การทดสอบประสิทธิภาพของโปรโตคอล TCP

TCP PERFORMANCE TEST

โดย

นางสาวปาณิสรา บุญเพิ่ม

นายพรชัย เปลี่ยมทรัพย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

การทดสอบประสิทธิภาพของโปรโตคอล TCP

TCP PERFORMANCE TEST

โดย

นางสาวปาณิสรา บุญเพิ่ม 47015019

นายพรชัย เปลี่ยมทรัพย์ 47015020

อาจารย์ที่ปรึกษา

ผศ. นภัทร สระเอี่ยม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การทดสอบประสิทธิภาพของโปรโตคอล TCP**

TCP PERFORMANCE TEST

ผู้จัดทำ

1. นางสาวปาณิสรา บุญเพิ่ม 47015019

2. นายพรชัย เปลี่ยมทรัพย์ 47015020

..... อาจารย์ที่ปรึกษา

(ผศ. นภัทร สระเอี่ยม)

การทดสอบประสิทธิภาพของโปรโตคอล TCP

TCP PERFORMANCE TEST

โดย นางสาวปาณิสรา บุญเพิ่ม 47015019
นายพรชัย เปลี่ยมทรัพย์ 47015020

อาจารย์ที่ปรึกษา ผศ. นภัทร สระเอี่ยม

บทคัดย่อ

ในปัจจุบันนี้ การสื่อสารด้วยระบบคอมพิวเตอร์ ได้มีความสำคัญต่อการดำเนินการต่างๆ เป็นอย่างมาก ทั้งด้านธุรกิจ การศึกษา หรือแม้แต่องค์กรของรัฐบาล ซึ่งทำให้การใช้งานคอมพิวเตอร์ไม่ได้ถูกจำกัดอยู่แค่การใช้งานส่วนบุคคลเท่านั้น แต่จะเป็นการแลกเปลี่ยนข้อมูลข่าวสารระหว่างคอมพิวเตอร์ โดยอาจจะเป็นทางอินเตอร์เน็ต หรือเครือข่ายภายในองค์กรเอง การส่งข้อมูลข่าวสารระหว่างเครือข่ายที่มีประสิทธิภาพจึงเป็นที่ต้องการ ดังนั้นโครงงานนี้จึงจัดทำขึ้นเพื่อทดสอบประสิทธิภาพของเครือข่าย และวิเคราะห์ระบบเครือข่ายบนโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP โดยใช้โปรแกรม Network Simulator (NS) ในการทดสอบ

ABSTRACT

Nowaday, communication by computer network is important to work, business, education and ect. 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 SACK TCP and Vegas TCP. To Test by Network Simulator (NS) Program .

กิตติกรรมประกาศ

โครงงานนี้จะสำเร็จไม่ได้ ถ้าไม่ได้ความช่วยเหลือจาก ผศ. นภัทร สระเอี่ยม อาจารย์ที่ปรึกษา ซึ่งคอยให้คำแนะนำต่างๆ ตลอดจนให้ความช่วยเหลือในโครงงานนี้ รวมทั้งอินเตอร์เน็ต ห้องสมุดของทางสถาบันซึ่งเป็นแหล่งข้อมูลหลักในการหาข้อมูล และการให้ความสนับสนุนจากพ่อและแม่

สารบัญ

หน้า

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันระบบเครือข่ายคอมพิวเตอร์ได้เข้ามามีบทบาทสำคัญในการติดต่อสื่อสารต่างๆ ดังนั้น ในการที่จะใช้คอมพิวเตอร์เชื่อมต่อกับระบบเครือข่ายนั้น จึงจำเป็นต้องมีความรู้ความเข้าใจถึงโปรโตคอลต่างๆ ที่ใช้เป็นมาตรฐานในการติดต่อสื่อสาร

ปริญญานิพนธ์เล่มนี้ นำเสนอโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP โดยใช้โปรแกรมจำลองการทำงานระบบเครือข่าย Network Simulator (NS) โดยเลือกให้โปรแกรม NS ทำงานบนระบบปฏิบัติการลีนุกซ์ (Linux) เนื่องจากระบบปฏิบัติการลีนุกซ์เป็นระบบปฏิบัติการที่ใช้ทรัพยากรน้อย ทำงานได้อย่างมีประสิทธิภาพและมีความเสถียรภาพในการทำงาน

1.2 ขอบเขตของปริญญานิพนธ์

จะเป็นการศึกษาในรายละเอียดของระบบเครือข่ายในส่วนของโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP รวมถึงองค์ประกอบต่างๆ ที่เกี่ยวข้อง แล้วทำการจำลองรูปแบบการทำงานเพื่อทดสอบประสิทธิภาพ จากนั้นนำผลที่เกิดขึ้นมาวิเคราะห์ และทำการเปรียบเทียบระหว่างโปรโตคอล TCP ทั้ง 2 ชนิด

1.3 วิธีการดำเนินงาน

ในส่วนแรก จะทำการศึกษาและรวบรวมข้อมูลที่เกี่ยวข้องกับโครงสร้างและการทำงานของโปรโตคอล TCP, SACK TCP, Vegas TCP รวมถึงศึกษาการใช้โปรแกรม NS ในการจำลองรูปแบบของระบบเครือข่าย ในส่วนที่ 2 จะทำการทดสอบประสิทธิภาพโดยใช้โปรแกรม NS จำลองการทำงานระบบเครือข่าย แล้วนำผลที่ได้ไปวิเคราะห์ และทำการเปรียบเทียบระหว่างโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP โดยในส่วนสุดท้ายจะสรุปถึงสิ่งที่มีผลกระทบต่อประสิทธิภาพในการทำงานของโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP

บทที่ 2
ทฤษฎีและหลักการ

2.1 โปรโตคอล TCP (Transmission Control Protocol)

โปรโตคอล TCP เป็นโปรโตคอลในชั้นขนส่งข้อมูล (Host-to-Host Layer) ของ TCP/IP ได้รับการออกแบบมาให้เป็นโปรโตคอลที่ไว้ใจได้เพื่อใช้ในการสื่อสารผ่านระบบเครือข่ายทั่วไปที่อาจมีความผิดพลาดเกิดขึ้นในขณะนำส่งข้อมูลอยู่เสมอ ทั้งนี้การสื่อสารผ่านระบบเครือข่ายทั่วไปนั้นมีความแตกต่างกันมากมายในเรื่องรูปแบบเครือข่าย ขนาดช่องสัญญาณ ขนาดแพ็กเก็ต และพารามิเตอร์ที่ใช้ ดังนั้น โปรโตคอล TCP จึงถูกออกแบบมาให้สามารถปรับตัวเข้ากับความแตกต่างเหล่านี้ได้เป็นอย่างดี รวมทั้งสามารถทนทานต่อความล้มเหลวในระหว่างการนำส่งข้อมูลที่อาจเกิดขึ้นเมื่อใดและในส่วนใดก็ได้

โปรโตคอล TCP ถูกกำหนดเป็นมาตรฐานรุ่นแรกเรียกว่า RFC 793 ต่อมาได้รับการปรับปรุงแก้ไขเรื่อยมาจนออกมาเป็นมาตรฐาน RFC 1122 และ RFC 1323 ตามลำดับ

โฮสต์ที่สนับสนุนการทำงานโปรโตคอล TCP จะต้องมีเอ็นติตี้ TCP ซึ่งอาจจะเป็นส่วนประกอบของโปรเซสฝั่งรับ หรือเป็นส่วนหนึ่งของระบบปฏิบัติการ มีหน้าที่ในการบริหารข้อมูลของ TCP และติดต่อกับชั้นสื่อสารที่ควบคุมแพ็กเก็ตไอพีโดยตรง เอ็นติตี้ TCP รับข้อมูลจากโปรเซสผู้ใช้แบ่งข้อมูลออกเป็นส่วนย่อยที่มีขนาดไม่เกิน 64 กิโลไบต์ (โดยปกติมีขนาดประมาณ 1,500 ไบต์) แล้วจึงส่งไปยังฝั่งรับในรูปแบบของดาต้าแกรมไอพีแพ็กเก็ต เมื่อแพ็กเก็ตเดินทางมาถึงก็จะถูกส่งต่อให้กับเอ็นติตี้ TCP ซึ่งจะสร้างข้อมูลให้กลับไปอยู่ในสภาพเดิม

เนื่องจากชั้นสื่อสารที่ควบคุมการรับ-ส่งแพ็กเก็ตไอพีไม่มีการรับประกันความสำเร็จในการนำส่งข้อมูล ดังนั้นจึงเป็นหน้าที่ของเอ็นติตี้ TCP ที่จะต้องกำหนดระยะเวลารอคอย (time wait) ให้เหมาะสม และทำการส่งแพ็กเก็ตซ้ำในกรณีที่จำเป็น แพ็กเก็ตที่เดินทางมาถึงอย่างสมบูรณ์ก็อาจอยู่ในลำดับที่ไม่ถูกต้อง จึงเป็นหน้าที่ของเอ็นติตี้ TCP ที่จะต้องจัดเรียงลำดับข้อมูลให้ถูกต้อง กล่าวโดยสรุปคือ เอ็นติตี้ TCP มีบทบาทสำคัญในการจัดการนำส่งข้อมูลให้แก่ผู้ใช้ได้อย่างถูกต้องสมบูรณ์บนพื้นฐานของการนำส่งข้อมูลแบบไอพีที่ไม่มีความสมบูรณ์ในตัวเอง

TCP จะกำหนดการสื่อสาร Process-to-Process โดยใช้ port number ตามตารางที่ 2.1 เป็นรายชื่อของ well-know port number ที่ใช้ใน TCP

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	User	Active users
13	Daytime	Returns the date and the time

ตารางที่ 2.1 รายชื่อของ well-know port number ที่ใช้ใน TCP

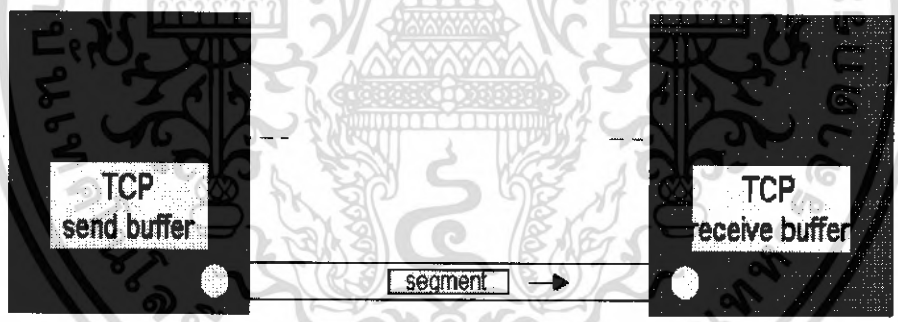
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File transfer Protocol (data connection)
21	FTP, Control	File transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple mail transfer protocol
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

ตารางที่ 2.1 (ต่อ) รายชื่อของ well-know port number ที่ใช้ใน TCP

2.2 บริการของ TCP

2.2.1 บริการส่งข้อมูลเป็นสตรีม

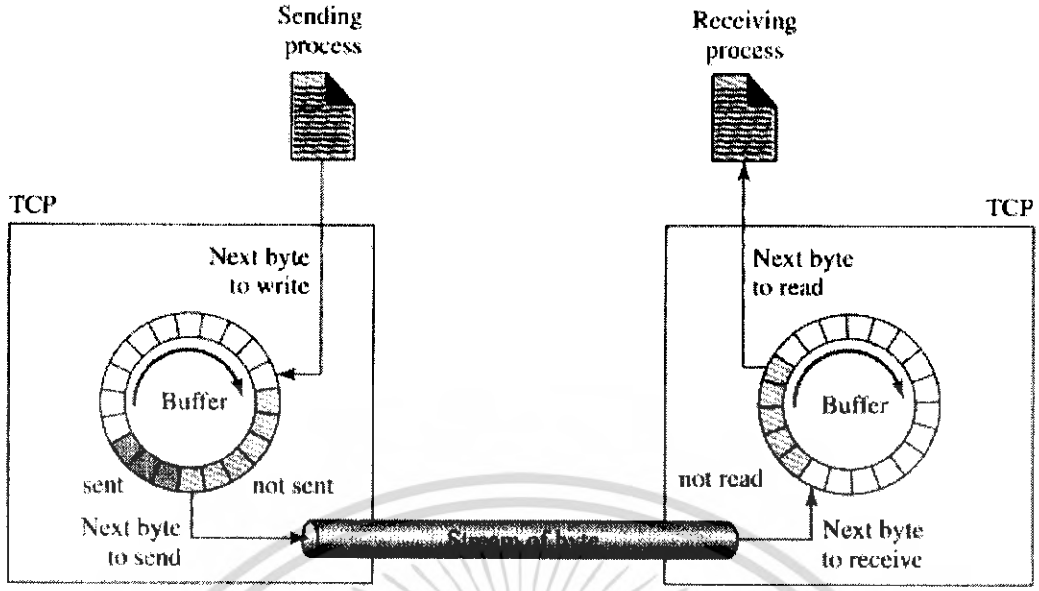
TCP จะมีการรับส่งข้อมูลเป็นไบต์ต่อเนื่องกันไปก่อนการรับส่งข้อมูลโปรเซสทั้งสองจะต้องสร้างการติดต่อก่อน เมื่อติดต่อได้แล้วจะเสมือนว่ามี " ท่อ " ที่ใช้ในการลำเลียงข้อมูล



รูปที่ 2.1 การส่งข้อมูลเป็นสตรีม

บัฟเฟอร์สำหรับรับข้อมูล และบัฟเฟอร์สำหรับส่งข้อมูล

เนื่องจากโปรเซสในการรับส่งข้อมูลอาจมีอัตราการรับส่งข้อมูลที่ไม่เท่ากัน ดังนั้น TCP จึงต้องมีบัฟเฟอร์เพื่อช่วยในการเก็บข้อมูล และบัฟเฟอร์จะมีอยู่สองแบบคือบัฟเฟอร์สำหรับรับข้อมูลและบัฟเฟอร์สำหรับส่งข้อมูล

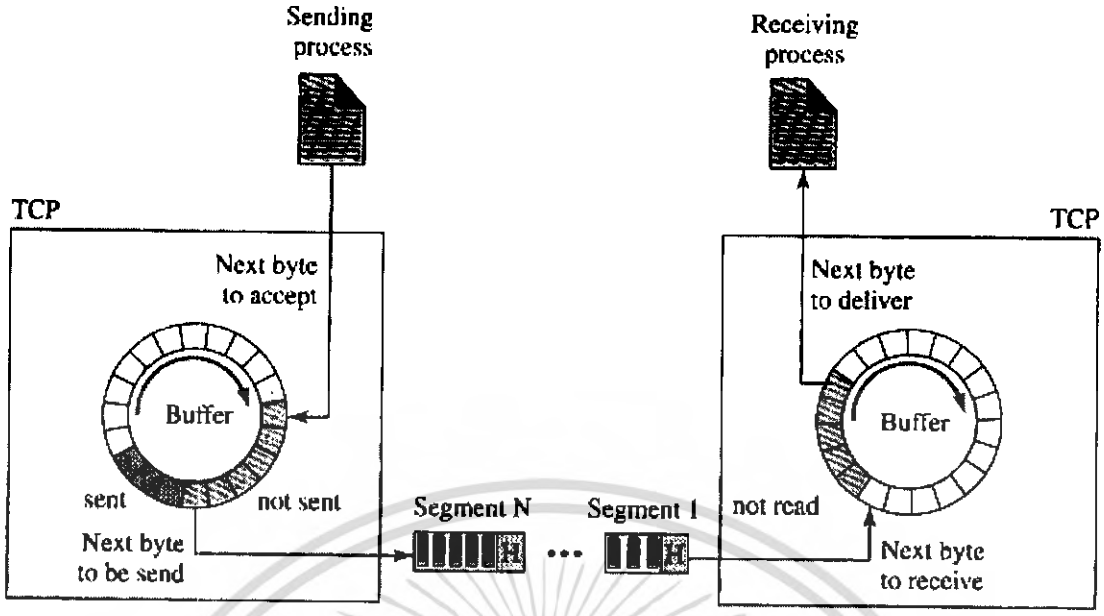


รูปที่ 2.2 บัฟเฟอร์ของการรับและส่งข้อมูล

จากรูปที่ 2.2 บัฟเฟอร์จะช่วยในการรับและส่งข้อมูล ซึ่งตัวอย่างนี้จะเป็นการส่งข้อมูลทิศทางเดียว โดยฝั่งส่งจะนำข้อมูลมาเก็บไว้ในบัฟเฟอร์ก่อนแล้วค่อยทยอยส่งออกไป เมื่อได้รับการตอบกลับมาว่าได้รับข้อมูลนั้นแล้วจึงทำการลบข้อมูลในส่วนนั้นออกไป เมื่อบัฟเฟอร์เต็มจะต้องหยุดรอจนกว่าจะมีการตอบรับกลับมา ส่วนทางด้านฝั่งรับเมื่อได้รับข้อมูลแล้วจะนำมาเก็บไว้ในบัฟเฟอร์ก่อนแล้วค่อยทยอยส่งให้กับโปรเซส แล้วทำการตอบกลับไปยังฝั่งส่งว่าได้รับข้อมูลแล้ว จากนั้นจะทำการลบข้อมูลส่วนนั้นทิ้งไปและรอรับข้อมูลส่วนใหม่เข้ามา

เซ็กเมนต์

ถึงแม้ว่า การใช้บัฟเฟอร์จะสามารถช่วยแก้ปัญหาในเรื่องของอัตราเร็วการรับส่งข้อมูลที่ไม่เท่ากันได้ แต่ยังต้องมีขั้นตอนมากกว่านั้นก่อนจะทำการส่งข้อมูลได้ เนื่องจากโปรโตคอลไอพีจะส่งข้อมูลเป็นแพ็กเก็ต ไม่ใช่แบบ stream of byte ดังนั้น TCP จะต้องมีการแบ่งข้อมูลออกเป็นแพ็กเก็ตให้มีขนาดเหมาะสม ซึ่งเรียกข้อมูลแบบนี้ว่า TCP เซ็กเมนต์ โดยจะเพิ่มเฮดเดอร์เข้าไปในแต่ละเซ็กเมนต์ จากนั้นจึงส่งให้กับไอพี แล้วไอพีจะทำการห่อหุ้ม (encapsulate) เซ็กเมนต์ เป็นไอพีดาต้าแกรม ดังรูปที่ 2.3



รูปที่ 2.3 TCP เซ็กเมนต์

TCP เซ็กเมนต์ กับ MSS (Maximum Segment Size)

ข้อดีของการกำหนด TCP เซ็กเมนต์ คือ โดยทั่วไปการกำหนดขนาดของข้อมูลที่จะทำการส่งอย่างมีประสิทธิภาพนั้นมีปัจจัยสำคัญคือ สื่อกลางในการสื่อสารซึ่งอยู่ชั้นที่ต่ำลงไปเกินกว่าชั้นโปรโตคอลประยุกต์ (Application Layer) สามารถควบคุมได้ เช่น ในชั้นเชื่อมโยงข้อมูล (Data Link Layer) ที่แตกต่างกันอย่าง อีเทอร์เน็ต กับ โทเคนริง ก็จะมีขนาดของ MTU (Maximum Transfer Unit) แตกต่างกัน ดังนั้นหากให้แอพพลิเคชั่นเป็นตัวกำหนดขนาดของข้อมูลก็จะทำให้ได้ผลดีเฉพาะกับเครือข่ายแบบหนึ่ง แต่หากชั้นเชื่อมโยงข้อมูลเปลี่ยนไปก็จะเกิดปัญหาขึ้นได้

TCP มีกลไกกำหนดปัญหาเหล่านี้ โดยจะมีการสอบถามขนาดของข้อมูลที่เหมาะสมเรียกว่า MSS ก่อนการเริ่มส่งข้อมูลเพื่อให้ปลายทางตอบกลับมาได้ว่าสามารถที่จะรับส่งข้อมูลขนาดของ MSS ได้โดยต้องทำการแฟรกเมนต์หรือไม่ หากปลายทางเห็นว่าไม่สามารถจะทำได้ก็จะส่งค่า MSS กลับมาให้ในขนาดที่ลดลง จึงเริ่มทำการรับส่งข้อมูลที่ขนาดของเซ็กเมนต์เท่า MSS ที่ตกลงกันได้ทั้ง 2 ฝั่ง โดยทั่วไปค่า MSS จะเท่ากับขนาดของ MTU-IP header - TCP-header ซึ่งขนาดของ MTU-40 ที่เป็นค่าดีฟอลต์จะเท่ากับ 536 ไบต์ สำหรับการส่งผ่านเครือข่ายที่ไม่ใช่ เครือข่ายท้องถิ่น (Local Area Network) และเพิ่ม 20 ไบต์ สำหรับไอพีเฮดเดอร์ 20 ไบต์ สำหรับ TCP เฮดเดอร์ เพื่อให้มีขนาดทั้งหมด 576 ไบต์

2.2.2 บริการส่งข้อมูลสองทิศทาง (Full Duplex)

TCP จะส่งข้อมูลแบบสองทิศทางเนื่องจากมีบัฟเฟอร์สำหรับการรับส่งข้อมูลที่แยกออกจากกัน ดังนั้นจึงสามารถรับและส่งข้อมูลได้ในเวลาเดียวกัน

2.2.3 บริการ Connection-Oriented

TCP เป็นโปรโตคอลแบบ connection-oriented ซึ่งจะมีการสร้างเส้นทางเสมือน (virtual connection) ระหว่างต้นทางและปลายทางก่อนที่จะทำการส่งข้อมูลกันได้ เมื่อการรับส่งข้อมูลนั้นเสร็จสิ้น เส้นทางนี้จะถูกยกเลิกทันที

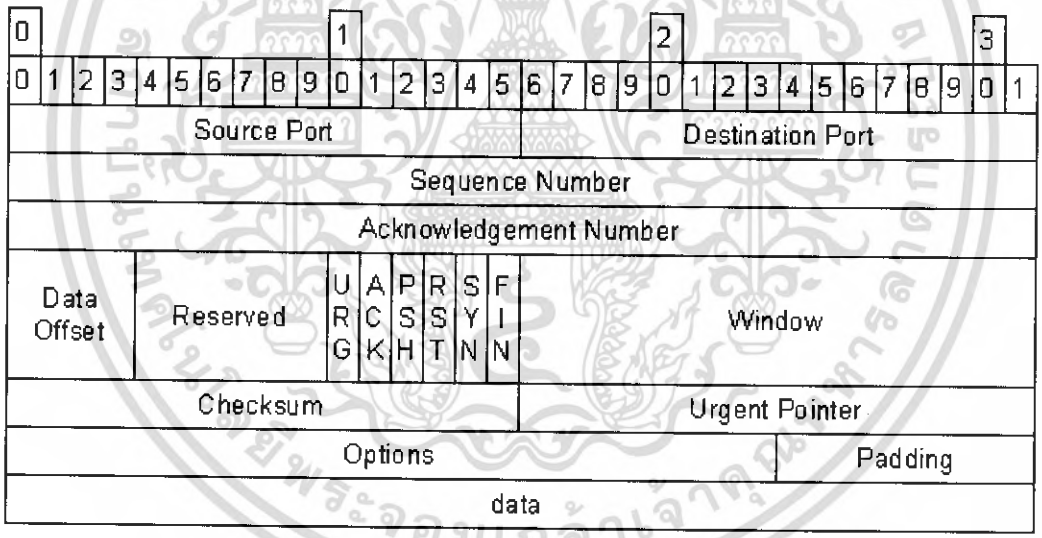
2.2.4 บริการรับประกันความถูกต้อง (Reliable)

TCP เป็นโปรโตคอลที่จะรับประกันความถูกต้องของข้อมูลโดยใช้กลไกของการตอบกลับเมื่อได้รับข้อมูลเรียบร้อยแล้ว

2.2.5 End-to-End Semantic

TCP มีความน่าเชื่อถือตรงที่มี end-to-end semantic เป็นพื้นฐาน โดยที่ ACK ถูกสร้างขึ้นจากฝั่งรับหลังจากที่ข้อมูลไปถึงฝั่งรับอย่างถูกต้องเท่านั้น ดังนั้นเมื่อฝั่งส่งได้รับ ACK มาแล้วจะรับรองได้ว่าข้อมูลนั้นได้ไปถึงฝั่งรับอย่างปลอดภัย ซึ่งกระบวนการนี้คือ end-to-end semantic จะเกิดขึ้นที่ TCP เลเยอร์ และ end-to-end semantic อาจถูกบุกรุกถ้าโหนดระหว่างทางสร้าง ACK ขึ้นแทนที่ปลายทาง

2.3 TCP เฮดเดอร์



รูปที่ 2.4 TCP เฮดเดอร์

จากรูปที่ 2.4 สามารถอธิบายได้ดังต่อไปนี้

Source Number (16 บิต) : หมายถึง พอร์ตที่โฮสต์ต้นทางใช้ในการสื่อสารกันของเซสชั่นนี้ และ TCP/IP จะใช้พอร์ตนี้ไปตลอดตราบใดที่การสื่อสารในเซสชั่นนี้ยังไม่ยุติลง โดยทั่วไปพอร์ตนี้เรียกว่า "ไคลเอนต์พอร์ต" คือ พอร์ตที่ไคลเอนต์เปิดขึ้นมาเพื่อรอการตอบรับจากเซิร์ฟเวอร์ (พิจารณาจากทิศทางของแพ็กเก็ตที่ส่งมาจากไคลเอนต์ไปยังเซิร์ฟเวอร์) ไคลเอนต์พอร์ตจะมีหมายเลขไม่แน่นอนและเปลี่ยนไปทุกครั้งที่มีการเริ่มการเชื่อมต่อใหม่ เป็นพอร์ตที่ถูกเปิดไว้ในระยะเวลาสั้นๆ (ephemeral port)

ค่าที่เป็นไปได้ของพอร์ตนี้ขึ้นอยู่กับการจัดสรรของระบบปฏิบัติการ ในการกำหนดขอบเขตของพอร์ตเหล่านี้ส่วนใหญ่จะมีค่าอยู่ในช่วง 1024-5000

Destination Port Number (16 บิต) : หมายถึง หมายเลขพอร์ตบนโฮสต์ปลายทางที่โฮสต์ต้นทางต้องการติดต่อด้วย โดยนัยแล้วจะหมายถึงแอพพลิเคชั่นที่ให้บริการอยู่บนพอร์ตนั้นที่โฮสต์ปลายทางนั่นเอง พอร์ตนี้จะเรียกอีกอย่างหนึ่งว่า "เซิร์ฟเวอร์พอร์ต" หมายเลขพอร์ตที่เปิดไว้จะขึ้นอยู่กับแอพพลิเคชั่นที่ให้บริการ โดยทั่วไปแอพพลิเคชั่นแต่ละประเภทจะมีหมายเลขพอร์ตเป็นมาตรฐานสำหรับให้ไคลเอนต์ได้เรียกใช้บริการ

Sequence Number (32 บิต) : เป็นฟิลด์ที่ระบุถึงหมายเลขลำดับที่ใช้อ้างอิงในการสื่อสารข้อมูลแต่ละครั้ง เพื่อให้ทั้ง 2 ฝ่าย ได้รับทราบตรงกันว่าเป็นข้อมูลของชุดใด การนำไปใช้งานจะได้ไม่ปะปนกัน และมีลำดับที่ถูกต้อง เนื่องจากการสื่อสารข้อมูลผ่าน TCP นั้น จังหวะและลำดับเป็นส่วนสำคัญของโปรโตคอลไม่ยิ่งหย่อนไปกว่าข้อมูลใน TCP เฮดเดอร์ รวมไปถึงการที่ข้อมูลแต่ละ TCP เซ็กเมนต์ อาจจะถูกทำการแฟรกเมนต์ในเลเยอร์ของไอพีถัดลงไป ทำให้ข้อมูลถูกแบ่งออกและส่งไปในลำดับที่ไม่เรียงกัน หากไม่มีจุดอ้างอิงของข้อมูลก็จะไม่สามารถอ่านข้อมูลกลับมาใหม่ได้อย่างสมบูรณ์และถูกต้อง การส่งข้อมูลและการตอบรับจะใช้ฟิลด์นี้เป็นตัวยืนยันระหว่างกันเสมอ และจะถูกควบคุมโดย TCP เอ็นติตี้

Reserved (16 บิต) : ส่วนนี้จะถูกกำหนดให้เป็น 0 ตลอด ข้อมูลส่วนนี้เป็นการสงวนไว้ใช้ในอนาคตเมื่อมีการปรับปรุงโปรโตคอล

Acknowledge Number (32 บิต) : ทำหน้าที่เช่นเดียวกับ sequence number ต่างกันตรงที่เป็น sequence number ที่ใช้ในการตอบรับค่าที่อยู่ใน acknowledge number คือ หมายเลขที่ใช้ในการตอบรับ ค่าทั้งใน sequence number และ acknowledge number ต้องพิจารณาประกอบกับ flag จึงจะใช้ความสามารถของ TCP เซ็กเมนต์ ได้อย่างสมบูรณ์ จะถูกควบคุมโดย TCP เอ็นติตี้

Header Length หรือ data offset HLEN (4 บิต) : เป็นตัวเลขที่บอกขนาดของ TCP เฮดเดอร์ โดยปกติจะมีความยาว 20 บิต แต่ถ้ามีการใช้ค่าออพชั่นจะทำให้ขนาดของเฮดเดอร์ยาวขึ้นตามข้อมูลที่ต้องเพิ่มมาจากออพชั่นนั้น แต่จะไม่เกิน 60 ไบต์ และยังใช้ในการคำนวนหาจุดเริ่มต้นของข้อมูลจริง

Flag (6 บิต) : เป็นข้อมูลในระดับบิตที่ใช้เป็นตัวบอกคุณสมบัติของ TCP เซ็กเมนต์ ที่กำลังส่งและเป็นตัวควบคุมจังหวะการรับส่งข้อมูล มีทั้งหมด 6 บิต ดังรูปที่ 2.5

URG	ACK	PSH	RST	SYN	FIN

รูปที่ 2.5 ตัวควบคุมจังหวะการรับส่งข้อมูล (Control Field)

URG แสดงว่า ข้อมูลในฟิลด์ urgent pointer นั้นนำมาใช้งานได้ เมื่อถูกกำหนดค่าให้เป็น '1'

ACK แสดงว่า ข้อมูลในฟิลด์ acknowledge number นำมาใช้งานได้ เมื่อถูกกำหนดค่าให้เป็น '1'

PSH (Push) เพื่อแจ้งให้ผู้รับข้อมูลทราบว่าควรจะส่งข้อมูลเซ็กเมนต์นี้ไปยังแอพพลิเคชันโดยเร็ว

RST (Reset) ยกเลิกการติดต่อ ถ้าผู้รับพบว่าบิตนี้เป็น '1' แสดงว่าเกิดปัญหาในการเชื่อมต่อ

SYN (Synchronous) ใช้ในการเริ่มต้นขอติดต่อกับปลายทาง โดยปกติจะกำหนดค่าให้ SYN = 1 และ ACK = 0 ถ้าการเชื่อมต่อเรียบร้อย ผู้ตอบจะตอบกลับมาด้วย SYN = 1 and ACK = 1 ซึ่งเป็นค่าที่แทนความหมาย connection request and connection accepted

FIN (Finish) ใช้ส่งเพื่อแจ้งให้ปลายทางทราบว่ายกเลิกการติดต่อ

ใช้ flag เพื่อให้รับทราบว่าข้อมูลใดที่ใช้งานและข้อมูลใดที่ไม่ได้ใช้งาน เช่น acknowledge number จะไม่ได้ใช้งานในตอนเริ่มต้นของการขอเชื่อมต่อ นั่นหมายถึงฝ่ายผู้รับไม่ต้องนำข้อมูลในฟิลด์นี้ไปประมวลผลใดๆ และจากการที่ TCP เฮดเดอร์ มีขนาดคงที่ ที่ 20 ไบต์ ดังนั้น ถึงแม้ข้อมูลในบางฟิลด์จะใช้งานหรือไม่ก็ตาม ข้อมูลนั้นจะถูกส่งไปยังปลายทางเสมอ

Window Size (16 บิต) : เป็นขนาดของการรับ- ส่งข้อมูลในแต่ละครั้ง มีหน่วยเป็นไบต์ ที่ทางฝ่ายผู้รับจะสามารถรับได้ เนื่องจากในการรับข้อมูลทางผู้รับจะต้องเตรียมหน่วยความจำในการพักข้อมูลที่มาจาก TCP และทำการดีมัลติเพล็กซ์ออกมา หากไม่มีการตกลงถึงขนาดที่ทางฝ่ายรับสามารถรับได้ ก็จะทำให้การสื่อสารข้อมูลไม่สมดุล และฝ่ายรับอาจจะประมวลผลไม่ทัน ซึ่งจะส่งผลให้ต้องส่งข้อมูลซ้ำหลายครั้ง ค่า window size = 0 บอกให้ทราบว่า เซ็กเมนต์ทั้งหมดที่ส่งมานับจนถึงหมายเลขลำดับที่ "ACK number -1" นั้นผู้รับได้รับถูกต้องแต่ขอให้ผู้ส่งหยุดพักการส่งข้อมูลชั่วคราว เมื่อผู้รับพร้อมที่จะทำงานต่อไปก็จะส่ง เซ็กเมนต์ใหม่ที่มีค่า ACK number เท่าเดิมแต่กำหนดค่า window size เป็นค่ามากกว่า 0

Checksum (16 บิต) : ฟิลด์ที่ใช้ในการตรวจสอบความถูกต้องของข้อมูลใน TCP เซ็กเมนต์ จากรูปที่ 2.6 โครงสร้างข้อมูลส่วนหัวจำลอง (pseudo header) กระบวนการตรวจสอบเริ่มด้วยการกำหนดให้ค่าของทุกบิตใน checksum เป็น 0 ทั้งหมดและไม่นำส่วนของข้อมูลจริงมาพิจารณา ถ้าจำนวนไบต์รวมที่เหลืออยู่เป็นเลขจำนวนคี่ก็ให้เติมไบต์ 0 เข้าไป จากนั้นจึงบวกเลขทั้งหมดเข้าด้วยกัน จำนวนละ 16 บิต ด้วยวิธีการแบบ 1's complement แล้วนำมาคำนวณหา 1's complement ร่วมกับค่า checksum ที่ได้รับมา ถ้าผลลัพธ์ที่เกิดขึ้นมีค่าเป็น 0 แสดงว่าข้อมูลใน เซ็กเมนต์ นั้นถูกต้อง

<table>
<tr><td colspan="3">Source IP Address</td></tr>
<tr><td colspan="3">Destination IP Address</td></tr>
<tr><td>00000000</td><td>Protocol = 6</td><td>TCP Length</td></tr>
</table>

รูปที่ 2.6 ข้อมูลส่วนหัวจำลองที่มี TCP checksum อยู่ด้วย

Urgent Pointer : ใช้ระบุหมายเลข sequence number ของ TCP เซ็กเมนต์ ล่าสุดที่อยู่ในโหมด urgent

Option : เป็นฟิลด์ที่มีขนาดไม่แน่นอน เตรียมไว้สำหรับการให้บริการพิเศษที่นอกเหนือไปจากที่มีอยู่ มีใช้งานอยู่ 2 รูปแบบ คือ แบบ 1 ไบต์ ประกอบด้วย option type เพียงอย่างเดียว และ แบบจำนวนไบต์ไม่คงที่ประกอบด้วย option type, option length และ option value

End of Option List กำหนดโดยให้ option type = 0, option length = 1 และไม่ต้องกำหนดค่าให้ option value จะเป็นการบ่งบอกว่าจบรายการในออพชั่นแล้ว

No Operation กำหนดโดยให้ option type = 1, option length = 1 และ option value ไม่ต้องกำหนดค่าให้ ตัวเลือกนี้ใช้ในการเติมเต็มเพื่อปรับให้ออพชั่นถัดไปอยู่ในขอบเขตของ 32 บิต

Maximum Receive Segment Size : กำหนดโดยให้ option type = 2, option length = 4 และ option value กำหนดให้ด้วยค่าสูงที่สุดของขนาดเซ็กเมนต์ที่ได้รับ (receive segment size) ตัวเลือกนี้จะใช้โดยผู้ส่งสำหรับระบุขนาดเซ็กเมนต์ของ TCP ที่ใหญ่ที่สุด เพื่อเตรียมไว้ในการรับข้อมูล ค่านี้จะถูกกำหนดในตอนเริ่มต้นของการเชื่อมต่อ (เมื่อมีการกำหนด Flag SYN เป็น 1) เท่านั้น ในกรณีที่ไม่ได้กำหนดขนาดให้ตัวเลือกนี้ ค่าดีฟอลต์ของขนาดเซ็กเมนต์ที่ได้รับจะเป็น 536 ไบต์ซึ่งจะเป็นค่าดีฟอลต์ของขนาดดาต้าแกรมของไอพี (576 ไบต์) ลบด้วยขนาดของเฮดเดอร์ที่เล็กที่สุดในดาต้าแกรมของไอพี (20 ไบต์) และขนาดเฮดเดอร์ที่เล็กที่สุดของ TCP (20 ไบต์) ตัวเลือกนี้มีประโยชน์เนื่องจากทำให้ TCP สามารถส่งข้อมูลด้วยเซ็กเมนต์ขนาดใหญ่ที่สุดเท่าที่จะเป็นไปได้

Window Scale : กำหนดโดยให้ option type = 3, option length = 3 และ option value กำหนดด้วย window size ที่ผู้รับควรจะใช้ ตัวเลือกนี้มีประโยชน์มากสำหรับเครือข่าย WAN ซึ่งเป็นระบบที่มีการใช้เวลาในการทำงานมาก และต้องการ Throughput ที่สูง window size สูงสุดของ TCP ปกติจะเป็น 65,353 ไบต์ (ค่ามากที่สุดที่ตัวเลขขนาด 16 บิต จะสามารถอ้างอิงได้ที่ 2^{16})

ถ้าพิจารณาถึงการเชื่อมต่อกันแบบข้ามประเทศเวลาที่ใช้ในการเดินทางไปกลับของข้อมูลอาจจะเป็น 30 มิลลิวินาที ก็ถือว่าเป็นเรื่องปกติ ที่ความเร็ว 4 เมกะบิตต่อวินาที การจะส่งข้อมูลขนาดเท่ากับ window size จะต้องใช้เวลา 16.4 มิลลิวินาที ซึ่งการดำเนินการนี้ไม่เป็นการใช้ทรัพยากรอย่างมีประสิทธิภาพ และปัญหาจะยิ่งแย่ลงไปอีก หากว่าความเร็วในการรับ-ส่งเพิ่มมากขึ้น ตัวเลือกนี้จะเปิดโอกาสให้สามารถเพิ่ม window size ให้มากกว่า 65,535 ไบต์ โดยค่า n ของ window scale จะแสดง window size จริง มีขนาดเท่ากับ (window size) * (2^n) เนื่องจากการปรับขนาดจะเป็นกำลังสอง ซึ่งเป็นลักษณะเดียวกับการทำ bitwise shift ตัวเลือกนี้จึงสามารถเรียกได้อีกชื่อว่า window shift จะเห็นได้ชัดว่าตัวเลือกนี้ สามารถดำเนินการได้เฉพาะกรณีที่ทั้งสองด้านที่ทำการเชื่อมต่อกันในระดับ TCP ยินยอมทั้งสองฝั่ง

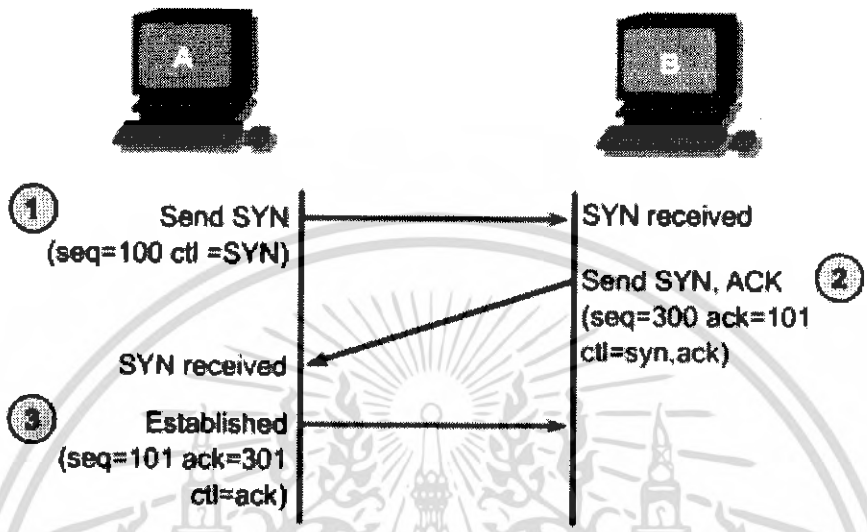
2.4 กลไกการติดต่อของ TCP

TCP เป็น โปรโตคอล แบบ connection-oriented จะต้องมีการสร้างเส้นทางเสมือน(virtual path) ระหว่างต้นทางและปลายทางก่อนที่จะทำการส่งข้อมูลกันได้

2.4.1 การสร้างการติดต่อ (Connection Establishment)

Three-Way Handshake

TCP จะส่งข้อมูลแบบสองทิศทาง คือสามารถรับ-ส่งข้อมูลได้พร้อมกัน การสร้างการติดต่อจะมี 3 ขั้นตอน หรือเรียกว่า three-way handshake ดังรูปที่ 2.7



รูปที่ 2.7 Three-Way Handshake

1. ไคลเอ็นต์จะส่ง SYN เซ็กเมนต์ ที่ประกอบด้วย หมายเลขพอร์ตของทั้งต้นทางและปลายทาง และหมายเลขลำดับเริ่มต้น ISN (initialization sequence number) เป็นหมายเลขไบต์แรกที่ไคลเอ็นต์จะส่งให้กับเซิร์ฟเวอร์

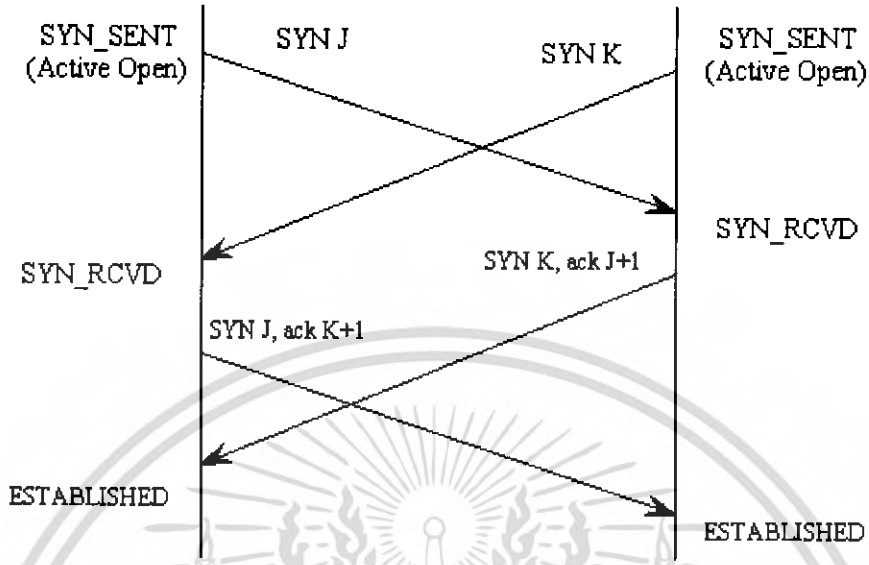
2. เซิร์ฟเวอร์จะส่ง SYN and ACK เซ็กเมนต์ มีจุดมุ่งหมายคือเป็นการตอบรับเซ็กเมนต์แรกที่ได้รับ โดยการใช้ flag ACK และหมายเลขตอบรับ ACK number หมายเลขนี้ได้มาจากการนำหมายเลขลำดับเริ่มต้น ISN ของไคลเอ็นต์บวกด้วย 1 ใช้เป็นเซ็กเมนต์เริ่มต้นของเซิร์ฟเวอร์ด้วยหมายเลขลำดับเริ่มต้นของเซิร์ฟเวอร์

3. ไคลเอ็นต์จะส่ง ACK เซ็กเมนต์ เพื่อเป็นการตอบรับ เซ็กเมนต์ที่สอง โดยการใช้ flag ACK พร้อมทั้งหมายเลขตอบรับ หมายเลขนี้ได้มาจากการนำหมายเลขลำดับเริ่มต้นของเซิร์ฟเวอร์บวกด้วย 1 และ SYN เซ็กเมนต์ที่ส่งไปจะเหมือนกับ SYN เซ็กเมนต์ตัวแรก ไคลเอ็นต์จะต้องกำหนด window size ที่เซิร์ฟเวอร์ โดยที่ SYN เซ็กเมนต์จะยอมให้นำพาข้อมูลจากไคลเอ็นต์ไปด้วย ถ้ามีการนำพาข้อมูลไปก็จะต้องใช้ SYN เซ็กเมนต์ใหม่แทน SYN เซ็กเมนต์เดิม โดยทั่วไปจะไม่นำพาข้อมูลไปด้วย

Simultaneous Open

ในกรณีที่โฮสต์ ทั้ง 2 พยายามสร้างการเชื่อมต่อระหว่างซ็อกเก็ตคู่เดียวกันจะทำให้เกิดลำดับขั้นตอนดังแสดงในรูปที่ 2.8 แต่จะมีการเชื่อมต่อขึ้นเพียงหนึ่งช่องทางเท่านั้น เนื่องจากการเชื่อมต่อในแต่ละช่องทางจะถูกกำหนดขึ้นโดยใช้หมายเลขซ็อกเก็ตผู้ส่งและผู้รับ ถ้าการเชื่อมต่อลำดับแรกสำเร็จก็จะ

ถูกบันทึกไว้ตารางการสื่อสาร เช่น (j, k) ถ้าการเชื่อมต่อลำดับที่สองสำเร็จในเวลาต่อมาข้อมูลจะถูกบันทึกไว้ที่เดียวกันคือ (j, k)



รูปที่ 2.8 Simultaneous Open

SYN Flooding Attack

ขั้นตอนการสร้างการเชื่อมต่อใน TCP จะมีความไวต่อปัญหาของระบบรักษาความปลอดภัยอย่างมากที่เรียกว่า SYN flooding attack ซึ่งจะเกิดขึ้นเมื่อมีผู้โจมตีส่ง SYN เซ็กเมนต์ จำนวนมากไปยังเซิร์ฟเวอร์ เพื่อหลอกว่า SYN เซ็กเมนต์ แต่ละอันนั้นมาจากไคลเอ็นต์ที่แตกต่างกัน โดยจะปลอมไอพีแอดเดรส ไคลเอ็นต์ในดาต้าแกรม เซิร์ฟเวอร์จะดำเนินการตามที่ไคลเอ็นต์สั่งให้เปิดการทำงาน แบ่งทรัพยากรที่สำคัญ เช่น การสร้าง TCP table และกำหนดไทม์เมอร์โดย TCP เซิร์ฟเวอร์นั้นจะส่ง SYN + ACK เซ็กเมนต์ ไปหลอกไคลเอ็นต์ว่ามีการสูญหายในระหว่างเวลานี้ อย่างไรก็ตามทรัพยากรจำนวนมากจะถูกครอบครองแต่ไม่มีการถูกใช้ ถ้าเป็นช่วงเวลาที่สั้นจำนวนของ SYN เซ็กเมนต์ จะมาก ในที่สุดเซิร์ฟเวอร์ก็จะไม่เหลือทรัพยากร และ SYN flooding attack นี้จะขึ้นอยู่กับกลุ่มของการโจมตีของการรักษาความปลอดภัยที่เรียกว่า denial of service attack โดยผู้โจมตีจะครอบครองระบบกับร้องขอการบริการของระบบ จะทำให้ระบบเกิดการล้มเหลวและปฏิเสธการบริการทุกการร้องขอ

การดำเนินการบางอย่างของ TCP จะมีแผนการในการบรรเทาผลของการโจมตีด้วย SYN เช่นมีการถูกกำหนดให้จำกัดการร้องขอการเชื่อมต่อในระหว่างที่มีเวลาจำกัด ในส่วนอื่นๆก็จะมีการกรองเอาดาต้าแกรมที่มาจากแอดเดรสของต้นทางที่ไม่ต้องการออกไป อีกหนึ่งแผนการอันใหม่จะเป็นการเลื่อนทรัพยากรออกไปจนกระทั่งมีการเชื่อมต่อที่สมบูรณ์โดยการใช้คุกกี้โดยเป็นวิธีของ STCP

Data Transfer

หลังจากที่การเชื่อมต่อถูกสร้างขึ้นแล้วนั้นก็จะมีการถ่ายโอนข้อมูลทั้งสองทิศทาง ไคลเอ็นต์และเซิร์ฟเวอร์สามารถส่งข้อมูลและ ACK ไปในทั้งสองทิศทางได้ซึ่งข้อมูลนั้นจะเดินทางไปในทิศทางเดียวกับ ACK ซึ่งจะถูกจัดการโดยเซ็กเมนต์เดิม ACK จะถูกนำพาไปกับข้อมูล

เซ็กเมนต์ที่เป็นข้อมูลจะส่งโดยไคลเอ็นต์ที่มีการกำหนด flag PSH (push) ดังนั้น TCP เซิร์ฟเวอร์จะรู้ถึงผู้ที่ส่งมายังเซิร์ฟเวอร์เพื่อทำการประมวลผลซึ่งในไม่ช้า TCP เซิร์ฟเวอร์ก็จะทำการรับเซ็กเมนต์จากเซิร์ฟเวอร์บนการจัดการอื่นๆนั้น จะไม่มีมีการกำหนด push flag การดำเนินการของ TCP ส่วนใหญ่จะมี ออพชั่นในการกำหนดหรือไม่กำหนด flag นี้

Pushing Data

เราจะเห็นได้ว่าการส่งของ TCP นั้นใช้บัฟเฟอร์เก็บข้อมูลที่เป็นสตรีมที่มาจากแอพพลิเคชั่นของโปรแกรมที่ส่งมาไว้ในหน่วยความจำ

การส่งของ TCP นั้นสามารถเลือกขนาดของเซ็กเมนต์ได้โดย TCP ทางผู้รับจะใช้บัฟเฟอร์ เช่น เมื่อข้อมูลมาถึงและมีการส่งข้อมูลต่อไปยังแอพพลิเคชั่นของโปรแกรม เมื่อแอพพลิเคชั่นของโปรแกรมพร้อมหรือสะดวกที่จะรับนี้เป็นประเภทของการเพิ่มความยืดหยุ่นที่มีประสิทธิภาพของ TCP

อย่างไรก็ตามเหตุการณ์นี้จะเกิดขึ้นเมื่อแอพพลิเคชั่นของโปรแกรมไม่จำเป็นต้องมีความยืดหยุ่น เช่น พิจารณาแอพพลิเคชั่นของโปรแกรมในการติดต่อสื่อสารกันกับอีกแอพพลิเคชั่นของโปรแกรมตามด้วยอื่นๆ อีกจนถึงปลายทางแอพพลิเคชั่นของโปรแกรมที่จุดหนึ่งต้องการส่ง keystroke ไปยังแอพพลิเคชั่นที่จุดอื่นๆ และรับผลตอบสนองโดยตรง การส่งจะถูกดีเลย์และการส่งต่อข้อมูลจะถูกดีเลย์อาจจะไม่เป็นการยอมรับของแอพพลิเคชั่นของโปรแกรม

TCP จะจัดการกับแต่ละสถานการณ์ได้โดย แอพพลิเคชั่นของโปรแกรมชุดที่ส่งจะสามารถร้องขอการดำเนินการ " push " ซึ่งหมายความว่าในการส่งของ TCP ต้องไม่มีการรอคอยวินโดว์มาให้เต็มก่อนมันจะต้องสร้างเซ็กเมนต์ และการส่งไปทันที การส่งของ TCP ต้องมีการกำหนด bit push ด้วยเหมือนกันเพื่อให้ TCP ที่ฝั่งรับนั้นรู้ว่าเซ็กเมนต์ นั้นมีการเพิ่มข้อมูลมา จำเป็นต้องถูกส่งไปยังแอพพลิเคชั่นของโปรแกรมและไม่มีการรอคอยข้อมูลที่เข้ามามากไปกว่านี้

แม้ว่าการดำเนินการ push สามารถถูกร้องขอโดยแอพพลิเคชั่นของโปรแกรมนั้นส่วนมาก การดำเนินการที่นิยมจะไม่สนใจในแต่ละการร้องขอ ซึ่ง TCP นั้นสามารถเลือกที่จะใช้รูปแบบนี้ได้หรือไม่ใช้รูปแบบนี้ก็ได้

Urgent Data

TCP เป็นโปรโตคอลแบบ stream - oriented ซึ่งหมายความว่าข้อมูลนั้นถูกนำมาจากชั้นโปรโตคอลประยุกต์ไปยัง TCP ขณะที่มีการส่งข้อมูลเป็นไบต์อย่างต่อเนื่อง แต่ละประเภทของข้อมูลจะมีตำแหน่งในสตรีมอย่างไรก็ตามเหตุการณ์ที่เกิดขึ้นนั้นแอพพลิเคชั่นจำเป็นต้องส่ง bytes urgent ซึ่งเป็นการส่งที่แอพพลิเคชั่นต้องการส่วนของข้อมูลเพื่อการอ่านเนื่องจากเป็นคำสั่งของ แอพพลิเคชั่นทางฝั่งรับ เมื่อผลของการดำเนินการกลับมานั้น แอพพลิเคชั่นฝั่งส่งจะหาข้อผิดพลาดทุกๆ อย่างที่จะทำให้การดำเนินการนั้นล้มเหลว แต่มันจะมีการส่งข้อมูลไปแล้วเป็นจำนวนมาก ถ้ามันปล่อยให้การควบคุมล้มเหลว

(control + c) ตัวอักษร 2 ตัวนี้จะถูกเก็บไว้ในหน่วยความจำที่ปลายทางของบัฟเฟอร์ทางฝั่งรับของ TCP มันจะถูกส่งไปยัง แอพพลิเคชั่นทางฝั่งรับหลังจากข้อมูลทั้งหมดถูกดำเนินการแล้ว

ผลของการส่งเซ็กเมนต์กับการกำหนดค่าบิต URG ของแอพพลิเคชั่นที่ทำการส่งจะแจ้ง TCP ที่ทำการส่งว่าส่วนของข้อมูลนั้นเป็น urgent TCP ที่ทำการส่งจะสร้างเซ็กเมนต์และแทรกข้อมูล urgent ที่เป็นจุดเริ่มต้นของเซ็กเมนต์ส่วนที่เหลือซึ่งสามารถบรรจุข้อมูลปกติจากบัฟเฟอร์

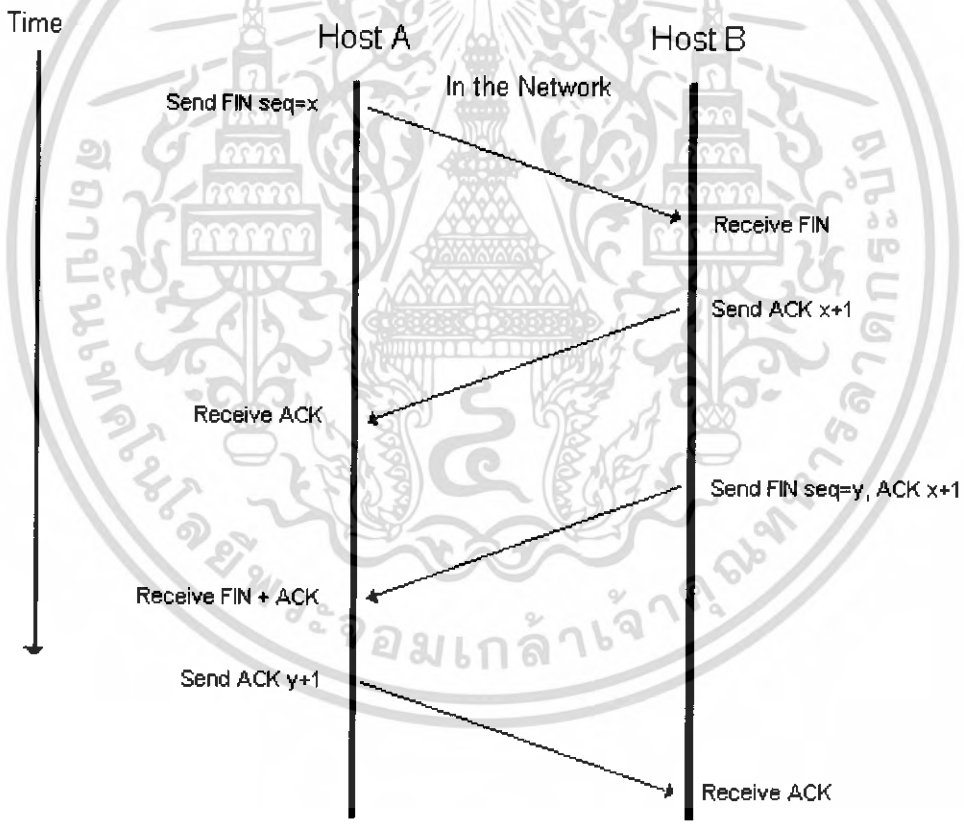
ฟิลด์ urgent pointer ในเฮดเดอร์จะกำหนดจุดสิ้นสุดของข้อมูล urgent และจุดเริ่มต้นของข้อมูลปกติ

เมื่อ TCP ฝั่งรับนั้นรับเซ็กเมนต์กับกำหนด bit URG ที่ได้มาจากข้อมูล urgent ของ เซ็กเมนต์จะใช้ค่าของ urgent pointer และทำการส่งมันไปยังแอพพลิเคชั่นทางฝั่งรับ

2.4.2 การยกเลิกการติดต่อ (Connection Termination)

Three or Four – Way Handshaking

การยกเลิกการติดต่อระหว่างไคลเอ็นต์กับเซิร์ฟเวอร์มี 4 ขั้นตอน ดังรูปที่ 2.9



รูปที่ 2.9 การยกเลิกการติดต่อ

1. ไคลเอ็นต์ส่ง FIN เซ็กเมนต์ (ส่ง EOF ให้แอพพลิเคชั่น)

Flag ของ ACK ก็แอคทีฟด้วย แต่ในกรณีนี้หมายเลขใน ACK จะไม่นำมาพิจารณาเกี่ยวข้องกับกระบวนการนี้ด้วย จากข้อกำหนด RFC 793 สำหรับการยุติการติดต่อหรือการปิดการเชื่อมต่อนั้นไม่ได้

กล่าวถึง flag ของ ACK ในเซ็กเมนต์แรกว่าจะต้องมีด้วยหรือไม่ แต่ในทางปฏิบัติสำหรับ FIN เซ็กเมนต์นั้นจะต้องมี flag ที่เซตอยู่ทั้ง FIN and ACK คู่กันเสมอ เซ็กเมนต์ที่มี flag FIN อย่างเดียวจะไม่มี หากพูดถึง FIN เซ็กเมนต์ก็ต้องมี FIN ACK เซตอยู่

2. เซิร์ฟเวอร์ส่ง ACK เซ็กเมนต์เพื่อเป็นการยืนยันว่าได้รับ FIN เซ็กเมนต์จากไคลเอ็นต์แล้ว จะสังเกตเห็นได้ว่า acknowledgment number จะได้มาจากการนำ sequence number ซึ่งอยู่ใน FIN เซ็กเมนต์บวกด้วย 1 (แอพพลิเคชั่นหยุดทำงาน)

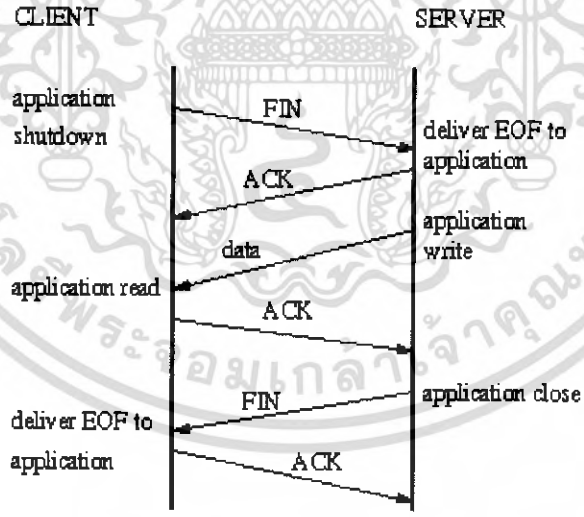
3. ในตอนนี้เซิร์ฟเวอร์ยังสามารถส่งข้อมูลที่ต้องการให้กับไคลเอ็นต์ได้ เมื่อเซิร์ฟเวอร์ไม่มีข้อมูลใดๆ ที่จะส่งให้กับไคลเอ็นต์แล้ว เซิร์ฟเวอร์จะส่ง FIN เซ็กเมนต์

4. ไคลเอ็นต์จะส่ง ACK เซ็กเมนต์เพื่อยืนยันว่าได้รับ FIN เซ็กเมนต์จากเซิร์ฟเวอร์แล้ว

เพื่อหลีกเลี่ยงปัญหา two – army problem ดังรูปที่ 2.10 จึงกำหนดให้มีการจับเวลาในระหว่างการขอยกเลิกช่องสัญญาณ ถ้าฝ่ายหนึ่งได้ส่งเซ็กเมนต์ขอยกเลิกการใช้ช่องสัญญาณไปแล้ว และถ้าไม่มีการตอบรับภายในระยะเวลา 2 เท่าของ packet lifetime ก็ให้ถือว่าช่องสัญญาณนั้นถูกยกเลิกโดยอัตโนมัติ

TCP Half-Close

การยุติการเชื่อมต่อโดยที่โฮสต์ส่ง FIN ACK ออกไป ไม่ได้หมายความว่าการสื่อสารจะยุติลงในทันที เป็นเพียงการแจ้งให้อีกฝ่ายหนึ่งทราบว่าจะไม่มีข้อมูลส่งไปอีก แต่โฮสต์จะยังคงสามารถรับข้อมูลเข้ามาได้อีกอยู่ต่อไป จนกว่าโฮสต์อีกฝั่งหนึ่งจะส่ง FIN ACK เพื่อขอยุติการส่งข้อมูลเช่นกัน เมื่อไม่มีฝ่ายใดต้องการจะส่งข้อมูลกัน นั่นคือการเชื่อมต่อได้ยุติลงอย่างสมบูรณ์



รูปที่ 2.10 TCP Half-Close

ในระหว่างที่โฮสต์ส่ง FIN ACK ออกไปและยังไม่ได้รับ FIN ACK กลับมานั้น เรียกว่า half-close คือ การปิดเพียงด้านส่งข้อมูลเพียงด้านเดียวแต่ยังคงเปิดด้านรับไว้รับข้อมูล ซึ่งคุณสมบัตินี้แอพพลิเคชั่นจะสามารถนำไปใช้ให้เห็นประโยชน์ได้ขึ้นอยู่กับลักษณะของการใช้งาน เช่น การสื่อสารข้อมูลบางประเภทโฮสต์ที่ทำหน้าที่รับข้อมูลอาจไม่มีความจำเป็นต้องโต้ตอบกับโฮสต์ที่ส่งข้อมูลในระดับ

ชั้นโปรโตคอลประยุกต์ (การโต้ตอบในระดับชั้นขนส่งข้อมูลยังคงมีอยู่โดยการจัดการของ TCP/IP) ดังนั้น จึงอาจจะสามารถทำงานได้อย่างต่อเนื่องแม้ว่า TCP จะอยู่ในสถานะ half-close ก็ตาม

การ Reset การติดต่อ

การ reset ในที่นี้ หมายถึง มีการขอยกเลิกการติดต่อกะทันหัน ซึ่งเหตุการณ์แบบนี้จะเกิดขึ้นใน 3 กรณี คือ

1. Denying a Connection : เมื่อ TCP ของต้นทางไม่สามารถติดต่อไปยังพอร์ตที่ต้องการของปลายทางได้ TCP ของปลายทางจะส่งเซ็กเมนต์ที่มีค่า flag RST เท่ากับ 1 มาให้กับต้นทางเพื่อยกเลิกการขอการติดต่อครั้งนี้

2. Aborting Connection : เมื่อเกิดเหตุการณ์ที่ผิดปกติบางอย่าง TCP สามารถส่ง RTS เซ็กเมนต์เพื่อขอหยุดการติดต่อได้

3. Terminating an Idle Connection : เมื่อ TCP ตรวจสอบได้ว่าไม่มีการตอบสนองจาก TCP ของอีกฝ่ายหนึ่งแล้ว ก็จะส่ง RTS เซ็กเมนต์เพื่อยกเลิกการติดต่อได้

2.5 สถานะของการเชื่อมต่อ

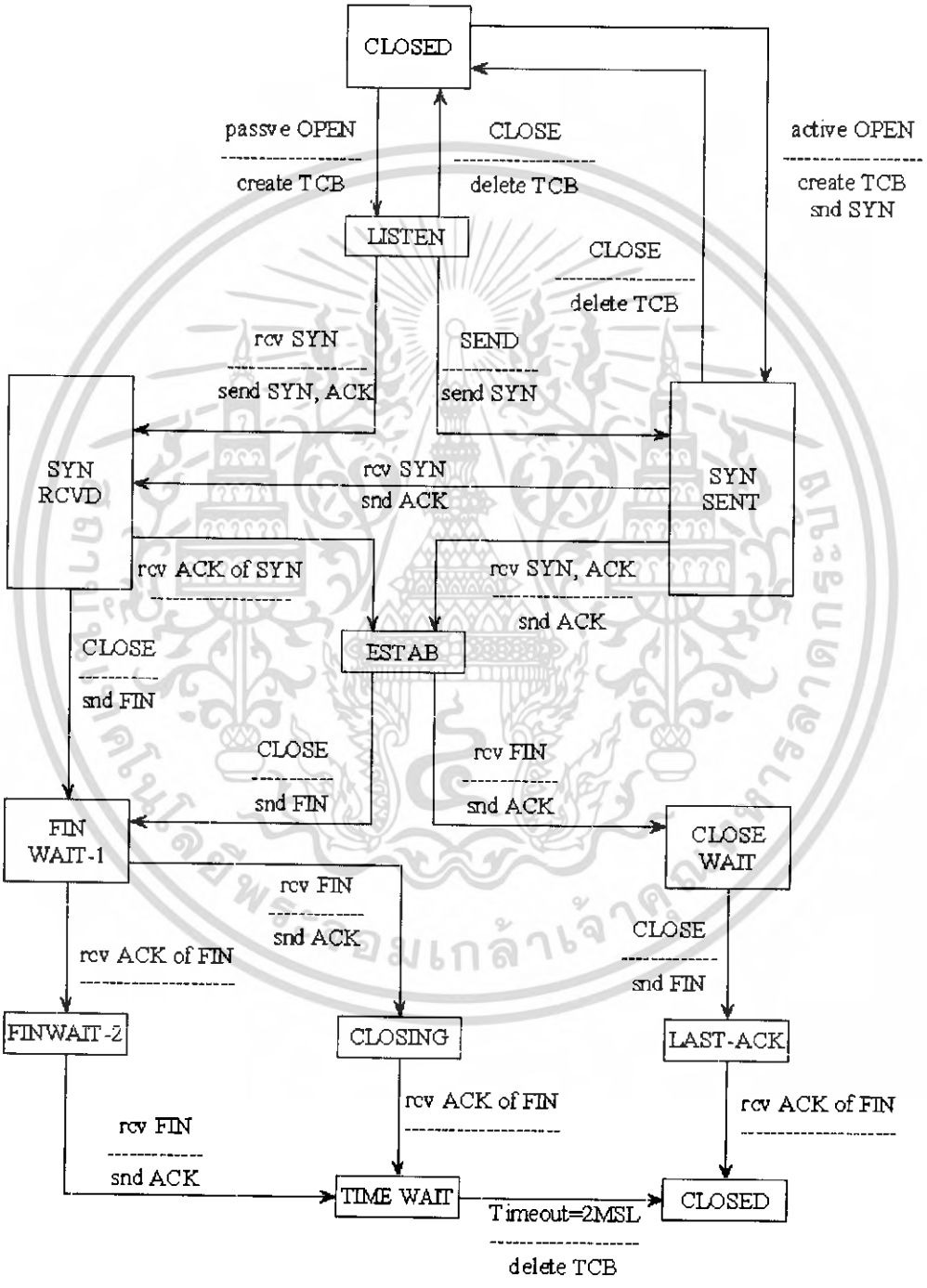
ในการพัฒนาโปรแกรม TCP นั้นจะต้องมีการเก็บเหตุการณ์ต่างๆ เอาไว้ในระหว่างขั้นตอนของการสร้างการติดต่อ การยกเลิกการติดต่อ และการส่งข้อมูล ดังนั้นในการพัฒนาจึงได้ใช้หลักการของ finite state machine ซึ่งสามารถที่จะกำหนดจำนวนของสถานะต่างๆ ได้ ซึ่ง ณ เวลาใดเวลาหนึ่ง machine จะมีเพียงสถานะเดียวเท่านั้น แต่ถ้ามีเหตุการณ์ใดเปลี่ยนไป machine จะสามารถเปลี่ยนสถานะได้ หรือเหตุการณ์นั้นๆ จะทำให้ machine มีการกระทำบางอย่างได้ หรือเหตุการณ์ต่างๆ จะเป็นอินพุตซึ่งจะส่งผลให้สถานะเปลี่ยนไปและสามารถที่จะสร้างเอาต์พุตออกมาได้ ดังตารางที่ 2.2

State	Description
CLOSED	There is no connection.
LISTEN	The sever is waiting for calls from the client.
SYN-SENT	A connection request is sent; waiting for acknowledgment
SYN-RCVD	A connection request is received.
ESTABLISHED	Connection is established.
FIN-WAIT-1	The application has requested the closing of the connection.
FIN-WAIT-2	The other side has accepted the closing of the connection.
TIME-WAIT	Waiting for retransmitted segments to die.
CLOSE-WAIT	The sever is waiting for the application to close.
LAST-ACK	The sever is waiting for the last acknowledgment.

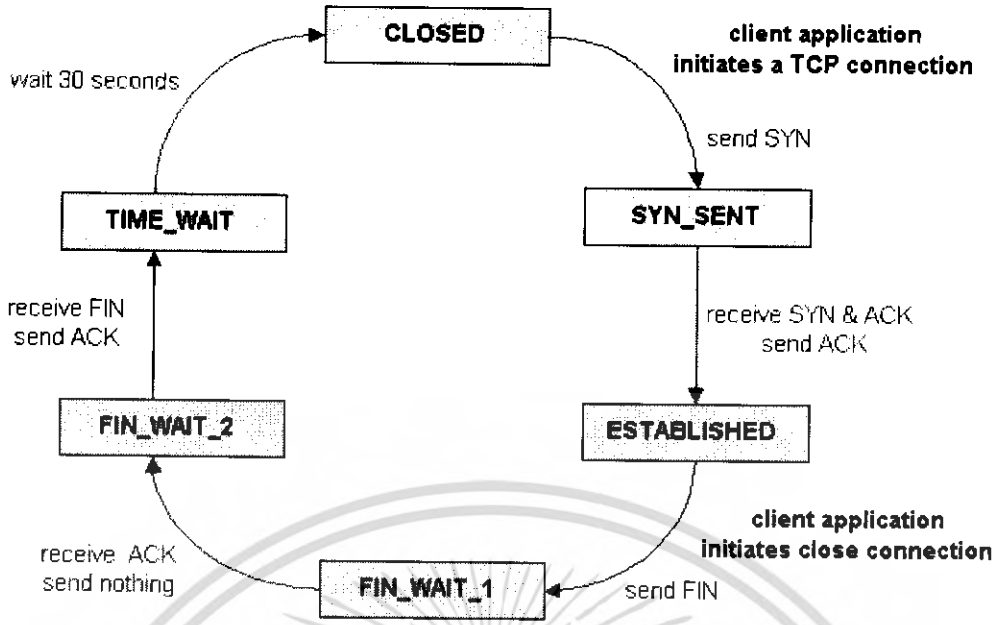
ตารางที่ 2.2 สถานะของการเชื่อมต่อ TCP

การเชื่อมต่อเริ่มต้นจากสถานะ CLOSED เมื่อเรียกใช้บริการ LISTEN หรือ CONNECT ก็จะเปลี่ยนสถานะไปจากเดิม ถ้าอีกฝ่ายหนึ่งเรียกใช้บริการตรงกันข้ามการเชื่อมต่อก็จะเกิดขึ้นและจะย้ายไปอยู่ในสถานะ ESTABLISHD เมื่อยกเลิกการติดต่อก็จะกลับไปอยู่ในสถานะ CLOSED อย่างเดิม

จากหลักการที่ได้กล่าวมานั้นเราสามารถนำมาเขียนเป็น state transition diagram ได้ ดังรูปที่ 2.11 ในการติดต่อสื่อสารโดยทั่วไปจะเกิดขึ้นระหว่างผู้ใช้ติดต่อกับผู้ให้บริการ



รูปที่ 2.11 State Transition Diagram

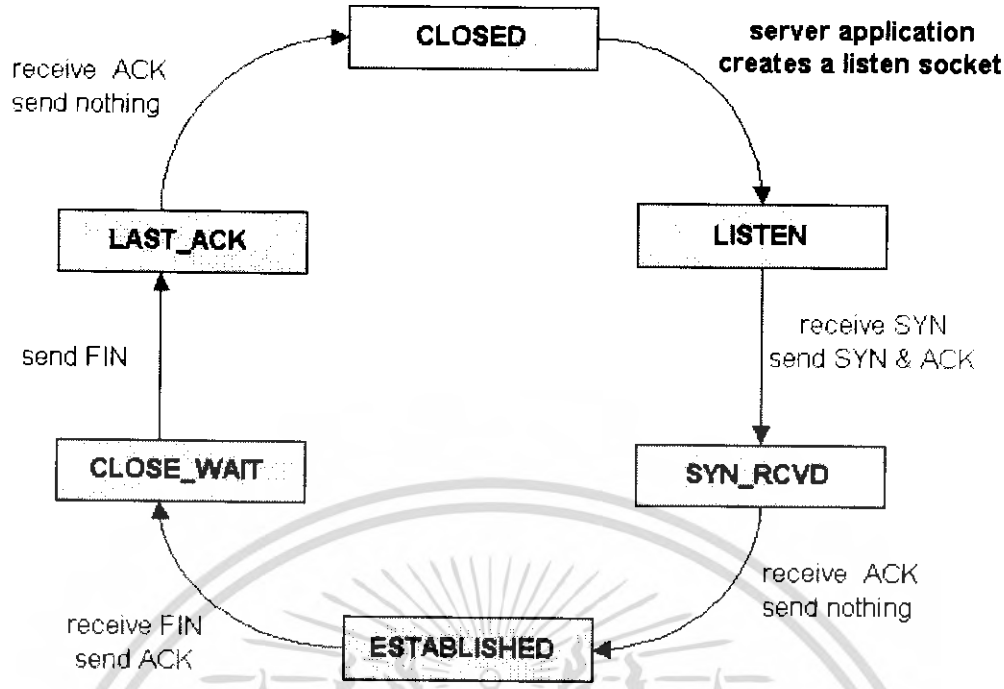


รูปที่ 2.12 Client Diagram

จากรูปที่ 2.12 กระบวนการทำงานเริ่มต้นจากเส้นทางของผู้ใช้บริการ (ผู้ส่ง) เมื่อผู้ใช้บริการส่งคำสั่ง CONNECT มาถึง เอ็นติตี้ TCP ทางฝั่งผู้ให้บริการ (ผู้รับ) จะสร้าง connection record สำหรับการเชื่อมต่อขึ้นมารองรับมีสถานะเป็น " SYN SENT " แล้วส่ง SYN เซ็กเมนต์ ตอบกลับไป ในขณะเดียวกันทางผู้ให้บริการอาจจะเริ่มการติดต่อกับโฮสต์ อื่นๆ อยู่ด้วย ดังนั้น connection record สร้างขึ้นเพื่อการเชื่อมต่อแต่ละช่องทางโดยเฉพาะ เมื่อ SYN+ACK เซ็กเมนต์ มาถึง ผู้รับจะส่ง ACK เซ็กเมนต์ กลับไปอีกครั้ง เมื่อเสร็จ 3 ขั้นตอนแล้ว ช่องการสื่อสารนั้นจะเป็น "established"

เมื่อผู้ใช้ต้องการยกเลิกการเชื่อมต่อก็จะส่ง FIN มายังผู้ให้บริการ โดยมีสถานะเป็น "FIN WAIT 1" เมื่อการตอบรับมาถึงสถานะของผู้ใช้จะเปลี่ยนเป็น "FIN WAIT 2" ช่องทางการสื่อสารถูกปิดลง เมื่อผู้ให้บริการยกเลิกการเชื่อมต่อบ้าง ก็จะเกิดกระบวนการแบบเดียวกัน ขณะที่ช่องทางการสื่อสารถูกปิดแล้วแต่ เอ็นติตี้ TCP ของทั้งคู่ยังมีสถานะ "LAST ACK" เมื่อสิ้นสุดระยะเวลารอคอยนี้ การเชื่อมต่อก็จะถูกยกเลิกและ connection record ของคู่นี้จะถูกลบทิ้ง

ถ้าเริ่มต้นจากผู้ให้บริการ โดยปกติจะใช้บริการ LISTEN และหยุดการทำงานเพื่อรอคอย การเรียกจากผู้ใช้บริการ เมื่อ SYN เซ็กเมนต์มาถึง ผู้ให้บริการจะตอบกลับด้วย "SYN RCVD" การเชื่อมต่อจะสมบูรณ์ หลังจากผู้ให้บริการได้รับแพ็กเก็ตตอบรับกลับมาอีกครั้ง และจะมีสถานะ "established" ดังรูปที่ 2.13



รูปที่ 2.13 Server Diagram

2.6 กลไกควบคุมการไหล (Flow Control)

บางเครือข่ายได้พยายามที่จะนำการควบคุมการไหลของข้อมูลมาใช้ เพื่อควบคุมความคับคั่ง (congestion) ถึงแม้ว่าการควบคุมการไหลสามารถนำมาใช้ที่ชั้นขนส่งข้อมูลเพื่อควบคุมไม่ให้โฮสต์ตัวหนึ่งส่งข้อมูลไปให้โฮสต์อีกตัวหนึ่ง จนข้อมูลล้นได้ และสามารถนำมาใช้เพื่อป้องกันไม่ให้ IMP ตัวหนึ่งส่งข้อมูลไปให้แก่ IMP ที่อยู่ข้างเคียงจนมากเกินไปเช่นกัน แต่เป็นการยากที่จะควบคุมปริมาณการสื่อสารทั้งหมดในเครือข่ายจากจุดสุดท้ายถึงจุดสุดท้าย แต่อาจจะกล่าวได้ว่าถ้าหากโฮสต์ได้ถูกบังคับให้หยุดส่งข้อมูลเพื่อควบคุมปริมาณการไหลแล้ว จะทำให้ subnet ไม่ทำงานมากเกินไปจนทำไม่ทันได้

สาเหตุที่การควบคุมการไหลไม่สามารถแก้ปัญหาของความแออัดได้เนื่องจากปกติการสื่อสารจะเป็นลักษณะเบิสท์ เวลาส่วนใหญ่ของผู้ใช้งานที่ทำงานแบบโต้ตอบทันที ก็คือนั่งอยู่ที่หน้าเครื่องคอมพิวเตอร์เคาะคีย์บอร์ดแล้วหยุดคิดบางอย่างไปชั่วระยะแล้วจึงกดคีย์ใหม่ แต่ถ้าหากว่าผู้ใช้งานต้องการตรวจดูหรือค้นหาบางอย่างในไฟล์ขนาดใหญ่ปริมาณการสื่อสารจะสูงกว่าปกติที่ใช้อยู่มาก ดังนั้นการควบคุมการไหลใดก็ตามที่ได้ปรับมาเพื่อให้ใช้กับค่าเฉลี่ยของอัตราการส่งขนาดนี้ จะไม่สามารถจัดการกับข้อมูลแบบเบิสท์ได้ดี ในทำนองเดียวกันถ้าหากการควบคุมการไหลที่ปรับมาให้ใช้ได้ดีกับเบิสท์ก็จะไม่สามารถจัดการกับค่าสูงสุดเมื่อมีผู้ใช้งานหลายๆคนส่งข้อมูลแบบเบิสท์พร้อมๆกันได้ (หากคนที่มีโทรศัพท์อยู่ครึ่งหนึ่งของทั้งหมดยกหูโทรศัพท์พร้อมทั้งใช้งานพร้อมกัน จะมีคนจำนวนมากที่ได้รับสัญญาณไม่ว่าง ระบบโทรศัพท์ได้ถูกออกแบบมาให้สามารถให้บริการได้เท่ากับค่าเฉลี่ย ไม่ใช่ออกแบบมาให้ใช้ได้กับกรณีที่แย่ที่สุด)

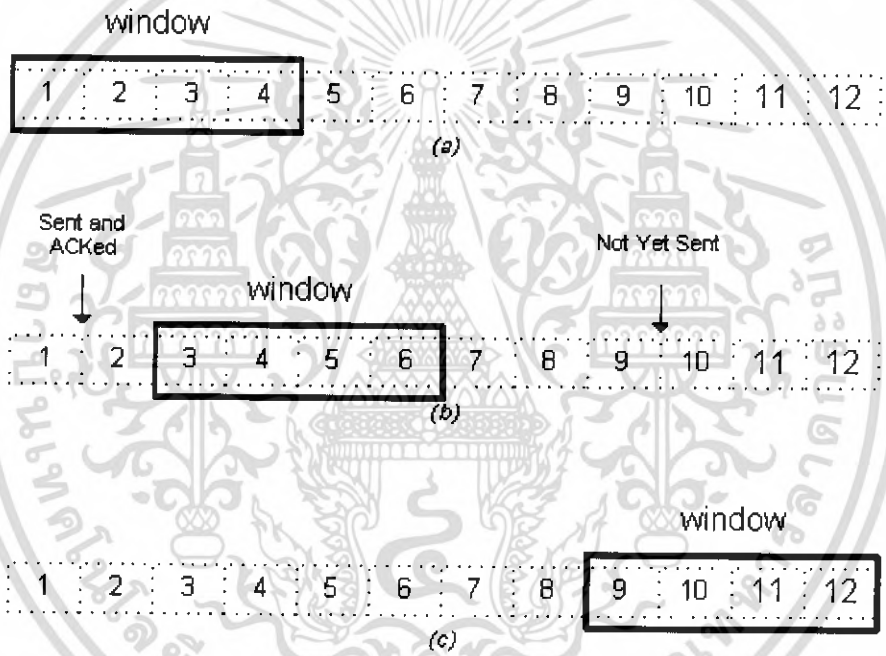
เมื่อมีความพยายามที่จะใช้การควบคุมการไหลเพื่อควบคุมความแออัดจะสามารถใช้ได้กับการสื่อสารดังต่อไปนี้

1. ในโปรเซสของผู้ใช้งาน หรือ ข้อความ (message) ที่เหลืออยู่เพียงหนึ่งต่อวงจรเสมือน
2. ในโฮสต์โดยที่ไม่ขึ้นอยู่กับวงจรเสมือนที่เปิดอยู่
3. ระหว่าง IMP ที่ต้นทางและปลายทางโดยไม่สนใจโฮสต์

นอกจากนี้จำนวนของวงจรเสมือนที่เปิดอาจจะจำกัดได้

2.6.1 Sliding Window Protocol

โปรโตคอลนี้จะกำหนดวินโดว์ซึ่งเป็นจำนวนแพ็กเก็ตที่ผู้ส่งสามารถส่งได้พร้อมกันในเวลาเดียวกัน จำนวนแพ็กเก็ตนี้เป็นจำนวนสูงสุดที่ฝั่งส่งสามารถส่งให้ฝั่งรับได้โดยไม่ต้องรอให้ฝั่งรับตอบ ACK กลับมา ดังรูปที่ 2.14 แสดงตัวอย่างของ sliding window protocol



รูปที่ 2.14 Sliding Window Protocol

จากรูปที่ 2.14 ขนาดของ sliding window ถูกกำหนดให้เป็น 4 ในส่วนบนของรูปที่ 2.14 แสดงให้เห็นว่าฝั่งส่งสามารถส่งแพ็กเก็ตข้อมูลที่ 1-4 ออกไปได้ ถ้าฝั่งรับทำการตอบแพ็กเก็ตที่ 1 and 2 กลับมา วินโดว์ก็จะขยับไปเป็นแพ็กเก็ตที่ 3-6 โดยที่ sending window และ receiving window ไม่จำเป็นต้องมีขอบเขตสูงสุดและต่ำสุดรวมถึงขนาดเท่ากัน

Sliding Window Protocol เป็นการใช้งานเครือข่ายอย่างมีประสิทธิภาพ เครือข่ายส่วนใหญ่เป็นแบบสองทิศทางและ window size ในโปรโตคอล TCP เปลี่ยนแปลงตลอดเวลาตามเงื่อนไขของระบบเครือข่าย

Sliding Window Protocol สามารถแก้ปัญหา dead lock ของฝั่งส่ง ถ้าหากว่ามีการตั้งเวลาน้อยเกินไป และยังสามารถแก้ปัญหาเรื่องการ synchronization เมื่อต้องพบกับเฟรมที่เป็นขยะและเฟรมที่หายไประหว่างการส่งได้อีกด้วย

การทำงานของ Sliding Window Protocol มีขั้นตอนดังนี้

1. เมื่อโฮสต์ ต้องการที่จะส่งข้อมูล TCP จะย้ายข้อมูลไปเก็บไว้ในบัฟเฟอร์ที่จะใช้ส่งข้อมูล ซึ่งข้อมูลส่วนนี้เรียกว่าเซ็กเมนต์และในเซ็กเมนต์อาจจะแบ่งเป็นหลายแพ็กเก็ต และในแต่ละแพ็กเก็ตจะถูกกำหนดหมายเลขลำดับไว้

2. ทุกๆ แพ็กเก็ตในเซ็กเมนต์จะถูกส่งไปให้โปรโตคอลไอพี เพื่อทำการส่งไปยังโฮสต์ปลายทาง

3. เซ็กเมนต์ข้อมูลจะยังคงถูกเก็บไว้ในบัฟเฟอร์จนกว่าจะได้รับการตอบรับจากโฮสต์ฝั่งรับก่อน และโฮสต์ฝั่งส่งจะตั้งเวลาเพื่อรอการตอบกลับ ถ้าโฮสต์ฝั่งรับไม่ตอบกลับภายในเวลาที่กำหนด ข้อมูลที่อยู่ในบัฟเฟอร์จะถูกส่งใหม่อีกครั้ง

4. เมื่อแพ็กเก็ตมาถึงฝั่งรับโฮสต์ฝั่งรับจะใช้หมายเลขลำดับในการเรียบเรียงแพ็กเก็ตให้ได้เซ็กเมนต์เหมือนเดิม

5. เมื่อโฮสต์ฝั่งรับได้รับแพ็กเก็ตครบและไม่มีการผิดพลาดใดๆ ก็จะส่งแพ็กเก็ตตอบกลับไปยังโฮสต์ฝั่งส่งว่าได้รับข้อมูลครบแล้ว

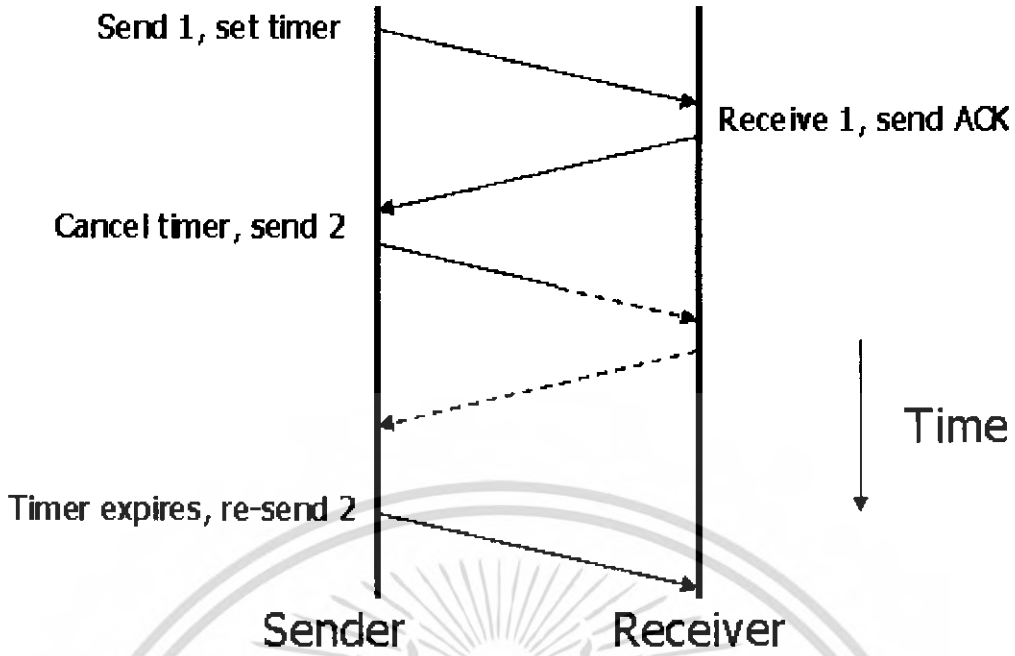
6. เมื่อโฮสต์ฝั่งส่งได้รับการตอบกลับมาเซ็กเมนต์ที่อยู่ในเซ็กเมนต์จะถูกลบทิ้งแล้วทำการส่งเซ็กเมนต์ถัดไป จนกว่าข้อมูลจะถูกส่งทั้งหมด

กระบวนการส่งข้อมูลแบบนี้จะทำให้มั่นใจว่าข้อมูลจะส่งถึงปลายทางอย่างแน่นอนและถูกต้อง ซึ่งการให้บริการนี้เรียกว่า connection-oriented

Sliding Window Protocol จะแบ่งได้ 3 ประเภท

1. A One Bit Sliding Window Protocol

มีขนาดเท่ากับ 1 หรือมีลักษณะ stop and wait ซึ่งฝั่งส่งจะต้องรอการตอบรับจากฝั่งรับก่อนที่จะส่งเฟรมต่อไป การตอบรับจะประกอบไปด้วยหมายเลขของเฟรมที่ได้รับเข้ามาโดยไม่มีความผิดพลาด เมื่อเฟรมนั้นกลับไปถึงฝั่งส่ง ฝั่งส่งจะดูว่าหมายเลขนี้ตรงกับที่คาดไว้หรือไม่ ถ้าหากว่าไม่ใช่ก็จะทำการส่งเฟรมมาซ้ำอีกครั้ง ดังรูปที่ 2.15



รูปที่ 2.15 Stop and Wait

2. A Protocol Using Go Back n

จากโปรโตคอลที่ผ่านๆ มาได้มีการละเลยเวลาที่ใช้ในการส่งเฟรมรวมทั้งเวลาในการตอบรับจากฝั่งส่งไปยังฝั่งรับ แต่ในความเป็นจริงแล้วเวลาเหล่านี้มีผลเป็นอย่างมากต่อประสิทธิภาพของการใช้ช่องสัญญาณ

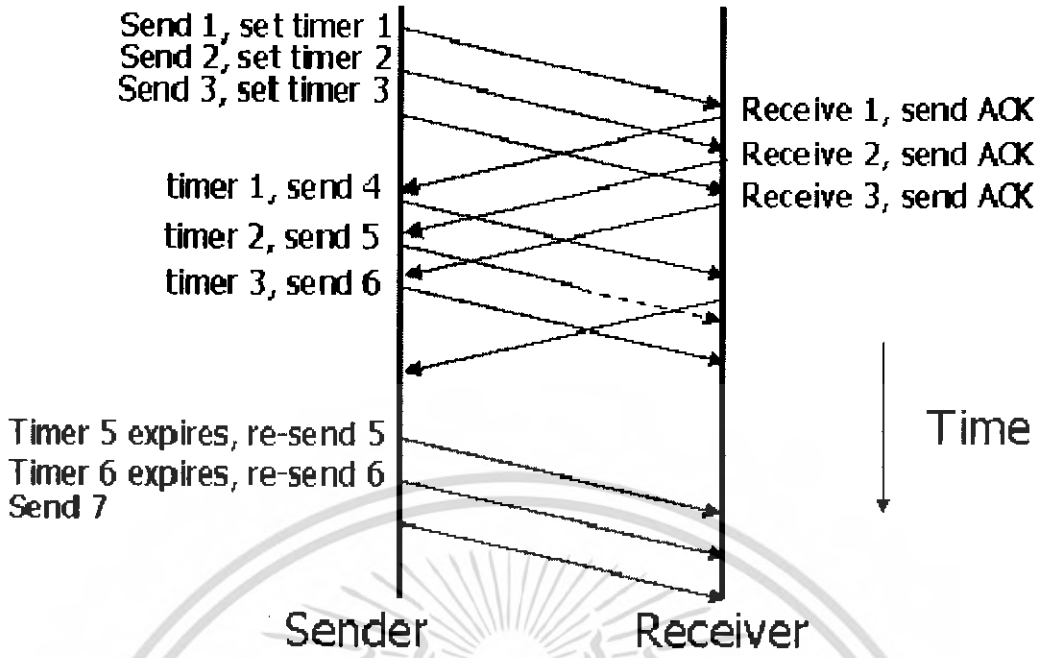
ปัญหาที่เกิดขึ้นมาจากการที่เราต้องการให้มีการตอบรับเสียก่อนที่จะมีการส่งเฟรมต่อไป ดังนั้นแทนที่จะให้ฝั่งส่ง ส่งครั้งละ 1 แต่เปลี่ยนเป็นส่งเฟรมได้คราวละ w และถ้าหากเลือก w ที่เหมาะสม จะทำให้ผู้ส่งสามารถส่งเฟรมได้ตลอดเท่ากับช่วงเวลาที่เฟรมเดินทางไปกลับ และจำนวนที่ส่งไม่ทำให้ วินโดว์เต็ม วิธีการนี้เรียกว่า pipelining ถ้าหากว่าความจุข้อมูลของช่องสัญญาณเท่ากับ b บิตต่อวินาที ขนาดเฟรมเท่ากับ 1 บิต เวลาเดินทางไปกลับเท่ากับ R วินาที สำหรับในแบบ stop and wait ความสามารถในการใช้ช่องสัญญาณ utilization จะเป็นไปดังสมการ (2.1)

$$U = \frac{1}{(1+bR)} \quad \text{ถ้า } 1 < bR \tag{2.1}$$

จะทำให้ความสามารถในการใช้ช่องสัญญาณน้อยกว่า 50 เปอร์เซ็นต์

การส่งแบบ pipelining ผ่านช่องทางการสื่อสารที่มีความผิดพลาดสูงจะเป็นเรื่องซับซ้อนมาก ถ้าเฟรมตรงกลางเกิดความผิดพลาดขึ้นระหว่างการส่งเฟรมเป็นจำนวนมาก จะมีเฟรมเป็นจำนวนมากที่ฝั่งรับได้รับก่อนที่จะรู้ว่ามีเฟรมที่ผิดพลาดเกิดขึ้น

วิธีแก้ปัญหา Go back n แสดงในรูปที่ 2.16

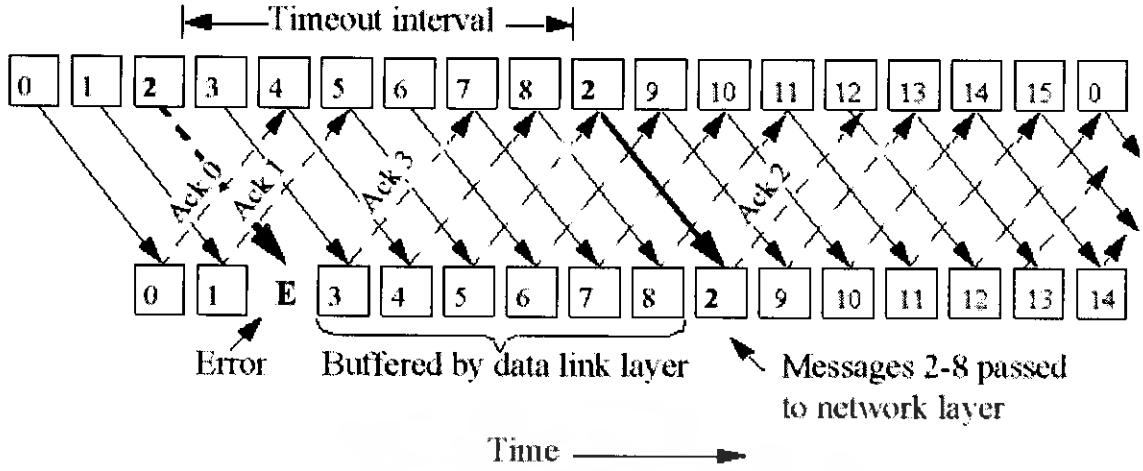


รูปที่ 2.16 Go back n

จากรูปที่ 2.16 เมื่อฝั่งส่งเริ่มส่งเฟรมข้อมูลหมายเลข 1,2,...,6 มายังฝั่งรับตามลำดับ ซึ่งฝั่งรับก็จะส่งเฟรมตอบรับ (ACK#1, ACK#2,...,ACK#6) กลับมาตามลำดับเฟรมที่ได้รับ เมื่อฝั่งรับตรวจพบเฟรมข้อมูลหมายเลข 5 ซึ่งอยู่ในสภาพที่ไม่สมบูรณ์ ก็จะลบเฟรมนั้นรวมทั้งเฟรมข้อมูลที่ตามมาทั้งหมด และจะไม่ส่งเฟรมตอบรับกลับมายังฝั่งส่ง ทางฝ่ายฝั่งส่งซึ่งยังไม่ทราบปัญหาที่เกิดขึ้น ก็จะส่งเฟรมข้อมูลตามมาเรื่อยๆ จนกระทั่งส่งเฟรมข้อมูลมาครบจำนวนที่วินโดว์นั้นกำหนด หรือเกิดไทม์เอาต์ขึ้น เนื่องจากไม่ได้รับเฟรมตอบรับภายในเวลาที่ต้องการ ดังนั้นฝั่งส่งจึงเริ่มส่งเฟรมข้อมูลมาใหม่ โดยเริ่มตั้งแต่เฟรมที่เสียหายหรือสูญหาย คือตั้งแต่เฟรมหมายเลข 5 วิธีการนี้เนื่องจากเป็นวิธีแบบง่ายๆ จึงอาจจะมีปัญหาในเรื่องประสิทธิภาพของระบบนั้นต่ำเกินไป โดยเฉพาะในระบบที่มีอัตราการเกิดข้อผิดพลาดในระหว่างการส่งข้อมูล

3. Selective Repeat

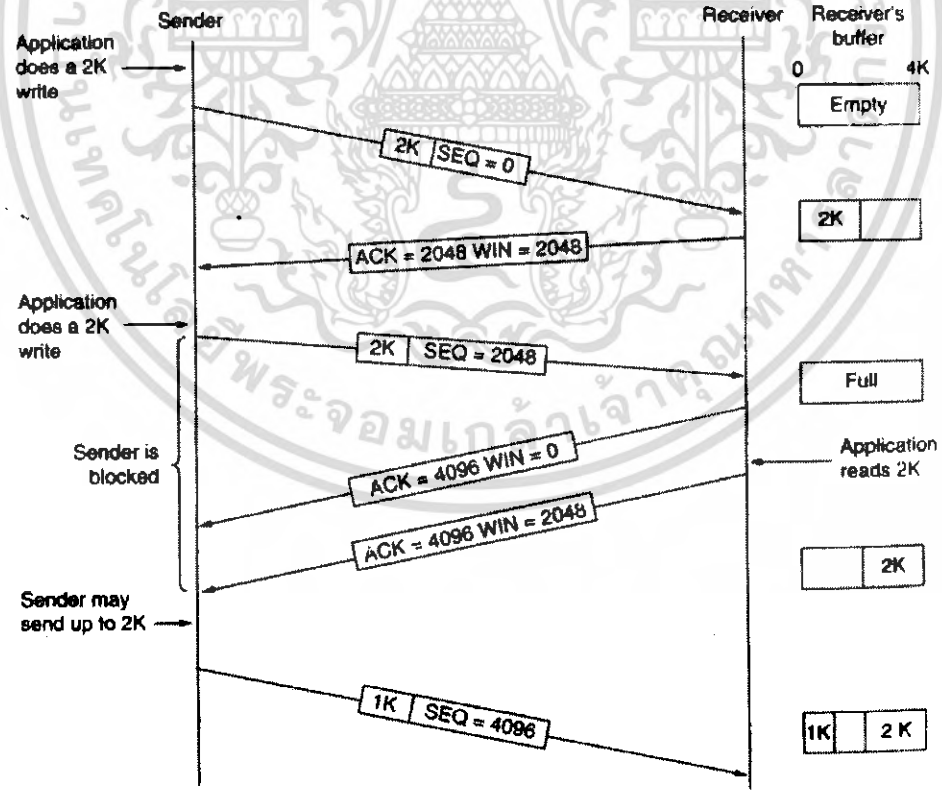
เป็นวิธีการแก้ปัญหาข้างต้นอีกแบบหนึ่ง ที่จะเลือกส่งเฟรมซ้ำเฉพาะเฟรมที่เสียหาย เมื่อฝั่งรับตรวจพบความผิดปกติของเฟรมหมายเลข 2 ก็จะลบเฉพาะเฟรมนั้นทิ้งไปและไม่ส่งเฟรมตอบรับกลับมา ส่วนเฟรมที่สมบูรณ์อื่นๆ ก็จะได้รับการนำส่งต่อไป ตามปกติ ทางฝั่งส่งเมื่อไม่ได้รับเฟรมตอบรับของเฟรมข้อมูลหมายเลข 2 ก็จะส่งเฉพาะเฟรมข้อมูล หมายเลข 2 มาใหม่แล้วจึงส่งเฟรมข้อมูลในลำดับอื่นๆ (หมายเลข 9,...) ต่อไป วิธีการนี้บังคับให้ทางฝั่งรับต้องมีบัฟเฟอร์สำหรับเก็บเฟรมข้อมูลที่มีขนาดเท่ากับ window size เพื่อเตรียมไว้ในการเก็บข้อมูลชั่วคราวในกรณีที่เกิดเฟรมไม่สมบูรณ์เกิดขึ้น ดังแสดงในรูปที่ 2.17



รูปที่ 2.17 Selective Repeat

2.7 วิธีทางการส่งผ่านข้อมูล

วิธีการควบคุมการส่งข้อมูลในโปรโตคอล TCP ไม่ได้ผูกติดกับแพ็กเก็ตตอบรับอย่างในชั้นเชื่อมโยงข้อมูล สมมติว่าผู้รับมีบัฟเฟอร์ ขนาด 4,096 ไบต์ ถ้าผู้ส่งจัดการส่งเซ็กเมนต์ขนาด 2,048 ไบต์ ผู้รับจะตอบกลับไปในเซ็กเมนต์ต่อไปว่ามีขนาดบัฟเฟอร์ เพียง 2,048 ไบต์ เท่านั้น



รูปที่ 2.18 การบริการหน้าต่างสื่อสารใน TCP

ต่อมาผู้ส่งได้ส่งเซ็กเมนต์ ขนาด 2,048 ไบต์ มา ผู้รับจะตอบกลับด้วยเซ็กเมนต์ต่อไปเป็น 0 ไบต์ ทำให้ผู้ส่งจะต้องหยุดส่งข้อมูลทันทีเพื่อรอให้ผู้รับจัดการนำเอาข้อมูลออกจากบัฟเฟอร์ก่อน แต่จะมีอยู่ 2 กรณี กรณีแรก ข้อมูลเร่งด่วนอาจถูกส่งออกมาได้ เช่น โปรเซสผู้ส่งอาจส่งคำสั่งมาบังคับให้ โปรเซสผู้รับหยุดการทำงานทันที หรือกรณีที่สอง ส่งเซ็กเมนต์มากระตุ้นให้ทางผู้รับส่งข้อมูลขนาดของเซ็กเมนต์ไปใหม่ เผื่อเนื้อที่ในบัฟเฟอร์ว่างบ้างแล้ว ข้อยกเว้นนี้ใช้ป้องกันข้อมูล เซ็กเมนต์จากผู้รับเกิดการสูญหาย ซึ่งจะทำให้เกิดสภาวะการรอคอยอย่างไม่มีสิ้นสุด

การทำงานโดยปกติ ผู้ส่งไม่ได้ถูกบังคับให้จัดการส่งข้อมูลที่ผู้ใช้มาส่งมาให้ในทันที ในเวลาเดียวกันผู้รับก็ไม่จำเป็นจะต้องส่งเซ็กเมนต์ตอบรับ ในทันทีที่ได้รับเซ็กเมนต์ใหม่เข้ามา เช่น เมื่อผู้รับ รับข้อมูลขนาด 2 กิโลไบต์ แรกเข้ามา ผู้รับอาจเลือกที่จะไม่ส่งข้อมูลขนาดเซ็กเมนต์ไป แล้วรอรับจนบัฟเฟอร์เต็ม วิธีนี้จะเปิดโอกาสให้ผู้ส่งสามารถส่งเซ็กเมนต์ขนาด 4 ไบต์ มาได้ทันทีที่ต้องการ ความคล่องตัวนี้ช่วยให้คู่สื่อสารสามารถปรับปรุงประสิทธิภาพการรับส่งข้อมูลให้เหมาะสมต่อสภาพแวดล้อมของตัวเอง

การทำงานของโปรแกรม Telnet ใช้สำหรับการเชื่อมต่อเข้ากับระบบโต้ตอบทันที (interactive) ของโฮสต์ผู้ให้บริการ การทำงานในลักษณะนี้ไม่เหมาะสมกับระบบเครือข่ายที่มีแบนด์วิดธ์แคบ (low bandwidth) วิธีการแก้ปัญหาคือ การดีเลย์แพ็กเก็ตตอบรับไว้อย่างน้อย 500 มิลลิวินาที โดยหวังว่าจะสามารถแทรกข้อมูลการตอบรับเข้าไปในแพ็กเก็ตอื่น ถ้ากำหนดให้การตอบสนองผู้ใช้เกิดขึ้นในทุกๆ 500 มิลลิวินาที จะทำให้ผู้ให้บริการส่งข้อมูลกลับมาเพียง 41 ไบต์ ซึ่งจะลดปริมาณข้อมูลทั้งหมดลงไปได้ครึ่งหนึ่ง

แม้ว่า ทางด้านผู้ให้บริการจะเพิ่มประสิทธิภาพการทำงานขึ้นได้บ้างแล้ว แต่ผู้ใช้เองยังต้องส่งข้อมูลเพียง 1 ไบต์ ผ่านแพ็กเก็ตขนาด 41 ไบต์ ในปี 1984 Nagle ได้เสนอวิธีการแก้ปัญหาโดย ส่งเฉพาะตัวอักษรแรก แล้วเก็บตัวอักษรที่ตามมาไว้ในบัฟเฟอร์ เมื่ออักษรตัวแรกได้รับการตอบรับแล้วจึงค่อยส่งตัวอักษรทั้งหมดที่อยู่ในบัฟเฟอร์ไปในเซ็กเมนต์เดียวกัน พร้อมกับเริ่มต้นเก็บตัวอักษรไว้ในบัฟเฟอร์อีก แล้วจึงส่งไปเมื่อเซ็กเมนต์ล่าสุดได้รับการตอบรับ อัลกอริทึมของ Nagle สามารถทำงานได้ดีในระบบที่ใช้ตัวอักษรเป็นหลัก ปัญหาอีกอย่างหนึ่งที่ทำให้ประสิทธิภาพโปรโตคอลของ TCP ลดลง เรียกว่า silly window syndrome ซึ่งจะอธิบายในหัวข้อที่ 2.7.1

2.7.1 Silly Window Syndrome

ค้นพบโดย Clark ปี 1982 RFC 1122 อธิบายปรากฏการณ์หนึ่งของ TCP ว่าเป็น silly window syndrome ปรากฏการณ์นี้อาจนำไปสู่ปัญหาทางด้านความเร็วในการรับ-ส่ง และก่อให้เกิดการใช้งานเครือข่ายอย่างไร้ประสิทธิภาพ ปรากฏการณ์นี้เกิดขึ้นเมื่อ window size ของ TCP ฝั่งผู้รับนั้นเล็กมาก ซึ่งอาจจะเป็นเพราะ window size ของ TCP ผู้รับนั้นเล็กมากจริงๆ หรืออาจจะเป็นเพราะสภาวะแวดล้อมต่างๆ เกี่ยวกับข้อมูลที่ไม่ได้รับการตอบกลับ และข้อมูลจำนวนเล็กน้อยที่ต้องการส่งประกอบรวมกันก็ได้

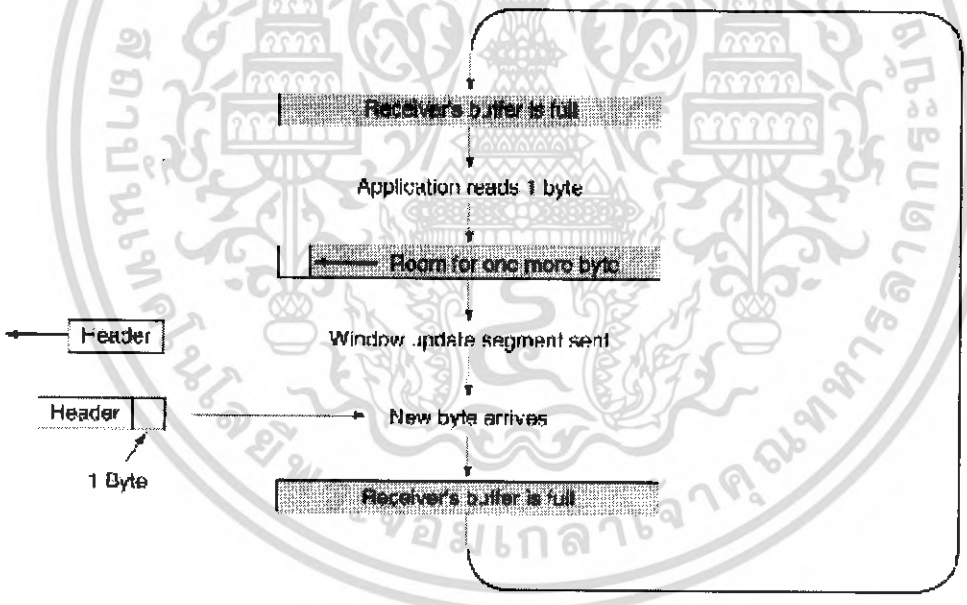
เช่น พิจารณาจากกรณีที่ window size ของฝั่งผู้รับเท่ากับ 1,024 ไบต์ ทางด้านผู้ส่งทำการส่งเซ็กเมนต์ข้อมูลขนาด 512 ไบต์ จำนวน 2 เซ็กเมนต์ เมื่อข้อมูลมาถึงผู้รับตอบรับ เซ็กเมนต์แรกกลับมา ผู้ส่งจะคำนวณ window size ให้มีขนาดเท่ากับ 512 ไบต์ ถ้าสมมุติว่าแอพพลิเคชั่นต้องการส่งข้อมูล

เร่งด่วนขนาด 64 ไบต์ ออกไป ในการทำงานผู้ส่งจะส่งข้อมูล 64 ไบต์ รวมกับ 448 ไบต์ (512-64) ที่ไม่ใช่ข้อมูล ถ้าหากว่าผู้รับตอบกลับข้อมูลด่วนนั้น window size จะถูกคำนวณโดยผู้ส่งให้เหลือเพียง 64 ไบต์ ผลที่เกิดขึ้น คือ window size ที่มีขนาดเล็กมากจะถูกส่งออกไปทำให้ทรัพยากรถูกใช้อย่างไม่มีประสิทธิภาพ

2.7.2 อัลกอริทึมของ Nagle และ Clack

RFC 1122 ให้คำอธิบายโดยละเอียดถึงวิธีการที่จะหลีกเลี่ยง silly window syndrome จุดที่สำคัญก็อยู่ที่โหนดผู้รับและผู้ส่ง นั่นคือผู้ส่งควรจะต้องถ่วงเวลาไว้บ้าง ถ้าข้อมูลที่ส่งนั้นมีขนาดเล็กเกินไป ยกเว้นว่าจะกำหนดตัวเลือกไว้ในฟิลด์ออพชั่นให้ส่งข้อมูลแบบ push เวลาที่รอคอยนี้ต้องกำหนดให้สัมพันธ์กับเวลาเดินทางไปกลับของข้อมูล ทางด้านรับนั้นก็ควรมีการรอคอยสักเล็กน้อยก่อนที่จะตอบรับกลับไป เพื่อดูว่าจะมีข้อมูลอื่นถูกส่งมาอีกหรือไม่ และจะสามารถตอบกลับไปได้ในคราวเดียว ซึ่งเวลาที่ใช้ในการรอคอยนั้นควรจะสัมพันธ์กับเวลาที่ใช้ในการส่งข้อมูลกลับ และไม่ควรนานเกินไปจนกระทั่งผู้ส่งต้องส่งข้อมูลมาใหม่ ดังรูปที่ 2.19

Minimum (1/2 Receive-Buffer-Size, Maximum Segment Size) (2.2)



รูปที่ 2.19 Silly Window Syndrome

จะเห็นได้ว่าอัลกอริทึมของ Nagle และ Clack สามารถนำมาใช้แก้ไขปัญหา silly window syndrome ร่วมกันได้ Nagle พยายามแก้ปัญหาในส่วนของผู้ส่ง ขณะที่ Clack แก้ปัญหาที่ผู้รับ

ทางด้านผู้รับข้อมูลยังสามารถนำบัฟเฟอร์มาช่วยปรับปรุงการทำงานให้ดีขึ้นสำหรับงานที่ไม่ต้องการการโต้ตอบทันทีทันใด วิธีการนี้จะช่วยลดการสื่อสารระหว่างโปรเซสผู้ใช้ กับ เอ็นติตี้ลงได้มาก ซึ่งจะทำให้ประสิทธิภาพโดยรวมดีตามไปด้วย

2.8 กลไกควบคุมความผิดพลาด (Error Control)

TCP เป็นโปรโตคอลที่รับประกันความถูกต้องของข้อมูล นั่นหมายความว่าก่อนที่ TCP จะส่งข้อมูลให้กับชั้นโปรโตคอลประยุกต์ ข้อมูลนั้นต้องไม่มีข้อผิดพลาดใดๆ ดังนั้น TCP จะต้องมีกระบวนการตรวจสอบสิ่งเหล่านี้คือ การตรวจสอบความถูกต้องของเซ็กเมนต์ การสูญหายของเซ็กเมนต์ การไม่เรียงลำดับของเซ็กเมนต์และการตรวจสอบเซ็กเมนต์ที่ซ้ำกัน

2.8.1 การตรวจสอบความถูกต้องของเซ็กเมนต์

TCP จะใช้ฟิลด์ checksum ในการตรวจสอบว่าเซ็กเมนต์นั้นถูกต้องหรือไม่ ถ้าพบว่าไม่ถูกต้อง TCP จะทำการทิ้งเซ็กเมนต์นั้นไป และจะไม่ส่งเซ็กเมนต์ตอบรับ (acknowledgment) กลับไปว่าได้รับเซ็กเมนต์นี้แล้ว

2.8.2 Sequence Numbers and Acknowledgement Numbers

2 ฟิลด์ นี้เป็นส่วนสำคัญในบริการการส่งข้อมูลที่มีความน่าเชื่อถือ ของ TCP โดยที่ TCP จะมองข้อมูลว่าเป็นลำดับของไบต์ที่ต้องส่งต่อกันมา โดยลำดับนี้นั้นจะเรียกว่า sequence number แต่ถ้ากลุ่มของข้อมูลที่ส่งมีขนาดใหญ่กว่า MSS แล้ว TCP จะทำการแบ่งข้อมูลนั้นเป็นหลายเซ็กเมนต์ซึ่งฟิลด์ที่เป็น sequence number ของเซ็กเมนต์นั้นจะใส่ค่า sequence number ของข้อมูลไบต์แรกที่อยู่ในเซ็กเมนต์นั้นเข้าไป เช่น การส่งข้อมูล 500,000 ไบต์ โดยไบต์นี้จะมี sequence number เป็น 0 และ MSS มีขนาด 1,000 ไบต์ ดังนั้น TCP ก็จะทำการแบ่งข้อมูลออกเป็น 500 เซ็กเมนต์ โดยเซ็กเมนต์แรกจะมี sequence number เป็น 0 เซ็กเมนต์ที่สองเป็น 1,000 และเซ็กเมนต์ที่สามเป็น 2,000 ซึ่งจะเป็นแบบนี้ต่อไปเรื่อยๆ

Acknowledgement Number ใช้ตอบว่าได้รับเซ็กเมนต์นั้นแล้วและบอกว่าต้องการข้อมูลอะไรต่อไปอีก โดยในฟิลด์ที่เป็น acknowledgement number ของเซ็กเมนต์ที่ตอบกลับมานั้นจะใส่ sequence number ของที่ถัดจาก sequence number ของไบต์สุดท้ายในเซ็กเมนต์ที่ได้รับมา เช่น ถ้า A ได้รับไบต์ 0 ถึง 535 จาก B แล้ว A ก็จะตอบกลับด้วยเซ็กเมนต์ที่มี acknowledgement number เป็น 536 ไปยัง B

อีกตัวอย่าง สมมติว่า ถ้า A ได้รับเซ็กเมนต์ที่ 1 ซึ่งมีไบต์ตั้งแต่ 0 ถึง 535 และเซ็กเมนต์ที่ 3 ที่มีไบต์ ตั้งแต่ 900 ถึง 1,000 แต่ A ยังไม่ได้รับเซ็กเมนต์ที่ 2 ที่มีไบต์ตั้งแต่ 535 ถึง 899 เลยนั้น A จะตอบกลับเฉพาะเซ็กเมนต์ที่ 1 เท่านั้นเพราะ TCP จะตอบกลับจนถึงไบต์แรกที่หายไปเท่านั้น ถ้าเกิดพบว่า ข้อมูลถูกส่งมาแบบ out of order คนที่จัดการเกี่ยวกับ TCP จะต้องตัดสินใจเองเพราะ TCP ใน RFCs ไม่ได้บอกกฎอะไรเกี่ยวกับเรื่องนี้ ซึ่งมี 2 ทางเลือกให้เลือกคือ

1. ทิ้งไบต์ที่ out of order ไปเลย
2. เก็บเอาไว้แล้วรอไบต์ที่หายมาเติมทีหลัง

ในกรณีหลังจะมีประสิทธิภาพดีกว่าในแง่ของแบนด์วิดธ์ของเครือข่ายแต่เราสนใจในกรณีแรกเพราะง่ายในการดำเนินการ

ถ้าสมมติให้ sequence number แรกเป็น 0 แต่ในความเป็นจริง TCP ทั้ง 2 ด้านของการเชื่อมต่อจะสุ่ม sequence number แรกขึ้นมา เพื่อทำการลดโอกาสที่จะซ้ำกับเซ็กเมนต์ที่ยังค้างอยู่ของการเชื่อมต่อที่ปิดไปแล้ว เมื่อใช้คู่โฮสต์ และ พอร์ตเหมือนกันกับการเชื่อมต่อเก่า

2.8.3 Reliable Data Transfer

เนื่องจาก TCP มีการใช้บริการของไอพี ที่ไม่ reliable ดังนั้น TCP ต้องให้ reliable data transfer เองโดยมีหลักดังนี้

1. Retransmission

การ retransmission ของข้อมูลที่หายไปหรือไม่ถูกต้องเป็นสิ่งสำคัญของ reliable data transfer โดย TCP จะใช้ positive acknowledgement และไทม์เมอร์ เพื่อให้รู้ว่ามันต้อง retransmission หรือไม่ TCP เองก็ไม่สามารถบอกได้ว่าเซ็กเมนต์หรือ ACK ที่ได้รับนั้นเกิดการสูญหายหรือเกิดความผิดพลาดหรือส่งมาช้าเกินกำหนดหรือเปล่า ทำให้ส่งได้หลายครั้งโดยไม่ต้อง acknowledgement เซ็กเมนต์ที่ยังค้างก่อน ตารางที่ 2.3 จะสรุปหลักการส่ง ACK ของ TCP ทางฝั่งรับ จากที่กล่าวมาจะเห็นได้ว่า TCP จะไม่ใช้ negative acknowledgement แต่ใช้ re – acknowledgement แทน

Event	TCP Receiver Action
เซ็กเมนต์มาแบบ in – order และข้อมูลที่ได้จนถึง sequence ตัวนี้ได้ ACK หมดแล้ว ไม่มี gap ในข้อมูลที่ได้มา	delayed ACK รอถึง 500 มิลลิวินาที สำหรับ in – order เซ็กเมนต์ถัดไป ถ้าไม่มีมาก็ส่ง ACK
เซ็กเมนต์มาแบบ in – order แต่เซ็กเมนต์บางตัวยังไม่ได้ ACK ไม่มี gap ในข้อมูลที่ได้มา	ส่ง ACK แบบ cumulative ครั้งเดียว
เซ็กเมนต์มาแบบ out – of – order ซึ่ง sequence number สูงกว่าที่ต้องการมี gap เกิดขึ้น	ส่ง duplicate ACK ทันทีโดยใส่ sequence number ของไบต์ที่ต้องการถัดไป
เซ็กเมนต์บางส่วนหรือทั้งหมด ที่จะเติมใน gap นั้นมาถึง	ส่ง ACK ทันที โดยเริ่มจากเซ็กเมนต์ล่างของ gap

ตารางที่ 2.3 TCP ACK Generation Recommendations

2. A Few Interesting Scenarios

สมมติว่ามีการส่งเซ็กเมนต์จากโฮสต์ A ไปยังโฮสต์ B ถ้าเซ็กเมนต์นั้นมี sequence number 92 มีข้อมูล 8 ไบต์ หลังจากทำการส่งแล้วโฮสต์ A จะรอให้โฮสต์ B ตอบกลับด้วย acknowledgement number 100 แต่ในกรณีที่ไม่ได้รับ ACK เพราะหมดเวลาเสียก่อน โฮสต์ A จะทำการส่งเซ็กเมนต์เดิมไปใหม่อีกครั้ง ดังนั้นโฮสต์ B ก็จะได้ duplicate segment ซึ่งโฮสต์ B ก็จะไม่รับเซ็กเมนต์นี้ที่ส่งมาซ้ำกับของเดิม

3. การสูญหายของเซ็กเมนต์

หากมีเซ็กเมนต์ที่ไม่สามารถเดินทางไปถึงปลายทางได้ เมื่อเป็นเช่นนี้ถ้าต้นทางไม่ได้รับเซ็กเมนต์ตอบรับกลับมาภายในเวลาที่กำหนด จะต้องทำการส่งเซ็กเมนต์นั้นกลับไปใหม่อีกครั้ง

4. การตรวจสอบ เซ็กเมนต์ซ้ำกัน

TCP ที่ต้นทางสามารถที่จะส่งเซ็กเมนต์ซ้ำกันได้ เหตุการณ์แบบนี้จะเกิดขึ้นได้ เช่น เมื่อต้นทางไม่ได้รับเซ็กเมนต์ตอบรับกลับมาภายในเวลาที่กำหนด จะต้องทำการส่งเซ็กเมนต์นั้นใหม่อีกครั้ง การแก้ปัญหาการส่งเซ็กเมนต์ซ้ำกันจะกระทำที่ปลายทาง เนื่องจาก TCP ที่ปลายทางจะต้องรับ เซ็กเมนต์โดยมีหมายเลขเรียงตามลำดับกันไป ถ้ามีหมายเลขที่ซ้ำกันก็จะทิ้งเซ็กเมนต์นั้นไป

5. การไม่เรียงลำดับของเซ็กเมนต์

เนื่องจาก TCP จะใช้บริการของไอพี ในการนำส่งข้อมูล ซึ่งไอพีเป็นโปรโตคอลในชั้นเครือข่าย (Network Layer) ที่ให้บริการแบบ connectionless และไม่มีการรับประกันความถูกต้องของข้อมูล ดังนั้น ดาต้าแกรมที่ถูกส่งโดยไอพีนั้นอาจจะเดินทางไปคนละเส้นทางกัน ทำให้อาจจะไปถึงปลายทางโดยไม่เรียงลำดับกันได้ ในการแก้ปัญหาของการไม่เรียงลำดับของเซ็กเมนต์จะกระทำที่ปลายทาง โดย TCP ที่ปลายทางจะไม่ตอบรับกลับในกรณีที่หมายเลขเซ็กเมนต์ไม่เรียงลำดับกัน ดังนั้น TCP ที่ต้นทางจะต้องส่งเซ็กเมนต์ที่ไม่มีการตอบรับกลับไปใหม่อีกครั้งหนึ่ง ซึ่งเป็นไปได้อีกว่าจะเกิดเหตุการณ์ส่งเซ็กเมนต์ซ้ำกัน แต่อย่างไรก็ตาม TCP ที่ปลายทางได้มีกระบวนการตรวจสอบอยู่แล้ว

6. การสูญหายของเซ็กเมนต์ตอบรับ

การสูญหายของเซ็กเมนต์ตอบรับ โดยเริ่มจาก TCP ต้นทางจะส่ง 3 เซ็กเมนต์ จากนั้น TCP ปลายทาง ได้ตอบรับด้วย ACK เซ็กเมนต์ แต่ปรากฏว่าเซ็กเมนต์ที่มี acknowledgment number เท่ากับ 1,601 นี้ได้สูญหายในระหว่างทาง เมื่อ TCP ต้นทางได้รับเฉพาะ เซ็กเมนต์ที่มี acknowledgment number เท่ากับ 1,801 TCP ต้นทางจะสามารถสรุปได้เลยว่า ได้รับไบต์ข้อมูลตั้งแต่ 1,201 ถึง 1,800 แล้ว ดังนั้นจะไม่ส่งเซ็กเมนต์ก่อนหน้านี้ไปอีก

7. กลไกการตอบกลับแพ็กเก็ต

มี 2 ประเภท ประเภทแรกคือ PAR (Positive Acknowledgement and Retransmission) กลไกการทำงานคือ เมื่อฝั่งส่งทำการส่งแพ็กเก็ตหนึ่ง ก็จะรอการตอบกลับจากฝั่งรับ แล้วค่อยส่งแพ็กเก็ตต่อไป ถ้าไม่ได้รับการตอบกลับภายในเวลาที่กำหนดก็จะส่งแพ็กเก็ตนั้นอีกครั้ง ปัญหาของกลไกนี้คือ ถ้าข้อมูลประกอบด้วยหลายๆ แพ็กเก็ตและการที่ฝั่งรับต้องส่งแพ็กเก็ตตอบรับต่อทุกๆ แพ็กเก็ตที่ได้รับนั้นอาจเป็นการสิ้นเปลืองแบนด์วิดธ์และเป็นขบวนการที่ไร้ประสิทธิภาพเนื่องจากฝั่งส่งจะใช้เวลาในการรอมากกว่าการส่งข้อมูล

กลไกที่สองจะแก้ปัญหานี้ ซึ่งกลไกนี้เรียกว่า “sliding window” มีกลไกในการทำงาน คือ ฝั่งรับสามารถยืนยันการรับหลายแพ็กเก็ตวิธีนี้จะช่วยลดจำนวนแพ็กเก็ตที่ต้องไหลเวียนในเครือข่าย และฝั่งส่งสามารถส่งทีละหลายๆ แพ็กเก็ตก่อนที่จะรอการตอบกลับ

2.9 การควบคุมความคับคั่ง (Congestion Control)

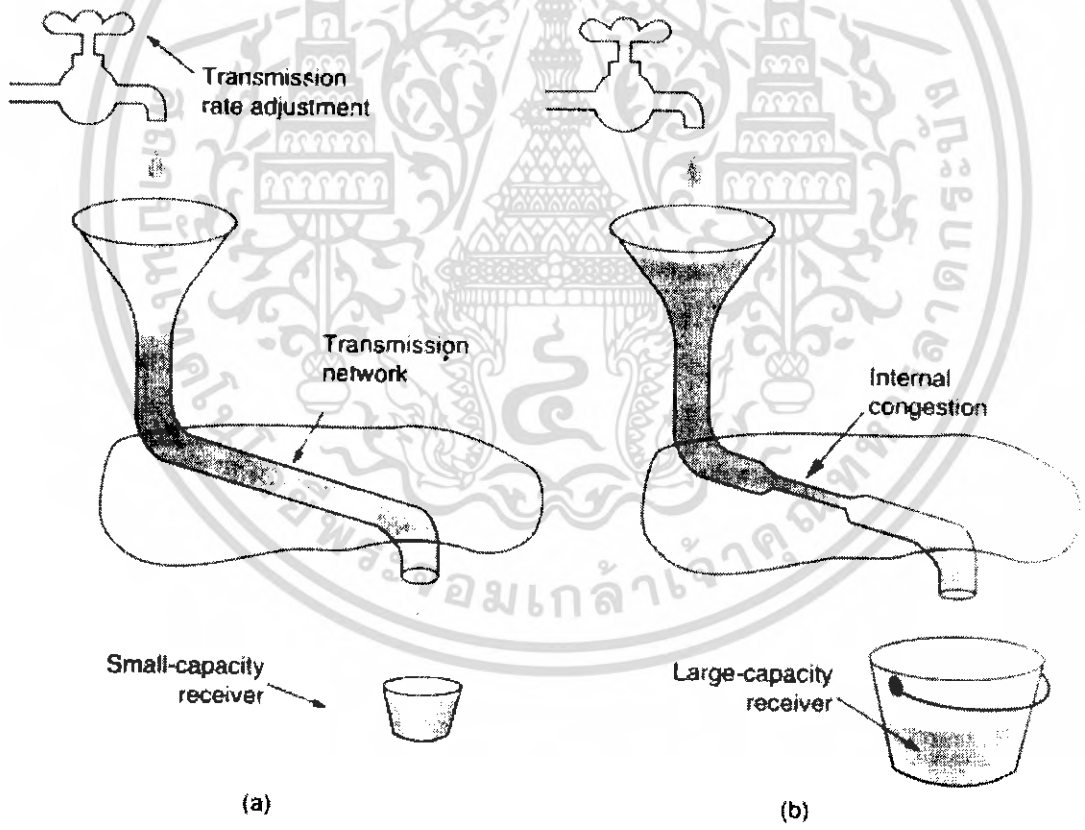
เมื่อปริมาณข้อมูลในเครือข่ายมีมากกว่าความสามารถในการรับ-ส่งข้อมูล ก็จะเกิดปัญหาความคับคั่ง (congestion) ของข้อมูลขึ้น ซึ่งมีลักษณะคล้ายกับปัญหาจราจรติดขัดในช่วงเวลาชั่วโมงเร่งด่วน

จะแก้ปัญหาโดยการใช้โปรโตคอล TCP เป็นตัวควบคุมการส่งหรือไม่ส่งข้อมูลไปยังชั้นสื่อสารควบคุมเครือข่าย จึงเป็นการควบคุมการส่งข้อมูลในอัตราที่ช้าลง

การแก้ปัญหาในทางทฤษฎีจะใช้หลักการของฟิสิกส์มาช่วย คือ กฎการรักษาแพ็กเก็ตแนวคิดพื้นฐานคือการไม่ส่งแพ็กเก็ตใหม่เข้าสู่ระบบเครือข่ายจนกว่าแพ็กเก็ตเดิมจะถูกส่งออกไปเรียบร้อยแล้ว

ขั้นตอนแรกของการแก้ปัญหาคือ จะต้องสามารถตรวจพบว่ามีความคับคั่งเกิดขึ้นให้ได้ ซึ่งในปัจจุบันนี้แพ็กเก็ตที่สูญหายในการนำส่ง เนื่องจากสัญญาณรบกวน (noise) เกิดขึ้นน้อยมาก (แม้ว่าในบางระบบ เช่นการสื่อสารไร้สายจะยังมีปัญหานี้อยู่ก็ตาม) ดังนั้นการที่แพ็กเก็ตเดินทางมาถึงจุดหมายก่อนหมดระยะเวลารอคอย จึงเกิดขึ้นเพราะความคับคั่งของข้อมูลเพียงสาเหตุเดียวโปรโตคอล TCP จึงใช้ตรวจสอบระยะเวลารอคอย เป็นการตรวจสอบความคับคั่งของข้อมูล

เมื่อการเชื่อมต่อเริ่มขึ้นต้องมีการกำหนด window size หรือขนาดสูงสุดของเซ็กเมนต์ที่อนุญาตให้ใช้ในการสื่อสาร โดยปกติฝั่งรับจะใช้ขนาดบัฟเฟอร์เป็นตัวกำหนด ถ้าฝั่งส่งใช้ตัวเลขนี้ในการสื่อสารอย่างเคร่งครัด ปัญหาในการสื่อสารจะไม่เกิดขึ้นทางฝั่งรับอย่างแน่นอน แต่อาจทำให้เกิดความคับคั่งในเครือข่าย



รูปที่ 2.20 แสดงการเปรียบเทียบระบบที่เกิดความคับคั่งของระบบ

จากรูปที่ 2.20 แสดงถึงตัวอย่างของระบบที่เกิดความคับคั่งขึ้น โดยใช้ภาพของ ของเหลวเป็นตัวเปรียบเทียบ รูปที่ 2.20 (a) มีท่อขนาดใหญ่ถ่ายเทของเหลวไปยังถังขนาดเล็ก ถ้าฝั่งส่งไม่ถ่ายเทของเหลว

มาเกินจนกว่าที่ถังจะรับได้ ก็จะไม่มีการสูญเสียเกิดขึ้น รูปที่ 2.20 (b) ถังที่รับของเหลวมีขนาดใหญ่มาก ปัญหาเกิดขึ้นมาจาก ท่อถ่ายเทของเหลวช่วงหนึ่งมีขนาดเล็กมาก ถ้าของเหลวไหลสู่ท่อในปริมาณมากและเร็วเกินไปของเหลวก็จะล้นออกจากท่อ ไม่ได้ล้นออกจากถังที่รอล้นออกอยู่

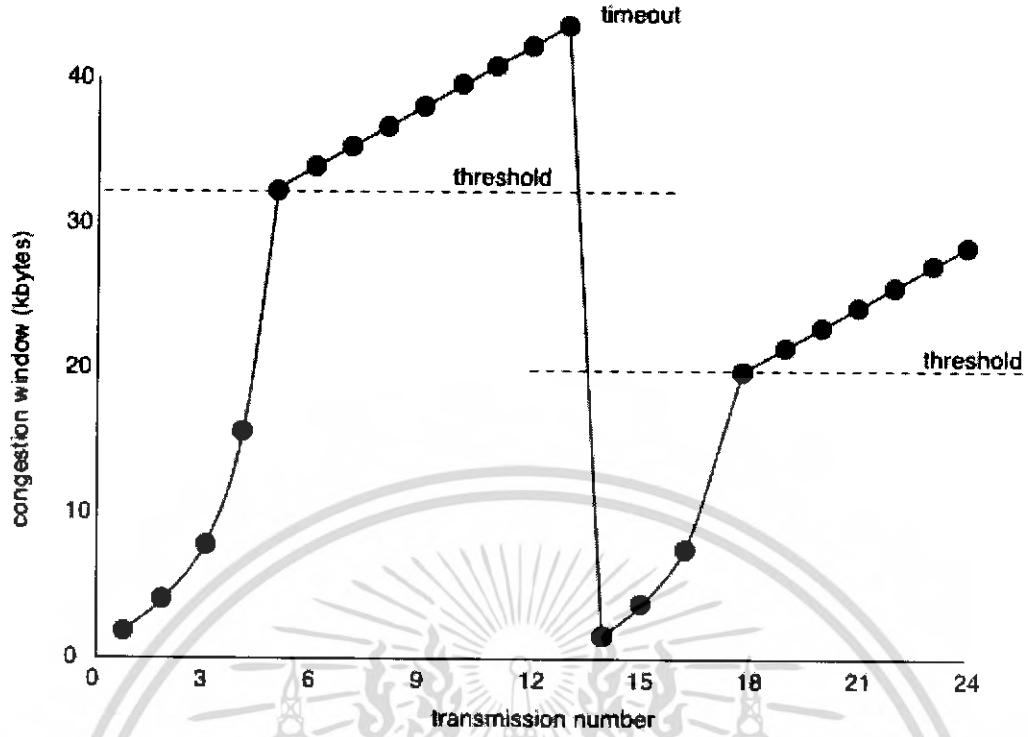
หนทางแก้ไขปัญหาในระบบจึงขึ้นอยู่กับความสามารถในการรับทราบปัญหา 2 แบบ คือ ความจุข้อมูลของระบบเครือข่ายและความจุข้อมูลของฝั่งรับ ซึ่งมีวิธีแก้ไขต่างกัน

ฝั่งส่งจะต้องเก็บตัวเลข 2 ตัว คือ window size ที่ฝั่งรับแจ้งให้ทราบ และขนาดของความคับคั่งของวินโดว์ (congestion windows) ซึ่งตัวเลขแต่ละตัวใช้กำหนดขนาดของเซ็กเมนต์ที่ฝั่งส่งจะส่งไปยังฝั่งรับ ซึ่งจะต้องเลือกใช้ค่าที่น้อยกว่าในระหว่างตัวเลข 2 ตัวนี้

เมื่อการเชื่อมต่อเริ่มขึ้น ฝั่งส่งจะใช้ขนาดเซ็กเมนต์สูงสุดที่ผู้ใช้อนุญาตเป็นขนาดความคับคั่งของวินโดว์ แล้วจึงเริ่มส่งเซ็กเมนต์ขนาดสูงสุดไปยังฝั่งรับ ถ้าได้รับการตอบรับก่อน หมดระยะเวลารอคอย ฝั่งส่งจะเพิ่มขนาดของความคับคั่งของวินโดว์ เป็น 2 เท่า แล้วจัดการส่งเซ็กเมนต์ขนาดเดิมจำนวนสองเซ็กเมนต์ออกมาติดๆ กัน ถ้าสำเร็จฝั่งส่งจะพยายามเพิ่มขนาดขึ้นเรื่อยๆครั้งละ 1 เท่าตัวของขนาดความคับคั่งของวินโดว์ ล่าสุด

ฝั่งส่งจะปรับขนาดของความคับคั่งของวินโดว์ไปเรื่อยๆ จนกว่าจะเกิดปัญหาขึ้น คือ แพ็กเก็ตตอบรับเดินทางมาไม่ถึงภายในเวลารอคอยที่กำหนด หรือส่งข้อมูลไปจนเต็มขนาดวินโดว์ที่ผู้ใช้กำหนด แต่อัลกอริทึมนี้จะทำงานได้อย่างช้าๆ ในปี 1988 Jacobson ได้พิสูจน์ให้เห็นว่าอัลกอริทึมนี้ทำงานได้อย่างรวดเร็ว ในปัจจุบันโปรแกรมที่เขียนขึ้นโดยใช้ข้อกำหนดของโปรโตคอล TCP ทุกชนิดจะต้องนำอัลกอริทึมนี้ไปใช้

การแก้ปัญหาความคับคั่งได้เพิ่มพารามิเตอร์ตัวที่สามคือ ค่าเทรีชโฮลด์มีค่าเริ่มต้นเป็น 64 กิโลไบต์ ถ้าแพ็กเก็ตตอบรับเดินทางมาไม่ทันระยะเวลารอคอย (time out) ค่าเทรีชโฮลด์จะถูกกำหนดให้ใหม่เป็นครึ่งหนึ่งของขนาดความคับคั่งของวินโดว์ ส่วนความคับคั่งของวินโดว์ถูกกำหนดให้เหลือเท่ากับขนาดสูงสุดของเซ็กเมนต์จากนั้นฝั่งส่งจะใช้วิธีการเดิมในการพยายามเพิ่มขนาดเซ็กเมนต์โดยใช้ค่าเทรีชโฮลด์เป็นตัวกำหนดค่าเพดาน เมื่อการสื่อสารดำเนินต่อไปฝั่งส่งจะค่อยๆ ปรับขนาดของความคับคั่งของวินโดว์ขึ้นไปครั้งละ 1 เซ็กเมนต์ ในทุกครั้งที่การส่งประสบความสำเร็จ



รูปที่ 2.21 ตัวอย่างอัลกอริทึมที่ใช้ควบคุมความคับคั่งบนระบบอินเตอร์เน็ต

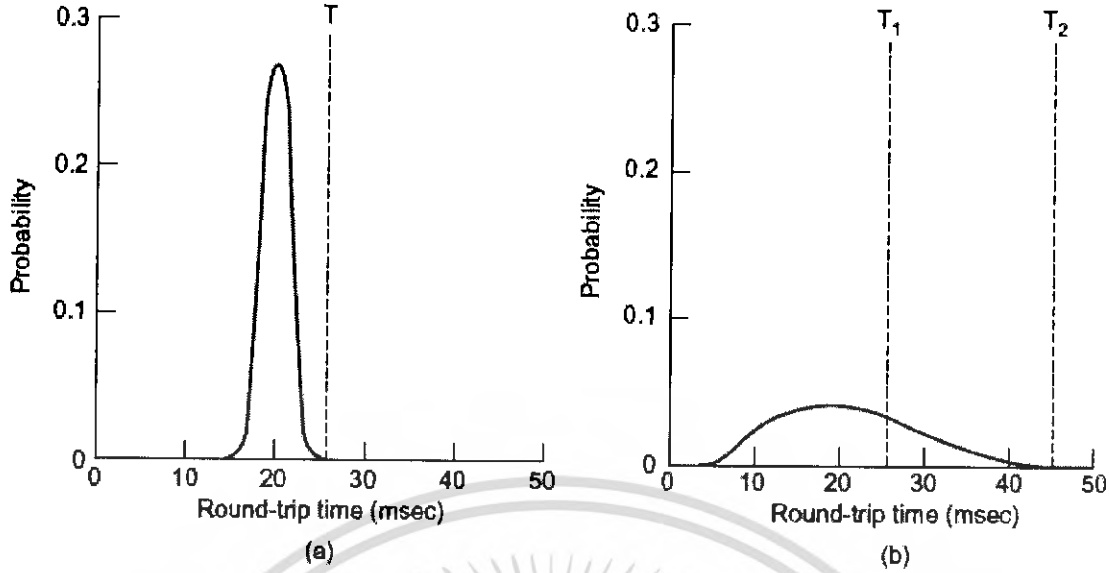
จากรูปที่ 2.21 กำหนดให้ค่าเซ็กเมนต์มีค่าเป็น 1 กิโลไบต์ ค่าเริ่มต้นความคับคั่งเป็น 64 กิโลไบต์ ฝั่งส่งเริ่มส่งข้อมูลแล้วเพิ่มขนาดหนึ่งเท่าตัวไปเรื่อยจนกระทั่งแพ็กเก็ตตอบรับเดินทางมาไม่ทันระยะเวลารอคอย จึงปรับค่าเป็น 32 กิโลไบต์ การสื่อสารดำเนินต่อไปโดยเริ่มจากการส่งเซ็กเมนต์ 1 กิโลไบต์ และเพิ่มขึ้นเท่าตัวจนเซ็กเมนต์มีขนาดเท่ากับค่าเทร็ชโฮลด์(เซ็กเมนต์ ที่ 0-5) การเพิ่มขนาดครั้งต่อไป (เซ็กเมนต์ ที่ 6-13) จะกระทำครั้งละ 1 กิโลไบต์ (ขนาดสูงสุดของเซ็กเมนต์) จากรูปที่ 2.21 การส่งเซ็กเมนต์ที่ 13 มีปัญหาคือแพ็กเก็ตตอบรับเดินทางมาไม่ทันเวลารอคอยค่าเทร็ชโฮลด์ จึงถูกกำหนดขึ้นมาใหม่ เป็น 20 กิโลไบต์ การส่งข้อมูลจึงดำเนินต่อไปโดยเริ่มต้นจากเซ็กเมนต์ขนาด 1 กิโลไบต์ แล้วเพิ่มขนาดเป็นเท่าตัวจนมีค่าเท่ากับค่าเทร็ชโฮลด์ (เซ็กเมนต์ที่ 14-18) แล้วค่อยๆเพิ่มขนาด 1 กิโลไบต์ (เซ็กเมนต์ที่19-24)

ในกรณีที่ไม่มีปัญหาเกิดขึ้นขนาดของเซ็กเมนต์จะค่อยๆ เพิ่มขนาด ไปจนกระทั่งเซ็กเมนต์มีขนาดเท่ากับขนาดวินโดว์ที่ฝั่งรับกำหนดและมีขนาดค่าคงที่ไปเรื่อยๆ

2.10 การบริหารการจับเวลา (Timer)

2.10.1 Retransmission Timer

โปรโตคอล TCP ใช้การจับเวลาที่สำคัญที่สุดคือ retransmission timer ซึ่งจะเริ่มต้นทันทีในแพ็กเก็ตได้ถูกส่งออกไปและหยุดเมื่อแพ็กเก็ตตอบรับเดินทางมาถึง



รูปที่ 2.22 (a) Probability Density สำหรับการมาถึงของแพ็กเก็ตตอบรับในชั้นสื่อสารเชื่อมต่อข้อมูล
(b) Probability Density สำหรับการมาถึงของแพ็กเก็ตตอบรับใน TCP

การกำหนดระยะเวลารอคอยที่เหมาะสม จะกำหนดให้มีค่ามากกว่าค่าจากการคำนวณเพียงเล็กน้อย ดังรูปที่ 2.22 (a) เนื่องจากการดีเลย์ในชั้นเชื่อมต่อข้อมูลแทบจะไม่เกิดขึ้นเลย นอกจากแพ็กเก็ตเกิดสูญหายในระหว่างการนำส่ง

โปรโตคอล TCP มีสภาพแวดล้อมของการทำงานที่แตกต่างออกไปมาก ทำให้ค่าจากการคำนวณหา ความเข้มข้นของการคาดหวัง (probability density function) มีลักษณะดังรูปที่ 2.22 (b) การประเมินเวลาจากฝั่งส่งไปยังผู้รับและจากฝั่งรับกลับมายังฝั่งส่ง หรือระยะเวลารอคอยนั้นถ้ากำหนดให้ระยะเวลารอคอยสั้นเกินไป ของ T1 ดังรูปที่ 2.22 (b) จะทำให้เกิดการส่งแพ็กเก็ตซ้ำโดยไม่จำเป็น แต่ถ้ากำหนดให้มีค่านานเกินไป ของ T2 ในรูปที่ 2.22 (b) จะทำให้ประสิทธิภาพลดลง ค่าการกระจายของระยะเวลาการรอคอยนี้ยังเปลี่ยนแปลงได้อย่างรวดเร็ว และยังเกิดขึ้นได้ตลอดเวลา เมื่อระบบเข้าสู่สภาวะคับคั่ง หรือออกจากสภาวะคับคั่ง

ค่า retransmission time นั้นสามารถหาได้จากสมการ (2.3)

$$\text{Re}transmission\ Time \quad = \quad 2 * RTT \tag{2.3}$$

หนทางแก้ไขคือ ใช้อัลกอริทึมที่พัฒนามาจาก Jacobson โดยการเชื่อมต่อแต่ละช่องสัญญาณจะต้องกำหนดค่า RTT (Round-Trip-Time) ดังสมการ (2.4)

$$RTT \quad = \quad \alpha\, RTT + (1-\alpha)M \tag{2.4}$$

โดยที่ RTT คือค่าโดยประมาณของระยะเวลาในการส่งแพ็กเก็ตไปถึงฝั่งรับและฝั่งรับส่งแพ็กเก็ตตอบรับกลับมาถึงฝั่งส่ง

M คือ ค่า RTT ปัจจุบัน

α smoothing factor มีค่าเท่ากับ $\frac{7}{8}$

การคำนวณหาค่าที่จะนำมาใช้จริง Jacobson ได้เสนอค่า β เป็นสัดส่วนกับค่าเบี่ยงเบนมาตรฐานของระยะเวลารอคอยที่เกิดขึ้นจริง และเสนอให้ใช้ค่าเบี่ยงเบนเฉลี่ยแทนค่าเบี่ยงเบนมาตรฐาน และใช้พารามิเตอร์ตัวใหม่คือ D ในทุกครั้งที่การส่งสำเร็จค่าแตกต่างที่เกิดขึ้นระหว่างค่าคาดหวังและค่าที่เกิดขึ้นจริง จะถูกนำมาคำนวณเป็นค่าพารามิเตอร์ D ดังสมการ (2.5)

$$D = \alpha D + (1+\alpha)|RTT - M| \tag{2.5}$$

ในปัจจุบันโปรโตคอล TCP จะนำสมการ (2.5) ไปใช้ และกำหนดค่าเวลารอคอยดังสมการ (2.6)

$$Timeout = RTT + 4D \tag{2.6}$$

ค่าตัวเลข 4 ขึ้นอยู่กับ เอ็นติตี้ TCP จะกำหนดเองมีผลดีคือ เป็นการเลื่อนบิตของตัวตั้งเพียง 1 ครั้ง และแพ็กเก็ตเกือบทั้งหมดจะเดินทางกลับมาก่อนจะหมดเวลารอคอย (4 เท่าของค่าเบี่ยงเบนเฉลี่ย) ทำให้ลดอัตราการส่งข้อมูลซ้ำลงไปได้มาก

ปัญหาหนึ่งของการใช้ค่า RTT เมื่อเกิดการส่งข้อมูลซ้ำขึ้นเนื่องจากแพ็กเก็ตตอบรับเดินทางกลับมาไม่ทันการตอบรับที่เกิดขึ้นในภายหลังนั้นฝั่งส่งไม่มีทางทราบได้เลยว่าเป็นการตอบรับแพ็กเก็ตเดิม หรือแพ็กเก็ตที่ส่งซ้ำไป การเดาที่ผิดพลาดจะส่งผลไปยังค่า RTT ที่จะนำมาใช้ Phil Karn วิธีแก้ปัญหาคือ จะไม่ทำการปรับปรุงค่า RTT เมื่อเกิดการส่งแพ็กเก็ตซ้ำแต่ให้ใช้การเพิ่มค่าระยะการเวลารอคอยเป็น 2 เท่า จนกว่าแพ็กเก็ตจะสามารถส่งผ่านไปได้เป็นปกติ เรียกว่า อัลกอริทึมของ Karn

2.10.2 Persistence Timer

ใช้สำหรับการป้องกันปัญหาการขัดแย้งแบบ dead lock ซึ่งจะเกิดขึ้นเมื่อฝั่งรับส่งแพ็กเก็ตตอบรับพร้อมกับแจ้ง window size เป็น 0 เพื่อบอกให้ฝั่งส่งหยุดการส่งข้อมูลชั่วคราว เมื่อฝั่งรับพร้อมจึงแจ้งข้อมูล window size ไปใหม่ แต่แพ็กเก็ตนี้เกิดสูญหายไปทำให้ทั้งฝั่งส่งและฝั่งรับต่างรอซึ่งกันและกัน การจับเวลาแบบ persistence timer จึงถูกนำมาใช้สำหรับฝั่งส่งซึ่งจะทำการจับเวลาเมื่อถูกสั่งให้หยุดทำงานชั่วคราวถ้าฝั่งรับไม่ส่งข้อมูลใดๆมาภายในเวลารอคอยที่กำหนด ฝั่งส่งจะเป็นฝ่ายส่งแพ็กเก็ตข้อมูลพิเศษ (probe) ไปถามเอง ซึ่งฝั่งรับจะต้องตอบรับด้วยการแจ้ง window size มาใหม่(อาจจะยังคงเป็น 0 อีกก็ได้)

2.10.3 Keepalive Timer

ถูกนำมาใช้เมื่อไม่มีการสื่อสารเกิดขึ้นเป็นระยะเวลานาน เนื่องจากทั้งฝั่งส่งและฝั่งรับไม่มีข้อมูลที่จะแลกเปลี่ยนกัน แต่ก็ยังคงสภาพการเชื่อมต่อเอาไว้ เมื่อหมดระยะเวลารอคอย ฝั่งส่ง (หรือฝั่งรับก็ได้)

จะส่งแพ็กเก็ตออกไปเพื่อตรวจดูว่าอีกฝ่ายหนึ่งนั้นยังคงทำงานอยู่ตามปกติหรือไม่ ถ้าไม่มีการตอบรับกลับมาทางฝั่งที่ส่งแพ็กเก็ตออกไปถาม ก็จะยกเลิกการเชื่อมต่อในทันที กระบวนการนี้จะเป็นการเพิ่มปริมาณข้อมูลเข้าสู่ระบบเครือข่าย และอาจยกเลิกการเชื่อมต่อที่สมบูรณ์แต่เกิดปัญหาอันเนื่องมาจากเรื่องอื่นๆ ได้

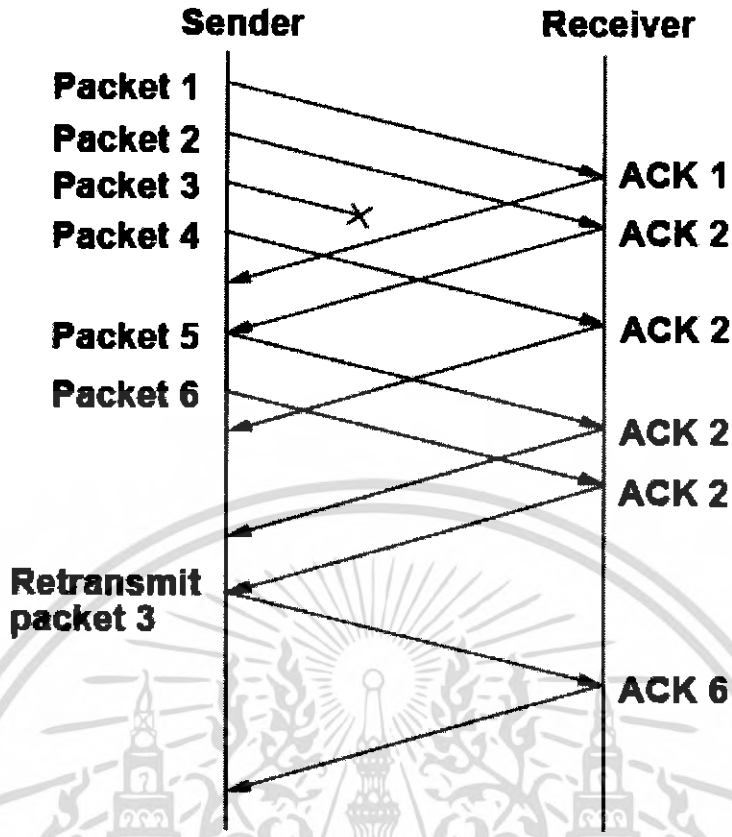
การจับเวลาแบบสุดท้ายเกิดขึ้นเมื่อเอ็นติตี้ TCP อยู่ในสถานะ TIMED WAIT เพื่อรอการยกเลิกการเชื่อมต่อ ซึ่งจะมีระยะเวลารอคอยเป็น 2 เท่าของอายุของแพ็กเก็ต (packet lifetime) เพื่อให้แน่ใจว่าจะไม่มีแพ็กเก็ตใดยังคงค้างอยู่ในระบบ เมื่อหมดระยะรอคอยนี้ การเชื่อมต่อจะถูกยกเลิกโดยสมบูรณ์

2.11 Fast Retransmit

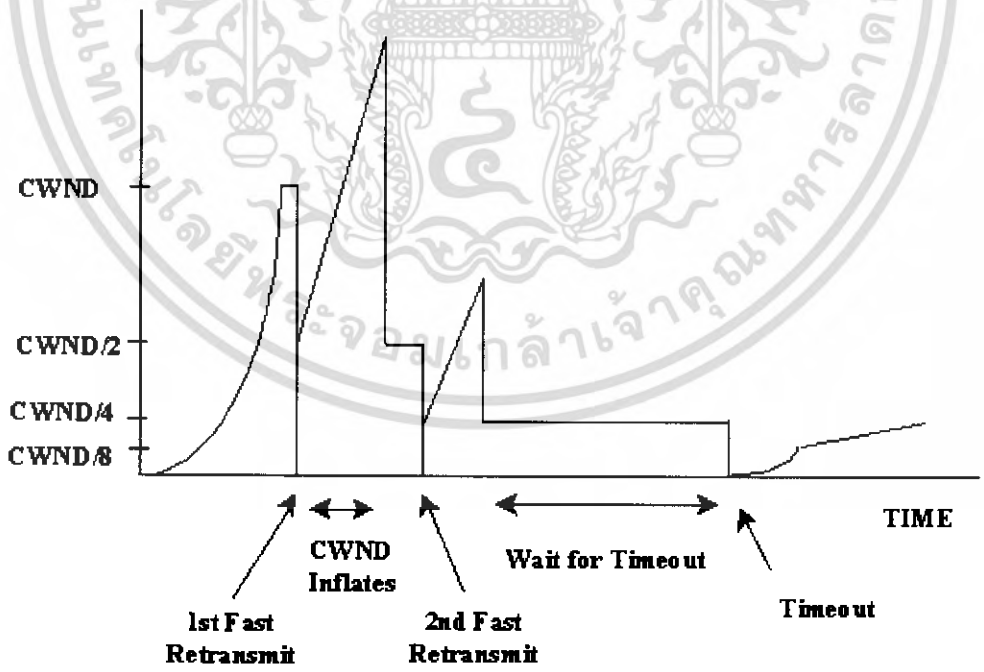
ปัญหาหนึ่งของ time out – triggered retransmission คือช่วงหมดระยะรอคอยที่ค่อนข้างยาวนาน เมื่อเซ็กเมนต์เกิดการสูญหาย ช่วงหมดระยะรอคอย ที่ยาวนานนี้เป็นผลให้ฝั่งส่งต้องดีเลย์การส่งกลับไปใหม่ เมื่อเกิดการสูญหายของแพ็กเก็ตจะเพิ่มดีเลย์ให้ end – to – end และที่ฝั่งส่งสามารถตรวจจับแพ็กเก็ตที่เกิดการสูญหายบ่อยๆ ก่อนจะเกิดเหตุการณ์หมดระยะรอคอยโดยจะเรียกว่า duplicate ACK duplicate ACK คือเซ็กเมนต์ของ ACK ที่ตอบกลับมาใหม่ซึ่งฝั่งส่งมีการรับการตอบกลับเข้ามาก่อนแล้ว เข้าใจว่าผลตอบสนองของฝั่งส่งที่ duplicate ACK เราต้องดูว่าทำไมที่ฝั่งรับ ส่ง duplicate ACK มาในตำแหน่งแรก ในตารางที่ 2.4 สรุปสั้นๆว่า TCP ฝั่งรับจะตอบ ACK เมื่อ TCP ฝั่งรับได้รับเซ็กเมนต์ที่มี sequence number ที่ใหญ่กว่า ลำดับของ sequence number มันจะตรวจจับช่องว่างในข้อมูลที่ถูกส่งอย่างต่อเนื่อง ซึ่งนั่นคือ เซ็กเมนต์ที่สูญหายไป ช่องว่างสามารถให้ผลลัพธ์ของการสูญหายหรือลำดับเซ็กเมนต์เข้าไปในเครือข่ายใหม่อีกครั้ง ตั้งแต่ TCP ไม่ได้ใช้ negative ACK ตามตารางที่ 2.3

เพราะฝั่งส่งจะส่ง sequence number ที่ใหญ่กลับไปกลับมาบ่อยๆ ซึ่งหนึ่งเซ็กเมนต์ที่มีการสูญหาย อาจจะเป็นไปได้ที่มีการส่ง duplicate ACK กลับไปกลับมาหลายครั้ง ถ้า TCP ฝั่งส่ง รับ duplicate ACK เข้ามา 3 อัน สำหรับที่ข้อมูลเหมือนกัน มันจะทำให้เห็นเซ็กเมนต์ต่อไปที่เซ็กเมนต์นั้นมี ACK 3 เวลาที่เกิดการสูญหาย ในกรณีที่รับ duplicate ACK เข้ามา 3 อัน TCP ฝั่งส่งจะใช้วิธี fast retransmit ในการส่งกลับไปใหม่ที่มีการสูญหายของเซ็กเมนต์ก่อนจะหมดเวลาของเซ็กเมนต์นั้น

จากคำอธิบายการทำงานข้างต้น สามารถแสดงให้เห็นได้ดังรูปที่ 2.23 และแสดงในรูปของกราฟได้ ดังรูปที่ 2.24 (จากความสัมพันธ์ระหว่าง cwnd และ เวลา)



รูปที่ 2.23 Fast Retransmit

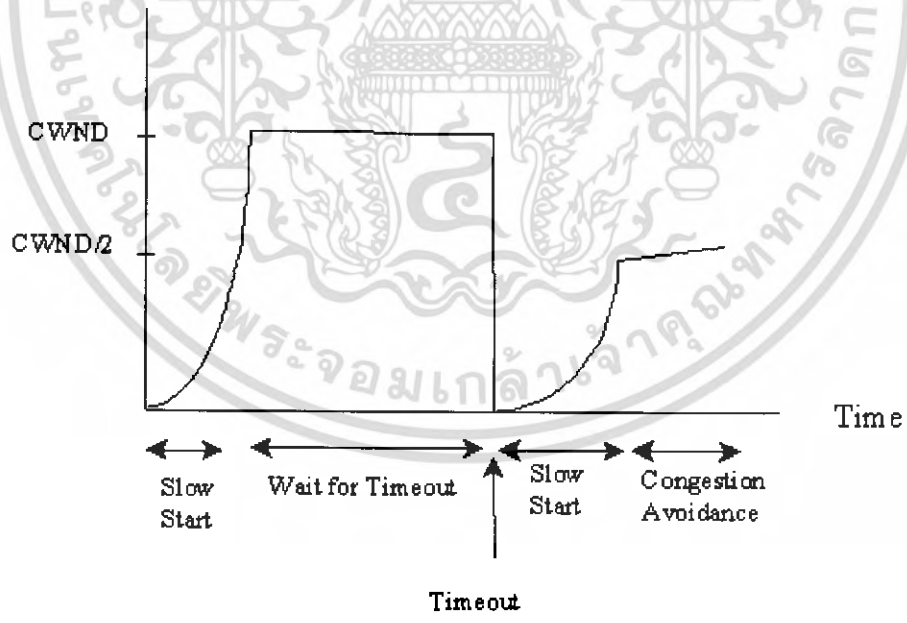


รูปที่ 2.24 กราฟของ Fast Retransmit

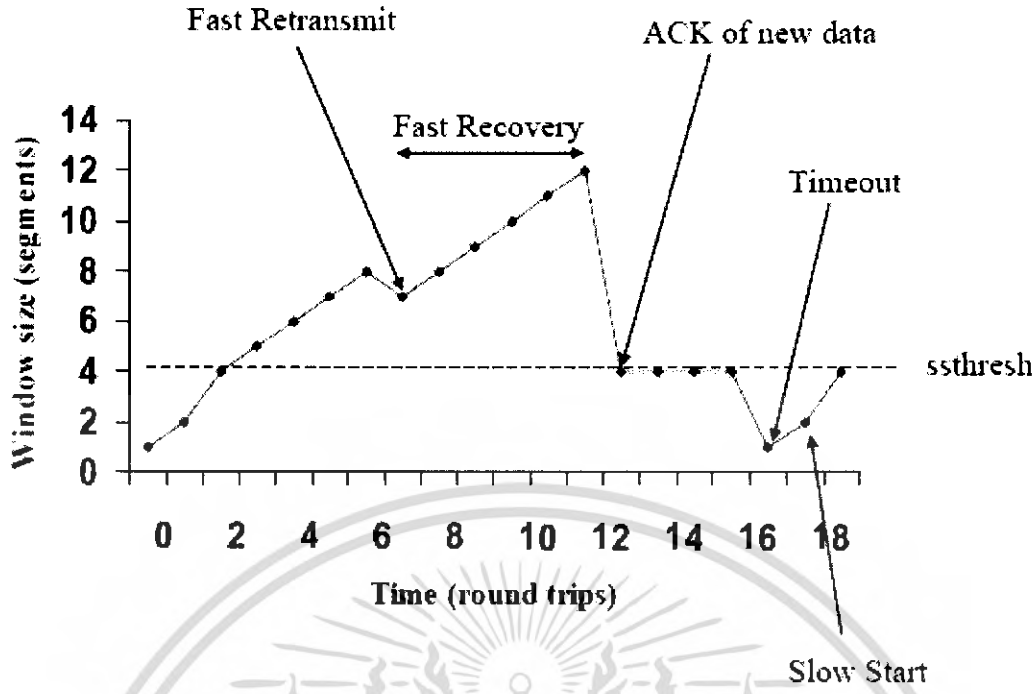
2.12 Slow Start

เมื่อ TCP เริ่มต้นการเชื่อมต่อ ค่าของ CongWin คือค่าเริ่มต้นที่ 1 MSS ผลลัพธ์ค่าเริ่มต้นของอัตราส่งหยาบๆคือ MSS/RTT เพราะการหาแบนด์วิดธ์ของการเชื่อมต่ออาจจะมากกว่า MSS/RTT มันอาจจะผิดพลาดจากการเพิ่มอัตราการส่งที่เป็นเชิงเส้นเท่านั้น และมีการรอคอยเป็นเวลาที่มากเกินไปก่อนที่อัตราการส่งจะลาดขึ้นไปอยู่ในระดับที่เหมาะสม ดังนั้นแทนการเพิ่มขึ้นของอัตราการส่งที่เป็นเชิงเส้นตลอดระยะเวลาที่ค่าเริ่มต้นเฟสของ TCP ฝั่งส่งจะเพิ่มอัตราการส่งแบบเอกซโปเนนเชียลจากส่วนของด้านบนสุดคือค่า CongWin ทั้งหมดของ RTT TCP ฝั่งส่งจะเพิ่มอัตราการส่งอย่างต่อเนื่องแบบเอกซโปเนนเชียล จนกระทั่งมีการสูญหาย ซึ่งเวลา CongWin เป็นการตัดในครึ่งหนึ่งและการเพิ่มขึ้นเป็นเชิงเส้น ดังนั้นระยะเวลาเริ่มต้นของเฟส ซึ่งเรียกว่า Slow Start (SS) ของ TCP ฝั่งส่งจะเริ่มจากทำการส่งอย่างช้าๆ แต่จะเพิ่มการส่งขึ้นแบบเอกซโปเนนเชียล ฝั่งส่งก็จะเพิ่มค่าของ CongWin เป็นแบบเอกซโปเนนเชียลจาก 1 MSS ของเวลาในการส่งเซ็กเมนต์ตอบกลับ TCP ฝั่งส่งจะส่งเซ็กเมนต์แรกไปในเครือข่ายและรอคอยการตอบกลับ ถ้านี้คือการตอบกลับก่อนเกิดการสูญหาย TCP ฝั่งส่งจะเพิ่มการคับคั่งของข้อมูลจาก 1 MSS และส่งเซ็กเมนต์ที่มีขนาดสูงสุดออกไป 2 เซ็กเมนต์ สำหรับแต่ละเซ็กเมนต์ที่ตอบกลับทำให้การคับคั่งของข้อมูลเท่ากับ 4 MSS และส่งเซ็กเมนต์ขนาดสูงสุดออกไป 4 เซ็กเมนต์ นี่คือวิธีการที่การตอบกลับมาถึงอย่างต่อเนื่อง จนกระทั่งสุดท้ายเกิดการสูญหายขึ้น ดังนั้นค่าสัมประสิทธิ์ของ CongWin คือ 2 เท่าของระยะเวลาทั้งหมดของ RTT Slow Start phase

จากการอธิบายข้างต้นนั้นสามารถแสดงได้ดังรูปกราฟในรูปที่ 2.25



รูปที่ 2.25 Slow Start



รูปที่ 2.26 Fast Retransmit, Fast Recovery, Slow Start, Time out

จากรูปที่ 2.26 เป็นการแสดงถึงรูปกราฟ จากความสัมพันธ์ระหว่าง ค่าของ window size และค่า round trip time ซึ่งในแต่ละช่วงนั้นจะเกิดปรากฏการณ์ต่างๆ เกิดขึ้น

2.13 ทฤษฎีและหลักการทำงานของคิว (Queue)

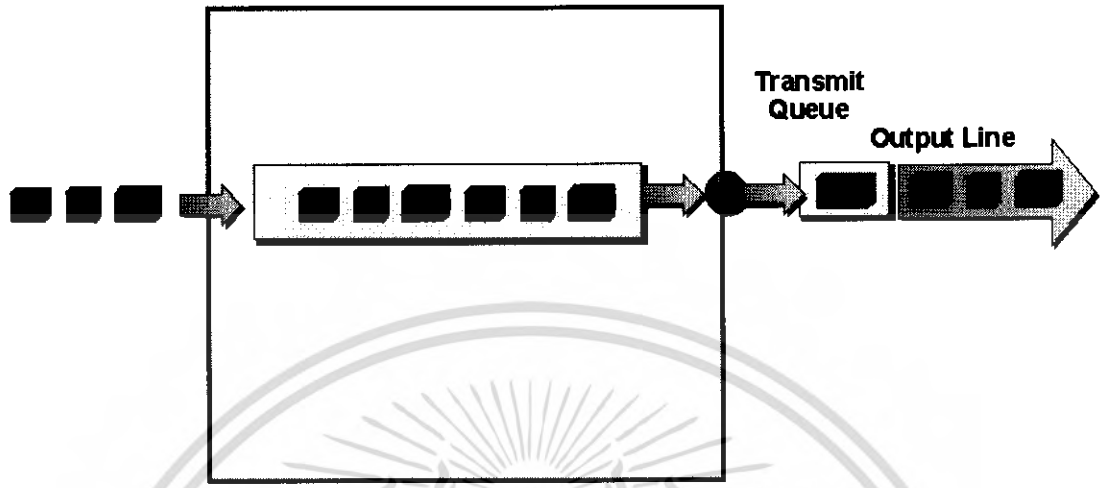
2.13.1 First In First Out (FIFO) และ Drop Tail

FIFO Queuing เป็นกลไกการทำงานพื้นฐาน กลไกในการทำงานนั้นแพ็กเก็ตจะถูกนำเข้ามาอย่างต่อเนื่องภายในระบบเครือข่าย ซึ่งภายในนี้จะมีคิวเดียวเท่านั้นที่เป็นคิวเดี่ยว โดยจะเข้ามาทางท้ายคิว และเคลื่อนย้ายออกไปทางส่วนหัวของคิว

FIFO เป็นวิธีการดั้งเดิมที่ใช้กันอย่างแพร่หลายในอุปกรณ์เครือข่ายอย่างง่าย แต่ยังไม่ใช่เป็นการทำงานที่ดีในการที่จะให้บริการระบบที่มีความหนาแน่นของปริมาณทราฟฟิกสูงๆ เนื่องจากไม่มีความยืดหยุ่นตามมาตรฐาน QoS ซึ่งต้องการการให้บริการเฉพาะอย่างที่แตกต่างออกไป

หลักการทำงานของ FIFO แสดงดังรูปที่ 2.27

FIFO



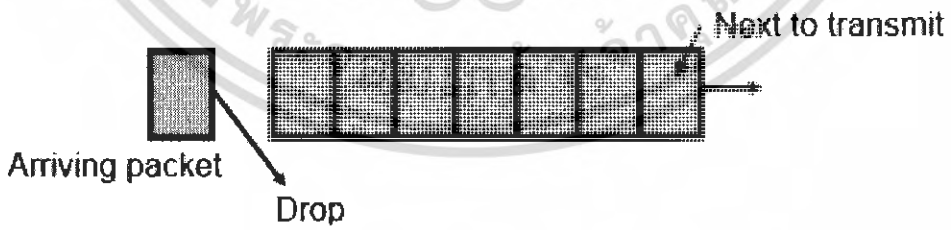
รูปที่ 2.27 หลักการทำงานของ FIFO Queuing

FIFO Queuing มีลักษณะดังนี้

1. จะมีความเร็วสูงสุด และมีความง่ายที่สุด เนื่องจากแพ็กเก็ตที่เข้ามาในระบบจะได้รับการบริการในทันที

2. ไม่มีการแยกแยะชนิดของข้อมูลที่เข้ามาว่ามีความแตกต่างกันอย่างไร

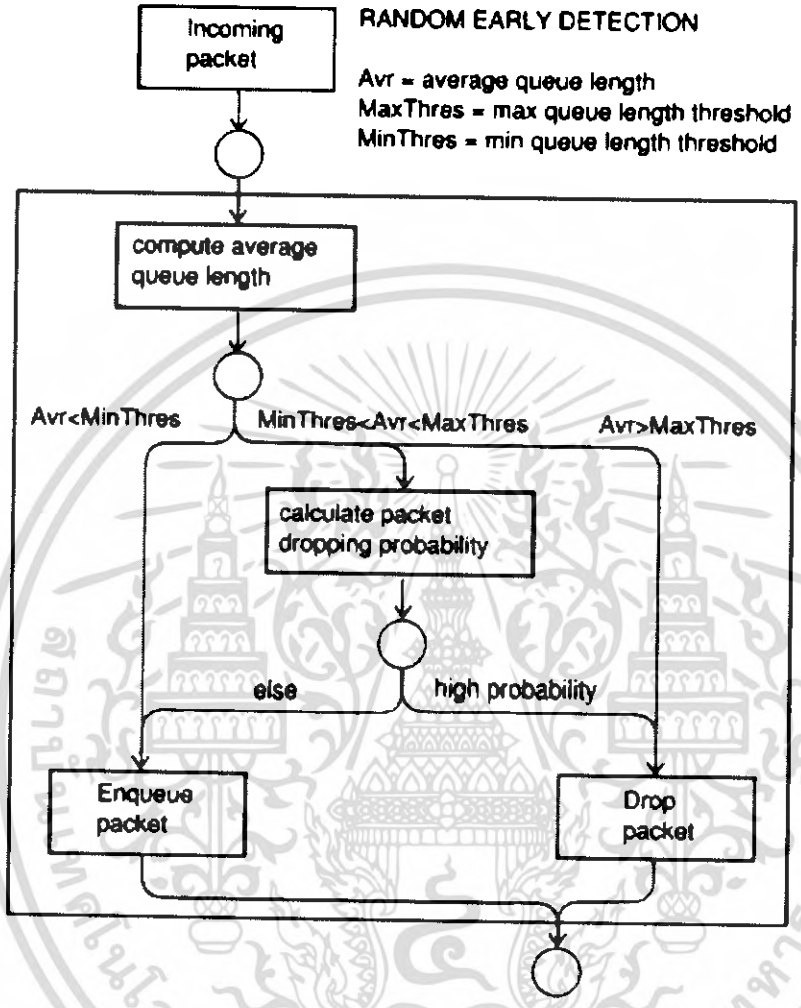
3. มีวิธีการจัดการบัฟเฟอร์แบบ DropTail ซึ่งการทำงานของวิธีการนี้คือ เมื่อมีแพ็กเก็ตของข้อมูลไหลเข้ามาในคิวของระบบจนเต็ม แพ็กเก็ตต่างๆที่เหลือจะถูกดรอปไม่สามารถเข้ามาในระบบได้ ทำให้มีปัญหาในเรื่องการสูญเสียของแพ็กเก็ตมาก ดังแสดงในรูปที่ 2.28



รูปที่ 2.28 แสดงการทำงานของ Drop Tail

2.13.2 Random Early Detection (RED) Queuing

วิธีการนี้ถูกออกแบบมาสำหรับปรับปรุงประสิทธิภาพของระบบเครือข่ายให้ดีขึ้น โดยการปรับปรุงวิธีการทำงานซึ่งทำให้แตกต่างจาก FIFO ดังแสดงในรูปที่ 2.29



รูปที่ 2.29 ขั้นตอนการทำงานของ RED Queuing

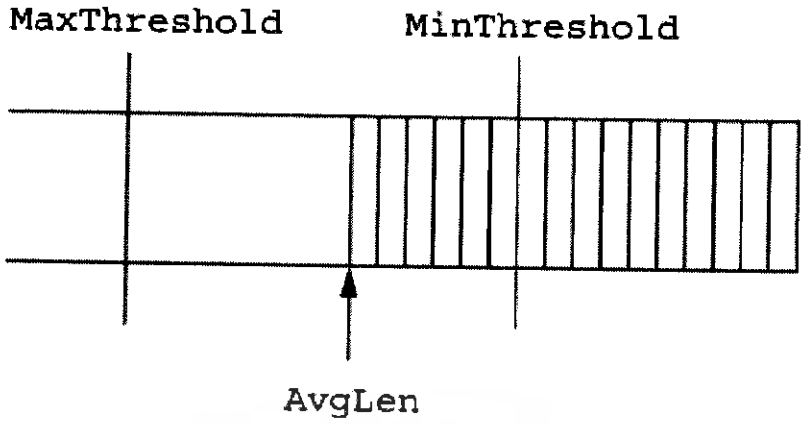
1. ตรวจจับความหนาแน่นในระบบเครือข่ายเบื้องต้น

2. อนุญาตให้มีความหนาแน่นของปริมาณทราฟฟิกสูงได้ชั่วคราวโดยใช้หลักการดูดซับ (absorbing) ความหนาแน่นของแพ็กเก็ตข้อมูลเอาไว้

3. ป้องกันการเกิดความหนาแน่นเพิ่มขึ้นมาอีก

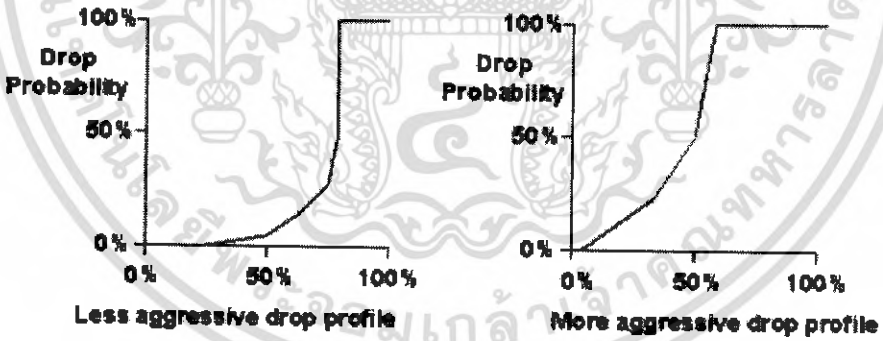
4. หลีกเลี่ยงการเกิดความหนาแน่นขึ้นพร้อมๆกัน ของการเชื่อมต่อต่างๆ โดยใช้การเลือกสุ่มการเชื่อมต่อหรือการไหลของแพ็กเก็ตที่ถูกทำเครื่องหมายไว้

โดยที่ RED นั้น จะทำเครื่องหมายหรือสัญลักษณ์ลงบนความยาวเฉลี่ยของคิวกับจุดจำกัดต่ำสุดและสูงสุด (minimum and maximum threshold) โดยค่าของความยาวเฉลี่ยนี้ จะใช้การคำนวณจากทุกช่วงเวลาที่มีแพ็กเก็ตเข้าออกจากคิว ซึ่งก็จะทำให้ได้ค่าของความยาวเฉลี่ยของคิว ดังรูปที่ 2.30



รูปที่ 2.30 การหาความยาวเฉลี่ยของคิว

ซึ่งถ้าความยาวเฉลี่ยของคิวมีค่าต่ำกว่าจุดจำกัดต่ำสุด แพ็กเก็ตต่างๆ ก็จะไหลเข้าไปในคิวโดยที่จะยังไม่มีการทำเครื่องหมายลงบนแพ็กเก็ตเหล่านี้ แต่เมื่อมีความเป็นไปได้ในการที่จะมีปริมาณของแพ็กเก็ตเพิ่มสูงขึ้นจนทำให้ค่าความยาวเฉลี่ยของคิวเพิ่มสูงขึ้นจนเกินระดับจุดจำกัดต่ำสุด จะมีการทำเครื่องหมายไว้บนแพ็กเก็ตบางส่วน และจะหยุดแพ็กเก็ตบางส่วนเหล่านี้ไว้ เพื่อที่จะทำให้สามารถรองรับปริมาณทราฟฟิกภายในคิวนี้ได้หรืออาจเรียกการทำงานแบบนี้ว่าเป็นการดูดซับความหนาแน่น แต่เมื่อมีจำนวนของแพ็กเก็ตที่ไหลเข้ามาสูงมากจนทำให้ความยาวเฉลี่ยของคิวนี้สูงเกินจุดจำกัดสูงสุด แพ็กเก็ตทั้งหมดจะถูกหยุดไว้และระบบจะไม่สามารถใช้งานได้ ซึ่งสามารถแสดงได้ดังรูปที่ 2.31



รูปที่ 2.31 กราฟแสดงการทำงานของ RED

กลไกการทำงานของ RED แบ่งออกเป็น 2 ขั้นตอน คือ

1. RED จะทำการคำนวณหาค่าความยาวเฉลี่ยของคิวจากจุดจำกัดต่ำสุดและสูงสุดนี้ในทันทีที่เริ่มทำงาน ซึ่งจะขึ้นอยู่กับการตั้งค่าของอุปกรณ์ เช่น เกตเวย์ เพื่อรองรับทราฟฟิก เป็นต้น

2. RED จะทำการสุ่มทำเครื่องหมายที่แพ็กเก็ตที่จะทำการหยุด ซึ่งจะให้มีความถี่ใดๆที่จะทำการหยุดแพ็กเก็ต

จากรูปที่ 2.31 จำนวนของแพ็กเก็ตที่จะถูกหยุดจะสามารถเขียนได้เป็นความน่าจะเป็นที่จะหยุดจำนวนของแพ็กเก็ตเท่าใดจากจำนวนแพ็กเก็ตทั้งหมด (discard probability) ซึ่งสามารถเขียนได้ดังสมการที่ (2.7)

$$P_B = \min_p \frac{(avg - \min_{th})}{(\max_{th} - \min_{th})} \quad (2.7)$$

ทั้งนี้ถ้าความหนาแน่นของปริมาณทราฟฟิกยังคงมีอยู่ต่อไป RED ก็จะทำการสุ่มเพื่อหยุด แพ็กเก็ตเพิ่มขึ้นเรื่อยๆ จนกว่าจะมีความหนาแน่นที่ลดลง ซึ่งความถี่ในการที่จะสุ่มทำเครื่องหมายกับแพ็กเก็ตต่างๆที่เพิ่มขึ้นอยู่กับความยาวเฉลี่ยของคิวที่จะเข้าถึงจุดจำกัดสูงสุดนั่นเอง

RED สามารถกำหนดให้ค่าความน่าจะเป็นในการทิ้งดาต้าแกรม ในขณะที่คิวเต็มได้โดยปราศจากการทิ้งดาต้าแกรมได้โดยการใช้เทคนิคที่ลอกเลียนแบบมาจาก TCP โดยแทนที่จะใช้ขนาดของคิวที่แท้จริง RED จะคำนวณค่าขนาดของคิวเฉลี่ย (average queue size) หรือ avg และใช้ขนาดเฉลี่ยไปกำหนดความน่าจะเป็น ค่า avg คือ ค่าเฉลี่ยที่มีช่วงกว้างเป็นเอกซโปเนนเชียลเวลาที่ดาต้าแกรมมาถึงจะแสดงได้ดังสมการ (2.8)

$$avg = (1-\gamma) * Old\_avg + \gamma * Current\_queue\_size \quad (2.8)$$

γ คือ ค่าระหว่าง 0 และ 1 ถ้า γ น้อย ค่าเฉลี่ยก็มีแนวโน้มที่จะน้อยตามด้วย ซึ่งจะเห็นได้ว่าสมการ (2.8) นั้น จะมี γ เป็นตัวกำหนด

2.14 ชนิดของ TCP

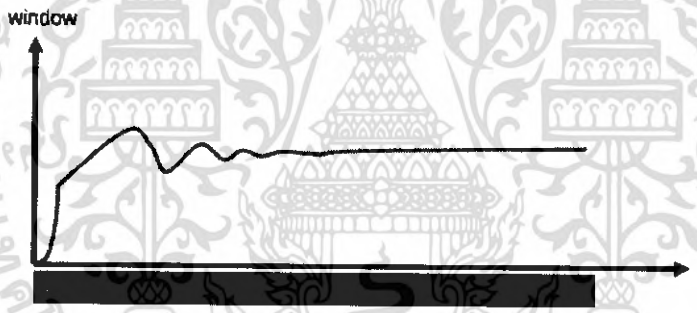
2.14.1 Vegas TCP

Vegas TCP ถูกเสนอโดย Brakmo มีอัลกอริทึมในการควบคุมความคับคั่งที่แตกต่างกับ New Reno มาก Vegas TCP นั้นในการควบคุมโดยทั่วไปอัตราการไหลของเซ็กเมนต์ของมันได้มาจากการประมาณจากแบนด์วิดธ์ที่ใช้ในเครือข่าย ในระหว่างที่มีการพัฒนารูปแบบใหม่ๆภายใน Vegas TCP ขึ้นมาจำนวนมาก สิ่งสำคัญที่ทำให้ Vegas TCP มีความแตกต่างไปจาก Reno TCP คือ การวางแผนในการประมาณค่าแบนด์วิดธ์ของมัน การศึกษา Vegas TCP แสดงให้เห็นว่า Vegas TCP มีประสิทธิภาพที่สูงกว่า Reno TCP เพราะมีการ retransmission แพ็กเก็ตน้อยกว่าและไม่มีอิทธิพลกับการเชื่อมต่อที่เกิดขึ้นใหม่ กับ RTT ที่ยาวนานกว่า ในขั้นแรก Vegas TCP จะอ่านและบันทึกคล็อกของระบบในแต่ละส่วนที่ถูกส่ง เมื่อ ACK มาถึง Vegas TCP จะอ่านค่าคล็อกอีกครั้งและทำการคำนวณแบบ RTT Vegas TCP จะใช้ค่า RTT ในการตัดสินใจว่าจะทำการ retransmit มี 2 กรณีคือ

1. เมื่อได้รับ ACK ซ้ำ Vegas TCP จะตรวจสอบความแตกต่างระหว่างค่าปัจจุบัน และบันทึกค่า timestamp เพื่อให้ส่วนที่สัมพันธ์กันมีความสำคัญกว่าค่าไม่ทันระยะเวลารอคอย ถ้าเป็นดังนี้ Vegas TCP จะทำการ retransmit ในส่วนที่ไม่ต้องรอให้มีการส่ง ACK ซ้ำกันถึง 3 ครั้ง

2. เมื่อไม่มีการรับ ACK ซ้ำ ถ้า ACK ที่ถูกส่งมาอันแรกหรืออันที่ 2 หลังจากการ retransmit Vegas TCP จะตรวจสอบ ถ้าเวลาในช่วงตั้งแต่ส่วนที่ถูกส่งให้มีขนาดใหญ่กว่าเวลาที่ไม่ทันระยะเวลารอคอย ถ้าเป็นดังนี้ Vegas TCP จะทำการ retransmit จะทำให้ส่วนอื่นๆที่สูญเสียก่อนหน้าที่จะมีการ retransmit โดยที่ไม่ต้องรอ ACK ที่ซ้ำกันถึง 3 ครั้ง

Vegas TCP จะมีการส่ง window size ที่เปลี่ยนแปลงตลอดซึ่งจะเกิดจากการวัดค่า RTT ในขณะที่ Reno TCP จะเพิ่ม window size ของมันอย่างต่อเนื่องจนกระทั่งมีการพบแพ็กเก็ตที่สูญหายและ Reno TCP จะตรวจสอบแพ็กเก็ตที่สูญหาย แสดงให้เห็นว่าเครือข่ายเกิดความคับคั่ง Vegas TCP จะใช้การประมาณค่าแบนด์วิดธ์ที่สลับซับซ้อนคือมันจะใช้ความแตกต่างระหว่างอัตราการไหลที่คาดหวังและอัตราการไหลที่ได้มา เพื่อนำไปประมาณค่าแบนด์วิดธ์ในเครือข่าย แนวคิดนี้นั้นเมื่อค่าทรูพุตที่ได้จากการคาดหวัง และค่าทรูพุต มีค่าเท่ากันเกือบทั้งหมด เครือข่ายจะไม่มีความคับคั่ง ในกรณีที่มีความคับคั่งค่าทรูพุตที่เป็นจริงจะน้อยกว่าค่าทรูพุตที่ได้มาจากการคาดหวังมากๆ



รูปที่ 2.32 กราฟแสดงการทำงานของ Vegas TCP

กลไกของ Vegas TCP

$$1.\ expected\_rate = \frac{cwnd(t)}{base\_RTT} \tag{2.9}$$

โดยที่ $cwnd(t)$ คือขนาดของความคับคั่งวินโดว์ ปัจจุบันและ $base\_rtt$ คือ RTT ต่ำสุดของการเชื่อมต่อ

$$2.\ actual\_rate = \frac{cwnd(t)}{RTT} \tag{2.10}$$

โดยที่ RTT คือ round – trip time ที่มีอยู่

3. การประมาณของฝั่งส่งมีการสำรองไว้ในคิวของเราเตอร์จากค่าความแตกต่างหรือ $diff$

$$diff = \exp ected\_rate - actual\_rate \tag{2.11}$$

4. ใช้ค่า $diff$ นี้และค่าความคับคั่งของวินโดว์ หรือ cwnd ซึ่งคือการปรับค่าใช้เหมาะสมดังนี้

$$cwnd = \begin{cases} cwnd + 1 & if\ diff < \alpha \\ cwnd - 1 & if\ diff > \beta \\ cwnd & e.w. \end{cases} \tag{2.12}$$

อัลกอริทึมจะมีผลเมื่อค่าทรูพุตที่คาดหวัง และค่าทรูพุตจริงนั้นใกล้เคียงกัน การเชื่อมต่ออาจจะไม่ใช้แบนด์วิดธ์ที่หาได้จากเครือข่าย ดังนั้นควรจะเพิ่มอัตราการไหล ในกรณีอื่นๆ เมื่อค่าทรูพุตจริงนั้นน้อยกว่าค่าทรูพุตที่คาดหวังมากๆ เครือข่ายจะมีความน่าจะเป็นที่จะมีความคับคั่งและจากนั้นการเชื่อมต่อควรจะลดอัตราการไหล ดังรูปที่ 2.32 จะแสดงค่าทรูพุตของ Vegas TCP

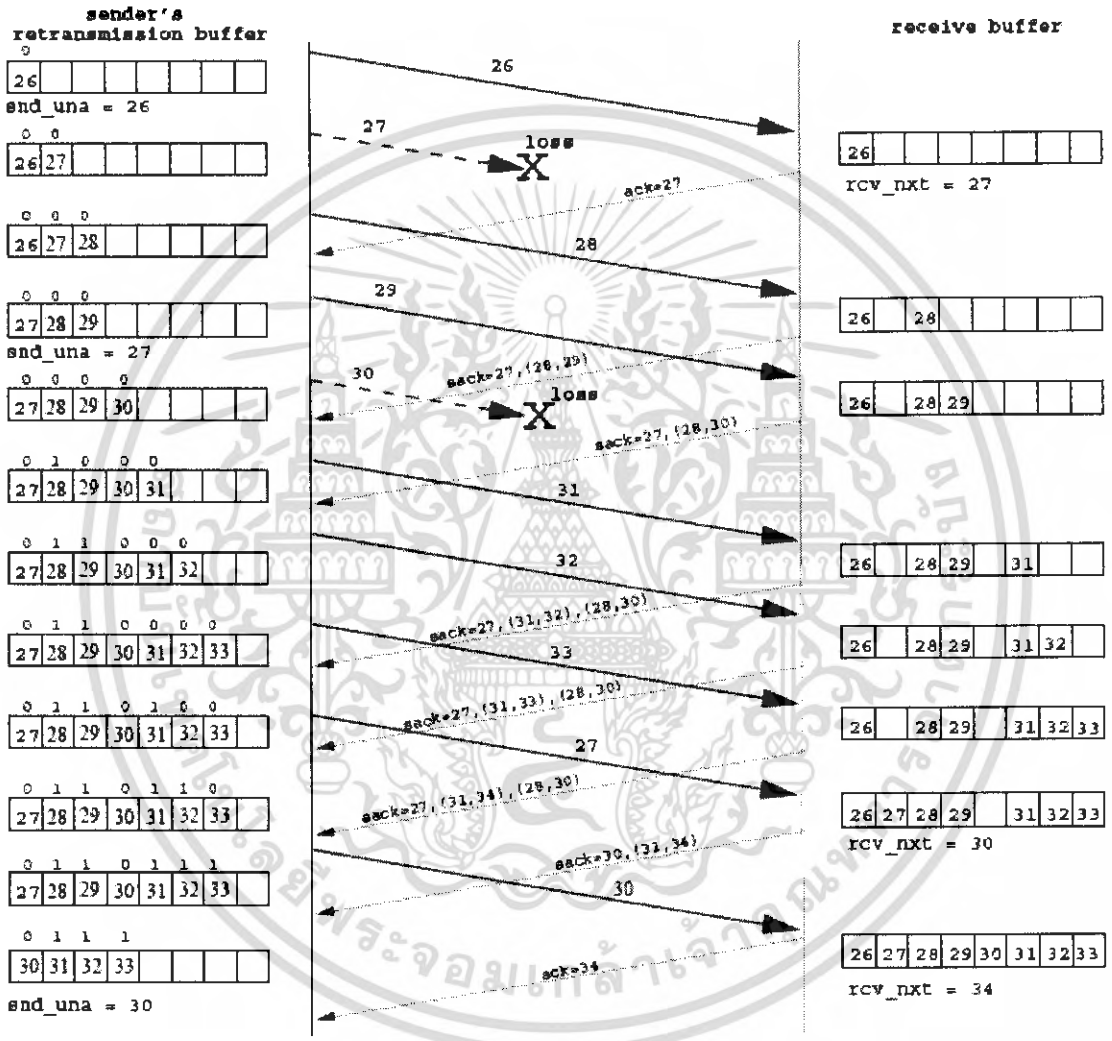
2.14.2 SACK TCP (Selective Acknowledgment)

New Reno TCP คือการดำเนินการของ TCP ปกติในระบบที่สำคัญ ในการดำเนินการนี้ผู้ส่งสามารถ recover แพ็กเก็ตที่สูญหายไปได้โดยใช้อัลกอริทึม fast retransmit และ fast recovery อย่างไรก็ตามเมื่อแพ็กเก็ตจำนวนมากสูญหายจาก single window ฝั่งส่งจะหยุดแต่ละแพ็กเก็ตที่กำลังทำการ retransmit ที่มีความสำเร็จในการรับ คือ 1 แพ็กเก็ตต่อ RTT

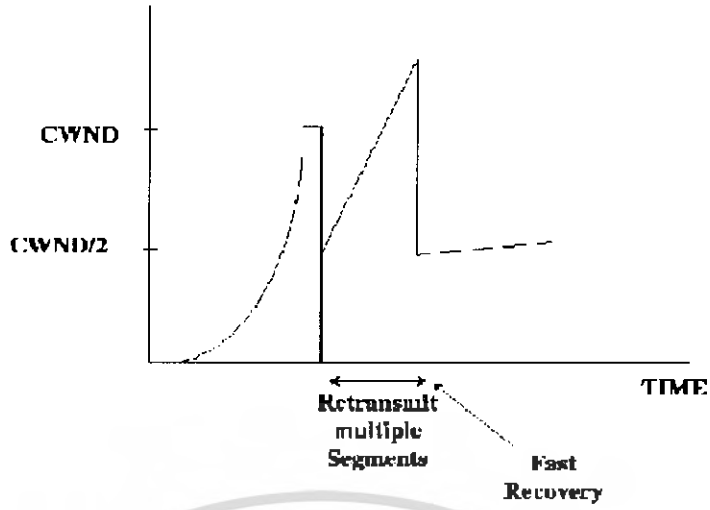
SACK TCP option ถูกเสนอใน RFC 2018 และ RFC 2883 ซึ่งได้ขจัดข้อจำกัดนี้ ใน SACK TCP ฝั่งรับสามารถแจ้งฝั่งส่งเกี่ยวกับแพ็กเก็ตที่รับเข้า ดังนั้นจะยินยอมให้ฝั่งส่งเลือกการ retransmit แพ็กเก็ตสูญหายอย่างเดียว

จากรูปที่ 2.33 SACK TCP จะเริ่มต้นส่งแพ็กเก็ตออกไปแบบ Slow Start คือส่งออกไปอย่างช้าๆ และเมื่อได้รับ ACK ตอบกลับมาก็จะทำการเพิ่มจำนวนแพ็กเก็ตที่ใช้ในการส่งมากขึ้นเรื่อยๆ จนกระทั่งเกิดความคับคั่งขึ้นที่เราเตอร์ ทำให้ข้อมูลบางส่วนเกิดการสูญหาย SACK TCP จะใช้การ fast recovery เมื่อฝั่งส่งข้อมูลได้รับ ACK ของข้อมูลทั้งหมด จะทำการ retransmit แพ็กเก็ตและตัดความคับคั่งของวินโดว์ลงครึ่งเท่า ระหว่างการ fast recovery SACK TCP จะปรับค่าที่เรียกว่า pipe ซึ่งเป็นค่าที่แสดงถึงจำนวนของแพ็กเก็ตที่ค้างอยู่ในเส้นทาง ฝั่งส่งจะส่งข้อมูลใหม่หรือข้อมูลเดิมเมื่อจำนวนแพ็กเก็ตในเส้นทางนั้นไม่น้อยกว่าค่าความคับคั่งของวินโดว์ ค่าของ pipe จะเพิ่มขึ้นเมื่อฝั่งส่งส่งแพ็กเก็ตใหม่หรือแพ็กเก็ตเดิมซ้ำ และจะลดลงเมื่อฝั่งส่งได้รับแพ็กเก็ตของ ACK ที่มี SACK TCP ซ้ำๆกัน pipe จะมีการใช้งานเมื่อมีการตัดสินใจส่งแต่ละแพ็กเก็ต ฝั่งส่งจะจำ ACK ที่ได้จาก SACK TCP ก่อนหน้านี้ เมื่อฝั่งส่งยอมให้ส่งแพ็กเก็ต มันจะทำการ retransmit แพ็กเก็ตต่อไป

SACK TCP จะป้องกันการดรอปของช่วงหมดระยะรอคอยในการ retransmit โดย จะส่งแพ็กเก็ตที่ถูกดรอป และทำการ Slow Start ฝั่งส่งจะมีการตรวจจับ ACK ซึ่งจะลด pipe ลง โดยใช้ 2 แพ็กเก็ต เมื่อเริ่มการ fast retransmit ในกรณีที่แพ็กเก็ตถูกดรอปลง pipe จะมีค่าลดลง และจะเพิ่มขึ้นเมื่อแพ็กเก็ตมีการ retransmit อย่างไรก็ตาม pipe จะถูกเพิ่มขึ้นเมื่อมีการส่งแพ็กเก็ตซ้ำเข้าไปใน pipe แต่จะไม่ลดสำหรับแพ็กเก็ตที่ถูกดรอป จากการทำงานของ SACK TCP ทั้งหมดที่ได้อธิบายมาสามารถแสดงในรูปของกราฟได้ดังรูปที่ 2.34

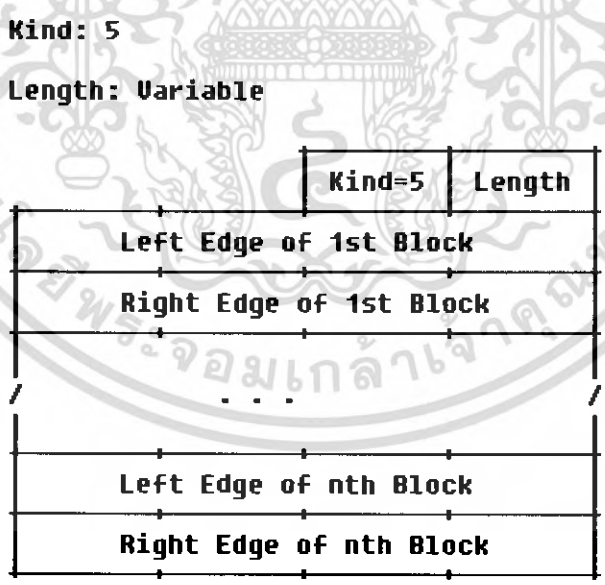


รูปที่ 2.33 การทำงานของ SACK TCP



รูปที่ 2.34 กราฟแสดงการทำงานของ SACK TCP

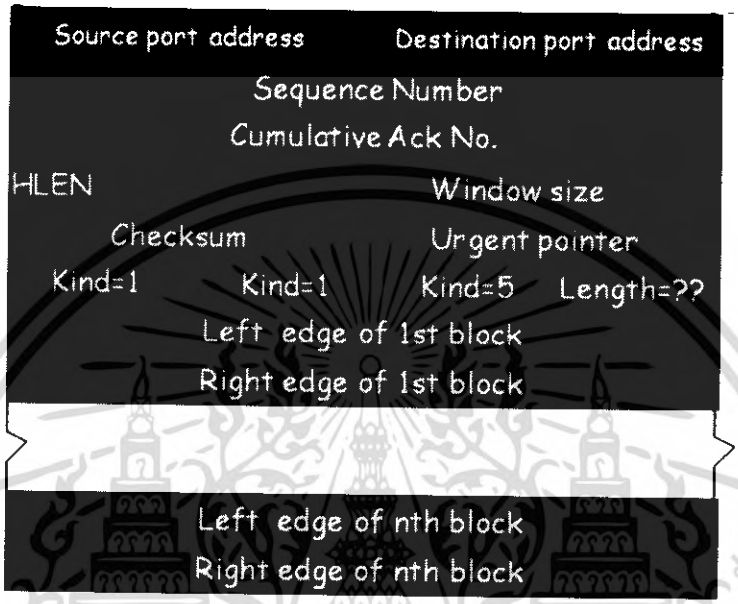
SACK TCP มีการแบ่งใช้ TCP option ออกเป็น 2 ส่วน คือ ออพชั่นแรกเรียกว่า "SACK Permitted" ใช้ทำให้ SACK TCP สามารถเข้าไปในการเชื่อมต่อ ซึ่งจะเป็นการกำหนดไปในทิศทางเดียวกันกับ SYN flag และในส่วนของออพชั่น อื่นๆ จะถูกใช้เป็น SACK option ของตัวมันเอง ซึ่งสามารถส่งในระหว่างที่ SACK TCP ทำให้ TCP เชื่อมต่อได้ โครงสร้างของ SACK option format แสดงในรูปที่ 2.35 ในขณะที่รูปที่ 2.36 แสดงถึง SACK – permitted option



รูปที่ 2.35 รูปแบบของ SACK (Format SACK)

SACK option ส่งโดยฝั่งรับ เพื่อเป็นการแจ้งฝั่งส่งเกี่ยวกับการรับ block ของข้อมูลที่ไม่ติดกัน ใน SACK TCP นั้นหมายถึง 32 บิต 2 จำนวนที่ไม่มีเครื่องหมายจาก left edge (sequence number แรกของ

block) และ right edge ของ block (sequence number ที่ไหลอย่างรวดเร็ว และ sequence number สุดท้าย ของ block) SACK block แรกมีคุณสมบัติคือ block ที่อยู่ติดกันนั้นจะถูกกระตุ้นด้วย ACK ส่วนที่เหลือ ของ SACK block จะถูกเติมด้วยการรายงานซ้ำเกี่ยวกับ SACK block ที่เพิ่งผ่านมาโดยส่วนใหญ่ (SACK block แรกที่เกิดขึ้นมาก่อน SACK option) ซึ่งไม่มีรูปแบบการรวมกลุ่มของ SACK block ใดๆ แล้ว ซึ่ง จะรวมอยู่ใน SACK option ปัจจุบัน

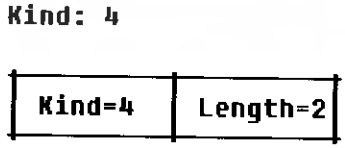


รูปที่ 2.36 รูปแบบของ SACK (Format SACK) เมื่อดูจาก TCP เฮดเดอร์

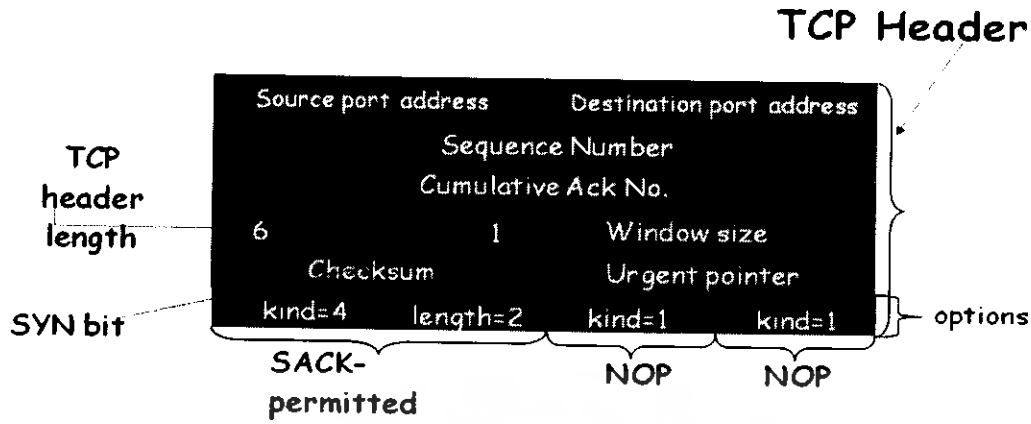
จากรูปที่ 2.36 นั้น จะแสดงถึง รูปแบบของ SACK เมื่อพิจารณาจาก TCP เฮดเดอร์

SACK option จะมีความยาว $8 * n + 2$ โดยที่ n คือจำนวนของ SACK block ที่ถูกรวมอยู่ ดังนั้นถ้าไม่มี TCP option อื่นๆ ทำงาน 40 ไบต์ ของ TCP เฮดเดอร์สามารถจัดให้มี SACK block มากที่สุด คือ 4 SACK block

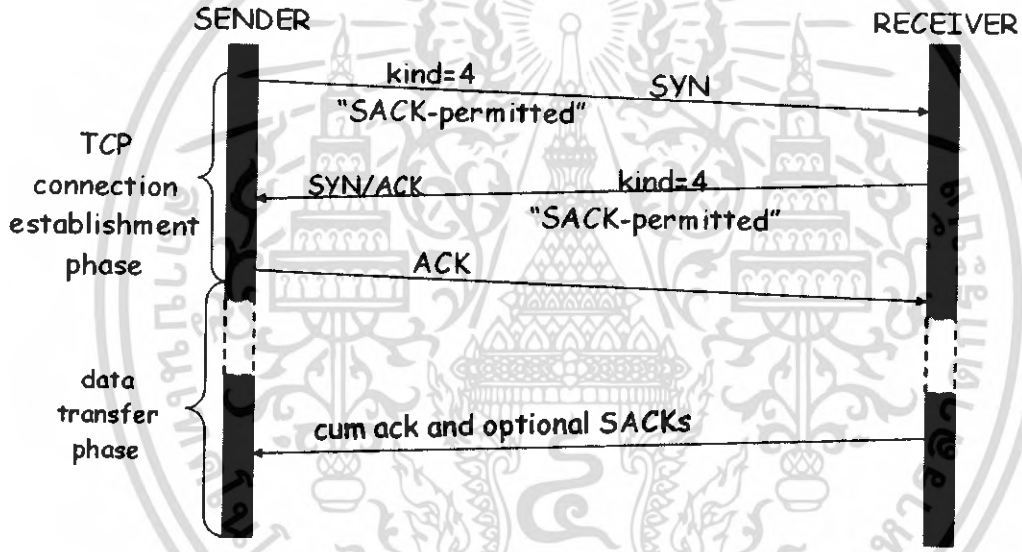
SACK Permitted Option มีรูปแบบดังรูปที่ 2.37 และเมื่อพิจารณาจาก TCP เฮดเดอร์จะเป็นไป ตามรูปที่ 2.38 และในการทำงานของ SACK Permitted Option นั้น สามารถแสดงได้ดังรูปที่ 2.39



รูปที่ 2.37 SACK TCP – Permitted Option



รูปที่ 2.38 SACK TCP – Permitted Option เมื่อพิจารณาจาก TCP เฮดเดอร์



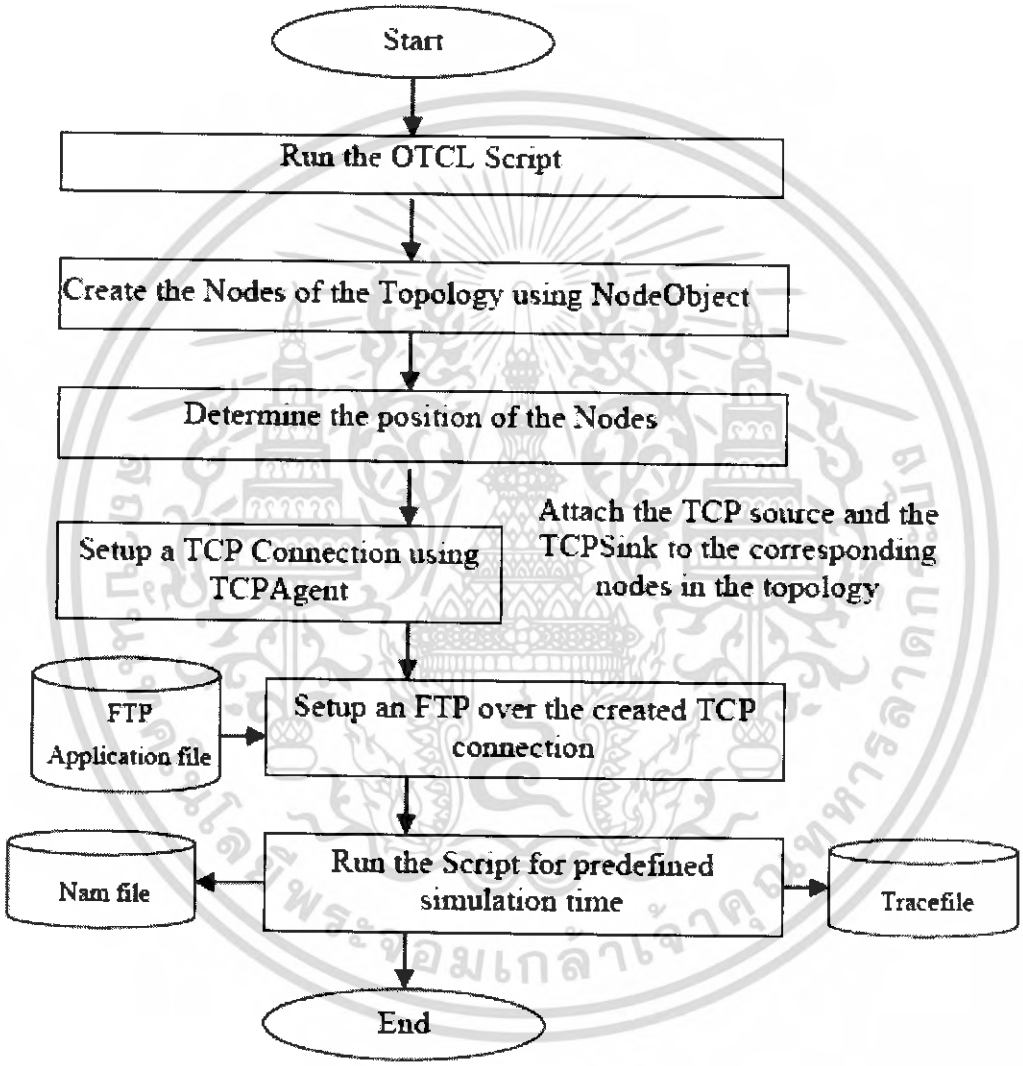
รูปที่ 2.39 แสดงการทำงานของ SACK TCP – Permitted Option

บทที่ 3

การทดลองและผลการทดลอง

3.1 ลำดับขั้นตอนในการทดสอบประสิทธิภาพ TCP

ในการทดสอบประสิทธิภาพ TCP ทั้ง Vegas TCP และ SACK TCP นั้นจะใช้โปรแกรม NS ในการทดสอบ ซึ่งจะมีขั้นตอนในการทำดังรูปที่ 3.1

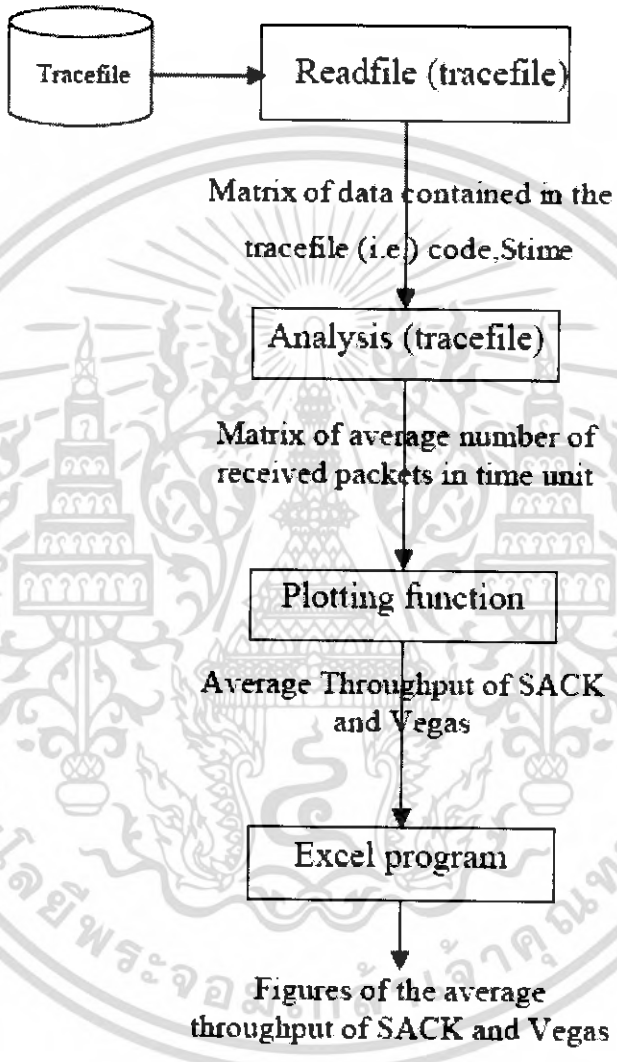


รูปที่ 3.1 ลำดับขั้นตอนในการทดสอบประสิทธิภาพ Vegas TCP และ SACK TCP

จากรูปที่ 3.1 นั้นแสดงถึงขั้นตอนในการทดสอบประสิทธิภาพของ Vegas TCP และ SACK TCP โดยจะเริ่มจากการเขียน OTcl Script ให้ถูกต้องตามหลักการเขียน ต่อจากนั้นให้ทำการสร้างโทโปโลยีและสร้างโหนด และจากนั้นทำการกำหนดตำแหน่งของโหนดว่าจะให้โหนดใดเป็นฝั่งส่ง โหนดใดเป็นฝั่งรับ และโหนดใดที่ทำหน้าที่เป็นเราเตอร์ และให้ทำการกำหนดทิศทางการเชื่อมต่อให้กับ TCP โดยใช้

TCP Agent จากนั้นให้ทำการกำหนด FTP ให้กับ TCP ที่ได้สร้างการเชื่อมต่อแล้ว เมื่อเสร็จเรียบร้อยแล้ว ให้นำโปรแกรมที่เขียนมาประมวลผล ซึ่งสามารถอ่านค่าที่จะนำมาวิเคราะห์และเปรียบเทียบได้จาก Trace File

3.2 ลำดับขั้นตอนในการวิเคราะห์และเปรียบเทียบประสิทธิภาพของ Vegas TCP และ SACK TCP



รูปที่ 3.2 ลำดับขั้นตอนในการวิเคราะห์และเปรียบเทียบประสิทธิภาพของ Vegas TCP และ SACK TCP

จากรูปที่ 3.2 นั้น จะแสดงถึงลำดับขั้นตอนในการวิเคราะห์ และเปรียบเทียบประสิทธิภาพ Vegas TCP และ SACK TCP โดยในขั้นแรกนั้นจะทำการอ่าน Trace File ที่ได้จากการทดสอบจากโปรแกรม NS จากนั้นจะทำการวิเคราะห์ถึงค่าเฉลี่ยของจำนวนแพ็กเก็ตที่รับได้ในเวลาที่กำหนด และนำค่าที่แสดงถึงค่าทรูพุตของ Vegas TCP และ SACK TCP มาทำการพล็อตเป็นกราฟเส้นโดยใช้โปรแกรม Excel

ซึ่งนอกเหนือจากการวิเคราะห์โดยใช้ค่าทรูพุตแล้ว ทางผู้ทำการทดสอบยังทำการวิเคราะห์โดยใช้จำนวนแพ็กเก็ตที่สูญหายกับจำนวนของแพ็กเก็ตที่ทำการส่งออกไปทั้งหมด และจำนวนแพ็กเก็ตที่ส่งออกไปทั้งหมดกับขนาดของคิว มาพล็อตกราฟโดยใช้โปรแกรม Excel และทำการวิเคราะห์เปรียบเทียบถึงประสิทธิภาพของตัวโปรโตคอล

3.3 การทดสอบประสิทธิภาพ TCP

การทดสอบในส่วนนี้เป็นการทดสอบเพื่อพิจารณาการทำงานของ SACK TCP และ Vegas TCP เพื่อวัดค่าทรูพุตที่เกิดขึ้นในระหว่างการส่ง โดยทำการเปลี่ยนพารามิเตอร์ต่างๆคือ

1. เปลี่ยนขนาดของคิว โดยใช้การจัดเรียงคิวแบบ RED
2. เปลี่ยนขนาดของคิว โดยใช้การจัดเรียงคิวแบบ DropTail
3. เปลี่ยนขนาดของแพ็กเก็ต

โครงสร้างของการจำลองที่ใช้ในการทดสอบ จะใช้รูปแบบเดียวกันทั้ง 3 การทดสอบ โดยจะมีโครงสร้าง ดังรูปที่ 3.3

รูปที่ 3.3 โครงสร้างของการจำลองที่ใช้ในการทดสอบ

จากรูปที่ 3.3 โครงสร้างที่ใช้ในการทดสอบเป็นการเชื่อมต่อจากทางฝั่งส่งคือโหนดที่ 0 ถึงโหนดที่ 3 ไปยังฝั่งรับคือโหนด 6 ถึงโหนดที่ 9 ซึ่งฝั่งส่งจะเป็นรูปแบบของ TCP ชนิด SACK TCP และ Vegas TCP และฝั่งรับจะเป็นรูปแบบของ TCPSink (หรือ Tahoe TCP) โดยจะทำการเชื่อมต่อผ่านช่องทางการเชื่อมต่อคือโหนดที่ 4 และโหนดที่ 5

ค่าพารามิเตอร์ที่สำคัญที่ใช้ในการจำลอง มีดังนี้

1. ในแต่ละด้านจะเชื่อมต่อด้วยความเร็ว 1 เมกะบิตต่อวินาที และค่าดีเลย์ 50 มิลลิวินาที
2. ระหว่างโหนดที่ 4 และ 5 เชื่อมต่อด้วยความเร็ว 0.1 เมกะบิตต่อวินาที และค่าดีเลย์ 10 มิลลิวินาที

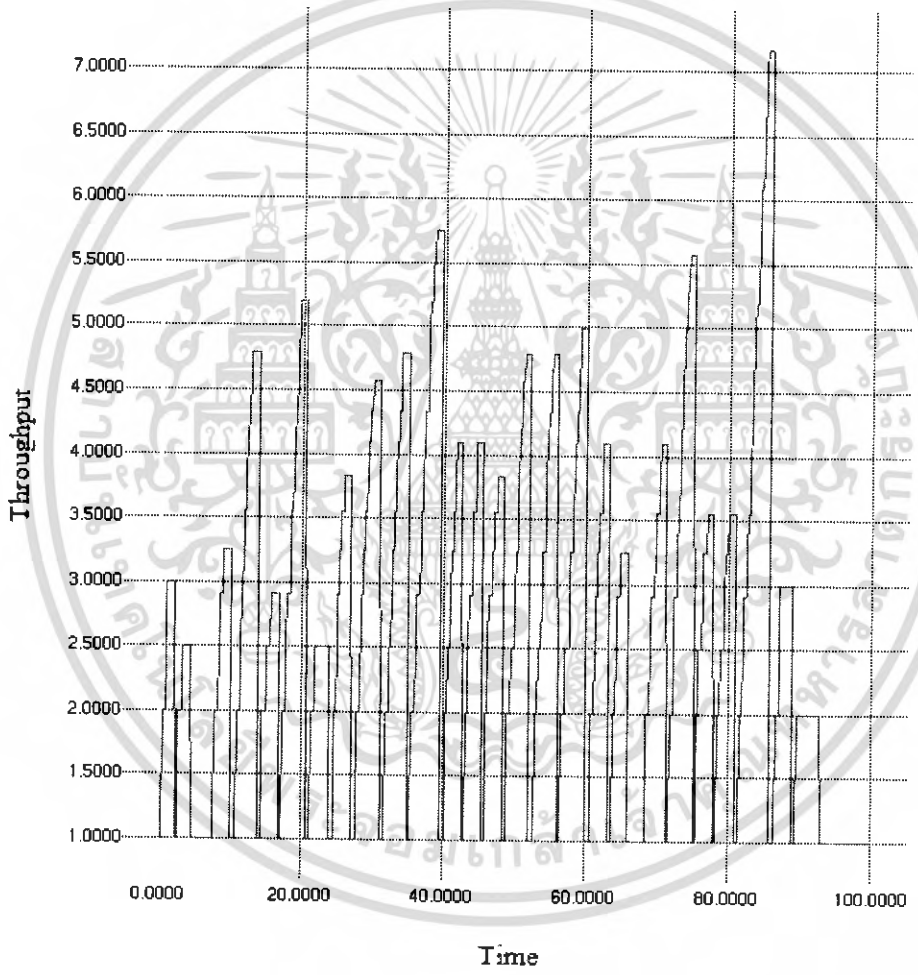
ในการทดสอบนั้นจะทำการพิจารณาทุกโหนด เพื่อพิจารณาถึงประสิทธิภาพของการทำงานของ SACK TCP และ Vegas TCP

สำหรับผลการทดสอบในแต่ละส่วนนั้นจะแสดงออกมาในรูปของกราฟซึ่งจะได้มาจากการพล็อตกราฟของโปรแกรม X Graph ซึ่งเป็นโปรแกรมที่ใช้งานร่วมกับโปรแกรม NS โดยจะทำการพิจารณาถึงการเดินทางในแต่ละแพ็กเก็ตที่ไปถึงช่องทางการเชื่อมต่อ หรือโหนดที่ 4

3.3.1 การทดสอบโดยการเปลี่ยนขนาดของคิว โดยใช้การจัดเรียงคิวแบบ RED

1. การทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของคิวโดยใช้การจัดเรียงคิวแบบ RED

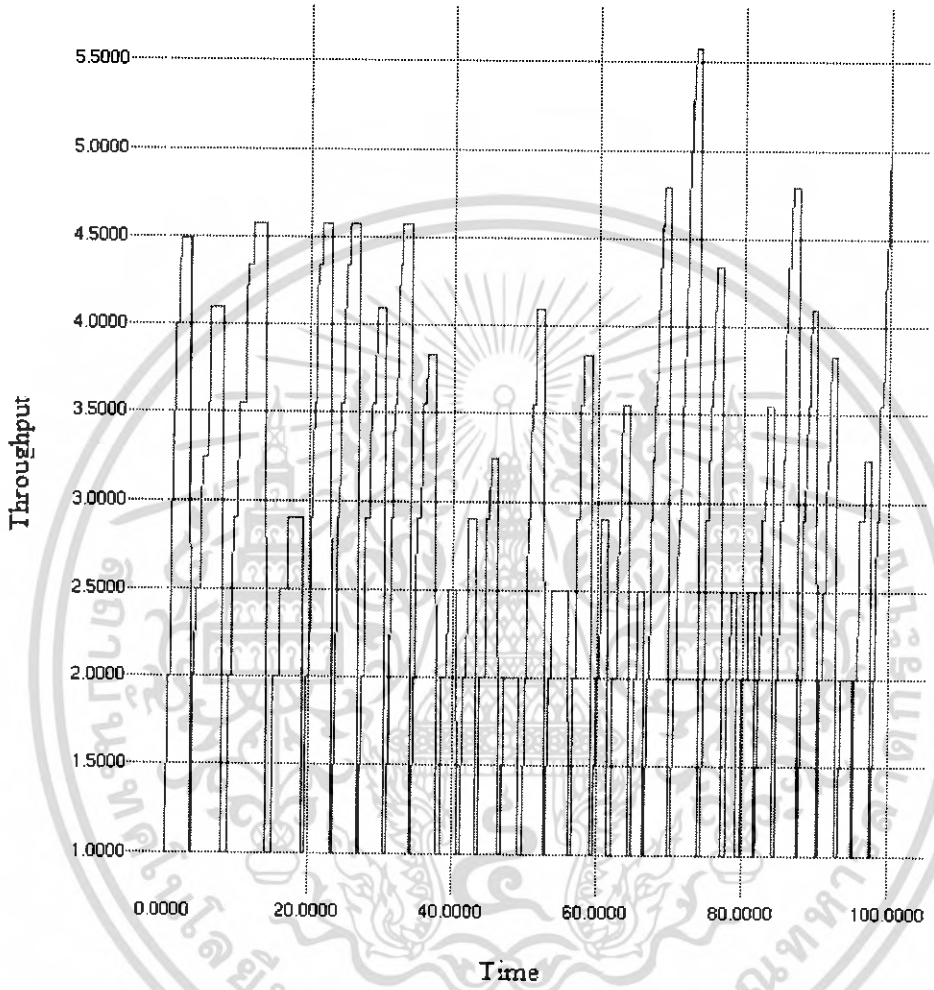
การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ขนาดของแพ็กเก็ตมีค่า 1,000 ไบต์ และใช้การจัดคิวแบบ RED ซึ่งจะทำการเปลี่ยนค่าขนาดของคิวชนิดนี้ 5 ค่าด้วยกัน คือ 10, 20, 30, 40 และ 50 ตามลำดับ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.4, 3.5, 3.6, 3.7, และ 3.8



รูปที่ 3.4 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED

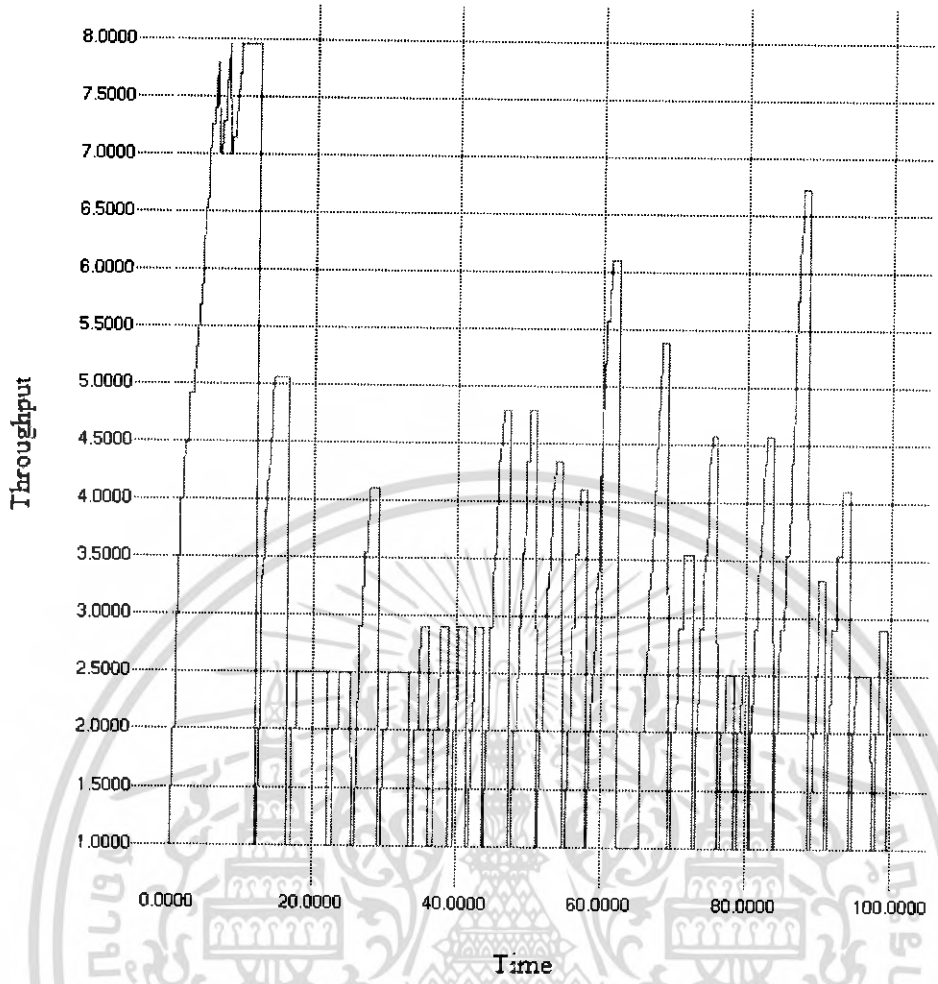
จากรูปที่ 3.4 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED โดยที่ SACK TCP นั้นจะเริ่มต้นส่งแพ็กเก็ตแบบ Slow Start หรือเป็นการเริ่มต้นส่งออกไปอย่างช้าๆ และจะเพิ่มจำนวนแพ็กเก็ตที่จะส่งออกไปให้มากขึ้นเรื่อยๆ ซึ่งจะส่งผลให้ค่าทรูพุตที่

ได้นั้นมีการเพิ่มขึ้นเรื่อยๆ และเมื่อเกิดความคับคั่งที่เราเตอร์มากๆ จนทำให้มีแพ็กเก็ตสูญหาย จากนั้น SACK TCP จะทำการ Fast Recovery โดยจะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ในการส่ง และหลังจากนั้นจะทำการ Fast Retransmission ซึ่งจะเห็นได้ว่าค่าทรูพุตที่ได้นั้นจะมีค่าลดลง และจะเป็นเช่นนี้ไปตลอด



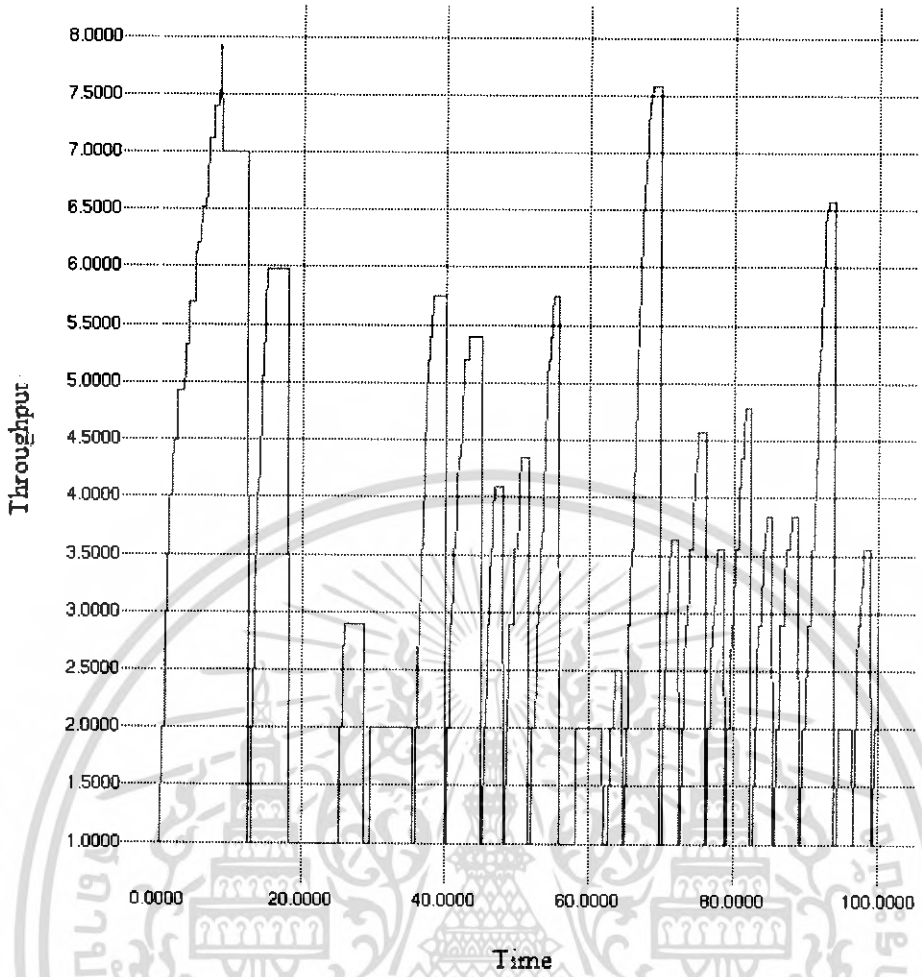
รูปที่ 3.5 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED

จากรูปที่ 3.5 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED ในหลักการทำงานนั้นจะเหมือนกับรูปที่ 3.4 แต่เมื่อทำการปรับขนาดของคิวให้เพิ่มขึ้นเป็น 20 แล้ว จะเห็นได้ว่าค่าทรูพุตที่ได้นั้นจะมีค่าลดลงเมื่อเปรียบเทียบกับรูปที่ 3.4 และมีค่าทรูพุตที่ไม่คงที่



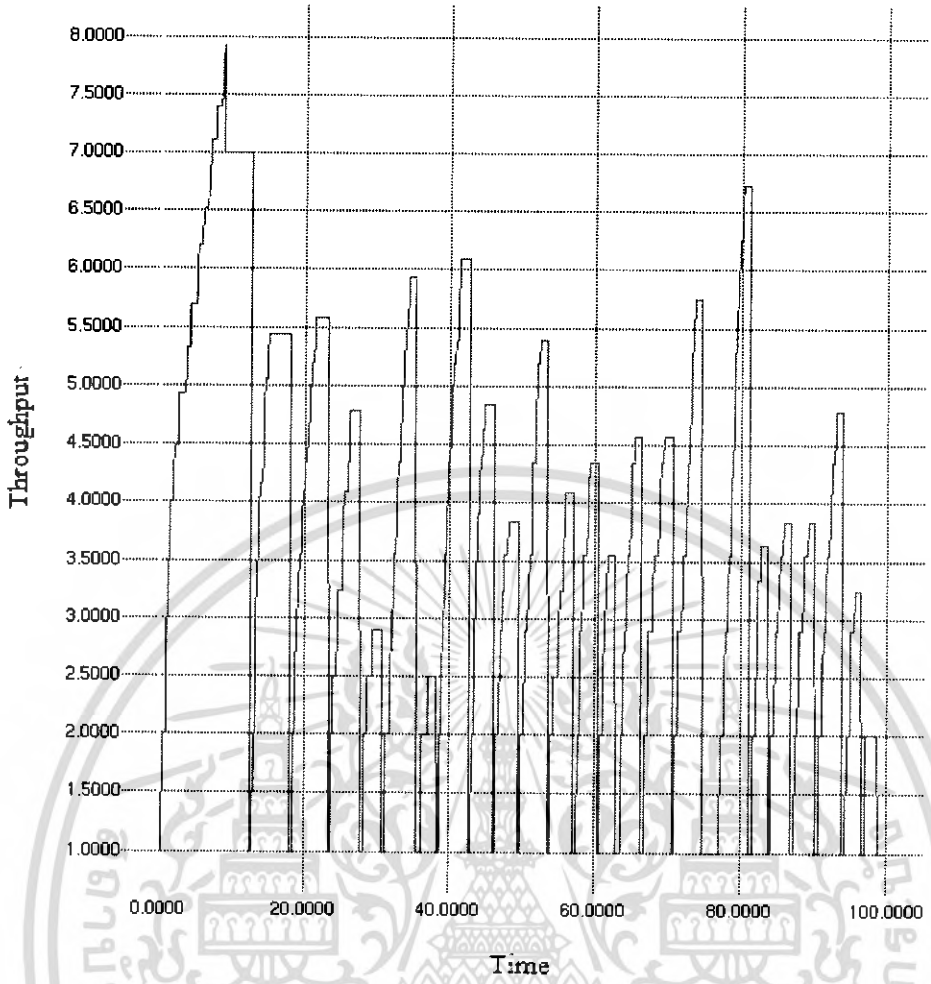
รูปที่ 3.6 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 30 โดยใช้การจัดเรียงคิวแบบ RED

จากรูปที่ 3.6 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 30 โดยใช้การจัดเรียงคิวแบบ RED ในหลักการทำงานนั้นจะเหมือนกับรูปที่ 3.4 แต่เมื่อทำการปรับขนาดของคิวให้เพิ่มขึ้นเป็น 30 แล้ว จะเห็นได้ว่าค่าทรูพุตที่ได้ในช่วงเริ่มต้นนั้นจะมีค่าสูงขึ้นและยาวนานกว่าเมื่อเปรียบเทียบกับรูปที่ 3.4 และ 3.5 ในช่วงเวลาที่ 0 จนถึง 20 วินาที หลังจากนั้นค่าทรูพุตที่ได้จะมีค่าไม่คงที่



รูปที่ 3.7 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 40 โดยใช้การจัดเรียงคิวแบบ RED

จากรูปที่ 3.7 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 40 โดยใช้การจัดเรียงคิวแบบ RED ในหลักการทำงานนั้นจะเหมือนกับรูปที่ 3.4 แต่เมื่อทำการปรับขนาดของคิวให้เพิ่มขึ้นเป็น 40 แล้ว จะเห็นได้ว่าค่าทรูพุตที่ได้ในช่วงเริ่มต้นนั้นจะมีค่าสูงขึ้น และจะมีช่วงเวลาที่มีค่าทรูพุตสูงนั้นกว้างขึ้น ในช่วงเวลา 0 ถึง 20 วินาที และหลังจากนั้นค่าทรูพุตจะมีค่าไม่คงที่



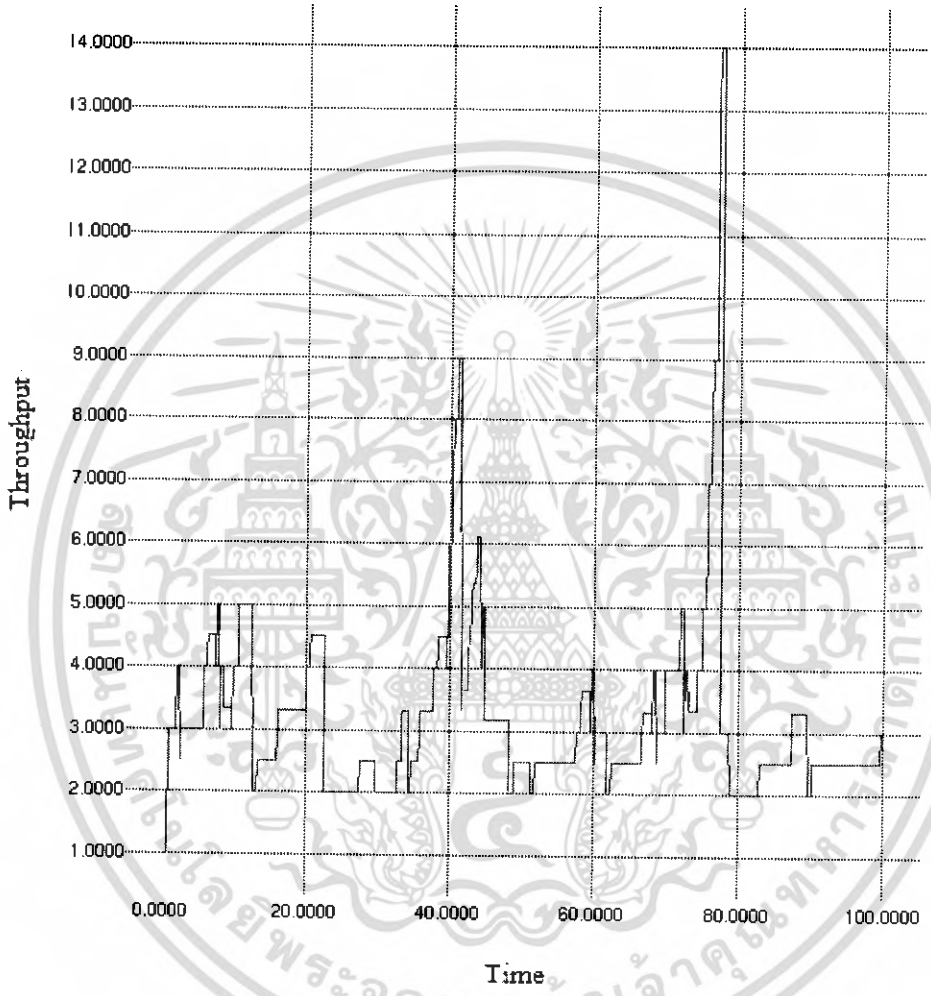
รูปที่ 3.8 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 50 โดยใช้การจัดเรียงคิวแบบ RED

จากรูปที่ 3.8 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 50 โดยใช้การจัดเรียงคิวแบบ RED ในหลักการทำงานนั้นจะเหมือนกับรูปที่ 3.4 แต่เมื่อทำการปรับขนาดของคิวให้เพิ่มขึ้นเป็น 50 แล้ว จะเห็นได้ว่าค่าทรูพุตที่ได้ในช่วงเริ่มต้นนั้นจะมีค่าสูงขึ้น และจะมีช่วงเวลาที่มีค่าทรูพุตสูงนั้นกว้างขึ้น ในช่วงเวลา 0 ถึง 20 วินาที

เมื่อทำการทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ RED ซึ่งผลที่ได้แสดงดังรูปที่ 3.4 จนถึง 3.8 นั้น สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที นั้นมีค่าน้อย และหลังจากนั้นค่าทรูพุตก็จะมีค่าเพิ่มขึ้นและลดลงไม่คงที่ แต่เมื่อทำการเพิ่มขนาดของคิวให้มากขึ้นค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที นั้นจะมีค่าสูง และหลังจากนั้นค่าทรูพุตก็จะมีค่าไม่คงที่

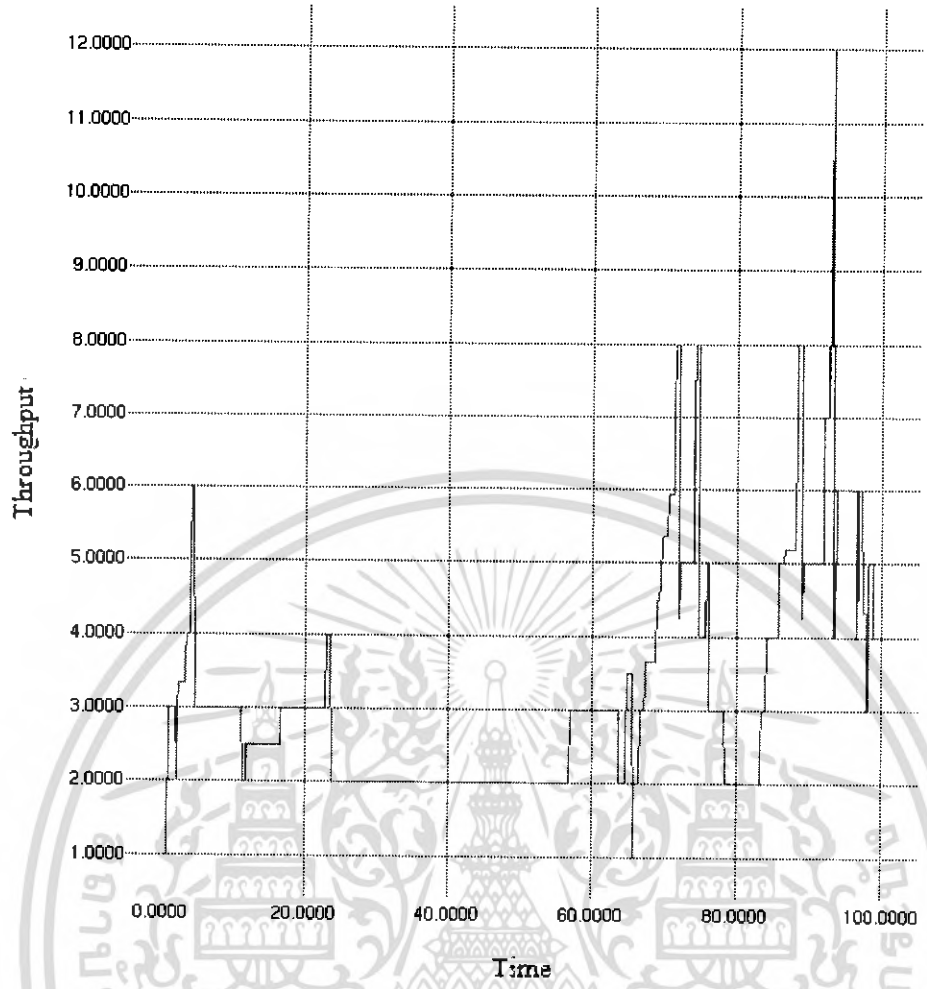
2. การทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของคิวโดยใช้การจัดเรียงคิวแบบ RED

การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ขนาดของแพ็กเก็ตมีค่า 1,000 ไบต์ และใช้การจัดคิวแบบ RED ซึ่งจะทำการเปลี่ยนค่าขนาดของคิวชนิดนี้ 5 ค่าด้วยกัน คือ 10, 20, 30, 40 และ 50 ตามลำดับ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.9, 3.10 และเมื่อปรับให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 นั้น ผลที่ได้คือ ค่าทรูพุตนั้นคงที่ ดังนั้น กราฟที่ได้จะเหมือนกันดังแสดงไว้ในรูปที่ 3.11



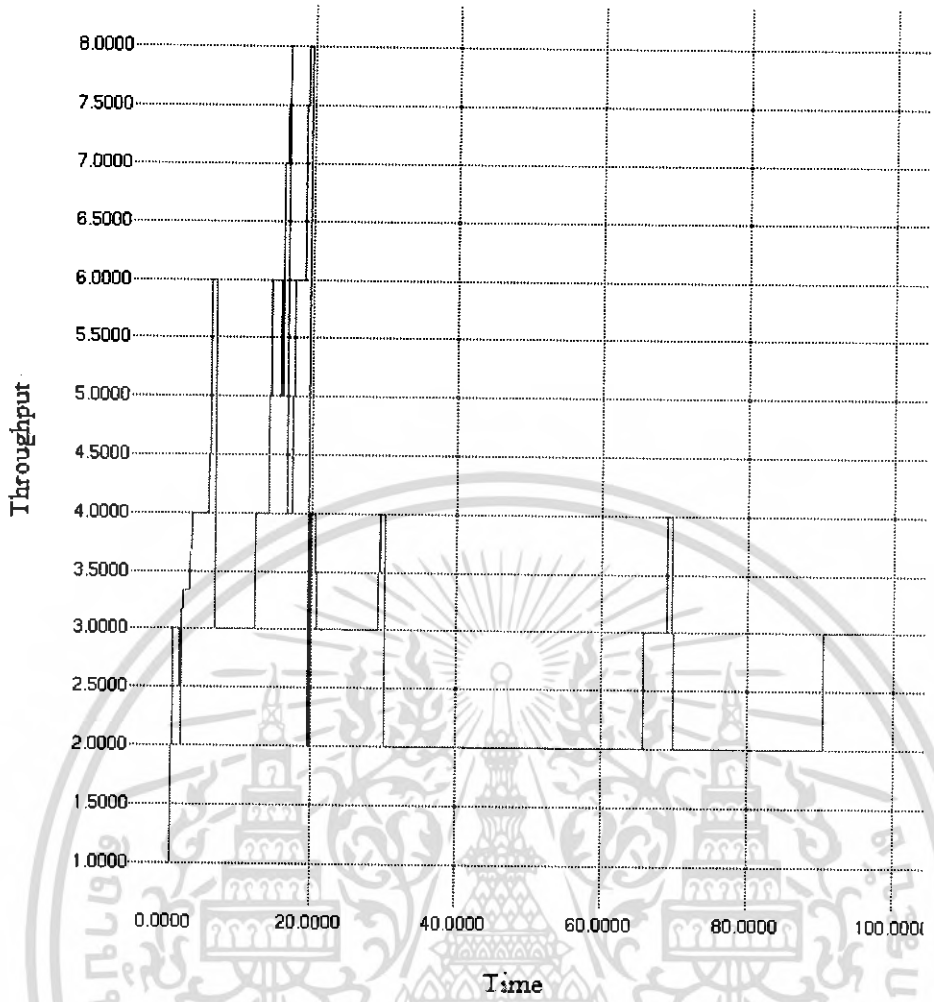
รูปที่ 3.9 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED

จากรูปที่ 3.9 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED โดยที่ Vegas TCP นั้นจะมีกลไกควบคุมปริมาณแพ็กเก็ตในเราเตอร์ตลอดเวลา ดังนั้นเมื่อมีปริมาณแพ็กเก็ตในเราเตอร์เพิ่มมากขึ้น Vegas TCP ก็จะทำการลดปริมาณการส่งแพ็กเก็ตลง จนถึงขั้นหยุดส่ง ส่งผลให้การสูญหายของแพ็กเก็ตน้อยลงไปด้วย ดังนั้นค่าทรูพุตที่ได้นั้นจะมีช่วงเวลาที่กว้าง และมีค่าทรูพุตต่ำสุดในระดับที่คงที่



รูปที่ 3.10 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED

จากรูปที่ 3.10 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED โดยที่จะมีหลักการทำงานเหมือนกับรูปที่ 3.9 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีค่าเท่ากับ 20 แล้วนั้น ค่าทรูพุตที่ได้นั้นในช่วงเวลาที่ 0 ถึง 20 วินาทีจะมีค่าสูงกว่ารูปที่ 3.9 และมีค่าทรูพุตต่ำสุดในระดับที่คงที่



รูปที่ 3.11 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ RED

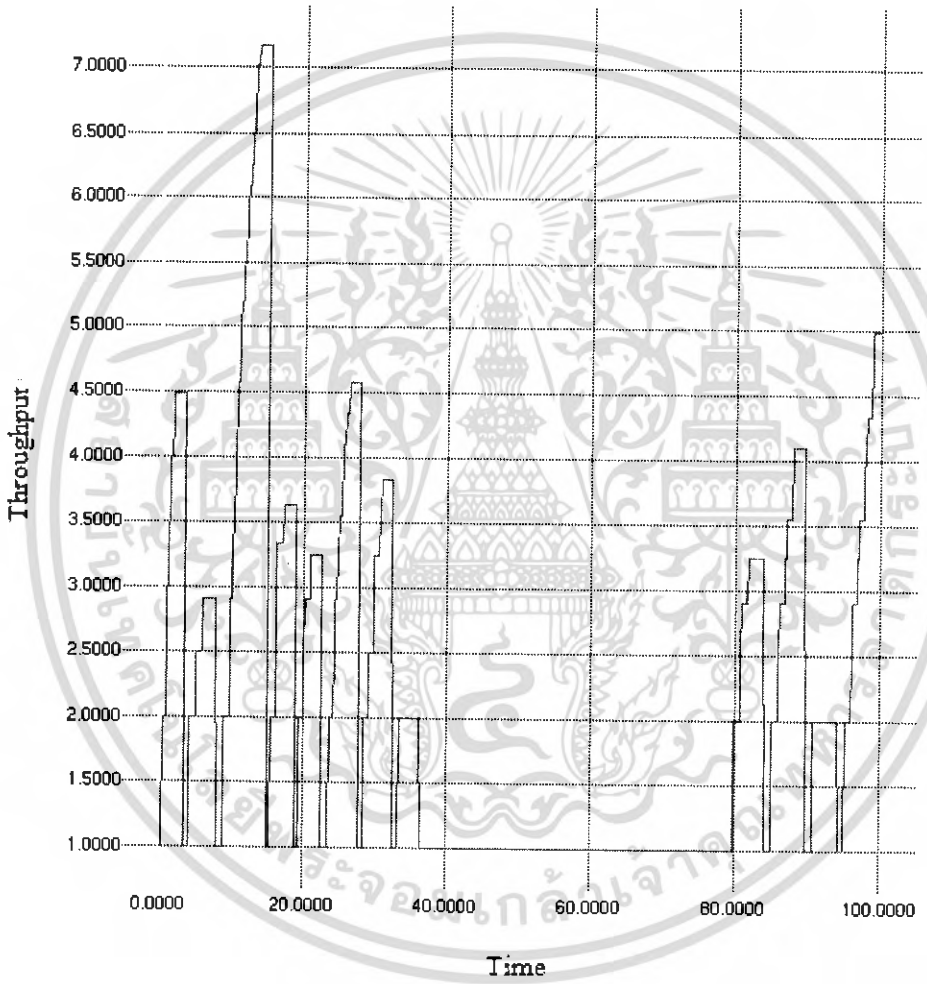
จากรูปที่ 3.11 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ RED โดยที่จะมีหลักการทำงานเหมือนกับรูปที่ 3.9 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีค่าเพิ่มขึ้นตั้งแต่ 30 จนถึง 50 แล้วนั้น จะได้ค่าทรูพุตที่มีค่าสูงกว่าในช่วงเวลา 0 ถึง 20 วินาที เปรียบเทียบกับรูปที่ 3.9 และเมื่อเวลาผ่านไปจนถึงจุดๆหนึ่งค่าทรูพุตจะมีค่าคงที่

เมื่อทำทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ RED ซึ่งผลที่ได้แสดงดังรูปที่ 3.9 จนถึง 3.11 นั้น สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตในช่วงเวลา 0 ถึง 20 วินาที มีค่าน้อยและหลังจากนั้นที่เวลาผ่านไปจนถึงจุดๆหนึ่งค่าทรูพุตจะมีค่าคงที่เท่ากัน เมื่อเพิ่มขนาดของคิวให้มากขึ้นค่าทรูพุตจะมีค่ามากขึ้นในช่วงเวลาที่ 0 ถึง 20 วินาที

3.3.2 การทดสอบโดยการเปลี่ยนขนาดของคิว โดยใช้การจัดเรียงคิวแบบ DropTail

1. การทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของคิวโดยใช้การจัดเรียงคิวแบบ DropTail

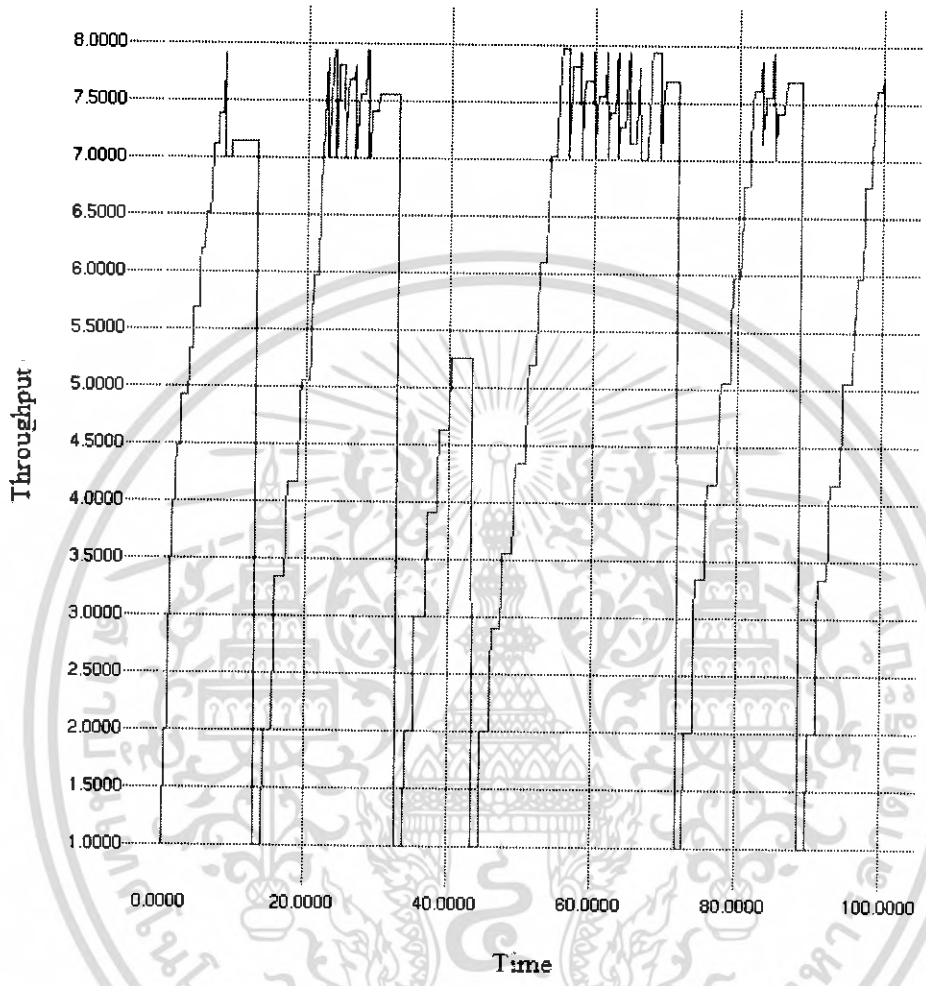
การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ขนาดของแพ็กเก็ตมีค่า 1,000 ไบต์ และใช้การจัดคิวแบบ DropTail ซึ่งจะทำการเปลี่ยนค่าขนาดของคิวชนิดนี้ 5 ค่าด้วยกัน คือ 10, 20, 30, 40 และ 50 ตามลำดับ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.12, 3.13 และเมื่อปรับให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 นั้น ผลที่ได้คือ ค่าทรูพุตนั้นคงที่ ดังนั้น กราฟที่ได้จะเหมือนกันดังแสดงไว้ในรูปที่ 3.14



รูปที่ 3.12 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ DropTail

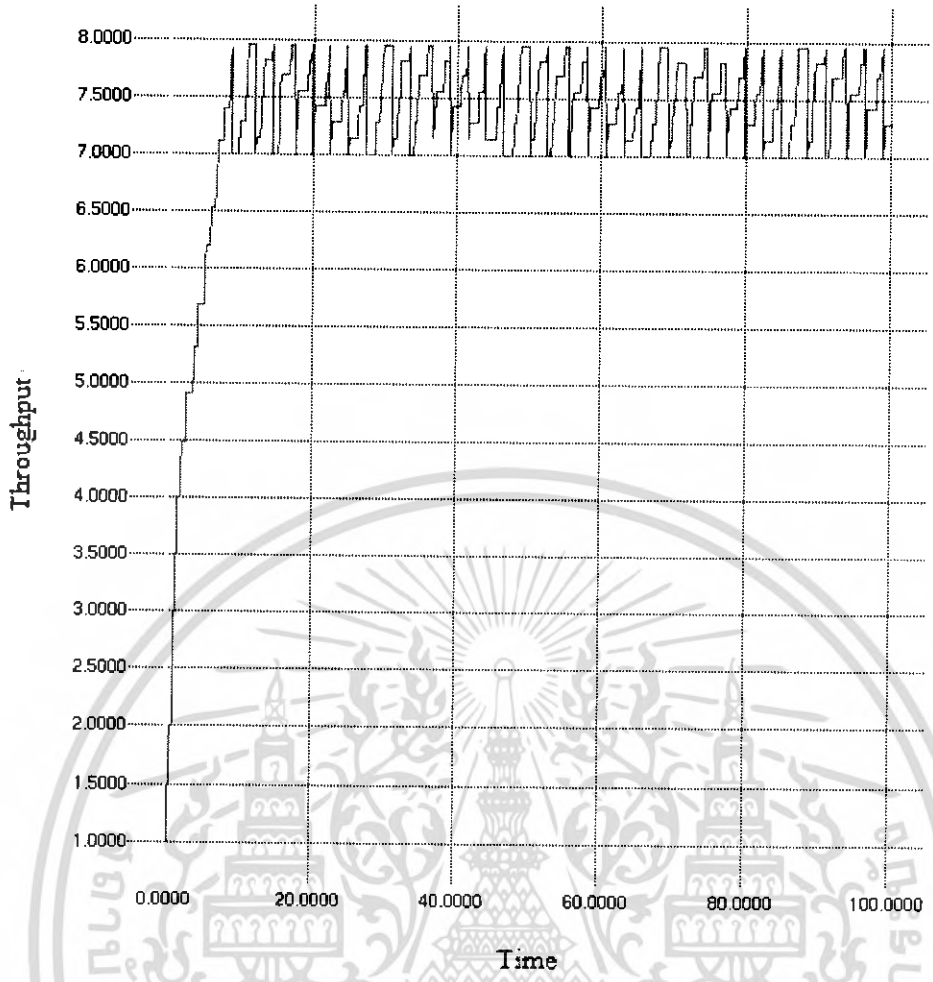
จากรูปที่ 3.12 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ DropTail โดยที่ SACK TCP นั้นจะเริ่มต้นส่งแพ็กเก็ตแบบ Slow Start หรือเป็นการเริ่มต้นส่งออกไปอย่างช้าๆ และจะเพิ่มจำนวนแพ็กเก็ตที่จะส่งออกไปให้มากขึ้นเรื่อยๆ ซึ่งจะส่งผลให้ค่าทรูพุตที่ได้นั้นมีการเพิ่มขึ้นเรื่อยๆ และเมื่อเกิดความคับคั่งที่เราเตอร์มากๆ จนทำให้มีแพ็กเก็ตสูญหาย

จากนั้น SACK TCP จะทำการ Fast Recovery โดยจะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ในการส่ง และหลังจากนั้นจะทำการ Fast Retransmission ซึ่งจะเห็นได้ว่าค่าทรูพุตที่ได้นั้นจะมีค่าลดลง และจะเป็นเช่นนี้ไปตลอด



รูปที่ 3.13 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ DropTail

จากรูปที่ 3.13 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ DropTail โดยที่หลักการทำงานนั้นจะเหมือนกันกับในรูปที่ 3.12 แต่เมื่อเพิ่มค่าขนาดของคิวให้มีค่าเท่ากับ 20 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลาที่ 0 ถึง 20 วินาที นั้นจะมีค่าที่สูงกว่ารูปที่ 3.12 และหลังจากนั้นค่าทรูพุตจะมีค่าไม่คงที่



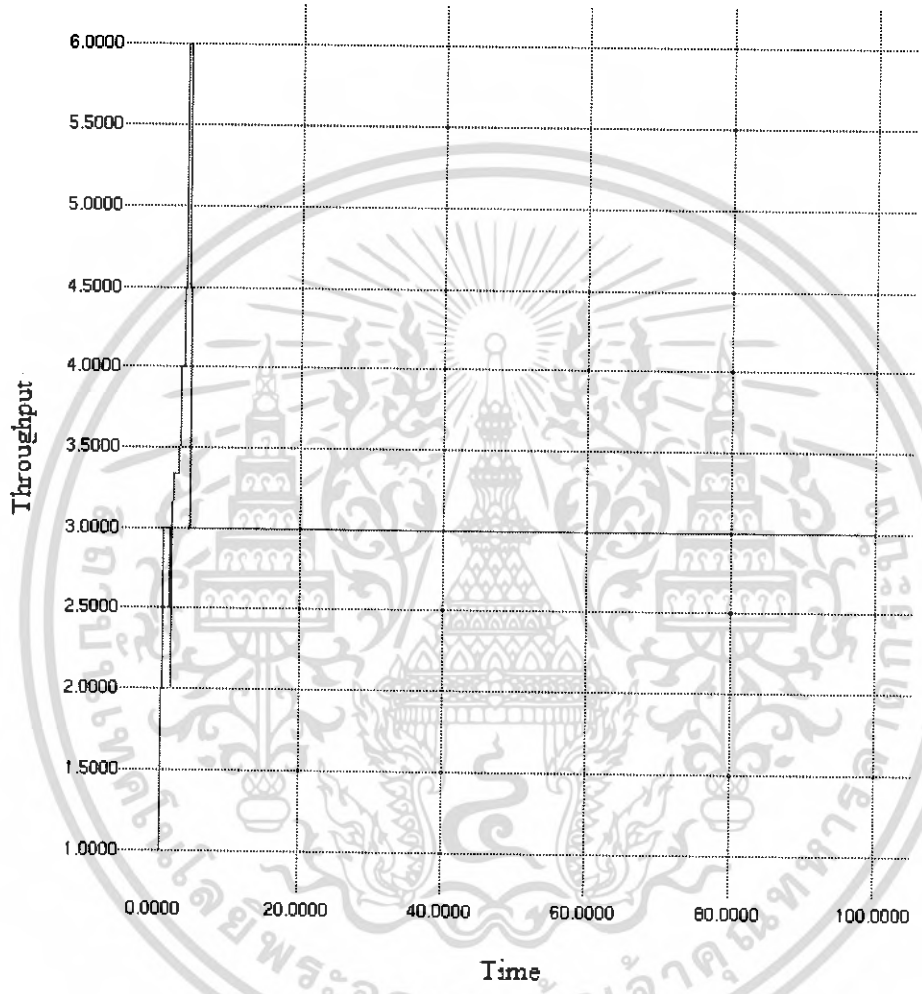
รูปที่ 3.14 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail

จากรูปที่ 3.14 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.12 แต่เมื่อเพิ่มค่าขนาดของคิวให้มีค่าเท่ากับ 30 จนถึง 50 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลาที่ 0 ถึง 20 วินาที นั้นจะมีค่าที่สูงขึ้นและหลังจากนั้นจะมีค่าทรูพุตที่ไม่คงที่

จากการทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ DropTail และผลที่ได้นั้นแสดงดังรูปที่ 3.12 ถึง 3.14 สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที นั้นมีค่าน้อยและเมื่อเวลาผ่านไปจะมีค่าไม่คงที่ แต่เมื่อเพิ่มขนาดของคิวให้มากขึ้นค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที จะมีค่าสูงขึ้นและหลังจากนั้นจะมีค่าไม่คงที่

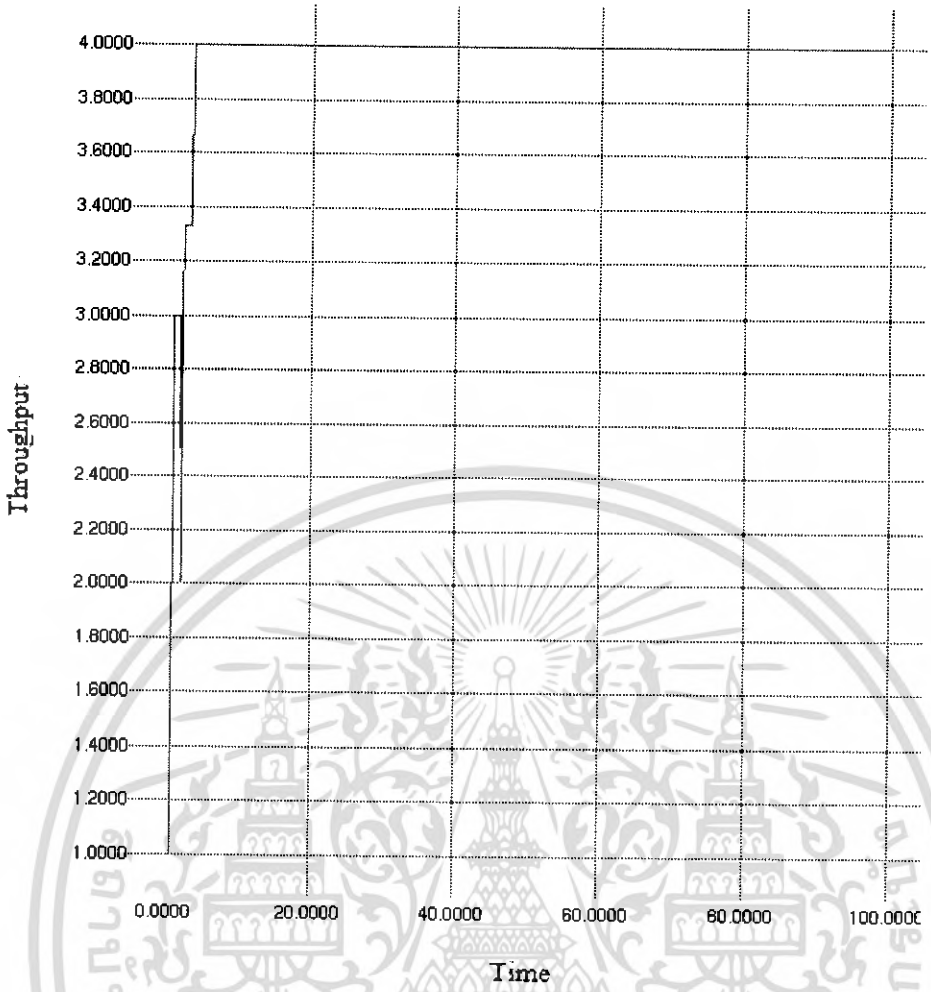
2. การทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของคิวโดยใช้การจัดเรียงคิวแบบ DropTail

การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ขนาดของแพ็กเก็ตมีค่า 1,000 ไบต์ และใช้การจัดคิวแบบ DropTail ซึ่งจะทำการเปลี่ยนค่าขนาดของคิวชนิดนี้ 5 ค่าด้วยกัน คือ 10, 20, 30, 40 และ 50 ตามลำดับ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.15 และเมื่อปรับให้ขนาดของคิวมีค่าเท่ากับ 20, 30, 40 และ 50 นั้น ผลที่ได้คือ ค่าทรูพุตนั้นคงที่ ดังนั้น กราฟที่ได้จะเหมือนกันดังแสดงไว้ในรูปที่ 3.16



รูปที่ 3.15 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ DropTail

จากรูปที่ 3.15 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10โดยใช้การจัดเรียงคิวแบบ DropTail โดยที่ Vegas TCP นั้นจะมีกลไกควบคุมปริมาณแพ็กเก็ตในเราเตอร์ตลอดเวลา ดังนั้นเมื่อมีปริมาณแพ็กเก็ตในเราเตอร์เพิ่มมากขึ้น Vegas TCP ก็จะทำการลดปริมาณการส่งแพ็กเก็ตลงจนถึงขั้นหยุดส่ง ส่งผลให้การสูญหายของแพ็กเก็ตน้อยลงไปด้วย ดังนั้นค่าทรูพุตที่ได้นั้นจะมีค่าสูงขึ้นในช่วงเวลาสั้นๆ แต่เมื่อเวลาผ่านไปถึงจุดๆหนึ่งค่าทรูพุตที่ได้นั้นจะมีระดับที่คงที่



รูปที่ 3.16 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 20, 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail

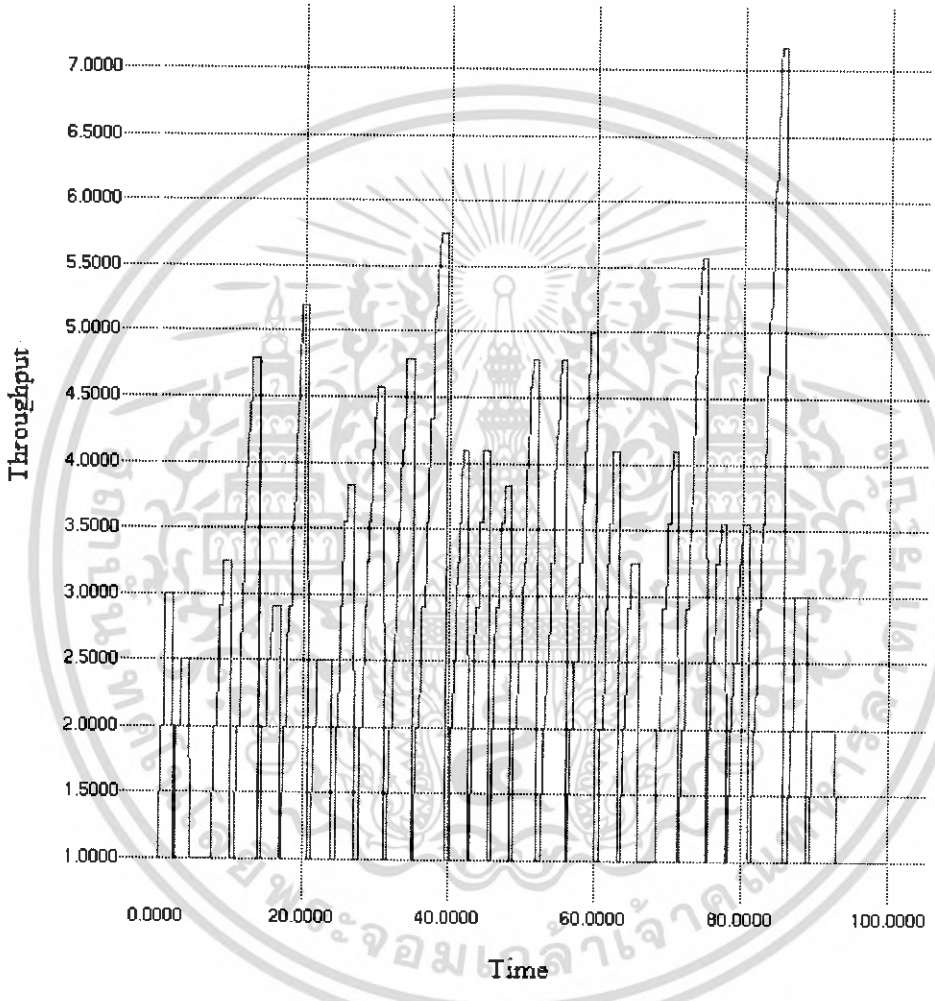
จากรูปที่ 3.16 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20, 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail โดยที่หลักการทำงานนั้นจะเหมือนกับรูปที่ 3.15 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีขนาดอยู่ที่ 20 จนถึง 50 นั้นจะเห็นได้ว่าค่าทรูพุตในช่วงเวลาสั้นๆ จะมีค่าที่ต่ำกว่าเมื่อเปรียบเทียบกับรูปที่ 3.15 และหลังจากนั้นค่าทรูพุตที่ได้จะมีค่าคงที่และสูงกว่าเมื่อเปรียบเทียบกับรูปที่ 3.15

เมื่อทำทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ DropTail ซึ่งผลที่ได้แสดงดังรูปที่ 3.15 และ 3.16 นั้น สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตในช่วงเวลาสั้นๆ มีค่าสูงกว่าเมื่อเพิ่มขนาดของคิวให้มากขึ้นแต่เมื่อเวลาผ่านไปค่าทรูพุตจะมีค่าคงที่และน้อยกว่าเมื่อเพิ่มขนาดของคิวให้มากขึ้น

3.3.3 การทดสอบโดยเปลี่ยนขนาดของแพ็กเก็ต

1. การทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของแพ็กเก็ต และใช้การจัดเรียงคิวแบบ RED

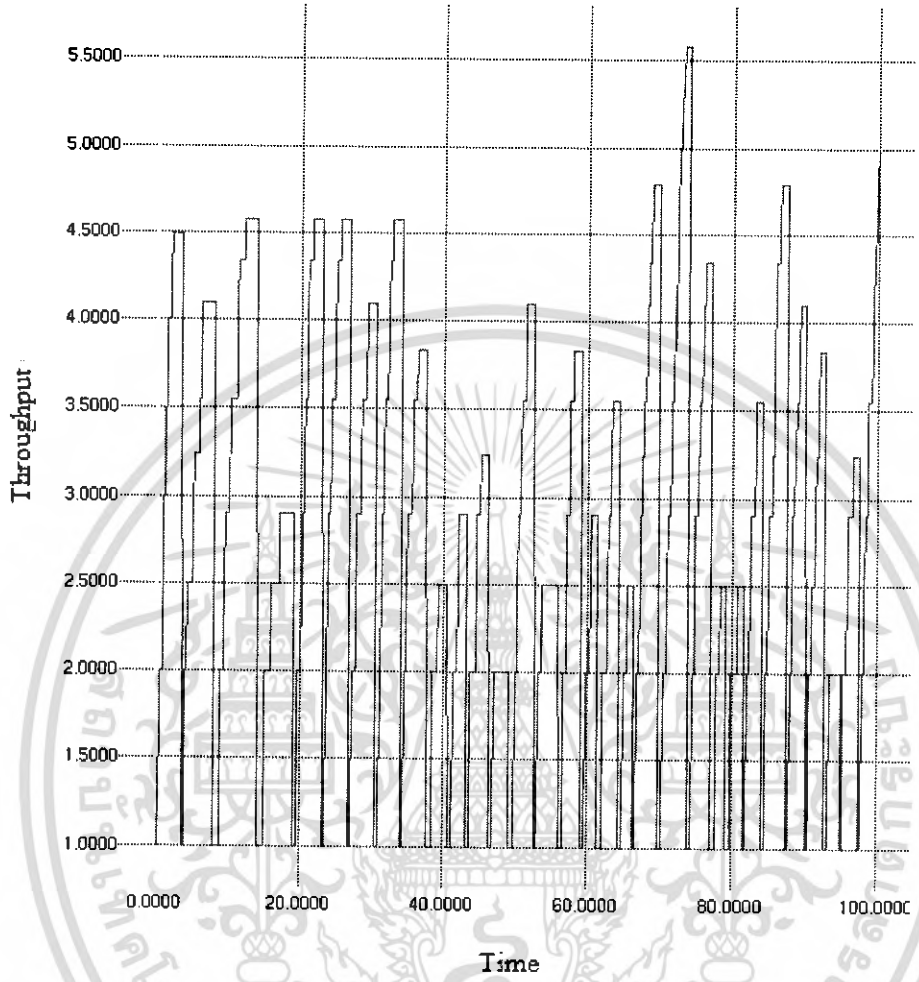
การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ในการจัดเรียงคิวจะใช้การจัดเรียงคิวแบบ RED โดยทำการปรับขนาดของคิวให้มีค่าเท่ากับ 10, 20, 30, 40 และ 50 ตามลำดับ และขนาดของแพ็กเก็ตจะกำหนดไว้ที่ 500 ไบต์ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.17, 3.18, 3.19, 3.20, และ 3.21



รูปที่ 3.17 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

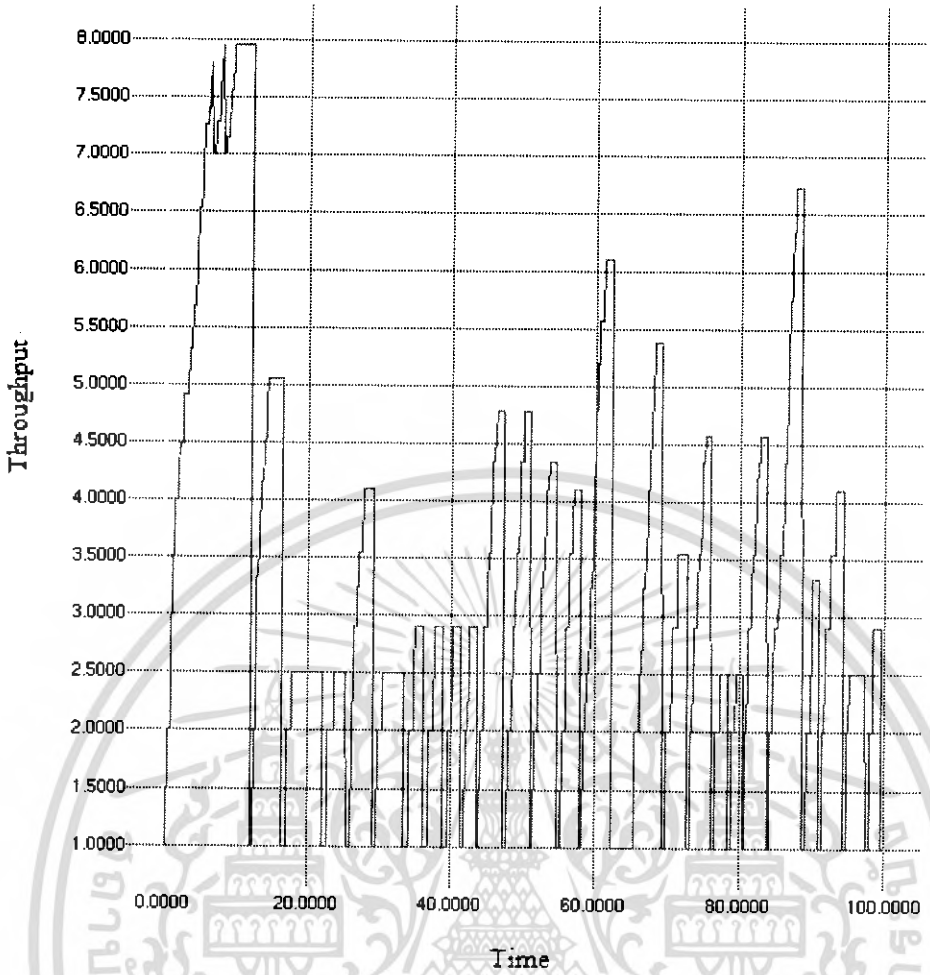
จากรูปที่ 3.17 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่ SACK TCP นั้นจะเริ่มต้นส่งแพ็กเก็ตแบบ Slow Start หรือเป็นการเริ่มต้นส่งออกไปอย่างช้าๆ และจะเพิ่มจำนวนแพ็กเก็ตที่จะส่งออกไปให้มากขึ้นเรื่อยๆ ซึ่งจะส่งผลให้ค่าทรูพุตที่ได้นั้นมีการเพิ่มขึ้นเรื่อยๆ และเมื่อเกิดความคับคั่งที่เราเตอร์มากๆ จนทำให้มีแพ็กเก็ตสูญหาย จากนั้น SACK TCP จะทำการ Fast Recovery โดยจะใช้ค่า

pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ในการส่ง และหลังจากนั้นจะทำการ Fast Retransmission ซึ่งจะเห็นได้ว่าค่าทรูพุตที่ได้นั้นจะมีค่าลดลง และจะเป็นเช่นนี้ไปตลอด



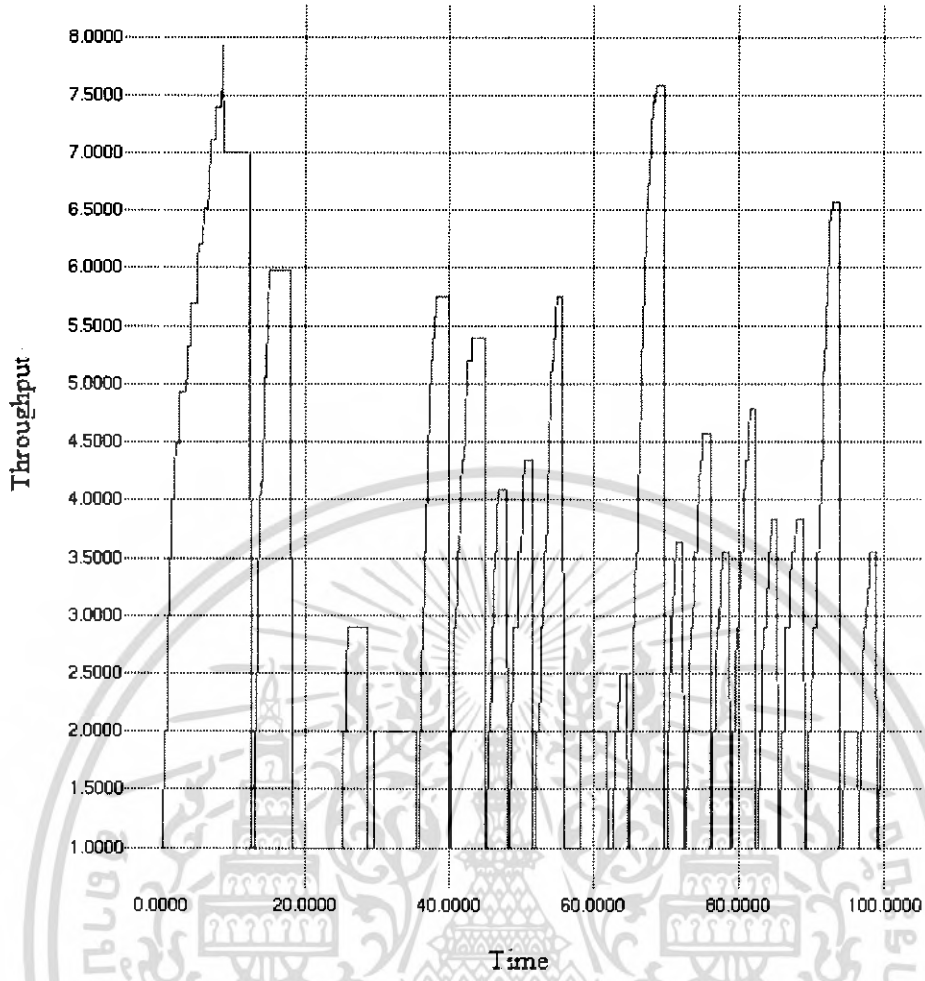
รูปที่ 3.18 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.18 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.17 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีค่าเท่ากับ 20 แล้วนั้น ค่าทรูพุตที่ได้จะมีค่าลดลงและไม่คงที่



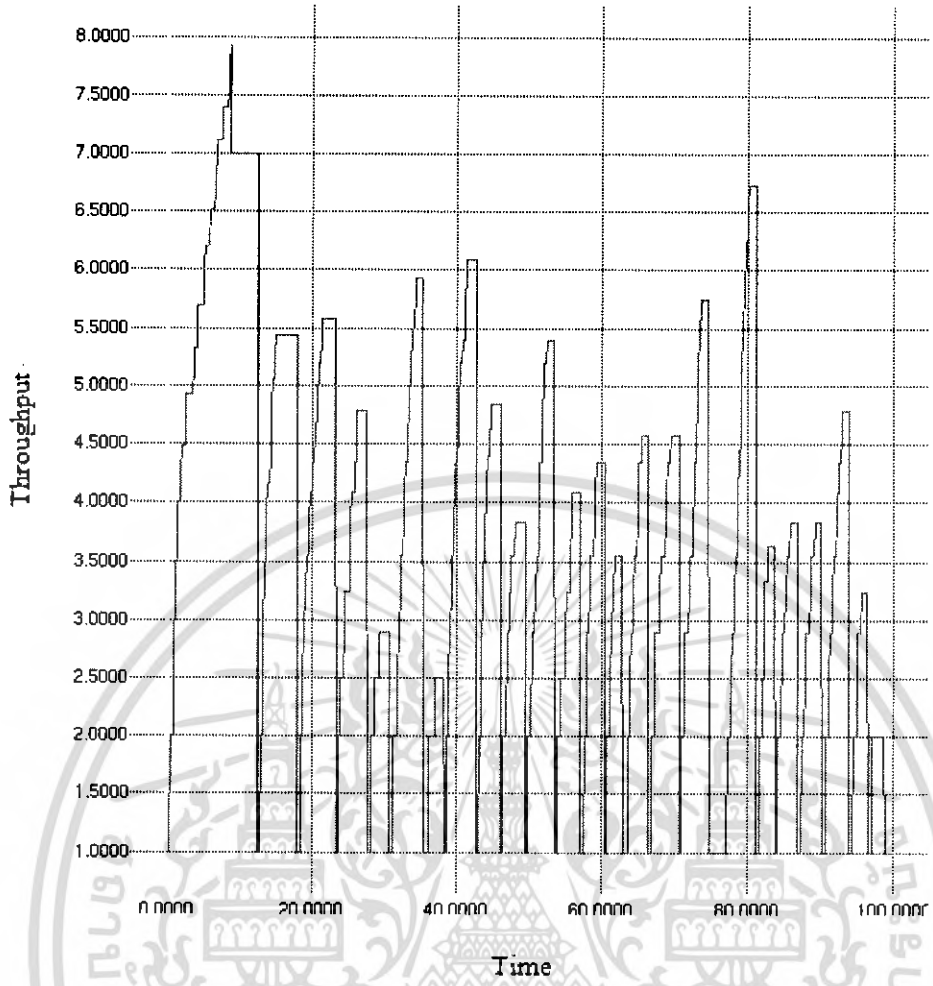
รูปที่ 3.19 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 30 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.19 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 30 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.17 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีค่าเท่ากับ 30 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลา 0 ถึง 20 วินาที นั้นจะมีค่าสูงขึ้นและเมื่อเวลาผ่านไปค่าทรูพุตจะมีค่าไม่คงที่



รูปที่ 3.20 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 40 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.20 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 40 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.17 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีค่าเท่ากับ 40 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลา 0 ถึง 20 วินาที นั้นจะมีค่าสูงขึ้นและเมื่อเวลาผ่านไปค่าทรูพุตจะมีค่าไม่คงที่



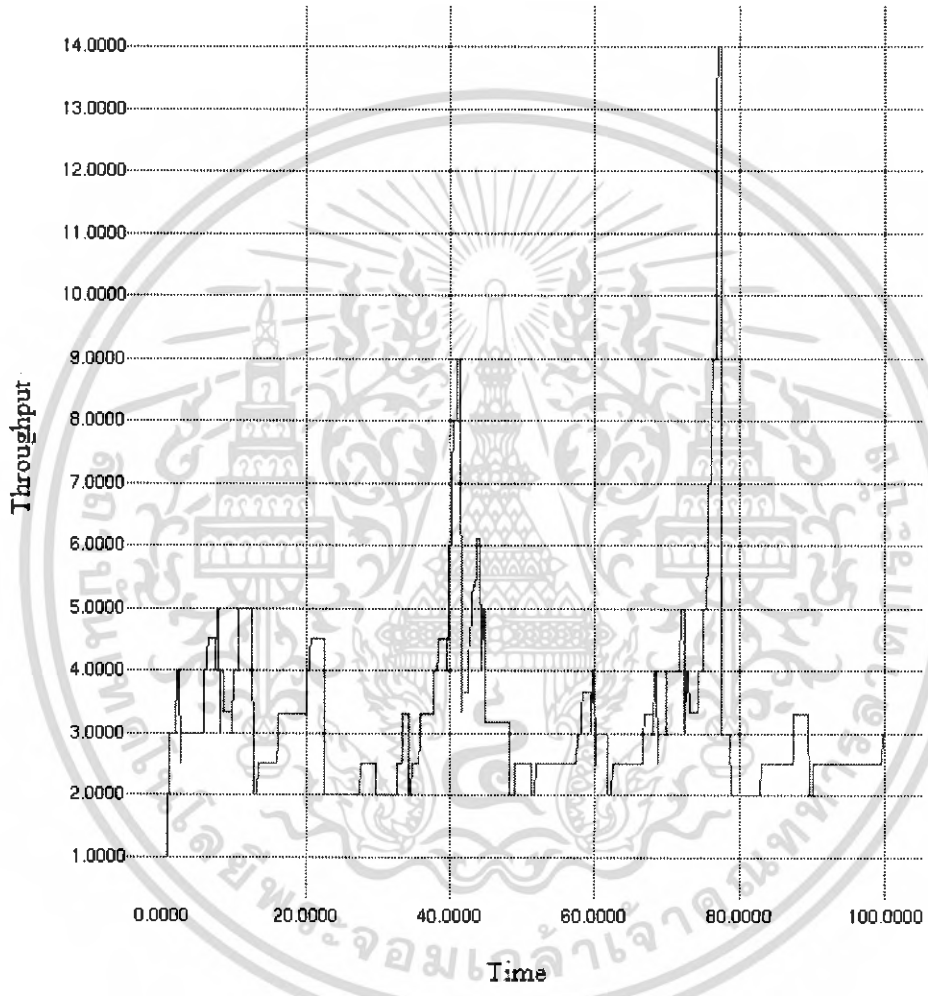
รูปที่ 3.21 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 50 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.21 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 50 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.17 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีค่าเท่ากับ 50 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลา 0 ถึง 20 วินาที นั้นจะมีค่าสูงขึ้นและเมื่อเวลาผ่านไปค่าทรูพุตจะมีค่าไม่คงที่

จากการทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ RED โดยกำหนดให้ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ และผลที่ได้นั้นแสดงดังรูปที่ 3.17 ถึง 3.21 สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที นั้นมีค่าน้อย และหลังจากนั้นค่าทรูพุตก็จะมีค่าเพิ่มขึ้นและลดลงไม่คงที่ แต่เมื่อทำการเพิ่มขนาดของคิวให้มากขึ้นค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที นั้นจะมีค่าสูง และหลังจากนั้นค่าทรูพุตก็จะมีค่าไม่คงที่

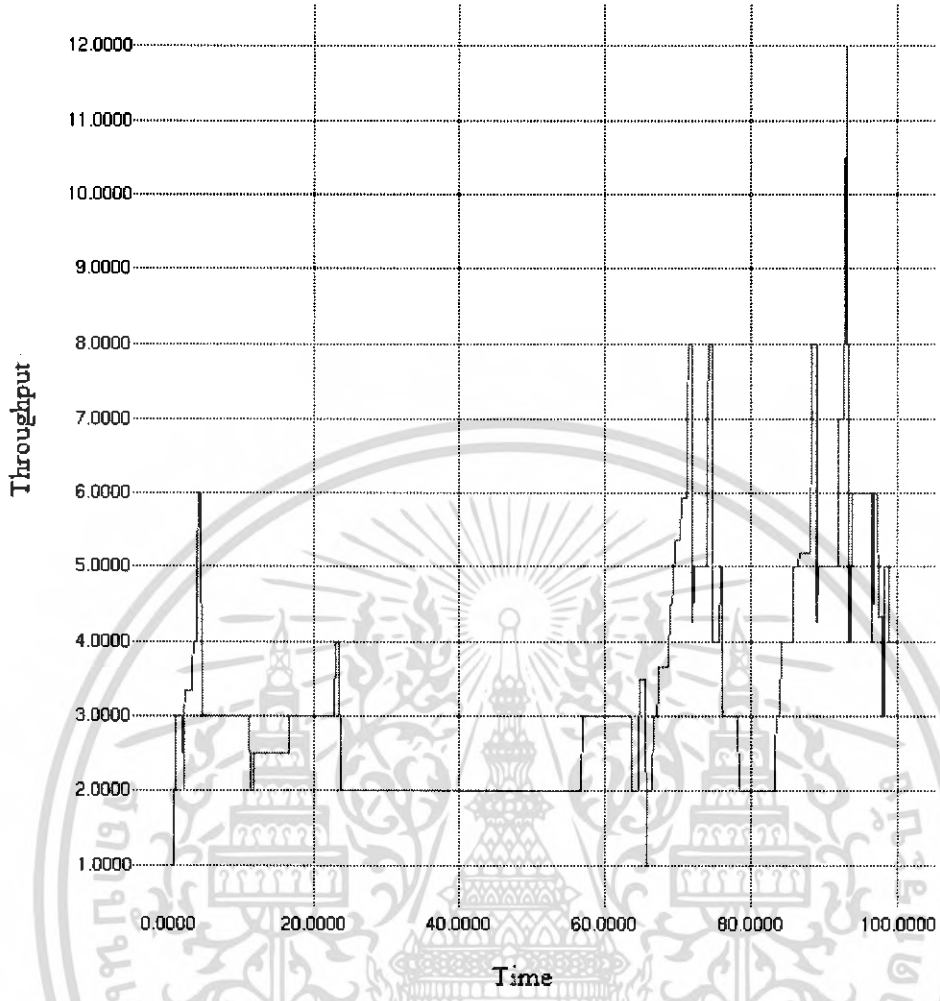
2. การทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของแพ็กเก็ต และใช้การจัดเรียงคิวแบบ RED

การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ในการจัดเรียงคิวจะใช้การจัดเรียงคิวแบบ RED โดยทำการปรับขนาดของคิวให้มีค่าเท่ากับ 10, 20, 30, 40 และ 50 ตามลำดับ และขนาดของแพ็กเก็ตจะกำหนดไว้ที่ 500 ไบต์ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.22, 3.23และเมื่อปรับให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 นั้น ผลที่ได้คือ ค่าทรูพุตนั้นคงที่ ดังนั้น กราฟที่ได้จะเหมือนกันดังแสดงไว้ในรูปที่ 3.24



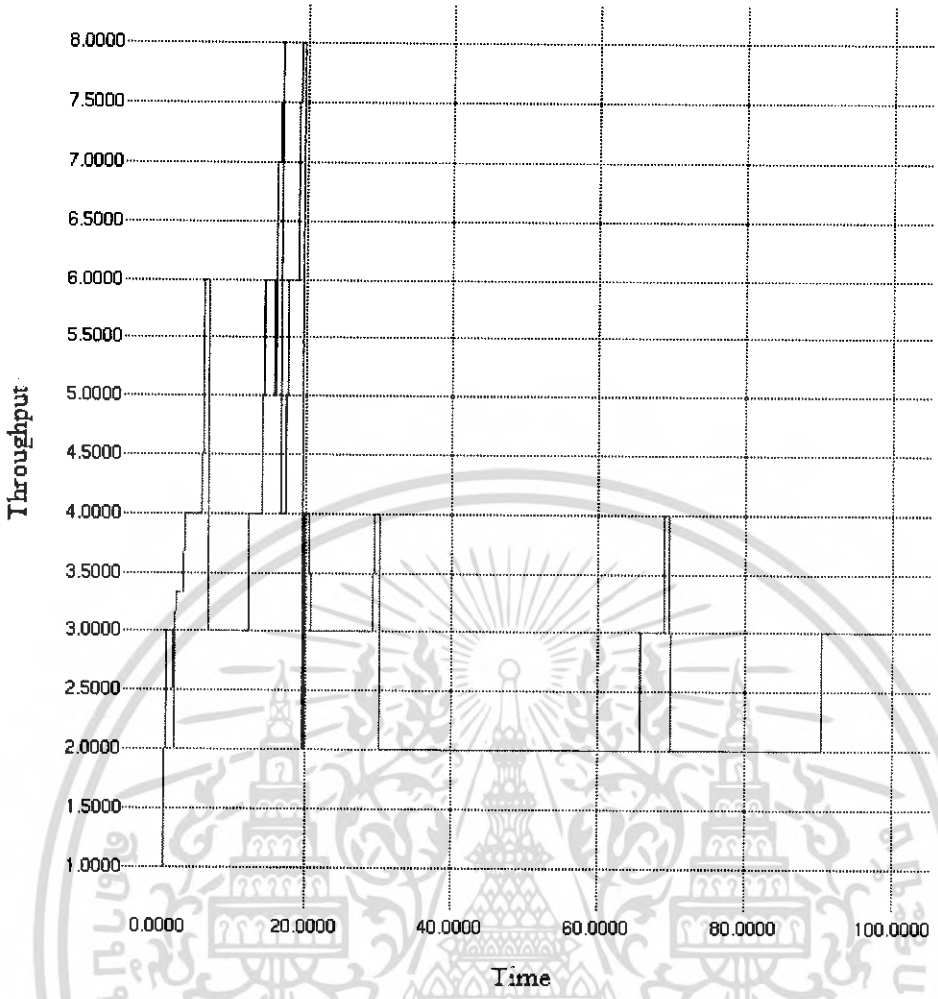
รูปที่ 3.22 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.22 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10โดยใช้การจัดเรียงคิวแบบ RED ซึ่งจะกำหนดให้ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ โดยที่ Vegas TCP นั้นจะมีกลไกควบคุมปริมาณแพ็กเก็ตในเราเตอร์ตลอดเวลา ดังนั้นเมื่อมีปริมาณแพ็กเก็ตในเราเตอร์เพิ่มมากขึ้น Vegas TCP ก็จะทำการลดปริมาณการส่งแพ็กเก็ตลงจนถึงขั้นหยุดส่ง ส่งผลให้การสูญหายของแพ็กเก็ตน้อยลงไปด้วย ดังนั้นค่าทรูพุตที่ได้นั้นจะมีช่วงเวลาที่กว้าง และมีค่าทรูพุตต่ำสุดในระดับที่คงที่



รูปที่ 3.23 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.23 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20โดยใช้การจัดเรียงคิวแบบ RED ซึ่งจะกำหนดให้ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ โดยที่การทำงานจะเหมือนกับรูปที่ 3.22 ค่าทรูพุตที่ได้นั้นในช่วงเวลาที่ 0 ถึง 20 วินาทีจะมีค่าสูงกว่ารูปที่ 3.22 และมีค่าทรูพุตต่ำสุดในระดับที่คงที่



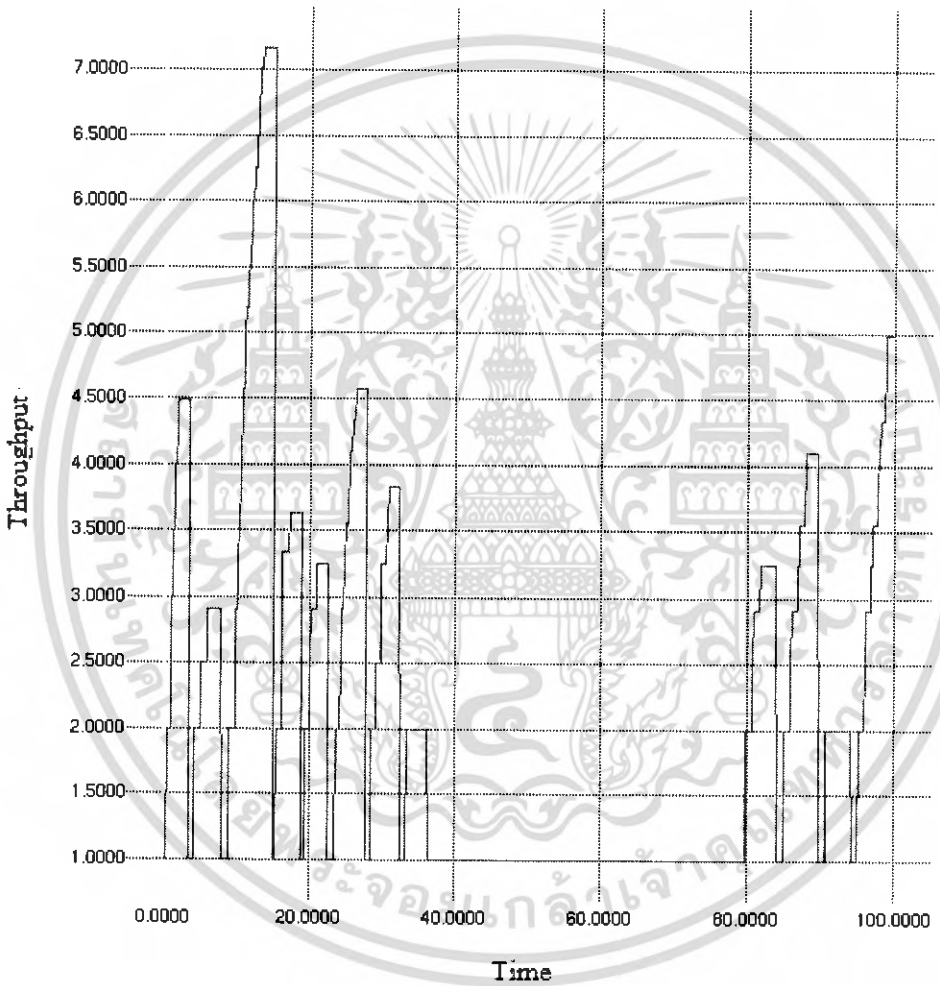
รูปที่ 3.24 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ RED และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.24 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 30,40 และ 50 โดยใช้การจัดเรียงคิวแบบ RED ซึ่งจะกำหนดให้ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ โดยที่การทำงานจะเหมือนกับรูปที่ 3.22 แต่เมื่อทำการเพิ่มขนาดของคิวเท่ากับ 30 จนถึง 50 จะได้ค่าทรูพุตที่มีค่าสูงกว่าในช่วงเวลา 0 ถึง 20 วินาที เปรียบเทียบกับรูปที่ 3.22 และเมื่อเวลาผ่านไปจนถึงจุดๆหนึ่งค่าทรูพุตจะมีค่าคงที่

เมื่อทำทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ RED ซึ่งผลที่ได้แสดงดังรูปที่ 3.22 จนถึง 3.24 นั้น สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตในช่วงเวลา 0 ถึง 20 วินาที มีค่าน้อยและหลังจากนั้นที่เวลาผ่านไปจนถึงจุดๆหนึ่งค่าทรูพุตจะมีค่าคงที่เท่ากัน เมื่อเพิ่มขนาดของคิวให้มากขึ้นค่าทรูพุตจะมีค่ามากขึ้นในช่วงเวลาที่ 0 ถึง 20 วินาที

3. การทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของแพ็กเก็ต และใช้การจัดเรียงคิวแบบ DropTail

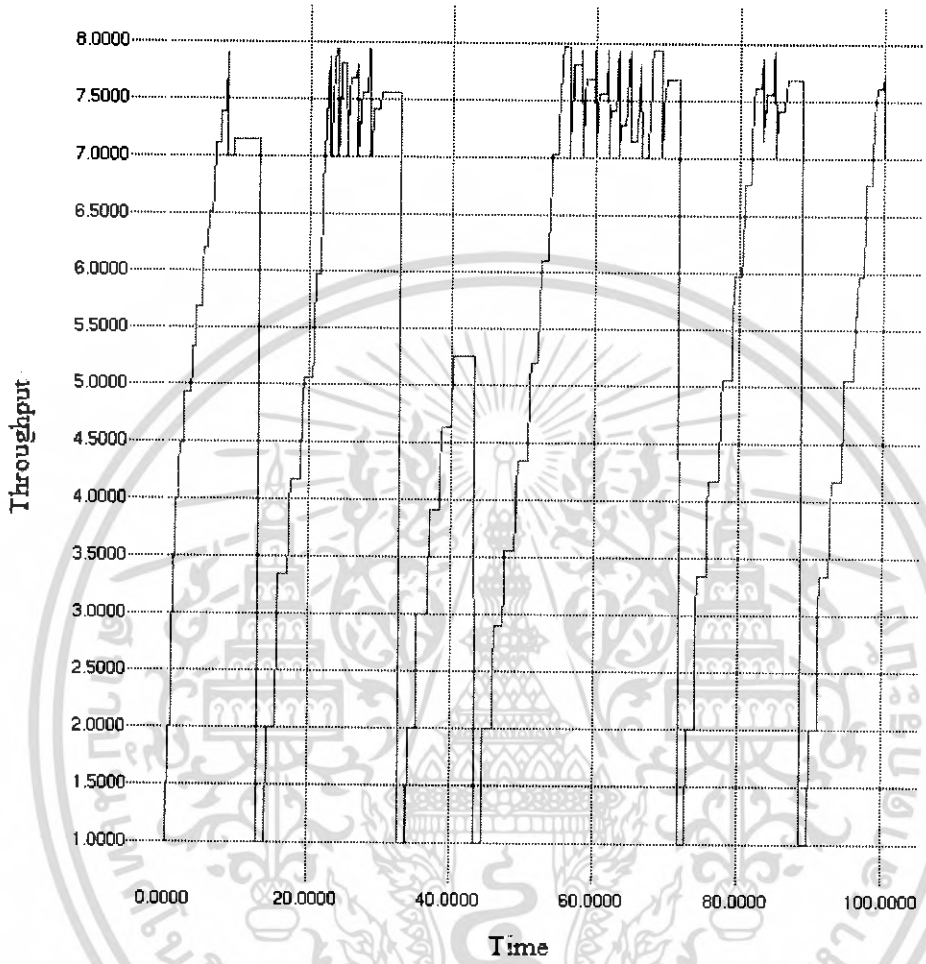
การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ในการจัดเรียงคิวจะใช้การจัดเรียงคิวแบบ DropTail โดยทำการปรับขนาดของคิวให้มีค่าเท่ากับ 10, 20, 30, 40 และ 50 ตามลำดับ และขนาดของแพ็กเก็ตจะกำหนดไว้ที่ 500 ไบต์ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.25, 3.26และเมื่อปรับให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 นั้น ผลที่ได้คือ ค่าทรูพุตนั้นคงที่ ดังนั้น กราฟที่ได้จะเหมือนกันดังแสดงไว้ในรูปที่ 3.27



รูปที่ 3.25 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ DropTail และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

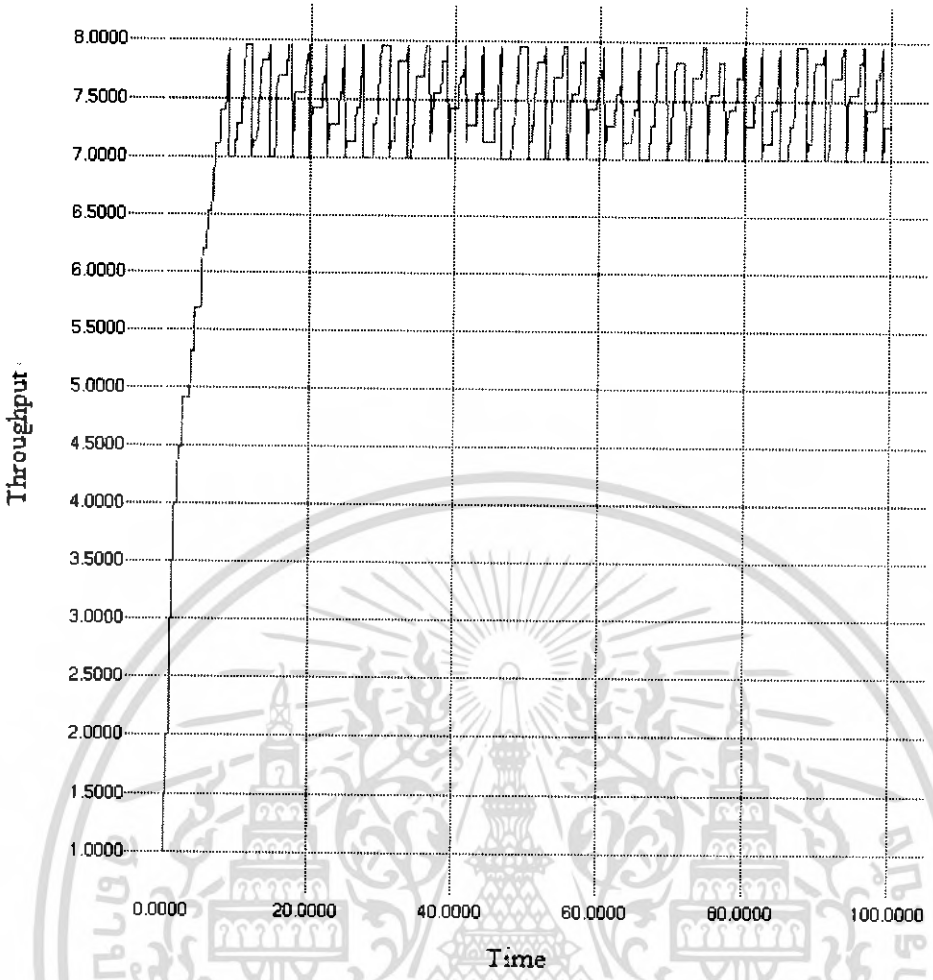
จากรูปที่ 3.25 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ DropTail ซึ่งกำหนดขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ โดยที่ SACK TCP นั้นจะเริ่มต้นส่งแพ็กเก็ตแบบ Slow Start หรือเป็นการเริ่มต้นส่งออกไปอย่างช้าๆ และจะเพิ่มจำนวนแพ็กเก็ตที่จะส่งออกไปให้มากขึ้นเรื่อยๆ ซึ่งจะส่งผลให้ค่าทรูพุตที่ได้นั้นมีการเพิ่มขึ้นเรื่อยๆ และเมื่อเกิดความคับคั่ง

ที่เราเตอร์มากๆ จนทำให้มีแพ็กเก็ตสูญหาย จากนั้น SACK TCP จะทำการ Fast Recovery โดยจะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ในการส่ง และหลังจากนั้นจะทำการ Fast Retransmission ซึ่งจะเห็นได้ว่าค่าทรูพุตที่ได้นั้นจะมีค่าลดลง และจะเป็นเช่นนี้ไปตลอด



รูปที่ 3.26 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ DropTail และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.26 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20 โดยใช้การจัดเรียงคิวแบบ DropTail และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.25 แต่เมื่อเพิ่มค่าขนาดของคิวให้มีค่าเท่ากับ 20 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลาที่ 0 ถึง 20 วินาที นั้นจะมีค่าที่สูงขึ้นและหลังจากนั้นจะมีค่าทรูพุตที่ไม่คงที่



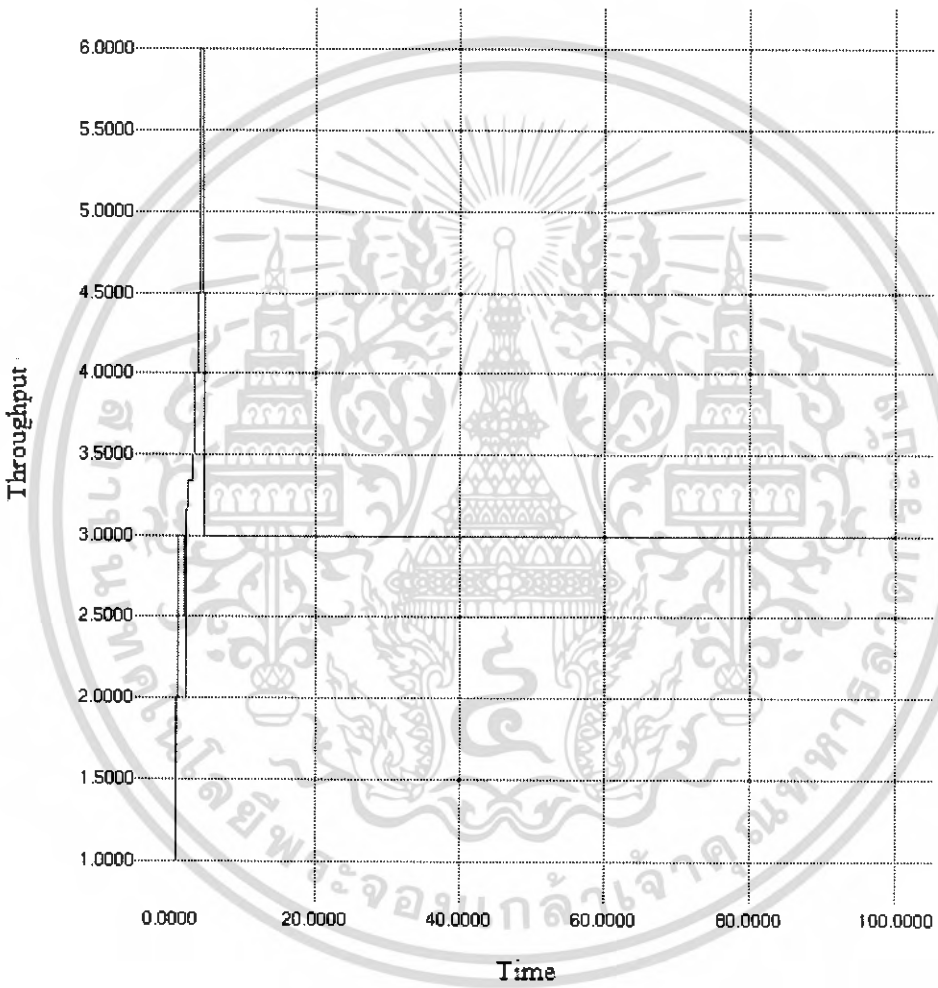
รูปที่ 3.27 กราฟแสดงค่าทรูพุตของ SACK TCP ที่มีขนาดของคิวเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.27 แสดงถึงค่าทรูพุตของ SACK TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail และกำหนดให้ขนาดของแพ็กเก็ตมีค่าเท่ากับ 500 ไบต์ โดยที่หลักการทำงานนั้นจะเหมือนกับในรูปที่ 3.25 แต่เมื่อเพิ่มค่าขนาดของคิวให้มีค่าเท่ากับ 30 จนถึง 50 แล้วนั้น ค่าทรูพุตที่ได้ในช่วงเวลาที่ 0 ถึง 20 วินาที นั้นจะมีค่าที่สูงขึ้นและหลังจากนั้นจะมีค่าทรูพุตที่ไม่คงที่

จากการทดสอบ SACK TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ DropTail และผลที่ได้นั้นแสดงดังรูปที่ 3.25 ถึง 3.27 สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที นั้นมีค่าน้อยและเมื่อเวลาผ่านไปจะมีค่าไม่คงที่ แต่เมื่อเพิ่มขนาดของคิวให้มากขึ้นค่าทรูพุตที่ช่วงเวลา 0 ถึง 20 วินาที จะมีค่าสูงขึ้นและหลังจากนั้นจะมีค่าไม่คงที่

4. การทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของแพ็กเก็ต และใช้การจัดเรียงคิวแบบ DropTail

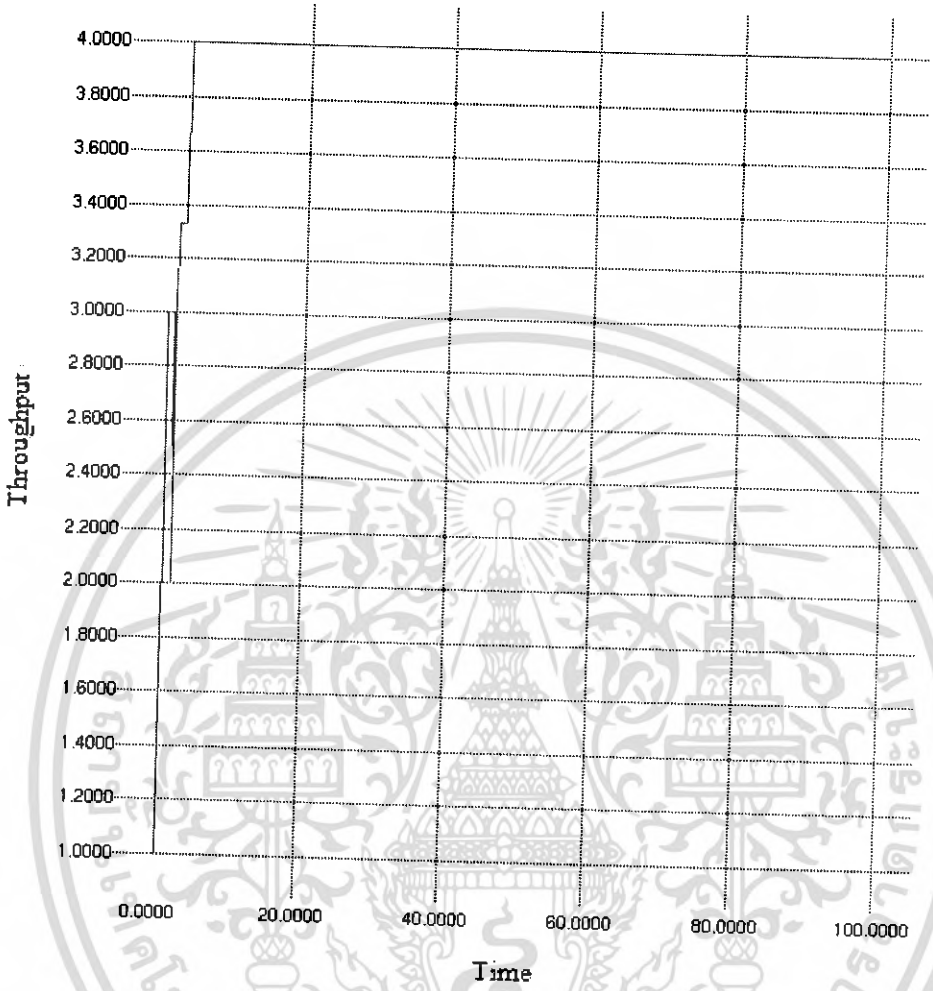
การทดสอบในส่วนนี้จะใช้ค่าพารามิเตอร์ตามที่กำหนดไว้ ในการจัดเรียงคิวจะใช้การจัดเรียงคิวแบบ DropTail โดยทำการปรับขนาดของคิวให้มีค่าเท่ากับ 10, 20, 30, 40 และ 50 ตามลำดับ และขนาดของแพ็กเก็ตจะกำหนดไว้ที่ 500 ไบต์ โดยค่าทรูพุตที่ได้นั้นดังแสดงในรูปที่ 3.28 และเมื่อปรับให้ขนาดของคิวมีค่าเท่ากับ 20, 30, 40 และ 50 นั้น ผลที่ได้คือ ค่าทรูพุตนั้นคงที่ ดังนั้น กราฟที่ได้จะเหมือนกันดังแสดงไว้ในรูปที่ 3.29



รูปที่ 3.28 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 10 โดยใช้การจัดเรียงคิวแบบ DropTail และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.28 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 10โดยใช้การจัดเรียงคิวแบบ DropTail ซึ่งจะกำหนดให้ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ โดยที่ Vegas TCP นั้นจะมีกลไกควบคุมปริมาณแพ็กเก็ตในเราเตอร์ตลอดเวลา ดังนั้นเมื่อมีปริมาณแพ็กเก็ตในเราเตอร์เพิ่มมาก

ขึ้น Vegas TCP ก็จะทำการลดปริมาณการส่งแพ็กเก็ตลงจนถึงขั้นหยุดส่ง ส่งผลให้การสูญหายของแพ็กเก็ตน้อยลงไปด้วย ดังนั้นค่าทรูพุตที่ได้นั้นจะมีช่วงเวลาที่กว้าง และมีค่าทรูพุตต่ำสุดในระดับที่คงที่



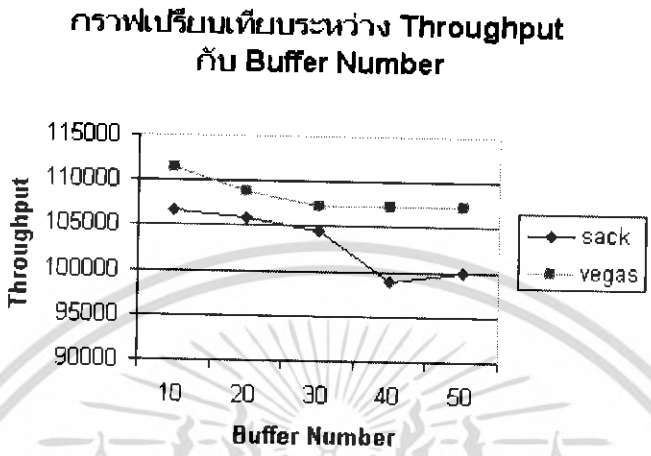
รูปที่ 3.29 กราฟแสดงค่าทรูพุตของ Vegas TCP ที่มีขนาดของคิวเท่ากับ 20, 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail และ ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.29 แสดงถึงค่าทรูพุตของ Vegas TCP ที่กำหนดให้ขนาดของคิวมีค่าเท่ากับ 20, 30, 40 และ 50 โดยใช้การจัดเรียงคิวแบบ DropTail ซึ่งจะกำหนดให้ขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์ โดยที่การทำงานจะเหมือนกับรูปที่ 3.28 แต่เมื่อทำการเพิ่มขนาดของคิวให้มีขนาดอยู่ที่ 20 จนถึง 50 นั้น ค่าทรูพุตในช่วงเวลาสั้นๆ จะมีค่าที่ต่ำกว่าเมื่อเปรียบเทียบกับรูปที่ 3.28 และหลังจากนั้นค่าทรูพุตที่ได้จะมีค่าคงที่และสูงกว่าเมื่อเปรียบเทียบกับรูปที่ 3.28

เมื่อทำทดสอบ Vegas TCP โดยทำการเปลี่ยนขนาดของคิวตั้งแต่ 10 จนถึง 50 โดยใช้การจัดเรียงคิวแบบ DropTail ซึ่งผลที่ได้แสดงดังรูปที่ 3.28 และ 3.29 นั้น สามารถสรุปได้ว่า ที่ขนาดของคิวมีค่าน้อยนั้นจะทำให้ค่าทรูพุตในช่วงเวลาสั้นๆ มีค่าสูงกว่าเมื่อเพิ่มขนาดของคิวให้มากขึ้นแต่เมื่อเวลาผ่านไปค่าทรูพุตจะมีค่าคงที่และน้อยกว่าเมื่อเพิ่มขนาดของคิวให้มากขึ้น

3.4 การวิเคราะห์และเปรียบเทียบประสิทธิภาพของ Vegas TCP และ SACK TCP เมื่อพิจารณาถึงค่าทรูพุตและขนาดของคิว

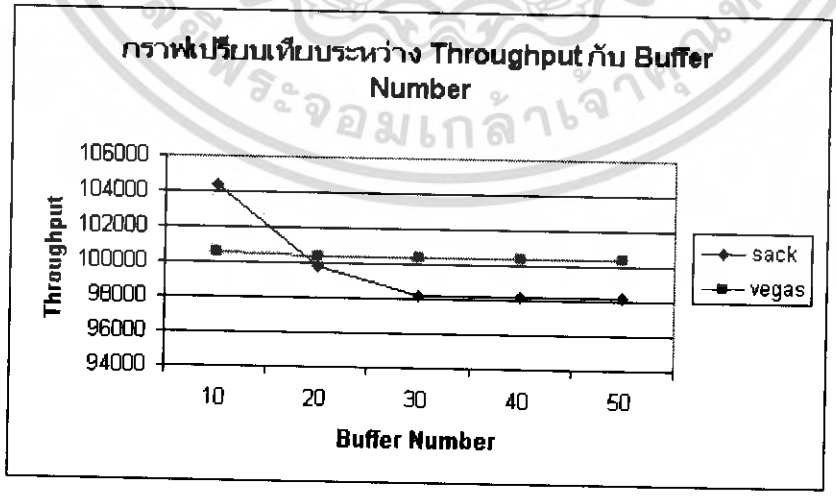
3.4.1 ใช้การจัดเรียงคิวแบบ RED โดยมีขนาดของแพ็กเก็ตเท่ากับ 1,000 ไบต์



รูปที่ 3.30 กราฟแสดงการเปรียบเทียบค่าทรูพุตกับขนาดของคิว ใช้การจัดเรียงคิวแบบ RED ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

จากรูปที่ 3.30 นั้นจะแสดงถึงกราฟที่แสดงค่าทรูพุตกับขนาดของคิวที่กำหนดไว้ที่ 10-50 ของ TCP ชนิด Vegas TCP และ SACK TCP โดยที่จะใช้การจัดเรียงคิวแบบ RED กำหนดขนาดของแพ็กเก็ตไว้ที่ 1,000 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y แสดงถึงค่าทรูพุต ซึ่งจากการเปรียบเทียบกันแล้วนั้นจะเห็นได้ว่า ค่าทรูพุตของ Vegas TCP นั้นจะสูงกว่า SACK TCP โดยตลอด

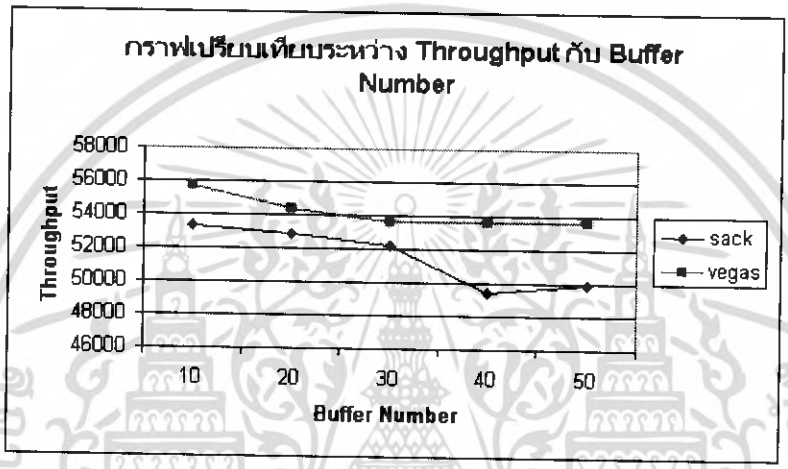
3.4.2 ใช้การจัดเรียงคิวแบบ DropTail โดยมีขนาดของแพ็กเก็ตเท่ากับ 1,000 ไบต์



รูปที่ 3.31 กราฟแสดงการเปรียบเทียบค่าทรูพุตกับขนาดของคิว ใช้การจัดเรียงคิวแบบ DropTail

จากรูปที่ 3.31 นั้นจะแสดงถึงกราฟที่แสดงค่าทรูพุตกับขนาดของคิวที่กำหนดไว้ที่ 10-50 ของ TCP ชนิด Vegas TCP และ SACK TCP โดยที่จะใช้การจัดเรียงคิวแบบ DropTail กำหนดขนาดของแพ็กเก็ตไว้ที่ 1,000 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y ซึ่งจากการเปรียบเทียบกันแล้วนั้นจะเห็นได้ว่า ค่าทรูพุตของ Vegas TCP นั้นจะต่ำกว่า SACK TCP เฉพาะในช่วงที่ขนาดของคิวมีค่าเท่ากับ 10 เท่านั้น เมื่อขนาดของคิวมีค่าเท่ากับ 20- 50 นั้นค่าทรูพุตของ Vegas TCP นั้นจะมีค่าที่สูงกว่า SACK TCP โดยตลอด และยังสังเกตได้อีกว่า ค่าทรูพุตของ Vegas TCP นั้นค่อนข้างที่จะคงที่ จะมีการเปลี่ยนแปลงเพียงเล็กน้อยเท่านั้น

3.4.3 ใช้การจัดเรียงคิวแบบ RED โดยมีขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

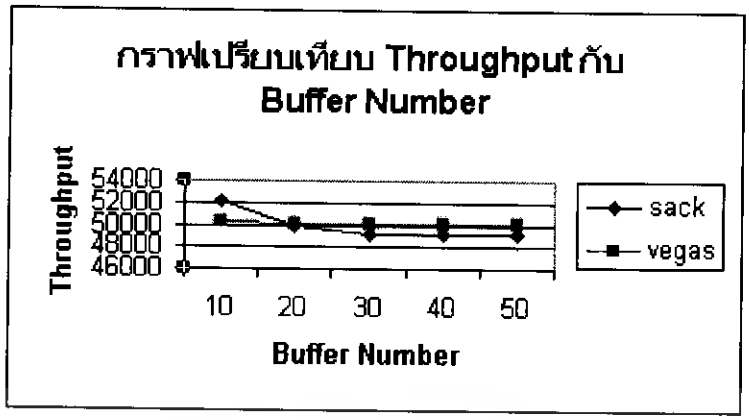


รูปที่ 3.32 กราฟแสดงการเปรียบเทียบค่าทรูพุตกับขนาดของคิว โดยใช้การจัดเรียงคิวแบบ RED ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.32 นั้นจะแสดงถึงกราฟที่แสดงค่าทรูพุตกับขนาดของคิวที่กำหนดไว้ที่ 10-50 ของ TCP ชนิด Vegas TCP และ SACK TCP โดยที่จะใช้การจัดเรียงคิวแบบ RED กำหนดขนาดของแพ็กเก็ตไว้ที่ 500 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y ซึ่งจากการเปรียบเทียบกันแล้วนั้นจะเห็นได้ว่า ค่าทรูพุตของ Vegas TCP นั้นจะสูงกว่า SACK TCP โดยตลอด

3.4.4 ใช้การจัดเรียงคิวแบบ DropTail โดยมีขนาดของแพ็กเก็ตเท่ากับ 500 ไบต์

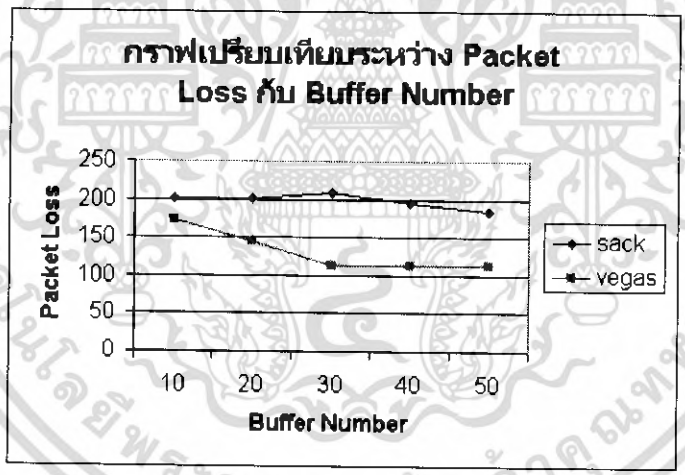
จากรูปที่ 3.31 นั้นจะแสดงถึงกราฟที่แสดงค่าทรูพุตกับขนาดของคิวที่กำหนดไว้ที่ 10-50 ของ TCP ชนิด Vegas TCP และ SACK TCP โดยที่จะใช้การจัดเรียงคิวแบบ DropTail กำหนดขนาดของแพ็กเก็ตไว้ที่ 500 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y ซึ่งจากการเปรียบเทียบกันแล้วนั้นจะเห็นได้ว่า ค่าทรูพุตของ Vegas TCP นั้นจะต่ำกว่า SACK TCP เฉพาะในช่วงที่ขนาดของคิวมีค่าเท่ากับ 10 เท่านั้น เมื่อขนาดของคิวมีค่าเท่ากับ 20- 50 นั้นค่าทรูพุตของ Vegas TCP นั้นจะมีค่าที่สูงกว่า SACK TCP โดยตลอด และยังสังเกตได้อีกว่า ค่าทรูพุตของ Vegas TCP นั้นค่อนข้างที่จะคงที่ จะมีการเปลี่ยนแปลงเพียงเล็กน้อยเท่านั้น



รูปที่ 3.33 กราฟแสดงการเปรียบเทียบค่าทรูพุตกับขนาดของคิว ใช้การจัดเรียงคิวแบบ DropTail ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

3.5 การวิเคราะห์และเปรียบเทียบประสิทธิภาพเมื่อพิจารณาถึงจำนวนแพ็กเก็ตที่สูญหายกับจำนวนแพ็กเก็ตที่ส่งทั้งหมด

3.5.1 ใช้การจัดเรียงคิวแบบ RED โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

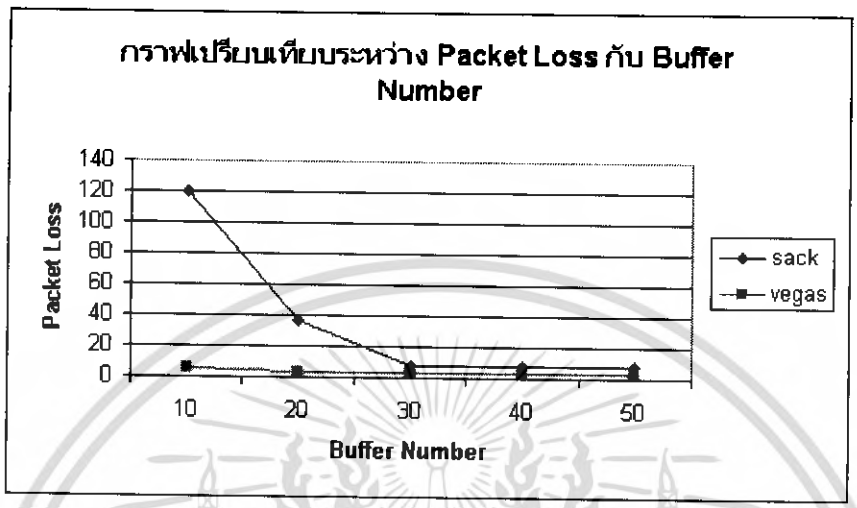


รูปที่ 3.34 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่สูญหายกับจำนวนแพ็กเก็ตที่ส่งทั้งหมด โดยใช้การจัดเรียงคิวแบบ RED ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

จากรูปที่ 3.34 นั้นแสดงถึงกราฟที่แสดงค่าระหว่างขนาดของคิวกับจำนวนของแพ็กเก็ตที่เกิดการสูญหายไป ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดขนาดของแพ็กเก็ตไว้ที่ 1,000 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y แสดงถึงจำนวนแพ็กเก็ตที่เกิดการสูญหาย จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่า ที่ขนาดของคิวเท่ากันนั้น SACK TCP นั้นจะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายไปมากกว่า Vegas TCP โดยตลอด และที่

ขนาดของคิวที่มากขึ้นนั้นจะเห็นได้ว่า Vegas TCP นั้น จะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายลดน้อยลงเรื่อยๆ จนกระทั่งคงที่

3.5.2 ใช้การจัดเรียงคิวแบบ DropTail โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์



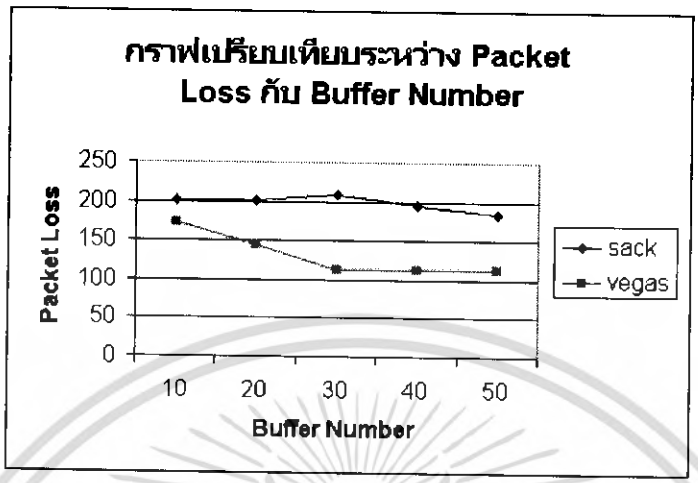
รูปที่ 3.35 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่สูญหายกับจำนวนแพ็กเก็ตที่ส่งทั้งหมด โดยใช้การจัดเรียงคิวแบบ DropTail ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

จากรูปที่ 3.35 นั้นแสดงถึงกราฟที่แสดงค่าระหว่างขนาดของคิวกับจำนวนของแพ็กเก็ตที่เกิดการสูญหายไป ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ DropTail และกำหนดขนาดของแพ็กเก็ตไว้ที่ 1,000 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y แสดงถึงจำนวนแพ็กเก็ตที่เกิดการสูญหาย จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่า ที่ขนาดของคิวเท่ากันนั้น SACK TCP นั้นจะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายไปมากกว่า Vegas TCP โดยตลอด และที่ขนาดของคิวที่มากขึ้นนั้นจะเห็นได้ว่า Vegas TCP นั้น จะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายลดน้อยลงจนกระทั่งคงที่ ซึ่งเมื่อใช้การจัดเรียงคิวแบบ DropTail นั้นจะสังเกตเห็นได้ว่า Vegas TCP นั้นจะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายไปน้อยมาก ส่วน SACK TCP นั้น เมื่อขนาดของคิวมีค่าเท่ากับ 10 นั้นจำนวนแพ็กเก็ตที่เกิดการสูญหายนั้นจะมีมาก และเมื่อขนาดของคิวเพิ่มขึ้น จำนวนแพ็กเก็ตที่เกิดการสูญหายนั้นจะมีจำนวนที่ลดน้อยลงเรื่อยๆ จนกระทั่งคงที่ ที่ขนาดของคิวเท่ากับ 30 จนถึง 50

3.5.3 ใช้การจัดเรียงคิวแบบ RED โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

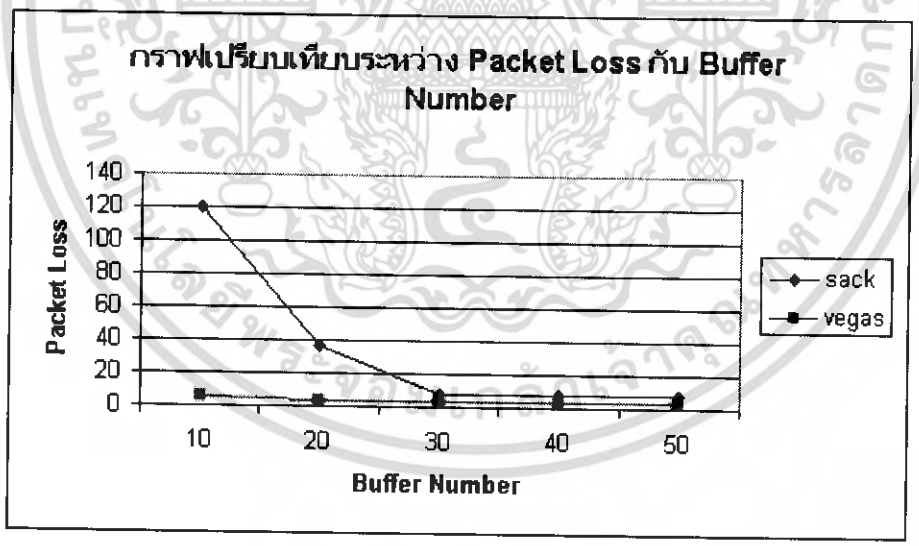
จากรูปที่ 3.36 นั้นแสดงถึงกราฟที่แสดงค่าระหว่างขนาดของคิวกับจำนวนของแพ็กเก็ตที่เกิดการสูญหายไป ซึ่งในที่นี้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดขนาดของแพ็กเก็ตไว้ที่ 500 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y แสดงถึงจำนวนแพ็กเก็ตที่เกิดการสูญหาย จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่า ที่ขนาดของคิวเท่ากันนั้น SACK TCP นั้นจะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายไปมากกว่า Vegas TCP โดยตลอด และที่ขนาดของ

คิวที่มากขึ้นนั้นจะเห็นได้ว่า Vegas TCP นั้น จะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายลดน้อยลงเรื่อยๆ จนกระทั่งคงที่



รูปที่ 3.36 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่สูญหายกับจำนวนแพ็กเก็ตที่ส่งทั้งหมด โดยใช้การจัดเรียงคิวแบบ RED ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

3.5.4 ใช้การจัดเรียงคิวแบบ DropTail โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์



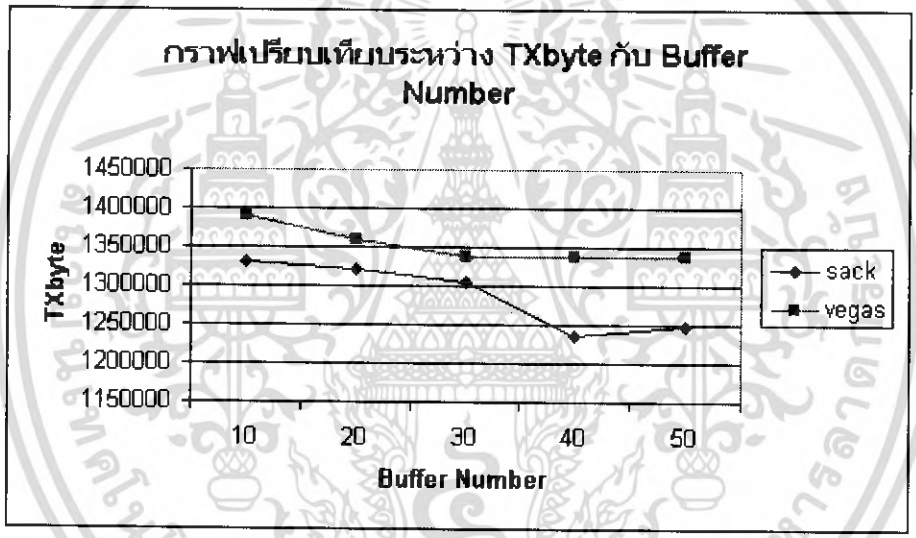
รูปที่ 3.37 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่สูญหายกับจำนวนแพ็กเก็ตที่ส่งทั้งหมด โดยใช้การจัดเรียงคิวแบบ DropTail ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.37 นั้นแสดงถึงกราฟที่แสดงค่าระหว่างขนาดของคิวกับจำนวนของแพ็กเก็ตที่เกิดการสูญหายไป ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ DropTail และกำหนดขนาดของแพ็กเก็ตไว้ที่ 500 ไบต์ โดยกำหนดให้แกน x แสดงถึงขนาดของคิว และแกน y แสดงถึง

จำนวนแพ็กเก็ตที่เกิดการสูญหาย จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่า ที่ขนาดของคิวเท่ากันนั้น SACK TCP นั้นจะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายไปมากกว่า Vegas TCP โดยตลอด และที่ขนาดของคิวที่มากขึ้นนั้นจะเห็นได้ว่า Vegas TCP นั้น จะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายลดน้อยลงจนกระทั่งคงที่ ซึ่งเมื่อใช้การจัดเรียงคิวแบบ DropTail นั้นจะสังเกตเห็นได้ว่า Vegas TCP นั้นจะมีจำนวนแพ็กเก็ตที่เกิดการสูญหายไปน้อยมาก ส่วน SACK TCP นั้น เมื่อขนาดของคิวมีค่าเท่ากับ 10 นั้นจำนวนแพ็กเก็ตที่เกิดการสูญหายนั้นจะมีมาก และเมื่อขนาดของคิวเพิ่มขึ้น จำนวนแพ็กเก็ตที่เกิดการสูญหายนั้นจะมีจำนวนที่ลดน้อยลงเรื่อยๆ จนกระทั่งคงที่ ที่ขนาดของคิวเท่ากับ 30 จนถึง 50

3.6 การวิเคราะห์และเปรียบเทียบประสิทธิภาพเมื่อพิจารณาถึงจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว

3.6.1 ใช้การจัดเรียงคิวแบบ RED โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

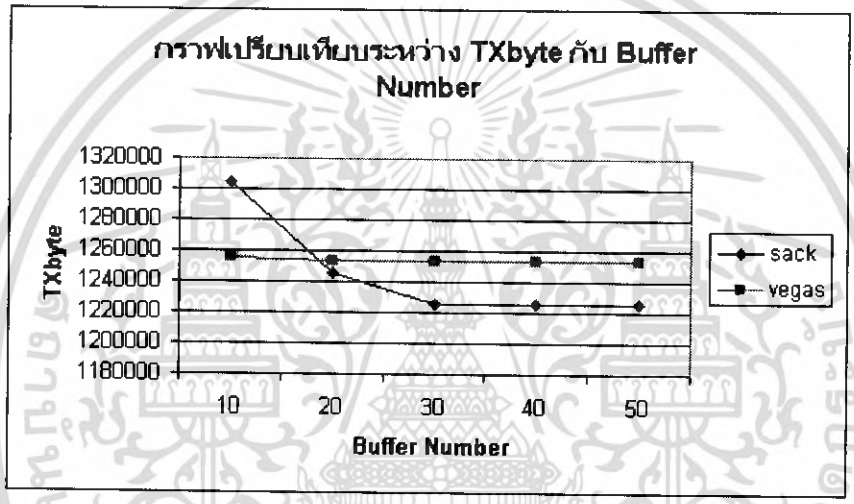


รูปที่ 3.38 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว โดยใช้การจัดเรียงคิวแบบ RED ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

จากรูปที่ 3.38 นั้นจะแสดงถึงกราฟที่แสดงค่าระหว่างจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดขนาดของแพ็กเก็ตไว้ที่ 1,000 ไบต์ โดยจะกำหนดให้แกน x คือขนาดของคิว และแกน y คือจำนวนของแพ็กเก็ตที่ส่งได้สำเร็จ จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่าในขนาดของคิวที่เท่ากันนั้นจำนวนของแพ็กเก็ตที่ส่งได้สำเร็จของ Vegas TCP นั้นมีจำนวนที่มากกว่า SACK TCP โดยตลอด และจากรูปที่ 3.38 ยังสามารถบอกได้อีกว่าที่ขนาดของคิวนั้นมีค่ามากขึ้นจำนวนแพ็กเก็ตที่ส่งได้สำเร็จนั้นก็จะยิ่งมีค่าที่น้อยลง

3.6.2 ใช้การจัดเรียงคิวแบบ DropTail โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

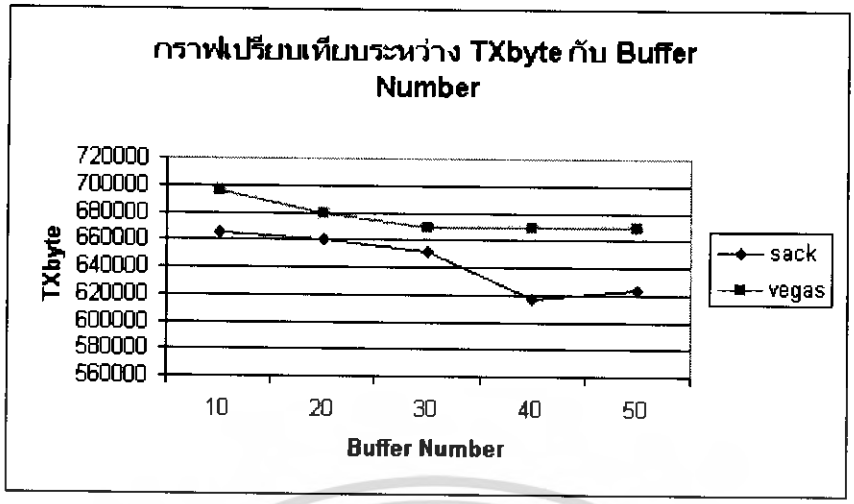
จากรูปที่ 3.39 นั้นจะแสดงถึงกราฟที่แสดงค่าระหว่างจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ DropTail และกำหนดขนาดของแพ็กเก็ตไว้ที่ 1,000 ไบต์ โดยจะกำหนดให้แกน x คือขนาดของคิว และแกน y คือจำนวนของแพ็กเก็ตที่ส่งได้สำเร็จ จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่าที่ขนาดของคิวมีค่าเท่ากับ 10 นั้น SACK TCP จะมีจำนวนแพ็กเก็ตที่ส่งได้สำเร็จมากกว่า Vegas TCP และเมื่อขนาดของคิวมีค่าเพิ่มขึ้นเป็น 20 จนถึง 50 นั้นจะเห็นได้ว่า Vegas TCP จะมีจำนวนแพ็กเก็ตที่ส่งได้สำเร็จมากกว่า SACK TCP โดยตลอด และยังสังเกตเห็นได้อีกว่าจำนวนแพ็กเก็ตที่ส่งได้สำเร็จของ Vegas TCP นั้นมีการเปลี่ยนแปลงค่อนข้างน้อยไม่เหมือนกับ SACK TCP



รูปที่ 3.39 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว โดยใช้การจัดเรียงคิวแบบ DropTail ขนาดแพ็กเก็ตเท่ากับ 1,000 ไบต์

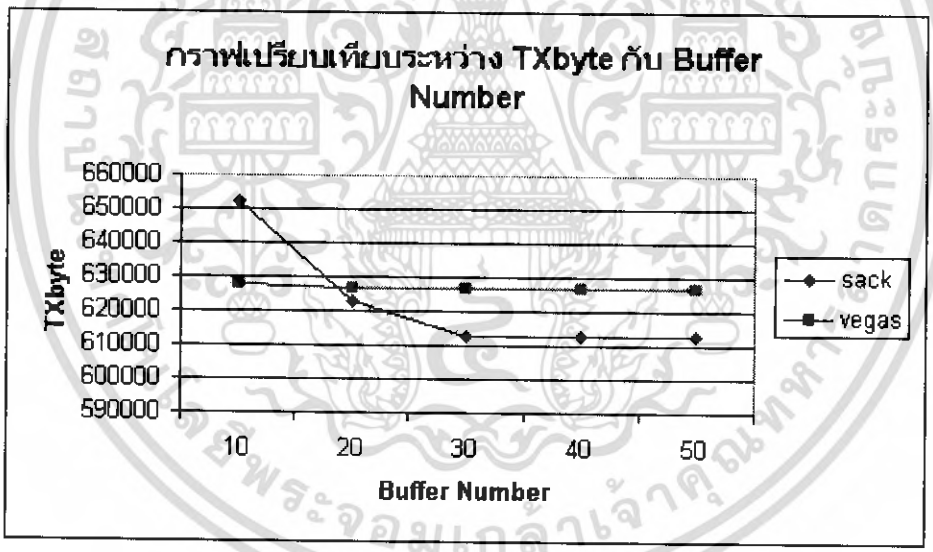
3.6.3 ใช้การจัดเรียงคิวแบบ RED โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.40 นั้นจะแสดงถึงกราฟที่แสดงค่าระหว่างจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ RED และกำหนดขนาดของแพ็กเก็ตไว้ที่ 500 ไบต์ โดยจะกำหนดให้แกน x คือขนาดของคิว และแกน y คือจำนวนของแพ็กเก็ตที่ส่งได้สำเร็จ จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่าในขนาดของคิวที่เท่ากันนั้นจำนวนของแพ็กเก็ตที่ส่งได้สำเร็จของ Vegas TCP นั้นมีจำนวนที่มากกว่า SACK TCP โดยตลอด และจากรูปที่ 3.38 ยังสามารถบอกได้อีกว่าที่ขนาดของคิวนั้นมีค่ามากขึ้นจำนวนแพ็กเก็ตที่ส่งได้สำเร็จนั้นก็จะยิ่งมีค่าที่น้อยลง



รูปที่ 3.40 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว โดยใช้การจัดเรียงคิวแบบ RED ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

3.6.4 ใช้การจัดเรียงคิวแบบ DropTail โดยกำหนดให้ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์



รูปที่ 3.41 กราฟแสดงการเปรียบเทียบจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว โดยใช้การจัดเรียงคิวแบบ DropTail ขนาดแพ็กเก็ตเท่ากับ 500 ไบต์

จากรูปที่ 3.41 นั้นจะแสดงถึงกราฟที่แสดงค่าระหว่างจำนวนแพ็กเก็ตที่ส่งได้สำเร็จกับขนาดของคิว ซึ่งในที่นี้ได้ทำการกำหนดขนาดของคิวไว้ที่ 10-50 โดยใช้การจัดเรียงคิวแบบ DropTail และกำหนดขนาดของแพ็กเก็ตไว้ที่ 500 ไบต์ โดยจะกำหนดให้แกน x คือขนาดของคิว และแกน y คือจำนวนของแพ็กเก็ตที่ส่งได้สำเร็จ จากการวิเคราะห์และเปรียบเทียบนั้นจะเห็นได้ว่าที่ขนาดของคิวมีค่าเท่ากับ 10 นั้น SACK TCP จะมีจำนวนแพ็กเก็ตที่ส่งได้สำเร็จมากกว่า Vegas TCP และเมื่อขนาดของคิวมีค่าเพิ่มขึ้นเป็น

20 จนถึง 50 นั้นจะเห็นได้ว่า Vegas TCP จะมีจำนวนแพ็กเก็ตที่ส่งได้สำเร็จมากกว่า SACK TCP โดยตลอด และยังสังเกตเห็นได้อีกว่าจำนวนแพ็กเก็ตที่ส่งได้สำเร็จของ Vegas TCP นั้นมีการเปลี่ยนแปลงค่อนข้างน้อยไม่เหมือนกับ SACK TCP

ความหมายและวิธีการวัดค่า ทรูพุต

ทรูพุต คือ อัตราส่วนจำนวนบิตข้อมูลทั้งหมดที่สามารถรับส่งได้สำเร็จแล้วระหว่างสถานีส่งกับสถานีรับต่อเวลาทั้งหมดที่ใช้ในการรับส่ง ซึ่งสามารถเขียนได้ดังสมการที่ (3.2) ดังต่อไปนี้

$$Throughput = \frac{Number\ of\ bits\ received}{Frist\ Packet's\ sended - Last\ Packet's\ received} \quad (3.2)$$

จากการทดสอบทั้งหมดสรุปได้ว่า

SACK TCP จะมีค่าทรูพุตลดลงอย่างรวดเร็ว เนื่องจากการทำงานของ SACK TCP นั้น จะเริ่มต้นการส่งแพ็กเก็ตแบบ Slow Start ซึ่งเป็นการเริ่มต้นการทำงานอย่างช้าๆ และจะทำการเพิ่มจำนวนแพ็กเก็ตข้อมูลที่ส่งขึ้นไปเรื่อยๆ แบบเอกซโปเนนเชียลเมื่อได้รับ ACK ตอบกลับมา จากนั้นเมื่อมีแพ็กเก็ตข้อมูลถูกส่งออกไปมากจะทำให้แพ็กเก็ตนั้นต้องรอการจัดเรียงคิว เมื่อมีการจัดเรียงคิวเต็มแล้วแพ็กเก็ตที่ถูกจัดเรียงคิวก็จะถูกทิ้งไป จากนั้น SACK TCP จะใช้ค่า pipe ในการพิจารณาจำนวนแพ็กเก็ตที่ใช้ในการส่งในระหว่างที่อยู่ในกระบวนการ Fast Recovery โดยที่ค่า pipe จะมีค่าเท่ากับ

$$pipe = cwnd - ndup \quad (3.1)$$

โดยที่ cwnd คือ Congestion Window และ ndup คือ จำนวน dup ACK โดยที่ค่า pipe จะถูกเพิ่มค่าขึ้น 1 ค่า เมื่อมีการส่งแพ็กเก็ตออกไป และจะถูกลดค่าลง 1 ค่า เมื่อมีการรับ dup ACK และเมื่อ SACK TCP นั้นออกจากการ Fast Recovery แล้ว จะเข้าสู่กระบวนการ Congestion Avoidance

จากการใช้การจัดเรียงคิวแบบ RED และแบบ DropTail ใน SACK TCP นั้นจะเห็นได้ว่าการจัดเรียงคิวแบบ DropTail นั้นจะมีการสูญหายของแพ็กเก็ตน้อยกว่า แต่ขนาดของแพ็กเก็ตทั้งหมดที่ส่งนั้นจะมีค่าน้อยตามไปด้วย และเมื่อทำการเปลี่ยนขนาดของคิวที่มากขึ้นนั้นค่าทรูพุตจะมีค่าที่คงที่ แต่เมื่อเปลี่ยนขนาดของแพ็กเก็ตนั้นการสูญหายของแพ็กเก็ตจะเท่าเดิม โดยที่แพ็กเก็ตทั้งหมดที่ส่งออกไปจะน้อยลง

ในส่วนของ Vegas TCP นั้นจะมีค่าทรูพุตที่ค่อยๆเปลี่ยนและคงที่ในช่วงเวลาสั้นๆ เนื่องจาก Vegas TCP นั้นจะมีการควบคุมความคับคั่งตลอดเวลา ดังนั้นเมื่อมีปริมาณแพ็กเก็ตมาก Vegas TCP ก็จะทำการลดปริมาณการส่งแพ็กเก็ตจนถึงขั้นหยุดส่ง ดังนั้นค่าทรูพุตจึงขึ้นอยู่กับปริมาณแพ็กเก็ตที่อยู่ในเราเตอร์โดยตรง หรือกล่าวได้ว่าถ้าปริมาณการเชื่อมต่อเพิ่มขึ้น ค่าทรูพุตของ Vegas TCP จะลดลง ดังนั้นเมื่อเปรียบเทียบกับ SACK TCP แล้ว Vegas TCP จะมีการสูญหายของแพ็กเก็ตน้อยกว่า SACK TCP

จากการใช้การจัดเรียงคิวแบบ RED และแบบ DropTail ใน Vegas TCP นั้นจะเห็นได้ว่าการจัดเรียงคิวแบบ DropTail นั้นจะมีการสูญหายของแพ็กเก็ตน้อยกว่า แต่ขนาดของแพ็กเก็ตทั้งหมดที่ส่งนั้นจะมีค่าน้อยตามไปด้วย และเมื่อทำการเปลี่ยนขนาดของคิวที่มากขึ้นนั้นค่าทรูพุตจะมีค่าที่คงที่ แต่เมื่อเปลี่ยนขนาดของแพ็กเก็ตนั้นการสูญหายของแพ็กเก็ตจะเท่าเดิม โดยที่แพ็กเก็ตทั้งหมดที่ส่งออกไปจะน้อยลง

การนำเอา SACK TCP และ Vegas TCP ไปใช้ในระบบเครือข่าย ซึ่งโปรโตคอลทั้งสองชนิดนี้มีข้อดีข้อเสียที่แตกต่างกัน จึงทำให้การนำไปใช้งานเพื่อให้มีประสิทธิภาพสูงสุดนั้น ถ้าต้องการรับส่งข้อมูลที่มีความสำคัญน้อยในระบบเครือข่ายนั้น เช่น เสียง (Voice) และมัลติมีเดีย ควรจะใช้ SACK TCP เนื่องจาก SACK TCP จะมีการสูญหายของแพ็กเก็ตมาก ในทางกลับกันถ้าหากมีการรับส่งข้อมูลที่มีความสำคัญมาก เช่น ข้อมูลทั่วไป ควรจะใช้ Vegas TCP เนื่องจาก Vegas TCP มีค่าทรูพุตที่คงที่และสามารถจัดการการคับคั่งของข้อมูลได้ดี

บทที่ 4
สรุปและวิจารณ์

4.1 สรุป

โครงงานนี้เป็นโครงงานซึ่งทำการทดสอบประสิทธิภาพบนโปรโตคอล TCP ซึ่งเป็นโปรโตคอลที่ใช้กันมากในปัจจุบัน โดยเฉพาะการใช้งานอินเตอร์เน็ตซึ่งในการพัฒนาระบบอินเตอร์เน็ตให้มีประสิทธิภาพที่ดีขึ้นนั้นจะต้องพิจาณาถึงองค์ประกอบที่เกี่ยวข้อง ดังเช่นในโครงงานนี้ที่ทำการทดสอบถึงองค์ประกอบที่ส่งผลกระทบต่อการทำงานของโปรโตคอล TCP โดยทำการจำลองการทำงานผ่านทางโปรแกรม NS แล้วนำผลที่ได้จากการทดสอบมาวิเคราะห์ และทำการสรุปถึงสิ่งที่มีผลกระทบต่อประสิทธิภาพในการทำงานของโปรโตคอล TCP เพื่อนำไปปรับปรุงค่าองค์ประกอบต่างๆ ให้มีความเหมาะสม ดังนั้นในการทำการทดสอบนั้น ผู้ทำการทดสอบจะต้องมีความรู้ความเข้าใจถึงรูปแบบและกลไกการทำงานของตัวโปรโตคอล

สำหรับการทดสอบในโครงงานนี้ได้เลือกทำการทดสอบกระบวนการทำงานของ TCP ชนิด SACK TCP และ Vegas TCP โดยใช้โปรแกรม NS จำลองการทำงานระบบเครือข่าย แล้วนำผลที่ได้ไปวิเคราะห์ และทำการเปรียบเทียบระหว่างโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP และสรุปถึงสิ่งที่มีผลกระทบต่อประสิทธิภาพในการทำงานของโปรโตคอล TCP ชนิด SACK TCP และ Vegas TCP

ในส่วนของอุปสรรคในการดำเนินการโครงงานนี้ ได้แก่ ปัจจัยต่างๆ ที่ส่งผลกระทบต่อประสิทธิภาพในการทำงานของโปรโตคอล TCP นั้น มีเป็นจำนวนมาก ทำให้การพิจารณาในแต่ละปัจจัยนั้นทำได้ยาก และได้รับผลที่ไม่ตรงกับความต้องการ สำหรับในส่วนของรูปแบบคำสั่งที่ใช้ในการจำลองการทำงาน ค่ากำหนดต่างๆ ที่ใช้ในการปรับเปลี่ยนในบางกรณีได้ทำการปรับเปลี่ยนแล้ว แต่ไม่เกิดการเปลี่ยนแปลง อาจเป็นเพราะค่ากำหนดมีการอธิบายที่ไม่ละเอียดพอ รวมทั้งเอกสารสำหรับอ้างอิงในส่วนของโปรแกรม NS นั้นมีไม่ครบถ้วน ทำให้เกิดอุปสรรคระหว่างการดำเนินงาน

4.2 แนวทางการพัฒนาต่อ

ในปริญญานิพนธ์นี้ ทางผู้จัดทำได้ทำการทดสอบประสิทธิภาพของ SACK TCP และ Vegas TCP โดยทำการทดสอบกับตัวกลางชนิดสายและกำหนดพารามิเตอร์บางส่วนใส่ลงไปในการทดสอบประสิทธิภาพนี้ ดังนั้นในแนวทางการพัฒนาต่อจึงควรจะกำหนดพารามิเตอร์ในส่วนที่ผู้จัดทำยังไม่ได้กำหนดลงไปในการทดสอบประสิทธิภาพนี้ อาจจะเปลี่ยนตัวกลางในการนำส่งข้อมูลเป็นแบบไร้สาย หรือนำเอาข้อเสียของ SACK TCP และ Vegas TCP ที่แสดงในการทดสอบประสิทธิภาพนี้มาวิเคราะห์แล้วทำการปรับปรุงแก้ไขโดยเขียนเป็นอัลกอริทึมขึ้นมาใหม่ แล้วนำอัลกอริทึมที่สร้างขึ้นมาใหม่ไปทดสอบประสิทธิภาพ และนำมาเปรียบเทียบกับอัลอริทึมแบบเก่าว่ามีข้อดีข้อเสียมากกว่ากันอย่างไร

ภาคผนวก

ภาคผนวก

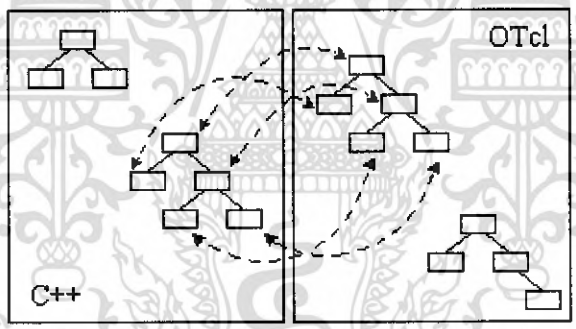
เครื่องมือสำหรับจำลองการทำงาน

1. โปรแกรม NS

โปรแกรม NS (Network Simulator) เป็นโปรแกรมที่ใช้สำหรับการจำลองระบบเครือข่ายที่นิยมใช้กันอย่างมากในกลุ่มผู้ที่ทำการพัฒนาระบบเครือข่าย โดยโปรแกรม NS นั้นสามารถทำงานได้ทั้งบนระบบยูนิกซ์ (เช่น ลีนุกซ์, FreeBSD เป็นต้น) และระบบวินโดว์และยังสนับสนุนการจำลองในหลายๆ รูปแบบด้วยกัน ดังเช่น โปรโตคอล TCP และ UDP การใช้ในงานประยุกต์ เช่น FTP, Telnet ระบบการจัดคิว เช่น DropTail, RED รวมทั้งการใช้งานในแบบมัลติคาสต์ทั้งบนระบบเครือข่ายทั่วไป (Wired Network) และในระบบเครือข่ายไร้สาย (Wireless Network)

โปรแกรม NS ถูกพัฒนามาจากภาษา C++ และภาษา OTcl (เป็นภาษา Tcl ที่มีลักษณะการเขียนโปรแกรมเป็นแบบเชิงวัตถุ) โดยจะใช้ภาษา C++ ในการจัดการกับตัวข้อมูล ส่วนภาษา OTcl จะใช้ในการการควบคุมกระบวนการ โดยที่จะมีการเชื่อมโยงระหว่าง 2 โปรแกรม ซึ่งจะใช้ฟังก์ชันการทำงานที่เรียกว่า OTcl Linkage ดังรูปที่ 1

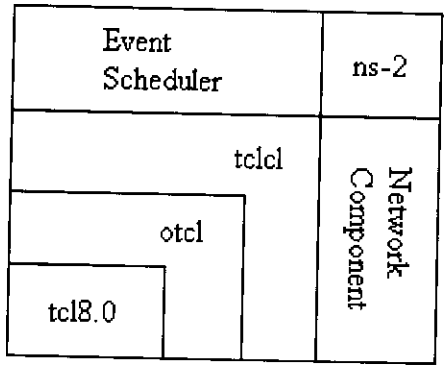
โดยผู้ที่จะทำการพัฒนารูปแบบการจำลองในรูปแบบใหม่ จะต้องทำการพัฒนาในส่วนนี้



รูปที่ 1 ความสัมพันธ์ระหว่างภาษา C++ กับภาษา OTcl

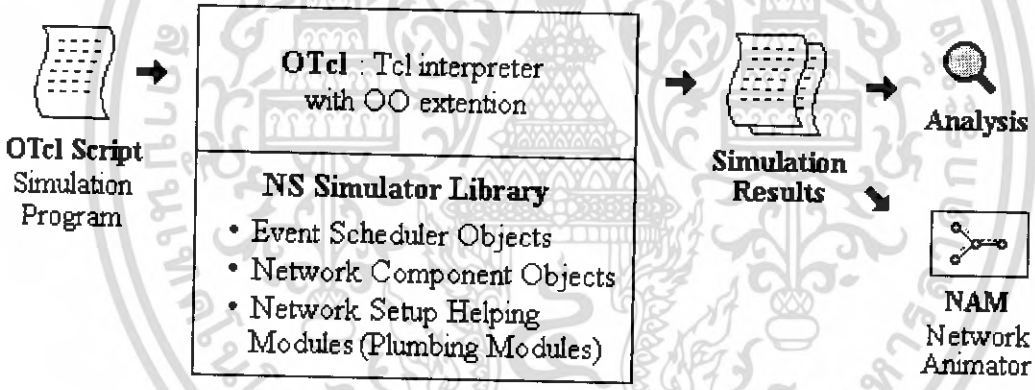
สำหรับการใช้โปรแกรม NS เพื่อจำลองระบบเครือข่ายนั้นจะใช้ภาษา Tcl ในการเรียกฟังก์ชันต่างๆ ที่ได้มีผู้พัฒนามาแล้ว คือ เรียกใช้องค์ประกอบในส่วนต่างๆ นำมาต่อกันเพื่อให้เกิดเป็นระบบเครือข่ายที่ต้องการ เรียกรูปแบบการทำงานนี้ว่า Event Scheduler

สำหรับโครงสร้างภายในตัวโปรแกรม NS จะเป็นดังรูปที่ 2



รูปที่ 2 โครงสร้างภายในของโปรแกรม NS

ในการใช้งานโปรแกรม NS เพื่อวิเคราะห์การทำงานนั้นจะมีลักษณะการทำงานดังรูปที่ 3 โดยเมื่อทำการเขียนคำสั่งกำหนดรูปแบบเครือข่ายเรียบร้อยแล้ว จึงทำการเรียกใช้โปรแกรม NS ให้ทำงาน จากนั้นเมื่อได้ผลการทำงานแล้วจึงนำมาวิเคราะห์ โดยอาจจะใช้โปรแกรมจำลองเส้นทางการเดินทางของข้อมูล ที่ชื่อว่า NAM เพื่อใช้ช่วยในการวิเคราะห์การทำงาน



รูปที่ 3 ขั้นตอนการทำงานในการวิเคราะห์ระบบเครือข่าย

ในการใช้งานโปรแกรม NS ในการจำลองการทำงานต่างๆ นั้น จะใช้คำสั่ง Tcl ในการกำหนดค่าต่างๆ โดยจะเรียกใช้รูปแบบการทำงานที่กำหนดไว้แล้ว ซึ่งในรูปแบบต่างๆ เหล่านี้จะถูกเรียกใช้จากคลาส Simulator ซึ่งเป็นคลาสหลักของโปรแกรม NS

1.1 รูปแบบคำสั่งที่ใช้งานซึ่งเป็นส่วนหลักของโปรแกรม

-รูปแบบคำสั่งที่ใช้ในโปรแกรม NS จะมีรูปแบบทั่วไปคือ

```
ns <tcl-script>
```

-สร้างตัวงาน

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

-ลำดับเหตุการณ์

$ns at <time> <event>

โดยที่ <time> เป็นเวลาที่กำหนดในลำดับเหตุการณ์

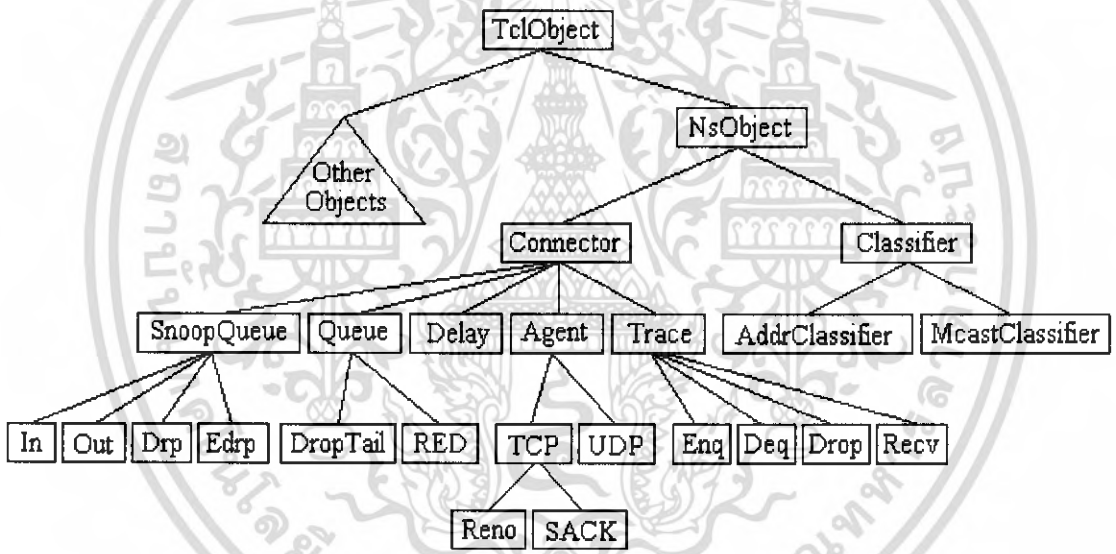
<event> เป็นเหตุการณ์ที่จะกระทำ

-เริ่มต้นตัวงาน

$ns run

1.2 องค์ประกอบของเครือข่ายในโปรแกรม NS

ในเครือข่ายหนึ่งๆ จะประกอบไปด้วยส่วนต่างๆ มากมายที่เชื่อมต่อกันในการทำงานร่วมกัน เช่น จุดเชื่อมต่อ (node) ของจุดหนึ่ง จะถูกเชื่อมต่อด้วยเส้นทางการเชื่อมต่อ (link) ไปยังจุดเชื่อมต่ออีกจุดหนึ่ง ซึ่งในแต่ละส่วนจะประกอบไปด้วยองค์ประกอบอีกมากมายที่ถูกกำหนดให้ทำงานเป็นส่วนๆ ดังนั้นในการพิจารณาองค์ประกอบของเครือข่ายในโปรแกรม NS จะแบ่งองค์ประกอบออกเป็นส่วนๆ ดังรูปที่ 4



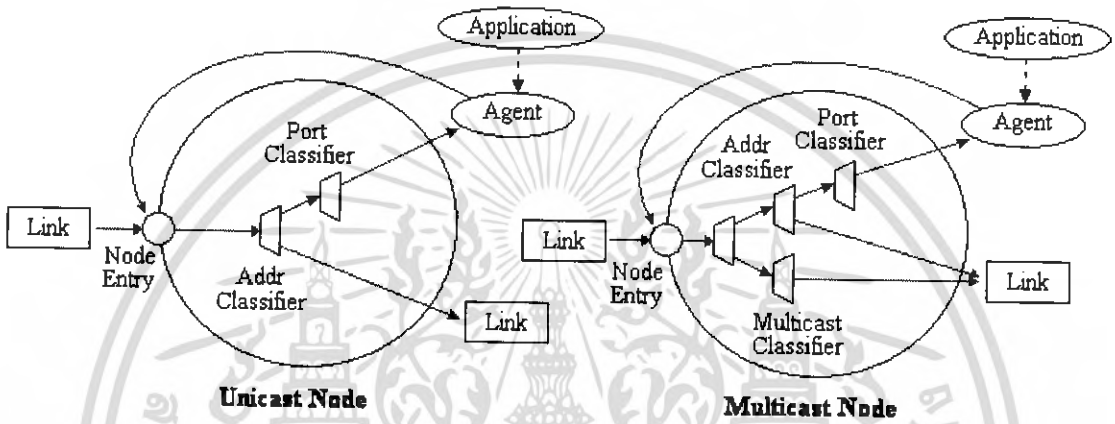
รูปที่ 4 ระดับชั้นขององค์ประกอบในโปรแกรม NS

จากรูปที่ 4 จะพิจารณาในส่วนที่เกี่ยวข้องโดยจะมี TelObject เป็นคลาสแม่ของคลาสทั้งหมด โดยมี NSObject นั้นเป็นคลาสลูก ซึ่ง NSObject จะเป็นคลาสแม่ขององค์ประกอบของเครือข่ายทั้งหมด ถ้าแบ่งองค์ประกอบต่างๆ นี้ ตามจำนวนของเส้นทางของผลลัพธ์ที่เป็นไปได้ จะสามารถแบ่งออกเป็น 2 ประเภท คือ แบบ Connector ซึ่งเป็นองค์ประกอบพื้นฐานต่างๆ ที่มีเส้นทางของผลลัพธ์ในทางเดียว เช่น Drop, Enq, Deq , Recv ที่ใช้ในการตามข้อมูล (trace) และแบบ Classifier ซึ่งจะเป็นองค์ประกอบที่ใช้ในการเลือกเส้นทาง (switching) เมื่อนำทั้งสองประเภทมาประกอบรวมกัน จะได้เป็นองค์ประกอบต่างๆ

1.2.1 จุดเชื่อมต่อ (Node)

จุดเชื่อมต่อจะประกอบไปด้วย จุดรับข้อมูล (Node Entry) และองค์ประกอบประเภท Classifier โดยโหนดสามารถแบ่งได้เป็น 2 ประเภท คือ ยูนิคาสต์โหนด และ มัลติคาสต์โหนด

ยูนิคาสต์โหนด ในส่วนของ Calssifier จะประกอบไปด้วย Addr Classifier ซึ่งใช้สำหรับการค้นหาเส้นทางในแบบยูนิคาสต์ (Unicast Routing) ซึ่งจะเชื่อมต่อกับโหนดอื่นๆ โดยเชื่อมต่อผ่านลิงค์ และ Port Classifier ซึ่งจะใช้เชื่อมต่อกับการทำงานในระดับสูง เช่น เชื่อมต่อกับโปรโตคอล TCP เป็นต้น ซึ่งจะเรียกว่า ตัวแทน (Agent) ดังรูปที่ 5



รูปที่ 5 โครงสร้างของ ยูนิแคส โหนด และ มัลติแคส โหนด

มัลติคาสต์โหนด ในส่วนของ Classifier จะแตกต่างกับแบบ ยูนิคาสต์โหนด ตรงที่จะประกอบไปด้วย Classifier แบบ Unicast Classifier และ Multicast Classifier ซึ่งทำให้สามารถทำงานได้ทั้งสองแบบ โดยจะมีตัวที่ทำหน้าที่แยกแพ็กเก็ตว่าเป็นแบบใด จะเรียกส่วนนี้ว่า สวิตช์ สำหรับ Multicast Classifier จะใช้สำหรับการค้นหาเส้นทางแบบมัลติคาสต์โดยจะมีตัวที่ทำหน้าที่กระจายข้อมูลว่าไปให้โหนดต่างๆ ที่เชื่อมต่อกันอยู่ จะเรียกส่วนนี้ว่า Replicators สำหรับโครงสร้างนั้นแสดงในรูปที่ 5

สำหรับการใช้งาน การค้นหาเส้นทางแบบมัลติคาสต์นั้น จะพิจารณาบิตสูงสุดของแอดเดรส ซึ่งจะเป็นตัวบอกว่าเป็นแบบยูนิคาสต์ หรือแบบมัลติแคส โดยถ้ามีค่าเป็น 0 จะเป็นยูนิคาสต์แอดเดรส ถ้าเป็น 1 จะเป็น มัลติคาสต์แอดเดรส

ในการกำหนดค่านั้น โปรแกรม NS จะกำหนดค่าเริ่มต้นเป็นแบบยูนิคาสต์โหนด ดังนั้นถ้าจะใช้แบบ มัลติคาสต์โหนด จะต้องกำหนดค่าเพิ่มเติม เพื่อบอกให้โปรแกรมรู้ก่อน โดยสรุปรูปแบบคำสั่งในการกำหนดรูปแบบ ดังนี้

-สร้างจุดเชื่อมต่อ (Node)

```
$ns node
```

-การเปลี่ยนแปลงค่าพารามิเตอร์ของจุดเชื่อมต่อ

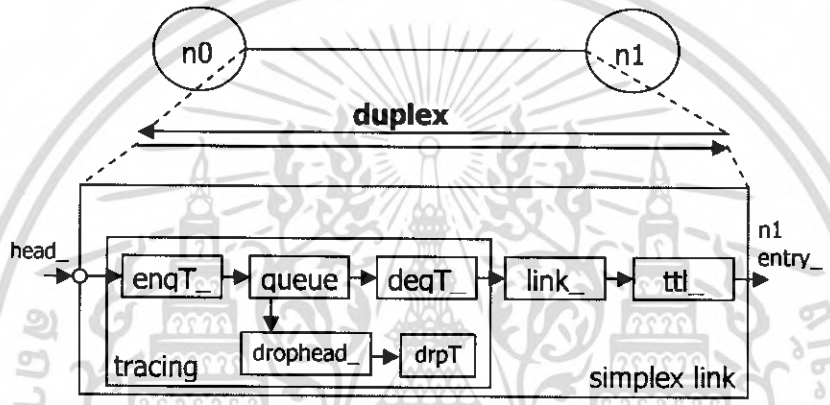
```
$ns node-config - <config-parameter> <optional>
```

1.2.2 เส้นทางการเชื่อมต่อ (Link)

เส้นทางการเชื่อมต่อ หรือ ลิงค์ เป็นส่วนประกอบสำคัญส่วนหนึ่งในเครือข่าย เป็นส่วนที่เชื่อมต่อจุดต่างๆ ในเครือข่ายเข้าด้วยกัน เช่น เป็นส่วนเชื่อมต่อระหว่างโหนด เพื่อทำหน้าที่ส่งผ่านแพ็กเก็ตข้อมูลระหว่างกัน

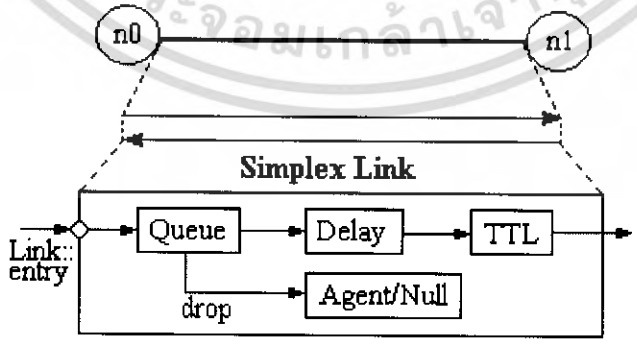
สำหรับในโปรแกรม NS นั้นยังได้สนับสนุนในหลายรูปแบบ เช่น แบบ Multi-access LAN รวมไปถึงการใช้งานไวร์เลส หรือบรอดคาสต์โดยจะพิจารณาในรูปแบบ จุดต่อจุด (point-to-point) ซึ่งถือได้ว่าเป็นลักษณะพื้นฐานของรูปแบบต่างๆ

ในลักษณะการเชื่อมต่อโดยทั่วไปนั้น จะเป็นในลักษณะการเชื่อมต่อแบบ 2 ทิศทาง (Duplex link) ดังแสดงในรูปที่ 6



รูปที่ 6 ลักษณะการเชื่อมต่อแบบ 2 ทิศทาง (Duplex link)

เมื่อพิจารณาแล้วจะสามารถแบ่งได้เป็นการเชื่อมต่อในแบบทิศทางเดียว (Simplex link) ใน 2 ทิศทาง เมื่อพิจารณาในโครงสร้างแล้ว จะมีลักษณะเหมือนกัน ดังนั้นแล้วจะพิจารณาในรูปแบบทิศทางเดียว ดังรูปที่ 7



รูปที่ 7 ลักษณะการเชื่อมต่อแบบทิศทางเดียว (Simplex link)

จากรูปที่ 7 จะเห็นว่าภายในจะประกอบไปด้วยองค์ประกอบ 5 ส่วนคือ

1. Link entry เป็นจุดเข้าของลิงค์ ซึ่งเป็นส่วนที่อยู่ด้านหน้าสุดของลิงค์

2. คิว เป็นส่วนหลักของลิงค์เป็นส่วนที่ทำหน้าที่จัดแพ็กเก็ตข้อมูล โดยทั่วไปจะมีเพียง 1 ส่วนเท่านั้นใน 1 ลิงค์

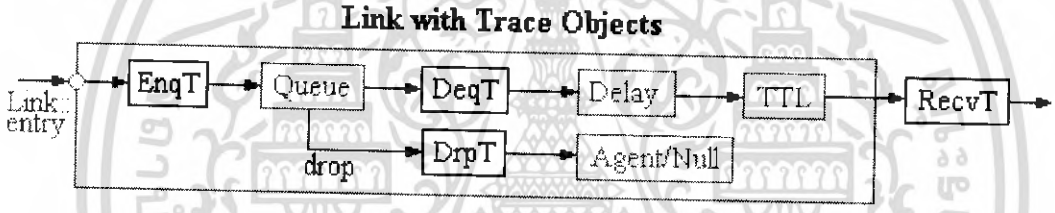
3. ดีเลย์ หรืออาจจะใช้คำว่า ลิงค์ ดีเลย์ ในส่วนนี้จะมีลักษณะของการหน่วงเวลาในการส่งผ่าน

4. TTL (Time to Live) เป็นค่าพารามิเตอร์ของเวลาในการส่งผ่านแพ็กเก็ตซึ่งจะมีการเปลี่ยนแปลงเมื่อแพ็กเก็ตส่งผ่านส่วนต่างๆ ไป

5. Agent/Null หรือบางทีจะใช้คำว่า drophead เป็นส่วนที่ใช้เก็บแพ็กเก็ตที่ไม่ได้ใช้งาน

1.2.3 การติดตามการเดินทางของแพ็กเก็ต (Tracing)

ในการทดสอบประสิทธิภาพโดยทั่วไปจะมีการติดตามการไหลของแพ็กเก็ตในส่วนต่างๆ ว่ามีลักษณะอย่างไร เมื่อมีการใช้งานก็ทำให้ต้องเพิ่มองค์ประกอบเข้าไป แทรกในจุดต่างๆ ในองค์ประกอบนั้นๆ ในส่วนนี้ก็ทำให้โครงสร้างของลิงค์ มีลักษณะดังรูปที่ 8



รูปที่ 8 องค์ประกอบของการตามข้อมูลในเส้นทางการเชื่อมต่อ

ส่วนที่เพิ่มเข้ามาจะประกอบไปด้วย

1. EnqT เป็นส่วนที่ทำหน้าที่ เก็บผลของแพ็กเก็ต เมื่อก่อนเข้าคิว

2. DeqT เป็นส่วนที่ทำหน้าที่เก็บผลของแพ็กเก็ต เมื่อออกจากคิว

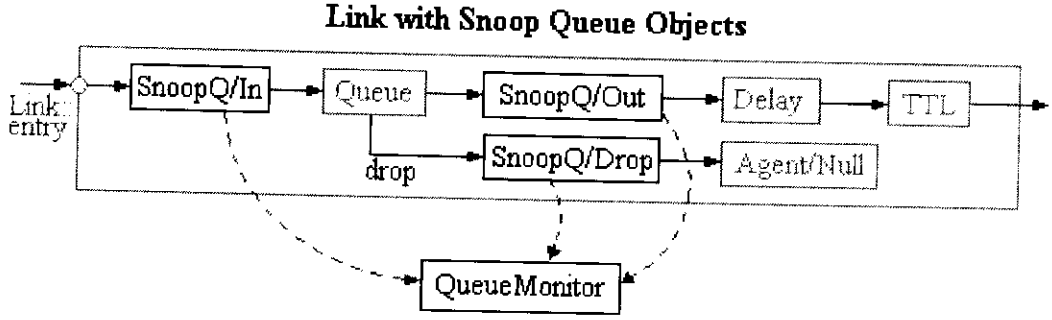
3. DrpT เป็นส่วนที่ทำหน้าที่เก็บผลของแพ็กเก็ต เมื่อแพ็กเก็ตถูกปล่อยมากจากคิว

4. RecvT เป็นส่วนที่ทำหน้าที่เก็บผลของแพ็กเก็ต เมื่อแพ็กเก็ตจะถูกส่งไปโหนดต่อไป

สำหรับรายละเอียดเกี่ยวกับการติดตามการไหลของแพ็กเก็ต (Tracing) นี้จะอธิบายเพิ่มเติมในส่วนต่อไป

1.2.4 การตรวจวัดการทำงานของการจัดคิว (Queue Monitor)

ในบางกรณี จะต้องการข้อมูลของการทำงานของระบบการจัดคิว (Queue) เพื่อนำไปวิเคราะห์ ไม่ว่าจะเป็นการจัดคิวในรูปแบบใด จะมีรูปแบบของการทำงานในรูปแบบเดียวกัน ดังรูปที่ 9



รูปที่ 9 องค์ประกอบของการตรวจวัดการทำงานของการจัดคิว

โดยจะทำการเพิ่มตัวตรวจจับการทำงานที่เรียกว่า SnoopQ ไปคั่นระหว่างจุดต่างๆ จะประกอบไปด้วย SnoopQ/In, SnoopQ/Out และ SnoopQ/Drop ส่วนต่างๆ เหล่านี้ จะประกอบเป็นระบบการตรวจวัดการทำงานของการจัดคิว

สำหรับรูปแบบคำสั่งที่ใช้งาน มีดังนี้

- การสร้างการเชื่อมต่อ แบบทิศทางเดียว (Simplex link)

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

- การกำหนดค่าการเชื่อมต่อแบบทิศทางเดียว (สำหรับโปรแกรม NAM)

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

- การสร้างการเชื่อมต่อ แบบสองทิศทาง (Duplex link)

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

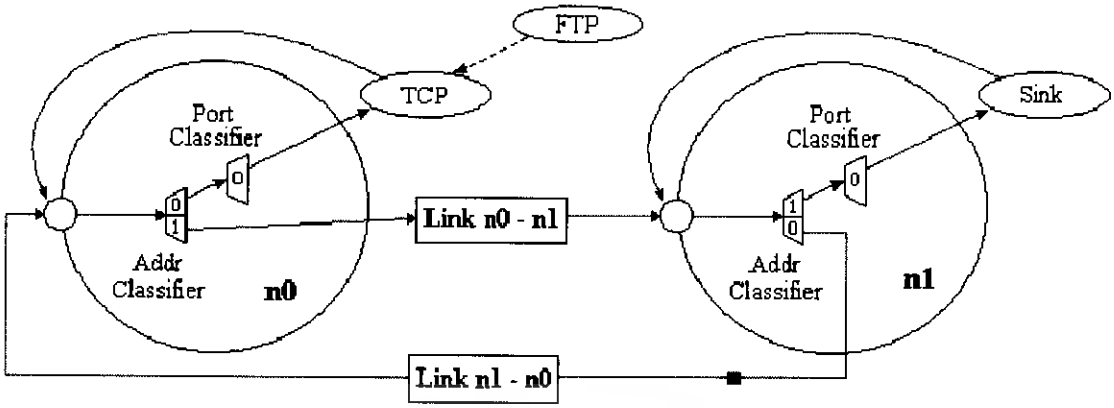
- การกำหนดค่าการเชื่อมต่อแบบสองทิศทาง (สำหรับโปรแกรม NAM)

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

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

1.2.5 ตัวแทนรูปแบบการทำงานในระดับสูง (Agent)

ตัวแทนรูปแบบการทำงานในระดับสูง หรือที่เรียกว่า Agent คือรูปแบบต่างๆ ของการทำงานในระดับที่สูงขึ้น ที่ทำหน้าที่ต่อจากการทำงานในระดับล่างๆ จะเชื่อมต่อกัน (node) โดยจะแบ่งการทำงานออกเป็นส่วนๆ มีลักษณะเป็นลำดับชั้นไปเรื่อยๆ จนถึงรูปแบบการทำงานในระดับชั้นโปรแกรมประยุกต์ เช่น TCP Agent, UDP Agent เป็นต้น เมื่อนำไปใช้งานในระบบแล้ว จะมีลักษณะดังรูปที่ 10



รูปที่ 10 ตัวแทนรูปแบบการทำงานในระดับสูง (Agent)

สำหรับโปรแกรม NS จะสนับสนุน Agent ในหลายรูปแบบที่มีการใช้งาน ดังตัวอย่างนี้ซึ่งจะเป็นเพียงส่วนหนึ่งที่โปรแกรม NS สนับสนุน

TCP	เป็นโปรโตคอล TCP แบบ Tahoe ทางด้านส่ง
TCP/Reno	เป็นโปรโตคอล TCP แบบ Reno ทางด้านส่ง
TCP/NewReno	เป็นโปรโตคอล TCP แบบ Reno ที่มีการพัฒนาทางด้านส่ง
TCP/Sack1	เป็นโปรโตคอล TCP แบบ SACK ทางด้านส่ง
TCP/Fack	เป็นโปรโตคอล TCP แบบ forward SACK ทางด้านส่ง
TCP/FullTcp	เป็นโปรโตคอล TCP แบบ 2 ทิศทางทางด้านส่ง
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	เป็นส่วนที่เก็บแพ็กเก็ตที่ถูกจำกัดทิ้ง

ในรูปแบบต่างๆ นี้ เป็นรูปแบบการทำงานที่เรียกว่าเป็นคลาสลูกของ Agent Class ซึ่งจะมีรูปแบบที่เป็นพื้นฐานเหมือนๆ กัน ดังนั้นในการกำหนดการทำงานต่างๆ เหล่านี้จะมีลักษณะเหมือนกันในส่วนนี้จะสรุปคำสั่งพื้นฐานที่ใช้กำหนดรูปแบบ ซึ่งจะรูปแบบดังนี้

- สร้างรูปแบบการทำงานของ Agent

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

โดยที่ <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>]

โดยที่ <pktclass> คือ เป็นการกำหนดคลาสให้การเชื่อมต่อ

- การกำหนดในรายละเอียดของการทำงานของ Agent

$<agent> set <argument>

1.2.6 ส่วนประยุกต์ใช้งาน (Applications)

ส่วนประยุกต์ใช้งานหรือ แอพพลิเคชั่น เป็นส่วนที่เชื่อมต่อระหว่างการทำงานในระดับชั้นขนส่งข้อมูล ที่เราเรียกว่า Agent กับการใช้งานในระดับโปรแกรมประยุกต์ ซึ่งเป็นรูปแบบระดับการทำงานในระดับสูง ในโปรแกรม NS นั้นจะแบ่งส่วนประยุกต์ใช้งานนี้ออกเป็น 2 ประเภท คือ ตัวกำเนิดทราฟฟิก (Traffic generators) และการจำลองโปรแกรมประยุกต์ (Simulated Applications)

สำหรับการเชื่อมต่อระหว่างระดับชั้นขนส่งข้อมูลกับระดับชั้นโปรแกรมประยุกต์นั้นจะใช้รูปแบบการรองรับการทำงานที่เรียกว่า Application Programming Interface (API) หรือซ็อคเก็ต (Sockets)

1.2.6.1 ตัวกำเนิดทราฟฟิก (Traffic generators)

ในโปรแกรม NS จะสนับสนุนอยู่ 4 ประเภทคือ

1. EXPOO Traffic

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

2. POO Traffic

เป็นตัวกำเนิดทราฟฟิกที่มีรูปแบบการทำงานแบบ Exponential On/Off เช่นกัน แต่ค่าช่วงเวลาทำการ และช่วงเวลาหยุดนี้ จะใช้รูปแบบของ Pareto

3. CBR Traffic

เป็นตัวกำเนิดทราฟฟิกที่มีค่าอัตราการส่งแพ็กเก็ตคงที่ ขนาดแพ็กเก็ตคงที่ โดยสามารถกำหนดให้มีการรบกวนเกิดขึ้นในระหว่างการส่งได้

4. TrafficTrace

เป็นตัวกำเนิดทราฟฟิก ที่มีรูปแบบการทำงานตามค่าในไฟล์ตามเส้นทางที่แพ็กเก็ตผ่าน

สำหรับรูปแบบคำสั่งที่ใช้กำหนดการทำงาน จะมีรูปแบบดังนี้

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

$<Traffic_name> attach-agent $<Agent_name>

หรือ จะใช้คำสั่ง

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

1.2.6.2 การจำลองโปรแกรมประยุกต์ (Simulated Application)

ในโปรแกรม NS จะสนับสนุนอยู่ 2 ประเภท คือ

1. FTP

เป็นรูปแบบการทำงานที่ใช้สำหรับการถ่ายโอนไฟล์ข้อมูล (file transfer)

2. Telnet

เป็นรูปแบบการทำงานที่ใช้สำหรับการติดต่อระหว่างเครื่องลูกข่ายกับเครื่องแม่ข่าย โดยจะมีรูปแบบการทำงานอยู่ 2 รูปแบบ คือ ถ้าค่า interval_ ไม่เป็น 0 นั้นจำนวนแพ็กเก็ตที่ส่งจะมีรูปแบบ exponential แต่ถ้าค่า interval_ มีค่าไม่เป็น 0 นั้นจำนวนแพ็กเก็ตที่ส่งจะมีรูปแบบของ TCP

รูปแบบคำสั่งที่ใช้กำหนดการทำงาน จะมีรูปแบบดังนี้

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

$<Application_name> attach-agent $<Agent_name>

หรือจะใช้คำสั่ง

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

1.3 กระบวนการหาเส้นทาง (Routing)

กระบวนการหาเส้นทางเป็นกระบวนการที่กำหนดรูปแบบลำดับการเดินทางของแพ็กเก็ตข้อมูลที่มีการสื่อสารกันในเครือข่าย สามารถแบ่งออกเป็น 2 ประเภท คือ

1.3.1 กระบวนการหาเส้นทางแบบยูนิคาสต์ (Unicast Routing)

รูปแบบกระบวนการหาเส้นทางแบบยูนิคาสต์มี 4 แบบคือ

1. Static Routing

เป็นรูปแบบที่ใช้เป็นค่าเริ่มต้นในการกำหนดกระบวนการหาเส้นทางในโปรแกรม NS ซึ่งเป็นกระบวนการที่ใช้แนวคิดแบบ Dijkstra's all-pairs SPF จะมีการเส้นทางเพียงครั้งเดียวเมื่อเริ่มต้นในการจำลองการทำงาน

2. Session Routing

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

3. DV Routing

เป็นกระบวนการที่ใช้รูปแบบ Distributed Bellman-Ford (หรือ Distance Vector) โดยวิธีนี้จะมีการปรับปรุงค่าเป็นระยะๆ

4. Manual Routing

เป็นกระบวนการที่ไม่ได้ใช้การคำนวณในแบบใดๆ แต่จะใช้การควบคุมจากผู้ใช้งานเอง

สำหรับคำสั่งที่ใช้กำหนดรูปแบบของการหาเส้นทาง จะมีรูปแบบดังนี้

```
$ns rtproto <routing-protocol> <args>
```

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

1.3.2 กระบวนการหาเส้นทางแบบมัลติคาสต์ (Multicast Routing)

รูปแบบกระบวนการหาเส้นทางแบบมัลติคาสต์ที่โปรแกรม NS สนับสนุนมี 4 แบบ คือ

1. Centralized Multicast หรือ CtrMcast

   เป็นกระบวนการที่มีลักษณะคล้ายแบบ PIM-SM

2. Dense Mode หรือ DM

   เป็นกระบวนการที่ใช้รูปแบบ dense-made-like

3. Shared Tree Mode หรือ ST

   เป็นรูปแบบของ sparse mode ซึ่งจะใช้โปรโตคอล shared-tree-multicast protocol

4. Bi-irectional Shared Tree Mode หรือ BST

   เป็นรูปแบบที่อยู่ในระหว่างการพัฒนา

สำหรับคำสั่งที่ใช้กำหนดรูปแบบกระบวนการหาเส้นทาง จะมีรูปแบบดังนี้

- กำหนดการใช้งานในรูปแบบมัลติคาสต์

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

หรือใช้คำสั่ง

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

- การกำหนดรูปแบบของมัลติคาสต์

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

โดยที่ <type> คือ CtrMcast, DM, ST, BST

1.4 การจัดการบริการแถวรอคอย (Queue Management)

การจัดการบริการแถวรอคอย หรือการจัดคิวให้แพ็กเก็ตที่เข้ามาในส่วนต่างๆ หรือ การจัดลำดับความสำคัญของแพ็กเก็ตต่างๆ ที่เข้ามารับบริการนั้นว่าแพ็กเก็ตตัวใดจะได้รับการบริการก่อนหรือหลัง ถือได้ว่าการจัดคิวเป็นส่วนสำคัญส่วนหนึ่งของระบบเครือข่าย ที่จะเป็นตัวกำหนดคุณภาพของการให้บริการ (Quality of Service : QoS) ในระบบเครือข่าย

1.4.1 Drop-Tail Queuing (FIFO)

เป็นกลไกการทำงานพื้นฐานที่สุดในการจัดคิว การทำงานจะจัดการกับแพ็กเก็ตที่เข้ามาอย่างต่อเนื่อง โดยลักษณะที่ตัวใดมาก่อนก็ได้รับการจัดการก่อน เป็นวิธีการดั้งเดิมที่ใช้กันอย่างแพร่หลายในอุปกรณ์เครือข่ายอย่างง่าย ซึ่งลักษณะการจัดการนั้นถือเป็นการทำงานที่ไม่ดี ในการที่จะให้บริการระบบที่มีความหนาแน่นของทราฟฟิกสูงๆ

1.4.2 Fair Queuing (FQ)

เป็นกลไกที่มีรูปแบบการทำงานแบบ Round Robin โดยจะทำการจัดเก็บแพ็กเก็ตที่เข้ามาลงไว้ในคิวที่ต่างกัน แล้วจึงให้บริการส่งทีละแพ็กเก็ตต่อคิว แล้วจึงไปให้บริการในคิวต่อไป

1.4.3 Stochastic Fairness Queuing (SFQ)

เป็นกลไกที่มีรูปแบบการทำงานแบบ Round Robin เช่นกัน แต่ทำการเปลี่ยนแปลงการจัดเก็บแพ็กเก็ตที่เข้ามาเอาไว้รวมกันในคิวเดียว โดยใช้ Hashing function ในการจัดการ

1.4.4 Deficit Round Robin Queuing (DRR)

เป็นกลไกที่ใช้การกระจายแพ็กเก็ตลงในคิวแบบ SFQ แต่กลไกการทำงานจะมีรูปแบบที่แตกต่างกับแบบ Round Robin คือ จะมีการกำหนดขนาดของแต่ละคิวไว้ (Quantrum size) คือถ้าคิวนี้มีแพ็กเก็ตที่ใหญ่เกินไปในการส่งขนาดแพ็กเก็ตที่เหลือที่ยังไม่ได้ส่งจะถูกนำไปรวมในครั้งต่อไป

1.4.5 Random Early Detection Queuing (RED)

เป็นกลไกที่มีการจัดคิวต่างจากลักษณะ FIFO โดยวิธีนี้จะพิจารณาความหนาแน่นของเครือข่ายเบื้องต้น เพื่อใช้การพิจารณาการควบคุมปริมาณข้อมูลที่จะเข้ามาในระบบ จะใช้การเลือกสุ่มการเชื่อมต่อหรือการไหลของแพ็กเก็ตที่ถูกทำเครื่องหมายไว้ ซึ่งจะเกี่ยวข้องกับค่าความยาวเฉลี่ยของคิว (Average queue size) ถ้าค่าความยาวเฉลี่ยของคิวนี้มีค่าสูงขึ้นเกินกว่าระดับจุดจำกัดที่กำหนดไว้แล้ว จะทำการหยุดแพ็กเก็ตบางส่วนไว้ เพื่อให้ความหนาแน่นในเครือข่ายลดลง

1.4.6 Class-Based Queuing (CBQ)

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

สำหรับรูปแบบคำสั่งที่ใช้ควบคุมการทำงานโดยทั่วไป การกำหนดขนาดของบัฟเฟอร์ที่ใช้ในการจัดคิว จะมีรูปแบบคือ

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

1.5 การติดตามเส้นทางการเดินทางแพ็กเก็ต (Trace and Monitoring)

วิธีการที่ใช้ในการติดตามเส้นทางการเดินทางของแพ็กเก็ตนั้นสามารถทำได้ในหลายวิธี ซึ่งโดยทั่วๆ ไปแล้วจะมีการเก็บบันทึกรายละเอียดการติดตามเส้นทางการเดินทางของแพ็กเก็ต ในขณะที่ทำการจำลองรูปแบบ หรือทำการบันทึกลงไปในไฟล์ เมื่อผ่านกระบวนการไปแล้ว ในการติดตามเส้นทางการเดินทางของแพ็กเก็ตนี้จะแบ่งออกเป็น 2 รูปแบบ คือ

1.5.1 การติดตามเส้นทางการเดินทางแพ็กเก็ต หรือ Trace

จะเป็นการบันทึกข้อมูลของแพ็กเก็ตแยกเป็นแต่ละแพ็กเก็ต โดยจะทำการบันทึกในแต่ละลำดับขั้นตอนในการทำงานซึ่งผ่านในส่วนต่างๆ เช่น แพ็กเก็ตที่มาถึงยังโหนด แพ็กเก็ตที่พร้อมที่จะออกจากโหนด เป็นต้น

ในการทำการติดตามเส้นทางการเดินทางของแพ็กเก็ตนั้นจะมีการเพิ่ม EnqT, DeqT, DrpT, RecvT เข้าไปในระหว่างเส้นทางการเชื่อมต่อ ดังที่กล่าวไว้แล้วในหัวข้อการเชื่อมต่อ ซึ่งในแต่ละส่วนจะทำหน้าที่เก็บข้อมูลการทำงานเอาไว้ โดยสามารถกำหนดส่วนที่เพิ่มไปได้ โดยใช้คำสั่ง

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

สำหรับรูปแบบคำสั่งที่ใช้ในการกำหนดการทำงาน จะมีรูปแบบดังนี้

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต

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

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต (สำหรับโปรแกรม NAM)

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

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต โดยระบุเฉพาะ

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

- การทำการติดตามเส้นทางการเดินทางของแพ็กเก็ต โดยระบุเฉพาะ (สำหรับโปรแกรม NAM)

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

- คำสั่งที่ให้บันทึกข้อมูลการทำงาน (เป็นคำสั่งที่ใช้ก่อนที่จะจบการทำงาน)

```
$ns flush-trace
```

สำหรับรูปแบบของไฟล์การติดตามเส้นทางการเดินทางของแพ็กเก็ต (Trace file) จะมีลักษณะชุดลำดับของค่าต่างๆ ดังรูปที่ 11

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)                    src_addr : node.port (3.0)
- : dequeue (at queue)                    dst_addr : node.port (0.0)
d : drop    (at queue)
```

```
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 11 รูปแบบของไฟล์ติดตามเส้นทางการเดินทางของแพ็กเก็ต

จากรูปที่ 11 ในแต่ละบรรทัด จะเป็นการทำงานในหนึ่งหน้าที่ โดยแต่ละบรรทัดก็จะแบ่งออกเป็นส่วนๆ ทั้ง 12 ส่วน ดังนี้

1. event เป็นรูปแบบการทำงาน จะมี 4 ประเภท คือ
   - r : receive คือ แพ็กเก็ตมาถึงโหนดแล้ว
   - + : enqueue คือ แพ็กเก็ตอยู่ในระหว่างการรอคิว
   - - : dequeue คือ แพ็กเก็ตที่ได้รับการจัดคิว
   - d : drop คือ แพ็กเก็ตไม่ได้รับการจัดคิว
2. time เป็นเวลาที่ทำงาน
3. from node เป็นส่วนที่ใช้บอกว่าแพ็กเก็ตถูกส่งมาจากที่ใด
4. to node เป็นส่วนที่ใช้บอกว่าแพ็กเก็ตจะถูกส่งไปที่ใด
5. packet type เป็นชนิดของแพ็กเก็ต
6. packet size เป็นขนาดของแพ็กเก็ต
7. flags ไม่มีการใช้งาน (จะมีลักษณะเป็น - - - - - - -)
8. fid เป็นส่วนที่ใช้กำหนดลำดับ IP 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

1.5.2 การเฝ้าดูการเดินทางของแพ็กเก็ต (Monitor)

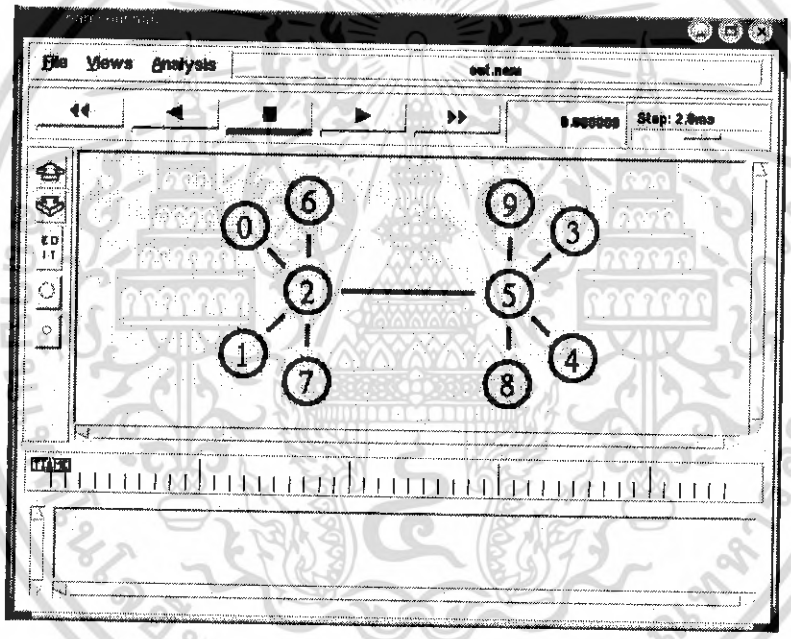
เป็นการบันทึกข้อมูลที่เจาะจงลงไปในส่วนที่สนใจ เช่น ในลักษณะของการสนใจปริมาณแพ็กเก็ตที่ได้รับ หรือจำนวนไบต์ที่ได้รับ เป็นต้น ซึ่งในแบบนี้ก็สามารถทำการเฝ้าดูโดยรวมทั้งหมดได้ หรือในลักษณะที่เรียกว่า per-flow ซึ่งเป็นรูปแบบของ flow monitor

สำหรับรูปแบบคำสั่งที่ใช้ในการกำหนดการทำงาน จะมีรูปแบบดังนี้

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

2. โปรแกรม NAM

โปรแกรม NAM ย่อมาจาก Network Animator เป็นโปรแกรมที่ใช้แสดงผลของโปรแกรม NS โดยเฉพาะ ดังรูปที่ 12



รูปที่ 12 โปรแกรม NAM

คำสั่งที่เรียกใช้งานโปรแกรม จะใช้รูปแบบภาษา Tcl/Tk โดยมีรูปแบบดังนี้

- เริ่มต้นการทำงานของโปรแกรม NAM

nam < option > < trace_file >

- กำหนดให้โปรแกรม NS บันทึกไฟล์

$ns namtrace - all < trace_file >

- กำหนดให้โปรแกรมเขียนลงไฟล์

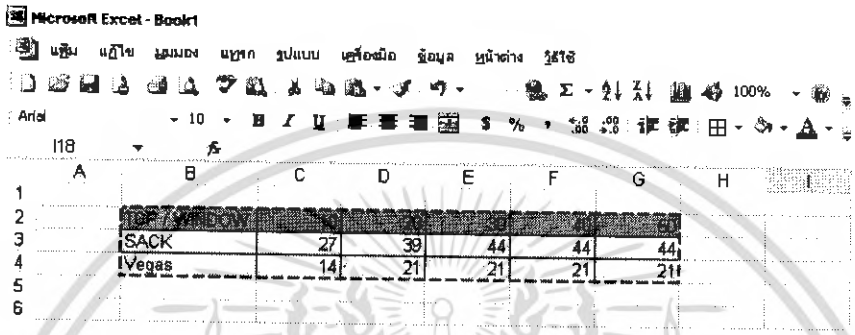
$ns flush – trace

3. โปรแกรม Excel

จะใช้โปรแกรม Excel มาทำการสร้างกราฟเส้น เพื่อวิเคราะห์และเปรียบเทียบประสิทธิภาพการทำงานของโปรโตคอล TCP ทั้ง 2 แบบ

วิธีการสร้างกราฟเส้นของโปรแกรม Excel

1. เลือกข้อมูลที่จะนำมาสร้างกราฟเส้นดังรูปที่ 13
2. เลือกไอคอนที่เป็นรูปกราฟบนแถบเครื่องมือ

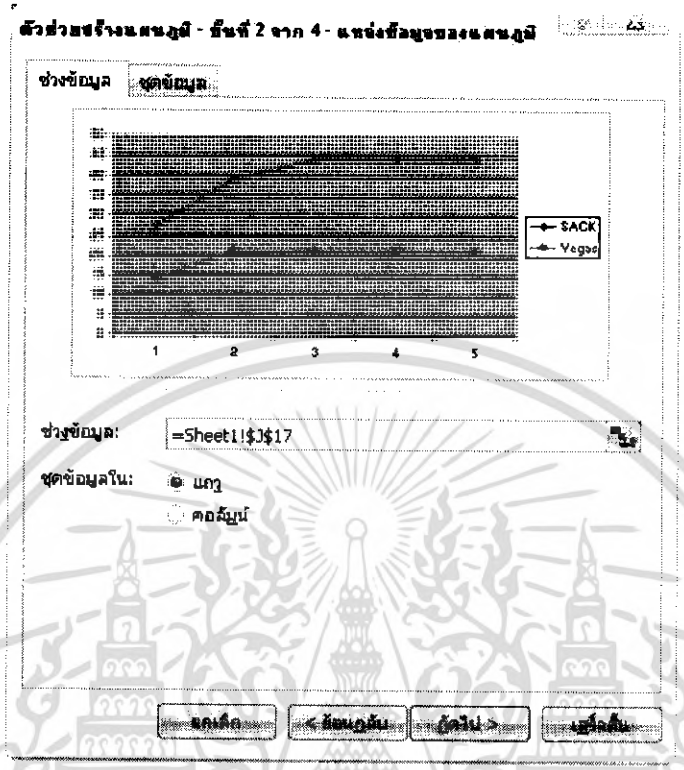


รูปที่ 13 โปรแกรม Excel

3. เลือกรูปแบบของกราฟ ในที่นี้จะเลือกเป็นกราฟเส้น ดังรูที่ 14

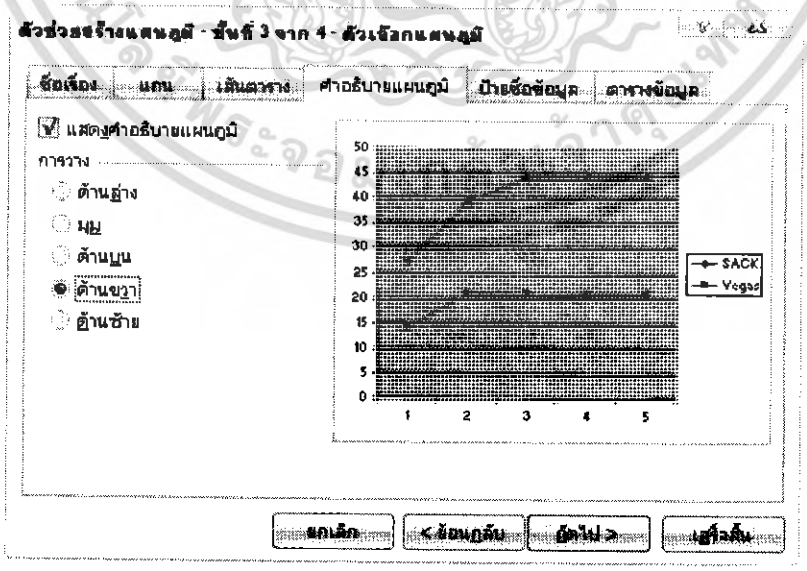
รูปที่ 14 การเลือกรูปแบบของกราฟ

5. คลิกตรงคำว่า ถัดไป เพื่อเลือกการแสดงกราฟให้เป็นแบบ แถว หรือ คอลัมน์ ในที่นี้เลือกให้เป็น แถว ดังรูที่ 15 เมื่อเลือกเสร็จแล้ว ก็คลิกคำว่า ถัดไป



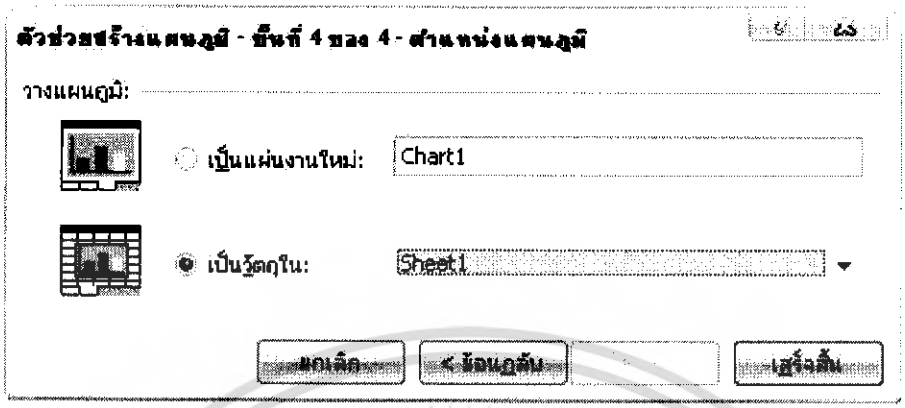
รูปที่ 15 การเลือกการแสดงของกราฟ

6. ใส่ชื่อ แล้วเลือกคำอธิบายแผนภูมิ และกำหนดค่าต่างๆ ของกราฟ ดังรูปที่ 16 แล้วคลิก ถัดไป

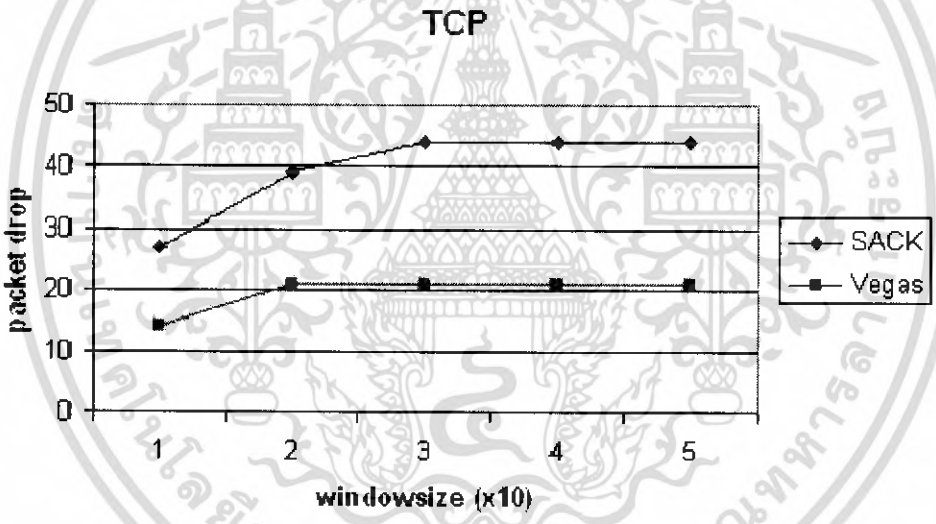


รูปที่ 16 กำหนดค่าต่างๆ ของกราฟ

7. เป็นขั้นตอนสุดท้ายในการสร้างกราฟ คลิกคำว่า เสร็จสิ้น ดังรูปที่ 17 และจะได้กราฟที่สมบูรณ์ดังรูปที่ 18



รูปที่ 17 ขั้นตอนสุดท้ายในการสร้างกราฟ



รูปที่ 18 ตัวอย่างกราฟเส้น

หนังสือและเอกสารอ้างอิง

[1] จตุชัย แพรจันทร์, อนุโชต วุฒิพรพงษ์, " เจาะระบบ Network ฉบับสมบูรณ์ ", ไอดีซี อินโฟดิสทริบิวเตอร์, 2546

[2] สุวัฒน์ ปุณณชัยยะ, ต้น ตันสุทธิวงษ์, สุพจน์ ปุณณชัย, " เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต ", โปรวิชั่น, 2545

[3] GIS – group, " เครือข่ายคอมพิวเตอร์ ", ฟิสิกส์ เซ็นเตอร์, 2536

[4] Andrew S. Tanenbaum, " Computer Network ", Pearson Education Indochina, 2004

[5] B. Braden et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998

[6] Behrouz A. Forouzan, Sophia, Chung, Fegan, " TCP/IP Protocol Suite ", Mc Graw Hill, 2006

[7] Douglas E Comer, " Internetworking With TCP/IP ", Prentice Hall, 2000

[8] H. Balakrishnan, "An Implementation of TCP Selective Acknowledgements", 1996

[9] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", in Computer Communication Review, July 1996, V. 26 N. 3, pp. 5-21.

[10] Larry L. Peterson & Bruce S. Davie, "Computer networks : a system approach 2nd ed.", Morgan Kaufmann Publisher, 2000

[11] Mahbub Hassan, Raj Jain, " High Performance TCP/IP Networking ", Prentice Hall, 2004

[12] Michael Santifaller, "TCP/IP and ONC/NFS Internetworking in a UNIX™ Environment 2 nd ed.", Addison-Wesley Publishing Company, 1994

[13] M. Mthis, J. Mahdasi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Option", RFC 2018, October 1996

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

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

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

[17] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, and A. Romanow, "An Extension to the Selective Acknowledgement (SACKs) option for TCP ", RFC 2883, July 2000

[18] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, vol.1 n.4, pp. 397-413, August 1993

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

[20] W. Stevens "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, 1997