ได้จากส่วนหัวของไอพีแพ็คเกตมาดูว่าตรงกับ FEC ใดหรือไม่หากไม่ตรงก็จะทำการส่งข้อมูลต่อไปที่

router ที่อยู่ถัดไปแบบปกติ แต่หากพบว่ามี FEC ที่ตรงกับที่อยู่ปลายทางดังกล่าว จะนำไปเทียบกับตาราง LIB (Label Information Base) ว่าจะกำหนดลาเบลอะไรให้แพ็คเกตดังกล่าวและต้องส่งไปที่ LSR ถัดไปตัวใด และซึ่งจะใส่ลาเบล (Push) และส่งไปที่ LSR ตัวนั้น เมื่อ LSR ถัดไปได้รับลาเบลแพ็คเกตนั้น ก็จะทำการดึงค่าในลาเบลมาเทียบกับ LIB ของตน ซึ่งตาราง LIB จะเก็บข้อมูลการเทียบระหว่าง อินเตอร์เฟซและลาเบลขาเข้า เทียบกับอินเตอร์เฟซและลาเบลขาออก ซึ่ง LSR ดังกล่าวจะดูว่าตรงกับข้อมูลใดและทำการเปลี่ยน (Swap) ลาเบลนั้นเป็นค่าใหม่และส่งไปที่ LSR ตัวถัดไป จนกระทั่งถึง Egress LSR เมื่อได้รับลาเบลแพ็คเกตและนำค่าในลาเบลมาเทียบกับ LIB ของตนจะพบว่าไม่มีลาเบลขาออกให้ทำการเปลี่ยน จึงทำการถอดลาเบลออก (Pop) และนำที่อยู่ปลายทางของไอพีแพ็คเกตมาเทียบกับค่าในตารางเราท์ติ้งและส่งให้อุปกรณ์ถัดไป



รูปที่ 3.4 กระบวนการส่งข้อมูลโดยใช้ลาเบลใน MPLS

จากรูปจะเห็นได้ว่าประกอบไปด้วย 3 LSR ซึ่ง LSR ซ้ายและขวา เป็น Edge LSR และ LSR ตรงกลางคือ Core LSR สมมติให้ LSP มีทิศทางจากซ้ายไปขวา ดังนั้น LSR ทางด้านซ้ายคือ ingress LSR และทางด้านขวาคือ Egress LSR ของ LSP นี้ ซึ่งเมื่อ Egress LSR รับแพ็คเกตปกติเข้ามาที่ไม่มีลาเบลประกอบอยู่จะทำการอ่านที่อยู่ปลายทางส่วนหัวของไอพีซึ่งพบว่าแพ็คเกตดังกล่าวต้องการไปยังซับเน็ทที่ 171.69/16 ซึ่งเมื่อนำมาเทียบกับตาราง LIB ของตนพบว่าอยู่ใน FEC 127 จึงทำการส่งข้อมูลออกไปที่ LSR ตัวถัดไปโดยแทรกลาเบลที่มีค่า 127 ไปด้วย เมื่อ Core LSR ได้รับ MPLS แพ็คเกตดังกล่าวและทำการตรวจสอบพบว่ามีลาเบลและค่าในลาเบลคือ 127 เมื่อเทียบกับ LIB ของตนพบว่าต้องส่งไปที่ LSR ตัวถัดไปด้วยลาเบล 15 เมื่อ Egress LSR ได้รับก็ทำการตรวจสอบลาเบลและนำค่ามาเทียบในตารางของตนพบว่าต้องทำการถอดลาเบลออกและอ่านที่อยู่ปลายทางจากส่วนหัวของไอพีแพ็คเกตเพื่อเทียบกับตารางเราท์ติ้งเพื่อส่งไปที่อุปกรณ์ถัดไป

โดยปกติแล้ว LSR ตัวสุดท้ายจะเป็นตัวที่ทำการ pop ข้อมูลของลาเบลออก แต่การทำงานนั้นจะทำให้เสียเวลา เนื่องจากว่าจะต้องตรวจสอบตารางถึงสองครั้ง โดยครั้งแรกจะต้องตรวจสอบตาราง LIB เพื่อตรวจสอบว่าจะต้องส่งต่อไปยังอินเตอร์เฟสและลาเบลใดต่อไป เมื่อพบว่าไม่มี

ข้อมูลลาเบลที่ต้องส่งต่อไป จึงไปตรวจสอบตารางเราท์ติ้งเทเบิ้ล โดยตรวจสอบจากไอพีแอดเดรสว่าต้องส่งไปยังที่ใด ซึ่งเป็นการเสียเวลามาก ดังนั้นในการจัดการลาเบลของ MPLS นั้นมีกลไกที่เรียกว่า penultimate hop poping ซึ่งจะทำให้เกิดการถอดลาเบลออกตั้งแต่โหนดก่อนสุดท้าย เพื่อที่ทำให้โหนดที่อยู่สุดท้ายไม่ต้องเสียเวลาในการตรวจสอบลาเบล เพียงแต่ทำการตรวจสอบจากเราท์ติ้งเทเบิ้ลอย่างเดียวเท่านั้น เพื่อลดภาระการทำงานของ Egress LSR

3.3 Label Distribution Protocol

MPLS สามารถทำให้ไอพีเนตเวิร์กสามารถทำงานได้อย่างมีประสิทธิภาพมากขึ้น แต่ในการทำงานของ MPLS นั้นก็ต้องมีการแลกเปลี่ยนลาเบลซึ่งกันและกัน ซึ่งจะทำได้โดยใช้โปรโตคอล LDP ซึ่งถูกพัฒนาขึ้นมาสำหรับ MPLS โดยเฉพาะ แต่ว่าในความเป็นจริงในการแลกเปลี่ยนลาเบลนั้นสามารถทำได้หลายวิธี เช่น อาจใช้ BGP ในการแลกเปลี่ยนก็ได้

LDP นั้นมีการทำงานที่เหมือนกับการรวมกันของ OSPF และ BGP โดยการทำงานที่เหมือน OSPF คือ LDP มีความสามารถให้โหนดที่อยู่ติดกันนั้นเรียนรู้เส้นทางด้วยตนเองได้โดยอัตโนมัติ และที่เหมือน BGP ตรงที่โหนดที่อยู่ติดกันสามารถใช้การเชื่อมต่อแบบ TCP เพื่อทำให้มีความน่าเชื่อถือ ส่วนคุณลักษณะเฉพาะของ LDP นั้นคือมีฟังก์ชันการทำงานให้เกิดการแลกเปลี่ยนข้อมูลลาเบลได้นั่นเอง

ก่อนที่จะมีการแลกเปลี่ยนข้อมูลลาเบลกันนั้น จะต้องมีการติดต่อเพื่อเรียนรู้เส้นทางกันก่อนว่าจะติดต่อกันได้อย่างไร โดยแต่ละเราท์เตอร์ทุกตัวจะต้องทำการส่ง Hello เมสเสจที่เป็นลักษณะมัลติคาสออกไปยังทุกอินเตอร์เฟสของตัวเอง จากนั้นจึงสามารถแลกเปลี่ยนลาเบลกันได้โดยผ่านเส้นทางที่เรียกว่าลาเบลสวิชชิ่งพาท (LSP)

รูปแบบของข้อมูลที่เป็น LDP มีหลายชนิดได้แก่

- LDP Request Message ใช้ในการขอลาเบลใหม่
- LDP Mapping Message ใช้ในบอกการแลกเปลี่ยนข้อมูลในการติดต่อระหว่างกัน
- LDP Withdraw Message ใช้เพื่อเป็นการบอกว่าอินเตอร์เฟสนี้ได้ถูกยกเลิกแล้ว
- LDP Release Message ใช้ตรงข้ามกับ Request เมสเสจ เพื่อเป็นการบอกว่าต้องการยกเลิก Request เมสเสจ ที่ได้ส่งไปก่อนหน้า
- LDP Notification Message ใช้ในการแจ้งเตือนเมื่อเกิดข้อผิดพลาดขึ้น

3.4 MPLS in NS-2

ระบบจำลองเครือข่ายที่ชื่อ NS-2 สามารถจำลองเครือข่ายที่เป็น MPLS ได้ โดย NS-2 จะมีคลาสและฟังก์ชั่นต่างๆที่เกี่ยวข้องซึ่งมีคุณสมบัติต่างๆพร้อมที่จะจำลองการทำงานของเครือข่าย MPLS ได้ค่อนข้างสมบูรณ์ในระดับหนึ่ง โดยจะถูกบรรจุอยู่ใน NS-2 ตั้งแต่รุ่น 2.15b เป็นต้นไป ซึ่งจะมีไฟล์ตัวอย่างแนบมาให้ด้วย

ซึ่งการใช้ NS ในการจำลอง MPLS (ต่อไปนี้เรียก MNS) มีจุดประสงค์เช่นเดียวกับการใช้แบบจำลองก็คือการจำลองเพื่อทดสอบหรือศึกษาการทำงานและประสิทธิภาพโดยไม่ต้องทำการติดตั้งเครือข่ายจริงๆ

โดยขอบเขตของ MNS ที่สามารถทำการจำลองได้ จะเป็นไปดังนี้

- จำลองการสวิทชิ่งใน MPLS เกี่ยวกับการสลับลาเบล การลดทอนของ TTL หรือการดึงลาเบลออกจาลาเบลสแต็ก เป็นต้น
- จำลองการทำงานของ LDP โดยเป็นจำลองเมสเซจเกี่ยวกับการร้องขอ(Request), การเทียบ(Mapping) การยกเลิก(Withdraw) การปลดปล่อย(Release) และการประกาศ(Notification)
- จำลองการทำงานของ CR-LDP โดยเป็นการจำลองเมสเซจในการร้องขอ และการเทียบ

นอกจากนี้ยังมีความสามารถเกี่ยวกับการสร้างและการจัดการ LSP อาทิเช่น

- รูปแบบ LSP ทริกเกอร์ ซึ่งประกอบด้วยการขับเคลื่อนโดยการควบคุม(Control-Driven) และการขับเคลื่อนตามข้อมูล (Data-driven)
- รูปแบบแผนการจองและกระจายลาเบล โดยเป็นแบบทวนน้ำในการขับเคลื่อนโดยการควบคุม และแบบทวนน้ำและตามน้ำโดยการขับเคลื่อนด้วยข้อมูล
- โหมดการควบคุมการกระจายลาเบล (Label Distribution Control Mode)โดยเป็นโหมดอิสระ (Independent) ในการขับเคลื่อนโดยการควบคุม และโหมดอิสระและโหมดตามคำสั่ง (Ordered) ในการขับเคลื่อนด้วยข้อมูล
- โหมดการสงวนลาเบล (Label Retention Mode) สำหรับโหมดอรุรักษ์นิยม (Conservative Mode)
- ER-LDP บนการทำงาน CR-LDP ซึ่งจะเป็นการสร้างเส้นทางที่กำหนดโดยผู้ใช้
- การรวมเส้นทาง (Aggregated flow) ซึ่งจะรวมเส้นทางหลายๆเส้นทางเป็นเส้นทางเดียว

3.4.1 โครงสร้างของ MPLS Node

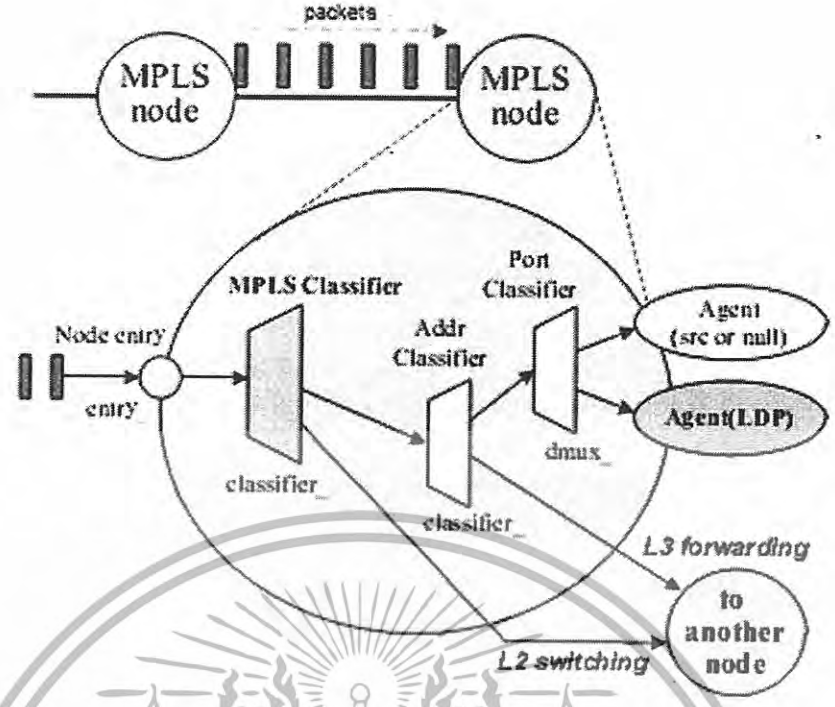
MPLS Network Simulator หรือ MNS เป็นส่วนที่มีการต่อเพิ่มจาก NS ปกติ ซึ่ง NS นั้นมีการทำงานที่อยู่บนพื้นฐานของการจำลองเครือข่ายไอพี โดยทั่วไปแล้วใน NS node จะประกอบไปด้วยเอเจนท์และตัวจำแนกโดยเอเจนท์คือออบเจคของผู้ส่งและผู้รับของแต่ละโปรโตคอล และตัวจำแนกเป็นออบเจคที่มีหน้าที่ในการจำแนกเพคเก็ทเพื่อใช้ในการส่งต่อแพ็คเกตไปยังโหนดถัดไป โดยเราจะแทรกตัวจำแนก MPLS และ LDP เอเจนท์ไปอยู่ในไอพีโหนด

ในรูปที่ 3.5 จะแสดงถึงโครงสร้างของ MPLS โหนดในการจำลอง MPLS โดย node entry จะแสดงจุดในการเข้าไปภายในโหนด คือส่วนแรกแรกในการควบคุมแพ็กเก็ทที่เข้ามายังโหนด โดย 'entry_' เป็นตัวแปรที่ได้จากการทำอินสแตนท์ ซึ่งจะเป็นตัวอ้างอิงสำหรับโหนดนี้ ตั้งแต่การ Multicasting ไม่ได้มีการสนับสนุนตัวแปรที่ต้องอ้างถึง

สิ่งแรกที่ต้องสนใจคือการจำแนกว่าแพ็คเกตที่เข้ามายังโหนดเป็นแพ็คเกตที่มีหรือไม่มีลาเบลประกอบอยู่ โดยหากไม่มีลาเบลก็จะถูกจัดการในลักษณะหนึ่ง แต่หากมีลาเบลก็จะถูกส่งให้ตัวจำแนก MPLS ในส่วนของการสับเปลี่ยนลาเบลและทำแพ็คเกตสวิทชิ่ง โดยตัวแปร 'classifier_' จะเป็นตัวอ้างอิงไปยังโหนดอื่นหรืออ้างอิงไปยังตัวจำแนกที่อยู่ซึ่งจะทำหน้าที่ส่งแพ็คเกตที่อยู่บนพื้นฐานไอพี และตัวจำแนกพอร์ทซึ่งจำทำหน้าที่เลือกเอเจนท์

โดย MPLS โหนดได้รับแพ็คเกตจะมีการทำงานดังนี้

1. จะมีการตรวจสอบว่าเป็นแพ็คเกตที่มีลาเบลหรือไม่มีลาเบลประกอบอยู่ หากเป็นแพ็คเกตที่มีลาเบล ตัวจำแนก MPLS จะดำเนินการทำสวิทชิ่งในระดับชั้นที่ 2 เนื่องจากเป็นการส่งแพ็คเกตโดยตรงไปยังโหนดถัดไป โดยเกิดขึ้นหลังจากได้มีการสับเปลี่ยนลาเบลไปแล้ว และหากรับมาเป็นแพ็คเกตที่ไม่มีลาเบลแต่ได้มีการสร้างเส้นทาง LSP สำหรับแพ็คเกตดังกล่าวไว้แล้ว ตัวจำแนก MPLS จะทำการส่งต่อไปยังโหนดถัดไปโดยเพิ่มลาเบลเข้าไป หรือหากไม่มีเส้นทาง ก็จะส่งไปยังตัวจำแนกที่อยู่ซึ่งจะทำการส่งต่อข้อมูลในระดับไอพี
2. ตัวจำแนกที่อยู่ จะทำงานในระดับชั้นที่ 3 ในการส่งแพ็คเกตไปยังปลายทาง
3. ถ้าโหนดถัดไปของแพ็คเกตดังกล่าวคือตัวมันเอง จะส่งแพ็คเกตไปยังตัวจำแนกพอร์ท

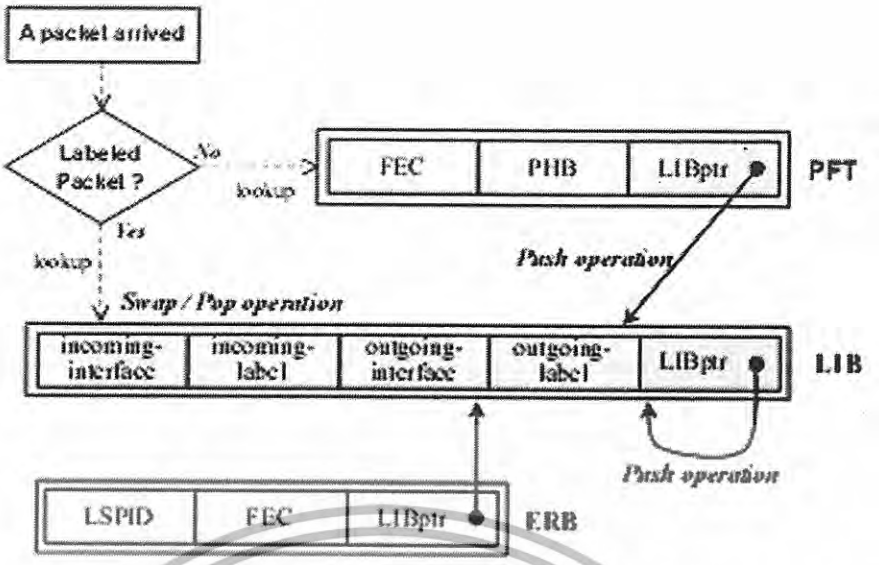


รูปที่3.5 แสดงของโครงสร้างของ MPLSโหนด

ในแต่ละ MPLS โหนดจะเก็บตาราง 3 ตารางในการจัดการข้อมูลที่เกี่ยวข้องกับ LSP, PFT (Partial Forwarding Table) , LIB (Label Information Base) และERB (Explicit Routing Information Base)

โดยตาราง PFT เป็นส่วนหนึ่งของตารางที่ใช้ในการส่งข้อมูลซึ่งประกอบด้วย FEC , PHB (Per-Hop-Behavior) และ LIBptr ขณะที่ตาราง LIB จะมีข้อมูลสำหรับ LSP และ ตาราง ERB จะมีข้อมูลสำหรับ ER-LSP

รูปที่ 3.6 จะแสดงโครงสร้างของตารางทั้งหลายและอัลกอริธึมพื้นฐานสำหรับการส่งแพ็คเกต โดย LIBptr ของแต่ละตารางเป็นพอยเตอร์ที่ชี้ไปยังข้อมูลในตาราง LIB



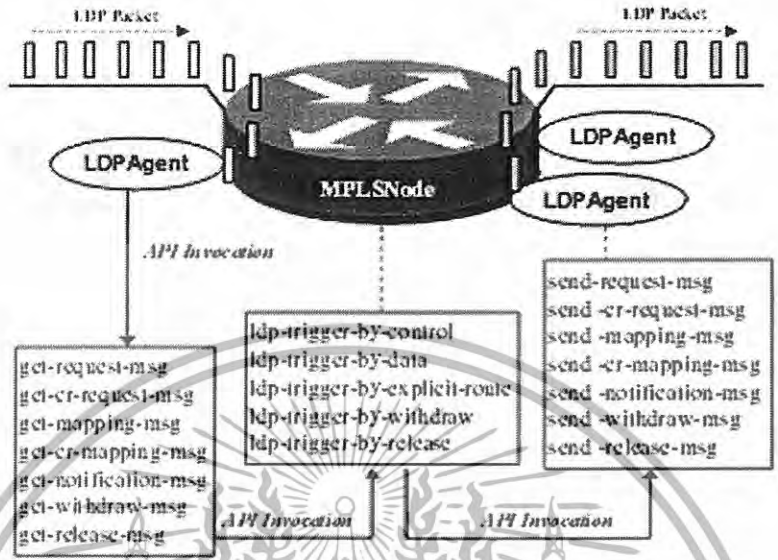
รูปที่ 3.6 แสดงโครงสร้างของตาราง MPLS แพ็คเกต switching

การเทียบค่าระหว่างตาราง PFT และตาราง LIB จะเกิดขึ้นก็ต่อเมื่อ MPLS โหนดได้รับแพ็คเกต ในกรณีของแพ็คเกตที่ไม่มีลาเบล MPLS โหนดจะค้นหาข้อมูล FEC ในตาราง PFT สำหรับแพ็คเกตดังกล่าวโดยดูจากที่อยู่ปลายทางของแพ็คเกตนั้น เมื่อพบ FEC ที่ต้องการจะใช้ LIBptr ของแถวนั้นอ้างอิงไปยังตาราง LIB โดยจะนำค่าลาเบลขาออกจากตาราง LIB นั้นมาบรรจุลงในลาเบลของแพ็คเกตดังกล่าว หากเมื่อเทียบ LIBptr ไปแล้วพบค่าว่าง หมายถึงเข้าสู่ Penultimate Hop Popping ซึ่งจะทำการถอดลาเบลออก และMPLS โหนดจะส่งแพ็กเก็ทต่อไปในรูปแบบการส่งในระดับชั้นที่ 3 หาก MPLS จะทำการใส่ลาเบลเข้าในแพ็คเกตโดยใช้ค่าในฟิลด์ลาเบลขาออกของตาราง LIB ที่ถูกอ้างอิงโดย LIBptr ของตาราง PFT แต่ในขณะเดียวกัน MPLS มีการทำลาเบลของแสต็กซึ่งจะมีการใส่ลาเบลเพิ่มเข้าไปเป็นลำดับชั้น โดยจะใส่ลาเบลไปจนกระทั่ง LIBptr เป็นค่าว่างคือไม่มี FEC ให้ทำการใส่เข้าไปในลาเบลอีกแล้ว ซึ่งหลังจากกระบวนการใส่หรือสับเปลี่ยนลาเบลเสร็จสิ้น MPLS โหนดจะทำการส่งแพ็คเกตดังกล่าวไปยังโหนดถัดไปโดยตรงโดยใช้ข้อมูลอินเอตร์เฟซขาออกจากตาราง LIB

ในกรณีที่เป็นแพ็คเกตที่มีลาเบลนั้นจะเป็นการสะดวกที่ MPLS โหนด จะระบุข้อมูลในตาราง LIB โดยการที่จะใช้ลาเบลเป็นดัชนีชี้ไปหาข้อมูลในตาราง LIB ซึ่ง MPLS โหนดจะใช้ข้อมูลลาเบลขาออกในแถวดังกล่าวมาทำการแทนที่ลาเบลของแพ็คเกตที่รับเข้ามา ซึ่งหากข้อมูลลาเบลขาออกเป็นค่าว่างนั่นหมายถึงให้ทำการถอดลาเบลออก

ตาราง ERB ใช้สำหรับใช้ในการเก็บเฉพาะข้อมูลที่เกี่ยวข้องกับ ER-LSP ซึ่งไม่มีผลในการส่งแพ็คเกตเลย โดยหากต้องการที่จะนำเส้นทางใหม่เข้าไปยังอยู่ใน ER-LSP ที่ถูกสร้างไปแล้ว

ก่อนหน้า โดยข้อมูลใหม่ที่มี LIBptr เหมือนกับข้อมูลในตาราง ERB ก็จะมีการใส่เข้าไปยังตาราง PFT



รูปที่ 3.7 API ของ LDP ที่ใช้ในการสร้างเครือข่าย MPLS

เมื่อ LDP เอเจนท์ได้รับ LDP เมสเซจจะมีการจัดการเมสเซจดังกล่าว มีการเลือกโหนดถัดไป มีการสร้างเมสเซจ LDP ใหม่ และมีการส่งเมสเซจไปยังเอเจนท์ของโหนดถัดไป ลำดับของการเรียกใช้ API จะเป็นไปดังรูปที่ 3.7 เมื่อ LDP เอเจนท์ได้รับ LDP เมสเซจ จากนั้น API ที่ทำการ get -*- message คือจะรับเมสเซจที่มีการร้องขอเข้าสู่โหนด จากนั้น LDP เอเจนท์จะทำการ เรียกใช้ API ldp-trigger-by -*- คือจะให้เอเจนท์เกิดการทำงานตามการร้องขอนั้นและจะทำการเลือกโหนดถัดไปที่จะส่งไปให้ ซึ่งเมื่อเลือกสำเร็จเอเจนท์จะทำการเรียก API API send-*-message เพื่อทำการส่งเมสเซจไปยังโหนดถัดไปนั้น

3.5 อธิบายการเขียนโปรแกรมในภาษา TCL สำหรับการจำลองเครือข่าย MPLS

โดยอันดับแรกต้องมีการสร้างซิมมูเลชั่นออปเจ็ค ก่อนโดยการเขียนโปรแกรมดังนี้

```
#Create a simulator object
set ns [new Simulator]
```

เป็นการสร้างออปเจ็คจากคลาสซิมมูเลเตอร์ (Simulator) โดยให้ตัวออปเจ็คนั้นชื่อว่า **ns** ซึ่งเป็นการกำหนดให้ออปเจ็คมีคุณสมบัติเดียวกันกับตัวคลาสซิมมูเลเตอร์เพื่อที่จะได้นำออปเจ็คนั้นมาทำการ จำลองเครือข่ายที่เราต้องการได้

```
#Turn on all control-driven method
Classifier/Addr/MPLS set control_driven_ 1
```

เป็นการสั่งให้โปรแกรมมีการทำงานในโหมดขับเคลื่อนด้วยการควบคุม (control-driven) โดยการกำหนดค่าให้เป็น 1 ก็คือการเปิดโหมดการทำงาน

```
#Turn on ldp and mpls traces
Agent/LDP set trace_ldp_ 1
Classifier/Addr/MPLS set trace_mpls_ 1
```

เป็นการสั่งให้โปรแกรมมีการเก็บเทรซไฟล์ของ ldp และบรรทัดต่อมาจะการเก็บเทรซไฟล์ของ MPLS โดยการกำหนดค่าให้เป็น 1 คือการเปิดโหมดในการเก็บเทรซไฟล์ของ MPLS

```
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
```

เป็นการสร้างไฟล์ที่จะสำหรับแสดงผลใน NAM (Network Animator) โดยมีชื่อว่า out.nam ซึ่ง "w" เป็นพารามิเตอร์ที่กำหนดให้มีการเปิดไฟล์ดังกล่าวเพื่อเขียนไฟล์ใหม่ และในบรรทัดต่อมาจะเป็นการกำหนดให้เก็บผลการเทรซทั้งหมดลงไฟล์ out.nam

```
#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf
```

เป็นการเปิดไฟล์ out.tr เพื่อเก็บผลการจากการเทรซทราฟฟิคของ NS ซึ่ง "w" เป็นพารามิเตอร์ที่กำหนดให้มีการเปิดไฟล์ดังกล่าวเพื่อเขียนไฟล์

```
#set IP node And MPLS node
set Node0 [$ns node]
set Node1 [$ns node]
$ns node-config -MPLS ON
set LSR2  [$ns node]
set LSR3  [$ns node]
set LSR4  [$ns node]
set LSR5  [$ns node]
```

```
$ns node-config -MPLS OFF
set Node6  [$ns node]
set Node7 [$ns node]
```

เป็นการสร้างโหนดแบบที่เป็น IP และ MPLS โหนด โดยคำสั่ง ***set Node0 [$ns node]*** นั้นจะเป็นการสร้าง IP โหนดโดยกำหนดให้มีชื่อเป็น Node0 ต่อมาคือคำสั่ง ***set Node0 [$ns node]*** ซึ่งเป็นลักษณะเดียวกันกับการสร้าง Node0 โดยมีการกำหนดให้มีชื่อคือ Node1 และถัดไปเป็นคำสั่ง ***$ns node-config -MPLS ON*** โดยเป็นการประกาศให้รู้ว่าทุกโหนดที่มีการสร้างโหนดต่อจากบรรทัดนี้จะเป็นการสร้างโหนดแบบ MPLS โดยที่คำสั่ง ***set LSR1 [$ns node]*** เป็นคำสั่งที่ต่อจากคำสั่ง ***$ns node-config -MPLS ON*** จึงทำให้โหนดที่ชื่อ LSR1 นั้นเป็น MPLS โหนด โดยเป็นเช่นนี้จนถึงโหนดที่ชื่อ LSP5 เพราะคำสั่ง ***$ns node-config -MPLS OFF*** เป็นการปิดการทำงานในการสร้างโหนดแบบ MPLS โดยจะเป็นการประกาศให้รู้ว่าถ้าทำการสร้างโหนดหลังจากคำสั่งนี้จะเป็นการสร้างโหนดแบบปกติซึ่งก็คือการสร้าง Node6 และ Node7 ตามลำดับ

```
#set Link between node
$ns duplex-link $Node0  $LSR2  1Mb  10ms DropTail
$ns duplex-link $Node1  $LSR3  1Mb  10ms DropTail
$ns duplex-link $LSR2   $LSR3  1Mb  10ms DropTail
$ns duplex-link $LSR2   $LSR4  1Mb  10ms DropTail
$ns duplex-link $LSR3   $LSR4  1Mb  10ms DropTail
$ns duplex-link $LSR3   $LSR5  1Mb  10ms DropTail
$ns duplex-link $LSR4   $LSR5  1Mb  10ms DropTail
$ns duplex-link $LSR5   $Node6 1Mb  10ms DropTail
$ns duplex-link $LSR4   $Node7 1Mb  10ms DropTail
```

เป็นการสร้างลิงค์ระหว่างโหนดโดยที่กำหนดให้มีลิงค์แบบดูเพล็กซ์ลิงค์ (duplex-link) จะเป็นการสร้างการเชื่อมต่อระหว่าง 2 โหนดแบบดูเพล็กซ์ คือส่งข้อมูลได้ 2 ทิศทาง ซึ่งจะประกอบไปด้วย 5 พารามิเตอร์คือ โหนดต้น(Source Node) , โหนดปลาย (Destination Node) , แบนวิธ (Bandwidth) , ความหน่วง (Delay) และชนิดของคิว (queue) โดยยกตัวอย่างจากคำสั่ง ***$ns duplex-link $Node0 $LSR2 1Mb 10ms DropTail*** จากคำสั่งจะอธิบายได้ว่า กำหนดให้ Node0 และ LSR2 เชื่อมต่อกันแบบดูเพล็กลิงค์โดยให้มีความเร็วในการส่งข้อมูลที่ 1 เมกะบิท (Mb) ต่อวินาทีและมีความหน่วง (delay) อยู่ที่ 10 มิลลิวินาทีโดยใช้คิวแบบดรอปเทล (DropTail) โดยคิวแบบดรอปเทลจะมีการทำงานดังนี้ ถ้าหาก แพ็คเกต ที่มีการเข้าคิวเกินกว่าที่บัฟเฟอร์ (buffer) จะรองรับได้จะทำการโยนแพ็คเกต (drop แพ็คเกต) ที่เข้าคิวล่าสุดทิ้งไป

```
$ns ldp-request-color    blue
$ns ldp-mapping-color    red
$ns ldp-withdraw-color   magenta
```

```
$ns ldp-release-color    orange
$ns ldp-notification-color  yellow
```

เป็นการกำหนดสีที่ใช้แสดงในโปรแกรม NAM (Network Animator) เพื่อแสดงทราฟฟิกแต่ละเหตุการณ์ที่เกิดขึ้นจาก LDP message จากคำสั่งด้านบนสามารถอธิบายได้ดังนี้ ถ้ามีการส่ง **ldp-request** ให้แสดงทราฟฟิกที่ส่งผ่านระหว่างโหนดเป็นสีฟ้า, **ldp-mapping** ให้แสดงเป็นสีแดง, **ldp-withdraw** ให้แสดงเป็นสีม่วง, **ldp-release** ให้แสดงเป็นสีส้ม และสุดท้าย **ldp-notification** ให้แสดงเป็นสีเหลือง

```
#Create Agent and Application
set Src0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$Src0 attach-agent $udp0
$ns attach-agent $Node0 $udp0
$Src0 set packetSize 500
$Src0 set interval 0.010

set Src1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$Src1 attach-agent $udp1
$ns attach-agent $Node1 $udp1
$Src1 set packetSize 500
$Src1 set interval 0.010

set Dst6 [new Agent/Null]
$ns attach-agent $Node6 $Dst6

set Dst7 [new Agent/Null]
$ns attach-agent $Node7 $Dst7
```

จากคำสั่งด้านบนนั้นเป็นการกำหนดให้ Node0 ซึ่งเป็น IP โดยมีรูปแบบเป็น CBR agent คือจะมีการส่งข้อมูลด้วยความเร็วคงที่โดยกำหนดให้มีชื่อ Src0 , Src1 ได้กำหนดให้แพ็คเกตมีขนาด 500 ไบท์ และกำหนดระยะเวลาระหว่างการส่งแต่ละแพ็คเกตเป็น 0.01 วินาที และต่อมาเป็นการสร้าง Null เอเจนต์ กำหนดให้มีชื่อเป็น Dst0 และ Dst1 และทำการกำหนด Null เอเจนต์ไปที่ Node7 และ Node6 ซึ่งเป็นการกำหนดให้เป็นโหนดปลายทาง

```
$ns connect $Src0 $Dst0
$ns connect $Src1 $Dst1
```

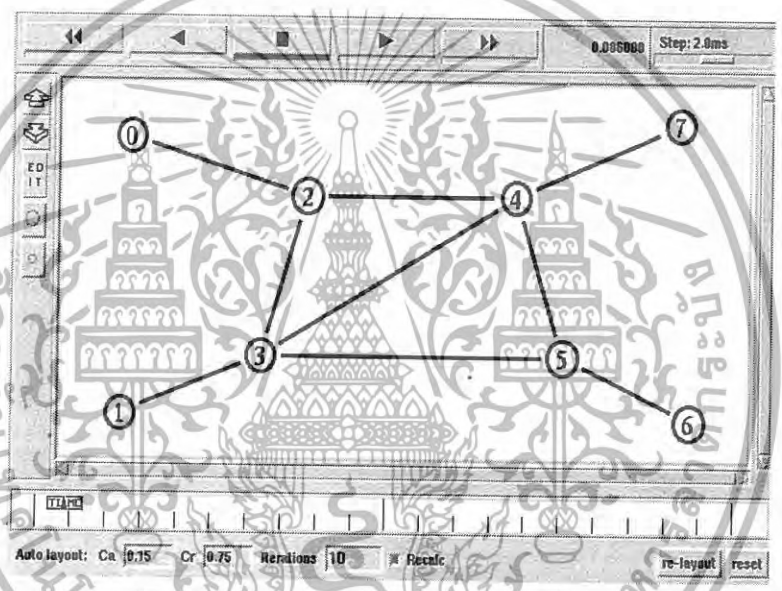
จากคำสั่งด้านบนเป็นการเชื่อมกันระหว่าง CBR เอเจนต์กับ NULL เอเจนต์ทั้งสองตัวซึ่งก็คือการเชื่อมระหว่างโหนดต้นทางคือ Src0 กับ Node ปลายทางซึ่งคือ Dst0 และเชื่อมระหว่างโหนดต้นทางคือ Src1 กับ Dst1

3.6 การจำลองเหตุการณ์ที่เกิดขึ้นจากเครือข่าย MPLS

ในส่วนนี้จะเป็นตัวอย่างการจำลองการทำงานของ LDP บนเครือข่าย MPLS

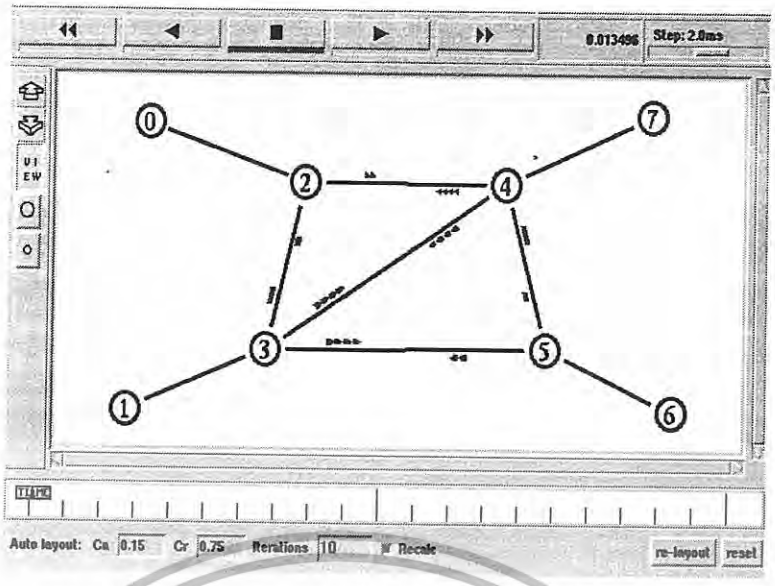
3.6.1 การทำงานของ Flow aggregation

Flow Aggregation คือการกำหนดเส้นทางการส่งของข้อมูลในลักษณะที่ให้ไปรวมในเส้นทางที่กำหนดไว้เสียก่อน โดยเราต้องระบุเส้นทางที่จะนำไปรวม โดยจะระบุด้วย FEC ซึ่งปกติเส้นทางในการส่งข้อมูลหรือ LSP จะอ้างอิงตามเส้นทางที่ได้จากตารางเราติ้ง โดยมีการสร้างโทโพโลจีในการจำลองเครือข่าย MPLS ดังนี้



รูปที่ 3.8 แสดงเหตุการณ์ก่อนที่จะเริ่มการจำลองเครือข่าย MPLS

โดยกำหนดให้โหนด 2 ถึงโหนด5 เป็น LSR นอกนั้นเป็นโหนดปกติ โดยในอันดับแรกได้มีการรัน ldp agent ที่ LSR แต่ละตัว ซึ่งจะเกิดการส่ง ldp mapping message เพื่อที่จะใช้ในการแลกเปลี่ยนข้อมูลและลาเบลของแต่ละ LSR และบันทุกลง LIB และเพื่อเป็นการสร้าง LSP ด้วย

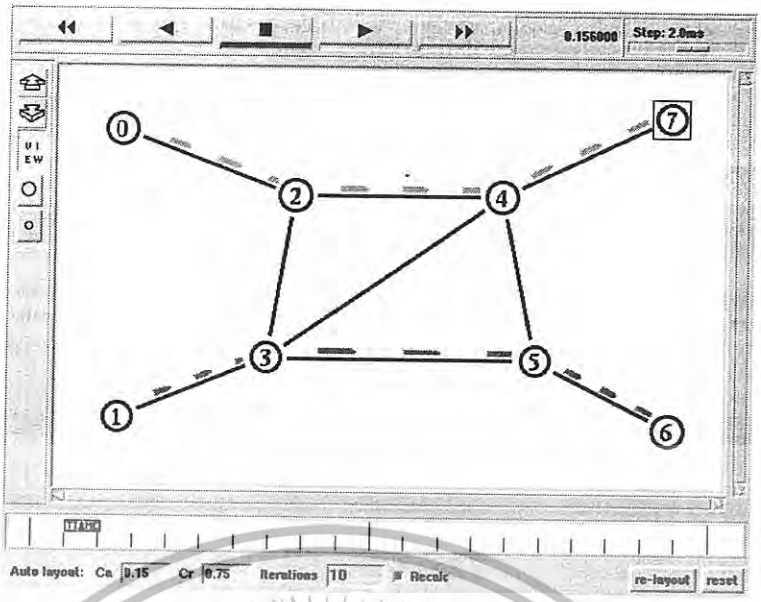


รูปที่ 3.9 แสดงการส่ง ldp mapping message

ที่เวลา 0.0134 วินาทีจะมีการส่งข้อความการเปรียบเทียบ (mapping message) ระหว่าง MPLS โหนดที่เชื่อมต่อกันซึ่งแสดงในรูปที่ 3.8 เพราะว่าในขณะที่เริ่มการจำลองเครือข่าย MPLS นั้น LSR แต่ละตัวยังไม่มีข้อมูลเกี่ยวกับเส้นทางว่าจะติดต่อกันได้อย่างไรบ้างในแต่ละโหนด จึงทำให้มีการส่งข้อความการเปรียบเทียบ หากันก่อนในช่วงแรก เพื่อสร้างตาราง LIB และสร้าง LSP

```
$ns at 0.1"$Src0 start"
$ns at 0.1"$Src1 start"
```

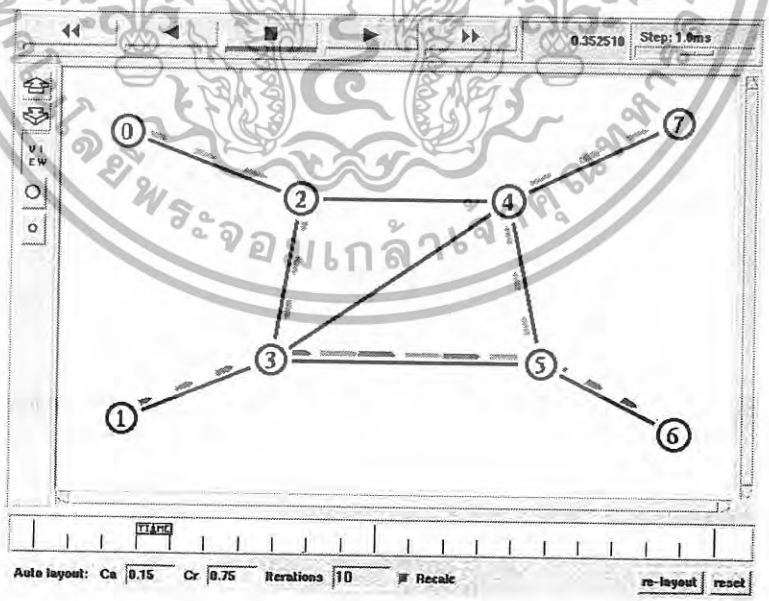
จากคำสั่งด้านบนจะทำให้ Src0 และ Src1 ซึ่งคือโหนด 0 และ 1 มีการส่งข้อมูลไปยังปลายทางคือโหนด 7 และ 6ที่เวลา 0.1 วินาทีโดยคำสั่ง start คือเริ่มให้มีการส่งข้อมูลออกมา ซึ่งในวินาทีที่ 0.1 เป็นต้นไปจะเริ่มมีการส่งแพ็คเกตจากโหนดต้นทางที่ 0 และ 1 ไปยังโหนดปลายทางที่โหนด 6 และ 7 ซึ่งแสดงในรูปที่ 3.10



รูปที่ 3.10 แสดงการส่งแพ็คเกตจากโหนดต้นทางไปยังโหนดปลายทาง

```
$ns at 0.3"[$LSR2 get-module MPLS] flow-aggregation 7-15-1"
```

จากคำสั่งด้านบนที่เวลา 0.3 วินาทีเป็นการสั่งให้ LSR2 ที่มีการส่งข้อมูลไปยัง โหนดที่ 7 นั้นให้มีการรวมการไหลของข้อมูล (Flow Aggregate) โดยให้มีการส่งข้อมูลไปรวมกันที่เส้นทางไปยัง LSR5 เสียก่อนและส่งไปยังปลายทางคือโหนดที่ 7 ซึ่งแสดงได้ดังรูปที่ 3.11



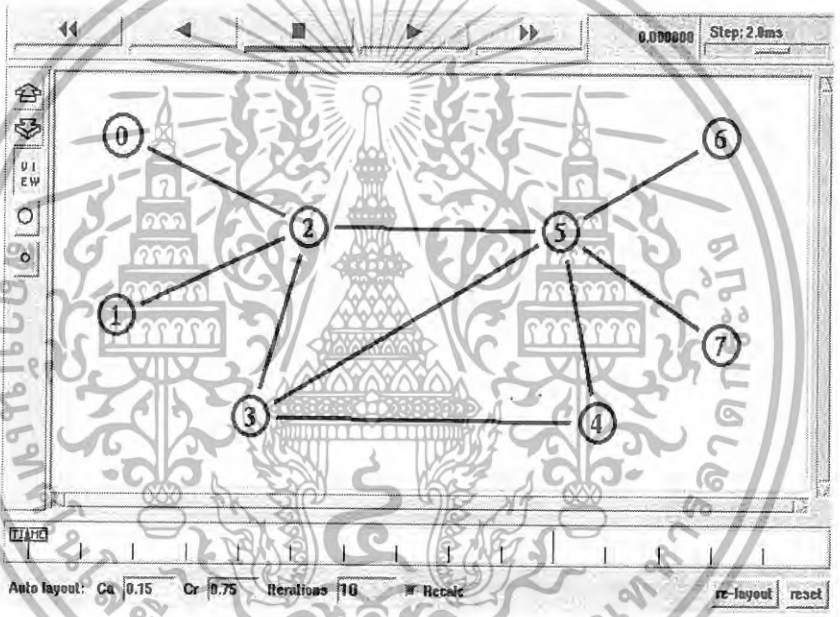
รูปที่ 3.11 ที่เวลา 0.35 วินาทีเป็นการรวมการไหลของข้อมูลให้ไปรวมที่ LSR5 และส่งต่อไปยังโหนดปลายทาง

ซึ่งจากรูปจะเห็นได้ว่า LSR2 หากจะส่งไปยังโหนด 7 จะส่งไปทางลิงค์บน แต่เรากำหนดให้รวมไปยังเส้นทางที่จะไป fec5 ทำให้ LSR2 จะส่งไปยัง LSR5 ก่อน ซึ่ง LSR5 จะส่งข้อมูลไปยังปลายทางต่อไป

3.6.2 การทดลองการทำ Explicit Route และการส่ง ldp release message

การทำ explicit-route คือการสร้าง explicit-route LSP หรือ ER-LSP ซึ่งจะเป็นการกำหนดเส้นทางที่จะส่งโดยจะระบุโหนดที่จะให้ส่งผ่านเลยโดยกำหนดด้วยค่า FEC ซึ่งจะมีการสร้างค่าในตาราง ERB สำหรับหาเส้นทางในการส่งข้อมูลผ่าน ER-LSP ที่สร้างขึ้น

โดยในการทดลองจะสร้างโทโพโลจีดังรูปที่ 3.12 ซึ่งกำหนดให้โหนด 2 ถึง 5 เป็น LSR เท่านั้น นอกจากนี้จะเป็นโหนดปกติ

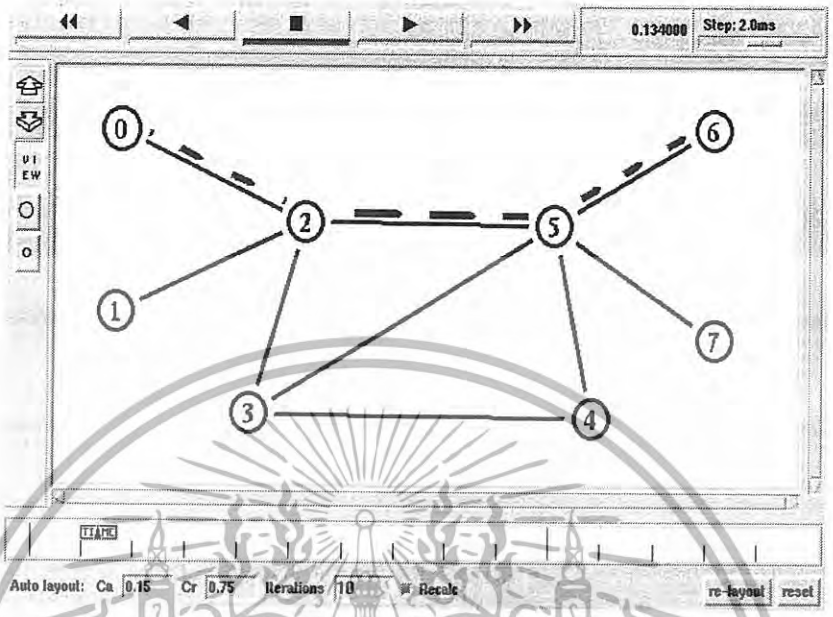


รูปที่ 3.12 แสดงโทโพโลจีของการทดลอง

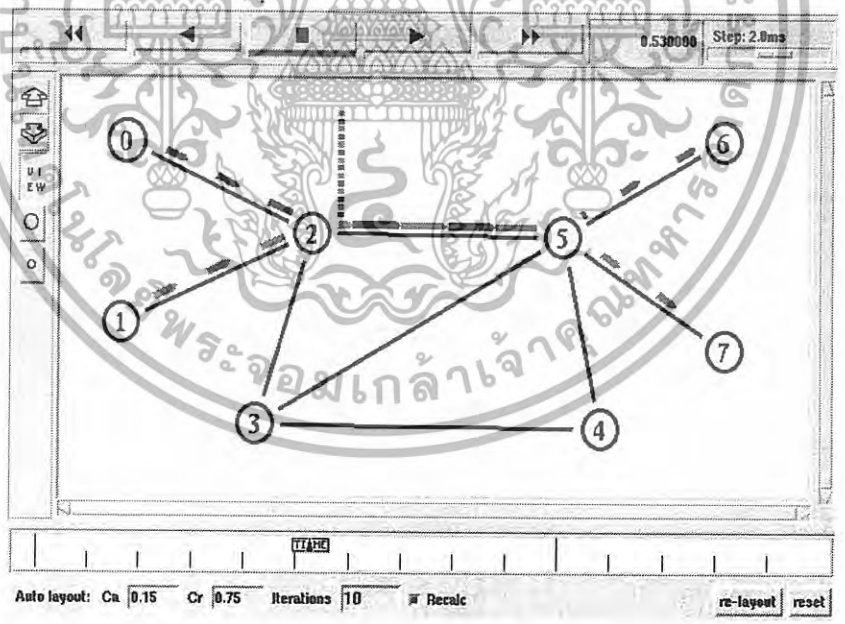
โดยในการทดลองการส่งข้อมูลจากโหนด 0 ไปยังโหนด 6 และจากโหนด 1 ไปยังโหนด 7 นั้นจะมีการใช้ลิงค์ด้านบนร่วมกัน เหตุผลที่ต้องกำหนดให้เป็นลักษณะนี้ก็เพื่อที่จะพยายามทำให้เกิดเหตุการณ์ที่ทำให้เกิดความคับคั่งของสัญญาณหรือที่เรียกว่า congestion ซึ่งทำให้ส่งข้อมูลไปได้ช้าเพราะเกิดการเข้าคิวในบัฟเฟอร์ของลิงค์ ซึ่งเราอาจทำการหลีกเลี่ยงโดยการระบุเส้นทางของบางทราฟฟิกด้วยการทำ Explcit Route

ในการทดลองที่เวลา 0.1 วินาทีได้กำหนดให้โหนด 0 นั้นทำการส่งข้อมูลไปก่อนซึ่งจะมีการส่งข้อมูลผ่านลิงค์ระหว่าง LSR2 กับ LSR5 ซึ่งแสดงได้ในรูปที่ 3.13 และในขณะเดียวกันวินาทีที่ 0.4 โหนด 1 ได้มีการส่งข้อมูลผ่านทางลิงค์ LSR2 กับ LSR5 เหมือนกัน ซึ่งข้อมูลขาเข้า

มากกว่าข้อมูลขาออก จึงทำให้ข้อมูลที่ส่งมานั้นต้องเข้าไปรออยู่ในคิวเพื่อรอเวลาที่จะถูกส่งข้อมูลออกมาดังรูป 3.14



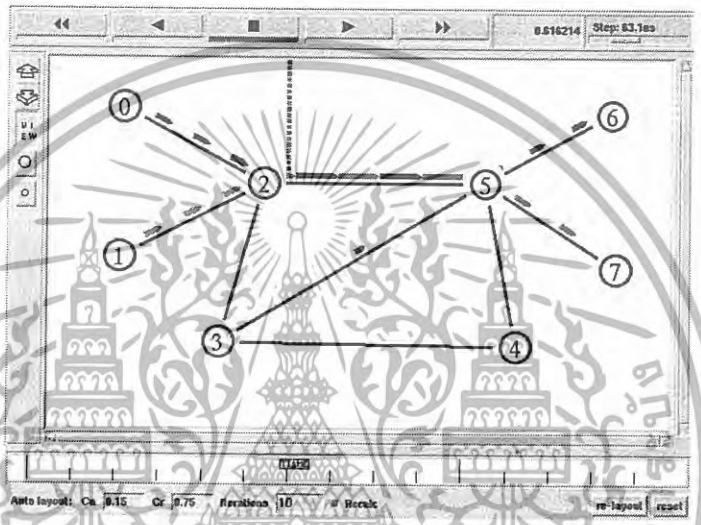
รูปที่ 3.13 แสดงการส่งข้อมูล ณ.เวลาที่ 0.2



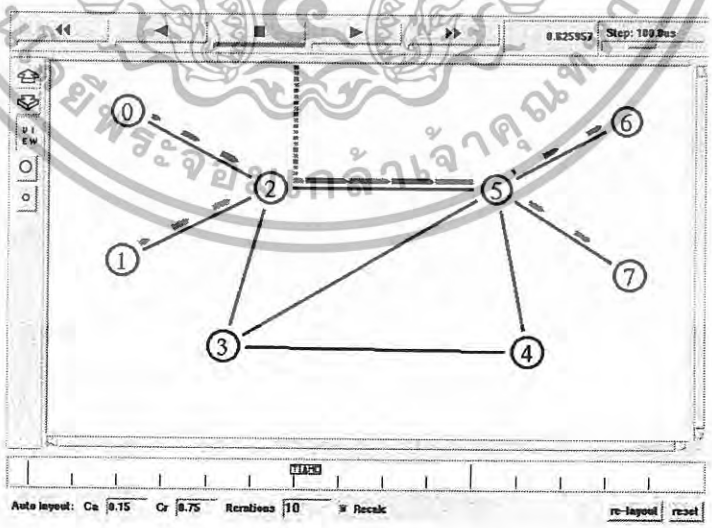
รูปที่ 3.14 แสดงการส่งข้อมูลของโหนด 0 และโหนด 1 โดยที่แพ็คเกตได้มีการเข้าคิว

```
$ns at 0.6 "[$LSR2 get-module MPLS] make-explicit-route 7
2_3_5 3000 -1"
```

ด้วยคำสั่งดังกล่าวได้มีการสร้างเส้นทางใหม่โดยกำหนดที่ LSR2 ว่าให้ข้อมูลที่จะไปยังโหนด7 ให้ส่งผ่านLSR3 LSR5 โดยกำหนดให้ flow id มีค่าเป็น 3000 ซึ่งในการกำหนดเส้นทางใหม่จะต้องมีการส่ง ldp-request-message เพื่อเป็นการร้องการสร้างเส้นทางใหม่ดังรูปที่ 3.15 ต่อจากนั้น LSR5 ซึ่งเป็นปลายทางของเส้นทางได้มีการส่ง ldp-mapping-message กลับมาเพื่อเป็นการแลกเปลี่ยนข้อมูลในการติดต่อ ซึ่งแสดงได้ในรูปที่ 3.16



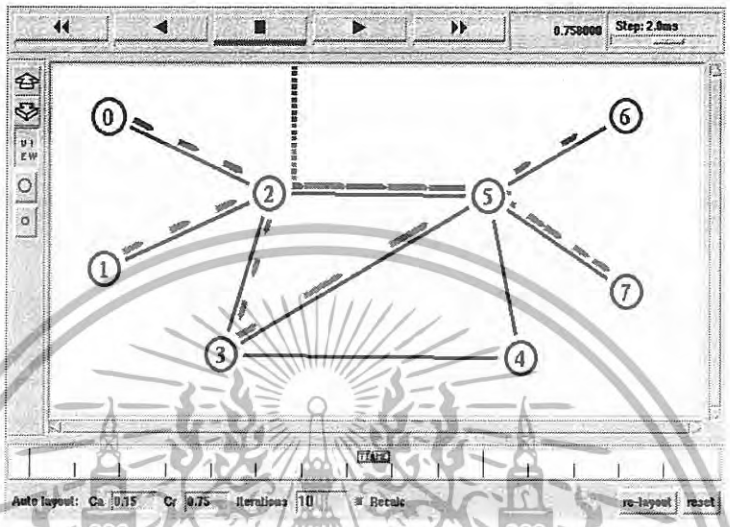
รูปที่ 3.15 แสดงการส่ง ldp-request-messageเพื่อขอลาเบลใหม่



รูปที่ 3.16 แสดงการส่ง ldp-mapping-messageเพื่อเป็นการแลกเปลี่ยนข้อมูลในการติดต่อระหว่างกัน

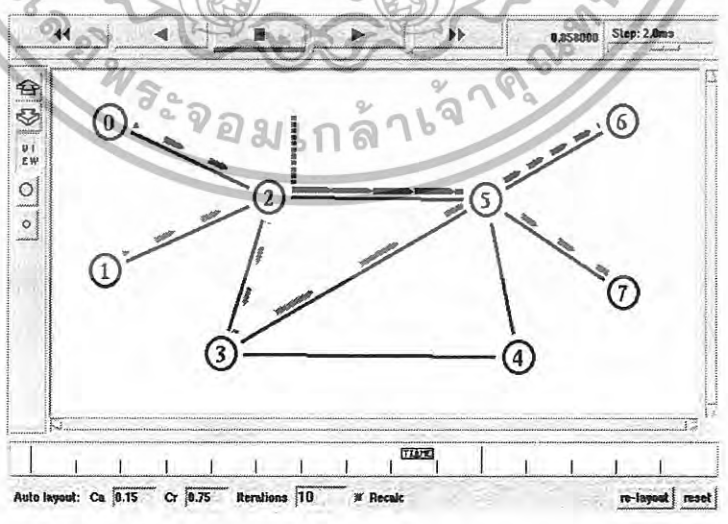
```
$ns at 0.7 "[$LSR2 get-module MPLS] flow-erlsp-install 7 -1
3000"
```

จากนั้นคำสั่งดังกล่าวจะเป็นการ install คือเป็นการใช้ ER-LSP mด้าร้างไว้โดยระบุ flow id คือ 3000 ซึ่งแสดงได้ดังรูปที่ 3.17



รูปที่ 3.17 แสดงการส่งข้อมูลตามเส้นทางที่ได้มีการกำหนดไว้โดยการทำ Explicit Route

จากรูปที่ 3.17 โหนด 1 ได้มีการส่งข้อมูลโดยใช้เส้นทางใหม่โดยส่งผ่าน LSR2 LSR3 LSR5 ไปยังโหนด 7 ส่วนข้อมูลที่ยังค้างอยู่ในคิวของลิงค์ 2 และ 5ก็ยังคงมีการส่งเส้นทางเดิมอยู่จนหมดคิว



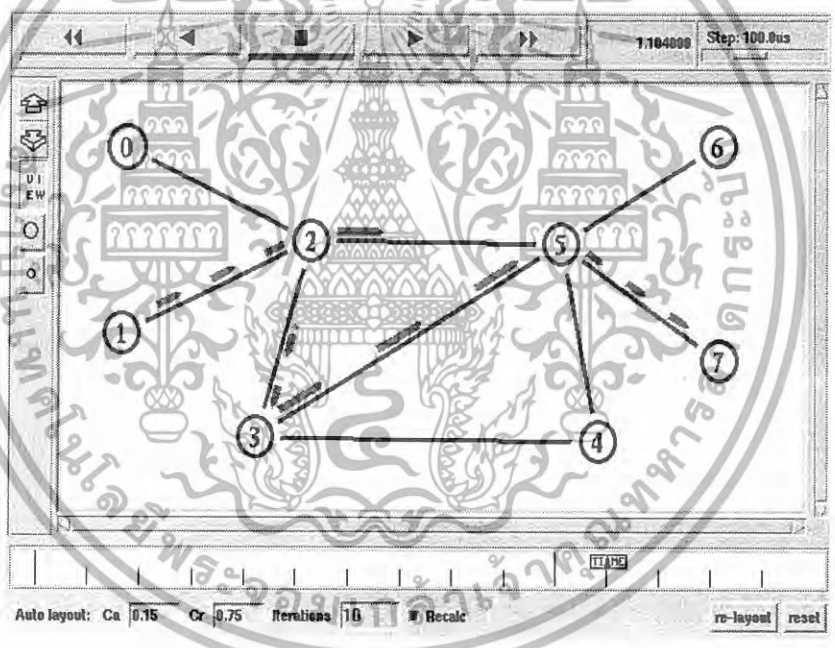
รูปที่ 3.18 แสดงการส่งข้อมูลที่ไม่มีข้อมูลจากโหนด 1 ค้างอยู่ในคิว

จากรูปที่ 3.18 นั้นจะเห็นว่าข้อมูลที่ส่งจากโหนด 0 ไปยังโหนด 6 นั้นยังมีข้อมูลค้างอยู่ในคิวอยู่ซึ่งเกิดตั้งแต่มีการใช้ลิงค์ร่วมกันในตอนแรก ทำให้ข้อมูลใหม่จาก 0 ที่ส่งมาก็ต้องรอคิวเช่นเดิม แต่จะไม่มีข้อมูลจากโหนด 1 อยู่ในคิวแล้วเนื่องจากที่ค้างตั้งแต่ก่อนเปลี่ยนเส้นทางได้ถูกส่งออกไปจนหมดแล้ว

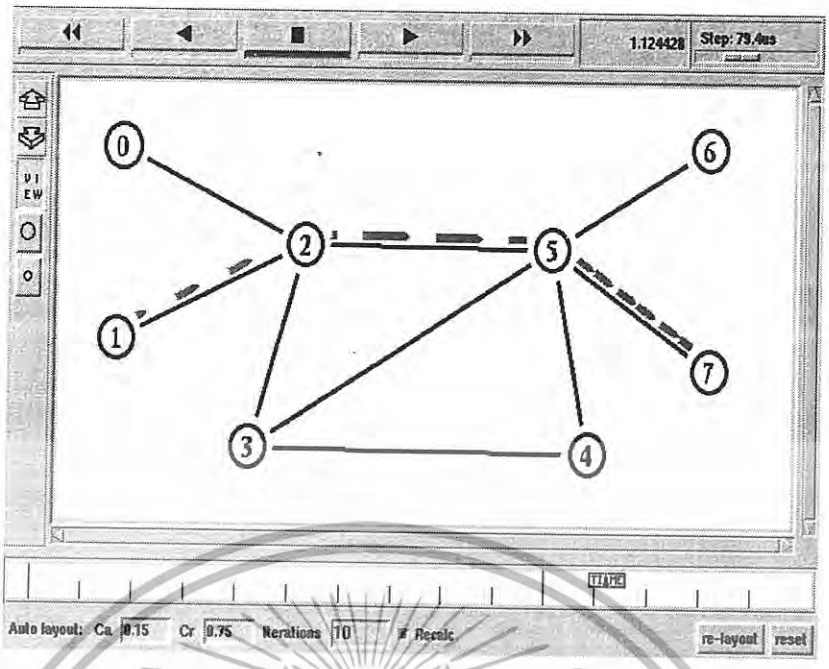
จากนั้นหากในกรณีที่เราต้องการยกเลิก ER-LSP ที่กำหนดไว้ก็สามารถทำได้ด้วย Release message ซึ่งมีรูปแบบคำสั่งดังนี้

```
$ns at 1.1 "[$LSR2 get-module MPLS] ldp-trigger-by-release 7
-1 3000"
```

ซึ่งทำให้เกิดการส่งข้อความดังรูปที่ 3.19 และจะทำให้การส่งข้อมูลกลับมาใช้เส้นทางเก่าซึ่งจะได้ผลดังรูปที่ 3.20



รูปที่ 3.19 แสดงการส่ง ldp-release-message

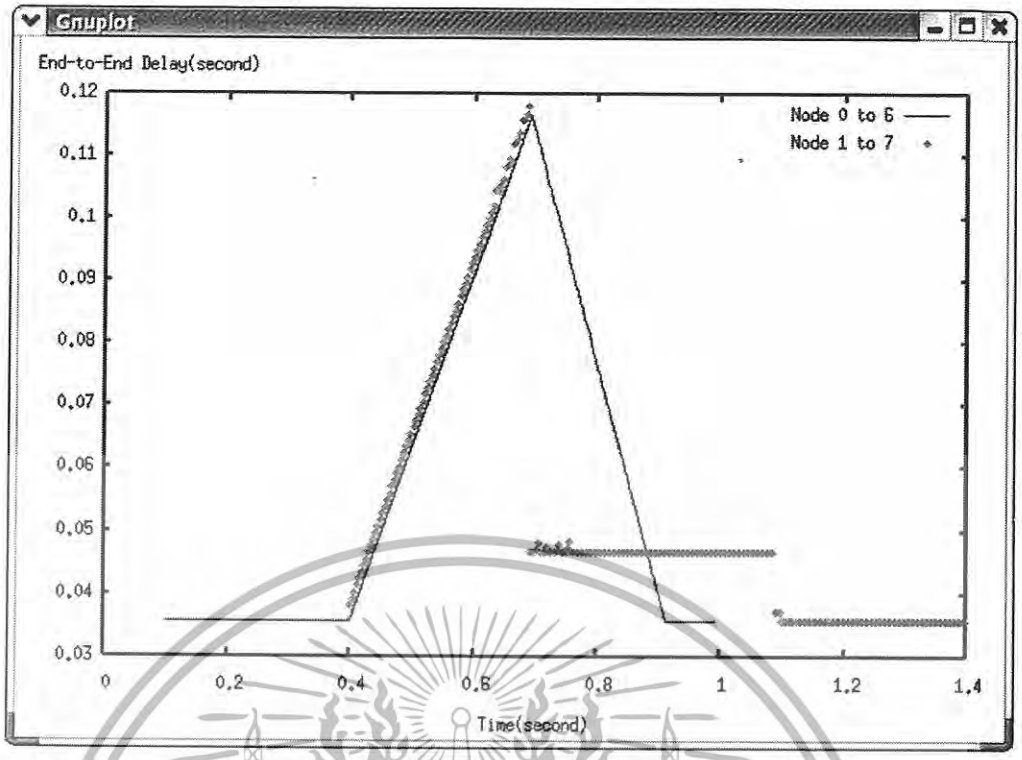


รูปที่ 3.20 แสดงการส่งข้อมูลหลังจากได้มีการส่ง ldp-release-message

3.6.3 เปรียบเทียบ End-to-End Delay ของการส่งข้อมูลในแต่ละเส้นทาง

โดยในส่วนนี้จะยกตัวอย่างกรณีที่เราทำการ explicit route ในกรณีที่เกิด congestion ขึ้น ซึ่งจะใช้ topology เช่นเดียวกับในหัวข้อที่ 3.6.2

โดยเราจะกำหนดให้โหนด 0 ส่งข้อมูลก่อนและตามด้วยโหนด 1 ซึ่งจะมีการใช้ลิงค์ร่วมกัน โดยเรากำหนดแบนวิธเพียงพอให้เกิดการรอคิวในลิงค์ ซึ่งเราจะทำการสร้าง ER-LSP สำหรับข้อมูลที่ส่งจากโหนด 1 เพื่อให้สามารถส่งข้อมูลได้เร็วขึ้น จากนั้นเมื่อโหนด 0 หยุดส่ง เราจะทำการ release ER-LSP mร้างขึ้นเพื่อให้กลับไปใช้เส้นทางเดิม

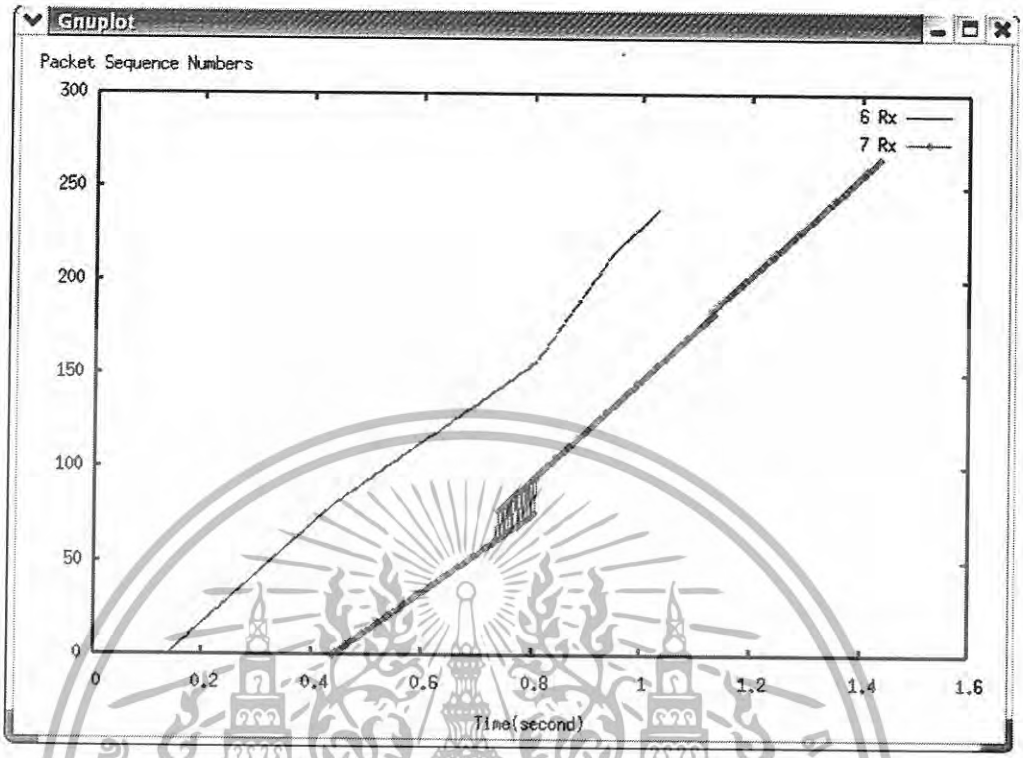


รูปที่ 3.21 แสดงการเปรียบเทียบ End-to-End delay ระหว่างการส่งข้อมูลจากโหนด 0 ไปยังโหนด 6 และจากโหนด 1 ไปยังโหนด 7

จากรูปที่ 3.21 จะเห็นว่าในช่วงแรกนั้นค่า end-to-end delay จากการส่งข้อมูลจากโหนด 0 ไปยังโหนด 6 จะมีค่าคงที่ เนื่องจากมีการใช้ลิงค์เพียงทราฟฟิกเดียวและไม่เกิดการรอคิว จากนั้นเมื่อเข้าสู่วินาทีที่ 4 ได้ให้มีการส่งข้อมูลจากโหนด 1 ซึ่งทำให้ใช้ลิงค์ร่วมกันและเกิดคิวในบัฟเฟอร์ จึงทำให้ค่า end-to-end delay ของทั้ง 2 ทราฟฟิกมีแนวโน้มสูงขึ้นไปเรื่อยๆ จนกระทั่งวินาทีที่ 0.7 ค่า end-to-end delay ของทราฟฟิกจากโหนด 1 จะลดลงเพราะว่าได้มีการเปลี่ยนเส้นทางในการส่งข้อมูลจากโหนด 1 ไปยังโหนด 7 โดยใช้เส้นทางใหม่ตามที่ได้มีการกำหนดโดยใช้วิธี Explicit Route ไว้ ขณะที่ค่าความหน่วงจากโหนด1 จะค่อยๆลดลงจนมาอยู่ในระดับเดิมเนื่องจากใช้ลิงค์แต่เพียงผู้เดียวและข้อมูลในคิวก็สามารถส่งออกมาได้เร็วกว่าข้อมูลที่เข้าไป

จากนั้นวินาทีที่ 1 ได้หยุดการส่งข้อมูลจากโหนด 0 จึงทำให้ลิงค์ว่าง และวินาที 1.1 ได้ทำการ release เส้นทางที่สร้างไว้ จึงกลับไปใช้เส้นทางเดิม ทำให้มีค่าความหน่วงอยู่ในระดับเดียวกับช่วงแรกของข้อมูลที่ส่งจากโหนด 0 ที่ใช้ลิงค์ผู้เดียว และไม่เกิดการรอคิว

3.6.4 เปรียบเทียบ แพ็คเกต Sequence Number ของการส่งข้อมูลในแต่ละเส้นทาง



รูปที่ 3.22 แสดงการเปรียบเทียบ Sequence Number ระหว่างการส่งข้อมูลจากโหนด 0 ไปยังโหนด 6 และจากโหนด 1 ไปยังโหนด 7

จากรูปเป็นกราฟแสดงลำดับของแพ็คเกตที่โหนด 6 และ7 ได้รับ โดยช่วงกราฟที่มีความชันสูงที่สุดจะหมายถึงแพ็คเกตลำดับดังกล่าวถูกส่งถึงปลายทางด้วยความเร็วสูงสุด โดยจะสังเกตได้ว่าช่วงที่เร็วที่สุดคือช่วงเวลาที่มีการส่งข้อมูลเพียงทราฟฟิกเดียวเท่านั้น

โดยในช่วงวินาทีที่ 0.7 ของทราฟฟิกจากโหนด 1 จะเป็นผลจากการใช้เส้นทางใหม่ที่กำหนดไว้ โดยจะเห็นได้ว่าลำดับของแพ็คเกตที่ปลายทางได้รับจะไม่ถูกลำดับเนื่องจากแพ็คเกตใหม่ที่ถูกส่งไปในเส้นทางใหม่จะถึงก่อนแพ็คเกตที่ยังค้างอยู่ในคิวของลิงค์ในเส้นทางเดิม

3.7 MPLS เทรซไฟล์เพื่อศึกษาการสร้าง LSP

จากหัวข้อที่ผ่านมาได้อธิบายถึงการสร้างเส้นทางที่เรียกว่า LSP ซึ่งจะเกิดจากการที่ ingress LSRนำที่อยู่ปลายทางของแพ็คเกตที่ได้รับมาเทียบกับ FEC กับตาราง PFT เพื่อจะนำไปเทียบกับตาราง KIB เพื่อทำการใส่ลาเบลเข้าไปในแพ็คเกตที่จะส่งข้อมูลเข้าไปใน MPLS โดเมน โดยในหัวข้อนี้จะพูดถึงการจำลองเครือข่าย MPLS โดยจะมุ่งเน้นไปการวิเคราะห์ผลการทดลองในรูปแบบของ MPLS เทรซไฟล์ ซึ่งเป็นเทรซไฟล์อีกรูปแบบหนึ่งนอกเหนือจากเทรซไฟล์ในรูปแบบปกติดังที่ได้กล่าวไปในหัวข้อของการแสดงผลและการเทรซในตอนต้น

ซึ่ง MPLS เทรซไฟล์จะเป็นการบันทึกผลข้อมูลจากออมเจคต่างๆที่อยู่ใน MPLS โหนดที่อยู่ใน MPLS โดเมนเท่านั้น โดยเทรซไฟล์จะแสดงถึงกระบวนการทำงานต่างของเทคโนโลยี MPLS และโปรโตคล LDP เช่น การแลกเปลี่ยนลาเบล การใส่ลาเบล การสับเปลี่ยนลาเบล หรือการถอดลาเบล เป็นต้น

โดยตัวอย่างสคริป TCL ที่จะใช้ในการจำลองเครือข่าย MPLSเพื่อศึกษาเทรซไฟล์ในหัวข้อนี้จะเป็นไปดังนี้

ตัวอย่าง สคริปการสร้างการจำลองเครือข่าย MPLS

```
#Create a simulator object
set ns [new Simulator]

Classifier/Addr/MPLS set control_driven_ 1

#Turn on all traces to stdout
Agent/LDP set trace_ldp_ 1
Classifier/Addr/MPLS set trace_mpls_ 1

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
global ns nf tf
$ns flush-trace
#Close the trace file
close $nf
#Close the Trace file

close $tf
#Execute nam on the trace file
exec nam out.nam &
exit 0
```

ตัวอย่าง สคริปการสร้างการจำลองเครือข่าย MPLS (ต่อ)

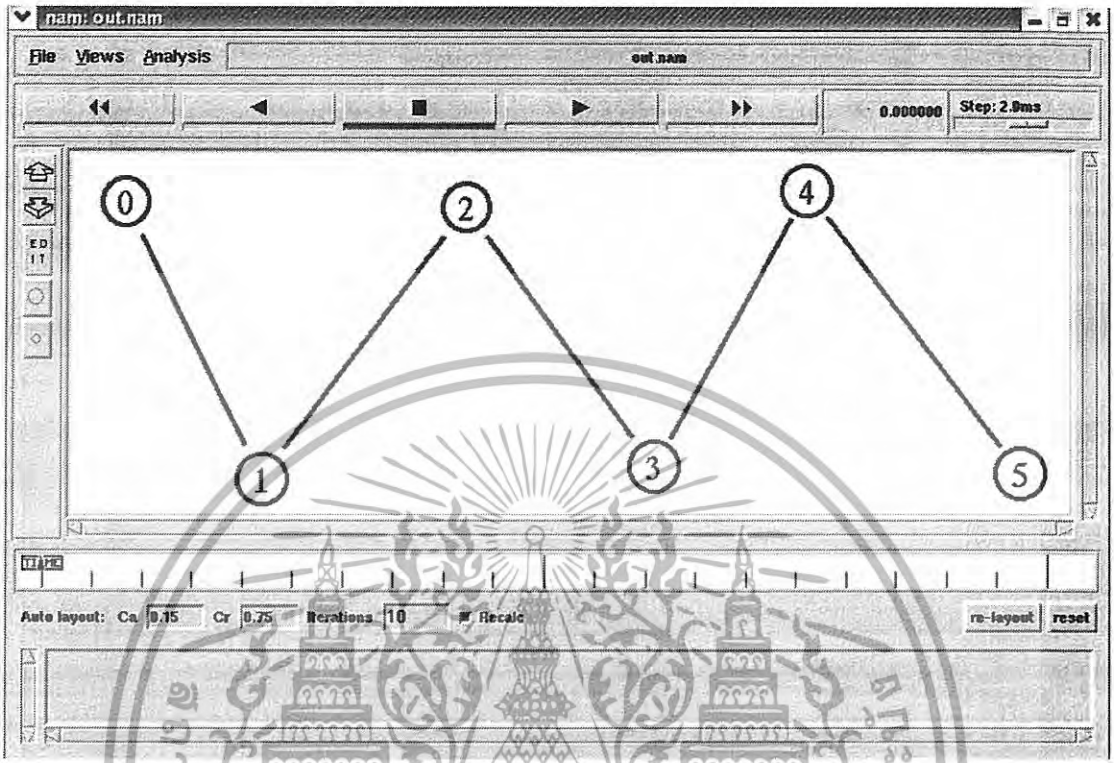
```
}

set Node0 [$ns node]
$ns node-config -MPLS ON
set LSR1 [$ns node]
set LSR2 [$ns node]
set LSR3 [$ns node]
set LSR4 [$ns node]
$ns node-config -MPLS OFF
set Node5 [$ns node]

$ns duplex-link $Node0  $LSR1 1Mb  10ms DropTail
$ns duplex-link $LSR1   $LSR2 1Mb  10ms DropTail
$ns duplex-link $LSR2   $LSR3 1Mb  10ms DropTail
$ns duplex-link $LSR3   $LSR4 1Mb  10ms DropTail
$ns duplex-link $LSR4   $Node5 1Mb  10ms DropTail

for {set i 1} {$i < 5} {incr i} {
    for {set j [expr $i+1]} {$j < 5} {incr j} {
        set a LSR$i
        set b LSR$j
        eval $ns LDP-peer $$a $$b
    }
}

set Src0 [new Agent/CBR]
$ns attach-agent $Node0 $Src0
$Src0 set packetSize_ 500
$Src0 set interval_ 0.010

set Dst0 [new Agent/Null]
$ns attach-agent $Node5 $Dst0

$ns connect $Src0 $Dst0

$ns at 0.0104 "[$LSR1 get-module MPLS] pft-dump"
$ns at 0.0104 "[$LSR1 get-module MPLS] lib-dump"
$ns at 0.0140 "[$LSR2 get-module MPLS] pft-dump"
$ns at 0.0104 "[$LSR2 get-module MPLS] lib-dump"
$ns at 0.0104 "[$LSR3 get-module MPLS] pft-dump"
$ns at 0.0104 "[$LSR3 get-module MPLS] lib-dump"
$ns at 0.0104 "[$LSR4 get-module MPLS] pft-dump"
$ns at 0.0104 "[$LSR4 get-module MPLS] lib-dump"

$ns at 0.0107 "[$LSR2 get-module MPLS] pft-dump"
$ns at 0.0107 "[$LSR2 get-module MPLS] lib-dump"
$ns at 0.0107 "[$LSR3 get-module MPLS] pft-dump"
$ns at 0.0107 "[$LSR3 get-module MPLS] lib-dump"
```

ตัวอย่าง สคริปการสร้างการจำลองเครือข่าย MPLS (ต่อ)

```
$ns at 0.022 "[$LSR1 get-module MPLS] pft-dump"
$ns at 0.022 "[$LSR1 get-module MPLS] lib-dump"
$ns at 0.022 "[$LSR2 get-module MPLS] pft-dump"
$ns at 0.022 "[$LSR2 get-module MPLS] lib-dump"
$ns at 0.022 "[$LSR3 get-module MPLS] pft-dump"
$ns at 0.022 "[$LSR3 get-module MPLS] lib-dump"
$ns at 0.022 "[$LSR4 get-module MPLS] pft-dump"
$ns at 0.022 "[$LSR4 get-module MPLS] lib-dump"

$ns at 0.032 "[$LSR1 get-module MPLS] pft-dump"
$ns at 0.032 "[$LSR1 get-module MPLS] lib-dump"
$ns at 0.032 "[$LSR2 get-module MPLS] pft-dump"
$ns at 0.032 "[$LSR2 get-module MPLS] lib-dump"
$ns at 0.032 "[$LSR3 get-module MPLS] pft-dump"
$ns at 0.032 "[$LSR3 get-module MPLS] lib-dump"
$ns at 0.032 "[$LSR4 get-module MPLS] pft-dump"
$ns at 0.032 "[$LSR4 get-module MPLS] lib-dump"

$ns at 0.1 "$Src0 start"

$ns at 2.0 "$Src0 stop"

$ns at 2.0 "[$LSR1 get-module MPLS] pft-dump"
$ns at 2.0 "[$LSR1 get-module MPLS] erb-dump"
$ns at 2.0 "[$LSR1 get-module MPLS] lib-dump"
$ns at 2.0 "[$LSR2 get-module MPLS] pft-dump"
$ns at 2.0 "[$LSR2 get-module MPLS] erb-dump"
$ns at 2.0 "[$LSR2 get-module MPLS] lib-dump"
$ns at 2.0 "[$LSR3 get-module MPLS] pft-dump"
$ns at 2.0 "[$LSR3 get-module MPLS] erb-dump"
$ns at 2.0 "[$LSR3 get-module MPLS] lib-dump"
$ns at 2.0 "[$LSR4 get-module MPLS] pft-dump"
$ns at 2.0 "[$LSR4 get-module MPLS] erb-dump"
$ns at 2.0 "[$LSR4 get-module MPLS] lib-dump"

$ns at 2.0 "finish"
#
# The last line finally starts the simulation
#
$ns run
```

โดยจากสคริปดังกล่าวจะเป็นการสร้างการจำลองเครือข่าย MPLS โดยจะกำหนดให้มีโหนดต้นทางหนึ่งโหนดและโหนดปลายทางหนึ่งโหนด และกำหนดให้มี LSR ใน MPLS โดเมน 4 โหนด

ต่อกับในลักษณะอนุกรม ดังนั้นจะประกอบด้วยทั้งหมด 6 โหนด ซึ่งเมื่อแสดงผลด้วย NAM จะได้ผลดังรูปที่ 3.23



รูปที่ 3.23 การแสดงผลด้วย NAM จาก mpls-6nodeTest.tcl

จากสคริปจะกำหนดให้โหนด 0 ในรูปเป็น โหนดต้นทางและโหนด 5 เป็น โหนดปลายทาง โดยใช้ทราฟฟิก CBR วินาทีที่ 0.1 ถึง 2.0 ซึ่งได้มีการสร้างการเทรซ MPLS ซึ่งจะได้ผลดังต่อไปนี้

ตัวอย่างเทรซไฟล์

```
0 1(1->2): U -1 L3 -1 -1 -1 0
0 1(1->2): U -1 L3 -1 -1 -1 0
0 2(2->1): U -1 L3 -1 -1 -1 0
0 2(2->3): U -1 L3 -1 -1 -1 0
0 3(3->2): U -1 L3 -1 -1 -1 0
0 3(3->4): U -1 L3 -1 -1 -1 0
0 4(4->3): U -1 L3 -1 -1 -1 0
0 4(4->3): U -1 L3 -1 -1 -1 0
0.010336 2(1->2): U -1 L3 -1 -1 -1 0
0.010336 2: 1 (Mapping 1) 0 0 *_1  [-1 *]  [-1 * -1]
```

ตัวอย่างเทรซไฟล์ (ต่อ)

```
0.010336: <mapping-msg> 1 -> 2 : fec(0), label(0) 1
0.010336 2(2->3): U -1 L3 -1 -1 -1 0
0.010336 1(2->1): U -1 L3 -1 -1 -1 0
0.010336 1: 2 (Mapping 1) 2 0 *_2 [-1 *] [-1 * -1]
0.010336: <mapping-msg> 2 -> 1 : fec(2), label(0) 2
0.010336 3(2->3): U -1 L3 -1 -1 -1 0
0.010336 3: 2 (Mapping 1) 2 0 *_2 [-1 *] [-1 * -1]
0.010336: <mapping-msg> 2 -> 3 : fec(2), label(0) 2
0.010336 3(3->4): U -1 L3 -1 -1 -1 0
0.010336 2(3->2): U -1 L3 -1 -1 -1 0
0.010336 2: 3 (Mapping 1) 3 0 *_3 [-1 *] [-1 * -1]
0.010336: <mapping-msg> 3 -> 2 : fec(3), label(0) 3
0.010336 2(2->1): U -1 L3 -1 -1 -1 0
0.010336 4(3->4): U -1 L3 -1 -1 -1 0
0.010336 4: 3 (Mapping 1) 3 0 *_3 [-1 *] [-1 * -1]
0.010336: <mapping-msg> 3 -> 4 : fec(3), label(0) 3
0.010336 3(4->3): U -1 L3 -1 -1 -1 0
0.010336 3: 4 (Mapping 1) 4 0 *_4 [-1 *] [-1 * -1]
0.010336: <mapping-msg> 4 -> 3 : fec(4), label(0) 4
0.010336 3(3->2): U -1 Push(penultimate) 2 0 32 0
0.010672 2(1->2): U -1 L3 -1 -1 -1 0
0.010672 2: 1 (Mapping 2) 1 0 *_1 [-1 *] [-1 * -1]
0.010671999999999999: <mapping-msg> 1 -> 2 : fec(1), label(0) 1
0.010672 2(2->3): U -1 Push(penultimate) 3 0 32 0
0.010672 3(4->3): U -1 L3 -1 -1 -1 0
0.010672 3: 4 (Mapping 2) 5 0 *_4 [-1 *] [-1 * -1]
0.010671999999999999: <mapping-msg> 4 -> 3 : fec(5), label(0) 4
0.010672 3(3->2): U -1 Push(penultimate) 2 0 32 0
0.020704 3(2->3): U -1 L3 -1 -1 -1 0
0.020704 3: 2 (Mapping 2) 0 1 *_1_2 [-1 *] [-1 * -1]
0.020704: <mapping-msg> 2 -> 3 : fec(0), label(1) 2
0.020704 3(3->4): U -1 Push(penultimate) 4 0 32 0
0.020704 4(3->4): U -1 L3 -1 -1 -1 0
0.020704 4: 3 (Mapping 2) 2 1 *_2_3 [-1 *] [-1 * -1]
0.020704: <mapping-msg> 3 -> 4 : fec(2), label(1) 3
```

ตัวอย่างเทรซไฟล์ (ต่อ)

```
0.020704 1(2->1): U -1 L3 -1 -1 -1 0
0.020704 1: 2 (Mapping 2) 3 2 *_3_2  [-1 *]  [-1 * -1]
0.020704: <mapping-msg> 2 -> 1 : fec(3), label(2) 2
0.020704 2(3->2): U -1 L3 -1 -1 -1 0
0.020704 2: 3 (Mapping 2) 4 2 *_4_3  [-1 *]  [-1 * -1]
0.020704: <mapping-msg> 3 -> 2 : fec(4), label(2) 3
0.020704 2(2->1): U -1 Push(penultimate) 1 0 32 0
0.021072 3(2->3): U -1 L3 -1 -1 -1 0
0.021072 3: 2 (Mapping 3) 1 3 *_1_2  [-1 *]  [-1 * -1]
0.021072: <mapping-msg> 2 -> 3 : fec(1), label(3) 2
0.021072 3(3->4): U -1 Push(penultimate) 4 0 32 0
0.021072 2(3->2): U -1 L3 -1 -1 -1 0
0.021072 2: 3 (Mapping 3) 5 3 *_4_3  [-1 *]  [-1 * -1]
0.021072: <mapping-msg> 3 -> 2 : fec(5), label(3) 3
0.021072 2(2->1): U -1 Push(penultimate) 1 0 32 0
0.031104 4(3->4): U -1 L3 -1 -1 -1 0
0.031104 4: 3 (Mapping 3) 0 4 *_1_2_3  [-1 *]  [-1 * -1]
0.031104: <mapping-msg> 3 -> 4 : fec(0), label(4) 3
0.031104 1(2->1): U -1 L3 -1 -1 -1 0
0.031104 1: 2 (Mapping 3) 4 4 *_4_3_2  [-1 *]  [-1 * -1]
0.031104: <mapping-msg> 2 -> 1 : fec(4), label(4) 2
0.031504 4(3->4): U -1 L3 -1 -1 -1 0
0.031504 4: 3 (Mapping 4) 1 5 *_1_2_3  [-1 *]  [-1 * -1]
0.031504000000000004: <mapping-msg> 3 -> 4 : fec(1), label(5) 3
0.031504 1(2->1): U -1 L3 -1 -1 -1 0
0.031504 1: 2 (Mapping 4) 5 5 *_4_3_2  [-1 *]  [-1 * -1]
0.031504000000000004: <mapping-msg> 2 -> 1 : fec(5), label(5) 2
0.114 1(0->5): U -1 Push(ingress) 2 5 32 4
0.124 1(0->5): U -1 Push(ingress) 2 5 32 4
0.128 2(0->5): L 5 Swap 3 3 31 4
0.134 1(0->5): U -1 Push(ingress) 2 5 32 4
0.138 2(0->5): L 5 Swap 3 3 31 4
0.142 3(0->5): L 3 Pop(penultimate) 4 0 30 0
0.144 1(0->5): U -1 Push(ingress) 2 5 32 4
0.148 2(0->5): L 5 Swap 3 3 31 4
```

ตัวอย่างเทรซไฟล์ (ต่อ)

```
0.152 3(0->5): L 3 Pop(penultimate) 4 0 30 0
0.154 1(0->5): U -1 Push(ingress) 2 5 32 4
0.156 4(0->5): U -1 L3 -1 -1 -1 0
                .....
1.986 4(0->5): U -1 L3 -1 -1 -1 0
1.988 2(0->5): L 5 Swap 3 3 31 4
1.992 3(0->5): L 3 Pop(penultimate) 4 0 30 0
1.994 1(0->5): U -1 Push(ingress) 2 5 32 4
1.996 4(0->5): U -1 L3 -1 -1 -1 0
```

รูปแบบพื้นฐานของเทรซไฟล์ของ MPLS จะแบ่งเป็น 10 ฟิลด์โดยเรียงตามลำดับดังนี้

- เวลาที่เกิดกระบวนการนั้นๆ
- โหนดที่เกิดกระบวนการนั้นๆ
- โหนดต้นทางแรกสุดและปลายทางสุดท้าย
- ลักษณะของแพ็คเกต (U คือแพ็คเกตที่ไม่ลาเบล, L แพ็คเกตที่มีลาเบล)
- ค่าของลาเบลขาเข้า
- กระบวนการที่เกิดขึ้น(Swap, Push, Pop เป็นต้น)
- อินเตอร์เฟซขาออก
- ลาเบลขาออก
- ค่า TTL

ตัวอย่างที่ 1

```
0.114 1(0->5): U -1 Push(ingress) 2 5 32 4
```

หมายถึงเมื่อเวลาดังกล่าว ที่โหนด 1 (LSR1) ซึ่งเป็น ingress LSR (ingress) ได้นำแพ็คเกตที่ไม่มีลาเบล (U) จึงไม่มีลาเบลขาเข้า (-1) โดยถูกส่งจากโหนด 0ไปยังโหนด 5 ซึ่งเป็นโหนดปลายทาง (0->5) ซึ่งทำการใส่ลาเบลที่มีค่า 5 (5) เข้าไป (Push) ซึ่งมีค่า TTL เป็น 32 (32) และขนาดเท่ากับ 4 (4) และจะส่งออกไปทางอินเตอร์เฟซ 2 (2)

```
0.128 2(0->5): L 5 Swap 3 3 31 4
```

หมายถึงเมื่อเวลาดังกล่าว ที่โหนด 2 (LSR2) ได้นำแพ็คเกตที่มีลาเบล (L) ซึ่งมีลาเบลขาเข้าเป็น 5 (5) โดยถูกส่งจากโหนด 0ไปยังโหนด 5 ซึ่งเป็นโหนดปลายทาง (0->5) ซึ่งทำการสับเปลี่ยนลาเบล (Swap) จากลาเบล 5 (5) ไปเป็นลาเบล 3 (3) ซึ่งมีค่า TTL เป็น 31 (31) และขนาดเท่ากับ 4 (4) และจะส่งออกไปทางอินเตอร์เฟซ 3 (3)

```
0.142 3(0->5): L 3 Pop(penultimate) 4 0 30 0
```

หมายถึงเมื่อเวลาดังกล่าว ที่โหนด 3 (LSR3) ซึ่งเป็น LSR ก่อนสุดท้ายในเส้นทาง (Penultimate) ได้นำแพ็คเกตที่มีลาเบล (L) ซึ่งมีลาเบลขาเข้าเป็น 3 (3) โดยถูกส่งจากโหนด 0ไปยังโหนด 5 ซึ่งเป็นโหนดปลายทาง (0->5) ซึ่งจะทำการถอดลาเบลออก (Pop) ซึ่งมีค่า TTL เป็น 30 (30) และขนาดเท่ากับ 0(0) และจะส่งออกไปทางอินเตอร์เฟซ 4 (4)

```
0.156 4(0->5): U -1 L3 -1 -1 -1 0
```

หมายถึงเมื่อเวลาดังกล่าว ที่โหนด 4 (LSR4) ซึ่งในที่นี้เป็น Egress LSR ได้นำแพ็คเกตที่ไม่มีลาเบล (U) โดยถูกส่งจากโหนด 0ไปยังโหนด 5 (0->5) และจะทำการสวิทชิ่งในระดับชั้นที่ 3 (L3)

จากตัวอย่างจะเห็นถึงกระบวนการต่างๆเกี่ยวกับการสวิทชิ่งโดยใช้ลาเบลของ MPLS ซึ่งจะเกิดขึ้นภายในโหนดของ MPLS โดเมน

ใน LDP นั้นแต่ละ LSR จะกระทำการแลกเปลี่ยนลาเบลกันตั้งแต่เริ่มแรกโดยในการจำลองนี้แต่ละ LSR จะมีการส่ง LDP เมสเซจเพื่อการแลกเปลี่ยนลาเบลและเพื่อแสดงการมีอยู่ของตนแก่ LSR ที่อยู่ติดกันโดยรอบ โดยจุดประสงค์เพื่อสร้างตาราง PFT และ LIB เพื่อใช้ในการสวิทชิ่งลาเบลยกตัวอย่างเช่น

นอกจากนี้ MPLS เทรซไฟล์ยังแสดงถึงกระบวนการของการเทียบลาเบล (Mapping) ซึ่งจะจัดกลุ่มเป็นชุดชุดละ 3 บรรทัดโดยเป็นรูปแบบดังนี้

```
0.031104 1(2->1): U -1 L3 -1 -1 -1 0
0.031104 1: 2 (Mapping 3) 4 4 *_4_3_2  [-1 *]  [-1 * -1]
0.031104: <mapping-msg> 2 -> 1 : fec(4), label(4) 2
```

บรรทัดแรกแสดงถึงแพ็คเกตที่โหนด 1 ได้รับมาจากโหนด 2 ซึ่งเป็น LDP เมสเซจ

บรรทัดที่สองจะหมายถึงความหมายของเมสเซจในบรรทัดแรกโดยหมายถึง ที่โหนด 1 ได้รับเมสเซจจากโหนด 2 มาทำการเทียบลาเบลเป็นครั้งที่ 3 ซึ่งต้องการเทียบไปยังปลายทางคือโหนด 4 และใช้ลาเบล 4 ส่งมา โดยเส้นทาง จะเริ่มจากโหนด 4 ผ่านโหนด 3 และโหนด 2 สรุปคือบรรทัดนี้โหนด 2 ต้องการบอกโหนด 1 ว่าสามารถไปยังโหนด 4 ผ่านทางโหนด 2 ได้ โดยใช้ลาเบล 4 ส่งออกทางอินเตอร์เฟซ 2 โดยเมื่อผ่านโหนด 2 ไปจะผ่านโหนด 3 และถึงโหนด 4

และบรรทัดสุดท้ายจะเป็นกระบวนการที่เกิดจากการตีความหมายที่ได้จากบรรทัดที่สองซึ่งจะทำการบันทึกค่าลงในตาราง PFT และ LIB โดยจะเก็บว่าหากจะใช้เส้นทางจาก FEC 4 จะต้องส่งออกไปทางอินเตอร์เฟซ 2 ด้วยลาเบล 4

ในการสร้าง LSP นั้นแต่ละ LSR จำเป็นต้องใช้ค่าในตาราง PFT และ LIB เพื่อที่จะนำมาเทียบเพื่อหาอินเตอร์เฟซและลาเบลที่ต้องการส่งออกไป ซึ่งแต่ละ LSR ใน MPLS โดเมนจำเป็นต้องมีการติดต่อกันเพื่อแลกเลี่ยนลาเบลและเรียนรู้เส้นทางภายในโดเมนโดยใช้ LDP เมสเซจ

ใน LDP นั้นแต่ละ LSR จะกระทำการแลกเปลี่ยนลาเบลกันตั้งแต่เริ่มแรก โดยในการจำลองนี้แต่ละ LSR จะมีการส่ง LDP เมสเซจเพื่อการแลกเปลี่ยนลาเบลและเพื่อแสดงการมีอยู่ของตนแก่ LSR ที่อยู่ติดกันโดยรอบ โดยจุดประสงค์เพื่อสร้างตาราง PFT และ LIB เพื่อใช้ในการสวิทชิ่งลาเบล

โดยในการอธิบายจะยกตัวอย่างการสร้างตารางของ LSR 2 โดยเราสามารถบันทึกตารางให้แสดงใน MPLS เทรซไฟล์ได้ด้วยคำสั่ง dump ดังที่ได้กล่าวถึงไปแล้ว

LDP เมสเซจเพื่อการเทียบ (Mapping LDP Message) จะถูกส่งตั้งแต่ตั้งแต่เริ่มการจำลองที่วินาทีที่ 0 ซึ่งแต่ละ LSR จะส่งออกไปยังทุกอินเตอร์เฟซของตนที่อยู่ใน MPLS โดเมน

```
0 1(1->2): U -1 L3 -1 -1 -1 0
0 1(1->2): U -1 L3 -1 -1 -1 0
0 2(2->1): U -1 L3 -1 -1 -1 0
0 2(2->3): U -1 L3 -1 -1 -1 0
0 3(3->2): U -1 L3 -1 -1 -1 0
0 3(3->4): U -1 L3 -1 -1 -1 0
0 4(4->3): U -1 L3 -1 -1 -1 0
0 4(4->3): U -1 L3 -1 -1 -1 0
```

โดยจะสังเกตได้ว่า Edge LSR ซึ่งคือ LSR1 และ LSR4 จะมีการส่งเมสเซจไปจำนวน 2 เมสเซจในแต่ละอินเตอร์เฟซ เนื่องจาก เมสเซจการเทียบนี้จะเป็นการบอกให้ LSR รู้ถึงการมีอยู่ของตนและเป็นการบอกโดยนัยว่าสามารถติดต่อมายังตนได้ หรือติดต่อไปยังปลายทางใดๆ โดยผ่านตนได้

ดังนั้น Edge LSR ซึ่งติดต่อกับโหนดที่อยู่นอก MPLS โดเมนด้วย จึงต้องทำการส่ง 2 เมสเซจโดย เมสเซจหนึ่งเพื่อบอกการมีอยู่ของตน และบอกถึงการมีอยู่ของโหนดนอก MPLS โดเมน โดยโหนด 2 แต่ LSR2 จะยังไม่ทราบถึงการมีอยู่ของ LSR ตัวอื่น จึงจะส่งข้อมูลเฉพาะของตนเท่านั้นออกไป ทาง 2 อินเตอร์เฟซของตน คือไปยัง LSR1 และ LSR3 ซึ่งในขณะนี้ ตาราง PFT และ LIB ของแต่ละ LSR จะยังไม่มีข้อมูลใดๆ

```
--) ___PFT dump___ [node: 2] (--
-----------------------------------------
  FEC     PHB     LIBptr  AltanativePath

  ___LIB dump___ [node: 2]
-----------------------------------------
   #    iIface   iLabel   oIface   oLabel   LIBptr
```

จากนั้นเมื่อเมสเซจที่ถูกส่งจาก LSR1 และ LSR3 ถูกส่งมาถึง LSR2 ซึ่งจะสังเกตได้จากผลการ เทรซดังนี้

```
0.010336 2(1->2): U -1 L3 -1 -1 -1 0
0.010336 2: 1 (Mapping 1) 0 0 *_1  [-1 *]  [-1 * -1]
0.010336: <mapping-msg> 1 -> 2 : fec(0), label(0) 1
0.010336 2(3->2): U -1 L3 -1 -1 -1 0
0.010336 2: 3 (Mapping 1) 3 0 *_3  [-1 *]  [-1 * -1]
0.010336: <mapping-msg> 3 -> 2 : fec(3), label(0) 3
```

ดังที่ได้กล่าวไปแล้วว่าจะจัดกลุ่มผลจากเมสเซจการเทียบเป็น 3 บรรทัด ซึ่ง 3 บรรทัดแรกคือ กระบวนการที่ LSR2 ได้รับเมสเซจจาก LSR1 และทำการค่าที่ได้จากการตีความเมสเซจดังกล่าวลง ในตาราง โดย LSR2 จะทราบว่าสามารถส่งข้อมูลไปยัง LSR1 ได้โดยใช้ลาเบล 0 ส่งไปทางอินเตอร์ เฟซ 1 โดยจะนำค่า FEC ไปใส่ในตาราง PFT ดังนี้

```
--) ___PFT dump___ [node: 2] (--
-----------------------------------------
```

```
FEC     PHB     LIBptr  AltanativePath
0       -1      0       -1
```

ซึ่ง FEC จะเป็นการจัดกลุ่มของปลายทางด้วยแนวคิดที่ว่า FEC เดียวกันจะถูกส่งไปในเส้นทางเดียวกันดังที่ได้กล่าวไปในหัวข้อ MPLS โดยใน NS จะจัดกลุ่มตามปลายทาง ซึ่ง FEC 0 จะหมายถึงปลายทางเป็นโหนด 0 ซึ่งค่า LIBptr จะเป็นตัวชี้ไปยังค่าในตาราง LIB ซึ่งเป็นค่าดัชนี โดยจะอ้างอิงไปยังดัชนีที่ 0 ซึ่งข้อมูลในตาราง LIB เมื่อถูกสร้างขึ้นจะมีดัชนีอยู่ซึ่งจะเริ่มจากค่า 0 เป็นค่าแรก

```
___LIB dump___ [node: 2]
----------------------------------------
#     iIface    iLabel    oIface    oLabel    LIBptr
0:    -1        1         1         0         -1
```

ซึ่งข้อมูลที่ 0 จะระบุถึงการหากมีข้อมูลเข้ามาด้วยลาเบล 1 จะทำการถอดลาเบลและส่งออกไปที่อินเตอร์เฟซ 1 ซึ่งลาเบล 1 นั้น LSR2 จะเป็นคนกำหนดขึ้นมาเองตามลำดับเมสเซจการเทียบที่ได้รับ ซึ่งเริ่มที่ค่า 1 และจากข้อมูลนี้หากเทียบย้อนกลับไปที่ PFT ก็คือจะไปยังโหนด 0 นั่นเอง

จากนั้น 3 บรรทัดถัดมาคือกระบวนการที่เกิดขึ้นเมื่อ LSR2 ได้รับเมสเซจจาก LSR3 ซึ่งทำให้ค่าในตารางเพิ่มเป็นดังนี้

```
--) ___PFT dump___ [node: 2] (--
----------------------------------------
FEC     PHB     LIBptr  AltanativePath
0       -1      0       -1
3       -1      1       -1
___LIB dump___ [node: 2]
----------------------------------------
#     iIface    iLabel    oIface    oLabel    LIBptr
0:    -1        1         1         0         -1
1:    -1        2         3         0         -1
```

ซึ่งจากตารางจะพบว่า LSR2 สามารถส่งข้อมูลไปยังโหนด 0 และ LSR3 ได้โดยตรง เพราะไม่ต้องกำหนดลาเบลออกไป

โดยในเวลานี้ LSR อื่นๆก็จะได้รับเมสเซจการเทียบจากอินเตอร์เฟซต่างๆของแต่ละตัวก็จะมีกระบวนการเรียนรู้การมีอยู่และวิธีในการที่จะส่งข้อมูลไปถึงโหนดนั้นๆซึ่งยังเป็นโหนดที่อยู่ติดกันอยู่

จากการที่เมื่อเริ่ม LSR1 ได้ส่งเมสเซจมา 2 เมสเซจซึ่งอีกเมสเซจที่ได้รับจาก LSR1 ก็คือ

```
0.010672 2(1->2): U -1 L3 -1 -1 -1 0
0.010672 2: 1 (Mapping 2) 1 0 *_1  [-1 *]  [-1 * -1]
0.010671999999999999: <mapping-msg> 1 -> 2 : fec(1), label(0) 1
```

ซึ่งจะเป็นการให้ LSR2 รับรู้ถึงการมีอยู่ของ LSR1 ซึ่งก็จะเกิดกระบวนการเรียนรู้เช่นเดียวกัน โดยจะบันทึกผลการเรียนรู้ดังกล่าวลงในตารางทั้ง 2 เช่นเดิม

```
--) ___PFT dump___ [node: 2] (--
-----------------------------------
  FEC    PHB    LIBptr  AltanativePath
  0      -1     0       -1
  3      -1     1       -1
  1      -1     2       -1

___LIB dump___ [node: 2]
-----------------------------------
   #    iIface   iLabel   oIface   oLabel   LIBptr
  0:    -1       1        1        0        -1
  1:    -1       2        3        0        -1
  2:    -1       3        1        0        -1
```

โดยข้อมูลใหม่ในตารางจะบอกถึงการที่จะส่งข้อมูลไปยัง LSR1 ซึ่งจะพบว่าต่อตรงกับ LSR2 เช่นเดียวกัน

ถึงขณะนี้ทุกโหนดจะเป็นผลจากการเรียนรู้จากเมสเซจที่มีการส่งตั้งแต่เริ่มแรกเท่านั้น โดยแต่ละโหนดจะรู้จัก LSR หรือ MPLS โหนดที่อยู่ติดกัน และโหนดปกติซึ่งอยู่ห่างออกไป 1 ระดับเท่านั้น ซึ่งในขณะนี้ LSR 2 จะสามารถส่งข้อมูลหาใครได้บ้าง สามารถดูได้จากตาราง PFT

ซึ่งหลังจากนั้นแต่ละโหนดจะทำการนำข้อมูลที่ได้เรียนรู้มาใหม่ส่งต่อไป โดยจะส่งออกไปยังอินเตอร์เฟซที่เหลือซึ่งไม่ใช่อินเตอร์เฟซเดิม โดย LSR2 จะนำข้อมูลเกี่ยวกับ LSR1 และโหนด 0 ส่งไปยัง LSR 3 และ นำข้อมูลของ LSR3 ส่งไปยัง LSR1 เพื่อขยายขอบเขตการเรียนรู้ และ LSR2 ก็จะได้รับเมสเซจที่เกิดจากการเรียนรู้ของ LSR3 เช่นเดียวกัน แต่จะไม่ได้รับจาก LSR1 เพราะ LSR1 ยังไม่ได้เรียนรู้อะไรเพิ่มเติม ซึ่งจากเทรซไฟล์ได้ดังนี้

```
0.020704 2(3->2): U -1 L3 -1 -1 -1 0
0.020704 2: 3 (Mapping 2) 4 2 * _4_3  [-1 *]  [-1 * -1]
0.020704: <mapping-msg> 3 -> 2 : fec(4), label(2) 3
0.021072 2(3->2): U -1 L3 -1 -1 -1 0
0.021072 2: 3 (Mapping 3) 5 3 * _4_3  [-1 *]  [-1 * -1]
0.021072: <mapping-msg> 3 -> 2 : fec(5), label(3) 3
```

โดย LSR2 จะเรียนรู้ได้ว่าสามารถส่งข้อมูลไปยัง LSR4 ผ่าน LSR3 ได้ โดยส่งออกไปด้วยลาเบล 2 ซึ่งลาเบล 2 นี้ได้จากการกำหนดของ LSR 3 โดยจะกำหนดเรียงตามลำดับการเรียนรู้และเริ่มจากค่า 1 จากนั้น LSR2 จะทำการเพิ่มค่าในตารางเกี่ยวกับการไปถึง LSR4 และจากนั้นก้ได้รับเมสเซจเกี่ยวกับการไปถึงโหนด 5 ซึ่งก็ส่งผ่าน LSR3 เช่นเดิมด้วยลาเบล 3 ซึ่งค่าในตารางจะได้ผลดังนี้

```
  --) ___PFT dump___ [node: 2] (--
-----------------------------------------

  FEC     PHB     LIBptr  AltanativePath
  0       -1      0       -1
  3       -1      1       -1
  1       -1      2       -1
  4       -1      3       -1
  5       -1      4       -1


  ___LIB dump___ [node: 2]
-----------------------------------------
```

#	iIface	iLabel	oIface	oLabel	LIBptr
0:	-1	1	1	0	-1
1:	-1	2	3	0	-1
2:	-1	3	1	0	-1
3:	-1	4	3	2	-1
4:	-1	5	3	3	-1

ซึ่งจากตรงจุดนี้เมื่อดูจากโทโพโลจีเราสามารถทราบได้ว่า LSR2 เรียนรู้เส้นทางไปยังทุกๆ โหนดแล้ว ถึงแม้ความจริงแล้ว LSR2 ไม่สามารถทราบได้ด้วยตนเอง แต่เนื่องจากมันไม่ได้รับเมสเซจเพื่อให้เกิดการเรียนรู้อีกแล้ว แต่นั่นก็ไม่ได้หมายถึงกระบวนการแลกเปลี่ยนลาเบลจะเสร็จสมบูรณ์ เนื่องจาก LSR2 ต้องส่งข้อมูลทุกอย่างที่เรียนรู้ใหม่ออกไปยังอินเตอร์เฟซที่เหลือ โดยส่งเมสเซจที่บอกถึงการมีอยู่ของ LSR4 และโหนด 5 ไปยัง LSR1 โดยหลังจากนั้นกระบวนการเรียนรู้จะสมบูรณ์ทั่วทั้งโทโพโลจี ซึ่งแต่ละ LSR จะมีค่าในตารางต่างๆดังนี้

--) ___PFT dump___ [node: 1] (--

FEC	PHB	LIBptr	AltanativePath
2	-1	0	-1
3	-1	1	-1
4	-1	2	-1
5	-1	3	-1

--) ___ERB dump___ [node: 1] (--

FEC	LSPid	LIBptr

___LIB dump___ [node: 1]

#	iIface	iLabel	oIface	oLabel	LIBptr
0:	-1	-1	2	0	-1
1:	-1	-1	2	2	-1
2:	-1	-1	2	4	-1

```
3:   -1    -1   2    5    -1

--) ___PFT dump___ [node: 2] (--
-----------------------------------------
FEC    PHB    LIBptr  AltanativePath
0      -1     0       -1
3      -1     1       -1
1      -1     2       -1
4      -1     3       -1
5      -1     4       -1

--) ___ERB dump___ [node: 2] (--
-----------------------------------------
FEC    LSPid    LIBptr

___LIB dump___ [node: 2]
-----------------------------------------
 #    iIface   iLabel   oIface   oLabel   LIBptr
0:    -1       1        1        0        -1
1:    -1       2        3        0        -1
2:    -1       3        1        0        -1
3:    -1       4        3        2        -1
4:    -1       5        3        3        -1

--) ___PFT dump___ [node: 3] (--
-----------------------------------------
FEC    PHB    LIBptr  AltanativePath
2      -1     0       -1
4      -1     1       -1
5      -1     2       -1
0      -1     3       -1
1      -1     4       -1
```

--) ___ERB dump___ [node: 3] (--

FEC LSPid LIBptr

___LIB dump___ [node: 3]

#	iIface	iLabel	oIface	oLabel	LIBptr
0:	-1	1	2	0	-1
1:	-1	2	4	0	-1
2:	-1	3	4	0	-1
3:	-1	4	2	1	-1
4:	-1	5	2	3	-1

--) ___PFT dump___ [node: 4] (--

FEC	PHB	LIBptr	AltanativePath
3	-1	0	-1
2	-1	1	-1
0	-1	2	-1
1	-1	3	-1

--) ___ERB dump___ [node: 4] (--

FEC LSPid LIBptr

___LIB dump___ [node: 4]

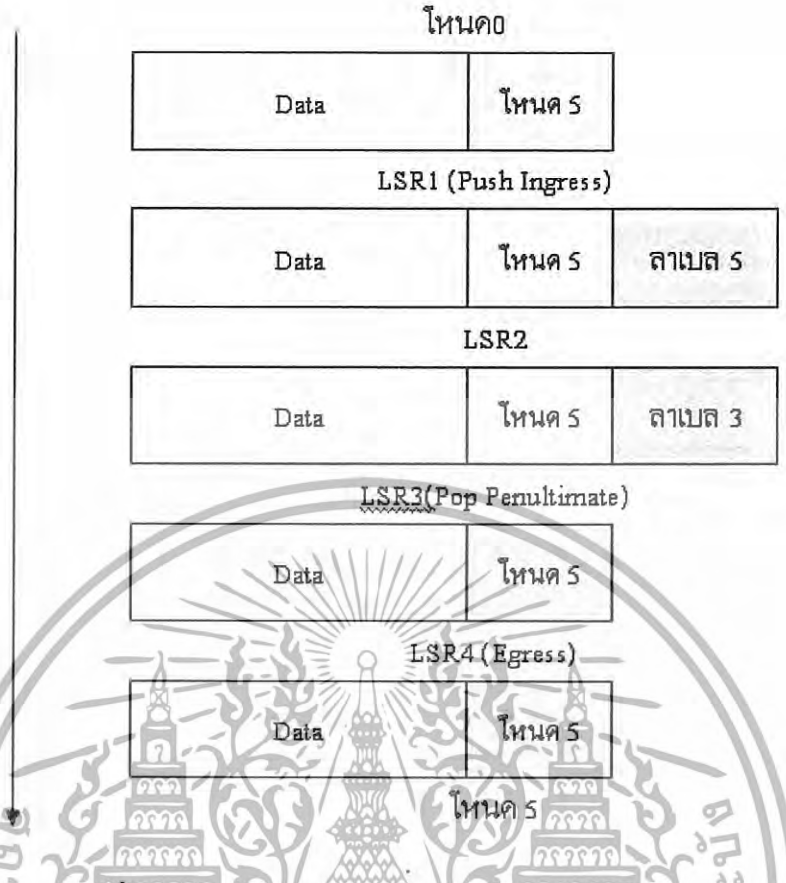
#	iIface	iLabel	oIface	oLabel	LIBptr
0:	-1	-1	3	0	-1
1:	-1	-1	3	1	-1
2:	-1	-1	3	4	-1

```
3:   -1    -1   3    5    -1
```

โดยเมื่อเสร็จสิ้นกระบวนการเรียนรู้ แต่ละโหนดก็จะสามารถส่งข้อมูลถึงกันได้หมด โดยในการจำลองได้กำหนดให้โหนด 0 ส่งข้อมูลไปยังโหนด 5 ซึ่งจากเทรซไฟล์จะได้ผลดังนี้

```
0.114 1(0->5): U -1 Push(ingress) 2 5 32 4
0.128 2(0->5): L 5 Swap 3 3 31 4
0.142 3(0->5): L 3 Pop(penultimate) 4 0 30 0
0.156 4(0->5): U -1 L3 -1 -1 -1 0
```

โดยเริ่มแรกโหนด 0 จะส่งแพ็คเกตไปยัง LSR1 ซึ่งเป็น ingress LSR ซึ่งอยู่นอก MPLS โดเมน ซึ่งไม่สามารถเทรซออกมาได้ จากนั้นเมื่อ LSR1 ได้รับแพ็คเกตดังกล่าว จะอ่านที่อยู่ปลายทางของแพ็คเกตนั้นและพบว่าอยู่ใน FEC 5 ซึ่งเมื่อเทียบไปจนถึงข้อมูลในตาราง LIB จะพบว่าต้องใส่ลาเบลเข้าไปด้วยค่า 5 ไปทางอินเตอร์เฟซ 2 จากนั้นเมื่อ LSR2 ได้รับจะอ่านค่าลาเบลและเทียบกับตาราง LIB และพบว่าต้องเปลี่ยนลาเบลออกไปเป็นค่า 3 และส่งออกไปยังอินเตอร์เฟซ 3 เมื่อ LSR3 ได้รับ และทำการเทียบตารางจะพบว่า ไม่มีลาเบลให้เปลี่ยนต่อจึงทำการถอดลาเบลออกซึ่งเป็นไปตามเทคนิค Penultimate Hop Popping โดยจะทำการถอดลาเบลตั้งแต่ LSR ก่อน Egress LSR เมื่อถอดลาเบลเสร็จจะส่งไปยัง LSR4 ซึ่ง LSR4 เมื่อพบว่าไม่มีเป็นแพ็คเกตที่ไม่มีลาเบลก็จะทำการส่งข้อมูลด้วยการสวิทในระดับชั้นที่ 3 ต่อไปจนถึงโหนด 5 ซึ่งเป็นโหนดปลายทาง โดยการส่งข้อมลจากโหนด 0 ไปโหนด 5 และการเปลี่ยนลาเบลจะเป็นลักษณะดังนี้



รูปที่3.24 แสดงการส่งข้อมูลโดยใช้ลาเบลจากการจำลอง

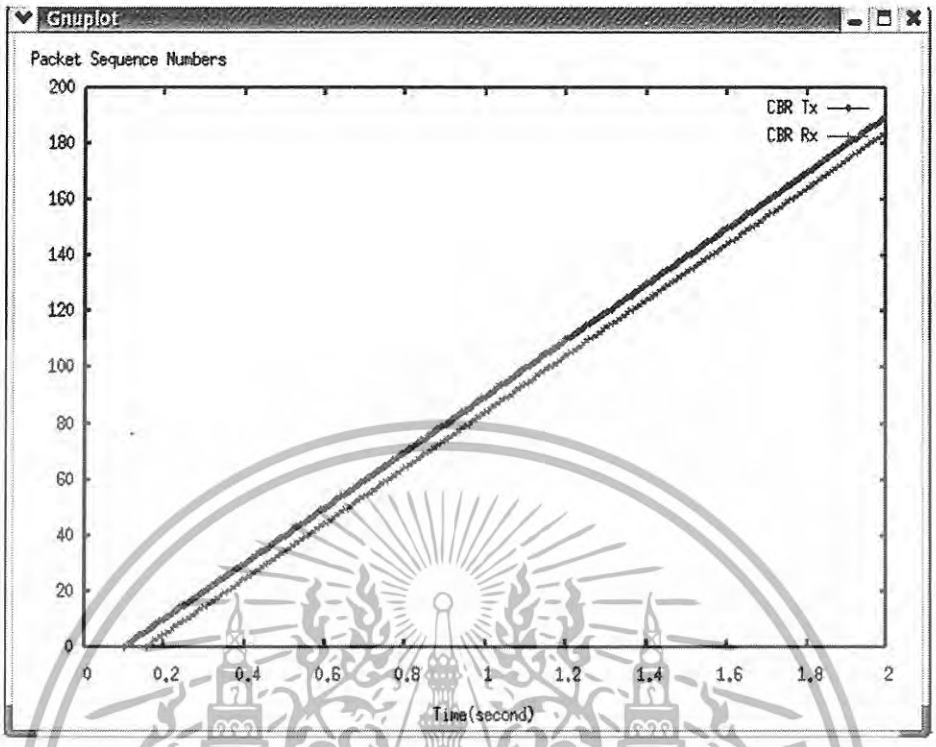
นอกจากนี้ในสคริปดังกล่าวยังได้สร้างเทรซไฟล์ในแบบมาตรฐานของ NS ซึ่งจะเก็บไว้ในในไฟล์ out.tr ซึ่งจะมีรูปแบบดังที่ได้กล่าวไปแล้วในตอนต้น ซึ่งจะไม่เห็นถึงการทำงานของ MPLS แต่จะแสดงให้เห็นถึงการทำงานปกติอาทิเช่น เข้าคิว ออกจากคิว รับข้อมูลเป็นต้น ซึ่งยังสามารถใช้ตัวอย่าง Perl สคริปในบทดังกล่าว ในการที่จะวิเคราะห์ผลและสแดงผลออกมาในรูปแบบของกราฟได้อีกด้วย

อาทิเช่นใช้ Perl สคริปในการแสดงลำดับของแพ็คเกตในการติดตามผลการสูญเสียข้อมูลระหว่างเส้นทาง ดังนี้

ตัวอย่าง สคริปการแสดงลำดับแพ็คเกตในการติดตามผลการสูญเสียข้อมูลระหว่างเส้นทาง

```
#!/usr/bin/perl
#
# simple perl script to print CBR sequences
#
open(cmd,">sqs");
open(cmd1,">sqr");
open(infile, "out.tr") or die "couldn't open the file";
$line_no=0;
while ($line = <infile>) {
    ++$line_no;
    @entry = split(" ", $line);
   if(($entry[0] eq "+")&&($entry[2] eq "0")&&($entry[3] eq
"1")&&($entry[4] eq "cbr")) {
     print cmd "$entry[1] $entry[10] \n";
    }
    if(($entry[0] eq "+")&&($entry[2] eq "4")&&($entry[3] eq
"5")&&($entry[4] eq "cbr")) {

     print cmd1 "$entry[1] $entry[10] \n";
    }
}
close(cmd);
close(cmd1);
close(infile);
# call plot_graph
plot_graph();
sub plot_graph
{
    $output_type=0;
    $outname="data1.eps";
    $x_label="Time(second)";
    $y_label="Packet Sequence Numbers";
    open(cmd, ">cmd.tmp");
    if($output_type==0){
     print cmd "set term x11 \n";
    }else{

     print cmd "set term post portrait color solid \'Roman\'
11\n";
     print cmd "set output \"$outname\"\n";
     print cmd "set size 1.0,0.5\n"
    }
    print cmd "set xlabel \"$x_label\" \n";
    print cmd "set ylabel \"$y_label\"\n";
    $output_files="sqs";
    $output_files1="sqr";
    print cmd "plot \"$output_files\" u 1:2 t \"CBR Tx\" w
linespoints,\"$output_files1\" u 1:2 t \"CBR Rx\" w
linespoints   ";
     print cmd "\n";
    close(cmd);
    system "gnuplot -persist cmd.tmp";}
```

โดยผลจากกราฟจะแสดงออกมาดังนี้



รูปที่ 3.25 กราฟแสดงลำดับของแพ็คเกต

จากกราฟเส้นบนคือลำดับของแพ็คเกตที่ถูกส่งออกมาจากโหนด 0 และเส้นล่างคือลำดับของแพ็คเกตที่โหนด 5 ได้รับ ซึ่งจะแสดงบนแกนของเวลา ซึ่งระยะห่างระหว่าง 2 เส้น คือระยะเวลาที่แพ็คเกตถูกส่งจากต้นทางถึงเวลาที่ปลายทางได้รับ จากกราฟจะเห็นได้ว่าเวลาหน่วงในการส่งนั้นคงที่ และแพ็คเกตไม่มีการสูญเสียเนื่องจากแพ็คเกตที่ต้นทางส่ง และที่ปลายทางได้รับนั้นมีค่าเท่ากัน

หรืออีกตัวอย่างเช่นการใช้สคริปในการหาเวลาหน่วงจากต้นทางถึงปลายทาง ดังนี้

ตัวอย่าง แสดงสคริปในการหาเวลาหน่วงจากต้นทางถึงปลายทาง

```
#!/usr/bin/perl
#
# simple perl script to print CBR sequences
#
open(cmd,">sqs");
open(cmd1,">sqr");
open(infile, "out.tr") or die "couldn't open the file";
$line_no=0;
while ($line = <infile>) {
++$line_no;
    @entry = split(" ", $line);
```

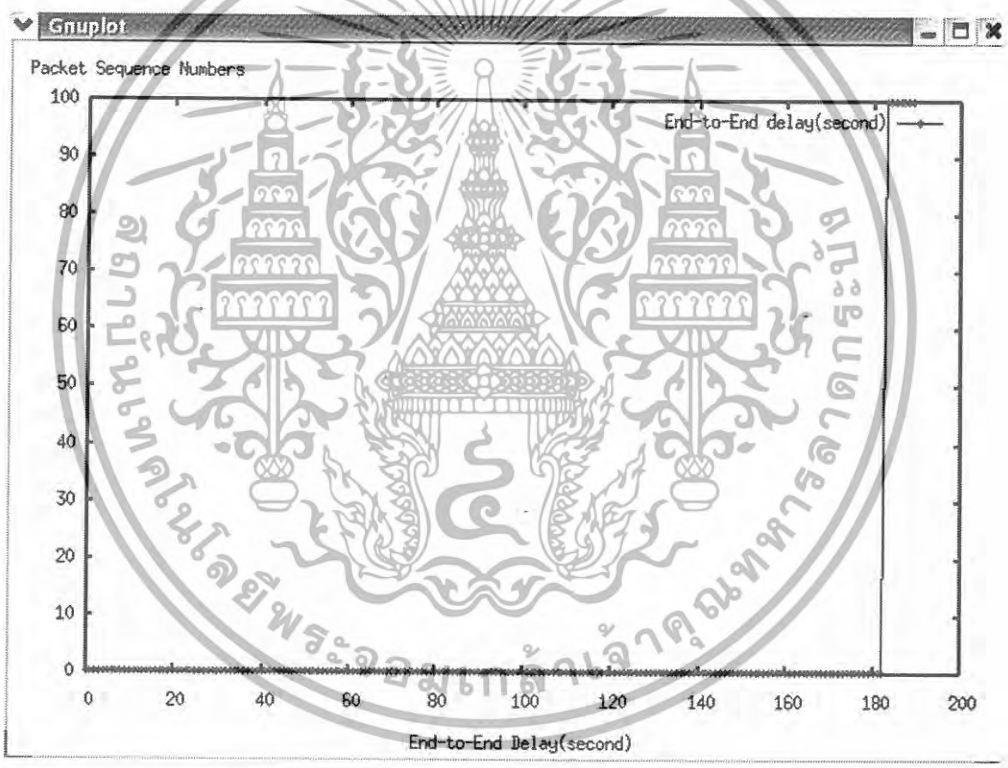
ตัวอย่าง แสดงสคริปในการหาเวลาหน่วงจากต้นทางถึงปลายทาง (ต่อ)

```
if(($entry[0] eq "+")&&($entry[2] eq "0")&&($entry[3] eq
"1")&&($entry[4] eq "cbr")) {
print cmd "$entry[1] $entry[10] \n";
}
if(($entry[0] eq "r")&&($entry[3] eq "5")&&($entry[4] eq
"cbr")) {

print cmd1 "$entry[1]  $entry[10]\n";
}
}
close(cmd);
close(cmd1);
close(infile);

open(infiles,"sqs");
open(cmd,">delay");
while ($lines = <infiles>) {
@entrys = split(" ", $lines);
open(infiler,"sqr");
$st=0;
while ($liner = <infiler>) {
@entryr = split(" ", $liner);
if($entrys[1] eq $entryr[1]){
$delay=$entryr[0]-$entrys[0];
print cmd "$entrys[1] $delay\n";
$st=1;
last;
}
}
if($st==0){
print cmd "$entrys[1]  100\n"
}
close(infiler)
}
close(cmd);
close(infiles);
close(infiler);

# call plot_graph

plot_graph();

sub plot_graph
{
$output_type=0;
$outname="data1.eps";
$y_label="Packet Sequence Numbers";
$x_label="End-to-End Delay(second)";
open(cmd, ">cmd.tmp");
if($output_type==0){
print cmd "set term x11 \n";
}else{
```

ตัวอย่าง แสดงสคริปในการหาเวลาหน่วงจากต้นทางถึงปลายทาง(ต่อ)

```
print cmd "set term post portrait color solid \'Roman\' 11\n";
      print cmd "set output \"$outname\"\n";
      print cmd "set size 1.0,0.5\n"
    }
    print cmd "set xlabel \"$x_label\" \n";
    print cmd "set ylabel \"$y_label\"\n";
    $output_files="delay";
    print cmd "plot \"$output_files\" u 1:2 t \"End-to-End
delay(second)\" w linespoints";
    print cmd "\n";
    close(cmd);
    system "gnuplot -persist cmd.tmp";
}
```

โดยผลจากกราฟจะแสดงออกมาดังนี้



รูปที่ 3.26 กราฟแสดงเวลาหน่วงจากต้นทางถึงปลายทาง

จากกราฟจะแสดงให้เห็นถึงเวลาหน่วงที่คงที่ตลอดเส้นทาง แต่สังเกตุได้ว่าตั้งแต่แพ็คเกตลำดับที่ 180 กว่าเป็นต้นไปมีดีเลย์สูงมาก ซึ่งในที่นี้คือถึงค่าอนันต์ เพราะปลายทางไม่ได้รับแพ็คเกตดังกล่าว เนื่องจากใน TCL สคริปเราได้กำหนดให้ให้ต้นทางหยุดส่งข้อมูลวินาทีที่ 2.0 แต่ก็ได้กำหนดให้หยุดการจำลองที่วินาทีที่ 2.0 เช่นเดียวกัน ทำให้เกิดแพ็คเกตที่ถูกส่งไปจากต้นทางก่อนวินาทีที่ 2.0 แต่ปลายทางยังไม่ได้รับ ทำให้แพ็คเกตมีค่าเวลาหน่วงหน่วงอนันต์

3. 8 การทดสอบประสิทธิภาพระหว่างเครือข่ายไอพีและเครือข่าย MPLS

ในส่วนนี้จะกล่าวถึงการจำลองเครือข่าย MPLS และเครือข่ายปกติ เพื่อทำการเปรียบเทียบประสิทธิภาพในด้านต่างๆ โดยมุ่งเน้นไปที่สมมติฐานที่ว่าด้วยเรื่องของการที่ LSR ใช้มีเวลาในการสวิทชิ่งน้อยกว่าเราเตอร์ปกติ เนื่องจากใช้เพียงลาเบลในการตัดสินใจเลือกเส้นทาง ไม่ต้องใช้ข้อมูลในเฮดเดอร์ของเลเยอร์ 3 ที่อยู่ถัดเข้าไป

ในการทดสอบประสิทธิภาพระหว่างเครือข่ายทั้งสองแบบนั้น ในเครือข่ายไอพีปกติจะกำหนดให้มีการสร้างโทโพโลยีที่มีโหนดจำนวน 10 โหนด โดยทำการกำหนดให้ทุกโหนดเป็นโหนดปกติทั้งหมดดังนี้

ตัวอย่าง แสดงสคริปการจำลองเครือข่ายไอพี

```
#Create a simulator object
set ns [new Simulator]

$ns color 0 orange
$ns color 1 magenta

Classifier/Addr/MPLS set control_driven_ 1

#Turn on all traces to stdout
Agent/LDP set trace_ldp_ 1
Classifier/Addr/MPLS set trace_mpls_ 1

#Open the nam trace file
set nf [open out.namip10 w]
$ns namtrace-all $nf

#Open the Trace file
set tf [open outip10.tr w]
$ns trace-all $tf

#set tmpls [open mplstrace.tr w]
#$ns trace-mpls $tmpls

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
      #Close the trace file
        close $nf
      #Close the Trace file

        close $tf
        #Execute nam on the trace file
        exec nam out.namip10 &
        exit 0
}
```

ตัวอย่าง แสดงสคริปการจำลองเครือข่ายไอพี(ต่อ)

```
set Node0 [$ns node]
set Node1 [$ns node]
set Node2  [$ns node]
set Node3  [$ns node]
set Node4   [$ns node]
set Node5   [$ns node]
set Node6   [$ns node]
set Node7   [$ns node]
set Node8   [$ns node]
set Node9  [$ns node]
set Node10 [$ns node]

$ns duplex-link $Node0 $Node2  10Mb  10ms DropTail
$ns duplex-link $Node1 $Node2  10Mb  10ms DropTail
$ns duplex-link $Node2  $Node3  10Mb  10ms DropTail
$ns duplex-link $Node3  $Node4  10Mb  10ms DropTail
$ns duplex-link $Node4  $Node8  10Mb  10ms DropTail
$ns duplex-link $Node2  $Node5  10Mb  10ms DropTail
$ns duplex-link $Node5  $Node6  10Mb  10ms DropTail
$ns duplex-link $Node5  $Node4  10Mb  10ms DropTail
$ns duplex-link $Node6  $Node7  10Mb  10ms DropTail
$ns duplex-link $Node6  $Node8  10Mb  10ms DropTail
$ns duplex-link $Node7  $Node8  10Mb  10ms DropTail
$ns duplex-link $Node7  $Node9  10Mb  10ms DropTail
$ns duplex-link $Node8  $Node10 10Mb  10ms DropTail


set Src0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$Src0 attach-agent $udp0
$ns attach-agent $Node0 $udp0
$Src0 set packetSize 500
$Src0 set interval 0.010

set Src1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
$Src1 attach-agent $udp1
$ns attach-agent $Node1 $udp1
$Src1 set packetSize 500
$Src1 set interval 0.010

set Dst0 [new Agent/Null]
$ns attach-agent $Node9 $Dst0

set Dst1 [new Agent/Null]
$ns attach-agent $Node10 $Dst1

$udp0 set fid_ 0
$udp1 set fid_ 1

$ns connect $udp0 $Dst0
$ns connect $udp1 $Dst1

$ns at 0.1 "$Src0 start"
```

ตัวอย่าง แสดงสคริปการจำลองเครือข่ายไอพี (ต่อ)

```
$ns at 0.1  "$Src1 start"

$ns at 1.8 "$Src0 stop"
$ns at 1.8 "$Src1 stop"

$ns at 2.0 "finish"
$ns run
```

ขณะที่เครือข่าย MPLS จะมีโทโพโลยีเช่นเดิมแต่จะกำหนดให้ โหนดที่ 0 และ 1 , 9 และ 10 เป็นโหนดปกติ และโหนดที่ 2 ถึง 8 เป็นโหนด MPLS หรือ LSR ดังนี้

ตัวอย่าง แสดงสคริปการจำลองเครือข่าย MPLS

```
#Create a simulator object
set ns [new Simulator]

$ns color 0 orange
$ns color 1 magenta

Classifier/Addr/MPLS set control_driven_ 1

#Turn on all traces to stdout
Agent/LDP set trace_ldp_ 1
Classifier/Addr/MPLS set trace_mpls_ 1

#Open the nam trace file
set nf [open out.nammp10 w]
$ns namtrace-all $nf

#Open the Trace file

set tf [open outmp10.tr w]
$ns trace-all $tf

#set tmpls [open mplstrace.tr w]
#$ns trace-mpls $tmpls

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
      #Close the trace file
        close $nf
      #Close the Trace file

        close $tf
        #Execute nam on the trace file
        exec nam out.nammp10 &
        exit 0
}
```

ตัวอย่าง แสดงสคริปการจำลองเครือข่าย MPLS (ต่อ)

```
set Node0 [$ns node]
set Node1 [$ns node]
$ns node-config -MPLS ON
set LSR2   [$ns node]
set LSR3   [$ns node]
set LSR4   [$ns node]
set LSR5   [$ns node]
set LSR6   [$ns node]
set LSR7   [$ns node]
set LSR8   [$ns node]
$ns node-config -MPLS OFF
set Node9  [$ns node]
set Node10 [$ns node]


$ns duplex-link $Node0 $LSR2  10Mb  10ms DropTail
$ns duplex-link $Node1 $LSR2  10Mb  10ms DropTail
$ns duplex-link $LSR2  $LSR3  10Mb  10ms DropTail
$ns duplex-link $LSR3  $LSR4  10Mb  10ms DropTail
$ns duplex-link $LSR4  $LSR8  10Mb  10ms DropTail
$ns duplex-link $LSR2  $LSR5  10Mb  10ms DropTail
$ns duplex-link $LSR5  $LSR6  10Mb  10ms DropTail
$ns duplex-link $LSR5  $LSR4  10Mb  10ms DropTail
$ns duplex-link $LSR6  $LSR7  10Mb  10ms DropTail
$ns duplex-link $LSR7  $LSR8  10Mb  10ms DropTail
$ns duplex-link $LSR6  $LSR8  10Mb  10ms DropTail
$ns duplex-link $LSR7  $Node9  10Mb  10ms DropTail
$ns duplex-link $LSR8  $Node10 10Mb  10ms DropTail

for {set i 2} {$i < 9} {incr i} {
     for {set j [expr $i+1]} {$j < 9} {incr j} {
          set a LSR$i
          set b LSR$j
          eval $ns LDP-peer $$a $$b
     }
}

$ns ldp-request-color         blue
$ns ldp-mapping-color         red
$ns ldp-withdraw-color        magenta
$ns ldp-release-color         orange
$ns ldp-notification-color    yellow


set Src0 [new Application/Traffic/CBR]
set udp0 [new Agent/UDP]
$Src0 attach-agent $udp0
$ns attach-agent $Node0 $udp0
$Src0 set packetSize 500
$Src0 set interval 0.010

set Src1 [new Application/Traffic/CBR]
set udp1 [new Agent/UDP]
```

ตัวอย่าง แสดงสคริปการจำลองเครือข่าย MPLS (ต่อ)

```
$Src1 attach-agent $udp1
$ns attach-agent $Node1 $udp1
$Src1 set packetSize 500
$Src1 set interval 0.010

set Dst0 [new Agent/Null]
$ns attach-agent $Node9 $Dst0

set Dst1 [new Agent/Null]
$ns attach-agent $Node10 $Dst1

$udp0 set fid_ 0
$udp1 set fid_ 1
$ns connect $udp0 $Dst0
$ns connect $udp1 $Dst1

$ns at 0.1  "$Src0 start"
$ns at 0.1  "$Src1 start"

$ns at 1.8 "$Src0 stop"
$ns at 1.8 "$Src1 stop"

$ns at 2.0 "finish"
$ns run
```

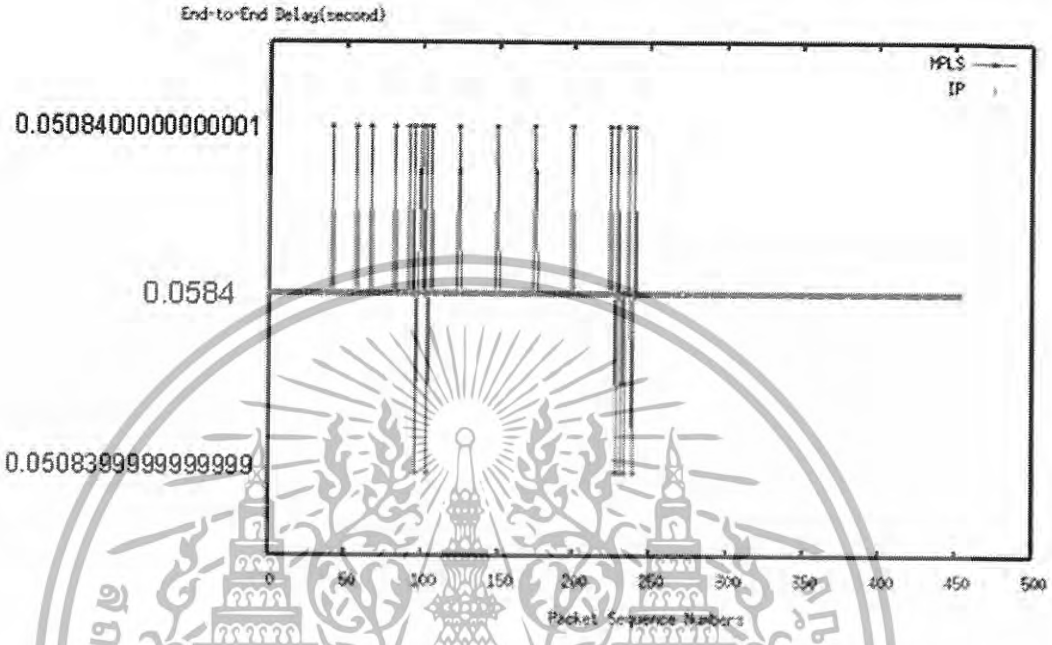
และเมื่อทำการรันสคริปต์ข้างต้นทั้งสองแล้วจะได้เครือข่ายไอพีและเครือข่าย MPLS ที่มีโทโพโลยีเดียวกัน ดังรูปที่ 3.27

รูปที่ 3.27 แสดงโทโพโลยีที่ใช้ในการทดลอง

จากนั้นจะนำผลลัพธ์ของการจำลองซึ่งอยู่ในรูปแบบของเทรซไฟล์มาทำการวิเคราะห์และวาดกราฟเพื่อเทียบประสิทธิภาพดังต่อไปนี้

3.8.1 การหาระยะเวลาหน่วงระหว่างต้นทางและปลายทาง (End-to-End Delay)

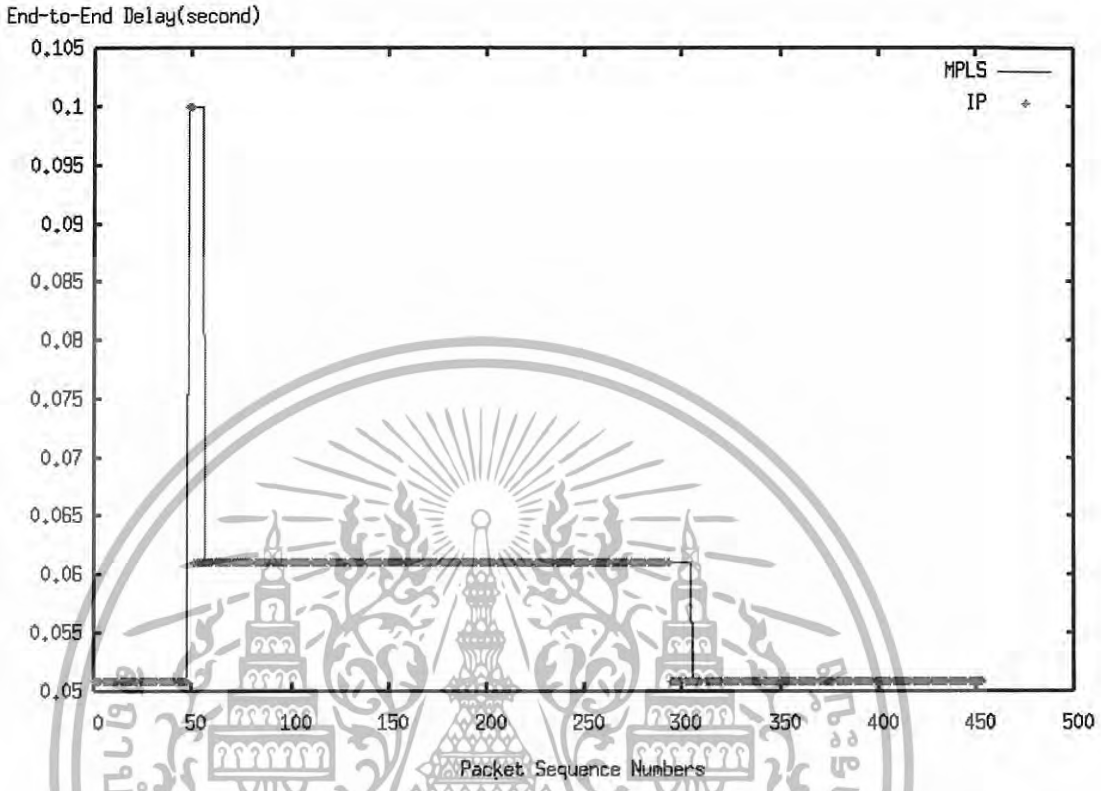
การเปรียบเทียบค่าระยะเวลาหน่วงของเครือข่ายไอพีและเครือข่าย MPLS โดยกำหนดให้ทุกลิงค์สามารถใช้งานได้ตามปกติ จะได้กราฟดังรูปที่ 3.28



รูปที่ 3.28 แสดงกราฟเปรียบเทียบเวลาหน่วงระหว่างเครือข่ายไอพีและเครือข่าย MPLS เมื่อสามารถใช้งานทุกลิงค์ได้ตามปกติ

จากกราฟพบว่าค่าระยะเวลาหน่วงระหว่างต้นทางกับปลายทาง ของทั้งเครือข่าย ไอพีและเครือข่าย MPLS นั้นมีค่าเท่ากันหมด เนื่องจากว่าการจำลองใน ns-2 แต่ละโหนดนั้นไม่มีเวลาสวิทชิ่ง หมายถึงไม่มีเวลาหน่วงที่จะเกิดจากกระบวนการหาเส้นทางที่จะใช้ส่ง แต่ในความเป็นจริงแล้วในระบบเครือข่ายจริงนั้น อุปกรณ์เนตเวิร์กจะมีเวลานี้อยู่ ซึ่งจะแตกต่างกันแล้วแต่ความสามารถของอุปกรณ์นั้น ซึ่งในทางทฤษฎีนั้น เราท์เตอร์ LSR ที่อยู่ในเครือข่าย MPLS นั้นจะสามารถทำงานได้ประสิทธิภาพดีกว่าเราท์เตอร์ในเครือข่ายไอพี แต่ในปัจจุบันนั้นเทคโนโลยีพัฒนาไปมากทำให้ความแตกต่างดังกล่าวนั้นน้อยมาก โดยจะสังเกตได้ว่าค่าเวลาหน่วงจะเปลี่ยนแปลงในช่วงที่แคบมากๆ โดยสรุปได้ว่าอาจเป็นผลจากเวลาของการประมวลผลของเครื่องที่ใช้ในการจำลอง ซึ่งอาจทำให้ค่าผิดเพี้ยนไปเล็กน้อย เนื่องจากในการทดลองแต่ละครั้ง ค่าความหน่วงที่แตกต่างนั้นจะเกิดกับแพ็คเกตใดก็ได้ไม่ซ้ำกัน

การเปรียบเทียบค่าระยะเวลาหน่วงของเครือข่ายไอพีและเครือข่าย MPLS โดยกำหนดให้เกิดกรณีที่ลิงค์ที่เชื่อมระหว่างโหนดที่ 2 และ โหนดที่ 5 ไม่ทำงาน



รูปที่ 3.29 แสดงกราฟเปรียบเทียบเวลาหน่วงระหว่างเครือข่ายไอพีและเครือข่าย MPLS เมื่อเมื่อลิงค์ที่เชื่อมระหว่างโหนดที่ 2 และ โหนดที่ 5 ไม่ทำงาน

จากรูปที่ 3.29 เป็นการเปรียบเทียบเวลาหน่วงระหว่างเครือข่ายไอพีและเครือข่าย MPLS โดยกำหนดให้มีการรันโปรโตคอลที่ใช้ในการหาเส้นทางที่เป็นแบบ Dynamic ที่เป็นลักษณะ Distance Vector เพื่อทำการค้นหาเส้นทางใหม่ในกรณีที่เส้นทางปกติใช้งานไม่ได้ โดยใช้คำสั่ง

```
$ns rtproto DV
```

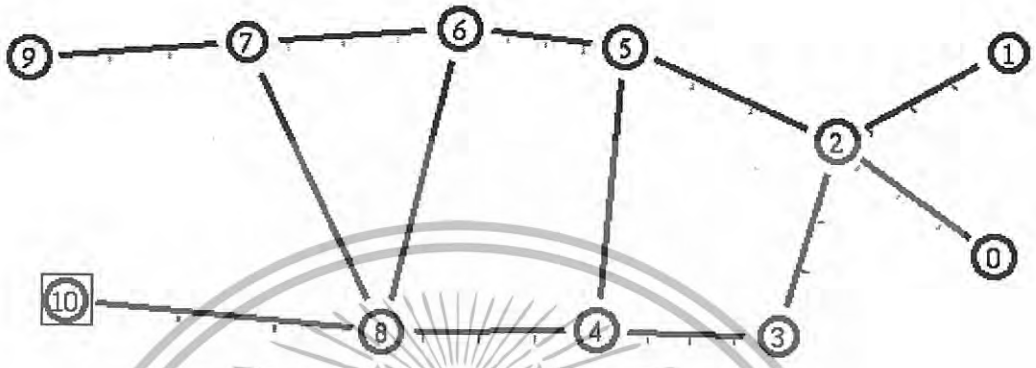
จากนั้นก็สั่งให้ลิงค์หยุดทำงานที่เวลาในการจำลองเท่ากับ 0.3 โดยใช้คำสั่ง

```
$ns rtmodel-at 0.3 down $Node2 $Node5
```

และสั่งให้ลิงค์กลับมาทำงานใหม่อีกครั้งที่เวลาที่ใช้ในการจำลองเท่ากับ 1.2 โดยใช้คำสั่ง

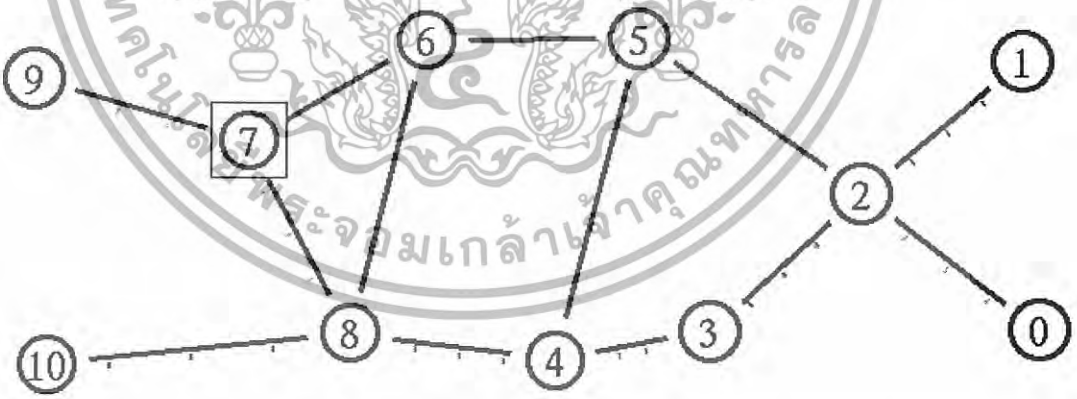
```
$ns rtmodel-at 1.2 up $Node2 $Node5
```

จากการทดลองจะพบว่า เมื่อลิงค์ทำงานได้ปกติจะมีการส่งแพ็คเกต โดยเส้นทางในการส่งแพ็คเกตจะเป็นดังรูปที่ 3.30 โดยโหนดที่ 0 จะส่งแพ็คเกตไปยังโหนดที่ 9 และโหนดที่ 1 ทำการส่งแพ็คเกตไปยังโหนดที่ 10



รูปที่ 3.30 แสดงเส้นทางการเดินทางของแพ็คเกตเมื่อเวลาปกติ

เมื่อลิงค์ระหว่างโหนดที่ 2 และโหนดที่ 5 ไม่ทำงานทำให้โหนดที่ 0 ไม่สามารถส่งแพ็คเกตไปยังโหนดที่ 9 ซึ่งเป็นโหนดปลายทางได้ เราท์เตอร์ของทั้งสองเครือข่ายจะมีการหาเส้นทางใหม่เพื่อที่ให้สามารถส่งแพ็คเกตไปยังโหนดที่ 9 จึงย้ายไปส่งเส้นทางใหม่ดังรูปที่ 3.31



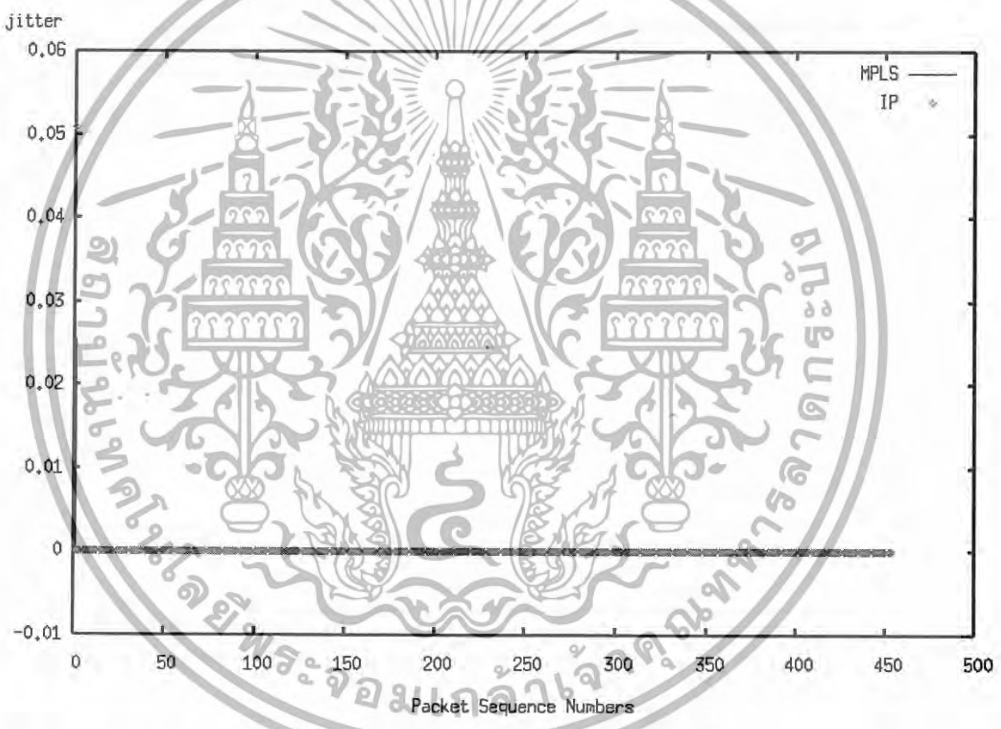
รูปที่ 3.31 แสดงเส้นทางการเดินทางของแพ็คเกตเมื่อลิงค์ไม่ทำงาน

จากกราฟจะพบว่า เมื่อช่วงเวลาปกติจะมีค่าระยะเวลาหน่วงค่าหนึ่ง เมื่อลิงค์ไม่ทำงานจะพบว่ามีเวลาหน่วงเกิดขึ้น เนื่องจากว่าเราท์เตอร์ของทั้งเครือข่ายไอพีและเครือข่ายMPLS ต้องทำการหาเส้นทางใหม่ที่มาแทนเส้นทางเดิม โดยใช้เราติ้งโปรโตคอล เพื่อปรับปรุงตารางเราติ้ง ซึ่งเส้นทางที่หามาใหม่นั้นจะมีจำนวน hop มากกว่าเดิม ทำให้ค่าเวลาหน่วงมีค่าเพิ่มขึ้น

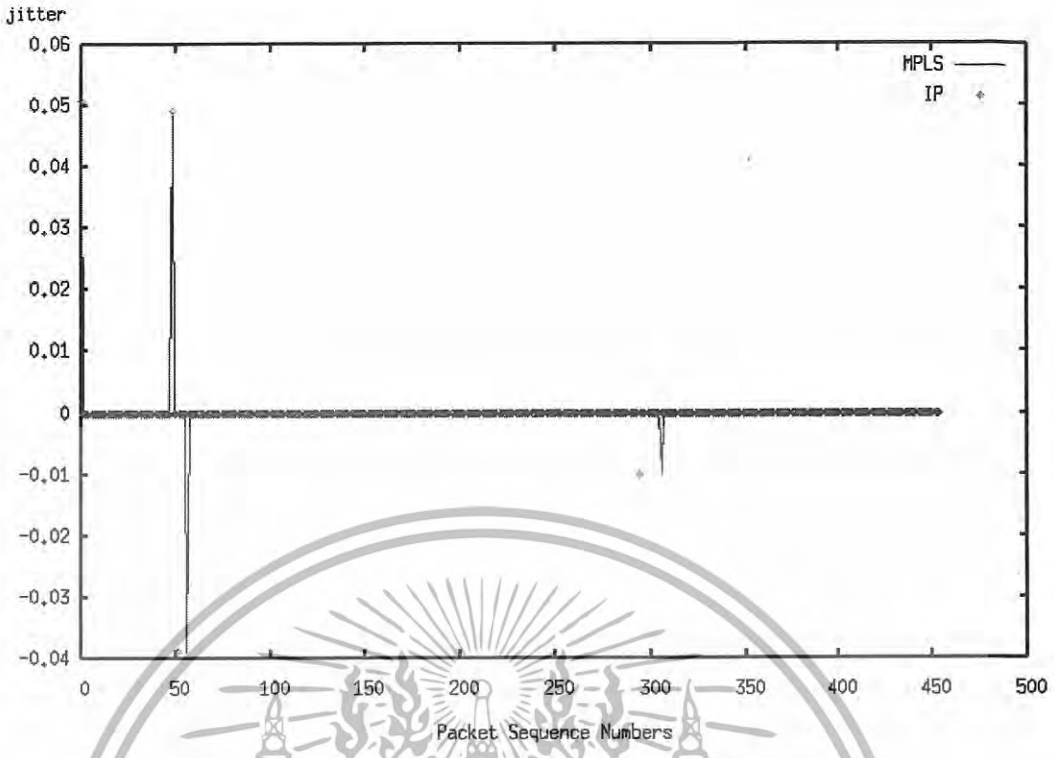
จนกระทั่งเมื่อลิงค์กลับมาใช้ได้เหมือนเดิมที่เวลาที่ 1.2 เราท์เตอร์ก็จะเปลี่ยนเส้นทางกลับมาส่งที่เส้นเดิม และพบว่าเครือข่ายที่เป็นแบบ MPLS นั้นจะใช้เวลาในการเรียนรู้เส้นทางใหม่มากกว่าเครือข่ายไอพีธรรมดา เนื่องจากว่าเครือข่ายที่เป็น MPLS จะต้องมีการแลกเปลี่ยนข้อมูลของลาเบลเพื่อการสร้างเส้นทางใหม่ทั้งตอนที่เส้นทางไม่ทำงานและตอนที่เส้นทางกลับมาใช้งานได้อีกครั้ง ซึ่งกระบวนการกาสร้างจะเกิดหลงจากมีการปรับปรุงตารางเราติ้งก่อน

3.8.2 การคำนวณหาค่าความแปรปรวนของเวลาหน่วง (jitter delay)

ในการหาค่า jitter จะแบ่งออกเป็นหาค่า jitter ในขณะที่ลิงค์ทุกเส้นสามารถทำงานได้ตามปกติ และจะหาในขณะที่มีลิงค์ระหว่างโหนดที่ 2 และ โหนดที่ 5 ไม่สามารถทำงานได้ ได้ผลการทดลองเป็นดังนี้



รูปที่ 3.32 แสดงค่าความแปรปรวนของเวลาหน่วงเมื่อลิงค์ทั้งหมดสามารถทำงานได้ตามปกติ

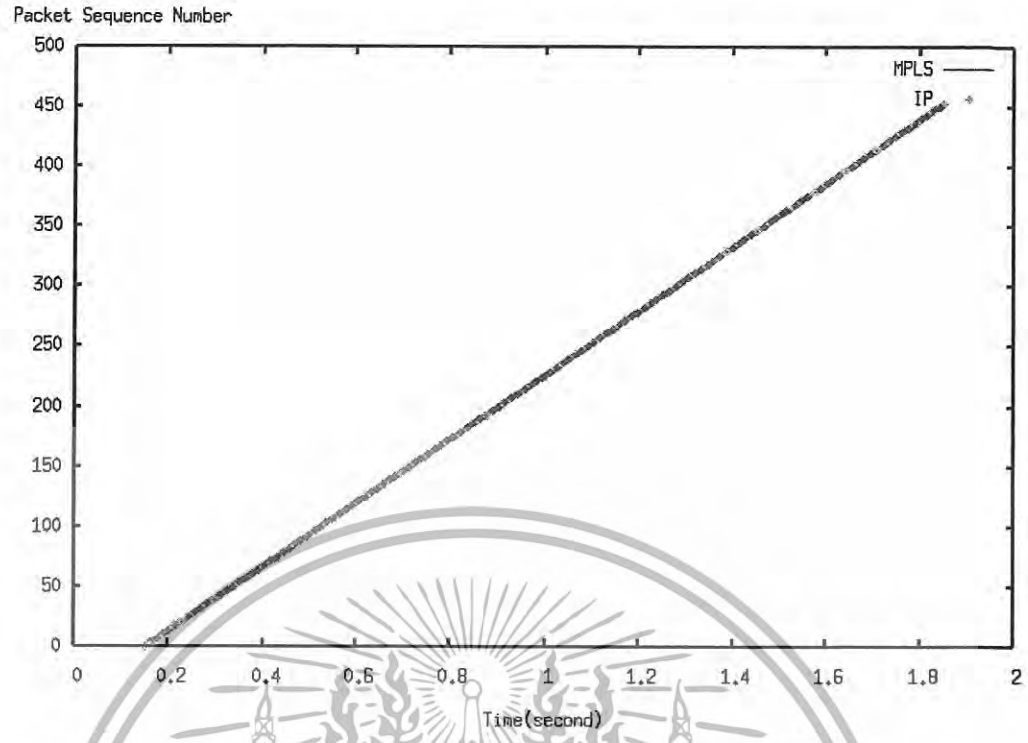


รูปที่ 3.33 แสดงค่าความแปรปรวนของเวลาหน่วงเมื่อลิงค์ระหว่างโหนดสองและโหนดห้าไม่ทำงาน

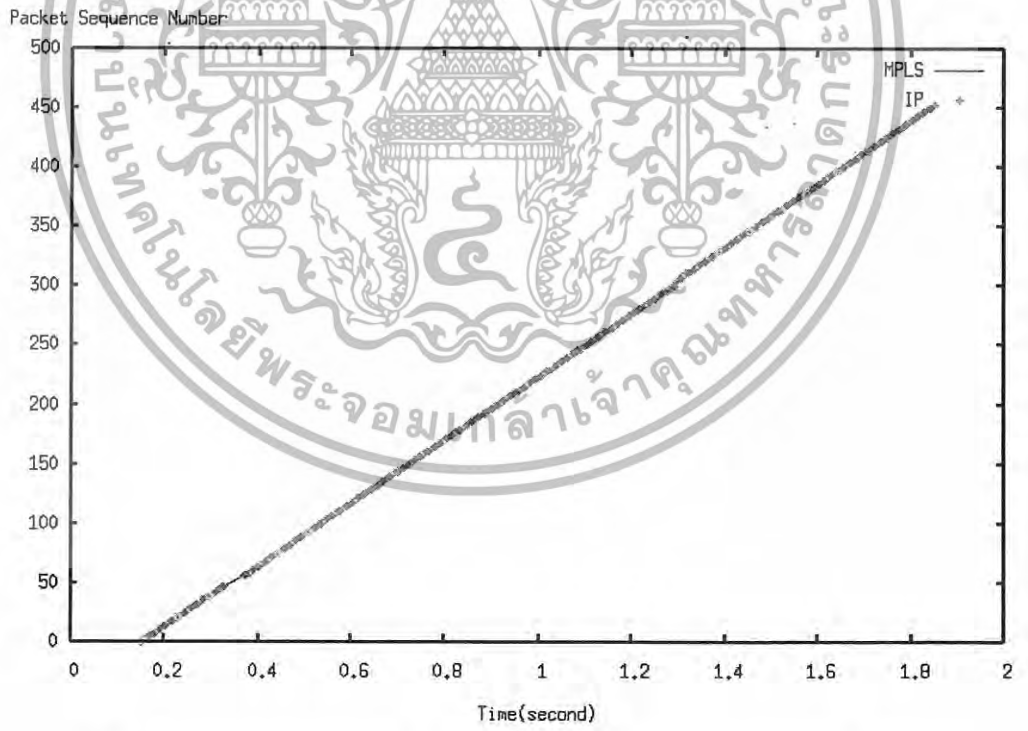
จากกราฟความแปรปรวนของเวลาหน่วงจะพบว่า โดยทั่วไปนั้นไม่มีการแปรปรวนของเวลาหน่วงคือมีค่าความแปรปรวนเท่ากับ 0 ในสถานการณ์ปกติ ทั้งเครือข่ายที่เป็นไอพีและเครือข่าย MPLS และเมื่อลิงค์ระหว่างโหนดที่ 2 และ โหนดที่ 5 ไม่สามารถทำงานได้ จะพบว่ามีค่าความแปรปรวนเกิดขึ้น เนื่องจากว่าต้องมีการเปลี่ยนแปลงเส้นทางในการส่งเพราะว่าลิงค์บางลิงค์ไม่สามารถทำงานได้ ทำให้ค่าเวลาหน่วงนั้นเปลี่ยนแปลงไป

3.8.3 การหาเลขลำดับแพ็คเกต (Sequence Number)

ในการหาเลขของลำดับแพ็คเกตนั้นจะแบ่งออกเป็นสองแบบคือ จะหาในขณะที่ลิงค์ทุกลิงค์สามารถทำงานได้ตามปกติ และเมื่อลิงค์ระหว่างโหนดสองและโหนดห้าไม่สามารถทำงานได้



รูปที่ 3.34 แสดงลำดับของแพ็คเกตเมื่อทุกลิงค์ทำงานได้ตามปกติ



รูปที่ 3.35 แสดงลำดับแพ็คเกตเมื่อลิงค์ระหว่างโหนดสองและโหนดห้าไม่ทำงาน

จะพบว่าเมื่อลิงค์ทุกลิงค์สามารถทำงานได้ตามปกตินั้นเลขลำดับของแพ็คเกตนั้นจะเรียงต่อเนื่องกัน หมายความว่าทุกแพ็คเกตสามารถส่งไปถึงผู้รับได้หมด โดยไม่หาย แต่เมื่อมีลิงค์ไม่ทำงานจะพบว่า เมื่อลิงค์เริ่มดาวน์ จะมีแพ็คเกต lost ไปบางส่วน ทั้งเครือข่ายไอพีและเครือข่าย MPLS จากนั้นเมื่อมีการค้นหาเส้นทางใหม่เรียบร้อยแล้วก็กลับมาส่งข้อมูลได้อีกครั้ง ซึ่งจะเห็นว่าทั้งเครือข่ายไอพีและเครือข่าย MPLS นั้น มีข้อมูลหายเท่าๆกัน แสดงว่าใช้เวลาในการค้นหาเส้นทางพอๆกัน

จากผลการทดลองการเปรียบเทียบประสิทธิภาพของเครือข่ายไม่ว่าจะเป็น เวลาหน่วง ค่าความแปรปรวนของเวลาหน่วง หรือว่าหมายเลขลำดับแพ็คเกต ของเครือข่ายทั้งสองแบบนั้นจะไม่เห็นความแตกต่างชัดเจนนัก เนื่องจากว่าการจำลองใน NS-2 นั้น โหนดต่างๆ เมื่อได้รับข้อมูลมาจะส่งออกไปยังปลายทางเลย ไม่มีเวลาหน่วงในการสวิทชิ่งคือเวลาที่ใช้เทียบเฮดเดอร์ของแพ็คเกตกับตารางเราท์ติ้งว่าจะส่งออกไปยังปลายทางใด ทำให้เมื่อผลการทดลองของเครือข่ายไอพีกับเครือข่าย MPLS นั้นค่าออกมาเท่าๆกัน

บทที่ 4
เครือข่ายไร้สายเฉพาะกิจ

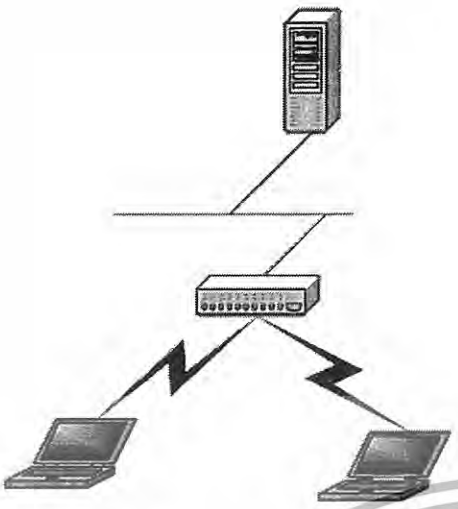
4.1 Mobile Ad Hoc Network

ระบบเครือข่ายเฉพาะกิจ (Mobile Ad hoc Network: MANET) เป็นระบบที่ใช้เทคโนโลยีไร้สายตามมาตรฐานของ IEEE 802.11 (Wireless Local Area Network Protocol) ในการติดต่อสื่อสารและแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์ขนาดพกพา โดยใช้พลังงานจากแบตเตอรี่ในตัวอุปกรณ์พกพา ซึ่งไม่จำเป็นต้องอาศัยสถานีแม่ (Access Point) ในการติดต่อสื่อสาร ซึ่งจะกล่าวถึงสถาปัตยกรรมของ IEEE 802.11 พื้นฐานที่จำเป็น การแก้ปัญหา Hidden และ Exposed Terminal โดยการใช้ RTS/CTS การทำงานของ Back off

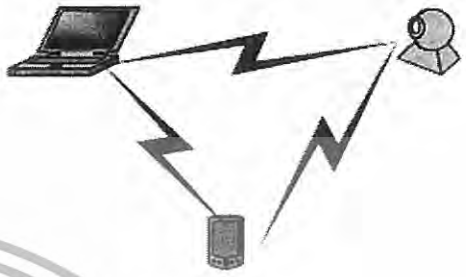
การติดต่อสื่อสารของระบบคอมพิวเตอร์เริ่มเปลี่ยนจากการใช้ระบบสายมาสู่การใช้สัญญาณวิทยุ (Radio Frequency: RF) เป็นสื่อกลางในการแลกเปลี่ยนข้อมูล ส่งผลให้เกิดการเปลี่ยนแปลงในการติดต่อสื่อสาร เนื่องจากผู้ใช้งานสามารถใช้งานเครื่องคอมพิวเตอร์ในการเข้าถึงข้อมูลได้อย่างไม่จำกัดเวลาและสถานที่ ระบบเครือข่ายสามารถแบ่งได้หลายประเภท แต่ถ้าพิจารณาตามรูปแบบการเชื่อมต่อกับโครงข่ายสามารถแบ่งได้เป็น 2 ประเภท คือ

1. อาศัยโครงข่ายพื้นฐานในการแลกเปลี่ยนข้อมูล (Infrastructure Network) ซึ่งระบบเครือข่ายประกอบไปด้วยอุปกรณ์เครื่องให้บริการ (Server) และเครื่องขอใช้บริการ (Client) รวมถึงอุปกรณ์การติดต่อสื่อสารอื่นๆ และต้องใช้อุปกรณ์เครือข่าย เช่น Hub, Switching, Access Point (AP) หรือ Router ทำหน้าที่เป็นเครื่องบริการการแลกเปลี่ยนข้อมูล ถ้าอุปกรณ์เครือข่ายทำงานผิดพลาดจะส่งผลให้การติดต่อสื่อสารในระบบเครือข่ายนั้นใช้ไม่ได้เครือข่ายในปัจจุบันส่วนใหญ่ใช้โครงข่ายพื้นฐานนี้

2. แบบที่ไม่มีโครงข่ายพื้นฐาน (Infrastructure-less Network) การติดต่อสื่อสารของอุปกรณ์ในระบบนี้ไม่จำเป็นต้องอาศัยอุปกรณ์เครือข่ายช่วยในการแลกเปลี่ยนข้อมูล ซึ่งในโครงข่ายแบบนี้ประกอบไปด้วยอุปกรณ์คอมพิวเตอร์ขนาดพกพาได้ (Mobile Devices) โดยใช้เทคโนโลยีไร้สายเป็นตัวกลาง (Transmission Medium) ในการติดต่อสื่อสาร และมีการทำงานแบบกระจาย (Distributed Computing) คือ ไม่มีจุดศูนย์กลางในการควบคุมการทำงานของระบบ แต่ใช้การทำงานร่วมกันเป็นหลัก ซึ่งเรียกอีกชื่อว่า เครือข่ายเฉพาะกิจ (Mobile Ad hoc Network : MANET) ดังรูปที่ 4.1



รูปที่ 4.1a. โครงข่ายพื้นฐาน



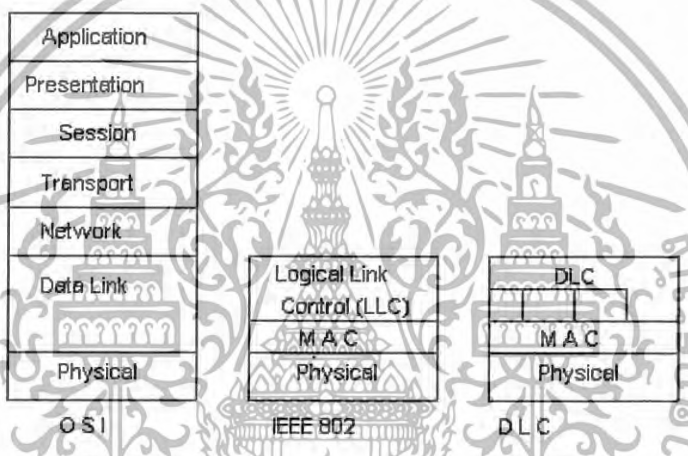
รูปที่ 4.1b. เครือข่าย MANET

MANET คือ ระบบที่ประกอบด้วยอุปกรณ์พกพาซึ่งสามารถแลกเปลี่ยนข้อมูลโดยใช้เทคโนโลยีไร้สาย และไม่จำเป็นต้องอาศัยสถานีแม่ (Access Point) ช่วยในการติดต่อสื่อสาร อุปกรณ์พกพาเหล่านี้ใช้พลังงานจากแบตเตอรี่และรูปแบบการเชื่อมต่อของเครือข่าย (Network Topology) สามารถเปลี่ยนแปลงได้ตลอดเวลา ตัวอย่างการประยุกต์ใช้งานเครือข่าย MANET เช่น ในสนามรบ สถานที่เกิดภัยธรรมชาติ ห้องเรียน ห้องประชุม หรือสถานพยาบาล ซึ่งไม่มีโครงข่ายพื้นฐานแบบคงที่อยู่ MANET แบ่งตามระยะทาง การติดต่อสื่อสารได้ 4 ประเภท คือ

1. ระบบเครือข่ายไร้สายสำหรับเครื่องคอมพิวเตอร์แบบสวมใส่ (Body Area Network: BAN)
2. ระบบเครือข่ายส่วนบุคคล (Personal Area Network: PAN)
3. ระบบเครือข่ายงานเฉพาะที่ (Local Area Network: LAN)
4. ระบบเครือข่ายงานบริเวณกว้าง (Wide Area Network: WAN)

ระบบเครือข่ายไร้สายสำหรับแลน (Wireless Local Area Network: WLAN) เริ่มมีบทบาทมากขึ้นในปัจจุบันเนื่องจากการเพิ่มจำนวนของเครื่องคอมพิวเตอร์ที่มีขนาดเล็ก เช่น เครื่อง Personal Digital Assistant (PDA) เครื่องคอมพิวเตอร์ส่วนตัวแบบพกพา (Notebook Computer) อุปกรณ์เหล่านี้ทำงานภายใต้สภาวะที่มีการเคลื่อนที่ และจำเป็นต้องอาศัยเครือข่ายไร้สายเป็นสื่อกลางในการแลกเปลี่ยนข้อมูล ส่งผลให้ความต้องการใช้งานระบบเครือข่ายไร้สายสำหรับแลนเพิ่มขึ้นอย่างมาก ระบบเครือข่ายไร้สายสำหรับแลนช่วยให้การทำงานมีความคล่องตัวสูงขึ้น เนื่องจากผู้ใช้งานสามารถเคลื่อนย้ายอุปกรณ์พกพาไร้สายไปตามที่ต่างๆ เช่น ภายในมหาวิทยาลัย สนามบิน ร้านอาหาร หรือโรงแรม ซึ่งสามารถเชื่อมต่อเข้ากับระบบอินเทอร์เน็ตได้ตลอดเวลา

MANET ในระบบเครือข่ายไร้สายสำหรับแลน เริ่มมีการใช้งานมากขึ้นในปัจจุบัน เนื่องจากระยะทางการติดต่อสื่อสารที่ไม่ไกลประมาณ 1 ถึง 2 จุด (Single or 2 Hops) และมีการเปลี่ยนรูปแบบการเชื่อมต่อกับเครือข่ายไม่บ่อยครั้ง แต่ MANET สำหรับเครือข่ายงานบริเวณกว้าง นั้นมีความซับซ้อนสูง เนื่องจากมีการเชื่อมต่อกับเครือข่ายหลายจุด (Multi-Hop) และมีการเปลี่ยนแปลงรูปแบบการเชื่อมต่อได้ตลอดเวลา ดังนั้นในการออกแบบและติดตั้ง MANET สำหรับเครือข่ายงานบริเวณกว้าง นั้นจะมีปัจจัยต่างๆที่ควรนำมาพิจารณา เช่น การกำหนดหมายเลขเครื่อง (Addressing) การหาเส้นทาง (Routing) การประหยัดพลังงานจากแบตเตอรี่ (Power Management) การประกันคุณภาพของบริการ (Quality of Service) และการรักษาความปลอดภัยของข้อมูล (Security) เทคโนโลยีพื้นฐานที่รองรับเครือข่าย MANET คือเทคโนโลยีไร้สาย ใช้มาตรฐานของ IEEE 802.11



รูปที่ 4.2 มาตรฐาน OSI และ IEEE 802

4.1.1 มาตรฐาน IEEE 802.11 Wireless LAN

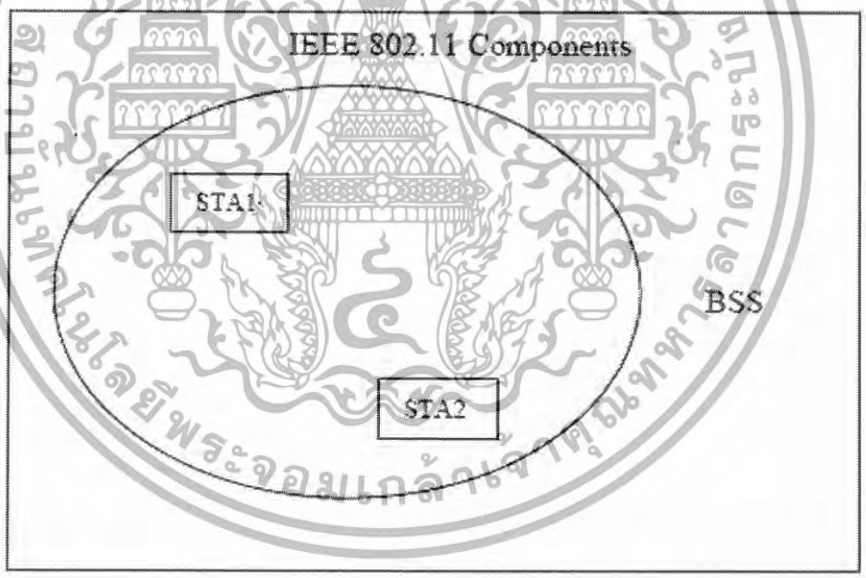
มาตรฐาน IEEE 802.11 Wireless LAN สร้างขึ้นในปี ค.ศ. 1997 โดยองค์กร The Institute of Electrical and Electronics Engineers ซึ่งระบบมีอัตราความเร็วในการรับส่งข้อมูลที่ 2 ล้านบิตต่อวินาที มาตรฐานนี้ตรงกับลำดับชั้นที่ 1 และ 2 ของ Open System Interconnect (OSI) ซึ่งแยกเป็น Physical (PHY) และ Medium Access Control (MAC) ดังรูปที่ 2.2 ในเวลาถัดมามีการแบ่ง IEEE 802.11 เป็น 2 มาตรฐานคือ IEEE 802.11a และ IEEE 802.11b ซึ่งใช้ Physical Layer เทคโนโลยีที่แตกต่างกัน สำหรับ IEEE 802.11b ใช้สัญญาณวิทยุที่ประมาณ 2.4 GHz ISM (Industrial, Scientific and Medical) มีอัตราเร็ว 11 ล้านบิตต่อวินาที มาตรฐาน IEEE 802.11b ประกาศใช้ในปี ค.ศ. 1999 และมีการใช้งานอย่างแพร่หลายในปัจจุบัน สำหรับมาตรฐาน IEEE 802.11a ใช้สัญญาณวิทยุที่ประมาณ 5 GHz และมีอัตราเร็วสูงถึง 54 ล้านบิตต่อวินาที ข้อกำหนดของ Physical Layer ใน IEEE 802.11 WLAN ประกอบด้วย 3 เทคโนโลยีคือ

1. Direct Sequence Spread Spectrum (DSSS) โดยใช้การส่งข้อมูลกระจายไปตามช่วงความถี่ที่ 2.4 GHz ถึง 2.4835 GHz เทคโนโลยีที่

2. Frequency Hopping Spread Spectrum (FHSS) ซึ่งใช้การแบ่งข้อมูลเป็นขนาดเล็ก และส่งไปตามช่องความถี่ที่กำหนดไว้แล้ว เช่น ข้อมูลชุดที่หนึ่งส่งไปโดยใช้ความถี่ที่ 1 ข้อมูลชุดที่สองส่งไปโดยใช้ความถี่ที่ 2 และข้อมูลชุดที่สามกระโดดไปใช้ความถี่ที่ 10 โดยไม่จำเป็นต้องเรียงช่องสัญญาณความถี่

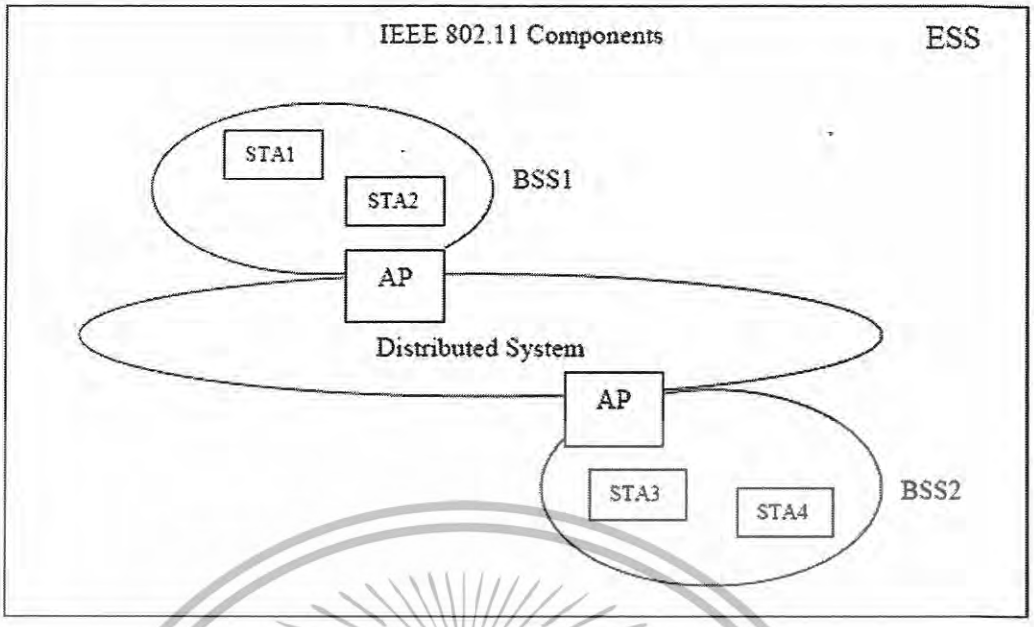
3. Infrared (IR) ซึ่งไม่เป็นที่นิยมใช้ เนื่องจากมีข้อจำกัดว่าต้องไม่มีสิ่งกีดขวางระหว่างเครื่องรับและเครื่องส่งสัญญาณ เพราะใช้แสงเป็นสื่อกลางในการส่งข้อมูล เช่น รีโมตคอนโทรลทั่วไป

Basic Service Set (BSS) เป็นสถาปัตยกรรมพื้นฐานในมาตรฐาน IEEE 802.11 หมายถึงกลุ่มของสถานี (STA) ที่อยู่รวมกันภายใต้การควบคุมของ DCF หรือ PCF (ควบคุมโดย AP) สถานีที่อยู่ใน BSS ไม่สามารถติดต่อสื่อสารกันได้ถ้าเคลื่อนที่ออกจาก BSS จากรูปที่ 4.3 ถ้า STA1 เคลื่อนที่ออกจาก BBS จะไม่สามารถติดต่อสื่อสารกับ STA2 ได้



รูปที่ 4.3 Basic Service Set

สำหรับเครือข่าย MANET มีชื่อเรียกว่า Independent BSS (IBSS) ซึ่งประกอบด้วย 2 สถานีขึ้นไปติดต่อสื่อสารโดยไม่ต้องอาศัย AP Extended Service Set (ESS) ดังรูปที่ 4.4 ประกอบด้วยหลาย BSS ที่มีการเชื่อมต่อกันโดยใช้ Distribution System (DS) ซึ่งทำหน้าที่ให้บริการและจัดการเรื่องหมายเลขของเครื่องสถานี เพื่อให้อนุญาตให้ STA เคลื่อนที่และติดต่อสื่อสารข้าม BSS ได้



รูปที่ 4.4 Extended Service Set

ส่วนข้อกำหนดของ Medium Access Control Layer แบ่งเป็น 2 ส่วนคือ

1. Point Coordination Function (PCF) ซึ่งใช้งานได้กับเครือข่ายที่มีในโครงข่ายพื้นฐานเท่านั้น การติดต่อสื่อสารในส่วนนี้ไม่มีการชนกัน (No Collision) ของเฟรมข้อมูล โดยสถานีแม่จะวนถาม (Pooling) สถานีลูกในระบบว่าต้องการส่งข้อมูลหรือไม่ สถานีแม่ส่วนใหญ่คือ AP ดังนั้นเครือข่าย MANET ไม่สามารถใช้หน้าที่ของ PCF ได้

2. Distributed Coordination Function (DCF) ซึ่งใช้โปรโตคอล Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) ซึ่งเครือข่าย MANET ใช้การทำงานของ DCF เป็นหลัก เนื่องจากไม่จำเป็นต้องใช้สถานีแม่ในการรับส่งข้อมูล ทุกสถานีมีสิทธิในการรับส่งข้อมูลเท่าเทียมกัน

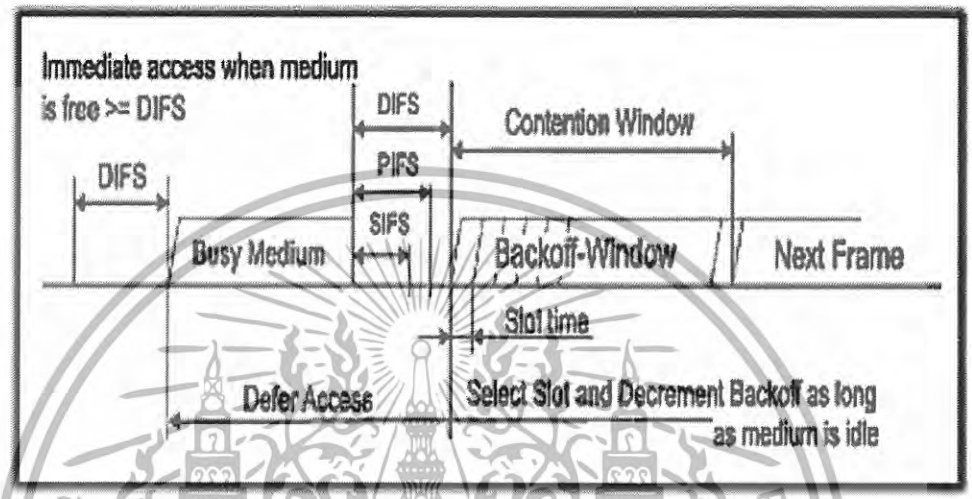
4.1.2 The IEEE 802.11 Distributed Coordination Function (DCF)

การทำงานของ DCF ซึ่งแบ่งได้เป็น 2 หมวดคือ 1. Basic Access (BA) ซึ่งใช้ Two-Way Hand shaking 2. Request-To-Send/Clear-To-Send (RTS/CTS) หรือ Four-Way Handshaking ทั้ง 2 หมวดใช้โปรโตคอล CSMA/CA มีวัตถุประสงค์เพื่อลดการชนกันของเฟรมข้อมูล เนื่องจากการทำงานใน DCF ใช้สื่อกลางรวมกัน ถ้ามีการส่งเฟรมข้อมูลจาก 2 สถานีพร้อมกันจะเกิดการชนกัน สำหรับ BA มีขั้นตอนการทำงานดังต่อไปนี้

1. สถานีที่ต้องการส่งข้อมูล ฟังช่องสัญญาณของสื่อกลางว่าไม่มีสถานีใดใช้อยู่

2. ถ้าช่องสัญญาณว่างเป็นเวลาที่มากกว่า DCF Interframe Space (DIFS) สถานีเริ่มส่งข้อมูลได้ และรอสัญญาณตอบรับ (Positive Acknowledgment: ACK) โดยมีระยะเวลาระหว่างเฟรมข้อมูลกับ ACK เป็น Short Interframe Space (SIFS) ดังรูปที่ 4.5

3. ถ้าช่องสัญญาณไม่ว่าง สถานีเริ่มใช้ขั้นตอน Back off เพื่อรอแบบสุ่มก่อนที่จะส่งข้อมูลได้เพื่อหลีกเลี่ยงการชนกันของเฟรมข้อมูล



รูปที่ 4.5 Basic Access Method

จากรูปที่ 4.5 Contention Window (CW) เป็นช่วงเวลาที่แบ่งเป็นช่องเวลา (Slot Time) เพื่อให้สถานีใน BSS หรือ IBSS สามารถแข่งขันกันเพื่อใช้สิทธิ์ในการเข้าใช้ช่องสัญญาณ แต่ในกรณีที่มีการแย่งใช้ช่องสัญญาณเดียวกัน จะเกิดการชนขึ้น ตามมาตรฐานเมื่อเกิดการชนของเฟรมข้อมูลขึ้นช่องสัญญาณจะต้องว่างเป็นเวลา Extended Interframe Space (EIFS) ก่อนที่สถานีต่างๆ เริ่มทำงานอีกครั้งหนึ่ง ค่า CW ของแต่ละสถานีถูกสุ่มขึ้นโดยมีค่าเป็นจำนวนเต็มอยู่ระหว่าง CWmin และ CWmax ขึ้นอยู่เทคโนโลยีใน PHY (เช่นค่า CWmin = 16 และ CWmax = 1024 สำหรับ FHSS)

ในการพยายามส่งครั้งแรก สถานีจะใช้ค่า CW เท่ากับ CWmin และถ้าส่งไม่สำเร็จเนื่องจากการชนกันหรือ Cyclic Redundancy Check (CRC) ของเฟรมข้อมูลผิด สถานีจะเพิ่มค่า CW ครั้งละสองเท่าทุกครั้งสำหรับการส่งที่ไม่สำเร็จแต่ไม่เกินค่า CWmax ในมาตรฐานของ IEEE 802.11 ไม่ได้อธิบายความสัมพันธ์ของค่า CW กับจำนวนสถานี ซึ่งความสัมพันธ์ของ CWmin และ CWmax กับจำนวนสถานีใน BSS เป็นดังต่อไปนี้

1. ค่า CWmin ที่น้อยเกินไป และสถานีที่มีจำนวนมาก ส่งผลให้มีการชนกันของเฟรมข้อมูลมาก และทำให้ประสิทธิภาพของระบบเครือข่ายลดลง

2. ค่า CWmin ที่มีค่ามากเกินไป และสถานีที่มีจำนวนน้อย ทำให้ประสิทธิภาพของระบบเครือข่ายแย่ลง เนื่องจากค่า Back off Time ที่มาก ทำให้เสียเวลารอนาน ก่อนที่จะมีสิทธิส่งเฟรมข้อมูล

3. ค่า CWmax ที่มีค่าน้อยเกินไป และสถานีที่มีจำนวนมาก ทำให้ประสิทธิภาพของระบบเครือข่ายลดลง

4. ค่า CWmax มีผลต่อต่อระบบไม่มาก แต่ไม่ควรลดค่า CWmax ลงในระบบเครือข่ายที่มีจำนวนสถานีมากในระหว่าง Sender ใช้ช่องสัญญาณอยู่

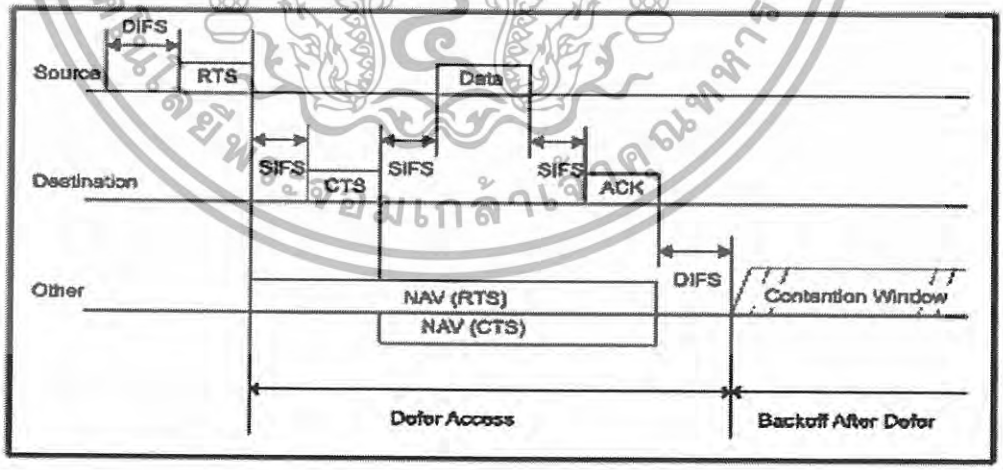
โดยสถานีอื่นจะทราบค่าว่าช่องสัญญาณนี้จะถูกใช้งานเป็นเวลาอีกนานเท่าไร และเก็บค่านี้ไว้ในตัวแปรที่ชื่อ Network Allocation Vector (NAV) ซึ่งหมายความว่า สถานีอื่นต้องรอเป็นเวลา NAV ก่อนที่สามารถเริ่มขั้นตอน Back off ได้ ค่า NAV มีประโยชน์สำหรับการจัดการพลังงานจากแบตเตอรี่ เนื่องจากในช่วงเวลาที่ช่องสัญญาณมีการใช้งาน สถานีอื่นจะทราบว่าช่องสัญญาณนี้จะถูกใช้งานเป็นเวลานานเท่าใด ทำให้สามารถหยุดการทำงาน (Sleep) เพื่อประหยัดพลังงานและทำงานต่อ (Awake) เมื่อเวลาผ่านเท่ากับ NAV การทำงานของ DCF ในหมวดที่ 2 คือ RTS/CTS เกิดขึ้นสืบเนื่องจากปัญหาของ Hidden และ Exposed Terminal จากรูปที่ 4.6a แสดง Hidden Terminal Problem สถานี STA1 ต้องการส่งเฟรมข้อมูลไปยัง STA3 ในขณะเดียวกัน STA2 ต้องการส่งเฟรมข้อมูลไปที่ STA3 และ STA1 กับ STA2 ไม่อยู่ใน BSS เดียวกัน ส่งผลให้เกิดการชนกันของเฟรมข้อมูลที่ STA3

รูปที่ 4.6 Hidden และ Exposed Terminal Problem

ปัญหาของ Exposed Terminal ในรูปที่ 2.6b คือ STA2 อยู่ในจังหวะที่ส่งข้อมูลไปที่ STA1 ทำให้ STA3 ไม่สามารถส่งข้อมูลไป STA4 ได้เนื่องจากได้ยินสัญญาณจาก STA2 ซึ่งในกรณีนี้ STA3 สามารถส่งข้อมูลไปยัง STA4 ได้ เพราะสัญญาณความถี่ของ STA3 ไม่รบกวน STA1 เพราะอยู่ต่าง BSS เพื่อแก้ปัญญา Hidden Terminal ดังที่กล่าวไป DCF มีโหมดที่ใช้สัญญาณ RTS/CTS ในการบอกให้สถานีอื่นในระบบทราบถึงการส่งข้อมูลเพื่อหลีกเลี่ยงการชนกันของเฟรมข้อมูล และเพิ่มประสิทธิภาพการส่งข้อมูลให้มากขึ้น หลักการทำงานของ RTS/CTS

แสดงในรูปที่ 4.7 ก่อนที่ STA1 จะเริ่มส่งเฟรมข้อมูล ต้องส่งเฟรมข้อมูลสั้นๆ ที่บรรจุค่า NAV และ RTS ไปที่ STA3 เพื่อบอกให้ผู้รับและสถานีอื่นในเครือข่ายทราบถึงเฟรมข้อมูลที่ต้องการส่ง STA1 รอเป็นเวลา SIFS เพื่อรอรับสัญญาณ CTS (บรรจุค่า NAV CTS) จาก STA3 การส่งเฟรมข้อมูลจึงเริ่มขึ้น ในกรณีที่ STA3 มีการติดต่อกับสถานีอื่นอยู่ซึ่ง STA1 ไม่สามารถรับสัญญาณนี้ได้ STA3 จะไม่ตอบ CTS กลับไปหา STA1 ทำให้ไม่เกิดการชนกันของเฟรมข้อมูล สำหรับการแก้ปัญหาปัญหา Exposed Terminal ในรูปที่ 4.6b เมื่อ STA2 ส่ง RTS ไปหา STA1 เพื่อขอส่งข้อมูล STA1 ตอบ CTS กลับไป STA2 เพื่อแสดงว่าพร้อมที่จะรับข้อมูล STA3 ได้รับข้อมูล RTS นี้ด้วยแต่ไม่ได้ยินสัญญาณ CTS ของ STA1 และเมื่อเวลาผ่านไปเท่ากับ SIFS ทำให้ STA3 ทราบว่า STA1 ไม่อยู่ใน BSS เดียวกับ STA3 ทำให้ STA3 สามารถส่งข้อมูลไปที่ STA4 ได้ ข้อสังเกต RTS/CTS อาจทำให้การทำงานใน BSS เดียวกันช้าลงเนื่องจากต้องใช้เวลานานในการโต้ตอบของ RTS/CTS แต่อาจเพิ่มประสิทธิภาพในกรณีของ ESS

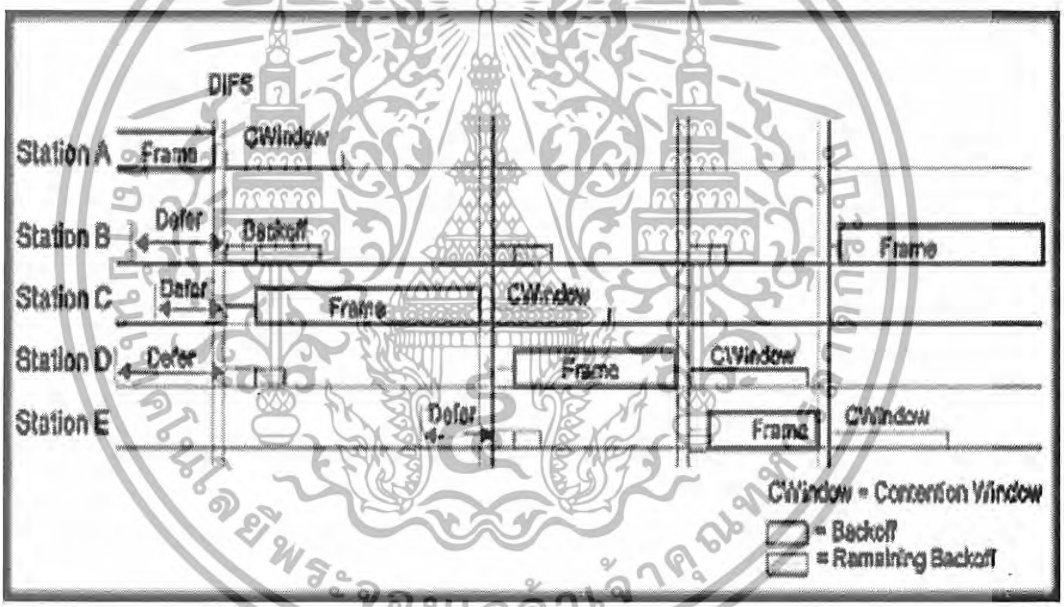
RTS และ CTS นั้น ใช้เพื่อ clear ช่องสื่อสารให้ว่าง ช่วยการันตีว่าไม่มีการเข้าใช้ AP พร้อมกัน และจะไม่เกิด collision ที่ AP แต่ใน network เล็กๆ จะไม่ใช้ RTS/CTS ก็ได้ เพราะมันเป็น overhead ที่ไม่เกี่ยวข้องกับข้อมูลที่จะส่งเลย หากข้อมูลที่จะสื่อสารมีขนาดเล็กๆ จะใช้ RTS/CTS ก็ไม่เหมาะสม เพราะ RTS/CTS มันกิน resource มากกว่าการส่งข้อมูลเสียอีก แถมยังทำให้เกิด latency (ความหน่วง) อีกด้วย แต่เราสามารถควบคุมได้ว่าจะใช้ RTS/CTS โดยการตั้ง threshold เอาไว้ (อุปกรณ์ต้อง support ด้วย) ว่าจะใช้ RTS/CTS เมื่อขนาดของข้อมูลใหญ่กว่า threshold ซึ่งเป็นการลด overhead ที่เกิดจาก RTS/CTS



รูปที่ 4.7 การทำงานของ RTS/CTS

4.1.3 ขั้นตอน Back off

Back off เริ่มขึ้นเมื่อสถานีพบว่า ช่องสัญญาณไม่ว่างโดยใช้วิธี BA หรือ CTS/RTS สถานีจะสุ่มค่า Back off Time โดยใช้ค่า CW ในการหาค่า Back off Time ซึ่งเท่ากับ Integer (CW*Random (0,1))*Slot Time (ค่า Slot Time มีค่าตาม PHY สำหรับ FHSS เป็น 50μs DSSS เป็น 20μs และสำหรับ IR เป็น 8μs) ค่า Back off Time มีการลดค่าลงหลังจากช่องสัญญาณว่างเป็นเวลาเท่ากับ DIFS เป็นต้นไปถัดจากนั้นเป็นช่วงเวลา CW ในช่วงเวลานี้สถานีใดมีค่า Back off Time เป็น 0 สามารถส่งเฟรมข้อมูลได้ทันที โดยมีข้อแม้ว่าต้องไม่มีสถานีอื่นใช้ช่องสัญญาณอยู่ในขณะนั้นแต่ถ้าพบว่ามีสถานีอื่นใช้ช่องสัญญาณไปแล้ว ต้องรอส่งในช่วง CW ถัดไป แต่กรณีค่า Back off Time ไม่เท่ากับศูนย์ สถานีจะลดค่า Back off Time ตามค่าของ Slot Time ใน CW ที่ผ่านไป แต่เมื่อช่องสัญญาณไม่ว่าง สถานีจะหยุดลดค่า Back off Time และจะลดค่า Back off Time อีกครั้งเมื่ออยู่ในช่วงเวลา CW ดังภาพที่ 4.8



รูปที่ 4.8 ขั้นตอนของ Back off

4.2 โปรโตคอล Dynamic Source Routing

Dynamic Source Routing หรือ DSR เป็นเราติ้งโปรโตคอลชนิดหนึ่งที่ใช้กันอย่างแพร่หลายในเครือข่ายเฉพาะกิจหรือที่เรียกว่า MANET (Mobile Ad-hoc Network) ซึ่งเป็นเครือข่ายของอุปกรณ์ไร้สายที่เชื่อมต่อกันโดยไม่ผ่านอุปกรณ์ที่ทำหน้าที่ส่งผ่านที่เรียกว่า Access Point ทำให้การเชื่อมต่อระหว่างอุปกรณ์จะเป็นการเชื่อมต่อกันโดยตรง แต่ในกรณีที่ต้นทางและปลายทางอยู่ไกลเกินกว่าที่สัญญาณวิทยุจะไปถึงได้ ทำให้มีความจำเป็นต้องมีการส่งผ่านอุปกรณ์ไร้สายอื่นที่อยู่ระหว่างทาง ดังนั้นจึงมีความจำเป็นที่จะต้องหาเส้นทางจากต้นทางไปถึงปลายทาง

DSR เป็นเราติ้งโปรโตคอลชนิดที่ขึ้นอยู่กับความต้องการหรือที่เรียกว่า On-Demand Protocol หรือเรียกอีกชื่อหนึ่งว่า Reactive Protocol โดยเส้นทางจะถูกสร้างขึ้นก็ต่อเมื่อต้นทางมีความต้องการจะส่งข้อมูลเท่านั้น ซึ่งตรงข้ามกับอีกชนิดที่จะมีการสร้างรอไว้ก่อนซึ่งเรียกว่า Proactive Protocol เช่นโปรโตคอล DSDV นอกจากนี้โปรโตคอลประเภท Reactive ยังมีโปรโตคอลชนิดอื่นอีกเช่น AODV หรือ TORA

โดยเมื่อต้นทางจะทำการส่งข้อมูล จะมีการพยายามหาเส้นทางเพื่อไปให้ถึงปลายทาง โดยจะมีการส่งบรอดคาสท์เฟรมเรียกว่า Route Request (RREQ) เพื่อให้ทุกอุปกรณ์ที่อยู่ภายในของเขตของกำลังสัญญาณไปถึงได้รับเฟรมนี้ โดยทุกอุปกรณ์ที่ได้รับจะตรวจสอบว่า ปลายทางที่มีการร้องขอจากเฟรมนี้ใช่ตัวมันเองหรือไม่ หรือมีเส้นทางที่จะไปถึงปลายทางดังกล่าวอยู่ในตารางเราติ้งของตัวเองหรือไม่ หากปลายทางคือตนเองก็จะส่งกลับไปบอกต้นทาง หรือมีเส้นทางที่จะไปยังปลายทางดังกล่าว ก็จะส่งกลับไปบอกต้นทางว่าสามารถไปถึงอุปกรณ์ปลายทางได้โดยผ่านทางตนผ่านเฟรมที่เรียกว่า Route Reply (RREP) แต่หากไม่มีข้อมูลเกี่ยวกับปลายทาง ก็จะส่งบรอดคาสท์เฟรม (RREQ) ต่อไป เพื่อให้อุปกรณ์อื่นๆ ได้รับ ซึ่งก็จะเป็นลักษณะนี้ไปเรื่อยๆ จนกระทั่งไปถึงปลายทาง ก็จะทำการส่งกลับไปยังต้นทาง โดยผ่านอุปกรณ์ตอนขามากลับไปยังต้นทาง เนื่องจากใน RREP จะมีการระบุเส้นทางลงไปด้วยว่าผ่านโหนดใดมาแล้วบ้างจากต้นทาง ซึ่งต้นทางจะบันทึกเส้นทางดังกล่าวไว้ และทำการส่งข้อมูลไปยังปลายทางโดยผ่านทางเส้นทางที่ได้มา โดยจะเลือกใช้เส้นทางที่มี RREP กลับมาถึงตนเองเร็วที่สุด และ RREP ที่ได้รับหลังจากนี้จะเก็บไว้เป็นเส้นทางสำรอง

DSR ที่ใช้ในการทดลองในโครงงานฉบับนี้ จะใช้เทคโนโลยีที่เรียกว่า Source Routing คือการที่ให้ต้นทางเป็นผู้กำหนดเส้นทางเอง โดยเฟรมที่ส่งออกไปจะมีการแนบเส้นทางไปด้วย เมื่ออุปกรณ์ถัดไปได้รับก็จะทำการอ่านเส้นทางและจะทราบได้ว่า ต้องส่งไปยังอุปกรณ์ใด เป็นอุปกรณ์ถัดไป

DSR เป็นเราติ้งโปรโตคอลพื้นฐานที่มีประสิทธิภาพ ซึ่งสามารถทำงานร่วมกับเครือข่าย IP และ Mobile IP ได้อย่างดีปละมีประสิทธิภาพ รวมถึงการหาเส้นทางได้อย่างรวดเร็ว และตอบสนองได้ทันทีหากมีการเปลี่ยนแปลงเส้นทาง

4.3 การใช้พลังงานบนเครือข่ายไร้สายเฉพาะกิจ

ในระบบเครือข่ายไร้สายเฉพาะกิจ (MANET) นั้น อุปกรณ์ในระบบจะไม่มีสายที่ใช้ในการเชื่อมต่อ ทั้งนี้เพื่อความคล่องตัวและสะดวกในการเพิ่มความสามารถในการเคลื่อนย้าย เนื่องจากไม่ต้องมีข้อจำกัดในการใช้สาย แต่อย่างไรก็ดีอุปกรณ์ไฟฟ้าจำเป็นต้องใช้พลังงานไฟฟ้าในการเป็นพลังงานขับเคลื่อนให้เกิดการทำงานและทำกิจกรรมต่างๆตามหน้าที่ของอุปกรณ์ ซึ่งปกติจะอยู่ใน

รูปแบบของปลั๊กไฟฟ้า แต่อุปกรณ์ไร้สายนั้น ไม่สะดวกที่จะมีปลั๊กเพื่อรับพลังงาน เนื่องจากจะเป็นอุปสรรคในการเคลื่อนที่ จึงมีความจำเป็นต้องใช้พลังงานจากแบตเตอรี่ที่ติดอยู่กับอุปกรณ์

ซึ่งพลังงานจากแบตเตอรี่โดยมากจะหมดไปกับกิจกรรมการรับและส่งข้อมูล เนื่องจากต้องใช้พลังงานไฟฟ้าในการแปลงข้อมูลดิจิตอลให้เป็นคลื่นที่มีกำลังในการส่งผ่านไปยังอากาศ ซึ่งการแปลงนี้จะเกิดขึ้นที่เสาอากาศ ซึ่งจะเป็นในลักษณะตรงกันข้ามในกรณีที่เกิดการรับข้อมูลคือใช้พลังงานในการเปิดเสาอากาศเพื่อดักจับคลื่นในอากาศและแปลงเป็นข้อมูลดิจิตอล ซึ่งจะมีสมการในการหาพลังงานของการส่งข้อมูลมีดังนี้

พลังงานในการส่งข้อมูล = พลังในการขับข้อมูลออกจากเสาต่อหน่วยเวลา * เวลาที่ข้อมูลถูกขับออกจากเสาอากาศ (4.1)

ขณะที่การรับข้อมูลจะมีเป็นดังนี้

พลังงานในการรับข้อมูล = พลังที่เสาใช้ในการรับข้อมูลเข้ามาต่อหน่วยเวลา * เวลาที่ใช้ในการรับข้อมูลเข้ามา (4.2)

ซึ่งจะเห็นได้ว่าพลังงานที่เสียไปจะขึ้นอยู่กับเวลาและพลังที่ใช้ในกิจกรรมต่อหน่วยเวลา ซึ่งพลังนี้โดยมากจะเท่ากันทุกเฟรม แต่เวลาที่ใช้จะขึ้นอยู่กับขนาดของเฟรมที่จะทำการรับและส่ง ยิ่งเฟรมมีขนาดใหญ่ ก็จะยิ่งใช้เวลามาก และยิ่งใช้พลังงานมากตามไปด้วย

ซึ่งด้วยเหตุผลข้างต้น คือการที่อุปกรณ์ไร้สายจำเป็นต้องใช้พลังงานจากแบตเตอรี่ เราจึงควรคำนึงถึงการใช้พลังงาน โดยเฉพาะอย่างยิ่งในเครือข่ายที่มีการใช้เราติ้งโปรโตคอล ที่จะมีการยอมให้อุปกรณ์ไร้สายในระบบนอกจากจะทำการรับและส่งข้อมูลของตัวเองแล้ว ต้องทำหน้าที่เหมือนตัวส่งผ่านข้อมูลให้กับอุปกรณ์อื่นด้วย ซึ่งจะเป็นการเสียพลังงานที่ไม่ใช่จุดประสงค์เพื่อตนเอง

กิจกรรมหลักของอุปกรณ์ไร้สายคือการดักจับและส่งข้อมูล โดยเมื่อดักจับแล้วจึงจะนำพิจารณาว่าจะรับหรือจะลบหรือจะแค่รับรู้เฉยๆ ซึ่งการรับหรือลบนั้น ผู้รับจำเป็นต้องได้รับข้อมูลที่มีกำลังสัญญาณมากพอ จึงจะสามารถนำมาจำแนกได้ ซึ่งถ้าปลายทางในระดับชั้นที่2 คือตัวมันเองก็จะทำการรับข้อมูลนั้นไปประมวลผล แต่ถ้ากำลังมากพอแต่ข้อมูลเสียหายไม่สามารถอ่านได้ก็จะทำการลบทิ้ง ขณะที่หากกำลังสัญญาณไม่มากพอก็จะไม่สามารถแปลความหมายได้ก็จะเป็นการรับรู้เฉยๆ ว่ามีข้อมูลเข้ามาแต่ไม่สามารถจำแนกได้ว่าเป็นข้อมูลใด

โดยในเครือข่ายเฉพาะกิจจะมีระดับการรับสัญญาณอยู่ 2 ระดับ ระดับแรกคือรับและประมวลผลได้เรียก Receive Threshold และระดับที่ 2 คือรับรู้เฉยๆเรียก Carrier Sense Threshold โดยระดับที่ 1 จะน้อยกว่าระดับที่ 2 ซึ่งก็คือหากอุปกรณ์ได้รับสัญญาณมาเกินระดับที่ 2 ก็จะ

สามารถรู้ได้ว่ามีข้อมูลเข้ามา แต่ถ้าไม่เกินระดับที่ 1 ก็จะไม่สามารถตีความหมายได้ คล้ายกับการได้ยินคนพูดแต่เบามากจนไม่ทราบว่าอีกฝ่ายพูดอะไร ซึ่งทั้ง 2 ระดับจะอยู่ในรูปของกำลังสัญญาณ หน่วยเป็นเดซิเบล ซึ่งอาจแปลงเป็นระยะทางจากอุปกรณ์ถึงจุดรับสัญญาณ

4.4 รูปแบบเทรซไฟล์ของเครือข่ายไร้สายเฉพาะกิจ

ในการจำลองระบบเครือข่ายไร้สายเฉพาะกิจ โดยใช้เครื่องมือที่เรียกว่า NS-2 นั้น ผลลัพธ์ของการจำลองจะออกมาในรูปแบบของเทรซไฟล์ ซึ่งเทรซไฟล์จะบอกถึงเหตุการณ์ที่เกิดขึ้นในช่วงเวลาต่างๆ และบอกถึงรายละเอียดของสิ่งที่เกิดขึ้นในการจำลอง เช่น การรับ การส่ง หรือการลบทิ้ง ซึ่งทั้งหมดจะถูกแสดงในเทรซไฟล์ รวมถึงบอกคุณลักษณะของข้อมูลที่เกิดเหตุการณ์นั้นๆ และโหนดที่เกิดเหตุการณ์นั้นๆ

ดังนั้นเราจำจำเป็นต้องเรียนรู้โครงสร้างของเทรซไฟล์เพื่อที่จะสามารถนำมาใช้ในการประมวลผลและวิเคราะห์ได้อย่างถูกต้อง

โดยรูปแบบเทรซไฟล์ของระบบเครือข่ายไร้สายเฉพาะกิจนั้น จะมีเครื่องหมายที่ใช้ในการระบุถึงความหมายของข้อมูลนั้นๆ ซึ่งจะประกอบไปด้วยเครื่องหมาย "-" และตามด้วยอักขระ 2 ตัว โดยตัวแรกจะเป็นตัวพิมพ์ใหญ่ ซึ่งจะมีความหมายดังนี้

- N: คุณลักษณะของโหนด
- I: ข้อมูลในระดับไอพี
- H: ข้อมูลในระดับโหนดต่อโหนด
- M: ข้อมูลในระดับ MAC
- P: ข้อมูลเกี่ยวกับแพ็คเกตนั้นๆ

ซึ่งอักขระตัวที่ 2 จะบอกถึงความหมายย่อยที่อยู่ภายใต้ความหมายหลักอันนั้น ซึ่งจะมีรูปแบบดังนี้

ตารางที่ 4.1 แสดงความหมายเทรซไฟล์ของเครือข่ายไร้สายเฉพาะกิจ

เหตุการณ์	คำย่อ	ฉลาก	ชนิด	ความหมาย
Wireless Event	s: ส่ง r: รับ d: ลบทิ้ง f: ส่งต่อ	-t	double	เวลา (* ของการจำลอง)
		-Ni	int	หมายเลขประจำโหนด
		-Nx	double	พิกัดของโหนด X
		-Ny	double	พิกัดของโหนด Y
		-Nz	double	พิกัดของโหนด Z
		-Ne	double	ระดับพลังงานของโหนด
		-Nl	string	ระดับชั้นของเครือข่าย (AGT, RTR, MAC, etc.)
		-Nw	string	เหตุผลของการดรอป
		-Hs	int	หมายเลขโหนดต้นทาง
		-Hd	int	หมายเลขโหนดปลายทาง, -1, -2
		-Ma	hexadecimal	ระยะทาง
		-Ms	hexadecimal	MAC Address ต้นทาง
		-Md	hexadecimal	MAC Address ปลายทาง
		-Mt	hexadecimal	ชนิดของMAC
		-P	string	ชนิดของแพ็คเกต(arp, dsr, imep, tora, etc.)
		-Pn	string	ชนิดของข้อมูลในแพ็คเกต (cbr, tcp)

หมายเหตุ: ค่าของฟิลด์ –Hd อาจมีทั้งค่า -1 หรือ -2 โดย -1 หมายความว่าเป็นแพ็คเกตที่มีการส่งแบบบรอดคาสท์และ -2 หมายความว่าเป็นแพ็คเกตที่รับและส่งระหว่างระดับชั้นภายในตัวโหนดเอง

ซึ่งข้อมูลจากตารางจะแสดงให้เห็นถึงฟิลด์หลักๆ ที่จะมีอยู่ในทุกแพ็คเกต และตารางต่อไปนี้จะแสดงถึงเทรซไฟล์ที่จะเกิดขึ้นตามแต่ชนิดของแพ็คเกต

ตารางที่ 4.2 แสดงความหมายของเทรซไฟล์แต่ละชนิดแพ็คเกต

เหตุการณ์	ฉลาก	ชนิด	ความหมาย
ARP	-Po	string	ชนิดของ arp คือเป็น Request หรือ Reply
	-Pms	int	MAC Address ต้นทาง
	-Ps	int	ที่อยู่ต้นทาง
	-Pmd	int	MAC Address ปลายทาง
	-Pd	int	ที่อยู่ปลายทาง
DSR	-Ph	int	จำนวนของโหนดที่ทำการส่งผ่าน
	-Pq	int	ฉลากของ Routing Request
	-Ps	int	เลขลำดับของ Route Request
	-Pp	int	ฉลากของ Routing Reply
	-Pn	int	เลขลำดับของ Route Request
	-Pl	int	ขนาดของ Reply
	-Pe	int->int	ต้นทาง->ปลายทาง ของ Source Routing
	-Pw	int	ฉลากของ Error Report
	-Pm	int	จำนวนของการผิดพลาด
	-Pc	int	แจ้งถึงใคร
	-Pb	int->int	การผิดพลาดจากใครถึงใคร
IP	-Is	int.int	ที่อยู่ต้นทางและพอร์ท
	-Id	int.int	ที่อยู่ปลายทางและพอร์ท
	-It	string	ชนิดของแพ็คเกต
	-Il	int	ขนาดของแพ็คเกต
	-If	int	หมายเลขของการไหลของข้อมูล
	-Ii	int	หมายเลขประจำแพ็คเกต
	-Iv	int	ค่า TTL

TCP	-Ps	int	เลขลำดับ
	-Pa	int	หมายเลขของ Acknowledgment
	-Pf	int	จำนวนครั้งที่ได้ทำการส่ง
	-Po	int	จำนวนที่ดีที่สุดของการส่งข้อมูล
CBR	-Pi	int	หมายเลขลำดับ
	-Pf	int	จำนวนครั้งที่ได้ทำการส่ง
	-Po	int	จำนวนที่ดีที่สุดของการส่งข้อมูล

ตัวอย่างเทรซไฟล์ของเครือข่ายเฉพาะกิจ

ในส่วนนี้จะยกตัวอย่างเทรซไฟล์พร้อมถึงอธิบายความหมายของเหตุการณ์นั้นๆ ตามพารามิเตอร์ที่ระบุ โดยจะอธิบายเฉพาะพารามิเตอร์ที่จะนำมาพิจารณาในโครงงานนี้เท่านั้น

```
s -t 35.871425732 -Hs 0 -Hd -1 -Ni 0 -Nx 592.07 -Ny 294.00 -Nz 0.00 -
Ne 99.999597 -Nl MAC -Nw --- -Ma 0 -Md ffffffff -Ms 0 -Mt 800 -Is
0.255 -Id 2.255 -It DSR -Il 84 -If 0 -Ii 2 -Iv 32 -P dsr -Ph 1 -Pq 1
-Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->16 -Pw 0 -Pm 0 -Pc 0 -Pb 0->0
```

โดยบรรทัดดังกล่าวหมายถึงเป็นเหตุการณ์การส่งข้อมูล (s) ของโหนด 0 (-Ni) ที่วินาทีที่ 35.871425732 (-t) โดยโหนดต้นทางคือโหนด 0 (-Hs) และปลายทางเป็นแบบบรอดคาสท์ (-Hd) โดยเป็นการส่งในระดับชั้น (-Nl) MAC และพลังงานคงเหลือของโหนดนี้คือ 99.999597 (-Ne) ซึ่งเป็นแพคเก็ทชนิด DSR (-P) มีหมายเลขการไหลข้อมูลคือ 0 (-If) โดยต้นทางในระดับชั้น MAC คือ 0 (-Ms) และปลายทางเป็นบรอดคาสท์ (-Md) โดยจุดที่ต้องพิจารณาเวลาเหตุการณ์ s คือพลังงานคงเหลือที่ได้จากเหตุการณ์นี้จะเป็นพลังงานคงเหลือก่อนที่จะทำการส่งข้อมูลออกไป

```
N -t 35.871426 -n 3 -e 99.999597
```

บรรทัดนี้แสดงถึงเหตุการณ์ว่ามีการเปลี่ยนแปลงพลังงานเกิดขึ้น โดยเปลี่ยนแปลงไปเป็น 99.99597 (-e) เนื่องจากมีการดักจับข้อมูล (N) ของโหนด 3 (-n) วินาทีที่ 35.871426 (-t) แต่ยังไม่สามารถระบุได้ว่าดักจับแล้วได้ความหมายว่าอย่างไรและจะกระทำการใดต่อ

```
r -t 35.880127177 -Hs 2 -Hd -2 -Ni 2 -Nx 339.46 -Ny 476.63 -Nz 0.00 -
Ne 99.998253 -Nl MAC -Nw --- -Ma 13a -Md 2 -Ms 3 -Mt 806 -P arp -Po
REPLY -Pms 3 -Ps 3 -Pmd 2 -Pd 2
```

สำหรับเหตุการณ์รับข้อมูล r นั้นจะมีลักษณะคล้ายกับเหตุการณ์ s โดยเหตุการณ์นี้หมายถึงเป็นการรับข้อมูล (r) ของโหนด 2 (-Ni) ในระดับ MAC (-Nl) วินาทีที่ 35.880127177 (-t) ซึ่งเป็นข้อมูลชนิด ARP (-P) โดยพลังงานคงเหลือคือ 99.998253 (-Ne) โดยต้นทางในระดับชั้น MAC

```
d -t 487.168867460 -Hs 7 -Hd 6 -Ni 7 -Nx 518.35 -Ny 732.51 -Nz 0.00 -
Ne 74.497180 -Nl MAC -Nw RET -Ma 13a -Md 6 -Ms 7 -Mt 800 -Is 5.0 -Id
6.0 -It cbr -Il 532 -If 1002 -Ii 582727 -Iv 28 -Pn cbr -Pi 1433 -Pf 4
-Po 16777215
```

เหตุการณ์นี้หมายถึงเป็นเหตุการณ์การลบทิ้งของข้อมูล (d) ของโหนด 7 (-Ni) ที่วินาทีที่ 487.168867460 (-t) โดยโหนดต้นทางคือโหนด 7 (-Hs) และปลายทางเป็นคือโหนด 6 (-Hd) โดยเป็นการกระทำในระดับชั้น (-Nl) MAC และพลังงานคงเหลือของโหนดนี้คือ 74.497180 (-Ne) ซึ่งเป็นแพคเก็ทชนิด CBR (-P) มีหมายเลขการไหลข้อมูลคือ 1002 (-If) โดยต้นทางในระดับชั้น MAC คือ 7 (-Ms) และปลายทางคือโหนด 6 (-Md)

4.5 รูปแบบการเคลื่อนที่ของอุปกรณ์ใน MANET

ในเครือข่ายเฉพาะกิจ (MANET) นั้น ทุกอุปกรณ์ในเครือข่ายจะมีความสามารถในการเคลื่อนที่ (Mobility) ซึ่งในการจำลองนั้น จะต้องมีการกำหนดลักษณะการเคลื่อนที่ของอุปกรณ์ดังกล่าวด้วย โดยได้มีการกำหนดรูปแบบการเคลื่อนที่ออกมาหลายรูปแบบ เช่น

- Random Waypoint เป็นรูปแบบที่ใช้กันอย่างแพร่หลายมากที่สุด เนื่องจากครอบคลุมแทบทุกความเป็นไปได้ในการเคลื่อนที่ หากแต่ว่าในความเป็นจรองไม่อาจเคลื่อนที่ได้ดังในการจำลอง โดยจะใช้การเคลื่อนที่โดยใช้หลักการสุ่ม โดยจะต้องกำหนดพารามิเตอร์จำนวน 3 ตัวด้วยกันคือเวลาหน่วงในการหยุด, ความเร็วสูงสุด และความเร็วต่ำสุด โดยเมื่อเริ่มจะทำการสุ่มเพื่อกำหนดพิกัดเริ่มต้นของแต่ละโหนด จากนั้นทุกโหนดจะอยู่ในตำแหน่งเดิมด้วยเวลาหน่วงที่กำหนดไว้ จากนั้นจะทำการสุ่มเพื่อหาพิกัดของตำแหน่งใหม่ที่จะเคลื่อนไป และจะเคลื่อนด้วยความเร็วที่ได้จากการสุ่มระหว่างความเร็วสูงสุดและต่ำสุด และจะเป็นไปในลักษณะนี้เรื่อยๆ
- Manhattan Grid เป็นรูปแบบการเคลื่อนที่ที่จำลองลักษณะการเคลื่อนที่ในเมืองที่มีถนนและอาคาร โดยจะทำการสุ่มจุดเริ่มต้นมาที่ถนนซักแห่งหนึ่งและจะทำการเคลื่อนที่ไปยังปลายทางที่ได้จากการสุ่ม ซึ่งจะเป็นนิศทางแนวตั้งหรือแนวนอนเท่านั้น ด้วยความเร็วจากการสุ่ม และจะหยุดอยู่ในตำแหน่งเดิมตามเวลาที่กำหนด ก่อนที่จะเคลื่อนตัวอีกครั้งหนึ่ง

บทที่ 5
การศึกษาพฤติกรรมการใช้พลังงาน ของอุปกรณ์บนเครือข่ายเฉพาะกิจ

5.1 บทนำ

ในเครือข่ายเฉพาะกิจ (MANET) นั้น จะเป็นกลุ่มของอุปกรณ์ไร้สาย ที่ทำการติดต่อสื่อสารกันโดยไม่จำเป็นต้องใช้สถานีแม่หรือที่เรียกว่า Access Point ซึ่งจะเป็นการติดต่อสื่อสารระหว่างอุปกรณ์โดยตรง ซึ่งจะเกิดการสื่อสารลักษณะนี้ก็ต่อเมื่อ ทั้ง 2 อุปกรณ์อยู่ในพื้นที่ของสัญญาณกันและกัน หรือหากอยู่นอกพื้นที่ก็จำเป็นต้องมีการใช้การส่งผ่านอุปกรณ์อื่นเพื่อไปให้ถึงปลายทาง ซึ่งจะเกิดกรณีนี้ได้จำเป็นต้องมีเราติ้งโปรโตคอลสำหรับเครือข่ายเฉพาะกิจนี้ ซึ่งจะเป็นการส่งข้อมูลจากต้นทางถึงปลายทางที่อยู่นอกพื้นที่ของสัญญาณกันและกันโดยผ่านอุปกรณ์ระหว่างทาง ซึ่งอุปกรณ์ระหว่างทางจะทำหน้าที่ในการส่งผ่านข้อมูลเพื่อให้ไปถึงปลายทาง ซึ่งจะทำหน้าที่คล้ายกับอุปกรณ์เราเตอร์ เนื่องจากมีหน้าที่ส่งผ่านข้อมูลและจะมีกลไกในการหาเส้นทางจากต้นทางไปยังปลายทางเช่นเดียวกัน

แต่อย่างไรก็ดี ในระบบเครือข่ายไร้สายเฉพาะกิจนั้น แต่ละโหนดอาจเป็นได้ทั้งผู้ส่งข้อมูล ผู้รับ หรือเป็นตัวกลางในการส่งผ่านข้อมูล ซึ่งแต่ละอุปกรณ์จะไม่สามารถหลีกเลี่ยงหน้าที่เหล่านี้ได้ หรือไม่สามารถปฏิเสธการเป็นตัวกลาง หรือหากทำได้ก็นับว่าไม่สมควรเนื่องจาก จะแต่ละอุปกรณ์จำเป็นต้องใช้ผู้อื่นเป็นตัวกลาง ซึ่งอุปกรณ์นั้นก็ควรยินยอมที่ให้ตนเองเป็นตัวกลางของผู้อื่นด้วยเช่นเดียวกัน ซึ่งอาจมองในแง่ของความยุติธรรม แต่ความยุติธรรมนี้สามารถนำมาพิจารณาได้อีกว่าผลประโยชน์ที่เราได้รับจากการกระทำของผู้อื่นนั้นเหมาะสมหรือคุ้มค่ากับสิ่งที่เรากระทำเพื่อผู้อื่นหรือไม่ หรือเมื่อเราทำให้ผู้อื่นแล้วเราสามารถทำงานให้ตนเองได้เต็มประสิทธิภาพและตามความต้องการหรือไม่

ซึ่งในระบบเครือข่ายนั้น การวัดประสิทธิภาพโดยมากจะเป็นการสัดประสิทธิภาพในแง่ของการส่งผ่านข้อมูล เช่น เวลาหน่วง, แบนด์วิธ หรือทรูพุธ แต่สิ่งหนึ่งที่อุปกรณ์ไร้สายต่างจากอุปกรณ์อื่นๆในระบบเครือข่ายและเป็นสิ่งหนึ่งที่ควรนำเอามาเป็นปัจจัยในการพิจารณาก็คือพลังงาน

ซึ่งปัจจัยที่เราจะนำมาพิจารณาเรื่องปริมาณการทำงานนั้นเราจะใช้พลังงาน ในการพิจารณาเนื่องจากอุปกรณ์ไร้สายเป็นเครื่องใช้ไฟฟ้า ซึ่งต้องการพลังงานไฟฟ้าในการหล่อเลี้ยงและเป็นพลังขับเคลื่อนให้เกิดการทำงาน แต่อุปกรณ์ไร้สายนั้นไม่สะดวกในการมีปลั๊กไฟฟ้าเพื่อรับพลังงาน จึงจำเป็นต้องอาศัยพลังงานจากแบตเตอรี่ ที่ติดมากับตัวเครื่อง ซึ่งอุปกรณ์ไร้สายนั้น จะพยายามลดขนาดของอุปกรณ์ให้เล็กและใช้พลังงานแต่ละกิจกรรมให้น้อยที่สุด เพื่อเป็นการประหยัดพลังงาน

โดยอุปกรณ์ไร้สายนั้น กิจกรรมหลักคือการรับส่งข้อมูล ซึ่งทั้ง 2 กิจกรรมนั้นจำเป็นต้องใช้พลังงานทั้งสิ้น ไม่ว่าจะเป็นการรับส่งข้อมูลเพื่อตัวเองหรือการรับส่งข้อมูลเพื่อผู้อื่นก็ตาม ดังนั้นจึงจำเป็นที่จะต้องตระหนักและพิจารณาการใช้พลังงานของอุปกรณ์ไร้สาย

โครงงานฉบับนี้จะนำเสนอการทดลองเพื่อหารูปแบบการสูญเสียพลังงานดังกล่าว โดยจะทดลองภายใต้สภาพแวดล้อมต่างๆและนำผลลัพธ์ซึ่งเป็นข้อมูลดิบมาวิเคราะห์และประมวลผล โดยการทดลองจะมีพารามิเตอร์ที่เปลี่ยนแปลงไป โดยจะพยายามให้ครอบคลุมทุกกรณีและทุกความเป็นไปได้ ซึ่งการทดลองนั้นจะกำหนดรูปแบบและเงื่อนไขต่างๆโดยใช้ TCL สคริปต์

และในโครงงานนี้จะนำเสนอวิธีการและแนวคิดรวมถึงหลักการในการวิเคราะห์พฤติกรรมและการสูญเสียพลังงานของอุปกรณ์ในเครือข่ายเฉพาะกิจ รวมถึงการให้ได้มาซึ่งรูปแบบของการสูญเสียพลังงานของอุปกรณ์โดยขึ้นกับปัจจัยต่างๆซึ่งเป็นสภาพแวดล้อม โดยในส่วนนี้จะอธิบายถึงภาพรวมและขั้นตอนการทำงาน รวมถึงแนวคิดที่ใช้และทฤษฎีที่จำเป็นและเกี่ยวข้อง

5.2 ขั้นตอนการดำเนินงาน

ใน Network Simulator 2 (NS-2) นั้น จะใช้ TCL สคริปท์ เป็นตัวกำหนดรูปแบบและเงื่อนไขในการจำลอง เช่น จำนวนโหนด รูปแบบการเชื่อมต่อ รวมถึงรูปแบบของข้อมูลที่ทำการแลกเปลี่ยน ซึ่ง 1 สคริปท์ก็ถือว่าเป็น 1 การจำลอง ซึ่งเมื่อนำไปประมวลผลด้วย NS-2 จะได้ผลลัพธ์ออกมาอยู่ในรูปของเทรซไฟล์ ซึ่งเป็นไฟล์ที่บันทึกเหตุการณ์ต่างๆที่เกิดขึ้นในการจำลองรวมถึงรายละเอียดของเหตุการณ์นั้นๆ ซึ่งข้อมูลต่างๆที่ได้จากเทรซไฟล์ จะเป็นข้อมูลดิบที่ยังไม่ผ่านการประมวลผลและอาจไม่สามารถนำมาใช้ในการวิเคราะห์ให้เกิดประโยชน์ได้ จึงจำเป็นต้องใช้โปรแกรมภาษา PERL ในการแยกและคัดกรองข้อมูล รวมถึงตีความข้อมูลที่ได้ และคำนวณค่าต่างๆ เพื่อให้ได้ข้อมูลที่เกิดประโยชน์และสามารถนำไปใช้ต่อ

ในการจำลองจำเป็นต้องมีการกำหนดรูปแบบการเชื่อมต่อ ซึ่งประกอบด้วย โหนดต้นทางและโหนดปลายทางของการเชื่อมต่อนั้น, เวลาที่เริ่มการเชื่อมต่อ, ปริมาณข้อมูลที่ส่ง และลักษณะข้อมูลที่ส่ง ซึ่งจะเรียกข้อกำหนดและเงื่อนไขต่างๆเหล่านี้ว่า "รูปแบบการเชื่อมต่อ" ซึ่งในการจำลองจำเป็นต้องมีการสร้างรูปแบบนี้ขึ้นเพื่อกำหนดพฤติกรรมและการทำงานของแต่ละโหนดในระบบ

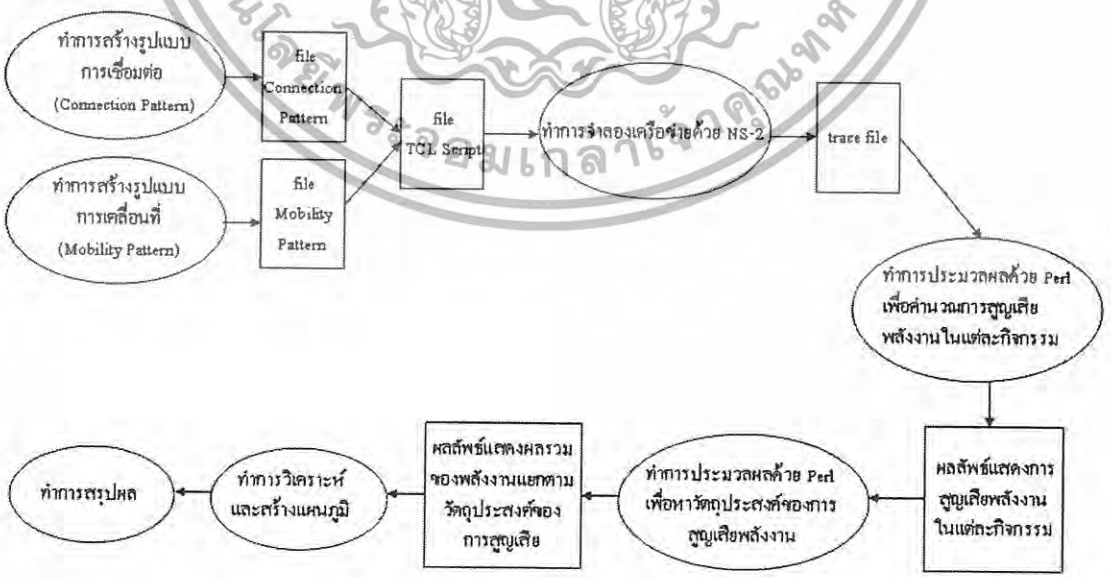
และในเครือข่ายเฉพาะกิจ ซึ่งเป็นเครือข่ายไร้สายนั้น จะมีความสามารถในการเคลื่อนที่ (Mobility) ดังนั้นเพื่อที่จะให้มีคุณสมบัติคล้ายระบบจริงจึงจะมีการกำหนดรูปแบบการเคลื่อนที่ที่เรียกว่า Scenario ด้วย ซึ่งจะเป็นการกำหนดพิกัดเริ่มต้นของแต่ละโหนด รวมถึงเวลาและการย้ายตำแหน่งของแต่ละโหนดตลอดทั้งการจำลอง

ซึ่งรูปแบบทั้ง 2 นั้น เป็นสิ่งที่จำเป็นต้องมีในการจำลองและใน TCL สคริปท์จำเป็นต้องมีการกำหนดรูปแบบทั้ง 2 นี้ด้วย ซึ่งใน NS-2 ได้มีคำสั่งในการสร้าง รูปแบบการเชื่อมต่อ และ scenario นี้ไว้แล้ว โดยสิ่งที่ผู้ใช้ต้องทำคือการใช้คำสั่ง และกำหนด parameter ให้กับคำสั่งนั้น

จากนั้นจะทำการนำรูปแบบทั้ง 2 มาเป็นองค์ประกอบในการจำลอง และทำการสร้าง TCL สคิปท์เพื่อการจำลอง ซึ่งจะทำการจำลองด้วย Network Simulator 2 (NS-2) แล้วจะได้ผลลัพธ์จากการจำลองออกมาอยู่ในรูปของเทรซไฟล์

เทรซไฟล์ที่ได้จากการจำลองนั้นเปรียบเสมือนข้อมูลดิบซึ่งเป็นบันทึกที่บอกถึงเหตุการณ์และรายละเอียดทุกอย่างที่เกิดในในการจำลอง ซึ่งเทรซไฟล์ของเครือข่ายเฉพาะกิจนั้น ค่อนข้างซับซ้อน มีปริมาณมาก และข้อมูลที่แสดงไม่สามารถนำไปทำให้เกิดประโยชน์ในการนำไปใช้งานได้ จึงจำเป็นต้องเขียนโปรแกรมภาษา PERL เพื่อทำการคัดแยกข้อมูล และทำการจำแนกและวิเคราะห์ข้อมูลเพื่อให้ได้ข้อมูลตามที่ต้องการ ซึ่งผลลัพธ์จะเป็นข้อมูลที่มีประโยชน์ และสามารถนำไปวิเคราะต่อและแสดงผลในรูปของแผนภูมิได้ โดยอาจสรุปขั้นตอนของการทำการทดลองได้ดังนี้

1.) ทำการสร้างรูปแบบการเชื่อมต่อ
2.) ทำการสร้างรูปแบบการเคลื่อนที่
3.) นำผลลัพธ์จาก 1.) และ 2.) มาใช้เป็นองค์ประกอบในการสร้าง TCL สคริปท์
4.) นำผลลัพธ์จาก 3.) มาทำการจำลองด้วย NS-2
5.) นำผลลัพธ์จาก 4.) ซึ่งอยู่ในรูปของเทรซไฟล์มาทำการประมวลผลด้วย Perl เพื่อคำนวณการสูญเสียพลังงานในแต่ละกิจกรรม
6.) นำผลลัพธ์จาก 5.) มาทำการประมวลผลด้วย Perl เพื่อหาวัตถุประสงค์ของการสูญเสียพลังงาน
7.) นำผลลัพธ์จาก 6.) มาทำการวิเคราะห์และสร้างแผนภูมิ
8.) นำผลลัพธ์จาก 6.) และ 7.) มาทำการสรุปผล



รูปที่ 5.1 แสดงขั้นตอนของการทำการทดลอง

5.3 การสร้างรูปแบบการเชื่อมต่อ

ใน NS-2 ได้กำหนดรูปแบบคำสั่งในการสร้างรูปแบบการเชื่อมต่อมีดังนี้

```
ns cbrgen.TCL [-type cbr|tcp] [-nn nodes] [-seed seed] [-mc connections] [-rate rate]
```

โดย - cbr|tcp คือรูปแบบของทราฟฟิกที่ต้องการสร้าง
- nodes คือจำนวนโหนดทั้งหมดในเครือข่าย
- seed คือค่าที่เอาไว้ใช้ในการแรนดอม
- connections คือจำนวนการเชื่อมต่อ
- rate คือจำนวนแพ็คเกตที่ส่งใน 1 วินาที

ซึ่งไฟล์ cbrgen.TCL นั้น จะอยู่ในไดเร็กทอรี ~ns/indep-utils/cmu-scen-gen ซึ่งเป็นไฟล์ที่มีเพื่อการสร้างการเชื่อมต่อของ TCP หรือ CBR โดยเฉพาะ ตัวอย่างคำสั่งในการสร้างการเชื่อมต่อเช่น

```
ns cbrgen.tcl -type cbr -nn 10 -seed 97 -mc 2 -rate 4 > outputtest
```

ซึ่งหมายถึงสร้างการเชื่อมต่อที่เป็น CBR จำนวน 2 ครั้ง โดยส่งที่อัตรา 4 แพ็คเกตต่อวินาที โดยใช้ค่า 97 ในการสุ่มเพื่อหาคู่ของการเชื่อมต่อจากโหนดจำนวน 10 โหนด ที่เวลาที่ได้จากการสุ่ม ซึ่งจะเก็บผลลัพธ์เป็นรูปแบบการเชื่อมต่อไว้ในไฟล์ที่ชื่อ outputtest ซึ่งจะได้ผลลัพธ์ดังนี้

ตารางที่ 5.1 แสดงรูปแบบการเชื่อมต่อ

```
#
# nodes: 10, max conn: 2, send rate: 0.25, seed: 97
#
#
# 0 connecting to 1 at time 67.94694506933304
#
set udp_(0) [new Agent/UDP]
$udp_(0) set class_ 1000
$ns_ attach-agent $node_(0) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(1) $null_(0)
```

ตารางที่ 5.1 แสดงรูปแบบการเชื่อมต่อ (ต่อ)

```
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 67.94694506933304 "$cbr_(0) start"
#
# 0 connecting to 2 at time 98.553513548594665
#
set udp_(1) [new Agent/UDP]
$udp_(1) set class_ 1001
$ns_ attach-agent $node_(0) $udp_(1)
set null_(1) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(1)
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 512
$cbr_(1) set interval_ 0.25
$cbr_(1) set random_ 1
$cbr_(1) set maxpkts_ 10000
$cbr_(1) attach-agent $udp_(1)
$ns_ connect $udp_(1) $null_(1)
$ns_ at 98.553513548594665 "$cbr_(1) start"
#
```

ซึ่งจากผลลัพธ์จะเห็นได้ว่ามีการสร้างการเชื่อมต่อจำนวน 2 ครั้ง คือ 1 และ 0 โดยครั้งแรกให้โหนด 0 ส่งข้อมูลหาโหนด 1 ที่วินาทีที่ 67.94694506933304 และครั้งที่ 2 ให้โหนด 0 ส่งหาโหนด 2 ที่เวลา 98.553513548594665 โดยกำหนดให้หมายเลขของการเชื่อมต่อเป็น 1000 และ 1001 ตามลำดับ และมีค่าเวลาหน่วงระหว่างแพ็คเกตคือ 0.25 วินาที ซึ่งเป็นส่วนกลับของอัตราส่วน 4 แพ็คเกตต่อวินาที

5.4 การสร้างรูปแบบการเคลื่อนที่

รูปแบบการเคลื่อนที่หรือเรียกอีกชื่อหนึ่งว่า scenario จะเป็นการกำหนดตำแหน่งและการเคลื่อนที่ของแต่ละโหนด ซึ่งจะกำหนดในรูปแบบของ พิกัด x และ y โดยจะมีคำสั่งในการสร้าง scenario ดังนี้

```
./setdest [-v version] [-n nodes] [-p pausetime] [-s maxspeed] [-t simtime] [-x maxx] [-y maxy] >
[outdir/movement-file]
```

โดย

- version คือเวอร์ชั่นของ scenario โดยกำหนดเป็น 1 สำหรับเวอร์ชั่น original 1999 CMU หรือกำหนดเป็น 2 สำหรับเวอร์ชั่น modified 2003 U.Michigan
- nodes คือจำนวนโหนดทั้งหมดในเครือข่าย
- pausetime คือเวลาหน่วงในการที่โหนดจะอยู่ในตำแหน่งเดิม
- maxspeed คือความเร็วสูงสุดที่โหนดจะเคลื่อนที่ได้
- simtime คือเวลาที่ใช้ในการจำลอง
- maxx คือขนาดของพื้นที่ในแนวแกน x
- maxy คือขนาดของพื้นที่ในแนวแกน y
- outdir/movement-file คือ ไฟล์ที่จะใช้เก็บเป็นผลลัพธ์

ซึ่งเราต้องทำการคอมไพล์ไฟล์โดยเข้าไปที่ indep-utils/cmu-scen-gen/setdest ก่อน ซึ่งใช้คำสั่ง make จากนั้นจึงจะสามารถใช้คำสั่ง setdest ได้ ยกตัวอย่างเช่น

```
./setdest -n 20 -p 2.0 -s 10.0 -t 1000 -x 500 -y 500 > scen-20-test
```

หมายถึงทำการสร้าง scenario โดยกำหนดให้มีจำนวนโหนดทั้งหมด 20 โหนด มีเวลาหน่วงในการที่แต่ละโหนดจะอยู่ในตำแหน่งเดิมคือ 2.0 วินาที ความเร็วสูงสุดที่โหนดจะใช้ได้ในการเคลื่อนที่คือ 10 เมตรต่อวินาที โดยใช้เวลาในการจำลองทั้งหมดคือ 1000 วินาที โดยมีพื้นที่ในการที่แต่ละโหนดจะเคลื่อนที่ไปได้คือ 500*500 เมตร โดยเก็บผลลัพธ์ที่เป็น scenario ไว้ที่ไฟล์ scen-20-test

ซึ่งผลลัพธ์จะเป็นการกำหนด พิกัดเริ่มต้นของแต่ละโหนด และจะระบุเวลาที่โหนดจะเคลื่อนที่ไปยังตำแหน่งใหม่ และพิกัดของตำแหน่งใหม่ ซึ่งพิกัดนั้นจะเกิดจากค่าสุ่มจากขอบเขตที่กำหนดให้ในรูปแบบของพิกัด x และ y ที่มากที่สุด และเวลาจะเกิดจากค่าหน่วงในการที่ให้โหนดหยุดอยู่กับที่

และความเร็วในการเคลื่อนที่ที่ได้จากค่าสุ่มจากการกำหนดค่าความเร็วสูงสุด ซึ่งรูปแบบผลลัพธ์จะเป็นดังนี้

```
$node_(0) set X_ 850.810197174671
$node_(0) set Y_ 347.203153236363
```

ซึ่งเป็นการกำหนดพิกัดเริ่มต้นของโหนด 0 คือตำแหน่งที่ (850.810197174671, 347.203153236363)

```
$ns_ at 1.000000000000 "$node_(0) setdest 392.148063417244 57.345610327992 1.793613553297"
```

ซึ่งเป็นการกำหนดให้เมื่อวินาทีที่ 1.0 โหนด 0 เคลื่อนที่ไปยังพิกัด (392.148063417244, 57.345610327992) ด้วยความเร็ว 1.793613553297 เมตรต่อวินาที ซึ่งสังเกตได้ว่า เริ่มเคลื่อนเมื่อวินาทีที่ 1.0 เนื่องจาก ได้กำหนดเวลาหน่วงในการโหนดอยู่ที่ตำแหน่งเดิมเป็นเวลา 1 วินาที

```
$ns_ at 303.504215693585 "$node_(0) setdest 392.148063417244 57.345610327992 0.000000000000"
```

โดยสังเกตที่บรรทัดนี้คือความเร็วจะเป็น 0.0000 นั่นหมายถึงเป็นเวลาที่โหนด 0 เคลื่อนที่จากจุดเดิมมาอยู่ที่ตำแหน่งใหม่พอดีในวินาทีที่ 303.504215693585 ซึ่งจะเกิดจากระยะห่างจากจุดเดิมและความเร็วในการเคลื่อนที่

```
$ns_ at 304.504215693585 "$node_(0) setdest 467.263611077990 149.622951895365 6.586556485855"
```

โดยเมื่อโหนด 0 เคลื่อนที่มายังตำแหน่งใหม่ก็จะอยู่ในตำแหน่งเดิม 1 วินาที แล้วจะเริ่มเคลื่อนที่ด้วยความเร็วและพิกัดจากการสุ่มอีกครั้ง

ซึ่งการเคลื่อนที่ของแต่ละโหนดจะเป็นไปในลักษณะนี้เรื่อยๆ ซึ่งอัลกอริธึมนี้เรียกว่า Random Waypoint ดังที่ได้กล่าวไปแล้วในบทก่อนหน้า

5.5 การจำลองเครือข่ายเฉพาะกิจด้วย TCL สคริปท์

หลังจากได้รูปแบบทั้ง 2 คือ รูปแบบการเชื่อมต่อ และรูปแบบการเคลื่อนที่แล้ว ต่อจากนี้เราจะนำรูปแบบทั้ง 2 มาเป็นองค์ประกอบของการจำลอง ซึ่งการจำลองจะใช้ TCL สคริปท์ ในการกำหนดสภาพแวดล้อมและเงื่อนไขของการทดลอง ซึ่งการจำลองเครือข่ายเฉพาะกิจหรือ MANET นั้น จำเป็นต้องมีการระบุ parameter ที่จำเป็นสำหรับเครือข่ายเฉพาะกิจที่เป็นเครือข่ายไร้สายนี้ เช่น มาตรฐานของระดับ MAC หรือชนิดของสัญญาณและเสาอากาศเป็นต้น ซึ่งในการทดลองสามารถใช้สคริปท์ด้านล่างนี้ได้เลย เพียงแต่ทำการแก้ไข parameter ต่างๆ ในส่วนของ "Define Option"

ตารางที่ 5.2 สคริปท์จำลองเครือข่ายเฉพาะกิจ

```
# Modified for new node structure
#
# ======================================================================
# Define options
#
# ======================================================================
set opt(chan)           Channel/WirelessChannel
set opt(prop)           Propagation/TwoRayGround
set opt(netif)          Phy/WirelessPhy
set opt(mac)            Mac/802_11
set opt(ifq)            CMUPriQueue
set opt(ll)         LL
set opt(ant)            Antenna/OmniAntenna

set opt(x)          1000  ;# X dimension of the topography
set opt(y)          1000      ;# Y dimension of the topography
set opt(cp)         "cbr10-5"
set opt(sc)         "mobility1000x1000-10"

set opt(ifqlen)         50            ;# max packet in ifq
set opt(nn)         10        ;# number of nodes
set opt(seed)           0.0
set opt(stop)           1000.0              ;# simulation time
set opt(tr)       result_DSR_1000x1000_10_cbr_5.tr;# trace file
set opt(rp)             DSR           ;# routing protocol script
#set opt(lm)             "off"         ;# log movement
set opt(agent)          Agent/DSDV
set opt(energymodel)    EnergyModel     ;
set opt(initialenergy)  100             ;# Initial energy in Joules
#set opt(logenergy)      "on"                ;# log energy every 150
seconds
#
# ======================================================================
# needs to be fixed later
#set AgentTrace                 ON
#set RouterTrace                ON
#set MacTrace                   ON

LL set mindelay_        50us
LL set delay_                   25us
LL set bandwidth_       0       ;# not used

Agent/Null set sport_           0
```

ตารางที่ 5.2 สคริปท์จำลองเครือข่ายเฉพาะกิจ (ต่อ)

```
Agent/Null set dport_           0

Agent/CBR set sport_            0
Agent/CBR set dport_            0

Agent/TCPSink set sport_        0
Agent/TCPSink set dport_        0

Agent/TCP set sport_            0
Agent/TCP set dport_            0
Agent/TCP set packetSize_       1460

Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above
it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

#
# ======================================================================
proc usage { argv0 } {
      puts "Usage: $argv0"
      puts "\tmandatory arguments:"
      puts "\t\t\[-x MAXX\] \[-y MAXY\]"
      puts "\toptional arguments:"
      puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
      puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\]\n"
}

proc getopt {argc argv} {
      global opt
      lappend optlist cp nn seed sc stop tr x y

      for {set i 0} {$i < $argc} {incr i} {
            set arg [lindex $argv $i]
            if {[string range $arg 0 0] != "-"} continue

            set name [string range $arg 1 end]
            set opt($name) [lindex $argv [expr $i+1]]
      }
}
#
# ======================================================================
# Main Program
```

ตารางที่ 5.2 สคริปท์จำลองเครือข่ายเฉพาะกิจ (ต่อ)

```
#
# ======================================================================
getopt $argc $argv
#source ../lib/ns-bsnode.TCL
#source ../mobility/com.TCL
# do the get opt again incase the routing protocol file added some
more
# options to look for
getopt $argc $argv

if { $opt(x) == 0 || $opt(y) == 0 } {
      usage $argv0
      exit 1
}
if {$opt(seed) > 0} {
      puts "Seeding Random number generator with $opt(seed)\n"
      ns-random $opt(seed)
}
#
# Initialize Global Variables
#
set ns_           [new Simulator]
set topo    [new Topography]

set tracefd [open $opt(tr) w]

$topo load_flatgrid $opt(x) $opt(y)

$ns_ use-newtrace
$ns_ trace-all $tracefd
#
# Create God
#
create-god $opt(nn)

#
#  Create the specified number of nodes $opt(nn) and "attach" them
#  the channel.
#  Each routing protocol script is expected to have defined a proc
#   create-mobile-node that builds a mobile node and inserts it into
the
#  array global $node_($i)
#
      #global node setting

        $ns_ node-config -adhocRouting $opt(rp) \
                  -llType $opt(ll) \
                  -macType $opt(mac) \
                  -ifqType $opt(ifq) \
                  -ifqLen $opt(ifqlen) \
                  -antType $opt(ant) \
                  -propType $opt(prop) \
                  -phyType $opt(netif) \
                  -channelType $opt(chan) \
                  -topoInstance $topo \
                  -energyModel $opt(energymodel) \
                  -rxPower 0.3 \
                  -txPower 0.6 \
                  -initialEnergy $opt(initialenergy) \
```

ตารางที่ 5.2 สคริปท์จำลองเครือข่ายเฉพาะกิจ (ต่อ)

```
                    -agentTrace ON \
                    -routerTrace ON \
                    -macTrace ON
#                   -movementTrace OFF

      for {set i 0} {$i < $opt(nn) } {incr i} {
            set node_($i) [$ns_ node]
            $node_($i) random-motion 0          ;#   disable   random
motion
      }
#
# Source the Connection and Movement scripts
#
if { $opt(cp) == "" } {
      puts "*** NOTE: no connection pattern specified."
        set opt(cp) "none"
} else {
      puts "Loading connection pattern..."
      source $opt(cp)
}
#
# Tell all the nodes when the simulation ends
#
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}
$ns_ at $opt(stop).00000001 "puts \"NS EXITING...\" ; $ns_ halt"

if { $opt(sc) == "" } {
      puts "*** NOTE: no scenario file specified."
        set opt(sc) "none"
} else {
      puts "Loading scenario file..."
      source $opt(sc)
      puts "Load complete..."
}
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp $opt(rp)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"

puts "Starting Simulation..."
$ns_ run
```

โดยส่วนของสคริปท์ในการกำหนดเงื่อนไขและสภาพแวดล้อมของการจำลองได้แยกออกมาจากส่วนหลักของสคริปท์โดยคจากตัวอย่าง ค่าที่กำหนดคือคือ

opt(chan)	ชนิดของช่องสัญญาณ
opt(prop)	ลักษณะการเดินทางของคลื่น
opt(netif)	ชนิดของอินเตอร์เฟต
opt(mac)	ชนิดของ Mac Layer
opt(ifq)	ชนิดของคิว
opt(ll)	ชนิดของ Link Layer

opt(ant)	ชนิดของเสาอากาศ
opt(x)	ขนาดพื้นที่ในแนวแกน x
opt(y)	ขนาดพื้นที่ในแนวแกน y
opt(cp)	ระบุไฟล์รูปแบบการเชื่อมต่อ
opt(sc)	ระบุไฟล์รูปแบบการเคลื่อนที่
opt(ifqlen)	จำนวนแพ็คเกตมากที่สุดในคิว
opt(nn)	จำนวนโหนดทั้งหมด
opt(seed)	ค่าที่ใช้ในการสุ่ม
opt(stop)	เวลาในการจำลอง (วินาที)
opt(tr)	ระบุเทรซไฟล์
opt(rp)	ระบุเราติ้งโปรโตคอล
opt(initialenergy)	กำหนดค่าพลังงานเริ่มต้นของแต่ละโหนด (จูลส์)

ซึ่งเมื่อได้ทำการกำหนดค่าทั้งหมดซึ่งเป็นเงื่อนไขที่จำเป็นในการกำหนดสภาพแวดล้อมของการจำลองแล้ว จึงใช้สคริปท์ดังกล่าวทำการจำลองด้วย NS-2 และจะได้ผลลัพธ์ซึ่งอยู่ในรูปของเทรซไฟล์ โดยชื่อของเทรซไฟล์จะเป็นไปตามที่กำหนดไว้ในสคริปท์ ซึ่งเทรซไฟล์ที่ได้จะเป็นเหมือนข้อมูลดิบที่บอกถึงเหตุการณ์ทั้งหมดที่เกิดขึ้นในการจำลอง รวมถึงรายละเอียดของแต่ละเหตุการณ์นั้น

5.6 การวิเคราะห์เทรซไฟล์ด้วย PERL

ดังที่ได้กล่าวไปแล้ว เกี่ยวกับการสูญเสียพลังงานของอุปกรณ์ไร้สายในเครือข่ายไร้สายเฉพาะกิจ โดยเฉพาะอย่างยิ่งในกรณีที่มีการส่งข้อมูลผ่านอุปกรณ์ไร้สายที่เป็นตัวกลางก่อนที่จะถึงปลายทาง ทำให้อุปกรณ์ระหว่างจำเป็นต้องเสียพลังงานในการรับและส่งเฟรมที่ไม่ใช่ของตน ซึ่งเป็นพลังงานที่เสียไปเพื่ออุปกรณ์อื่น ดังนั้นการสูญเสียพลังงานของอุปกรณ์นั้น ไม่อาจคำนึงเฉพาะข้อมูลที่ตนรับและส่งเท่านั้น

ในระบบเครือข่ายไร้สายเฉพาะกิจ จำเป็นต้องมีข้อความที่ใช้ในการควบคุมการรับส่งข้อมูล เรียก Control Message ซึ่งหลักๆแล้วประกอบด้วย RTS (Request To Send), CTS (Clear To Send) และ ACK (Acknowledgement) ซึ่งทั้ง 3 ข้อความนี้เป็นกลไกที่ใช้ในการลดการเกิดการชนกันของข้อมูลในเครือข่าย ซึ่งรายละเอียดจะเป็นไปดังที่ได้กล่าวไปแล้วในเบื้องต้นในบทที่ 4 ซึ่งแน่นอนว่าการรับส่งข้อมูลประเภทนี้ก็จำเป็นต้องใช้พลังงานในการกระทำกิจกรรมดังกล่าวเช่นกัน ซึ่งเป็นพลังงานที่เสียไปกับข้อมูลที่ไม่ใช่ข้อมูลที่ต้องการส่งจริงๆ จึงจำเป็นต้องนำมาทำการพิจารณาด้วย

ระบบเครือข่ายไร้สายเฉพาะกิจที่มีการส่งข้อมูลผ่านหลายอุปกรณ์กว่าจะถึงปลายทางนั้น จำเป็นที่จะต้องมีเราติ้งโปรโตคอลที่ใช้ในการหาเส้นทาง เช่น DSR, AODV เป็นต้น ซึ่งในการค้นหาเส้นทางก็จำเป็นต้องใช้ข้อความในการแลกเปลี่ยนระหว่างอุปกรณ์ ซึ่งก็ด้วยเหตุผลเดิมนั่นก็คือต้องมีการสูญเสียพลังงานในการแลกเปลี่ยนข้อมูลดังกล่าวนี้ด้วย

ในระบบเครือข่าย การรับและส่งข้อมูลจำเป็นต้องทราบถึงที่อยู่ในระดับที่ 2 เช่น MAC Address ซึ่งจำเป็นต้องมีกลไกในการให้ได้มาซึ่ง MAC Address โดยใช้ที่อยู่ในระดับที่สูงขึ้นไป ซึ่ง ARP (Address Resolution Protocol) เป็นกลไกที่ใช้ในการแก้ปัญหานี้ ซึ่งมีการส่ง ARP request และ reply เพื่อให้ได้มาซึ่งที่อยู่ดังกล่าว และก็เป็นข้อมูลอีกชนิดที่จำเป็นต้องนำมาพิจารณาการสูญเสียพลังงานด้วย

จากตัวอย่างเหตุผลดังกล่าวในข้างต้น จะเห็นได้ว่า การสูญเสียพลังงานของอุปกรณ์ไม่ได้เกิดจากตัวอุปกรณ์เองเท่านั้น และไม่ได้เกิดจากเฉพาะข้อมูลจากระดับบนเท่านั้นซึ่งเป็นข้อมูลจริงๆ ที่ผู้ใช้ต้องการส่ง

โดยการที่อุปกรณ์ต้องทำการรับส่งข้อมูลที่ไม่ใช่ข้อมูลที่ตนต้องการนั้น เป็นการสูญเสียข้อมูลเพื่อคนอื่น ที่อาจทำให้เกิดกรณีที่สูญเสียไปจนไม่เพียงพอที่จะทำงานของตนเอง ซึ่งอาจทำให้เกิดความไม่ยุติธรรมขึ้น เนื่องจากอาจมีบางโหนดที่ใช้พลังงานของโหนดอื่นเพื่อกิจกรรมของตนเอง ในขณะที่บางโหนดใช้พลังงานเพื่อกิจกรรมของคนอื่นโดยที่ไม่เพียงพอในการทำกิจกรรมของตนเอง แต่ในขณะเดียวกันเมื่อมองในมุมกลับกันก็จะพบว่ามีโหนดอื่นได้สูญเสียพลังงานในการทำกิจกรรมแก่ตนเช่นเดียวกัน ซึ่งอาจนำมาเปรียบเทียบเพื่อหาความยุติธรรมในมุมมองระหว่างพลังงานที่ใช้ไปเพื่อผู้อื่น และพลังงานที่ได้โหนดอื่นทำงานให้ตน ซึ่งเหมือนเป็นการได้รับพลังงานนั้น

ซึ่งในเทรซไฟล์นั้น จะมีลักษณะดังที่ได้กล่าวไปแล้วดังบทที่ 4 ซึ่งจะระบุเหตุการณ์ที่เกิดขึ้น ซึ่งเหตุการณ์ที่เราจะนำมาพิจารณาจะมีดังนี้

- S คือการที่โหนดทำการส่งข้อมูลออกจากตนเอง โดยเราต้องพิจารณาการส่งข้อมูลในระดับ เท่านั้น เนื่องจากในระดับอื่นจะไม่เกิดการสูญเสียพลังงาน เพราะเป็นการส่งข้อมูลระหว่างระดับชั้นภายในโหนดเท่านั้น แต่ระดับ MAC เป็นการส่งข้อมูลออกจากอุปกรณ์จริงๆ
- R คือการที่โหนดทำการรับข้อมูลเข้าหาตนเองโดยเราต้องพิจารณาการสรับข้อมูลในระดับ MAC เท่านั้น เนื่องจากในระดับอื่นจะไม่เกิดการสูญเสียพลังงาน เพราะเป็นการรับข้อมูลระหว่างระดับชั้นภายในโหนดเท่านั้น แต่ระดับ MAC เป็นการรับข้อมูลออกจากอุปกรณ์จริงๆ

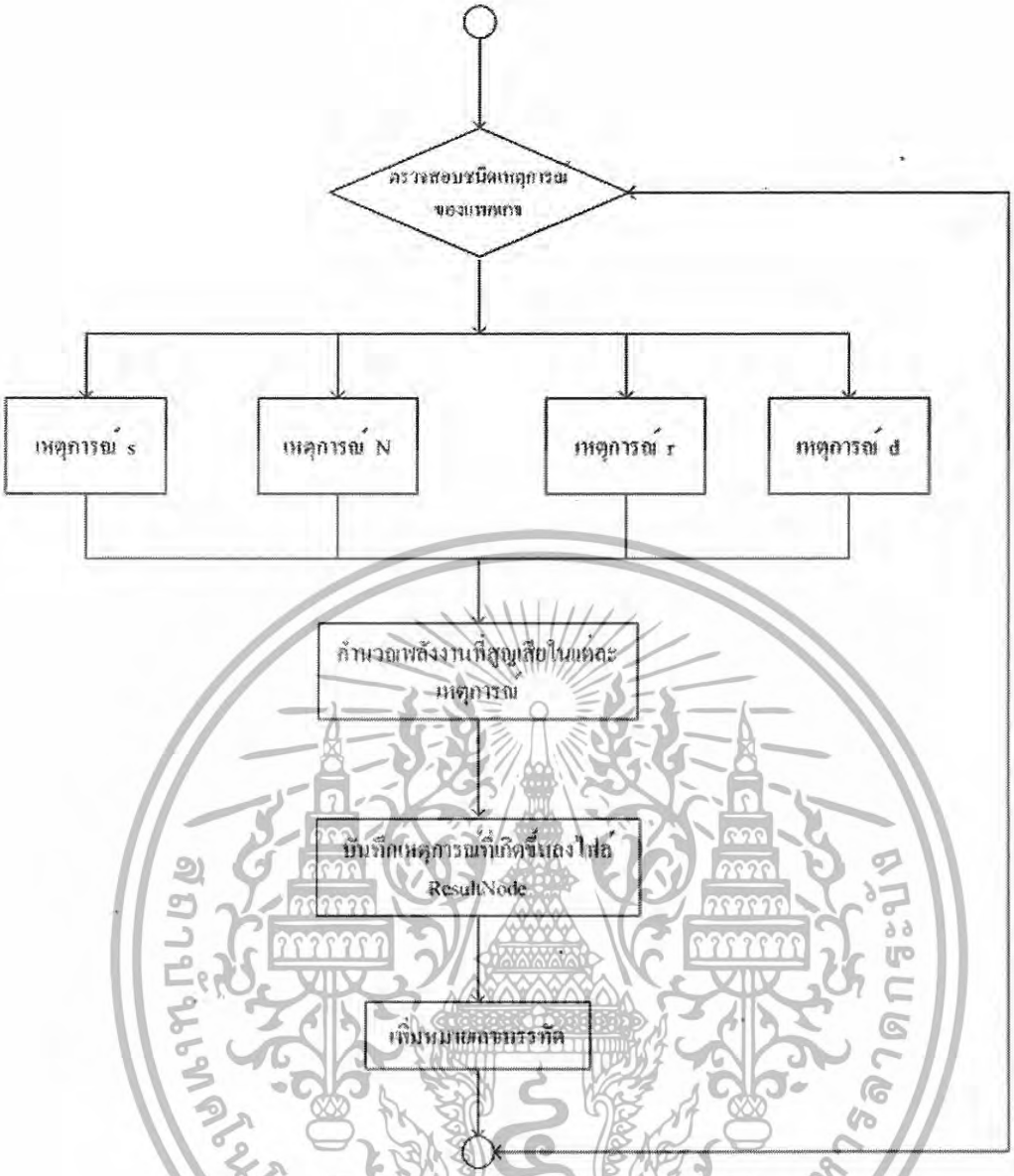
- D คือการที่โหนดที่ทำการรับข้อมูลมาพร้อมกันแล้วมีความจำเป็นต้องลบข้อมูลนั้นทิ้ง เนื่องจากเกิดการชนกันของข้อมูล ซึ่งข้อมูลจะเกิดความเสียหายทำให้ไม่สามารถรับและอ่านได้
- N คือการที่โหนดทำการดักจับข้อมูลผ่านทางเสาอากาศ ซึ่งจะเป็นกระบวนการก่อนที่จะตัดสินใจว่าจะเลือกรับหรือจะลบทิ้ง โดยจะเป็นเหตุการณ์ที่แสดงถึงพลังงานที่เปลี่ยนแปลงไปจากการดักจับเฟรม โดยยังไม่ตัดสินใจว่าจะรับหรือจะลบทิ้งหรือจะรับเฉยๆ ซึ่งหากเป็นกรณีหลังจะนับว่าเป็นการฟัง Listen เท่านั้น

ซึ่งจะแบ่งการวิเคราะห์เทรซไฟล์ด้วย PERL ออกเป็น 2 ส่วน ดังนี้

- การประมวลผลเทรซไฟล์เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม
- การประมวลผลเพื่อจำแนกจุดประสงค์ของการสูญเสียพลังงาน

5.7 การประมวลผลเทรซไฟล์เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม

ในการหาค่าพลังงานของแต่ละโหนดนั้น เราจะหาค่าพลังงานที่สูญเสียไปแต่ละเหตุการณ์ เพื่อที่ว่าจะนำมาวิเคราะห์ว่าเมื่อมีเหตุการณ์ต่างๆ เกิดขึ้นนั้น โหนดมีการสูญเสียพลังงานไปเพื่อตนเอง หรือว่าเพื่อโหนดอื่นอย่างไรบ้าง โดยเรามีอัลกอริธึมในการหาค่าพลังงานดังรูปที่ 5.2



รูปที่ 5.2 แสดงโครงสร้างการทำงานในการประมวลผลเทรซไฟล์

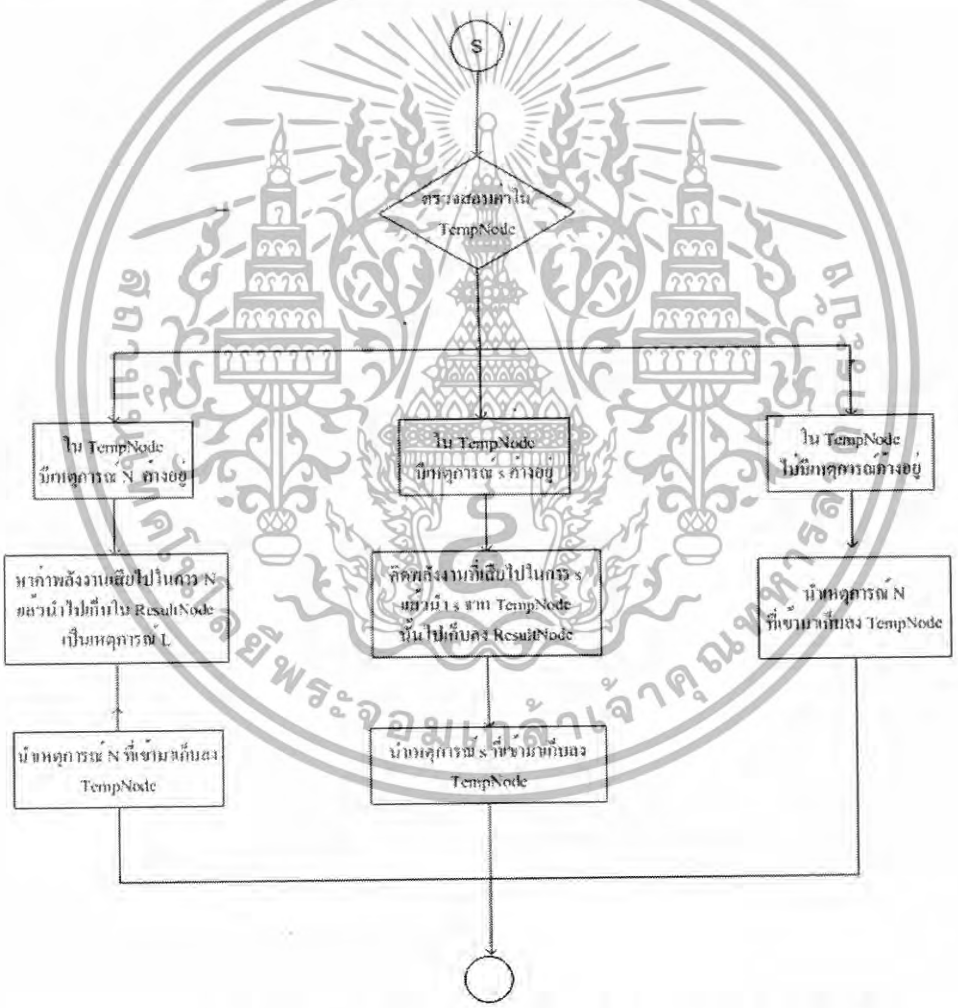
เมื่อได้รับเหตุการณ์แต่ละเหตุการณ์เข้ามาเราจะต้องตรวจสอบก่อนว่าเหตุการณ์นั้นเป็นชนิดแพ็คเกตอะไร เพื่อนำไปจำแนกในการทดลอง ซึ่งแพ็คเกตที่เข้ามานั้นก็เป็นได้ทั้งแพ็คเกตที่เป็นข้อมูลจริง และแพ็คเกตควบคุมต่างๆ และต้องทำการอ่านไฟล์ที่เป็นรูปแบบในการเชื่อมต่อว่าในการจำลองนี้ หมายเลขการไหลของข้อมูลหมายเลขนี้มีการส่งจากต้นทางไปปลายทางใดบ้าง

ในการคิดค่าพลังงานที่ใช้ไปของแต่ละเหตุการณ์นั้น แต่ละโหนดจะมีการสร้างไฟล์เอาไว้โหนดละสองไฟล์ คือไฟล์ ResultNode และไฟล์ TempNode โดยไฟล์ ResultNode จะทำการเก็บเหตุการณ์ที่เกิดขึ้นทั้งหมดของโหนดนั้นที่ทำการหาค่าพลังงานที่ใช้ไปของแต่ละเหตุการณ์เรียบร้อยแล้ว ส่วนในไฟล์ TempNode จะเป็นที่เก็บข้อมูลชั่วคราวในกรณีที่เมื่อเกิดเหตุการณ์ที่ยังไม่สามารถคิดพลังงานที่ใช้ไปในเหตุการณ์นั้นได้

จะกำหนดอาร์เรย์ที่เก็บพลังงานปัจจุบันของแต่ละโหนด โดยค่าพลังงานเริ่มต้นของแต่ละโหนดจะกำหนดให้มีค่าเท่ากับ 100 และจะเปลี่ยนแปลงก็ต่อเมื่อมีการบันทึกเหตุการณ์และการใช้พลังงานลงใน ResultNode และได้กำหนดอาร์เรย์อีกหนึ่งชุดเพื่อเก็บว่าโหนดที่เกิดเหตุการณ์ก่อนหน้าเหตุการณ์นี้คือโหนดอะไร เพื่อนำมาใช้ในการหาค่าพลังงานในบางกรณีซึ่งไม่สามารถหาจากโหนดตัวเองได้

5.7.1 กรณีที่มีเหตุการณ์ s เกิดขึ้น

ในกรณีที่มีเหตุการณ์ s เกิดขึ้นนั้นหมายถึงว่าโหนดนั้นมีการส่งข้อมูล ซึ่งการจำลองใน NS-2 นั้นจะกำหนดให้มีการสูญเสียพลังงานเมื่อเป็นเหตุการณ์ที่อยู่ในระดับแมคเลเยอร์ ซึ่งจะสังเกตได้จากฟิลด์ -Nl ในการหาค่าพลังงานที่ใช้ไปของเหตุการณ์ s นั้นจะสามารถหาได้ดังนี้



รูปที่ 5.3 แสดงอัลกอริธึมการทำงานเมื่อมีเหตุการณ์ s เกิดขึ้น

จากรูปที่ 5.3 เป็นอัลกอริธึมในการคิดพลังงานเมื่อมีเหตุการณ์ s เกิดขึ้น จะทำการตรวจสอบว่า เหตุการณ์ที่เกิดขึ้นนั้นเป็นของโหนดใด โดยตรวจสอบจากฟิลด์ -Ni จากนั้นจะทำการคำนวณพลังงานที่ใช้ไปเมื่อเกิดเหตุการณ์ดังกล่าว โดยแต่ละโหนดจะมีไฟล์สองชนิดคือ

ไฟล์ที่ชื่อว่า ResultNode และไฟล์ TempNode ซึ่งในการคิดนั้นเราจะเปิดไฟล์ TempNode ของโหนดนั้นว่ามีเหตุการณ์อะไรที่ติดค้างอยู่ในนั้นหรือไม่ ซึ่งเหตุการณ์ที่จะค้างอยู่นั้นมีอยู่ทั้งหมดสามกรณี คือ

5.7.1.1 ในกรณีที่มี เหตุการณ์ N ค้างอยู่ในไฟล์ TempNode ของโหนดนั้น

ตัวอย่าง เมื่อมีเหตุการณ์ s เกิดขึ้นดังนี้

```
s -t 46.970184013 -Hs 1 -Hd 2 -Ni 1 -Nx 250.15 -Ny 335.61 -
 Nz 0.00 -Ne 94.2349909 -Nl MAC -Nw --- -Ma 13a -Md 2 -Ms 1
 -Mt 800 -Is 1.0 -Id 2.0 -It cbr -Il 600 -If 1000 -Ii 0 -Iv
 32 -Pn cbr -Pi 0 -Pf 0 -Po 1
```

หมายความว่าเป็นเหตุการณ์ที่โหนด 1(-Ni) มีการส่งข้อมูลที่เป็น cbr (-it) โดยก่อนที่จะมีการส่งข้อมูลมีพลังงานเริ่มต้นเท่ากับ 94.2349909

ค่าพลังงานปัจจุบันที่เก็บอยู่ในอาร์เรย์ $energy มีค่าเท่ากับ 94.23518289 9999998416 และมีค่าที่เก็บอยู่ใน ไฟล์ ResultNode1 และ TempNode1 ดังนี้

ResultNode1

```
44.869327692 r 0.000239999999990914 DSR
44.870038611 r 0.000191999999998416 arp
44.870114 L 0.000106000000002382
44.870477 L 9.09999999976208e-05
```

TempNode1

```
44.870791 N 94.2349909
```

จากตัวอย่างจะเห็นว่ามีเหตุการณ์ N ค้างอยู่ใน TempNode แล้วมีเหตุการณ์ s เข้ามา แสดงว่า เหตุการณ์ N ดังกล่าวเป็นการฟังเฉยๆ ไม่ได้มีการรับหรือว่าการทิ้งข้อมูลแต่อย่างใด เราจะทำการย้ายเหตุการณ์ดังกล่าวไปเก็บไว้ที่ไฟล์ ResultNode ของโหนดนั้น โดยจะกำหนดใหม่ว่าเป็น เหตุการณ์ L ซึ่งหมายถึงเป็นการฟังและก็เก็บพลังงานที่ใช้ในการฟัง โดยพลังงานที่ใช้ไปในการ L จะหาได้ดังนี้

พลังงานที่ใช้ในการ L = พลังงานปัจจุบัน – พลังงานที่ได้จากเหตุการณ์ N ใน TempNode (5.1)

เมื่อคำนวณพลังงานเรียบร้อยแล้วจะทำการย้ายเหตุการณ์ดังกล่าวไปไว้ที่ ResultNode1 แล้วก็ปรับค่าที่อยู่ในอาร์เรย์ energy ให้เป็นค่าพลังงานปัจจุบันใหม่คือ 94.2349909 ได้ผลดังนี้

ResultNode1

```
44.869327692 r 0.000239999999990914 DSR
44.870038611 r 0.000191999999998416 arp
44.870114 L 0.000106000000002382
44.870477 L 9.09999999976208e-
44.870791 L 0.000191999999998416
```

TempNode1

```
46.970184013 s 94.2349909 cbr 1000 1 2
```

หมายความว่าโหนด 1 นั้นมีเหตุการณ์ฟังเกิดขึ้นที่เวลา 44.870791 โดยพลังงานที่ใช้ในการฟังมีค่าเท่ากับ 0.000191999999998416

ที่ไฟล์ TempNode1 มีการเก็บเหตุการณ์ s ลงไปแทนเนื่องจากว่าไม่สามารถคิดพลังงานที่ใช้ไปในเหตุการณ์ s ได้จึงเก็บไว้ที่ TempNode1 ก่อน

5.7.1.2 ในกรณีที่มีเหตุการณ์ s ค้างอยู่ในไฟล์ TempNode ของโหนดนั้น

ตัวอย่าง เมื่อมีเหตุการณ์ s เกิดขึ้นดังนี้

```
s -t 58.220406 -Hs 1 -Hd 2 -Ni 1 -Nx 250.15 -Ny 335.61 -Nz
0.00 -Ne 89.2349389 -Nl MAC -Nw --- -Ma 13a -Md 2 -Ms 1 -Mt
800 -Is 1.0 -Id 2.0 -It cbr -Il 600 -If 1000 -Ii 0 -Iv 32 -
Pn cbr -Pi 0 -Pf 0 -Po 1
```

หมายความว่าเป็นเหตุการณ์ที่โหนด 1(-Ni) มีการส่งข้อมูลที่เป็น cbr (-it) โดยก่อนที่จะมีการส่งข้อมูลมีพลังงานเริ่มต้นเท่ากับ 89.2349389

ค่าพลังงานปัจจุบันที่เก็บอยู่ในอาร์เรย์ $energy มีค่าเท่ากับ 89.23695889846 และมีค่าที่เก็บอยู่ในไฟล์ ResultNode1 และค่าที่ค้างอยู่ใน TempNode1 ดังนี้

ResultNode1

```
58.211745 L 0.000106000000002382
58.212108 L 9.09999999976208e-05
58.212423 L 0.00140100000000132
58.217106 L 9.20000000093069e-05
```

TempNode1

```
58.218405 s 89.23695889846 ctrl
```

ในการคิดพลังงาน จะทำการคิดพลังงานที่สูญเสียไปของเหตุการณ์ s ที่ค้างอยู่ใน ไฟล์ TempNode1 จะหาได้จาก

พลังงานที่ใช้ในการ s = พลังงานที่เหลือใน TempNode – พลังงานที่ได้จากเหตุการณ์ s (5.2)

จากนั้นก็จะย้ายเหตุการณ์ s ที่อยู่ใน TempNode1 ไปเก็บไว้ที่ไฟล์ ResultNode1 ของโหนดนั้น โดยจะนำพลังงานที่ใช้ไปเก็บลงไปด้วย และเปลี่ยนค่าปัจจุบันของอาร์เรย์ energy ให้มีค่าเท่ากับ 89.2349389 และนำเหตุการณ์ s ที่พึ่งเข้ามาใหม่มาเก็บไว้ใน TempNode1 เพื่อรอการคิดพลังงานต่อไป จึงได้

ResultNode1

```
58.211745 L 0.00010600000000002382
58.212108 L 9.09999999976208e-05
58.212423 L 0.00140100000000132
58.217106 L 9.20000000093069e-05
58.218405 s 0.00201999846 ctrl
```

TempNode1

```
58.220406 s 89.2349389 cbr 1000 1 2
```

5.7.1.3 กรณีที่เข้ามาเปิดไฟล์ TempNode ของโหนดนั้นแล้วไม่มีเหตุการณ์ใดค้างอยู่

ตัวอย่าง เมื่อมีเหตุการณ์ s เกิดขึ้นดังนี้ แล้วใน TempNode1 ไม่ได้มีเหตุการณ์อะไร ค้างอยู่เลย

```
s -t 365.027206958 -Hs 1 -Hd 2 -Ni 1 -Nx 250.15 -Ny 335.61 -
Nz 0.00 -Ne 63.3920594 -Nl MAC -Nw --- -Ma 13a -Md 2 -Ms 1 -
Mt 800 -Is 1.0 -Id 2.0 -It cbr -Il 600 -If 1000 -Ii 0 -Iv 32
-Pn cbr -Pi 0 -Pf 0 -Po 1
```

ResultNode1

```
364.998211 L 0.00140100000000132
365.002894 L 9.09999999976208e-05
365.026529420 s 0.000212000000004764 ctrl
365.027196958 r 9.09999999976208e-05 ctrl
```

TempNode1

```
```

เราจะทำการเพิ่มเหตุการณ์ s ที่เข้ามาใหม่เข้าไปเก็บไว้ในไฟล์ TempNode1 เพื่อรอการคิดค่าพลังงาน จึงได้ผลดังนี้

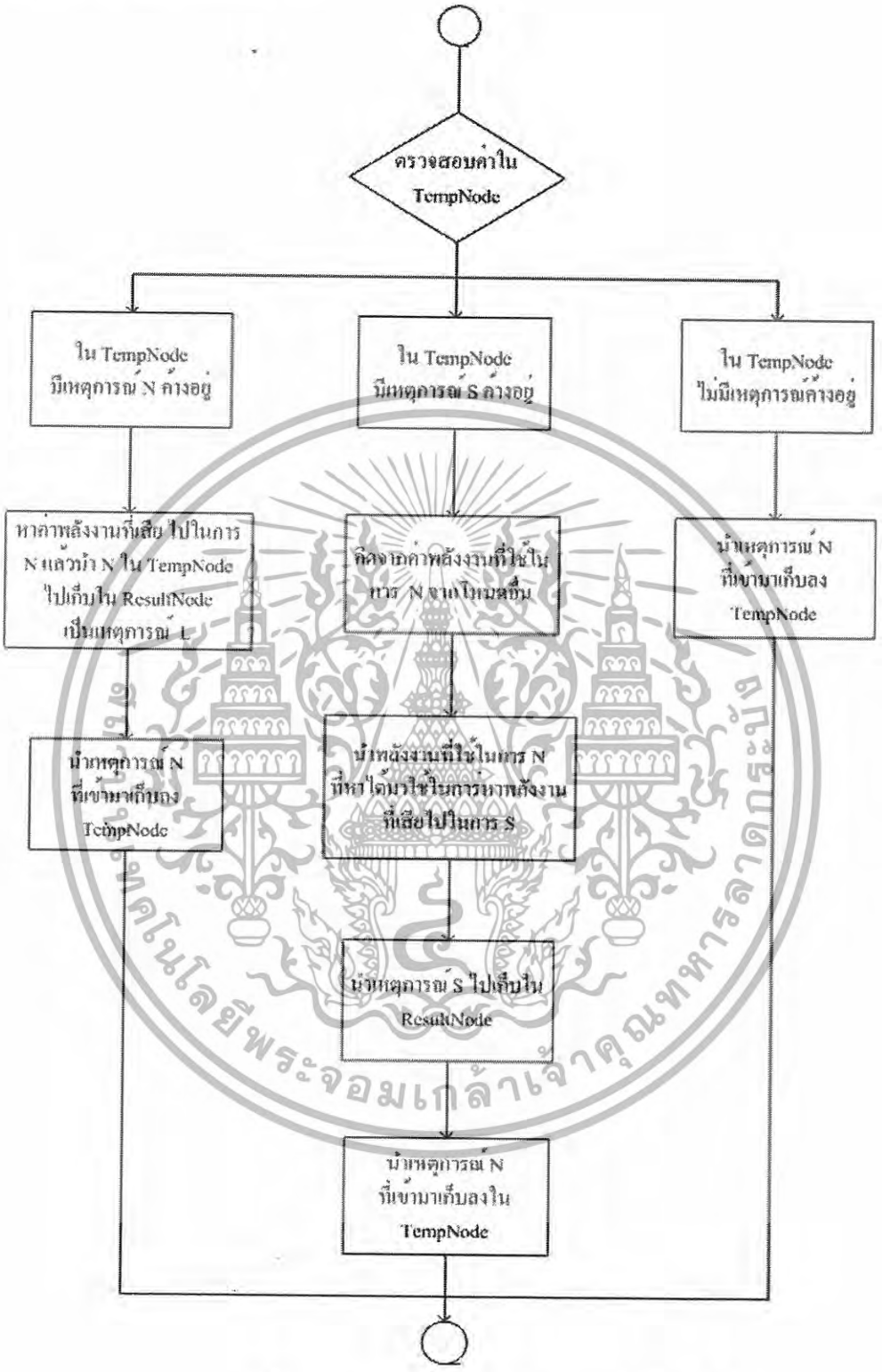
ResultNode1

```
364.998211 L 0.00140100000000132
365.002894 L 9.09999999976208e-05
365.026529420 s 0.000212000000004764 ctrl
365.027196958 r 9.09999999976208e-05 ctrl
```

TempNode1

```
365.027206958 s 63.3920594 cbr 1000 1 2
```

5.7.2 กรณีที่มีเหตุการณ์ N เกิดขึ้น



รูปที่ 5.4 แสดงการทำงานเมื่อมีเหตุการณ์ N เกิดขึ้น

จากรูปที่ 5.4 เมื่อมีเหตุการณ์ N เกิดขึ้น เรายังไม่สามารถสรุปได้ว่าจะเป็นการรับฟัง การรับหรือว่าเป็นการทิ้ง เราจะทำการตรวจสอบว่า เหตุการณ์ที่เกิดขึ้นนั้นเป็นของโหนดใด โดยตรวจสอบจากฟิลด์ -n จากนั้นก็จะทำการเปิดไฟล์ TempNode ของโหนดนั้นว่ามีเหตุการณ์อะไรที่ติดค้างอยู่ในนั้นหรือไม่ ซึ่งเหตุการณ์ที่จะค้างอยู่นั้นมีอยู่ทั้งหมดสามกรณี

5.7.2.1 กรณีที่มีเหตุการณ์ N ค้างอยู่ใน TempNode

```
N -t 364.109016 -n 5 -e 63.34583029680
```

หมายความว่าเป็นเหตุการณ์ที่โหนดที่ 5(-n) มีการฟังเกิดขึ้น โดยพลังงานที่เหลือจากการฟังครั้งนี้จะมีค่าเท่ากับ 63.34583029680

ค่าพลังงานปัจจุบันที่เก็บอยู่ในอาร์เรย์ $energy มีค่าเท่ากับ 63.43549326 และมีค่าที่เก็บอยู่ในไฟล์ ResultNode5 และค่าที่ค้างอยู่ใน TempNode5 ดังนี้

ResultNode5

```
364.100830 L 0.0001060000000002382
364.101193 L 9.099999999976208e-05
364.101508 L 0.00140199999999879
364.106190 L 9.099999999976208e-05
364.107030 L 0.000105000000004907
```

TempNode5

```
364.108745 N 63.43538725999991608
```

ถ้าหากว่าถ้า N ใหม่ที่เข้ามานั้นมีพลังงานไม่เท่ากับพลังงานงานของเหตุการณ์ N ที่ค้างอยู่ใน TempNode5 สามารถสรุปได้เลยว่า N ที่ค้างอยู่ใน TempNode5 นั้นเป็นเหตุการณ์ L คือการฟัง ให้เราทำการคิดพลังงานที่เสียไปในเหตุการณ์นั้น แล้วทำการย้ายเหตุการณ์ N จาก TempNode5 ไปเก็บเอาไว้ใน ResultNode5 โดยเก็บเป็นเหตุการณ์ L ซึ่งหมายถึงการฟัง จากนั้นก็เอาเหตุการณ์ N ที่เข้ามาใหม่ไปเก็บใน TempNode5 แทน โดยพลังงานในการ L หาได้ดังนี้

พลังงานที่ใช้ในการ L = พลังงานปัจจุบัน – พลังงานที่ได้จากเหตุการณ์ N ใน TempNode (5.3)

เมื่อคิดพลังงานเสร็จเรียบร้อยแล้ว ก็ให้เปลี่ยนค่าพลังงานปัจจุบันที่เก็บไว้ในอาร์เรย์ energy ให้มีค่าเท่ากับ 63.4353872599991608 และไฟล์ ResultNode5 และ TempNode5 จะมีค่าดังนี้

ResultNode5

```
364.100830 L 0.000106000000002382
364.101193 L 9.09999999976208e-05
364.101508 L 0.00140199999999879
364.106190 L 9.09999999976208e-05
364.107030 L 0.000105000000004907
364.108745 L 0.0001060000008392
```

TempNode5

```
364.109016 N  63.34583029680
```

5.7.2.2 กรณีที่มีเหตุการณ์ s ค้างอยู่ใน TempNode

ตัวอย่างช่วงลำดับเหตุการณ์ที่เกิดขึ้นในแทรซไฟล์มีดังนี้

```
r -t 2.559128379 -Hs 10 -Hd -1 -Ni 10 -Nx 455.75 -Ny 193.65
-Nz 0.00 -Ne 0.999798 -Nl MAC -Nw --- -Ma 0 -Md ffffffff -Ms
1 -Mt 800 -Is 1.255 -Id 2.255 -It DSR -Il 32 -If 0 -Ii 1 -Iv
32 -P dsr -Ph 1 -Pq 1 -Ps 1 -Pp 0 -Pn 1 -Pl 0 -Pe 0->0 -Pw 0
-Pm 0 -Pc 0 -Pb 0->0
s -t 2.564602459 -Hs 2 -Hd -2 -Ni 2 -Nx 330.16 -Ny 364.69 -
Nz 0.00 -Ne 0.999798 -Nl MAC -Nw --- -Ma 0 -Md ffffffff -Ms
2 -Mt 806 -P arp -Po REQUEST -Pms 2 -Ps 2 -Pmd 0 -Pd 1
N -t 2.564603 -n 44 -e 0.999606
N -t 2.564603 -n 1 -e 0.999405
N -t 2.564603 -n 47 -e 0.999606
N -t 2.564603 -n 4 -e 0.999606
```

เราจะพิจารณา N -t 2.564603 -n 1 -e 0.999405 ก่อนหน้าเหตุการณ์ดังกล่าวมีเหตุการณ์ที่เกิดขึ้นของโหนด 1 คือเหตุการณ์ s ซึ่งถูกเก็บเอาไว้ใน TempNode1 ดังนี้

TempNode1

```
2.558455546 s 1.000000 DSR
```

เราต้องคิดเหตุการณ์ s ที่ค้างอยู่ใน TempNode ก่อน แต่เนื่องจากว่ากรณีนี้จะไม่สามารถคิดเหมือนกรณีอื่นๆได้ เนื่องจากว่าข้อมูลพลังงานที่ได้จากเหตุการณ์ s นั้น

เป็นพลังงานที่ยังไม่ได้เสียไปในการ s ครั้งนั้น เราจึงจำเป็นต้องรอเหตุการณ์ถัดไปของโหนดนี้ จึงจะสามารถหาพลังงานของกิจกรรม s นี้ได้ นั่นคือ เหตุการณ์ N ที่เกิดขึ้นซึ่งมีพลังงานเท่ากับ 0.999405 ในกรณีหมายความว่า หลังจาก s แล้วเกิดเหตุการณ์ N เราจะทำการหาพลัง s ที่ค้างอยู่ใน TempNode โดยหาพลังงานที่เสียไปในการ N ครั้งนี้ โดยจะหาจากเหตุการณ์ของโหนดที่เกิดเหตุการณ์ก่อนหน้า(บรรทัดก่อนหน้า) ซึ่งก็คือเหตุการณ์ของโหนด 44 ได้แก่ N -t 2.564603 -n 44 -e 0.999606

โดยดูจาก TempNode ของโหนด 44 ซึ่งภายในนั้นต้องเป็นเหตุการณ์ N เช่นเดียวกัน เนื่องจากสมมติฐานว่าจะเป็นการ N จากข้อมูลเดียวกันซึ่งจะเสียพลังงานเท่ากัน โดยจะหาพลังงานจากการ N ครั้งนั้นของโหนด 44 ออกมา แล้วนำไปหาพลังงานที่ใช้ในการ s ของโหนด 1 ได้ดังนี้

พลังงานที่ใช้ในการ s ของ x = พลังงานของ x – ((พลังงานของy-พลังงานที่เหลือจากการ n ของ y)+พลังงานที่เหลือจากการ n ของ x)

กำหนดให้ x คือโหนดปัจจุบันและ y คือโหนดก่อนหน้า เมื่อเราทำการหาค่าพลังงานในการ s เสร็จเรียบร้อยแล้ว เราก็นำเหตุการณ์ s นั้นย้ายไปเก็บในไฟล์ ResultNode1 และนำเหตุการณ์ N ของโหนด 1 ที่เข้ามาใหม่เก็บลงใน TempNode1 จึงได้

ResultNode1

```
2.558455546 s 0.000403 DSR
```

TempNode1

```
2.564603 N 0.999405
```

แต่หากกรณีที่โหนดก่อนหน้าไม่ใช่ N จะทำการไปหาพลังงานที่ใช้ในการ N จากเหตุการณ์ถัดไปแทน ซึ่งจะใช้แนวคิดเดียวกัน

5.7.2.3 กรณีที่เข้ามาเปิดไฟล์ TempNode ของโหนดนั้นแล้วไม่มีเหตุการณ์ใดค้างอยู่

เมื่อมีเหตุการณ์ N เกิดขึ้น แล้วใน TempNode1 ไม่ได้มีเหตุการณ์อะไรค้างอยู่เลย

```
N -t 2.565880 -n 1 -e 0.999102
```

TempNode1

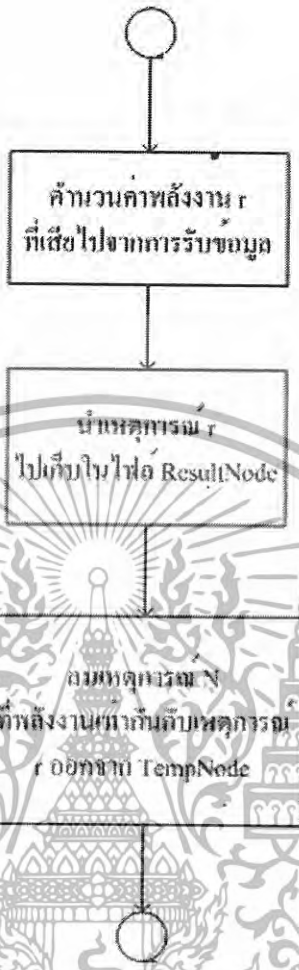
```

```

จะทำการเพิ่มเหตุการณ์ N ที่เข้ามาใหม่เข้าไปเก็บไว้ในไฟล์ TempNode1 เพื่อรอการคิดค่าพลังงาน จึงได้ผลดังนี้

TempNode1

```
2.565880 N 0.999102
```

5.7.3 กรณีที่มีเหตุการณ์ r เกิดขึ้น



รูปที่ 5.5 แสดงการทำงานเมื่อมีเหตุการณ์ r เกิดขึ้น

จากรูปที่ 5.5 กรณีที่เหตุการณ์ที่เข้ามานั้นเป็นเหตุการณ์ r หมายถึงว่า โหนดนั้นมีการรับข้อมูล ซึ่งการที่จะมีเหตุการณ์ r ได้โหนดนั้นจะต้องมีเหตุการณ์ N มาก่อนแล้ว ซึ่งเราสามารถดูได้จากพลังงานคงเหลือซึ่งจะมีค่าเท่ากัน

ตัวอย่าง กรณีที่มีเหตุการณ์ r เกิดขึ้น และพลังงานปัจจุบันมีค่าเท่ากับ 1.0003546821

```
r -t 2.565870027 -Hs 2 -Hd -2 -Ni 2 -Nx 330.16 -Ny 364.69 -Nz
0.00 -Ne 0.999309 -Nl MAC -Nw --- -Ma 4fe -Md 2 -Ms 1 -Mt 0
```

TempNode2

```
2.565518 N 0.999309
```

กรณีที่มี r เข้ามานั้นเราจะทำการตรวจสอบก่อนว่าค่าพลังงานของเหตุการณ์ r มีค่าเท่ากับค่าพลังงานของเหตุการณ์ N ที่อยู่ใน TempNode ของโหนดนั้นหรือไม่ ซึ่งจะเห็นว่า พลังงานของเหตุการณ์ N ที่อยู่ใน TempNode2 กับพลังงานของเหตุการณ์ r ที่เข้ามาใหม่นั้นมีค่าเท่ากัน แสดงว่า N นั้นเป็นการฟังเพื่อรับข้อมูล โดยเราสามารถหาพลังงานที่ใช้ในการ r ได้ดังนี้

พลังงานที่ใช้ในการ r = พลังงานปัจจุบัน – พลังงานจากการ r

เมื่อทำการคำนวณค่าพลังงานที่ใช้ในการ r เสร็จเรียบร้อยแล้วให้ลบเหตุการณ์ N ที่อยู่ใน TempNode2 ออก แล้วนำเหตุการณ์ r ไปใส่ใน ResultNode2 จึงได้

ResultNode2

```
2.565518 r 0.0010456821 ctrl
```

5.7.4 กรณีที่มีเหตุการณ์ d เกิดขึ้น

รูปที่ 5.6 แสดงการทำงานเมื่อมีเหตุการณ์ d เกิดขึ้น

จากรูปที่ 5.6 กรณีที่เหตุการณ์ที่เข้ามานั้นเป็นเหตุการณ์ d หมายถึงว่า โหนดนั้นมีการทิ้งข้อมูลเนื่องจากมีการชนกันของข้อมูล ซึ่งการที่จะมีเหตุการณ์ d ได้โหนดนั้นจะต้องมีเหตุการณ์ N มาก่อนแล้วสองครั้ง ซึ่งเราสามารถดูได้จากพลังงานคงเหลือซึ่งจะมีค่าเท่ากัน

ตัวอย่าง กรณีที่มีเหตุการณ์ d เกิดขึ้น และพลังงานปัจจุบันมีค่าเท่ากับ 0.7554830

```
d -t 31.549029625 -Hs 2 -Hd -1 -Ni 2 -Nx 285.81 -Ny 400.18 -Nz
0.00 -Ne 0.754316 -Nl MAC -Nw COL -Ma 0 -Md ffffffff -Ms 2f -Mt
800 -Is 9.255 -Id 11.255 -It DSR -Il 120 -If 0 -Ii 97 -Iv 32 -P
dsr -Ph 3 -Pq 1 -Ps 2 -Pp 0 -Pn 2 -Pl 0 -Pe 0->16 -Pw 0 -Pm 0 -Pc
0 -Pb 0->0
```

กรณีที่มี d เข้ามานั้นเราจะทำการตรวจสอบก่อนว่าค่าพลังงานของเหตุการณ์ d มีค่าเท่ากับค่าพลังงานของเหตุการณ์ N ที่อยู่ใน TempNode ของโหนดนั้นหรือไม่ ถ้าหากว่าเท่ากันเราจะทำการลบเหตุการณ์ N ใน TempNode ออก แล้วจะนำเหตุการณ์ d ไปใส่ใน ResultNode โดยพลังงานที่สูญเสียไปในเหตุการณ์ d เราสามารถหาได้ดังนี้

พลังงานที่ใช้ในการ d = พลังงานปัจจุบัน – พลังงานจากการ d

ที่ไฟล์ ResultNode2 จะมีผลลัพธ์ดังนี้

ResultNode2

```
31.549029625 d 0.001167 DSR
```

ซึ่งกระบวนการประมวลผลการสูญเสียพลังงานจะเป็นไปในลักษณะนี้เสมอจนครบทุกบรรทัดในเทรซไฟล์ ซึ่งก็จะทำให้ได้ผลลัพธ์เป็นกิจกรรมและพลังงานที่เสียไปในแต่ละกิจกรรม

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม

```
#!/usr/bin/perl

@energy;
for($i=0;$i<50;$i++) { #50->50 node #วนสร้างไฟล์เพื่อที่จะทำการเขียนตามจำนวนที่กำหนด
  $cmdresult =""; #กำหนดชื่อไฟล์และบัฟเฟอร์ที่จะใช้เก็บข้อมูลก่อนเขียนลงไฟล์
  $cmdtemp ="";
  $ResultNode="";
  $TempNode="";

  $cmdresult ="cmdresult$i";
  $cmdtemp ="cmdtemp$i";
  $ResultNode ="ResultNode$i";
  $TempNode = "TempNode$i";
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
  open($cmdresult,">$ResultNode");
  print $cmdresult "#Node $i from file result_DSR_500x500_30_cbr_5.tr
\n"; #ทำการแนบฉลากเพื่อบอกรายละเอียดของไฟล์

  open($cmdtemp,">$TempNode");

  @energy[$i]=100.000000; #ประกาศอะเรย์เพื่อใช้เก็บพลังงานคงเหลือของแต่ละโหนด
                          ให้ค่าเริ่มต้นพลังงานเท่ากับ 100
}
open(infile,"result_DSR_500x500_50_cbr_10.tr") or die "couldn't open
the file";

@previousNode;      #ประกาศตัวแปรและกำหนดค่าเริ่มต้น
$line_no=0;
$linetmpr=0;
$previous = 0;
$node = 0;
$PktType ="";
$filecp="";
$flowID="";
$source="";
$dest="";
$wait="";

while ($line = <infile>) { #ทำการวนอ่านไฟล์ทีละบรรทัดจนจบไฟล์
      ++$line_no;

       $mod =$line_no % 50;   #50->50 node
      @entry = split(" ", $line);    #นำบรรทัดมาแบ่งเป็นคอลัมน์ด้วยช่องว่าง

      if($entry[4] eq "cp") {          #เช็คเงื่อนไขเพื่อดึงรูปแบบการเชื่อมต่อจากไฟล์
            $filecp = $entry[5];

            open(infilecp,"$filecp") or die "couldn't open the file";
            while ($linecp = <infilecp>) {
                  ++$line_no;
                  @entrycp = split(" ", $linecp);
                  if($entrycp[2] eq "connecting"){
                        $src = $entrycp[1];
                        $dst = $entrycp[4];
                  }
                  if($entrycp[2] eq "class_"){
                        $fid = $entrycp[3];

                        open(cmdcp,">>connectionPattern");
                        print cmdcp "$fid $src $dst \n";
                        close(cmdcp);
                  }
            }
            close(infilecp);
      }

      #print "filecp = $filecp \n";
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
      if($entry[34] eq "DSR"){          #ตรวจสอบชนิดของเหตุการณ์
            $PktType = $entry[34];
      }elsif ($entry[34] eq "cbr"){
            $PktType =$entry[34];

            open(cpt,"connectionPattern") or die "couldn't open the
file";
            while ($lcp = <cpt>) {
                 @cp = split(" ", $lcp);
                 if($cp[0] eq $entry[38]){
                    $flowID=$cp[0];
                    $source=$cp[1];
                    $dest=$cp[2];
                 }
            }
            close(cpt);

      }elsif($entry[30] eq "arp"){
            $PktType =$entry[30];
      }else{
            if($entry[22] eq 0){
              $PktType ="ack";
            }else{
              $PktType ="ctrl";
            }
      }
      #print "$PktType \n";

      if($entry[0] eq "s"){    #เป็นเหตุการณ์การส่งข้อมูล
            if($entry[18] eq "MAC"){        #ทำงานในระดับ MAC
                 $previousNode[$mod]= $entry[8];
                 $cmds = "cmdtemp$entry[8]";
                 $temps ="TempNode$entry[8]";
                 open($cmds,"$temps");
                 $lines = <$cmds>;
                 @entrytmps = split(" ", $lines);
                 close($cmds);

                 if($entrytmps[1] eq "N")          #ตรวจสอบเหตุการณ์ใน TempNode
                 {

                  $cmdresults = "cmdresult$entry[8]";
                  $ResultNodes ="ResultNode$entry[8]";
                  $energyuseds = $energy[$entry[8]]-$entrytmps[2];
                  $energy[$entry[8]]=$entrytmps[2];
                  open($cmdresults,">>$ResultNodes");
                  print  $cmdresults  "$entrytmps[0]  L  $energyuseds
\n";          #เก็บผลลัพธ์ของเหตุการณ์ฟังลงไฟล์
                  close($cmdresults);
                  open($cmds,">$temps");
                  print   $cmds   "$entry[2]   $entry[0]   $entry[16]
$PktType$flowID $source $dest\n";
                  close($cmds);
                 }
                 elsif($entrytmps[1] eq "s")    #ตรวจสอบเหตุการณ์ใน TempNode
                 {
                  $cmdresults = "cmdresult$entry[8]";
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
                    $ResultNodes ="ResultNode$entry[8]";
                    $energyuseds =$entrytmps[2]-$entry[16];

                    open($cmdresults,">>$ResultNodes");
                    print  $cmdresults  "$entrytmps[0]  $entrytmps[1]
$energyuseds      $entrytmps[3]      $entrytmps[4]      $entrytmps[5]
$entrytmps[6]\n"; #เก็บผลลัพธ์ของเหตุการณ์ส่งลงไฟล์
                    close($cmdresults);
                    $energy[$entry[8]]=$energy[$entry[8]]-
$energyuseds;
                    open($cmds,">$temps");
                    print  $cmds  "$entry[2]  $entry[0]  $entry[16]
$PktType $flowID $source $dest\n";
                    close($cmds);

                   }else{       #ไม่มีค่าใน TempNode

                    open($cmds,">$temps");
                    print  $cmds  "$entry[2]  $entry[0]  $entry[16]
$PktType $flowID $source $dest\n";
                    close($cmds);

                   }
            }
      }
      elsif($entry[0] eq "N"){      #เกิดการเปลี่ยนแปลงของพลังงาน

            $previousNode[$mod]= $entry[4];
            $cmdn = "cmdtemp$entry[4]";
            $tempn ="TempNode$entry[4]";
            open($cmdn,"$tempn");
            $linen = <$cmdn>;
            @entrytmpn = split(" ", $linen);
            close($cmdn);

            @waits = split(" ", $wait);
            if($waits[1] eq "-t")   #ตรวจสอบว่ามีเหตุการณ์ที่ต้องรอการคำนวณพลังงานหรือไม่
            {

                  $cmdwait="cmdtemp$waits[4]";
                  $tempwait ="TempNode$waits[4]";
                  open($cmdwait,"$tempwait");
                  $linewait = <$cmdwait>;

                  @tempwait = split(" ", $linewait);

                  close($cmdwait);

                  if($tempwait[1] eq"s"){   #ตรวจสอบเหตุการณ์ใน TempNode
                  $energywait   =   $tempwait[2]-(($energy[$entry[4]]-
$entry[6])+$waits[6]);

                  $cmdresultwaitnode ="cmdresult$waits[4]";
                  $resultwaitnode="ResultNode$waits[4]";
                  open($cmdresultwaitnode,">>$resultwaitnode");
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
                print $cmdresultwaitnode "$tempwait[0] $tempwait[1]
$energywait $tempwait[3] $tempwait[4] $tempwait[5] $tempwait[6]\n";
#เก็บผลลัพธ์ของเหตุการณ์ส่งลงไฟล์
                close($cmdresultwaitnode);

                $energy[$waits[4]]=$energy[$waits[4]]-$energywait;

                  open($cmdwait,">$tempwait");
                print $cmdwait "$waits[2] $waits[0] $waits[6] ";
                close($cmdwait);
                $wait="";
                }
            }

            if($entrytmpn[1] eq "N"){      #ตรวจสอบเหตุการณ์ใน TempNode

                #if energy from this msg equals N from temp ->clear
temp

                  if($entry[6] eq $entrytmpn[2]){
                  $cmdn = "cmdtemp$entry[4]";
                  $tempn ="TempNode$entry[4]";
                  open($cmdn,">$tempn");
                  print cmdn "";
                  close($cmdn);
                   }
                  else{
                   $cmdresultn = "cmdresult$entry[4]";
                   $ResultNoden ="ResultNode$entry[4]";
                   $energyusedn = $energy[$entry[4]]-$entrytmpn[2];
                   $energy[$entry[4]]=$entrytmpn[2];
                   open($cmdresultn,">>$ResultNoden");
                   print $cmdresultn "$entrytmpn[0] L $energyusedn
\n";   #เก็บผลลัพธ์ของเหตุการณ์ฟังลงไฟล์
                   close($cmdresultn);

                   open($cmdn,">$tempn");
                   print   $cmdn   "$entry[2]   $entry[0]   $entry[6]
$PktType $flowID $source $dest \n";
                   close($cmdn);
                   }
            }
            elsif($entrytmpn[1] eq "s"){   #ตรวจสอบเหตุการณ์ใน TempNode
                if($mod eq 0){
                  $previous = 49; #50 Nodes
                }else{
                  $node = $mod-1;
                  $previous = $previousNode[$node];
                }

                $cmdpre = "cmdtemp$previous";
                $temppre = "TempNode$previous";

                open($cmdpre,"$temppre");
                $linepre = <$cmdpre>;
                @entrytmppre = split(" ", $linepre);
                close($cmdpre);
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
                if($entrytmppre[1] eq "N"){ #ตรวจสอบเหตุการณ์ใน TempNode ก่อนหน้า

                $energyuseds          =          $energy[$entry[4]]-
((($energy[$previous]-$entrytmppre[2]) + $entry[6]);

                $cmdresultnode ="cmdresult$entry[4]";
                $resultnode="ResultNode$entry[4]";
                open($cmdresultnode,">>$resultnode");
                print  $cmdresultnode  "$entrytmpn[0]  $entrytmpn[1]
$energyuseds    $entrytmpn[3]     $entrytmpn[4]     $entrytmpn[5]
$entrytmpn[6]\n"; #เก็บผลลัพธ์ของเหตุการณ์ฟังลงไฟล์
                close($cmdresultnode);
                $energy[$entry[4]]=$energy[$entry[4]]-$energyuseds;
                open($cmdn,">$tempn");
                print  $cmdn  "$entry[2]  $entry[0]  $entry[6]  $PktType
$flowID $source $dest\n";
                close($cmdn);

                }
                else{
                  $wait=$line;
                }
            }
            else{ #กรณีที่ไม่มีค่าใน TempNode
                open($cmdn,">$tempn");
                print  $cmdn  "$entry[2]  $entry[0]  $entry[6]  $PktType
$flowID $source $dest\n";
                close($cmdn);
            }


      } #ตรวจสอบว่าเป็นเหตุการณ์รับในระดับ MAC
      elsif(($entry[0] eq "r") && ($entry[18] eq "MAC")){

            $previousNode[$mod]= $entry[8];
            $cmdr = "cmdtemp$entry[8]";
            $tempr ="TempNode$entry[8]";
            open($cmdr,"$tempr");

            while ($liner = <$cmdr>) {
               ++$linetmpr;
               @entrytmpr = split(" ",$liner);

              if(($entrytmpr[1]     eq     "N")&&($entrytmpr[2]     eq
$entry[16])){#ตรวจสอบเหตุการณ์ใน TempNode

                $energyusedr = $energy[$entry[8]]-$entry[16];
                $energy[$entry[8]]=$entry[16];

                $cmdresultr = "cmdresult$entry[8]";
                $Tempresultr ="ResultNode$entry[8]";
                open($cmdresultr,">>$Tempresultr");
                print  $cmdresultr  "$entry[2]  $entry[0]  $energyusedr
$PktType $flowID $source $dest\n";  #เก็บผลลัพธ์ของเหตุการณ์รับลงไฟล์
                close($cmdresultr);

            }
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
            }
            close($cmdr);
            open($cmdr,">$tempr");
            print $cmdr "";
            close($cmdr);
      }
      elsif($entry[0] eq "d"){ #ตรวจสอบว่าเป็นเหตุการณ์ลบทิ้ง
            $previousNode[$mod]= $entry[8];
            $cmdd = "cmdtemp$entry[8]";
            $tempd ="TempNode$entry[8]";
            open($cmdd,"$tempd");

            while ($lined = <$cmdd>) {
            ++$linetmpd;
            @entrytmpd = split(" ",$lined);

                if(($entrytmpd[1]    eq    "N")&&($entrytmpd[2]    eq
$entry[16])){      #ตรวจสอบเหตุการณ์ใน TempNode

                $energyusedd = $energy[$entry[8]]-$entry[16];
                $energy[$entry[8]]=$entry[16];

                $cmdresultd = "cmdresult$entry[8]";
                $Tempresultd ="ResultNode$entry[8]";
                open($cmdresultd,">>$Tempresultd");
                print $cmdresultd "$entry[2] $entry[0] $energyusedd
$PktType $flowID $source $dest\n"; #เก็บผลลัพธ์ของเหตุการณ์ลบทิ้งลงไฟล์
                close($cmdresultd);

              }
            }
            close($cmdd);

            open($cmdd,">$tempd");
            print $cmdd "";
            close($cmdd);
            }

      $flowID="";
      $source="";
      $dest="";
}

for($i=0;$i<50;$i++) {
  $cmdresult ="";
  $cmdtemp ="";
  $ResultNode="";
  $TempNode="";

  $cmdresult ="cmdresult$i";
  $cmdtemp ="cmdtemp$i";
  $ResultNode ="ResultNode$i";
  $TempNode = "TempNode$i";

  open($cmdtemp,"$TempNode");
  $lastline = <$cmdtemp>;
  @lasttemp = split(" ",$lastline);
  close($cmdtemp);
```

ตารางที่ 5.3 แสดง Perl เพื่อหาพลังงานที่ใช้ในแต่ละกิจกรรม (ต่อ)

```
   open($cmdresult,">>$ResultNode");
   if($lasttemp[1] eq "N"){#ตรวจสอบเหตุการณ์ใน TempNode

   $used=$energy[$i]-$lasttemp[2];
   print $cmdresult "$lasttemp[0] L $used \n";
   $energy[$i]=$energy[$i]-$used;
   }
   elsif($lasttemp[1] eq "r"){ #ตรวจสอบเหตุการณ์ใน TempNode
     $used=$energy[$i]-$lasttemp[2];
     print $cmdresult "$lasttemp[0] $lasttemp[1] $used \n";
     $energy[$i]=$energy[$i]-$used;

   }
   elsif($lasttemp[1] eq "d"){#ตรวจสอบเหตุการณ์ใน TempNode
     $used=$energy[$i]-$lasttemp[2];
     print $cmdresult "$lasttemp[0] $lasttemp[1] $used \n";
     $energy[$i]=$energy[$i]-$used;
   }

   print $cmdresult "Remaining Energy of Node $i = $energy[$i]\n";
   close($cmdresult); #บันทึกพลังงานคงเหลือลงบรรทัดสุดท้ายของผลลัพธ์

   open($cmdtemp,">$TempNode");
   print $cmdtemp "";
   close($cmdtemp);

}

close($cmdresult);
close($cmdtemp);
close(infile);
```

ซึ่งจาก Perl สคริปดังกล่าวจะได้ผลลัพธ์ออกมาดังนี้

```
2.725855298 s 0.000182000000000015 ctrl
2.727066899 r 0.000268999999999964 DSR
2.727076899 s 0.000182000000000127 ack
2.727784500 r 0.000105999999999939 ctrl
2.727794500 s 0.000182000000000015 ctrl
2.729070100 r 0.000287999999999955 DSR
2.729080100 s 0.000182999999999933 ack
2.729514100 s 0.000393999999999894 ctrl
2.730181545 r 9.10000000000633e-05 ctrl
2.730191545 s 0.00289899999999998 cbr 1000 1 2
2.735338989 r 9.20000000000092e-05 ack
2.735639 L 0.000104999999999911
2.736002 L 9.10000000000633e-05
2.736317 L 0.000394000000000005
2.737639 L 9.09999999999522e-05
2.738605213 r 0.000106000000000005 ctrl
```

รูปที่ 5.7 ผลลัพธ์จากการประมวลผลเทรซไฟล์

โดยจะมีการเก็บเอาไว้ในไฟล์ที่ชื่อว่า ResultNode ของแต่ละโหนดว่าแต่ละโหนดเช่น เหตุการณ์ของโหนด 1 จะเก็บไว้ที่ไฟล์ ResultNode1 ว่าในการจำลองนั้นมีเหตุการณ์ใดเกิดขึ้นบ้าง เหตุการณ์นั้นใช้พลังงานไปเท่าไร และเป็นเหตุการณ์ของแพ็คเกตชนิดใด และในกรณีที่เป็นชนิด cbr จะมีการบันทึกหมายเลขประจำการไหลของข้อมูล โหนดต้นทางและโหนดปลายทางด้วย

5.8 การประมวลผลเพื่อจำแนกจุดประสงค์ของการสูญเสียพลังงาน

การสูญเสียพลังงานในระบบเครือข่ายไร้สายเฉพาะกิจนั้น แต่ละอุปกรณ์สามารถสูญเสียพลังงานเพื่อวัตถุประสงค์ของตนเอง และสามารถสูญเสียเพื่อให้กิจกรรมของโหนดอื่นดำเนินไปได้ เช่นการเป็นตัวกลางในการส่งผ่านข้อมูลไปยังปลายทาง ซึ่งจะเป็นการสูญเสียพลังงานให้ต้นทางและปลายทางดังกล่าวไม่ใช่ของตน ในส่วนนี้จะนำเสนอวิธีการในการที่จะนำมาซึ่งผลลัพธ์ เป็นการแสดงให้เห็นถึงระดับการใช้พลังงานเพื่อวัตถุประสงค์ของตนเอง หรือเพื่อกิจกรรมของผู้อื่น

โดยการที่จะสูญเสียพลังงานเพื่อจุดประสงค์ของใครนั้นสามารถดูได้จากการส่งข้อมูล ที่เป็นตัวข้อมูลทีผู้ใช้ต้องการส่งจริงๆ เช่น CBR และ TCP ซึ่งจะมีข้อมูลหมายเลขของการไหลข้อมูล (flow id) ซึ่งหมายเลขหนึ่งจะเป็นการจับคู่ระหว่างต้นทางและปลายทางหนึ่งเท่านั้น นั่นหมายความว่าเราสามารถรู้ต้นทางและปลายจริงๆของแพ็คเกตได้จากหมายเลขนี้ ซึ่งหากเป็นการสูญเสียพลังงานเพื่อวัตถุประสงค์ของตน จะเป็นการส่งเฟรมที่มีต้นทางคือตนเอง และเป็นการรับเฟรมที่มีปลายทางคือตนเองเท่านั้น นอกจากนั้นเป็นการเสียพลังงานเพื่อผู้อื่นทั้งสิ้น

ซึ่งการทดลองนี้จะทำการวิเคราะห์หาเพื่อดูระดับความแตกต่างระหว่างพลังงานที่ใช้ในกิจกรรมเพื่อกิจกรรมของตนเอง (Itself), พลังงานที่ใช้ในกิจกรรมเพื่อผู้อื่น (Give) และพลังงานรวมที่ผู้อื่นใช้ในการทำกิจกรรมให้กับตนเอง (Take) ซึ่งทั้ง 3 ส่วนนี้ สามารถเอามาเป็นปัจจัยในการมองถึงความยุติธรรมในการใช้พลังงานในเครือข่ายเฉพาะกิจได้

โดยพลังงานที่ใช้เพื่อกิจกรรมของตนเอง (Itself) นั้น จะสามารถดูได้จากการรับและส่งข้อมูล ที่มีต้นทางเป็นของตน หรือปลายทางเป็นของตนตามลำดับ ซึ่งในกรณีที่ส่งข้อมูลที่ต้นทางคือตนเอง นั่นหมายถึงว่าเป็นการส่งข้อมูลที่ตนเองมีความต้องการที่จะส่ง ขณะที่การรับข้อมูลที่ปลายทางเป็นของตนนั่นหมายความว่าเป็นการรับข้อมูลของตนเองที่ตนเองต้องการ ดังนั้นพลังงานที่ใช้เพื่อกิจกรรมของตนเองนั้นจะต้องเพียงพอกับความต้องการใช้งาน

พลังงานที่ใช้ในกิจกรรมเพื่อผู้อื่น (Give) จะดูจากการที่มีการรับและส่งข้อมูลที่ตนเองไม่ใช่ต้นทางหรือปลายทางนั้นจริงๆ นั่นหมายความว่า อุปกรณ์ที่ทำกิจกรรมในลักษณะนี้ จะทำหน้าที่เป็นตัวกลางในการส่งผ่านข้อมูลของอุปกรณ์อื่น ซึ่งก็แปลว่าเป็นการทำกิจกรรมเพื่อผู้อื่น (Give) นั่นเอง โดยการใช้พลังงานเพื่อผู้อื่นนั้น จะเป็นการใช้พลังงานเพื่ออุปกรณ์ที่เป็นต้นทางและปลายทางของข้อมูลสำหรับกิจกรรมนั้นๆ เช่น อุปกรณ์ ค. ทำการส่งข้อมูลโดยต้นทางและปลายทางจริงๆของ

ข้อมูลคืออุปกรณ์ ก. และ ข. ตามลำดับ นั่นหมายถึง อุปกรณ์ ค. ได้มีการสูญเสียพลังงานเพื่ออุปกรณ์ ก. และ ข. นั่นเอง เนื่องจากทำหน้าที่เป็นตัวกลางในการส่งข้อมูลระหว่างอุปกรณ์ทั้งสอง โดยแต่ละอุปกรณ์ไม่ควรเสียพลังงานในส่วนนี้เกินกว่าการสูญเสียพลังงานเพื่อกิจกรรมของตนเอง ซึ่งไม่ควรจะสูญเสียพลังงานเพื่อผู้อื่นจนไม่สามารถใช้พลังงานในการทำกิจกรรมเพื่อตนเองได้ หรือในกรณีใกล้เคียงกันหากอุปกรณ์ ค. ทำการรับข้อมูลที่อุปกรณ์ ก. ส่งมาเพื่อให้ส่งต่อไปที่อุปกรณ์ ข. แล้วการสูญเสียพลังงานของ ค. นั้นถือเป็นการสูญเสียให้ผู้อื่น (Give) ซึ่งอุปกรณ์ ก. และ ข. จะได้รับผลประโยชน์จากกิจกรรมนี้ (Take)

พลังงานรวมที่ผู้อื่นใช้ในการทำกิจกรรมให้กับตนเอง (Take)นั้น จะมองได้จากมีอุปกรณ์อื่นสูญเสียพลังงานเพื่อกิจกรรมของตนเองเท่าไหร่ โดยเมื่อมีอุปกรณ์ใดๆก็ตามสูญเสียพลังงานเพื่ออุปกรณ์อื่นแล้ว จะต้องมีอุปกรณ์ที่ได้รับผลประโยชน์จากการสูญเสียพลังงานในครั้งนั้น ซึ่งก็คืออุปกรณ์ต้นทางและปลายทางของกิจกรรมนั้นนั่นเอง จากกิจกรรมเมื่อตัวอย่างที่แล้ว อุปกรณ์ ก. และ ข. จะได้รับผลประโยชน์จากการสูญเสียพลังงานจากอุปกรณ์ ค. ทั้งคู่ โดยสามารถสรุปได้ว่าเมื่อมีอุปกรณ์ใดอุปกรณ์หนึ่งสูญเสียพลังงานเพื่อผู้อื่นนั้น จะมีอุปกรณ์ที่ได้รับผลประโยชน์จากการสูญเสียครั้งนั้นอย่างน้อย 2 อุปกรณ์เสมอ

5.8.1 ตัวอย่างลักษณะการสูญเสียพลังงาน

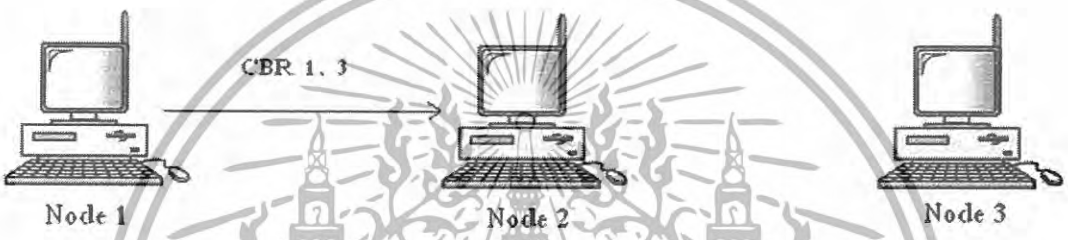
กำหนดให้มีโหนดทั้งหมดสามโหนด โดยกำหนดให้โหนดที่หนึ่งนั้นเป็นโหนดต้นทาง และกำหนดให้โหนดที่สามเป็นโหนดปลายทาง มีการรับส่งข้อมูลที่เป็น cbr ถึงกัน โดยให้โหนดที่สองเป็นตัวกลาง เพื่อศึกษาการสูญเสียพลังงานไปในกิจกรรม Itself, Give และ Take โดยกำหนดให้ทุกกิจกรรมที่เกิดขึ้นมีการสูญเสียพลังงานเท่ากันคือ 1 หน่วย

รูปที่ 5.8 แสดงเหตุการณ์ที่โหนด 1 มีการส่งข้อมูล

ตารางที่ 5.4 แสดงค่าพลังงานของแต่ละโหนดเมื่อโหนด 1 เริ่มมีการส่งข้อมูล

Node	Initial Energy	Itself	Give	Take	Remaining
Node 1	10	1	-	-	9
Node 2	10	-	-	-	10
Node 3	10	-	-	-	10

จากตารางที่ 5.4 จะเห็นได้ว่าเมื่อมีการส่งข้อมูลออกมาจากโหนด 1 ซึ่งเป็นโหนดต้นทาง โหนด 1 นั้นจะสูญเสียพลังงานในการทำกิจกรรมเพื่อตัวเอง โดยพลังงานที่โหนด 1 เสียไปนั้น จะอยู่ในรูปของ Itself

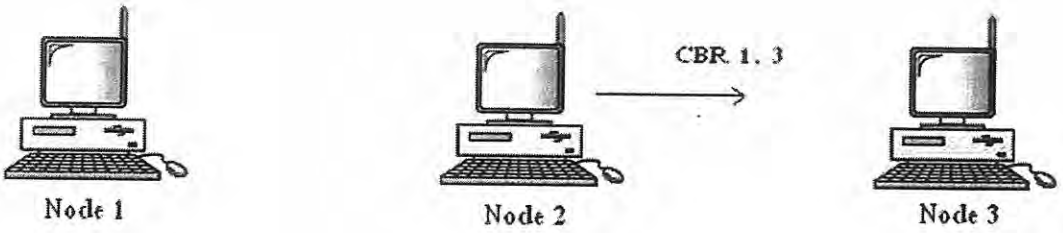


รูปที่ 5.9 แสดงเหตุการณ์เมื่อโหนดสองมีการรับข้อมูลจากโหนด 1

ตารางที่ 5.5 แสดงค่าพลังงานของแต่ละโหนดเมื่อโหนด 2 มีการรับข้อมูลจากโหนด 1

Node	Initial Energy	Itself	Give	Take	Remaining
Node 1	9	-	-	1	9
Node 2	10	-	1	-	9
Node 3	10	-	-	1	10

จากตารางที่ 5.5 จะเห็นได้ว่าเมื่อโหนด 2 มีการรับข้อมูลที่ส่งมาจากโหนด 1 แล้ว โหนดสองจะเสียพลังงานไปในกิจกรรมที่ทำเพื่อคนอื่น เนื่องจากว่าโหนดสองไม่ใช่ทั้งโหนดต้นทางและโหนดปลายทาง ซึ่งพลังงานส่วนนี้นั้นจะอยู่ในรูปของ Give แล้วเมื่อมีการเสียพลังงาน Give นั้นก็ย่อมมีโหนดที่ได้รับประโยชน์จากการ Give นี้ นั่นก็คือโหนด 1 และโหนด 3 ซึ่งพลังงาน ที่ได้รับนั้นจะอยู่ในรูปของ Take ซึ่งจะมีค่าพลังงานเท่ากับการ Give นั่นเอง



รูปที่ 5.10 แสดงเหตุการณ์เมื่อโหนด 2 มีการส่งข้อมูลไปยังโหนด 3

ตารางที่ 5.6 แสดงค่าพลังงานของแต่ละโหนดเมื่อโหนด 2 มีการส่งข้อมูลไปยังโหนด 3

Node	Initial Energy	Itself	Give	Take	Remaining
Node 1	9	-	-	1	9
Node 2	9	-	1	-	8
Node 3	10	-	-	1	10

จากตารางที่ 5.6 เมื่อโหนด 2 มีการส่งข้อมูลไปยังโหนด 3 โหนดสองจะเสียพลังงานในการส่งพลังงานเพื่อคนอื่น เนื่องจากว่าโหนด 2 ไม่ใช่ทั้งโหนดต้นทางและโหนดปลายทาง ซึ่งพลังงานที่ใช้ในการส่งออกจากโหนด 2 ไปยังโหนด 3 นั้นจะอยู่ในรูปของพลังงาน Give แล้วเมื่อมีการเสียพลังงาน Give นั้นก็ย่อมมีโหนดที่ได้รับประโยชน์จากการ Give นี้ นั่นก็คือโหนด 1 และโหนด 3 ซึ่งพลังงานที่ได้รับนั้นจะอยู่ในรูปของ Take ซึ่งจะมีค่าพลังงานเท่ากับการ Give นั่นเอง

รูปที่ 5.11 แสดงเหตุการณ์เมื่อโหนด 3 ได้รับข้อมูลจากโหนด 2

ตารางที่ 5.7 แสดงค่าพลังงานของแต่ละโหนดเมื่อโหนด 3 ได้รับข้อมูลจากโหนด 2

Node	Initial Energy	Itself	Give	Take	Remaining
Node 1	9	-	-	-	9
Node 2	8	-	-	-	8
Node 3	10	1	-	-	9

จากตารางที่ 5.7 เมื่อโหนด 3 ได้รับข้อมูล จะมีการสูญเสียพลังงานในการรับข้อมูลของตัวเอง เนื่องจากว่าโหนดสามนั้นเป็นโหนดปลายทาง พลังงานที่สูญเสียไปในการรับข้อมูลดังกล่าวจึงอยู่ในรูปของ itself นั่นเอง

จากตัวอย่างสามารถสรุปการเสียพลังงานเมื่อเกิดเหตุการณ์ ทั้งสามกรณีได้ดังตารางที่ 5.8

ตารางที่ 5.8 แสดงค่าพลังงานสรุปที่แต่ละโหนดเสียไปเมื่อโหนด 1 ส่งข้อมูลไปยังโหนด 3

Node	Initial Energy	Itself	Give	Take	Remaining
Node 1	10	1	-	2	9
Node 2	10	-	2	-	8
Node 3	10	1	-	2	9

ตารางที่ 5.9 แสดง Perl ในการวิเคราะห์วัตถุประสงค์ของการใช้พลังงานและคำนวณหาพลังงานทั้งหมดที่ใช้ในแต่ละวัตถุประสงค์

```
#!/usr/bin/perl

#### Edit here ##############################################
@area = qw(1250); #พื้นที่ที่ใช้ในการจำลอง
@nodeNo = (10);   # จำนวนโหนดที่ใช้ในการจำลอง
@con_type = qw(cbr); # ชนิดของทราฟฟิก
@max_con = qw(5); # จำนวน connection

###########################################################

for (my $ct=0;$ct<@con_type;$ct++) {
      for (my $n=0;$n<@nodeNo;$n++) {
            for (my $t=5;$t<8;$t++) {


$node="";
$src="";
$dst="";
$file="";

for(my $i=0;$i<$nodeNo[$n];$i++) {

  $cmdresult ="";
```

ตารางที่ 5.9 แสดง Perl ในการวิเคราะห์วัตถุประสงค์ของการใช้พลังงานและคำนวณหาพลังงานทั้งหมดที่ใช้ในแต่ละวัตถุประสงค์ (ต่อ)

```
 $ResultNode="";

  $cmdcbr ="";
  $resultcbr = "";

  $cmdresult
="cmdresult$i"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-t$t";
  $ResultNode
="ResultNode$i"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-t$t";

  $cmdcbr ="cmdcbr$i"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-t$t";
  $resultcbr    =    "cbr$i"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-
t$t";

open($cmdcbr,">$resultcbr") or die "couldn't open the file";




print      $cmdcbr      "#Node      $i"."      CBR      from      file
result_DSR_$area[$a]x$area[$a]_$nodeNo[$n]_$con_type[$ct]_$max_con[$n
]-t$t.tr \n";

close($cmdcbr);

#เปิดไฟล์ ResultNode ของแต่ละโหนดเพื่ออ่านข้อมูลมาใช้
open($cmdresult,"$ResultNode") or die "couldn't open the file";
      while($line = <$cmdresult>){

      @entry = split(" ", $line);

      #เช็คเงื่อนไขเพื่อแยกชนิดของข้อมูลตามวัตถุประสงค์การใช้งาน
      if($entry[0] eq "#Node"){
            $node=$entry[1];
            $file=$entry[4];
      }
         elsif($entry[3] eq "cbr"){
               $src=$entry[5];
                 $dst=$entry[6];

        open($cmdcbr,">>$resultcbr") or die "couldn't open the file";

        if(($node eq $src)||($node eq $dst)){
        #กรณีที่โหนดนั้นเป็นต้นทางหรือปลายทางจะให้เป็นข้อมูล cbr แบบ itself


            print $cmdcbr "$entry[0] $entry[1] $entry[2] $entry[3]
$entry[4] $entry[5] $entry[6] itself\n";
        }elsif(($node ne $src) && ($node ne $dst)){
        # กรณีที่โหนดนั้นไม่ได้เป็นทั้งต้นทางและปลายทางโหนดนั้นจะเป็น cbr ชนิด give ให้กับต้นทางและปลายทาง และที่ต้นทาง
และปลายทางจะมี cbr ชนิด take เกิดขึ้น


            print $cmdcbr "$entry[0] $entry[1] $entry[2] $entry[3]
$entry[4] $entry[5] $entry[6] give \n";
```

ตารางที่ 5.9 แสดง Perl ในการวิเคราะห์วัตถุประสงค์ของการใช้พลังงานและคำนวณหาพลังงานทั้งหมดที่ใช้ในแต่ละวัตถุประสงค์ (ต่อ)

```
            $takeSrc=$entry[5];
            $takeDest=$entry[6];


      $cmdtakeSrc="cmdcbr$takeSrc"."_$nodeNo[$n]_$con_type[$ct]_$max_
con[$n]-t$t";
            $resultSrc=
"cbr$takeSrc"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-t$t";



      $cmdtakeDst="cmdcbr$takeDest"."_$nodeNo[$n]_$con_type[$ct]_$max
_con[$n]-t$t";
            $resultDst=
"cbr$takeDest"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-t$t";

            open($cmdtakeSrc,">>$resultSrc");
            print   $cmdtakeSrc   "$entry[0]   $entry[1]   $entry[2]
$entry[3] $entry[4] $entry[5] $entry[6] take from $i\n";

close($cmdtakeSrc);

            open($cmdtakeDst,">>$resultDst");
              print  $cmdtakeDst  "$entry[0]  $entry[1]  $entry[2]
$entry[3] $entry[4] $entry[5] $entry[6] take from $i\n";

            close($cmdtakeDst);
              }

       }
       elsif(($entry[1]      ne      "L")&&($entry[0]      ne
"Remaining")&&($entry[0] ne "#Node")){ #ข้อมูลที่นอกเหนือการฟัง จะให้เก็บชนิดข้อมูลด้วย
            open($cmdcbr,">>$resultcbr") or die "couldn't open the
file";
            print    $cmdcbr    "$entry[0]    $entry[1]    $entry[2]
$entry[3]\n";
       }
       elsif($entry[0] eq "Remaining"){ #เก็บพลังงานคงเหลือลงไฟล์
            #print "n $node re $entry[6]\n";
            open($cmdcbr,">>$resultcbr") or die "couldn't open the
file";
            print   $cmdcbr   "Remaining   Energy   Of   Node   $i   =
$entry[6]\n";
            close($cmdcbr);

       }elsif($entry[1] eq "L"){
       # กรณีที่เป็นการฟังจะไม่สามารถระบุชนิดของข้อมูลได้ให้ใส่ชนิดว่าเป็น unknown
       open($cmdcbr,">>$resultcbr") or die "couldn't open the file";
            print $cmdcbr "$entry[0] $entry[1] $entry[2] unknown\n";
         }
        }

close($cmdcbr);
close($cmdresult);
}
```

ตารางที่ 5.9 แสดง Perl ในการวิเคราะห์วัตถุประสงค์ของการใช้พลังงานและคำนวณหาพลังงานทั้งหมดที่ใช้ในแต่ละวัตถุประสงค์ (ต่อ)

```
open(infile,">plotgraph"."DSR_$area[$a]x$area[$a]_$nodeNo[$n]_$con_ty
pe[$ct]_$max_con[$n]-t$t");
print          infile          "#conclusion          from          file
result_DSR_$area[$a]x$area[$a]_$nodeNo[$n]_$con_type[$ct]_$max_con[$n
]-t$t.tr\n";

close(infile);

for($i=0;$i<$nodeNo[$n];$i++) {

$arp=0;
$DSR=0;
$itself=0;
$give=0;
$take=0;
$ctrl=0;
$ack=0;
$unknown=0;

$enarp=0.0;
$enDSR=0.0;
$enitself=0.0;
$entake=0.0;
$engive=0.0;
$enctrl=0.0;
$enack=0.0;
$enunknown=0.0;
$remain=0.0;
  $cmdcbr ="";
  $resultcbr = "";

  $cmdcbr ="cmdcbr$i"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-t$t";
  $resultcbr =   "cbr$i"."_$nodeNo[$n]_$con_type[$ct]_$max_con[$n]-
t$t";

#อ่านไฟล์ของแต่ละโหนดเพื่อนำมาทำไฟล์สรุปรวม
open($cmdcbr,"$resultcbr") or die "couldn't open the file";
      while($line = <$cmdcbr>){

            @entry = split(" ", $line);
            #เชคเพื่อแยกชนิดของแพ็คเกต
            if($entry[3] eq "arp") {
                  ++$arp;
                  $enarp += $entry[2];
            }elsif ($entry[3] eq "DSR") {
                  ++$DSR;
                  $enDSR += $entry[2];
            }elsif ($entry[7] eq "itself"){
            ++$itself;
                  $enitself  += $entry[2];
            }elsif ($entry[7] eq "give"){
                  ++$give;
                  $engive  += $entry[2];
            }elsif ($entry[7] eq "take"){
                  ++$take;
                  $entake  += $entry[2];
```

ตารางที่ 5.9 แสดง Perl ในการวิเคราะห์วัตถุประสงค์ของการใช้พลังงานและคำนวณหาพลังงานทั้งหมดที่ใช้ในแต่ละวัตถุประสงค์ (ต่อ)

```
            }elsif ($entry[3] eq "ctrl") {
                  ++$ctrl;
                  $enctrl += $entry[2];
            }elsif ($entry[3] eq "ack") {
                  ++$ack;
                  $enack += $entry[2];
            }elsif ($entry[0] eq "Remaining"){
                  $remain = $entry[6];
            }elsif($entry[1] eq "L"){
                  ++$unknown;
                  $enunknown += $entry[2];
            }
      }
#เขียนลงไฟล์สรุปว่าแพ็คเกตของโหนดแต่ละชนิดนั้นมีจำนวนและพลังงานที่สูญเสียไปในชนิดนั้นเท่าใด
open(infile,">>plotgraph"."DSR_$area[$a]x$area[$a]_$nodeNo[$n]_$con_t
ype[$ct]_$max_con[$n]-t$t") or die "couldn't open the file";
print infile "-n $i -itself $itself -enitself $enitself -give $give -
engive $engive -take $take -entake $entake -arp $arp -enarp $enarp -
DSR $DSR -enDSR $enDSR -ctrl $ctrl -enctrl $enctrl -ack $ack -enack
$enack -unknown $unknown -enunknown $enunknown -remain $remain\n";
close(infile);
close($cmdcbr);
    }
  }
 }
}
```

```
58.950293 L 0.00145999999999447 unknown
58.955167 L 9.99999999976208e-05 unknown
58.956113017 r 0.000105000000004907 ctrl
58.956123017 s 0.000182999999992717 ctrl
58.961302415 r 0.00145899999999699 cbr 1000 0 1 itself
58.961312415 s 0.000183000000006928 ack
58.962512 L 0.000105000000004907 unknown
58.962876 L 9.99999999976208e-05 unknown
```

รูปที่ 5.12 แสดงตัวอย่างผลลัพธ์จากการวิเคราะห์วัตถุประสงค์

โดยผลลัพธ์ที่ได้มาตามรูปที่ 5.12 นั้นจะมีการเก็บเป็นไฟล์แยกของแต่ละโหนดว่าโหนดนั้นมีการใช้พลังงานไปเพื่อวัตถุประสงค์อะไรบ้าง โดยจะแยกแพ็คเกตชนิด cbr ออกตามวัตถุประสงค์เป็นสามกรณีคือ Itself, Give และ Take และเหตุการณ์ที่เป็นการฟัง นั้นไม่ได้มีการรับหรือว่าการทิ้งข้อมูล เป็นการฟังเฉยๆ จึงให้มีชนิดเป็น Unknown เนื่องจากว่าไม่ทราบ

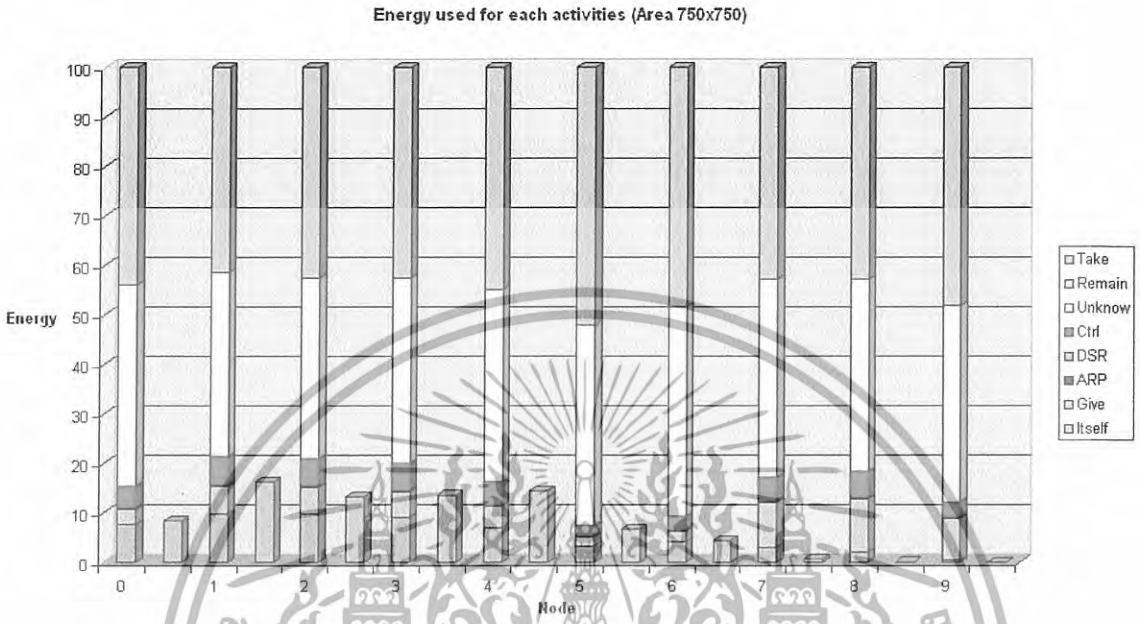
```
#ConclusionDSR_1000x1000_10_cbr_5
-n 0 -itself 24344 -enitself 38.7521030000012 -give 13506 -engive 22.704623000001
-n 1 -itself 37426 -enitself 60.1135840000011 -give 8232 -engive 12.9730370000008
-n 2 -itself 30641 -enitself 46.2783209999993 -give 6258 -engive 10.7450600000002
-n 3 -itself 38489 -enitself 69.5000160000001 -give 28928 -engive 52.4556100000005
-n 4 -itself 34218 -enitself 53.5657699999997 -give 14850 -engive 26.382828000002
-n 5 -itself 19038 -enitself 30.3553060000013 -give 32324 -engive 53.783685999999
-n 6 -itself 17255 -enitself 24.5782069999993 -give 19261 -engive 34.446859999999
-n 7 -itself 10684 -enitself 16.4031330000002 -give 17372 -engive 33.094658999998
-n 8 -itself 7523 -enitself 10.50991 -give 21493 -engive 33.072059 -take 1279 -en
-n 9 -itself 1790 -enitself 2.57112199999969 -give 28888 -engive 50.6708490000032
```

รูปที่ 5.13 แสดงผลการสรุปรวมพลังงานที่ใช้แต่ละวัตถุประสงค์ของแต่ละโหนด

ผลลัพธ์ที่ได้จะทราบว่าโหนดใด มีการใช้พลังงานไปตามวัตถุประสงค์ใด จำนวนเท่าใดโดย

–n	หมายเลขโหนด
-itself	จำนวนแพ็คเกต CBR ที่โหนดนั้นเสียพลังงานไปเพื่อกิจกรรมของตัวเอง
-enitself	พลังงานรวมที่โหนดนั้นเสียไปเพื่อกิจกรรมของตัวเอง (CBR)
-give	จำนวนแพ็คเกต CBR ที่โหนดนั้นเสียพลังงานไปเพื่อกิจกรรของโหนดอื่น
-engive	พลังงานรวมที่โหนดนั้นเสียพลังงานไปเพื่อกิจกรรมของโหนดอื่น(CBR)
-take	จำนวนแพ็คเกต CBR ของกิจกรรมที่โหนดนั้นได้รับจากกิจกรรมของโหนดอื่น
-entake	พลังงานรวมของกิจกรรมที่โหนดนั้นได้รับจากกิจกรรมของโหนดอื่น (CBR)
-arp	จำนวนแพ็คเกตของโหนดนั้นที่เกี่ยวกับ ARP Request และ ARP Reply
-enarp	พลังงานรวมที่โหนดนั้นเสียไปเพื่อกิจกรรมที่เกี่ยวกับ ARP Request และ ARP Reply
-DSR	จำนวนแพ็คเกตของโหนดนั้นที่เกี่ยวกับเราติ้งโปรโตคอล DSR
-enDSR	พลังงานรวมที่เสียไปในกิจกรรมที่เกี่ยวกับเราติ้งโปรโตคอล DSR
-ctrl	จำนวนแพ็คเกตของโหนดนั้นที่เกี่ยวกับแพ็คเกตควบคุม เช่น RTS , CTS
-enctrl	พลังงานรวมที่โหนดนั้นเสียไปในกิจกรรมที่เกี่ยวกับแพ็คเกตควบคุม
-ack	จำนวนแพ็คเกตของโหนดนั้นที่เกี่ยวกับ Acknowledgement
-enack	พลังงานรวมที่เสียไปในกิจกรรมที่เกี่ยวกับ Acknowledgement
-unknown	จำนวนแพ็คเกตทั้งหมดที่โหนดนั้นใช้ในการฟัง (ไม่ทราบชนิด)
-enunknown	พลังงานรวมที่โหนดนั้นเสียไปในการฟัง
-remain	พลังงานที่เหลือของโหนดนั้น

จากผลการวิเคราะห์วัตถุประสงค์ของการใช้พลังงานและคำนวณหาพลังงานทั้งหมดที่ใช้ในแต่ละวัตถุประสงค์ นำมาทำการสร้างแผนภูมิได้ดังนี้



รูปที่ 5.14 แสดงแผนภูมิแสดงการใช้พลังงานแยกตามวัตถุประสงค์

ซึ่งจะเห็นได้ว่ามีข้อมูลที่ไม่ใช่ข้อมูลที่ผู้ใช้ต้องการรับและส่งจริงๆมากมาย ซึ่งทุกชนิดก็ทำให้เกิดการสูญเสียพลังงานได้ทั้งสิ้น และข้อมูลแต่ละชนิดก็มีความจำเป็นต้องมีการแลกเปลี่ยนเพื่อทำให้การสื่อสารสามารถดำเนินไปได้ ดังนั้นเราจึงจำเป็นต้องนำมาพิจารณาด้วย เพื่อที่จะสามารถทราบได้ว่า พลังงานทั้งหมดที่เสียไป หมดไปกับข้อมูลชนิดไหนบ้างและปริมาณเท่าไหร่ และใช้ในการรับส่งข้อมูลจริงๆเท่าไหร่

บรรณานุกรม

คณิตสรณ์ สุริยะไพบูลย์วัฒนา. 2546. "Fundamental of TCL Programming." เอกสารอัดสำเนา.

ถิรพล วงศ์สะอาดสกุล. ม.ป.ป. "เครือข่ายเฉพาะกิจไร้สาย IEEE 802.11 IEEE 802.11 Ad Hoc Wireless Networks." [Online]. เข้าถึงได้จาก : http://www.bu.ac.th/knowledgecenter/epaper/jan_june2005/tirapol.pdf

ประวิทย์ ชุมชู. ม.ป.ป. การจำลองระบบเครือข่ายด้วย NS2 พื้นฐานสำหรับนักวิจัย. เอกสารอัดสำเนา.

ศักดิ์พงษ์ เสรีเศวตรัตน์. 2549. "Instruction : Setup NS-2 version 2.29.3." [Online]. เข้าถึงได้จาก : http://elearning.it.kmitl.ac.th/mod/forum/discuss.php?d=779.

Aeil Ahn and Woojik Chun. n.d. "Overview of MPLS Network Simulator : Design And Implement." [Online]. Available : http://heim.ifi.uio.no/johanmp/mpls.html

Ahmed Safwat and Hossam Hassanein. ET.AL . 2002. "Energy-Aware Routing in MANETs: Analysis and Enhancements." Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems. 2002. 46 – 53.

Bor-rong Chen and C. Hwa Chang. n.d. "Mobility Impact on Energy Conservation of Ad Hoc Routing Protocols." [Online]. Available :http://www.ece.tufts.edu/~hchang/Projects2003/SSGRR03_brchen.pdf

Brent B. Welch and Ken Jones. ET.AL . 2003. **Practical Programming in Tcl and Tk, Fourth Edition.** Prentice Hall PTR.

Eric W. Gray. 2001. **MPLS Architecture.** [Online]. Available : http://www.informit.com/articles/article.asp?p=167856&rl=1

Information Sciences Institute of University of Southern California. 2006. **The Network Simulator - ns – 2 .** [Online]. Available : http://www.isi.edu/nsnam/ns/.

Jim Guichard and Ivan Pepelnjak. 2000. **MPLS and VPN Architectures. :** USA. Cisco Press

Jung Hyon Jun and Young-June Choi. ET.AL. n.d. "Affinity-Based Power Saving MAC Protocol in Ad Hoc Networks" 2005. Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on. 2005. 363- 372.

Kevin Fall and Kannan Vardhan. 2006. **The *ns* Manual (formerly *ns* Notes and Documentation).** n.p.

Stephan A. Thomas. 2001 **IP Switching and Routing Essentials.** : USA. Wiley.

The International Engineering Consortium. n.d. "Multiprotocol Label Switching." [Online]. Available : http://iec.org

Vivek Alwayn. 2001 "Advanced MPLS Design and Implementation." n.p.

WANG Kun and XU Yin-long. ET.AL. n.d. "Power-Aware On-Demand Routing Protocol for MANET." [Online].
Available : http://ieeexplore.ieee.org/iel5/9027/28651/01284112.pdf?arnumber=1284112

ภาคผนวก

ภาคผนวก ก.

การใช้งานภาษา TCL เบื้องต้น

ภาษา TCL (Tool Command Language) เป็นภาษาที่มีวยากรณ์ที่ผู้ใช้สามารถเข้าใจได้ง่ายและสามารถใช้งานได้กับภาษาอื่นๆ และสามารถใช้งานได้อย่างแพร่หลายอาทิเช่น สามารถใช้เขียนเวปแอพพลิเคชั่น (Web application) หรือ เดสทอปแอพพลิเคชั่น (desktop application) ใช้งานทางด้านเน็ตเวิร์ค (network) ด้านผู้ดูแลระบบ เป็นต้น

โดยภาษา TCL มีลักษณะเด่นดังนี้

- เป็นภาษาที่พัฒนาได้ง่ายและมีความรวดเร็วในการพัฒนา
- สามารถใช้งานเกี่ยวกับกราฟิกอินเตอร์เฟส (Graphic Interface) ได้
- สามารถใช้งานได้หลากหลายแพทฟอร์ม (platform)
- เป็นภาษาที่ง่ายต่อการรวมเข้ากับภาษาอื่นๆและง่ายต่อการนำไปใช้งาน
- เป็นภาษาที่ใช้งานง่าย
- สามารถใช้งานได้ฟรี (free) ซึ่งปัจจุบันมีโอเพนต์ซอสต์ (open source) มากมายให้ใช้งาน

การใช้งานคำสั่ง *puts*

ในภาษา TCL นั้นจะใช้คำสั่ง ***puts*** ในการแสดงผลออกทางหน้าจอ ซึ่งเหมือนกับคำสั่ง ***printf*** ในภาษา C หรือ คำสั่ง ***system.out.print*** ในภาษา Java ซึ่งการใช้งานคำสั่ง ***puts*** นั้นจะมีลักษณะการใช้งานดังนี้

ตัวอย่างที่ 1

```
puts "Hello World";          # ผลลัพธ์ Hello World
puts "Hello World"           # ผลลัพธ์ Hello World
puts  Hello World            # ผลลัพธ์ Hello World
```

จากตัวอย่างที่ 1 ทั้ง 3 คำสั่งจะแสดงข้อความออกทางหน้าจอว่า Hello World ซึ่งจะอยู่คนละบรรทัดกัน โดยสามารถใส่เซมิโคลอน (Semicolon (;)) หรือไม่ใส่ก็ได้ หรือจะไม่ใส่เครื่องหมายฟันหนู (Double quotes (" ")) ก็ได้ ซึ่งจะได้ผลลัพธ์เหมือนกัน โดยคำสั่ง ***puts*** นั้นจะขึ้นบรรทัดใหม่ให้และสามารถใช้เครื่องหมาย ; ได้ในเครื่องหมายฟันหนู (" ") ด้วย ดังตัวอย่างที่2

ตัวอย่างที่ 2

```
puts "Hello World; - semicolon inside the quotes
# ผลลัพธ์ Hello World ; - semicolon inside the quotes
```

คำสั่ง ***puts*** จะใช้ควบคู่กับเครื่องหมาย {} และ " " ซึ่งทั้งสองเครื่องหมายนี้จะแสดงผลต่างกันดังนี้

ตัวอย่างที่ 3

```
set a 5                    # เป็นการกำหนดค่าให้กับตัวแปร a ซึ่งให้มีค่าเท่ากับ 5
puts {$a}                  # การใช้ puts คู่กับ {} ผลลัพธ์คือ $a
puts "$a"                  # การใช้ puts คู่กับ " "   ผลลัพธ์คือ 5
```

ในตัวอย่างที่ 3 จะสังเกตว่าการใช้งานคำสั่ง ***puts*** ร่วมกับเครื่องหมายทั้งสองนั้นจะทำให้แสดงผลต่างกัน โดยการใช้คู่กับเครื่องหมาย {} นั้นจะเหมือนกับข้างใน {} เป็นตัวอักษรซึ่งจะได้ผลออกมาตามที่ได้ใส่เข้าไป ส่วนเครื่องหมาย " " นั้นจะเป็นการแสดงค่าจากการที่ได้กำหนดค่าไว้

การใช้งานคำสั่ง *set*

คำสั่ง ***set*** เป็นคำสั่งที่ใช้สำหรับกำหนดค่าให้กับตัวแปร ที่ต้องการ ซึ่งมีรูปแบบดังนี้

```
set ชื่อตัวแปร ค่าที่กำหนดให้ตัวแปร
```

ตัวอย่างที่ 1

```
set ns "network_simulator"  # กำหนดค่า ns ให้เก็บคำว่า network_simulator
set ns network_simulator    # กำหนดค่า ns ให้เก็บคำว่า network_simulator
```

จากตัวอย่างที่ 1 เป็นการถ้าทำการใช้คำสั่ง puts เพื่อแสดงผลของตัวแปร ns ออกมาจะได้คำว่า network simulator ซึ่งคำสั่ง set คำสั่งแรกกับคำสั่งที่สองจะให้ค่าเหมือนกัน ซึ่งแสดงว่าสามารถไม่ใส่เครื่องหมาย " " ก็ได้ แต่ต้องเป็นคำที่ติดกัน ถ้าใช้ ***set*** ในคำสั่งที่สองนั้นมีช่องว่างระหว่างค่าที่กำหนดให้ตัวแปร ns จะทำให้โปแกรมเกิดข้อผิดพลาด (error) ได้ ดังนี้

ตัวอย่างที่ 2

```
set ns network simulator    # เกิดข้อผิดพลาดเพราะมีช่องว่างระหว่างค่าที่กำหนด
```

การจะนำตัวแปรที่ได้กำหนดค่าไปใช้งานนั้นจะต้องใช้เครื่องหมาย $ นำหน้าตัวแปรที่ได้กำหนดไว้ ซึ่งมีลักษณะการใช้งานดังนี้

ตัวอย่างที่ 3

```
set a "Hello"          #  กำหนดค่าให้ตัวแปรเก็บคำว่า Hello
puts "$a"              # ผลลัพธ์ Hello
puts a                 # ผลลัพธ์ a
```

จากตัวอย่างได้มีการกำหนดค่า Hello ให้กับตัวแปร a ซึ่งในการที่จะทำให้แสดงค่าของตัวแปร a ออกมานั้นต้องใช้เครื่องหมาย $ นำหน้าตัวแปร a ดังคำสั่ง ***puts*** คำสั่งแรก ส่วนคำสั่งที่สองนั้นจะได้ค่าออกมาเหมือนกับการแสดงตัวอักษร a ออกทางหน้าจอเท่านั้น

การใช้งานคำสั่ง *expr*

คำสั่ง ***expr*** มีไว้ใช้ในการนำค่าตัวแปรมาทำกระบวนการทางคณิตศาสตร์ (Arithmetic operator) โดยมีแบบคำสั่งดังนี้

```
expr ตัวแปรที่1(+ - * /) ตัวแปรที่ 2 .....ตัวแปรที่ n
```

ตัวอย่างที่ 1

```
set a 21               # กำหนดให้ตัวแปร a มีค่าเท่ากับ 21
set b 7                # กำหนดให้ตัวแปร b มีค่าเท่ากับ 7
puts [expr $a/$b]      # ผลลัพธ์ 3
puts $a/$b             # ผลลัพธ์ 21/7
```

จากตัวอย่างที่ 1 จะสังเกตเห็นว่าในคำสั่ง ***puts*** คำสั่งแรกนั้นจะเป็นการดำเนินการทางคณิตศาสตร์ด้วยคำสั่ง ***expr*** โดยการนำค่าตัวแปร b มาหารด้วยค่าของตัวแปร a ซึ่งค่าที่ได้ก็จะเป็น 3 ส่วนในคำสั่ง ***puts*** คำสั่งที่สองนั้นจะเป็นการแสดงผลออกทางหน้าจอเฉยๆ โดยไม่ได้มีการทำ

กระบวนการทางคณิตศาสตร์ เพราะฉะนั้นเวลาจะทำกระบวนการทางคณิตศาสตร์กับตัวแปรใดๆ ต้องใช้คำสั่ง ***expr***

โดยในกระบวนการทางคณิตศาสตร์นั้นจะคำนึงถึงลำดับความสำคัญของเครื่องหมายทางคณิตศาสตร์ว่าตัวดำเนินการตัวไหนจะทำก่อน ซึ่งแสดงในตารางที่ 1 ดังนี้

ตาราง แสดงลำดับความสำคัญของตัวดำเนินการทางคณิตศาสตร์ในภาษา TCL

- ~ !	Unary minus, bitwise NOT, logical NOT.
* / %	คูณ,หาร,หารเอาเศษ
+ -	บวก,ลบ
<< >>	เลื่อนด้านซ้าย(Left shift), เลื่อนด้านขวา(right shift)
< > <= >=	เครื่องหมายเปรียบเทียบ: น้อยกว่า, มากกว่า, น้อยกว่าเท่ากับ, มากกว่าเท่ากับ หรือเท่ากับ
== != eq ne	เท่ากับ, ไม่เท่ากับ, สตริงเท่ากับ (Tcl 8.4), สตริง ไม่เท่ากับ (not equal) (Tcl 8.4).
&	ตัวดำเนินการแบบบิต AND.
^	ตัวดำเนินการแบบบิต XOR.
|	ตัวดำเนินการแบบบิต OR.
&&	Logical AND.
||	Logical OR.

จากตารางแสดงลำดับความสำคัญของตัวดำเนินการทางคณิตศาสตร์นั้นจะสามารถแสดงได้ดังตัวอย่างที่ 2 ดังนี้

ตัวอย่างที่ 2

```
set a 21            # กำหนดให้ตัวแปร a มีค่าเท่ากับ 21
set b 4             # กำหนดให้ตัวแปร b มีค่าเท่ากับ 4
set c 3             # กำหนดให้ตัวแปร c มีค่าเท่ากับ 3
set d 2             # กำหนดให้ตัวแปร d มีค่าเท่ากับ 2
puts [expr $a-$b+$c*$d]                 # ผลลัพธ์เท่ากับ 23
puts [expr ($a-$b+$c)*$d]               # ผลลัพธ์เท่ากับ 40
```

จากตัวอย่างที่ 2 จะสังเกตเห็นว่าจะทำการคูณระหว่างตัวแปร c ที่มีค่าเท่ากับ 3 กับตัวแปร d มีค่าเท่ากับ 2 ก่อน เพราะว่าการคูณนั้นมีลำดับความสำคัญมากกว่าการบวกและการลบ โดยจะได้ผลลัพธ์เท่ากับ 6 จากนั้นจะทำการลบก่อน เพราะว่าการบวกและการลบนั้นมีลำดับความสำคัญเท่ากัน โดยการนำค่าตัวแปร a ไปลบกับค่าของตัวแปร b ซึ่งจะได้ผลลัพธ์เท่ากับ 17 และจากนั้นจึงนำผลลัพธ์ที่ได้มาบวกกันซึ่งผลที่ได้จะมีค่าเท่ากับ 23 ส่วนใน ***puts*** คำสั่งที่สองนั้นจำทำในเครื่องหมายวงเล็บก่อนโดยจะนำค่าตัวแปร a มาลบกับตัวแปร b ก่อนแล้วเอามาบวกกับตัวแปร a ซึ่งจะได้ผลลัพธ์คือ 20 จากนั้นก็เอา 20 ไปคูณกับค่าตัวแปรของ d ซึ่งมีค่าคือ 2 ซึ่งจะได้ผลลัพธ์เท่ากับ 40

ถ้าลำดับความสำคัญของตัวแปรเท่ากันจะมีการทำโดยเรียงลำดับการมาก่อนหลังของสมการดังตัวอย่างที่ 2 นั้นจะเห็นว่าการบวกและการลบนั้นมีลำดับความสำคัญเท่ากัน จึงทำการลบก่อนเพราะว่าการลบนั้นมาก่อนการบวก และถ้าในสมาการนั้นมีเครื่องหมาย ( ) ก็จะทำในเครื่องหมาย ( ) ก่อนโดยการใช้คำสั่ง ***expr*** นั้นยังสามารถดำเนินการทางด้านค่าความจริง (Boolean) ได้ด้วยซึ่งจะแสดงในตัวอย่างที่ 3 ดังนี้

ตัวอย่างที่ 3

```
set a 5                  # กำหนดค่าให้กับตัวแปร a มีค่าเท่ากับ 5
set b 4                  # กำหนดค่าให้กับตัวแปร b มีค่าเท่ากับ 4
puts [expr $a>$b]        # ผลลัพธ์เท่ากับ 1 ซึ่ง 1 ก็คือจริง (True)
puts [expr $a<$b]        # ผลลัพธ์เท่ากับ 0 ซึ่ง 0 ก็คือเท็จ (False)
```

จากตัวอย่างที่ 3 จะเห็นว่าได้มีการเปรียบเทียบค่าของตัวแปร a ซึ่งมีค่าเท่ากับ 5 กับตัวแปร b ซึ่งมีค่าเท่ากับ 4 โดยคำสั่ง ***puts*** คำสั่งแรกนั้นจะเป็นการเปรียบเทียบว่า 5 > 4 ซึ่งผลที่ได้จะออกมาเป็นค่าความจริงคือ 1 ซึ่งมีค่าเท่ากับจริง ส่วนคำสั่งที่สองนั้นจะเป็นการเปรียบเทียบว่า 4 > 5 ซึ่งผลที่ได้จะออกมาเป็นค่าความจริงคือ 0 ซึ่งมีค่าเท่ากับเท็จ

การใช้เครื่องหมาย \ (Backslash)

ในภาษา TCL จะมีเครื่องหมาย \ ที่เป็นคำสั่งพิเศษ ซึ่งแสดงได้ดังตารางที่ 2 ดังนี้

ตาราง แสดงการใช้งานเครื่องหมาย \ (Backslash) ในภาษา TCL

ตัวอักษร (String)	เอาท์พุต (output)	(เลขฐานสิบหก) Hex Value
\b	Backspace	0x08
\f	เคลียหน้าจอ (clear screen)	0x0c
\n	ขึ้นบรรทัดใหม่ (New Line)	0x0a
\t	แทป (Tab)	0x09
\v	แทปแนวตั้ง (Vertical Tab)	0x0b
\0dd	เลขฐานแปด (Octal Value)	d เป็นเลข 1-7
\xhh	เลขฐานสิบหก	h เป็นเลข 0-9,A-F,a-f

โดยคำสั่งเครื่องหมาย \ นี้สามารถใช้ได้กับเครื่องหมายฟันหนู " " ภายใต้คำสั่ง ***puts*** เท่านั้น

ตัวอย่างที่ 1

```
set name Hello                    # กำหนดค่าให้ตัวแปร name คือ Hello
set surname World                 # กำหนดค่าให้ตัวแปร surname คือ World
puts "My name: $name \n Surname: $surname" # แสดงค่าออกทางหน้าจอ
```

ผลลัพธ์

```
My name: Hello
Surname: World
```

จะเห็นว่าได้มีการใส่เครื่องหมาย \n เข้าไปในประโยคโดยประโยคที่อยู่ต่อจาก \n จะทำการขึ้นบรรทัดใหม่ นอกจากนี้ \ ยังสามารถใช้ในการแสดงอักขระพิเศษที่อาจจะซ้ำกับคำสงวน (reserve word) ในภาษา TCL ได้ อย่างเช่นเราต้องการแสดงผล $100 ในข้อความ จะทำได้อย่างไร

ตัวอย่างที่ 2

```
set a 100.00                    # กำหนดให้ตัวแปร a มีค่าเท่ากับ 100
puts "I've money $a"            # ผลที่ได้คือ I've money 100.00
puts "I've money $$a"           # ผลที่ได้คือ I've money $100.00
puts "I've money \$a"           # ผลที่ได้คือ I've money $a
puts "I've money \$$a"          # ผลที่ได้คือ I've money $100.00
```

การจัดกลุ่มตัวแปรโดยใช้เครื่องหมาย { } (Braces)

การใช้งานเครื่องหมาย { } ในการแสดงผลกับคำสั่ง puts จะมีความแตกต่างจากการใช้เครื่องหมายฟันหนู " " คือถ้าใช้เครื่องหมาย { } จะทำให้ข้อความที่อยู่ในเครื่องหมาย { } ถูกแสดงผลออกมาเป็นเหมือนสตริง (String) ธรรมดา ซึ่งคำสั่งต่างๆ หรือ ตัวแปรจะไม่มีการทำงานใน { } ดังตัวอย่างที่ 1

ตัวอย่างที่ 1

```
set a 5          # เป็นการกำหนดค่าให้กับตัวแปร a ซึ่งให้มีค่าเท่ากับ 5
puts {$a}        # การใช้puts คู่กับ {}      ผลลัพธ์ $a
puts "$a"        # การใช้puts คู่กับ " "  ผลลัพธ์ 5
```

การจัดกลุ่มตัวแปรโดยใช้เครื่องหมาย [ ] (Square Brackets)

เครื่องหมาย [ ] จะถูกมองว่ามีคำสั่งอยู่ภายใน ซึ่งตัวแปลภาษา (interpreter) จะทำคำสั่งภายใน [ ] และส่งผลลัพธ์การทำงานออกมาแทนที่ [ ] ในรูปแบบสตริง โดยจะมีรูปแบบการใช้งานดังนี้

```
puts [readsensor [selectsensor]]
```

จากรูปแบบการทำงาน จะมองว่า readsensor และ selectsensor เป็นคำสั่งที่ต้องการทำงาน โดยจะมีการทำงานจากตัว selectsensor ก่อนแล้วจะส่งผลลัพธ์ไปเป็นอินพุต (input) ให้กับตัว readsensor ต่อและเมื่อทำคำสั่งใน readsensor เสร็จก็จะนำผลลัพธ์ที่ได้ไปเป็นอินพุตของ puts ต่อไป ซึ่งจะแสดงได้ในตัวอย่างที่ 1 ดังนี้

ตัวอย่างที่ 1

```
set a Hello              # กำหนดให้ตัวแปร a เก็บคำว่า Hello
puts $a                  # แสดงค่าของตัวแปร a ซึ่งผลลัพธ์ที่ได้ก็คือ Hello
set b [set a Hi]         # กำหนดค่าให้ a เก็บคำว่า Hi แล้วจะไปเป็นอินพุตให้ b
                         # ซึ่ง b ก็ถูกกำหนดค่าให้เก็บคำว่า Hi ด้วย
puts $a                  # ผลลัพธ์ Hi
puts $b                  # ผลลัพธ์ Hi
```

ขอบเขตของตัวแปรในภาษา TCL

สามารถแบ่งออกได้เป็น 2 ประเภทดังนี้

1. โลคอล (local) เป็นตัวแปรที่มองเห็นได้แค่ในขอบเขตหนึ่งๆ
2. โกลบอล (global) เป็นตัวแปรที่ถูกกำหนดไว้แต่แรกจึงทำให้มองเห็นตัวแปรได้ในทุกส่วนของโปรแกรม

ตัวอย่างที่ 1

```
set x 7                  # ตัวแปรโกลบอล
proc sum {a b} {
     set x 5             # ตัวแปรโลคอล
     puts "Variable in side Procedure is: $x"
     print 5
}
proc print {a} {
     puts "Variable in Procedure print is: $a"
}
sum 7 8
puts "Variable out side Procedure is $x"
```

ผลลัพธ์

```
Variable in side Procedure is: 5
Variable in Procedure print is: 5
Variable out side Procedure is: 7
```

จากตัวอย่างที่ 1 นั้นจะสังเกตเห็นว่าถ้ามีการประกาศตัวแปรไว้ในส่วนเริ่มของโปรแกรมเลยจะเป็นการประกาศตัวแปรแบบโกลบอลคือตัวแปรนี้จะมองเห็นได้ในทุกส่วนของโปรแกรม ส่วนตัวแปรโลคอลนั้นจะมองเห็นได้ในแค่ขอบเขตหนึ่งซึ่งจากตัวอย่างก็จะมองเห็นได้แค่ภายในโพซีเยอร์

การสร้างโพรซีเยอร์ (Procedure) ในภาษา TCL

ในภาษา TCL จะใช้คำสั่ง proc ในการสร้างโพรซีเยอร์ถ้าเทียบกับภาษาอื่นก็คือฟังก์ชัน (function) นั่นเอง โดยเมื่อประกาศโพรซีเยอร์แล้วจะสามารถอ้างถึงโพรซีเยอร์นั้นๆ ด้วยชื่อโพรซีเยอร์ได้เลย พร้อมทั้งสามารถส่งอาร์กูเมนท์ (Argument) ต่างๆไปด้วยได้ โดยการสร้างโพรซีเยอร์มีรูปแบบดังนี้

```
proc arglist body
```

ตัวอย่างที่ 1

```
proc Cal {a b} {
   set c [expr {sqrt($a * $a + $b * $b)}]
   return $c
}
puts "The diagonal of a 3, 4 is [Cal 3 4]"
# ผลลัพธ์คือ The diagonal of a 3, 4 right triangle is 5.0
puts "The diagonal of a 3, 4 is [cal 3 4]"
# เกิดข้อผิดพลาดเพราะเรียกชื่อโพรซีเยอร์ผิด
puts "The diagonal of a 3, 4 is [Cal 3]"
# เกิดข้อผิดพลาดเพราะส่งอาร์กูเมนท์ ไปไม่ครบ
```

จากตัวอย่างที่ 1 ได้มีการประกาศ ***proc*** ชื่อว่า Cal ที่รับอาร์กูเมนท์ สองตัวคือ a กับ b จากนั้นจะกำหนดให้ตัวแปร c นั้นทำการเก็บค่าของการทำการถอดรากที่สอง (square root) ของค่าที่รับมาจากอาร์กูเมนท์ a*a แล้วไปบวกกับค่าที่รับมาจากอาร์กูเมนท์ b*b แล้วเอามาบวกกัน และจากนั้นคืนค่า c ออกมา โดยในคำสั่ง puts เราได้มีการเรียกใช้โพรซีเยอร์ Cal โดยเราส่งพารามิเตอร์ (parameter) 3 กับ 4 เข้าไป โดย 3 จะเข้าไปที่อาร์กูเมนท์ a และ 4 จะเข้าไปที่อาร์กูเมนท์ b แล้วจากนั้นมาทำกระบวนการถอดรากที่สองของ (3*3)+(4*4) ซึ่งจะได้ 5 โดยจะเก็บค่า 5 ไว้ที่ตัวแปร c แล้วจะส่งค่ากลับ (return) ตัวแปร c ซึ่งมีค่าเท่ากับ 5 ออกมาจึงได้ผลลัพธ์ออกมาคือ The diagonal of a 3, is 5.0

จะสังเกตเห็นว่าชื่อโพรซีเยอร์นั้นตัวอักษรเล็กและใหญ่จะมีผลโดยต้องเรียกชื่อ ***proc*** ให้ถูกต้องถึงจะมีการทำงานตามโพรซีเยอร์ ซึ่งเป็นตาม puts คำสั่งที่สอง จะเห็นว่าได้ทำการเรียก ***proc*** cal ซึ่งไม่ตรงกับ ***proc*** ที่ประกาศไว้ คือ Cal จึงทำให้เกิดข้อผิดพลาดขึ้น

และในการส่งพารามิเตอร์ เข้าไปในโพรซีเยอร์นั้น จะต้องส่งไปให้ครบกับจำนวนอาร์กูเมนท์ที่ประกาศไว้ในโพรซีเยอร์ซึ่งในตัวอย่างที่ 1 ตรงส่วน ***puts*** คำสั่งที่สามนั้นจะเห็นว่าเกิด ข้อผิดพลาดขึ้นเพราะว่าเราได้ส่งพารามิเตอร์ ไปไม่ครบจำนวนของอาร์กูเมนท์ที่รับจึงทำให้เกิดข้อผิดพลาดขึ้น โดยจะมีการแก้ปัญหาการส่งพารามิเตอร์ไม่ครบกับจำนวนอาร์กูเมนท์ โดยมีวิธีแก้ดังตัวอย่างที่ 2 ดังนี้

ตัวอย่างที่ 2

```
proc Cal {a {b 4}} {
   set c [expr {sqrt($a * $a + $b * $b)}]
   return $c
}
puts "The diagonal of a 3, 4 is [Cal 3]"
# ผลลัพธ์คือ The diagonal of a 3, 4 is 5.0
puts "The diagonal of a 3, 4 is [Cal 3 8]"
# ผลลัพธ์คือ The diagonal of a 3, 4 is 8.54
```

จากตัวอย่างที่ 2 จะสังเกตเห็นว่าเราได้มีการกำหนดค่าดีฟอลท์ (default) ให้กับอาร์กูเมนท์ b มีค่าเป็น 4 คือถ้าเวลาเราเรียกโพรซีเยอร์ โดยไม่ได้ส่งพารามิเตอร์ ไปให้อาร์กูเมนท์ b ก็จะทำให้โปรแกรมไม่เกิดการข้อผิดพลาดโดยโปรแกรมจะกำหนดค่าดีฟอลท์ให้ตัวแปร b เป็น 4 เลยดังคำสั่ง ***puts*** คำสั่งแรกเราได้ทำการเรียกโพรซีเยอร์ Cal โดยการส่งพารามิเตอร์ คือ 3 ไปแค่ตัวเดียว ซึ่งผลลัพธ์ก็จะได้ค่าคือ 5.0 โดยสาเหตุที่โปรแกรมไม่มีการผิดพลาด เนื่องจากการส่งพารามิเตอร์ไปไม่ครบเหมือนดังตัวอย่างที่ 1 นั้นเพราะว่าเราได้ทำการกำหนดค่าดีฟอลท์ ไว้ให้กับตัวแปร b คือ 4 ไว้ จึงทำให้เราส่งไปเพียงแค่พารามิเตอร์ตัวเดียวได้โดยที่โปรแกรมจะไม่เกิดข้อผิดพลาดแต่ถ้าเราส่งพารามิเตอร์ ไปทั้งสองตัวทั้งๆที่เราได้กำหนดค่าดีฟอลท์ไว้แล้วดังคำสั่ง puts คำสั่งที่สองนั้นตัวโปรแกรมจะเป็นการส่งค่า 3 ไปอาร์กูเมนท์ a และส่งค่า 8 ไปอาร์กูเมนท์ b จึงทำให้ได้ผลลัพธ์เป็น The diagonal of a 3, 4 is 8.54

นอกจากนี้ยังมีข้อกำหนดให้มีตัวแปรอาร์กูเมนท์พิเศษที่ช่วยในการรับพารามิเตอร์ ที่ส่งมามากกว่าตัวอาร์กูเมนท์ ที่ได้ประกาศไว้ใน โพรซีเยอร์โดยจะใช้ คำว่า ***args*** ซึ่งพารามิเตอร์ที่เกินอาร์กูเมนท์ของโพรซีเยอร์ทั้งหมดจะถูกเก็บอยู่ในนี้ โดย ***args*** จะมีค่าดีฟอลท์เป็นสตริงเปล่าๆ โดยถ้าส่งค่ามาไม่ครบโปรแกรมก็จะไม่เกิดข้อผิดพลาด ซึ่งแสดงได้ดังตัวอย่างที่ 3 ดังนี้

ตัวอย่างที่ 3

```
proc test {a {b ""} args} {
   puts "a is: $a"
   puts "b is: $b"
   puts "args is: $args"
}
test 1 2 3 4 5 6        # เรียกใช้ proc test โดยส่งพารามิเตอร์ 1 2 3 4 5 6 เข้าไป
test 1 2                # เรียกใช้ proc test โดยส่งพารามิเตอร์ 1 2 เข้าไป
```

ผลลัพธ์

```
a is: 1                          # ผลลัพธ์จากการเรียก test คำสั่งแรก
b is: 2                          # ผลลัพธ์จากการเรียก test คำสั่งแรก
args is: 3 4 5 6                 # ผลลัพธ์จากการเรียก test คำสั่งแรก
a is: 1                          # ผลลัพธ์จากการเรียก test คำสั่งที่สอง
b is: 2                          # ผลลัพธ์จากการเรียก test คำสั่งที่สอง
args is:                         # ผลลัพธ์จากการเรียก test คำสั่งที่สอง
```

จากตัวอย่างที่ 3 จะได้ผลลัพธ์ดังนี้ในการเรียกใช้ ***proc*** test ครั้งแรกนั้นเลข 1 จะถูกส่งไปที่ a เลข 2 จะถูกส่งไปที่ b และเลข 3 4 5 6 จะถูกส่งไปที่ args และจะแสดงผลออกมาโดย a จะแสดงเลข 1 b จะแสดงเลข 2 และ ***args*** จะแสดงเลข 3 4 5 6 ส่วนในการเรียกเรียกใช้ procedure test ครั้งที่สองนั้นมีการส่งพารามิเตอร์ที่มีค่าเป็น 1 ไปที่ a และ 2 ไปที่ b โดยไม่มีการส่งไปที่ ***args*** เลยซึ่งผลที่ได้จะแสดงค่า 1 กับ 2 และ empty string ออกมาโดยที่ไม่เกิดข้อผิดพลาดเพราะว่า ***args*** นั้นมีค่าดีฟอลท์เป็นสตริงเปล่าๆ

คำสั่งควบคุม (Control Structure Commands)

การใช้ *if-then-else*

ในภาษา TCL มีรูปแบบการใช้งาน if ดังนี้

```
if expression ?then? body1 ?else? ?body2?
```

ตัวอย่างที่ 1

```
set x 1
set y 5
if {$x == 0} {
   puts stderr "Divide by zero!"
} else {
   set slope [expr $y/$x]
   puts $slope
}
```

จากตัวอย่างจะได้ผลลัพธ์เป็น 5 โดยในเงื่อนไขของ ***if*** ถ้าค่า x เป็น 0 จะแสดงข้อความออกมาว่า divide by zero! แต่ถ้าค่า x ไม่ใช่ 0 ก็จะไปเข้าเงื่อนไขของ ***else*** แล้วจึงทำตามคำสั่งใน body ของ ***else*** คือการนำค่าของ y มาหารด้วยค่าของ x แล้วเก็บในตัวแปร slope แล้วแสดงผลตัวแปร slope ออกมาทางหน้าจอ

การใช้ *switch*

ในภาษา TCL มีรูปแบบการใช้งาน ***switch*** ได้ 2 แบบ ดังนี้

```
switch value pat1 body1 pat2 body2 ...
```

และ

```
switch  value { pat1 body1 pat2 body2 ... }
```

โดย value จะคือค่าตัวแปรที่รับเข้ามาแล้วเอามาเปรียบเทียบกับ pat ที่ได้มีการกำหนดไว้ หาก value มีค่าตรงกับ pat อันใด ก็จะทำงานในส่วนของตัวโปรแกรม (body) ของ pat นั้น

ตัวอย่างที่ 1

```
set x "ONE"
set y 1
set z "ONE"
switch $x {
      "$z" {      # ค่า $z นี้จะถูกมองว่าเป็นสตริงคำว่า $z จึงไม่ถูกทำงานในส่วนนี้
      set y1 [expr $y+1]
           puts "MATCH \$z. $y + $z is $y1"
       }
       ONE {
           set y1 [expr $y+1]
           puts "MATCH ONE. $y + one is $y1"
       }
       TWO {
           set y1 [expr $y+2]
           puts "MATCH TWO. $y + two is $y1"
      }
       THREE {
           set y1 [expr $y+3]
           puts "MATCH THREE. $y + three is $y1"
       }
       default {
           puts "$x is NOT A MATCH"
       }
      }
```

จากตัวอย่างที่ 1 นั้น ***switch*** $x นั้นจะมีค่าคือ ONE โดยจะไปทำ pat เป็นสตริงคำว่า ONE จึงทำให้ได้ผลลัพธ์ออกมาเป็น MATCH ONE. 1 + one is 2 โดยจะสังเกตเห็นว่า $z จะมีค่าเท่ากับ ONE เหมือนกับค่าของ $x แต่ในการทำงานของ ***switch*** จะไม่ทำการแทนค่า $z แต่จะมองว่า $z เป็นแค่สตริงตัวหนึ่งเท่านั้น

การใช้ *while* loop

ในภาษา TCL มีรูปแบบการใช้งาน ***while*** loop ดังนี้

```
while booleanExpr body
```

ตัวอย่างที่ 1

```
set x 1
while {$x < 5} {
        puts "x is $x"
        set x [expr $x + 1]
        # สามารถใช้คำสั่ง incr ได้โดยใส่ incr x เข้าไปแทนในบรรทัดนี้
       }
puts "exited first loop with X equal to $x\n"
```

ผลลัพธ์

```
x is 1
x is 2
x is 3
x is 4
exited first loop with X equal to 5
```

จากตัวอย่างจะเป็นการทำไปเรื่อยๆเมื่อ x มีค่าน้อยกว่า 5 โดยจะมีการเพิ่มค่าของตัวแปร x ขึ้นทีละ 1 แล้วแสดงผลค่า x ออกมาจนกระทั่ง x มีค่าเป็น 5 ซึ่ง 5 ไม่ได้ที่ค่าน้อยกว่า 5 จึงทำให้ while loop หยุดทำงานแล้วจึงแสดงข้อความว่า exited first loop with X equal to 5 ออกมา

การใช้ *for*

การใช้งาน ***for*** ในภาษา TCL นั้นจะมีรูปแบบการใช้ ดังนี้

```
for start test next body
```

start คือ ค่าเริ่มต้นของการวนลูป (loop)

test คือ ค่าที่ใช้สำหรับตรวจสอบเงื่อนไข

next คือ จำนวนค่าที่เพิ่มขึ้นต่อรอบ หลังจาก start

body คือคำสั่งที่ต้องทำระหว่างลูป

ในการเพิ่มค่าในส่วนของ next ที่ทำการเพิ่มค่าขึ้นต่อรอบนั้นสามารถใช้คำสั่ง ***incr*** ได้โดยจะมีรูปแบบการใช้งานดังนี้

```
incr ชื่อตัวแปร ค่าที่จะกำหนดให้เพิ่ม
```

โดยดีฟอลท์ของการเพิ่มค่าโดยการใช้คำสั่ง ***incr*** นั้นจะคือจะเพิ่มค่าขึ้นทีละ 1 (หากไม่ใส่ตัวเลขใดๆเลยในส่วนของค่าที่จะกำหนดให้เพิ่ม)

ตัวอย่างที่ 1

```
for {set i 0} {$i < 10} {incr i} {
     puts "I inside first loop: $i"
}
```

ผลลัพธ์

```
I inside first loop: 0
I inside first loop: 1
I inside first loop: 2
I inside first loop: 3
I inside first loop: 4
I inside first loop: 5
I inside first loop: 6
I inside first loop: 7
I inside first loop: 8
I inside first loop: 9
```

จากตัวอย่างที่ 1 จะเป็นการใช้ for โดยมีการทำงานในลักษณะที่มีการวนโดยจะมีการกำหนดค่าเริ่มต้นเท่ากับ 0 แล้วจากนั้นจะไปตรวจสอบกับเงื่อนไขถ้าเงื่อนไขเป็นจริงก็จะมีการทำคำสั่งภายในตัวโปรแกรม (body) จากนั้นจึงทำการเพิ่มค่าขึ้นไปเรื่อยๆและจะทำการตรวจสอบเงื่อนไขไปเรื่อยๆ และทำคำสั่งในตัวโปรแกรม จนกว่าการตรวจสอบจะได้เงื่อนไขเป็นเท็จจึงหยุดการทำงานของลูป

การใช้งานคำสั่ง *break* และ *continue*

คำสั่ง ***break*** และ ***continue*** นั้นจะนิยมใช้ในลูป เช่น ***while*** หรือ ***for*** โดย

continue จะมีผลให้ลูปนั้นทำงานต่อไปโดยผ่านรอบที่กำลังทำอยู่

break จะมีผลทำให้หยุดการทำงานของลูปและออกจากการทำงานในลูปไปทำคำสั่งที่ต่อจากลูปนั้น

ตัวอย่างที่ 1

```
set x 0
while {$x < 10} {                    # ลูปจะทำไปเรื่อยๆเมื่อ x มีค่าน้อยกว่า 9
      incr x 1                       # เพิ่มค่าให้ตัวแปร x ที 1
      if  "$x==3" continue           # ทำการข้ามการทำงานรอบที่ตัวแปร x มีค่าเท่ากับ 3
      if {$x > 7} break              # สั่งให้หยุดการทำงานเมื่อมีค่ามากกว่า 7
      puts "x is $x"
}
puts "exited first loop with X equal to $x\n"
```

ผลลัพธ์

```
      x is 1
      x is 2
      x is 4
      x is 5
      x is 6
      x is 7
      exited first loop with X equal to 8
```

จากตัวอย่างที่ 1 เป็นการทำลูปวนไปเรื่อยๆเมื่อ x มีค่าน้อยกว่า 10 โดยจะมีการเพิ่มค่าของตัวแปร x ขึ้นทีละ 1 โดยใช้คำสั่ง ***incr*** แล้วแสดงผลค่า x ออกมาโดยเมื่อวนรอบมาถึงรอบที่ x มีค่าเท่ากับ 3 จะไปเข้าเงื่อนไขของ ***if*** ที่ตัวแปร x มีค่าเท่ากับ 3 จึงทำให้มีการทำคำสั่ง ***continue*** โดยจะหลุดจากการทำงานในลูปที่ x มีค่าเป็น 3 โดยในการแสดงผลจะไม่แสดง x is 3 ออกมาและไปทำงานในลูปถัดไป และแสดงค่าออกมาเรื่อยๆจนกระทั่งตัวแปร x มีค่าเป็น 8 จึงทำให้ไปเข้าเงื่อนไข ***if*** ที่ x มีค่ามากกว่า 7 จะไปทำคำสั่ง ***break*** และหยุดการทำงานโดยออกไปทำคำสั่งนอกลูป คือไปทำคำสั่ง ***puts*** "exited first loop with X equal to $x\n" และแสดงผลออกมาโดยที่ค่า x มีค่าเป็น 8

ลิสต์ (List) ในภาษา TCL

รูปแบบการใช้ลิสต์

ลิสต์เป็นโครงสร้างข้อมูลพื้นฐาน(basic data structure) ที่เก็บข้อมูลหลายๆอันไว้ในตัวแปรเดียว ซึ่งมีรูปแบบในการสร้างลิสต์ ดังนี้

```
      set lst {{item 1} {item 2} {item 3}}
      set lst [split "item 1.item 2.item 3" "."] # การใช้คำสั่ง split
      set lst [list "item 1" "item 2" "item 3"]  # การใช้คำสั่ง list
```

set lst {{item 1} {item 2} {item 3}} จะหมายความว่าได้ประกาศตัวแปร lst ที่มีสมาชิกเป็น item1 item2 item3 ตามลำดับ

set lst [split "item 1.item 2.item 3" "."] เป็นการสร้างลิสต์โดยใช้คำสั่ง split ร่วมด้วย โดยคำสั่ง split นั้นจะคอยแยกคำตามเครื่องหมาย . โดยจะแยกออกมาเป็นชุดแล้วไปกำหนดค่าให้กับตัวแปร lst ซึ่งทำให้มีสมาชิกเป็น item1 item2 item3 ตามลำดับ

set lst [list "item 1" "item 2" "item 3"] เป็นการสร้างลิสต์โดยใช้คำสั่ง list ร่วมด้วย โดยจะแบ่งแยกสมาชิกตามตัวที่อยู่ในเครื่องหมาย " " โดยมีสมาชิกเป็น item1 item2 item3 ตามลำดับ

ในการเข้าถึงสมาชิกแต่ละตัวในลิสต์จะใช้คำสั่ง ***lindex*** ช่วยโดยมีค่าเริ่มต้นที่ 0 โดยยังสามารถวัดขนาดสมาชิกของลิสต์ได้โดยใช้คำสั่ง ***llength*** และสามารถใช้คำสั่ง ***foreach*** เพื่อช่วยในการแสดงข้อมูลในลิสต์ได้

ตัวอย่างที่ 1

```
set x "a b c"
puts "Item 2 of the list {$x} is: [lindex $x 2]\n"
#ใช้คำสั่ง lindex เพื่อช่วยเข้าถึงสมาชิกในลิสต์
set y [split 7/4/1776 "/"]
# การใช้ลิสต์ ร่วมกับคำสั่ง split
puts "We celebrate on the [lindex $y 1]'th day of the [lindex
$y 0]'th month\n"
set z [list puts "arg 2 is $y" ]
puts "A command resembles: $z\n"
set i 0;
     foreach j $x {
               puts "$j is item number $i in list x"
          incr i;     }
```

ผลลัพธ์

```
Item2of the list{a b c} is: c
We celebrate on the 4'th day of the 7'th month
A command resembles: puts {arg 2 is 7 4 1776}
a is item number 0 in list x
b is item number 1 in list x
c is item number 2 in list x
```

จากตัวอย่างที่ 1 ในส่วนของคำสั่ง ***puts "Item 2 of the list {$x} is: [lindex $x 2]\n"*** จะเป็นการแสดงค่าของลิสต์ x และจะเป็นการใช้คำสั่ง ***lindex*** ในการหาสมาชิกตัวที่ 2 ของลิสต์ x โดยจะมีสมาชิกคือ a b และ c ซึ่งสมาชิกตัวที่ 2 ของลิสต์ x คือตัว c โดยสมาชิกใน

ลิสต์จะเริ่มนับจากตัวแรกของลิสต์คือจะมี index คือ 0 และในคำสั่ง ***puts "We celebrate on the [lindex $y 1]'th day of the [lindex $y 0]'th month\n"*** ซึ่งจะเป็นการหาสมาชิกตัวที่ 1 และ 0 ของลิสต์ y มีสมาชิกเป็น 7 4 1776 ซึ่งในการ สร้างลิสต์ของ y นั้นจะเป็นรูปแบบดังนี้ ***set y [split 7/4/1776 "/"]*** จากการสร้าง list y นั้นได้มีการใช้คำสั่ง split ช่วยในการแบ่งแยกกลุ่มของสตริงโดยจะแยกคำโดยดูจากเครื่องหมาย / ซึ่งแยกออกมาได้สมาชิกดังนี้ 7 4 1776 ส่วน index ตัวที่ 1 และ 0 ของลิสต์ y ก็คือ 4 กับ 7 นั่นเอง และในคำสั่ง ***puts "A command resembles: $z\n"*** นั้นจะเป็นการแสดงค่าสมาชิกทั้งหมดในลิสต์ z ซึ่งได้มีการกำหนดให้มีสมาชิก 2 ตัวคือ ***puts*** กับ ***arg*** 2 is 7 4 1776 จึงทำให้แสดงค่าออกมาดังผลลัพธ์ และในส่วนของคำสั่ง ***foreach*** นั้นจะเป็นการวนแสดงค่าในลิสต์ x จนหมดและแสดงผลลอกมาตามผลลัพธ์ข้างต้น

การเพิ่มลดสมาชิกภายในลิสต์

ในการเพิ่มหรือลดจำนวนสมาชิกในลิสต์ นั้นมีรูปแบบ ดังนี้

การใช้ ***lappend***

lappend	ลิสต์ที่ต้องการนำเอาคำไปต่อ	คำที่จะนำไปต่อ

การใช้ ***linsert***

linsert	ลิสต์ที่ต้องการนำเอาคำไปแทรก	คำที่จะนำไปแทรก

การใช้ ***lreplace***

lreplace	ลิสต์ที่ต้องการนำคำไปแทนที่	ตำแหน่งเริ่มต้น	ตำแหน่งสุดท้าย	คำที่จะนำไปแทน

ตัวอย่างที่ 1

```
set b [list a b {c d e} {f {g h}}]
      puts "Treated as a list: $b\n"
set b [split "a b {c d e} {f {g h}}"]
      puts "Transformed by split: $b\n"
set a [concat a b {c d e} {f {g h}}]
      puts "Concated: $a\n"
lappend a {ij K lm}
# {ij K lm} ถือว่าเป็นค่าเดียว โดยจะเป็นการนำ {ij K lm} ไปต่อท้าย list a
   puts "After lappending: $a\n"
      puts "Length A is: [llength $a]\n"
set b [linsert $a 3 "1 2 3"]
# "1 2 3" ถือว่าเป็นค่าเดียว นำคำเข้าไปแทรกในตำแหน่งที่ 3 ของ list a
   puts "List B After linsert at position 3: $b\n"
      puts "Length B is: [llength $b]\n"
set b [lreplace $b 3 5 "AA" "BB"]
```

ตัวอย่างที่ 1 (ต่อ)

```
      puts "After lreplacing 3 positions with 2 values at
position 3: $b\n"
# นำคำเข้าไปแทนที่ในตำแหน่งเริ่มต้นคือ 3 ถึง ตำแหน่งสุดท้ายคือ 5 ของลิสต์ b
puts "Length B is: [llength $b]\n"
```

ผลลัพธ์

```
Treated as a list: a b {c d e} {f {g h}}
Transformed by split: a b \{c d e\} \{f \{g h\}\}
Concated: a b c d e f {g h}
After lappending: a b c d e f {g h} {ij K lm}
Length A is: 8
List B After linsert at position 3: a b c {1 2 3} d e f {g h}
{ij K lm}
Length B is: 9
After lreplacing 3 positions with 2 values at position 3: a b
c AA BB f {g h} {ij K lm}
Length B is: 8
```

จากตัวอย่างที่ 1 จะเป็นการใช้คำสั่ง ***concat*** ในการแยกกลุ่มของคำโดยจะแยกออกมาได้เป็น 7 ชุดซึ่งก็คือ a b c d e f {g h} โดยจะถือว่าเป็นลิสต์ที่ถูกเก็บไว้ในลิสต์ a โดยจะมีสมาชิกทั้งหมด 7 ตัว หลังจากนั้นได้ใช้คำสั่ง ***lappend*** ในการต่อชุดของคำเข้าไปที่ลิสต์ a โดยทำการต่อชุดของคำ {ij K lm} จึงทำให้จำนวนสมาชิกในลิสต์ a เป็น 8 โดยคำสั่ง ***lappend*** นั้นจะเป็นการเอาคำไปต่อท้ายสมาชิกทั้งหมดในลิสต์ ส่วนคำสั่ง ***linsert*** นั้นจะเป็นการแทรกคำเข้าไปโดยต้องระบุตำแหน่งที่จะให้ทำการแทรกเข้าไปด้วยโดยเมื่อแทรกแล้วจะทำให้ตำแหน่งสมาชิกภายในลิสต์หลังจากคำที่แทรกเข้าไปนั้นเลื่อนไปด้วย และสุดท้ายเป็นคำสั่ง ***lreplace*** จะเป็นคำสั่งที่ใช้สำหรับแทนคำโดยจะต้องมีการกำหนดว่าจะให้เริ่มแทนคำที่ตำแหน่งไหนและสิ้นสุดการแทนคำที่ตำแหน่งไหนด้วย ซึ่งจากตัวอย่างได้เริ่มแทนคำจากตำแหน่งที่ 3 ของลิสต์ b โดยแต่เดิมลิสต์ b จะมีสมาชิกดังนี้ a b c {1 2 3} d e f {g h} {ij K lm} โดยจะเริ่มทำการแทนคำจากตำแหน่งที่ 3 ก็คือ {1 2 3} โดยทำการแทน AA และ BB ลงไปและได้สิ้นสุดการแทนที่ตำแหน่งที่ 5 ก็คือ e โดยหลังจากการแทนคำจะได้สมาชิกดังนี้ a b c AA BB f {g h} {ij K lm} โดยจะเหลือจำนวนสมาชิกในลิสต์ b 8 ตัว

การเข้าถึงไฟล์ (File Access)

ในการเข้าถึงไฟล์นั้นจะมีการเข้าถึงได้หลายกรณี ดังนี้

การเปิดไฟล์

```
open fileName ?access? ?permission?
```

fileName คือชื่อของไฟล์ที่ต้องการเปิด

access คือ mode ที่จะใช้ในการเปิดไฟล์โดยสามารถแบ่งได้เป็น

r คือการเปิดไฟล์ขึ้นมาสำหรับอ่าน โดยจะต้องมีไฟล์นั้นอยู่แล้ว

r+ คือการเปิดไฟล์ขึ้นมาสำหรับอ่านและเขียน โดยจะต้องมีไฟล์นั้นอยู่แล้ว

w คือการเปิดไฟล์ขึ้นมาสำหรับเขียน โดยไม่จำเป็นต้องมีไฟล์นั้นอยู่

w+ คือการเปิดไฟล์ขึ้นมาสำหรับอ่านและเขียน โดยไม่จำเป็นต้องมีไฟล์นั้นอยู่

a คือการเปิดไฟล์ขึ้นสำหรับเขียน โดยต้องมีไฟล์นั้นอยู่ โดยจะทำการเขียนต่อท้ายไฟล์

a+ คือการเปิดไฟล์ขึ้นมาสำหรับเขียน โดยไม่จำเป็นต้องมีไฟล์นั้นอยู่ โดยจะทำการเขียนต่อท้ายไฟล์นั้น

permission คือ ตัวเลขที่ใช้สำหรับกำหนดสิทธ์ในการเข้าถึงไฟล์โดย default จะคือ rw-rw-rw- (0666) โดยสามารถใช้เพื่อกำหนดสิทธ์ในการเข้าถึงสำหรับไฟล์ได้

การปิดไฟล์

close fileID เป็นคำสั่งในการปิดไฟล์หลังจากที่ได้มีการเปิดไฟล์ไว้

ตัวอย่างที่ 1

```
set infile [open "myfile.txt" r]
set number 0
     while { [gets $infile line] >= 0 } {
          incr number
     }
     close $infile
puts "Number of lines: $number"
set outfile [open "report.out" w]
puts $outfile "Number of lines: $number"
close $outfile
```

ไฟล์ myfile.txt มีรูปแบบดังนี้

```
Hello
World
Test
Read
Book
```

จากตัวอย่างจะเป็นการเปิดไฟล์ myfile.txt ขึ้นมาอ่านโดยจะเป็นการวนนับจำนวน บรรทัดใน file myfile.txt และแสดงผลออกมาว่า Number of lines: 5 โดยจะทำการนับทุกครั้งที่มีการขึ้นบรรทัดใหม่และจากนั้นได้มีการสร้างไฟล์ report.out ขึ้นมาเพื่อทำการเขียนจำนวนบรรทัดที่นับได้ในไฟล์ myfile.txt โดยจะเขียนคำว่า Number of lines: 5 ลงไปในไฟล์

ไฟล์ report.out มีรูปแบบดังนี้

```
Number of lines: 5
```

ภาคผนวก ข.

การใช้งานภาษา PERL เบื้องต้น

Perlเป็นภาษาที่มีรากฐานมาจากภาษา C Perl ย่อมาจาก Practical Extraction and Report Language พัฒนาพัฒนาขึ้นในปี 1986 โดยนาย Larry Wall

เหตุผลสำคัญที่ทำให้เป็นที่นิยมคือ

1. สามารถใช้งานได้ฟรี
2. ติดตั้งมาแล้วกับ Unix Standard หรือ Linux
3. สามารถติดตั้งบน Win 32 ได้
4. เขียนง่ายเพราะมีรากฐานมาจากภาษา C ซึ่งเป็นที่นิยมกัน
5. ความสามารถพิเศษด้านการติดต่อระบบ เพื่อการ Admin ระบบ (โดยไม่ต้อง Telnet เข้าไป Admin)
6. มี Function สำเร็จรูปมาให้
7. มี Site Reference มากมายที่สอนการใช้
8. Web Hosting ที่ให้บริการเช่าพื้นที่สร้าง .com .org .net etcฯ โดยประมาณ 90 % เป็น Unix Standard ดังนั้นจึงมี Perl ให้ใช้ จึงทำให้ Website ทั่วโลกกว่า 80 % จึงใช้ Perl
9. Free Web Hosting โดยส่วนใหญ่มีแนวโน้มที่จะเริ่มให้บริการ Perl (ที่ Hypermart.net มีให้ใช้)

ตัวแปร

ตัวแปรใน Perl นั้นมีอยู่ 2 แบบหลักๆ คือ Scalar และ Array การกำหนดชื่อตัวแปรมีข้อกำหนดดังนี้

1. ถ้าเป็นตัวแปร Scalar ให้ใช้เครื่องหมาย $ นำหน้า
2. ถ้าเป็นตัวแปร Arrays ให้ใช้เครื่องหมาย @ นำหน้า
3. หนังสือที่ตามด้วยเครื่องหมาย $ และ @ ต้องเป็นตัวอักษรเท่านั้น เช่น $a , @day , ไม่ใช่ $1 หรือ @+day
4. ไม่ใช่สัญลักษณ์ใดร่วมในตัวแปร เช่น $hi! @question?
5. ตัวอักษรตัวเล็กและตัวใหญ่ให้ความหมายเป็นคนละตัวแปร เช่น $hi กับ $Hi หรือ $Day กับ $day เป็นต้น

ในการกำหนดตัวแปรที่เป็น Scalar นั้นจะใช้สำหรับการเก็บข้อมูลๆ เดียว ซึ่งข้อมูลนั้นอาจจะเป็นตัวเลข (Integer) หรือข้อความ (String) ในการประกาศจะใช้เครื่องหมาย $ นำหน้าเสมอ ยกตัวอย่างเช่น

ตัวอย่างที่ 1

```
$name="NS";
$faculty='it';
$phone= 1234567 ;
```

เครื่องหมาย " " และ ' ' จะให้ความหมายเหมือนกัน แต่ในการประมวลผลจะที่แตกต่างกัน คือ ถ้า Perl พบเครื่องหมาย " " จะมีการนำไปประมวลผล แต่ถ้าพบเครื่องหมาย ' ' Perl จะไม่สนใจ ฉะนั้นถ้าตัวแปรที่มีความหมายไม่มาก ไม่ต้องการเก็บเพื่อการประมวลผลภายหลัง ก็ควรใช้ ' ' เพื่อเป็นการประหยัดทรัพยากรของเครื่อง

ข้อมูลที่เป็น Integer ไม่นิยมใช้เครื่องหมาย " " หรือ ' ' จะไม่ใส่เครื่องหมายใดๆเช่น

ตัวอย่างที่ 2

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$num1="12345";
$num2= 12345print"$num1<br>\n";
print "$num2<br>\n";
```

ผลลัพธ์

```
12345
12345
```

ตัวแปร Scalar จะเก็บข้อมูลเพียงข้อมูลเดียว ถ้าหากเราประกาศตัวแปรชื่อเดียวกันซ้ำจะยึดตามตัวที่ประกาศล่าสุด เช่น

ตัวอย่างที่ 3

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$name="network";
$name= "ns";
print "$name";
```

ผลลัพธ์

```
ns
```

จากผลลัพธ์ข้างต้น มีการแสดงค่าของตัวแปร $name ออกมามีค่าเท่ากับ ns เนื่องจากการประกาศตัวแปรชื่อซ้ำกัน ทำให้ $name มีค่าเท่ากับตัวที่ประกาศล่าสุดนั่นเอง

การประกาศตัวแปรเป็น Array

การเก็บที่เป็นแบบ Array จะเก็บข้อมูลที่มีลักษณะเป็นชุดหรือกลุ่มของข้อมูล ในการเก็บข้อมูลที่เป็นแบบนี้ จะมีการประกาศตัวแปรโดยใช้เครื่องหมาย @ นำหน้าเสมอ เช่น

```
@day  =("Monday","Tuesday","Wednesday","Thursday","Friday",
"Saturday"','Sunday");
```

หมายความว่าตัวแปร day เป็น Array มีข้อมูลที่เก็บอยู่เป็นชุดของข้อมูลคือMonday, Tuesday , Wednesday ,Thursday , Friday , Saturday , Sunday และเมื่อต้องการให้แสดงข้อมูลก็สามารถทำได้โดยใช้คำสั่ง print โดยให้แสดงตัวแปร @day ออกมา

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
@day  =("Monday","Tuesday","Wednesday","Thursday"','Friday",
"Saturday"','Sunday");
print "@day";
```

ผลลัพธ์

```
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

จะเห็นว่าเมื่อเราพิมพ์คำสั่ง print "@day"; จะเป็นการสั่งให้แสดงข้อมูลทั้งหมดที่อยู่ในตัวแปร day ออกมาทั้งหมด แต่หากถ้าเราต้องการดึงข้อมูลเฉพาะตำแหน่งที่เราต้องการก็สามารถทำได้ โดยเปลี่ยนจากเครื่องหมาย @ ที่อยู่หน้าตัวแปร Array มาเป็นเครื่องหมาย $ แล้วทำการระบุตำแหน่งของข้อมูลที่ต้องการลงไป(ซึ่งข้อมูลที่อยู่ตัวแรกนั้นจะเริ่มต้นที่ตำแหน่ง 0) ในเครื่องหมาย [ ] เช่น

ตัวอย่างที่ 2

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
@day  =("Monday","Tuesday","Wednesday","Thursday"','Friday",
"Saturday"','Sunday");
print "@day <p>";
print "$day[0]<br>\n"
print "$day[1]<br>\n"
print "$day[2]<br>\n"
print "$day[3]<br>\n"
print "$day[4]<br>\n"
print "$day[5]<br>\n"
print "$day[6]<br>\n"
```

ผลลัพธ์

```
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

การประกาศตัวแปรที่เป็น Associative Array

Associative Array เป็นการเก็บข้อมูล Array ที่เป็นคู่ๆ โดยกำหนดชุดของข้อมูลจับคู่กันตามลำดับ 0 คู่กับ 1 , 2 คู่กับ 3 , 4 คู่กับ 5 ไปเรื่อยๆ ในการเก็บข้อมูลแบบนี้ สามารถทำได้โดยต้องประกาศเครื่องหมาย % เอาไว้หน้าตัวแปร ยกตัวอย่างเช่น

```
% day = ("Monday", 1 ,"Tuesday" ,2 ,"Wednesday" ,3 );
```

ในการอ้างถึงข้อมูล เราจะสามารถทำได้โดยอ้างถึงข้อมูลตัวแรกของแต่ละคู่ได้เลย เช่น

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
% day = ("Monday", 1 ,"Tuesday ", 2 ,"Wednesday " , 3 );
Print"$day{'Monday'}<br>\n";
print"$day{'Tuesday'}<br>\n";
print "$day {'Wednesday'}<br>\n";
```

จากตัวอย่างเป็นการประกาศ Associative Array ที่ชื่อ day แล้วมีการกำหนดค่าให้กับตัวแปร จากนั้นก็สั่งให้ทำการพิมพ์ข้อมูลแต่ละคู่ของตัวแปรออกมา

ผลลัพธ์

```
Monday  1
Tuesday 2
Wednesday 3
```

Operator

ในการพัฒนาโปรแกรมจำเป็นต้องใช้ Operator เพื่อกำหนดเงื่อนไขและทิศทางการทำงานของโปรแกรม Perl ได้กำหนด Operator โดยแยกตาม คุณสมบัติและการใช้งาน ดังนี้

ตารางที่ 2.3 ตัวดำเนินการทางคณิตศาสตร์ในภาษา Perl

Operator	ตัวอย่าง	ความหมาย
+	$a + $b	ผลบวกของ $a และ $b
-	$a - $b	ผลต่างของ $a และ $b
*	$a * $b	ผลคูณของ $a และ $b
/	$a / $b	ผลหารของ $a หารด้วย $b
%	$a % $b	เศษของการหาร $a ด้วย $b
**	$a ** $b	ผลของ $a ยกกำลังด้วย $b

ตารางที่ 2.4 ตัวดำเนินการที่ใช้ในการกำหนดค่า (Assignment Operators) ในภาษา Perl

Operator	ตัวอย่าง	ความหมาย
=	$var = 5	ให้ตัวแปร $var มีค่าเท่ากับ 5
++	$var++ or ++$var	เพิ่มค่าของ $var ขึ้นอีก 1 และเก็บใน $var
--	$var-- or --$var	ลดค่าของ $var ลงอีก 1 และเก็บใน $var
+=	$var += 3	เพิ่มค่าของ $var ขึ้นอีก 3 และเก็บใน $var
-=	$var -= 2	ลดค่าของ $var ลงอีก 2 และเก็บใน $var
.=	$str .= "ing"	นำข้อความ "ing" เพิ่มต่อท้ายเข้าไปใน $var
*=	$var *= 4	คูณค่าใน $var ด้วย 4 และเก็บใน $var
/=	$var /= 2	นำค่าใน $var หารด้วย 2 และเก็บใน $var
**=	$var **= 2	นำค่าใน $var ยกกำลัง 2 และเก็บใน $var
%=	$var %= 2	หารค่าใน $var ด้วย 2 และนำเศษจากการหารเก็บใน $var
x=	$str x= 20	ซ้ำค่าในตัวแปร $str 20 รอบและเก็บใน $str

ตารางที่ 2.5 ตัวดำเนินการกับข้อความ (String Operators) ในภาษา Perl

Operator	ตัวอย่าง	ความหมาย
.	$a . $b	นำข้อความใน $b ไปต่อท้ายข้อความใน $a
X	$a x $b	ซ้ำข้อความใน $a เป็นจำนวน $b รอบ
substr()	substr($a, $o, $l)	จะให้ข้อความใน $a ที่จุด $o เป็นความยาว $l
index()	index($a, $b)	ตำแหน่งของ $b ใน $a

ตารางที่ 2.6 ตัวดำเนินการเปรียบเทียบ (Relational Operators) ในภาษา Perl

กับค่าตัวเลข	กับข้อความ	ตัวอย่าง	ความหมาย
==	eq	$str eq "Word"	เท่ากันกับ
!=	ne	$str ne "Word"	ไม่เท่ากันกับ
>	gt	$var > 10	มากกว่า
>=	ge	$var >= 10	มากกว่าหรือเท่ากันกับ
<	lt	$var < 10	น้อยกว่า
<=	le	$var <= 10	น้อยกว่าหรือเท่ากันกับ

ตารางที่ 2.7 Pattern Matching Operators ในภาษา Perl

Operator	ตัวอย่าง	ความหมาย
=~ //	$a =~ /ns/	ให้ค่าเป็นจริงถ้าหาก $a มี "ns"
=~ s//	$a =~ s/p/r	เปลี่ยน 'p' ใน $a เป็น 'r'
=~ tr//	$a =~ tr/a-z/A-Z	แปลงตัวอักษรตามที่กำหนด
!~ //	$a !~ /ns/	ให้ค่าเป็นจริงถ้าหาก $a ไม่มี "ns"

ตารางที่ 2.8 ตัวดำเนินการทางตรรกะ (Logical Operators) ในภาษา Perl

Operator	ตัวอย่าง	ความหมาย
&&	$a && $b	ให้ผลเป็นจริงเมื่อ $a และ $b เป็นจริงทั้งคู่ (and)
||	$a || $b	ให้ผลเป็นจริงเมื่อ $a หรือ $b ตัวใดตัวหนึ่งเป็นจริง (or)
!	! $a	ให้ผลเป็นจริงเมื่อ $a เป็นเท็จ (not)

การกำหนดค่าตัวแปร

การกำหนดค่าให้กับตัวแปรสเกลาร์

การกำหนดค่าให้กับตัวแปรสเกลาร์ สามารถทำได้โดยการใช้ Arithmetic Operator ในการเปลี่ยนแปลงค่า เช่น เพิ่มค่า ลดค่า เป็นต้น ตัวอย่างการใช้งาน

กำหนดให้ $a=10 $b=5 ให้คำนวณค่า $c ตามเงื่อนไขที่กำหนดในตารางที่ 9.

ตารางที่2.9 แสดงเงื่อนไขในการคำนวณในภาษา Perl

เงื่อนไขการคำนวณ	กำหนดค่าตัวแปร	ผลลัพธ์ (print "$c";)
$a + $b	$c = $a + $b	15
$a - $b	$c = $a - $b	10
$a * $b	$c = $a * $b	50
$a / $b	$c = $a / $b	2
$a % $b	$c = $a %$b	0
$a ** $b	$c = $a ** $b	100000

การกำหนดค่าให้กับตัวแปร Array

การกำหนดค่าตัวแปรให้กับตัวแปร Array เราสามารถทำได้โดยสามารถเพิ่มหรือลดจำนวนข้อมูลของตัวแปรซึ่งสามารถทำได้ดังนี้

```
@fruit=("Apple","Orange");
```

ในการเพิ่มค่านั้นเราสามารถทำได้โดยเพิ่มค่าเข้าไปในตัวแปร @fruit โดยตรง เช่น

```
@fruit=("Apple","Orange","Grape","Cherry");
#เพิ่มค่าเข้าไปใน Array
@fruit=("Apple","Orange");
@morefruit=("Grape","Cherry");
@morefruitr=("Grape","Cherry",@fruit);
```

ผลลัพธ์

```
@morefruit=("Grape","Cherry","Apple","Orange");
```

การเพิ่มค่า Array

การดำเนินการเพิ่มค่าตัวแปร Array เราสามารถทำได้โดย Function push ดังนี้

รูปแบบการใช้ Function push

```
push = (ข้อมูลตัวแปรเก่า,ข้อมูลตัวแปรใหม่);
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print"Content-type:text/html\n\n";
@morefruit=("Grape","Cherry","Apple","Orange");
push(@morefruit,"Strawberry","Pineapple");
print "@morefruit";
```

ผลลัพธ์

```
Grape Cherry Apple Orange Strawberry Pineapple
```

Function push จะเพิ่มค่าตัวแปรให้กับตัวแปรตัวสุดท้าย ตัวแปร @morefruit กำหนดให้มีค่าเท่ากับ Grape Cherry Apple Orange Function push จะเพิ่มค่าตัวแปร Strawberry Pineapple ลงไปในตัวแปร @morefruit โดยไปต่อที่ตำแหน่งสุดท้าย

ตัวอย่างที่ 2

```
#!/usr/bin/perl
print"Content-type:text/html\n\n";
@fruit=("Apple","Orange");
@morefruit=("Grape","Cherry");
push(@morefruit,@fruit);
print "@morefruit";
```

ผลลัพธ์

```
Grape Cherry Apple Orange
```

การลดค่า Array

ในการดำเนินการลดค่าตัวแปร Array เราสามารถทำได้โดยใช้Function pop ดังนี้

รูปแบบการใช้ Function pop

```
pop = (ข้อมูลตัวแปรเก่า);
```

ตัวอย่าง

```
#!/usr/bin/perl
print"Content-type:text/html\n\n";
@morefruit=("Grape","Cherry","Apple","Orange");
pop(@morefruit);
print "@morefruit";
```

ผลลัพธ์

```
Grape Cherry Apple
```

จากตัวอย่างจะเห็นว่า Function pop จะตัดข้อมูลตัวสุดท้ายออก ตัวแปร @morefruit กำหนดให้มีค่าเท่ากับ Grape Cherry Apple Orange พอใช้คำสั่ง pop (@morefruit); แล้ว สั่งให้แสดง @morefruit จะเหลือข้อมูลแค่สามตัว คือ Grape Cherry Apple

Control Structures

หลักการการควบคุม(Control Structures) นั้น โดยทั่วไปทุกภาษาจะเหมือนกัน แต่จะแตกต่างกันที่ รายละเอียดของคำสั่ง

เงื่อนไข *if*

คำสั่ง ***if*** เป็นคำสั่งสำหรับสร้างเงื่อนไขการทำงานของโปรแกรมว่า เมื่อทำงานมาถึงคำสั่ง if ให้ตรวจสอบว่าเงื่อนไขที่อยู่ใน if นั้นเป็นจริงหรือไม่ ถ้าเป็นจริงก็จะให้ทำงานตามที่อยู่ในเงื่อนไขนั้น ถ้าเป็นเท็จก็จะไม่ทำ โดยมีรูปแบบคำสั่งดังนี้

```
if(เงื่อนไข){งานที่จะต้องทำ}
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$a=500;
      if ( $a == 500 ){print "ตัวแปร a มีค่าเท่ากับ 500";}
```

ผลลัพธ์

```
ตัวแปร a มีค่าเท่ากับ 500
```

กำหนดให้ $a มีค่าเท่ากับ 500 คำสั่งสร้างเงื่อนไขว่า ถ้า $a เท่า 500 ให้พิมพ์คำว่า "ตัวแปร a มีค่าเท่ากับ 500 แต่ถ้าพบว่า $a มีค่าไม่เท่ากับ 500 ก็จะไม่ทำงานอะไร

เงื่อนไข *if else*

ลักษณะคล้ายคำสั่ง if แต่เพิ่มสถานการณ์ว่าถ้าตรวจสอบเงื่อนไขที่ if ไม่เป็นจริงให้ไปทำ

เหตุการณ์ที่อยู่ใน ***else*** โดยมีรูปแบบคำสั่งดังนี้

```
if( เงื่อนไข ){งานที่จะต้องทำ} else {งานที่จะต้องทำ}
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$a=600;
     if  ( $a == 500 ) {
               print"ตัวแปร a  มีค่าเท่ากับ 500";
     } else {
              print"ตัวแปรa  มีค่าไม่เท่ากับ500";
    }
```

ผลลัพธ์

```
ตัวแปรa มีค่าไม่เท่ากับ500
```

จากตัวอย่างที่ 1 กำหนดให้ $a มีค่าเท่ากับ 600 คำสั่งสร้างเงื่อนไขว่า ถ้า $a เท่า 500 ให้พิมพ์คำว่า ตัวแปร a มีค่าเท่ากับ 500 ถ้าไม่เท่ากับ 500 ให้พิมพ์คำว่า ตัวแปรa มีค่าไม่เท่ากับ 500 พอตรวจสอบแล้วไม่เป็น ไปตามเงื่อนไขของ if จริงได้ผลลัพธ์ออกมาเป็นตัวแปร a มีค่าไม่เท่ากับ500

เงื่อนไข *elsif*

คำสั่ง ***elsif*** เป็นคำสั่งสำหรับสร้างเงื่อนไขการทำงานของโปรแกรมว่า ถ้าเหตุการณ์แรกเป็นจริง ก็จะให้ทำงานตามที่กำหนดไว้ แต่ถ้าเป็นเท็จให้เข้าสู่เงื่อนไขที่ 2 ที่เตรียมไว้ ถ้าเป็นจริงก็ให้ทำงานตามที่กำหนด แต่ถ้าไม่เป็นจริงก็ให้ทำงานตามที่กำหนดไว้ถัดไป มีรูปแบบคำสั่งดังนี้

```
if (เงื่อนไข){
} else
```

หรือ

```
if( เงื่อนไข ){งานที่จะต้องทำ} else {งานที่จะต้องทำ}
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$a=20;
$b = 30;
if ($a > $b) {
          print "a is bigger than b";
} elsif ($a == $b) {
         print "a is equal to b";
} else {
     print "a is smaller than b";
}
```

ผลลัพธ์

```
a is smaller than b
```

กำหนดให้ $a มีค่าเท่ากับ 20 และกำหนดให้ $b มีค่าเท่ากับ 30

คำสั่งสร้างเงื่อนไขว่า ถ้า $a มากกว่า $b ให้พิมพ์คำว่า "a is bigger than b" แต่ถ้าไม่พบว่า $a มากกว่า $b ก็จะเข้าสู่สถานการณ์ถัดไป ถ้า $a เท่ากับ $b ให้พิมพ์ว่า "a is equal to b" แต่ถ้าไม่ใช่ทั้งเท่ากับ และมากกว่า $b ให้พิมพ์ว่า "a is smaller than b" จากการทำเงื่อนไขพบว่าไม่ตรงเงื่อนไขใดเลย ทั้งมากกว่าและเท่ากับ เลยแสดงข้อความออกมาว่า a is smaller than b

คำสั่ง *for*

for เป็นคำสั่งสร้างเงื่อนไขให้ทำงานวนรอบซ้ำๆกัน จนกว่าตัวแปรที่กำหนดจะมีค่าครบตามเงื่อนไข โดยจะนำเอาเรื่องของการกำหนดค่าให้กับตัวแปรหรือ Assignment Operators เข้ามาเกี่ยวข้องด้วย การ Assignment ค่าให้กับ Operators ก็คือการสร้างเงื่อนไขและกำหนดค่าให้กับตัวแปรเป็นค่าต่างๆ จนครบที่กำหนด โดยรูปแบบคำสั่งดังนี้

```
for ( สถานการณ์1 ; สถานการณ์ 2 ; สถานการณ์ 3; ){ งานที่จะต้องทำ}
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
for($i=1;$i<=6;$i++) {
              print"Hi hello<br>";
          }
```

ผลลัพธ์

```
Hi hello
Hi hello
Hi hello
Hi hello
Hi hello
Hi hello
```

กำหนดให้ $i มีค่าเท่ากับ 1 แล้วทำการตรวจสอบเงื่อนไขว่า i<=6 ถ้าเป็นไปตามเงื่อนไข ให้พิมพ์คำว่า Hi hello 1 ครั้งแล้วขึ้นบรรทัดใหม่ แล้วเพิ่มค่า i ขึ้น 1 จากนั้นก็วนรอบต่อไปเรื่อยๆ จนกระทั่งเงื่อนไข i<=6 เป็นเท็จ ก็จะออกจากลูป

คำสั่ง foreach

foreach เป็นคำสั่งวนลูปที่ทำงานร่วมกับตัวแปร Array โดยจะต้องกำหนดตัวแปร Array ขึ้นมาก่อนดังเช่น

@car = ("toyota","nissan","misubishi","isuzu"); foreach จะทำการแบ่งชุดตัวแปร @car เป็นตัว ๆ โดยรูปแบบคำสั่งเป็นดังนี้

```
กำหนดตัวแปร Array
foreach ตัวแปร scalar ( ตัวแปร Array ) { งานที่จะต้องทำ}
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
@fruit = ("Apple","Orange","Grpae","Cherry");
foreach $allfruit (@fruit)
                  {
                        print "$allfruit<br>\n";
                  }
```

ผลลัพธ์

```
Apple
Orange
Grape
Cherry
```

จากตัวอย่างได้มีการกำหนดให้ @fruit มีค่าเท่ากับ Apple Orange Grape Cherry คำสั่ง foreach ก็จะแยกชุดตัวแปร @fruit และเก็บไว้ใน $allfruit จากนั้นให้พิมพ์ $allfruit ออกมาทีละบรรทัด

คำสั่ง *while*

คำสั่ง while เป็นคำสั่งสร้างเงื่อนไขให้ทำงานวนรอบซ้ำๆกัน จนกว่าตัวแปรที่กำหนดจะมีค่าครบตามเงื่อนไข ***while*** จะนำเอาเรื่องของการกำหนดค่าให้กับตัวแปรหรือ Assignment Operators เข้ามาเกี่ยวข้องด้วย โดยมีรูปแบบคำสั่งดังนี้

กำหนดค่าตัวแปรเริ่มต้น(Assignment Operators)

```
while ( เงื่อนไข ){ งานที่จะต้องทำ}
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$a = 1;
while ($a<= 10) {
print $a++;
 }
```

ผลลัพธ์

```
12345678910
```

จากตัวอย่างกำหนดให้ $a มีค่าเท่ากับ 1 จากนั้นสร้างเงื่อนไขว่าให้ $a มีค่าน้อยกว่าหรือเท่ากับ10 ลงมา และให้พิมพ์ค่า $a จึงได้ผลลัพธ์ คือ 1 - 10

การดำเนินจัดการ file ข้อมูล

Perl มีความสามารถเปิด file ใน server เพื่ออ่าน เขียน แก้ไข หรือสร้าง file ใหม่ได้ ตัวอย่างเช่น Program Counter สำหรับนับตัวเลขผู้เข้าเยี่ยมชม Website หลักการคือเปิด file นับตัวเลขใน file เขียนทับลงไปใหม่ แล้วรายงานจำนวนผู้เยี่ยมชมออกทางหน้า Website เป็นต้น คำสั่งที่เกี่ยวข้องกับการดำเนินการจัดการ file ข้อมูลมีดังนี้

- ***open*** เปิด file
- ***close*** ปิด file
- เขียน file ใหม่ หรือ ถ้ามี file เก่าอยู่แล้วก็เขียนทับ file เก่า
- >> เขียนต่อจาก file เก่า

เปิดและปิดไฟล์

คำสั่ง ***open*** จะเปิด file ที่มีอยู่แล้ว สามารถใช้ได้ทั้ง Unix และ Windows ถ้าเป็น Unix จะต้องคำนึงถึงสิทธิผู้ใช้ด้วยว่าสิทธิ์ของผู้ใช้นั้นสามารถทำอะไรกับไฟล์ได้บ้าง เช่น ถ้าไม่ใช่เจ้าของ file ก็ไม่สามารถเปิด file นั้นอ่านได้ แต่ถ้าเป็นวินโดวส์ไม่ต้อง คำนึงถึงเรื่องนี้ รูปแบบคำสั่ง ***open*** และ ***close*** file เพื่ออ่านอย่างเดียว

```
open( FILE , "..path.. filename" );
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
open(File,"data.dat");
@AllFile = <File>;
close(File);
print "@AllFile";
```

ผลลัพธ์

```
200
```

คำสั่ง open จะเปิด file ที่ชื่อ data.dat ซึ่งภายในเก็บข้อมูลคือ 200 แล้วเก็บไว้ตัวแปร File จากนั้นก็จะเก็บค่าของชุดตัวแปร File ทั้งหมดไว้ในตัว Array @Allfile จากนั้นก็สั่งให้ปิดไฟล์ โดยใช้คำสั่ง close(File); สั่งพิมพ์ค่าของ @Allfile ทั้งหมดออกทางจอภาพ จึงได้ข้อมูลทั้งหมดในไฟล์นั้นออกมาก็คือ 200 (ถ้าเป็นเซิฟเวอร์Unix/Linux จะต้องเปลี่ยนโหมดให้เป็น 777 โดยใช้คำสั่ง chmod 777 data.dat โดย 777 หมายความว่า ใครก็ได้สามารถ เขียน อ่าน file data.dat ได้)

การสร้างไฟล์หรือเขียนไฟล์ใหม่

เราใช้เครื่องหมาย ">" เพื่อเขียน file ใหม่กรณีที่ไม่มี file เก่า ถ้ามี file เก่าก็จะเขียนทับลงไป โดยมีรูปแบบคำสั่งดังนี้

```
open( FILE , "..path.. />filename" );
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$newdata = Network;
open(File,">data.dat");
print File "$newdata\n";
close(File);
open(File,"data.dat");
@AllFile = <File>;
close(File);
print "@AllFile";
```

ผลลัพธ์

```
Network
```

จากตัวอย่างกำหนดตัวแปร $newdata = Network คำสั่ง open ครั้งแรกจะเปิด file ที่ชื่อ data.dat หรือสร้าง file ใหม่ขึ้นมาถ้าไม่มี data.dat จากนั้นก็เอาค่าตัวแปร $newdata เก็บลงใน data.dat เสร็จแล้วก็ close file คำสั่ง open ครั้งที่ 2 จะเปิด file ที่ชื่อ data.dat แล้วเก็บไว้ตัวแปร File จากนั้นก็จะเก็บค่าของชุดตัวแปร File ทั้งหมดไว้ในตัว Array @Allfile เสร็จแล้วก็ close file สั่งพิมพ์ค่าของ @Allfile ทั้งหมดออกทางจอภาพ

การเขียนข้อความต่อจากไฟล์เดิมที่มีอยู่แล้ว

เราใช้เครื่องหมาย ">>" เพื่อเขียน file ต่อจาก file เดิม โดยมีรูปแบบคำสั่งดังนี้

```
open( FILE , "..path.. />>filename" );
```

ตัวอย่างที่ 1

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$newdata = Simulator;
open(File,">>data.dat");
print File "$newdata\n<br>";
close(File);
open(File,"data.dat");
@AllFile = <File>;
close(File);
print "@AllFile";
```

ผลลัพธ์

```
Network
Simulator
```

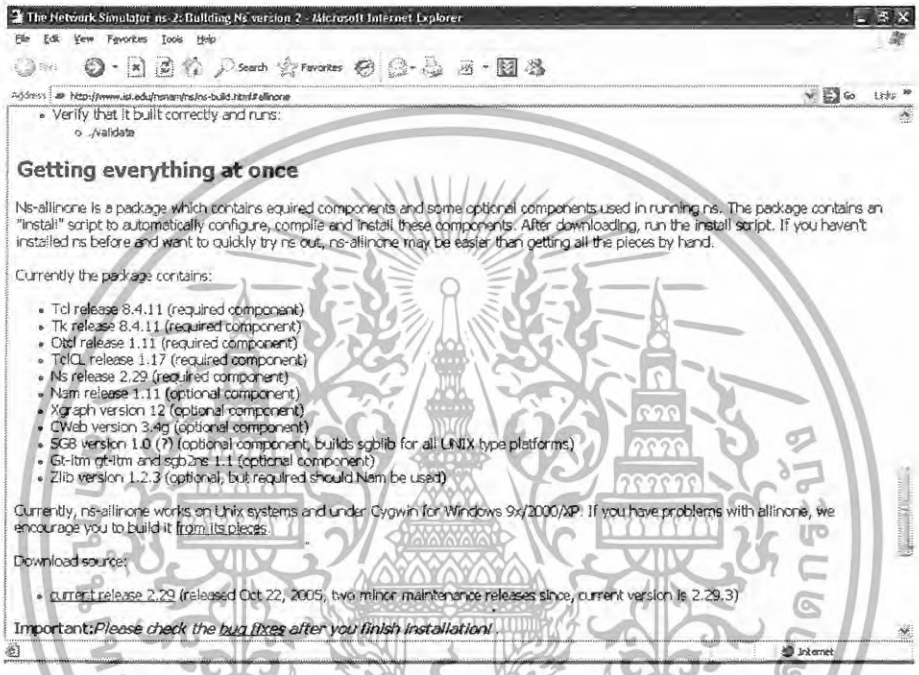
จากตัวอย่างได้มีการกำหนดตัวแปร $newdata = Simulator คำสั่ง open ครั้งแรกจะเปิด file ที่ชื่อ data.dat จากนั้นก็เอาค่าตัวแปร $newdata พิมพ์ต่อท้ายข้อความใน data.dat ซึ่งมีข้อความว่า Network อยู่แล้ว เสร็จแล้วก็ close file คำสั่ง open ครั้งที่ 2 จะเปิด file ที่ชื่อ data.dat แล้วเก็บไว้ตัวแปร File จากนั้นก็จะเก็บค่าของชุดตัวแปร File ทั้งหมดไว้ในตัว Array @Allfile เสร็จแล้วก็ close file สั่งพิมพ์ค่าของ @Allfile ทั้งหมดออกทางจอภาพ

ภาคผนวก ค.

วิธีการติดตั้ง NS2 บน Red Hat Linux 9

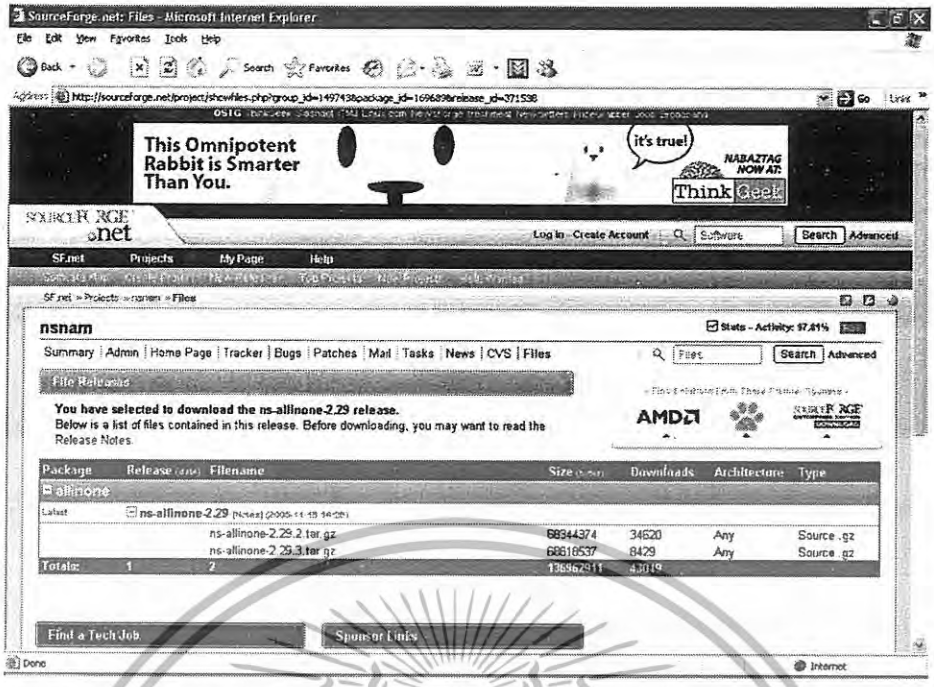
ขั้นตอนที่ 1 ดาวน์โหลด (Download) ไฟล์ ns-allinone-2.29.3.tar

ทำการดาวน์โหลด ไฟล์ ns-allinone-2.29.3.tar ซึ่งใน NS เวอร์ชันนี้ได้ทำการแก้ไขเพื่อให้สามารถติดตั้งบนเรดแฮตลินุกซ์ 9 (Red Hat Linux 9) ได้แล้ว โดยสามารถทำการดาวน์โหลด ไฟล์ได้จาก http://www.isi.edu/nsnam/ns/ns-build.html#allinone จะปรากฏหน้าเวปเพจ (Web page) ดังรูปที่ 1. ดังนี้



รูปที่ 1. หน้าเวปเพจที่ลิงค์ (link) ไปยังที่ดาวน์โหลดไฟล์ ns-allinone-2.29.3.tar

จากนั้นให้เลือกไปที่ current release 2.29 เพื่อลิงค์เข้าไปสู่ที่ดาวน์โหลดไฟล์ ns-allinone-2.29.3.tar จะปรากฏหน้าเวปเพจ ดังนี้



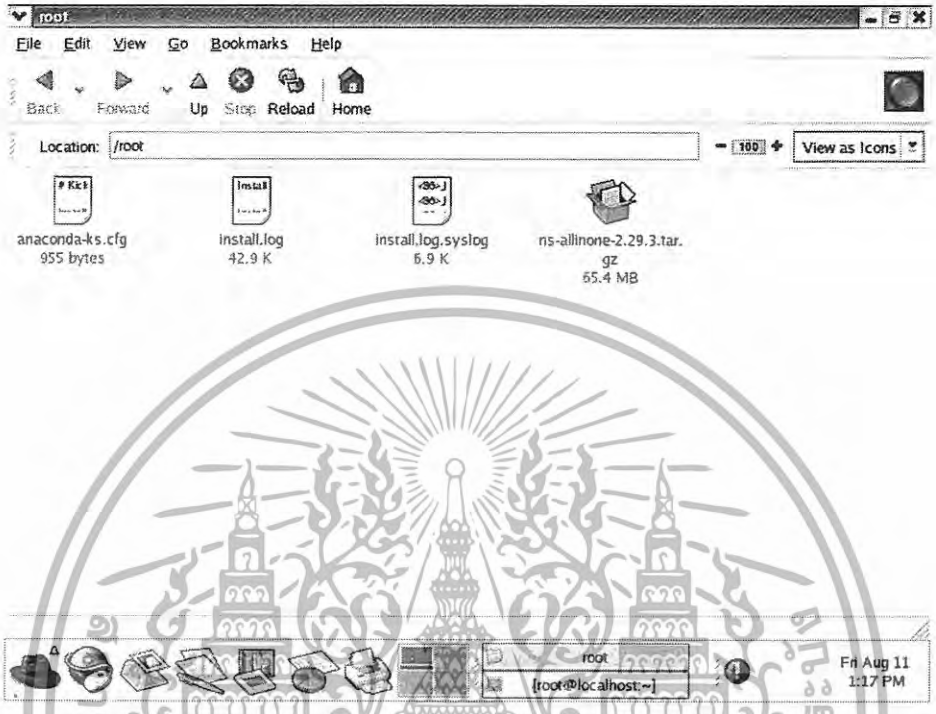
รูปที่ 2. หน้าเวปเพจ ที่ใช้ดาวน์โหลดไฟล์ ns-allinone-2.29.3.tar

ให้ทำการดาวน์โหลดไปไว้ที่เรดแฮตลินุกซ์ 9 ซึ่งสามารถทำการติดตั้งได้หลายกรณี ได้แก่

- /root/ เป็น path ของ root ซึ่งมีสิทธิ์การใช้งานเหมือนผู้ดูแลระบบ
- /home/ชื่อ user/ เป็น path ของผู้ใช้ (user) ที่ได้ทำการกำหนดไว้
- /usr/local/ เป็น path ของผู้ใช้ทั่วไป (public user) ที่ทำให้ผู้ใช้คนอื่นๆสามารถใช้งาน NS2 ได้

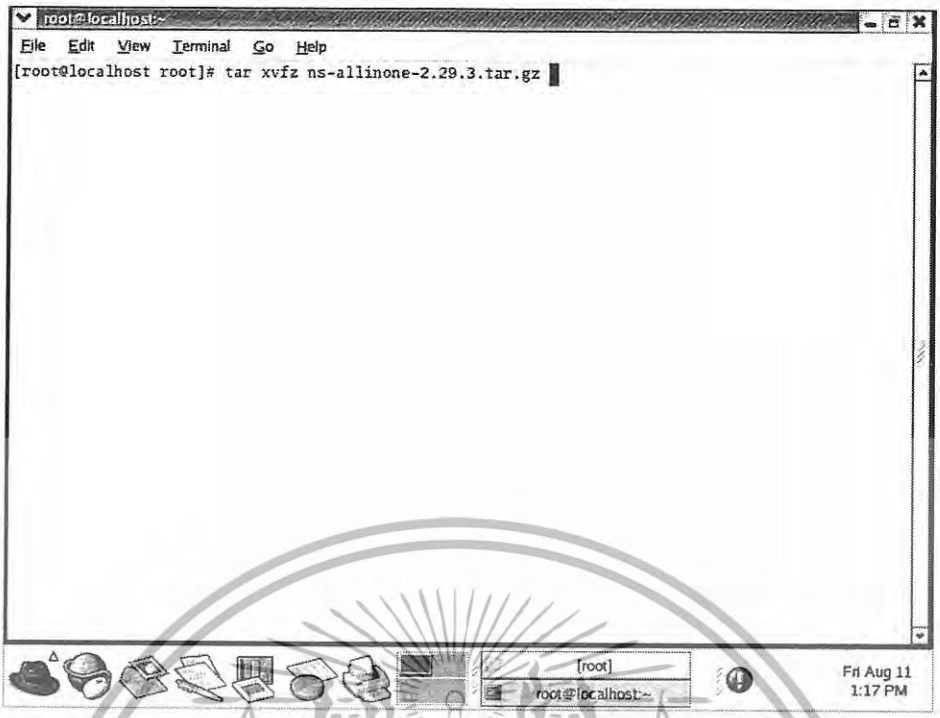
ขั้นตอนที่ 2 ติดตั้ง ns-allinone-2.29.3.tar ลงบนเรดแฮตลินุกซ์ 9

หลังจากที่ได้ทำการดาวน์โหลด ns-allinone-2.29.3.tar ไปไว้ที่เรดแฮตลินุกซ์ 9 โดยนำไปไว้ใน path /root ดังรูปที่ 3.



รูปที่ 3. แสดงการนำไฟล์ ns-allinone-2.29.3.tar ไปใส่ใน path /root

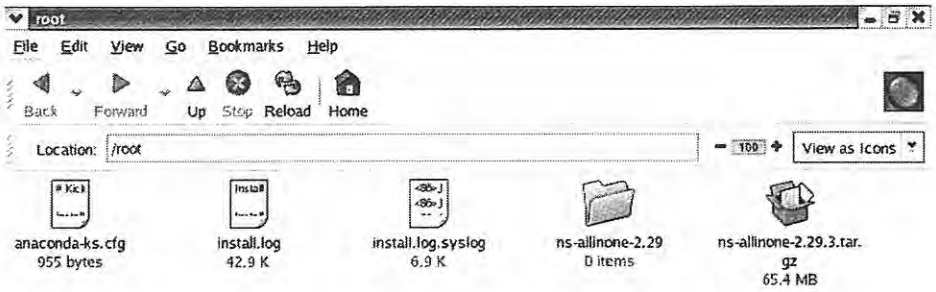
จากนั้นให้ทำการแตกไฟล์ ns-allinone-2.29.3.tar โดยใช้คำสั่ง tar xvfz ns-allinone-2.29.3.tar.gz ซึ่งแสดงวิธีดังรูปที่ 4. และ 5. ดังนี้



รูปที่ 4. แสดงวิธีการแตกไฟล์ ns-allinone-2.29.3.tar

รูปที่ 5. ขั้นตอนขณะกำลังทำการแตกไฟล์ ns-allinone-2.29.3.tar

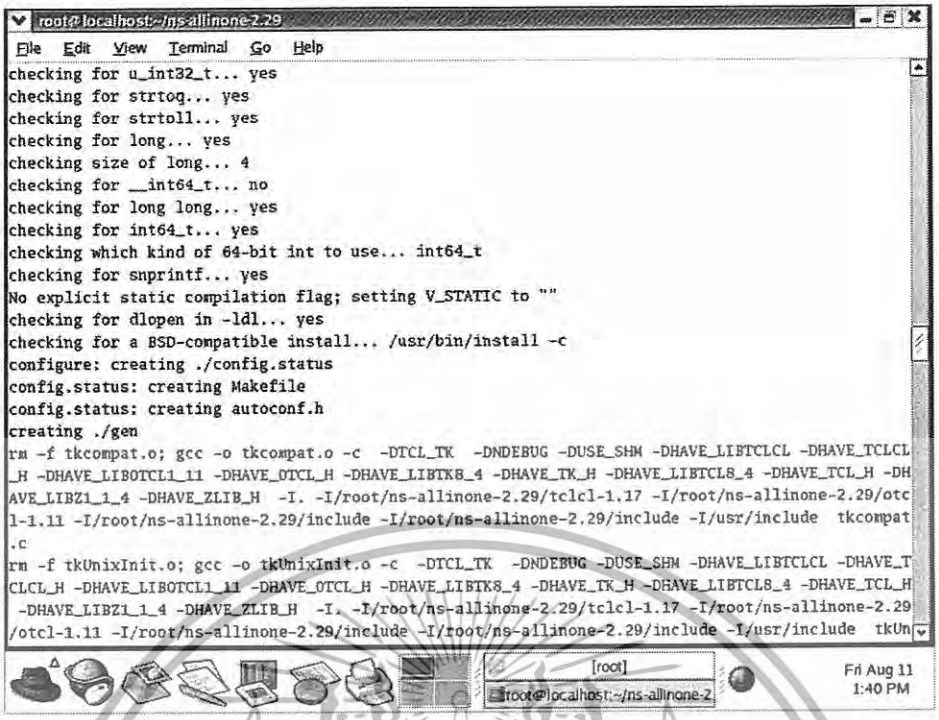
เมื่อทำการแตกไฟล์ ns-allinone-2.29.3.tar เสร็จแล้วจะปรากฏ folder ns-allinone-2.29 ขึ้นใน path /root ดังรูปที่ 6. ดังนี้



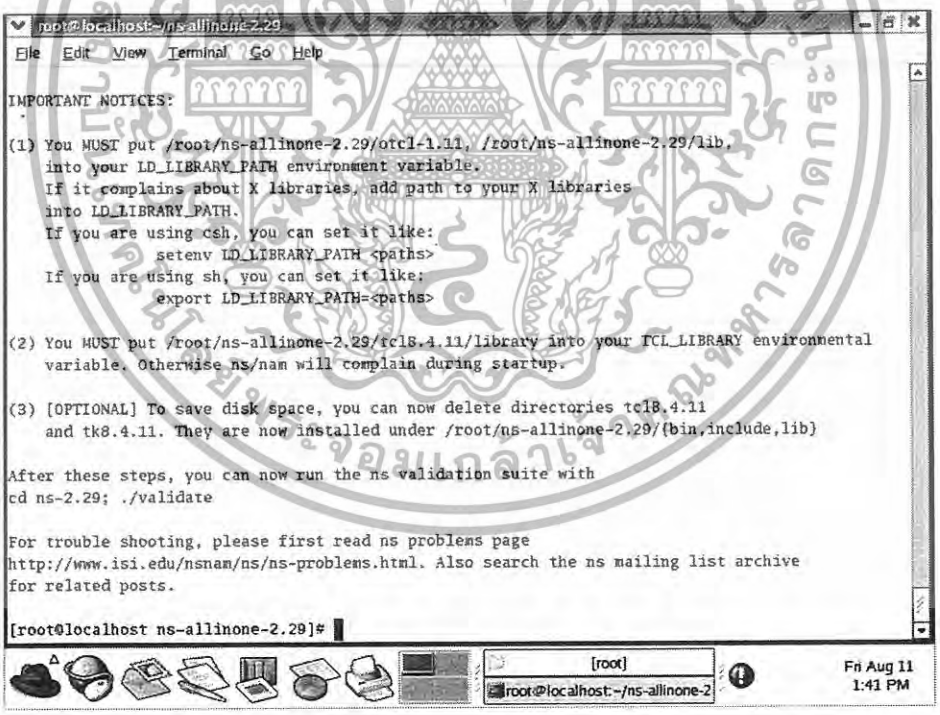
รูปที่ 6. ทำการแตกไฟล์ ns-allinone-2.29.3.tar เสร็จเรียบร้อยแล้ว

จากนั้นให้ทำการติดตั้งโปรแกรม ns-allinone-2.29 โดยให้เข้าไปที่ path ns-allinone-2.29 จากนั้นให้ใช้คำสั่ง ./install เพื่อติดตั้งโปรแกรมซึ่งแสดงดังรูปที่ 7.

รูปที่ 7. install โปรแกรมไปที่ path /root



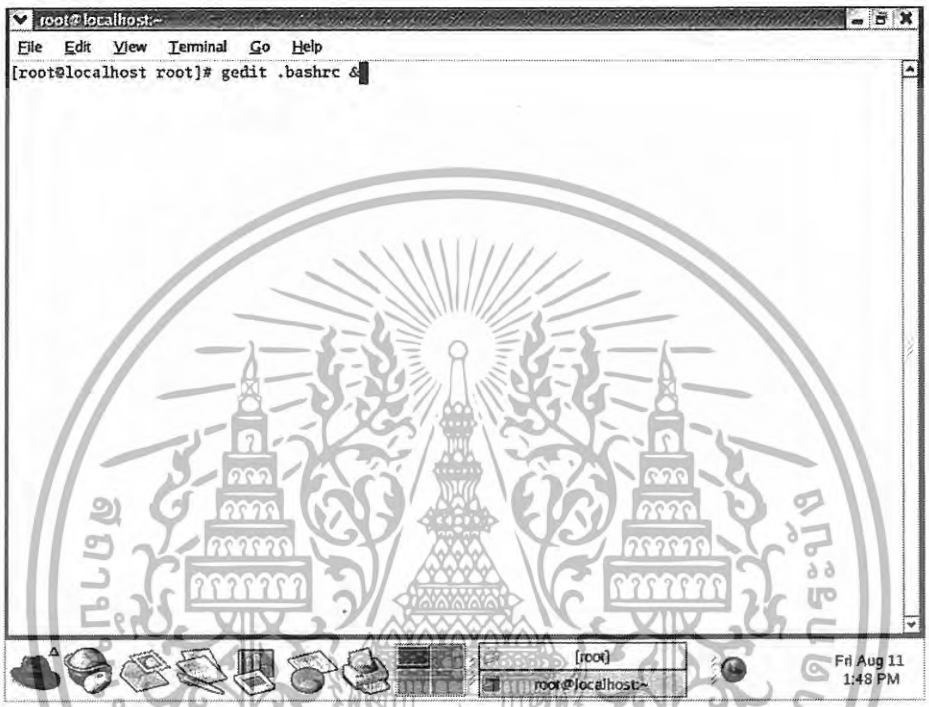
รูปที่ 8. ขณะกำลัง install โปรแกรม ns-allinone-2.29



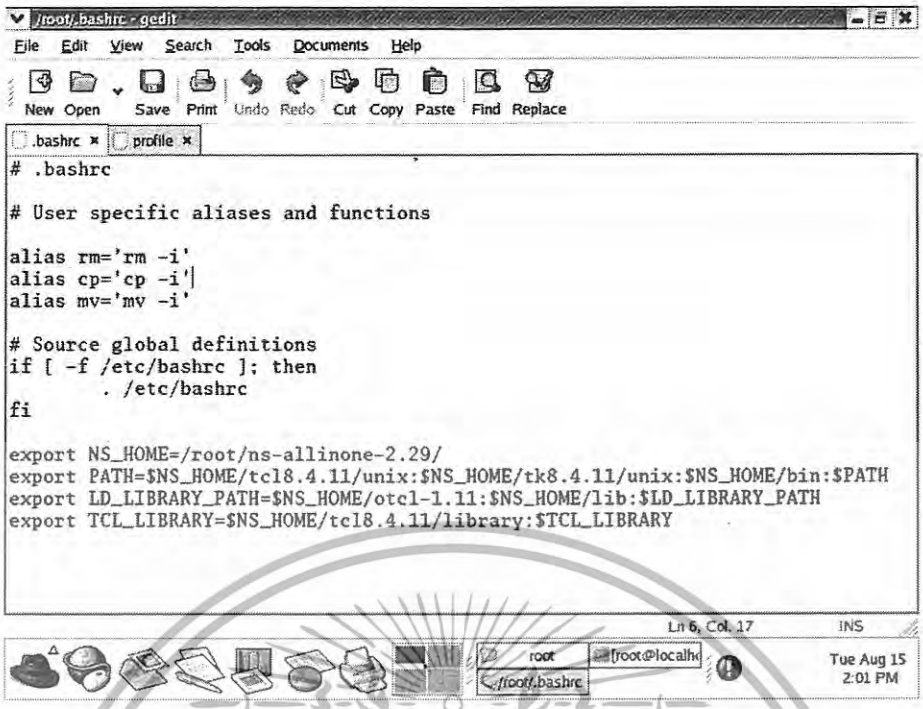
รูปที่ 9. เมื่อ install โปรแกรม ns-allinone-2.29 เสร็จเรียบร้อยแล้ว

เมื่อทำการ install โปรแกรม ns-allinone-2.29 เสร็จเรียบร้อยแล้วให้ทำการหาไฟล์ .bashrc เพื่อทำการเพิ่ม path เข้าไป โดย path ที่เพิ่มเข้าไปเป็นดังนี้

```
export NS_HOME=/root/ns-allinone-2.29/
export PATH=$NS_HOME/tcl8.4.11/unix:$NS_HOME/tk8.4.11/unix:$NS_HOME/bin:$PATH
export LD_LIBRARY_PATH=$NS_HOME/otcl1.11:$NS_HOME/lib:$LD_LIBRARY_PATH
export TCL_LIBRARY=$NS_HOME/tcl8.4.11/library:$TCL_LIBRARY
```



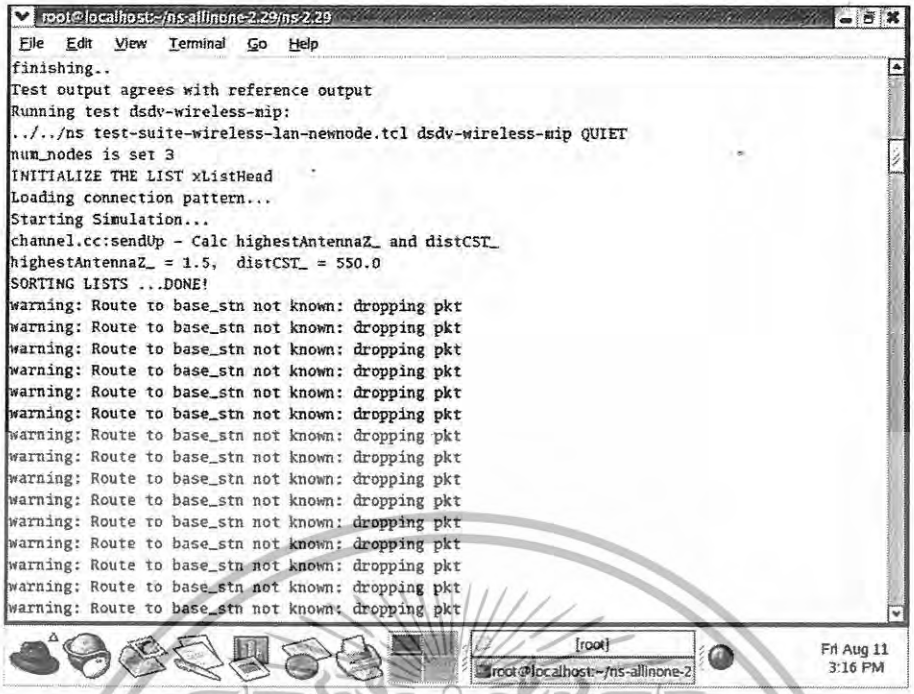
รูปที่ 10. แสดงการใช้ gedit เพื่อทำการเปิดไฟล์ .bashrc



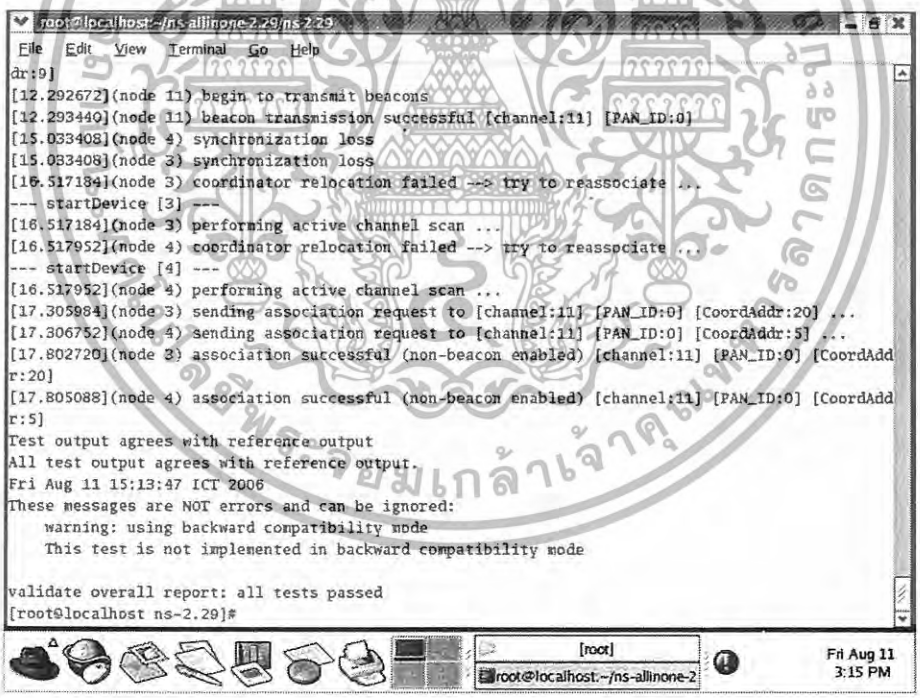
รูปที่ 11. แสดงการเพิ่ม path เข้าไปในไฟล์ .bashrc

เมื่อทำการเพิ่ม path เรียบร้อยแล้วให้ทำการ save ไฟล์ .bashrc จากนั้นให้ทำการ validate ตัว ns-allinone-2.29 โดยให้เข้าไปที่ path ns-allinone-2.29/ns-2.29 โดยพิมพ์คำสั่ง ./validate

รูปที่ 12. แสดงการ validate ตัวโปรแกรม ns-allinone-2.29



รูปที่ 13. ขั้นตอนขณะ validate โปรแกรม ns-allinone-2.29



รูปที่ 14. ขั้นตอนการเสร็จสิ้นการ validate โปรแกรม ns-allinone-2.29

เมื่อการ validate สิ้นสุดลงจะปรากฏข้อความมาว่า all tests passed ซึ่งแสดงว่าได้ทำการ validate ผ่านแล้วซึ่งทำให้สามารถใช้งาน ns2 ได้ โดยการทดสอบลองทำการ run tcl file โดยใช้คำสั่ง ns ชื่อไฟล์.tcl ซึ่งถ้าไม่เกิดปัญหาอะไรก็แสดงว่าสามารถทำการ run โปรแกรม ns2 ได้แล้ว