การทดสอบประสิทธิภาพของโปรโตคอล TCP

TCP PERFORMANCE TEST

โดย

นาย โยธิน ปรีชาสุข 44010390

นาย วริษฐ์ บุญสมจินต์ 44010427

นาย วันชัย เลิศล้ำมีชัย 44010436

อาจารย์ที่ปรึกษา

ผศ. นภัทร สระเอี่ยม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

การทดสอบประสิทธิภาพของโปรโตคอล TCP

TCP PERFORMANCE TEST

โดย

นาย โยธิน ปรีชาสุข

นาย วริษฐ์ บุญสมจินต์

นาย วันชัย เลิศล้ำมีชัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การทดสอบประสิทธิภาพของโปรโตคอล TCP**

TCP PERFORMANCE TEST

ผู้จัดทำ

1. นาย โยธิน ปรีชาสุข **44010390**
2. นาย วริษฐ์ บุญสมจินต์ **44010427**
3. นาย วันชัย เลิศล้ำมีชัย **44010436**

.....อาจารย์ที่ปรึกษา

(ผศ. นภัทร สระเอี่ยม)

การทดสอบประสิทธิภาพของโปรโตคอล TCP

TCP PERFORMANCE TEST

โดย นาย โยธิน ปรีชาสุข 44010390
นาย วริษฐ์ บุญสมจินต์ 44010427
นาย วันชัย เลิศล้ำมีชัย 44010436

อาจารย์ที่ปรึกษา ผศ. นภัทร สระเอี่ยม

บทคัดย่อ

ในปัจจุบันนี้ การสื่อสารด้วยระบบคอมพิวเตอร์ ได้มีความสำคัญต่อการดำเนินการต่าง ๆ เป็นอย่างมาก ทั้งด้านธุรกิจ, การศึกษา หรือแม้แต่องค์กรของรัฐบาล ซึ่งทำให้การใช้งานคอมพิวเตอร์ไม่ได้ถูกจำกัดอยู่แค่การใช้งานส่วนบุคคลเท่านั้น แต่จะเป็นการแลกเปลี่ยนข้อมูลข่าวสารระหว่างคอมพิวเตอร์ โดยอาจจะเป็นทางอินเทอร์เน็ต หรือเครือข่ายภายในองค์กรเอง การส่งข้อมูลข่าวสารระหว่างเครือข่ายที่มีประสิทธิภาพจึงเป็นที่ต้องการ ดังนั้นโครงการนี้จึงจัดทำขึ้นเพื่อทดสอบประสิทธิภาพของเครือข่าย และวิเคราะห์ระบบเครือข่ายบนโปรโตคอล TCP

ABSTRACT

Nowaday, communication by computer network is important to work, business, education and etc. If agency of government, the usage in computer is unlimited in personal computer but will be exchanged the information with others by Internet or internal network organization. The efficiency of data transmission is required. So this project concerns about the efficiency of network test and analyze network system on protocol TCP.

สารบัญ

หน้า

สารบัญรูปภาพ

หน้า

สารบัญรูปภาพ (ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันนี้การติดต่อสื่อสารต่างๆ ทั้งด้านเสียงและข้อมูลนั้น เครือข่ายคอมพิวเตอร์ได้เข้ามามีบทบาทมากขึ้น และคาดการณ์ได้ว่าในอนาคตอันใกล้นี้ทุกๆ สิ่งจะต้องใช้คอมพิวเตอร์เข้ามามีส่วนร่วมมากมาย ดังนั้นเพื่อการที่จะสามารถใช้คอมพิวเตอร์เชื่อมกับเครือข่ายได้นั้นจำเป็นจะต้องมีความรู้ความเข้าใจใน โปรโตคอลต่างๆ ที่เป็นมาตรฐานของเครือข่าย รู้ถึงระบบทราฟฟิกของการถ่ายโอนข้อมูลในเครือข่ายอินเตอร์เน็ต

ปริญญานิพนธ์เล่มนี้ นำเสนอรูปแบบของการจำลองการทำงานของเครือข่าย โดยใช้โปรแกรมจำลอง Network Simulator (NS) ซึ่งสามารถทำงานบนระบบปฏิบัติการได้หลายชนิด สำหรับในโครงงานนี้จะเลือกทำงานบนระบบปฏิบัติการ ลีนุกซ์ (Linux) โดยสาเหตุที่เลือกใช้เพราะว่า ข้อดีที่มีมากมายของตัวลีนุกซ์เองที่เป็นโปรแกรมเสรี (freeware) ทำให้ไม่ต้องเสียค่าใช้จ่ายซื้อระบบปฏิบัติการมาใช้งาน และตัวลีนุกซ์เป็นระบบปฏิบัติการที่ใช้ทรัพยากรในตัวเองน้อย ทำงานได้อย่างมีประสิทธิภาพ มีความเสถียรภาพมั่นคงในการทำงาน และใช้งานทรัพยากรน้อย

1.2 ขอบเขตของปริญญานิพนธ์

ขอบเขตของปริญญานิพนธ์ จะเป็นการศึกษาในรายละเอียดของเครือข่ายในส่วนของโปรโตคอลทีซีพี ซึ่งจะประกอบด้วยองค์ประกอบต่างๆ ที่เกี่ยวข้อง โดยในโครงงานนี้จะพิจารณาในส่วนขององค์ประกอบที่สนใจจะศึกษา แล้วทำการจำลองรูปแบบ เพื่อทำการทดสอบประสิทธิภาพที่ได้ แล้วนำการวิเคราะห์ของผลที่เกิดขึ้น

1.3 วิธีการดำเนินงาน

ในการดำเนินงานนั้น ในส่วนแรกจะทำการรวบรวมข้อมูลและรายละเอียดที่เกี่ยวกับการทำงานและโครงสร้างของข้อมูลในการใช้งานบนโปรโตคอลทีซีพี รวมทั้งศึกษาการใช้งานโปรแกรม NS ในการกำหนดการจำลองรูปแบบเครือข่าย และการตีความของผลได้ จากนั้นหลังจากที่ได้ทำการศึกษาข้อมูลต่างๆ เป็นอย่างดีแล้ว ก็ทำการสรุปถึงสิ่งที่มีผลกระทบต่อประสิทธิภาพการทำงาน แล้วทำการทดสอบโดยใช้การจำลองการทำงานของโปรแกรม NS แล้วนำผลที่ได้ไปวิเคราะห์ โดยในส่วนสุดท้ายก็ทำการสรุปถึงสิ่งที่มีผลกระทบต่อประสิทธิภาพในการทำงาน

บทที่ 2
ทฤษฎีและหลักการ

2.1 การควบคุมความคับคั่งของข้อมูลในโปรโตคอล TCP

เมื่อปริมาณข้อมูลในระบบเครือข่ายมีมากกว่าความสามารถในการรับส่งข้อมูลเมื่อใด ก็จะเกิดปัญหาความคับคั่งของข้อมูล ซึ่งมีลักษณะคล้ายกับปัญหาการจราจรติดขัดในช่วงเวลาเร่งด่วน แม้ว่าโปรโตคอลในชั้นควบคุมเครือข่ายจะพยายามแก้ไขปัญหานี้แล้วก็ตาม แต่ส่วนที่จะแก้ปัญหาได้อย่างแท้จริงเป็นส่วนของโปรโตคอล TCP ซึ่งเป็นตัวที่ควบคุมการส่งหรือไม่ส่งข้อมูลไปยังชั้นสื่อสารควบคุมเครือข่าย การควบคุมในจุดนี้จึงเท่ากับเป็นการส่งข้อมูลในอัตราที่ช้าลงซึ่งจะช่วยแก้ปัญหาได้ดีกว่า

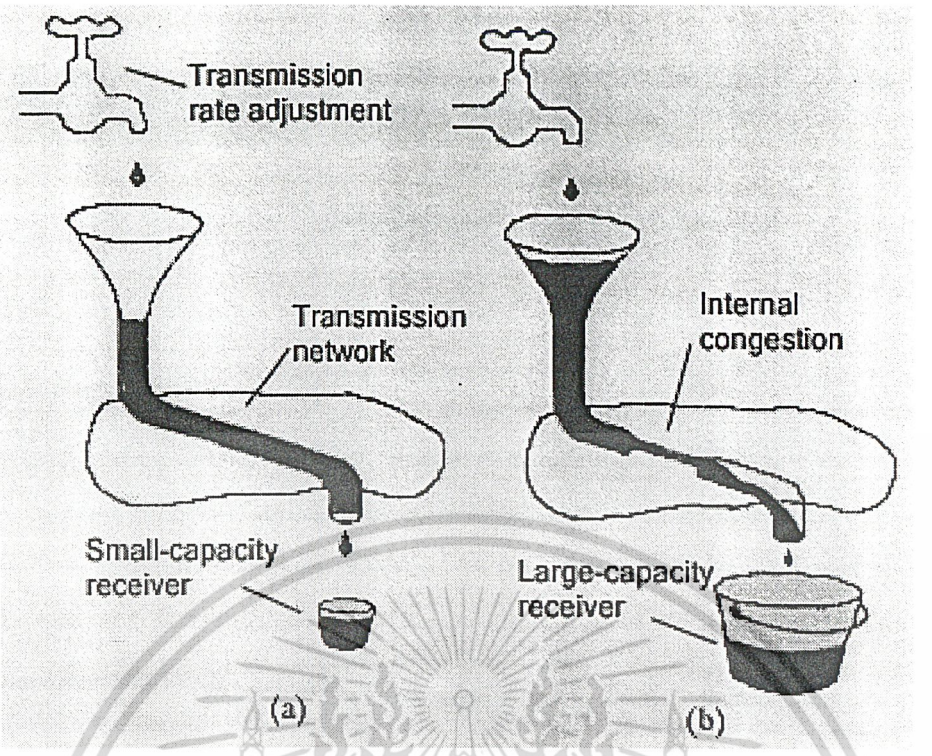
การแก้ปัญหาในทางทฤษฎีนั้นใช้หลักการทางฟิสิกส์เข้ามาช่วย คือ กฎการอนุรักษ์แพ็กเก็ต (The law of conversation of packets) แนวความคิดพื้นฐานคือการไม่ส่งแพ็กเก็ตเข้าสู่ระบบเครือข่ายจนกว่าแพ็กเก็ตเดิมจะถูกส่งออกไปเรียบร้อยแล้ว โปรโตคอล TCP พยายามนำหลักการนี้ไปใช้โดยจัดการปรับขนาดของเซ็กเมนต์ตลอดเวลาที่ยังมีการสื่อสารเกิดขึ้น

ขั้นตอนแรกของการแก้ปัญหา คือ จะต้องสามารถตรวจพบว่ามีความคับคั่งเกิดขึ้นให้ได้ เมื่อก่อนนี้การตรวจจับทำได้ลำบากมาก เนื่องจากแพ็กเก็ตที่เดินทางมาไม่ทันระยะเวลารอคอยนั้นอาจเกิดได้จาก 1. สัญญาณรบกวนที่เกิดขึ้นในสายสื่อสาร หรือ 2. แพ็กเก็ตถูกลบทิ้งโดย router ที่เกิดความคับคั่งสูง ผู้รับและผู้ส่งข้อมูลไม่สามารถแยกความแตกต่างระหว่าง 2 เหตุการณ์นี้ได้

ในปัจจุบันนี้ แพ็กเก็ตที่สูญหายในระหว่างการนำส่งเนื่องจากสัญญาณรบกวนนั้นเกิดขึ้นน้อยมาก (แม้ว่าในบางระบบ เช่น การสื่อสารไร้สายจะยังคงมีปัญหานี้อยู่ก็ตาม) ดังนั้นการที่แพ็กเก็ตที่เดินทางมาถึงจุดหมายไม่ทันระยะเวลารอคอยจึงเกิดขึ้นจากความคับคั่งของข้อมูลเพียงสาเหตุเดียว

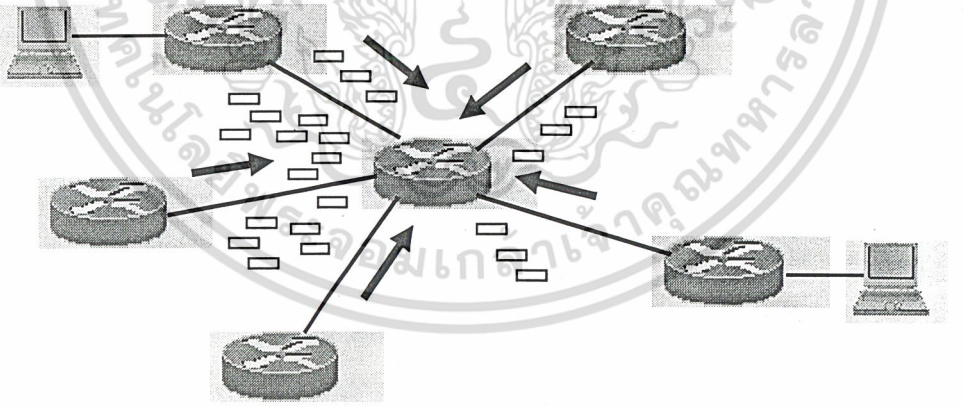
เมื่อการเชื่อมต่อเริ่มเกิดขึ้นก็ต้องมีการกำหนดขนาดของหน้าต่างสื่อสาร ซึ่งก็คือขนาดสูงสุดของเซ็กเมนต์ที่อนุญาตให้ใช้ในการสื่อสาร โดยปกติผู้รับจะใช้ขนาดบัฟเฟอร์เป็นตัวกำหนด ถ้าผู้ส่งใช้ตัวเลขนี้ในการสื่อสารอย่างเคร่งครัด ปัญหาในการสื่อสารจะไม่เกิดขึ้นที่ทางฝ่ายผู้รับอย่างแน่นอน แต่อาจทำให้เกิดความคับคั่งขึ้นในระบบเครือข่าย

จากในรูปที่ 2.1 แสดงตัวอย่างของระบบที่เกิดความคับคั่งโดยใช้ภาพของของเหลวเป็นตัวเปรียบเทียบ ในรูปที่ 2.1(a) มีท่อขนาดใหญ่ถ่ายเทของเหลวไปยังถังขนาดเล็ก ตราบเท่าที่ผู้ส่งไม่ถ่ายเทของเหลวมากจนเกินกว่าที่ถังจะรับได้ ก็จะไม่มีการสูญเสียเกิดขึ้น ในรูปที่ 2.1(b) ถังที่รับของเหลวมีขนาดใหญ่มาก ปัญหาเกิดขึ้นมาจากท่อถ่ายของเหลวช่วงหนึ่งมีขนาดเล็กมาก ถ้าของเหลวไหลเข้าสู่ท่อในปริมาณมากและรวดเร็วเกินไป ของเหลวก็จะล้นออกจากท่อไม่ได้ล้นออกจากถังที่รอรับอยู่



รูปที่ 2.1 แสดงการเปรียบเทียบระบบที่เกิดความคับคั่ง 2 ระบบ

โดยรูปที่ 2.1(b) สามารถแสดงได้เป็นแผนภาพดังรูปที่ 2.2 แสดงถึงการเชื่อมโยงระหว่างสถานี A และ B และมีแพ็กเก็ตจำนวนมากไหลเข้าสู่เราเตอร์ศูนย์กลาง จนกระทั่งเราเตอร์ไม่สามารถนำส่งแพ็กเก็ตได้ทัน



รูปที่ 2.2 การเชื่อมโยงระหว่างสถานีที่เกิดความคับคั่ง

เมื่อ TCP ควบคุมกระแสข้อมูลด้วยการประกาศค่าหน้าต่าง เพื่อให้อีกฝ่ายทราบขนาดข้อมูลที่จะส่ง แต่วิธีนี้เป็นการควบคุมกระแสข้อมูลที่มองโดยรวมระหว่างต้นทางกับปลายทางโดยไม่คำนึงถึงกระแสข้อมูลระหว่างทางแต่อย่างใด เมื่อเราเตอร์ศูนย์กลางมีปัญหาข้อมูลล้นเกินไปแต่สถานี A ที่เป็นฝ่ายส่งไม่มีทางทราบถึงปัญหานี้ทำได้แต่รอการตอบรับจากสถานี B

เมื่อสถานี A รอการตอบรับจนกระทั่งหมดเวลา ก็จะถือว่าแพ็กเก็ตที่ส่งออกไปสูญหายและต้องส่งซ้ำใหม่ ในทำนองเดียวกับสถานีอื่นในเครือข่ายที่ส่งข้อมูลผ่านเราเตอร์ตัวเดียวกันก็จะส่งแพ็กเก็ตซ้ำเช่นเดียวกับสถานี A โดยแพ็กเก็ตที่ส่งซ้ำนั้นจะยิ่งสร้างความคับคั่งให้สูงขึ้น และเป็นตัวถ่วงเวลาแพ็กเก็ตอื่นให้เดินทางช้าลงไปอีกทำให้ต้องมีการส่งซ้ำเพิ่มตามมา จึงทำให้ปัญหายิ่งลุกลามมากขึ้นจนอาจทำให้การสื่อสารทั้งหมดในเครือข่ายหยุดชะงักได้

หนทางแก้ปัญหาในระบบดังกล่าวจึงขึ้นอยู่กับความสามารถในการรับทราบปัญหาทั้ง 2 แบบ คือ ความจุข้อมูลของระบบเครือข่ายและความจุข้อมูลของผู้รับซึ่งมีวิธีแก้ไขแตกต่างกัน ผู้ส่งจะต้องเก็บตัวเลขที่นอกเหนือจากขนาดของหน้าต่างสื่อสารที่ผู้รับแจ้งให้ทราบ นั่นคือผู้ส่งต้องเก็บขนาดของหน้าต่างความคับคั่ง (congestion window) ด้วย ซึ่งหน้าต่างความคับคั่งนี้มีไว้กำหนดปริมาณข้อมูลที่ส่งได้ ซึ่งเป็นเสมือนหน้าต่างกำหนดความจุตลอดทั้งเส้นทางเชื่อมโยงว่าสามารถรับแพ็กเก็ตได้ปริมาณเท่าใด

โดยตัวเลขทั้งสองตัวนี้ใช้กำหนดขนาดของเซ็กเมนต์ที่ผู้ส่งจะส่งไปยังผู้รับซึ่งจะต้องเลือกค่าที่น้อยกว่าในระหว่างตัวเลขทั้งสองตัวนี้ เช่น ถ้าผู้ใช้อนุญาตให้ใช้ขนาด 8 กิโลไบต์ แต่ผู้ส่งทราบว่าขนาดที่จะไม่ทำให้เกิดความคับคั่งจะต้องไม่เกิน 4 กิโลไบต์ ผู้ส่งก็จะส่งเซ็กเมนต์ 4 กิโลไบต์ ในทางกลับกันผู้รับอนุญาตให้ใช้ขนาด 8 กิโลไบต์ แต่ระบบเครือข่ายสามารถรองรับข้อมูลได้มากถึง 32 กิโลไบต์ ผู้ส่งก็จะส่งข้อมูลขนาด 8 กิโลไบต์ได้เต็มตามขนาดที่ผู้รับอนุญาต

เมื่อการเชื่อมต่อเริ่มขึ้น ผู้ส่งจะใช้ขนาดเซ็กเมนต์สูงสุดที่ผู้ใช้อนุญาตเป็นขนาดหน้าต่างสื่อสารความคับคั่ง หลังจากนั้นจึงเริ่มส่งเซ็กเมนต์ขนาดสูงสุดไปยังผู้รับ ถ้าได้รับการตอบรับก่อนหมดระยะเวลารอคอย ผู้ส่งจะเริ่มขนาดหน้าต่างสื่อสารความคับคั่งเป็นสองเท่าแล้วจัดการส่งเซ็กเมนต์ขนาดเดิมจำนวนสองเซ็กเมนต์ออกมาติดกัน ถ้าประสบผลสำเร็จผู้ส่งก็จะพยายามเพิ่มขนาดขึ้นไปเรื่อยๆ ครั้งละ 1 เท่าตัวของขนาดหน้าต่างสื่อสารความคับคั่งล่าสุด (ที่ประสบผลสำเร็จ)

ผู้ส่งจะปรับขนาดหน้าต่างสื่อสารความคับคั่งไปเรื่อยๆจนกว่าจะเกิดปัญหาขึ้น คือ แพ็กเก็ตตอบรับเดินทางมาไม่ถึงภายในเวลาที่กำหนดหรือส่งข้อมูลไปจนเต็มขนาดหน้าต่างสื่อสารที่ผู้ใช้กำหนด เช่น ถ้าผู้ส่งประสบความสำเร็จในการส่งข้อมูลขนาด 1024 , 2048 และ 4096 ไบต์ แต่ล้มเหลวเมื่อส่งข้อมูลขนาด 8192 ไบต์ ผู้ส่งจะต้องกำหนดขนาดหน้าต่างสื่อสารความคับคั่งไว้ที่ 4096 ไบต์ หลังจากนั้นผู้ส่งจะส่งข้อมูลออกมาครั้งละไม่เกิน 4096 ไบต์ แม้ว่าหน้าต่างสื่อสารที่ผู้ใช้กำหนดจะมีขนาดใหญ่กว่านี้ก็ตาม

การทำงานข้างต้นนี้ Jacobson เป็นคนเสนอซึ่งเป็นวิธีสองวิธีให้ทำงานร่วมกันแก้ปัญหาความคับคั่งเรียกว่า การเริ่มต้นอย่างช้า (Slow Start) และการหลีกเลี่ยงความคับคั่ง (Congestion Avoidance)

การเริ่มต้นอย่างช้าหมายถึงให้ TCP นำส่งแพ็กเก็ตทีละน้อยหลังจากเริ่มต้นสถาปนาการเชื่อมโยงและเพิ่มขึ้นจนกระทั่งถึงจุดที่เครือข่ายจะรองรับได้ ข้อดีของการเริ่มต้นอย่างช้าคือ ไม่เพิ่มภาระเครือข่ายในทันทีทันใดและเป็นการทดสอบไปพร้อมกันว่าเครือข่ายในขณะนั้นสามารถรองรับปริมาณข้อมูลได้มากน้อยเพียงใด แต่ถ้าเพิ่มปริมาณส่งไปถึงจุดๆหนึ่งจนแพ็กเก็ตเกิดสูญหายจากความคับคั่ง ขั้นตอนหลีกเลี่ยงความแออัดจะรับหน้าที่ต่อเพื่อรักษาระดับการนำส่งแพ็กเก็ตให้อยู่ในปริมาณที่เหมาะสม

2.2 แนวคิดพื้นฐานของการจัดการโปรโตคอล TCP

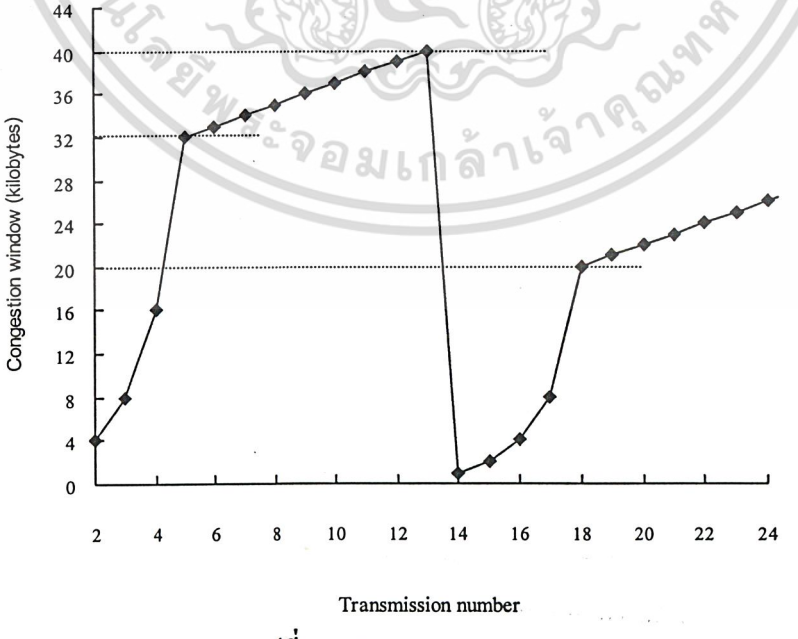
เมื่อทั้งสองฝ่ายสถาปนาการเชื่อมต่อได้แล้ว ฝ่ายส่งจะกำหนดค่า cwnd (ค่าสูงสุดของเซ็กเมนต์) เท่ากับหนึ่งเซ็กเมนต์ ซึ่งหมายถึง TCP ส่งข้อมูลโดยไม่ต้องรอการตอบรับได้เพียงหนึ่งเซ็กเมนต์และใช้ค่าอีกค่าหนึ่งกำหนดเพดานของเซ็กเมนต์เรียกว่า ssthresh (Slow Start threshold) ซึ่งปกติกำหนดให้เท่ากับ 64 กิโลไบต์

ค่า cwnd เริ่มต้นจาก 1 และจะเพิ่มขึ้นครั้งละ 1 ต่อเซ็กเมนต์ตอบรับที่ได้ ดังนั้นในขั้นแรกเมื่อได้รับเซ็กเมนต์ตอบรับแล้ว ค่า cwnd จะเพิ่มค่าเป็น 2 หรือส่งได้ 2 เซ็กเมนต์ จากนั้นเมื่อมีการตอบรับทั้งสองเซ็กเมนต์ ค่า cwnd จะเพิ่มเป็น 4 และเพิ่มเป็น 8 หรือเพิ่มขึ้นแบบเอ็กซ์โพเนนเชียล จนกระทั่งถึงขนาดหน้าต่างที่กำหนดหรือเกิดแพ็กเก็ตสูญหายทำให้ต้องส่งซ้ำ

หากเกิดกรณีส่งเซ็กเมนต์ซ้ำ ค่า cwnd จะนำมาหาร 2 แล้วเก็บเข้าไปในตัวแปรอีกตัวแปรหนึ่งคือค่าที่เรียกว่า ssthresh (Slow Start threshold) ถัดจากนั้นให้เริ่มกระบวนการเริ่มต้นอย่างช้าใหม่อีกครั้งหรือกลับไปเซ็ตค่า cwnd เท่ากับ 1 แต่ในขั้นต่อมานี้ค่า cwnd จะมีขีดจำกัดการเพิ่มแบบเอ็กซ์โพเนนเชียลที่ไม่เกินค่า ssthresh

เมื่อ cwnd เพิ่มค่าขึ้น ssthresh จะหยุดการเพิ่มแบบเอ็กซ์โพเนนเชียล (หยุดขั้นตอนเริ่มต้นอย่างช้า) แล้วเริ่มเข้าสู่กระบวนการใหม่คือหลีกเลี่ยงความคับคั่ง ในขั้นนี้ค่า cwnd จะเพิ่มขึ้นครั้งละ 1/cwnd ต่อทุกเซ็กเมนต์ตอบรับหรือเพิ่มแบบเชิงเส้น ถ้าหากไม่มีปัญหาการส่งเซ็กเมนต์ซ้ำอีก ค่า cwnd ก็จะเข้าสู่สมดุลตามค่าหน้าต่างฝ่ายรับอีกครั้ง

โดยปกติแล้วหากไม่มีปัญหาใดๆ ค่า cwnd ก็จะเพิ่มขึ้นไปได้ถึง 64 เซ็กเมนต์ (ให้เซ็กเมนต์ใหญ่สุดเท่ากับ 1 กิโลไบต์) ในที่นี้สมมติว่ามีปัญหาเซ็กเมนต์สูญหายทำให้ต้องส่งซ้ำดังนั้นค่า ssthresh จะมีค่าเท่ากับครึ่งหนึ่งของค่า cwnd ล่าสุดหรือเท่ากับ 32 จากนั้นกระบวนการเริ่มต้นอย่างช้าจะเริ่มต้นขึ้น ดังรูปที่ 2.3 โดยเพิ่มค่า cwnd จาก 1,2,4,8,16 และหยุดอยู่ที่ 32



รูปที่ 2.3 Congestion Window

ถัดจากนั้นกระบวนการหลีกเลี่ยงความคับคั่งจะเริ่มทำงานโดยเพิ่มค่า cwnd เป็นฟังก์ชันเชิงเส้นสมมติให้การส่งครั้งที่ 13 เกิดปัญหาอีก ค่า ssthresh จะถูกปรับเปลี่ยนเป็น 20 และเริ่มส่งเซ็กเมนต์ตามการเริ่มต้นอย่างช้าใหม่ ในคราวนี้ cwnd จะเพิ่มค่าแบบเอ็กซ์โพเนนเชียลได้ไม่เกิน 20

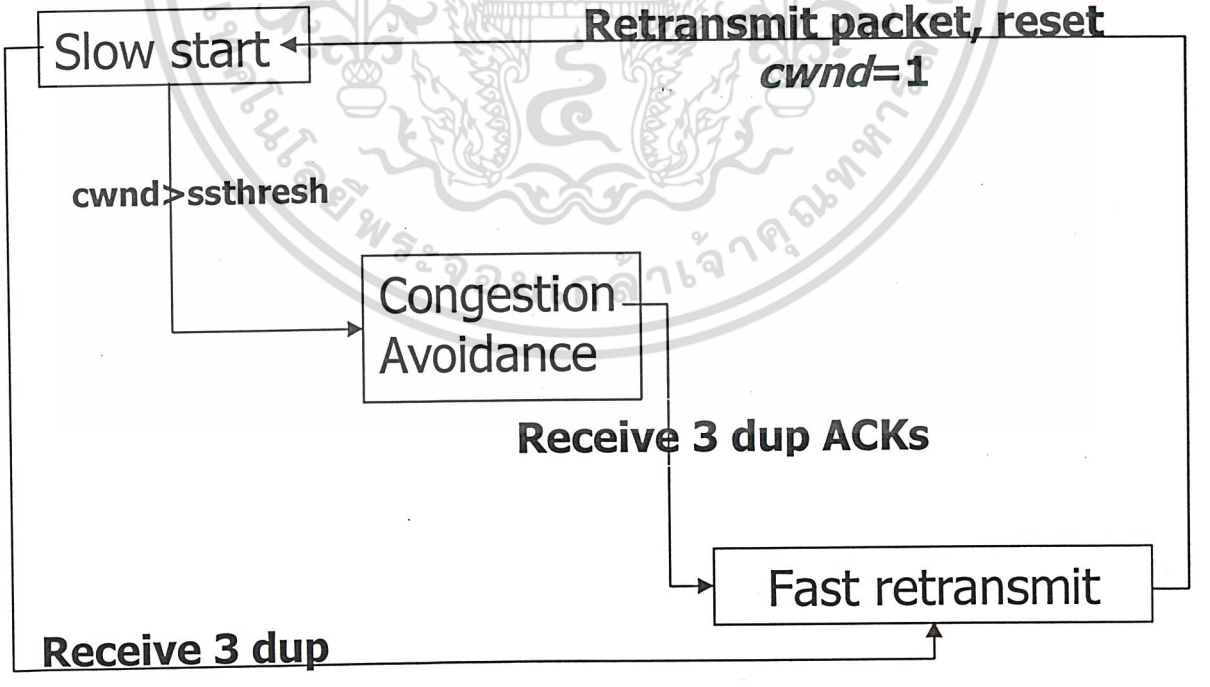
2.3 ชนิดของโปรโตคอล TCP

2.3.1 Tahoe TCP

TCP สมัยใหม่จะประกอบไปด้วยจำนวนของอัลกอริธึมที่ใช้ควบคุมความคับคั่งของเครือข่ายในเพื่อช่วยทำให้ค่า Throughput ในการใช้งานมีค่าสูงขึ้น ในระยะแรก TCP จะใช้กระบวนการ Go-back-n ในการให้สัญญาณการ Acknowledgement และใช้การหน่วงเวลาเพื่อทำกระบวนการ Retransmit เพื่อทำการส่งข้อมูลซ้ำในระหว่างการส่งข้อมูล สิ่งเหล่านี้เป็นส่วนช่วยให้เครือข่ายมีความคับคั่งไม่มากนัก

Tahoe TCP จะมีทั้งจำนวนอัลกอริธึมใหม่และส่วนของการปรับปรุง เพื่อให้ใช้งานได้ง่ายขึ้น อัลกอริธึมนี้ประกอบด้วย Slow-Start, Congestion Avoidance และ Fast Retransmit ส่วนการปรับปรุงนั้นก็ประกอบด้วยการใช้ค่า round-trip time ที่ใช้ในการตั้งค่าเวลา (Timeout) นับถอยหลังในการ Retransmit

ส่วนของอัลกอริธึมในการ Fast Retransmit เป็นส่วนที่ได้ถูกเพิ่มเติมขึ้นในภายหลัง โดยมีหลักการดังนี้คือ หลังจากได้รับสัญญาณ Acknowledgement ซ้ำๆ กันแล้ว ผู้ส่งข้อมูลจะอนุมานว่า แพ็กเก็ตได้เกิดการผิดพลาดขึ้นและจะทำการส่งข้อมูลซ้ำ (Retransmit) โดยไม่มีการรอให้หมดเวลา (Timeout) ในการตอบรับข้อมูลและจะเป็นการนำไปสู่การใช้ช่องสัญญาณที่คุ้มค่ารวมทั้งให้ค่า Throughput ที่ดีขึ้น



รูปที่ 2.4 Flowchart ของ Tahoe TCP

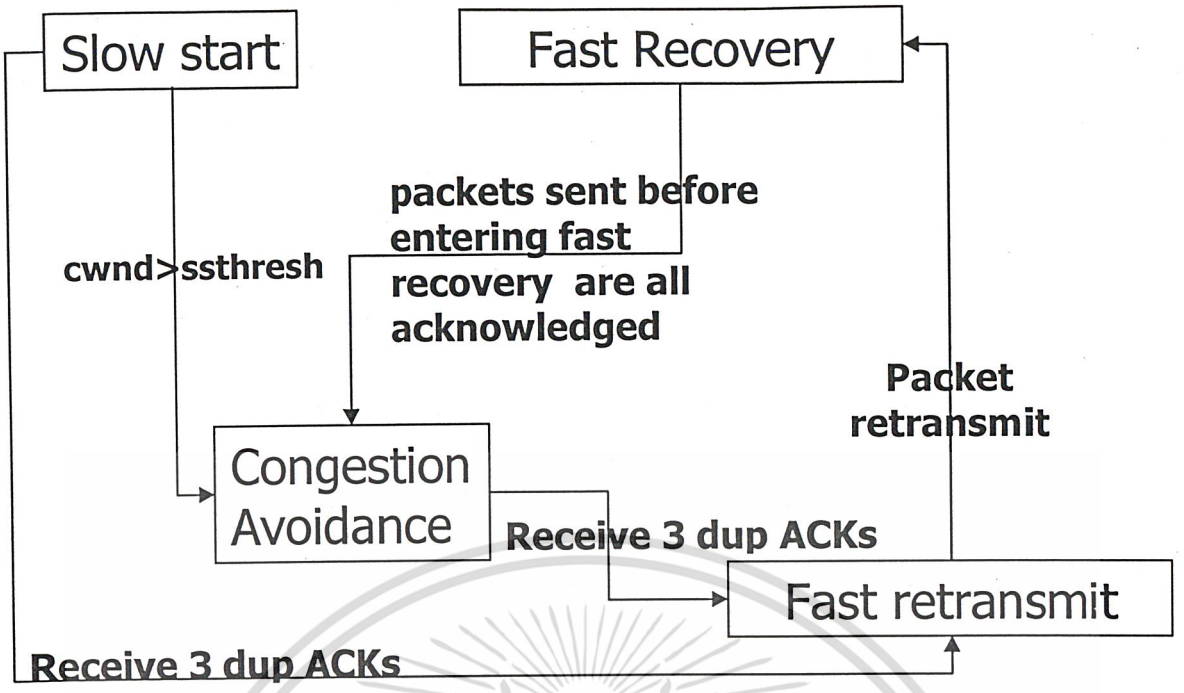
2.3.2 Reno TCP

Reno TCP นั้นมีรูปแบบเหมือนกับ Tahoe แต่มีการปรับปรุงกระบวนการ Fast Retransmit โดยเพิ่มการ Fast Recovery อัลกอริทึมใหม่นี้จะป้องกันเส้นทางการสื่อสาร (ท่อ) ที่จะว่างลงหลังจากการ Fast Retransmit และเพื่อหลีกเลี่ยงความจำเป็นในการเริ่มต้นทำกระบวนการ Slow-Start หลังจากที่แพ็กเก็ตสูญหายไป กระบวนการ Fast Recovery จะทำงานหลังจากการรับสัญญาณ ACK ที่ถูกส่งซ้ำมาเมื่อแพ็กเก็ตได้ออกจากท่อ (pipe)ไปแล้ว 1 แพ็กเก็ต ดังนั้นในระหว่างการ Fast Recovery ตัวส่ง TCP จะสามารถคาดคะเนจำนวนข้อมูลที่ยังคงไม่ถูกส่งออกไปได้

กระบวนการ Fast Recovery จะเริ่มขึ้นโดยตัวส่งทีซีพีได้รับ Threshold ของสัญญาณ ACK ที่ส่งซ้ำมา Threshold นี้รู้จักกันในชื่อ tcprexmtthresh ที่ถูกตั้งค่าไว้ที่ 3 แต่ละครั้งของการรับ Threshold ของสัญญาณ ACK ที่ส่งซ้ำมา ผู้ส่งจะทำการ Retransmit 1 แพ็กเก็ตและลดความคับคั่งของวินโดว์ลง 1/2 เท่า ใน Reno ตัวส่งจะใช้การเพิ่มของ ACK ที่ส่งซ้ำมาเพื่อกำหนดสัญญาณ clock ของแพ็กเก็ตขาออก แทนการ Slow-start ที่เป็นรูปแบบของ Tahoe TCP

ใน Reno ความสามารถในการใช้งานวินโดว์ของตัวส่งเป็น min(awin,cwnd+ndup) ซึ่ง awin เป็นวินโดว์ของตัวรับ, cwnd เป็น Congestion Window ความคับคั่งของวินโดว์ของตัวส่ง และ ndup ถูกตั้งค่าไว้ที่ 0 จนถึงจำนวนของ ACK ที่ถูกส่งซ้ำมาถึงค่า Threshold tcprexmtthresh ดังนั้นระหว่างการ Fast Recovery ตัวส่งจะทำการขยายขนาดวินโดว์จากการได้รับ ACK ที่ถูกส่งซ้ำ จากการสังเกตในแต่ละ ACK ที่ถูกส่งซ้ำนั้นบ่งชี้ว่าบางแพ็กเก็ตได้ถูกส่งออกจากเน็ตเวิร์คและไปถึงยังด้านรับแล้ว หลังจากกระบวนการ Fast RecoveryและRetransmitting 1 แพ็กเก็ตแล้ว ด้านส่งจะรอจนกระทั่งได้รับสัญญาณ ACK ที่ส่งซ้ำกลับมาเพียงครึ่งวินโดว์ และจะส่งแพ็กเก็ตใหม่สำหรับแต่ละการส่ง ACK ซ้ำกลับมา การรับสัญญาณ ACK ของ Data ใหม่ที่เรียกว่าการ Recovery ACK นั้นด้านส่งจะทำการ Fast Recovery โดยการตั้งค่า ndup เป็น 0

อัลกอริทึมในการ Fast Recovery ของ Reno ถูกใช้ในกรณีที่แพ็กเก็ตเดี่ยวถูก drop จากวินโดว์ของข้อมูล ที่ด้านส่งใน Reno จะ Retransmit แพ็กเก็ตที่ drop 1 แพ็กเก็ตต่อ round-trip time Reno ได้ปรับปรุงจาก Tahoe TCP เมื่อแพ็กเก็ต 1 แพ็กเก็ตถูก drop จากวินโดว์ของข้อมูลจะมีผลจากปัญหาเมื่อแพ็กเก็ตหลายๆอันถูก drop ลงจากวินโดว์ของข้อมูล ปัญหานี้ง่ายในการ simulates เมื่อเชื่อมต่อ Reno TCP ด้วยวินโดว์ขนาดใหญ่ที่จะส่งผลต่อแพ็กเก็ตที่สูญเสียหลังจากกระบวนการ Slow- Start ในเครือข่ายด้วย drop-tail gateways



รูปที่ 2.5 Flowchart ของ Reno TCP

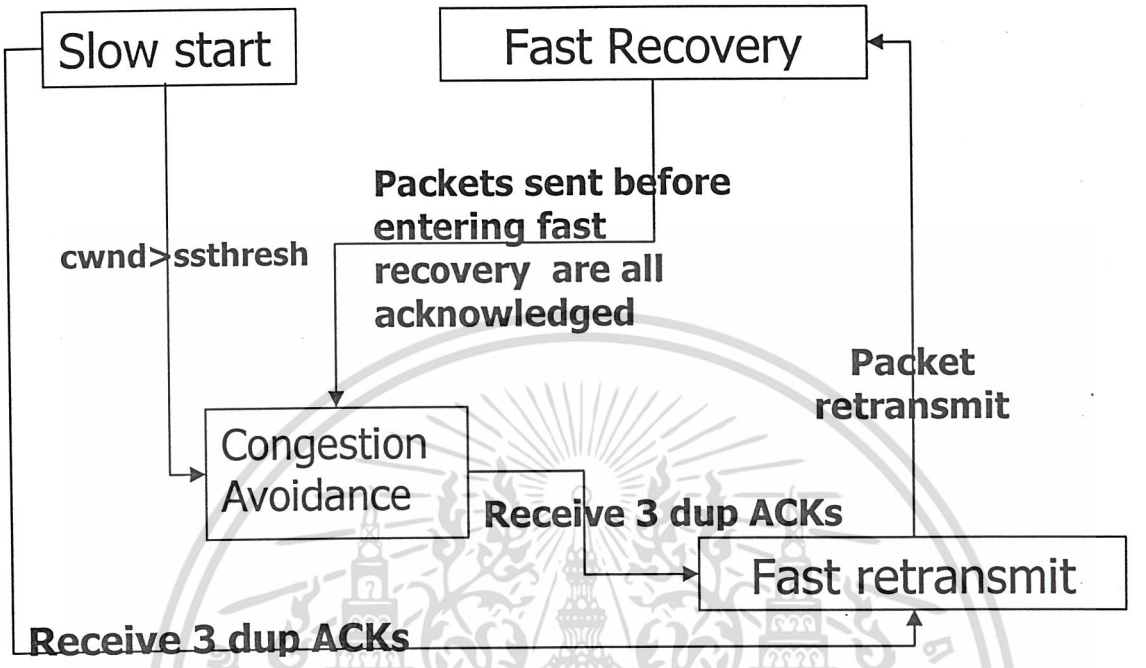
2.3.3 NewReno TCP

NewReno TCP แสดงการเปลี่ยนแปลงอย่างง่ายๆของ TCP ที่จะหลีกเลี่ยงบางปัญหาของ Reno TCP โดยปราศจากการเพิ่มของ SACK ในเวลาเดียวกัน เราใช้ NewReno TCP เพื่อสำรวจข้อจำกัดพื้นฐานของคุณสมบัติของTCP ที่ขาดไปของ SACK

NewReno TCP จะมีการเปลี่ยนแปลงของ Reno algorithm ที่ด้านส่งจะไม่รอเวลาการ Retransmit ของ Reno เมื่อแพ็กเก็ตหลายๆ อันได้สูญเสียจากวินโดว์ การเปลี่ยนแปลงเกี่ยวข้องกับพฤติกรรมของด้านส่งระหว่างการ Fast Recovery เมื่อส่วนเล็กๆของ ACK ถูกรับได้แต่ไม่ทั้งหมด ซึ่งจะค้างอยู่ที่จุดเริ่มของช่วงระยะเวลาของการ Fast Recovery ใน Reno ส่วนเล็กๆของ ACK จะนำ TCP ออกจากการ Fast Recovery โดยการทำให้น้อยลง (deflating) การใช้วินโดว์จะกลับไปยังขนาดของ Congestion Window ส่วนใน NewReno ส่วนเล็กๆ ของ ACK จะไม่นำ TCP ออกจากการ Fast Recovery การแทนด้วยส่วนเล็กๆ ของ ACK ที่ได้รับระหว่างการ Fast Recovery จะถูกแก้ไขด้วยการบ่งชี้ว่า แพ็กเก็ต จะติดตามสัญญาณ ACK ในลำดับของส่วนที่สูญหายและจะทำการ Retransmit ดังนั้นเมื่อแพ็กเก็ตหลายๆอันสูญหายในวินโดว์ของข้อมูล NewReno จะสามารถสร้างใหม่ได้ โดยปราศจากกระบวนการรอการRetransmission การRetransmit ของแพ็กเก็ต 1 แพ็กเก็ตที่สูญหายต่อ round-trip time จนกระทั่งการสูญหายของแพ็กเก็ตทั้งหมดจากวินโดว์จะได้ทำการRetransmitแล้ว NewReno ยังคงอยู่ในกระบวนการ Fast Recovery จนกระทั่งส่วนทั้งหมดของข้อมูลที่ยังค้างอยู่ เมื่อ Fast Recovery ได้เริ่มการส่ง ACK

การแนะนำการเปลี่ยนแปลงเพิ่มเติมของอัลกอริทึมการ Fast Recovery ของ TCP ได้แนะนำว่าด้านส่งข้อมูลจะส่งแพ็กเก็ตใหม่สำหรับทุกๆ สองค่าของการส่งสัญญาณ ACK ที่ได้รับระหว่างการ Fast

Recovery เพื่อเป็นการลดภาระของการ ACK สิ่งนี้มิใช่เครื่องมือใน NewReno เพราะว่าเราต้องการที่จะพิจารณาชุดของการเปลี่ยนแปลงของ Reno ที่จะจำเป็นในการหลีกเลี่ยงการรอเวลา (Timeout) โดยไม่จำเป็นของการ Retransmission



รูปที่ 2.6 Flowchart ของ NewReno TCP

2.3.4 SACK TCP

ดังเช่น Reno นั้น SACK TCP จะใช้การ Fast Recovery เมื่อด้านส่งข้อมูลได้รับ tcprexmtthresh ซ้ำด้านส่งจะทำการ Retransmit แพ็กเก็ตและตัดความคับคั่งของวินโดว์ (Congestion Window) ลงครึ่งเท่าระหว่างการ Fast Recovery SACK จะปรับค่าที่เรียกว่า pipe ซึ่งแสดงจำนวนของแพ็กเก็ตที่ค้างอยู่ในเส้นทาง ด้านส่งจะส่งข้อมูลใหม่ หรือส่งข้อมูลเดิมเมื่อจำนวนของแพ็กเก็ตในเส้นทางไม่น้อยกว่าค่าความคับคั่งของวินโดว์ (Congestion Window) ค่าของ pipe จะเพิ่มขึ้นเมื่อด้านส่งส่งแพ็กเก็ตใหม่หรือส่งแพ็กเก็ตเดิมซ้ำ (Retransmit) และจะลดลงเมื่อด้านส่งได้รับแพ็กเก็ตของ ACK ซ้ำๆ ที่มี SACK ซึ่งข้อมูลใหม่จะถูกรับที่ด้านรับ

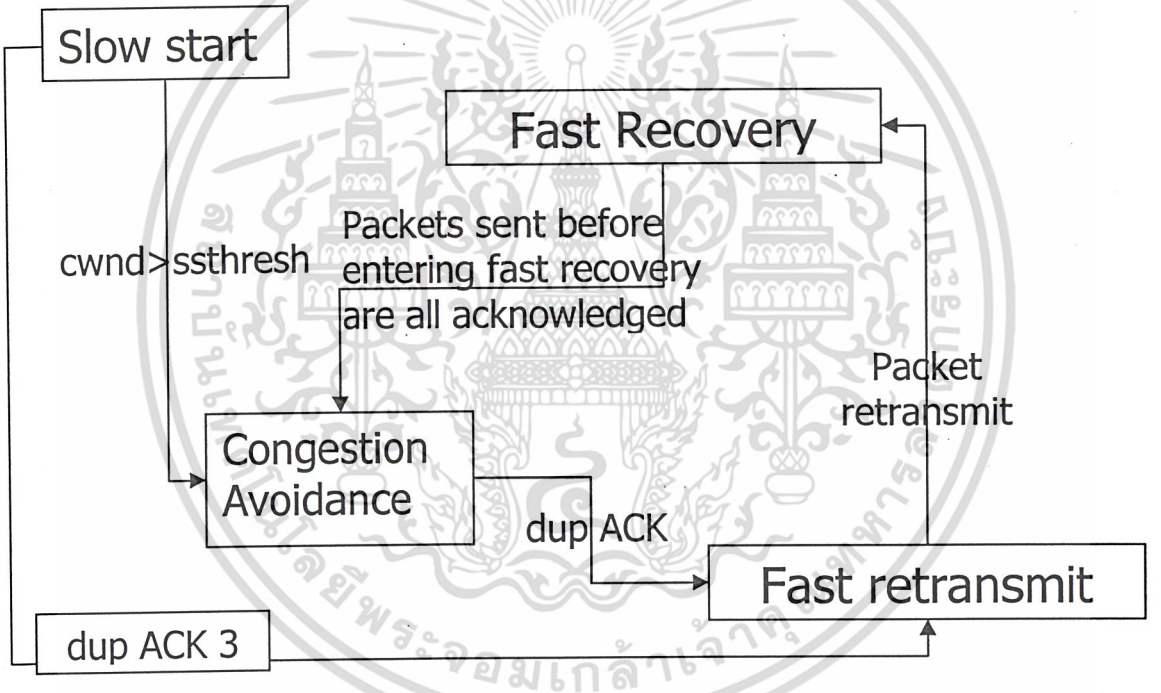
การใช้ pipe จะใช้เมื่อส่งแพ็กเก็ตจากการตัดสินของการส่งแต่ละแพ็กเก็ต ผู้ส่งจะรักษาโครงสร้างของข้อมูล ซึ่งจะจำการ acknowledgement ได้จาก SACK ก่อนหน้านี้ เมื่อผู้ส่งได้ยอมให้ส่งแพ็กเก็ต มันจะทำการ Retransmit แพ็กเก็ตต่อไปจากรายการของแพ็กเก็ตที่อนุมานจากการผิดพลาดที่ด้านรับ ถ้าแพ็กเก็ตและวินโดว์ของด้านรับมีขนาดใหญ่ไม่ใหญ่มากผู้ส่งจะส่งแพ็กเก็ตใหม่

เมื่อทำการส่ง แพ็กเก็ต ใหม่จะเกิดการ drop ลง การใช้ SACK จะป้องกันการ drop ของ timeoutในการ Retransmit โดยจะส่งแพ็กเก็ตที่ถูก drop และทำการ Slow-start หลังจากนั้น

ด้านส่งจะทำการ Fast Recovery เมื่อได้รับ acknowledgement ของข้อมูลทั้งหมด ซึ่งกำลังค้างอยู่เมื่อได้ทำการ Fast Recovery

ด้านส่ง SACK จะมีการตรวจจับ ACK ในส่วนของ ACK ด้านส่งจะลด pipe ลงโดยใช้ 2 แพ็กเก็ต เมื่อเริ่มการ Fast Retransmit pipe จะถูกลดสำหรับแพ็กเก็ตที่จะถูก drop ลง และจะเพิ่มขึ้นสำหรับแพ็กเก็ตที่ถูก Retransmit ดังนั้นการลดของ pipe ที่เกิดจาก 2 แพ็กเก็ต เมื่อ ACK แรกได้รับและได้แสดงว่าแพ็กเก็ตได้ส่งออกไป อย่างไรก็ตามเพื่อการรับ ACK ได้ pipe จะถูกเพิ่มเมื่อแพ็กเก็ตที่ถูกส่งซ้ำถูกส่งเข้าไปใน pipe แต่จะไม่ลดสำหรับแพ็กเก็ตที่ถูก drop ดังนั้นเมื่อ ACK ส่งมาถึงมันจะแสดงว่า 2 แพ็กเก็ตได้ถูกส่งออกจาก pipe เป็นแพ็กเก็ตเริ่มต้นและแพ็กเก็ตที่ส่งซ้ำ เพราะว่าด้านส่งจะลด pipe ด้วย 2 แพ็กเก็ตที่มากกว่า สำหรับ ACK SACK ที่ด้านส่งจะไม่ทำการเร็วกว่า Slow-start

มีจำนวนของสัดส่วนเพื่อการควบคุมอัลกอริธึมของความคับคั่งของ TCP ที่ใช้การเลือกการ Acknowledgement การใช้ SACK ในการsimulateจะถูกออกแบบให้มีการขยายของอัลกอริธึมที่ควบคุมความคับคั่งของ Reno ซึ่งจะทำให้เกิดการเปลี่ยนของอัลกอริธึมที่ควบคุมความคับคั่งของ Reno เพียงเล็กน้อย



รูปที่ 2.7 Flowchart ของ SACK TCP

2.3.5 Vegas TCP

ทุกวิธีที่นำสู่การพัฒนาของ Tahoe TCP และ Reno TCP ในปี 1996 Brakmo ได้แนะนำ TCP Vegas ที่ดีกว่าโดย Vegas จะเพิ่มกลไกในการ Retransmit ของ Reno ขั้นแรก Vegas จะอ่านและบันทึก clock ของระบบในแต่ละส่วนที่ถูกส่ง เมื่อสัญญาณ ACK มาถึง Vegas จะอ่านค่า clock อีกครั้งและทำการคำนวณแบบ RTT Vegas จะใช้ค่า RTT ที่จะตัดสินว่าจะทำการ Retransmit ใน 2 กรณีดังนี้

1.เมื่อได้รับสัญญาณ ACK ซ้ำ

Vegas จะตรวจความแตกต่างระหว่างค่าปัจจุบัน และบันทึกค่า Timestamp เพื่อให้ส่วนที่สัมพันธ์กันมีความสำคัญกว่าค่า Timeout ถ้าเป็นดังนี้ Vegas จะทำการ Retransmit ในส่วนที่ไม่ต้องรอให้มีการส่งสัญญาณ ACK ซ้ำกันถึง 3 ครั้ง ในกรณีอื่นที่มีการสูญเสีย ด้านส่งจะไม่รับ ACK ทั้ง 3 ครั้ง ดังนั้น Reno จะต้องตอบในเวลา Timeout

2. เมื่อไม่มีการรับสัญญาณ ACK ซ้ำ

ถ้า ACK ที่ถูกส่งมาอันแรกหรืออันที่สองหลังจากการ Retransmit Vagas จะตรวจถ้าเวลาช่วงในตั้งแต่ส่วนที่ถูกส่งมีขนาดใหญ่กว่าเวลา Timeout ถ้าเป็นดังนี้ Vegas จะทำการ Retransmit จะทำให้ส่วนอื่นๆ ที่สูญเสียก่อนหน้าที่จะมีการ Retransmit ที่ไม่มีการรอสัญญาณ ACK ที่ซ้ำถึง 3 ครั้ง

สำหรับกระบวนการ Congestion avoidance ของ Vegas จะใช้พื้นฐานการเปลี่ยนแปลงจำนวนของข้อมูลในเครือข่าย Vegas จะกำหนดให้ BaseRTT ของการเชื่อมต่อให้เป็น RTT ของเซ็กเมนต์เมื่อการเชื่อมต่อไม่มีความคับคั่ง ในทางปฏิบัติ Vegas จะตั้งค่า BaseRTT ให้น้อยที่สุดของ round trip time ที่วัดได้ RTT ของเซ็กเมนต์แรกส่งโดยการเชื่อมต่อ ก่อนที่เราเตอร์จะเข้าคิวเพิ่มขึ้นเนื่องมาจากทราฟฟิกที่สร้างจากการเชื่อมต่อนี้ Vegas ใช้ค่านี้ทำการคำนวณค่า Throughput ที่คาดไว้ ในขั้นที่สอง Vegas จะคำนวณค่าอัตราเร็วในการส่งข้อมูล สิ่งนี้ทำได้โดยบันทึกเวลาในการส่งสำหรับแบ่งเซ็กเมนต์ , บันทึกจำนวนของ bytes ที่ถูกส่งระหว่างเวลาที่เซ็กเมนต์ถูกส่งและได้รับสัญญาณ ACK , ทำการประมวลผล RTT สำหรับแบ่งเซ็กเมนต์เมื่อสัญญาณ ACK มาถึง และแบ่งจำนวนของ bytes ที่ถูกส่งโดย sample RTT การคำนวณนี้ถูกทำหนึ่งครั้งต่อ round trip time สำหรับในขั้นที่สาม Vegas จะเปรียบเทียบค่า Throughput ที่คาดไว้แล้วและปรับขนาดของวินโดว์ ค่า Diff เป็นค่าความแตกต่างระหว่างค่าของ Throughput ที่ได้จากการปฏิบัติและค่าที่ได้จากการคาดการณ์ไว้ ดังนั้น Vegas จะกำหนดค่า Threshold 2 ค่าคือค่า α และ β ที่จะมีน้อยกว่าและมากกว่าข้อมูลในเครือข่าย ตามลำดับ

cwnd + 1 if Diff < α

cwnd - 1 if Diff > β

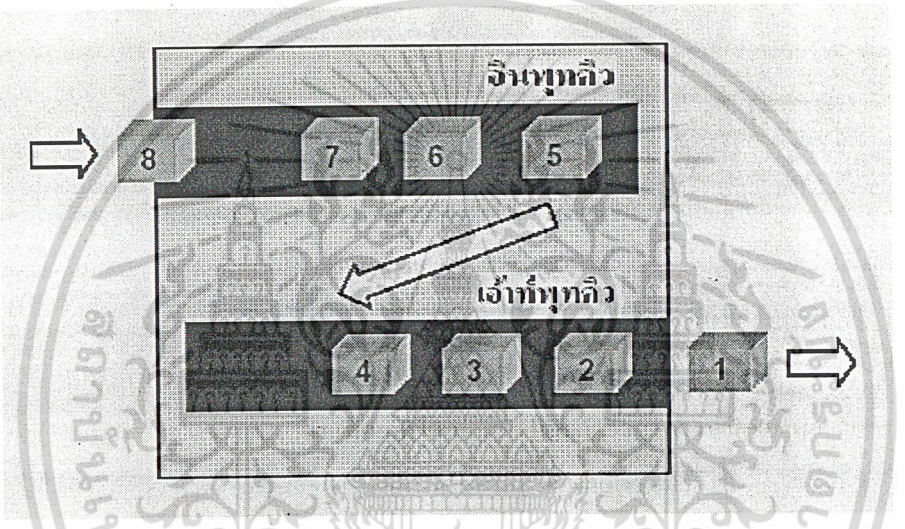
cwnd otherwise($\alpha \leq$ Diff $\leq \beta$)

Vegas มี 3 เทคนิคในการเพิ่ม throughput และลดการสูญเสียของแพ็กเก็ตและการ Retransmit เทคนิคแรกคือ congestion avoidance เป็นการปรับค่าความคับคั่งวินโดว์ของ TCP แบบเส้นตรงทั้งขาขึ้นและขาลงโดยจะมีค่าที่คงที่ของจำนวนบัฟเฟอร์ในเครือข่าย อัลกอริธึมของ congestion avoidance ของ Vegas จะรักษาจำนวนของส่วนในการส่งให้พอที่จะสามารถตั้งสัญญาณ clock ได้ เพื่อการป้องกันเทคนิคที่ 2 จะใช้ 2 สิ่งที่จะป้องกันการสูญหายของแพ็กเก็ตง่ายกว่า Reno และใช้การลดที่ต่ำกว่า Reno ในเทคนิคที่ 3 Vegas จะลดลงครึ่งหนึ่งในการเพิ่มของอัตราการ Slow-start ที่จะป้องกันการสูญหายของแพ็กเก็ต

2.4 ทฤษฎีและหลักการทำงานของการจัดการคิวแบบ DropTail และ RED Queuing

2.4.1 First In First Out (FIFO), DropTail Queuing

FIFO Queuing เป็นกลไกการทำงานพื้นฐานที่สุด ซึ่งในการทำงานของมันนั้นแพ็กเก็ต จะถูกนำเข้ามาอย่างต่อเนื่องภายในระบบเครือข่าย ซึ่งภายในนี้จะมีคิวอยู่เพียงคิวเดียวเท่านั้นเป็นคิวเดี่ยว ซึ่งมันไม่ต้องการการจัดแบ่งหรือการจัดเรียงของแพ็กเก็ตใดๆเลย โดยจะเข้ามาทางท้ายของคิว และเคลื่อนย้ายออกไปทางส่วนหัวของคิว FIFO เป็นวิธีการดั้งเดิมที่ใช้กันกันอย่างแพร่หลายในอุปกรณ์เครือข่ายอย่างง่าย แต่อย่างไรก็ดีมันไม่ได้เป็นการทำงานที่ดี ในการที่จะให้บริการระบบที่มีความหนาแน่นของปริมาณทราฟิกสูงๆ เช่น ไม่ได้มีความยืดหยุ่นในระบบ QoS ซึ่งต้องการการให้บริการเฉพาะอย่างที่แตกต่างออกไป ซึ่งโครงสร้างของ FIFO จะแสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 โครงสร้างของ FIFO Queuing

ซึ่งลักษณะของ FIFO Queuing ก็คือ

- จะมีความเร็วและง่ายที่สุดเพราะแพ็กเก็ตที่เข้ามาในระบบจะได้รับการบริการทันที
- ไม่มีการแยกแยะชนิดของข้อมูลที่เข้ามาว่ามีความแตกต่างกันอย่างไร
- DropTail คือวิธีการจัดการบัฟเฟอร์ของ FIFO Queuing
- วิธีการนี้เมื่อมีแพ็กเก็ตของข้อมูลจำนวนมากไหลเข้ามาในคิวของระบบจนเต็ม แพ็กเก็ตต่างๆที่เหลือจะถูกหยุด (drop) ไม่สามารถเข้ามาในระบบได้ ทำให้มีปัญหาเรื่อง การสูญเสียของแพ็กเก็ตมาก

2.4.2 Random Early Detection (RED) Queuing

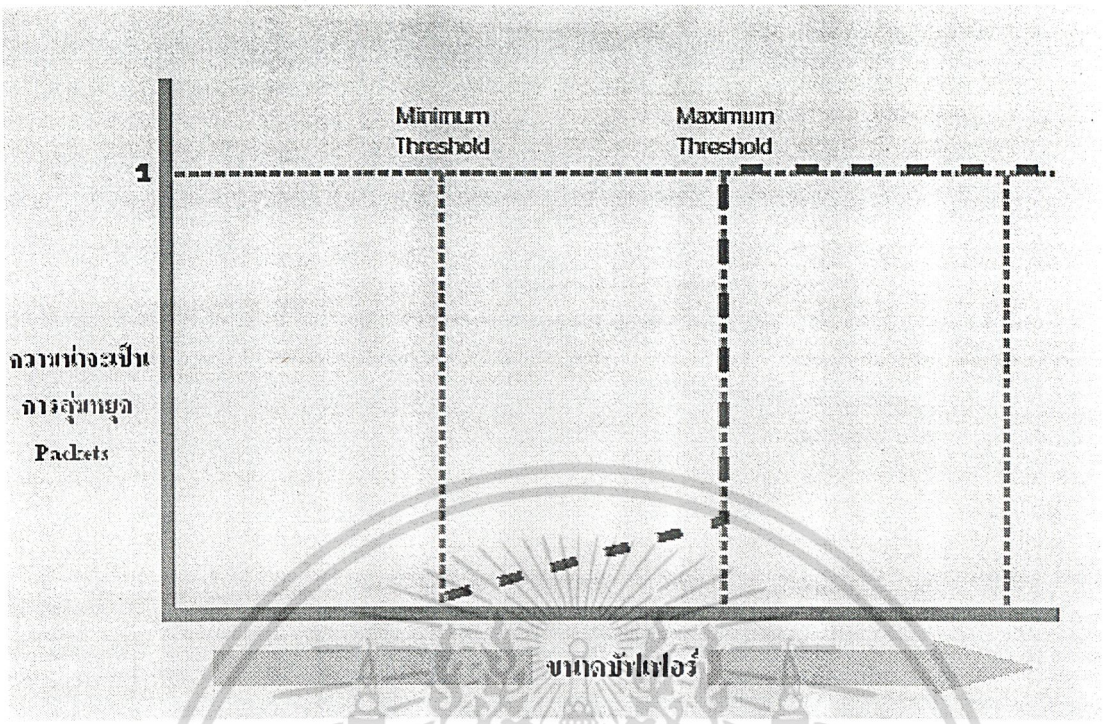
การที่การทำงานบนเครือข่ายไม่สามารถทำงานได้เต็มประสิทธิภาพ อาจจะเป็นเพราะมีความหนาแน่นของทราฟฟิกสูงเกินไป จึงทำให้มีการลดทอนประสิทธิภาพของการทำงานลงไป วิธีการแบบ RED นี้ถูกออกแบบมาสำหรับปรับปรุงประสิทธิภาพของระบบเครือข่ายให้ดีขึ้น โดยการปรับปรุงที่ต่างออกไปจากคิวแบบ FIFO ซึ่งมันมีจุดประสงค์คือ

- ตรวจจับความหนาแน่นในระบบเครือข่ายเบื้องต้น
- อนุญาตให้มีความหนาแน่นของปริมาณทราฟฟิกสูงๆ ได้ชั่วคราวโดยใช้หลักการดูดซับ (Absorbing) ความหนาแน่นของแพ็กเก็ตข้อมูลเอาไว้
- ป้องกันการเกิดความหนาแน่นเพิ่มมาอีก
- หลีกเลี่ยงการเกิดความหนาแน่นขึ้นพร้อมๆกันของการเชื่อมต่อต่างๆ โดยใช้การเลือกสุ่มการเชื่อมต่อหรือการไหลของแพ็กเก็ตที่ถูกทำเครื่องหมายไว้

เป้าหมายของ RED ที่จะทำตามดังที่กล่าวก็คือ มันจะมีการทำเครื่องหมายหรือสัญลักษณ์ลงบนความยาวเฉลี่ยของคิวกับ จุดจำกัด ต่ำสุดและสูงสุด (Minimum & Maximum Threshold) โดยค่าของความยาวเฉลี่ยของใน RED คิวนี้ จะใช้การคำนวณจากทุกช่วงเวลาที่ มี Packets เข้าออกจากคิวนั่นเอง ซึ่งก็จะทำให้ได้ค่าของความยาวเฉลี่ยของคิว

รูปที่ 2.9 การหาความยาวเฉลี่ยของคิว

ซึ่งถ้าความยาวเฉลี่ยของคิวมีค่าต่ำกว่าจุดจำกัดต่ำสุด แพ็กเก็ตต่างๆ ก็จะไหลเข้าไปในคิวโดยที่จะยังไม่มีการทำเครื่องหมายลงบนแพ็กเก็ตเหล่านี้ แต่เมื่อมีความเป็นไปได้ในการที่จะมีปริมาณของแพ็กเก็ตเพิ่มสูงขึ้นจนทำให้ค่าความยาวเฉลี่ยของคิวเพิ่มสูงขึ้นจนเกินระดับจุดจำกัดต่ำสุด ก็จะมีการทำเครื่องหมายไว้บนแพ็กเก็ตบางส่วน และจะหยุดแพ็กเก็ตบางส่วนเหล่านี้ไว้ เพื่อที่จะทำให้ยังสามารถรองรับปริมาณทราฟฟิกภายในคิวนี้ได้หรืออาจเรียกการทำงานแบบนี้ว่าเป็นการดูดซับ (Absorbing) ความหนาแน่นก็ได้ แต่เมื่อมีจำนวนของแพ็กเก็ตที่ไหลเข้ามาสูงมากจนทำให้ความยาวเฉลี่ยของคิวนี้สูงเกินจุดจำกัดสูงสุด แพ็กเก็ตทั้งหมดก็จะถูกหยุดไว้และระบบก็จะไม่สามารถใช้งานได้ซึ่งสามารถแสดง ได้ดังรูปกราฟที่ 2.10



รูปที่ 2.10 กราฟแสดงการทำงานของ RED

ซึ่งวิธีการทำงานของ RED นั้นสามารถแบ่งได้เป็น 2 ขั้นตอนก็คือ

อันดับแรก มันจะคำนวณหาค่าของความยาวเฉลี่ยของคิวจากจุดจำกัด ต่ำสุดและสูงสุดนี้ในทันทีที่เริ่มทำงานซึ่งจะขึ้นอยู่กับการตั้งค่าของอุปกรณ์อย่างเกตเวย์เพื่อรองรับทราฟฟิกเป็นต้น

อันดับสอง เป็นวิธีการที่จะเลือกสุ่มทำเครื่องหมายแพ็กเก็ตที่จะทำการหยุด ซึ่งจะให้มีความถี่เท่าใดที่จะทำการหยุดแพ็กเก็ต ซึ่งจากกราฟรูปที่ 2.10 นั้นจำนวนของแพ็กเก็ตที่จะถูกหยุดนั้นจะสามารถเขียนได้เป็นความน่าจะเป็นที่จะหยุดจำนวนของแพ็กเก็ตเท่าใดจากจำนวนแพ็กเก็ตทั้งหมด (Discard Probability) ซึ่งสามารถเขียนได้ดังสมการที่ (2.1)

$$p_b = \max_p \frac{(avg - \min_{th})}{(\max_{th} - \min_{th})} \qquad (2.1)$$

ทั้งนี้ถ้าความหนาแน่นของปริมาณทราฟฟิกยังคงมีอยู่ต่อไป RED ก็จะทำการสุ่มเพื่อทำการหยุดแพ็กเก็ตเพิ่มขึ้นเรื่อยๆ จนกว่าจะมีความหนาแน่นที่ลดลง ซึ่งความถี่ในการที่จะสุ่มทำเครื่องหมายกับแพ็กเก็ตต่างๆที่เพิ่มขึ้นอยู่กับ ความยาวเฉลี่ยของคิวที่จะเข้าถึงจุดจำกัดสูงสุดนั่นเอง

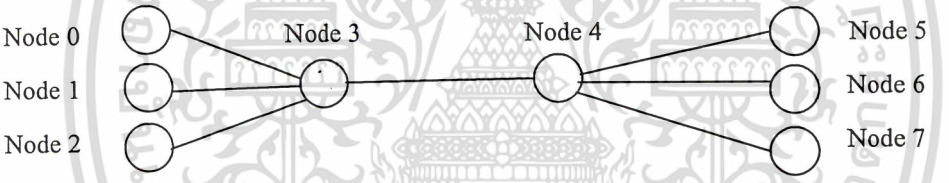
บทที่ 3
การทดลองและผลการทดลอง

3.1 การทดสอบประสิทธิภาพเมื่อพิจารณาจำนวนแพ็กเก็ตข้อมูลที่สูญหายในหน้าต่างข้อมูล

การทดลองในส่วนนี้ เป็นการทดสอบเพื่อพิจารณาการทำงานของรูปแบบของ TCP ชนิดต่าง ๆ ตั้งแต่ Tahoe TCP, Reno TCP, Newreno TCP และ SACK TCP เมื่อมีแพ็กเก็ตข้อมูลสูญหายไประหว่างการส่ง โดยพิจารณาจำนวนแพ็กเก็ตข้อมูลที่สูญหายไป (dropped packet) ต่อขนาดหน้าต่างที่ใช้ในการรับส่งข้อมูล (window size) โดยจะแบ่งการพิจารณาออกเป็น 4 ส่วน ซึ่งแต่ละแบบจะมีผลต่อรูปแบบของ TCP ที่แตกต่างกัน ดังนี้

1. จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 1 แพ็กเก็ต (One Packet loss)
2. จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 2 แพ็กเก็ต (Two Packet losses)
3. จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 3 แพ็กเก็ต (Three Packet losses)
4. จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 4 แพ็กเก็ต (Four Packet losses)

รูปแบบโครงสร้างของการจำลองที่ใช้ในการทดสอบ จะใช้รูปแบบเดียวกันทั้ง 4 การทดสอบข้างต้น โดยมีโครงสร้าง ดังรูปที่ 3.1



รูปที่ 3.1 แบบจำลองการทดสอบระหว่าง TCP ชนิดต่าง ๆ

โครงสร้างที่ใช้ในการทดสอบเป็นการเชื่อมต่อ จากด้านส่ง คือ โหนดที่ 0 ถึงโหนดที่ 2 จะเป็นรูปแบบของ TCP แต่ละแบบในการจำลองโดยเชื่อมต่อกับด้านรับ คือ โหนดที่ 5 ถึง โหนดที่ 7 ซึ่งเป็นตัวรับแพ็กเก็ตข้อมูล เรียกว่า TCPSink โดยจะเชื่อมต่อผ่านช่องการเชื่อมต่อ คือ โหนดที่ 3 กับ โหนดที่ 4

ค่าพารามิเตอร์ที่สำคัญที่ใช้ในการจำลอง มีดังนี้

1. ในแต่ละด้าน จะเชื่อมต่อกันด้วย ความเร็ว 10 Mbps และค่า Delay 0.1 ms
2. ระหว่างโหนดที่ 3 กับโหนดที่ 4 เชื่อมต่อกันด้วย ความเร็ว 1 Mbps และค่า Delay 100 ms
3. ใช้การจัดคิวแบบ Droptail

ในการจำลองนั้นจะพิจารณาเฉพาะ โหนดที่ 0 กับโหนดที่ 5 เพื่อพิจารณาถึงประสิทธิภาพของรูปแบบการทำงานของ TCP แต่ละรูปแบบ โดยที่โหนดที่เหลือถูกนำมาใช้เพื่อจำกัดจำนวนแพ็กเก็ตที่จะสูญหายของการเชื่อมต่อของตัวแรก

สำหรับผลการจำลองในแต่ละการทดลอง จะแสดงออกมาในรูปของกราฟ โดยพิจารณาการเดินทางในแต่ละแพ็กเก็ตที่ไปถึงช่องทางการเชื่อมต่อ หรือโหนดที่ 3 โดยที่แกน x แสดงเวลาที่แพ็กเก็ต

มาถึงและส่งแพ็กเก็ตออกไป แกน y แสดงหมายเลขแพ็กเก็ต (packet number) โดยจะถูกทำการ mod 60 เพื่อให้รูปกราฟมีความกระชับและมีความเหมาะสมในการพิจารณา โดยที่สัญลักษณ์สี่เหลี่ยมทึบ จะแทนแพ็กเก็ตข้อมูลที่มาถึงโหนดที่ 3 และแพ็กเก็ตที่จะถูกส่งออกจากโหนดที่ 3 ถ้าแพ็กเก็ตข้อมูลที่มาถึงโหนดที่ 3 แล้วต้องรอคิวในการส่งแพ็กเก็ตออกไป ในกราฟจะเห็นเป็นลักษณะของจุดที่อยู่ใกล้กัน และถ้าแพ็กเก็ตใช้เวลาในการรอน้อยก็จะเห็นเป็นจุด ๆ เดียว สำหรับสัญลักษณ์สี่เหลี่ยมโปร่ง จะแทนค่าการตอบกลับ (ACK) และสัญลักษณ์วงกลม จะแทนแพ็กเก็ตข้อมูลที่สูญหายไปจากการส่ง (Dropped Packet)

3.1.1 จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 1 แพ็กเก็ต (One Packet loss)

การจำลองในส่วนนี้จะพิจารณาเมื่อจำนวนแพ็กเก็ตที่สูญหายมีจำนวน 1 แพ็กเก็ตต่อหน้าต่างข้อมูล (window) จะมีลักษณะดังรูปที่ 3.2 ถึง 3.5 ซึ่งจะแสดงการทำงานในแต่ละรูปแบบของ TCP

การทำงานของ TCP แบบ Tahoe จะแสดงในรูปที่ 3.2 จะมีการทำงานดังนี้ TCP จะเริ่มการส่งข้อมูลในรูปแบบการส่งแบบ Slow-Start ก็คือจะทำงานเริ่มต้นอย่างช้า ในการส่งแพ็กเก็ตที่ 0 -13 โดยจะทำการเพิ่มจำนวนแพ็กเก็ตข้อมูลที่ส่งในลักษณะ exponential เมื่อถึงการส่งแพ็กเก็ตที่ 14 จำนวนข้อมูลในโหนดที่ 3 (router) เต็ม ทำให้แพ็กเก็ตที่ 14 ถูกทิ้งไป (drop) จากแถวคิวการรอคอย โดยที่จำนวนแพ็กเก็ตข้อมูลก่อนหน้านี้จำนวน 7 แพ็กเก็ต (แพ็กเก็ตที่ 7-13) ถูกส่งสำเร็จได้รับการตอบกลับมา ทำให้ด้านส่ง (โหนดที่ 0) ทำการส่งแพ็กเก็ตออกไปอีก 14 แพ็กเก็ต คือ แพ็กเก็ตที่ 15-28

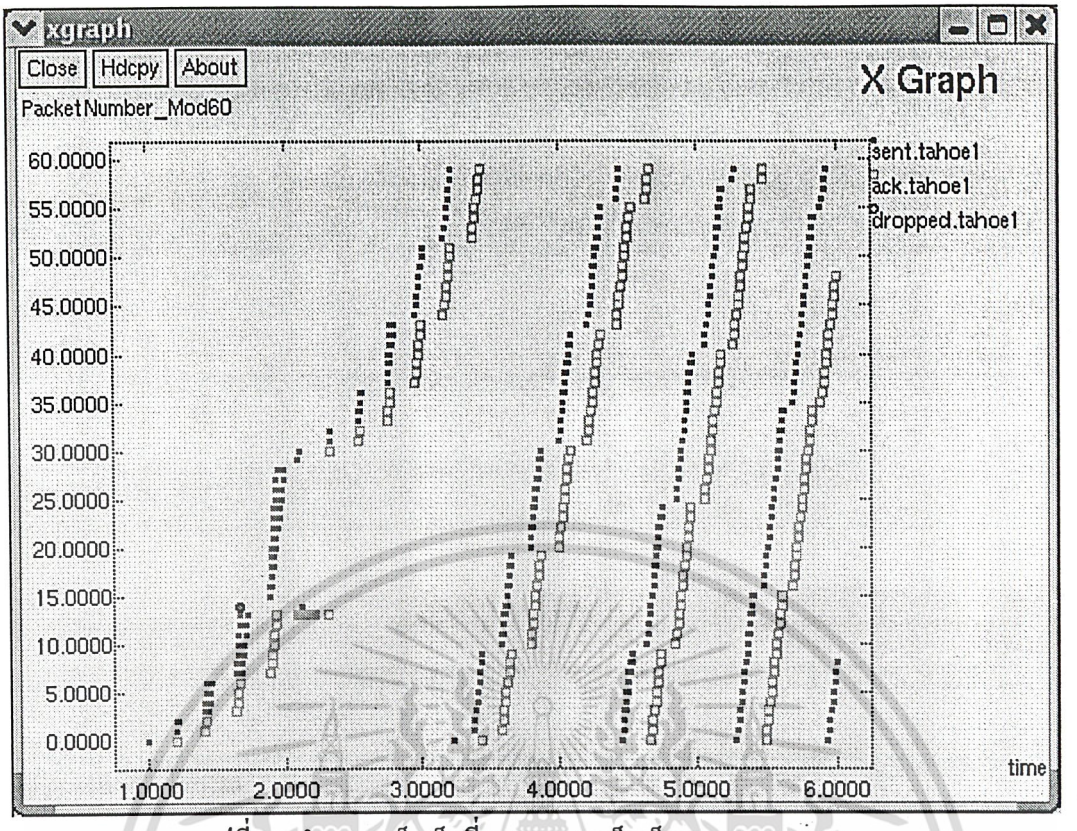
เมื่อด้านส่งได้รับ ACK ตัวแรกจากแพ็กเก็ตที่ 13 แล้วหลังจากนั้นแล้วก็จะได้รับ ACK จากหมายเลขแพ็กเก็ตที่ 13 อีก 16 ตัวซึ่ง ACK ประเภทนี้เรียกว่า duplicate ACK (dup ACK) โดยที่ dup ACK จำนวน 16 ตัวนี้มาจากด้านรับที่ทำการตอบกลับแพ็กเก็ตที่ 15-30 เมื่อด้านส่งได้รับ dup ACK ตัวที่ 3 (เป็น ACK ตัวที่ 4 ของแพ็กเก็ตที่ 13) ซึ่งถึงค่า duplicate ACK threshold ทำให้ TCP จะเข้ากระบวนการ Fast Retransmission และ Slow-Start และทำการกำหนดค่า ssthresh (Slow-Start threshold) โดยลดลงเหลือเท่ากับ 7 และทำการลดค่า cwnd (Congestion Window) เป็น 1 และทำการส่งแพ็กเก็ตที่ 14 ซ้ำไปใหม่

เมื่อด้านรับได้รับแพ็กเก็ตที่ 15-30 และได้รับแพ็กเก็ตที่ 14 อีกครั้ง และเมื่อด้านส่งได้รับ dup ACK ทั้งหมด ก็จะเข้าสู่กระบวนการ Slow-Start ซึ่งจะทำการเพิ่มขนาด cwnd ทีละ 1 โดยเริ่มจากส่งแพ็กเก็ตที่ 31

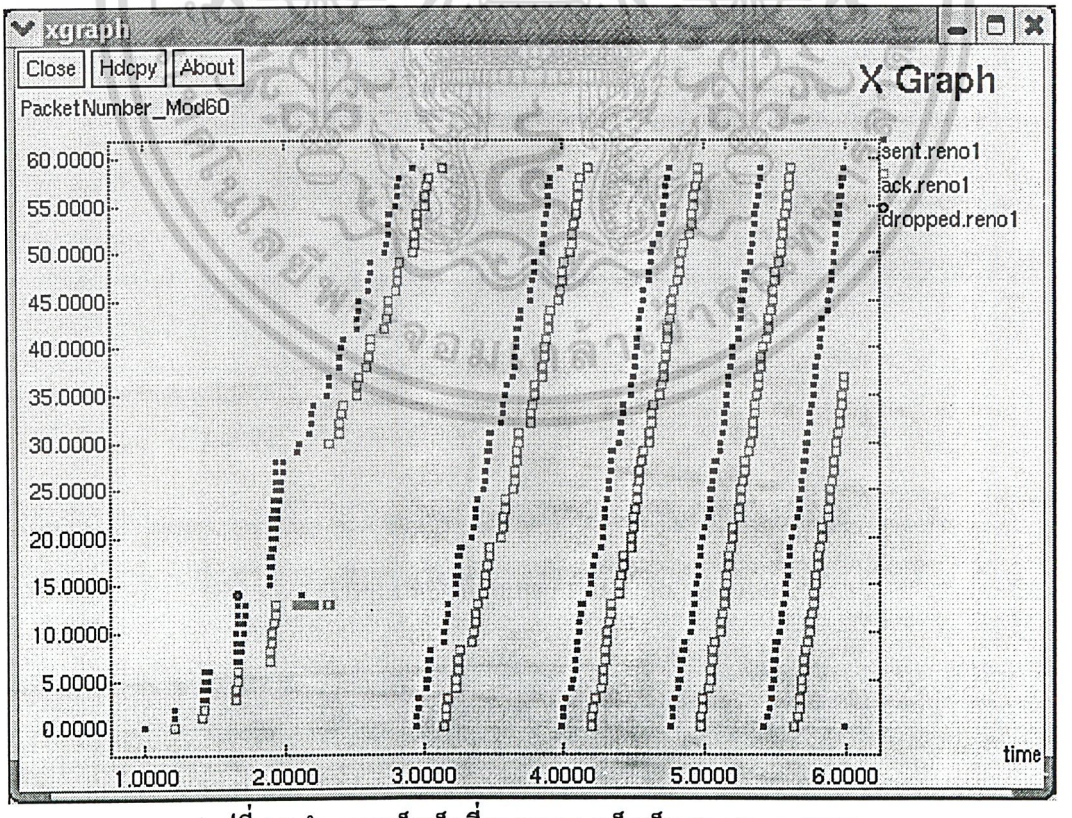
เมื่อถึงการส่งแพ็กเก็ตที่ 43 ก็จะถึงค่า ssthresh ซึ่งถูกกำหนดจากก่อนหน้านี้ซึ่ง เท่ากับ 7 ก็จะเข้าสู่กระบวนการ Congestion Avoidance ซึ่งจะทำการเพิ่มขนาดหน้าต่างทีละ 1 ต่อรอบการส่ง

การทำงานของ TCP แบบ Reno จะแสดงในรูปที่ 3.3 กระบวนการในการทำงานตอนเริ่มต้นจะเหมือนกับ TCP แบบ Tahoe จนกระทั่งได้รับ dup ACK ตัวที่ 3 ซึ่ง Reno จะใช้กระบวนการ Fast Recovery โดยทำการลดค่า cwnd และ ssthresh ให้เท่ากับ 7 ในระหว่าง Fast Recovery ค่า usable window จะเพิ่มจากค่า cwnd โดยจำนวน dup ACK เมื่อได้รับ dup ACK 6 ตัวสุดท้าย ตัวส่งจะสามารถส่งแพ็กเก็ตที่ 31-36 ออกไปได้ เมื่อได้รับ ACK จากแพ็กเก็ตที่ 30 ก็จะออกจาก Fast Recovery ไปสู่ Congestion Avoidance ด้วยค่า cwnd เท่ากับ 7

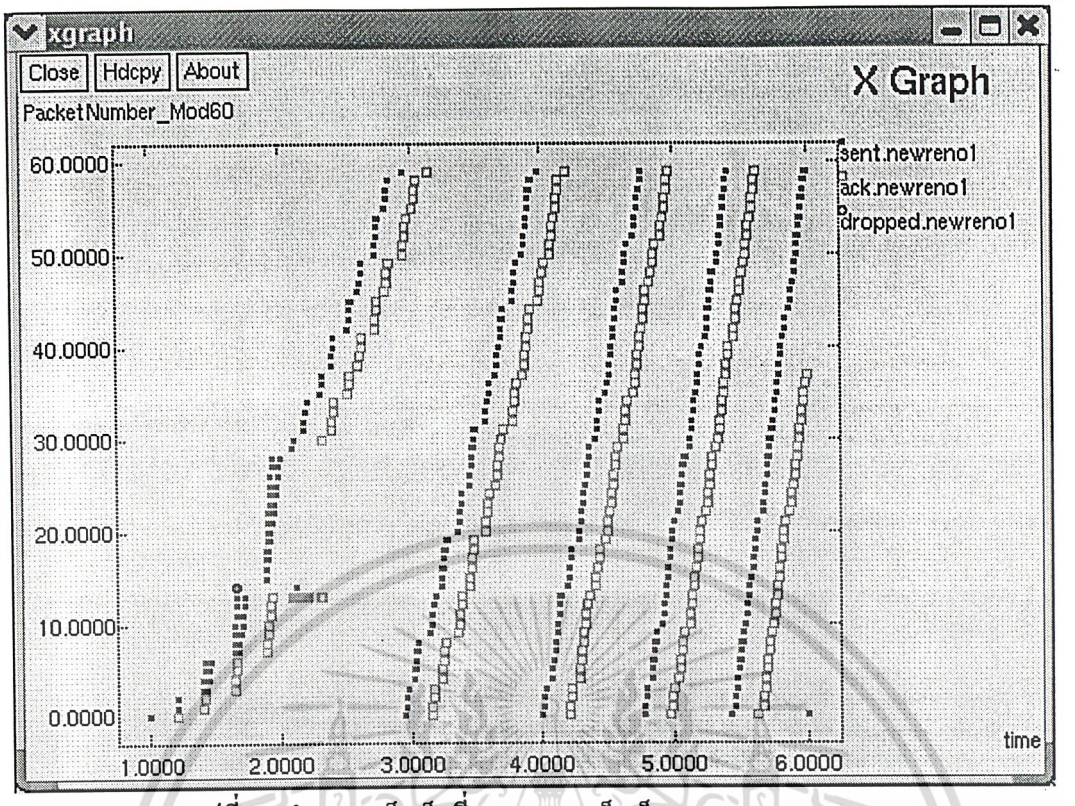
สำหรับการทำงานของ TCP แบบ NewReno และ SACK จะเหมือนกับแบบ Reno TCP โดยมีลักษณะดังรูปที่ 3.4 และ 3.5 ตามลำดับ



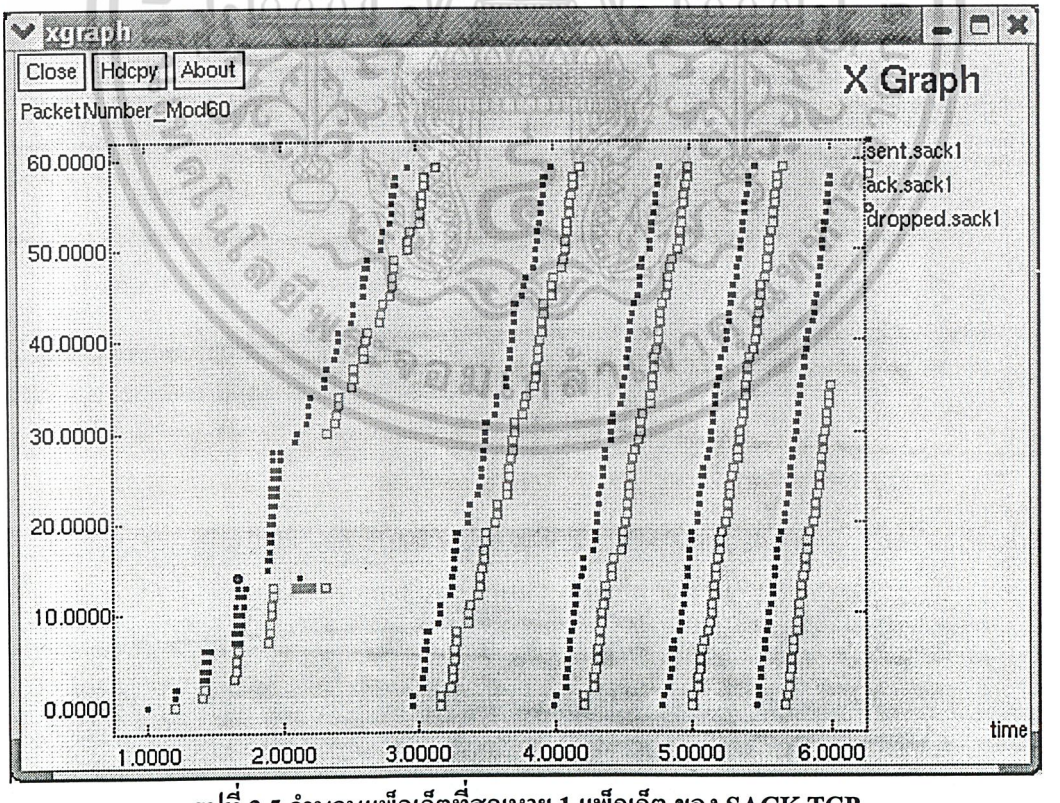
รูปที่ 3.2 จำนวนแพ็กเก็ตที่สูญหาย 1 แพ็กเก็ต ของ Tahoe TCP



รูปที่ 3.3 จำนวนแพ็กเก็ตที่สูญหาย 1 แพ็กเก็ต ของ Reno TCP



รูปที่ 3.4 จำนวนแพ็กเก็ตที่สูญหาย 1 แพ็กเก็ต ของ NewReno TCP



รูปที่ 3.5 จำนวนแพ็กเก็ตที่สูญหาย 1 แพ็กเก็ต ของ SACK TCP

3.1.2 จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 2 แพ็กเก็ต (Two Packet losses)

การจำลองในส่วนนี้จะพิจารณาเมื่อจำนวนแพ็กเก็ตที่สูญหายมีจำนวน 2 แพ็กเก็ตต่อหน้าต่างข้อมูล (window) จะมีลักษณะดังรูปที่ 3.6 ถึง 3.9 ซึ่งจะแสดงการทำงานในแต่ละรูปแบบของ TCP

การทำงานของ TCP แบบ Tahoe จะแสดงในรูปที่ 3.6 กระบวนการทำงานในช่วงที่ทำการส่งแพ็กเก็ตที่ 14 ซ้ำไปอีกครั้ง (ซึ่งเป็นแพ็กเก็ตตัวแรกที่สูญหาย) จะมีรูปแบบการทำเหมือนกับในแบบแรก หลังจากนั้น เมื่อได้รับ dup ACK จำนวน 16 ตัวแล้ว จะได้รับ ACK จากแพ็กเก็ตที่ 27 ก็จะเข้ากระบวนการ Slow-Start โดยเริ่มจากการส่งแพ็กเก็ตที่ 28 เมื่อส่งแพ็กเก็ตที่ 41 ก็จะถึงค่า ssthresh หลังจากนั้น ก็จะเข้าสู่กระบวนการ Congestion Avoidance

การทำงานของ TCP แบบ Reno จะแสดงในรูปที่ 3.7 สำหรับการทำงานโดยทั่วไปของ Reno TCP จะต้องทำการรอคอยค่าเวลา retransmission timeout สำหรับกรณีที่มีจำนวนแพ็กเก็ตที่สูญหาย จำนวน 2 แพ็กเก็ตขึ้นไป แต่ในกรณีนี้กระบวนการ Fast Retransmission ของ Reno TCP จะไม่รอคอยค่าเวลา retransmission timeout แต่จะทำการกระบวนการ Fast Retransmission และ Fast Recovery ในการแก้ปัญหา ซึ่งจะทำให้การลดค่า cwnd ลดลงครึ่งหนึ่งเป็นจำนวน 2 ครั้ง

แพ็กเก็ตที่สูญหาย คือ แพ็กเก็ตที่ 14 และ 28 โดยเมื่อได้รับ dup ACK หมายเลข 13 ด้านส่งจะส่งแพ็กเก็ตที่ 31-35 ออกไป โดยจะมีขนาดหน้าต่างที่ใช้เท่ากับ 20 จากแพ็กเก็ตที่ 28 ที่สูญหายไปนั้น ค่า ACK ตัวแรกจากแพ็กเก็ตที่ 27 จะออกจาก Fast Recovery และ cwnd เท่ากับ 7 เมื่อได้รับ dup ACK ของแพ็กเก็ตที่ 27 ตัวที่ 3 ก็จะเข้าสู่ Fast Recovery ครั้งที่ 2 ผู้ส่งจะทำการส่งแพ็กเก็ตที่ 28 กลับไปใหม่ โดยกำหนดค่า cwnd เท่ากับ 3 และเมื่อได้รับ ACK จากแพ็กเก็ตที่ 36 ก็จะออกจาก Fast Recovery และค่า cwnd และ ssthresh เท่ากับ 3 โดยจะเข้าสู่กระบวนการ Congestion Avoidance

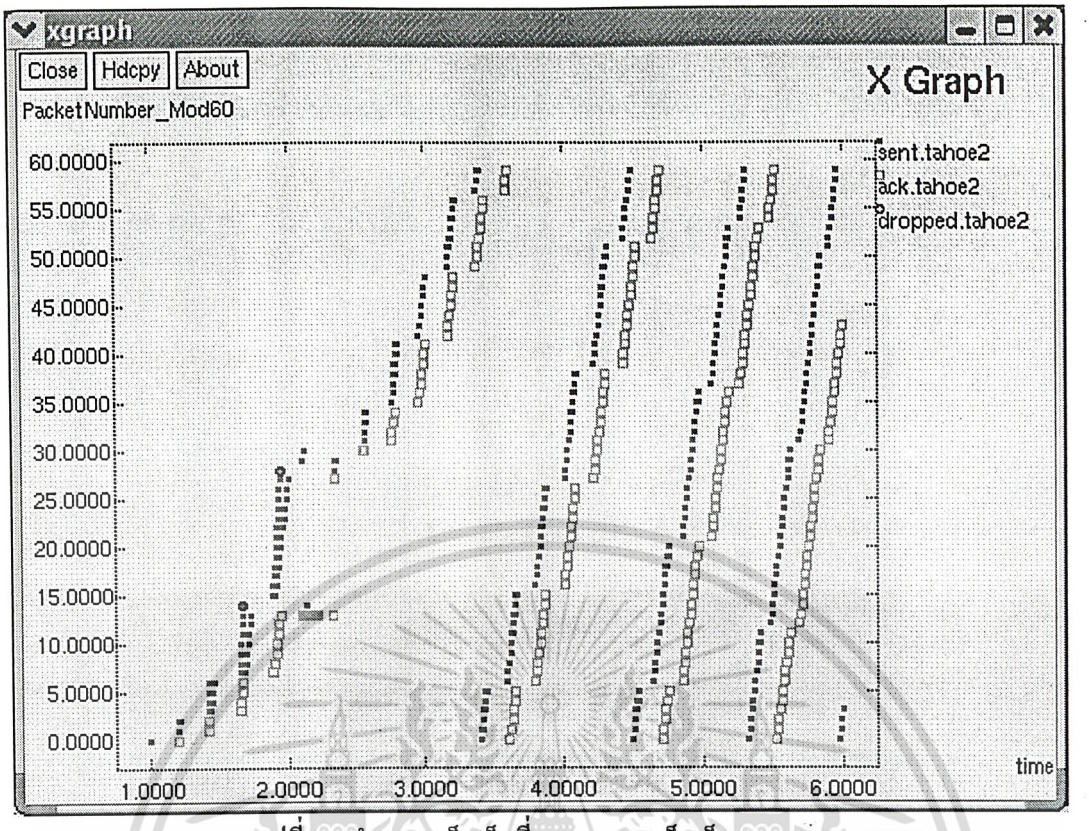
การทำงานของ TCP แบบ NewReno จะแสดงในรูปที่ 3.8 จะมีลักษณะการทำงานเช่นเดียวกับแบบ Reno TCP จนกระทั่งได้รับ ACK จากแพ็กเก็ตที่ 27 ตัวแรก โดย TCP แบบ NewReno จะทำการส่งแพ็กเก็ตที่ 28 กลับไปใหม่ โดยไม่ออกจาก Fast Recovery เมื่อได้รับ partial ACK จบครบ ก็จะออกจากกระบวนการ Fast Recovery ไปเข้าสู่กระบวนการ Congestion Avoidance

การทำงานของ TCP แบบ SACK จะแสดงในรูปที่ 3.9 จะมีลักษณะการทำงานเช่นเดียวกับ Reno TCP จนกระทั่งผู้ส่งได้รับ ACK ของแพ็กเก็ตที่ 13 ตัวที่ 3 โดย SACK TCP จะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ในการส่ง ในระหว่างอยู่ในกระบวนการ Fast Recovery โดยที่ค่า pipe จะมีค่าเท่ากับ

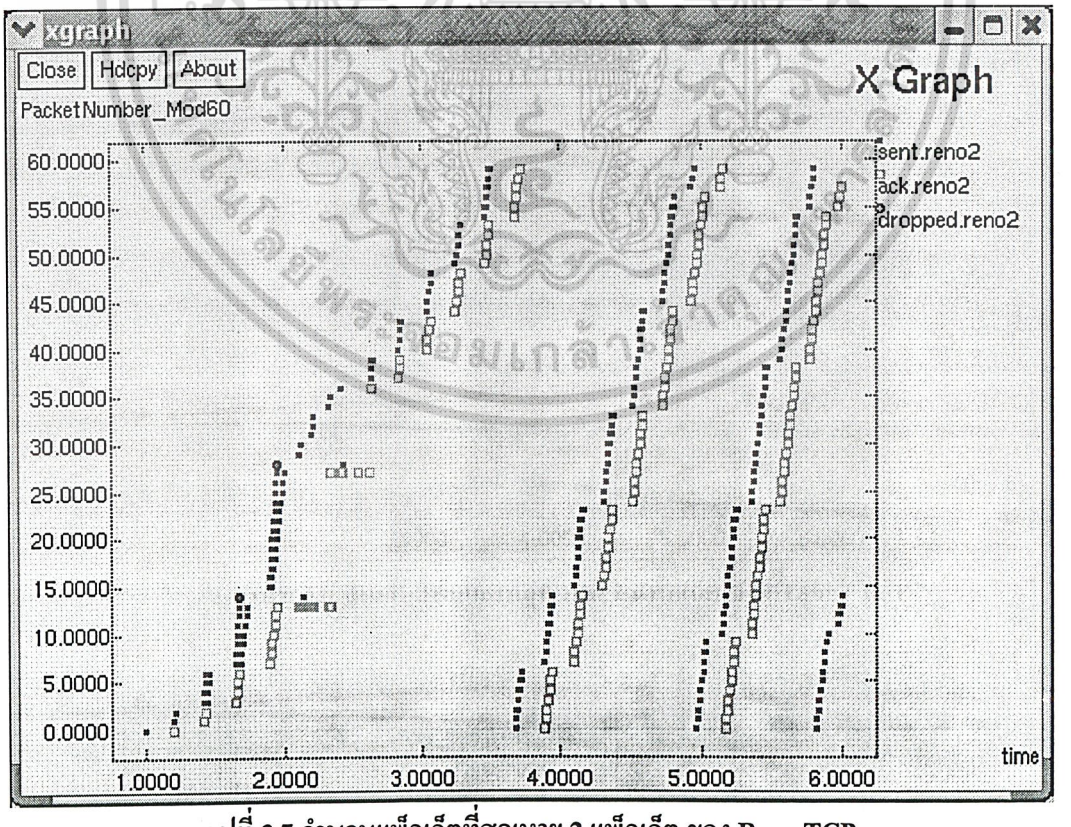
pipe = cwnd – ndup โดยที่ cwnd คือ Congestion Window , ndup คือ จำนวน dup ACK

โดยที่ค่า pipe จะถูกเพิ่มค่าขึ้น 1 เมื่อมีการส่งแพ็กเก็ตออกไป และจะถูกลดค่าลง 1 เมื่อมีการรับ dup ACK

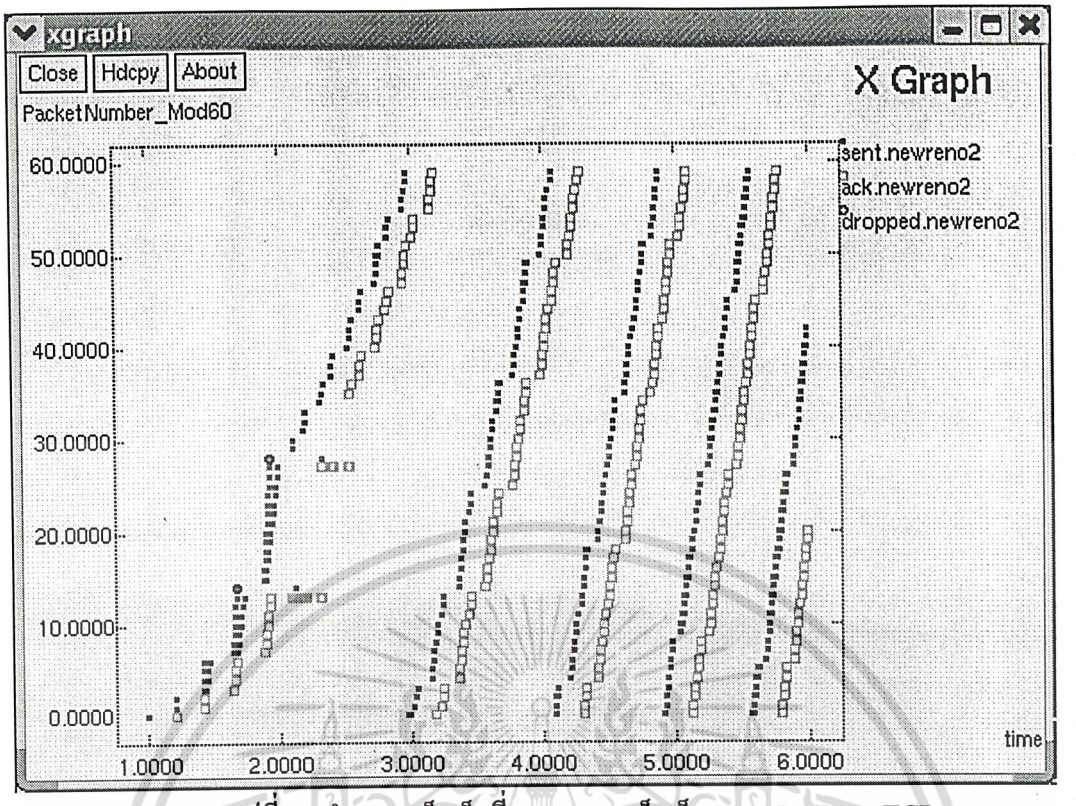
เพราะฉะนั้น ค่า pipe จะมีค่าเท่ากับ 14 และเมื่อได้รับค่า ACK จากแพ็กเก็ต 27 ตัวแรก ค่า pipe จะเท่ากับ 7 และเนื่องจากเป็น partial ACK เพราะฉะนั้นค่า pipe จะลดลงอีก 2 เท่ากับ 5 ซึ่งยังคงน้อยกว่าค่า cwnd ซึ่งมีค่าเท่ากับ 7 จึงทำการส่งแพ็กเก็ต 35 และ 36 หลังจากนั้นจำนวน dup ACK ที่เหลือ ส่งให้ส่งแพ็กเก็ตที่ 37-39 และจะออกจาก Fast Recovery เมื่อได้รับ การตอบกลับจากแพ็กเก็ตที่ 33 ไปเข้าสู่กระบวนการ Congestion Avoidance



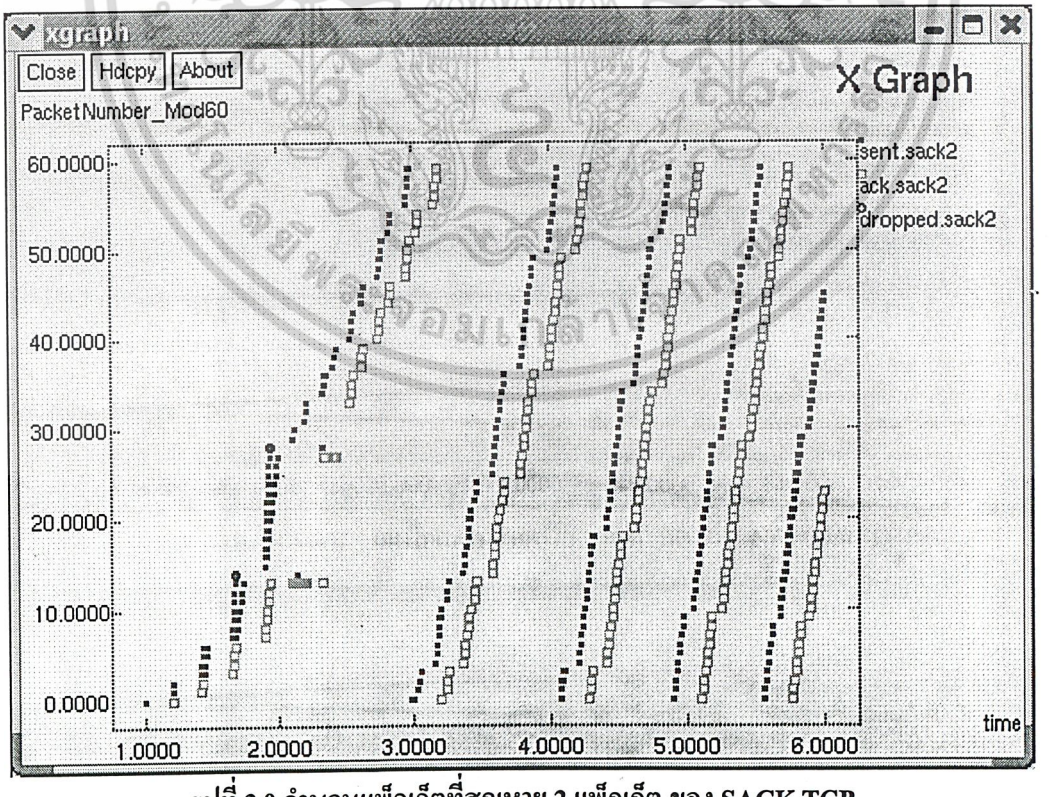
รูปที่ 3.6 จำนวนแพ็กเก็ตที่สูญหาย 2 แพ็กเก็ต ของ Tahoe TCP



รูปที่ 3.7 จำนวนแพ็กเก็ตที่สูญหาย 2 แพ็กเก็ต ของ Reno TCP



รูปที่ 3.8 จำนวนแพ็กเก็ตที่สูญหาย 2 แพ็กเก็ต ของ NewReno TCP



รูปที่ 3.9 จำนวนแพ็กเก็ตที่สูญหาย 2 แพ็กเก็ต ของ SACK TCP

3.1.3 จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 3 แพ็กเก็ต (Three Packet losses)

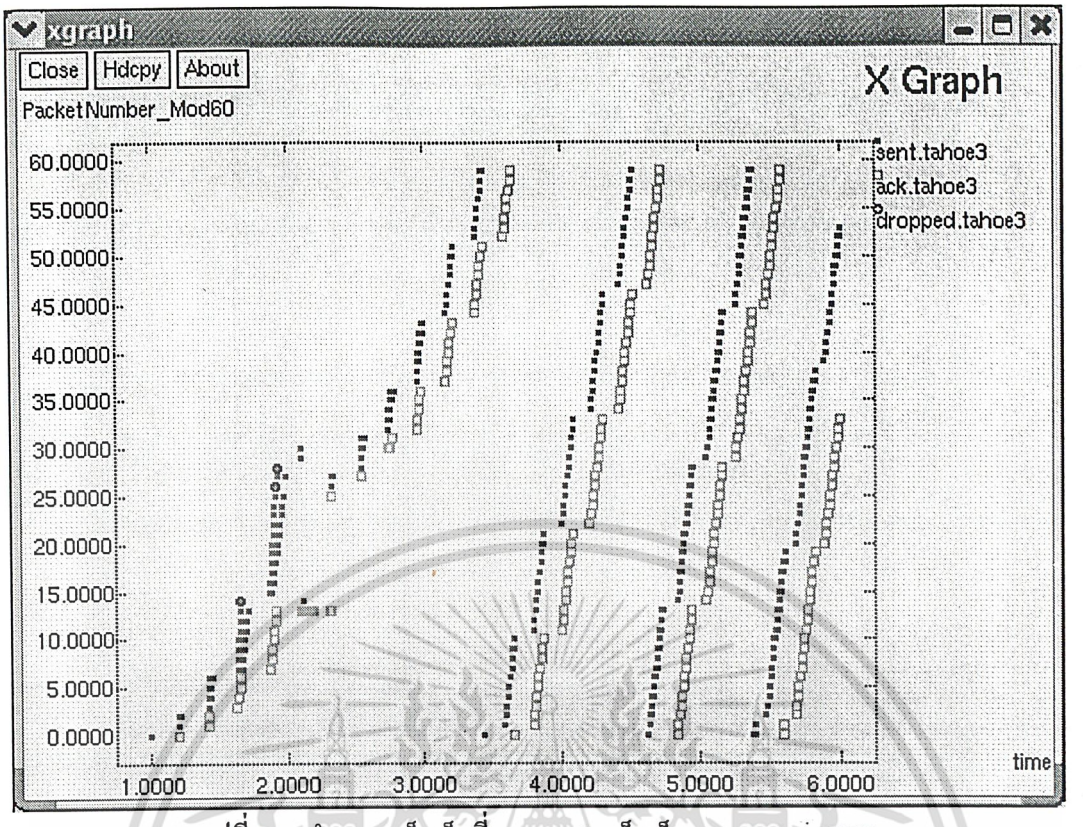
การจำลองในส่วนนี้จะพิจารณาเมื่อจำนวนแพ็กเก็ตที่สูญหายมีจำนวน 3 แพ็กเก็ตต่อหน้าต่างข้อมูล (window) จะมีลักษณะดังรูปที่ 3.10 ถึง 3.13 ซึ่งจะแสดงการทำงานในแต่ละรูปแบบของ TCP

การทำงานของ TCP แบบ Tahoe จะแสดงในรูปที่ 3.10 การทำงานจะเหมือนกับการแบบแรก จนกระทั่งทำการส่งแพ็กเก็ตที่ 14 ไปใหม่ และได้รับ dup ACK จำนวน 14 แพ็กเก็ต และเมื่อได้รับ ACK จากแพ็กเก็ตที่ 25 ก็จะเข้าสู่ Slow-Start จนกระทั่งถึงแพ็กเก็ตที่ 43 ก็จะเข้าสู่ Congestion Avoidance

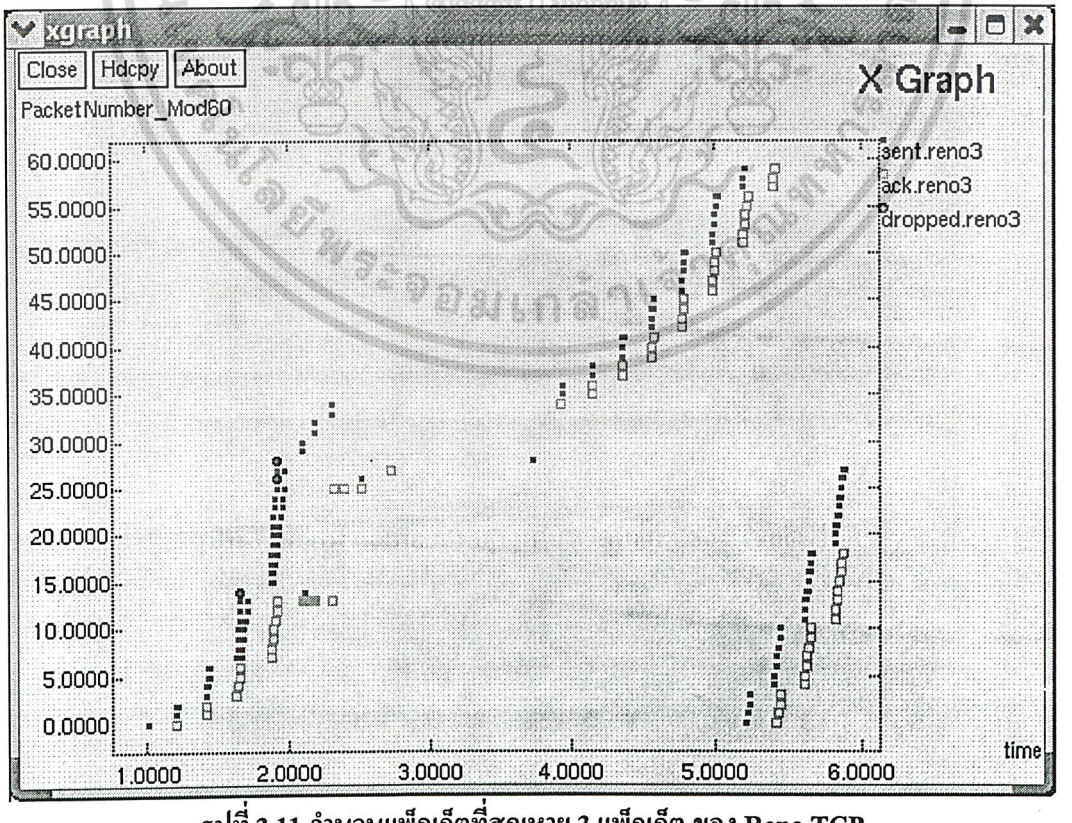
การทำงานของ TCP แบบ Reno จะแสดงในรูปที่ 3.11 การทำงานจะเป็นลักษณะเดียวกับแบบจำนวน 2 แพ็กเก็ตที่สูญหาย โดยที่เมื่อได้รับ ACK จากแพ็กเก็ต 25 ตัวแรก ตัว Reno จะออกจาก Fast Recovery แต่เมื่อได้รับค่า ACK เพิ่มเติมเข้ามาอีก 3 ค่า ก็จะกลับไปใน Fast Recovery เหมือนเดิม โดยที่ cwnd เท่ากับ 3 และขนาดหน้าต่างเท่ากับ 6 เมื่อได้รับ ACK จากแพ็กเก็ต 27 จะออกจาก Fast Recovery อีกครั้ง ด้วยค่า cwnd เท่ากับ 3 โดยตอนนี้ด้านส่งจะถูกหยุด จะต้องรอค่า retransmission timeout เมื่อค่า timeout หมด ก็จะถูกเข้าไปทำในส่วน Slow-Start ทำการส่งแพ็กเก็ตที่ 28 ออกไป เมื่อได้รับ ACK จากแพ็กเก็ต 34 จึงไปทำในส่วน Congestion Avoidance

การทำงานของ TCP แบบ NewReno จะแสดงในรูปที่ 3.12 การทำงานจะเหมือนกับแบบ Reno จนกระทั่งได้รับ ACK ค่าแรกจากแพ็กเก็ตที่ 25 โดยที่ NewReno เมื่อได้รับ ACK นี้แล้ว จะทำการส่งแพ็กเก็ตที่ 26 ซ้ำกลับไป และกำหนดขนาดหน้าต่างเท่ากับ 7 เมื่อได้รับ dup ACK จากแพ็กเก็ต 25 จำนวน 4 ครั้ง ขนาดหน้าต่างจะเพิ่มเป็น 11 ทำให้สามารถส่งแพ็กเก็ตที่ 36-39 จนถึงเมื่อได้รับ ACK จากแพ็กเก็ตที่ 39 ด้านส่งจะออกจาก Fast Recovery มาเป็น Congestion Avoidance และค่า cwnd เท่ากับ 7

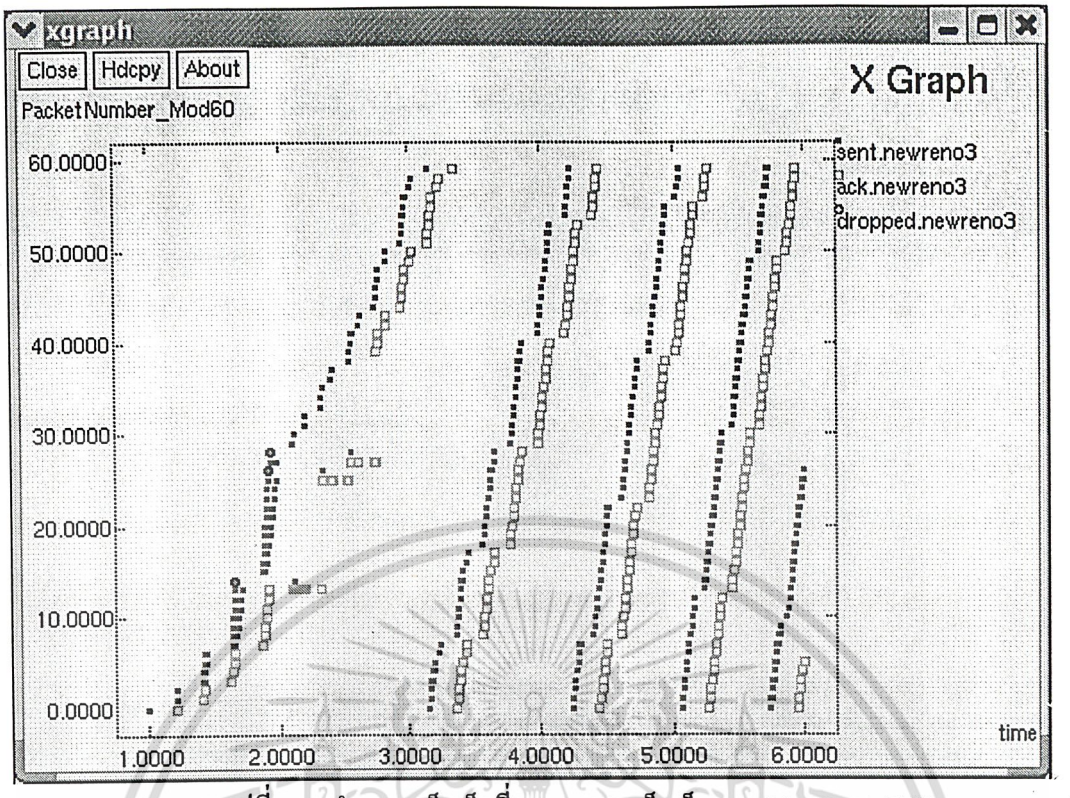
การทำงานของ TCP แบบ SACK จะแสดงในรูปที่ 3.13 การทำงานจะเหมือนกับแบบ Reno จนกระทั่งได้รับ dup ACK จากแพ็กเก็ต 13 ทั้งหมด 14 ค่า สำหรับในช่วง Fast Recovery นั้น จะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ส่ง โดยเริ่มต้นค่า pipe จะเท่ากับ 14 เมื่อได้รับค่า ACK จากแพ็กเก็ต 25 ตัวแรก ค่า pipe ลดลงเหลือเท่ากับ 7 และเนื่องจากเป็น partial ACK เพราะฉะนั้นค่า pipe จะลดลงอีก 2 จะได้ค่า pipe เท่ากับ 5 ซึ่งยังคงน้อยกว่าค่า cwnd ซึ่งมีค่าเท่ากับ 7 จึงทำการส่งแพ็กเก็ต 33 และ 34 หลังจากนั้นจำนวน dup ACK ที่เหลือก็จะส่งให้แพ็กเก็ตที่ 35-37 และออกจาก Fast Recovery เมื่อได้รับ การตอบกลับจากแพ็กเก็ตที่ 31 ไปเข้าสู่ Congestion Avoidance โดยที่มีค่า cwnd เท่ากับ 7



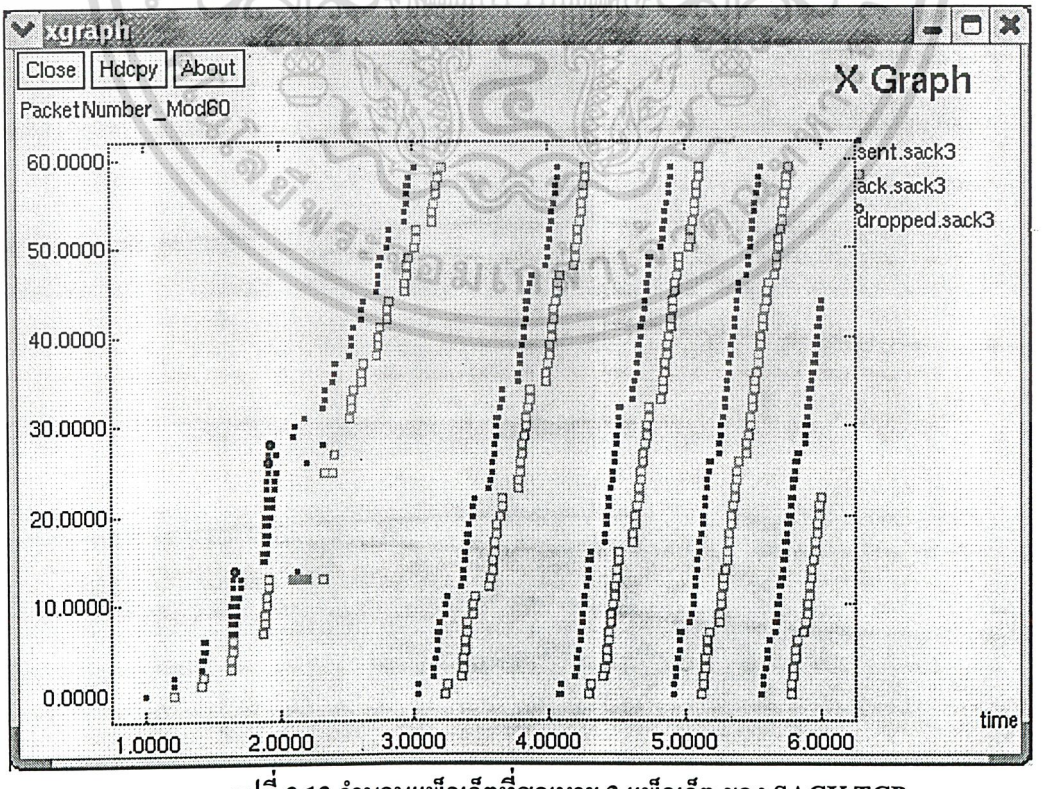
รูปที่ 3.10 จำนวนแพ็กเก็ตที่สูญหาย 3 แพ็กเก็ต ของ Tahoe TCP



รูปที่ 3.11 จำนวนแพ็กเก็ตที่สูญหาย 3 แพ็กเก็ต ของ Reno TCP



รูปที่ 3.12 จำนวนแพ็กเก็ตที่สูญหาย 3 แพ็กเก็ต ของ NewReno TCP



รูปที่ 3.13 จำนวนแพ็กเก็ตที่สูญหาย 3 แพ็กเก็ต ของ SACK TCP

3.1.4 จำนวนแพ็กเก็ตข้อมูลที่สูญหายจำนวน 4 แพ็กเก็ต (Four Packet losses)

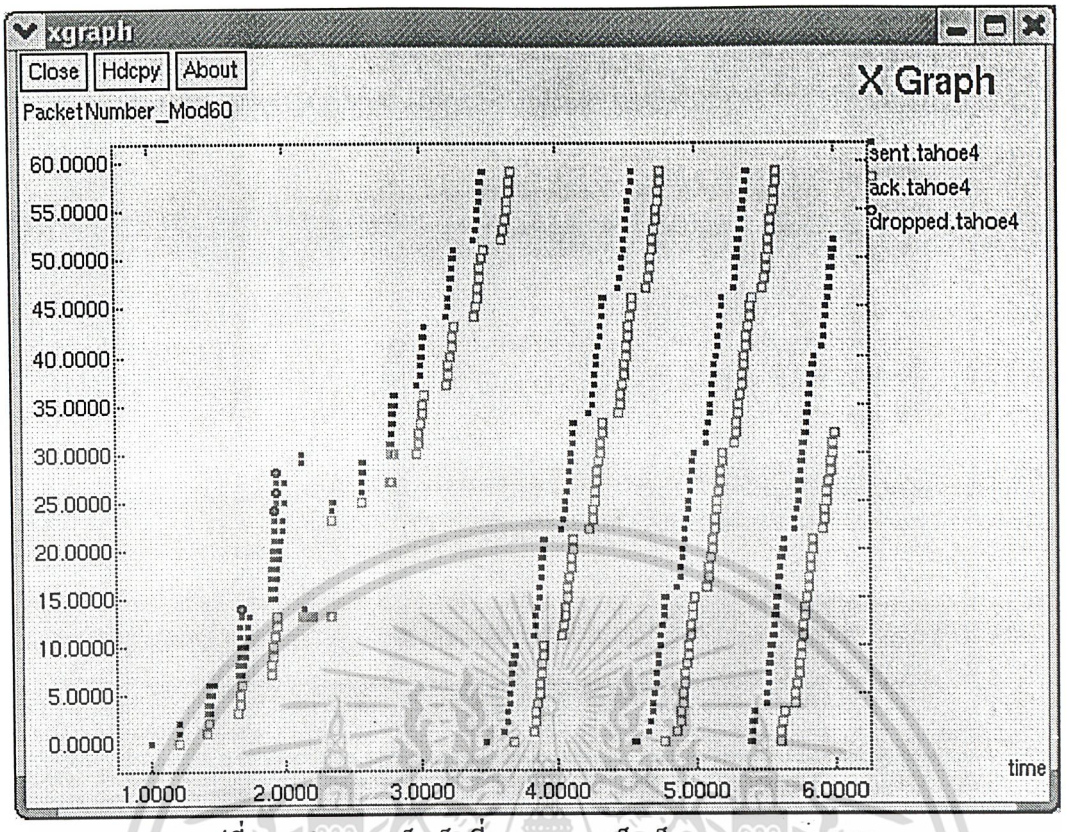
การจำลองในส่วนนี้จะพิจารณาเมื่อจำนวนแพ็กเก็ตที่สูญหายมีจำนวน 4 แพ็กเก็ตต่อหน้าต่างข้อมูล (window) จะมีลักษณะดังรูปที่ 3.14 ถึง 3.17 ซึ่งจะแสดงการทำงานในแต่ละรูปแบบของ TCP โดยจะแสดงให้เห็นความแตกต่างระหว่าง NewReno TCP กับ SACK TCP เมื่อจำนวนแพ็กเก็ตที่สูญหายมีเป็นจำนวนมาก

การทำงานของ TCP แบบ Tahoe จะแสดงในรูปที่ 3.14 การทำงานจะเหมือนกับแบบแรกจนกระทั่งการส่งแพ็กเก็ตที่ 14 ซ้ำกลับไปใหม่ เมื่อได้รับ dup ACK ทั้ง 13 แพ็กเก็ต และได้รับ ACK จากแพ็กเก็ต 23 ก็จะเข้าสู่ Slow-Start โดยมีค่า cwnd เท่ากับ 2 และเพิ่มขึ้นเมื่อได้รับ ACK ตอบกลับมาจนกระทั่งแพ็กเก็ตที่ 42 ก็จะเข้าสู่กระบวนการ Congestion Avoidance

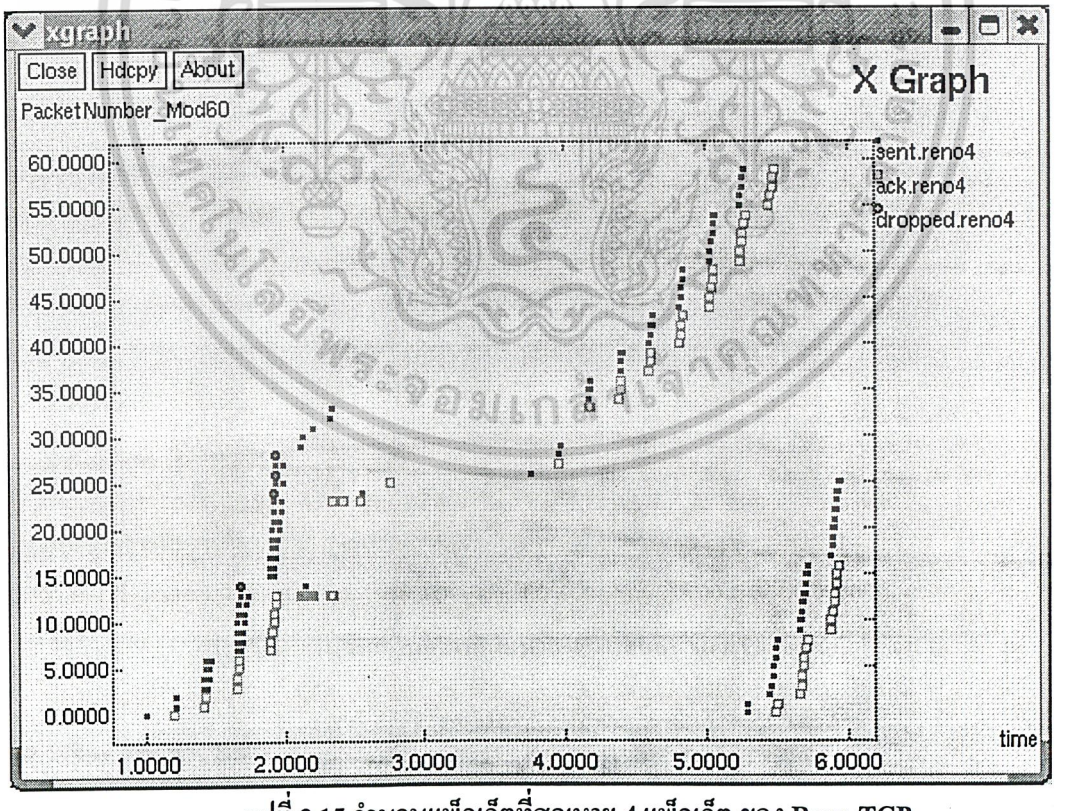
การทำงานของ TCP แบบ Reno จะแสดงในรูปที่ 3.15 การทำงานจะเป็นลักษณะเดียวกับแบบจำนวน 3 แพ็กเก็ตที่สูญหาย โดยที่เมื่อได้รับ ACK จากแพ็กเก็ต 23 ตัวแรก ด้านส่งก็จะออกจาก Fast Recovery แต่เมื่อได้รับค่า ACK เข้ามาอีก 3 ตัว ก็จะกลับไปใน Fast Recovery อีกครั้ง เมื่อได้รับ ACK จากแพ็กเก็ต 25 ก็จะออกจาก Fast Recovery และจะต้องรอค่า retransmission timeout ถึงจะทำกระบวนการ Slow-Start ต่อไป โดยทำการส่งแพ็กเก็ตที่ 26 ออกไป และเมื่อได้รับ ACK จากแพ็กเก็ต 27 ก็จะเข้าสู่กระบวนการ Congestion Avoidance

การทำงานของ TCP แบบ NewReno จะแสดงในรูปที่ 3.16 การทำงานจะเหมือนกับแบบ Reno จนกระทั่งได้รับ ACK ค่าแรกจากแพ็กเก็ตที่ 23 ก็จะทำการส่งแพ็กเก็ตที่ 24 ซ้ำกลับไปใหม่ และกำหนดขนาดหน้าต่างเท่ากับ 7 เมื่อด้านส่งได้รับ dup ACK จากแพ็กเก็ตที่ 23 จำนวน 3 ครั้ง ขนาดหน้าต่างก็จะเพิ่มเป็น 10 ทำให้สามารถส่งแพ็กเก็ตที่ 35-37 ได้ จนกระทั่งเมื่อได้รับ ACK จากแพ็กเก็ตที่ 42 ก็ออกจาก Fast Recovery ไปเข้ากระบวนการ Congestion Avoidance และค่า cwnd เท่ากับ 7

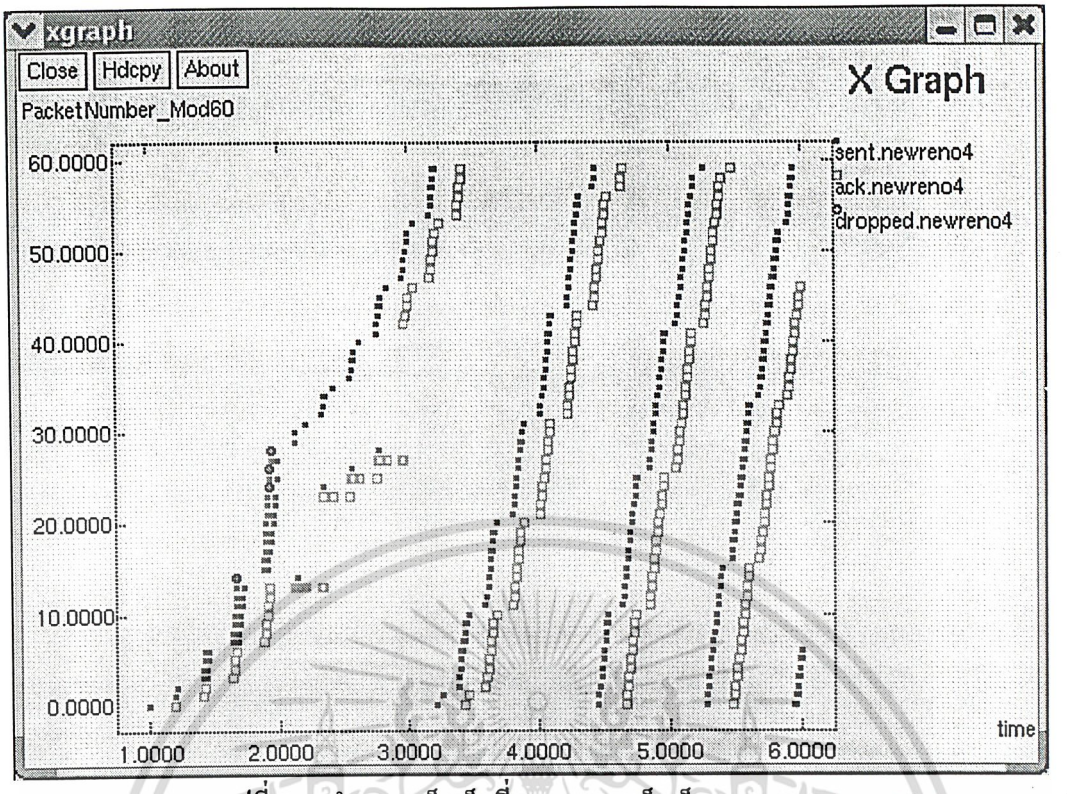
การทำงานของ TCP แบบ SACK จะแสดงในรูปที่ 3.17 การทำงานจะเหมือนกับแบบ Reno จนกระทั่งได้รับ dup ACK จากแพ็กเก็ต 13 ทั้งหมด 13 ค่า สำหรับในช่วง Fast Recovery นั้น จะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ส่ง โดยเริ่มต้นค่า pipe จะเท่ากับ 14 เมื่อได้รับค่า ACK จากแพ็กเก็ต 23 ตัวแรก ค่า pipe ลดลงเหลือเท่ากับ 6 และเนื่องจากเป็น partial ACK เพราะฉะนั้นค่า pipe จะลดลงอีก 2 จะได้ค่า pipe เท่ากับ 4 ซึ่งยังคงน้อยกว่าค่า cwnd ซึ่งมีค่าเท่ากับ 7 จึงทำการส่งแพ็กเก็ต 31 และ 32 หลังจากนั้นจำนวน dup ACK ที่เหลือก็จะส่งให้แพ็กเก็ตที่ 33-36 และออกจาก Fast Recovery เมื่อได้รับการตอบกลับจากแพ็กเก็ตที่ 30 ไปเข้าสู่ Congestion Avoidance โดยที่มีค่า cwnd เท่ากับ 7



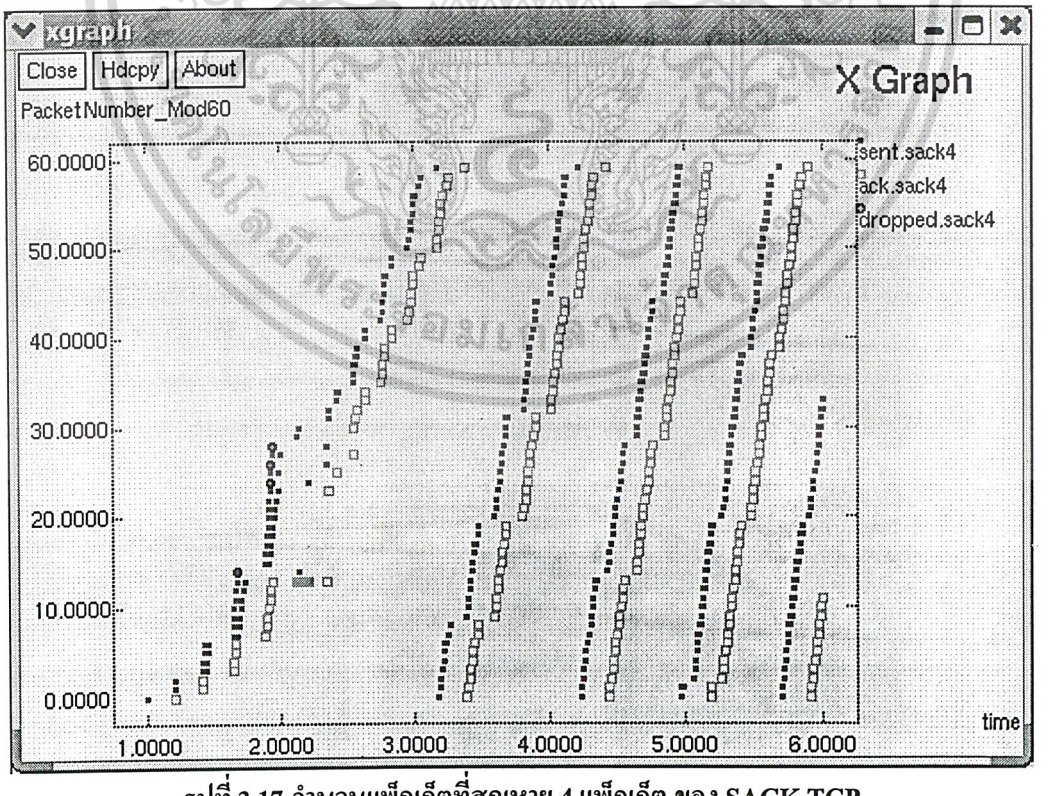
รูปที่ 3.14 จำนวนแพ็กเก็ตที่สูญหาย 4 แพ็กเก็ต ของ Tahoe TCP



รูปที่ 3.15 จำนวนแพ็กเก็ตที่สูญหาย 4 แพ็กเก็ต ของ Reno TCP



รูปที่ 3.16 จำนวนแพ็กเก็ตที่สูญหาย 4 แพ็กเก็ต ของ NewReno TCP



รูปที่ 3.17 จำนวนแพ็กเก็ตที่สูญหาย 4 แพ็กเก็ต ของ SACK TCP

สรุปการทำงานของแต่ละรูปแบบของ TCP

1. Tahoe TCP

Tahoe TCP เป็นรูปแบบพื้นฐานที่ใช้ในแต่ละแบบของ TCP โดยใช้แนวคิด Slow-start , Fast Retransmit ,Congestion Avoidance ร่วมกัน สำหรับกระบวนการFast Retransmit นั้น เมื่อได้รับ dup ACK จากหมายเลขแพ็กเก็ตเดิม ผู้ส่งก็จะตีความหมายว่า แพ็กเก็ตเกิดการสูญหาย และทำการส่งแพ็กเก็ตซ้ำกลับไปใหม่ โดยที่ไม่รอค่าเวลา retransmission timeout และจะไปเริ่มที่กระบวนการ Slow-start เสมอ

ผลจากการทดสอบทั้ง 4 จะพบว่า จะให้ผลลัพธ์ออกมาใกล้เคียงกัน โดยที่ Tahoe TCP จะทำการส่งแพ็กเก็ตซ้ำโดยใช้กระบวนการ Fast Retransmit และตามด้วยกระบนการ Slow-start

2. Reno TCP

Reno TCP เป็นรูปแบบที่ดัดแปลงจากแบบ Tahoe TCP โดยทำการปรับปรุงกระบวนการ Fast Retransmit โดยรวมเอากระบวนการ Fast Recovery เข้าไปด้วย เพื่อป้องกันไม่ให้เกิดช่วงเวลาว่างในบัฟเฟอร์หลังจากผ่านกระบวนการ Fast Retransmit โดยที่จะไม่ต้องกลับไปเริ่มที่กระบวนการ Slow-start และใช้ตัว dup ACK ที่รับเข้ามาเป็นตัวกำหนดเวลาในการส่งแพ็กเก็ตถัดไป

ทำให้ประสิทธิภาพดีขึ้นเมื่อเกิดการสูญหายของแพ็กเก็ตเพียงตัวเดียวต่อหน้าต่างข้อมูล แต่จะให้ประสิทธิภาพที่ไม่ดี ถ้าหากมีเกิดการสูญหายของแพ็กเก็ตเป็นจำนวนมาก เพราะอาจจะเกิดปัญหาที่จะต้องรอค่าเวลา retransmission timeout

3. NewReno TCP

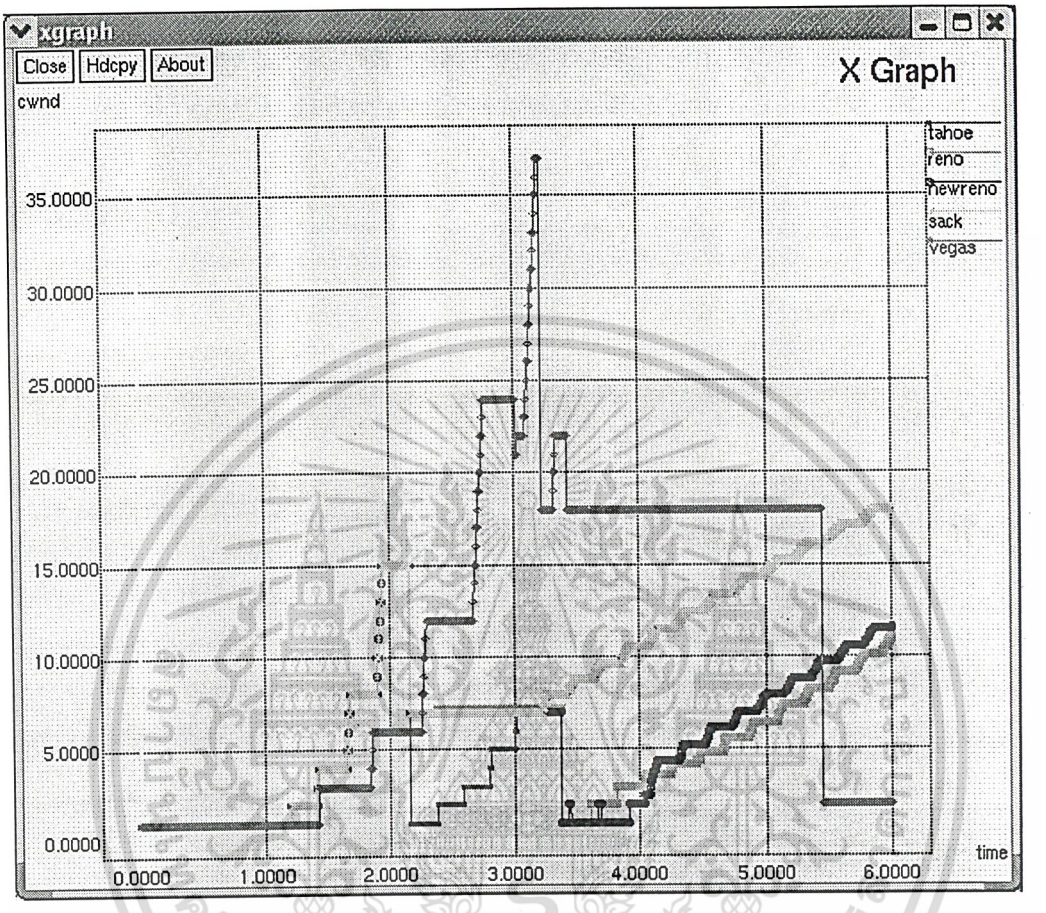
NewReno TCP เป็นรูปแบบของ Reno TCP ที่ทำการปรับปรุงมีประสิทธิภาพดีขึ้น โดยปรับเปลี่ยนเมื่อได้รับค่า ACK ในระหว่าง Fast Recovery รูปแบบการทำงานนี้ จะไม่ต้องออกจากกระบวนการนี้ เพราะฉะนั้นเมื่อมีจำนวนแพ็กเก็ตที่สูญหายหลายตัว ก็จะทำให้สามารถจัดการได้เลย ไม่ต้องรอเวลา retransmission timeout โดยจะสามารถทำการส่งแพ็กเก็ตได้เพียง 1 แพ็กเก็ตต่อรอบการส่ง (round-trip time) ซึ่งทำให้ประสิทธิภาพด้อยกว่าในแบบ SACK TCP เมื่อเกิดการสูญหายของจำนวนแพ็กเก็ตมาก ๆ

4. SACK TCP

SACK TCP เป็นรูปแบบที่ปรับปรุงให้มีการเก็บบันทึกค่าจาก ACK ทำให้ทราบแพ็กเก็ตที่สูญหายไป ต้องการการส่งซ้ำ โดยใช้ตัวแปร pipe ในการกำหนดการส่งแพ็กเก็ตออกไป ในระหว่างที่อยู่ใน Fast Recovery

ทำให้ SACK TCP สามารถจัดการกับการสูญหายของแพ็กเก็ตที่มีเป็นจำนวนมากได้ดีกว่าทั้งสามแบบข้างต้น

เมื่อทำการพิจารณาค่าขนาดหน้าต่าง (Congestion Window:cwnd) ระหว่างรูปแบบ TCP แต่ละชนิด จะได้ดังรูปที่ 3.18

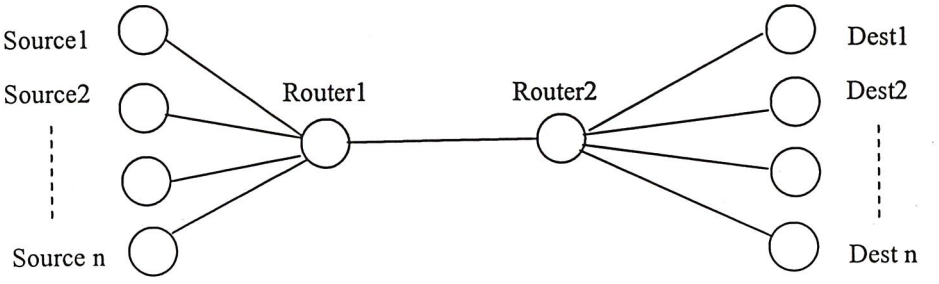


รูปที่ 3.18 การเปรียบเทียบค่า cwnd ในแต่ละชนิดของ TCP

3.2 การทดสอบประสิทธิภาพระหว่าง Reno กับ Vegas TCP ในการจำลองเครือข่ายที่ใช้งาน

การทดลองส่วนนี้เป็นการทดสอบที่ใกล้เคียงกับการใช้งานจริงเพื่อเปรียบเทียบประสิทธิภาพระหว่าง TCP แบบ Reno กับ TCP แบบ Vegas ภายใต้การจัดการคิวแบบ RED (Random Early Detection) ซึ่งการจัดการคิวแบบ RED นี้ เป็นวิธีการที่นิยมใช้งานจริงในปัจจุบัน และค่อนข้างมีประสิทธิภาพ

การทดสอบประสิทธิภาพของ TCP แบบ Reno และ TCP แบบ Vegas นี้ จะวัดค่าตัวแปรเพื่อเปรียบเทียบกัน 2 ตัว คือ ค่า Throughput และจำนวน Packets ที่ถูก drop ภายใต้การเชื่อมต่อที่แตกต่างกัน 3 โครงข่าย คือ โครงข่าย ISDN, โครงข่าย E-1 และ โครงข่าย E-3 โดยมีรูปแบบการ Simulation ดังรูปที่ 3.19



รูปที่ 3.19 แบบจำลองการทดสอบระหว่าง Reno กับ Vegas TCP

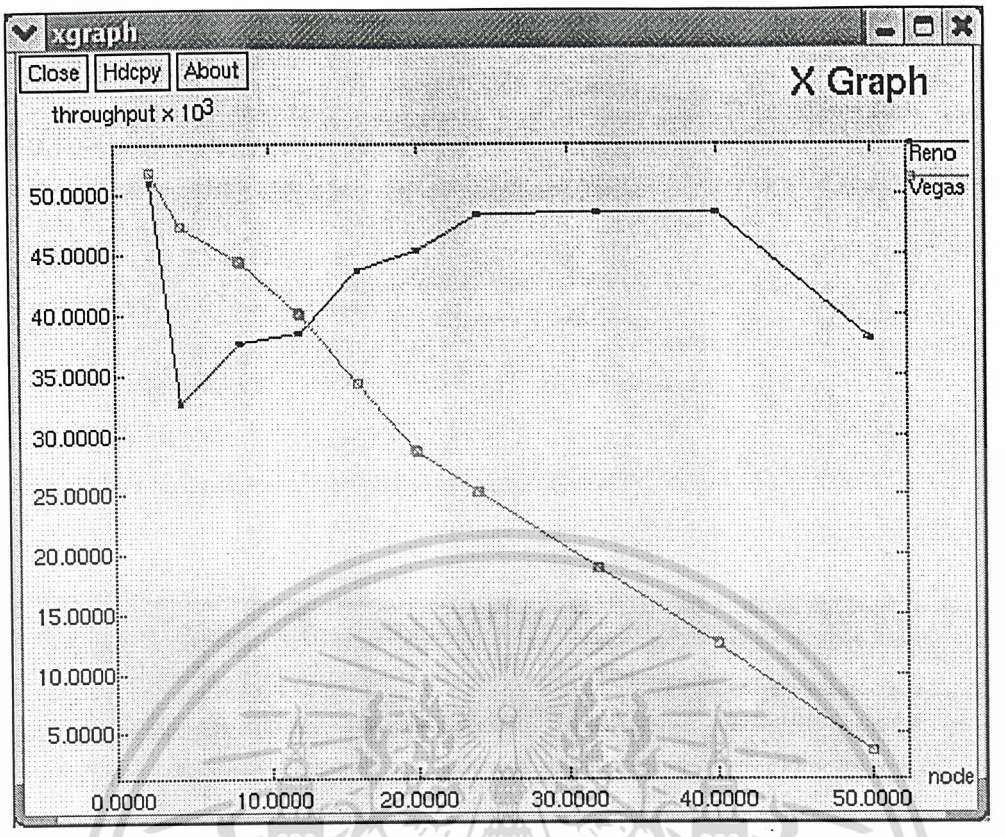
จากรูปที่ 3.19 เราจะใช้การเปลี่ยนแปลงจำนวนโหนด (เครื่องคอมพิวเตอร์) ที่เชื่อมต่อกันผ่านโครงข่าย แต่ละโครงข่ายโดยมีค่าเริ่มต้นที่ 2 โหนด และปรับค่าไปเรื่อยๆ จนถึง 50 โหนด โดยในแต่ละโหนดนั้นจะกำหนดให้มีการติดต่อสื่อสารแบบ 1 ต่อ 1 ซึ่งค่าพารามิเตอร์ที่สำคัญที่ใช้ในการจำลองการทำงาน มีดังนี้

1. LAN ทั้ง 2 ฝั่งใช้ Fast Ethernet ความเร็ว 100 Mbps
2. ค่า Delay ระหว่าง Node ถึง Router เท่ากับ 0.1 ms
3. ค่า Propagation delay ระหว่าง Router1 ถึง Router2 เท่ากับ 80 ms
4. Router ใช้การจัดการคิวแบบ RED โดยมีขนาดบัฟเฟอร์เท่ากับ 50 packets
5. สายสัญญาณที่เชื่อมต่อระหว่าง Router 2 ตัว มีมาตรฐาน 3 แบบ คือ ISDN (56 kbps), E-1 (2.048 Mbps) และ E-3 (34.368 Mbps)

<u>ความหมายและวิธีการวัดค่า Throughput</u>

Throughput คืออัตราส่วนจำนวน bits ข้อมูลทั้งหมดที่สามารถรับส่งได้สำเร็จแล้วระหว่างสถานีส่งกับสถานีรับต่อเวลาทั้งหมดที่ใช้ในการรับส่ง ซึ่งสามารถเขียนได้ดังสมการที่ (3.1) ดังต่อไปนี้

$$\text{Throughput} = \frac{\text{Number of bits received}}{\text{เวลาที่ส่ง TCP Packet แรก - เวลาที่รับ ACK Packet สุดท้าย}} \quad (3.1)$$

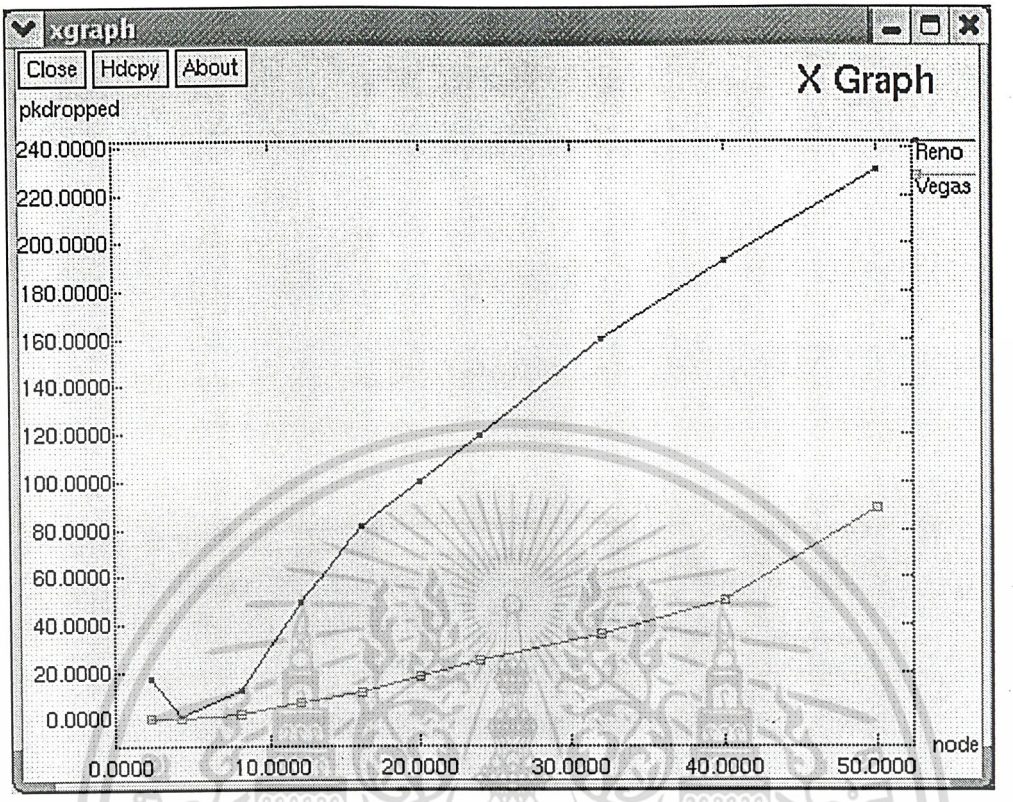


รูปที่ 3.20 กราฟแสดงค่า Throughput ต่อจำนวน Node โดยเชื่อมต่อผ่าน Bandwidth เท่ากับ 56 kbps

จำนวน Nodes	2	4	8	12	16	20	24	32	40	50
Reno throughput(kbps)	51	32.6	37.6	38.4	43.6	45.4	48.3	48.5	48.6	38
Vegas throughput(kbps)	52	47.3	44.4	40	34	28.6	25	18.8	12.5	3.6

ตารางที่ 3.1 ค่า Throughput ต่อจำนวน Node โดยเชื่อมต่อผ่าน Bandwidth เท่ากับ 56 kbps

จากกราฟรูปที่ 3.20 และตารางที่ 3.1 จะเห็นว่าเมื่อขนาด Bandwidth link มีค่าน้อยๆ จำนวนแพ็กเก็ตก็จะถูกส่งออกไปอย่างช้าๆ ทำให้เกิดความคับคั่งของแพ็กเก็ตที่ Router ตัวที่ 1 และจากการที่ TCP แบบ Reno จะส่งแพ็กเก็ตไปเรื่อยๆ จนกว่าแพ็กเก็ตจะ drop จึงจะลดปริมาณการส่งและเพิ่มขึ้นอย่างรวดเร็วเป็นเช่นนี้สลับกันไป ดังนั้นค่า Throughput จึงไม่ขึ้นกับปริมาณแพ็กเก็ตที่อยู่ใน Router ซึ่งตรงข้ามกับ TCP แบบ Vegas ที่พยายามควบคุมปริมาณแพ็กเก็ตใน Router ตลอดเวลา ดังนั้นเมื่อมีปริมาณแพ็กเก็ตใน Router มาก TCP แบบ Vegas ก็จะลดปริมาณการส่งแพ็กเก็ตจนถึงขั้นหยุดส่ง ดังนั้นค่า Throughput จึงขึ้นอยู่กับปริมาณแพ็กเก็ตที่อยู่ใน Router โดยตรง หรือกล่าวได้ว่า ถ้าปริมาณการเชื่อมต่อเพิ่มขึ้นค่า Throughput ของ TCP แบบ Vegas จะลดลง

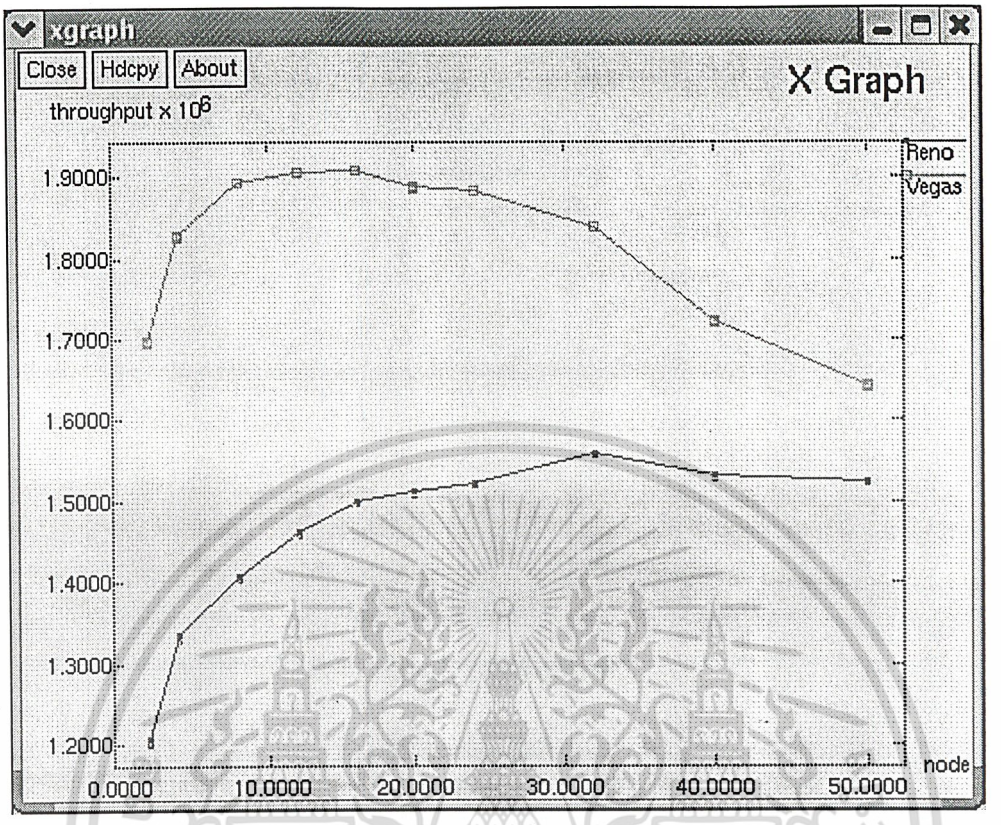


รูปที่ 3.21 กราฟแสดงจำนวนแพ็กเก็ตที่ Drop ต่อจำนวน Node โดยที่ Bandwidth เท่ากับ 56 kbps

จำนวน Nodes	2	4	8	12	16	20	24	32	40	50
Reno Packets dropped	17	1	12	49	81	100	119	160	193	231
Vegas Packets dropped	0	0	2	7	11	18	25	36	50	89

ตารางที่ 3.2 จำนวนแพ็กเก็ตที่ Drop ต่อจำนวน Node โดยที่ Bandwidth เท่ากับ 56 kbps

จากกราฟรูปที่ 3.21 และ ตารางที่ 3.2 ดังที่กล่าวไว้จากหน้าก่อนนี้ คือ TCP แบบ Reno จะส่งข้อมูลโดยไม่คำนึงถึงปริมาณแพ็กเก็ตที่อยู่ใน Router ดังนั้นเมื่อส่งเข้ามาจนบัฟเฟอร์เต็มแพ็กเก็ตก็จะถูก drop และเมื่อถูก drop แล้ว TCP แบบ Reno ก็มีกระบวนการ Fast Retransmit คือ เริ่มการส่งอย่างรวดเร็วและเพิ่มปริมาณการส่งแบบเอ็กซ์โพเนนเชียล ดังนั้นปริมาณแพ็กเก็ตที่ drop ก็จะเพิ่มขึ้นเรื่อยๆและในปริมาณที่มาก ซึ่งต่างกับ TCP แบบ Vegas ที่มีการควบคุมปริมาณแพ็กเก็ตใน Router ตลอดเวลา ดังนั้นปริมาณแพ็กเก็ตที่ drop ของ TCP แบบ Vegas จึงน้อยกว่า TCP แบบ Reno

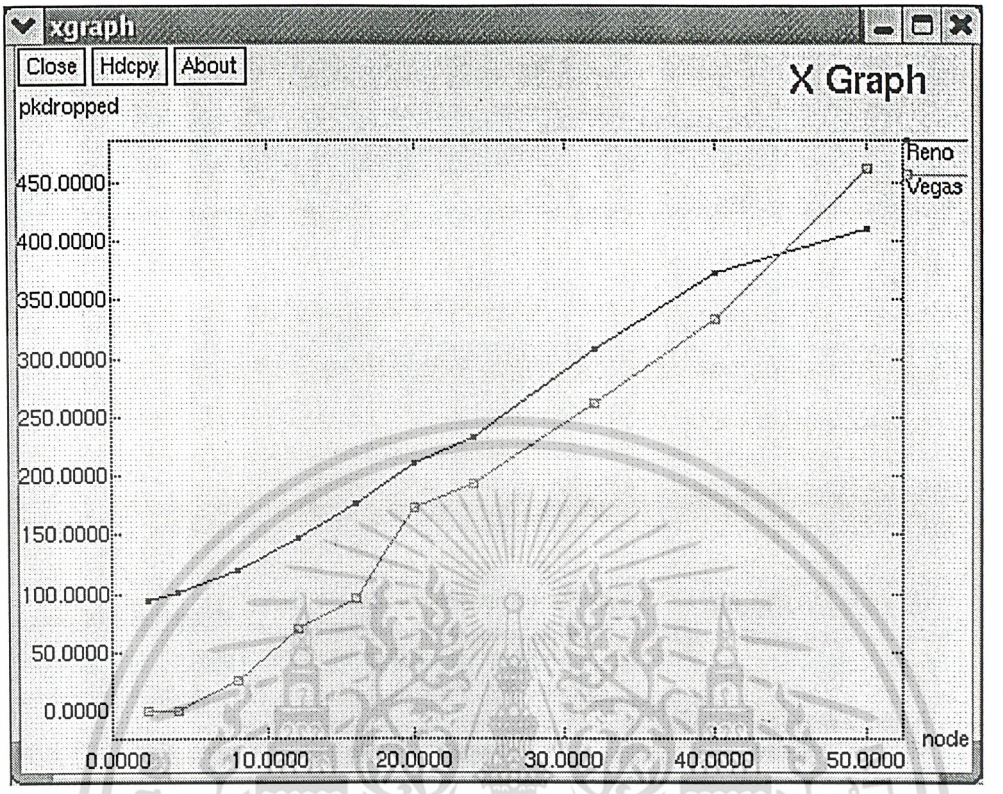


รูปที่ 3.22 กราฟแสดง Throughput ต่อจำนวน Node โดยเชื่อมต่อผ่าน Bandwidth เท่ากับ 2.048 Mbps

จำนวน Nodes	2	4	8	12	16	20	24	32	40	50
Reno throughput(kbps)	1209	1335.4	1408	1462	1501	1511	1523	1559.8	1531.4	1525.2
Vegas throughput(kbps)	1696.3	1828	1892.4	1905	1907.3	1887	1883.2	1839.5	1723	1643.5

ตารางที่ 3.3 ค่า Throughput ต่อจำนวน Node โดยเชื่อมต่อผ่าน Bandwidth เท่ากับ 2.048 Mbps

จากกราฟรูปที่ 3.22 และ ตารางที่ 3.3 จะเห็นว่าเมื่อขนาด Bandwidth link มีค่าเพิ่มขึ้นปริมาณแพ็กเก็ตก็จะถูกส่งออกจาก Router ได้เร็วขึ้น ทำให้ความคับคั่งของแพ็กเก็ตที่ Router ลดลง ดังนั้น TCP แบบ Reno และ TCP แบบ Vegas ก็จะมีการส่งปริมาณข้อมูลด้วยอัตราเร็วพอๆ กัน แต่ค่า Throughput ของ TCP แบบ Vegas จะดีกว่า เนื่องจาก TCP แบบ Vegas จะควบคุมปริมาณแพ็กเก็ตให้เกิดการ drop น้อยที่สุด ในขณะที่ TCP แบบ Reno ไม่ได้ควบคุมในเรื่องนี้

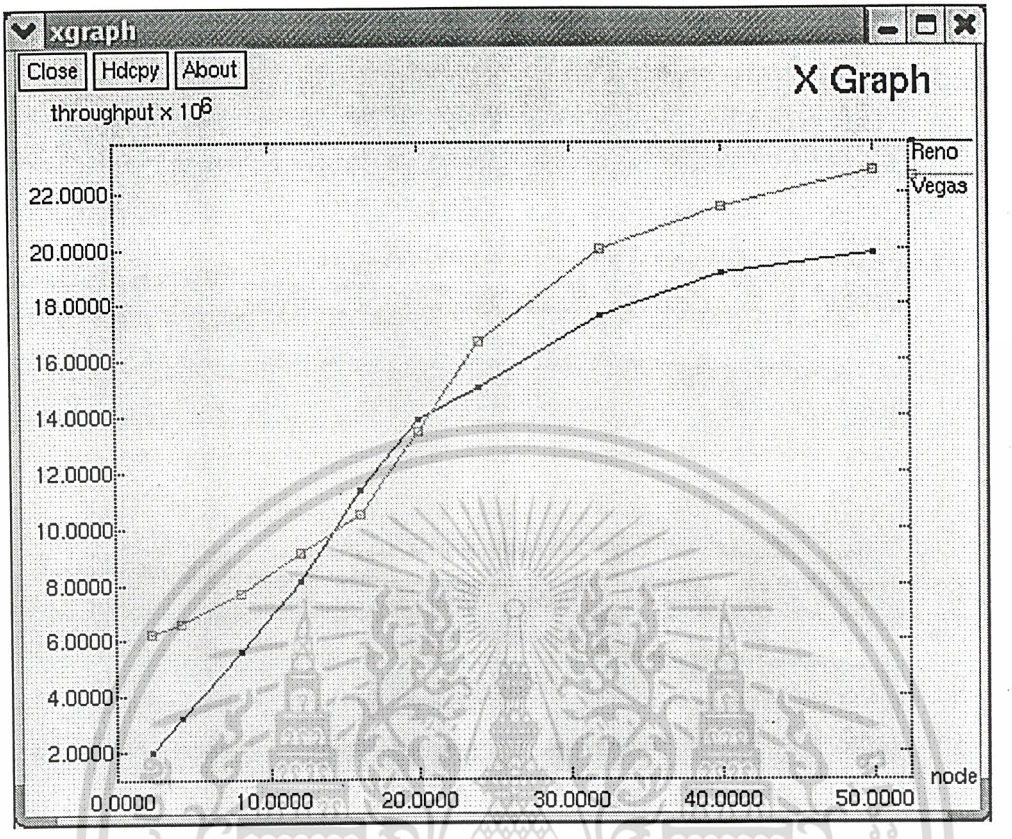


รูปที่ 3.23 กราฟแสดงจำนวนแพ็กเก็ตที่ Drop ต่อจำนวน Node โดยที่ Bandwidth เท่ากับ 2.048 Mbps

จำนวน Nodes	2	4	8	12	16	20	24	32	40	50
Reno Packets dropped	94	102	120	147	176	211	232	307	372	410
Vegas Packets dropped	0	1	26	71	96	173	194	261	334	462

ตารางที่ 3.4 จำนวนแพ็กเก็ตที่ Drop ต่อจำนวน Node โดยที่ Bandwidth เท่ากับ 2.048 Mbps

จากกราฟรูปที่ 3.23 และ ตารางที่ 3.4 ด้วยเหตุผลจากหน้าก่อนนี้ทำให้ปริมาณแพ็กเก็ตที่ drop ของ TCP แบบ Reno มากกว่า TCP แบบ Vegas แต่จะสังเกตว่าที่จำนวนโหนดเยอะขึ้น (ตั้งแต่ 50 โหนดขึ้นไป) ปริมาณแพ็กเก็ตที่ drop ของ TCP แบบ Vegas เริ่มเยอะขึ้นจนมากกว่า TCP แบบ Reno ซึ่งเหตุผลก็จะเป็นไปตามที่ได้กล่าวไว้ในส่วนของ Bandwidth เท่ากับ 56 kbps

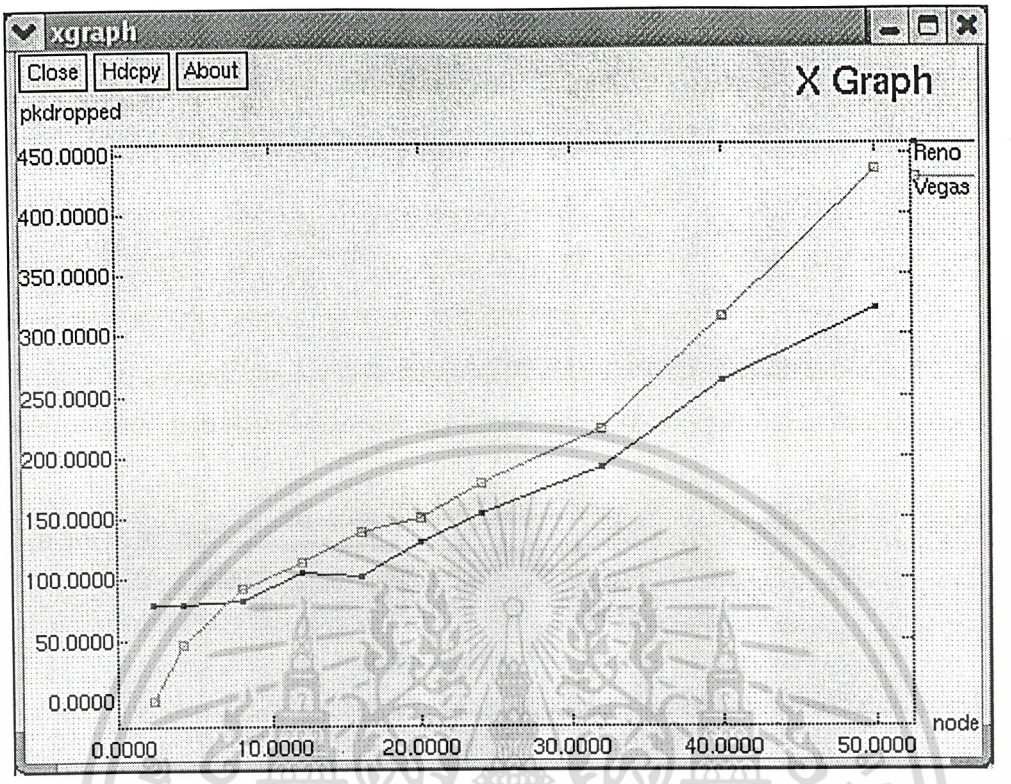


รูปที่ 3.24 กราฟแสดง Throughput ต่อจำนวน Node โดยเชื่อมต่อผ่าน Bandwidth เท่ากับ 34.368 Mbps

จำนวน Nodes	2	4	8	12	16	20	24	32	40	50
Reno throughput(Mbps)	2	3.2	5.6	8.1	11.3	13.8	15	17.6	19.1	19.8
Vegas throughput(Mbps)	6.2	6.6	7.7	9.1	10.5	13.4	16.7	20	21.4	22.7

ตารางที่ 3.5 ค่า Throughput ต่อจำนวน Node โดยเชื่อมต่อผ่าน Bandwidth เท่ากับ 34.368 Mbps

จากกราฟรูปที่ 3.24 และ ตารางที่ 3.5 จะเห็นว่าเมื่อขนาด Bandwidth link มีค่ามากๆ ปริมาณแพ็กเก็ตก็จะถูกส่งออกไปจาก Router อย่างรวดเร็วจึงไม่มีปัญหาความคับคั่งของแพ็กเก็ตที่ Router ดังนั้น TCP แบบ Reno และ TCP แบบ Vegas จึงใช้การส่งปริมาณข้อมูลที่พอๆกัน ผลที่ได้จะเห็นว่าประสิทธิภาพไม่ต่างกันมาก โดยค่า Throughput ของ TCP แบบ Vegas จะดีกว่าเล็กน้อย



รูปที่ 3.25 กราฟแสดงจำนวนแพ็กเก็ตที่ Drop ต่อจำนวน Node โดยที่ Bandwidth เท่ากับ 34.368 Mbps

จำนวน Nodes	2	4	8	12	16	20	24	32	40	50
Reno Packets dropped	79	79	82	106	103	130	154	191	263	323
Vegas Packets dropped	0	46	92	114	138	150	179	223	316	436

ตารางที่ 3.6 จำนวนแพ็กเก็ตที่ Drop ต่อจำนวน Node โดยที่ Bandwidth เท่ากับ 34.368 Mbps

จากกราฟรูปที่ 3.25 และ ตารางที่ 3.6 ดังที่กล่าวไว้จากหน้าก่อนนี้ คือ ประสิทธิภาพจะต่างกันไม่มาก โดยถ้าพิจารณาจากจำนวนแพ็กเก็ตที่ drop นั้น TCP แบบ Reno จะดีกว่าเล็กน้อย

3.3 แนวทางการพัฒนาประสิทธิภาพของโปรโตคอล TCP

เนื่องจากว่า TCP แบบ Vegas ใช้วิธีการจัดการกับจำนวนแพ็กเก็ตที่รอคิว คือ จะควบคุมค่าจำนวนแพ็กเก็ตที่รอคิว ($\bar{q}_v$) ในขณะที่ TCP แบบ Reno จะพยายามใช้งาน bandwidth ให้ได้มากที่สุด จนกว่าจะเกิดการสูญหายของแพ็กเก็ต ดังนั้นจึงมีแนวความคิดที่จะนำ Vegas มาต่อใช้งานร่วมกับ Reno เพื่อเพิ่มประสิทธิภาพให้กับโปรโตคอล TCP

แต่จะพบว่าเมื่อมีการใช้งานร่วมกันแล้ว ค่า Throughput โดยเฉลี่ยของตัว Vegas กลับต่ำกว่าค่า Throughput โดยเฉลี่ยของตัว Reno คือเกิดความไม่เท่าเทียมกันในการใช้ช่องสัญญาณ สาเหตุมาจากการตั้งค่าพารามิเตอร์ที่ใช้สำหรับ Vegas คือ α และ β ซึ่งใช้สำหรับการกำหนดค่าขนาดหน้าต่าง (congestion window : cwnd) โดยค่าปกติแล้ว $\alpha = 1$ และ $\beta = 3$ ดังนั้นจึงต้องมีการปรับปรุงค่า α และ β เพื่อให้เกิดความเท่าเทียมกันในการใช้ช่องสัญญาณ โดยคิดจากค่าพารามิเตอร์ที่ใช้สำหรับ Vegas ตามสมการดังต่อไปนี้

cwnd + 1 if Diff $< \alpha$

cwnd - 1 if Diff $> \beta$

cwnd otherwise($\alpha \leqslant$ Diff $\leqslant \beta$)

เมื่อนำ Vegas มาต่อการใช้งานร่วมกับ Reno จะได้ว่า

$$\bar{q}_v \leq \beta$$

$$q_r \varepsilon \left[0, B - \bar{q}_v \right]$$

โดยที่ B คือ ขนาดบัฟเฟอร์ของ Router

ถ้าให้ $\bar{q}_r$ คือ ค่าเฉลี่ยจำนวนแพ็กเก็ตที่รอคิวใน Buffer ของ Reno

จะได้ $\bar{q}_r = \dfrac{B - \bar{q}_v}{2}$

ดังนั้น อัตราส่วนของ throughput ของ Vegas ต่อ throughput ของ Reno จะเท่ากับ

$$\frac{\lambda_v}{\lambda_r} = \frac{\bar{q}_v}{\bar{q}_r} = \frac{2 \cdot \bar{q}_v}{B - \bar{q}_r} \qquad (3.2)$$

จากสมการที่ (3.2) ถ้าต้องการให้ throughput ของ Reno ใกล้เคียงกับของ Vegas ก็คือต้องทำให้สมการ (3.2) มีค่าเท่ากับ 1 โดยการเปลี่ยนแปลงที่ค่า α และ β

ให้ W = window size

d = RTT min

D = RTT

เพราะฉะนั้น ค่า Diff จะมีค่าเท่ากับ

$$Diff = (\frac{W}{d} - \frac{W}{D}) \cdot d = W \frac{(D-d)}{D} \quad (3.3)$$

หรือกล่าวได้ว่า ค่า Diff ก็คือ จำนวนแพ็กเก็ตที่อยู่ในคิว เพราะฉะนั้น $Diff \approx \bar{q_v}$

กำหนดให้

Wvmax คือ ค่าขนาดหน้าต่างข้อมูลของ Vegas เมื่อบัฟเฟอร์เต็ม

Wrmax คือ ค่าขนาดหน้าต่างข้อมูลของ Reno เมื่อบัฟเฟอร์เต็ม

จากหลักการที่ว่า Vegas พยายามทำให้ $q_v \approx \bar{q_v}$ อยู่ตลอดเวลา และ $Diff \approx \bar{q_v}$

จะได้ว่า

$$W_{v\max} \cdot \frac{D-d}{d} = \bar{q_v}$$

หรือ

$$W_{v\max} = \bar{q_v} \cdot \frac{D}{D-d} = \bar{q_v} \cdot \frac{\tau + B/\mu}{(B-1)/\mu}$$

เมื่อ μ คือ อัตราเร็วในการส่งข้อมูลของ Bandwidth link (packets/s)

τ คือ ค่าการหน่วงเวลา (round-trip propagation delay)

เมื่อให้ $d = \tau + 1/\mu$ และ $D = \tau + B/\mu$ เป็นค่าของ RTTmin และ RTT

จะได้ $W_{r\max} = B + \mu\tau - W_{v\max}$ if $d \approx \tau$ (3.5)

หลังจากบัฟเฟอร์ของคิวเต็ม ความน่าจะเป็นที่แพ็กเก็ตจาก Reno จะถูก Dropped เท่ากับ

$$p_r = \frac{q_{r\max}}{q_{r\max} + q_{v\max}} = \frac{B - \bar{q_v}}{B} \quad (3.6)$$

และความน่าจะเป็นที่แพ็กเก็ตจาก Vegas จะถูก Dropped เท่ากับ

$$p_v = \frac{q_{v\max}}{q_{r\max} + q_{v\max}} = \frac{\bar{q_v}}{B} \quad (3.7)$$

ถ้าสมมติให้แพ็กเก็ตจาก Reno ถูก dropped ไป ขนาดของ window จะลดลงจาก $W_{r\max}$ ไปเป็น $\frac{W_{r\max}}{2}$

จะได้ค่าเฉลี่ยของขนาดหน้าต่างแพ็กเก็ต $\bar{W_r}$ มีค่าเท่ากับ

$$\bar{W_r} = \frac{W_{r\max} + \frac{W_{r\max}}{2}}{2} = \frac{3}{4} \cdot W_{r\max} \quad (3.8)$$

จากนิยาม ค่าเฉลี่ยของความยาวคิว จะเป็นสัดส่วนของ $\bar{W}_r$ คือ

$$\bar{q}_r = \bar{W}_r \cdot \frac{B}{B + \mu\tau} \tag{3.9}$$

แทนสมการที่ (3.8) ในสมการที่ (3.9) จะได้

$$\bar{q}_r = W_{r\max} \cdot \frac{3B}{4(B + \mu\tau)} \tag{3.10}$$

แทนสมการที่ (3.10) ในสมการที่ (3.2) จะได้

$$\frac{\lambda_v}{\lambda_r} = \frac{4 \cdot \bar{q}_v (B + \mu\tau)}{W_{r\max} 3B}$$

$$\frac{\lambda_v}{\lambda_r} = \frac{4 \cdot \bar{q}_v}{3(B - \bar{q}_v)} \tag{3.11}$$

ให้สมการ (3.11) เท่ากับ 1 จะได้

$$\bar{q}_v = \frac{3}{7}B$$

ดังนั้น จะสามารถกำหนดค่า α และ β ได้ตามสมการข้างล่างนี้

$$\alpha = [\frac{3}{7}B - n] \quad \text{และ} \quad \beta = [\frac{3}{7}B]$$

เมื่อ n คือเลขจำนวนเต็มที่สามารถปรับค่าได้เพื่อเพิ่มประสิทธิภาพให้โปรโตคอล TCP

จากตัวอย่างที่กล่าวมาเป็นแนวคิดการพัฒนาที่พิจารณาเพียง Source ของ Reno และ Vegas อย่างละโหนด เพื่อให้ง่ายต่อการวิเคราะห์ ต่อมาถ้าต้องการพิจารณาการเชื่อมต่อโดยให้มี Source ของ Reno จำนวน Nr โหนด และ Source ของ Vegas จำนวน Nv โหนด สามารถกำหนดค่าต่างๆได้ดังนี้
กำหนดให้

$$N_r \cdot W_{r\max} = B + \mu\tau - N_v \cdot W_{v\max} \quad if \quad d \approx \tau$$

เมื่อ Wrmax คือขนาดหน้าต่างข้อมูลของ Reno เมื่อบัฟเฟอร์เต็ม หลังจากบัฟเฟอร์ของคิวเต็ม ให้ความน่าจะเป็นที่แพ็กเก็ตจาก Reno และ Vegas จะถูก Dropped มีค่าดังต่อไปนี้ตามลำดับ

$$p_r = \frac{B - N_v \cdot \bar{q}_v}{B} \quad \text{และ} \quad p_v = \frac{N_v \cdot \bar{q}_v}{B}$$

และเมื่อใช้การหาผลเฉลยดังตัวอย่างข้างต้นก่อนหน้านี้ก็จะทำให้ได้ค่า

$$\bar{q_v} = \frac{3B}{4N_r + 3N_r} \quad \text{และจาก} \quad \bar{q_v} \leq \beta$$

ดังนั้น จะสามารถกำหนดค่า α และ β ได้ตามสมการด้านล่างนี้

$$\alpha = \frac{3B}{4N_r + 3N_r} - n \quad \text{และ} \quad \beta = \frac{3B}{4N_r + 3N_r}$$

จากแนวทางการพัฒนาที่ได้คิดค้นขึ้นจึงได้นำมาพิสูจน์โดยใช้การ Simulation เพื่อเปรียบเทียบค่าอัตราส่วน Throughput ของ Reno ต่อ Throughput ของ Vegas จากค่า α และ β ที่เป็นค่า default กับค่า α และ β ที่ได้ทำการปรับปรุงค่าแล้ว ซึ่งค่าพารามิเตอร์ที่สำคัญที่ใช้ในการจำลองการทำงาน มีดังนี้

1. LAN ทั้ง 2 ฝั่งใช้ Fast Ethernet ความเร็ว 100 Mbps
2. ค่า Delay ระหว่าง Node ถึง Router เท่ากับ 0.1 ms
3. ค่า Propagation delay ระหว่าง Router1 ถึง Router2 เท่ากับ 80 ms
4. Router ใช้การจัดการคิวแบบ DropTail (เนื่องจากต้องการพิจารณาการจัดการแพ็กเก็ตภายใน Router โดยขึ้นอยู่กับค่า α และ β เท่านั้น)
5. สายสัญญาณที่เชื่อมต่อระหว่าง Router 2 ตัว ใช้มาตรฐาน E-1(2.048 Mbps)

การจำลองการทำงานจะแบ่งออกเป็น 2 ส่วน คือ

ส่วนแรกจะวัดค่าอัตราส่วน Throughput ของ Reno ต่อ Throughput ของ Vegas โดยมีการปรับเปลี่ยนจำนวนโหนดของทั้ง Reno และ Vegas โดยให้ขนาดบัฟเฟอร์ของ Router คงที่

ส่วนที่สองจะวัดค่าอัตราส่วน Throughput ของ Reno ต่อ Throughput ของ Vegas โดยมีการปรับเปลี่ยนขนาดของบัฟเฟอร์ โดยที่จำนวนโหนดของทั้ง Reno และ Vegas คงที่

ผลการจำลองการทำงานส่วนแรก

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
4	16	6771	352	3	19.236
8	12	3761	227	3	16.568
12	8	2600	186	3	13.978
16	4	1997	180	3	11.094

ตารางที่ 3.7 ค่าอัตราส่วน Throughput เมื่อ β =3, α = 1 และ Buffer = 500

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
4	16	2098	1547	23	1.356
8	12	2137	1335	22	1.601
12	8	1907	1256	20	1.518
16	4	1747	1207	19	1.447

ตารางที่ 3.8 ค่าอัตราส่วน Throughput เมื่อ β คือค่าที่ปรับปรุง (n =1) และ Buffer = 500

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
4	16	2218	1514	23	1.465
8	12	2194	1299	22	1.689
12	8	1945	1197	20	1.625
16	4	1762	1146	19	1.537

ตารางที่ 3.9 ค่าอัตราส่วน Throughput เมื่อ β คือค่าที่ปรับปรุง (n =2) และ Buffer = 500

N_r	N_v	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
4	16	2271	1501	23	1.513
8	12	2239	1268	22	1.766
12	8	1977	1148	20	1.722
16	4	1776	1090	19	1.629

ตารางที่ 3.10 ค่าอัตราส่วน Throughput เมื่อ β คือค่าที่ปรับปรุง (n =3) และ Buffer = 500

จากตารางที่ 3.7 จะเห็นว่าเมื่อนำ TCP แบบ Reno กับ TCP แบบ Vegas มาต่อใช้งานร่วมกัน โดยให้มีจำนวนโหนดที่แตกต่างกันและใช้ค่า α และ β เป็นค่า default จะเห็นว่าค่า Throughput โดยเฉลี่ยของ TCP แบบ Reno มีค่ามากกว่าค่า Throughput โดยเฉลี่ยของ TCP แบบ Vegas ค่อนข้างมาก (มากกว่า 10เท่า) ซึ่งเกิดความไม่เท่าเทียมกันอย่างมากในการใช้ช่องสัญญาณ ต่อมาพิจารณาตารางที่ 3.8, 3.9 และ 3.10 จะเห็นว่าหลังจากที่ได้ทำการปรับปรุงค่า α และ β แล้วนั้นค่า Throughput โดยเฉลี่ยของ TCP แบบ Reno ค่อนข้างใกล้เคียงกับค่า Throughput โดยเฉลี่ยของ TCP แบบ Vegas อย่างเห็นได้ชัด และเมื่อพิจารณาที่ค่า n จะเห็นได้ว่าค่า n เท่ากับ 1 ให้ผลลัพธ์ที่ค่อนข้างดีที่สุด

ผลการจำลองการทำงานส่วนที่สอง

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
100	26281	6957	3	3.777
200	29798	3261	3	9.138
300	30613	2429	3	12.603
400	30958	1975	3	15.675
500	30833	1934	3	15.943

ตารางที่ 3.11 ค่าอัตราส่วน Throughput เมื่อ $\beta = 3$, $\alpha = 1$ โดยที่ Nr และ Nv เท่ากับ 5

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
100	19590	13793	8	1.420
200	20527	12777	17	1.606
300	19517	13820	25	1.412
400	18175	15099	34	1.204
500	18524	14572	42	1.271

ตารางที่ 3.12 ค่าอัตราส่วน Throughput เมื่อ β คือค่าที่ปรับปรุง (n =1) โดยที่ Nr และ Nv เท่ากับ 5

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
100	19742	13683	8	1.443
200	20049	13261	17	1.512
300	19820	13508	25	1.467
400	18340	14930	34	1.228
500	18847	14240	42	1.323

ตารางที่ 3.13 ค่าอัตราส่วน Throughput เมื่อ β คือค่าที่ปรับปรุง (n =2) โดยที่ Nr และ Nv เท่ากับ 5

Buffer	ACK_R	ACK_V	β	$\frac{ACK_R}{ACK_V}$
100	21684	11691	8	1.855
200	21147	12120	17	1.745
300	20500	12812	25	1.600
400	18757	14502	34	1.293
500	19051	14031	42	1.358

ตารางที่ 3.14 ค่าอัตราส่วน Throughput เมื่อ β คือค่าที่ปรับปรุง (n =3) โดยที่ Nr และ Nv เท่ากับ 5

จากตารางที่ 3.11 จะเห็นว่าเมื่อนำ TCP แบบ Reno กับ TCP แบบ Vegas มาต่อใช้งานร่วมกัน โดยปรับเปลี่ยนขนาดของบัฟเฟอร์ที่ Router และใช้ค่า α และ β เป็นค่า default จะเห็นว่าค่า Throughput โดยเฉลี่ยของ TCP แบบ Reno มีค่ามากกว่าค่า Throughput โดยเฉลี่ยของ TCP แบบ Vegas ค่อนข้างมาก และจะแปรผันตามขนาดของบัฟเฟอร์ กล่าวคือเมื่อขนาดของบัฟเฟอร์เพิ่มขึ้นอัตราส่วน Throughput โดยเฉลี่ยของ TCP แบบ Reno ต่อ Throughput โดยเฉลี่ยของ TCP แบบ Vegas ก็จะเพิ่มตามไปด้วย ซึ่งจะทำให้เกิดความไม่เท่าเทียมกันในการใช้ช่องสัญญาณมากขึ้น ต่อมาพิจารณาตารางที่ 3.12, 3.13 และ 3.14 จะเห็นว่าหลังจากที่ได้ทำการปรับปรุงค่า α และ β แล้วนั้นค่า Throughput โดยเฉลี่ยของ TCP แบบ Reno ค่อนข้างใกล้เคียงกับค่า Throughput โดยเฉลี่ยของ TCP แบบ Vegas อย่างเห็นได้ชัด และเมื่อพิจารณาที่ค่า n จะเห็นได้ว่าค่า n เท่ากับ 1 และ 2 ให้ผลลัพธ์ที่ใกล้เคียงกันแต่ค่อนข้างดีกว่าค่า n ที่เท่ากับ 3

บทที่ 4
สรุปและวิจารณ์

4.1 สรุป

โครงงานนี้เป็นโครงงานซึ่งทำการทดสอบประสิทธิภาพบนโปรโตคอลทีซีพี ซึ่งเป็นโปรโตคอลที่มีการใช้งานกันมาก โดยเฉพาะการใช้งานอินเทอร์เนต ในการพัฒนาระบบอินเทอร์เนตให้มีประสิทธิภาพให้ดีขึ้น ก็จะต้องพิจารณาถึงองค์ประกอบที่เกี่ยวข้อง ดังเช่นในโครงงานนี้ที่ทำการทดสอบถึงองค์ประกอบที่ส่งผลกระทบต่อการทำงานของโปรโตคอลทีซีพี โดยจะทำการจำลองการทำงานโดยใช้โปรแกรม NS แล้วนำผลที่ได้มาวิเคราะห์ และทำการสรุปผลที่ได้ เพื่อนำไปปรับปรุงค่าองค์ประกอบต่าง ๆ ดังกล่าวให้มีความเหมาะสม เพราะฉะนั้นในการจะทำการทดสอบนั้น ผู้ทดสอบจะต้องมีความรู้ความเข้าใจในรูปแบบและกลไกการทำงานของตัวโปรโตคอล

สำหรับการทดสอบในโครงงานนี้ได้เลือกทำการทดสอบกระบวนการทำงานของแต่ละประเภทของโปรโตคอลทีซีพีที่มีการพัฒนามา ได้แก่ Tahoe, Reno, NewReno, SACK และ Vegas TCP รวมทั้งทดสอบรูปแบบจำลองโครงข่ายที่ใช้งานจริง โดยเลือกเอาประเภททีซีพีที่ใช้ในปัจจุบัน มาเปรียบเทียบกับตัวที่มีประสิทธิภาพดีด้วย รวมทั้งการปรับเปลี่ยนค่าพารามิเตอร์ ซึ่งจะส่งผลการทำงานและประสิทธิภาพที่ได้รับด้วย

สำหรับอุปสรรคในการดำเนินโครงงานนี้ ในส่วนของปัจจัยต่าง ๆ ที่ส่งผลกระทบต่อประสิทธิภาพของการทำงานมีอยู่มากมาย ทำให้การพิจารณาในแต่ละปัจจัยอาจจะทำได้ยาก และได้รับผลที่อาจจะไม่ต้องกับความต้องการได้รับ สำหรับในส่วนของรูปแบบคำสั่งที่ใช้ในการจำลองการทำงาน ค่ากำหนดต่าง ๆ ที่ใช้ในการปรับเปลี่ยน ในบางกรณีได้ทำการปรับเปลี่ยนแล้ว ไม่มีเปลี่ยน อาจจะเป็นเพราะค่ากำหนดที่มีการอธิบายไม่ละเอียด อาจจะทำให้เกิดความเข้าใจผิดได้ รวมทั้งเอกสารสำหรับอ้างอิงในโปรแกรมจำลองนี้ มีไม่ครบถ้วน ทำให้เกิดอุปสรรคในระหว่างการดำเนินงาน

4.2 แนวทางการพัฒนาต่อ

ในปริญญานิพนธ์นี้ ทางผู้จัดทำได้ทำการทดสอบประสิทธิภาพของการทำงานของโปรโตคอล TCP ชนิดต่าง ๆ แล้ว โดยทำการทดสอบโดยสนใจในเพียงบางปัจจัยที่ส่งผลกระทบเท่านั้น ซึ่งไม่ครบถ้วนในส่วนที่เกี่ยวข้องต่าง ๆ ดังนั้นในแนวทางการพัฒนาต่อจึงควรทำการทดสอบโดยคำนึงถึงปัจจัยอื่น ๆ ให้ครบถ้วนและทำการปรับค่าให้ตรงกับการใช้งานจริง รวมทั้งนำรูปแบบที่เสนอไปทดลองใช้ หรือทำดัดแปลงสมการให้มีประสิทธิภาพดีขึ้นหรือนำรูปแบบต่าง ๆ มาผสมผสานกัน

ภาคผนวก

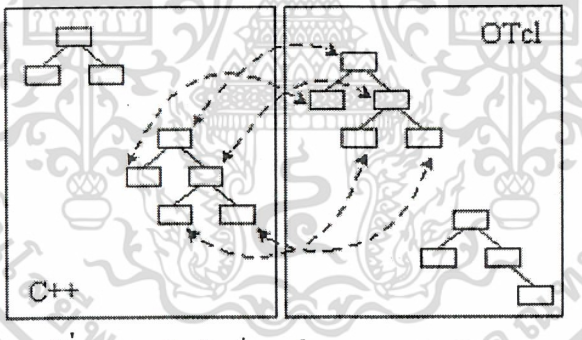
ภาคผนวก

เครื่องมือสำหรับจำลองการทำงาน

1. โปรแกรม NS

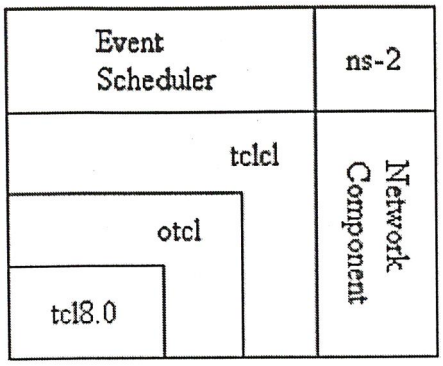
โปรแกรม NS (Network Simulator) เป็นโปรแกรมที่ใช้สำหรับการจำลองระบบเครือข่าย โดยตัวโปรแกรมสามารถทำงานได้บนหลายแพลตฟอร์ม ทั้งในระบบ Unix (ทั้ง FreeBSD, Linux, SunOS, Solaris) และบนระบบ Windows OS ทำให้เป็นที่นิยมใช้กันอย่างมากในกลุ่มผู้ทำการพัฒนาระบบเครือข่าย โดยจะนำโปรแกรมนี้มาจำลองเครือข่ายในลักษณะต่าง ๆ แล้ววิเคราะห์ผลซึ่งได้จากการจำลอง เพื่อนำไปปรับปรุงและพัฒนาต่อไป โดยโปรแกรม NS นี้สนับสนุนการจำลองในหลายรูปแบบ ดังเช่น โปรโตคอล TCP, UDP การใช้ในลักษณะงานประยุกต์ เช่น FTP, Telnet, Web ระบบการจัดคิว เช่น DropTail, RED, CBQ รวมทั้งการใช้งานในแบบมัลติคาสต์ ทั้งบนระบบเครือข่ายทั่วไป (Wired Network) และในระบบเครือข่ายไร้สาย (Wireless Network)

โปรแกรม NS ถูกพัฒนาขึ้นโดยใช้ภาษา C++ และภาษา OTcl (เป็นภาษา Tcl ที่มีลักษณะการเขียนโปรแกรมเป็นแบบเชิงวัตถุ) โดยจะใช้ภาษา C++ ในการจัดการกับตัวข้อมูล ส่วนภาษา OTcl จะใช้ในการควบคุมกระบวนการ โดยสาเหตุที่ใช้ 2 ภาษา ก็เพราะว่า ภาษา OTcl สามารถจัดการกับการควบคุมได้ง่ายและรวดเร็วกว่า แต่ภาษา C++ จะสามารถจัดการตัวข้อมูลได้ดีกว่า ดังนั้นในการพัฒนาโปรแกรมก็จะต้องมีการเชื่อมโยงระหว่าง 2 โปรแกรม ซึ่งจะใช้ฟังก์ชันการทำงานที่เรียกว่า OTcl linkage ดังรูปที่ 1



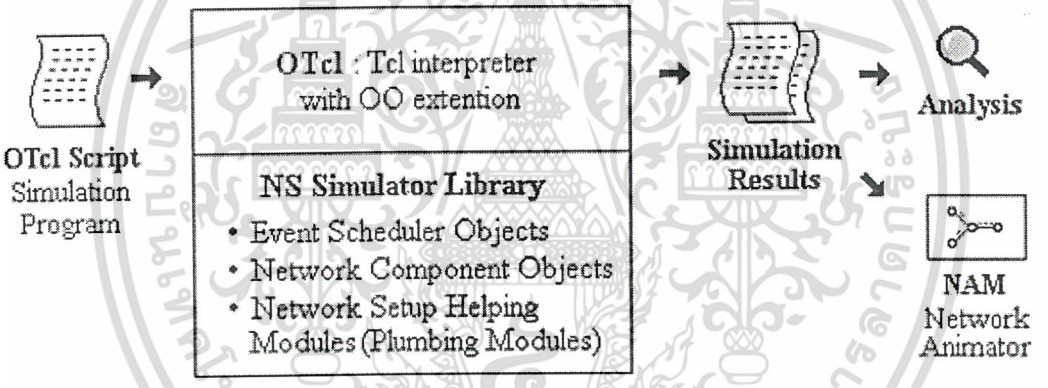
รูปที่ 1 ความสัมพันธ์ระหว่างภาษา C++ กับ OTcl

โดยผู้ที่จะทำการพัฒนารูปแบบการจำลองในรูปแบบใหม่ ก็จะต้องทำการพัฒนาในส่วนนี้ แต่สำหรับการใช้โปรแกรม NS เพื่อจำลองระบบเครือข่ายนั้น จะใช้ภาษา Tcl ในการเรียกฟังก์ชันต่าง ๆที่ได้มีผู้พัฒนามาแล้ว ก็คือเรียกใช้องค์ประกอบในส่วนต่าง ๆ (ซึ่งจะเรียกใช้การทำงานของคลาสต่าง ๆ อีกทีหนึ่ง) นำมาต่อกัน เพื่อให้เกิดเป็นระบบเครือข่ายที่ต้องการ ซึ่งจะเรียกรูปแบบการทำงานนี้ว่า Event Scheduler สำหรับโครงสร้างภายในตัวโปรแกรม NS จะเป็นดังรูปที่ 2



รูปที่ 2 โครงสร้างภายในของโปรแกรม NS

ในการใช้งานโปรแกรม NS เพื่อวิเคราะห์การทำงานนั้น จะมีลำดับขั้นตอนการทำงานดังรูปที่ 3 โดยเมื่อเราทำการเขียนคำสั่งกำหนดรูปแบบเครือข่ายเรียบร้อยแล้ว ก็ทำการเรียกใช้โปรแกรม NS ให้ทำงาน จากนั้นเมื่อได้ผลการทำงานออกมาแล้ว ก็นำผลที่ได้ไปวิเคราะห์ โดยอาจจะใช้โปรแกรมจำลองเส้นทางการเดินทางของข้อมูล ที่ชื่อว่า NAM เพื่อใช้ช่วยในการวิเคราะห์การทำงานได้ดีขึ้น



รูปที่ 3 ลำดับขั้นตอนการทำงานในการวิเคราะห์ระบบเครือข่าย

สำหรับรูปแบบคำสั่งที่ใช้ในโปรแกรม NS จะมีรูปแบบทั่วไป ดังนี้

ns < tcl-script >

ในการใช้งานโปรแกรม NS ในการจำลองการทำงานต่าง ๆนั้น จะใช้คำสั่ง Tcl ในการกำหนดค่าต่าง ๆ โดยจะเรียกใช้รูปแบบการทำงานที่กำหนดไว้แล้ว ซึ่งในรูปแบบต่าง ๆ เหล่านี้จะถูกเรียกใช้จากคลาส Simulator ซึ่งเป็นคลาสหลักของโปรแกรม NS

รูปแบบคำสั่งที่ใช้งานซึ่งเป็นส่วนหลักของโปรแกรม

- สร้างตัวตารางงาน

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

- ลำดับเหตุการณ์

$ns at <time> <event>

โดย <time> เป็นเวลาที่กำหนดในลำดับเหตุการณ์

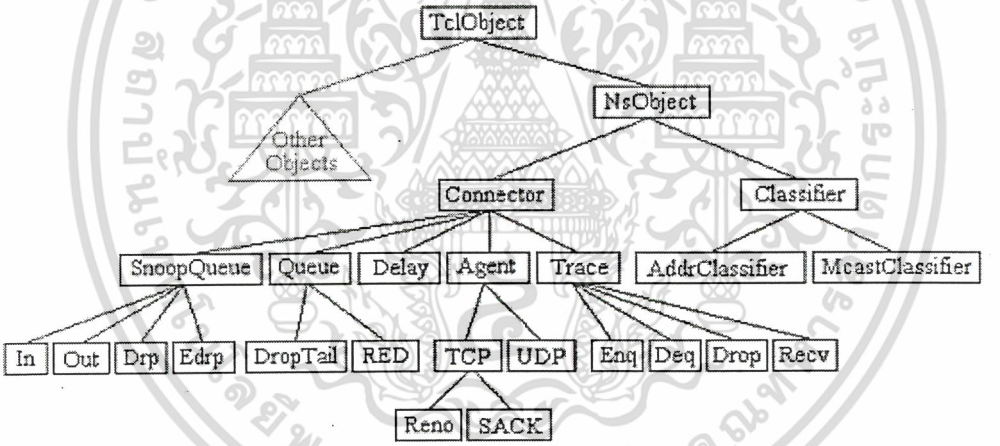
<event> เป็นเหตุการณ์ที่จะกระทำ

- เริ่มต้นตัวตารางงาน

$ns run

1.1 องค์ประกอบของเครือข่ายใน NS

ในเครือข่ายหนึ่ง ๆ จะประกอบไปด้วยส่วนต่าง ๆ มากมาย ที่เชื่อมต่อกัน ในการทำงานร่วมกัน ดังเช่น จุดเชื่อมต่อ (node) ของจุดหนึ่ง จะถูกเชื่อมต่อด้วยเส้นทางการเชื่อมต่อ (link)ไปยังจุดเชื่อมต่ออีกจุดหนึ่ง ซึ่งในแต่ละส่วนจะประกอบไปด้วยองค์ประกอบอีกมากมาย ที่ถูกกำหนดให้ทำงานเป็นส่วน ๆ ดังนั้นในการพิจารณาเครือข่ายอย่างละเอียด จึงควรพิจารณาถึงองค์ประกอบในของในแต่ละส่วน โดยในหัวข้อนี้ จะมาพิจารณาองค์ประกอบของเครือข่ายใน NS ที่จะใช้พิจารณา ในโปรแกรม NS จะแบ่งองค์ประกอบเป็นส่วน ๆ ดังรูปที่ 4



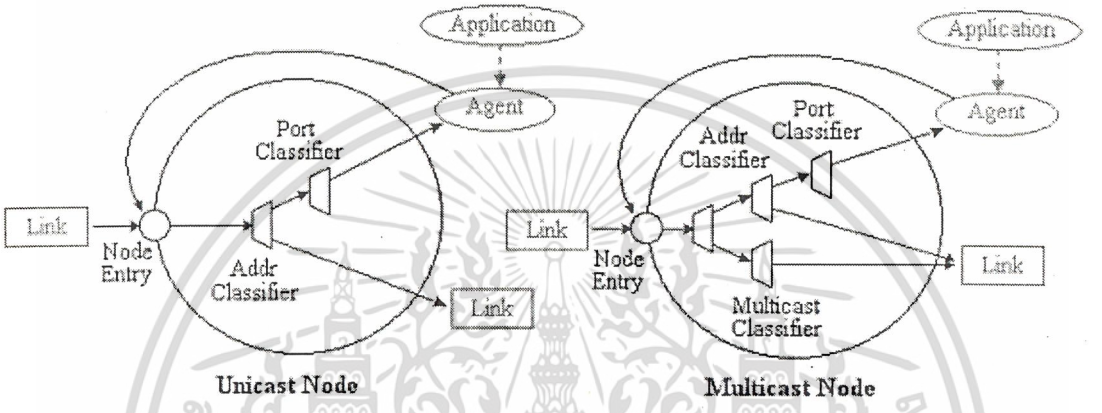
รูปที่ 4 ระดับชั้นขององค์ประกอบในโปรแกรม NS

ในรูปเป็นเพียงส่วนหนึ่ง โดยจะพิจารณาในส่วนที่เกี่ยวข้องนี้ โดยจะมี TclObject เป็นคลาสแม่ของคลาสทั้งหมด โดยมี NSObject เป็นคลาสลูก ซึ่งจะเป็นคลาสแม่ขององค์ประกอบของเครือข่ายทั้งหมด ถ้าเราแบ่งองค์ประกอบต่าง ๆ นี้ ตามจำนวนของเส้นทางของผลลัพธ์ที่เป็นไปได้ จะสามารถแบ่งออกเป็น 2 ประเภท คือ แบบ Connector และแบบ Classifier ก็คือ องค์ประกอบพื้นฐานต่าง ๆ ที่มีเส้นทางของผลลัพธ์ในทางเดียว เช่น Enq, Deq, Drop, Recv ที่ใช้ในการตามข้อมูล (trace) จะจัดอยู่ในประเภท Connector ส่วนประเภท Classifier จะเป็นองค์ประกอบประเภทที่ใช้ในการเลือกเส้นทาง (switching) เมื่อเรานำทั้งสองประเภทมาประกอบรวมกัน ก็จะได้เป็นองค์ประกอบต่าง ๆ

1.1.1 จุดเชื่อมต่อ (Node)

จุดเชื่อมต่อจะประกอบไปด้วย จุดรับข้อมูล (node entry) และองค์ประกอบประเภท Classifier โดยโหนดสามารถแบ่งได้เป็น 2 ประเภท คือ Unicast node และ Multicast node

Unicast node ในส่วนของ Classifier จะประกอบด้วย Addr Classifier ซึ่งใช้สำหรับการค้นหาเส้นทางในแบบ unicast (Unicast Routing) ซึ่งจะเชื่อมต่อกับโหนดอื่น ๆ ต่อโดยเชื่อมต่อผ่าน link และอีกส่วนหนึ่งเรียกว่า Port Classifier ซึ่งใช้เชื่อมต่อกับการทำงานในระดับสูง ดังเช่น เชื่อมต่อกับทีซีพี เป็นต้น ซึ่งจะเรียกว่า ตัวแทน (Agent) ดังแสดงได้ดังรูปที่ 5



รูปที่ 5 โครงสร้างของ Unicast Node และ Multicast Node

Mulitcast node ในส่วนของ Classifier จะแตกต่างกับแบบ Unicast node โดยจะประกอบไปด้วย Classifier แบบ Unicast node และ Multicast classifier ซึ่งทำให้สามารถทำงานได้ทั้งสองแบบ โดยจะมีตัวที่ทำหน้าที่แยกแพ็กเก็ตว่าเป็นแบบใด จะเรียกส่วนนี้ว่า switch สำหรับ Multicast Classifier จะใช้สำหรับการค้นหาเส้นทางแบบ multicast routing โดยจะมีตัวที่ทำหน้าที่กระจายข้อมูลว่าไปให้โหนดต่าง ๆ ที่เชื่อมต่อกันอยู่ จะเรียกส่วนนี้ว่า Replicators สำหรับโครงสร้างนั้นแสดงในรูปที่ 5

สำหรับการใช้งาน Mulitcast routing นั้น จะพิจารณาบิตสูงสุดของ Address ซึ่งเป็นตัวบอกว่าเป็นแบบ Unicast หรือแบบ multicast โดยถ้ามีค่าเป็น 0 จะเป็น Unicast Address ถ้าเป็น 1 จะเป็น Multicast Address

ในการกำหนดค่านั้น โปรแกรม NS จะกำหนดค่าเริ่มต้นเป็นแบบ Unicast node ดังนั้นถ้าจะใช้แบบ Multicast node จะต้องกำหนดค่าเพิ่มเติม เพื่อบอกให้โปรแกรมรู้ก่อน โดยสรุปรูปแบบคำสั่งในการกำหนดรูปแบบ ดังนี้

- สร้างจุดเชื่อมต่อ (node)

```
$ns node
```

- การเปลี่ยนแปลงค่าพารามิเตอร์ของจุดเชื่อมต่อ

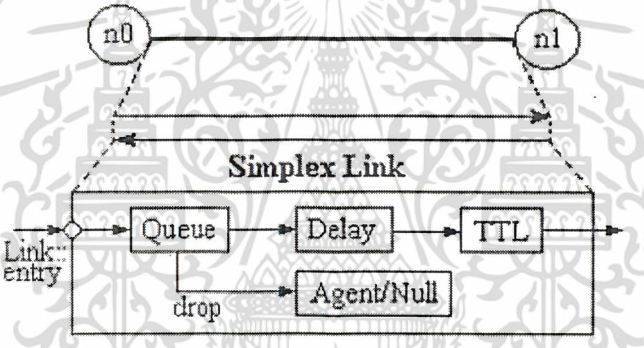
```
$ns node-config - <config-parameter> <optional>
```

1.1.2 เส้นทางการเชื่อมต่อ (Link)

เส้นทางการเชื่อมต่อ หรือ link เป็นส่วนประกอบสำคัญส่วนหนึ่งในเครือข่าย เป็นส่วนที่เชื่อมต่อจุดต่าง ๆ ในเครือข่ายเข้าด้วยกัน ดังที่เห็นกันได้ก็คือ เป็นส่วนเชื่อมต่อระหว่างโหนด เพื่อทำหน้าที่ส่งผ่านแพ็กเก็ตข้อมูลระหว่างกัน

สำหรับในโปรแกรมNS นั้นก็ได้สนับสนุนในหลายรูปแบบ เช่น แบบ multi-access LAN รวมไปถึงการใช้งาน Wireless หรือ broadcast media โดยเราจะพิจารณาในรูปแบบ จุดต่อจุด (point-to-point) ซึ่งถือได้ว่าเป็นลักษณะพื้นฐานของรูปแบบต่าง ๆ

ในลักษณะการเชื่อมต่อโดยทั่วไปนั้น จะเป็นในลักษณะการเชื่อมต่อแบบ 2 ทิศทาง (duplex-link) เมื่อพิจารณาแล้วก็สามารถแบ่งได้เป็นการเชื่อมต่อในแบบทิศทางเดียว (simplex link) ใน 2 ทิศทาง เมื่อพิจารณาในโครงสร้างแล้ว จะมีลักษณะเหมือนกัน ดังนั้นแล้วเราจะพิจารณาในรูปแบบทิศทางเดียว ดังรูปที่ 6



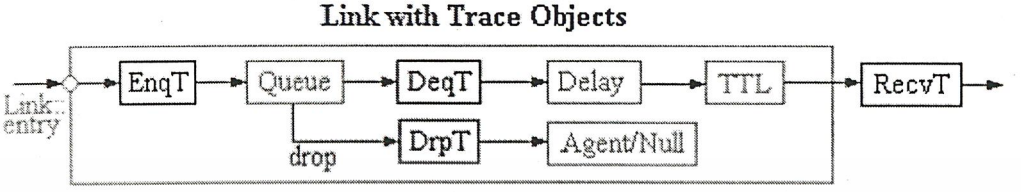
รูปที่ 6 เส้นทางการเชื่อมต่อ (link)

จากรูป จะเห็นว่าภายในนั้นจะประกอบไปด้วยองค์ประกอบ 5 ส่วน คือ

1. Link entry เป็นจุดเข้าของ link ซึ่งเป็นส่วนที่อยู่ด้านหน้าสุดของ link
2. Queue เป็นส่วนหลักของ link เป็นส่วนที่ทำหน้าที่จัดแพ็กเก็ตข้อมูล โดยทั่วไปจะมีเพียง 1 ส่วนเท่านั้นใน 1 link
3. Delay หรืออาจจะใช้คำว่า link delay ในส่วนนี้จะลักษณะของการหน่วงเวลาในการส่งผ่าน
4. TTL (Time To Live) เป็นค่าพารามิเตอร์ของเวลาในการส่งผ่านแพ็กเก็ต ซึ่งจะมีการเปลี่ยนแปลงเมื่อแพ็กเก็ตส่งผ่านส่วนต่าง ๆไป
5. Agent/Null หรือบางทีจะใช้คำว่า drophead เป็นส่วนที่ใช้เก็บแพ็กเก็ตที่ไม่ได้ใช้งาน

การติดตามการเดินทางของแพ็กเก็ต (Tracing)

ในการทดสอบประสิทธิภาพโดยทั่วไป จะมีการติดตามการไหลของแพ็กเก็ตในส่วนต่าง ๆ ว่ามีลักษณะเช่นไร เมื่อมีการใช้งานก็ทำให้ต้องเพิ่มองค์ประกอบเข้าไป แทรกในจุดต่าง ๆ ในองค์ประกอบนั้น ๆ ในส่วนนี้ก็ทำให้โครงสร้างของ link มีลักษณะดังรูปที่ 7



รูปที่ 7 องค์ประกอบของ Traceใน link

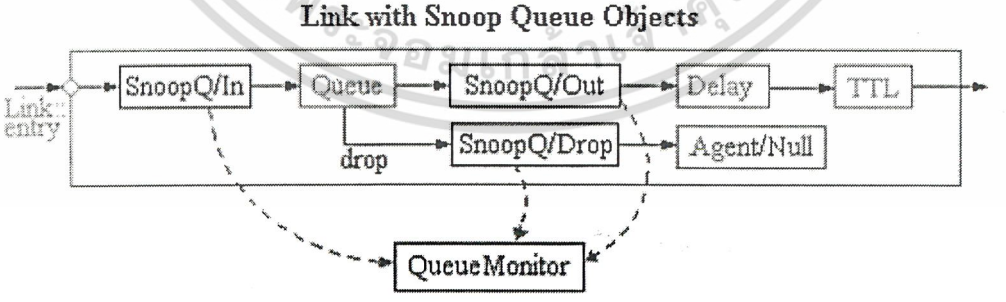
ส่วนที่เพิ่มเข้าไปจะประกอบไปด้วย

1. EnqT เป็นส่วนที่ทำหน้าที่ เก็บผลของแพ็กเก็ต เมื่อก่อนเข้า queue
2. DeqT เป็นส่วนที่ทำหน้าที่เก็บผลของแพ็กเก็ต เมื่อออกจาก queue
3. DrpT เป็นส่วนที่ทำหน้าที่เก็บผลของแพ็กเก็ต เมื่อแพ็กเก็ตถูกปล่อยมาจาก queue
4. RecvT เป็นส่วนที่ทำหน้าที่เก็บผลของแพ็กเก็ต เมื่อแพ็กเก็ตจะถูกส่งไปโหนดต่อไป

สำหรับรายละเอียดเกี่ยวกับการติดตามการไหลของแพ็กเก็ต (Tracing) นี้จะอธิบายเพิ่มเติมในส่วนต่อไป

การตรวจวัดการทำงานของการจัดคิว (Queue Monitor)

ในบางกรณีนั้น เราต้องการข้อมูลของการทำงานของระบบการจัดคิว (Queue) เพื่อไปวิเคราะห์ ไม่ว่าเป็นการจัดคิวในรูปแบบใด จะมีรูปแบบของการทำงานในรูปแบบเดียวกัน ดังรูปที่ 8



รูปที่ 8 องค์ประกอบของการตรวจวัดการทำงานของการจัดคิว

โดยจะทำการเพิ่มตัวตรวจจับการทำงาน ที่เรียกว่า SnoopQ ไปคั่นระหว่างจุดต่าง ๆ จะประกอบด้วย SnoopQ/In, SnoopQ/Out และ SnoopQ/Drop ส่วนต่าง ๆ เหล่านี้ จะประกอบเป็นระบบ การตรวจวัดการทำงานของการจัดคิว

สำหรับรูปแบบคำสั่งที่ใช้งาน มีดังนี้

- การสร้างการเชื่อมต่อ แบบทิศทางเดียว (Simplex-link)

$ns simplex-link <n1> <n2> <bw> <delay> <queue_type> <args>

- การกำหนดค่า การเชื่อมต่อ แบบทิศทางเดียว (สำหรับโปรแกรม NAM)

$ns simplex-link-op <n1> <n2> <option> <args>

- การสร้างการเชื่อมต่อ แบบสองทิศทาง (Duplex-link)

$ns duplex-link <n1> <n2> <bw> <delay> <queue_type> <args>

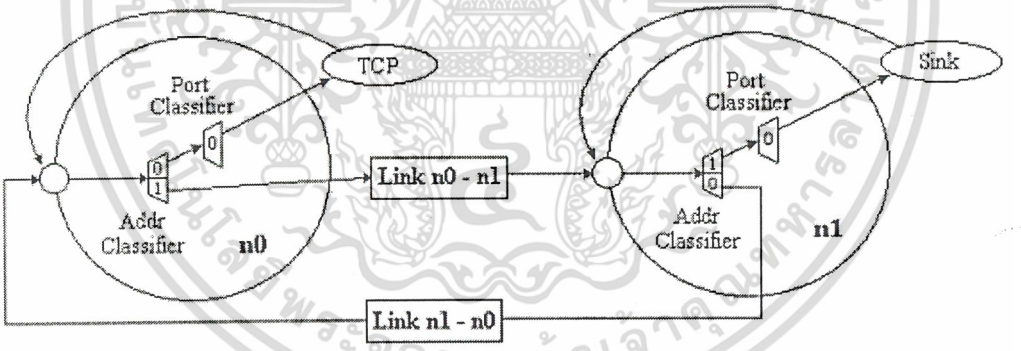
- การกำหนดค่า การเชื่อมต่อแบบสองทิศทาง (สำหรับโปรแกรม NAM)

$ns duplex-link-op <n1> <n2> <option> <args>

โดย <queue_type> เป็นประเภทของการจัดคิว เช่น DropTail, RED, CBQ, FQ, SFQ, DRR

1.1.3. ตัวแทนรูปแบบการทำงานในระดับสูง (Agent)

ตัวแทนรูปแบบการทำงานในระดับสูง หรือที่เรียกว่า Agent ก็คือ รูปแบบต่าง ๆ ของการทำงานในระดับที่สูงขึ้น ที่ทำหน้าที่ต่อจากการทำงานในระดับล่าง ๆ จะเชื่อมต่อกัน (node) โดยจะแบ่งการทำงานออกเป็นส่วน ๆ มีลักษณะเป็นลำดับชั้นไปเรื่อย ๆ จนถึงรูปแบบการทำงานในระดับชั้นโปรแกรมประยุกต์ ดังเช่น TCP Agent, UDP Agent เป็นต้น เมื่อนำไปใช้งานในระบบแล้ว จะมีลักษณะดังรูปที่ 9



รูปที่ 9 ตัวแทนรูปแบบการทำงานในระดับสูง (Agent)

สำหรับโปรแกรม NS ก็ได้สนับสนุน Agents ในหลายรูปแบบที่มีการใช้งาน ดังตัวอย่างข้างล่าง ซึ่งเป็นเพียงส่วนหนึ่งที่โปรแกรม NS สนับสนุน

TCP	เป็นโปรโตคอล TCP แบบ Tahoe ทางด้านส่ง
TCP/Reno	เป็นโปรโตคอล TCP แบบ Reno ทางด้านส่ง
TCP/Newreno	เป็นโปรโตคอล TCP แบบ Reno ที่มีการพัฒนา ทางด้านส่ง
TCP/Sack1	เป็นโปรโตคอล TCP แบบ SACK ทางด้านส่ง
TCP/Fack	เป็นโปรโตคอล TCP แบบ forward SACK ทางด้านส่ง

TCP/FullTcp เป็นโปรโตคอล TCP แบบ 2 ทิศทาง ทางด้านส่ง

TCP/Vegas เป็นโปรโตคอล TCP แบบ Vegas ทางด้านส่ง

TCPSink เป็นโปรโตคอล TCP แบบ Tahoe และ Reno ทางด้านรับ

TCPSink/DelAck เป็นโปรโตคอล TCP แบบ delayed-ACK ทางด้านรับ

TCPSink/Sack1 เป็นโปรโตคอล TCP แบบ SACK ทางด้านรับ

UDP เป็นโปรโตคอลมาตรฐานของ UDP

LossMonitor เป็นส่วนที่ตรวจเช็คการสูญเสียจากการตรวจวัดจำนวนแพ็กเก็ตที่ได้รับ

Null เป็นส่วนที่เก็บแพ็กเก็ตที่ถูกกำจัดทิ้ง

ในรูปแบบต่าง ๆ นี้ เป็นรูปแบบการทำงานที่เรียกว่าเป็นคลาสลูกของ Agent Class ซึ่งจะมีรูปแบบที่เป็นพื้นฐานเหมือน ๆ กัน ดังนั้นในการกำหนดการทำงานต่าง ๆ เหล่านี้จะมีลักษณะเหมือนกัน ในส่วนนี้จะสรุปคำสั่งพื้นฐานที่ใช้กำหนดรูปแบบ ซึ่งจะรูปแบบดังนี้

- สร้างรูปแบบการทำงานของ Agent

  set <Agent_name> [new Agent/<AgentType>]

  โดย <Agent_name> เป็นชื่อตัวแปรที่ใช้แทน Agent

  <AgentType> เป็นชนิดของ Agent เช่น TCP, TCP/Reno, UDP เป็นต้น

- การเชื่อมต่อ Agent กับโหนด

  $ns attach-agent <node> <agent>

- การเชื่อมต่อระหว่างรูปแบบการทำงานของ Agent

  $ns connect <source_agent> <destination_agent>

- การกำหนดการเชื่อมต่อระหว่าง Agent แบบสมบูรณ์

  set <Agent_name> [$ns create-connection <scr_type> <scr> <dst_type> <dst> <pktclass>]

  โดย <pktclass> คือ เป็นการกำหนดคลาสให้การเชื่อมต่อ

- การกำหนดในรายละเอียดของการทำงานของ Agent

  $<agent> set <argument>

1.1.4. ส่วนประยุกต์ใช้งาน (Applications)

ส่วนประยุกต์ใช้งานหรือ Applications เป็นส่วนที่เชื่อมต่อระหว่างการทำงานในระดับชั้นทรานสปอต์ (transport) ที่เราเรียกว่า Agent กับการใช้งานในระดับโปรแกรมประยุกต์ ซึ่งเป็นรูปแบบระดับการทำงานในระดับบน ในโปรแกรม NS นั้นจะแบ่งในส่วนประยุกต์ใช้งานนี้เป็น 2 ประเภท คือ ตัวกำเนิดทราฟฟิก (Traffic generators) และการจำลองโปรแกรมประยุกต์ (Simulated applications)

สำหรับการเชื่อมต่อระหว่างระดับชั้นทรานสปอต์กับระดับชั้นโปรแกรมประยุกต์นั้น ก็จะใช้รูปแบบการรองรับการทำงานที่เรียกว่า Application Programmning Interface (API) หรือที่รู้จักกันในชื่อว่า ซ็อคเก็ต (Sockets)

1.1.4.1. ตัวกำเนิดทราฟฟิก (Traffic generators)

ในโปรแกรม NS จะสนับสนุนอยู่ 4 ประเภท คือ

1. EXPOO Traffic

เป็นตัวกำเนิดทราฟฟิก ที่มีรูปแบบการทำงานแบบ Exponential On/Off โดยอัตราการส่งแพ็กเก็ตจะคงที่ ในช่วงเวลาทำการ (on periods) และจะไม่มีการส่งแพ็กเก็ตในช่วงเวลาหยุด (off periods) ขนาดแพ็กเก็ตมีขนาดคงที่

2 POO Traffic

เป็นตัวกำเนิดทราฟฟิก ที่มีรูปแบบการทำงานแบบ Exponential On/Off เช่นกัน แต่ค่าช่วงเวลาทำการ และช่วงเวลาหยุดนี้ จะใช้รูปแบบของ Pareto

3. CBR Traffic

เป็นตัวกำเนิดทราฟฟิก ที่มีค่าอัตราการส่งแพ็กเก็ตคงที่ ขนาดแพ็กเก็ตคงที่ โดยสามารถกำหนดว่าให้มีการรบกวนเกิดขึ้นในระหว่างการส่งได้

4. TrafficTrace

เป็นตัวกำเนิดทราฟฟิก ที่มีรูปแบบการทำงานตามค่าในไฟล์การตามเส้นทางที่แพ็กเก็ตผ่าน

สำหรับรูปแบบคำสั่งที่ใช้กำหนดการทำงาน จะมีรูปแบบดังนี้

set <Traffic_name> [new Application/Traffic/<Traffic_type>]

$<Traffic_name> attach-agent $<Agent_name>

หรือจะใช้คำสั่งว่า

Set <Traffic_name> [$<Agent_name> attach-app <Traffic_type>]

1.1.4.2. การจำลองโปรแกรมประยุกต์ (Simulated Application)

ในโปรแกรม NS จะสนับสนุนอยู่ 2 ประเภท คือ

1. FTP

เป็นรูปแบบการทำงานที่ใช้สำหรับการถ่ายโอนไฟล์ข้อมูล (file transfer)

2. Telnet

เป็นรูปแบบการทำงานที่ใช้สำหรับการติดต่อระหว่างเครื่องลูกข่ายกับเครื่องแม่ข่าย โดยจะมีรูปแบบการทำงานอยู่ 2 รูปแบบ คือ ถ้าค่า interval_ ไม่เป็นศูนย์ นั้นจำนวนแพ็กเก็ตที่ส่งจะมีรูปแบบ exponential แต่ถ้าค่า interval_ มีค่าไม่เป็นศูนย์ นั้นจำนวนแพ็กเก็ตที่ส่งจะมีรูปแบบของทีซีพี

สำหรับรูปแบบคำสั่งที่ใช้กำหนดการทำงาน จะมีรูปแบบดังนี้

set <Application_name> [new Application/<Application_type>]

$<Application_name> attach-agent $<Agent_name>

หรือจะใช้คำสั่ง

set <Application_name> [$<Agent_name> attach-app <Application_type>]

1.2 กระบวนการหาเส้นทาง (Routing)

กระบวนการหาเส้นทาง เป็นกระบวนการที่จะกำหนดรูปแบบลำดับการเดินทางของแพ็กเก็ตข้อมูลที่มีการสื่อสารกันในเครือข่าย สามารถแบ่งออกเป็น 2 ประเภท คือ ยูนิแคส (Unicast Routing) และมัลติแคส (Multicast Routing)

1.2.1 กระบวนการหาเส้นทางแบบยูนิแคส (Unicast Routing)

รูปแบบกระบวนการหาเส้นทางแบบยูนิแคส ที่โปรแกรม NS สนับสนุน มี 4 แบบคือ

1. Static Routing

เป็นรูปแบบที่ใช้เป็นเค่าเริ่มต้นในการกำหนดกระบวนการหาเส้นทางในโปรแกรม NS เป็นกระบวนการที่ใช้แนวคิดแบบ Dijkstra's all-pairs SPF จะมีการหาเส้นทางเพียงครั้งเดียวเมื่อเริ่มต้นในการจำลองการทำงาน

2. Session Routing

เป็นกระบวนการที่ใช้แนวคิวแบบ Dijkstra's all-pair SPF เช่นกัน แต่การหาเส้นทางนั้น นอกจากจะเกิดขึ้นเมื่อตอนเริ่มต้นแล้ว ก็ยังมีการกระทำอีกครั้ง เมื่อมีการเปลี่ยนแปลงรูปแบบของเครือข่าย

3. DV Routing

เป็นกระบวนการที่ใช้รูปแบบ Distributed Bellman-Ford (หรือ Distance Vector) โดยวิธีนี้จะมีการปรับปรุงค่าเป็นระยะ ๆ

4. Manual Routing

เป็นกระบวนการที่ไม่ได้ใช้การคำนวณในแบบใด ๆ แต่จะใช้การควบคุมจากผู้ใช้งานเอง

สำหรับคำสั่งที่ใช้กำหนดรูปแบบของการหาเส้นทาง จะมีรูปแบบดังนี้

$ns rtproto <routing-protocol> <args>

โดย <routing-protocol> คือ Static, Session, DV, Manual

1.2.2 กระบวนการหาเส้นทางแบบมัลติแคส (Multicast Routing)

รูปแบบกระบวนการหาเส้นทางแบบมัลติแคส ที่โปรแกรม NS สนับสนุน มี 4 แบบคือ

1. Centralized Multicast หรือ CtrMcast

เป็นกระบวนการที่มีลักษณะคล้ายแบบ PIM-SM

2. Dense Mode หรือ DM

เป็นกระบวนการที่ใช้รูปแบบโปรโตคอลแบบ dense-made-like

3. Shared Tree Mode หรือ ST

เป็นรูปแบบของ sparse mode ซึ่งจะใช้โปรโตคอลshared-tree-multicast protocl

4. Bi-directional Shared Tree Mode หรือ BST

เป็นรูปแบบที่อยู่ในระหว่างการพัฒนา

สำหรับคำสั่งที่ใช้กำหนดรูปแบบกระบวนหาเส้นทาง จะมีรูปแบบดังนี้

- กำหนดการใช้งานในรูปแบบมัลติแคส

```
set ns [new Simulator –multicast on]
```

หรือใช้คำสั่งว่า

```
set ns [new Simulator]
$ns multicast
```

- การกำหนดรูปแบบของมัลติแคส

```
$ns mrtproto <type>
```

โดย <type> คือ CtrMcast, DM, ST, BST

1.3 การจัดการการบริการแถวรอคอย (Queue Management)

การจัดการการบริการแถวรอคอย หรือการจัดคิวให้แพ็กเก็ตที่เข้ามาในส่วนต่าง ๆ หรือก็คือ การจัดลำดับความสำคัญของแพ็กเก็ตต่าง ๆ ที่เข้ามารับบริการนั้นว่าแพ็กเก็ตตัวใดจะได้รับการบริการก่อนหรือหลัง ถือได้ว่าการจัดคิวเป็นส่วนสำคัญส่วนหนึ่งของระบบเครือข่าย ที่จะเป็นกำหนดคุณภาพของการให้บริการ (Quality of Service : QoS) ในระบบเครือข่าย

ในโปรแกรม NS นั้นได้สนับสนุนการจัดคิวในหลายวิธี ดังเช่น

1.3.1 Drop-Tail Queuing (FIFO)

เป็นกลไกการทำงานพื้นฐานที่สุดในการจัดคิว การทำงานจะจัดการกับแพ็กเก็ตที่เข้ามาอย่างต่อเนื่อง โดยลักษณะที่ตัวใดมาก่อนก็ได้รับการจัดการก่อน เป็นวิธีการดั้งเดิมที่ใช้กันอย่างแพร่หลายในอุปกรณ์เครือข่ายอย่างง่าย ซึ่งลักษณะการจัดการนั้นถือเป็นการทำงานที่ไม่ดี ในการที่จะให้บริการระบบที่มีความหนาแน่นของทราฟฟิกสูง ๆ

1.3.2 Fair Queuing (FQ)

เป็นกลไกที่มีรูปแบบการทำงานแบบ Round Robin โดยจะทำการจัดเก็บแพ็กเก็ตที่เข้ามาลงไว้ในคิวที่ต่างกัน แล้วจึงให้บริการส่งทีละแพ็กเก็ตต่อคิว แล้วจึงไปให้บริการในคิวต่อไป

1.3.3 Stochastic Fairness Queuing (SFQ)

เป็นกลไกที่มีรูปแบบการทำงานแบบ Round Robin เช่นกัน แต่ทำการเปลี่ยนแปลงการจัดเก็บแพ็กเก็ตที่เข้ามา เอาไว้รวมกันในคิวเดียว โดยใช้ Hashing Function ในการจัดการ

1.3.4 Deficit Round Robin Queuing (DRR)

เป็นกลไกลที่ใช้การกระจายแพ็กเก็ตลงในคิวแบบ SFQ แต่กลไกการทำงานจะมีรูปแบบที่แตกต่างกับแบบ Round Robin คือ จะมีการกำหนดขนาดของแต่คิวไว้ (Quantrum size) ก็คือถ้าคิวนี้ มีแพ็กเก็ตที่ใหญ่เกินไป ในการส่งขนาดแพ็กเก็ตที่เหลือที่ยังไม่ได้ส่งจะถูกนำไปรวมในครั้งต่อไป

1.3.5 Radom Early Detection Queuing (RED)

เป็นกลไกที่มีการจัดคิวต่างจากลักษณะ FIFO โดยวิธีการนี้จะพิจารณาความหนาแน่นของเครือข่ายเบื้องต้น เพื่อใช้การพิจารณาการควบคุมปริมาณข้อมูลที่จะเข้ามาในระบบ จะใช้การเลือกสุ่มการเชื่อมต่อหรือการไหลของแพ็กเก็ตที่ถูกทำเครื่องหมายไว้ ซึ่งจะเกี่ยวข้องกับค่าความยาวเฉลี่ยของคิว (Average queue size) ถ้าค่าความยาวเฉลี่ยของคิวนี้มีค่าสูงขึ้นเกินกว่าระดับจุดจำกัดที่กำหนดไว้แล้ว ก็จะทำการหยุดแพ็กเก็ตบางส่วนไว้ เพื่อให้ความหนาแน่นในเครือข่ายลดลง

1.3.6 Class-Based Queuing (CBQ)

เป็นกลไกที่มีการกำหนดค่า Priority ให้กับแพ็กเก็ต โดยจะให้บริการกับแพ็กเก็ตที่มีค่า Priority สูงก่อน และจะมีการควมคุมการทำงานของจุดใช้งานร่วม (link-sharing)

สำหรับรูปแบบคำสั่งที่ใช้ควบคุมการทำงานในแต่ละแบบจะไม่ขอกล่าวในที่นี้ โดยจะขอกล่าวถึงคำสั่งที่ใช้ควบคุมโดยทั่วไป การกำหนดขนาดของบัฟเฟอร์ที่ใช้ในการจัดคิว จะมีรูปแบบคือ

$ns queue-limit <n1> <n2> <limit>

1.4 การติดตามเส้นทางการเดินทางของแพ็กเก็ต (Trace and Monitoring)

วิธีการที่ใช้ในการติดตามเส้นทางการเดินทางของแพ็กเก็ตนั้น สามารถทำได้ในหลายวิธี โดยทั่วไปแล้วจะมีการเก็บบันทึกรายละเอียดการติดตามเส้นทางการเดินทางของแพ็กเก็ต ในขณะที่ทำการจำลองรูปแบบ หรือทำการบันทึกลงไปในไฟล์ เมื่อผ่านกระบวนการไปแล้ว ในการติดตามเส้นทางการเดินทางของแพ็กเก็ตนี้จะแบ่งออกเป็น 2 รูปแบบ คือ

1.4.1 การติดตามเส้นทางการเดินทางแพ็กเก็ต หรือที่เรียกว่า Trace

จะเป็นการบันทึกข้อมูลของแพ็กเก็ตแยกเป็นแต่ละแพ็กเก็ต โดยจะทำการบันทึกในแต่ละลำดับขั้นในการทำงานซึ่งผ่านในส่วนต่างๆ เช่น แพ็กเก็ตที่มาถึงยังโหนด, แพ็กเก็ตที่พร้อมที่จะออกจากโหนด เป็นต้น

ในการทำการติดตามเส้นทางการเดินทางของแพ็กเก็ตนั้น จะมีการเพิ่ม EnqT, DeqT, DrpT, RecvT เข้าไปในระหว่างเส้นทางการเชื่อมต่อ ดังที่กล่าวไว้แล้วหัวข้อการเชื่อมต่อ ซึ่งในแต่ละส่วนจะทำหน้าที่เก็บข้อมูลการทำงานเอาไว้ โดยสามารถกำหนดส่วนที่เพิ่มเข้าไปได้ โดยใช้คำสั่งว่า

$ns create-trace <type> <file> <source> <destination>

สำหรับรูปแบบคำสั่งที่ใช้ในการกำหนดการทำงาน จะมีรูปแบบดังนี้

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต

```
$ns trace-all <trace_file>
```

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต(สำหรับ NAM)

```
$ns namtrace-all <namtrace_file>
```

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต โดยระบุเฉพาะ

```
$ns trace-queue <n1> <n2>
```

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต โดยระบุเฉพาะ (สำหรับ NAM)

```
$ns namtrace-queue <n1> <n2>
```

- คำสั่งที่ให้บันทึกข้อมูลการทำงาน (เป็นคำสั่งที่ใช้ก่อนที่จะจบการทำงาน)

```
$ns flush-trace
```

สำหรับรูปแบบของไฟล์การติดตามเส้นทางการเดินทางของแพ็กเก็ต (Trace file)
จะมีลักษณะชุดลำดับของค่าต่าง ๆ ดังรูปที่ 10

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop    (at queue)

src_addr : node.port (3.0)
dst_addr : node.port (0.0)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

รูปที่ 10 รูปแบบของไฟล์ติดตามเส้นทางการเดินทางของแพ็กเก็ต

ในแต่ละบรรทัด จะเป็นการทำงานในหนึ่งหน้าที่ โดยแต่ละบรรทัดก็จะแบ่งออกเป็นส่วน ๆ ทั้ง 12 ส่วน ดังนี้

1. event เป็นรูปแบบการทำงาน จะมีอยู่ด้วยกัน 4 ประเภท คือ

   r : receive คือ แพ็กเก็ตมาถึงโหนดแล้ว

   + : enqueue คือ แพ็กเก็ตอยู่ในระหว่างการรอคิว

- : dequeue คือ แพ็กเก็ตที่ได้รับการจัดคิว

d : drop คือ แพ็กเก็ตไม่ได้รับการจัดคิว

2. time เป็นเวลาที่กระทำงาน

3. from node เป็นส่วนที่ใช้บอกว่าแพ็กเก็ตถูกส่งมาจากที่ใด

4. to node เป็นส่วนที่ใช้บอกว่าแพ็กเก็ตจะถูกส่งไปที่ใด

5. packet type เป็นชนิดของแพ็กเก็ต

6. packet size เป็นขนาดของแพ็กเก็ต

7. flags ไม่มีการใช้งาน (จะมีลักษณะเป็น - - - - - - -)

8. fid เป็นส่วนที่ใช้กำหนด ลำดับIP ใน IPv6 ซึ่งยังไม่ได้ใช้งาน แต่จะใช้กำหนดสีให้กับกลุ่มแพ็กเก็ตในโปรแกรม NAM

9. source Address เป็นที่อยู่ของด้านส่ง จะมีรูปแบบเป็น node.port

10. destination Address เป็นที่อยู่ของด้านรับ จะมีรูปแบบเป็น node.port

11. sequence Number เป็นลำดับหมายเลขของแพ็กเก็ต

12. packet ID เป็นการกำหนดค่าของแพ็กเก็ต

1.4.2 การเฝ้าดูการเดินทางของแพ็กเก็ต (Monitor)

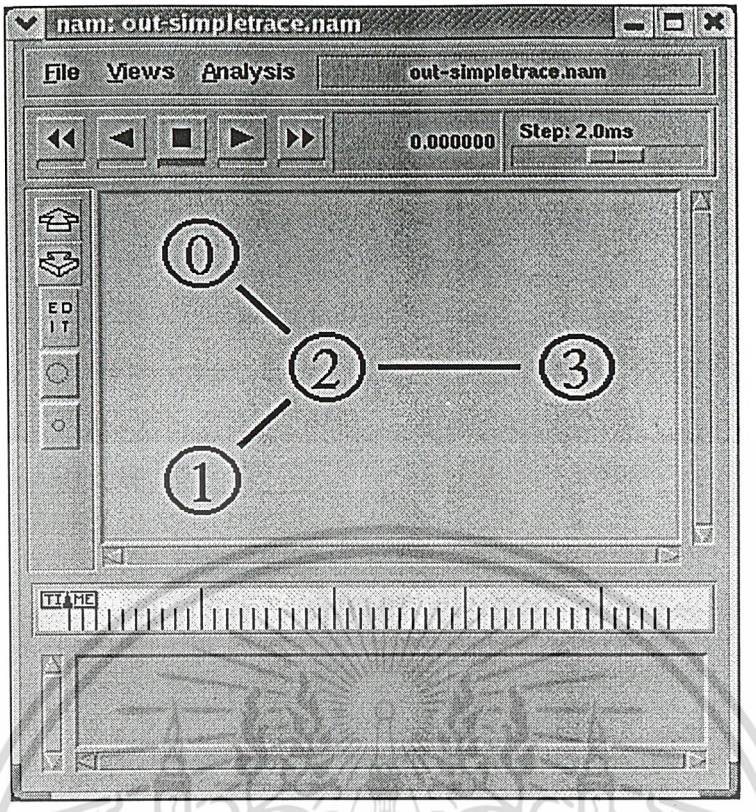
เป็นการบันทึกข้อมูลที่เจาะจงลงไปในส่วนที่สนใจเช่น ในลักษณะของการสนใจปริมาณแพ็กเก็ตที่ได้รับ หรือจำนวนไบต์ที่ได้รับ เป็นต้น ซึ่งในแบบนี้ก็สามารถทำการเฝ้าดูโดยรวมทั้งหมดได้หรือในลักษณะที่เรียกว่า per-flow ซึ่งเป็นรูปแบบของ flow monitor

สำหรับรูปแบบคำสั่งที่ใช้ในการกำหนดการทำงาน จะมีรูปแบบดังนี้

$ns monitor-queue <n1> <n2> <qtrace> <optional:sampleinterval>

2. โปรแกรม NAM

โปรแกรม NAM หรือ Network Animator เป็นโปรแกรมที่ใช้ในการแสดงผลที่ออกมาสำหรับโปรแกรม NS โดยเฉพาะ เพื่อใช้ช่วยในการวิเคราะห์การทำงาน ซึ่งจะทำให้มีความเข้าใจในการทำงานได้ดีขึ้น ลักษณะโปรแกรม NAM แสดงได้ดังรูปที่ 11



รูปที่ 11 โปรแกรม NAM

สำหรับคำสั่งที่เรียกใช้งานโปรแกรม จะอยู่ในรูปแบบภาษา Tcl/Tk จะมีรูปแบบ ดังนี้

- เริ่มต้นการทำงานของโปรแกรม NAM

nam <option> < trace_file>

- การกำหนดให้โปรแกรม NS ทำการบันทึกไฟล์

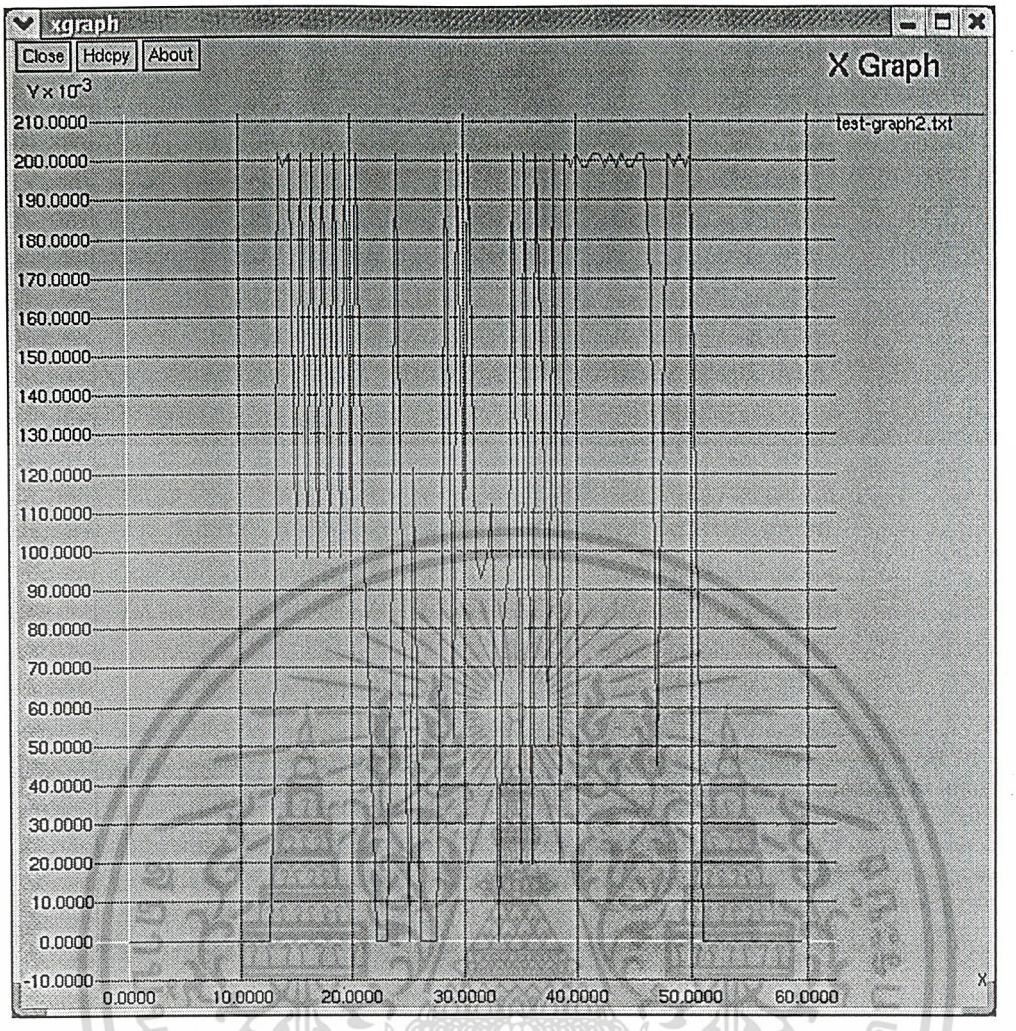
$ns namtrace-all <trace_file>

- การกำหนดให้โปรแกรมทำการเขียนลงไฟล์

$ns flush-trace

3. โปรแกรม Xgraph

โปรแกรม Xgraph เป็นโปรแกรมที่ใช้ในการเขียนกราฟ (plotting program) ในลักษณะกราฟ 2 มิติ หรือกราฟ x-y ซึ่งเป็นที่นิยมใช้กันมากในกลุ่มที่ทำการวิเคราะห์เกี่ยวกับเครือข่าย โดยเฉพาะในกลุ่มที่ใช้งานโปรแกรมวิเคราะห์ NS ซึ่งจะมีฟังก์ชันที่ช่วยในการทำงาน ทั้งการย่อ ขยายกราฟ หรือการเลือกดูเฉพาะส่วน จะช่วยให้การวิเคราะห์ผลที่สะดวกขึ้น รวมทั้งจะมีฟังก์ชันสำหรับการพิมพ์กราฟ โดยในโปรแกรมนี้สามารถกำหนดสีให้กับกราฟต่าง ๆ ได้ในกรณีที่เขียนกราฟหลาย ๆ กราฟ ลักษณะโปรแกรม Xgraph จะมีลักษณะดังรูปที่ 12



รูปที่ 12 โปรแกรม Xgraph

สำหรับข้อมูลที่จะนำมาเขียนกราฟ จะเป็นลักษณะข้อมูลในรูปแบบ x-y โดยข้อมูลจะถูกแบ่งโดยช่องว่าง ไม่ว่าจะเป็น ช่องว่างเดียวหรือการใช้การเว้นระยะแบบ tab หรือถูกแบ่งโดยเครื่องหมาย , ; หรือ :

รูปแบบคำสั่งสำหรับการเรียกใช้งานโปรแกรม Xgraph จะมีรูปแบบดังนี้

xgraph <parameter> <filename>

ในการใช้งานโปรแกรม xgraph เพื่อเขียนกราฟนั้น โดยส่วนมากเราจะนำเอาข้อมูลที่ได้จากการติดตามเส้นทางการเดินทางของข้อมูล (trace file) มาใช้ในการเขียนกราฟ ซึ่งข้อมูลที่ได้มานี้ ก็จะมีรูปแบบตามที่ได้กล่าวมาแล้ว โดยจะเป็นข้อมูลในหลายคอลัมน์ แต่ข้อมูลที่ต้องการนำมาใช้ในการเขียนกราฟ จะใช้เพียงบางคอลัมน์เท่านั้น ดังนั้นจะต้องมีกระบวนการที่จะนำข้อมูลที่ต้องการออกมา ซึ่งสามารถทำได้ในหลายวิธีด้วยกัน โดยส่วนมากแล้วจะใช้วิธีการเขียนโปรแกรมย่อย หรือที่เรียกว่า สคริพส์ (script) แทรกไว้ในการกำหนดทำงานของ NS สำหรับภาษาที่นิยมใช้ในการเขียนสคริพส์นี้ จะใช้ภาษา Awk หรือ Perl Script เหตุที่นิยมใช้ 2 ภาษานี้ในการเขียนสคริพส์ ก็เพราะว่า ทั้ง 2 ภาษานี้เป็นภาษาที่มีคำสั่งสำหรับการจัดการกับข้อความต่างๆ เป็นอย่างดี และมีรูปแบบคำสั่งที่ใช้งานไม่ยาก

กิตติกรรมประกาศ

โครงงานนี้จะไม่สำเร็จได้ หากไม่ได้ความช่วยเหลือจากบุคคลหลาย ๆ ท่านที่ให้ความช่วยเหลือในด้านต่างๆ โดยเฉพาะ ผศ. นภัทร สระเอี่ยม อาจารย์ที่ปรึกษาซึ่งให้คำแนะนำตั้งแต่เริ่มต้นตลอดจนเสร็จสิ้น, นาย ปวีณ ศรีวิโรจน์ และนาย มงคล เฟื่องฟูตระกูล รุ่นพี่ซึ่งทำโครงงานในด้านมาก่อน ซึ่งถือว่าเป็นแนวทางแรกที่โครงงานดำเนินไปได้ และอินเตอร์เน็ตซึ่งแหล่งความรู้หลักในการหาข้อมูลในการทำโครงงานนี้

หนังสือและเอกสารอ้างอิง

[1] สุรศักดิ์ สงวนพงษ์ , " สถาปัตยกรรมและโปรโตคอลทีซีพี/ไอพี " , ซีเอ็ดยูเคชั่น , 2543

[2] ทรงเกียรติ ภาวดี , "เริ่มเขียนสคริปต์ด้วยภาษา PERL" , วิตตี้ กรุ๊ป , 2542

[3] Brent B. Welch , " Practical Programming in Tcl and Tk " , Prentice Hall Ptr , 1995

[4] OTcl Tutorial," ftp://ftp.tns.lcs.mit.edu/pub/otcl/doc/ "

[5] Steven McCanne and Sally Floyd , "NS (Network Simulator)" URL:http://www-nrg.ee.lbl.gov/ns

[6] NS Program , " http://www.isi.edu/nsnam/ns/"

[7] The NS Manuals," http://www.isi.edu/nsnam/ns/ns-documentation.html "

[8] Larry Wall and Randai L.Schwartz , "Programming Perl" , O'Reilly & Associates , 1991

[9] Alfred V. Aho , " The AWK Pragamming Language " , Addison-Wesley Publishing , 1988

[10] J. Olsen , " Stochastic modeling and simulation of the TCP protocol "

[11] Kevin Fall and Sally Floyd , "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP"

[12] J.Mo, Richard J. La, V. Anantharm and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas" , 1998

[13] "Comparison of TCP Reno and TCP Vegas"

[14] INFOCOM2000 , "Modeling TCP Latency"

[15] Jong Suk Ahn, Peter B. Danzig, Zhen Liu and Limin Yan , "Evaluation of TCP Vegas: Emulation and Experiment"

[16] Sally Floyd, "Simulation Tests" , Jul 1995. URL http://www-nrg.ee.lbl.gov/nrg-papers.html

[17] Sally Floyd, "TCP and Succesive Fast Retransmits" URL ftp://ftp.ee.lbl.gov/papers/fastrans.ps

[18] J.Mundarath, N.Vendataranaiah, R. Huang , "TCP Variants Simulation-Based Analysis"

[19] Sally Floyd , "Issues of TCP with SACK" , Mar 1996. URL ftp://ftp.ee.lbl.gov/papers/issues_sa.pz.Z

[20] M. Allman , A. Falk , " On the EffectiveEvaluation of TCP "