การปรับปรุงการลดทอนข้อมูล TCP ด้วย ECN

IMPROVEMENT OF TCP PACKET DROPPING USING ECN

วรุต อนุชชาลาคม
WARUT ANUTCHALAKHOM

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2546

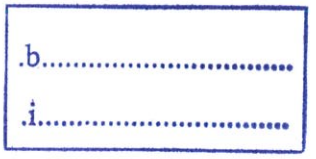
ISBN 974-324-541-3

การปรับปรุงการลดทอนข้อมูล TCP ด้วย ECN

IMPROVEMENT OF TCP PACKET DROPPING USING ECN

วรุต อนุชชาลาคม

WARUT ANUTCHALAKHOM



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2546

ISBN 974-324-541-3

IMPROVEMENT OF TCP PACKET DROPPING USING ECN

WARUT ANUTCHALAKHOM

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

ISBN 974-324-541-3

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การปรับปรุงการลดทอนข้อมูล TCP ด้วย ECN

IMPROVEMENT OF TCP PACKET DROPPING USING ECN

ชื่อนักศึกษา นายวรุต อนุชชาลาคม

รหัสประจำตัว 42061020

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมไฟฟ้า

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.บรรจง ปิยธำรง

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.วิศิษฏ์	หิรัญกิตติ	
ดร.วรวัฒน์	ลิ้มโภคา	
รศ.ประทีป	บัญญัตินพรัตน์	
รศ.บรรจง	ปิยธำรง	

วัน/เดือน/ปี ที่สอบ 25 เมษายน 2546 เวลา 14.00-16.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-404)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่.....29.....**เดือน**.....เมษายน.....**พ.ศ.**.....2546.....

หัวข้อวิทยานิพนธ์	การปรับปรุงการลดทอนข้อมูล TCP ด้วย ECN
นักศึกษา	นายวรุต อนุชชาลาคม
รหัสประจำตัว	42061020
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมไฟฟ้า
พ.ศ.	2546
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.บรรจง ปิยธำรง

บทคัดย่อ

Transmission control protocol (TCP) ถูกใช้งานอย่างแพร่หลายตามการเติบโตของ Internet โดยพื้นฐานของกลไก TCP ผู้ส่งข้อมูลจะทราบถึงความแออัด (congestion) ของช่องทางที่ใช้ในการส่งข้อมูลจากการที่ packet นั้นโดนคัดทิ้ง (drop) ซึ่งทำให้เกิดการสูญหายของ packet ในเครือข่าย (network) และ ทำให้ต้องส่ง packet เหล่านั้นซ้ำใหม่ (packet retransmit) จึงมีแนวความคิดที่จะทำการแจ้ง TCP sender ให้ทราบล่วงหน้าหลังจากมีการคาดการณ์ว่าจะเกิด congestion ก่อนที่จะเกิดการคัด packet ทิ้งในเครือข่าย Explicit Congestion Notification (ECN) จึงถูกนำเสนอขึ้นมาเพื่อสนับสนุนแนวทางนี้ โดยในขั้นแรกได้มีการเสนอให้ใช้กับ queue management แบบ Random Early Detection (RED) แต่ในวิทยานิพนธ์ฉบับนี้ได้นำ Explicit Congestion Notification มาใช้ร่วมกับ queue management แบบ Drop-Tail ซึ่งวิทยานิพนธ์ฉบับนี้เรียกว่า Drop-Tail with ECN (DT-ECN) และได้ทำการจำลองเหตุการณ์ (simulation) ด้วย Network Simulator 2 (NS-2) เพื่อวิเคราะห์และจากผลลัพธ์ที่ได้แสดงให้เห็นว่าจำนวน packet ที่ถูกคัดทิ้งมีปริมาณลดลงอย่างชัดเจนเมื่อเทียบกับ queue ที่ได้รับความนิยมแบบ Drop-Tail, RED และ RED ที่มี ECN

Thesis Title	Improvement of TCP Packet Dropping Using ECN
Student	Mr.Warut Anutchalakhom
Student ID.	42061020
Degree	Master of Engineering
Programme	Electrical Engineering
Year	2003
Thesis Advisor	Assoc.Prof.Bunjong Piyatamrong

ABSTRACT

For the Transmission Control Protocol (TCP), a TCP sender relies on packet dropping as the indication of congestion from the intermediate nodes to avoid packet loss and packet retransmission over the network. A prediction mechanism on the intermediate nodes that predicts the chance of congestion in the near future can send some information to a TCP sender to drop the sending packets in advance. The Explicit Congestion Notification (ECN) implemented on Random Early Detection (RED) gateways has been proposed to pass on the information. In this thesis, the combination between Explicit Congestion Notification mechanism and a normal Drop-Tail gateway is proposed. This mechanism, the Drop-Tail with ECN (DT-ECN), is simulated using the Network Simulator version 2 (NS-2) to evaluate and generate the results that show greater benefits of Drop-Tail with ECN than the others in terms of packet dropping.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำจาก รศ.บรรจง ปิยธำรง ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ข้าพเจ้ารู้สึกซาบซึ้งในความอนุเคราะห์อย่างดียิ่งจากท่านและขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ด้วย

ขอขอบพระคุณคณาจารย์ของสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ประสิทธิ์ประสาทความรู้ให้แก่ข้าพเจ้าได้นำมาคิดวิจัยวิทยานิพนธ์ฉบับนี้ได้สำเร็จ

ขอขอบคุณสมาชิกในครอบครัวและเพื่อนๆทุกคน ที่ให้การสนับสนุนและมอบกำลังใจแก่ข้าพเจ้าเป็นอย่างดีเสมอมา

ด้วยคุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าหวังเป็นอย่างยิ่งว่า นิสิตนักศึกษา หรือผู้สนใจ จะสามารถนำไปเป็นส่วนหนึ่งในการค้นคว้าเพิ่มเติมเพื่อให้เกิดประโยชน์สูงสุดแก่การพัฒนาต่อไป

ซึ่งคุณค่า ประโยชน์สูงสุดและความดีของวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้แก่ครอบครัว อันเป็นที่รักของข้าพเจ้า

วรุต อนุชชาลาคม

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

จากอดีตถึงปัจจุบัน Internet มีการเติบโตอย่างต่อเนื่องและรวดเร็ว ในโลกของ Internet มี protocol อยู่ 2 ชนิด ที่มีบทบาทอย่างมาก คือ Transmission Control Protocol (TCP) และ Internet Protocol (IP) รวมเรียกว่าชุด TCP/IP protocol โดยที่ IP ถูกใช้เป็น protocol หลักในชั้นของ network layer ส่วน TCP ถูกใช้ในชั้นของ transport layer เนื่องจากมีคุณสมบัติในด้าน flow control, reliability และ connection-oriented mechanism [1] TCP ใช้กลไกในการควบคุมปริมาณในการส่งข้อมูลแบบ window-based end-to-end คือ TCP จะทำการตรวจสอบสภาวะของ network และควบคุมอัตราการส่งข้อมูลด้วยตัวเอง อุปกรณ์ประเภท gateway และ router ที่อยู่ระหว่างเส้นทางมีหน้าที่จัดส่ง packet ไปตามเส้นทางโดยไม่ต้องทำการสื่อสารโดยตรงกับ TCP แต่อย่างใด TCP sender (TCP ที่ฝั่งส่ง) จะใช้การตรวจพบว่า packet ถูก drop (คัดทิ้ง) หรือสูญหายและจะทำการเปลี่ยนแปลงขนาดของ congestion window [2] ให้เหมาะสมเพื่อควบคุมปริมาณในการส่งข้อมูลด้วยตัวเอง แต่ด้วยกลไกของ TCP เองจะเกิดความล่าช้าในการตรวจพบ packet ถูก drop หรือสูญหาย ยิ่งไปกว่านั้นในขณะที่ TCP ยังไม่ทราบถึงความแออัดที่เกิดขึ้นแล้วพยายามส่งข้อมูลจำนวนมากเพิ่มเข้าไปในเครือข่าย จะทำให้เกิดความแออัดมากขึ้นไปอีกซึ่งจะทำให้ packet ถูก drop เป็นจำนวนมากและยังส่งผลต่อปริมาณการส่งผ่านข้อมูลที่ลดลงด้วย วิธีการที่สามารถนำมาช่วยแก้ปัญหานี้ควรจะต้องทำงานอยู่ที่ gateway หรือ router เพื่อจะได้ตรวจพบความแออัดได้อย่างรวดเร็วและทำการแจ้งให้ TCP ปรับอัตราการส่งให้เหมาะสม ซึ่งจะช่วยลดปริมาณ packet ถูก drop ลงได้

1.2 ความมุ่งหมายและวัตถุประสงค์ในการทำวิทยานิพนธ์

1. เพื่อปรับปรุงประสิทธิภาพของเครือข่าย TCP/IP ในด้านการลดปริมาณ packet ที่ถูก drop จากเครือข่ายและทำให้มีอัตราการส่งข้อมูลที่สูง
2. อยู่ภายใต้มาตรฐาน TCP/IP เพื่อให้เกิดความเข้ากันได้กับระบบ TCP/IP ที่ถูกพัฒนาขึ้นมาใช้งานอยู่แล้วเป็นจำนวนมาก
3. ง่ายในการพัฒนาขึ้นมาใช้งาน ซึ่งจะช่วยให้เกิดการพัฒนาบนระบบที่หลากหลายได้อย่างรวดเร็วและมีค่าใช้จ่ายที่ต่ำ

1.3 สมมติฐานของการศึกษา

จากกลไกของ TCP เองที่เกิดความล่าช้าในการตรวจพบ packet ถูก drop การลดปริมาณ packet ถูก drop โดยการแจ้งข้อมูลสภาพแออัดจากกลไกที่อยู่ใน router ที่อยู่ในเครือข่ายให้ TCP ทราบล่วงหน้าจะเป็นการช่วยให้เครือข่ายมีปริมาณ packet ถูก drop ลดลงและนำไปสู่ปริมาณการส่งผ่านข้อมูลที่เพิ่มขึ้น กลไกใน router ในกล่าวถึงนี้ก็คือกลไกเพื่อจัดการ queue ซึ่งมีหลากหลายชนิด แต่เนื่องจากต้องการให้เกิดความง่ายในการพัฒนาแต่มีประสิทธิภาพจึงได้ทำการพัฒนาบน Drop-Tail ที่มีใช้อยู่แล้วใน router ทั่วไปที่สามารถทำการพัฒนาเพิ่มได้ไม่ยาก และทั้งนี้ยังคงทำงานร่วมกับ TCP/IP ทั่วๆไปได้เพราะยังคงพัฒนาอยู่บน TCP/IP header มาตรฐาน

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในงานวิจัย

วิธีการที่นำมาใช้ใน gateway หรือ router เพื่อกำหนดหรือลดปริมาณการส่งของ TCP มีทั้งแบบแจ้งข้อมูลว่ามีการคาดการณ์ว่าจะเกิดความแออัดแล้วให้ TCP ทำการปรับลดปริมาณการส่งเหมือนกับ TCP พบ packet ถูก drop และแบบที่ทำการคำนวณอัตราการส่งที่เหมาะสมให้กับ TCP โดยตรงเลย Explicit Congestion Notification (ECN) [3] ถูกนำเสนอขึ้นมาเพื่อใช้แจ้งข้อมูลว่ามีสภาพแออัดโดยกำหนดลงไปใน IP header [4] เพื่อให้อุปกรณ์ประเภท IP gateway หรือ IP router ส่งสัญญาณแจ้งเตือนก่อนที่จะเกิดความแออัดในเครือข่าย โดยในขั้นแรกได้เสนอให้ทำการใส่กลไก ECN ลงใน gateway ที่จัดการ queue แบบ active management queue ได้แก่ Random Early Detection (RED) [5, 6] ซึ่ง queue แบบ RED เป็น queue ที่ยังมีการนำไปใช้ในงานจริงไม่มาก เนื่องจากยังไม่มีงานวิจัยที่แสดงผลดีของ RED อย่างชัดเจนและยากต่อการกำหนดค่า parameter ทั้ง 4 (minimum threshold min_{th}, maximum threshold max_{th}, maximum drop probability max_p, queue weight w_q) เพื่อให้ได้ประสิทธิภาพดี [7, 8]

ถัดมาคือวิธีการแบบที่มีการคำนวณอัตราการส่งให้กับ TCP โดยตรงคือวิธี TCP Rate Control [9] ที่อาศัยการคำนวณขนาดของ window และแก้ไขค่าใน receiver's advertised window ใน TCP acknowledgment (ACK) เพื่อส่งกลับไปที่ TCP sender โดย gateway เพื่อให้สอดคล้องกับสภาพ network ระหว่างทาง แต่มีความยากต่อการกำหนดค่า parameter เพื่อให้ได้ประสิทธิภาพดีในหลายรูปแบบของ network และ ใน paper ทำการทดสอบกับ TCP Reno แบบเก่าเท่านั้น ซึ่งยังไม่มีการทดสอบวิธีการนี้เมื่อใช้งานกับ TCP SACK ที่นิยมใช้ในระบบปฏิบัติการรุ่นใหม่ๆ

1.5 ขอบเขตของการวิจัย

วิทยานิพนธ์ฉบับนี้ทำการศึกษาเพื่อปรับปรุงประสิทธิภาพของการสื่อสารด้วย TCP/IP บนเครือข่ายแบบ WAN ขนาดใหญ่ที่มีสภาพความแออัดที่สูงเนื่องจากมี traffic จำนวนมากและไม่

แน่นอนเกิดขึ้นใน Internet อยู่ตลอดเวลา โดยการพัฒนากลไกที่ทำงานอยู่ภายใน router ที่สามารถคาดการณ์สภาพความแออัดในเครือข่ายและทำการแจ้งเตือน TCP เพื่อลดปริมาณของ packet ที่ถูก drop และเพิ่มปริมาณการส่งข้อมูลบนเครือข่ายการสื่อสาร TCP/IP โดยการเลือกใช้ TCP Reno และ TCP SACK ที่มีการใช้แพร่หลายในปัจจุบันเป็นตัวแทนของพฤติกรรมของ TCP ที่ทำงานบนเครือข่าย WAN แบบ T3 ที่มี bandwidth 45Mbps การเชื่อมต่อของ TCP ฝั่งส่งและ TCP ฝั่งรับจะกระทำผ่าน router จำนวน 2 ตัว เพื่อเป็นตัวแทนของเครือข่าย Internet ที่ประกอบไปด้วย router จำนวนมาก เพื่อไม่ให้การกำหนดเส้นทางในระดับ IP ของ router มามีผลในการเลือกเส้นทางจึงทำการกำหนดให้มีเส้นทางเชื่อมต่อระหว่าง router ทั้ง 2 เพียงแค่ 1 เส้นทางและเป็นคอขวดของเครือข่ายจำลองนี้ สภาพความแออัดที่เกิดขึ้นจากการส่งผ่านข้อมูลของ FTP ที่มีการส่งข้อมูลอย่างต่อเนื่องตามรูปแบบ traffic ที่ถูกกำหนดขึ้นมา 6 แบบเพื่อเป็นตัวแทนของ traffic ในแบบต่างๆมีตั้งแต่ปริมาณ traffic น้อย, มาก, น้อยไปมาก และ มากไปน้อย ในด้านการทดสอบประสิทธิภาพได้นำ Network Simulator 2 (NS-2) ที่เป็นโปรแกรมจำลองเครือข่ายมาใช้ในการสร้างกลไกที่นำเสนอและสร้างเครือข่ายจำลองขึ้นมาเพื่อทดสอบ ถ้า NS-2 มีความถูกต้อง ผลที่ได้จากการจำลองจะเหมือนกับผลที่ได้จากการทำงานบนเครือข่ายจริงๆ

1.6 โครงร่างของวิทยานิพนธ์

เนื้อหาในวิทยานิพนธ์ฉบับนี้ถูกนำเสนออย่างเป็นลำดับเรียงตามบทจาก 2 ถึง 8 ดังนี้

บทที่ 2 Protocols และชุด TCP/IP protocol: จะบรรยายถึงประวัติความเป็นมาของ Internet และโครงสร้างพื้นฐานของชุด TCP/IP protocol เพื่อให้เกิดความเข้าใจถึงภาพรวมในการสื่อสารด้วย TCP/IP

บทที่ 3 Transmission Control Protocol (TCP): ถัดจากบทที่ 2 ซึ่งได้กล่าวถึงพื้นฐาน TCP/IP ไว้บ้างแล้ว บทนี้จะบรรยายถึงการทำงานของ TCP อย่างละเอียด โดยเฉพาะเรื่องของ flow control และ congestion control เพื่อให้เกิดความเข้าใจต่อพฤติกรรมในการส่งข้อมูลของ TCP โดยเฉพาะเรื่องที่มีผลต่ออัตราการส่งข้อมูล

บทที่ 4 Network Simulator 2 (NS-2): สำหรับบทนี้เป็นการแนะนำโปรแกรมสำหรับสร้างเครือข่ายจำลองที่ดีโปรแกรมหนึ่ง การจำลองการทำงานเป็นวิธีที่ได้รับความนิยมเนื่องจากมีค่าใช้จ่ายต่ำ แต่ต้องทำการปรับแต่งค่าตัวแปรต่างๆให้มีความใกล้เคียงเครือข่ายที่ต้องการจำลองให้มากที่สุดจึงจะได้ผลที่ถูกต้องน่าเชื่อถือ

บทที่ 5 นำเสนอวิธี Drop-Tail with ECN: จากกลไกที่ถูกนำมาใช้ในเครือข่ายแบบ TCP/IP ทั่วไปทำให้เกิดพฤติกรรมตามที่กล่าวไว้ในบทที่ 2 และ 3 ซึ่งในบทนี้จะนำเสนอกลไกที่นำเสนอโดยวิทยานิพนธ์ฉบับนี้ที่ถูกปรับปรุงเพื่อเพิ่มประสิทธิภาพให้กับการสื่อสารด้วยเครือข่าย TCP/IP

บทที่ 6 การทำการ simulation: ด้วยแนวคิดในบทที่ 5 เราได้ทำการสร้างสถานการณ์ขึ้นมาหลายๆ แบบ และทำการจำลองเครือข่ายด้วย NS-2 เพื่อนำไปเปรียบเทียบกับกลไกอื่นๆที่เป็นที่รู้จักที่ถูกนำมาทดสอบด้วยสถานการณ์แบบเดียวกัน

บทที่ 7 ผลที่ได้จากการทำ simulation: แสดงถึงผลที่ได้จากการจำลองทั้งหมด 96 รูปแบบ ผลลัพธ์ในแต่ละรูปแบบถูกนำมาประมวลผลและรวบรวมในรูปของ ปริมาณข้อมูลที่ส่งได้ และปริมาณข้อมูลที่ถูกคัดทิ้ง รวมถึงได้ทำการวิเคราะห์ผลที่ได้ในส่วนท้ายด้วย

บทที่ 8 บทสรุปและงานวิจัยลำดับถัดไป: เป็นการสรุป วิจารณ์ เสนอแนะจากผลที่ได้ และข้อจำกัดเพื่อเป็นแนวทางการในการนำไปประยุกต์ใช้ในลำดับต่อไป

บทที่ 2

Protocols และชุด TCP/IP protocol

เพื่อให้เกิดความเข้าใจในพื้นฐานการทำงานของ TCP/IP และการทำงานร่วมกับ TCP/IP แบบภาพกว้าง บทนี้จะเริ่มจากการบรรยายถึงประวัติความเป็นมาของ Internet เพื่อให้ผู้อ่านทราบถึงความเป็นมาและความเปลี่ยนแปลงของ Internet ต่อมาจะบรรยายถึงโครงสร้างพื้นฐานของชุด TCP/IP protocol เพื่อให้เกิดความเข้าใจถึงภาพรวมในการสื่อสารด้วย TCP/IP ถัดมาจะบรรยายถึงการกำหนดมาตรฐานสำหรับใช้กับ Internet จากนั้นเราจะพูดถึงโครงสร้างของ OSI reference model รวมทั้งการเปรียบเทียบกับ TCP/IP และสุดท้ายจะเป็นการยกตัวอย่างการเชื่อมต่อข้ามเครือข่ายเพื่อให้เกิดความเข้าใจต่อชุด TCP/IP protocol ดียิ่งขึ้น

2.1 ประวัติโดยย่อของระบบเครือข่าย Internet

ในปัจจุบันได้มีการพัฒนา protocol ใหม่ๆ และ กลไกใหม่ๆมาใช้ในเครือข่ายคอมพิวเตอร์อย่างกว้างขวาง ปัจจัยที่ทำให้มีการพัฒนา protocol และ กลไกในการสื่อสารข้อมูลและระบบเครือข่ายคอมพิวเตอร์ก็คือ การเติบโตของ Internet Internet ในปัจจุบันมีที่มาจาก Arpanet ที่เริ่มวางเครือข่ายในปี ค.ศ. 1966 และได้เริ่มเปิดใช้งานเมื่อปี ค.ศ. 1969 โดยเริ่มต้นจากเพียง 4 packet-switching node ที่เชื่อมต่อ host computer และ terminal Link ที่เชื่อมต่อระหว่าง node สามารถส่งผ่านข้อมูลได้สูงสุดที่ 50 kbps โดยมี Advanced Research Projects Agency (ARPA) ของ U.S. Department of Defense (DoD) เป็นผู้ให้ทุนสนับสนุน Arpanet มุ่งเน้นในการพัฒนาเทคโนโลยีด้าน packet-switching และ protocol ที่ใช้ในการสื่อสารในระบบคอมพิวเตอร์

Application ที่ถูกพัฒนาใช้กับ ARPANET หลายๆตัวได้ถูกเพิ่มเติมความสามารถเพื่อให้เกิดประโยชน์สอดคล้องกับการพัฒนาระบบเครือข่าย application ที่มีความสำคัญมาก 2 ตัวคือ TELNET และ FTP TELNET ถูกใช้ในการควบคุมคอมพิวเตอร์จากระยะไกล เมื่อตอนที่ ARPANET ได้ถูกจัดตั้งขึ้น คอมพิวเตอร์แต่ละแบบใช้ terminal ในการควบคุมที่แตกต่างกันออกไป TELNET ทำให้เกิดความเข้ากันได้ของการควบคุมจากระยะไกล ถ้าชนิดของ computer นั้นรองรับ "TELNET terminal" ทำให้ terminal ตัวเดียวทำการสั่งงานได้กับ computer ทุกๆเครื่อง ส่วน FTP เป็น มาตรฐานในการถ่ายโอน file ข้ามเครือข่ายจาก computer เครื่องหนึ่งไปยังอีกเครื่องหนึ่ง อาจฟังดูเหมือนจะเป็นเรื่องที่ง่ายๆแต่จริงๆแล้ว computer แต่ละชนิดจะมีลักษณะหลายๆอย่างที่แตกต่างกัน เช่น ขนาดของ word ที่ไม่เท่ากัน การเรียงลำดับจัดเก็บก็แตกต่างกัน และรูปแบบของ word ก็ยังไม่เหมือนกัน

ตารางที่ 2.1 วิวัฒนาการของระบบเครือข่าย Internet

ปี ค.ศ.	เหตุการณ์
1966	เริ่มโครงการ ARPA packet-switching
1969	Arpanet node แรกเริ่มเปิดใช้งาน
1972	เริ่มใช้งาน e-mail
1973	ทำการเชื่อมต่อ Arpanet ออกสู่ประเทศอื่นๆนอกอเมริกา
1975	Arpanet ถูกโอนมาเป็นของ Defense Communications Agency
1980	มีการริเริ่มใช้ TCP/IP
1981	มีคอมพิวเตอร์แม่ข่ายเพิ่มขึ้นทุกๆ 20 วัน
1983	ระบบ TCP/IP มีความสมบูรณ์ในการใช้งาน
1986	NSFNET backbone ถูกสร้างขึ้นมา
1990	Arpanet ถูกยกเลิก
1991	เริ่มมีการใช้ Gopher
1991	เริ่มมีการใช้ WWW
1992	เริ่มมีการใช้ Mosaic
1995	เปลี่ยนแปลงผู้ถือ Internet backbone
1996	OC-3 (155 Mbps) backbone ถูกสร้างขึ้นมา

อย่างไรก็ตาม TELNET และ FTP ถือได้ว่ามีประโยชน์มาก แต่ตัวที่ถือว่าเป็น “killer app” (โปรแกรมที่ดีมากๆ) ตัวแรกของ ARPANET คือ electronic mail ก่อนที่จะมี ARPANET ได้มีการใช้งาน electronic mail มาบ้างแล้ว แต่จำกัดอยู่ที่ภายในระบบ computer เดียว ในปี ค.ศ. 1972 Ray Tomlinson ของ Bolt Beranek and Newman (BBN) ได้เขียนโปรแกรมสำหรับจัดส่ง mail ข้ามเครือข่ายคอมพิวเตอร์ ที่ใช้กับ computer จำนวนมากได้สำเร็จ ภายในปี ค.ศ. 1973 ARPA ได้ทำการสำรวจพบว่า 3 ใน 4 ของปริมาณข้อมูลใน ARPANET คือ e-mail นั่นเอง E-mail เป็นตัวดึงดูดให้ผู้คนสนใจในตัว ARPANET จนทำให้เกิดการเพิ่มขยาย node และเพิ่มความเร็วของ link อย่างต่อเนื่อง

ARPANET เติบโตขึ้นเรื่อยๆ และเป็นที่ได้รับความสนใจไม่เฉพาะแค่จากหน่วยงานราชการ และ สถานศึกษา แต่ยังรวมไปถึงผู้ที่ใช้งานอย่างเป็นทางการภายใน DoD เอง การบริหารจัดการ network กลายเป็นหัวข้อที่สำคัญขึ้นเรื่อยๆ รวมถึงเสถียรภาพของระบบเครือข่าย ในปี ค.ศ. 1975 ได้ทำการถ่ายโอนความรับผิดชอบ ARPANET จากการสนับสนุนโดย ARPA มาเป็นของ Defense Communications Agency

ด้วยความก้าวหน้าทางเทคโนโลยีทำให้ ARPA ประสบความสำเร็จในการประยุกต์ใช้ packet-switching ส่งไปในคลื่นวิทยุ (packet radio) และ ดาวเทียม (SATNET) เนื่องจาก network ทั้ง 3 แบบต้องการการจัดการที่แตกต่างกัน เนื่องจากสภาพแวดล้อมที่ไม่เหมือนกัน การกำหนดค่าให้กับระบบก็จะต้องแตกต่างกันด้วย เช่น ขนาดสูงสุดของ packet (maximum packet size) ที่มีขนาดไม่เท่ากัน การเชื่อม network ทั้ง 3 แบบเข้าด้วยกัน ทำให้เกิดปัญหาใหม่ขึ้น Vint Cerf และ Bob Kahn ของ ARPA จึงได้เริ่มพัฒนาวิธีการและ protocol สำหรับ internetworking (การเชื่อมต่อข้ามเครือข่าย) พวกเขาได้เผยแพร่งานวิจัยออกมาเมื่อ พฤษภาคม ค.ศ. 1974 [CERF74] ซึ่งเป็นแนวทางไปสู่ transmission control protocol ข้อเสนอแนะนี้ได้ถูกนำมาพัฒนาต่อโดยกลุ่ม ARPANET แล้วนำเสนอในชื่อ TCP และ IP protocol ซึ่งเรียกรวมๆว่าชุด TCP/IP protocol protocol ชุดนี้ถือได้ว่าเป็นการวางรากฐานให้กับ internet ในปี ค.ศ. 1982-1983 ARPANET ได้ทำการเปลี่ยน NCP protocol ที่ใช้อยู่เดิมมาเป็น TCP/IP ทำให้เครือข่ายอื่นๆที่เชื่อมต่อด้วยทั่วโลกเปลี่ยนมาเป็น TCP/IP

ARPANET และ INTERNET มีประโยชน์อย่างมาก ในปี ค.ศ. 1980-1981 The Nation Science Foundation (NSF) ได้ขยายการรองรับไปยังกลุ่มนักวิจัยคอมพิวเตอร์ผ่านทาง CSNET ในปี ค.ศ. 1986 NSF ขยายการรองรับ Internet ไปยังกลุ่มนักวิจัยอื่นๆผ่านทาง NSFNET ในตอนแรก NSFNET ถูกออกแบบเพื่อเชื่อมต่อระหว่าง 6 ศูนย์ super computer ในที่สุด NSF ได้เสนอให้ใช้ backbone ของศูนย์ super computer เพื่อเป็น backbone ของ packet-switched network แทน ARPANET ในปี ค.ศ. 1990 ARPANET ก็ได้ปิดตัวลง

2.2 Protocols และชุด TCP/IP protocol

เราจะทำการเริ่มต้นด้วยการแนะนำหลักการของแต่ละชั้นของโครงสร้าง protocol (layered protocol architecture) หลังจากนั้นเราจะเน้นไปที่ชุดของ TCP/IP protocol TCP/IP เป็นพื้นฐานและกรอบข้อกำหนดของ Internet ถูกพัฒนาขึ้นมาสำหรับทำการเชื่อมต่อคอมพิวเตอร์หลายๆแบบเข้าหากัน ผู้ผลิตคอมพิวเตอร์แทบทุกรายจะสนับสนุน TCP/IP โครงสร้างอื่นที่เป็นที่รู้จักกันดีคือโครงสร้างอ้างอิง OSI (OSI reference model) OSI เป็นโครงสร้างที่เป็นมาตรฐานที่นิยมใช้ในการบรรยายหน้าที่ของแต่ละส่วนย่อยในการสื่อสาร แต่ไม่มีการพัฒนาเพื่อใช้งานจริง

2.2.1 ประโยชน์ของ protocol layer

เมื่อ computer, terminal และ อุปกรณ์สำหรับประมวลผลอื่นๆ มีความต้องการที่จะแลกเปลี่ยนข้อมูลกัน ขั้นตอนที่ต้องใช้อาจมีความยุ่งยากมาก เช่นตัวอย่างการถ่ายโอนแฟ้มข้อมูลระหว่างคอมพิวเตอร์ 2 เครื่อง จำเป็นจะต้องมีช่องทางให้ข้อมูลเดินทางผ่านระหว่างคอมพิวเตอร์

ทั้งสอง จะโดยการเชื่อมต่อกันโดยตรงหรือผ่านเครือข่ายก็แล้วแต่ แต่ยังไม่พอ เพราะอย่างน้อยจะต้องมีการทำสิ่งเหล่านี้ด้วย

1) คอมพิวเตอร์ต้นทางต้องกำหนดเส้นทางเดินของข้อมูลโดยตรงหรือแจ้งให้เครือข่ายทราบถึงคอมพิวเตอร์ปลายทาง
2) คอมพิวเตอร์ต้นทางจะต้องตรวจสอบให้แน่ใจว่าคอมพิวเตอร์ปลายทางเตรียมพร้อมรับข้อมูล
3) โปรแกรมประยุกต์สำหรับโอนย้ายแฟ้มข้อมูลบนคอมพิวเตอร์ต้นทางจะต้องตรวจสอบว่าโปรแกรมจัดการแฟ้มที่ฝั่งปลายทางพร้อมที่จะรับและจัดเก็บแฟ้มนี้สำหรับผู้ใช้ที่ได้ระบุไว้
4) ถ้ารูปแบบของแฟ้มที่ใช้ในคอมพิวเตอร์ทั้งสองไม่เหมือนกันจะต้องมีการแปลงรูปแบบ

เห็นได้ชัดเจนว่าขั้นตอนทั้งหมดต้องอาศัยความร่วมมือกันระหว่างคอมพิวเตอร์ 2 เครื่อง แทนการสร้างทุกขั้นตอนทั้งหมดภายใน module เดียว การแยกออกเป็นส่วนย่อยๆที่แต่ละส่วนเป็นอิสระจากกันให้มากที่สุด ทำให้สะดวกในการพัฒนาและทำให้มีความยืดหยุ่นมากกว่า ในโครงสร้างของ protocol เราแบ่งแต่ละส่วนเป็นชั้นๆเรียงในแนวตั้ง แต่ละ layer ในโครงสร้างแบบนี้จะทำงานเฉพาะงานที่มีลักษณะเกี่ยวข้องกันเท่านั้น และเปิดช่องบริการให้กับ layer ที่สูงกว่าข้างบนถัดไป การแก้ไขหรือเปลี่ยนแปลง layer ใดๆจะไม่กระทบต่อ layer อื่นๆ

แน่นอนว่า protocol แบบเดียวกันต้องถูกนำมาใช้ในคอมพิวเตอร์ทั้ง 2 ด้าน การสื่อสารจะกระทำระหว่างจุดต่อจุด ระหว่าง layer ที่ชั้นเดียวกันของคอมพิวเตอร์ทั้งสอง คุณสมบัติที่สำคัญของ protocol มีดังนี้

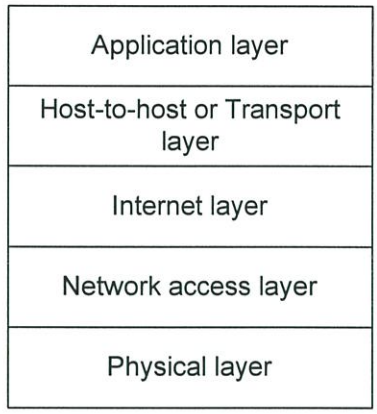
- Syntax: กำหนดรูปแบบของข้อมูล
- Semantics: ข้อมูลสำหรับควบคุมและการจัดการข้อผิดพลาด
- Timing: กำหนดความเร็วและลำดับ

2.2.2 โครงสร้างของ TCP/IP protocol

TCP/IP เป็นผลงานจากการวิจัยและพัฒนา เริ่มต้นจากบนเครือข่ายแบบ packet-switched network ที่เรียกว่า ARPANET จากการสนับสนุนของ Defense Advanced Research Projects Agency (DARPA) และทั่วๆไปเราเรียกรวมๆว่าชุดของ TCP/IP protocol ชุด protocol นี้ประกอบไปด้วยหลายๆ protocol ย่อยที่กำหนดให้เป็นมาตรฐานโดย Internet Architecture Board

2.2.2.1 TCP/IP layers

มาตรฐานของ TCP/IP ไม่ได้มีการกำหนดอย่างเป็นทางการเหมือนในกรณีของ OSI แต่จากพื้นฐานของ protocol มาตรฐานที่ได้มีการพัฒนา เราสามารถจัดแบ่งชั้นของ TCP/IP ออกได้เป็น 5 layer ที่ค่อนข้างอิสระจากกันดังแสดงในรูป 2.1



รูปที่ 2.1 โครงสร้าง Protocol Layer

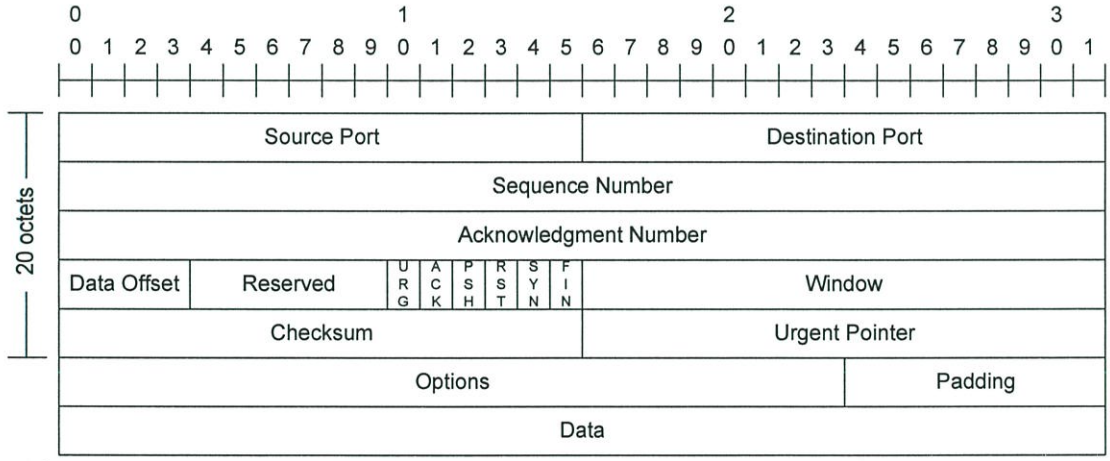
Physical layer กำหนดถึงมาตรฐานการเชื่อมต่อทางกายภาพระหว่างอุปกรณ์ที่ใช้ส่งข้อมูลกับสื่อกลางในการส่งข้อมูล ในชั้นนี้จะกำหนดคุณสมบัติของสื่อกลาง, คุณสมบัติของสัญญาณ, อัตราการส่ง และ อื่นๆในลักษณะดังกล่าว Network access layer กำหนดการแลกเปลี่ยนข้อมูลระหว่างคอมพิวเตอร์ปลายทางและเครือข่ายที่ต่อเชื่อมกัน คอมพิวเตอร์ต้นทางที่ต้องการส่งข้อมูลจะต้องบอกที่อยู่ของคอมพิวเตอร์ปลายทาง เพื่อเครือข่ายจะได้จัดเส้นทางที่เหมาะสมให้ข้อมูลกลุ่มนี้ผ่าน คอมพิวเตอร์ต้นทางอาจจะร้องขอบริการแบบพิเศษ เช่น ต้องการให้มีลำดับความสำคัญที่สูงกว่าปกติ ที่อาจจะเปิดให้บริการจากเครือข่าย software ที่อยู่ใน layer นี้จะขึ้นอยู่กับชนิดของเครือข่ายที่ใช้ มาตรฐานต่างๆส่วนใหญ่พัฒนาขึ้นมาใช้กับ circuit switching, packet switching (เช่น X.25), local area network (เช่น Ethernet) และอื่นๆ การแยกหน้าที่นี้ออกมาดูแลโดย layer นี้ทำให้การใช้เครือข่ายของ layer ที่สูงกว่าไม่ต้องสนใจว่า network ข้างล่างจะเป็นชนิดใด Network access layer ยังกำหนดถึงวิธีการเข้าถึงและการจัดเส้นทางการส่งผ่านข้อมูลข้ามเครือข่ายสำหรับคอมพิวเตอร์ 2 เครื่องที่ต่ออยู่บนเครือข่ายเดียวกัน แต่ถ้าคอมพิวเตอร์ทั้งสองอยู่บนเครือข่ายที่ต่างกัน กระบวนการในการที่จะให้ข้อมูลเดินทางข้ามหลายๆเครือข่ายจะเป็นหน้าที่ของ internet layer ที่ชั้นสูงขึ้นถัดไป Internet protocol ถูกใช้ที่ในชั้นนี้เพื่อทำหน้าที่กำหนดเส้นทางการเดินทางข้ามเครือข่าย protocol นี้ไม่เฉพาะแต่ใช้ในคอมพิวเตอร์ปลายทางเท่านั้น แต่ยังใช้ใน router ด้วย router คือ ตัวประมวลผลรูปแบบหนึ่งที่ต่ออยู่ตรงกลางเชื่อม 2 เครือข่ายหรือมากกว่าเข้าด้วยกัน และมีหน้าที่หลักในการจัดส่งข้อมูลจาก network หนึ่งไปยังอีก network หนึ่ง เพื่อให้ข้อมูลเดินทางจากต้นทางไปยังปลายทางได้อย่างถูกต้อง

โดยธรรมชาติของโปรแกรมประยุกต์ที่ต้องมีการแลกเปลี่ยนข้อมูลกัน จะต้องมีการกำหนดวิธีการขึ้นมาเพื่อให้แน่ใจว่าข้อมูลทั้งหมดถูกส่งมาจากต้นทางถึงปลายทางครบถ้วนและเรียงตามลำดับก่อนหลังถูกต้อง ซึ่งกลไกที่สร้างเสถียรภาพและความถูกต้องให้กับการส่งข้อมูลนี้ไม่ขึ้นกับลักษณะเฉพาะของโปรแกรมประยุกต์ ดังนั้นจึงควรที่จะกำหนดชั้นของ protocol ขึ้นมาอีก 1 ชั้น เพื่อทำหน้าที่นี้ เราเรียกว่าชั้นของ host-to-host layer หรือ transport layer Transmission

control protocol (TCP) เป็นที่รู้จักกันดีในการทำหน้าที่ในชั้น protocol นี้ ชั้นสุดท้ายที่อยู่บนสุดของโครงสร้างคือ application layer ซึ่งจะประกอบไปด้วยกลไกที่รองรับการใช้งานของผู้ใช้ และเป็นชั้นที่มีความหลากหลายมากที่สุด ตัวอย่างเช่น โปรแกรมขนย้ายแฟ้ม

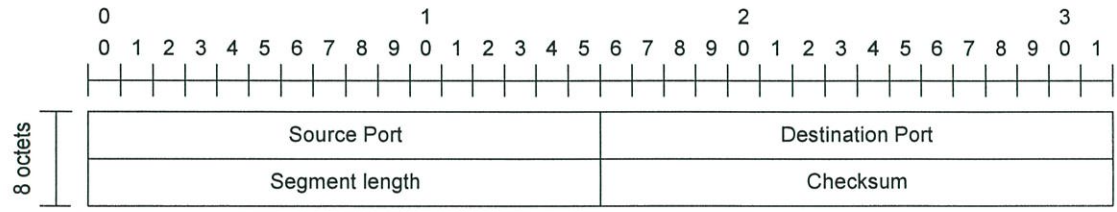
2.2.2.2 TCP กับ UDP

สำหรับโปรแกรมประยุกต์ส่วนใหญ่ที่ทำงานอยู่บนชุด TCP/IP protocol จะใช้ TCP ในชั้นของ transport layer เพื่อทำหน้าที่ดูแลความถูกต้องของการแลกเปลี่ยนข้อมูลระหว่างกันของโปรแกรม ในรูปที่ 2.2 แสดงรูปแบบส่วนหัวที่ใช้ใน TCP ซึ่งมีขนาดต่ำสุดอยู่ที่ 20 octets (1 octet = 1 byte) หรือ 160 bit port ต้นทางและปลายทางถูกระบุไว้เพื่อให้ทราบว่าข้อมูลเป็นของโปรแกรมใด port ใน TCP มีลักษณะคล้ายกับ Service Access Point (SAP) ใน OSI model sequence number ใช้แสดงลำดับของกลุ่มข้อมูล acknowledgment number ใช้แสดงลำดับของกลุ่มข้อมูลที่ได้รับแล้ว และ window ใช้เพื่อควบคุมปริมาณการส่งและควบคุมความผิดพลาด (flow control และ error control) checksum มีขนาด 16 bit เพื่อตรวจสอบว่ามีข้อมูลผิดพลาดภายใน TCP segment หรือไม่

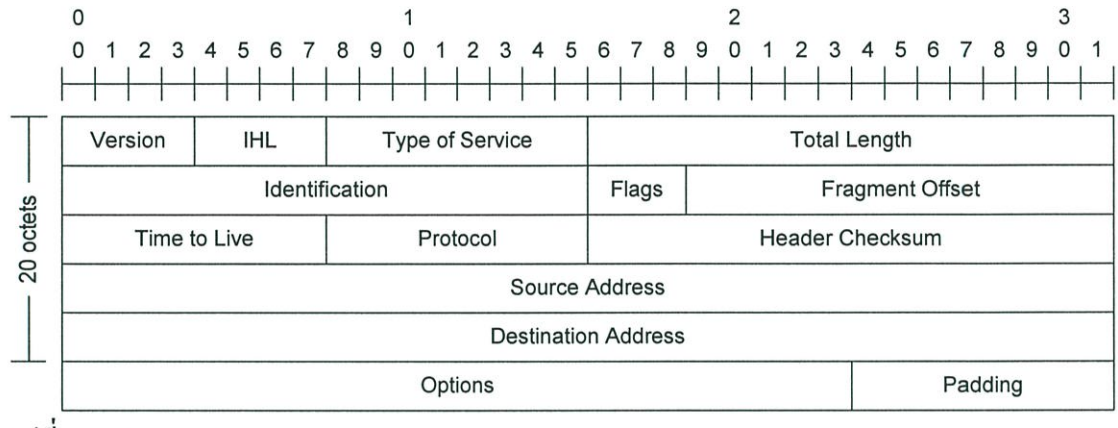


รูปที่ 2.2 Header format ของ TCP

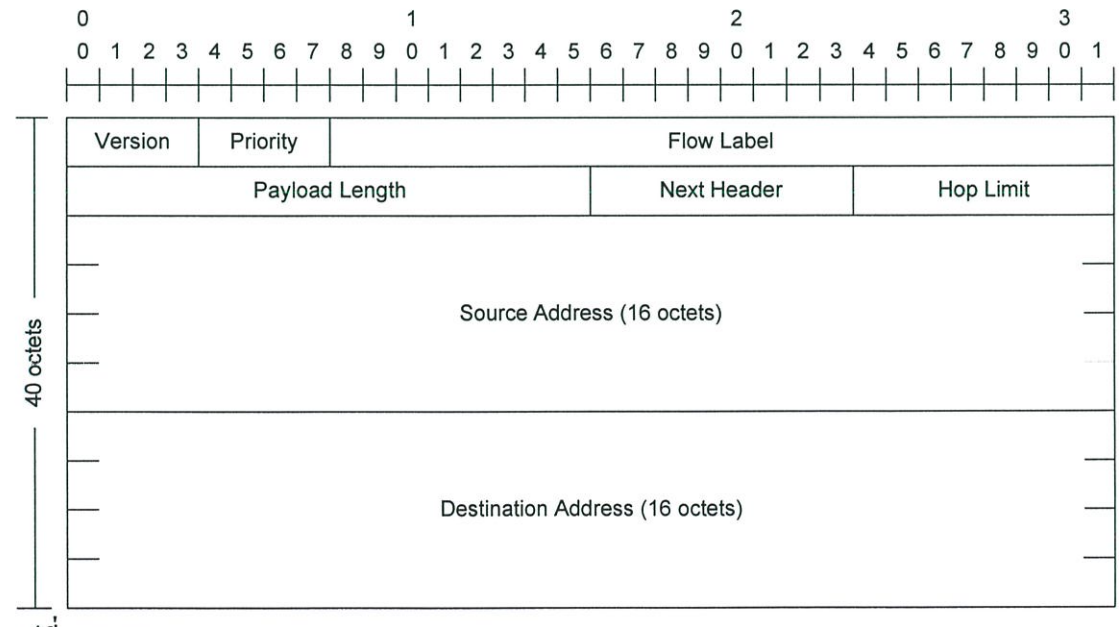
นอกจาก TCP แล้วยังมี protocol อีกหนึ่งตัวในชั้น transport ที่เป็นส่วนหนึ่งในชุด TCP/IP protocol นั่นก็คือ User Datagram Protocol (UDP) UDP จะทำหน้าที่จัดส่งข้อมูลให้กับโปรแกรมชั้นบนในแบบ connectionless ซึ่งหมายความว่าจะไม่มีการประกันว่าข้อมูลที่ส่งไปจะไปถึงปลายทาง, ไปถึงแบบเรียงลำดับ หรือมีความซ้ำซ้อนหรือไม่ UDP มีลักษณะพิเศษคือ กลไกที่ใช้มีความซับซ้อนน้อยกว่า TCP มีขนาดชุดข้อมูลเล็กกว่า ทำให้มีการนำไปใช้ในงานบางประเภท เช่น SNMP (Simple Network Management Protocol) ที่เป็นมาตรฐานของการบริหารจัดการบนเครือข่าย TCP/IP รูปแบบส่วนหัวของ UDP แสดงในรูปที่ 2.3



รูปที่ 2.3 Header format ของ UDP



รูปที่ 2.4 Header format ของ IPv4



รูปที่ 2.5 Header format ของ IPv6

2.2.2.3 IPv4 และ IPv6

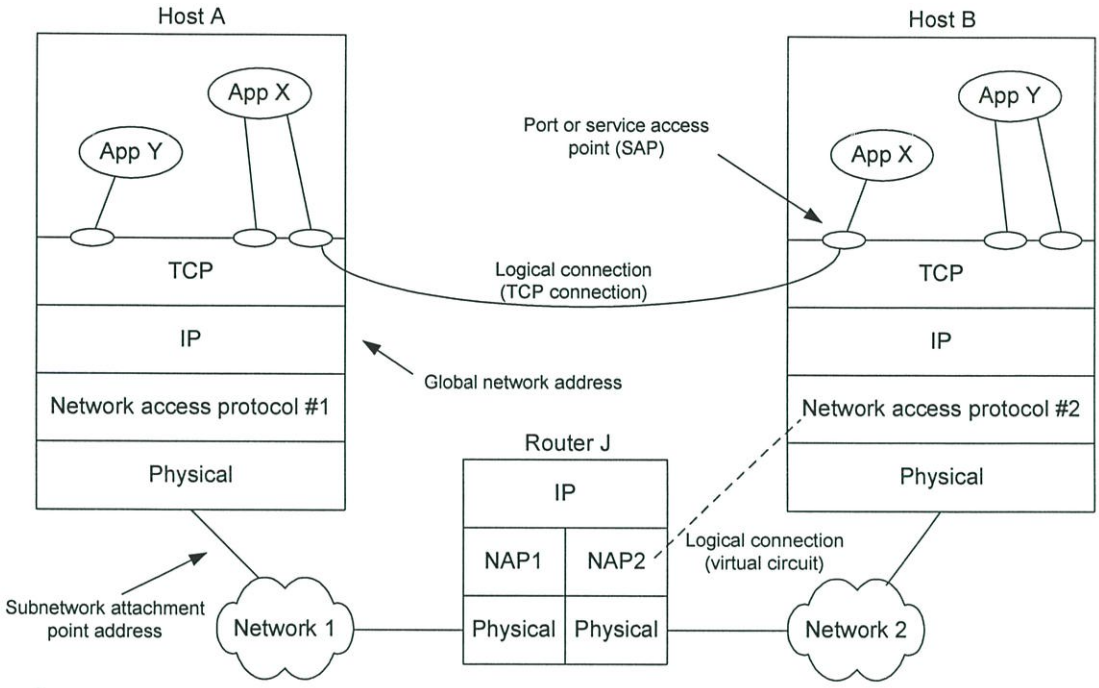
ในรูปที่ 2.4 แสดงรูปแบบส่วนหัวของ IPv4 (Internet Protocol version 4) ซึ่งมีขนาดต่ำสุดอยู่ที่ 20 octets หรือ 160 bit ในส่วนหัวจะประกอบด้วย ที่อยู่ต้นทางและปลายทางมีขนาดอย่างละ 32 bit ในส่วน header check-sum มีไว้ตรวจสอบความผิดพลาดของข้อมูลในส่วนหัวเอง Protocol field ใช้แสดงประเภทของ protocol ที่อยู่ภายในส่วนข้อมูลว่าเป็น TCP หรือ UDP หรืออาจจะเป็นรหัสของ protocol ใน layer ที่สูงกว่าอื่นๆที่ทำงานอยู่บน IP Flag และ Fragment offset field ใช้ในการแตกและรวมกลุ่มข้อมูล

ในปี ค.ศ. 1995 The Internet Engineering Task Force ที่เป็นกลุ่มที่คอยกำหนดมาตรฐานของ protocol ใน Internet ได้กำหนดมาตรฐานใหม่ของ IP ซึ่งรู้จักในนามของ IPng (IP next generation) ข้อกำหนดนี้ถูกยอมรับให้เป็นมาตรฐานในปี ค.ศ. 1996 ในชื่อ IPv6 (Internet Protocol version 6) IPv6 ถูกปรับปรุงให้มีขีดความสามารถสูงกว่า IP ในปัจจุบัน (IPv4) ในแง่ของ รองรับความเร็วที่สูงขึ้นของเครือข่ายที่พัฒนาไปมากกว่าเดิมมาก และการส่งรวมกันทั้งข้อมูล, ภาพ และเสียง ซึ่งกำลังเป็นที่นิยม แต่สาเหตุใหญ่ที่แท้จริงของการพัฒนาครั้งนี้คือ ความต้องการ IP address (เลขที่อ้างอิงประจำเครื่องที่ไม่ซ้ำกัน) ที่มากขึ้นจนเกินขนาดสูงสุดที่ IPv4 ที่มีขนาดของเลขที่อยู่ที่ 32 bit จะรองรับได้ ตามการขยายตัวของ Internet ทำให้เครือข่ายต่างๆก็ต้องการจะเชื่อมต่อเข้าสู่ Internet ซึ่งทุกเครือข่ายที่เชื่อมเข้ามาจะต้องการ IP address เพื่อกำหนดให้กับคอมพิวเตอร์และอุปกรณ์เครือข่าย ดังนั้น IPv6 จึงได้ขยายขนาดของ IP address ออกเป็น 128 bit ในรูปที่ 2.5 ซึ่งคาดว่าจะสามารถรองรับการใช้งานไปได้อีกนาน แต่ในปัจจุบันยังอยู่ในขั้นตอนการเปลี่ยนจาก IPv4 มาเป็น IPv6 ซึ่งคงต้องใช้เวลาอีกหลายปี

2.2.2.4 การทำงานของ TCP/IP

ในรูป 2.6 แสดงให้เห็นถึงการทำงานของ TCP/IP Protocol เพื่อความชัดเจนจึงได้แสดงภาพในลักษณะการเชื่อมต่อข้ามเครือข่ายที่มีการแบ่งเป็น subnetwork (เครือข่ายย่อยๆ) ระหว่าง 2 subnetwork เชื่อมต่อกันผ่าน router เรากำหนดให้ Network access protocol ที่ใช้อธิบายในรูปเป็นคนละชนิดกันโดยแสดงด้วยตัวเลข 1 และ 2 (เช่น Ethernet, Token ring) IP ถูกนำมาใช้ในคอมพิวเตอร์ทั้งสองและใน router คอยทำหน้าที่ส่งผ่านกลุ่มข้อมูลข้ามระหว่างเครือข่าย 1 กับ 2 ในโลกของ Internet จริงๆการส่งข้อมูลระหว่างคอมพิวเตอร์อาจจะต้องเดินทางข้าม router มากกว่า 1 ตัว TCP ถูกใช้ในคอมพิวเตอร์ปลายทางทั้งสองเท่านั้น เพื่อทำหน้าที่ตรวจสอบข้อผิดพลาดและรักษาความถูกต้องให้กับโปรแกรมที่เรียกใช้ในระดับ application layer เพื่อให้การติดต่อสื่อสารมีความสมบูรณ์แบบ ทุก ๆ หน่วยที่ต้องการติดต่อกันจะต้องมี IP address ที่เป็นของตัวเอง การกำหนดเลขที่นั้นมี 2 ระดับ คือ ระดับแรก แต่ละ host บนเครือข่ายย่อย ต้องมี IP address ที่เป็นเฉพาะของตัวเอง เพื่อใช้กำหนดเส้นทางในการจัดส่งข้อมูล เลขที่ชุดนี้ถูกใช้โดย IP สำหรับทำ

routing ระดับที่สอง แต่ละโปรแกรมประยุกต์ภายใน host ต้องมีเลขที่ของตัวเองที่ไม่ซ้ำกับใครใน host นั้น ซึ่งจะอนุญาตให้ host-to-host protocol (TCP) จัดส่งข้อมูลให้กับ process ที่ถูกต้อง เราเรียกเลขที่อย่างหลังนี้ว่า port



รูปที่ 2.6 การทำงานของ TCP/IP

ทีนี้เรากลับมาดูการทำงานกันต่อ สมมุติว่า มี process หนึ่ง เลขที่ 1 อยู่บน host A ต้องการส่งข้อมูลไปยัง process อื่นที่มี port เลขที่ 2 บน host B process บน host A จะส่งข้อมูลลงไปที่ TCP ที่ชั้นล่าง พร้อมกำกับคำสั่งลงไปว่าส่งไป B ที่ port 2 จากนั้น TCP จะทำการส่งผ่านข้อมูลลงไปที่ IP อีกต่อหนึ่ง (การทำงานของ IP จะไม่ขึ้นกับ port) ตอนนี้ข้อมูลทุกอย่างพร้อมแล้วสำหรับจัดส่งไปที่ B ต่อไป IP จะส่งต่อข้อมูลนี้ลงไปที่ network access layer (ในที่นี้คือ Ethernet) เพื่อส่งต่อไปยัง router J (เป็นจุดเชื่อมต่อแรกในเส้นทางไปยัง host B)

การควบคุมกระบวนการเหล่านี้ ทั้งคำสั่งควบคุมและข้อมูลจะถูกจัดส่งไปตามรูป 2.7 เริ่มจาก process ที่ต้องการส่งข้อมูลเตรียมการสร้างชุดข้อมูล (block) ที่จะส่งไปยัง TCP TCP จะแตก block ออกเป็นชุดข้อมูลที่ย่อยลงไปตามข้อกำหนด เพื่อง่ายต่อการจัดส่งและการจัดการ แต่ละส่วนย่อยจะมีข้อมูลควบคุมที่ TCP เติมลงไปในส่วนหัวดังที่แสดงในรูป 2.2 ทั้งหมดถูกเรียกรวมว่า TCP segment ข้อมูลควบคุมนี้จะถูกใช้โดย TCP ที่ฝั่งรับ (host B) ตัวอย่างของข้อมูลควบคุมในส่วนหัวมีดังนี้

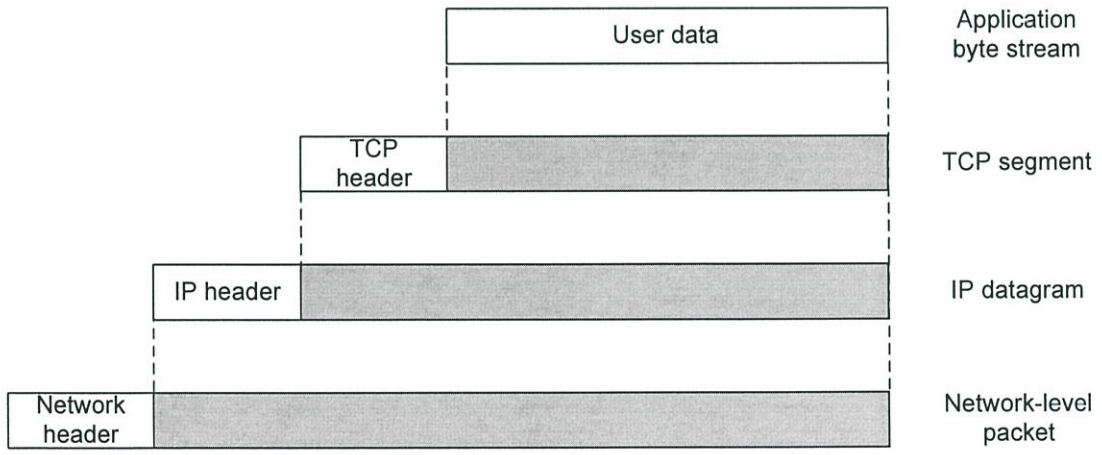
- Destination port: เมื่อ TCP ที่ host B รับ segment นี้ขึ้นมาจะทราบทันทีว่าข้อมูลในเซกเมนต์นี้ เป็นของ application ใด

- Segment number: TCP จะใส่ตัวเลขลำดับของ segment เรียงกันไปเรื่อยๆ เพื่อแก้ปัญหาที่ segment มาถึงไม่เรียงลำดับที่ TCP ฝั่งรับ ก็ยังสามารถเรียงลำดับใหม่ให้ถูกต้องได้
- Checksum: TCP ฝั่งส่งจะใส่รหัสที่ถูกคำนวณรหัสเพื่อเป็นตัวแทนของข้อมูล เมื่อฝั่งรับได้รับ ก็จะทำงานคำนวณรหัสนี้ขึ้นมาจากข้อมูลที่ได้รับเพื่อเปรียบเทียบกับ รหัสที่ได้รับ ถ้าไม่ตรงกันแสดงว่าข้อมูลผิดพลาด

ต่อไป TCP จะส่งต่อแต่ละ segment ไปยัง IP พร้อมกับข้อมูลควบคุมเพื่อส่งต่อไปยัง host B segment เหล่านี้จะถูกจัดส่งข้ามเครือข่ายหรือหลายๆเครือข่ายโดย router ที่เชื่อมระหว่างเครือข่าย โดยอาศัยข้อมูลควบคุมที่มา IP จะทำการแปะข้อมูลส่วนหัวลงไป (แสดงในรูปที่ 2.4) ในแต่ละ segment รวมกันเรียกว่า IP datagram ตัวอย่างของข้อมูลควบคุมที่อยู่ใน IP header คือ ที่อยู่ของ host ปลายทาง (ในที่นี้คือ host B)

สุดท้าย แต่ละ IP datagram จะถูกส่งต่อไปยัง network access layer เพื่อจัดส่งข้ามเครือข่ายย่อยแรก network access layer ก็มี header ที่จะต้องใส่เข้าไปเพิ่มเช่นกัน เพื่อควบคุม packet หรือ frame จากนั้น packet จะถูกส่งข้ามเครือข่ายย่อยไปยัง router packet header ประกอบด้วยข้อมูลที่จำเป็นในการติดต่อกันภายในเครือข่ายย่อยเช่น

- Destination subnetwork address: เครือข่ายย่อยจะต้องรู้ว่าอุปกรณ์ใดจะเป็นผู้รับ packet
- Facilities requests: Network access protocol อาจร้องขอให้มีการกำหนดความสำคัญในการจัดส่ง



รูปที่ 2.7 Protocol Data Units ใน TCP/IP

ที่ router J packet header จะถูกแกะออกเพื่อที่จะนำเอา IP header มาตรวจสอบ ข้อมูลที่สำคัญอย่างแรกเลยคือ address ปลายทาง IP module ที่อยู่ใน router จะทำการส่ง datagram ไปยังเครือข่ายย่อยที่ 2 ไปยัง host B กรณีนี้ datagram จะถูกแปะด้วย network header เข้าไปใหม่ก่อนจัดส่ง เมื่อข้อมูลมาถึง host B กระบวนการที่ใช้จะเกิดขึ้นกลับกันกับที่ host A header ในทุกชั้น

จะถูกแกะออก ส่วนที่เป็นข้อมูลจริงๆจะถูกส่งไปที่ application layer ที่ระบุไว้ในตอนแรกอย่างถูกต้อง

2.2.2.5 โปรแกรมประยุกต์บน TCP/IP

จำนวนของ application ที่เป็นมาตรฐานถูกเสนอเพื่อใช้งานบน TCP มีจำนวนมาก ตัวอย่างดังนี้

Simple Mail Transfer Protocol (SMTP) ให้บริการ electronic mail กลไกของมันสามารถทำการส่งข้อความข้าม host ได้ รวมทั้งสามารถทำ mailing lists (กลุ่มของผู้ใช้เพื่อรับข่าวสาร), return receipts (แจ้งสถานะถึงผู้ส่งว่าได้รับแล้ว) และ forwarding (ส่งต่อไปยังผู้อื่น) SMTP protocol ไม่ได้กำหนดวิธีการสร้างข้อความ ดังนั้นโปรแกรมที่ใช้สำหรับสร้างข้อความ จำเป็นต้องใช้จากภายนอก หลังจากที่ข้อความถูกสร้าง SMTP จะรับข้อความนั้นแล้วใช้ TCP ในการส่งผ่านข้อความไปยัง SMTP module บนเครื่องอื่น จากนั้น SMTP module จะจัดเก็บข้อความนี้ไว้ในmailbox (ตู้จดหมาย) ของผู้ใช้

File Transfer Protocol (FTP) ถูกใช้ในการขนย้ายแฟ้มจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งภายใต้คำสั่งควบคุมจากผู้ใช้ สามารถใช้ขนย้ายได้ทั้งแฟ้มแบบ text และ binary และยังสามารถควบคุมสิทธิ์ในการเข้าถึงของผู้ใช้ได้ด้วย เมื่อผู้ใช้ต้องการจะขนย้ายแฟ้ม FTP จะสร้างการเชื่อมต่อแบบ TCP ไปยังคอมพิวเตอร์ปลายทางเพื่อใช้แลกเปลี่ยนข้อมูลควบคุม เช่น ชื่อผู้ใช้ และรหัสผ่านของผู้ใช้ ซึ่งสิทธิ์ในการเข้าถึงทำได้ถึงระดับของแฟ้มเลยทีเดียว ถ้าการขนย้ายแฟ้มได้รับการตอบตกลง จะมีการสร้างการเชื่อมต่อที่ 2 เพื่อใช้ขนย้ายข้อมูล แฟ้มจะถูกขนผ่านโดยเส้นทางที่ 2 โดยไม่มี overhead ของ header หรือรหัสควบคุมที่ application level เมื่อการขนส่งเสร็จสิ้น ช่องทางที่ใช้ควบคุมจะถูกใช้ส่งสัญญาณเพื่อบอกว่าเสร็จสิ้น

TELNET มีความสามารถในการ logon เข้ามาควบคุมเครื่องจากระยะไกล ซึ่งจะทำให้ผู้ใช้ที่อยู่หน้า terminal หรือ คอมพิวเตอร์ส่วนบุคคล logon ไปที่คอมพิวเตอร์ที่อยู่ไกลออกไป เสมือนกับผู้ใช้นั้นอยู่ควบคุมที่หน้าจอของคอมพิวเตอร์ที่อยู่ไกลนั้นจริงๆ protocol นี้ถูกออกแบบเพื่อรองรับ terminal แบบง่ายๆด้วย TELNET ถูกออกแบบแยกเป็น 2 ส่วนคือ User TELNET ทำหน้าที่แปลงตัวอักษรจากการที่ผู้ใช้กดแป้นพิมพ์ไปเป็นข้อมูลส่งออกเครือข่าย และในทางกลับกันส่วน Server TELNET ทำหน้าที่ติดต่อกับ application ทำให้ application นั้นทำงานเสมือนกับ terminal ที่ต่อที่เครื่องนั้นเอง การติดต่อระหว่าง User และ Server TELNET กระทำผ่านการเชื่อมต่อแบบ TCP

2.3 มาตรฐานบน Internet

หลายๆ protocol ที่เกี่ยวข้องกับชุด TCP/IP protocol จะต้องได้รับการรับรอง หรือกำลังอยู่ในขั้นตอนพิจารณา แต่ถ้าอย่างเป็นทางการแล้วละก็ Internet Architecture Board (IAB) จะทำหน้าที่ในการพิจารณาและประกาศมาตรฐาน โดยเผยแพร่ในรูปแบบ series (ชุด) ของเอกสารเรียกว่า Requests for Comments (RFCs)

เนื้อหาต่อไปนี้จะกล่าวถึงขั้นตอนมาตรฐานในการพัฒนาชุด TCP/IP protocol ให้ได้รับมาตรฐาน

2.3.1 Internet และมาตรฐานของ Internet

Internet คือ กลุ่มของเครือข่ายจำนวนมากที่เชื่อมต่อกันโดยใช้ชุด TCP/IP protocol Internet เริ่มจากการพัฒนาของ ARPANET และต่อๆมา โดย Defense Advanced Research Projects Agency (DARPA) เพื่อรองรับความต้องการทางทหารและราชการ

IAB คือคณะกรรมการประสานงานในการออกแบบและจัดการขอบเขตรวมถึงการดำเนินงานของ Internet ด้วย และมาตรฐานของ protocol สำหรับเครื่องต้นทางปลายทาง เพื่อความเข้ากันได้ในการเชื่อมต่อ IAB มีหน่วยงานย่อย 2 หน่วยงานคือ Internet Engineering Task Force (IETF) และ Internet Research Task Force (IRTF)

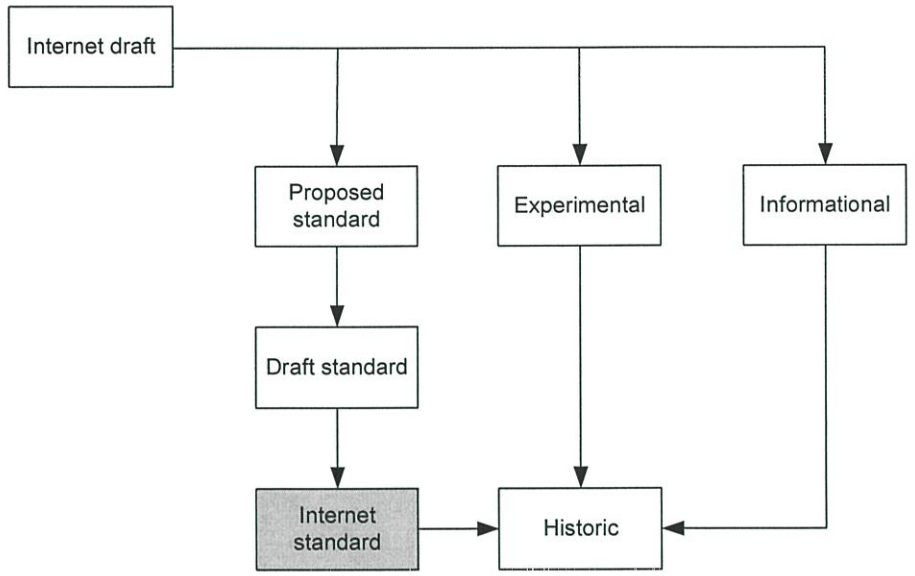
IETF จะดูแลงานด้านการนำเสนอ RFCs ซึ่งมาจากเอกสารงานวิจัยของกลุ่มนักวิจัยและพัฒนาใน Internet เอกสารในชุดนี้จะต้องเป็นเรื่องที่เกี่ยวข้องกับการสื่อสารคอมพิวเตอร์ เพื่อนำเข้าการประชุม เพื่อกำหนดออกมาเป็นมาตรฐาน การพิจารณาขั้นสุดท้ายว่า RFCs จะได้เป็น Internet standards หรือไม่นั้น จะอยู่ที่การพิจารณาของ IAB โดยพิจารณาจากข้อเสนอแนะของ IETF

เงื่อนไขที่จำเป็นในการพิจารณามีดังนี้

- มีเสถียรภาพและเป็นที่รู้จักดี
- มีความสามารถเชิงเทคนิค
- มีการพัฒนาขึ้นมาอย่างหลากหลาย, อิสระจากกัน และทำงานร่วมกันได้
- มีการสนับสนุนจากสังคม Internet
- เป็นประโยชน์ต่อบางส่วนหรือทุกส่วนของ Internet

2.3.2 ขั้นตอนเพื่อการเป็นมาตรฐาน

มาตรฐานของ Internet ในปัจจุบันถูกกำหนดใน RFC 2026 (The Internet Standards Process – Revision 3. October 1996) รูปที่ 2.8 แสดงขั้นตอนในการพิจารณาขอเป็นมาตรฐาน (Standards track)



รูปที่ 2.8 Internet Standards และ Nonstandards Track

ซึ่งขั้นตอนในช่วงหลังๆในการพิจารณาและทดสอบจะมีความเข้มข้นขึ้นเรื่อยๆ แต่ละขั้นตอน IETF จะเป็นผู้ที่ให้ข้อมูลสนับสนุนแก่ IAB ที่ทำหน้าที่พิจารณาอนุมัติ กรอบสีขาวในรูปแสดงถึงสถานะชั่วคราวที่ต้องใช้เวลาที่ระบุไว้อย่างน้อยรอที่จุดนั้นๆ ดังนี้ วิธีการที่เสนอต้องใช้เวลาอย่างน้อย 6 เดือนที่เป็นมาตรฐานที่เสนอ (Proposed standard) และอย่างน้อยอีก 4 เดือนที่เป็นมาตรฐานฉบับร่าง (Draft standard) เพื่อให้มีการวิจารณ์และข้อแนะนำอย่างเพียงพอ ส่วนกรอบสีเทาจะแสดงถึงสถานะอย่างถาวร

การเริ่มต้นสำหรับการนำเสนอผลงานฉบับร่าง (Internet draft) คือการทำเอกสารเบื้องต้นเพื่อการพิจารณาอย่างคร่าวๆในตอนแรก เอกสารนี้จะถูกนำไปเผยแพร่ใน Internet เพื่อวิเคราะห์แนะนำและแก้ไข Internet draft อาจจะต้องมีการแก้ไขใหม่ (Revision) อีกหลายๆครั้งก่อนที่จะได้เป็น RFC

ข้อกำหนดใน Standards track สำหรับการจะได้ปรับจาก Internet draft มาเป็น Proposed standard จะต้องมีเสถียรภาพ ทำการแก้ปัญหาที่มีการแนะนำไปแล้ว น่าจะเป็นที่รู้จักดี มีการตอบรับและวิจารณ์จากสังคม และสังคมให้ความสนใจถึงผลประโยชน์ที่ได้รับ

สำหรับ Proposed standard จะได้เลื่อนขั้นเป็น Draft standard จะต้องพัฒนาขึ้นมาอย่างน้อย 2 รูปแบบ (code) ที่แตกต่างกัน อิสระจากกัน แต่ต้องทำงานร่วมกันได้ และจะต้องผ่านการใช้งานเป็นอย่างดี Draft standard จะถูกพิจารณาให้เป็นข้อกำหนดสุดท้าย (Final specification) การแก้ไขข้อกำหนดจะเป็นเพียงการแก้ปัญหาใหม่ที่เกิดขึ้น

หลังจากผ่านขั้นตอนข้างต้น ถ้า Draft standard มีคุณลักษณะตรงตามข้อกำหนดทั้งหมด จะถูกปรับสถานะเป็น Internet standard Internet standard จะมีรายละเอียดเชิงเทคนิคที่สมบูรณ์มาก

และต้องมีประโยชน์อย่างมากต่อสังคม Internet ที่จุดนี้วิธีการนั้นจะได้รับ STD number แทนที่ RFC number สุดท้ายถ้ามาตรฐานนี้ถูกยกเลิกจะมีการบันทึกไว้เป็นหลักฐาน

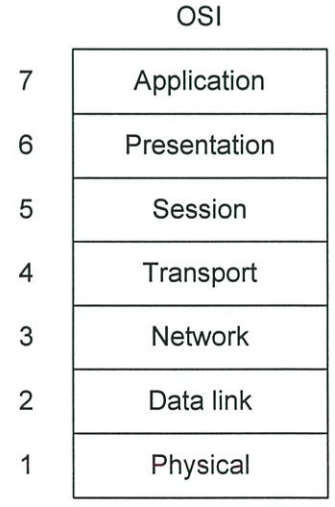
ยังมีขั้นตอนพิเศษที่ไม่ได้อยู่ในรูปแบบข้างต้นของ Standards track คือ protocol หรือข้อกำหนดอื่นๆ ที่ยังไม่ต้องการเป็นมาตรฐาน อาจจะเผยแพร่ในรูปของ RFC แต่ต่อไปอาจจะนำกลับมาเสนอใหม่เพื่อเข้าสู่ Standards track

RFC เป็นข้อมูลที่ถูกนำมาเผยแพร่เพื่อให้ข้อมูลที่เป็นประโยชน์ต่อสังคม Internet

2.4 โครงสร้างของ OSI reference model

OSI (Open Systems Interconnection) เป็น model อ้างอิงที่ถูกพัฒนาขึ้นมาโดย ISO (International Organization for Standardization) เพื่อเป็นกรอบในการพัฒนา protocol มาตรฐานต่อๆไป OSI ประกอบไปด้วย 7 layer ดังนี้

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

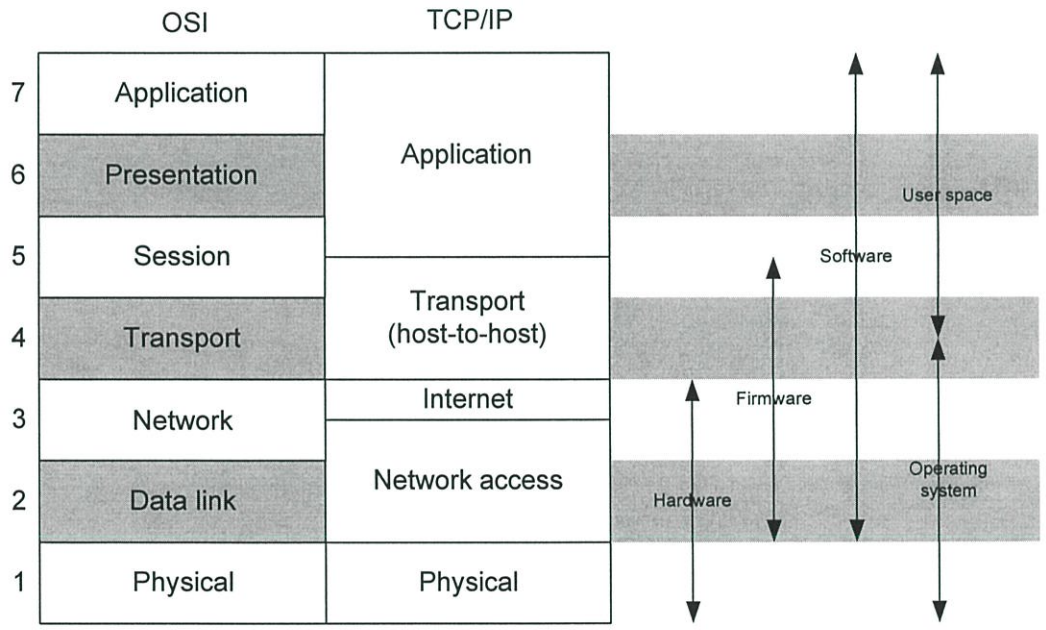


รูปที่ 2.9 OSI Layer

รูปที่ 2.9 แสดงโครงสร้างแบบชั้น (layer) ของ OSI model จุดมุ่งหมายของ OSI model คือให้ protocol ถูกพัฒนาเพื่อให้มีการแบ่งการทำงานของแต่ละ layer ผู้ที่ออกแบบ OSI คาดว่าด้วย

โครงสร้างแบบนี้จะเป็นต้นแบบให้มีการพัฒนา protocol ตามโครงสร้างนี้ ซึ่งส่งผลดีอย่างมากต่อการสื่อสารคอมพิวเตอร์ แม้กระทั่งการเข้าไปสู่แนวทางพัฒนา protocol เฉพาะของแต่ละผลิตภัณฑ์ด้วย และยังจะมีผลต่อการพัฒนา TCP/IP แต่ทั้งหมดนี้ไม่ได้เกิดขึ้นตามที่ผู้ออกแบบ OSI คาดหวัง ถึงแม้ว่าหลายๆ protocol ที่ดีได้ถูกพัฒนาตามคำอธิบายของ OSI แต่ไม่ได้พัฒนาขึ้นมาทั้ง 7 layer ตามที่ได้วางไว้ ตัวอย่างเช่น โครงสร้าง TCP/IP ที่ใช้กันแพร่หลาย ด้วยเหตุผลคือ TCP/IP ได้ถูกพัฒนาจนสมบูรณ์และมีการใช้งานอย่างแพร่หลายมากแล้ว ในช่วงเวลาที่ OSI model กำลังถูกพัฒนาขึ้นมา เมื่อธุรกิจต้องการเชื่อมต่อข้ามเครือข่าย จึงมีเฉพาะ TCP/IP เท่านั้นที่ตรงตามความต้องการและใช้งานได้จริง เหตุผลอื่นคือ OSI model ถูกออกแบบไม่ให้ซับซ้อนจึงมี layer มากถึง 7 layer ในขณะที่ TCP/IP ทำงานด้วย layer ที่น้อยกว่า

รูปที่ 2.10 แสดงถึง layer ของโครงสร้าง TCP/IP กับ OSI และแสดงหน้าที่อย่างคร่าวๆของทั้งสองโครงสร้าง ในรูปยังแนะนำถึงการพัฒนาแต่ละ layer ว่าควรรับผิดชอบโดยส่วนไหน



รูปที่ 2.10 การเปรียบเทียบระหว่าง OSI Reference Model กับ TCP/IP

2.5 Internetworking (การเชื่อมต่อข้ามเครือข่าย)

ในหลายๆกรณี เรามอง LAN และ WAN ไม่แตกต่างกัน องค์กรอาจจะมี LAN หลายๆ แบบโดยอยู่ที่ความเหมาะสมของแต่ละสถานที่ สถานการณ์ เพื่อสนับสนุนการเชื่อมต่อ หรืออาจจะเป็น LAN แบบเดียวกันหลายๆ LAN เพื่อสนับสนุนความต้องการในด้านประสิทธิภาพและความปลอดภัย และองค์กรอาจจะมี LAN อยู่หลายๆที่แต่ทำการเชื่อมต่อเข้าหากันผ่าน WAN แบบรวมศูนย์ในการแลกเปลี่ยนข้อมูล

การเชื่อมต่อกลุ่มของเครือข่ายจากมุมมองของผู้ใช้อาจจะมองง่ายๆว่าเป็นเครือข่ายที่ใหญ่ขึ้น แต่ถ้าแต่ละส่วนของเครือข่ายมีการกำหนดเลขที่เฉพาะลงไป และมีกลไกพิเศษที่ถูกใช้สำหรับการสื่อสารข้ามหลายๆเครือข่าย โดยทั้งหมดนี้เรานิยมเรียกว่า Internet และแต่ละส่วนย่อยของเครือข่ายเราเรียกว่า subnetwork Internet ถูกก่อตั้งจากการเริ่มต้นอย่างค่อยเป็นค่อยไปจากเครือข่ายแบบ packet-switching ของนักวิจัย และถูกใช้เป็นพื้นฐานในการพัฒนาของเทคโนโลยี internetworking และยังนำมาใช้เป็นเครือข่ายเฉพาะภายในองค์กรเรียกว่า intranet

แต่ละส่วนของ subnetwork ใน Internet สนับสนุนการสื่อสารในกลุ่มของอุปกรณ์ที่ต่อเข้ากับ subnetwork นั้น อุปกรณ์เหล่านี้เราเรียกว่า end systems (ESs) จากนั้น subnetwork ถูกต่อเข้ากับอุปกรณ์ที่อ้างจากเอกสารของ ISO ว่าเป็น intermediate systems (ISs) ISs จะเป็นตัวกลางในการเชื่อมต่อเครือข่ายและการจัดหาเส้นทางในการส่งข้อมูลระหว่างอุปกรณ์ที่ต่ออยู่คนละ subnetwork ใน Internet

ISs มีอยู่ 2 ชนิดที่สำคัญคือ bridge และ router ข้อแตกต่างระหว่างอุปกรณ์ทั้ง 2 คือ ชนิดของ protocol ที่ถูกใช้สำหรับ Internetworking bridge จะเป็นอุปกรณ์ที่ทำงานที่ layer 2 ของโครงสร้าง 7 layer ของ OSI และทำหน้าที่ส่งผ่าน frame ระหว่าง network ที่เป็นแบบเดียวกัน ส่วน router ทำงานที่ layer 3 ของ OSI และทำหน้าที่ส่งผ่าน packet ระหว่าง network ที่แตกต่างกันได้ด้วย เนื่องจาก router เป็นอุปกรณ์ที่มีบทบาทเป็นอย่างมากในการเชื่อมต่อเครือข่ายใน Internet จึงนำมาขยายความเพิ่มเติมในหัวข้อถัดไป

2.5.1 Router

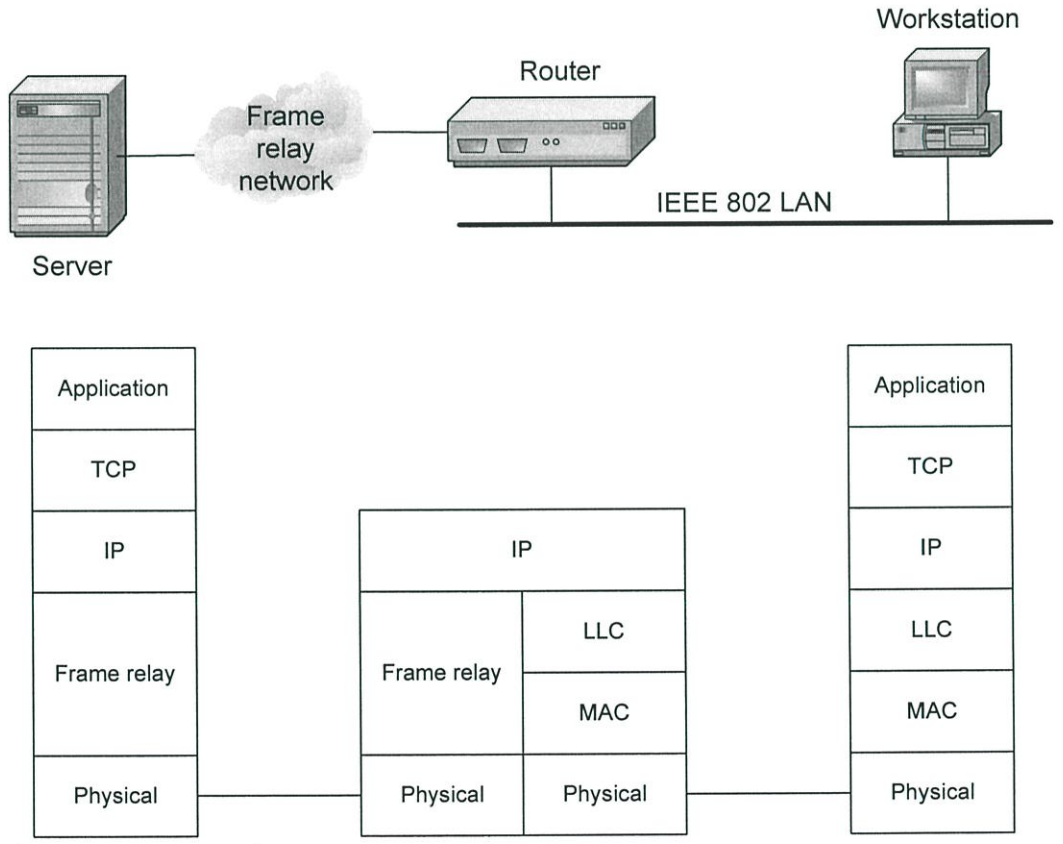
Router ถูกนำมาใช้ในการเชื่อมต่อของการสื่อสารข้ามเครือข่ายที่หลากหลายประเภท หน้าที่พื้นฐานของ router มีดังนี้

1. สร้างการเชื่อมต่อระหว่างเครือข่าย
2. จัดหาเส้นทางและจัดส่งข้อมูลข้ามเครือข่าย
3. ทำหน้าที่ในข้อที่ 1 และ 2 โดยไม่ต้องการการแก้ไขโครงสร้างของเครือข่ายย่อยที่เชื่อมต่อด้วย

จากข้อที่ 3 นั้นหมายความว่า router จะต้องอำนวยความสะดวกให้กับเครือข่ายที่ต่างประเภทกัน ด้วยคุณสมบัติดังต่อไปนี้

- Addressing schemes (วิธีการอ้างที่อยู่): เครือข่ายต่างประเภทกันอาจมีวิธีการกำหนด address ให้กับอุปกรณ์ต่างกัน เช่น IEEE 802 LAN ใช้ address ขนาด 16-bit และ 48-bit สำหรับอ้างถึงอุปกรณ์แต่ละตัว X.25 ที่เป็นเครือข่ายแบบ packet-switching ใช้ตัวเลขฐานสิบขนาด 12 ตัวเลข (เข้ารหัสแบบ 4-bit ต่อ 1 ตัวเลข รวมแล้วใช้ 48-bit) ทำให้การเชื่อมเครือข่ายขนาดใหญ่ต้องมีการกำหนดการอ้างอิง address ในรูปของ directory

- Maximum packet sizes (ขนาดสูงสุดของ packet): packet จากเครือข่ายหนึ่งอาจถูกแตกออกให้เล็กลงเพื่อส่งข้ามไปยังเครือข่ายอื่นๆ กระบวนการนี้เรียกว่า segmentation ตัวอย่างเช่น Ethernet กำหนดให้ขนาดสูงสุดของ packet ไม่เกิน 1500 bytes แต่ขนาดสูงสุดไม่เกิน 1000 bytes ถูกใช้ใน X.25 ดังนั้น packet ที่ส่งจาก Ethernet เมื่อไปถึง router จะถูกแตกออกเป็น 2 ส่วนเป็นอย่างน้อย เพื่อส่งผ่านไปยังเครือข่าย X.25
- Interfaces: ทั้ง hardware และsoftware มีการติดต่อกับเครือข่ายหลายประเภท ดังนั้น router จะต้องไม่ขึ้นกับชนิดของเครือข่าย
- Reliability: บริการของเครือข่ายบางชนิด เป็นแบบ end-to-end virtual circuit ที่มีเสถียรภาพสูง บางชนิดไม่ค่อยมีเสถียรภาพ การจัดการของ router จะต้องไม่ขึ้นกับเสถียรภาพของเครือข่าย



รูปที่ 2.11 ตัวอย่างการเชื่อมต่อของ TCP/IP

2.5.2 ตัวอย่างของ Internetworking

รูปที่ 2.11 แสดงการเชื่อมต่อเพื่อแสดงถึงการสื่อสารของ protocol ในกรณีนี้ server ต่ออยู่กับ frame relay WAN และ workstation ต่ออยู่กับ IEEE 802 LAN และมี router ทำหน้าที่เชื่อม 2

เครือข่ายเข้าด้วยกัน router ยังช่วยให้การเชื่อมต่อระหว่าง server กับ workstation เกิดขึ้นโดยทั้ง 2 ไม่ต้องสนใจรายละเอียดของเครือข่ายที่ติดต่อเป็นคนละชนิดกัน

2.6 สรุป

จากความรู้พื้นฐาน TCP/IP ในบทนี้เป็นการเตรียมความพร้อมให้ผู้อ่านได้เข้าใจถึงการทำงานแบบภาพรวมของทั้งชุด protocol ซึ่งเป็นประโยชน์ในการศึกษาเรื่องต่อๆไปเพื่อให้ทราบว่าเรื่องที่ย่อยๆลงไปเป็นส่วนประกอบย่อยตรงส่วนไหนของชุด TCP/IP protocol ในบทต่อไปเราจะพูดถึง TCP อย่างเดียวเป็นหลัก เพราะเป็น protocol ที่มีความซับซ้อนและมีผลต่อประสิทธิภาพในการส่งข้อมูลอย่างมาก

บทที่ 3

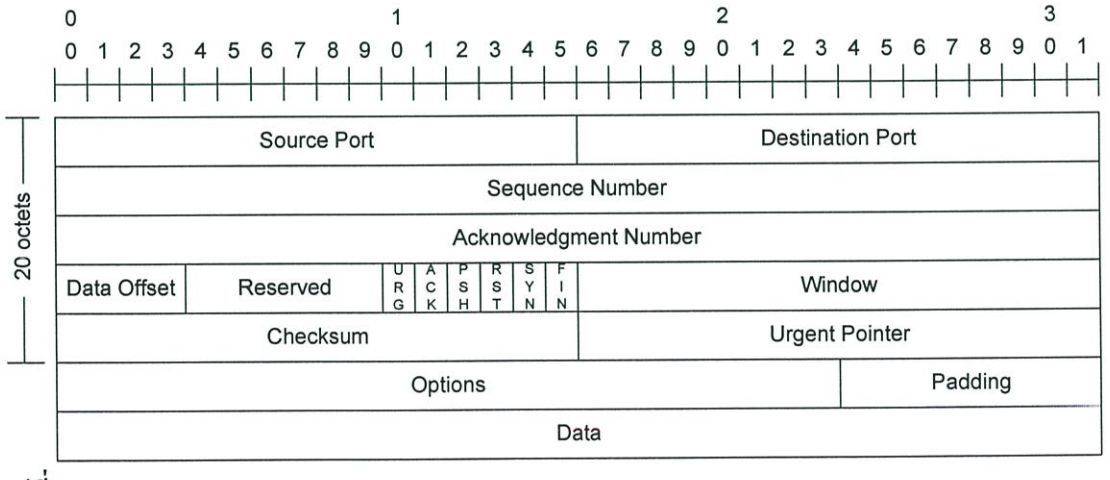
Transmission Control Protocol (TCP)

เพื่อประสิทธิภาพของการเชื่อมต่อเครือข่าย การออกแบบและการพัฒนาของ transport protocol เป็นสิ่งที่สำคัญมาก transport protocol ให้บริการ interface ระหว่าง application และอำนวยความสะดวกในการเชื่อมต่อ TCP เป็น transport protocol แบบ connection-oriented จะแบ่งข้อมูลที่ส่งมาจาก application ออกเป็นส่วนย่อยๆ จากนั้นจะใช้กลไกในการจัดส่ง (transmission) และการจัดส่งซ้ำ (retransmission) ที่ส่งผลกับความแออัดของช่องทางข้อมูล (congestion) ในเครือข่าย

บทนี้เราจะพูดถึง TCP ที่เป็น transport protocol ที่ถูกใช้อย่างแพร่หลาย โดยมีการเรียกใช้งานโดย application ส่วนใหญ่บนชุด TCP/IP protocol เราเริ่มจากภาพรวมของ TCP ต่อมาจะอธิบายถึงหน้าที่ของ flow and error control ของ TCP จากนั้นจะอธิบายเรื่อง congestion control ที่มีความซับซ้อนใน TCP

3.1 TCP header format (รูปแบบส่วนหัวของ TCP)

TCP ใช้ protocol data unit (TCP header กับ ข้อมูลจำนวนหนึ่ง) เพียงแบบเดียว เรียกว่า TCP segment ดังแสดงในรูป 3.1 เนื่องจาก header แบบเดียวต้องสามารถรองรับกลไกทั้งหมดของ protocol ส่งผลให้ TCP header มีขนาดค่อนข้างใหญ่ โดยขนาดเล็กสุดคือ 20 byte



รูปที่ 3.1 Header format ของ TCP

ต่อไปนี้คือรายละเอียดของแต่ละ field (เนื้อที่ใน header ที่กำหนดไว้สำหรับเก็บข้อมูลประเภทหนึ่ง) ใน TCP header

- Source port (16 bit): เลขที่ของ TCP application บน host ต้นทาง
- Destination port (16 bit): เลขที่ของ TCP application บน host ปลายทาง
- Sequence number (32 bit): เลขแสดงลำดับของ byte แรกของชุดข้อมูลภายใน segment นั้น ยกเว้นถ้า SYN flag ถูก set ถ้า SYN ถูก set เลขลำดับใน field นี้จะหมายถึง initial sequence number (ISN) และข้อมูล byte แรกจะเป็น ISN+1
- Acknowledgement number (32 bit): ใช้แสดงลำดับของข้อมูล byte ถัดไปที่คาดว่าจะได้รับ
- Data offset (4 bit): จำนวนของ word ขนาด 32-bit ใน header
- Reserved (6 bit): เผื่อไว้สำหรับอนาคต
- Flags (6 bit):

  URG: แสดงว่ามีการใช้ Urgent pointer field

  ACK: แสดงว่ามีการใช้ Acknowledgement field

  PSH: push function

  RST: reset (ยกเลิก) การสื่อสาร

  SYN: แสดงว่าเป็นการ synchronization (การปรับให้ตรงกัน) ของลำดับข้อมูล

  FIN: หมดข้อมูลที่จะส่ง และจบการสื่อสาร
- Window (16 bit): ใช้ในการจัดสรรพื้นที่ในกลไก flow control จะแสดงถึงจำนวนข้อมูลที่ต้องการรับ โดยนับข้อมูลตัวแรกจากลำดับใน acknowledgement field
- Checksum (16 bit): รหัสตรวจสอบข้อมูลผิดพลาด
- Urgent pointer (16 bit): ชี้ไปยัง byte สุดท้ายในชุดของ urgent data เพื่อให้รับรู้ว่ามี urgent data เข้ามามากเท่าไร
- Options (มีขนาดไม่แน่นอน): เป็นข้อมูลพิเศษเพิ่มเติม

field จำนวนมากใน TCP header ถูกออกแบบเผื่อเอาไว้เป็นอย่างดี sequence number และ acknowledgement number ถูกออกแบบให้แสดงถึงจำนวนข้อมูลเป็น byte มากกว่าที่จะเป็นการแสดงเฉพาะแค่ใน segment นั้น ตัวอย่างถ้า segment ประกอบด้วยลำดับที่ 1000 และมีข้อมูลจำนวน 600 byte sequence number จะแสดงถึงลำดับของ byte แรกในกลุ่มข้อมูล segment ถัดไปในทาง logical จะประกอบด้วย sequence number 1600 แสดงว่า TCP มองชุดข้อมูลที่จัดส่งแบบ logical TCP จะรับชุดของข้อมูลจากผู้ใช้และนำมารวมกันเป็น segment จนพอดีและใส่ตัวเลขอ้างอิงของแต่ละ segment

PUSH และ URGENT flag ถูกใช้สนับสนุนบริการ 2 อย่างดังนี้

- Data stream push: โดยปกติ TCP จะรวบรวมข้อมูลให้ได้มากระดับหนึ่งถึงจะทำการสร้าง segment ในการสื่อสาร แต่ผู้ใช้ TCP สามารถร้องขอให้ TCP ทำการส่งข้อมูล

เท่าที่มีรวมทั้งข้อมูลที่มาพร้อมคำสั่ง PUSH ออกไปโดยไม่ต้องรอให้ครบจำนวนตาม segment ปกติ และในฝั่งรับ TCP จะส่งข้อมูลให้ผู้ใช้ในรูปแบบเดียวกัน ผู้ใช้มักจะใช้คุณสมบัตินี้เหมือนกับเป็น logical break ของข้อมูล

- Urgent data signaling: เพื่อสนับสนุนการแจ้งข้อมูลให้กับ TCP ปลายทาง ถ้ามีการบอกว่า "Urgent" แสดงว่าข้อมูลชุดนี้ที่ส่งมาเป็นแบบด่วน แต่จะขึ้นอยู่กับผู้ใช้ปลายทางว่าจะตอบสนองต่อข้อมูลนี้อย่างไร

ผู้ใช้ TCP (หมายถึง application เช่น FTP) ร้องขอให้ TCP ส่งข้อมูลด้วยคำสั่ง SEND TCP จะนำข้อมูลที่จะส่งมาเก็บไว้ที่ send buffer (หน่วยความจำที่พักข้อมูลก่อนการจัดส่ง) ถ้า PUSH flag ถูกตั้งไว้ ข้อมูลทั้งหมดใน send buffer รวมทั้งข้อมูลที่เพิ่งรับเข้ามาจะถูกจัดส่งทันที (อาจส่งออกมามากกว่า 1 segment) โดย segment สุดท้ายถูกระบุด้วย PUSH flag ถ้า PUSH flag ไม่ได้ตั้งไว้ TCP จะเก็บข้อมูลนั้นไว้ใน send buffer และจะจัดส่งทีละ 1 segment หรือมากกว่าเมื่อต้องการ ตัวอย่างเช่น TCP จะรอจนกระทั่งข้อมูลจำนวนมากเข้ามาสะสมแล้วจึงค่อย เพราะ segment ที่มีขนาดใหญ่กว่าจะมีประสิทธิภาพในการส่งข้อมูลมากกว่า (overhead ต่ำ) ผู้ใช้อาจจะตั้ง URGENT flag กับข้อมูลมาพร้อมกับคำสั่ง SEND ในกรณีนี้ segment ที่ตอบรับจะถูกตั้ง URGENT flag กลับมาด้วย

ข้อมูลใน segment จะเดินทางข้ามเครือข่ายมายัง TCP ปลายทางและถูกจัดเก็บไว้ใน delivery buffer (หน่วยความจำที่พักข้อมูลก่อนที่ TCP รับเข้ามาก่อนส่งไปยัง application) ที่สัมพันธ์กับการสื่อสารนั้น ถ้าข้อมูลที่เข้ามาถูกตั้ง PUSH flag ไว้ ข้อมูลทั้งหมดใน delivery buffer จะถูกส่งไปยังผู้ใช้ปลายทางทันทีด้วยคำสั่ง RECEIVE ถ้าข้อมูลไม่ถูกตั้ง PUSH flag TCP จะจัดส่งเมื่อสถานการณ์เหมาะสม ตัวอย่างเช่น TCP อาจจะรอจนกระทั่งข้อมูลเข้ามาสะสมจนมีขนาดมากพอ เพื่อลดการขัดจังหวะระบบบ่อยๆ ถ้าข้อมูลที่เข้ามาถูกตั้ง URGENT flag ผู้ใช้จะได้รับสัญญาณว่าข้อมูลนั้นเป็นแบบ URGENT

Checksum field เป็นการนำทั้ง segment รวมกับ pseudo header (ข้อมูลควบคุมพิเศษนอกเหนือ TCP header) ที่เวลาของการคำนวณ (ที่ทั้งฝั่งส่งและฝั่งรับ) pseudo header คือ ข้อมูลจาก IP header ดังนี้ source IP, destination IP, protocol รวมกับ ความยาวของ segment การที่ต้องรวม pseudo header นั้นเพื่อป้องกันข้อผิดพลาดให้กับ TCP เองจากการจัดส่งของ IP ดังนั้นถ้า IP จัดส่ง segment ไปยัง hostที่ไม่ถูกต้อง แม้จะไม่มี bit ที่ผิดพลาดใน segment เลย TCP ฝั่งรับก็ยังสามารถตรวจจับข้อผิดพลาดที่เกิดขึ้นได้ TCP checksum ถูกคำนวณด้วยวิธี 16-bit one's complement ของ 16-bit word ใน pseudo header, TCP header และ TCP body เพื่อความสะดวกในการคำนวณ checksum field ถูกกำหนดให้ตอนเริ่มต้นมีค่าเป็นศูนย์

RFC 793 ได้กำหนด option ไว้เพียงแบบเดียว คือ maximum segment size 16-bit option นี้อาจจะถูกใช้ใน segment ที่ทำการร้องขอเริ่มต้นการสื่อสาร ขนาด segment สูงสุดมีหน่วยเป็น byte ตั้งแต่มีการนำเสนอ RFC มี option อีก 2 แบบที่เป็นที่ยอมรับอย่างกว้างขวางคือ

- Window scale factor: โดยปกติ window field ใน TCP header มีไว้บอกขนาดที่เตรียมไว้รับมูล มีหน่วยเป็น byte แต่เมื่อมีการใช้ window scale option ค่าของ window field จะถูกคูณด้วย 2^F เมื่อ F เป็นค่าของ window scale option ค่าสูงสุดของ F ที่ TCP ยอมรับคือ 14 option นี้ถูกใช้เฉพาะใน segment ที่ร้องขอเริ่มต้นการสื่อสาร
- Timestamp (การบันทึกเวลา): option นี้จริงๆแล้วถูกใช้ใน data segment และกำหนดไว้ 2 optional field TCP อาจบันทึก Timestamp Value field ในทุกๆ segment ที่ทำการจัดส่ง เมื่อ segment นั้นถูกตอบรับจากอีกฝั่ง จะมีการตอบกลับด้วย Timestamp Echo Reply field ด้วยเวลาเดียวกับที่ส่งมาใน Timestamp Value field ใน segment ที่ตอบรับ option นี้มีประโยชน์ในการใช้ monitor (เฝ้าติดตามตรวจสอบ) ค่า round trip time (เวลาเดินทางไป-กลับ) ของข้อมูล

3.2 TCP flow control

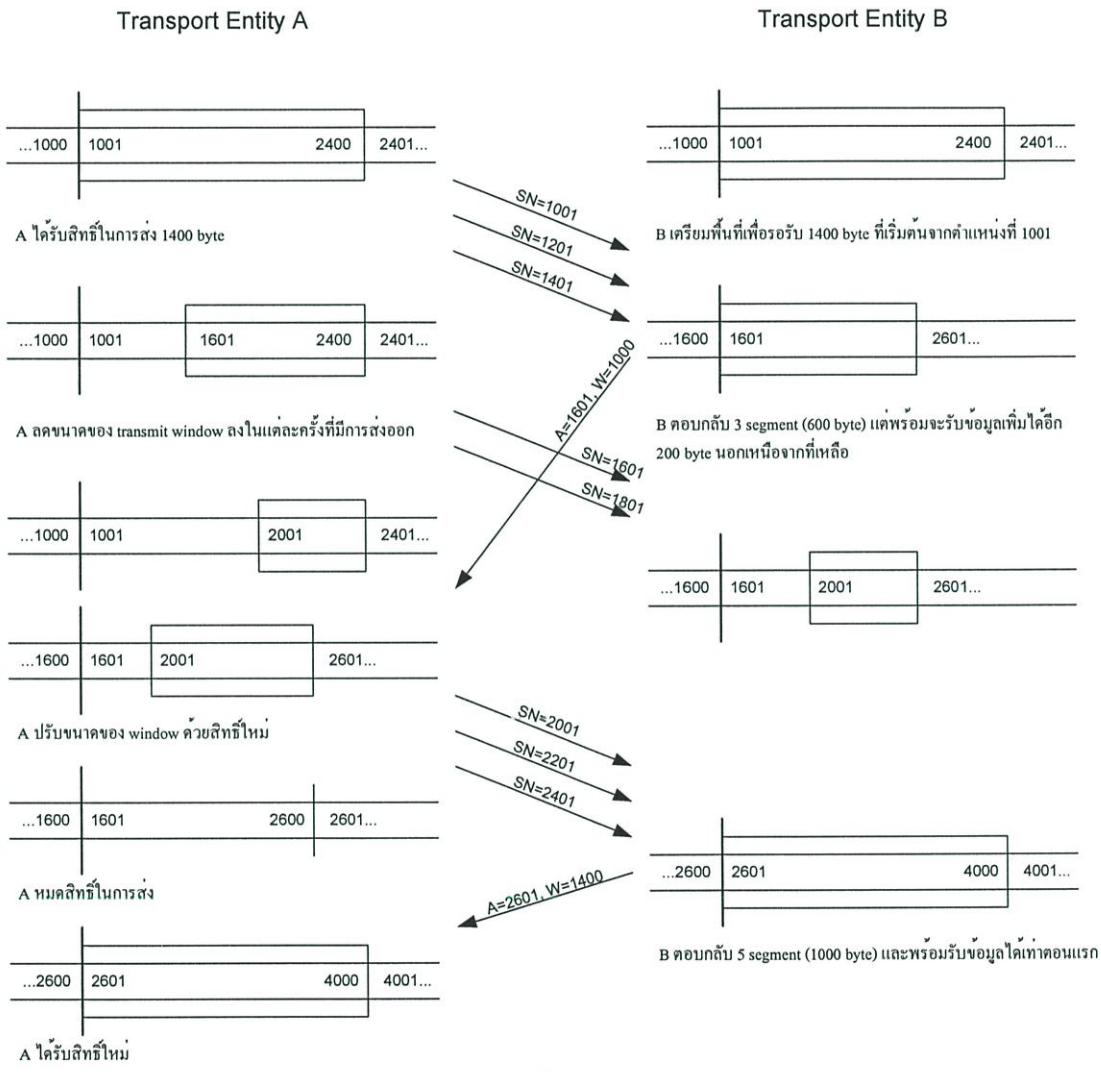
protocol ส่วนใหญ่มักจะต้องมีการควบคุม flow control TCP ใช้กลไกของ sliding window ซึ่งมีความแตกต่างจากกลไกที่ใช้ใน protocol อื่นๆ เช่น LLC, HDLC และ X.25 ตรงที่ว่า TCP ไม่ใช้การตอบรับของข้อมูลที่ได้รับแล้ว มาเป็นตัวกำหนดสิทธิ์ในการส่งข้อมูลใหม่

แต่กลไก flow control ของ TCP เรียกว่า credit allocation scheme ด้วยวิธีการนี้ ข้อมูลแต่ละส่วนที่ถูกส่งจะถูกพิจารณาด้วย sequence number เมื่อ TCP ส่ง segment TCP จะใส่ sequence number ของ byte แรกในชุดข้อมูลนั้นเข้าไปด้วย TCP ฝั่งรับจะตอบรับ segment ที่เข้ามาด้วยข้อความในรูปแบบนี้ (A = i, W = j) อธิบายได้ดังนี้

- ถ้าทุกๆ byte ถึง sequence number $i - 1$ ถูกตอบรับแล้ว byte ที่คาดว่าจะได้รับถัดไปคือ sequence number i
- ได้สิทธิ์ในการส่ง window (W) ของข้อมูลเพิ่มเติมอีก j byte ดังนั้นข้อมูลอีก j byte อ้างถึงได้จาก sequence number ที่ i ถึง $i + j - 1$

รูปที่ 3.2 แสดงถึงกลไกแบบ sliding window ของ TCP เพื่อให้ดูง่าย เราแสดงการไหลของข้อมูลในแบบทิศทางเดียวและสมมติให้ มีการส่งข้อมูลทีละ 200 byte ในแต่ละ segment เริ่มจากขั้นตอนเริ่มต้นเชื่อมต่อ sequence number ของทั้งฝั่งส่งและรับจะถูกกระบวนการปรับให้ตรงกัน และ A จะได้รับอนุญาตให้ส่งข้อมูลได้ 1400 byte เริ่มจาก byte ที่ 1001 หลังจากทำการส่งไป 600 byte ภายใน 3 segment A จะลดขนาดของ window ลงเหลือ 800 byte (เหลือสิทธิ์ในการส่ง 1601 ถึง 2400) หลังจากที่ B ได้รับ segment ทั้ง 3 แล้ว B ตอบรับข้อมูลทั้งหมดด้วย 1601 และให้สิทธิ์

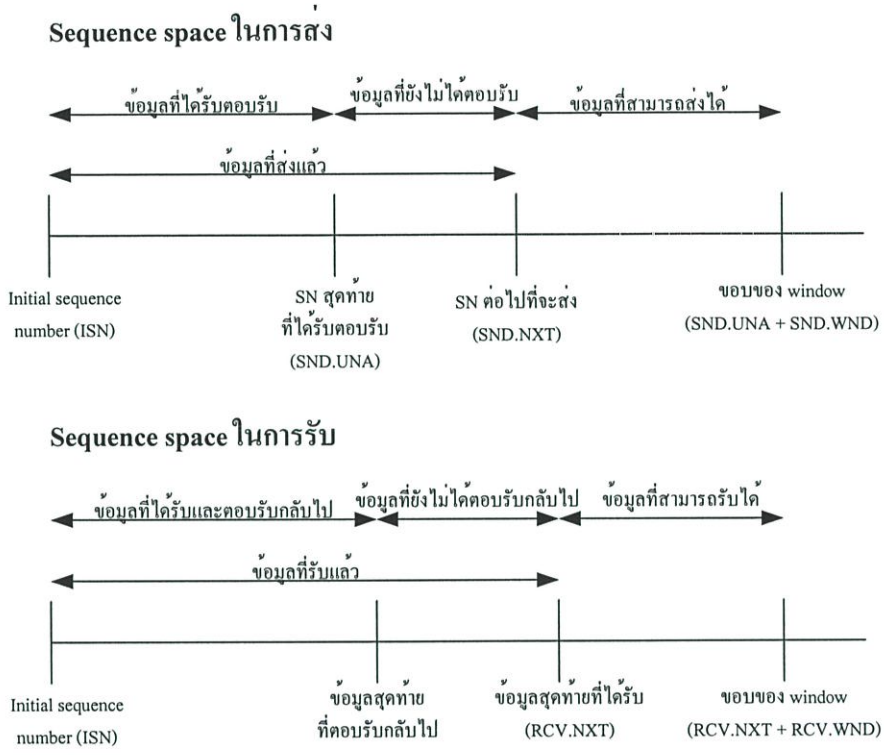
เป็น 1000 byte หมายความว่า A สามารถส่งข้อมูลลำดับที่ 1601 ถึง 2600 (5 segment) ช่วงเวลาที่ข้อความของ B มาถึง A A กำลังส่ง 2 segment ประกอบด้วย byte ลำดับที่ 1601 ถึง 2000 (ซึ่งยังเป็นการใช้สิทธิ์จากตอนเริ่มต้น) จากนั้น A ยังเหลือสิทธิ์อยู่ 400 byte (2 segment) การแลกเปลี่ยนข้อมูลกันแต่ละครั้ง A จะเลื่อนส่วนหางของ window ทุกครั้งที่ส่ง แต่จะเลื่อนส่วนหัวของ window ก็ต่อเมื่อได้รับสิทธิ์เพิ่ม



รูปที่ 3.2 ตัวอย่างของกลไกในการจัดสรรสิทธิ์ในการส่งของ TCP

รูปที่ 3.3 แสดงให้เห็นถึงการทำงานของกลไกนี้จากทั้งฝั่งส่งและรับ โดยปกติทั้ง 2 ฝั่งจะต้องมีกลไกนี้ทั้งคู่ เพราะว่าข้อมูลมีการแลกเปลี่ยนกันแบบ 2 ทิศทาง (full duplex) กลไกในการกำหนดสิทธิ์นี้มีความยืดหยุ่นสูงมาก ดังตัวอย่างนี้ ข้อความสุดท้ายที่ส่งมาจาก B คือ (A = i, W = j) และข้อมูล byte สุดท้ายที่ B ได้รับคือ $i - 1$ ดังนั้น

- เพื่อเพิ่มสิทธิ์จำนวน k ($k > j$) เมื่อไม่มีข้อมูลเพิ่มเติมเข้ามา B จะใช้ข้อความว่า (A = i, W = k)
- เพื่อตอบรับ segment ที่เข้ามาที่ประกอบด้วยข้อมูลขนาด m byte ($m < j$) โดยไม่มีการเพิ่มสิทธิ์ B จะใช้ข้อความว่า (A = i + m, W = j - m)



รูปที่ 3.3 การทำงานของ sequence space ของ flow control

ด้านรับไม่จำเป็นจะต้องตอบรับทุก segment อย่างทันที แต่จะรอให้มีการสะสมจำนวนหนึ่งก่อน ด้านรับจำเป็นจะต้องมีการกำหนดจำนวนของข้อมูลที่อนุญาตให้ฝั่งส่ง เพื่อให้แน่ใจว่ามีพื้นที่ buffer เพียงพอต่อการรองรับ segment ใหม่ที่จะเข้ามา เช่นในรูปที่ 3.2 ข้อความที่แจ้งสิทธิ์แรกจาก B กำหนดไว้ที่ 1000 byte และข้อความที่สองกำหนดไว้ที่ 1400 byte

ด้วย flow control แบบเก่าๆ อาจจะเป็นการลด throughput (อัตราการส่งผ่านข้อมูล) ของการสื่อสารกันในเครือข่ายที่มีค่าการหน่วงมากๆ แต่ด้วยวิธีนี้ ฝั่งรับสามารถเพิ่ม throughput ได้โดยการปรับขนาดของสิทธิ์ในการส่ง เช่น ถ้า buffer ด้านรับเต็ม แต่คาดการณ์ว่าจะได้พื้นที่คืนมา 1000 byte ภายใน round-trip propagation time เป็นไปได้ที่จะให้สิทธิ์ขนาด 1000 byte ไปทันที ถ้าด้านรับมีพื้นที่ 1000 byte ได้ทันเวลาก่อนที่ข้อมูลจากฝั่งส่งจะมาถึง จะเป็นการเพิ่ม throughput แต่ถ้าข้อมูลจากฝั่งส่งมาถึงก่อนแล้วไม่มีที่เก็บบาง segment จะถูกทิ้งไป ทำให้เกิดการส่งซ้ำใหม่ สรุปได้ว่าการควบคุม flow control ที่ดีทำให้ protocol มีความซับซ้อน

3.2.1 ผลของขนาดของ window ต่อประสิทธิภาพ

เราสามารถกำหนด throughput สูงสุดของ TCP connection throughput ขึ้นอยู่กับ window size, propagation delay และ data rate

W = TCP window size (มีขนาดเป็น byte)

R = Data rate (bps) ที่ TCP ฝั่งต้นทางขึ้นอยู่กับแต่ละการเชื่อมต่อ

D = Propagation delay (วินาที) ระหว่าง TCP ต้นทางและปลายทางผ่านการเชื่อมต่อ

เพื่อความง่าย เราจะตัด overhead bit ใน TCP segment ออกไป สมมติว่า TCP ต้นทางเริ่มต้นส่งข้อมูลผ่านช่องทางสื่อสารมายังปลายทางใช้เวลา D วินาที สำหรับ byte แรกเดินทางมาถึงปลายทาง และเวลาเพิ่มเติมอีก D วินาที สำหรับข้อความตอบรับกลับถึงต้นทาง ด้วยเงื่อนไขนี้ถ้าไม่มีการจำกัดปริมาณของข้อมูลในการส่ง สามารถจะส่งได้ทั้งหมด 2RD bit หรือ RD/4 byte ในความเป็นจริง ต้นทางถูกจำกัดขนาดของ window ไว้ที่ W byte จนกระทั่งได้รับการตอบรับ ดังนั้นถ้า W > RD/4 หมายความว่าการส่งข้อมูลมีประสิทธิภาพสูงสุด แต่ถ้า W < RD/4 ค่าประสิทธิภาพโดยทั่วไป คือ อัตราส่วนของ W กับ RD/4 เราสรุปเป็นสูตรได้ว่า normalized throughput หรือ S มีค่าดังนี้

$S = 1$ เมื่อ $W > RD/4$

$S = (4W)/(RD)$ เมื่อ $W < RD/4$

จากขนาดสูงสุดของ window คือ $2^{16} - 1 = 65,535$ byte ซึ่งมากพอสำหรับ application ทั่วๆไป เช่น 1-Gbps Ethernet ที่เชื่อมกันด้วยระยะทาง 100 เมตร หรือแม้กระทั่ง T-1 (1,544 Mbps) ผ่านดาวเทียม แต่ก็มีบางกรณีที่ขนาดของ window ธรรมดาไม่สามารถทำงานได้ดี เช่น Optical SDH (synchronous digital hierarchy) ที่ทำงานที่ 155 Mbps เชื่อมต่อกันด้วยระยะทางที่ไกลมาก ในกรณีนี้ window scale factor จะถูกนำมาใช้เพื่อเพิ่มประสิทธิภาพ เช่นอาจใช้ window scale factor เท่ากับ 4 ทำให้ขนาดของ window เพิ่มเป็น $2^{20} - 1 = 1,048,575$ byte

3.2.2 กรรมวิธีในการจัดส่งซ้ำ (retransmission)

TCP มีการควบคุมทั้ง error control และ flow control TCP ไม่มีการใช้ negative acknowledgement เหมือนใน link control protocol แต่ TCP จะใช้เฉพาะ positive acknowledgement และ retransmission เมื่อไม่ได้รับ acknowledgement ภายในช่วงเวลาหนึ่ง

มีอยู่ 2 เหตุการณ์ที่ทำให้เกิดการส่ง TCP segment ซ้ำ คือ อย่างแรก segment ได้รับความเสียหาย ในระหว่างการเดินทาง แต่เดินทางถึงปลายทาง ในกรณีนี้ checksum ที่กำกับมาใน segment

จะเป็นตัวบอกด้านฝั่งรับว่า มีข้อผิดพลาดให้ทำการคัด segment นั้นทิ้ง ส่วนอย่างที่สอง segment ไปไม่ถึงปลายทาง และรวมถึงฝั่งส่งไม่ทราบว่า segment ไม่ได้ถูกจัดส่งถึงปลายทางอย่างสมบูรณ์

ถ้า segment เดินทางไม่สำเร็จ จะไม่มี ACK ตอบกลับ และจะต้องทำการส่งซ้ำ ด้วยเหตุการณ์นี้จะต้องมี timer (นาฬิกาจับเวลา) ที่ตั้งเวลาไว้ให้แต่ละ segment ที่ถูกจัดส่ง ถ้า timer บอกว่าหมดเวลา ก่อนที่จะมีการตอบรับ segment นั้น ฝั่งส่งจะทำการส่ง segment นั้นซ้ำ

หลักในการออกแบบใน TCP คือค่าของ retransmission timer ถ้าน้อยเกินไป จะเกิดการส่ง segment ซ้ำจำนวนมาก ทำให้เกิดปริมาณข้อมูลในเครือข่ายเพิ่มขึ้น แต่ถ้าตั้งไว้มากเกินไป จะทำให้ TCP ช้าต่อการตอบสนองต่อการจัดส่ง segment ซ้ำ ค่าของ retransmission timer ควรจะตั้งไว้มากกว่า round-trip delay (เวลาที่ส่ง segment ไปและได้รับ ACK ตอบกลับ) เห็นได้ว่าค่า delay ที่แปรเปลี่ยนไปได้ตลอดแม้แต่ภายใต้การใช้งานเครือข่ายเหมือนเดิม ดังนั้นสถิติของค่า delay ควรจะเปลี่ยนแปลงไปตามเงื่อนไขของ Internet

มี 2 กรรมวิธีที่ใช้คือ ตั้งเวลาคงที่ให้กับ timer เพื่อเป็นการศึกษาถึงพฤติกรรมคร่าวๆของ Internet วิธีนี้ไม่ค่อยมีประสิทธิภาพในการตอบสนองต่อการเปลี่ยนแปลงของเครือข่าย ถ้าตั้งเวลาไว้มากไปก็จะตอบสนองช้า ถ้าตั้งเวลาไว้น้อยไปจะเกิดการส่งซ้ำมาก ยิ่งเพิ่มความแออัดให้กับเครือข่าย ส่วนแบบปรับตัวได้เอง (Adaptive) ก็มีปัญหาในตัวมันเอง สมมติว่า TCP มีการเก็บข้อมูลเวลาที่ใช้จนกว่าจะได้รับ acknowledgement และนำมาเฉลี่ย เพื่อตั้งเวลาให้ retransmission timer ซึ่งค่านี้ยังไม่น่าเนื่องจากเหตุผล 3 อย่างนี้

- TCP ฝั่งรับอาจจะไม่ตอบรับทันที เพราะว่าถ้าสะสมไว้จำนวนหนึ่งแล้วตอบรับจะมีประสิทธิภาพสูงกว่า
- เมื่อทำการส่ง segment ในเครือข่าย ผู้ส่งไม่สามารถรู้ว่า ACK ที่ได้รับเป็นการตอบรับ segment ที่ส่งครั้งแรก หรือ segment ที่ทำการส่งซ้ำ
- สภาวะของ Internet อาจมีการเปลี่ยนแปลงอยู่ตลอดเวลา

จากปัญหาข้างต้นเหล่านี้ทำให้ไม่มีวิธีใดที่สมบูรณ์ แต่ว่าได้มีการแนะนำการคำนวณเวลาให้กับ timer ใน RFC 793

3.2.3 นาฬิกาจับเวลาสำหรับการจัดส่งซ้ำแบบปรับได้ (Adaptive retransmission timer)

ถ้าสภาวะของเครือข่ายหรือ Internet เปลี่ยนไป ค่าของเวลาที่ตั้งใน retransmission timer ก็จะกลายเป็นมีค่ามากไปหรือน้อยไปทันที โดยทั่วไปแล้ว TCP จะประเมินค่าของ round-trip delay โดยการตรวจสอบรูปแบบของค่า delay ของ segment ล่าสุดจำนวนหนึ่ง และจะนำมากำหนดให้ timer มีค่ามากกว่า round-trip delay อยู่จำนวนหนึ่ง

วิธีการหนึ่งที่ง่ายๆ เพื่อหาค่าเฉลี่ยของการสำรวจ round-trip time ของ segment จำนวนหนึ่ง ถ้าค่าเฉลี่ยที่คาดเดาค่า round-trip delay ในอนาคตมีความแม่นยำ จะส่งผลให้ retransmission timer มีประสิทธิภาพดีตามไปด้วย ตัวอย่างของสมการในการคิดค่าเฉลี่ยแสดงดังนี้

$$ARTT(K+1) = \frac{1}{K+1}\sum_{i=1}^{K+1} RTT(i) \qquad (3.1)$$

เมื่อ RTT(i) คือค่า round-trip time ที่สำรวจจากการจัดส่ง segment ที่ i และ ARTT(K) คือค่าเฉลี่ยของ round-trip time ของ segment ก่อนหน้าจำนวน K segment

สมการนี้ถูกนำมาจัดรูปใหม่ได้เป็น

$$ARTT(K+1) = \frac{K}{K+1}ARTT(K) + \frac{1}{K+1}RTT(K+1) \qquad (3.2)$$

ด้วยสมการนี้ทำให้เราไม่ต้องทำการคำนวณ summation ทั้งหมดซ้ำในแต่ละครั้ง ในแต่ละ term ใน summation ถูกกำหนดให้ใช้ค่า weight เดียวกัน ทำให้แต่ละ term ถูกคูณด้วยค่าคงที่ 1/(K+1) โดยทั่วไปเราจะใช้ค่า weight ที่มากกว่ากับ segment ที่ใหม่กว่า เพราะว่า segment ที่ใหม่กว่าน่าจะแสดงถึงสถานะในอนาคตได้ใกล้เคียงกว่า วิธีการสำหรับคาดเดาค่าถัดไปจากกลุ่มของเวลาในอดีตที่ถูกแนะนำใน RFC 793 คือการเฉลี่ยแบบ exponential

$$SRTT(K+1) = \alpha \times SRTT(K) + (1-\alpha) \times RTT(K+1) \qquad (3.3)$$

เมื่อ SRTT(K) คือค่าประมาณของ smoothed round-trip time เปรียบเทียบกับ สมการที่ 3.2 โดยการใช้ค่าคงที่ α ($0 < \alpha < 1$) ที่เป็นอิสระจากจำนวนที่เคยสำรวจในอดีต เรายังคงใช้สถานการณ์ที่ผ่านมาในอดีตมาคำนวณ แต่เราให้ความสำคัญน้อยลงตามลำดับที่ผ่านไป เพื่อให้เข้าใจดียิ่งขึ้น เราจะพิจารณาโดยการขยายความ สมการที่ 3.3

$$SRTT(K+1) = (1-\alpha)RTT(K+1) + \alpha(1-\alpha)RTT(K) + \alpha^2(1-\alpha)RTT(K-1) + \ldots + \alpha^K(1-\alpha)RTT(1) + \alpha^{K+1}RTT(0)$$

ทั้ง α และ $(1 - \alpha)$ มีค่าน้อยกว่า 1 แต่ละ term ที่ต่อเนื่องกันไปในสมการจะมีค่าน้อยลง เช่น เราแทนค่า $\alpha = 0.8$ ได้เป็น

$$SRTT(K+1) = 0.2RTT(K+1) + 0.16RTT(K) + 0.128RTT(K-1) + ...$$

สังเกตได้ว่าค่าที่สำรวจยิ่งผ่านมานานมากก็จะมีผลต่อค่าเฉลี่ยน้อยลง

ขนาดของ coefficient (ค่าสัมประสิทธิ์) ที่น้อยกว่า จะให้ความสำคัญต่อค่าที่สำรวจล่าสุดมากกว่า เช่น อาจกล่าวได้ว่าค่า α ที่ 0.5 เป็นการให้ความสำคัญต่อค่าที่สำรวจ 4-5 ครั้งล่าสุด ต่างจาก ค่า α ที่ 0.875 ที่เฉลี่ยความสำคัญอย่างค่อนข้างเท่าเทียมกันไปยังค่าที่สำรวจ 10 ครั้งล่าสุดหรือมากกว่านั้น ประโยชน์ของการใช้ค่าที่เล็กกว่าของ α คือค่าเฉลี่ยจะมีผลตอบสนองต่อการเปลี่ยนแปลงของเครือข่ายอย่างรวดเร็ว แต่ข้อเสียคือถ้ามีปริมาณของข้อมูลในเครือข่ายเพิ่มขึ้นอย่างรวดเร็วในเวลาสั้นๆ จะทำให้การใช้ค่า α ที่เล็กเกินไปพยายามปรับตัวให้สอดคล้องกับเครือข่าย แต่ไม่สามารถปรับได้ทัน จึงไม่มีประสิทธิภาพ

สมการที่ 3.3 ถูกใช้ใน RFC 793 เพื่อประเมินค่า round-trip time ปัจจุบัน อย่างที่กล่าวไปแล้วว่า retransmission timer ควรถูกกำหนดให้มีค่ามากกว่าค่าคาดการณ์ของ round-trip time เราอาจใช้วิธีบวกค่าคงที่เข้าไป ดังนี้

$$RTO(K+1) = SRTT(K+1) + \Delta$$

เมื่อ RTO คือ retransmission timer (อาจเรียกได้อีกชื่อว่า retransmission timeout) และ Δ คือค่าคงที่ ข้อเสียของสมการนี้คือ Δ ไม่ได้เป็นอัตราส่วนที่พอเหมาะกับ SRTT เช่นถ้าค่าของ SRTT มาก ในขณะที่ Δ มีค่าน้อย จะทำให้ค่ารวมของ RTT น้อยไปจนเกิดการส่ง segment ซ้ำ แต่ถ้าค่าของ SRTT น้อยแต่ Δ มีค่าค่อนข้างมากจะเกิดปัญหาในเรื่องของการหน่วงเวลามากไปในการจัดส่ง segment ที่หายไป แต่ใน RFC 793 ได้กำหนดให้ใช้ค่าที่เป็นอัตราส่วนต่อ SRTT คือ

$$RTO(K+1) = MIN(UBOUND, MAX(LBOUND, \beta \times SRTT(K+1))) \qquad (3.4)$$

เมื่อ UBOUND และ LBOUND ถูกกำหนดให้เป็นค่าคงที่ของ upper bound และ lower bound ของค่า timer และ β เป็นค่าคงที่ RFC 793 ไม่ได้แนะนำค่าที่เหมาะสมตายตัวลงไป แต่ค่าตามตัวอย่างจะใช้ α ระหว่าง 0.8 ถึง 0.9 และ β ระหว่าง 1.3 ถึง 2.0

3.2.4 กฎเกณฑ์ในการพัฒนาทางเลือกของ TCP

มาตรฐาน TCP ได้กำหนดข้อมูลทางเทคนิคที่ชัดเจนของ protocol เพื่อให้การสื่อสารระหว่าง TCP เป็นไปอย่างถูกต้อง อย่างไรก็ตาม ได้มีการเปิดช่องทางไว้สำหรับ option (ทางเลือก)

ที่จะพัฒนาก็ได้ ไม่พัฒนาก็ได้ ถึงแม้การพัฒนาของ protocol ที่แตกต่างกัน 2 แบบจะสามารถสื่อสารกันได้ แต่อาจจะมีปัญหาด้านประสิทธิภาพ กลุ่มของการออกแบบสำหรับ option ที่ถูกจัดไว้มีดังนี้

- Send policy
- Deliver policy
- Accept policy
- Retransmit policy
- Acknowledge policy

3.2.4.1 Send policy

ถ้าไม่รวมการ PUSH ข้อมูล และทำการปิด transmission window แล้ว การส่งข้อมูลของ TCP จะเป็นแบบอิสระ โดยข้อมูลจะถูกจัดส่งเมื่อ TCP คิดว่าเหมาะสม ข้อมูลที่ส่งมาจากผู้ใช้จะถูกจัดเก็บไว้ใน buffer ใน transmit buffer TCP อาจจะสร้าง segment ขึ้นมาสำหรับแต่ละชุดข้อมูลของผู้ใช้หรืออาจจะรอจนข้อมูลสะสมจนมากพอแล้วจึงค่อยสร้าง segment และจัดส่ง ในทางปฏิบัติจะขึ้นอยู่กับการพิจารณาด้านประสิทธิภาพ ถ้าการส่งที่ไม่บ่อยและส่งทีละมากๆ จะมี overhead ต่ำ ทั้งในแง่ของการสร้าง segment และการประมวลผล ในอีกแง่หนึ่งถ้าการส่งมีความถี่มากและส่งทีละน้อยๆ จะทำให้ระบบตอบสนองได้รวดเร็วกว่า

3.2.4.2 Deliver policy

ถ้าไม่รวมการ PUSH ข้อมูลแล้ว TCP ทางฝั่งรับจะเป็นอิสระในการจัดส่งข้อมูลไปให้ผู้ใช้ TCP อาจจัดส่งข้อมูลเรียงตามลำดับ segment ที่ได้รับ หรือจะเก็บสะสมไว้ใน buffer จนได้จำนวนที่เหมาะสมแล้วจึงทำการจัดส่ง ในทางปฏิบัติจะขึ้นอยู่กับการพิจารณาด้านประสิทธิภาพ ถ้ามีการจัดส่งไม่บ่อยและส่งทีละมากๆ ผู้ใช้อาจจะได้รับข้อมูลที่ไม่ค่อยต่อเนื่องนัก แต่ในทางกลับกันถ้ามีการจัดส่งบ่อยแต่ส่งทีละน้อยๆ จะทำให้มีการประมวลผลของ TCP ทั้ง 2 ด้านใน software ของผู้ใช้มากเกินความจำเป็น และยังเป็นการเพิ่มจำนวนของการขัดจังหวะระบบปฏิบัติการอย่างไม่จำเป็นอีกด้วย

3.2.4.3 Accept policy

เมื่อ segment เดินทางแบบเรียงลำดับข้ามเครือข่ายมายัง TCP ฝั่งรับ TCP จะจัดเก็บข้อมูลไว้ใน receive buffer เพื่อพักไว้ก่อนจัดส่งไปให้กับผู้ใช้ แต่เป็นไปได้ที่แต่ละ segment จะเดินทางมาถึงแบบไม่เรียงลำดับ ในกรณีนี้ TCP ฝั่งรับมี 2 ทางเลือก คือ

- In order: จะทำการรับเฉพาะ segment ที่เรียงตามลำดับเข้ามา แต่ถ้ามี segment ไหนข้ามลำดับ segment นั้นจะถูกคัดทิ้ง
- In window: รับเฉพาะ segment ที่อยู่ภายในขอบเขตของ window ฝั่งรับ

แบบ In-order policy นั้นง่ายในการพัฒนา แต่เป็นการเพิ่มภาระให้กับเครือข่าย เนื่องจากจะทำให้ TCP ฝั่งส่งเกิด time out และจัดส่ง segment ที่สมบูรณ์แต่ถูกคัดทิ้งเพราะมาถึงผิดลำดับซ้ำอีกครั้ง ยิ่งกว่านั้น ถ้าเพียง segment เดียวหายไปในเครือข่าย จะส่งผลให้ segment ทั้งหมดที่ตามมาต้องถูกจัดส่งซ้ำไปด้วย หลังจาก TCP ฝั่งส่งเกิด time out ของ segment ที่หายไป

แบบ In-window policy อาจช่วยลดปริมาณการส่งซ้ำลงได้ แต่กลไกจะมีความซับซ้อนมากขึ้นในด้านการตรวจสอบ การจัดการ buffer และสถานะของข้อมูลที่รับเข้ามาแบบไม่เรียงลำดับ

3.2.4.4 Retransmit policy

TCP ดูแลแถว (queue) ของ segment ที่ถูกส่งแต่ยังไม่ได้รับ acknowledgement ข้อกำหนดของ TCP ระบุว่า TCP จะส่ง segment ซ้ำถ้าฝั่งส่งไม่ได้รับ acknowledgement ภายในเวลาหนึ่ง การพัฒนา TCP จะต้องทำตาม 1 ใน 3 วิธีการจัดส่งซ้ำนี้

- First only: ใช้ retransmission timer เพียงชุดเดียวเพื่อดูแลทั้ง queue ถ้าได้รับ acknowledgement จะย้าย segment ที่ได้รับตอบรับแล้วออกจาก queue และตั้ง timer ใหม่ ถ้า timer บอกหมดเวลา จะจัดส่ง segment แรกใน queue ซ้ำและตั้งเวลาใหม่
- Batch: ใช้ retransmission timer เพียงชุดเดียวเพื่อดูแลทั้ง queue ถ้าได้รับ acknowledgement จะย้าย segment ที่ได้รับตอบรับแล้วออกจาก queue และตั้ง timer ใหม่ ถ้า timer บอกหมดเวลา จะจัดส่งทุก segment ใน queue ซ้ำและตั้งเวลาใหม่
- Individual: ใช้ timer หนึ่งตัวต่อ segment ใน queue ถ้าได้รับ acknowledgement จะย้าย segment ที่ได้รับตอบรับแล้วออกจาก queue และยกเลิก timer ที่ใช้คู่กับ segment ที่ได้รับตอบรับ ถ้า timer บอกหมดเวลา การจัดส่งซ้ำจะเกิดขึ้นเฉพาะ segment ที่ไม่ได้รับตอบรับ และจะตั้งเวลาเฉพาะ timer ที่สัมพันธ์กับ segment ที่ส่งใหม่เท่านั้น

แบบ First-only policy จะมีประสิทธิภาพในแง่ของการสร้างปริมาณข้อมูลในเครือข่าย เพราะว่าทำการส่งซ้ำเฉพาะ segment ที่ไม่ได้รับ ACK แต่ว่าการที่ไม่มี timer สำหรับ segment ที่ 2 ใน queue ทำให้ต้องรอจน segment แรกได้รับ ACK ก่อน ซึ่งทำให้เกิด delay มากในกรณีที่ไม่ได้รับ ACK มากกว่า 1 segment แบบ Individual policy แก้ปัญหานี้โดยลงทุนใช้ timer จำนวนมาก ทำให้การพัฒนามีความซับซ้อนมาก ส่วนแบบ Batch policy ก็สามารถขจัดปัญหาด้าน delay ที่เกิดในแบบแรกได้ แต่จะสร้างปริมาณข้อมูลในเครือข่ายขึ้นมากโดยไม่จำเป็น

ผลที่ได้ในทางปฏิบัติของวิธีการเหล่านี้จะขึ้นอยู่กับ Accept policy ที่ฝั่งรับด้วย ถ้าฝั่งรับใช้ In-order accept policy จะทำให้เกิดการคัดทิ้งของ segment ที่ได้รับหลังจากมีบาง segment

หายไป ซึ่งเหมาะสมถ้าใช้คู่กับ Batch retransmission ถ้าฝั่งรับใช้ In-window accept policy การใช้ First-only หรือ Individual retransmission policy ที่ฝั่งส่งจะเป็นการเหมาะสมที่สุด แต่ในเครือข่ายคอมพิวเตอร์ขนาดใหญ่ที่มีการผสมผสานของวิธีการต่างๆ เราอาจจะนำ Accept policy ทั้ง 2 แบบมาใช้พร้อมกัน

3.2.4.5 Acknowledge policy

เมื่อ segment เดินทางมาถึงที่ฝั่งรับ TCP มี 2 วิธีการ ในการตอบ acknowledgement

- Immediate: เมื่อได้รับ segment เข้ามาจะทำการสร้าง segment ที่ไม่มีข้อมูล (ว่างเปล่า) เพื่อตอบ ACK กลับไปทันที
- Cumulative: เมื่อได้รับ segment เข้ามา จะบันทึกเวลาที่ต้องการการตอบรับไว้ แต่จะรอว่ามีข้อมูลที่ต้องการส่งออกหรือไม่ ถ้ามีก็จะฝาก ACK ไปกับการส่งข้อมูลนั้น (เรียกว่า piggyback) เพื่อไม่ให้เกิดการหน่วงเวลาที่ยาวนาน จะมีการตั้งเวลาไว้ ถ้าหมดเวลาที่ตั้งไว้จะทำการตอบ ACK กลับไปทันที โดยที่ใน segment นั้นไม่มีข้อมูลอื่น

แบบ Immediate policy เป็นแบบที่ง่ายและตอบสนองรวดเร็ว แต่เกิดการส่งข้อมูลที่มากเกินจำเป็น จากการส่ง segment เปล่า และเป็นการเพิ่มภาระให้กับเครือข่าย แต่จากการทำงานที่ TCP รับ segment และตอบรับด้วย ACK ทันที จะทำให้ข้อมูลถูกนำออกจาก buffer และส่งให้กับ application ทันที ทำให้เกิดการขยายของ receive window และการส่ง ACK เปล่ากลับไปจะทำให้เกิดการเพิ่มสิทธิ์ในการส่งของ TCP ฝั่งส่งอีกด้วย

เนื่องจาก overhead จำนวนมากที่เกิดจาก Immediate policy ดังนั้นแบบ Cumulative policy จึงเป็นที่นิยมใช้ แต่วิธีการนี้ต้องมีการประมวลผลอย่างมากที่ฝั่งรับ และการประเมินค่า round-trip delay ยังเป็นงานที่ซับซ้อนอีกด้วย

3.3 กลไกควบคุมความแออัดของ TCP (TCP Congestion Control)

สภาพความแออัดในเครือข่ายหรือใน Internet สร้างปัญหาใหญ่ให้กับระบบปลายทางดังต่อไปนี้ ทำให้ throughput ลดลง และเพิ่มเวลาในการตอบสนอง ในเครือข่ายแบบ packet-switching หรือ frame relay การเลือกเส้นทางทำโดย dynamic routing ที่สามารถกระจายความแออัดออกอย่างทั่วถึง ไม่ให้มารวมกันอยู่ที่จุดไม่กี่จุด คล้ายกันกับกลไกหาเส้นทางของ Internet ที่สามารถกระจายความแออัดของ router และเส้นทางในเครือข่าย เพื่อหลีกเลี่ยงความแออัด อย่างไรก็ตามวิธีการเหล่านี้มีประสิทธิภาพเมื่อใช้กับ load ที่ไม่เท่ากัน (unbalanced load) และกับปริมาณข้อมูลที่มีการเพิ่มขึ้นและลดลงอย่างรวดเร็ว แต่ท้ายที่สุดความแออัดสามารถถูกควบคุมได้จากการจำกัดจำนวนข้อมูลที่ถูกส่งเข้าไปใน Internet ให้เหมาะสม ซึ่งหัวข้อนี้จะอธิบายในตอนถัดไป

การควบคุมความแออัดของ TCP/IP มีความซับซ้อนและยากต่อการศึกษา เนื่องจากเหตุผลต่อไปนี้

- IP เป็นแบบ connectionless และ stateless ซึ่งไม่มีการเก็บสถานะของการเชื่อมต่อในการตรวจสอบ จึงมีประโยชน์น้อยมากต่อการนำมาควบคุมความแออัด
- TCP ใช้ end-to-end flow control ซึ่งจะนำมาควบคุมความแออัดได้ในทางอ้อม และเพราะว่า delay ในเครือข่ายหรือใน Internet แปรผันมากและอาจจะหน่วงนาน (เมื่อเทียบกับขนาดของ segment) จน TCP มองว่าอยู่ในสภาวะที่ไม่น่าเชื่อถือ
- เนื่องจากไม่มีการระบุและบังคับใช้กลไกอย่างชัดเจน ทำให้ TCP จากแต่ละผู้พัฒนาไม่สามารถทำงานร่วมกันได้เป็นอย่างดีในระดับของ flow control

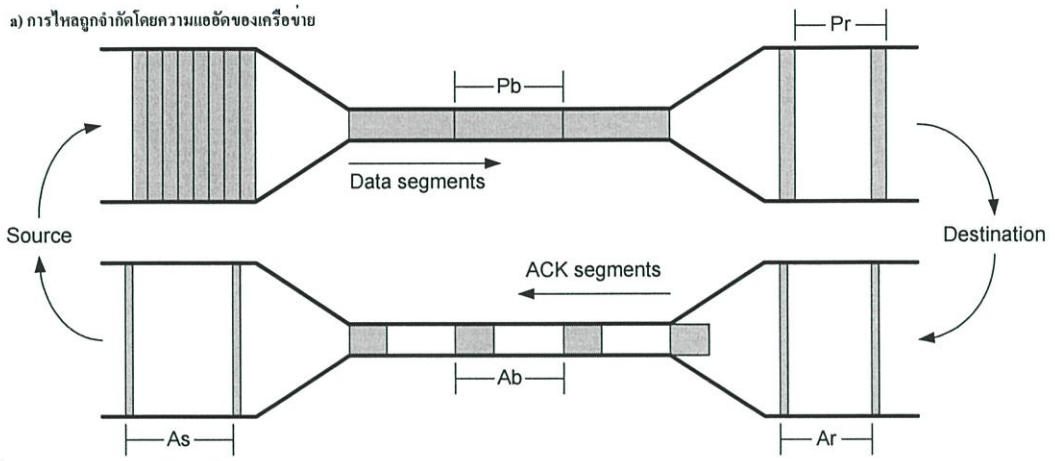
ภายใต้สภาวะแวดล้อมของ connectionless ใน IP ICMP Source Quench เป็นเครื่องมือเดียวที่ควบคุม flow อย่างหยาบๆ และไม่มีประโยชน์มากนักต่อการควบคุมความแออัด ส่วนใน TCP มีเครื่องมือเดียวที่นำมาใช้ควบคุมความแออัด คือ sliding window ซึ่งกลไกนี้ถูกออกแบบมาเพื่อใช้จัดการแบบ end-to-end อย่างไรก็ตามด้วยการพัฒนาของวิธีการที่ฉลาดที่นำมาใช้ในกลไกนี้ ทำให้สามารถตรวจจับสภาพความแออัด (congestion detection), หลีกเลี่ยงความแออัด (congestion avoidance) และ การปรับสู่สภาพปกติ (recovery) หัวข้อนี้เราจะอธิบายถึงสิ่งที่สำคัญมาก และกลไกที่ถูกพัฒนาอย่างแพร่หลาย เราจะเริ่มจากความสัมพันธ์ระหว่าง TCP flow และ congestion control

3.3.1 TCP flow และ congestion control

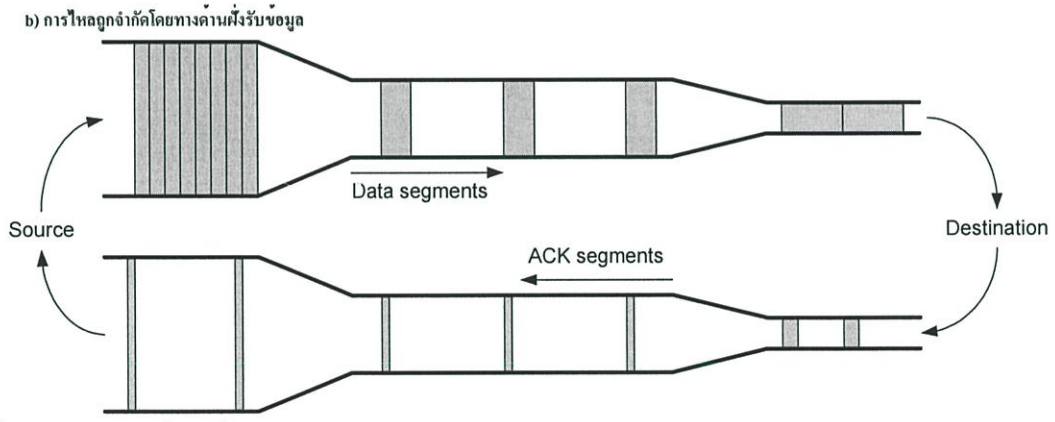
ใน data link control protocol ใช้เทคนิคของ sliding-window flow control เพื่อฝั่งรับกำหนดจังหวะให้กับฝั่งส่ง ฝั่งรับจะตอบด้วย acknowledge frame เท่านั้น เพื่อเป็นการขยาย window เพื่อเพิ่มขนาด buffer วิธีการนี้ถูกนำมาใช้ใน TCP เช่นกัน แต่อัตราการส่งของ TCP จะถูกกำหนดโดยอัตราการได้รับ ACK ของ segment ก่อนหน้า อย่างไรก็ตามในกรณีของ TCP อัตราของ ACK ที่เข้ามาจะถูกกำหนดโดยคอขวด (bottleneck) ในเส้นทางเดินระหว่างต้นทางและปลายทาง และคอขวดอาจเป็นได้ทั้งจากปลายทางและใน Internet

รูปที่ 3.4a แสดงถึงคอขวดที่เกิดจากภายใน Internet อธิบายรูปนี้ได้ว่า มีท่อต่อเชื่อมระหว่างต้นทางและปลายทาง ความกว้างของท่อจะแสดงถึงอัตราการส่งผ่านข้อมูล ระบบต้นทางและปลายทางทั้งคู่อยู่บนเครือข่ายความเร็วสูง และตัวระบบเองสามารถทำงานด้วยความเร็วสูงด้วย ท่อเล็กๆตรงกลางหมายถึง link ที่มีความเร็วต่ำที่เป็นจุดคอขวด แต่ละ segment แสดงด้วยสี่เหลี่ยมที่พื้นที่ในสี่เหลี่ยมหมายถึงปริมาณของข้อมูล ดังนั้นเมื่อ segment ถูกส่งเข้าไปในท่อที่เล็ก ก็จะใช้เวลาเดินทางมาก เวลา Pb คือค่าต่ำสุดของ segment ที่ใช้สำหรับเดินทางผ่านท่อที่เล็กที่สุด เมื่อ segment เดินทางมาถึงปลายทางด้านรับ ซึ่งมีขนาดของท่อที่ใหญ่ขึ้น ทำให้อัตราการส่งข้อมูล

ขณะนั้นเพิ่มขึ้น แต่ interarrival time (ช่วงเวลาที่แต่ละ segment เดินทางมาถึง) ยังคงเท่าเดิม ดังนั้น เวลา Pr จะเท่ากับ Pb ถ้าฝั่งรับตอบ acknowledge segment ตามที่เข้ามา เวลาของ ACK segment ที่ส่งจากฝั่งรับจะถูกกำหนดโดย segment ที่เข้ามา ทำให้ Ar = Pr สุดท้ายถ้าช่องเวลา (time slot) สำหรับ Pb ใหญ่พอสำหรับข้อมูลผ่าน ก็น่าจะใหญ่พอสำหรับ ACK segment ด้วย ดังนั้น Ab = Ar



รูปที่ 3.4a การเคลื่อนที่ของ TCP segment



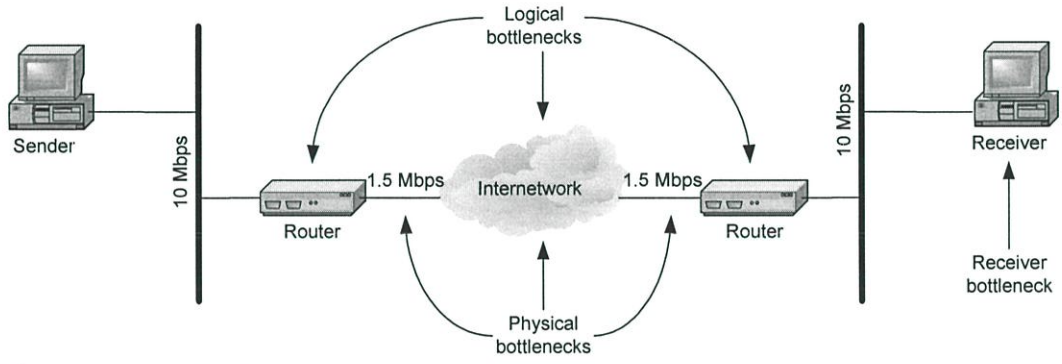
รูปที่ 3.4b การเคลื่อนที่ของ TCP segment

การเดินทางของ ACK กลับมาที่ฝั่งส่งข้อมูล ทำหน้าที่เหมือนเป็นการให้จังหวะ (pacing signal) ในสภาวะสมดุล หลังจากผ่านช่วงเริ่มต้นที่มีการส่งข้อมูลอย่างหนักแล้ว อัตราของ segment ของฝั่งส่งจะเท่ากันกับอัตราการเดินทางมาถึงของ ACK ดังนั้นอัตราของ segment ของฝั่งส่งจะเท่ากับ link ที่ช้าที่สุดในเส้นทาง ในกรณีนี้ TCP จะตรวจสอบพบคอขวดอย่างอัตโนมัติ และพยายามรักษาให้เกิดการไหลของข้อมูลอย่างคงที่ เราเรียกพฤติกรรมนี้ว่า self-clocking

พฤติกรรม self-clocking ยังคงเหมือนเดิม ในกรณีที่คอขวดย้ายไปอยู่ที่ด้านฝั่งรับ ในรูปที่ 3.4b แสดงถึงกรณีนี้ เราสมมติว่า link ที่ช้าที่สุดในเครือข่ายมีขนาดค่อนข้างใหญ่ประมาณครึ่งหนึ่ง

ของขนาดของท่อที่ฝั่งส่ง แต่ขนาดของท่อที่ฝั่งรับมีขนาดเล็กที่สุด ทำให้ ACK ถูกส่งออกมาด้วยอัตราเท่ากับปริมาตรของฝั่งรับ และผลของ ACK ที่ให้จังหวะแก่ฝั่งส่งทำได้เร็วที่สุดแค่นั้น

รูปที่ 3.4a และ 3.4b แสดงถึงสิ่งที่สำคัญ คือ ฝั่งส่งไม่มีทางรู้ว่า ACK ที่ได้รับเป็นผลมาจากความแออัดของ Internet (congestion control) หรือผลจากฝั่งรับ (flow control) ถ้า ACK มาค่อนข้างช้าเนื่องจากความแออัดของเครือข่าย การที่ฝั่งส่งจัดส่ง segment ให้ช้ากว่าจังหวะของ ACK จะช่วยลดความแออัดลงได้ แต่ถ้าจังหวะของ ACK ช้าเนื่องจาก flow control ของฝั่งรับ การเปลี่ยน send policy จะช่วยแก้ปัญหาได้



รูปที่ 3.5 ความสัมพันธ์ของ TCP flow และ congestion control

คอขวดในเส้นทางระหว่างฝั่งส่งและฝั่งรับสามารถเกิดขึ้นได้ในทุกๆตำแหน่งทั้ง logical หรือ physical รูปที่ 3.5 แสดงถึงความเป็นไปได้ในการเกิดคอขวด ในตัวอย่าง ถ้าฝั่งส่งให้อัตราการส่งทั้งหมดของ LAN กับ TCP connection เดียว ดังนั้นผู้ส่งจะมีอัตราการส่งอยู่ที่ 10 Mbps ในกรณีนี้ link ขนาด 1.5 Mbps ที่ต่อเชื่อมอยู่ระหว่าง router แต่ละข้างกับ Internet จะกลายเป็นคอขวด ในที่นี้คือคอขวดทาง physical เมื่อระบบสมดุล TCP จะมีประสิทธิภาพสูงสุดได้เท่ากับอัตราสูงสุดของคอขวด แต่ว่าในหลายๆกรณีจะมีคอขวดทาง logical เกิดขึ้น เนื่องจากการจัดการ queue ใน router, ใน switch หรือ ที่ฝั่งรับเอง การหน่วงของ queue จะแกว่งขึ้นลงจาก load ที่เข้ามา ทำให้ยากในการควบคุมให้อยู่ในสภาวะสมดุล

การแกว่งตัวของ delay นั้นเป็นธรรมชาติของ IP-based Internet ซึ่งท้าทายการออกแบบ flow policy ของ TCP เพื่อแก้ปัญหา ถ้า TCP flow ช้ามาก จะทำให้ Internet มีอัตราการใช้งานเครือข่ายที่ต่ำกว่าความสามารถจริงที่เครือข่ายสามารถรองรับได้มาก (underutilization) แต่ถ้า TCP ฝั่งส่งจำนวนหนึ่งทำการส่งมากเกินขอบเขต TCP flow จะเบียดเสียดกันหนาแน่น แต่ถ้า TCP ฝั่งส่งจำนวนมากทำการส่งมากๆพร้อมๆกัน จะส่งผลให้เกิดการสูญหายของ segment ในระหว่างเดินทาง ทำให้เกิดการส่งซ้ำ หรือการหน่วงเวลาอย่างมากของ ACK จนทำให้ต้องส่งซ้ำตามมา โดยเฉพาะการส่งซ้ำแบบนี้สามารถทำให้เกิด positive feedback effect คือ หลายๆ segment ถูกส่ง

ซ้ำ, ความแออัดยิ่งเพิ่มมากขึ้น, เวลาหน่วงมากขึ้น และ segment ถูกคัดทิ้งเพิ่มมากขึ้น สรุปก็คือ ผลลัพธ์ที่ได้จากการส่งมากเกินไป จะยิ่งทำให้ประสิทธิภาพแย่ลง

เราอาจมองว่ากลไก sliding-window ของ TCP ถูกออกแบบมาสำหรับ end-to-end flow control แต่ว่าไม่ใช่แค่นั้นมันยังถูกใช้เพื่องานด้าน congestion control ด้วย ตั้งแต่การเผยแพร่ของ RFC 793 ทำให้จำนวนของวิธีการที่ปรับปรุงในความสามารถด้านนี้ถูกพัฒนาขึ้นมามาก ตารางที่ 3.1 แสดงถึงวิธีการที่เป็นที่นิยม ไม่มีวิธีการไหนที่เพิ่มเติมหรือไม่ทำตามมาตรฐานของ TCP ดั้งเดิม แต่ว่าวิธีการเหล่านี้ต่างพัฒนากฎเกณฑ์ภายใต้เงื่อนไขตามที่มาตรฐาน TCP ได้กำหนดไว้ ตารางนี้แสดงถึงวิธีการที่ถูกใช้ใน RFC 1122 และยังแสดงถึงวิธีการที่ถูกพัฒนาใน Berkeley UNIX ที่เป็นที่นิยมแพร่หลาย TCP code 2 แบบนั้นของ Berkeley UNIX คือ Tahoe และ Reno ที่มักถูกนำมาพูดถึงในการเปรียบเทียบผลกับวิธีการ congestion control ของ TCP แบบอื่นๆ

ตารางที่ 3.1 วิธีการที่ถูกพัฒนาเพื่อ congestion control

Measure	RFC 1122	TCP Tahoe	TCP Reno
RTT Variance Estimation	✓	✓	✓
Exponential RTO Backoff	✓	✓	✓
Karn's Algorithm	✓	✓	✓
Slow Start	✓	✓	✓
Dynamic Window Sizing on Congestion	✓	✓	✓
Fast Retransmit		✓	✓
Fast Recovery			✓

3.3.2 การจัดการ retransmission timer

3 วิธีการต่อไปนี้ถูกนำมาใช้คำนวณค่า retransmission timer (RTO) ค่าของ timer นั้นมีผลอย่างมากต่อการตอบสนองต่อความแออัดของ TCP วิธีการเหล่านี้คือ

- RTT Variance Estimation
- Exponential RTO Backoff
- Karn's Algorithm

3.3.2.1 RTT Variance Estimation (Jacobson's algorithm)

วิธีการนี้ถูกกำหนดลงไปในมาตรฐาน TCP และถูกกล่าวไว้แล้วในสมการที่ 3.3 และ 3.4 ด้วยวิธีการนี้ทำให้ TCP สามารถปรับตัวด้วยการเปลี่ยนแปลง round-trip time อย่างไรก็ตามวิธีนี้ไม่ได้ออกแบบไว้สำหรับการเปลี่ยนแปลงที่รวดเร็วมากๆ เช่นใน 3 กรณีนี้

- ถ้าอัตราการส่งข้อมูลของ TCP ค่อนข้างช้า จะทำให้การหน่วงของการส่งมีค่าค่อนข้างมาก เมื่อเปรียบเทียบกับ propagation time และการแกว่งของ delay ที่เกิดจากการแกว่งของขนาดใน IP datagram มีผลชัดเจน ดังนั้นวิธีการคำนวณ SRTT มีบทบาทอย่างมากต่อคุณสมบัติของข้อมูล ซึ่งไม่ใช่คุณสมบัติของเครือข่าย
- ภาระและสภาวะของ Internet อาจจะเปลี่ยนอย่างรวดเร็วตามการเพิ่มขึ้นของปริมาณข้อมูลจากแหล่งอื่น ทำให้เกิดการเปลี่ยนแปลง RTT อย่างมาก
- TCP ทั้ง 2 ฝั่งอาจจะตอบรับแต่ละ segment ทันทีเพราะจากการหน่วงของการประมวลผลเอง และจากการที่ต้องการสะสมให้ได้จำนวนที่มากพอก่อนค่อยตอบกลับ

TCP ดั้งเดิมได้กำหนดการแก้ปัญหานี้ไว้โดยการคูณค่าคงที่เข้ากับ RTT ที่ประเมินได้ จากสมการนี้

$$RTO(K+1) = \beta \times SRTT(K+1)$$

เมื่อค่าโดยปกติของ $\beta = 2$ ในสภาวะสมดุลที่มีการแกว่งของ RTT น้อย ผลของสมการนี้จะทำให้ค่า RTO สูงเกินความจำเป็น และในสภาวะไม่สมดุล ค่าแค่ 2 ไม่พอต่อการป้องกันการส่งซ้ำ

วิธีที่มีประสิทธิภาพมากกว่าคือการประเมินความเปลี่ยนแปลงของ RTT และใช้เป็นข้อมูลให้กับการคำนวณค่า RTO วิธีการหนึ่งที่นำมาใช้ได้คือ การหาค่าเบี่ยงเบนมาตรฐาน (standard deviation) แต่เนื่องจากต้องทำการคำนวณแบบยกกำลังและถอดราก ดังนั้นการวัดความเปลี่ยนแปลงที่ง่ายกว่าที่ถูกนำมาใช้คือ การหาค่าเบี่ยงเบนเฉลี่ย (mean deviation) ดังสมการนี้

$$MDEV(x) = E\left[\left|X - E(X)\right|\right]$$

เหมือนกับการประเมินค่า RTT การหาค่าเฉลี่ยง่ายๆถูกนำมาใช้หา MDEV

$$AERR(K+1) = RTT(K+1) - ARTT(K)$$

$$ADEV(K+1) = \frac{1}{K+1}\sum_{i=1}^{K+1}\left|AERR(i)\right|$$

$$= \frac{K}{K+1}ADEV(K) + \frac{1}{K+1}\left|AERR(K+1)\right|$$

เมื่อ ARTT(K) คือค่าเฉลี่ยง่ายๆที่ถูกกำหนดในสมการ 3.1 และ AERR(K) คือตัวแทนของค่าเบี่ยงเบนเฉลี่ยที่ถูกวัดที่เวลา K

จากนิยามของ ARRT แต่ละ term ใน summation ของ ADEV ถูกกำหนดให้มีค่า weight เท่ากัน คือ แต่ละ term ถูกคูณด้วยค่าคงที่เดียวกันคือ 1/(K+1) ถ้าเราต้องการกำหนดค่า weight ที่มากกว่าให้กับค่าที่ใหม่ที่สุด เพราะเป็นค่าที่น่าจะบอกถึงพฤติกรรมในอนาคตได้ดีกว่าค่าที่ผ่านไปนานแล้ว Jacobson แนะนำให้ใช้การประเมินแบบ dynamic ในการประเมินค่า RTT โดยการใช้วิธีการเดียวกับ exponential smoothing technique ที่ถูกใช้ในการคำนวณหา SRTT ขั้นตอนที่สมบูรณ์ที่เสนอโดย Jacobson แสดงได้ดังนี้

$$
\begin{aligned}
SRTT(K+1) &= (1-g)\times SRTT(K) + g\times RTT(K+1) \\
SERR(K+1) &= RTT(K+1) - SRTT(K) \\
SDEV(K+1) &= (1-h)\times SDEV(K) + h\times |SERR(K+1)| \\
RTO(K+1) &= SRTT(K+1) + f\times SDEV(K+1)
\end{aligned}
\qquad (3.5)
$$

เหมือนที่กำหนดใน RFC 793 (สมการที่ 3.3) SRTT เป็นการประเมินแบบ exponential ของ RTT ด้วย (1-g) ที่มีค่าเท่ากับ α และเปลี่ยนจากการคูณค่าประเมิน SRTT ด้วยค่าคงที่ (สมการที่ 3.4) ค่าประเมินของค่าเบี่ยงเบนเฉลี่ยหลายๆค่าจะถูกนำมารวมกันเป็น SRTT เพื่อเป็นค่า retransmission timer บนพื้นฐานของประสบการณ์ Jacobson ได้เสนอค่าคงที่ต่อไปนี้ในงานวิจัยที่เผยแพร่

$$
\begin{aligned}
g &= 1/8 = 0.125 \\
h &= 1/4 = 0.25 \\
f &= 4
\end{aligned}
$$

ในตอนแรก Jacobson เสนอให้ใช้ค่า f = 2 แต่หลังจากนั้นได้มีการวิจัยเพิ่มและสรุปออกมาว่าให้ใช้ f = 4 และค่านี้เป็นค่ามาตรฐานที่ถูกใช้ในปัจจุบัน จากการใช้งานพบว่า algorithm ของ Jacobson นั้นช่วยปรับปรุงประสิทธิภาพของ TCP เป็นอย่างมาก แต่ว่าด้วยการสนับสนุนของ 2 องค์ประกอบนี้

- Exponential RTO Backoff Algorithm ใช้ในการหาค่า RTO สำหรับ segment ที่ถูกส่งซ้ำ
- Karn's Algorithm ใช้ในการหาตัวแทนของ round-trip เพื่อนำมาคำนวณใน Jacobson's algorithm

3.3.2.2 Exponential RTO Backoff

เมื่อ TCP ฝั่งส่งบอกหมดเวลาสำหรับรอตอบรับ segment TCP จะส่ง segment นั้นซ้ำ RFC 793 จะใช้ค่า RTO เดิมในการส่ง segment ซ้ำ อย่างไรก็ตามเพราะว่า time out น่าจะเกิดจากความแออัดของเครือข่ายที่แสดงได้จากการถูกคัดทิ้งของ packet หรือเวลาหน่วงใน round-trip time ทำให้การใช้ค่า RTO เท่าเดิมนั้นไม่เป็นวิธีการที่ดี

พิจารณาจากกรณีนี้ มี TCP connection ที่ถูกใช้งานอยู่จำนวนหนึ่ง จากต้นทางปลายทางที่หลากหลายใน Internet พื้นที่ที่เกิดความแออัดส่งผลให้ segment ในหลายๆ connection ในจำนวนนี้เกิดสูญหาย หรือ ถูกหน่วง ยิ่งกว่านั้นในช่วงเวลานั้น หลายๆ segment กำลังถูกส่งซ้ำเข้าไปใน Internet ทำให้ความแออัดยิ่งเพิ่มสูงขึ้น ผู้ส่งทั้งหมดก็จะรอ RTO หมดเวลาแล้วก็ส่งซ้ำอีก เป็นรูปแบบที่วนเวียนส่งซ้ำไปเรื่อยๆแบบนี้ เรียกได้ว่าเกิดความแออัดขึ้นแล้ว

วิธีการที่ดีกว่านั้นคือให้ RTO ฝั่งส่งทำการเพิ่ม RTO ในแต่ละครั้งที่ทำการส่ง segment เดิมซ้ำ เรียกว่ากระบวนการ backoff ในกรณีนี้หลังจากที่ส่งซ้ำครั้งแรกแล้ว TCP จะรอด้วยเวลาที่มากกว่าในตอนส่งครั้งแรก แต่ถ้าต้องส่งซ้ำครั้งที่สอง TCP ก็จะรอด้วยเวลาที่มากขึ้นไปอีก วิธีนี้อาจช่วยให้ความแออัดใน Internet ขณะนั้นลดลง

วิธีการง่ายๆในการพัฒนา RTO backoff คือการคูณ RTO แต่ละ segment ด้วยค่าคงที่ในแต่ละการส่งซ้ำ

$$\text{RTO} = q \times \text{RTO} \tag{3.6}$$

สมการ 3.6 ทำให้ค่า RTO มากขึ้นแบบ exponential ในแต่ละการส่งซ้ำ ค่าของ q ที่นิยมใช้คือ 2 ด้วยการคูณด้วย 2 ในแต่ละครั้งทำให้วิธีการนี้ถูกเรียกอีกอย่างว่า binary exponential backoff ซึ่งยังเป็นวิธีการเดียวกับที่ใช้ใน CSMA/CD protocol ใน Ethernet

3.3.2.3 Karn's Algorithm

ถ้าไม่มี segment ถูกส่งซ้ำ วิธีการเลือกตัวอย่าง round-trip ของ Jacobson จะมีประสิทธิภาพดี RTT ของแต่ละ segment ถูกนำมารวมในการคำนวณ สมมติว่า segment หมดเวลา และต้องถูกจัดส่งใหม่ ACK ที่ได้รับตามเข้ามาจะเป็นได้ 2 กรณีดังนี้

- เป็น ACK ที่ตอบรับการส่งครั้งแรกของ segment ในกรณีนี้แสดงว่า RTT มีค่ามากกว่าที่คาดการณ์ไว้ และบ่งบอกถึงสภาวะที่แท้จริงในเครือข่าย
- เป็น ACK ที่ตอบรับการส่งครั้งที่สอง

TCP ฝั่งส่งไม่มีทางรู้ได้เลยว่าเป็น ACK ในกรณีที่ 1 หรือ 2 ถ้าเป็นกรณีที่ 2 แล้ว TCP ใช้วัดว่าเป็น RTT จากการส่งครั้งแรกจนถึงได้รับ ACK เวลาที่วัดได้จะค่อนข้างมาก เมื่อนำค่า RTT ที่ไม่ถูกต้องไปใช้ใน Jacobson's algorithm จะทำให้ได้ SRTT ที่สูงเกินความจำเป็น มากกว่านั้น ผลของการนำค่า SRTT ที่ได้มาประมวลผลซ้ำในรอบถัดไปจะยิ่งเพิ่มความผิดพลาดอย่างมากไปเรื่อยๆ

แม้แต่เป็นการวัด RTT จากการส่งครั้งที่ 2 จนได้รับ ACK ถ้า ACK อันนี้เป็นการตอบรับจากการส่งครั้งแรก จะทำให้ RTT ที่ได้มีค่าน้อยมาก ทำให้ได้ค่าของ SRTT และ RTO ต่ำมาก ซึ่งส่งผลให้เกิด positive feedback ทำให้เกิดการส่งซ้ำตามมาเป็นจำนวนมาก

Karn's algorithm แก้ปัญหานี้โดย

- ไม่ใช้ RTT ที่ถูกวัดสำหรับ segment ที่ถูกส่งซ้ำมามีผลต่อค่า SRTT และ SDEV (สมการที่ 3.5)
- คำนวณ backoff RTO โดยใช้สมการที่ 3.6 เมื่อเกิดการส่งซ้ำ
- ใช้ค่า backoff RTO สำหรับ segment ต่อๆมาจนกระทั่งได้รับ ACK สำหรับ segment ที่ยังไม่ถูกส่งซ้ำ

เมื่อได้รับ ACK สำหรับ segment ที่ยังไม่ถูกส่งซ้ำ Jacobson's algorithm จะถูกนำมาใช้คำนวณค่า RTO ต่อไปทันที

3.3.3 การจัดการ window

วิธีการเพิ่มเติมเพื่อปรับปรุงประสิทธิภาพของ retransmission timer โดยการจัดการที่ send window ขนาดของ send window ของ TCP มีผลอย่างมากในแง่ประสิทธิภาพ และป้องกันการเกิดความแออัด เราจะพูดถึง 4 วิธีการที่มักนำมาพัฒนาใน TCP ที่ใช้ในปัจจุบัน

- Slow start
- Dynamic window sizing on congestion
- Fast retransmit
- Fast recovery

3.3.3.1 Slow start

ขนาดของ send window ที่ใหญ่ที่ใช้ใน TCP จะทำให้ segment จำนวนมากถูกส่งออกไปก่อนที่จะได้รับ ACK โดยปกติธรรมชาติของ self-clocking ของ TCP (ในรูป 3.4) นั้นจังหวะของ TCP จะเป็นไปอย่างลงตัว

วิธีการหนึ่งที่สามารถทำได้ คือ TCP ฝั่งส่งเริ่มต้นการส่งด้วย window ขนาดใหญ่ เพื่อใช้คาดเดาขนาดของ window ที่เป็นไปได้ใหญ่สุดในการเชื่อมต่อนี้โดยประมาณ วิธีนี้มีอาจสร้าง

ปัญหาให้กับระบบเพราะผู้ส่งอาจจะสร้าง segment จำนวนมาก (flood) ใน Internet ก่อนที่ผู้ส่งจะรับรู้จาก time out ว่าปริมาณที่ส่งนั้นมากเกินไป ดังนั้นวิธีที่เหมาะสมกว่าคือการค่อยๆเพิ่มขนาดของ window ทีละน้อยๆ

Jacobson แนะนำวิธีการที่เรียกว่า slow start TCP ใช้ congestion window ที่มีหน่วยเป็น segment แทน byte แต่ละช่วงเวลา การส่งของ TCP จะอยู่ภายใต้สมการนี้

$$awnd = MIN[credit, cwnd] \tag{3.7}$$

เมื่อ

awnd = window ที่ได้รับอนุญาต มีหน่วยเป็น segment คือ จำนวนของ segment ที่ TCP อนุญาตให้ส่งได้โดยไม่ต้องได้รับ ACK

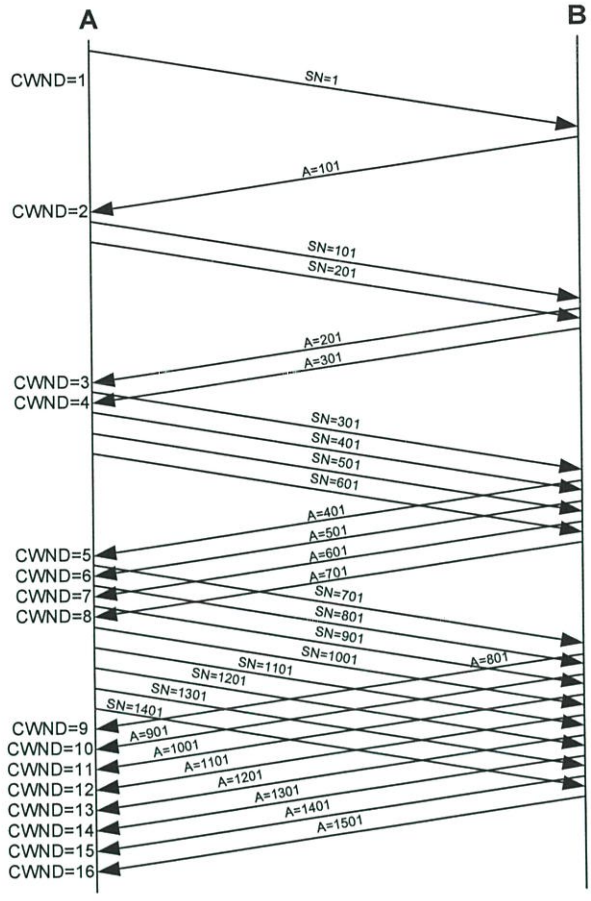
cwnd = congestion window มีหน่วยเป็น segment คือ window ที่ใช้โดย TCP ในตอนเริ่มต้นและเมื่อต้องการลดจำนวนลงของ flow ในสภาพที่แออัด

credit = จำนวนของ credit ที่ยังไม่ได้ใช้ ที่ได้รับจาก ACK ล่าสุด มีหน่วยเป็น segment เมื่อแต่ละ ACK ที่เข้ามาค่านี้จะถูกคำนวณด้วย window / segment size เมื่อ window คือค่าใน field ใน segment ที่เข้ามา (จำนวนของข้อมูลที่ TCP จะตอบรับ)

เมื่อ TCP เปิดช่องทางสื่อสารใหม่ TCP จะตั้งค่า cwnd = 1 หมายความว่า TCP อนุญาตให้ส่งได้ 1 segment เพื่อรอจนกว่าจะได้รับ ACK ก่อนจึงจะส่ง segment ที่ 2 ได้ แต่ละครั้งที่ได้รับ ACK ค่าของ cwnd จะเพิ่มขึ้น 1 จนกระทั่งถึงค่าสูงสุดที่ถูกกำหนด

กลไก slow start จะทำหน้าที่ตรวจสอบ Internet ด้วยว่าปริมาณข้อมูลที่ส่งไม่มากเกินไปที่จะถูกส่งเข้าไปในเครือข่ายที่แออัดอยู่แล้ว เมื่อ ACK มาถึง TCP จะมีสิทธิ์ขยาย window จนขนาดของ window ถูกควบคุมโดยจังหวะของ ACK แทนที่การควบคุมของ cwnd

ชื่อ slow start มักทำให้คนทั่วไปเข้าใจผิดว่าเป็นการเพิ่มขึ้นอย่างช้าๆ แต่จริงๆแล้ว cwnd เพิ่มขึ้นแบบ exponential เมื่อได้รับ ACK แรก cwnd ของ TCP จะมีค่าเพิ่มเป็น 2 และสามารถส่ง 2 segment เมื่อได้รับ ACK ของทั้ง 2 segment จากหลักที่ว่า TCP จะเลื่อน window ไป 1 segment ทุกๆการมาถึงของ ACK และเพิ่ม cwnd ขึ้นทีละ 1 สำหรับแต่ละ ACK ที่มา ดังนั้นที่จุดนี้ TCP สามารถส่งได้ 4 segment เมื่อได้รับ ACK ของทั้ง 4 TCP จะส่งได้อีก 8 segment รูปที่ 3.6 แสดงปรากฏการณ์นี้ ในตัวอย่างนี้ A ทำการส่งหลายๆ segment ที่มีขนาด segment ละ 100 byte และหลังจากผ่านไปประมาณ 4 round-trip time A จะมีกำลังส่งจนภายในท่อเต็มไปด้วยการไหลของ segment อย่างต่อเนื่อง



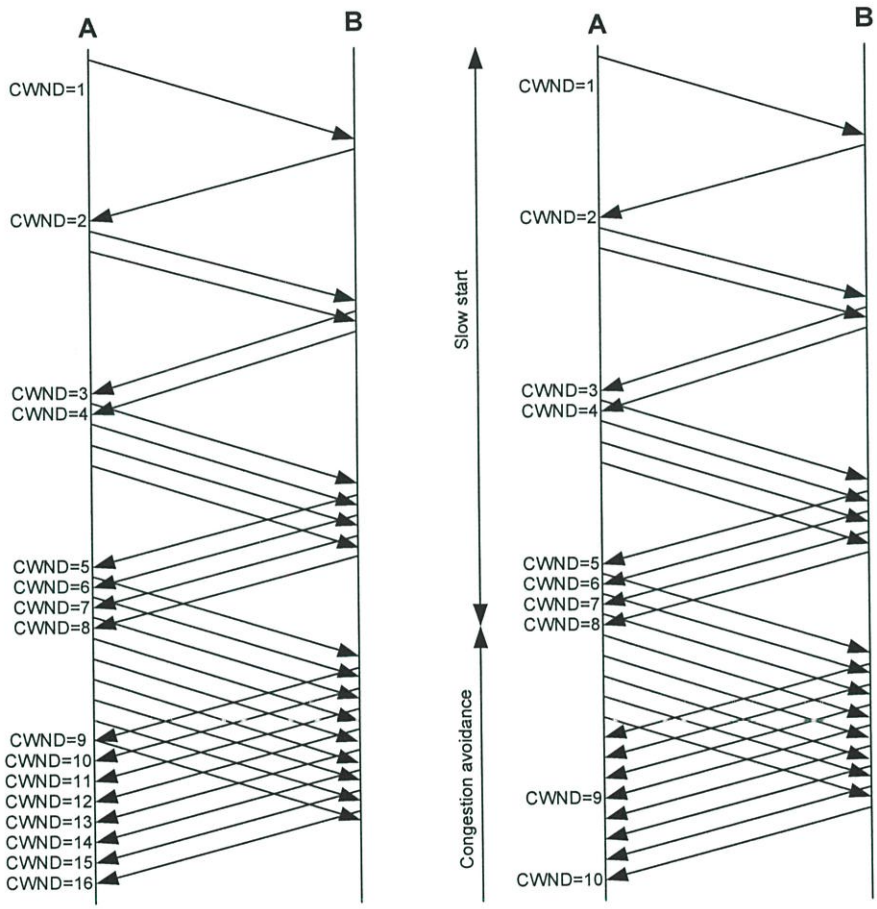
รูปที่ 3.6 ผลของ slow start

3.3.3.2 Dynamic window sizing on congestion

กลไก slow-start พิสูจน์แล้วว่ามีประสิทธิภาพในการเริ่มต้นเชื่อมต่อ มันทำให้ TCP ฝั่งส่งกำหนดขนาดที่เหมาะสมของ window ได้อย่างรวดเร็ว แต่ในกรณีที่เป็นการเปลี่ยนแปลงของสภาพความแออัดในเครือข่ายอย่างรวดเร็ว ไม่ว่าในตอนนั้นจะก่อนหรือหลัง cwnd มีค่าสูงสุดที่ได้รับจากฝั่งตรงข้าม การเกิด segment สูญหายเป็นสัญญานบอกว่ามีสภาพแออัด แต่ไม่มีความชัดเจนว่าแออัดแค่ไหน จากนั้นจะทำการตั้ง cwnd เป็น 1 ใหม่ และเริ่มต้นเข้าสู่ขั้นตอน slow-start ใหม่

วิธีที่กล่าวข้างต้นถือว่าใช้ได้แต่ยังไม่ดีพอ Jacobson กล่าวว่า "ง่ายมากที่จะทำให้เครือข่ายอิ่มตัว แต่ยากมากที่จะทำให้กลับสู่สภาวะปกติ" พูดได้อีกอย่างหนึ่งว่า เมื่อเกิดสภาพความแออัดแล้วอาจจะต้องใช้เวลานานพอสมควรถึงจะกลับเข้าสู่สภาวะปกติ ซึ่งการเติบโตของ cwnd แบบ exponential ภายใต้กลไก slow-start อาจจะเป็นตัวการให้เกิดความแออัด ดังนั้น Jacobson จึงได้เสนอให้ใช้ slow-start เฉพาะตอนเริ่มต้น และใช้การเพิ่ม cwnd แบบเชิงเส้นหลังจากนั้น เมื่อ time out เกิดขึ้นจะเกิดกระบวนการดังนี้

- ตั้งขนาดของ slow-start threshold ให้เป็นครึ่งหนึ่งของขนาดของ congestion window ปัจจุบัน ดังนี้ ssthresh = cwnd/2
- ตั้ง cwnd = 1 และเริ่มต้นกลไก slow-start จนกระทั่ง cwnd = ssthresh (ตอนนี้ cwnd จะเพิ่มทีละ 1 ต่อแต่ละ ACK ที่ได้รับ)
- เมื่อ cwnd ≥ssthresh การเพิ่มของ cwnd จะเพิ่มแค่ทีละ 1 ต่อแต่ละ round-trip time

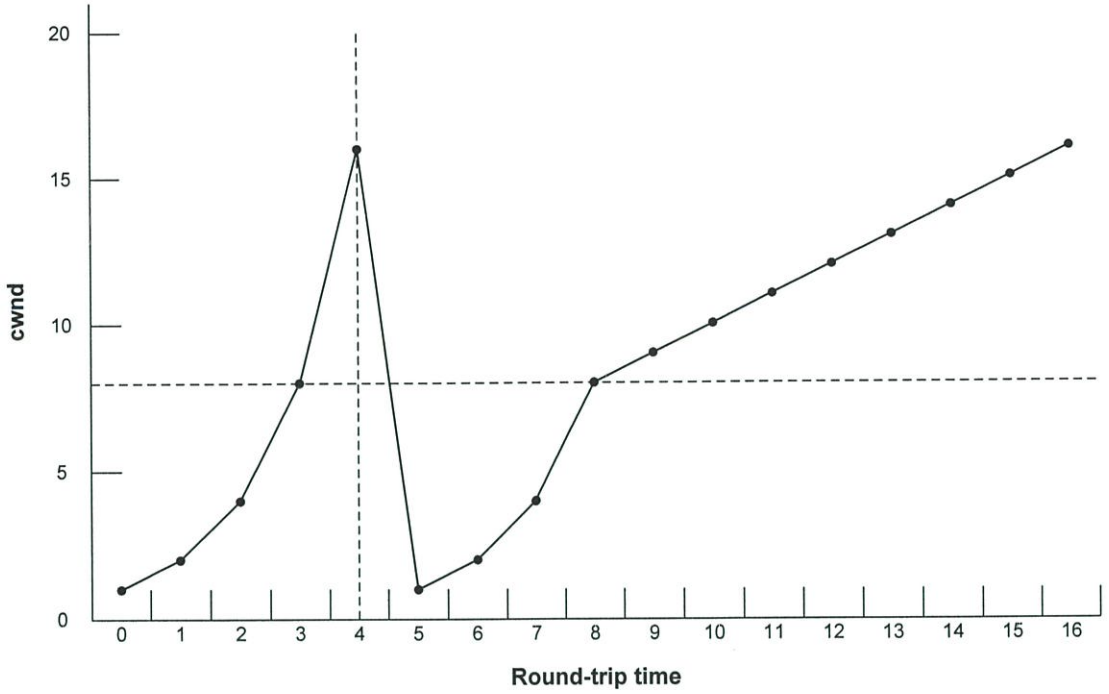


a) Slow start ที่ตอนสุดท้ายมี time out

b) Slow start และ congestion avoidance

รูปที่ 3.7 Slow start และ Congestion avoidance

รูปที่ 3.7 แสดงการทำงานของ slow-start รูปที่ 3.7a แสดงการทำงานซ้ำกับรูปที่ 3.6 แต่เพิ่ม segment สุดท้ายที่ส่งแล้วเกิด time out พฤติกรรมหลังจาก time out แสดงในรูป 3.7b ค่าของ ssthresh ถูกตั้งไว้ที่ 8 ก่อนจะถึงค่า threshold นี้ TCP จะใช้ exponential slow-start ในการขยายตัวของ window แต่จากนั้น cwnd จะเพิ่มแบบเป็นเชิงเส้น พฤติกรรมนี้แสดงอย่างชัดเจนในรูปที่ 3.8 TCP ใช้ 11 round-trip time ในการกู้สถานการณ์ให้กลับมายังจุดสูงสุดของ cwnd ในขณะที่ตอนเริ่มต้นใช้เพียง 4 round-trip time



รูปที่ 3.8 ขนาดของ cwnd ในระหว่าง Slow start และ Congestion avoidance

3.3.3.3 Fast retransmit

Retransmission timer (RTO) ถูกใช้โดย TCP ฝั่งส่งเพื่อกำหนดเวลาในการส่ง segment ซ้ำ นั้นโดยปกติจะมีค่ามากกว่าเวลา round-trip time จริงๆมาก กลไกของทั้งจาก RFC 793 และ Jacobson ต่างระบุให้ค่า RTO มากกว่าค่าประเมิน SRTT ด้วยเงื่อนไขเหล่านี้ทำให้ RTT ที่ประเมินไม่ถูกต้อง

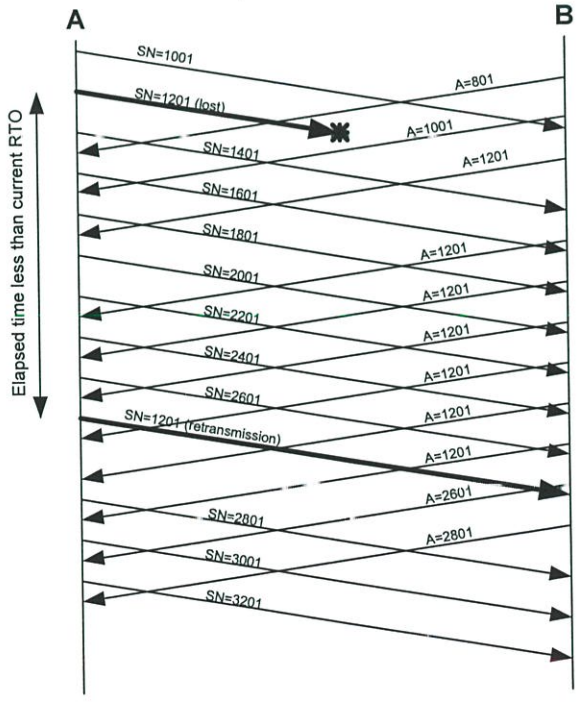
- RTO ถูกคำนวณบนพื้นฐานการคาดเดา RTT ถัดไป ซึ่งเป็นการประเมินจากค่าที่ผ่านไปแล้วของ RTT ถ้าความหน่วงเพิ่งเกิดขึ้น ค่าประเมินของ RTT จะน้อยกว่า RTT จริงๆ
- ในทำนองเดียวกัน ถ้าความหน่วงที่ฝั่งรับแกว่งตัวขึ้นลงอย่างรวดเร็ว ค่าประเมินของ RTT จะยิ่งไม่น่าเชื่อถือ
- ระบบปลายทางอาจไม่ตอบ ACK ที่แต่ละ segment แต่จะสะสมเพื่อตอบ ACK ทีละหลายๆ segment หรือรอจนมีข้อมูลที่จะส่งออกแล้วส่งไปพร้อมกัน ทำให้เกิดการแกว่งตัวของค่า RTT

จากสาเหตุเหล่านี้ ถ้า segment สูญหาย TCP อาจจะช้าที่จะส่งซ้ำ ถ้า TCP ปลายทางใช้ In-order accept policy จะทำให้ segment จำนวนมากสูญหาย แม้จะใช้ In-window accept policy การส่งซ้ำที่ช้าก็ยังเป็นปัญหา เพื่อให้เห็นได้ชัดเจน สมมติว่า A ส่ง segment จำนวนหนึ่ง แล้ว segment แรกในจำนวนนี้ได้สูญหายไป แต่ในขณะนั้น send window ยังไม่เต็ม และ RTO ยังไม่หมดเวลา A จะยังคงส่งต่อไปโดยไม่ต้องรอรับ ACK ต่อมา B ได้รับทุกๆ segment ยกเว้น segment แรก แต่ B

ต้องเก็บทุก segment ที่เข้ามาลง buffer จนกระทั่งได้รับการส่งซ้ำ segment ที่หายไป B ไม่สามารถยกเลิก buffer ของตัวเองได้จนกว่าจะได้รับ segment ที่หายไป ถ้าการส่งซ้ำของ segment ที่หายไปใช้เวลานาน B จะมี buffer ไม่พอที่จะเก็บ segment ที่เข้ามา ทำให้เกิดการคัด segment ที่เกินทิ้ง

Jacobson เสนอ 2 วิธีการเรียกว่า Fast retransmit และ Fast recovery ที่ภายใต้เงื่อนไขบางอย่างจะมีประสิทธิภาพดี Fast retransmit ใช้ประโยชน์ของกฎเกณฑ์ใน TCP ที่ว่า ถ้า TCP ได้รับ segment ผิดลำดับ TCP จะส่ง ACK สำหรับ segment สุดท้ายที่ได้รับที่ยังเรียงลำดับออกไปทันที TCP จะยังคงส่ง ACK นี้ซ้ำในทุกๆ segment ที่เข้ามา จนกระทั่ง segment ที่หายไปนั้นเข้ามาเติมช่องโหว่ใน buffer เมื่อช่องโหว่ถูกเติมเต็ม TCP จะส่ง ACK แบบสะสมสำหรับทุก segment ที่ได้รับที่เรียงลำดับ

เมื่อ TCP ต้นทางรับ ACK ที่ซ้ำกัน จะตีความว่า 1) segment ที่ระบุใน ACK ถูกหน่วงทำให้ไปถึงผิดลำดับ 2) segment นั้นหายไป ในกรณีที่1 segment ไปถึงในที่สุดดังนั้น TCP ไม่ควรส่งซ้ำ แต่ในกรณีที่2 การตอบ ACK ที่ซ้ำกัน เป็นการเตือนระบบต้นทางว่า segment สูญหายให้ส่งซ้ำใหม่ เพื่อให้แน่ใจว่าเป็นกรณีที่2 เท่านั้น Jacobson แนะนำว่า TCP ฝั่งส่งจะต้องรอจนได้รับ ACK ซ้ำ 3 ครั้งที่ segment เดิม (รวมทั้งหมดคือ 4 ACK ที่ segment เดิม) ภายใต้เงื่อนไขนี้เหมือนกับว่า segment ที่หายไปถูกส่งซ้ำอย่างทันทีแทนการรอให้หมดเวลา



รูปที่ 3.9 Fast retransmit

รูปที่ 3.9 แสดงการทำงานของ fast retransmit A ส่ง segment ขนาด segment ละ 200 byte จำนวนหนึ่ง segment ที่ 1201 ได้หายไป แต่โดยปกติ A จะยังไม่รู้จนกระทั่ง RTO หมดเวลา

ดังนั้น A จึงทำการส่ง segment ต่อไปจนหมด window B ได้รับ segment 1001 และตอบด้วย ACK 1201 จากนั้น B ได้รับ segment 1401 แต่ว่า segment นี้ผิดลำดับ B จะส่ง ACK 1201 ซ้ำ และจะยังคงส่งซ้ำในแต่ละ segment ที่เข้ามาจนกว่า segment 1201 จะเข้ามา ที่เวลาที่ A ได้รับ ACK 4 ครั้ง A ได้ส่ง 7 segment ที่เกิน 1201 ไปแล้ว A จะส่งซ้ำ segment 1201 ทันที และจะกลับไปส่ง segment ที่ค้างไว้ต่อไป

A สามารถสมมติว่า segment ที่ตามหลัง 1201 ได้รับแล้ว เพราะว่าถ้า B ไม่ได้รับ segment เหล่านั้น ก็จะไม่เกิด ACK ซ้ำตอบออกมา

3.3.3.4 Fast recovery

เมื่อ TCP ทำการส่ง segment ซ้ำโดยใช้ fast retransmit TCP รู้ (หรืออาจจะสมมติ) ว่า segment นั้นหายไป ทั้งที่ยังไม่หมดเวลารอตอบรับ segment นั้น ดังนั้น TCP ควรจะหลีกเลี่ยงความแออัดที่เกิดขึ้น วิธีการหนึ่งที่ใช้ได้คือ slow-start / congestion avoidance ที่ถูกใช้เมื่อเกิด time out นั่นคือ ตั้งค่าให้ ssthresh = cwnd/2 และให้ cwnd = 1 และเริ่มต้นขั้นตอน exponential slow-start จนกระทั่ง cwnd = ssthresh และต่อไปจะเพิ่ม cwnd แบบเชิงเส้น Jacobson กล่าวว่าวิธีนี้ยังไม่ดีพอในกรณีที่มี ACK ซ้ำตอบมามากๆ ดังนั้นจึงเสนอกลไก fast recovery ที่มีหลักการทำงานดังนี้

1. เมื่อได้รับ ACK ซ้ำครั้งที่ 3
    a. ตั้งค่า ssthresh = cwnd/2
    b. ส่งซ้ำ segment ที่หายไป
    c. ตั้งค่า cwnd = ssthresh+3

เหตุผลของการบวก 3 เข้ากับ ssthresh คือเป็นจำนวนของ segment ที่ส่งไปยังปลายทางแล้ว

2. แต่ละครั้งที่ได้รับ ACK ซ้ำเพิ่ม (ที่ segment เดิม) ให้เพิ่ม cwnd ขึ้นอีก 1 และส่ง segment ข้อมูลต่อไปถ้ามี การบวกนี้เพราะรู้ว่ามีจำนวน segment ที่ส่งไปและได้รับแล้วเพิ่มอีก 1
3. เมื่อได้รับ ACK ถัดไปที่เป็นการตอบรับ segment ใหม่ ให้ตั้งค่า cwnd = ssthresh

3.4 สรุป

จากเนื้อหาในบทนี้ทำให้เราเข้าใจถึงกลไก flow control และ congestion control ที่ใช้ในการตรวจจับสภาวะในเครือข่ายและควบคุมอัตราการส่งข้อมูลด้วยตัว TCP เอง แต่เนื่องจาก Internet ไม่ได้กำหนดมาตรฐานตายตัวลงไปทำให้วิธีการที่เลือกใช้และประสิทธิภาพของ TCP ของแต่ละผู้พัฒนาหรือแต่ละเวอร์ชันมีประสิทธิภาพแตกต่างกันออกไป ในบทต่อไปเราจะแนะนำเครื่องมือในการจำลองสภาพเครือข่ายซึ่งสามารถนำมาใช้ทดสอบวิธีการต่างๆที่นำเสนอในบทนี้ได้

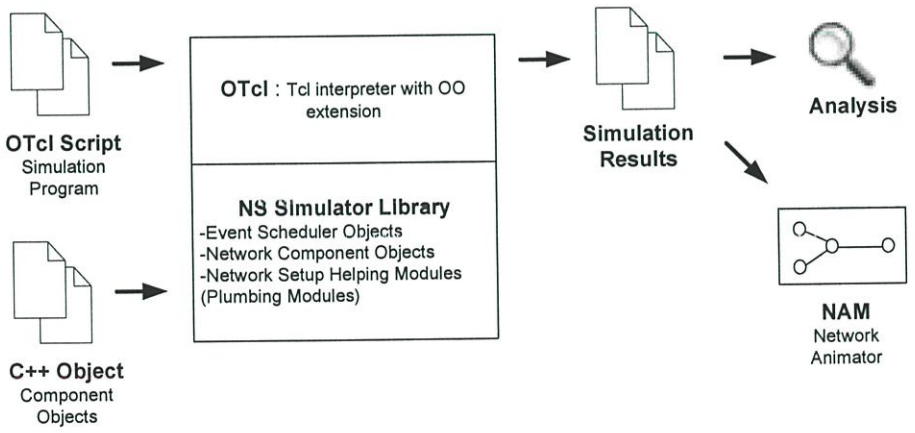
บทที่ 4

Network Simulator 2 (NS-2)

Network Simulator 2 (NS-2) เป็นโปรแกรมจำลองการทำงานของระบบเครือข่าย (simulator) แบบ ตามเหตุการณ์ (event driven) พัฒนาขึ้นมาโดยนักวิจัยจาก Lawrence Berkeley Laboratory ที่ University of California at Berkeley เขียนด้วยภาษา C++ และ OTcl สามารถจำลอง IP network ได้หลากหลายรูปแบบทั้ง LAN และ WAN อีกทั้งสามารถจำลองการทำงานของ protocol ต่างๆได้ทั้ง TCP, UDP และยังสามารถสร้าง traffic ที่ส่งเข้ามาในเครือข่ายจำลองได้ทั้งแบบ FTP, Telnet, Web, CBR และ VBR ในส่วนของ queue management มีกลไกให้เลือกทั้งแบบ Drop-Tail, RED และ CBQ นอกจากนี้ NS-2 ยังเปิดโอกาสให้มีการพัฒนาเพิ่มเติมความสามารถเข้าไปได้ด้วยซึ่งมีความยืดหยุ่นต่อการใช้งาน

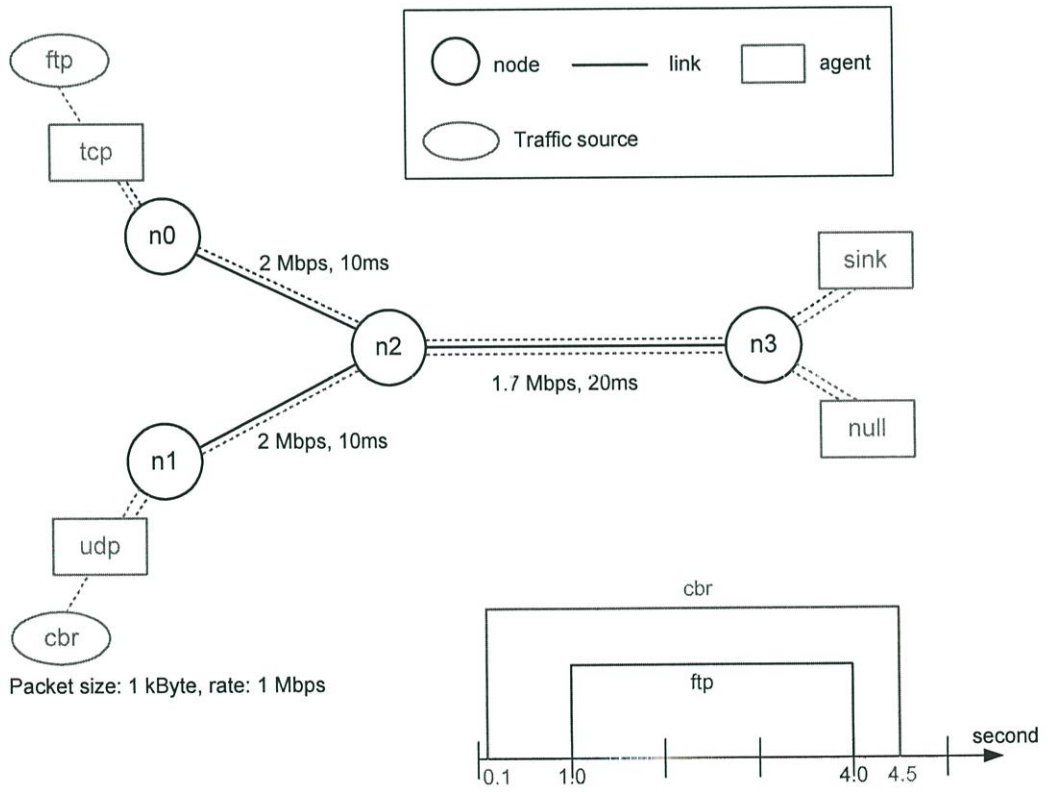
4.1 โครงสร้างของ NS-2

ดังแสดงในรูป 4.1 ในมุมมองของผู้ใช้งาน NS-2 เป็นตัวแปรภาษา Object-oriented Tcl (OTcl) ที่มีการจัดการจำลองเหตุการณ์ (simulation event scheduler), ส่วนประกอบของเครือข่ายแบบ object library และการตั้งค่าเครือข่าย พูดได้ว่าการใช้ NS-2 คือการเขียนโปรแกรมภาษา OTcl เพื่อตั้งค่าและ run เครือข่ายจำลอง ผู้ใช้ต้องเขียน OTcl script เพื่อกำหนด event scheduler, กำหนดการเชื่อมต่อกันของแต่ละส่วนประกอบ (topology setup) และ กำหนดเวลาเริ่มต้นสิ้นสุดให้กับ traffic ที่จะเดินทางเข้าไปในเครือข่ายจำลองผ่าน event scheduler



รูปที่ 4.1 มุมมองของผู้ใช้งานต่อ NS-2

ส่วนประกอบหลักของ NS-2 ที่อยู่เบื้องหลัง network object คือ event scheduler เหตุการณ์ใน NS-2 คือ Packet ID ที่ไม่ซ้ำกันเป็นการแสดงถึง packet ที่ถูกกำหนดเวลาไว้แล้วและตัวชี้ไปยัง object ที่ดูแลเหตุการณ์นี้โดยตรง ใน NS-2 event scheduler เก็บข้อมูลของเวลาในการจำลองและทำการเคลื่อนไหวทุกเหตุการณ์ใน event queue เมื่อถึงเวลาที่เหมาะสมกับเหตุการณ์นั้น Network component (อุปกรณ์เครือข่าย) จะสื่อสารกับอุปกรณ์ตัวอื่นเพื่อการส่งผ่าน packet ระหว่างกัน ซึ่งงานประเภทนี้ใช้เวลาในการจำลองไม่มากนัก ทุกๆ network component จำเป็นจะต้องใช้การควบคุมเวลาเพื่อจัดการกับ packet (เช่น เวลาในการ delay) ที่ต้องอาศัยความช่วยเหลือจาก event scheduler ตัวอย่างเช่น network switch ตัวหนึ่งจำลองเป็น switch ที่มีค่า delay เป็น 20 microsecond ร้องขอเหตุการณ์สำหรับ packet จาก scheduler หลังจากเวลานั้น 20 microsecond เมื่อครบเวลา 20 microsecond scheduler จะเอาเหตุการณ์ออกจาก queue และส่งให้กับอุปกรณ์เครือข่ายตัวนี้ ซึ่งจะส่ง packet ต่อไปยังอุปกรณ์ output link



รูปที่ 4.2 ตัวอย่างการทำ simulation ของ network topology ง่ายๆ

NS-2 ไม่ได้ถูกเขียนขึ้นมาด้วย OTcl อย่างเดียว แต่ยังเขียนด้วย C++ ด้วย เหตุผลที่สำคัญคือเรื่องของประสิทธิภาพ NS-2 แยก data path และ control path ออกจากกัน เพื่อลดเวลาสำหรับการประมวล packet และ event (เวลาที่ต้องใช้จริงไม่ใช่เวลาที่จำลอง) event scheduler และ network component พื้นฐานจะถูกเขียนขึ้นมาด้วย C++ object ที่ได้จาก C++ สามารถถูกเรียกใช้

โดยตัวแปรภาษา OTcl ผ่านตัวเชื่อม OTcl ที่จะทำหน้าที่จับคู่ระหว่าง OTcl object กับ C++ object และทำให้การควบคุมและการตั้งค่าตัวแปรใน C++ object เหมือนกับทำกับ OTcl object

4.2 ตัวอย่างการทำ simulation อย่างง่าย

หัวข้อนี้จะแสดงตัวอย่างของ NS-2 simulation script ง่ายๆพร้อมทั้งคำอธิบาย ตัวอย่างนี้เป็น OTcl script ที่เขียนเพื่อสร้างเครือข่ายง่ายๆและจำลองสถานการณ์ในรูปที่ 4.2

ในเครือข่ายนี้ประกอบไปด้วย 4 node (n0, n1, n2, n3) ดังแสดงในรูปที่ 4.2 Link ที่ใช้เป็นแบบ Duplex ถูกนำมาเชื่อมระหว่าง n0 กับ n2 และ n1 กับ n2 มี bandwidth 2 Mbps และ delay 10 ms Duplex link อีกเส้นหนึ่งนำมาเชื่อมระหว่าง n2 และ n3 มี bandwidth 1.7 Mbps และ delay 20 ms แต่ละ node ใช้ queue แบบ Drop-Tail ที่มีขนาด 10 packet "tcp" agent ถูกนำมาต่อกับ n0 เพื่อให้เกิดการสื่อสารกับ "sink" agent ที่ต่อกับ n3 เราใช้ค่า default กับขนาดสูงสุดของ packet ที่ "tcp" agent สามารถส่งออกคือ 1 kByte "sink" agent จะสร้างและส่ง ACK กลับมาที่ผู้ส่ง ("tcp" agent) "udp" agent ที่ต่อกับ n1 กำหนดให้สื่อสารกับ "null" agent ที่ต่ออยู่กับ n3 "null" agent จะรับ packet แล้วปล่อยหายไป "ftp" และ "cbr" ที่เป็นตัวสร้าง traffic ต่ออยู่กับ "tcp" และ "udp" ตามลำดับ และ "cbr" ถูกกำหนดให้สร้าง traffic ที่มี packet ขนาด 1 kByte ที่อัตรา 1 Mbps "cbr" ถูกกำหนดให้เริ่มที่เวลา 0.1 sec และหยุดที่ 4.5 sec ส่วน "ftp" เริ่มที่ 1.0 sec และหยุดที่ 4.0 sec

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

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

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

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
```

```
    #Close the NAM trace file
    close $nf
    #Execute NAM on the trace file
    exec nam out.nam &
    exit 0
}

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

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

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

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


#Setup a TCP connection
set tcp [new Agent/TCP]
```

```
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

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



#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

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



#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
```

```
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

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

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

อธิบาย script ตัวอย่างได้ดังนี้ โดยทั่วไป NS-2 script จะเริ่มต้นด้วย Simulator object instance

- set ns [new Simulator]: สร้าง NS-2 simulator object instance และกำหนดตัวแปร ns บรรทัดนี้มีไว้เพื่อ
  - เตรียมการรูปแบบ packet
  - สร้าง scheduler (default คือ calendar scheduler)
  - เลือกค่า default ของรูปแบบ address

Simulator object มีหน้าที่ดังต่อไปนี้

- สร้าง compound object เช่น node และ link
- เชื่อมต่อ network component object ที่ถูกสร้างขึ้น
- ตั้งค่าให้กับ network component
- สร้างการเชื่อมต่อระหว่าง agent (เช่น สร้างการเชื่อมต่อระหว่าง “tcp” และ “sink” ในตัวอย่าง)
- กำหนดตัวเลือกแสดงผลใน NAM

หน้าที่หลักส่วนใหญ่ คือ การตั้งค่าการทำ simulation และ scheduling แต่หน้าที่เช่นการตั้งค่าแสดงผลใน NAM ก็เป็นหน้าที่อีกประเภทหนึ่ง

- $ns color fid color: เป็นการกำหนดสีของ packet สำหรับการ flow ของ packet ที่ระบุโดย flow id (fid) หน้าที่นี้ของ Simulator object เพื่อการแสดงผลใน NAM อย่างเดียว จะไม่มีผลต่อการจำลองเครือข่าย
- $ns namtrace-all file-descriptor: เป็นการบอก Simulator ให้บันทึกเหตุการณ์ด้วยรูปแบบของ NAM และยังกำหนดชื่อแฟ้มที่จะให้เขียนทีหลังด้วยคำสั่ง $ns flush-trace หน้าที่นี้คล้ายกับหน้าที่ของ trace-all ที่ใช้บันทึกการจำลองด้วยรูปแบบปกติ
- proc finish {}: ถูกเรียกหลังการ simulation โดยคำสั่ง $ns at 5.0 “finish” ด้วยคำสั่งนี้จะทำให้เกิดกระบวนการที่ต้องทำหลังการจำลอง
- set n0 [$ns node]: เป็นการสร้าง node node ใน NS-2 เป็น compound object ที่ประกอบด้วย address และ port classifier แต่ผู้ใช้สามารถสร้าง node ที่มี address และ port classifier แยกกัน แล้วนำมาต่อกันอีกทีก็ได้
- $ns duplex-link node1 node2 bandwidth delay queue-type: สร้าง link แบบทางเดียว 2 เส้น ด้วย bandwidth และ delay ที่ระบุ และเชื่อมต่อกับทั้ง 2 node ที่ระบุ ใน NS-2 output queue ของ node ถูกพัฒนาให้เป็นส่วนหนึ่งของ link ดังนั้นผู้ใช้จะกำหนดชนิดของ queue เมื่อทำการสร้าง link ในตัวอย่างนี้ Drop-Tail queue ถูกนำมาใช้ แต่ถ้าเราต้องการใช้ RED queue ก็ทำการเปลี่ยนข้อความจาก “DropTail” ไปเป็น “RED” อย่างง่ายๆ NS-2 สร้าง link เหมือนสร้าง node link ก็เป็น compound object และผู้ใช้สามารถสร้าง sub-object และต่อเข้ากับ node ได้
- $ns queue-limit node1 node2 number: บรรทัดนี้กำหนดขนาดให้กับ queue ทั้ง 2 simplex link ที่ต่อ node1 เข้ากับ node2 ตามตัวเลขที่ระบุ
- $ns duplex-link-op node1 node2 …: บรรทัดนี้และอีก 2 บรรทัดที่ตามมา ถูกใช้เพื่อการแสดงผลของ NAM

ที่จุดนี้การสร้างพื้นฐานเครือข่ายเสร็จสิ้นแล้ว ต่อไปเราจะสร้าง traffic agent เช่น TCP และ UDP, traffic source เช่น FTP และ CBR และต่อเข้ากับ node ที่สร้างไว้แล้ว

- set tcp [new Agent/TCP]: บรรทัดนี้แสดงการสร้าง TCP agent แต่โดยทั่วไป ผู้ใช้สามารถสร้าง agent หรือ traffic source ด้วยวิธีนี้ agent และ traffic source เป็น object พื้นฐาน (ไม่ใช่ compound object) ส่วนใหญ่พัฒนาด้วย C++ และ link เข้ากับ OTcl ดังนั้นจึงไม่ต้องบอกให้ simulator object สร้าง object instance เพื่อสร้าง agent หรือ traffic source ผู้ใช้จำเป็นต้องรู้ชื่อ class ของ object เหล่านี้เพื่อเรียกใช้เอง (Agent/TCP, Agent/TCPSink, Application/FTP เป็นต้น) ข้อมูลเหล่านี้หาได้จากหนังสือ “ The ns Manual”

- $ns attach-agent node agent: attach-agent ทำการผูก agent object ที่สร้างเข้ากับ node object
- $ns connect agent1 agent2: หลังจาก 2 agent ที่จะสื่อสารกันถูกสร้างแล้ว ต่อไปคือการเชื่อมต่อกันทาง logical ระหว่าง agent ทั้ง 2

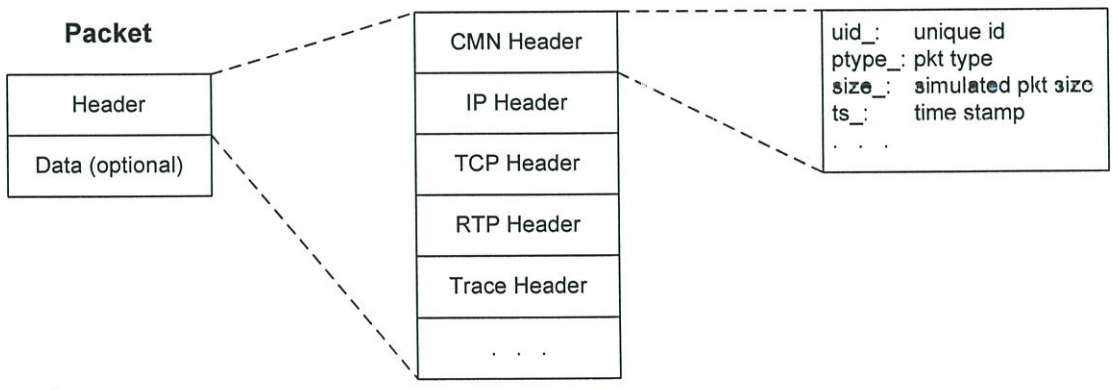
สมมติว่าการตั้งค่าเครือข่ายทั้งหมดเสร็จสิ้น ต่อไปจะต้องมากำหนดถึง สถานการณ์ที่จะให้เกิดขึ้นในเครือข่ายจำลองนี้

- $ns at time "string": เป็นการบอกให้ simulator object สร้าง scheduler เพื่อจัดการทำงานตามข้อความที่ระบุใน “string” ที่เวลาที่ระบุ เช่น $ns at 0.1 "$cbr start" จะเป็นการบอกให้ scheduler ทำการเรียกใช้ start ของ cbr traffic source object ซึ่งจะเริ่มการส่งข้อมูล cbr ใน NS ปกติแล้ว traffic source จะไม่ได้ส่งข้อมูลจริงๆ แต่จะแจ้งให้ agent ที่ผูกกันอยู่ว่ามีข้อมูลจำนวนหนึ่งที่จะส่ง ส่วน agent รู้เพียงแต่ว่ามีข้อมูลจะส่งมากแค่ไหน แล้วทำการสร้าง segment เพื่อจัดส่ง

หลังจากการตั้งค่าเครือข่าย, scheduling และกระบวนการหลังจบการจำลองแล้ว สุดท้ายคือการส่งคำสั่งให้คำสั่งที่เขียนไว้ทั้งหมดทำงานด้วย $ns run

4.3 NS-2 packet format

NS-2 packet ประกอบด้วยชั้น (stack) ของ header และตามด้วยพื้นที่ข้อมูล ดังรูปที่ 4.3 ตามที่กล่าวไว้ในตัวอย่างการ simulation ว่า packet header format ถูกกำหนดเริ่มต้น เมื่อ simulator object ถูกสร้างขึ้น ซึ่งเป็นชั้นของ header ที่ถูกกำหนดเฉพาะ header ที่มีการลงทะเบียนใช้งาน ชั้น header นี้ยังรวมถึง header ที่ถูกใช้งานโดย object ส่วนใหญ่ เช่น IP header, TCP header, RTP header (UDP ใช้ RTP header) และ Trace header โดย offset ของแต่ละ header ในชั้นจะถูกบันทึกไว้



รูปที่ 4.3 NS-2 packet format

ปกติแล้ว packet จะมี header (แต่ตัวชี้พื้นที่ข้อมูลว่างเปล่า) ดังนั้น packet สามารถส่งข้อมูลจริงๆ (จาก application) โดยการสร้าง data space ซึ่งมีบาง application และ agent พัฒนาเพื่อรองรับในส่วนนี้ เพราะว่ามันไม่ค่อยมีประโยชน์ที่จะส่งข้อมูลไปในการจำลองในเวลาจำลอง อย่างไรก็ตาม ถ้าเราต้องการพัฒนา application เพื่อส่งข้อมูลให้อีก application ข้ามเครือข่าย เราอาจจะต้องใช้ความสามารถนี้ด้วยการแก้ไขเล็กน้อยในส่วนของการพัฒนา agent วิธีการอื่นที่เป็นไปได้คือการสร้าง header ขึ้นมาใหม่สำหรับ application นี้และแก้ไข agent ให้เขียนข้อมูลที่ได้รับจาก application ลงใน header ที่กำหนดขึ้นใหม่

4.4 Trace format

จากตัวอย่างที่แล้วที่มีการสร้าง NAM trace file เพื่อใช้แสดงผลใน NAM เรานำมาเพิ่มเติมให้มีการสร้าง Trace file อีกแบบหนึ่งที่จะนำไปใช้ในการวิเคราะห์ต่อไป ส่วนที่เป็นสีเทาคือ code เก่า ส่วนสีเข้มคือ code ที่เพิ่มเข้าไป

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

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

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

```
    exit 0
}
```

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

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

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

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

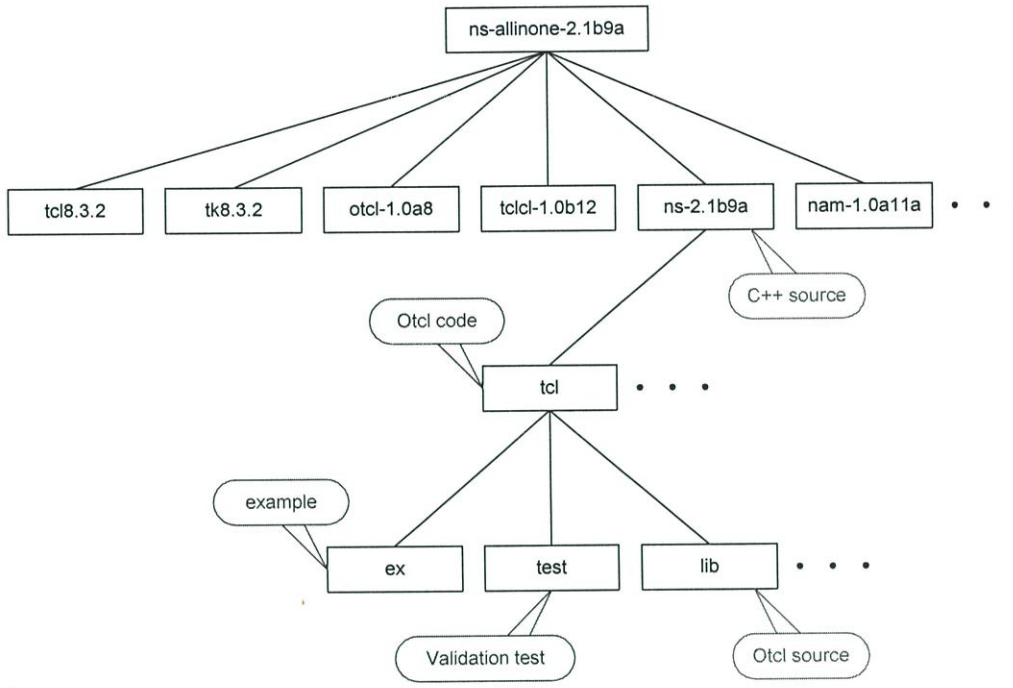
รูปที่ 4.4 ตัวอย่าง Trace format

เมื่อจบการ simulation แล้วเปิด file "out.tr" จะได้ Trace format ที่มีรูปแบบดังรูปที่ 4.4 แต่ละบรรทัดจะเริ่มต้นด้วย event (+, -, d, r) ตามด้วยเวลาจำลอง (มีหน่วยเป็นวินาที) และจาก node (from node) ไปยัง node (to node) ซึ่งจะเป็นตัวกำหนดเหตุการณ์ที่เกิดขึ้นบน link ต่อไปคือ packet type (pkt type) และ packet size (pkt size) มีหน่วยเป็น byte จากนั้นจะเป็น flag ถ้าเป็น "-------" แสดงว่าไม่มีการกำหนด flag ต่อไปคือ flow id (fid) ที่ใช้กำหนดสีให้แต่ละ flow เพื่อแสดงใน NAM ถัดไปเป็น address ต้นทาง (src addr) และ address ปลายทาง (dst addr) แสดงในรูปแบบ node.port ต่อไปคือ sequence number (seq num) ที่ใช้ใน network layer protocol สุดท้ายคือ packet id (pkt id) ที่ไม่ซ้ำกันในแต่ละ packet

4.5 โครงสร้าง directory ของ NS-2

ในหัวข้อนี้จะเป็นการศึกษาถึงโครงสร้าง directory และแฟ้มของ NS-2 ns-allinone เป็น software package ขนาดใหญ่ในรูปแบบการบีบอัดแบบ gz มีขนาดถึง 50MB ns-allinone package เป็นการรวมเอา package ย่อยๆที่จำเป็นต่อการใช้งานมารวมไว้ด้วยกัน ดังนี้ NS 2.1b9a, NAM 1.0a11a, Tcl 8.3.2, TclCL 1.0b12, Tk 8.3.2, OTcl 1.0a8, Zlib 1.1.3, Xgraph 12.1 เมื่อทำการแตก file ออกมาจะเห็นโครงสร้างดังแสดงในรูป 4.5 ภายใต้ directory "ns-allinone-2.1b9a" นั้น "ns-2.1b9a" เป็นที่รวมของ code ทั้งหมดของตัว simulator (ทั้ง C++ และ OTcl), validation test OTcl

script และ ตัวอย่างของ Otcl script ภายใต้ "tcl" อีกชั้นหนึ่งจะพบ OTcl code, ตัวอย่าง และ การทดสอบทั้งหมด ส่วน C++ code ส่วนใหญ่ที่เป็นการพัฒนา event scheduler และ object class ของ network component พื้นฐานจะอยู่ที่ directory หลัก เช่น ถ้าจะหา code การพัฒนาของ UDP agent ต้องไปที่ directory "ns-allinone-2.1b9a/"ns-2.1b9a" แล้วเปิดแฟ้ม "udp.h" และ "udp.cc"

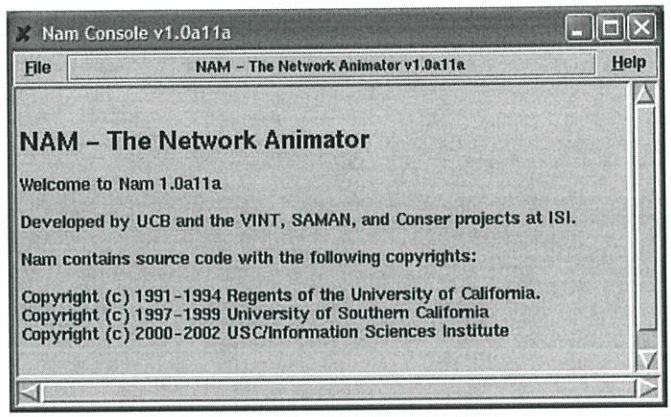


รูปที่ 4.5 โครงสร้าง directory ของ NS-2

ภายใต้ directory "tcl" จะประกอบด้วยกลุ่มของ directory ดังนี้ ภายใต้ "lib" เก็บ OTcl source code สำหรับพื้นฐานส่วนใหญ่ใน NS-2 (agent, node, link) ภายใต้ "ex" เป็นที่เก็บตัวอย่างที่หลากหลายให้ศึกษา และสุดท้ายภายใต้ "test" เป็นที่เก็บ code ในการตรวจสอบตัว NS-2 เอง หลังจากการติดตั้ง โดยทำการเปรียบเทียบค่าได้จากการทำงานจริงกับค่าที่ถูกต้อง

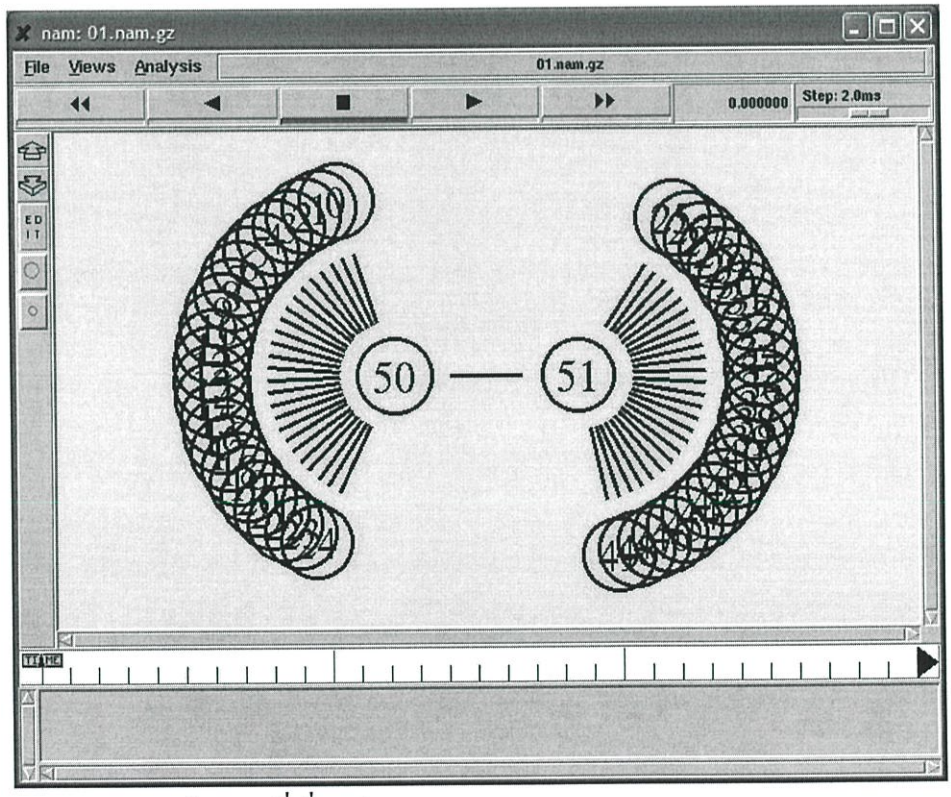
4.6 Network Animator (NAM)

หลังการทำ simulation จบสิ้นเราจะได้ output file จำนวนหนึ่ง (ขึ้นอยู่กับ script ที่เขียน) ที่ประกอบด้วยข้อมูลต่างๆที่เกิดขึ้นในขณะ simulation เราสามารถนำ output file นี้มาแสดงผลแบบ graphic ได้โดย NAM (Network Animator) ซึ่งเป็นโปรแกรมที่ผนวกมากับ NS-2 จะมีลักษณะการทำงานคล้ายกับ CD Player ในแง่ของการ play, forward, rewind, pause และอื่นๆ เพื่อสังเกตการณ์ทำงานของระบบที่เราสร้างขึ้น แต่ถ้าต้องการวิเคราะห์เชิงตัวเลขอย่างละเอียดแล้วละก็จำเป็นจะต้องเขียน script ขึ้นมาเป็นพิเศษเพื่อนำ Trace file ไปประมวลผลต่อ



รูปที่ 4.6 NAM Console

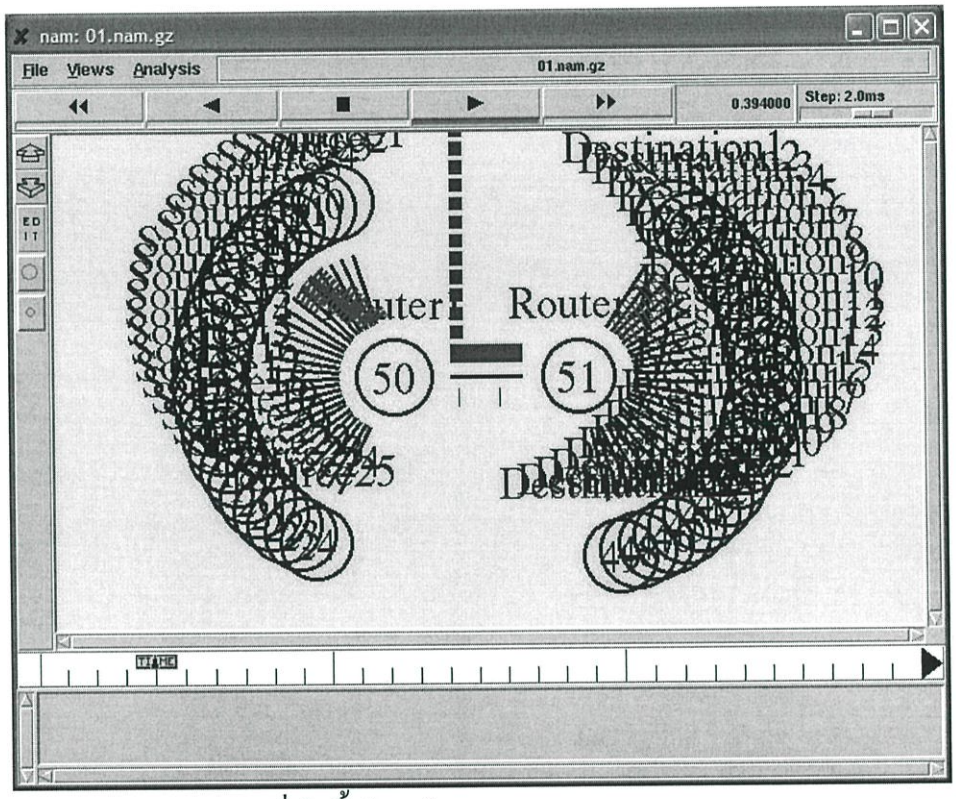
เมื่อเรียกใช้ NAM จาก command line ใน UNIX จะทำให้ NAM สร้าง window ขึ้นมา (ในรูป 4.6) เรียก window นี้ว่า NAM console ซึ่งจะเป็นตัวควบคุมการเรียกใช้ NAM trace file ที่ต้องการ



รูปที่ 4.7 NAM แสดงลักษณะที่เชื่อมต่อ (topology)

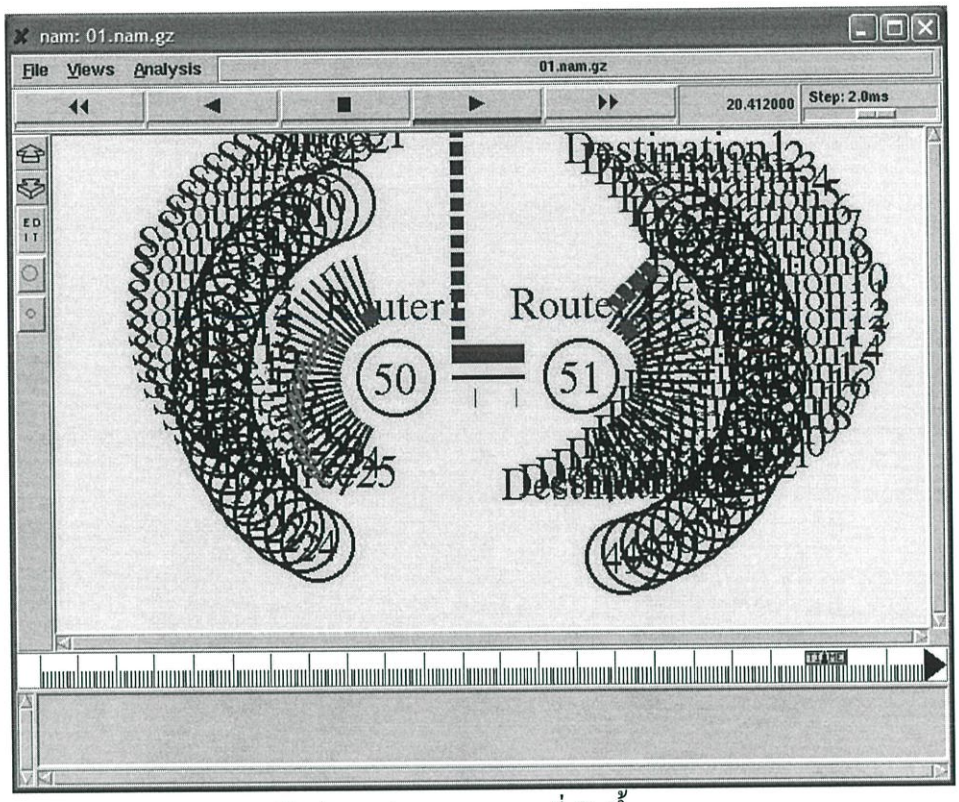
เมื่อทำการเปิด NAM trace file ที่ต้องการแล้ว NAM จะสร้าง window ที่ 2 ขึ้นมาพร้อมทั้งทำการวาด network topology ที่ถูกบันทึกอยู่ใน NAM trace file (แสดงในรูป 4.7 ชื่อ file คือ 01.nam.gz) วงกลมตรงกลางที่มีตัวเลข 50 และ 51 หมายถึง node ที่เป็น router ส่วนวงกลม

หลายๆวงที่วางซ้อนกันทางด้านซ้ายมือมีเลข 0 ถึง 24 และวงกลมที่วางซ้อนกันทางด้านขวามือมีเลข 25 ถึง 49 หมายถึง node ที่เป็นผู้ส่งและผู้รับตามลำดับ เส้นตรงที่เชื่อมแต่ละ node เข้าหากันคือ link



รูปที่ 4.8 NAM แสดง packet ที่เกิดขึ้นในแต่ละเวลา

จากนั้นเรากดปุ่ม play (เล่น) เหมือนกับการสั่งบน CD Player (โปรแกรมหรือเครื่องเล่น CD) ที่มีทั้ง เดินหน้า, ถอยหลัง, เดินหน้าเร็ว, ถอยหลังเร็ว และ หยุด เพื่อสั่งให้ NAM แสดงถึงการเคลื่อนที่ของ packet ตามผลจากการทำ simulation ด้วย NS-2 ในรูป 4.8 แสดงการ play จนถึงเวลา 0.394 วินาที แถบเส้นประสีน้ำเงิน(สีเข้ม)แสดงถึง packet ที่ไหลมาอย่างต่อเนื่อง เส้นประสีน้ำเงิน (สีเข้ม)ที่เป็นขีดเล็กๆที่แสดงใต้ link ที่เชื่อมระหว่าง node 50 กับ 51 คือ ACK ที่ตอบรับจากฝั่งรับ และเส้นประที่ต่อขึ้นไปในแนวตั้งที่ link ระหว่าง node 50 กับ 51 ใกล้กับ node 50 แสดงถึง packet ที่อยู่ใน queue เพื่อรอการจัดส่งไปใน link ในรูป 4.9 แสดงภาพที่เวลา 20.412 วินาที มีแถบเส้นประสีแดง(สีอ่อน)ที่แสดงถึง packet ที่เพิ่งถูกส่งออกมากำลังเดินทางอยู่ใน link ที่ต่อเชื่อมฝั่งส่งและ node 50 ที่เป็นของ traffic อีกกลุ่มหนึ่งที่เพิ่มเข้ามาในเครือข่าย



รูปที่ 4.9 NAM สามารถแสดงสีแต่ละกลุ่มของ packet ที่เกิดขึ้น

4.7 สรุป

เราได้แนะนำให้รู้จักกับโปรแกรมสร้างเครือข่ายจำลอง NS-2 ที่ช่วยให้การศึกษาการทำงานของเครือข่ายเป็นไปอย่างสะดวกสบาย และรวดเร็วขึ้นมาก เมื่อเทียบกับการหาอุปกรณ์ของจริงมาทำการทดสอบซึ่งบางกรณีอาจทำไม่ได้เลย เนื่องจาก NS-2 ได้เตรียม library ของ network component ที่เป็นที่นิยมเอาไว้ให้อย่างมากมาย และยังเปิดโอกาสให้ผู้ใช้นำมาปรับแต่งหรือเขียน component ของตัวเองขึ้นมาใหม่ทั้งหมดก็ได้ ในบทต่อไปเราจะนำเสนอกลไกแบบหนึ่งเพื่อช่วยเพิ่มประสิทธิภาพให้กับระบบเครือข่าย

บทที่ 5

นำเสนอวิธี Drop-Tail with ECN

จากกลไกของ TCP เพื่อตรวจสอบว่า packet ถูก drop ในระหว่างทาง จาก 2 เหตุการณ์คือ TCP sender ได้รับ ACK packet ข้ามลำดับ 3 ครั้ง หรือ สัญญาณ timeout จาก retransmit timer ซึ่งทั้ง 2 เหตุการณ์ต้องใช้เวลานานกว่าที่ TCP sender จะทราบว่าเกิด congestion ขึ้นใน link ระหว่างทางและทำการปรับตัวให้เหมาะสมกับสภาวะ network จริงๆได้

กลไก Explicit Congestion Notification (ECN) ได้ถูกนำเสนอขึ้นมาเพื่อแก้ปัญหาของ TCP ที่กล่าวมาข้างต้น โดยมีข้อดีดังนี้คือ ป้องกันการคัด packet ทิ้งโดยไม่จำเป็น ทำให้ไม่ต้องทำการส่ง packet เดิมซ้ำเป็นรอบที่สองหรือมากกว่านั้น ซึ่งช่วยลดปริมาณ traffic ใน network ให้ลดลง และ ทำให้ TCP sender รู้ว่ามีการเกิด congestion ได้เร็วขึ้น โดยไม่จำเป็นต้องรอจนกว่าจะได้รับ ACK packet ข้ามลำดับ 3 ครั้ง หรือ สัญญาณ timeout จาก retransmit timer

ECN ถูกนำเสนอให้ใช้งานร่วมกับ Random Early Detection (RED) เป็นอย่างแรก เพื่อให้ RED เป็นตัวกำหนดสัญญาณที่แสดงถึงสภาพแออัดให้อีกทีหนึ่ง แต่ในบทนี้เราจะนำเสนอวิธีการนำ ECN มาประยุกต์ใช้กับ Drop-Tail

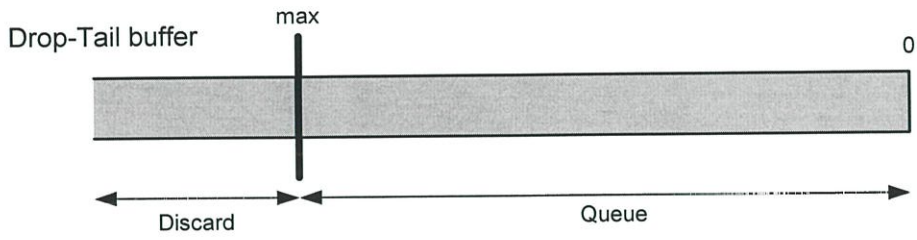
5.1 การจัดการ Queue

การจัดการ queue ที่จะพูดถึงนี้เป็นการจัดการพื้นที่ในการพักข้อมูล (เรียกว่า queue หรือ buffer) ที่อุปกรณ์ระหว่างทางของการส่งผ่านข้อมูล ที่เรียกว่า gateway หรือ router นั่นเอง โดยปกติแล้วเมื่อมี packet เข้ามาที่ link หนึ่ง เพื่อให้ router จัดส่งไปยังอีก link หนึ่ง router จะไม่จัดส่งให้อย่างทันที เนื่องจากเป็นวิธีที่ไม่มีประสิทธิภาพ แต่จะทำการเก็บเข้า queue ไว้ก่อน แล้วจึงค่อยนำออกจาก queue เมื่อมีความเหมาะสม ในวิทยานิพนธ์ฉบับนี้จะพูดถึงกลไกในการจัดการการนำเข้านำออก 2 ประเภทคือ Drop-Tail และ RED ดังต่อไปนี้

5.1.1 Drop-Tail

Queue แบบแรกที่จะพูดถึงนี้ใช้วิธีการง่ายๆ และ เป็นที่นิยมใช้กันโดยทั่วไป คือ Drop-Tail จากชื่อของ queue ก็ได้บ่งบอกถึงลักษณะได้อย่างชัดเจนว่า จะทำการคัดทิ้งจากส่วนหาง อธิบายการทำงานจากรูปที่ 5.1 ได้ว่าเมื่อมี packet เดินทางเข้ามา จะถูกจัดการแบบ FIFO (First In First Out หรือ เข้าก่อนออกก่อน) คือ ส่วนจัดการจะตรวจสอบว่ามีพื้นที่พอหรือไม่ (packet ที่เก็บอยู่ใน queue มีจำนวนถึงค่า max หรือไม่) ถ้าสามารถจัดเก็บได้ก็จะนำเข้าไปใส่ต่อท้ายใน queue แต่ถ้า queue เต็ม (packet ที่เก็บไว้เต็ม max) ก็จะคัด packet นั้นทิ้งไป ส่วนการนำเอา packet ไปใช้จะนำ

ออกจากตำแหน่ง 0 ก่อน (ส่วนหัว) packet ที่อยู่ตำแหน่งอื่นจะขยับมาทางส่วนหัว 1 ตำแหน่ง และจะมีที่ว่างใน queue เพิ่มอีกสำหรับ 1 packet



รูปที่ 5.1 การทำงานของ queue แบบ Drop-Tail

5.1.2 Random Early Detection (RED)

อีกวิธีการหนึ่งที่ถูกนำมาใช้จัดการกับความแออัดใน Internet คือวิธีการแบบที่เป็นการคัด packet ทิ้งแบบ proactive (คัดทิ้งก่อนที่ buffer จะเต็ม) เพื่อเป็นการปรับปรุงประสิทธิภาพของเครือข่าย วิธีการนี้ออกแบบมาให้ใช้กับ queue แบบ FIFO แบบ queue เดียว และสามารถถูกเลือกนำมาใช้กับ queue ใน router แบบต่างๆได้เพื่อความยืดหยุ่นต่อ traffic ที่เกิดขึ้น

จากเหตุที่ว่า เมื่อมีปริมาณ traffic จำนวนมากเกิดอย่างทันทีทันใดในเครือข่าย buffer ของ router จะถูกใช้งานจนเต็มและจากนั้นจะคัดทิ้ง packet ที่ไม่สามารถจัดเก็บลง buffer ได้ แต่ผลกระทบที่สำคัญต่อ TCP คือเป็นสัญญาณเข้าสู่ congestion control ซึ่งจะเป็นการปรับลดปริมาณการจัดส่งข้อมูลลงเพื่อแก้ปัญหาความแออัด

แต่ก่อให้เกิด 2 ปัญหาตามมาคือ อย่างแรก packet ที่หายไปต้องถูกจัดส่งซ้ำใหม่ ซึ่งเป็นการเพิ่มปริมาณข้อมูลให้กับเครือข่าย และยังเป็นการเพิ่มความหน่วงให้กับข้อมูลที่ส่งด้วย อย่างที่สองคือ ปรากฏการณ์ที่เรียกว่า “global synchronization” ด้วยปริมาณข้อมูลที่มากๆที่เกิดขึ้น buffer จะถูกใช้จนเต็มอย่างรวดเร็วและเกิดการคัด packet ทิ้งจำนวนมาก ทำให้ TCP connection จำนวนมากเข้าสู่ congestion control พร้อมๆกันจำนวนมาก ผลที่ตามมาคือ ปริมาณ traffic ในเครือข่ายมีปริมาณต่ำกว่าที่เครือข่ายรองรับอย่างมาก (underutilization)

วิธีแก้ปัญหาอย่างง่ายๆอย่างหนึ่งคือการเพิ่มขนาดของ buffer ให้มากๆเพื่อลดปริมาณการคัด packet ทิ้ง แต่ไม่ใช่วิธีที่ดีจาก 2 ประเด็นนี้ อย่างแรกคือ buffer ขนาดใหญ่ก่อให้เกิดความหน่วงที่มากตามขนาดที่เพิ่มขึ้นของ buffer และที่แย่คือถ้าปริมาณ traffic ที่มากขึ้นเรื่อยๆก็จะทำให้เกิด buffer ไม่พอเช่นเดิม

วิธีที่ดีกว่าคือ เมื่อเริ่มอยู่ในสภาพแออัดให้ส่งสัญญาณบอก TCP connection ทีละ 1 connection เพื่อให้ลดปริมาณการส่งลง และทำการตรวจดูความแออัดหลังจากนั้น ถ้าจำเป็นจะส่ง

สัญญาณให้อีก 1 TCP connection ทำการลดปริมาณการส่งลง และตรวจดูความแออัดอีกครั้ง ทำอย่างนี้ซ้ำๆกันจนกว่าจะสามารถควบคุมความแออัดในเครือข่ายได้

อธิบายกลไก RED ได้ว่า RED จะทำการคำนวณหาค่าเฉลี่ย (average) ของขนาด queue โดยการใช้ low-pass filter ร่วมกับ exponential weighted moving average จากนั้นจะนำมาเปรียบเทียบกับ 2 threshold คือ minimum threshold และ maximum threshold ถ้าค่า average ของขนาด queue น้อยกว่า minimum threshold จะไม่เกิดการ drop packet ถ้าค่า average ของขนาด queue มากกว่าหรือเท่ากับ maximum threshold จะทำการ drop ทุกๆ packet แต่ถ้าค่า average ของขนาด queue มากกว่าหรือเท่ากับ minimum threshold และ น้อยกว่า maximum threshold จะทำการสุ่ม drop packet ด้วยความน่าจะเป็น p_a ที่คำนวณได้

สามารถแสดงในรูป Algorithm ได้ดังนี้

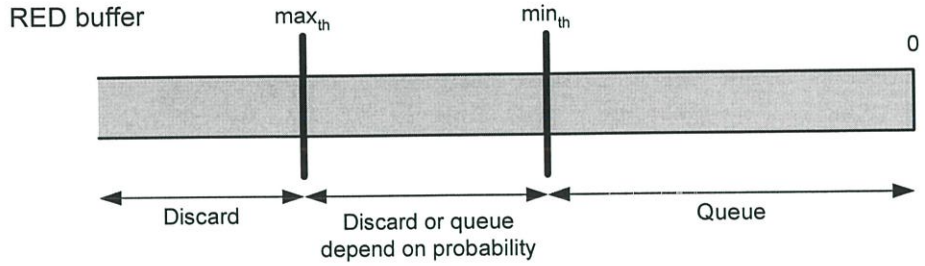
```
for each packet arrival do
    calculate the average queue size avg
    if min_th ≤avg < max_th
        calculate probability p_a
        with probability p_a
            drop the arriving packet
    else
        if max_th ≤avg
            drop the arriving packet
        else
            enqueue the packet
```

โดย

avg:	เป็นค่าเฉลี่ยขนาด queue จากการคำนวณ
min_{th}:	เป็นค่า threshold ต่ำสุดที่กำหนด
max_{th}:	เป็นค่า threshold สูงสุดที่กำหนด
p_a:	เป็นค่าความน่าจะเป็นจากการคำนวณ

อธิบายการทำงานจากรูปที่ 5.2 ได้ว่าเมื่อมี packet เดินทางเข้ามา จะเกิดการคำนวณหาค่า average ของขนาด queue และเปรียบเทียบค่าที่ได้ว่าอยู่ในเงื่อนไขใดจาก 3 เงื่อนไข ที่มี min_{th} และ

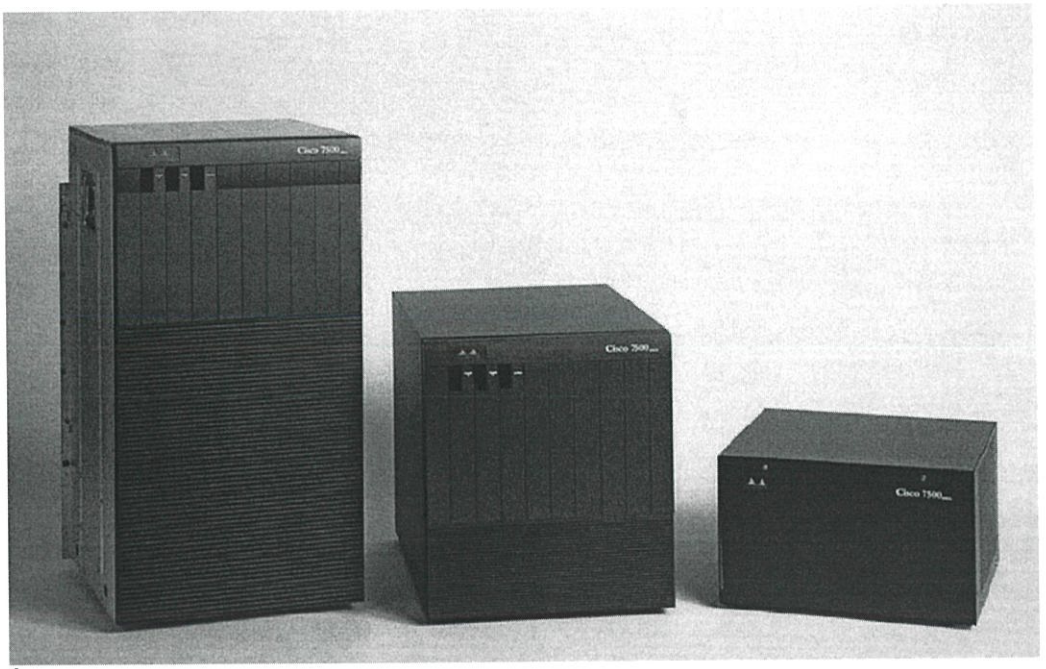
max_{th} เป็นตัวแบ่ง เพื่อจะ 1) เข้า queue ทันที 2) ใช้ความน่าจะเป็นช่วยว่าจะ drop หรือ เข้า queue หรือ 3) drop ทันที



รูปที่ 5.2 การทำงานของ queue แบบ RED

ตัวอย่างของการนำ RED ไปประยุกต์ใช้งานในผลิตภัณฑ์ทางธุรกิจ (commercial product) ได้แก่ Cisco IOS 11.2 GS ที่ใช้กับ core router รุ่น 12000 (รูปที่ 5.3) และ Cisco IOS 12.0 ที่ใช้กับ router รุ่น 7500 (รูปที่ 5.4) โดย Cisco จะเรียกว่า Weighted RED (WRED) [10] และ Distributed Weighted RED (dWRED) [11]

รูปที่ 5.3 Router ในตระกูล 12000 ของ Cisco

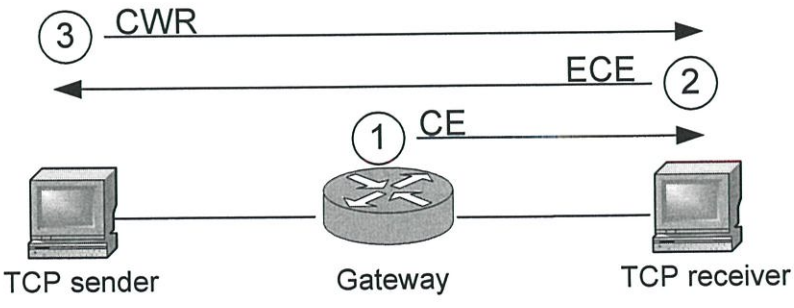


รูปที่ 5.4 Router ในตระกูล 7500 ของ Cisco

5.2 การทำงานของ Explicit Congestion Notification (ECN)

การส่งผ่านข้อมูลของ ECN นั้นเป็นการส่งข้อมูลแจ้งสภาพแออัด (congestion) ในเครือข่ายแทรกลงไปใน TCP/IP header เลยเพื่อไม่ให้เกิด overhead เพิ่มในระบบเครือข่าย จึงได้มีการกำหนด 4 bit ใน IP header และ TCP header เพื่อแสดงถึงข้อมูลของ ECN ในการสนับสนุนดังนี้ ใน IP header มีการกำหนด 2 bit คือ ECN-Capable Transport (ECT) เพื่อแสดงว่ามีการรองรับ ECN ทั้งต้นทางปลายทาง และ Congestion Experienced (CE) ถูก set จาก gateway เพื่อแสดงว่าเกิด congestion ส่วนใน TCP header มีการกำหนด 2 bit คือ ECN Echo (ECE) เพื่อส่งสถานะ congestion กลับไปยัง TCP sender และ Congestion Window Reduced (CWR) เพื่อบอกว่า TCP sender ได้มีการรับรู้ว่าเกิด congestion ขึ้นแล้ว

สามารถแสดงการทำงานของ ECN ได้ 3 ขั้นตอนดังแสดงในรูปที่ 5.5

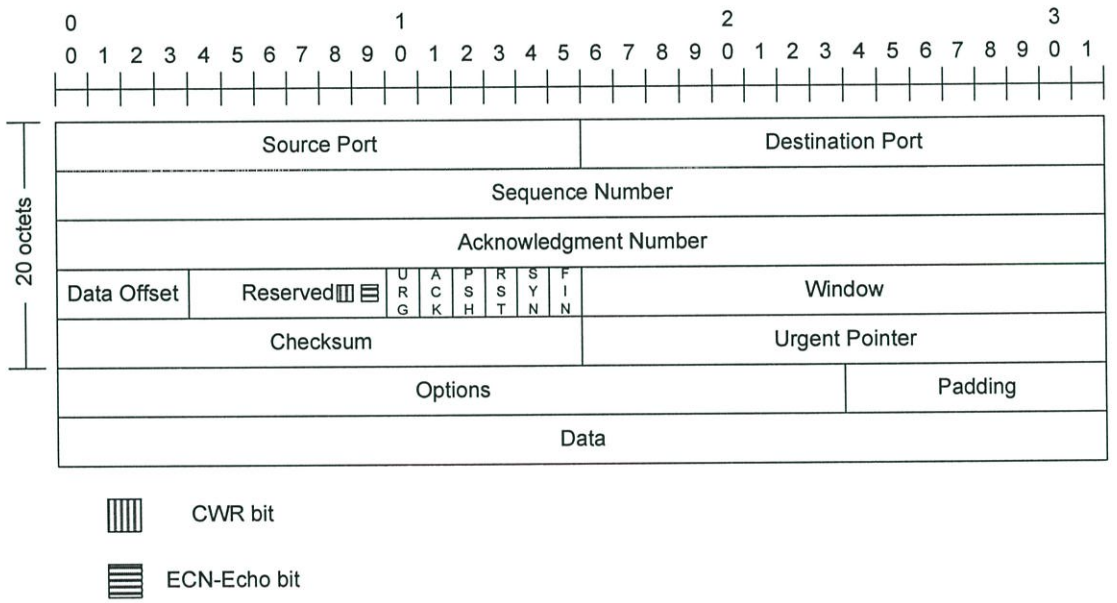


รูปที่ 5.5 ลำดับการทำงานของ ECN

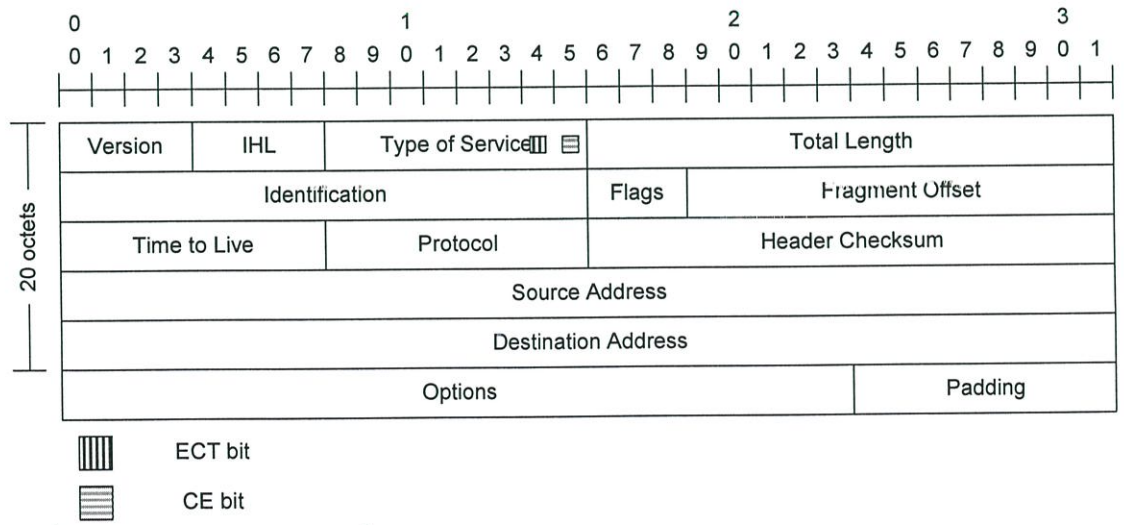
โดยที่

1. Gateway ทำการ mark ที่ CE bit
2. TCP receiver ส่งสัญญาณ ECE
3. TCP sender ส่งสัญญาณ CWR ไปยัง receiver

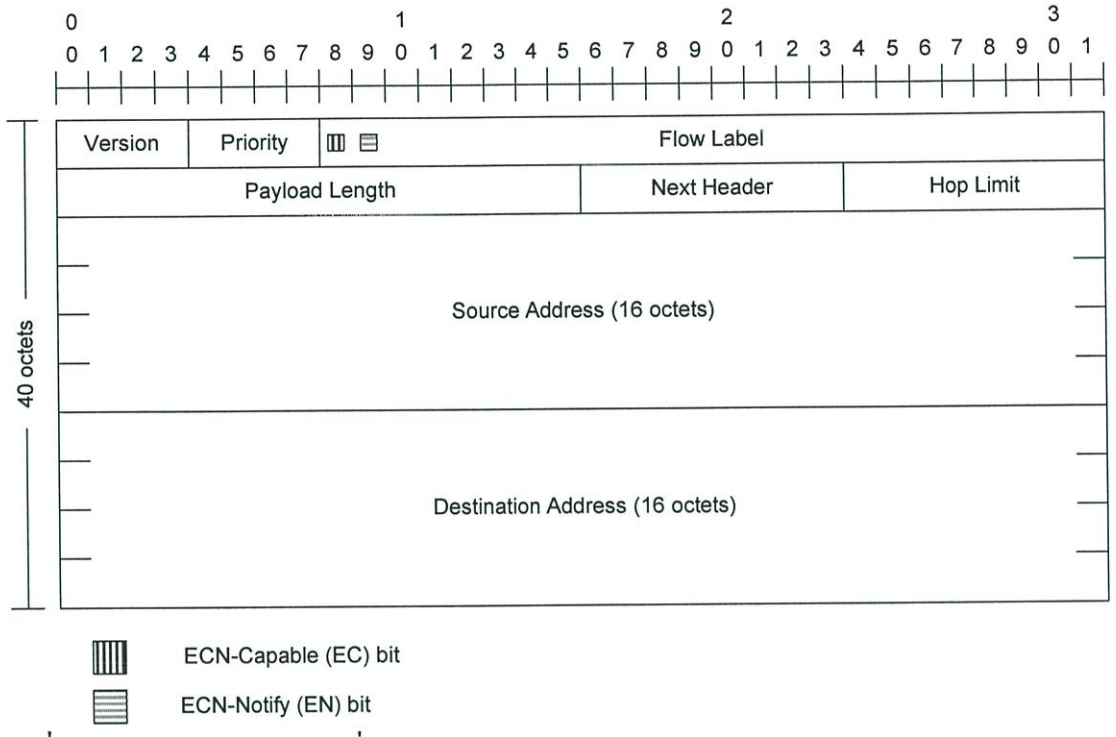
ตำแหน่งของ bit ที่ถูกกำหนดเพิ่มเติมสำหรับสนับสนุนกลไก ECN ใน header format ของ TCP, IPv4 และ IPv6 ถูกแสดงในรูปที่ 5.6 – 5.8 ตามลำดับ



รูปที่ 5.6 TCP header format ที่สนับสนุน ECN



รูปที่ 5.7 IPv4 header format ที่สนับสนุน ECN



รูปที่ 5.8 IPv6 header format ที่สนับสนุน ECN

5.3 Drop-Tail ที่มี ECN

Drop-Tail with ECN (DT-ECN) [12] ถูกนำเสนอขึ้นมาเพื่อลดปริมาณ packet ถูกคัดทิ้งในเครือข่าย โดยเป็นการนำเสนอแนวทางการใช้ ECN ร่วมกับ gateway แบบ Drop-Tail ซึ่งมีคุณสมบัติพื้นฐานคือ เป็น queue แบบ FIFO (First In, First Out) ซึ่งเมื่อมี packet เข้ามาแล้วไม่มีพื้นที่พอที่จะเก็บ packet นั้นได้ ก็จะทำการคัด packet นั้นทิ้ง โดยมีการแก้ไขให้ packet ที่เข้ามาอยู่ใน queue แต่อยู่ส่วนที่เกินจากค่า threshold แต่ยังไม่เกินขนาดสูงสุดของ queue มีการ set CE bit แต่ถ้าเกินขนาดสูงสุดของ queue ก็จะทำการคัด packet ทิ้งตามปกติ เมื่อ packet ที่มีการ set CE bit ได้ถูกจัดส่งออกไปจาก queue แล้วเดินทางถึง TCP receiver แล้ว TCP receiver จะทำการตรวจสอบสถานะของ CE bit ว่ามีการ set มาหรือไม่ ถ้ามีก็จะตอบกลับด้วย ACK ที่มีการ set ECE ซึ่งเมื่อเดินทางถึง TCP sender แล้ว TCP sender จะทราบว่ากำลังจะเกิด congestion ในเส้นทางนั้น จะทำการตอบด้วย CWR แล้วจึงทำการปรับลด congestion window ลงเพื่อลดอัตราการส่งข้อมูล ขั้นตอนทั้ง 3 ถูกกำหนดขึ้นเพื่อให้เกิด reliability ในกลไก เมื่อมี packet ที่มีการ set bit หายไปในเครือข่าย และ ไม่มีการกำหนด packet เพิ่มเป็นพิเศษจึงไม่เป็นการสร้างปริมาณ traffic เพิ่มในเครือข่าย

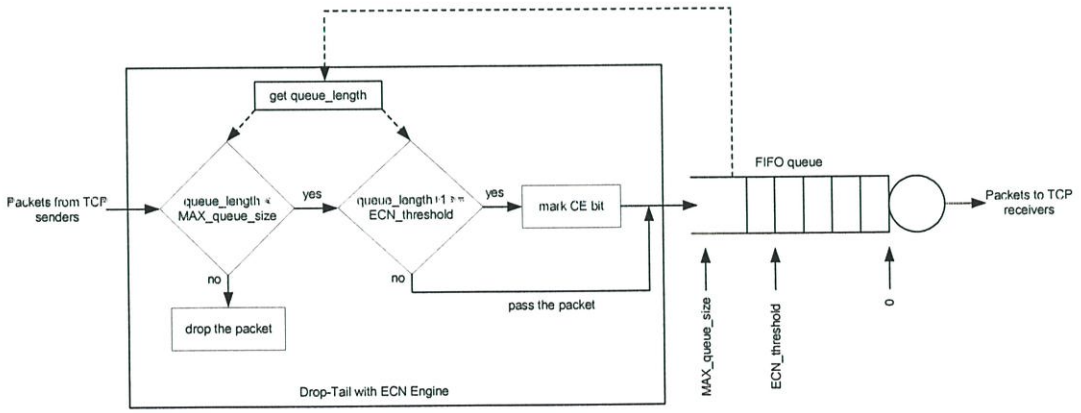
สามารถแสดงในรูป Algorithm ได้ดังนี้

```
ECN_threshold = MAX_queue_size * (Queue_utilization / 100)
for each packet arrival do
        if queue_length < MAX_queue_size
                if queue_length+1 >= ECN_threshold
                        mark CE bit on the arriving packet
                enqueue the packet
        else
                drop the arriving packet
```

โดย

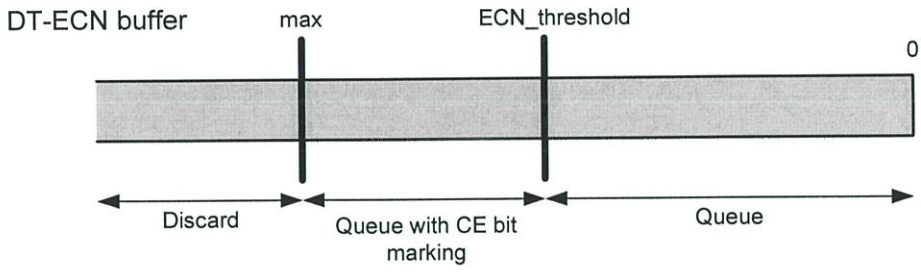
ECN_threshold:	เป็นระดับการใช้งานของ queue ที่ตั้งไว้จะให้มีการ mark CE bit
MAX_queue_size:	ขนาดสูงสุดของ queue
Queue_utilization:	percent การใช้งานของ queue ที่ตั้งไว้จะให้มีการ mark CE bit
queue_length:	ขนาดของ queue ที่ถูกใช้ในขณะนั้น

จาก algorithm ของ DT-ECN ข้างต้นถูกแสดงการทำงานในรูปแบบ diagram ในรูปที่ 5.9 เพื่อความง่ายต่อการทำความเข้าใจ กลไก DT-ECN ถูกวางไว้เพื่อจัดการกับ packet ที่เข้ามาทางด้านซ้ายมือ ก่อนที่จะจัดส่งเข้าไปเก็บใน queue get queue_length ทำหน้าที่แสดงปริมาณของข้อมูลที่อยู่ใน queue ถูกใช้เปรียบเทียบกับ ขนาดสูงสุดของ queue และ ECN_threshold ทำให้เกิด 3 เส้นทางที่เป็นไปได้ คือ คัดทิ้ง, นำเข้าใน queue และ ทำการเขียน CE bit ก่อนนำเข้าใน queue



รูปที่ 5.9 การทำงานภายในของ Drop-Tail with ECN

อีกรูปที่แสดงการทำงานของ DT-ECN ดังรูป 5.10 โดยใช้มุมมองของการตอบสนองต่อ packet ที่เข้ามาในแต่ละช่วงของ Queue โดยการแบ่งช่วงของค่า max และ ECN_threshold ทำให้เกิด 3 ช่วง คือ คัดทิ้ง, นำเข้าใน queue และ ทำการเขียน CE bit ก่อนนำเข้าใน queue



รูปที่ 5.10 การทำงานของ queue แบบ DT-ECN

ECN_threshold เป็นตัวแปรตัวเดียวของ DT-ECN ที่เป็นตัวกำหนดขอบเขตการแจ้งเตือนสภาพแออัด การกำหนดค่าของ ECN_threshold ที่แตกต่างกันออกไปจะทำให้ queue แบบ DT-ECN มีการตอบสนองต่อสภาพแวดล้อมที่แตกต่างกันออกไปด้วย ถ้าตั้งไว้เหมาะสมกับสภาพแวดล้อมย่อมส่งผลให้มีประสิทธิภาพสูงสุด ในบทนี้เรายังไม่ได้แนะนำเรื่องการตั้งค่า ECN_threshold ที่เหมาะสมไว้เนื่องจากต้องอาศัยการจำลองเครือข่ายเพื่อศึกษาหาค่าที่เหมาะสม

5.4 สรุป

จากบทนี้เราได้สร้างกลไกแบบใหม่ขึ้นมาที่เรียกว่า DT-ECN ซึ่งมีโครงสร้างไม่ซับซ้อน แต่มีความสามารถในการแจ้งเตือนถึงสภาพความแออัดของเครือข่ายก่อนที่จะเกิดการคัด packet ทิ้ง ในบทต่อไปเราจะกำหนดวิธีการในการทดสอบกลไกแบบ DT-ECN กับกลไกแบบอื่นๆในรูปแบบต่างๆ โดยใช้ NS-2 เป็นเครื่องมือหลักในการจำลองเครือข่าย

บทที่ 6

การ simulation

ในบทที่แล้วได้แนะนำกลไก DT-ECN เพื่อใช้ลดปริมาณการคัด packet ทิ้งในทางทฤษฎี แต่ยังไม่ได้พูดถึงในแง่ของประสิทธิภาพที่จะได้รับจากกลไกนี้ ในบทนี้จะนำเข้าสู่การสร้าง DT-ECN ขึ้นมาด้วย NS-2 จากนั้นจะทำการกำหนดเงื่อนไขในการจำลองเครือข่ายอย่างละเอียด เพื่อที่จะนำกลไก DT-ECN มาทดสอบภายใต้เงื่อนไขที่ได้กำหนดนี้ โดยต้องสอดคล้องกับวัตถุประสงค์และขอบเขตของวิทยานิพนธ์ฉบับนี้ที่กล่าวไว้ในบทนำ

6.1 เครื่องมือที่ใช้ทำการ simulation

เครื่องมือที่ใช้ในการจำลองเครือข่ายเพื่อวัดประสิทธิภาพในวิทยานิพนธ์ฉบับนี้ประกอบไปด้วย

Hardware: Notebook Computer แบบ Pentium III 600MHz, 256MB RAM, 20GB HDD
OS: Red Hat Linux release 7.1, Seawolf, Kernel 2.4.2-2
Application: NS 2.1b9a, NAM 1.0a11a, Tcl 8.3.2, TclCL 1.0b12, Tk 8.3.2, OTcl 1.0a8, Zlib 1.1.3, Xgraph 12.1, GNU Awk 3.0.6, gcc 2.96, GNU Make 3.79.1

Hardware ที่ใช้ในการจำลองเครือข่ายในวิทยานิพนธ์ฉบับนี้เป็น Personal Computer (PC) แบบทั่วๆไป ซึ่งมีความเร็วเหมาะสมต่อการทำการจำลอง (CPU กับ memory มีผลต่อเวลาที่ใช้ในการจำลอง ความจุของ Hard disk มีผลต่อขนาดและจำนวนของ trace file ที่สามารถเก็บได้ ยิ่งการจำลองที่ซับซ้อนและต้องทดสอบหลายๆรูปแบบก็ยิ่งต้องการ CPU ที่เร็วขึ้น memory ที่มากขึ้น และ Hard disk ที่มีความจุสูงขึ้น) ระบบปฏิบัติการแบบ Linux ถูกเลือกใช้เนื่องจากมีความเข้ากันได้กับระบบ UNIX อื่นๆและทำงานได้บน PC เพื่อรองรับการทำงานของ NS-2 ส่วน Application หลัก คือ NS-2 ที่เป็น simulator แบบ object-oriented และ discrete event driven พัฒนาขึ้นมาโดย University of California at Berkeley (UC Berkeley) เขียนขึ้นมาจากภาษา C++ และ OTcl ซึ่ง NS-2 ถูกใช้ในงานวิจัยและการเรียนการสอนในต่างประเทศอย่างแพร่หลาย (แสดงรายการผลงานวิจัยเป็นจำนวนมากที่ใช้ NS-2 ในภาคผนวก ข) NS 2.1b9a เป็น version ที่ถูกเผยแพร่เมื่อ 13 เมษายน 2545

เมื่อทำการเตรียมเครื่องมือที่กล่าวมาข้างต้นเรียบร้อยแล้วจากนั้นก็มาถึงในส่วนของการพัฒนา code ขึ้นมา ซึ่งสามารถแบ่งการพัฒนาออกเป็น 3 ส่วน คือ

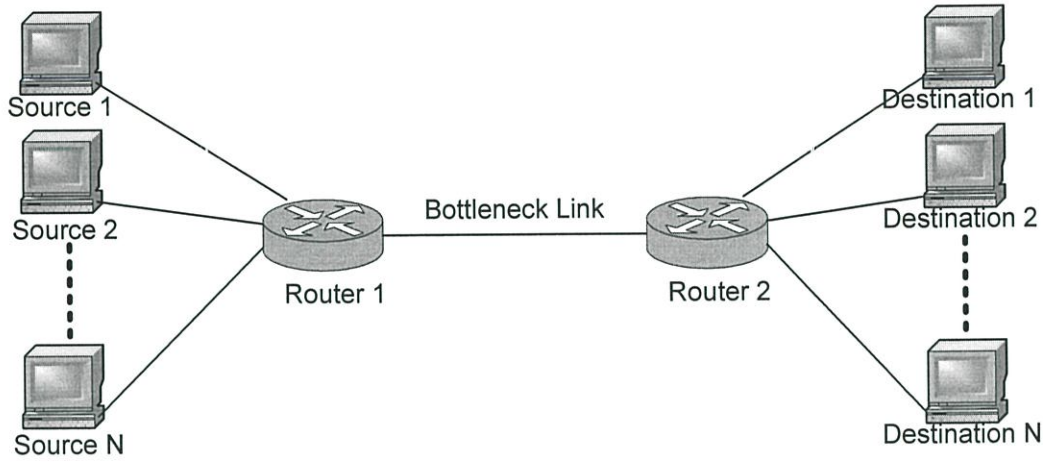
1. พัฒนา queue object ขึ้นมาใหม่ด้วยภาษา C++ ตามแนวทางที่กำหนดโดย NS-2 (แสดง source code ของ DT-ECN ในภาคผนวก ก.1 และ ก.2)
2. พัฒนา script ด้วยภาษา OTcl (แสดงตัวอย่างบางส่วนในภาคผนวก ก.3) เพื่อใช้กำหนด topology, setup ค่าต่างๆให้กับ object ที่เรียกใช้ และเวลาที่เหตุการณ์ต่างๆเกิดขึ้นให้กับ NS-2 จากนั้นทำการ run เพื่อให้ได้ผลลัพธ์ที่เป็น trace file ที่ประกอบไปด้วย เวลา, ตำแหน่งที่ packet อยู่, จุดที่ packet จะเดินทางไป, ชนิดของ packet และ ข้อมูลอื่นๆ
3. พัฒนา script ด้วยภาษา OTcl ที่เรียกใช้ Awk ช่วยในการประมวลผล (แสดงตัวอย่างบางส่วนในภาคผนวก ก.4) เพื่อเลือกข้อมูลที่ต้องการบางส่วน (filter) จาก trace file ที่ได้มาประมวลผล

6.2 การ simulation

วิทยานิพนธ์ฉบับนี้ได้ทำการ simulation โดยใช้ network topology ที่มีลักษณะแบบ wide area network (WAN) ที่ประกอบด้วยกลุ่มของ source ทางซ้ายมือ ส่งข้อมูลผ่าน router 2 ตัว ไปยังกลุ่มของ destination ทางขวามือ ดังแสดงในรูปที่ 6.1 โดยมีการกำหนดค่าต่างๆเพื่อใช้ในการจำลองเครือข่ายดังนี้

- Link ทั้งหมดมีขนาด bandwidth เป็น 45Mbps (ตามมาตรฐาน T3) และมีค่า propagation delay เป็น 10 msec
- TCP sender ที่เลือกใช้มี 2 แบบ คือ แบบ TCP-Reno ที่มีกลไก Fast-Retransmit และ Fast-Recovery และแบบ TCP Selective Acknowledgment (TCP SACK) [13] ที่เป็นมาตรฐานล่าสุดที่ถูกปรับปรุงคุณภาพจาก TCP Reno ในสภาวะที่หลายๆ packet ถูก drop ภายใน window เดียวกัน [13] (ตัวอย่างของ OS ที่ ใช้ SACK เป็นมาตรฐานและกำหนดให้ถูกใช้งานโดยเริ่มต้น เช่น Sun Solaris 8 [14] ที่ทำตาม RFC 2018 และ RFC 1323, Sun Solaris 9, Linux kernel 2.1.90 เป็นต้นไป , Microsoft Windows 98)
- TCP connection ทั้งหมดถูกเรียกใช้จาก application แบบ FTP ที่มีการส่งข้อมูลอยู่ตลอดเวลา (bulk-data connection)
- Maximum segment size (MSS) ของ TCP ถูกตั้งเป็น 1460 byte
- TCP receiver's buffer size ถูกตั้งให้มีค่าสูงสุดของ 16-bit window size
- ตัวแปร ssthresh ถูกตั้งให้มีค่าเริ่มต้นที่ 65536 byte
- Buffer size ที่ router ถูกกำหนดให้มีค่าเป็น 2 เท่าของ bandwidth-delay product (BDP) ซึ่งมีขนาด 675 kbyte

- threshold ที่ใช้ set ECN bit กำหนดให้มีค่าเป็น 25%, 50%, 75%, 80% และ 90% ของ buffer size ตามลำดับ การที่กำหนดขึ้นมาให้กระจายครอบคลุมตั้งแต่น้อยจนมากก็เพื่อจะได้เห็นถึงการตอบสนองในแบบต่างๆมากที่สุด



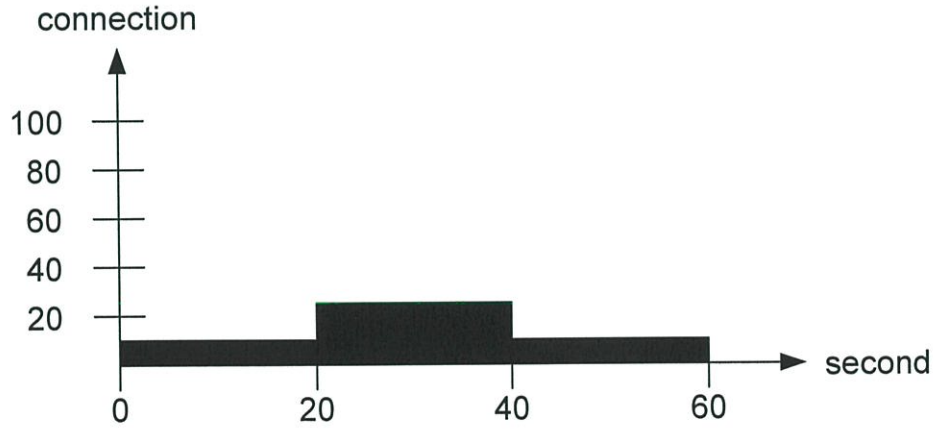
รูปที่ 6.1 Network topology แบบ WAN ที่ใช้ทดสอบ

วิทยานิพนธ์ฉบับนี้ได้ลองทดสอบกับรูปแบบของ traffic ที่ถูกจ่ายเข้ามาใน network หลายๆรูปแบบ เช่น จำนวน connection รวมสูงสุดจากน้อยๆ ไปจนถึงมากๆ (25 – 100 connection) และ connection แบบที่ค่อยๆเพิ่มขึ้นจนหนาแน่น, แบบที่เริ่มจากหนาแน่นแล้วค่อยๆลดลงไปจนถึงแบบที่หนาแน่นคงที่ตลอดเวลา ดังที่ได้กำหนดตัวแทนของ traffic ไว้ 6 รูปแบบ คือ แบบ A, B, C, D, E และ F ดังนี้

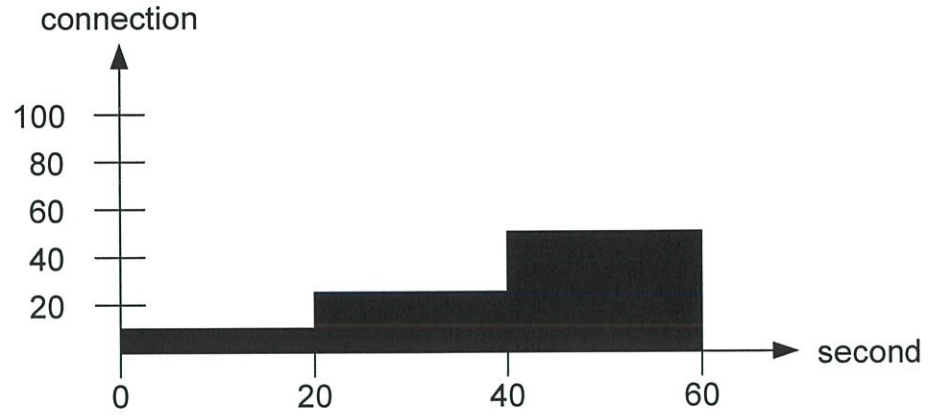
- แบบ A เป็นกรณีที่มี traffic จำนวน 10 connection อยู่ตลอดเวลา simulation (0.0 second – 60.0 second) แต่ในช่วงกลางของเวลา (20.0 second – 40.0 second) ได้มี traffic อีก 15 connection เข้ามาเป็นระยะเวลา 20 second แล้วก็หายไป (แสดงด้วยรูปที่ 6.2) เพื่อใช้เป็นตัวแทนของปริมาณ traffic ที่น้อยที่สุดใน 6 แบบ
- แบบ B เป็นแบบที่ปริมาณ traffic มีปริมาณเพิ่มขึ้นจนจบการ simulation คือ เริ่มจาก 10 connection ที่เวลาเริ่มต้น 0.0 second และเพิ่มไปเป็น 25 connection ที่เวลา 20.0 second แล้วเพิ่มอีกเท่าตัวไปเป็น 50 connection ที่เวลา 40.0 second และคงที่ด้วยจำนวนนี้ไปจนจบการ simulation ที่เวลา 60.0 second (แสดงด้วยรูปที่ 6.3) เพื่อใช้เป็นตัวแทนของปริมาณ traffic ที่มากพอสมควรและเป็นแบบที่มีปริมาณเพิ่มขึ้นเรื่อยๆ
- แบบ C เป็นแบบที่ปริมาณ traffic เริ่มจากมากไปหาน้อย โดยเริ่มจาก 50 connection ที่เวลาเริ่มต้น 0.0 second แล้วลดลงเหลือ 35 connection ที่เวลา 20.0 second และลดลงในช่วงสุดท้ายเหลือ 10 connection ที่เวลา 40.0 second และคงที่ด้วยจำนวนนี้ไปจนจบการ simulation ที่เวลา 60.0 second

(แสดงด้วยรูปที่ 6.4) เพื่อใช้เป็นตัวแทนของปริมาณ traffic ที่มากพอสมควรและเป็นแบบที่มีปริมาณลดลงเรื่อยๆ

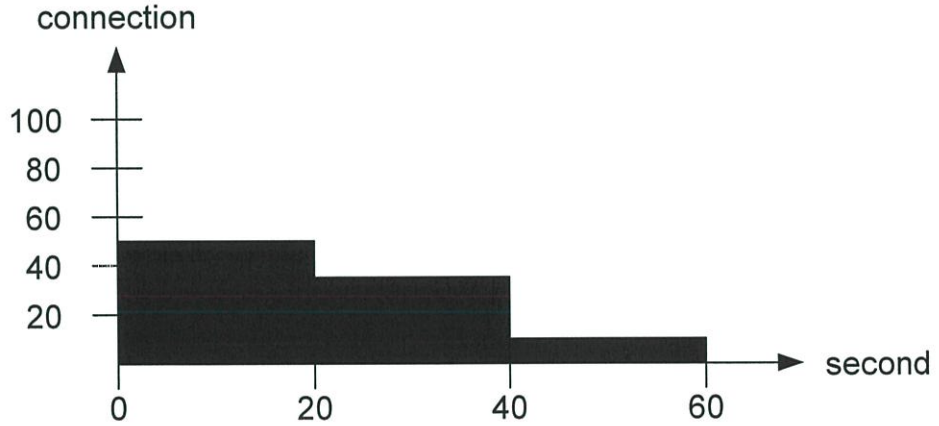
- แบบ D เป็นแบบที่คล้ายกับ แบบ B แต่ทำการเพิ่มปริมาณ traffic เป็น 2 เท่า คือ เริ่มจาก 20 connection ที่เวลาเริ่มต้น 0.0 second และเพิ่มไปเป็น 50 connection ที่เวลา 20.0 second แล้วเพิ่มอีกเท่าตัวไปเป็น 100 connection ที่เวลา 40.0 second และคงที่ด้วยจำนวนนี้ไปจนจบการ simulation ที่เวลา 60.0 second (แสดงด้วยรูปที่ 6.5) เพื่อใช้เป็นตัวแทนของปริมาณ traffic ที่มากที่สุดและเป็นแบบที่มีปริมาณเพิ่มขึ้นเรื่อยๆ
- แบบ E เป็นแบบที่คล้ายกับ แบบ C แต่ทำการเพิ่มปริมาณ traffic เป็น 2 เท่า โดยเริ่มจาก 100 connection ที่เวลาเริ่มต้น 0.0 second แล้วลดลงเหลือ 70 connection ที่เวลา 20.0 second และลดลงในช่วงสุดท้ายเหลือ 20 connection ที่เวลา 40.0 second และคงที่ด้วยจำนวนนี้ไปจนจบการ simulation ที่เวลา 60.0 second (แสดงด้วยรูปที่ 6.6) เพื่อใช้เป็นตัวแทนของปริมาณ traffic ที่มากที่สุดและเป็นแบบที่มีปริมาณลดลงเรื่อยๆ
- และสุดท้ายแบบ F เป็นแบบที่มีความหนาแน่นของ traffic สูงสุด โดยมีการใช้งาน 100 connection ตลอดทั้ง 0.0 second – 60.0 second (แสดงด้วยรูปที่ 6.7) เพื่อใช้เป็นตัวแทนของปริมาณ traffic ที่มากที่สุดและมีปริมาณมากที่สุดคงที่ตลอดเวลา



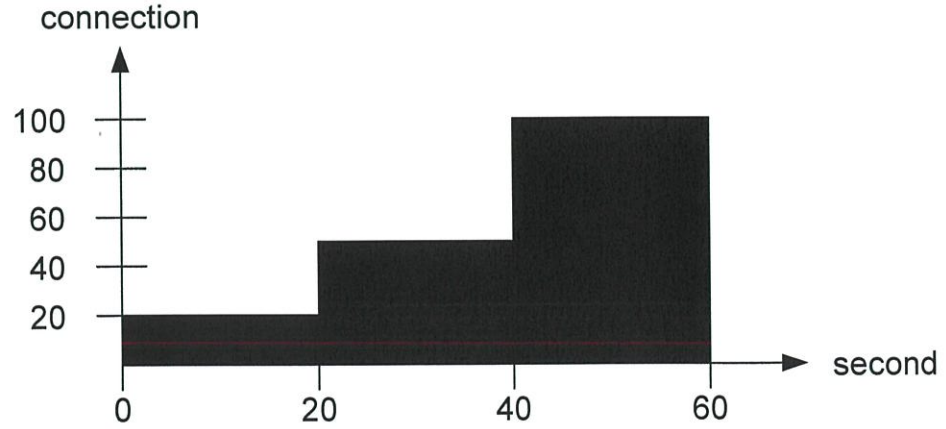
รูปที่ 6.2 Traffic แบบ A



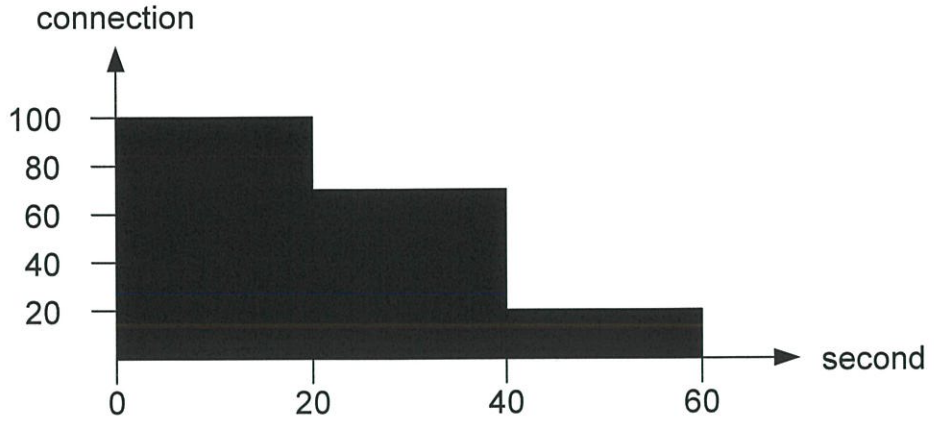
รูปที่ 6.3 Traffic แบบ B



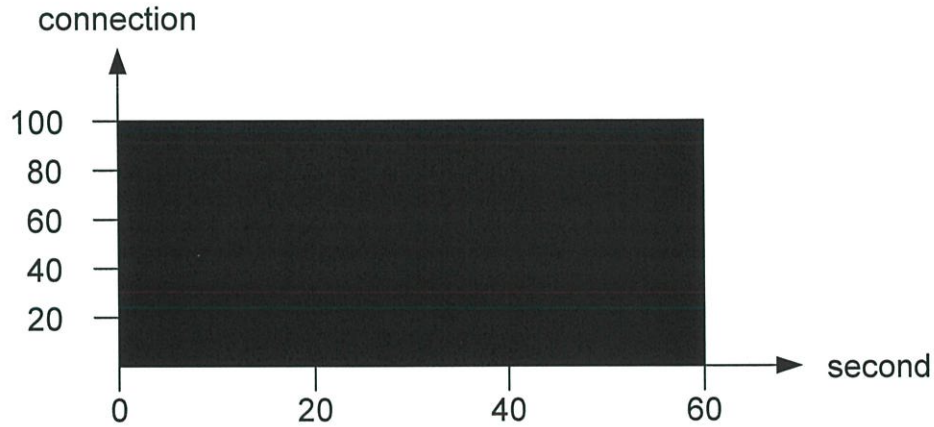
รูปที่ 6.4 Traffic แบบ C



รูปที่ 6.5 Traffic แบบ D



รูปที่ 6.6 Traffic แบบ E



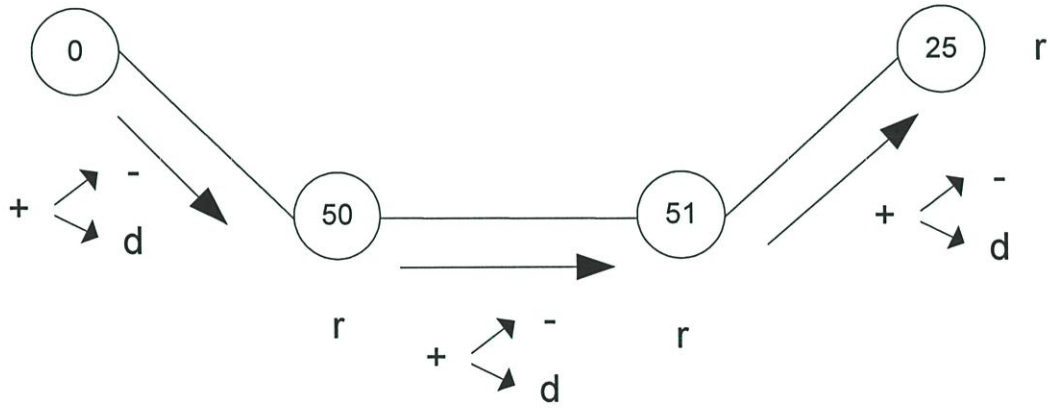
รูปที่ 6.7 Traffic แบบ F

วิธีการต่างๆที่เรานำมาทำการ simulation เพื่อเปรียบเทียบกันมีดังนี้ Drop-Tail, Drop-Tail with ECN at 25%, Drop-Tail with ECN at 50%, Drop-Tail with ECN at 75%, Drop-Tail with ECN at 80%, Drop-Tail with ECN at 90%, RED with ECN และ RED without ECN เรียงตามลำดับ ในส่วนของ parameter ที่ใช้กับ RED เราได้ set ตามที่ Floyd S. แนะนำไว้ใน [5] คือ min_{th} = 1/4 of queue size, $max_{th} = 2 * min_{th}$, $max_p = 0.02$, $w_q = 0.002$ ทั้งหมดนี้นำมาทดสอบกับ Traffic แบบ A, B, C, D, E, F และทดสอบกับทั้ง TCP Reno และ TCP SACK แสดงว่าเราต้องทำการ simulation ทั้งหมด 8x6x2 = 96 รูปแบบ เพื่อให้ได้เหตุการณ์ที่เกิดขึ้นในรูปของ trace file ทั้งหมด 96 file

6.3 การประมวลผล trace file

หลังจากจบการจำลองเครือข่ายจะได้ trace file ที่แสดงถึงเหตุการณ์ที่เกิดขึ้นตามรูปแบบที่กล่าวไว้ในหัวข้อ 4.4 ตัวอย่างข้างล่างคือบางส่วนของ trace file ที่ได้จากการทดสอบ Drop-Tail with ECN ที่ 75% กับ traffic แบบ A ที่เลือกมาเฉพาะส่วนที่จำเป็นต่อการอธิบายถึงวิธีการเก็บข้อมูล

```
+ 0.181685 0 50 tcp 1500 ------N 1 0.0 25.0 7 140
- 0.181685 0 50 tcp 1500 ------N 1 0.0 25.0 7 140
r 0.191952 0 50 tcp 1500 ------N 1 0.0 25.0 7 140
+ 0.191952 50 51 tcp 1500 ------N 1 0.0 25.0 7 140
- 0.191952 50 51 tcp 1500 ------N 1 0.0 25.0 7 140
r 0.202219 50 51 tcp 1500 ------N 1 0.0 25.0 7 140
+ 0.202219 51 25 tcp 1500 ------N 1 0.0 25.0 7 140
- 0.202219 51 25 tcp 1500 ------N 1 0.0 25.0 7 140
r 0.212485 51 25 tcp 1500 ------N 1 0.0 25.0 7 140
.
.
.
+ 0.469883 50 51 tcp 1500 ------N 1 0.0 25.0 165 2127
d 0.469883 50 51 tcp 1500 ------N 1 0.0 25.0 165 2127
```



รูปที่ 6.8 อธิบายเหตุการณ์จาก trace file

จากรูปที่ 6.8 เป็นตัวอย่างของ node id ที่ได้จากการทดสอบ Drop-Tail with ECN ที่ 75% กับ traffic แบบ A มี 50 เป็น node id ของ router 1 และ 51 เป็น node id ของ router 2 ส่วน 0 เป็น node id ของ computer ด้านส่ง และ 25 เป็น node id ของ computer ด้านรับ จากตัวอย่าง trace file นี้ผลที่ได้ใน 9 บรรทัดแรกจะบรรยายถึงเส้นทางและเวลาที่ packet ผ่านจาก 0 -> 50 -> 51 -> 25 ส่วนใน 2 บรรทัดล่างสุดจะแสดงให้เห็นถึงการคัด packet ทิ้ง แต่เหตุการณ์ที่เราสนใจจริงๆคือการส่ง packet ข้าม bottleneck link สำเร็จหรือไม่สำเร็จซึ่งสามารถตรวจสอบได้จากเหตุการณ์ 2 อย่างต่อไปนี้

1. "- time 50 51 ..." เครื่องหมาย - คือ packet ถูกส่งผ่านไปใน bottleneck link ได้สำเร็จ โดย link ที่ใช้อยู่ระหว่าง node id 50 กับ 51
2. "d time 50 51 ..." ตัวอักษร d คือ packet ถูก drop ทิ้งก่อนส่งเข้าไปใน bottleneck link โดย link ที่ระบุอยู่ระหว่าง node id 50 กับ 51

trace file ทั้งหมด 96 แบบจะถูกนำมาประมวลผลโดย Awk script ที่ถูกเขียนขึ้นมาเฉพาะเพื่อเลือกและประมวลผลเฉพาะเหตุการณ์ที่เป็นการส่ง packet ข้าม bottleneck link ว่าสำเร็จหรือไม่สำเร็จ และจะทำการบวกสะสมเหตุการณ์ทั้ง 2 แบบไปเรื่อยจนครบ 60.0 second ในที่นี้คือผลรวมของข้อมูลที่ส่งผ่านสำเร็จในหน่วย byte (Bytes sent) และ ผลรวมของข้อมูลที่ถูกคัดทิ้งในหน่วย byte (Bytes dropped) เพื่อใช้ตัวเลขจากค่าเหล่านี้แสดงถึงประสิทธิภาพของ queue แต่ละชนิดภายใต้สภาพ traffic ทั้ง 6 รูปแบบ

6.4 สรุป

จากบทนี้เราได้ทำการ simulation จนได้ trace file และนำมาประมวลผลโดย Awk script จนได้ตัวเลขที่นำมาใช้วัดประสิทธิภาพของ queue ทั้ง 8 แบบ ในบทต่อไปเราจะนำผลมาแสดงในรูปแบบตารางและกราฟเพื่อสะดวกในการวิเคราะห์ต่อไป

บทที่ 7

ผลจากการทำ Simulation

7.1 ผลจากการทำ Simulation

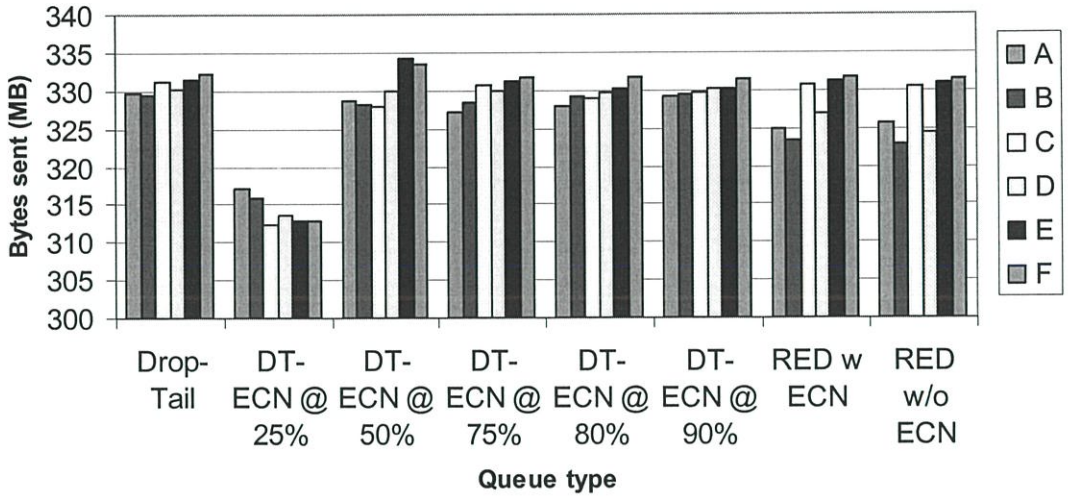
วิทยานิพนธ์ฉบับนี้วัดประสิทธิภาพของ queue ทั้ง 8 แบบ โดยดูจาก ผลรวมของข้อมูลที่ส่งผ่านสำเร็จในหน่วย byte (Bytes sent) และ ผลรวมของข้อมูลที่ถูกคัดทิ้งในหน่วย byte (Bytes dropped) ในรูปของตัวเลขในตารางที่ 7.1, 7.2, 7.3, 7.4 และในรูปของกราฟแท่ง ในรูปที่ 7.1, 7.2, 7.3, 7.4

ตารางที่ 7.1 ปริมาณ Bytes sent เมื่อใช้ TCP Reno

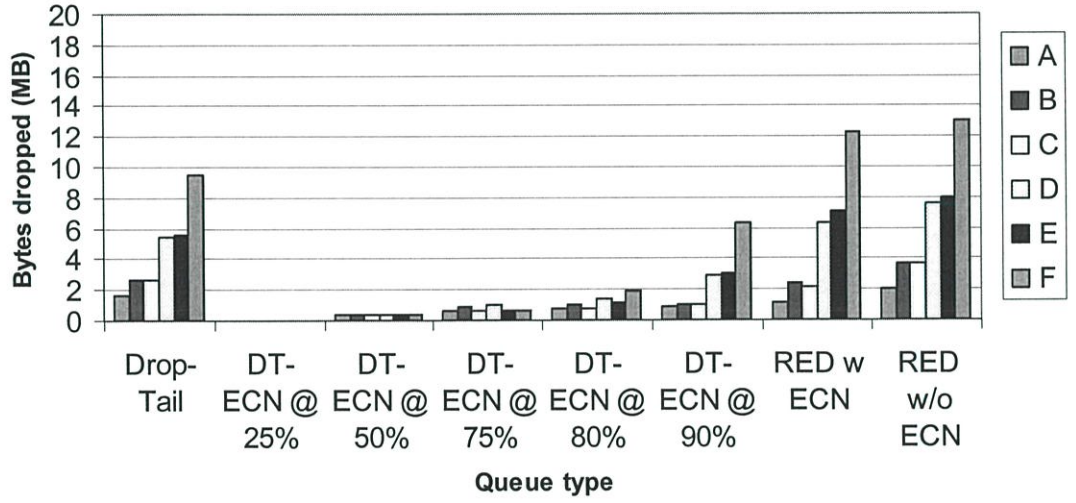
	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	DT-ECN @ 80%	DT-ECN @ 90%	RED w ECN	RED w/o ECN
A	329588500	317098000	328577500	327248500	327817000	329168500	324961000	325607500
B	329532500	315795500	328050500	328442000	329160500	329319500	323399000	322890500
C	331103000	312348500	328025000	330624500	329052500	329663000	330620000	330560000
D	330272500	313489000	329890000	329813500	329731000	330217000	326879500	324503500
E	331523500	312826000	334180000	331286500	330103000	330109000	331148500	330953500
F	332239000	312853000	333370000	331669000	331807000	331327000	331747000	331477000
Total	1984259000	1884410000	1982093000	1979084000	1977671000	1979804000	1968755000	1965992000

ตารางที่ 7.2 ปริมาณ Bytes dropped เมื่อใช้ TCP Reno

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	DT-ECN @ 80%	DT-ECN @ 90%	RED w ECN	RED w/o ECN
A	1695000	0	342000	681000	762000	882000	1122000	1972500
B	2694000	0	342000	880500	1017000	1021500	2442000	3708000
C	2680500	0	342000	681000	747000	1002000	2160000	3636000
D	5410500	0	409500	988500	1444500	2859000	6291040	7578040
E	5607000	0	342000	681000	1089000	3081000	7126500	8013000
F	9472500	0	342000	681000	1870500	6319500	12328500	13095000
Total	27559500	0	2119500	4593000	6930000	15165000	31470040	38002540



รูปที่ 7.1 กราฟแสดง Bytes sent เมื่อใช้ TCP Reno



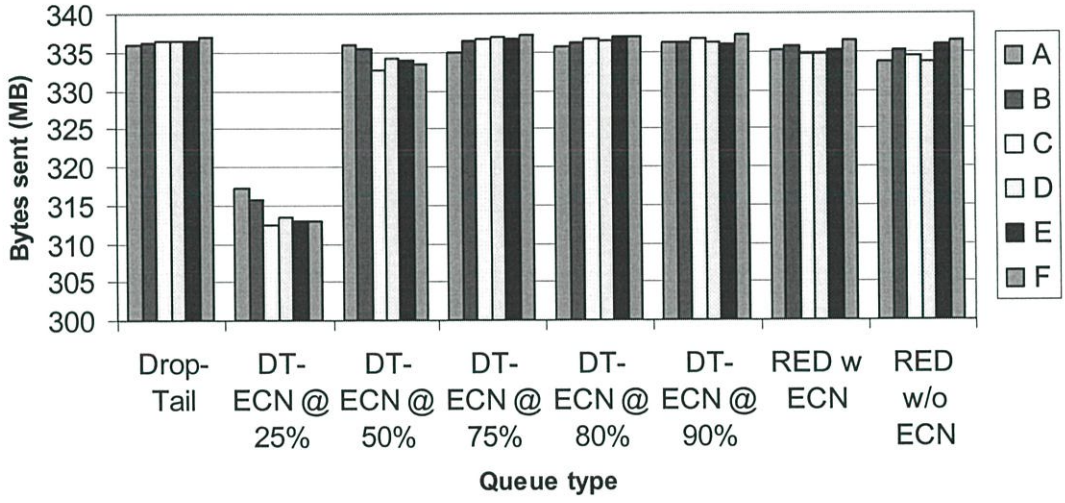
รูปที่ 7.2 กราฟแสดง Bytes dropped เมื่อใช้ TCP Reno

ตารางที่ 7.3 ปริมาณ Bytes sent เมื่อใช้ TCP SACK

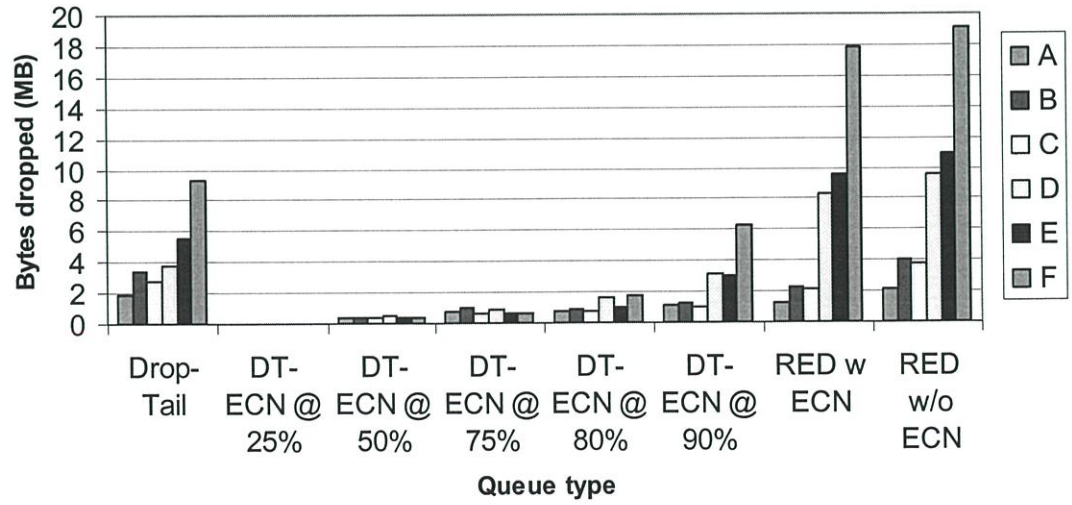
	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	DT-ECN @ 80%	DT-ECN @ 90%	RED w ECN	RED w/o ECN
A	335884000	317098000	336022000	335054500	335594500	336106000	335314000	333788500
B	336149000	315795500	335399000	336561500	336273500	336186500	335573000	335276000
C	336572000	312348500	332624000	336834500	336648500	336618500	334751000	334389500
D	336428500	313489000	334150000	336919000	336430000	336322000	334721500	333767500
E	336572500	312826000	333824500	336790000	336929500	335927500	335284000	336010000
F	336985000	312853000	333311500	337174000	337079500	337217500	336437500	336433000
Total	2018591000	1884410000	2005331000	2019333500	2018955500	2018378000	2012081000	2009664500

ตารางที่ 7.4 ปริมาณ Bytes dropped เมื่อใช้ TCP SACK

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	DT-ECN @ 80%	DT-ECN @ 90%	RED w ECN	RED w/o ECN
A	1917000	0	342000	757500	747000	1101000	1243500	2095500
B	3351000	0	342000	991500	849000	1264500	2302500	4021500
C	2737500	0	342000	681000	747000	1012500	2197500	3790500
D	3729880	0	447000	859500	1695000	3172500	8332540	9576040
E	5496000	0	342000	681000	1021500	3072000	9616500	10912500
F	9306000	0	342000	681000	1803000	6333000	17812500	19065000
Total	26537380	0	2157000	4651500	6862500	15955500	41505040	49461040



รูปที่ 7.3 กราฟแสดง Bytes sent เมื่อใช้ TCP SACK



รูปที่ 7.4 กราฟแสดง Bytes dropped เมื่อใช้ TCP SACK

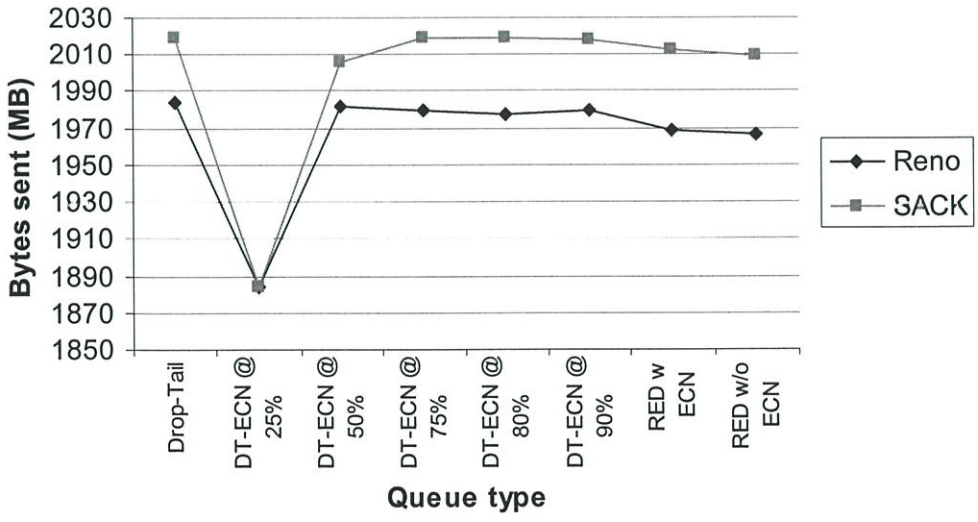
7.2 วิเคราะห์ผลจากการทำ Simulation

จากผลในตารางที่ 7.3 ถ้าดูที่ปริมาณการส่งผ่านข้อมูลแบบรวมเฉพาะ TCP SACK ของ DT-ECN เมื่อทำการเพิ่มค่า threshold ให้มากขึ้นจาก 25% ไปเป็น 50% จะได้ปริมาณการส่งผ่านข้อมูลที่สูงขึ้นเรื่อยๆ เมื่อเพิ่มค่า threshold เป็น 75% ปริมาณการส่งผ่านข้อมูลก็ยังคงเพิ่มขึ้นอีก แต่ถ้าเพิ่มค่า threshold ขึ้นไปอีกเป็น 80% และ 90% ปริมาณการส่งผ่านข้อมูลจะเริ่มลดลง โดยมีจุดที่ได้การส่งผ่านข้อมูลสูงสุดอยู่ที่ค่า threshold 75% ในรูป 7.5 แสดงจุดสูงสุดอยู่ที่ threshold 75% ให้เห็นอย่างง่ายๆ

จากผลในตารางที่ 7.1 จากปริมาณการส่งผ่านข้อมูลแบบรวมเฉพาะ TCP Reno เราพบว่า DT-ECN ที่ threshold 50% มีปริมาณการส่งผ่านข้อมูลสูงสุดในการทดสอบด้วย traffic แบบ E และ F ส่วน Drop-Tail มีปริมาณการส่งผ่านข้อมูลสูงสุดใน traffic แบบ A, B, C และ D และยังมีผลรวมของปริมาณการส่งผ่านข้อมูลสูงสุดอีกด้วย

ตารางที่ 7.5 ผลต่างของปริมาณ Bytes sent ของ TCP SACK เทียบกับ TCP Reno

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	DT-ECN @ 80%	DT-ECN @ 90%	RED w ECN	RED w/o ECN
Reno (Byte)	1984259000	1884410000	1982093000	1979084000	1977671000	1979804000	1968755000	1965992000
SACK (Byte)	2018591000	1884410000	2005331000	2019333500	2018955500	2018378000	2012081000	2009664500
ผลต่าง (Byte)	34332000	0	23238000	40249500	41284500	38574000	43326000	43672500
ผลต่าง (%)	1.730217678	0	1.172397057	2.033743894	2.087531243	1.948374688	2.200680125	2.221397646

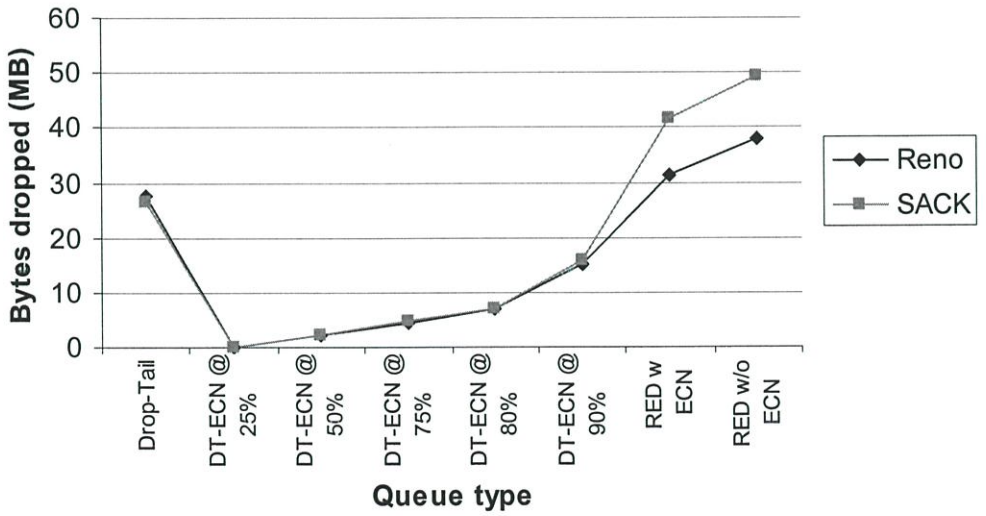


รูปที่ 7.5 กราฟแสดงผลรวมของปริมาณการส่งข้อมูลของ TCP Reno และ TCP SACK

การทดสอบของวิทยานิพนธ์ฉบับนี้ใช้ TCP 2 แบบ คือ TCP Reno และ TCP SACK เพื่อเป็นตัวแทนพฤติกรรมของ TCP ในการทดสอบ เห็นได้ชัดเจนว่า TCP SACK ให้ throughput สูงกว่า TCP Reno ในทุกกรณี ยกเว้น DT-ECN ที่ 25% ที่ไม่มีความแตกต่าง (จากรูปที่ 7.5 และตารางที่ 7.5) ที่ TCP SACK ให้ throughput สูงกว่า TCP Reno เนื่องจาก TCP SACK ถูกปรับปรุงในเรื่องของการส่ง packet ซ้ำเมื่ออยู่ในสภาพแออัดมากๆที่มี packet ถูก drop มากๆ ซึ่งทำให้ TCP SACK เป็นที่นิยมในการนำมาใช้เป็นมาตรฐานกับ OS รุ่นใหม่ๆ ในกรณีของ DT-ECN ที่ 25% ทั้ง TCP Reno และ TCP SACK ได้ผลเท่ากันเพราะที่ threshold 25% นั้นมีการ utilization เครือข่ายไม่เต็มที่เพราะมีการแจ้งว่าเกิดความแออัดล่วงหน้ามากเกินไปทั้งๆที่ยังไม่เกิดความแออัด ทำให้ไม่มีการคัด packet ทิ้งเกิดขึ้นเลยจึงไม่ได้ใช้คุณสมบัติพิเศษของ TCP SACK ถ้าดูเฉพาะ TCP SACK ผลลัพธ์ที่ได้ของ DT-ECN ที่ 75% มีการส่งผ่านข้อมูลสูงสุดในแบบ B, C, D, E และ F

ตารางที่ 7.6 ผลต่างของปริมาณ Bytes dropped ของ TCP SACK เทียบกับ TCP Reno

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	DT-ECN @ 80%	DT-ECN @ 90%	RED w ECN	RED w/o ECN
Reno (Byte)	27559500	0	2119500	4593000	6930000	15165000	31470040	38002540
SACK (Byte)	26537380	0	2157000	4651500	6862500	15955500	41505040	49461040
ผลต่าง (Byte)	-1022120	0	37500	58500	-67500	790500	10035000	11458500
ผลต่าง (%)	-3.708775558	0	1.769285209	1.273677335	-0.974025974	5.212660732	31.88747139	30.15193195



รูปที่ 7.6 กราฟแสดงผลรวมของปริมาณการ drop ข้อมูลของ TCP Reno และ TCP SACK

ถ้าดูเฉพาะผลของการคัด packet ทิ้งของ DT-ECN ทั้ง TCP Reno และ TCP SACK เป็นไปในทิศทางเดียวกันคือ ที่ค่า threshold น้อยๆที่ 25% จะไม่เกิดการคัด packet ทิ้งเลย แต่เมื่อค่า threshold เพิ่มมากขึ้นเป็นที่ 50%, 75% และ 80% การคัด packet ทิ้งก็มีเพิ่มขึ้นตามไปด้วยตามลำดับ โดย threshold ที่ 90% จะเกิดการคัด packet ทิ้งจำนวนมากที่สุดในกลุ่ม DT-ECN แต่ DT-ECN ที่ทุกค่า threshold ก็ยังมีการคัด packet ทิ้งที่น้อยกว่า Drop-Tail, RED ที่มี ECN และ ที่ไม่มี ECN อยู่มาก จากรูปที่ 7.6 และตารางที่ 7.6 ยังพบอีกว่าปริมาณการคัด packet ทิ้งของ DT-ECN ที่ 50%, 75% และ 80% ใกล้เคียงกันมากของทั้ง TCP Reno และ TCP SACK ด้วยความแตกต่างกันเพียง 1.769285209%, 1.273677335% และ -0.974025974% ตามลำดับ แสดงให้เห็นว่าปริมาณการคัด packet ทิ้งของ DT-ECN ไม่ขึ้นอยู่กับชนิดของ TCP

นอกจากนี้ยังพบว่า RED ที่มี ECN มีประสิทธิภาพสูงกว่า RED ที่ไม่มี ECN เกือบทุกรูปแบบของ traffic ในด้าน throughput (TCP Reno: B, C, D, E, F และ TCP SACK: A, B, C, D, F) และดีกว่าทุกรูปแบบของ traffic ในด้าน packet dropping เนื่องจากความแตกต่างของวิธีการจัดการของ RED ที่มี ECN ใช้การสุ่มเพื่อ mark CE bit แต่ RED ที่ไม่มี ECN ใช้การสุ่มเพื่อคัดทิ้ง ดังนั้นที่การ setup ค่าตัวแปรแบบเดียวกัน RED ที่มี ECN จะมีการใช้งาน buffer ได้มีประสิทธิภาพมากกว่า และยังพบอีกว่า Drop-Tail มีประสิทธิภาพสูงกว่า RED ที่มี ECN ในทุกกรณี ในด้าน throughput และ ดีกว่าในด้าน packet dropping ในแบบ D, E, F ของทั้ง TCP Reno และ TCP SACK

บทที่ 8
สรุปผลการวิจัยและข้อเสนอแนะ

วิทยานิพนธ์ฉบับนี้ได้นำเสนอแนวทางในการประยุกต์รวมใช้ Explicit Congestion Notification กับ queue แบบ Drop-Tail เพื่อทำให้ TCP มีการปรับตัวเองเข้ากับสภาพ network ได้รวดเร็วขึ้น ช่วยลดปริมาณ packet ที่ถูก drop ให้ลดลงเป็นอย่างมาก และยังเป็นการเพิ่ม throughput ให้กับ network ที่มีความแออัดอีกด้วย การกำหนดค่าให้กับ threshold มีผลต่อประสิทธิภาพของ DT-ECN เป็นอย่างมาก ถ้ากำหนดไว้น้อยเกินไป queue จะถูกใช้งานไม่เต็มที่เพราะมีการแจ้งสภาพแออัดล่วงหน้าที่มากเกินไป แต่ถ้าตั้ง threshold ไว้มากเกินไปก็จะมีการแจ้งสภาพแออัดไม่ทันต่อสภาพแออัดที่เกิดขึ้นจริงจึงมีประสิทธิภาพใกล้เคียงกับ Drop-Tail ยิ่งค่า threshold มากเท่าไหร่ DT-ECN ก็จะตอบสนองคล้ายกับ Drop-Tail มากขึ้นเรื่อยๆ จากการที่ TCP SACK เป็นที่นิยมใช้กับ OS รุ่นใหม่ๆ ดังนั้นถ้ามองเฉพาะ TCP SACK DT-ECN ที่ threshold 75% มีความเหมาะสมเป็นอย่างมากที่จะถูกนำมาใช้งาน และที่สำคัญ DT-ECN มีความง่ายในการที่จะพัฒนาขึ้นใช้งานบน gateway ซึ่งไม่ต้องมีการคำนวณที่ซับซ้อนและไม่ต้องทำการจัดการ traffic แบบ per flow

ในปัจจุบันอุปกรณ์ขนาดพกพาถูกนำมาใช้ในชีวิตประจำวันมากขึ้นและมีแนวโน้มที่อุปกรณ์เหล่านี้จะมี Wireless LAN (WLAN) ติดมาเป็นอุปกรณ์พื้นฐาน ซึ่งอุปกรณ์ขนาดพกพาเหล่านี้ใช้พลังงานจาก Battery ที่มีขนาดจำกัด การส่งข้อมูลไปในเครือข่ายแล้วเกิดการคัดทิ้งจนต้องเกิดการจัดส่งใหม่ถือว่าเป็นการเสียพลังงานไปโดยเปล่าประโยชน์เนื่องจากต้องมีการประมวลผลข้อมูลเดิมซ้ำแล้วซ้ำอีก ซึ่ง DT-ECN อาจสามารถนำมาใช้เพื่อลดการสูญเสียพลังงานโดยไม่จำเป็นในอุปกรณ์พกพาเหล่านี้ได้

เนื่องจาก DT-ECN ที่นำเสนอถูกตั้งค่า threshold ไว้แบบคงที่อาจจะมีประสิทธิภาพไม่ดีที่สุดเมื่ออยู่ภายใต้สภาพแวดล้อมและ traffic ในแบบอื่นที่ไม่ได้กล่าวไว้ในวิทยานิพนธ์ฉบับนี้ แต่อย่างน้อย DT-ECN มีประสิทธิภาพดีภายใต้สภาพแวดล้อมแบบ WAN ที่มี bandwidth มากที่มีความแออัดมาก

งานวิจัยในอนาคต

เรายังสามารถทำการวิจัยเพิ่มเติมเพื่อปรับปรุงประสิทธิภาพของกลไกที่ใช้ให้มากกว่านี้และด้วยวิธีทดสอบที่ครอบคลุมกว่าที่นำเสนอในวิทยานิพนธ์ฉบับนี้ ตามแนวทางดังต่อไปนี้ กำหนด traffic ที่นำมาทดสอบให้ใกล้เคียงกับสภาพการใช้งานจริงมากขึ้นโดยใช้วิธีการแบบสุ่ม, ทำการทดสอบด้วยช่วงเวลาที่มากกว่า 60 วินาที เพื่อให้เห็นการตอบสนองของเครือข่ายในระยะยาว

, ทำการทดสอบด้วยรูปแบบ Application ที่หลากหลายมากขึ้น เช่น ใช้ connection แบบ TELNET, HTTP ผสมกับ FTP เพื่อให้เครือข่ายจำลองนี้สมจริงยิ่งขึ้น, วิจัยเพิ่มเติมในด้านการประหยัดพลังงานของการลดลงของการส่ง packet ซ้ำในอุปกรณ์ขนาดพกพา และวิจัยเรื่องปริมาณการลดลงของ CPU utilization เนื่องจากการลดลงของการส่ง packet ซ้ำเพื่อใช้ CPU ทำงานด้านอื่นที่เป็นประโยชน์กว่า

เอกสารอ้างอิง

[1] Information Sciences Institute University of Southern California. "TRANSMISSION CONTROL PROTOCOL." Request for Comments (RFC) 793, 1981.

[2] Jacobson V. "Congestion Avoidance and Control." Proceedings of SIGCOMM, 1988, pp. 314-329.

[3] Floyd S. "TCP and Explicit Congestion Notification." ACM Computer Communication Review, vol. 24, no. 5, 1994. pp. 10-23.

[4] Ramakrishnan K., Floyd S. and Black D. "The Addition of Explicit Congestion Notification (ECN) to IP." Request for Comments (RFC) 3168, 2001.

[5] Floyd S. and Jacobson V. "Random Early Detection Gateways for Congestion Avoidance." IEEE/ACM Transactions on Networking, 1993.

[6] Braden B., Clark D., Crowcroft J., Davie B., Deering S., Estrin D., Floyd S., Jacobson V., Minshall G., Partridge C., Peterson L., Ramakrishnan K., Shenker S., Wroclawski J. and Zhang L. "Recommendations on Queue Management and Congestion Avoidance in the Internet." Request for Comments (RFC) 2309, 1998.

[7] Feng W., Kandlur D., Saha D. and Shin K. "A Self-Configuring RED Gateway." INFOCOM, 1999.

[8] May M., Bolot J., Diot C. and Lyles B. "Reasons not to Deploy RED." tech. rep., INRIA, France, 1999.

[9] Aweya J., Ouellette M., Montuno D. and Yao Z. "Enhancing network performance with TCP rate control." Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE, vol. 3, 2000. pp. 1712-1718.

[10] Cisco Systems. **Weighted Random Early Detection on the Cisco 12000 Series Router.**

[11] Cisco Systems. **Cisco 7500 Series Router, Data Sheet**. 2002.

[12] Piyatamrong B. and Anutchalakhom W. "Performance Evaluation on Drop-Tail and RED Gateways." Digital Signal Processing and Its Applications, Proceedings: 1, 2003. pp. 287-290.

[13] Mathis M., Mahdavi J., Floyd S. and Romanow A. "TCP Selective Acknowledgment Options." Request for Comments (RFC) 2018, 1996.

[14] Sun Microsystems, Inc. **What's New in the Solaris 8 Operating Environment.** 2000.

[15] Stalling W. **High-Speed Networks: TCP/IP and ATM Design Principles**. New Jersey : Prentice-Hall, Inc. 1998.

[16] Fall K. and Varadhan K. **The ns Manual.** The VINT Project by UC Berkeley, LBL, USC/ISI and Xerox PARC. 2002.

[17] Chung J. and Claypool M. **NS by Example**. Worcester Polytechnic Institute. 2002.

ภาคผนวก ก

Source code ของ Drop-Tail with ECN

ก.1 แฟ้ม “drop-tail-WRT.cc”

เป็น C++ code ที่ถูกพัฒนาขึ้นมาตามหลักการของกลไกแบบ Drop-Tail with ECN ที่นำเสนอโดยวิทยานิพนธ์ฉบับนี้เพื่อใช้จำลองกลไก DT-ECN ในสภาพแวดล้อมของ Network Simulator 2 มีหน้าที่หลักในการจัดการกับ packet ใน 3 รูปแบบ คือ คัดทิ้ง, นำเข้าใน queue และทำการเขียน CE bit ก่อนนำเข้าใน queue

```
// This file: drop-tail-WRT.cc

#include "flags.h"
#include "drop-tail-WRT.h"

static class DropTailWRTClass : public TclClass {
 public:
        DropTailWRTClass() : TclClass("Queue/DropTailWRT") {}
        TclObject* create(int, const char*const*) {
                return (new DropTailWRT);
        }
} class_drop_tail_WRT;

void DropTailWRT::reset()
{
        Queue::reset();
}

int
DropTailWRT::command(int argc, const char*const* argv) {

        if (argc==2) {
```

```
                if (strcmp(argv[1], "printstats") == 0) {
                        print_summarystats();
                        return (TCL_OK);
                }
        }
        if (argc == 3) {
                if (!strcmp(argv[1], "packetqueue-attach")) {
                        delete q_;
                        if (!(q_ = (PacketQueue*) TclObject::lookup(argv[2])))
                                return (TCL_ERROR);
                        else {
                                pq_ = q_;
                                return (TCL_OK);
                        }
                }
        }
        return Queue::command(argc, argv);
}

/*
 * drop-tail-WRT
 */
void DropTailWRT::enque(Packet* p)
{
        if (summarystats) {
          Queue::updateStats(qib_?q_->byteLength():q_->length());
        }
        q_->enque(p);
        int qlimBytes = qlim_ * mean_pktsize_;
        if ((!qib_ && q_->length() >= qlim_) ||
            (qib_ && q_->byteLength() >= qlimBytes)) {
                if (drop_front_) { /* remove from head of queue */
```

```
                        Packet *pp = q_->deque();
                        drop(pp);
                } else {
                        q_->remove(p);
                        drop(p);
                }
        } else {
// ---------------------------------------------------------------------
// Mark CE bit when the queue length reach the threshold (0.00-1.00) of maximum capacity
// ---------------------------------------------------------------------
            if (q_->length() >= (qlim_ * WRT_thresh_)) {
                hdr_flags* hf = hdr_flags::access(p);
                hf->ce() = 1;
            }
// ---------------------------------------------------------------------
        }
}

Packet* DropTailWRT::deque()
{
    if (summarystats && &Scheduler::instance() != NULL) {
        Queue::updateStats(qib_?q_->byteLength():q_->length());
    }
        return q_->deque();
}

void DropTailWRT::print_summarystats()
{
        //double now = Scheduler::instance().clock();
    printf("True average queue: %5.3f", true_ave_);
    if (qib_)
        printf(" (in bytes)");
```

```
    printf(" time: %5.3f\n", total_time_);
}
```

ก.2 แฟ้ม "drop-tail-WRT.h"

เป็น header file (.h) ที่ใช้ประกอบกับ code ภาษา C++ ของกลไกแบบ Drop-Tail with ECN ซึ่งเป็นส่วนที่กำหนดโครงสร้างและประกาศตัวแปรให้กับกลไกแบบ Drop-Tail with ECN ที่นำเสนอโดยวิทยานิพนธ์ฉบับนี้

```
// This file: drop-tail-WRT.h

#ifndef ns_drop_tail_WRT_h
#define ns_drop_tail_WRT_h

#include <string.h>
#include "queue.h"
#include "config.h"

/*
 * A bounded, drop-tail-WRT queue
 */
class DropTailWRT : public Queue {
 public:
        DropTailWRT() {
                q_ = new PacketQueue;
                pq_ = q_;
                bind_bool("drop_front_", &drop_front_);
                bind_bool("summarystats_", &summarystats);
                bind_bool("queue_in_bytes_", &qib_);  // boolean: q in bytes?
                bind("mean_pktsize_", &mean_pktsize_);
                bind("WRT_thresh_", &WRT_thresh_);
                //                      _RENAMED("drop-front_", "drop_front_");
        }
        ~DropTailWRT() {
                delete q_;
        }
```

```
protected:
        void reset();
        int command(int argc, const char*const* argv);
        void enque(Packet*);
        Packet* deque();
        PacketQueue *q_;        /* underlying FIFO queue */
        int drop_front_; /* drop-from-front (rather than from tail) */
        int summarystats;
        void print_summarystats();
        int qib_;       /* bool: queue measured in bytes? */
        int mean_pktsize_;      /* configured mean packet size in bytes */
        double WRT_thresh_;     /* Warut thresh 0.00 - 1.00 */
};

#endif
```

ก.3 แฟ้ม "10.tcl"

เป็น OTcl script ที่ใช้สร้างเครือข่ายจำลองขึ้นมาเพื่อทดสอบ Drop-Tail with ECN ที่ 75% กับ traffic แบบ A ซึ่งในการทำงานจะมีการเรียกใช้กลไกแบบ Drop-Tail with ECN ที่ถูกแสดงไว้ใน ก.1 ในวิทยานิพนธ์ฉบับนี้ใช้การสร้างเครือข่ายจำลองขึ้นมาทั้งหมด 96 รูปแบบในการทดสอบ

```
#############################################################################
# This file: 10.tcl
# Simulate "DropTailWRT @ 75%"
# Scenario A
# TCP Source 1-10 start from 0.0 - 60.0
# TCP Source 11-25 start from 20.0 - 40.0
# Simulation time = x minutes x seconds
# 10.nam.gz = x MB
# 10.tr = x MB
# Usage: ns 10.tcl

set ns [new Simulator]

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

set nf [open 10.nam w]
$ns namtrace-all $nf

set tf [open 10.tr w]
$ns trace-all $tf

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

### build topology with 52 nodes
# Source 1-10
```

```
for {set i 1} {$i <=10} {incr i 1} {
  set n($i) [$ns node]
}

# Source 11-25
for {set i 11} {$i <=25} {incr i 1} {
  set n($i) [$ns node]
}

# Destination 1-10
for {set i 101} {$i <=110} {incr i 1} {
  set n($i) [$ns node]
}

# Destination 11-25
for {set i 111} {$i <=125} {incr i 1} {
  set n($i) [$ns node]
}

# Router 1-2
for {set i 201} {$i <=202} {incr i 1} {
  set n($i) [$ns node]
}

#############################################################################
# Setup physical nodes

for {set i 1} {$i <=10} {incr i 1} {
  $ns at 0.0 "$n($i) label Source$i"
}

for {set i 11} {$i <=25} {incr i 1} {
```

```
  $ns at 0.0 "$n($i) label Source$i"
}

for {set i 101} {$i <=110} {incr i 1} {
  $ns at 0.0 "$n($i) label Destination[expr $i-100]"
}

for {set i 111} {$i <=125} {incr i 1} {
  $ns at 0.0 "$n($i) label Destination[expr $i-100]"
}

for {set i 201} {$i <=202} {incr i 1} {
  $ns at 0.0 "$n($i) label Router[expr $i-200]"
}

#################################################################################
# Setup Links

for {set i 1} {$i <=10} {incr i 1} {
  $ns duplex-link [set n($i)] $n(201) 45Mb 10ms DropTail
}

for {set i 11} {$i <=25} {incr i 1} {
  $ns duplex-link [set n($i)] $n(201) 45Mb 10ms DropTail
}

for {set i 101} {$i <=110} {incr i 1} {
  $ns duplex-link $n(202) [set n($i)] 45Mb 10ms DropTail
}

for {set i 111} {$i <=125} {incr i 1} {
  $ns duplex-link $n(202) [set n($i)] 45Mb 10ms DropTail
```

```
}

# DropTail with ECN @ 75%
Queue/DropTailWRT set drop_front_ false
Queue/DropTailWRT set WRT_thresh_ 0.75
$ns duplex-link $n(201) $n(202) 45Mb 10ms DropTailWRT

# Buffer size: 2*(45Mb/8)*(6*(10/1000)) = 0.675MB
# Buffer size: 0.675MB / 1500 = 450 packets
$ns queue-limit $n(201) $n(202) 450
$ns queue-limit $n(202) $n(201) 450

#Monitor the queue for link n201-n202 (for NAM)
$ns duplex-link-op $n(201) $n(202) queuePos 0.5

#############################################################################
# Layout nodes

set step [expr 3.14/100]
set degree [expr 3.14/2]

$ns duplex-link-op $n(201) $n(202) orient right

for {set i 1} {$i <=10} {incr i 1} {
  set degree [expr $degree+$step]
  $ns duplex-link-op [set n($i)] $n(201) orient $degree
}

for {set i 11} {$i <=25} {incr i 1} {
  set degree [expr $degree+$step]
  $ns duplex-link-op [set n($i)] $n(201) orient $degree
}
```

```
set degree [expr 3.14*3/4]

for {set i 101} {$i <=110} {incr i 1} {
  set degree [expr $degree-$step]
  $ns duplex-link-op $n(202) [set n($i)] orient $degree
}

for {set i 111} {$i <=125} {incr i 1} {
  set degree [expr $degree-$step]
  $ns duplex-link-op $n(202) [set n($i)] orient $degree
}

#############################################################################
# Setup type of nodes

for {set i 1} {$i <=10} {incr i 1} {
  set tcp($i) [new Agent/TCP/Reno]
  $tcp($i) set window_ 65535
  $tcp($i) set packetSize_ 1460
  $tcp($i) set ssthresh_ 65535
  $tcp($i) set ecn_ 1
  $tcp($i) set fid_ 1
  $ns attach-agent $n($i) $tcp($i)
  set sink($i) [new Agent/TCPSink]
  $ns attach-agent $n([expr $i+100]) $sink($i)
  $ns connect $tcp($i) $sink($i)
  set ftp($i) [new Application/FTP]
  $ftp($i) attach-agent $tcp($i)
}

for {set i 11} {$i <=25} {incr i 1} {
```

```
 set tcp($i) [new Agent/TCP/Reno]
 $tcp($i) set window_ 65535
 $tcp($i) set packetSize_ 1460
 $tcp($i) set ssthresh_ 65535
 $tcp($i) set ecn_ 1
 $tcp($i) set fid_ 2
 $ns attach-agent $n($i) $tcp($i)
 set sink($i) [new Agent/TCPSink]
 $ns attach-agent $n([expr $i+100]) $sink($i)
 $ns connect $tcp($i) $sink($i)
 set ftp($i) [new Application/FTP]
 $ftp($i) attach-agent $tcp($i)
}

#########################################################################
# Begin the script

proc finish {} {

      global ns nf tf
      $ns flush-trace
          close $nf
          close $tf
          exec gzip -f 10.nam
      exec nam 10.nam.gz &
      exit 0
}

### set operations
for {set i 1} {$i <=10} {incr i 1} {
  $ns at 0.0 "$ftp($i) start"
}
```

```
for {set i 11} {$i <=25} {incr i 1} {
  $ns at 20.0 "$ftp($i) start"
}

for {set i 11} {$i <=25} {incr i 1} {
  $ns at 40.0 "$ftp($i) stop"
}

for {set i 1} {$i <=10} {incr i 1} {
  $ns at 60.0 "$ftp($i) stop"
}

$ns at 61.0 "finish"

$ns run
```

ก.4 แฟ้ม "a01.tcl"

เป็น OTcl script ที่มีการเรียกใช้ Awk เพื่อประมวลผล trace file ที่ได้จาก NS-2 ในตัวอย่างนี้ใช้เพื่อประมวลผลจากผลที่ได้จาก Drop-Tail with ECN ที่ 75% เมื่อทดสอบด้วย traffic แบบ A ในวิทยานิพนธ์ฉบับนี้ใช้การสร้างเครือข่ายจำลองขึ้นมาทั้งหมด 96 รูปแบบในการทดสอบ ดังนั้นก็ต้องมีการประมวลผลทั้ง 96 รูปแบบ

```
#############################################################################
# This file: a01.tcl
# Passed bytes & Dropped bytes
# post-processing
# Execution time = x minutes x seconds
# Usage: tcl a01.tcl

set awkCode {
 BEGIN {
  router1 = 50
  router2 = 51
  interval = 0.1
  end_time = 60
  time_slot = end_time / interval
   for(i=0;i<=time_slot;i++) {
    P[i] = 0.0
    D[i] = 0.0
   }
  byte_pass = 0.0
  byte_drop = 0.0
 }
 {
  time_stamp = $2
  time_stamp2 = time_stamp / interval
  t_index = int(time_stamp2)
  if (time_stamp2 > t_index) {
```

```
    t_index += 1
  }
  if($1 == "-" && $3 == router1 && $4 == router2) {
    P[t_index] += $6
    byte_pass += $6
  }
  if($1 == "d" && $3 == router1 && $4 == router2) {
    D[t_index] += $6
    byte_drop += $6
  }
}
END {
  t = 0.0
  for(i=0;i<=time_slot;i++) {
    print t, P[i] >> "/tmp/P.x.tmp"
    print t, D[i] >> "/tmp/D.x.tmp"
    t += interval
  }
  print byte_pass >> "/tmp/O.info.txt"
  print byte_drop >> "/tmp/O.info.txt"
  print byte_drop/byte_pass >> "/tmp/O.info.txt"
 }
}

set f1 [open /tmp/P.x w]
set f2 [open /tmp/D.x w]
set f3 [open /tmp/O.info.txt w]

puts $f1 "TitleText: Scenario A - Passed bytes"
puts $f1 "Device: Postscript"

puts $f2 "TitleText: Scenario A - Dropped bytes"
```

```
puts $f2 "Device: Postscript"


exec rm -f /tmp/P.x.tmp
exec rm -f /tmp/D.x.tmp
exec touch /tmp/P.x.tmp
exec touch /tmp/D.x.tmp


exec awk $awkCode /tmp/I.tr


puts $f1 \"1
exec cat /tmp/P.x.tmp >@ $f1
puts $f1 ""


puts $f2 \"1
exec cat /tmp/D.x.tmp >@ $f2
puts $f2 ""


close $f1
close $f2
close $f3
#exec xgraph -bb -tk -x Time -y Byte /tmp/D.x &
```

ภาคผนวก ข

รายการผลงานวิจัยที่ใช้ Network Simulator 2 (NS-2)

ข.1 งานวิจัยที่เกี่ยวข้องกับ NS เอง

ในหัวข้อนี้แสดงกายการงานวิจัยที่เกี่ยวข้องกับ NS เองในด้าน: the simulator, visualization, emulation, simulation scaling.

[1] Bajaj S., Breslau L., Estrin D., Fall K., Floyd S., Haldar P., Handley M., Helmy A., Heidemann J., Huang P., Kumar S., McCanne S., Rejaie R., Sharma P., Varadhan K., Xu Y., Yu H. and Zappala D. "Improving Simulation for Network Research." Technical Report 99-702, University of Southern California, 1999. (Superceeds an older tech report 98-678: ``Virtual InterNetwork Testbed: Status and Research Agenda".)

[2] Breslau L., Estrin D., Fall K., Floyd S., Heidemann J., Helmy A., Huang P., McCanne S., Varadhan K., Xu Y. and Yu H. "Advances in Network Simulation." IEEE Computer, 2000. pp. 59-67. Superceeds USC tech report 99-702b.

[3] Estrin D., Handley M., Heidemann J., McCanne S., Xu Y. and Yu H. "Network Visualization with the VINT Network Animator Nam." Technical Report 99-703, University of Southern California, 1999.

[4] Fall K. "Network Emulation in the Vint/NS Simulator." 1999.

This paper provides a short description of the issues involved in implementing the emulation facility in the NS simulator. The current version is a draft; it was submitted in Dec., 1998 to ISCC '99.

[5] George F., Riley, Ammar M., and Fujimoto R. "Stateless Routing in Network Simulations." Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, p. to appear. San Francisco, CA, USA, IEEE, 2000.

[6] Heidemann J., Bulusu N., Elson J., Intanagonwiwat C., Lan K., Xu Y., Ye W., Estrin D. and Govindan R. "Effects of Detail in Wireless Network Simulation." SCS Communication Networks and Distributed Systems Modeling and Simulation Conference, 2000.

[7] Huang P., Estrin D. and Heidemann J. "Enabling Large-scale Simulations: Selective Abstraction Approach to The Study of Multicast Protocols."

Submitted for review to Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98)

ข.2 งานวิจัยที่มีการนำ NS ไปใช้

ในหัวข้อนี้แสดงกายการงานวิจัยที่นำ NS ไปประยุกต์ใช้กับการวิจัยด้านต่างๆ

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน TCP**

[1] Bakshi B., Krishna P., Pradhan D., and Vaidya N. "Performance of TCP over Wireless Networks." 17th Intl. Conf. on Distributed Computing Systems (ICDCS), Baltimore, 1997.
(This paper was based on work done using ns version 1)

[2] Balakrishnan H., Padmanabhan V. and Katz R. "The Effects of Asymmetry on TCP Performance" Proc. Third ACM/IEEE Mobicom Conference, Budapest, Hungary, 1997.

[3] Balakrishnan H., Padmanabhan V., Seshan S., Stemm M. and Katz R. "TCP Behavior of a Busy Web Server: Analysis and Improvements." Technical Report UCB/CSD-97-966. (Extended version of paper to appear in Proc. IEEE INFOCOM, San Francisco, CA, 1998).
Simulations were done using ns-1, but the relevant modules have since been ported and worked on in ns-2.

[4] Cardwell N., Savage S. and Anderson T. "Modeling the Performance of Short TCP Connections." University of Washington, 1998.
This paper uses ns simulations and some experiments to compare a few different closed-form models: a parameterized short flow model that assumes zero packet loss, an adaptation of the steady-state model from [Padhye et al 98], and a model for short flows that assumes random loss.

[5] Cohen R. and Ramanathan S. "TCP for High Performance in Hybrid Fiber Coaxial Broad-Band Access Networks." Journal Ton, vol.6, no. 1, 1998. pp 15-29.

[6] Fall K. and Floyd S. "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP." Computer Communications Review, 1996. (Abstract)
This study uses the script ``test-sack'' from the ns distribution.
(This paper was based on work done using ns version 1)

[7] Floyd S. and Fall K. "Router Mechanisms to Support End-to-End Congestion Control." LBNL Technical Report

[8] Floyd S. "Issues of TCP with SACK." Technical report, 1996.
(This paper was based on work done using ns version 1)

[9] Henderson T. and Katz R. "On Improving the Fairness of TCP Congestion Avoidance." Proc. IEEE Globecom `98.

[10] Lakshman T., Madhow U. and Suter B. "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance." INFOCOM'97.

This paper is based on work with modified versions of ns-1.

[11] Low S., Peterson L. and Wang L. "Understanding TCP Vegas: A Duality Model"

It presents a model of the TCP Vegas congestion control mechanism as a distributed optimization algorithm.

[12] Mahdavi N. "The Rate-Halving Algorithm for TCP Congestion Control"

It includes a copy of the current working draft, a tarfile with sources for the "ns" simulator, and a page with some sample comparisons between RH, Reno, and NewReno.

[13] Mathis M. and Mahdavi J. "Forward Acknowledgement: Refining TCP Congestion Control." SIGCOMM 96, Stanford, CA, 1996.

(This paper was based on work done using ns version 1)

[14] Suter B., Lakshman T., Stiliadis D. and Choudhury A. "Design Considerations for Supporting TCP with Per-flow Queueing." INFOCOM'98.

This paper is based on work with modified versions of ns-1.

[15] Visweswaraiah V. and Heidemann J. "Improving Restart of Idle TCP Connections." Technical Report 97-661, University of Southern California, 1997.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Diff-serve/int-serve**

[1] Clark D. and Fang W. "Explicit Allocation of Best Effort Packet Delivery Service." ACM Transactions on Networking, 1998.

This paper use ns-1 to evaluate the algorithms to be used in interior routers and edge routers. I am in the process of porting modules to ns-2.

[2] Oechslin P. and Crowcroft J. "Differentiated End to End Internet Services using a Weighted Proportional Fair Sharing TCP."

This paper has been submitted to ACM CCR.

Code used for the work done in this paper is based on ns2.1b1 and is available from the contributed code page under "Contributed code to ns-2".

[3] Welzl M. "A Stateless QoS Signaling Protocol for the Internet." ICPADS'00 (International Conference on Parallel and Distributed Systems - IEEE Computer Society), Iwate, Japan, 2000.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Scheduling or queue management in routers**

[1] Legout A. and Biersack E. "Revisiting the Fair Queueing Paradigm for End-to-End Congestion Control." IEEE Network Magazine, 2002.

[2] Sanneck H. and Carle G. "Predictive Loss Pattern Queue Management for Internet Routers."

[3] Ziegler T. and Clausen H. "Congestion Avoidance with BUC (Buffer Utilization Control Gateways) and RFCN (Reverse Feedback Congestion Notification)."
Submitted to International Phoenix Conference for Computers and Communication (IPCCC), Scottsdale, Arizona, 1997.
(This paper was based on work done using ns version 1)

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Multimedia**

[1] Kouvelas I., Hardman V. and Crowcroft J. "Network Adaptive Continuous-Media Applications Through Self Organised Transcoding."
This paper has been submitted to NOSDAV '98.
Code used for the work done in this paper is based on ns2.1b1 and is available from the contributed code page under "Contributed code to ns-2".

[2] McCanne S., Jacobson V. and Vetterli M. "Receiver-driven Layered Multicast." SIGCOMM '96, Stanford, CA, 1996.
(This paper was based on work done using ns version 1)

[3] McCanne S. "Scalable Compression and Transmission of Internet Multicast Video." Ph.D. Dissertation, University of California, Berkeley, 1996.

[4] Rejaie R., Handley M. and Estrin D. "Quality Adaptation for Congestion Controlled Video Playback over the Internet." SIGCOMM'99, Boston, MA, 1999.

[5] Welzl M. "Adaptive Multimedia Communication over Satellite Routed IP." ICC 2000 (International Conference on Communications - IEEE Communications Society), New Orleans, Louisiana, USA, 2000.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Multicast**

[1] Birman K., Hayden M., Ozkasap O., Xiao Z., Budiu M. and Minsky Y. "Bimodal Multicast." ACM Transactions on Computer Systems, 1999. pp. 41-88. Also available as Technical Report, Department of Computer Science, Cornell University, TR99-1745, 1999.

[2] Haenle C. and Hofmann M. "A Comparison of Reliable Multicast Protocols using the Network Simulator ns-2." Proceedings of IEEE Conference on Local Computer Networks (LCN), Boston, MA, USA, 1998.

[3] Helmy A. and Estrin D. "'STRESS' Testing Applied to a Multicast Routing Protocol." Tech Report USC-CS-TR 97-657, Department of Computer Science, University of Southern California.
(Also submitted for review to infocom '98).

[4] Legout A. and Biersack E. "PLM: Fast Convergence for Cumulative Layered Multicast Transmission Schemes." Proceedings of ACM SIGMETRICS'2000, Santa Clara, California, USA, 2000.

[5] Legout A. and Biersack E. "Pathological Behaviors for RLM and RLC." Proceedings of NOSSDAV'2000, Chapel Hill, North Carolina, USA, 2000.

[6] Ozkasap O., Xiao Z. and Birman K. "Scalability of Two Reliable Multicast Protocols." Technical Report, Department of Computer Science, Cornell University, TR99-1745, 1999.

[7] Ozkasap O., Renesse R., Birman K. and Xiao Z. "Efficient Buffering in Reliable Multicast Protocols." Proceedings of NGC99 (Networked Group Communication), Lecture Notes in Computer Science 1736, Springer Verlag, Pisa, Italy, 1999. pp. 188-203.
Also available as Technical Report, Department of Computer Science, Cornell University, TR99-1757, 1999.

[8] Varadhan K., Estrin D., and Floyd S. "Impact of Network Dynamics on End-to-End Protocols: Case Studies in Reliable Multicast." IEEE Symposium on Computers and Communications, Athens, Greece, 1998.

[9] Vicisano L., Rizzo L. and Crowcroft J. "TCP-like Congestion Control for Layered Multicast Data Transfer." INFOCOM '98.
Code used for the work done in this paper is based on ns2.1b1 and is available from the contributed code page under "Contributed code to ns-2".

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Internet Traffic Modeling**

[1] Feldmann A, Gilbert A., Huang P. and Willinger W. "Dynamics of IP traffic: A study of the role of variability and the impact of control." SIGCOMM'99, Boston, MA, 1999.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Web Caching**

[1] Yu H., Breslau L. and Shenker S. "A Scalable Web Cache Consistency Architecture." SIGCOMM'99, Boston, MA, 1999.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Wireless Sensor Networks**

[1] Intanagonwiwat C., Govindan R. and Estrin D. "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks." MobiCOM 2000, Boston, MA, 2000.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Wireless-Networked Mobile Robots**

[1] Ye W., Vaughan R., Sukhatme G., Heidemann J., Estrin D. and Mataric M. "Evaluating Control Strategies for Wireless-Networked Robots Using an Integrated Robot and Network Simulation." Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001), Seoul, Korea, 2001.

- **งานวิจัยที่มีการนำ NS ไปใช้ในด้าน Satellite Networks**

[1] Connors D. and Pottie G. "The Performance of HTTP Over Satellite Random Access Channels." Proceedings of 2000 Western MultiConference (WMC2000), 2000.

[2] Henderson T. and Katz R. "UC Berkeley Transport protocols for Internet-compatible satellite networks." IEEE Journal on Selected Areas in Communications, vol. 17, no. 2, 1999. pp. 345-359.

[3] Henderson T. and Katz R. "UC Berkeley Network Simulation for LEO Satellite Networks." 18th AIAA International Communications Satellite Systems Conference (ICSSC), Oakland, CA, 2000.

[4] Henderson T. and Katz R. "UC Berkeley On Distributed, Geographic-based Packet Routing for LEO Satellite Networks." Proceedings of IEEE Globecom 2000, San Francisco, CA, 2000.

[5] Karaliopoulos M., Tafazolli R. and Evans B. "Enhancing TCP's performance over GEO satellite links with split- connections and link level retransmissions." London Communications Symposium, London, UK, 2000.

[6] Karaliopoulos M., Tafazolli R. and Evans B. "TCP Performance on Split Connection GEO Satellite Links." Proceedings of the 19th AIAA International Communications Satellite Systems Conference (ICSSC '01), Toulouse, France, 2001.

[7] Koyabe M. and Fairhurst G. "Performance of Reliable Multicast Protocols via Satellite at EHF with Persistent Fades." 7th Ka-Band Utilization Conference, Santa Margherita Ligure, Genoa, Italy, 2001.

[8] Wood L. "Internetworking with satellite constellations." PhD thesis, Centre for Communication Systems Research, University of Surrey, 2001.

[9] Wood L., Pavlou G. and Evans B. "Effects on TCP of routing strategies in satellite constellations." Special issues on Satellite-Based Internet Technology and Services, IEEE Communications Magazine, vol. 39, no. 3, 2001. pp. 172-181.

[10] Wood L., Pavlou G. and Evans B. "Managing diversity with handover to provide classes of service in satellite constellation networks." Proceedings of AIAA International Communication Satellite Systems Conference (ICSSC '01), vol. 3, session 35, no. 194, Toulouse, France, 2001.

ภาคผนวก ค
ผลงานที่ได้รับตีพิมพ์

ผู้แต่ง:	Piyatamrong B. and Anutchalakhom W.
เรื่อง:	Performance Evaluation on Drop-Tail and RED Gateways
ตีพิมพ์ในวารสาร:	Digital Signal Processing and Its Applications
สถานที่:	Moscow, Russia
เป็นเอกสารลำดับที่:	1 (Proceedings: 1)
วันที่:	March 2003
หน้าที่:	287-290

PERFORMANCE EVALUATION ON DROP-TAIL AND RED GATEWAYS

Piyatamrong B., Anutchalakhom W.

Department of Computer Engineering, Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Chalongkrung Rd., Ladkrabang, Bangkok 10520, Thailand
Phone: (662) 326-9969 Ext. 107, E-mail: kpbanjon@kmitl.ac.th, warut@octl.com

Abstract

The Drop-Tail and RED are well-known queue types deploy on gateways nowadays. RED is a newer queue type with the active queue management feature. They both manage to pass or drop the packet on different ways. A question on this 2 types is what case we choose to deploy which one. This paper uses simulations to explore the behaviors of Drop-Tail and RED gateways on various kind of traffic in term of throughput and packet dropping. We include both Reno TCP and SACK TCP, the last two standard implementations for TCP, in our simulations. The results show Drop-Tail performs better than RED on most case in our environments.

1. Introduction

In the world of Internet, TCP and IP are 2 key protocols. IP is used as a main protocol in network layer. TCP positions in transport layer that on top of IP with features of flow control, reliability and connection-oriented mechanism [2]. To improve the performance of TCP, the upper application layer protocols will receive the benefits also. The window-based end-to-end mechanism that used to maintain throughput in the network does not need cooperation from a gateway that just passes a packet across the network interfaces. TCP itself probes the network state to determine a size of congestion window [3] to utilize the network link bandwidth.

In [1, 4], Explicit Congestion Notification (ECN) has been proposed to appear on IP header and deploy onto IP gateways or IP routers for sending congestion notification comes along with IP header to notify congestion in the network. Firstly, ECN is used to enhance an active management queue, Random Early Detection (RED) [5, 6]. RED is still limited to deploy in the real world because of lack of information to setting 4 parameters (minimum

threshold min_{th}, maximum threshold max_{th}, maximum drop probability max_p, queue weight w_q) to have optimal performance [7, 8].

2. The Explicit Congestion Notification

To support the ECN mechanism, totally 4 more bits on both IP and TCP header are redefined. The 2 bits on the IP header are ECN-Capable Transport (ECT) and Congestion Experienced (CE). The ECT provides a flag to notify that both the end systems are ECN capabled and the CE is set by a gateway when congestion occurs. The others 2 more bits on the TCP header are ECN Echo (ECE) and the Congestion Window Reduced (CWR) .The ECE bit is a flag that is bounced back to notify that the congestion is occurred and the CWR signal is used to confirm that the congestion window has been reduced. The order of the flows in the ECN mechanism is shown in Figure 1.

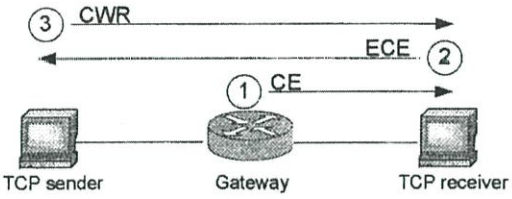


Fig. 1. Ordering flows of ECN mechanism.

The Drop-Tail with ECN (DT-ECN) is expressed in the term of algorithm as the following.

```
        ECN_threshold = MAX_queue_size *
(Queue_utilization / 100)
        for each packet arrival
        if queue_length < MAX_queue_size
                if queue_length+1 >=
ECN_threshold
                        mark CE bit on the arriving
packet
                enqueue the packet
        else
                drop the arriving packet
```

Where:

ECN_threshold: level of queue that begins to mark CE bit.

MAX_queue_size: maximum queue size.

Queue_utilization: percentage of queue that begins to mark CE bit.

queue_length: size of in used queue.

3. The Simulation Processes

The simulation process is done by using a wide area network (WAN) topology that consists of a group of sources on the left that send data via 2 routers to a group of destinations on the right, as shown in Figure 2. The environmental setting are:

- All network links have the same 45Mbps bandwidth (T3 standard).
- The propagation delay is set to 10 msec.
- The TCP senders are implemented for both TCP styles, i.e. TCP-Reno and TCP Selective Acknowledgment (SACK).
- All TCP connections called by FTP applications that always have data to send (bulk-data connection).
- The maximum segment size (MSS) of TCP is set 1460 bytes.
- The TCP receiver's buffer size is set to the maximum 16-bit window size TCP protocol field.
- The ssthresh variable of TCP is initialized to 65536 bytes.
- All 2 routers buffer sizes are set to twice of the bandwidth-delay product (BDP) of the network (675 kbytes).
- The ECN thresholds are set to 25%, 50% and 75% of the buffer size respectively.

The Network Simulator version 2.1b9a (NS-2) is an object-oriented, discrete event driven network that used in this paper. With the above conditions, a newly queue object module, DT-ECN, is implemented in C++ language to incorporate with the NS-2 framework and OTcl scripts are written to setting up the topology, the object values and the time that all the events occurred. After running the codes, the results are stored in the term of a trace file that contains information of time, position of a packet, destination of a packet, type of packet and any useful

information. The trace file is then passed the filtering and calculating processes using Awk scripts to generating the final results.

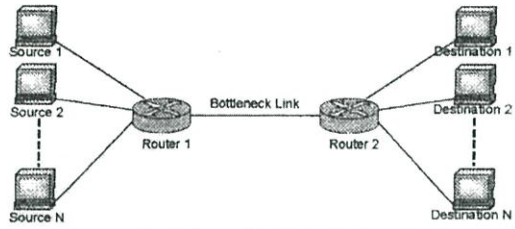


Fig. 2. Network topology for WAN in this simulation.

Various traffic patterns are used for the simulation. For instance:

- The total number of connections are varied from a very little to a very massive connections says 25 – 100 connections.

- The arrival and the departure of the number of connections is fluctuated from a little to a huge and falling down from a huge to a little number of connections and the final case is that all the connections enter at all time.

As defined in 6 different scenarios as A, B, C, D, E and F.

- Scenario A is a case that there are 10 connections at all the time of the simulation (0.0 seconds – 60.0 seconds) but in the middle of the period of time (20.0 seconds – 40.0 seconds) we increase traffic by 15 connections for 20 seconds.

- Scenario B is a case that the connections enter the network increasingly starting from 10 connections, 25 connections, and double to 50 connections.

- Scenario C is a case that inverses the scenario B by all the 50 connections enter the network aggressively at 0.0 seconds and decrease to 35 connections and 10 connections.

- Scenario D is a case that looks like scenario B with double of connection.

- Scenario E is a case that looks like scenario C with double of connection.

- The Last, Scenario F is a most aggressively case that all the 100 connections enter the network at all the 60.0 seconds period of time.

On the four RED parameters, the recommendation from Floyd in [5] as min_{th} = 1/4 of queue size, $max_{th} = 2 * min_{th}$, $max_p = 0.02$, $w_q = 0.002$ is employed.

4. Simulation Results

The measurement of the performance of the all 6 types of queue by summary of data that successful sent in byte (Bytes sent) and summary of data that has been dropped in byte (Bytes dropped). The results in term of value are shown in Table 1, 2, 3 and 4.

Table 1. Total Bytes sent on TCP Reno.

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	RED w ECN	RED w/o ECN
	329588500	317098000	328577500	327248500	324961000	325607500
	329532500	315795500	328050500	328442000	323399000	322890500
	331103000	312348500	328025000	330624500	330620000	330560000
	330272500	313489000	329890000	329813500	326879500	324503500
	331523500	312826000	334180000	331286500	331148500	330953500
	332239000	312853000	333370000	331669000	331747000	331477000

Table 2. Total Bytes dropped on TCP Reno.

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	RED w ECN	RED w/o ECN
	1695000	0	342000	681000	1122000	1972500
	2694000	0	342000	880500	2442000	3708000
	2680500	0	342000	681000	2160000	3636000
	5410500	0	409500	988500	6291040	7578040
	5607000	0	342000	681000	7126500	8013000
	9472500	0	342000	681000	12328500	13095000

Table 3. Total Bytes sent on TCP SACK.

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	RED w ECN	RED w/o ECN
	335884000	317098000	336022000	335054500	335314000	333788500
	336149000	315795500	335399000	336561500	335573000	335276000
	336572000	312348500	332624000	336834500	334751000	334389500
	336428500	313489000	334150000	336919000	334721500	333767500
	336572500	312826000	333824500	336790000	335284000	336010000
	336985000	312853000	333311500	337174000	336437500	336433000

Table 4. Total Bytes dropped on TCP SACK.

	Drop-Tail	DT-ECN @ 25%	DT-ECN @ 50%	DT-ECN @ 75%	RED w ECN	RED w/o ECN
	1917000	0	342000	757500	1243500	2095500
	3351000	0	342000	991500	2302500	4021500
	2737500	0	342000	681000	2197500	3790500
	3729880	0	447000	859500	8332540	9576040
	5496000	0	342000	681000	9616500	10912500
	9306000	0	342000	681000	17812500	19065000

5. Conclusions

This paper evaluated Drop-Tail and RED gateways with bulk-data connection on WAN topology. On TCP, we used both Reno TCP and SACK TCP for covering most case of implemented TCP. We also added ECN onto both queue types for complete comparison. Firstly, Drop-Tail without ECN always performs better than RED without ECN in term of throughput and packet dropping. Secondly, Drop-Tail with ECN at 75% performs better than RED with ECN from 10 of 12 cases in term of throughput (Reno: A, B, C, D, E and SACK: B, C, D, E, F) and always performs better in term of packet dropping. The result is point out that Drop-Tail is still valuable to deploy at least on this environments.

References

[1] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, Vol. 24 No. 5, October 1994, pp. 10-23.

[2] Information Sciences Institute University of Southern California, "TRANSMISSION CONTROL PROTOCOL," *Request for Comments (RFC) 793*, September 1981.

[3] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of SIGCOMM 1988*, August 1988, pp. 314-329.

[4] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," *Request for Comments (RFC) 3168*, September 2001.

[5] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, August 1993.

[6] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," *Request for Comments (RFC) 2309*, April 1998.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A Self-Configuring RED Gateway," *INFOCOM '99*, March 1999.

[8] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to Deploy RED," *tech. rep.*, INRIA, France, June 1999.

ประวัติผู้เขียน

นายวรุต อนุชชาลาคม เกิดเมื่อวันที่ 24 กุมภาพันธ์ พ.ศ. 2517 ที่จังหวัดชลบุรี สำเร็จการศึกษาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2538 ด้านการทำงาน ในปี พ.ศ. 2539 ได้เริ่มเข้าทำงานในตำแหน่ง System Engineer ที่บริษัท Nation Information Technology ถัดมาในปี พ.ศ. 2540 เข้าทำงานในตำแหน่ง Product Developer ที่บริษัท Logic จากนั้นในปี พ.ศ. 2541 เข้าทำงานในตำแหน่ง System Engineer ที่บริษัท MFEC และในปี พ.ศ. 2543 ถึงปัจจุบัน ทำงานในตำแหน่ง Technical Consultant ที่บริษัท Open Computing Technologies Limited (OCT)

ประสบการณ์การทำงานที่ผ่านมาส่วนใหญ่เป็นการออกแบบและติดตั้งระบบคอมพิวเตอร์ UNIX ขนาดใหญ่, ระบบเครือข่าย, ระบบรักษาความปลอดภัย และระบบ Internet