การจัดรูปนัยของโพรโทคอลทีซีพีโดยใช้คาเฟ่โอบีเจ

นางสาวสุชาดา จุลจันทร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2544

ISBN 974-03-1274-8

FORMALIZATION OF THE TCP PROTOCOL USING CAFEOBJ

Miss Suchada Junchan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2001

ISBN 974-03-1274-8

หัวข้อวิทยานิพนธ์	การจัดรูปนัยของโพรโทคอลทีซีพีโดยใช้คาเฟ่โอบีเจ
โดย	นางสาวสุชาดา จุลจันทร์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	รองศาสตราจารย์. ดร.วันชัย ริ้วไพบูลย์
อาจารย์ที่ปรึกษาร่วม	ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

..... อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์)

..... อาจารย์ที่ปรึกษาร่วม

(ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ)

..... กรรมการ

(อาจารย์ ดร.อานนท์ รุ่งสว่าง)

สุชาดา จุลจันทร์ : การจัดรูปนัยของโพรโทคอลทีซีพีโดยใช้คาเฟ่โอบีเจ. (FORMALIZATION OF THE TCP PROTOCOL USING CAFEOBJ) อ. ที่ปรึกษา : รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์, อ.ที่ปรึกษาร่วม : ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ 134 หน้า. ISBN 974-03-1274-8.

วิทยานิพนธ์นี้เสนอวิธีการใหม่ในการกำหนดรูปนัยโพรโทคอลทีซีพีด้วยภาษาคาเฟ่โอบีเจ โดยใช้แผนภาพเอดีเจ และแผนภาพการเปลี่ยนสถานะ ในการเขียนข้อกำหนดโพรโทคอลทีซีพี

วิธีการจัดรูปนัยใหม่นี้สามารถกำหนดหลักนามธรรมข้อมูลของโพรโทคอลทีซีพีด้วยแผนภาพเอดีเจ และสามารถแสดงกระบวนการทำงานของทีซีพีได้จากแผนภาพการเปลี่ยนสถานะ อีกทั้งมีการใช้แผนภาพเส้นกำหนดเวลาเพื่อแสดงถึงการรับและส่งข้อมูลระหว่างทีซีพี รวมไปถึงกฎการเชื่อมแผนภาพเอดีเจ แผนภาพการเปลี่ยนสถานะ และแผนภาพเส้นกำหนดเวลาเข้าด้วยกัน

ข้อกำหนดโพรโทคอลทีซีพีนี้ยึดมาตรฐานตามอาร์เอฟซี793 โดยที่ข้อกำหนดไม่ครอบคลุมกรณีที่ผิดพลาด กรณีมีตัวชี้บ่งการเร่ง และกรณีหมดเวลา ซึ่งวิธีรูปนัยศึกษากรณีการ-ติดตั้งการเชื่อมต่อ การรับส่งข้อมูล และการยกเลิกการติดต่อ รวมไปถึงข้อกำหนดของระบบการ-เรียกใช้งานจากผู้ใช้ผ่านคำสั่งเรียกไปยังโพรโทคอลทีซีพี ข้อกำหนดดังกล่าวกำหนดและทวนสอบด้วยภาษาคาเฟ่โอบีเจ ซึ่งผลลัพธ์ได้รับการตรวจสอบในส่วนวากยสัมพันธ์และการพิสูจน์ทางคณิตศาสตร์ด้วยภาษาคาเฟ่โอบีเจเช่นกัน

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2544 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

4270686321 : MAJOR COMPUTER ENGINEERING

KEY WORD: TCP / FORMALIZATION / CAFEOBJ / THE STATE TRANSITION DIAGRAM / FORMAL METHOD

SUCHADA JUNCHAN : FORMALIZATION OF THE TCP PROTOCOL USING CAFEOBJ. THESIS ADVISOR : ASSOC. PROF. WANCHAI RIVEPIBOON, THESIS COADVISOR : ASST. PROF. WIWAT VATANAWOOD, 134 pp.

ISBN 974-03-1274-8.

This thesis proposes new formalization of the TCP protocol specification using CafeOBJ formal language. This technique uses the ADJ Diagram and the State Transition Diagram for specifying the complexity and variety services of the TCP Protocol.

This new technique can specify both the data abstraction and the system procedure which are important for specifying and verifying complex system. The data abstraction of the TCP protocol is defined by using ADJ Diagram and the procedure of this specification is analyzed by using the State Transition Diagram. In addition, the combination rules of ADJ Diagram and State Transition Diagram are provided. Furthermore, the flow of sending and receiving data between two TCPs illustrate by using Timed Lined Diagram. In the same manner, the combination rules of the ADJ Diagram, State Transition Diagram and Timed Lined Diagram are provided.

The connection establishment, data transfer and termination of TCP specification and user calls system specification are specified and verified according to RFC793. Ignorance timeout interval, no urgent and no error are provided. The results show the TCP specification is syntactically verified by using CafeOBJ Interpreter.

Department Computer Engineering Student's signature .....

Field of study Computer Engineering Advisor's signature .....

Academic year 2001 Co-advisor's signature .....

กิตติกรรมประกาศ

ข้าพเจ้าใคร่ขอกราบขอบพระคุณรองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า และผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ อาจารย์ที่ปรึกษาร่วม ที่ท่านทั้งสองเป็นให้ความรู้ คำปรึกษา พร้อมชี้แนะแนวทาง ตลอดจนให้ความช่วยเหลือต่างๆ ในการทำวิจัยของข้าพเจ้าอย่างดียิ่งจนสำเร็จลุล่วงด้วยดี

ขอกราบขอบพระคุณอาจารย์ในห้องปฏิบัติบัติการวิศวกรรมซอฟต์แวร์ ผู้ช่วยศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี และ อาจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์ ที่ให้คำแนะนำข้อคิดเห็นต่างๆ ในการทำวิทยานิพนธ์ รวมถึงดูแลด้านอื่นๆ อีกด้วย ซึ่งข้าพเจ้ามิอาจลืมเลือนได้

ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่ข้าพเจ้าได้รับความสะดวกสบายจากสิ่งแวดล้อมรวมทั้งเครื่องมือต่างๆ ในห้องปฏิบัติการวิศวกรรมซอฟต์แวร์ ซึ่งทำให้ข้าพเจ้าทำงานได้อย่างมีประสิทธิภาพ

ขอขอบคุณบรรดาเพื่อนๆ พี่ๆ น้องๆ ที่ได้ให้คำแนะนำ และกำลังใจแก่ข้าพเจ้าตลอดเวลาที่ศึกษาในภาควิชาวิศวกรรมคอมพิวเตอร์แห่งนี้

ท้ายที่สุด ข้าพเจ้าใคร่ขอกราบขอบพระคุณบิดา มารดา พี่ และน้อง ที่ให้ความรัก กำลังใจ และคำแนะนำอันเป็นประโยชน์ต่องานวิจัย ตลอดจนให้การอุปการะแก่ข้าพเจ้าเสมอมา

สุชาดา จุลจันทร์

สารบัญ

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

หน้า

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การพัฒนาซอฟต์แวร์คอมพิวเตอร์ในปัจจุบันมีการเขียนข้อกำหนดต่าง ๆ ไม่ว่าจะเป็น ข้อกำหนดระบบ (System Specification) หรือ ข้อกำหนดโปรแกรม (Program Specification) เพื่อใช้อธิบายพฤติกรรมของระบบ ซึ่งวิธีเขียนข้อกำหนดที่ได้รับความนิยมคือ วิธีอรูปนัย (Informal Method) และ วิธีกึ่งรูปนัย (Semi-Formal Method) โดยใช้ภาษาธรรมชาติ (Natural Language) หรือ แผนภาพ (Diagram) ในการอธิบายพฤติกรรมของระบบ ทั้งสองวิธีนี้ประสบปัญหาด้านแปลความหมายและการทำความเข้าใจต่อความต้องการของผู้ใช้งาน เนื่องจากตัวภาษาธรรมชาติเองมีความกำกวม ดังนั้นจึงเกิดเป็นวิธีรูปนัย (Formal Method) [1] ขึ้น วิธีรูปนัยนี้พยายามลดความกำกวมของภาษาธรรมชาติ โดยการนำสัญกรณ์ (Notation) และทฤษฎีทางคณิตศาสตร์มาเขียนข้อกำหนดให้ชัดเจนและเข้าใจตรงกัน โดยเรียกข้อกำหนดที่ได้ว่า "ข้อกำหนดรูปนัย" (Formal Specification) วิธีรูปนัยนี้ยังสามารถพิสูจน์ข้อกำหนดที่เขียนขึ้นว่ามีความถูกต้องมากน้อยเพียงใดด้วยการพิสูจน์ตามหลักคณิตศาสตร์

วิธีรูปนัย (Formal method) พัฒนามากว่า 50 ปี โดยในระยะ 15 ปีที่ผ่านมา งานวิจัยเน้นประเด็นความสนใจในด้านข้อกำหนด (Specification) และด้านการทวนสอบ (Verification) ของระบบมากเป็นพิเศษ โดยเฉพาะ "ระบบการสื่อสาร" ซึ่งเป็นระบบที่ได้รับความสนใจมากที่สุด เนื่องจากระบบนี้มีวิธีการทำงานที่ซับซ้อนและใช้งานกันอย่างกว้างขวาง จึงจำเป็นอย่างยิ่งที่ต้องการข้อกำหนดที่ไม่คลุมเครือ กระชับ และชัดเจน

วิธีรูปนัยที่ใช้กำหนดระบบหนึ่งๆ นั้น สิ่งสำคัญที่ต้องพิจารณาคือ รูปแบบระบบที่นำมากำหนดมีกระบวนการทำงานอย่างไร การเลือกวิธีรูปนัยที่เหมาะสำหรับการจัดรูปนัยระบบดังกล่าว รวมไปถึงภาษาข้อกำหนดรูปนัยที่เลือกใช้ งานวิจัยที่ผ่านๆ มาจึงมีการเสนอวิธีรูปนัยและการทวนสอบความจริงของข้อกำหนดของระบบการติดต่อสื่อสารเป็นจำนวนมาก เช่น วิธีกระบวนการลำดับการสื่อสาร (Communicating Sequential Process) [2] วิธีเครื่องสถานะจำกัด (Finite State Machine) [3] วิธีการเปลี่ยนสถานะ (State Transition Technique) [4] และอื่นๆ ซึ่งแต่ละวิธีมีความเหมาะสมต่อระบบแตกต่างกันออกไป

ในงานวิจัยนี้ กำหนดรูปนัยของโพรโทคอลทีซีพี เนื่องจากโพรโทคอลนี้ที่มีความน่าเชื่อถือในการติดต่อสื่อสารในระบบเครือข่ายและใช้งานกันอย่างกว้างขวางมากที่สุด ภาษาที่เลือกในการ-

จัดรูปนัยนี้ คือ ภาษาคาเฟ่โอบีเจ (CafeOBJ) และจัดรูปนัยด้วยวิธีการจัดรูปนัยใหม่ โดยใช้แผนภาพเอดีเจ (ADJ Diagram) แผนภาพการเปลี่ยนสถานะ (State Transition Diagram) และ แผนภาพเส้นกำหนดเวลา (Timed Lined Diagram)

1.2 วัตถุประสงค์ของงานวิจัย

เขียนข้อกำหนดทีซีพีจากแผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะโดยใช้ภาษาข้อกำหนดรูปนัยคาเฟ่โอบีเจ

1.3 ขอบเขตของงานวิจัย

1.3.1 ใช้แผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะในการกำหนดโพรโทคอลทีซีพีตามอาร์เอฟซี 793 (RFC793 [8])

1.3.2 ใช้ภาษาคาเฟ่โอบีเจในการเขียนข้อกำหนดโพรโทคอลทีซีพีจากแผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะ

1.3.3 ข้อกำหนดโพรโทคอลทีซีพีนี้พิจารณาการรับส่งข้อมูลที่เสร็จสมบูรณ์ ไม่มีข้อผิดพลาดในการรับส่งของข้อมูล และในการรับส่งข้อมูลไม่พิจารณาขอบเขตของการหมดเวลา (timeout interval)

1.3.4 ข้อกำหนดโพรโทคอลทีซีพีนี้ครอบคลุมการเรียกใช้งานจากผู้ใช้ (user) โดยใช้คำสั่งงาน (command) "เปิด" (open), "ส่ง" (send), "รับ" (receive), "ปิด" (close) และ "ยกเลิก" (abort) ส่วนต่อประสาน (interface) ระหว่างผู้ใช้ (user) และทีซีพี (TCP) ตามรายละเอียดขั้นตอนการทำงาน

1.3.5 งานวิจัยนี้ครอบคลุม

1.3.5.1 การเริ่มการติดต่อของโพรโทคอลทีซีพี (Connection Establishment)

1.3.5.2 การส่งข้อมูล (Data Transfer)

1.3.5.3 การยกเลิกการติดต่อของโพรโทคอลทีซีพี (Termination)

1.3.6 งานวิจัยนี้ทำการตรวจสอบผลลัพธ์ที่ได้โดย

1.3.6.1 พิสูจน์ข้อกำหนดทีซีพีด้วยภาษาคาเฟ่โอบีเจ

1.3.6.2 ตรวจสอบการเรียกคำสั่งงานจากรายละเอียดในอาร์เอฟซี 793

1.3.6.3 อธิบายกระบวนการการทำงานที่จำเป็นในข้อกำหนดทีซีพีนี้ด้วยตารางการตัดสินใจ (Decision Table)

1.4 ขั้นตอนและวิธีดำเนินงานวิจัย

1.4.1 ศึกษาโพรโทคอลทีซีพี

1.4.2 ออกแบบแผนภาพเอดีเจ

1.4.3 เขียนกำหนดด้วยภาษาคาเฟ่โอบีเจ

1.4.4 ทวนสอบด้วยภาษาคาเฟ่โอบีเจ

1.4.5 สรุปและเขียนวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 ข้อกำหนดโพรโทคอลทีซีพี

1.5.2 วิธีกำหนดข้อกำหนดใหม่ในระบบเฉพาะ (domain specific application)

1.5.3 ช่วยในการจำลองโปรแกรมที่เรียกใช้งานโพรโทคอลทีซีพี

1.5.4 ช่วยในการออกแบบทีซีพีเวอร์ชันอื่นๆ

บทที่ 2

แนวคิดและทฤษฎีที่เกี่ยวข้อง

2.1 คาเฟ่โอบีเจ (CafeOBJ)

ภาษาคาเฟ่โอบีเจ (CafeOBJ) [5,6] เป็นภาษาข้อกำหนดที่พัฒนามาจากภาษาโอบีเจ (OBJ) ซึ่งอธิบายข้อกำหนดระบบในเชิงพีชคณิต (Algebraic Language) เป็นภาษาที่ออกแบบมาเพื่อระบบติดต่อสื่อสารโดยเฉพาะมีการใช้ตรรกะ (Logic) ในการอธิบายระบบได้อย่างชัดเจน อีกทั้งยังมีตรรกะที่เพิ่มเข้าไปใหม่ในภาษานี้ คือ การเขียนตรรกะใหม่ (Rewriting Logic) และพีชคณิตเชิงพฤติกรรม (Behavioral Algebra) โดยการเขียนตรรกะใหม่ใช้เขียนอธิบายกระบวนการที่ทำงานอย่างพร้อมเพรียงกัน (Concurrent Process) และพีชคณิตเชิงพฤติกรรมใช้เขียนข้อกำหนดเชิงพฤติกรรม (Behavioral Specification) ตรรกะทั้งสองที่เพิ่มขึ้นนี้เป็นส่วนที่ใช้ในการเขียนข้อกำหนดรูปนัยเชิงวัตถุ (Formal Object Oriented Specification)

ข้อกำหนดรูปนัยคาเฟ่โอบีเจมีลักษณะที่สำคัญ ดังต่อไปนี้

- การโปรแกรมและข้อกำหนดการเท่ากัน (Equational Specification and Programming) เป็นวิธีที่รับมาจากภาษาโอบีเจ และเป็นพื้นฐานของภาษาเชิงพีชคณิต ใช้อธิบายโปรแกรมเชิงหน้าที่ (Functional Programming)
- ข้อกำหนดการเขียนตรรกะใหม่ (Rewriting Logic Specification) เป็นวิธีที่ใช้อธิบายกระบวนการการทำงานที่ทำงานพร้อมๆ กัน
- ข้อกำหนดเชิงพฤติกรรม (Behavioral Specification) เป็นข้อกำหนดที่อธิบายถึงพฤติกรรมของวัตถุ โดยใช้ชนิดข้อมูลแฝง (Hidden Sort) และแนวคิดเชิงพฤติกรรม (Behavioral Concept)
- การปรับให้เป็นเชิงวัตถุ (Object Orientation) เป็นวิธีที่ใช้เขียนข้อกำหนดเชิงวัตถุ ซึ่งมีวิธีที่ใช้ในการอธิบาย 2 แบบคือ การเขียนตรรกะใหม่ (Rewriting Logic) และข้อกำหนดเชิงพฤติกรรม (Behavioral Specification)
- ระบบมอดูลที่มีประสิทธิภาพ (Powerful Module System) เป็นการเขียนข้อกำหนดเพื่ออธิบายระบบที่มีลักษณะต่างๆ คือระบบที่นำมอดูลจากภายนอกเข้ามาประกอบ โปรแกรมที่มีพารามิเตอร์ และนิพจน์มอดูล
- ระบบชนิดข้อมูลที่มีประสิทธิภาพ (Powerful Type System) สามารถเขียนข้อกำหนดที่มีชนิดข้อมูลย่อย (Subtype) เพื่อเพิ่มความชัดเจนลงไปยิ่งขึ้น โดยใช้พีชคณิตของชนิดข้อมูลที่มีลำดับ (Order Sorted Algebra) โดยในภาษาคาเฟ่โอบีเจชนิดข้อมูลหนึ่งๆ จะเรียกว่า “ซอท” (Sort)

2.1.1 วากยสัมพันธ์ของภาษาคาเฟโอบีเจ

มอดูล (Module) ในภาษาคาเฟโอบีเจ [12] มี 2 ประเภท ขึ้นอยู่กับหน้าที่และการทำงานของมอดูลนั้นๆ ประเภทแรกคือ ประเภทเดี่ยว (a unique model) ประกาศด้วยสัญลักษณ์ *module! mod! module* หรือ *mod* และ ประเภทที่สองคือ ประเภทหมวดหมู่ (A class of models) ประกาศ (declare) ด้วยสัญลักษณ์ *module** หรือ *mod** เช่น *mod* INDEX* ดังแสดงในรูปที่ 2.1

```
mod* INDEX{
        pr(INT)

        *[ Index ]*

        op init-index : -> Index     -- initial
        op dec_ : Index -> Index  -- method
        op inc_ : Index -> Index    -- method
        bop val : Index -> Int       -- attribute

        var X : Index

        eq val(init-index) = 0 .
        eq val(inc X) = (val(X) + 1 ) .
        ceq val(dec X) = (val(X) - 1 ) if val(X) =/= 0 .
}
```

รูปที่ 2.1 ตัวอย่างมอดูล INDEX

รายละเอียดต่างๆ ในมอดูลมีดังต่อไปนี้

- การประกาศนำเข้า (Import Declaration)

ส่วนย่อย (Elements) ในมอดูลสามารถอ้างอิงถึงมอดูลอื่นเพื่อนำเข้ามาทำงานร่วมกัน โดยภาษาคาเฟโอบีเจมีการนำเข้า 3 แบบ คือ การนำเข้าแบบป้องกัน (protecting) ประกาศด้วยสัญลักษณ์ *protecting* หรือ *pr* การนำเข้าแบบขยาย (extending) ประกาศด้วยสัญลักษณ์ *extending* หรือ *ex* และการนำเข้าแบบใช้งาน (using) ประกาศด้วยสัญลักษณ์ *using* หรือ *us*

- การนำเข้าแบบป้องกัน สามารถป้องกันมอดูลที่นำเข้ามาใช้งานร่วมกันได้ โดยมอดูลดังกล่าวจะไม่ถูกทำลายและไม่เสียหาย
- การนำเข้าแบบขยาย อนุญาตให้ขยายการทำงานของมอดูลที่นำเข้ามาใช้งานออกไปได้ และการทำงานเดิมยังคงอยู่โดยไม่ถูกทำลาย
- การนำเข้าแบบใช้งาน อนุญาตให้เรียกใช้งานมอดูลที่นำเข้ามาได้อย่างเต็มที่ ทั้งการเรียกใช้แบบธรรมดา สร้างการทำงานใหม่ หรือแม้กระทั่งทำลายการทำงานเดิม

- การประกาศชนิดข้อมูล (Sort Declaration)

ชนิดข้อมูล (Sort) คือ เซตของประเภทส่วนย่อย (A set of classified elements) แบ่งชนิดข้อมูลออกเป็น 2 ประเภท คือ ชนิดข้อมูลแฝงค่า (Hidden Sort) ซึ่งประกาศด้วยสัญลักษณ์ *[ชื่อชนิดข้อมูล]* เช่น *[Bool]* และชนิดข้อมูลที่สังเกตค่าได้ (Visible Sort) ซึ่งประกาศด้วยสัญลักษณ์ **[ชื่อชนิดข้อมูล]** เช่น **[Index]**

- การประกาศระเบียน (Record Declaration)

การประกาศระเบียน คือ การประกาศโครงสร้างของข้อมูลแบบชนิดระเบียน ซึ่งโครงสร้างนี้ประกอบด้วยเขตข้อมูล (field) ต่างๆ ที่มีการประกาศชนิดตัวแปรข้อมูลอยู่ภายใน ดังรูปที่ 2.2 เป็นต้น

- การประกาศการดำเนินการ (Operation Declaration)

การประกาศการดำเนินการ ประกาศได้ 3 วิธีคือ การดำเนินการเติมหน้า (prefix operation) การดำเนินการเติมกลาง (infix operation) และการดำเนินการเติมหลัง (postfix operation) ประกาศด้วยสัญลักษณ์ op ตัวอย่างการประกาศการดำเนินการเติมหลัง
op dec_ : Index -> Index หากไม่มีเครื่องหมาย _ ที่ตัวแปรดำเนินการทั้งการดำเนินการเติมหน้าและหลัง จะเป็นการประกาศการดำเนินการแบบมาตรฐาน ตัวอย่างเช่น
op initIndex : -> Index

- การประกาศการดำเนินการเชิงพฤติกรรม (Behavioral Operation Declaration)

การประกาศการดำเนินการเชิงพฤติกรรม ใช้สำหรับกำหนดการดำเนินการแบบเฉพาะเจาะจงสำหรับข้อมูลชนิดแฝงค่า (Hidden Sort) ซึ่งประกาศด้วยสัญลักษณ์ *bop* เช่น
bop val : Index -> Int

- การประกาศตัวแปร (Variable Declaration)

ตัวแปรแต่ละตัวเป็นส่วนหนึ่งของชนิดข้อมูล (Sort) ซึ่งแทนด้วยพจน์ของชนิดข้อมูลนั้นๆ เช่น *var X : Index*

```
record Segment{
        sourceport : Int
        destport : Int
        seqnumber : Int
        acknumber : Int
        offset : Int        -- value = 0 ; where data begins
        reserved : Int      -- value =0 ; for future use
        urg : Bool
        ack : Bool
        psh : Bool
        rst : Bool
        syn : Bool
        fin : Bool
        window : Int        -- value = 0 ( for no Segment )
        checksum : Int      -- value = 0 ( checking data in correctly)
        urgentptr : Int     -- noUrgentprt
        options : Int       -- value = 0 (endOfOption),
                               value = 1 (noOperation),
                               value = 2 (maximumSegmentSize)
        padding : Int       -- value = 0 ;Tcp header end and
                               data begin on 32 bit boundary
        data : Int
}
```

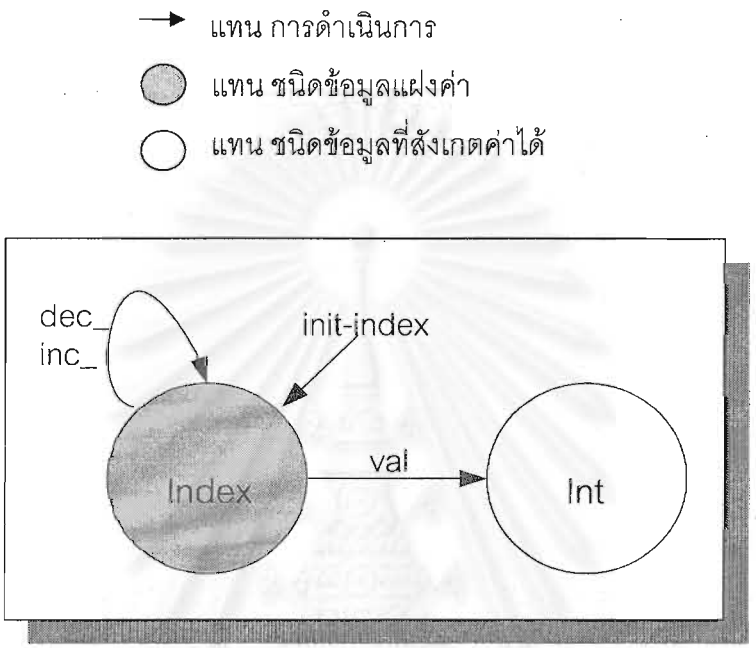
รูปที่ 2.2 ตัวอย่างระเบียน Segment

- การประกาศสมการ (Equation Declaration)

การประกาศสมการแบ่งออกได้ 2 ประเภท คือ สมการแบบปกติ (A Plain Equation) และสมการแบบมีเงื่อนไข (A Conditional Equation) ซึ่งประกาศด้วยสัญลักษณ์ *eq* และ *ceq* ตามลำดับ ตัวอย่างการประกาศสมการแบบปกติ *eq val(initIndex) = 0* . และสมการแบบมีเงื่อนไข *ceq val(dec X) = (val(X) – 1) if val(X) =/= 0* .

2.1.2 แผนภาพเอดีเจ (ADJ Diagram)

ตระกูลโอบีเจเสนอสัญกรณ์ (Notation) ด้วยภาพ เรียกว่า แผนภาพเอดีเจ [6] ดังแสดงในรูปที่ 2.3 ซึ่งใช้อธิบายพฤติกรรมของข้อกำหนดตามข้อกำหนดในรูปที่ 2.1

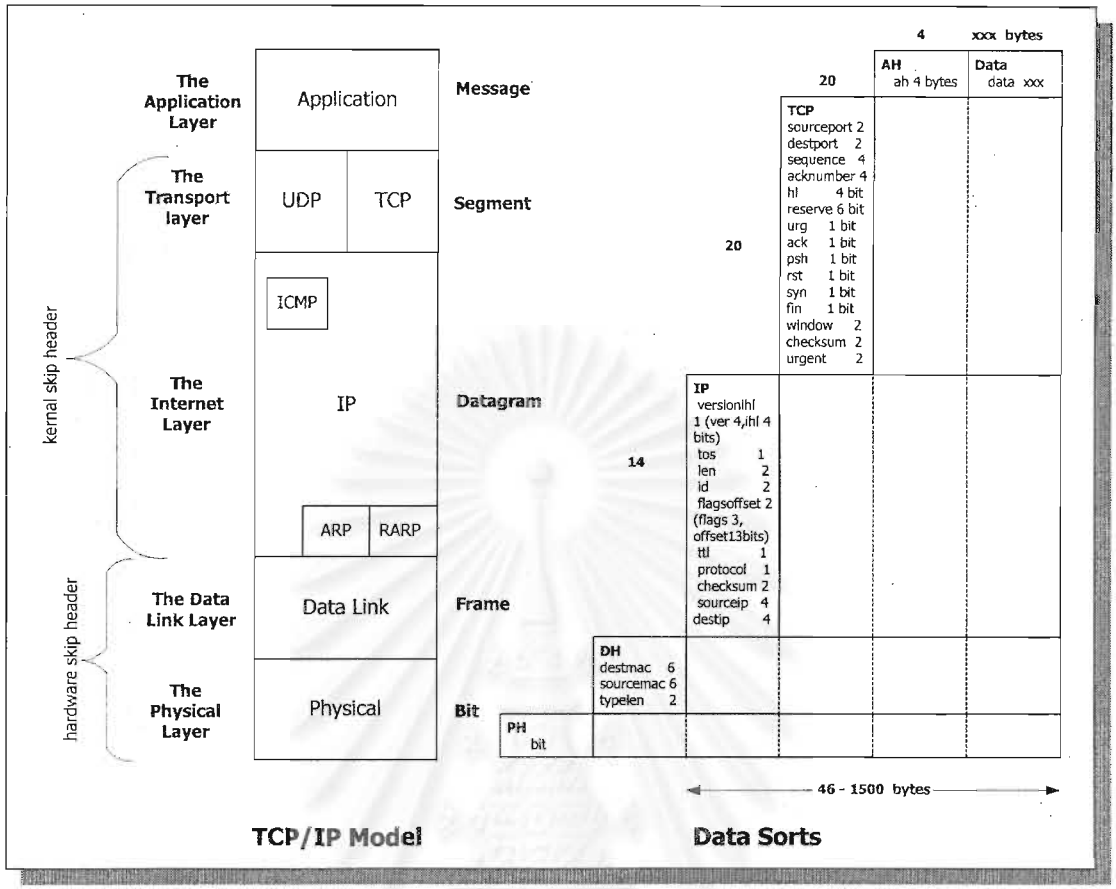


รูปที่ 2.3 ตัวอย่างแผนภาพเอดีเจ Index

bop val : Index -> Int ในแผนภาพในรูปที่ 2.3 นี้ หมายถึงการดำเนินการ val มีการรับค่าชนิดข้อมูลแฝงค่า Index เข้ามาเพื่อดำเนินการแล้วส่งค่าออกมาเป็นชนิดข้อมูล Int

2.2 ทีซีพี (Transmission Control Protocol)

ทีซีพี (TCP) (ดังมาตรฐานที่ระบุในอาร์เอฟซี 793 [8]) เป็นการขนส่งข้อมูลแบบกระแสข้อมูลเชิงวัตถุ (connection-oriented byte stream) จากโพรโทคอลหนึ่งไปยังอีกโพรโทคอลหนึ่ง ออกแบบเพื่อบรรจุอยู่ในแบบจำลองทีซีพีไอพี (รูปที่ 2.4) ซึ่งสนับสนุนการทำงานของแอพพลิเคชันที่ใช้งานหลายๆ เครือข่าย ทั้งนี้ทีซีพีที่กล่าวนี้มีการรับรองความถูกต้องในการรับส่งข้อมูล ไม่มีปัญหาในการรับส่งข้อมูลขณะที่สิ้นสุดเวลาในการรับส่งและเมื่อมีข้อผิดพลาดสามารถส่งข้อมูลใหม่ได้อีกครั้ง



รูปที่ 2.4 แบบจำลองทีซีพีไอพีและชนิดข้อมูล [8,9,10]

ทีซีพีออกแบบตามสถาปัตยกรรมแบบจำลองทีซีพีไอพี ซึ่งชั้นของทีซีพีจัดอยู่ในชั้นการขนส่ง (The Transport Layer) บนชั้นอินเทอร์เน็ต (The Internet Layer) อีกทีหนึ่ง มีหน้าที่รับส่งเซก-เมนต์ (Segment) ของข้อมูลแนบไปเป็นเดทาแกรม (Datagram) ในชั้นอินเตอร์เน็ต แล้วชั้นอินเตอร์เน็ตนี้จะส่งข้อมูลไปตามที่อยู่ของต้นทางและเป้าหมาย (Addressing source and destination) ต่อไป

2.2.1 ทีซีพี : ส่วนต่อประสาน (Interface)

ทีซีพีมีส่วนต่อประสานสองทางด้วยกัน ทางที่หนึ่งคือ ประสานกับผู้ใช้งานหรือกระบวนการการประยุกต์ต่างๆ (Application processes) และทางที่สอง คือ ประสานกับโพรโทคอลลำดับล่าง เช่น อินเทอร์เน็ตโพรโทคอล ส่วนต่อประสานระหว่างกระบวนการการประยุกต์นั้นประกอบด้วยชุดของการเรียกซึ่งเหมือนกับคำสั่งเรียกจากระบบปฏิบัติการที่กระบวนการประยุกต์ใช้จัดการกับไฟล์ต่างๆ เช่น คำสั่งเรียก "เปิด" และ "ปิด" ที่ใช้ในการเรียกเปิดและปิดการ

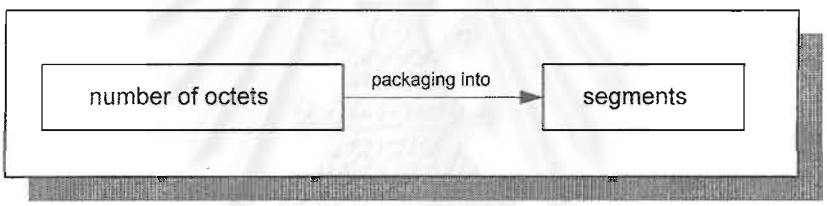
ติดต่อ และคำสั่งเรียก "ส่ง" และ "รับ" ที่ใช้ในการส่งข้อมูลและรับข้อมูลเมื่อทำการติดต่อเรียบร้อยแล้ว คำสั่งเรียกเหล่านี้มีการส่งพารามิเตอร์ของที่อยู่ ชนิดของการให้บริการ ลำดับความสำคัญ ความ-ปลอดภัย และข้อมูลควบคุมต่างๆ ส่งรวมไปด้วย

2.2.2 ทีซีพี : การดำเนินการ (Operation)

จุดมุ่งหมายหลักของทีซีพี คือ คู่กระบวนการติดต่อสื่อสารมีความน่าเชื่อถือในการรับส่งข้อมูล การจัดการบริการดังกล่าวนี้จึงจำเป็นต้องครอบคลุมรายละเอียดดังต่อไปนี้

- การถ่ายโอนข้อมูล (Data Transfer)

ทีซีพีสามารถส่งข้อมูลแบบกระแสต่อเนื่องออกเตต (Continuous stream of octets) ของผู้ใช้งาน โดยทำการบรรจุออกเตตของข้อมูลลงไปยังเซกเมนต์ (Segment) (ดูรูปที่ 2.5) แล้วส่งผ่านระบบอินเทอร์เน็ตต่อไป



รูปที่ 2.5 การบรรจุเซกเมนต์

- ความน่าเชื่อถือ (Reliability)

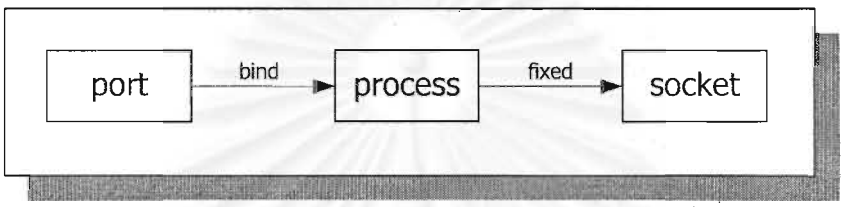
ทีซีพีสามารถกู้ข้อมูลที่เสียหาย สูญหาย และซ้ำซ้อน อันเนื่องมาจากสาเหตุของระบบการติดต่อสื่อสารอินเทอร์เน็ต ความสามารถเหล่านี้ทำได้โดยการกำหนดเลขลำดับการส่ง (A Sequence Number) และการกำหนดเลขลำดับการรับ (An Acknowledgement Number) ของทีซีพีของผู้รับ ถ้าเลขลำดับการรับไม่สามารถรับข้อมูลในระยะเวลาที่ได้กำหนดไว้ ข้อมูลจะถูกส่งมาใหม่ ทั้งนี้ผู้รับจะต้องทำการเรียงลำดับข้อมูลตามเลขลำดับการส่งดังกล่าวที่อาจรับข้อมูลซึ่งไม่ได้เรียงลำดับเอาไว้ และอาจต้องกำจัดข้อมูลที่ซ้ำซ้อนออกไป

- การควบคุมสายงาน (Flow Control)

ทีซีพีมีค่า "วินโดว์" (windows) สำหรับเลขลำดับการรับ เพื่อแสดงปริมาณข้อมูลที่ผู้รับได้รับข้อมูลจากผู้ส่ง

- การรับส่งหลายทาง (Multiplexing)

ทีซีพีมีชุดที่อยู่หรือช่องทาง (A Set of Addresses or Ports) ในทุกๆ แม่ข่าย (Host) โดยที่ทีซีพีติดต่อสื่อสารซึ่งกันและกันในเครือข่ายด้วยการใช้ชุดที่อยู่หรือช่องทางดังกล่าว ซึ่งเก็บอยู่ในชั้นของอินเทอร์เน็ตที่เรียกว่า "ซ็อกเก็ต" คู่ของซ็อกเก็ตหนึ่งๆ บ่งถึงกระบวนการติดต่อซึ่งกันและกันในระบบเครือข่าย โดยที่แม่ข่ายทำการยึดเหนี่ยวช่องทางไปยังกระบวนการเพื่อทำการตรึงซ็อกเก็ตไว้ใช้ในการติดต่อเครือข่ายตามชุดที่อยู่หรือช่องทางที่ระบุไว้ในซ็อกเก็ต (ดังรูปที่ 2.6)



รูปที่ 2.6 กระบวนการตรึงซ็อกเก็ต

- การเชื่อมต่อ (Connection)

ความน่าเชื่อถือและการควบคุมสายงานที่ได้กล่าวไว้ข้างต้นจำเป็นต้องกำหนดค่าเริ่มต้นของทีซีพีและกำหนดข้อมูลแสดงสถานภาพ (Status Information) สำหรับกระแสข้อมูลนั้นๆ ซึ่งข้อมูลแสดงสถานภาพ ข้อมูลในซ็อกเก็ต ข้อมูลเลขลำดับการส่ง ข้อมูลเลขลำดับการรับ และขนาดของข้อมูล ข้อมูลเหล่านี้เกี่ยวข้องกับการเชื่อมต่อทั้งสิ้น การเชื่อมต่อแต่ละคู่การติดต่อนั้นคือ การติดต่อกันระหว่างกระบวนการผ่านคู่ซ็อกเก็ต เมื่อทีซีพีมีการติดต่อสำเร็จเรียบร้อยแล้ว และสิ้นสุดการติดต่อจะคืนทรัพยากรต่างๆ ที่เคยนำมาใช้งานให้กับผู้ใช้งานดังเดิม

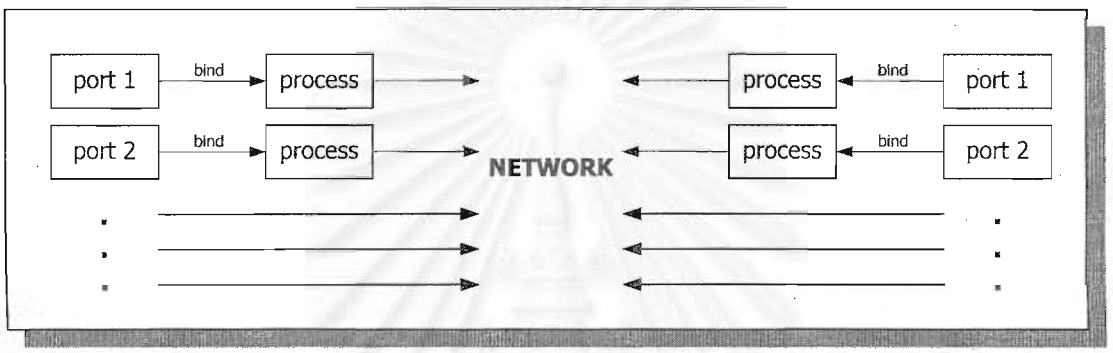
- การทำก่อนและความมั่นคง (Precedence and Security)

ผู้ใช้ทีซีพีต้องระบุค่าความมั่นคงและค่าการทำก่อนในการติดต่อ ซึ่งมีค่าโดยปริยายสำหรับค่าความมั่นคงและค่าการทำก่อนเมื่อค่าทั้งสองนี้มิได้เรียกใช้งาน

2.2.3 ทีซีพี : ส่วนย่อยของระบบระหว่างเครือข่าย (Element of the Internetwork System)

สภาพแวดล้อมระหว่างโครงข่ายประกอบด้วยเครือแม่ข่ายที่ติดต่อกันในโครงข่ายผ่านเกตเวย์ (Gateway) โดยการรับส่งข้อความ (message) จากเครื่องแม่ข่ายหนึ่งไปยังอีกเครื่องหนึ่งนั้นเรียกว่า "กระบวนการ" (Processes) กระบวนการดังกล่าวนี้ทำงานเหมือนตัวทำงาน

(Active Agent) ปลายทาง ไฟล์ข้อมูล อุปกรณ์ต่างๆ มีการติดต่อสื่อสารซึ่งกันและกัน กระบวนการก็เช่นเดียวกันมีการติดต่อสื่อสารดังกล่าวด้วย ดังนั้นจึงมองว่าการติดต่อสื่อสารสิ่งต่างๆ ทั้งหมดนี้เป็นเหมือนการติดต่อสื่อสารระหว่างกระบวนการหนึ่งไปยังอีกกระบวนการหนึ่ง ด้วยเหตุดังกล่าวกระบวนการจำเป็นต้องสามารถติดต่อระหว่างหลายๆ กระบวนการพร้อมกันและรวมไปถึงการติดต่อ ไปยังกระบวนการภายนอกอีกด้วย แต่ละกระบวนการมีช่องทางเพื่อใช้ในการติดต่อไปยังกระบวนการของแม่ข่ายอื่นๆ (ดังรูปที่ 2.7) เช่นกัน



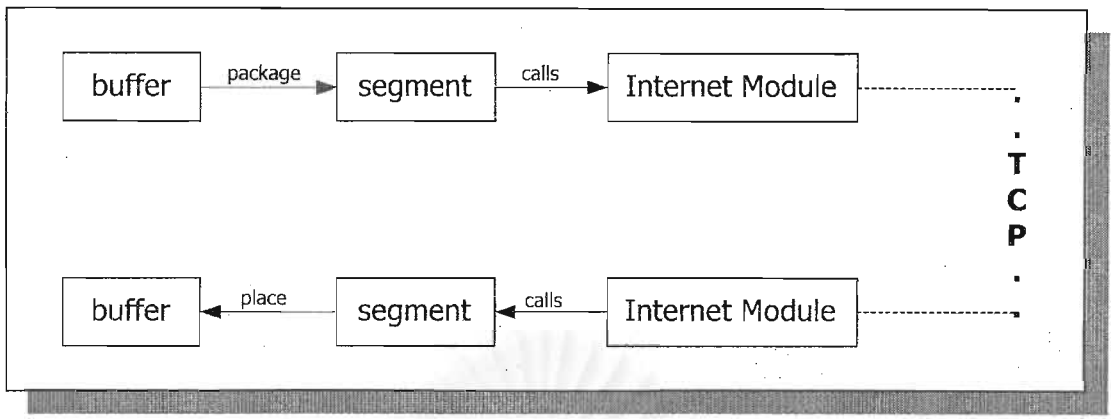
รูปที่ 2.7 กลไกกระบวนการติดต่อสื่อสารระหว่างเน็ตเวิร์ค

2.2.4 ทีซีพี : แบบจำลองการดำเนินการ (Model of Operation)

ทีซีพีบรรจุข้อมูลจากบัฟเฟอร์ (Buffer) ไปยังเซกเมนต์ (Segment) และรอคำสั่งเรียกจากมอดูลของอินเทอร์เน็ต (The internet moduie) เพื่อส่ง ข้อมูลดังกล่าวไปยังเซกเมนต์ของทีซีพีปลายทาง ในทางตรงกันข้ามทีซีพีปลายทางจะรับข้อมูลจากเซกเมนต์แล้วใส่ข้อมูลไปยังบัฟเฟอร์ (ดูรูปที่ 2.8) สิ่งที่สำคัญอีกประการคือ ทีซีพีรวบรวมข้อมูลที่ใช้ในการควบคุมการรับส่งข้อมูลเก็บไว้ในเซกเมนต์ด้วย เพราะข้อมูลส่วนนี้ใช้สำหรับตรวจสอบความน่าเชื่อถือและลำดับการรับส่งข้อมูล

2.2.5 ทีซีพี : สภาพแวดล้อมของแม่ข่าย (The Host Environment)

การทำงานของทีซีพีเทียบได้กับมอดูลในระบบปฏิบัติการ ซึ่งผู้ใช้สามารถเข้าถึงทีซีพีนี้เช่นเดียวกับการเข้าถึงระบบไฟล์ข้อมูล ทีซีพีไม่สามารถใช้คำสั่งเรียกจากโปรแกรมขับเครือข่าย (The network device driver) ได้โดยตรง แต่สามารถใช้คำสั่งเรียกจากมอดูลของโพรโทคอลอินเทอร์เน็ต



รูปที่ 2.8 ระบบการสื่อสาร

2.2.6 ทีซีพี : การติดตั้งและการยกเลิกการเชื่อมต่อ (Connection Establishment and Clearing)

ทีซีพีมีตัวระบุช่องทางสำหรับแยกกระแสข้อมูลและระบุที่อยู่ของแต่ละทีซีพี การระบุที่อยู่ของอินเทอร์เน็ต (internet address) นั้นเพื่อเป็นตัวชี้ไปยังทีซีพีโดยผ่านดัวบ่งชี้ช่องทาง (port identifies) ใช้สร้างซ็อกเก็ตที่ไม่ซ้ำกันทั้งเครือข่ายในช่วงเวลาที่ทำการติดต่อซึ่งกันและกัน

การเชื่อมต่อจัดให้มีการเรียก "เปิด" โดยช่องทางภายในและต่างอาร์กิวเมนต์ของซ็อกเก็ตอื่นๆ จะทำการเชื่อมต่อกัน ในทางตรงกันข้ามทีซีพีจัดให้มีชื่อในการเชื่อมต่อเฉพาะที่ (A local connection name) เพื่อให้ผู้ใช้งานใช้อ้างถึงการเชื่อมต่อในภายหลัง สิ่งที่จำเป็นอีกอย่างคือ เก็บข้อมูลเกี่ยวกับการเชื่อมต่อไว้ใน "กลุ่มระเบียนควบคุมการขนส่งข้อมูล" (Transmission Control Block) ดังแสดงรายละเอียดในรูปที่ 2.9

การติดตั้งการเชื่อมต่อเป็นตัวประสานการทำงาน หรือรอคอยการเรียก "เปิด" คำสั่งการรอคอยการเรียก "เปิด" (passive OPEN) หมายถึง กระบวนการที่รอการร้องขอการติดต่อ

กระบวนการต่างๆ มีผลก็ต่อเมื่อมีการรอคอยการเรียก "เปิด" ซึ่งคู่กับ การ "เปิด" การทำงาน (active OPEN) จากกระบวนการอื่นๆ จากนั้นทีซีพีจะแจ้งให้ทราบว่าได้ทำการติดตั้งการเชื่อมต่อเป็นที่เรียบร้อยแล้ว

กระบวนการติดตั้งการเชื่อมต่อใช้บิตควบคุม syn ในการรับส่งสื่อสารกัน 3 ขั้นตอนด้วยกันเรียกว่า "การจับมือ 3 วิถี" (3-way handshake) การเชื่อมต่อนี้เริ่มจากเซกเมนต์ที่บรรจุบิตควบคุม syn และรอคำสั่งเรียก "เปิด" จากผู้ใช้ เพื่อทำการสร้างกลุ่มระเบียนควบคุมการขนส่งข้อมูล (TCB) เมื่อเริ่มการเชื่อมต่อมีการเชื่อมต่อระหว่างซ็อกเก็ตภายในและซ็อกเก็ตภายนอกเข้าด้วยกัน เมื่อเกิดการเชื่อมต่อสถานะจะเปลี่ยนไปเป็นสถานะการติดตั้ง (ESTABLISHED)

การเชื่อมต่อมีความสมบูรณ์เมื่อมีสถานะเป็นสถานะการติดตั้งนั่นเอง การยกเลิกการเชื่อมต่อมีการทำงานเช่นเดียวกับการติดตั้งการเชื่อมต่อ แต่ต่างกันตรงการใช้บิตควบคุม fin เพื่อยกเลิกการเชื่อมต่อที่ใช้ในการสื่อสาร

2.2.7 ทีซีพี : ข้อกำหนดฟังก์ชัน (Function Specification)

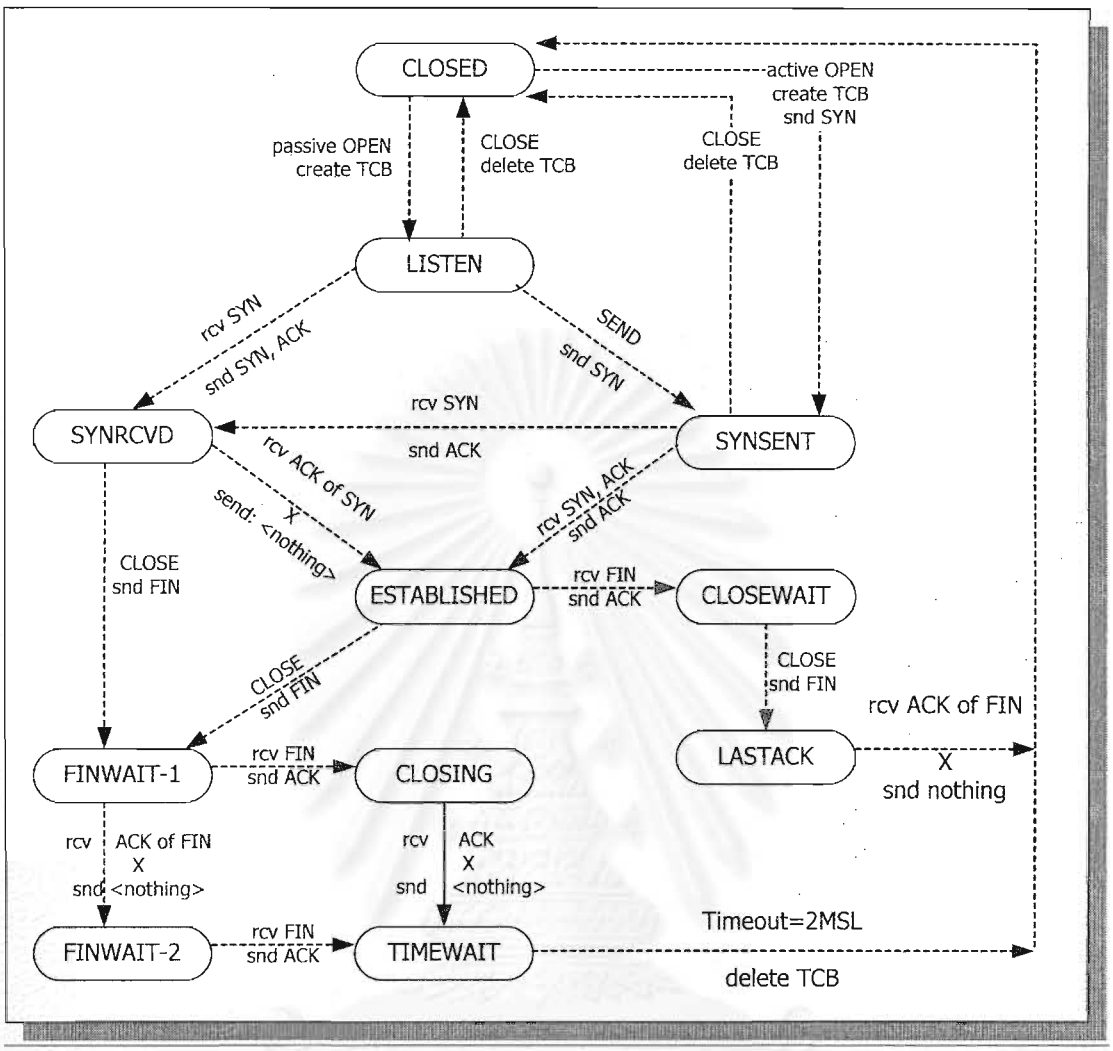
เซกเมนต์ของทีซีพี (TCP Segments) ส่งข้อมูลเหมือนกับการส่งข้อมูลของเดทาแกรมในชั้นอินเทอร์เน็ต (ดูรูปที่ 2.4) การเชื่อมต่อของทีซีพี มีความจำเป็นต่อการเก็บรายละเอียดต่างๆ ที่ใช้ในการรับส่งข้อมูลในการเชื่อมต่อลงในระเบียนการเชื่อมต่อ (A Connection Record) เรียกว่า "กลุ่มระเบียนควบคุมการขนส่งข้อมูล" (TCB) (ดังรูปที่ 2.9)

```
record Tcb{
snd-una : Int      -- send unacknowledged
snd-nxt : Int      -- send next
snd-wnd : Int      -- send window
snd-up : Int       -- send urgent pointer
snd-wl1 : Int      -- send sequence number used for last window update
snd-wl2 : Int      -- send acknowledgement number used for last window  update
iss : Int          -- initial send sequence number
rcv-nxt : Int      -- receive next
rcv-wnd : Int      -- receive window
rcv-up : Int       -- receive urgent pointer
irs : Int          -- initial receive sequence number
seg-seg : Int      -- segment sequence number
seg-ack : Int      -- segment acknowledgement number
seg-len : Int      -- segment length
seg-wnd : Int      -- segment window
seg-up : Int       -- segment urgent pointer
seg-prc : Int      -- segment security and precedence value
}
```

รูปที่ 2.9 กลุ่มระเบียนควบคุมการขนส่งข้อมูล (TCB)

กระบวนการเชื่อมต่อดำเนินไปตามสถานะต่างๆ ตามข้อกำหนดเชิงหน้าที่ของทีซี-พี (ดังรูปที่ 2.10) ซึ่งสถานะต่างๆ ประกอบด้วย สถานะฟัง (LISTEN) สถานะส่ง (SYNSENT) สถานะรับ (SYNRCVD) สถานะติดตั้ง (ESTABLISHED) สถานะรอฟิน1 (FINWAIT-1) สถานะรอ-ฟิน2 (FINWAIT-2) สถานะรอปิด (CLOSEWAIT) สถานะทำการปิด (CLOSING) สถานะตอบรับ-สุดท้าย (LASTACK) สถานะรอเวลา (TIMEWAIT) และสถานะปิด (CLOSED) ดังแสดงราย-ละเอียดการทำงานดังนี้

- สถานะฟัง เป็นสถานะที่แสดงถึงการรอคอยการร้องขอการเชื่อมต่อจากทีซีพีระยะไกล (remote TCP) และช่องทาง
- สถานะส่ง เป็นสถานะที่แสดงถึงการเข้าคู่การร้องขอการเชื่อมต่อหลังจากที่ส่งการร้องขอการติดต่อไปแล้ว
- สถานะรับ เป็นสถานะที่แสดงถึงการยืนยันการเชื่อมต่อจากการตอบรับหลังจากที่ผู้รับและผู้-ส่งมีการรับและส่งการร้องขอการติดต่อเรียบร้อยแล้ว
- สถานะติดตั้ง เป็นสถานะที่แสดงถึงการเปิดการเชื่อมต่อ ส่งข้อมูลไปยังผู้ใช้งาน
- สถานะรอฟิน1 เป็นสถานะที่แสดงถึงการรอคอยการยกเลิกการเชื่อมต่อจากทีซีพี ระยะไกล หรือการตอบรับการส่งการร้องขอยกเลิกการเชื่อมต่อก่อนหน้า
- สถานะรอฟิน2 เป็นสถานะที่แสดงถึงการรอคอยการร้องขอยกเลิกการเชื่อมต่อจากทีซีพี ระยะไกล
- สถานะรอปิด เป็นสถานะที่แสดงถึงการรอคอยการร้องขอยกเลิกการเชื่อมต่อจากผู้ใช้งาน ภายนอก (local user)
- สถานะทำการปิด เป็นสถานะที่แสดงถึงการรอคอยการตอบรับที่ร้องขอยกเลิกการเชื่อมต่อ จากทีซีพีระยะไกล
- สถานะตอบรับสุดท้าย เป็นสถานะที่แสดงถึงการรอคอยการตอบรับที่ร้องขอยกเลิกการ เชื่อมต่อก่อนหน้าที่ส่งไปยังทีซีพีระยะไกล
- สถานะรอเวลา เป็นสถานะที่แสดงถึงการรอคอยช่วงเวลาหนึ่งที่มากพอต่อทีซีพีระยะไกล ที่รับ การตอบรับจากการร้องขอยกเลิกการเชื่อมต่อ
- สถานะปิด เป็นสถานะที่แสดงถึงการไม่มีสถานะการเชื่อมต่อใดๆ อีกแล้ว



รูปที่ 2.10 ข้อกำหนดเชิงหน้าที่ของทีซีพี

2.3 งานวิจัยที่เกี่ยวข้อง (Literatures Survey)

2.3.1 วิธีรูปนัยสำหรับข้อกำหนดของโพรโทคอลสื่อสาร (A Formal Specification Technique for Communication Protocol) [3]

ในงานวิจัยของ Layuan เสนอวิธีรูปนัยสำหรับข้อกำหนดของโพรโทคอลสื่อสาร โดยวิธีรูปนัยดังกล่าวนี้เป็นการนำวิธีการหลายๆ อย่างมาผสมกัน แล้วนำข้อดีของแต่ละวิธีมาใช้งานอย่างเหมาะสม เรียกการรวมวิธีดังกล่าวนี้ว่า Hybrid Formal Method (FCA) ซึ่งประกอบด้วย เครื่องสถานะจำกัด (Finite State Machine (FSM)) กระบวนการลำดับการสื่อสาร

(Communicating Sequential Processes (CSP)) และชนิดข้อมูลนามธรรม (Abstract Data Type (ADT)) โดยแสดงดังสมการต่อไปนี้

Hybrid Model (FCA) = (FSM, CSP, ADT)

โดยที่ ADT ใช้อธิบายรายละเอียดข้อมูลในโพรโทคอล (e.g. address size = 16;)

CSP กระบวนการดำเนินตามลำดับการสื่อสาร

FSM = (S,P,I,O,T)

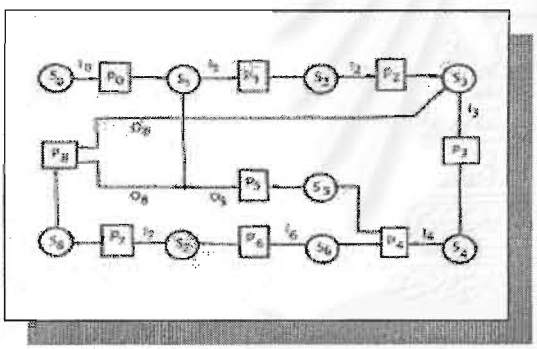
S = a finite set of states

P = a set of processes

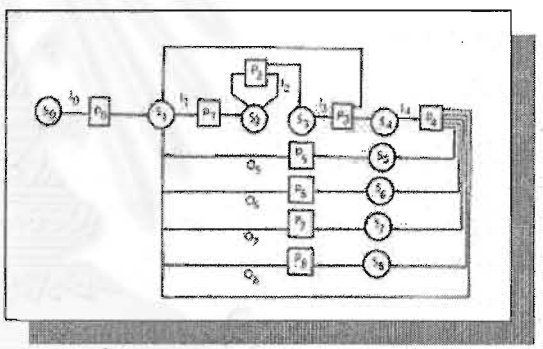
I = a set of input

O = a set of output

T = a state transition function (T: I x S -> S)



รูปที่ 2.11 แผนภาพสถานะการขนส่งข้อมูล[3]



รูปที่ 2.12 แผนภาพสถานะการรับข้อมูล[3]

โดยที่ ? แทน input

! แทน output

-> แทน sequential command

=> แทน assignment operator

|| แทน parallel operator

P_0: c?x: initialize -> S_1 => S; ...(1)

P_1: c?x: data req -> S_2=> S; ...(2)

วิธีเครื่องสถานะจำกัด กระบวนการลำดับการสื่อสาร และชนิดข้อมูลนามธรรม ทำให้กระบวนการทำงานของโพรโทคอลทั้งหมดครอบคลุมรายละเอียดข้อมูล (c?x) กระบวนการทำงาน เช่น กระบวน P_0 และ P_1 ในสมการที่ (1) และ (2) ตามลำดับ และสถานะของโพรโทคอล เช่น S_1 => S; ในสมการที่ (1) และ (2)

2.3.2 การจัดรูปนัยโพรโทคอลและวิธีทวนสอบ (Protocol Specification & Verification Methods: An Overview) [2]

ในงานวิจัยของ Michael และ Stephen พิจารณาถึงความจำเป็นของวิธีการที่เหมาะสมต่อการจัดรูปนัยและทวนสอบข้อมูลของโพรโทคอลติดต่อสื่อสาร วิธีที่เลือกใช้คือ ลำดับ-กระบวนการติดต่อสื่อสาร (Communicating Sequential Processes (CSP)) และลำดับเวลากระบวนการติดต่อสื่อสาร (Timed CSP) ระบบที่พิจารณาประกอบด้วยกระบวนการและสภาพแวดล้อม วิธีดังกล่าวนี้รวมพฤติกรรมของกระบวนการผู้ส่งและผู้รับเข้าด้วย แต่มิได้พิจารณาหลักนามธรรมข้อมูล (The abstraction) ร่วมด้วย

2.3.3 วิธีรูปนัยและการทวนสอบสำหรับโพรโทคอลกระบวนคำสั่ง : กรณีศึกษา (Formal specification and verification of a procedural protocol: case study) [4]

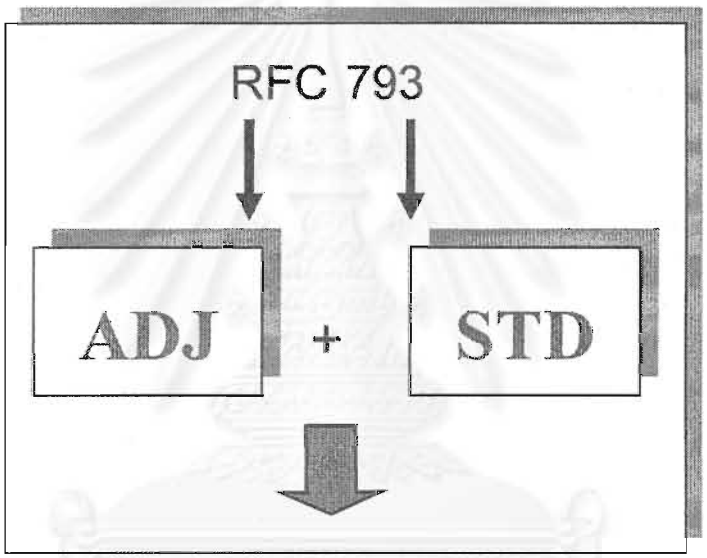
ในงานวิจัยของ Lai เสนอกรณีศึกษาซึ่งใช้วิธีการเปลี่ยนสถานะ (The State Transition technique) เอ็นพีเอ็น (The Numerical Petri Nets) และโพรทีนเครื่องมืออัตโนมัติ (PROTEAN) ในการจัดรูปและทวนสอบโพรโทคอลกระบวนคำสั่ง ในงานวิจัยนี้แสดงให้เห็นถึงประโยชน์ในการใช้วิธีจัดรูปนัยและการทวนสอบเหตุการณ์ที่สถานะของโพรโทคอลเปลี่ยนแปลงไป ถึงแม้ว่าวิธีนี้มีประโยชน์ แต่ยังไม่เพียงพอต่อการจัดรูปนัยโพรโทคอลกระบวนคำสั่ง เช่น การโยก-ย้ายถ่ายโอนงาน (Job Transfer Manipulating) วิธีดังกล่าวนี้ ไม่สามารถกำหนดพารามิเตอร์เพื่อระบุกระบวนการทำงานของระบบ

บทที่ 3

วิเคราะห์และออกแบบ

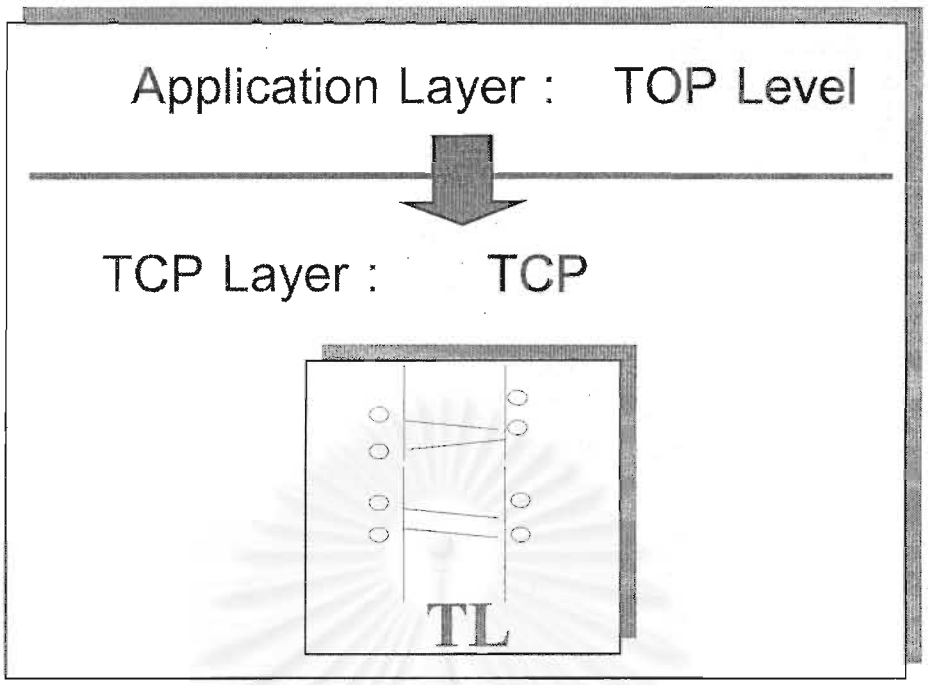
3.1 ขั้นตอนการวิเคราะห์

ใช้แผนภาพเอดีเจแสดงรายละเอียดข้อมูลของโพรโทคอลทีซีพีจากอาร์เอฟซี 793 [8] และใช้แผนภาพการเปลี่ยนสถานะแสดงกระบวนการทำงานต่างๆ ของทีซีพี พร้อมทั้งสร้างกฎที่ใช้เชื่อมแผนภาพทั้งสองเข้าด้วยกัน ผลลัพธ์ที่ได้ คือ ข้อกำหนดโพรโทคอลทีซีพี (ดูรูปที่ 3.1



รูปที่ 3.1 ขั้นตอนการวิเคราะห์ข้อกำหนดโพรโทคอลทีซีพี

จากข้อกำหนดโพรโทคอลทีซีพี ใช้แผนภาพเส้นกำหนดเวลาเพื่อใช้อธิบายการทำงานภายในชั้นทีซีพีดังกรณีศึกษาในบทที่ 5 พร้อมทั้งแสดงการเรียกใช้งานข้อกำหนดโพรโทคอลทีซีพีจากระดับบนดังกรณีศึกษาในบทที่ 5 เช่นกัน (ดูรูปที่ 3.2)



รูปที่ 3.2 ขั้นตอนอธิบายการเรียกใช้จากกรณีศึกษา

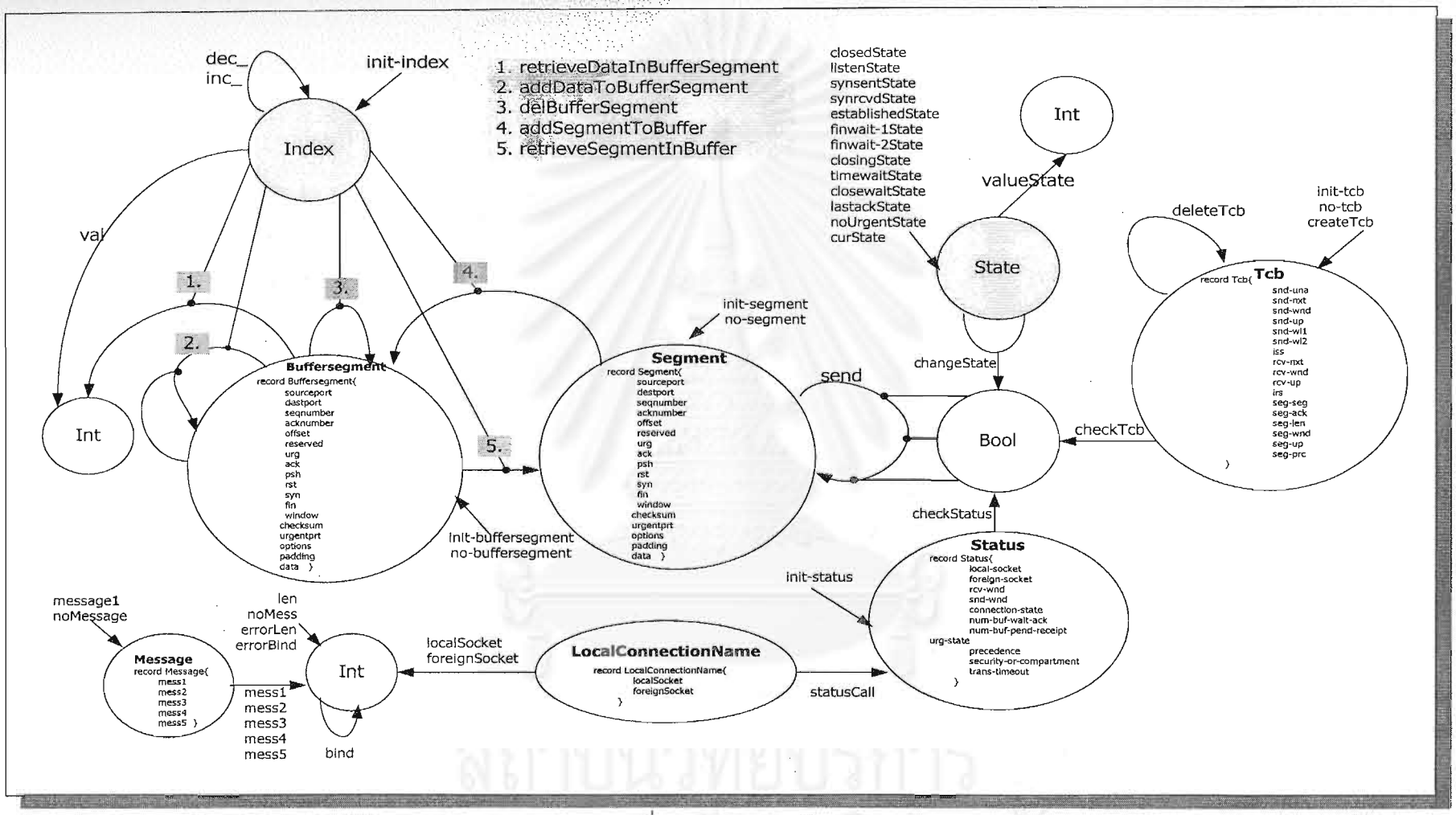
3.2 การเลือกเครื่องมือ

3.2.1 ภาษาคาเฟ่โอบีเจ (CafeOBJ Language)

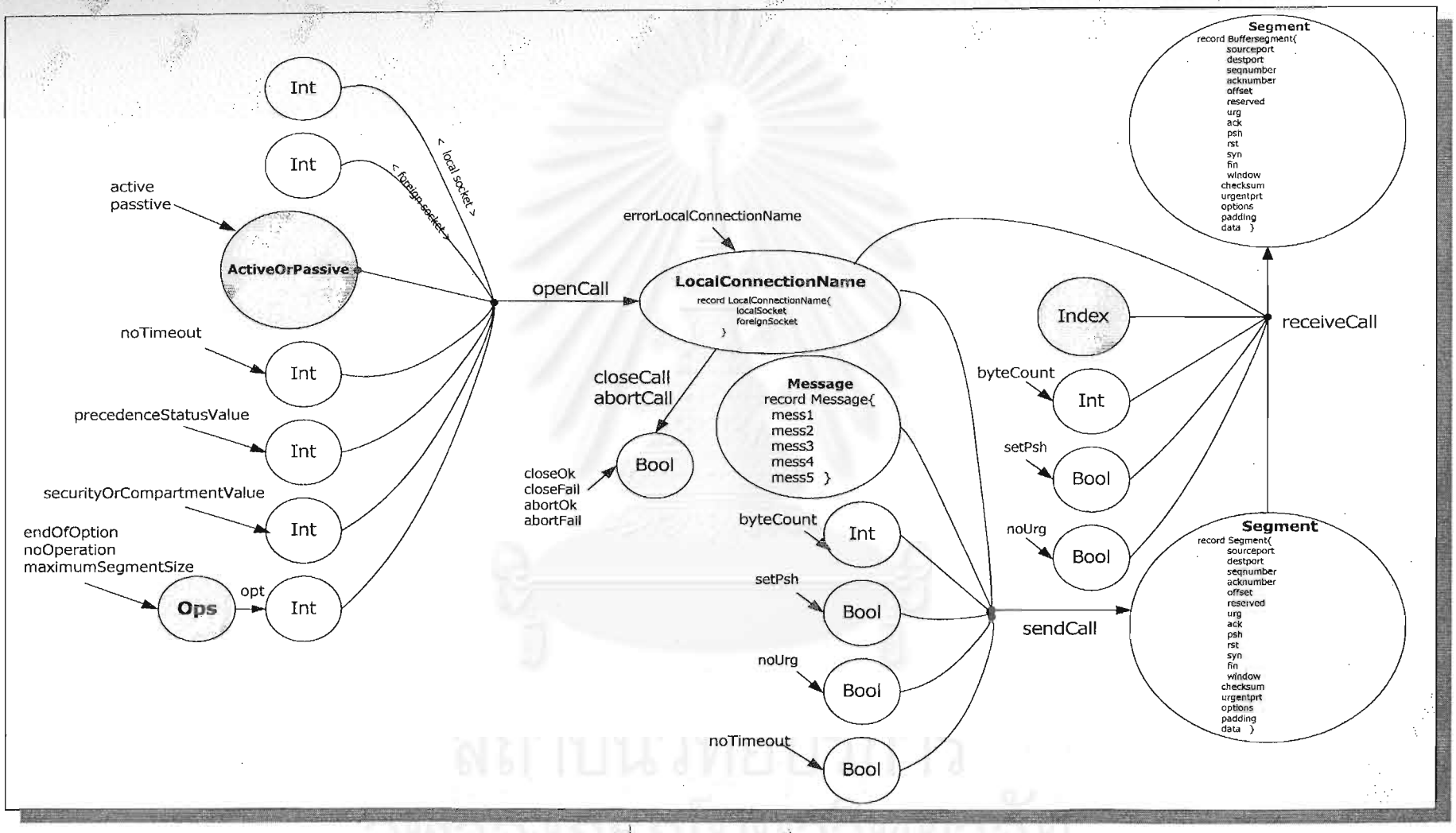
เนื่องจากทีซีพีมีการขนส่งข้อมูลแบบกระแสข้อมูลเชิงวัตถุ งานวิจัยนี้จึงเลือกภาษาคาเฟ่โอบีเจในการกำหนดรูปนัย เพราะเนื่องจากภาษาคาเฟ่โอบีเจพัฒนามาจากภาษาโอบีเจ (OBJ) อีกทั้งยังมีแม่แบบภาษาเป็นแบบพีชคณิต ทำให้ข้อกำหนดสามารถทวนสอบได้ดังพีชคณิต

3.2.2 แผนภาพเอดีเจ (The ADJ Diagram)

ภาษาคาเฟ่โอบีเจอธิบายการทำงานในรูปของการดำเนินการ (Operation) และสมการ (Equation) โดยที่ทั้งการดำเนินการและสมการต่างพิจารณาถึงชนิดข้อมูลนำเข้าและส่งออก (input/output) ที่กระทำในกระบวนการหนึ่งๆ งานวิจัยนี้จึงเลือกแผนภาพเอดีเจเพื่ออธิบายรายละเอียดชนิดข้อมูลของทีซีพี เพราะสามารถแสดงชนิดข้อมูลนำเข้าและส่งออกของการดำเนินการได้



รูปที่ 3.3 แผนภาพเอดีเจ



รูปที่ 3.4 แผนภาพเอดีเจ (ต่อ)

3.2.3 แผนภาพการเปลี่ยนสถานะ (State Transition Diagram)

กระบวนการทำงานทั้งหมดของทีซีพีอธิบายด้วยแผนภาพการเปลี่ยนสถานะของทีซีพี [8] ซึ่งเป็นแผนภาพที่อธิบายข้อกำหนดเชิงหน้าที่ของทีซีพี สถานะต่างๆ ในแผนภาพเหล่านี้ (ดังรูปที่ 2.11) อธิบายถึงลำดับกระบวนการทำงานทั้งหมดของทีซีพี

แผนภาพเอดีเจสามารถอธิบายรายละเอียดชนิดข้อมูลของทีซีพี และแผนภาพการเปลี่ยนสถานะของทีซีพีสามารถอธิบายถึงลำดับกระบวนการทำงานทั้งหมดของทีซีพี จึงทำให้งานวิจัยนี้นำแผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะมาเป็นเครื่องมือช่วยในการเขียนข้อกำหนดทีซีพีวิเคราะห์ การใช้แผนภาพทั้งสองร่วมกันทำให้สามารถระบุข้อกำหนดได้ครอบคลุมทั้งรายละเอียดของข้อมูลและกระบวนการทำงานทั้งหมด การพิจารณาการใช้งานร่วมกันของแผนภาพทั้งสอง จึงจำเป็นต้องมีกฎที่ใช้ในการเชื่อมแผนภาพทั้งสองเข้าด้วยกัน ดังแสดงรายละเอียดในบทที่ 4 ต่อไป

3.2.4 แผนภาพเส้นกำหนดเวลา (The Timed Line Diagramd)

งานวิจัยนี้ใช้เส้นกำหนดเวลาซึ่งสามารถแสดงการรับส่งข้อมูลไปมาระหว่างทีซีพีไปยังอีกทีซีพีหนึ่ง โดยมีสถานะต่างๆ ที่กำหนดไว้ในแผนภาพการเปลี่ยนสถานะ ความสำคัญอีกประเด็น คือ เส้นกำหนดเวลานี้เข้าใจง่ายและง่ายต่อการสังเกตุการรับส่งข้อมูลไปมาระหว่างทีซีพี การใช้เส้นกำหนดเวลานี้มีกฎการเชื่อมเส้นกำหนดเวลา แผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะเข้าด้วยกันเช่นกัน ดังแสดงรายละเอียดในบทที่ 5

3.3 แผนภาพเอดีเจของโพรโทคอลทีซีพี

แผนภาพเอดีเจที่แสดงรายละเอียดข้อมูลต่างๆ ของโพรโทคอลทีซีพี (รายละเอียดตามอาร์เอฟซี 793) ดังรูปที่ 3.33.4 ซึ่งจะแสดงรายละเอียดข้อมูลของแผนภาพเอดีเจตามกระบวนการทำงานต่างๆ ดังต่อไปนี้

3.3.1 ส่วนต่อประสาน (Interface)

คำสั่ง "เรียก" มีทั้งหมด 5 คำสั่งด้วยกัน คือ คำสั่งเรียกเปิด คำสั่งเรียกส่ง คำสั่งเรียกรับ คำสั่งเรียกปิด และคำสั่งเรียกยกเลิก

3.3.1.1 คำสั่งเรียก "เปิด"

คำสั่งเรียก "เปิด" ทำการรับข้อมูลดังนี้ "local socket, foreign socket, active/passive, timeout, precedence, security/compartment, options" เพื่อเปิดการเชื่อมต่อตามซ็อกเก็ตที่ระบุไว้ (local socket, foreign socket) ใช้ติดต่อไปยังที่อยู่ปลายทางในชั้นอินเตอร์เน็ต

โดยที่ชนิดข้อมูลต่างๆ ที่รับเข้ามาดำเนินการ op openCall : Int Int ActiveOrPassive Int Int Int Int -> localConnectionName มีความหมายตามลำดับ (รูปที่ 3.5) ดังนี้

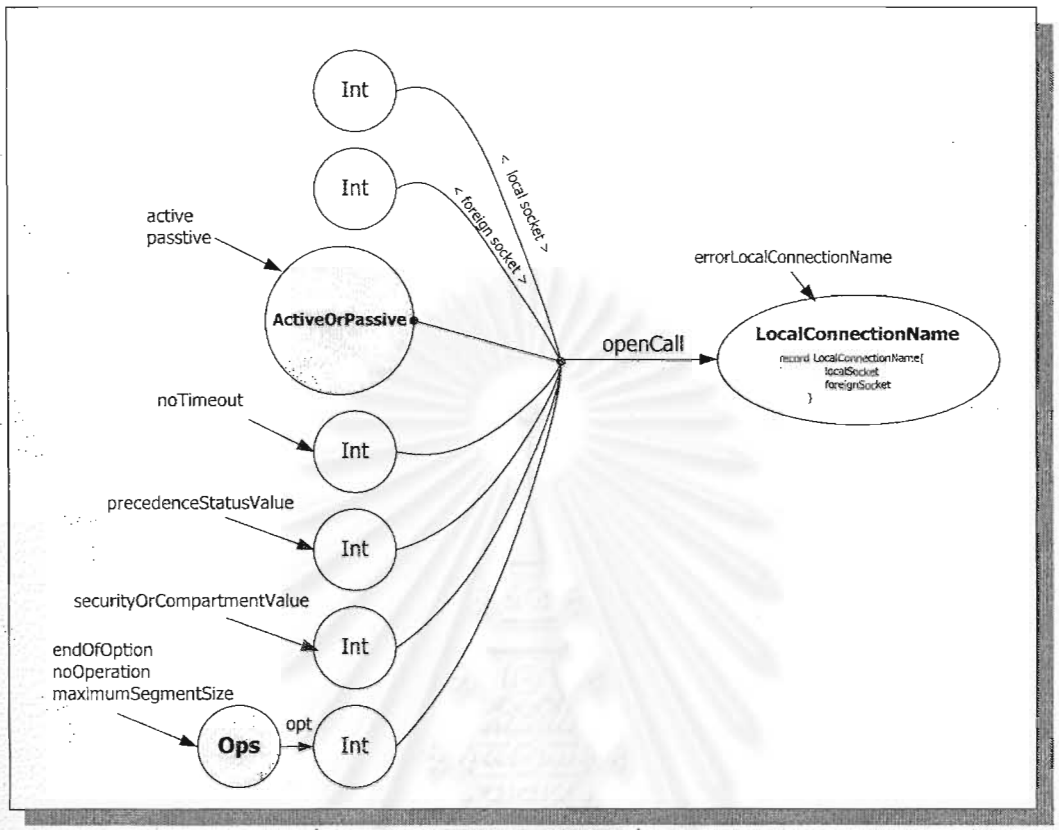
- Int แทน ที่อยู่ต้นทาง
- Int แทน ที่อยู่ปลายทาง
- ActiveOrPassive แทน ชนิดการทำการเรียกเปิดแบบ active หรือ passive
- Int แทน การพิจารณาการหมดเวลา โดยที่
  - noTimeout หมายถึง ไม่มีการพิจารณาการหมดเวลา
  - timeout หมายถึง ช่วงเวลาในการหมดเวลา
- Int แทน การทำก่อน (โดยมีค่าโดยปริยาย = 0)
- Int แทน ค่าความมั่นคง (โดยมีค่าโดยปริยาย = 1)
- Int แทน ค่าทางเลือก จะมี 3 ค่าด้วยกัน คือ 0,1 และ 2 โดยที่
  - ค่า 0 หมายถึง หมดรายการทางเลือก opt(endOfOption)
  - ค่า 1 หมายถึง ไม่มีการดำเนินการแล้ว opt (noOperation)
  - ค่า 2 หมายถึง ขนาดเซกเมนต์สูงสุด opt (maximumSegmentSize)

เมื่อบิต active/passive ถูกกำหนดให้เป็น passive สถานะของทีซีพีต้นทางจะถูกเปลี่ยนไปเป็น LISTEN (ดังสมการ eq3 ในรูปที่ 3.6) เพื่อรอการเรียกเปิดแบบ active (ซึ่งมีสถานะเป็น SYNSENT (ดังสมการ eq4 ในรูปที่ 3.6)) จากคู่ foreign socket ของทีซีพีปลายทาง แล้วรอคำสั่งเรียกส่ง

เมื่อเปิดการติดต่อแล้วจะสร้างกลุ่มระเบียนควบคุมการขนส่งข้อมูล (ดังสมการ eq5 ในรูปที่ 3.6) ไม่ว่าจะเป็นการเรียกเปิดแบบ active หรือ passive ก็ตาม เมื่อเรียกเปิดแล้วจะนำข้อมูลที่ได้รับมาเก็บเอาไว้เพื่อใช้งานในลำดับต่อไป

ทั้งนี้โอเอสจะทวนสอบสิทธิของผู้ใช้งานจากค่าการทำก่อนและค่าความมั่นคงที่ได้รับมาในการเปิดการติดต่อ หากไม่มีการเรียกใช้งานค่าดังกล่าวจะเรียกใช้ค่าโดยปริยาย

ผลลัพธ์หลังจากเปิดการเชื่อมต่อจะได้ชื่อในการเชื่อมต่อเฉพาะที่ เพื่อใช้อ้างถึงคู่กระบวนการเชื่อมต่อดังกล่าวนี้ในลำดับต่อๆ ไป (ดังสมการ eq1 และ eq2 ในรูปที่ 3.6)



รูปที่ 3.5 แผนภาพเอดีเจของคำสั่งเรียก "เปิด"

```
mod* OPENCALL{
        pr(SEGMENT + STATE + TCB + STATUS + STATUSCALL)
        *[ ActiveOrPassive ]*
-- OPEN (1:local port, 2:foreign socket, 3:active (1) /passive(0) 4:[,timeout] 5:[,precedence] 6:[,security/compartment]
7:[,options])
        op openCall : Int Int ActiveOrPassive Int Int Int Int -> LocalConnectionName          -- method
        op active : -> ActiveOrPassive
        op passive : -> ActiveOrPassive
        op timeout : -> Int
        vars I1 I2 I4 I5 I6 I7 : Int
        var S : Segment
        var O : Ops
        var P : ActiveOrPassive
        var C : Tcb
        var L : LocalConnectionName
        ceq openCall( I1, I2, P, I4, I5, I6, I7) = LocalConnectionName { localSocket  = I1, foreignSocket = I2 }
                if ( I1 > 0 ) and ( I2 > 0 ) and
                ( I1 =/= I2 ) and
                ( P == active or P == passive ) and
                I4 == noTimeout and
                I5 == precedenceStatusValue and
                I6 == securityOrCompartmentValue and
                I7 == opt(maximumSegmentSize) .          -- correct local + foreign port          ...(eq1)
        ceq openCall( I1, I2, P, I4, I5, I6, I7) = errorLocalConnectionName
                if ( I1 < 1 or I2 < 1 or  I1 == I2 ) and
                ( P == active or P == passive ) and
                I4 == noTimeout and
                I5 == precedenceStatusValue and
                I6 == securityOrCompartmentValue and
                I7 == opt(maximumSegmentSize) .          -- error IP          ...(eq2)
        ceq curState = listenState
                if  openCall(  I1,  I2,  P,  noTimeout,  precedenceStatusValue,  securityOrCompartmentValue,
opt(maximumSegmentSize) ) ==  LocalConnectionName { localSocket  = I1, foreignSocket = I2 } .          ...(eq3)
        ceq curState = synsentState
                if  openCall(  I1,  I2,  P,  noTimeout,  precedenceStatusValue,  securityOrCompartmentValue,
opt(maximumSegmentSize) ) ==  LocalConnectionName { localSocket  = I1, foreignSocket = I2 } .          ...(eq4)
        ceq checkTcb(createTcb) = true
                if openCall(I1,I2,P,I4,I5,I6,I7) == LocalConnectionName { localSocket  = I1, foreignSocket = I2 } .
                                                                                          ...(eq5)
}
```

รูปที่ 3.6 ข้อกำหนดรูปนัยคำสั่งเรียก "เปิด"

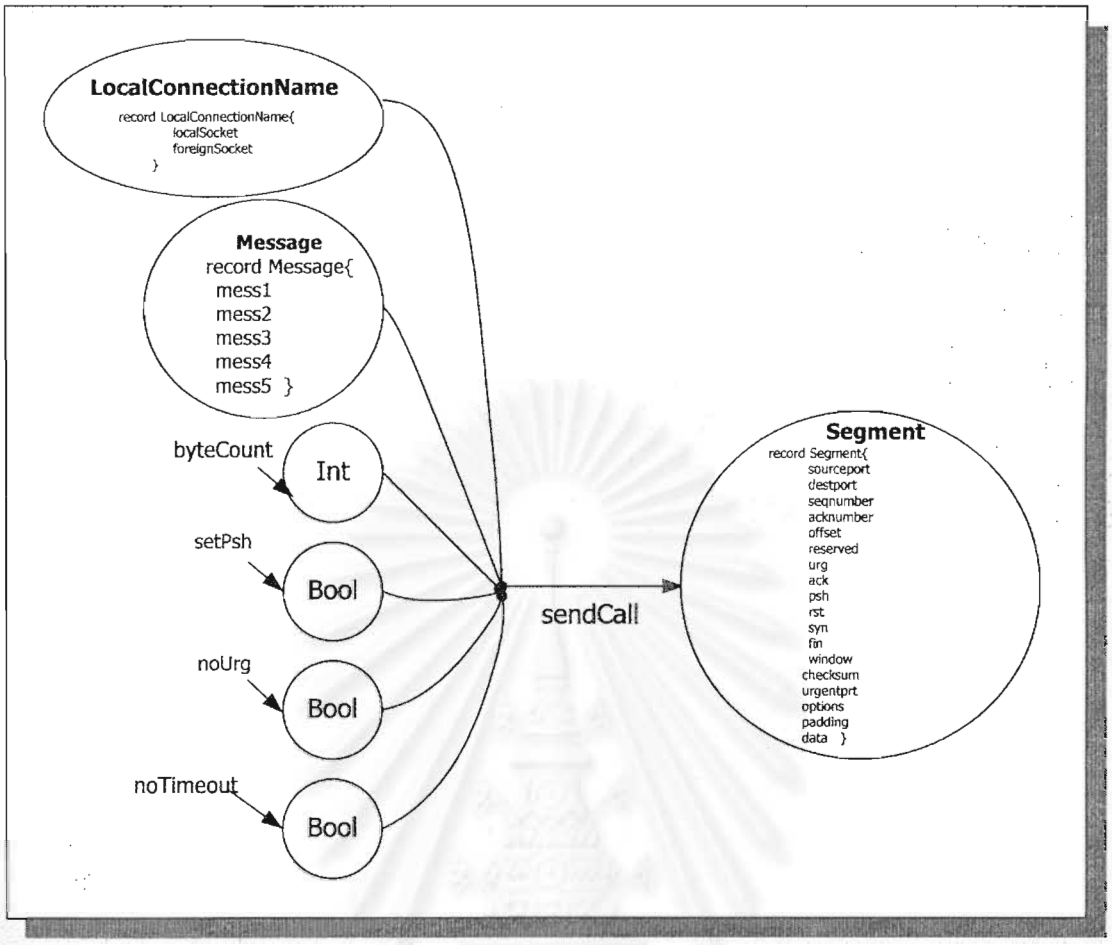
3.3.1.2 คำสั่งเรียก "ส่ง"

คำสั่งเรียก "ส่ง" ทำการรับข้อมูลดังนี้ "local connection name, buffer address, byte count, PUSH flag, URGENT flag, timeout" เพื่อทำการเรียก "ส่ง" ข้อมูลไปยังทีซีพีปลายทาง ดังการดำเนินการ sendCall (รูปที่ 3.7)

โดยที่ชนิดข้อมูลต่างๆ ที่รับเข้ามาดำเนินการ op sendCall : LocalConnectionName Message Int Bool Bool Bool -> Segment มีความหมายตามลำดับดังนี้

- LocalConnectionName แทน ชื่อในการเชื่อมต่อเฉพาะที่ (Local Connection Name)
- Message แทน ข้อมูล
- Int แทน จำนวนเซกเมนต์
- Bool แทน ตัวบ่งชี้การผลัก (PUSH flag) โดยที่
  - resetPsh หมายถึง ไม่มีตัวบ่งชี้การผลัก
  - setPsh หมายถึง มีตัวบ่งชี้การผลัก
- Bool แทน ตัวบ่งชี้การเร่ง (URG flag) โดยที่
  noUrg หมายถึง ไม่มีตัวบ่งชี้การเร่ง
- Bool แทน การพิจารณาการหมดเวลา โดยที่
  - noTimeout หมายถึง ไม่มีการพิจารณาการหมดเวลา
  - timeout หมายถึง ช่วงเวลาในการหมดเวลา

ข้อมูลที่บรรจุอยู่ใน Message จะถูกส่งไปยังเส้นทางที่เปิดไว้ตามชื่อในการเชื่อมต่อเฉพาะที่ โดยขั้นตอนแรกข้อมูลต่างๆ จะเก็บไว้ในบัฟเฟอร์ซึ่งถูกจัดเป็นเซกเมนต์ก่อนส่งไปยังทีซีพีปลายทาง โดยที่ข้อมูลนี้จะถูกส่งไปยังทีซีพีปลายทางก็ต่อเมื่อบิต Push มีค่าเป็น setPsh ส่งไปทีละเซกเมนต์ในปริมาณเท่ากับจำนวนเซกเมนต์ที่ได้ระบุมา ทั้งนี้ทีซีพีต้นทางและปลายทางต้องมีการเรียกเปิดถูกต้องเป็นที่เรียบร้อยแล้ว (ดังรายละเอียดข้อกำหนดรูปนัยการเรียก "ส่ง" Module SENDCALL ในภาคผนวก ก)



รูปที่ 3.7 แผนภาพเอดีเจของคำสั่งเรียก "ส่ง"

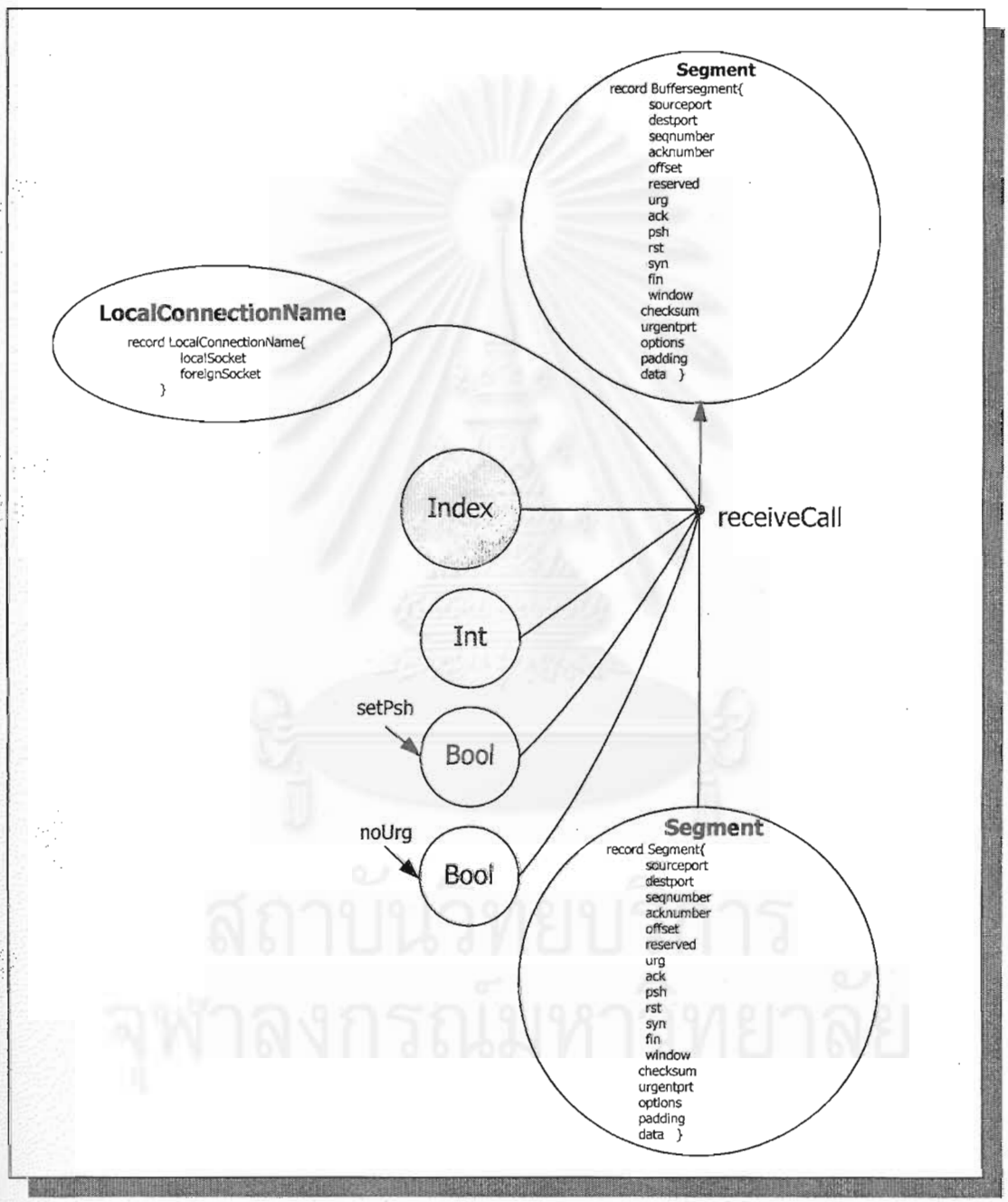
3.3.1.3 คำสั่งเรียก "รับ"

คำสั่งเรียก "รับ" ทำการรับข้อมูลดังนี้ "local connection name, buffer address, byte count, PUSH flag, URGENT flag, Segment" เพื่อทำการเรียก "รับ" ข้อมูลแล้วนำไปเก็บไว้ยังบัฟเฟอร์ ดังการดำเนินการ receiveCall (รูปที่ 3.8)

โดยที่ชนิดข้อมูลต่างๆ ที่รับเข้ามาดำเนินการ op receiveCall : LocalConnectionName Id Int Bool Bool Segment -> Buffersegment ความหมายตามลำดับดังนี้

- LocalConnectionName แทน ชื่อในการเชื่อมต่อเฉพาะที่ (Local Connection Name)
- Id แทน ที่อยู่บัฟเฟอร์
- Int แทน จำนวนเซกเมนต์

- Bool แทน ตัวบ่งชี้การผลัก (PUSH flag) โดยที่
  resetPsh หมายถึง ไม่มีตัวบ่งชี้การผลัก
  setPsh หมายถึง มีตัวบ่งชี้การผลัก
- Segment แทน เซกเมนต์ที่ถูกส่งมา



รูปที่ 3.8 แผนภาพเอดีเจของคำสั่งเรียก "รับ"

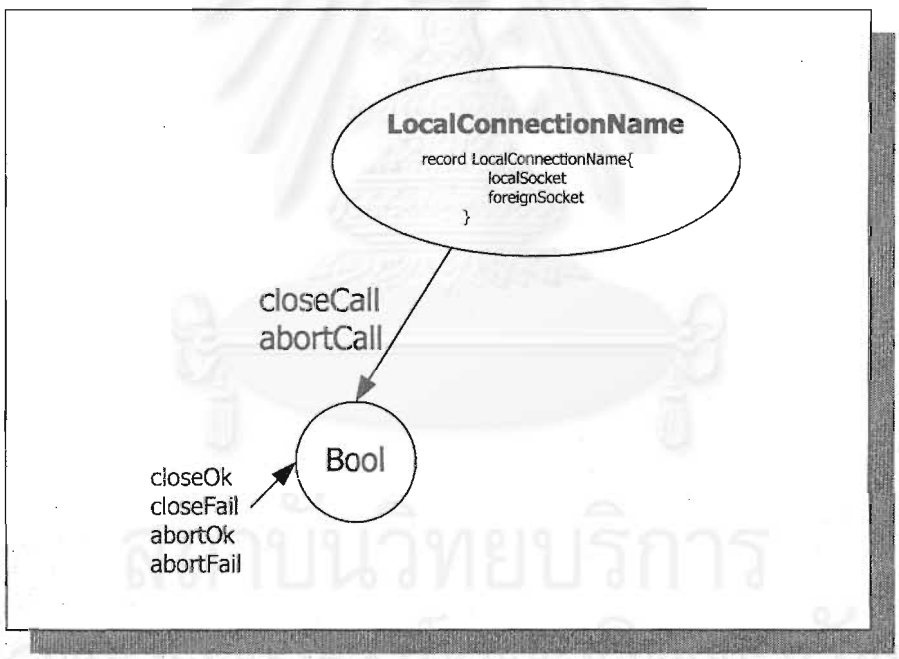
เมื่อบิต Push มีค่าเป็น setPsh ทีซีพีจะรับเซกเมนต์ตามเส้นทางที่เปิดไว้ตามชื่อในการเชื่อมต่อเฉพาะที่ แล้วจะนำเซกเมนต์ที่ได้รับมาไปเก็บตามที่อยู่บัฟเฟอร์ที่ได้ระบุมาและเก็บในปริมาณเท่ากับจำนวนเซกเมนต์ที่ได้รับมา อีกทั้งสถานะจะเปลี่ยนไปเป็น ESTABLISEHD (ดังรายละเอียดข้อกำหนดรูปนัยการเรียก "รับ" Module RECEIVECALL ในภาคผนวก ก)

3.3.1.4 คำสั่งเรียก "ปิด"

คำสั่งเรียก "ปิด" ทำการรับข้อมูล "local connection name" เพื่อทำการเรียก "ปิด" การเชื่อมต่อ ดังการดำเนินการ closeCall (รูปที่ 3.9)

โดยที่ชนิดข้อมูลต่างๆ ที่รับเข้ามาดำเนินการ op closeCall : LocalConnectionName -> Bool มีความหมาย ดังนี้

- LocalConnectionName แทน ชื่อในการเชื่อมต่อเฉพาะที่ (Local Connection Name)



รูปที่ 3.9 แผนภาพเอดีเจของคำสั่งเรียก "ปิด" และ "ยกเลิก"

เมื่อทำการปิดการเชื่อมต่อที่ได้ระบุไว้ตามชื่อในการเชื่อมต่อเฉพาะที่แล้วจะลบกลุ่มระเบียนควบคุมการขนส่งข้อมูลเป็นการคืนทรัพยากรที่เคยเรียกใช้งาน (ดังสมการ eq1 ในรูปที่ 3.10)

```
mod* CLOSECALL{
        pr(RECEIVECALL)
-- CLOSE(l1:local connection name)
        op closeCall : LocalConnectionName -> Bool                    -- method
        op closeOk : -> Bool
        op closeFail : -> Bool

        var C : Tcb
        var L : LocalConnectionName

        ceq closeCall(L) = closeOk if L =/= errorLocalConnectionName .
        ceq closeCall(L) = closeFail if L == errorLocalConnectionName .
        ceq deleteTcb(C) = no-tcb if closeCall(L) == closeOk .          ... (eq1)
}
```

รูปที่ 3.10 ข้อกำหนดรูปนัยคำสั่งเรียก "ปิด"

3.3.1.5 คำสั่งเรียก "ยกเลิก"

คำสั่งเรียก "ปิด" ทำการรับข้อมูล "local connection name" เพื่อทำการเรียก "ยกเลิก" การเชื่อมต่อ ดังการดำเนินการ abortCall (รูปที่ 3.11)

โดยที่ชนิดข้อมูลต่างๆ ที่รับเข้ามาดำเนินการ op abortCall : LocalConnectionName -> Bool มีความหมาย ดังนี้

- LocalConnectionName แทน ชื่อในการเชื่อมต่อเฉพาะที่ (Local Connection Name)

เมื่อทำการยกเลิกการเชื่อมต่อที่ได้ระบุไว้ตามชื่อในการเชื่อมต่อเฉพาะที่แล้วจะลบกลุ่มระเบียนควบคุมการขนส่งข้อมูลซึ่งเป็นการคืนทรัพยากรที่เคยเรียกใช้งาน (ดังสมการ eq1 ในรูปที่ 3.10)

3.3.1.6 คำสั่งเรียก "สถานภาพ"

คำสั่งเรียก "สถานภาพ" ทำการรับข้อมูล "local connection name" เพื่อทำการเรียก "สถานภาพ" การเชื่อมต่อ ดังการดำเนินการ statusCall (รูปที่ 3.12)

โดยที่ชนิดข้อมูลต่างๆ ที่รับเข้ามาดำเนินการ op statusCall : LocalConnectionName -> Status มีความหมาย ดังนี้

- LocalConnectionName แทน ชื่อในการเชื่อมต่อเฉพาะที่ (Local Connection Name)

ทั้งนี้ข้อมูลต่างๆ จะถูกเก็บไว้ใน record Status ในมอดูล STATUS (ในภาคผนวก ก) ส่งค่าการเก็บออกมาเป็นจริงหรือเท็จดังสมการในรูปที่ 3.13

```
mod* ABORTCALL{
        pr(TCB + OPENCALL)

        op abortCall : LocalConnectionName -> Bool                    -- method
        op abortOk : -> Bool
        op abortFail : -> Bool

        var C : Tcb
        var L : LocalConnectionName

        ceq abortCall(L) = abortOk if L =/= errorLocalConnectionName .
        ceq abortCall(L) = abortFail if L == errorLocalConnectionName .
        ceq deleteTcb(C) = no-tcb if abortCall(L) == abortOk .          ... (eq1)
}
```

รูปที่ 3.11 ข้อกำหนดรูปนัยคำสั่งเรียก "ยกเลิก"

```
mod* STATUSCALL{
        pr(STATUS)

        op statusCall : LocalConnectionName -> Status          -- method

        var L : LocalConnectionName

        ceq checkStatus(statusCall(L)) = true if localSocket(L) > 0 and foreignSocket(L) > 0 .
        ceq checkStatus(statusCall(L)) = false if ( localSocket(L) < 1 ) or ( foreignSocket(L) < 1 ) .
}
```

รูปที่ 3.12 ข้อกำหนดรูปนัยคำสั่งเรียก "สถานภาพ"

3.3.2 การดำเนินการ (Operation)

ทีซีพีมีการส่งข้อมูลแบบกระแสต่อเนื่องในรูปของเซกเมนต์ เพื่อส่งผ่านระบบอินเทอร์เน็ต โดยในการรับส่งข้อมูลระหว่างทีซีพีนั้น มีการตรวจสอบข้อมูลว่าสูญหายหรือซ้ำซ้อนหรือไม่ด้วยบิต syn และ ack ตรวจสอบหรือแสดงปริมาณข้อมูลที่ได้รับส่งจากบิต Windows

ทั้งนี้ในการรับส่งข้อมูลกระบวนการหนึ่งๆ จะส่งข้อมูลไปตามที่อยู่หรือช่องทางที่ได้ระบุเอาไว้ อีกทั้งกระบวนการจัดการเชื่อมต่อทั้งหมดนี้ต้องเก็บข้อมูลแสดงรายละเอียดต่างๆ ที่ใช้ในการรับส่งข้อมูลการเชื่อมต่อไว้ด้วย

งานวิจัยนี้จึงออกแบบระเบียนเซกเมนต์ (ดังรูปที่ 2.2) โดยที่บิต syn ack และ windows มีหน้าที่ตรวจสอบการรับส่งข้อมูลและปริมาณข้อมูลตามลำดับ โดยส่งข้อมูลตามช่องทางต้นทาง (sourceport) และช่องทางปลายทาง (destport) (มีชนิดข้อมูลเป็นเลขจำนวนนับหมายถึงมีจำนวนช่องทางหลายช่องทางด้วยกันเพื่อสนับสนุนการรับส่งหลายๆ ทาง) ที่ระบุในระเบียนเซกเมนต์และเก็บข้อมูลแสดงสภาวะต่างๆ ที่ใช้ในการเชื่อมต่อไว้ในระเบียนสถานภาพดังแสดงรายละเอียดในรูปที่ 3.13

```
record Status{
        local-socket : Int
        foreign-socket : Int
        rcv-wnd : Int
        snd-wnd : Int
        connection-state : Int
        num-buf-wait-ack : Int
        num-buf-pend-receipt : Int
        urg-state : Int                 -- no urg-state = 0
        precedence-status : Int         -- default value = 0
        security-or-compartment : Int   -- default value = 1 (have security), value = 0 ( no secure)
        trans-timeout : Int             -- default value = 0
}
```

รูปที่ 3.13 ระเบียนสถานภาพ

กระบวนการตรึงซ็อกเก็ตตามช่องทางที่ยึดเหนี่ยวไว้ตามกระบวนงานนั้นๆ ดังรูปที่ 2.6 สามารถแสดงการทำงานโดยจะเก็บ localSocket และ foreignSocket ซึ่งอยู่ในระเบียน LocalConnectionName ไว้ใน local-socket (สมการที่ 3) และ foreign-socket (สมการที่ 4) ในระเบียนสถานภาพตามลำดับ จากนั้นจะทำการตรึงด้วยการดำเนินการ bind เพื่อใช้งานในการเชื่อมซ็อกเก็ตเข้ากับช่องทาง sourceport (สมการที่ 5) และ destport (สมการที่ 6) ในระเบียน Segment ต่อไป

eq local-socket(Status) = localSocket(LocalConnectionName) . ...(1)

eq foreign-socket(Status) = foreignSocket(LocalConnectionName) . ...(2)

eq sourceport(Segment) = bind(localSocket(LocalConnectionName)) . ...(3)

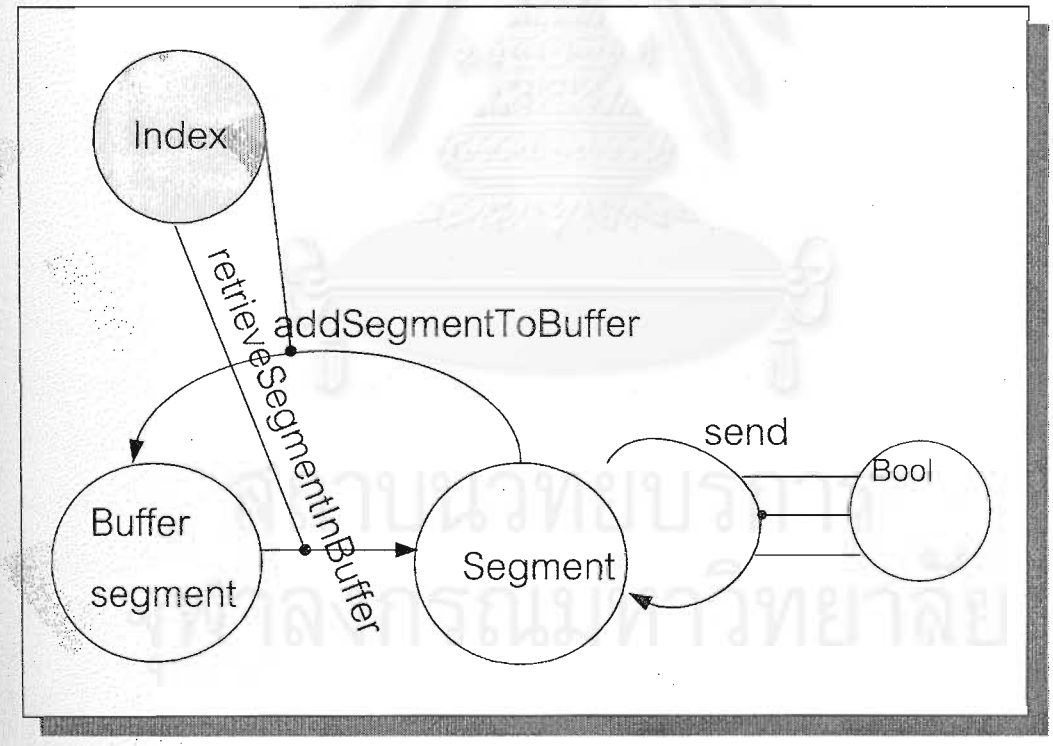
eq destport(Segment) = bind(foreignSocket (LocalConnectionName)) . ...(4)

3.3.3 ส่วนย่อยของระบบระหว่างเครือข่าย (Element of the Internetwork System)

กระบวนการจำเป็นต้องสามารถติดต่อหลายๆ ทางพร้อมกัน ซึ่งแต่ละกระบวนการมีช่องทางเพื่อใช้ในการติดต่อ งานวิจัยนี้จึงออกแบบกระบวนการที่ทำการติดต่อกับช่องทางหลายๆ ทางด้วย sourceport และ destport ซึ่งมีชนิดข้อมูลเป็นจำนวนเต็ม หมายถึงสามารถติดต่อหลาย ๆ ทางพร้อมกันได้

3.3.4 แบบจำลองของการดำเนินการ (Model of Operation)

ทีซีพีบรรจุข้อมูลจากบัฟเฟอร์ไปยังเซกเมนต์และรอคำสั่งเรียกจากอินเทอร์เน็ตมอดูลเพื่อส่งเซกเมนต์ไปยังทีซีพีปลายทาง ทีซีพีปลายทางเมื่อได้รับข้อมูลในรูปของเซกเมนต์ แล้วจะนำไปเก็บยังบัฟเฟอร์ของผู้รับเพื่อนำข้อมูลไปใช้งานต่อๆ ไป



รูปที่ 3.14 แบบจำลองของการดำเนินการ

งานวิจัยนี้จึงออกแบบ (ดูรูป 3.14) การดำเนินการ retrieveSegmentInBuffer คือ

op retrieveSegmentInBuffer : Index Buffersegment -> Segment

ซึ่งแสดงถึงการบรรจุข้อมูลจากบัฟเฟอร์ไปยังเซกเมนต์ แล้วรอคำสั่งเรียกจากอินเทอร์เน็ต มอดูล เมื่อได้รับคำสั่งแล้วทำการส่งไปยังทีซีพีปลายทางด้วยการดำเนินการ send ดัง

op send : Segment Bool Bool Bool -> Segment

การดำเนินการ send จะส่งได้ก็ต่อเมื่อ การตรวจสอบ State Tcb และ Status ที่ส่งค่ากลับมาเป็น Bool ตัวที่ 1 2 และ 3 ตามลำดับมีความถูกต้อง เซกเมนต์ดังกล่าวจึงถูกส่งไปยังทีซีพีปลายทาง

ทีซีพีปลายทางได้รับข้อมูลเซกเมนต์ คือ output ของการดำเนินการ send นี้ แล้วเซกเมนต์ที่ได้รับถูกนำไปเก็บยังบัฟเฟอร์ของผู้รับด้วยการดำเนินการ ดัง

op addSegmentToBuffer : Index Segment -> Buffersegment

3.3.5 สภาพแวดล้อมแม่ข่าย (Host Environment)

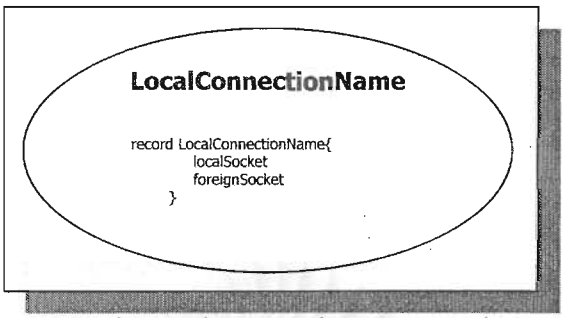
ทีซีพีใช้คำสั่งเรียกจากมอดูลของโพรโทคอลอินเทอร์เน็ตดังนี้ คำสั่งเรียก "เปิด" "ส่ง" "รับ" "ปิด" "ยกเลิก" และ "สถานภาพ" (ดังแสดงรายละเอียดที่ได้กล่าวมาแล้วในหัวข้อ 3.2.1)

3.3.6 การติดตั้งและการลบล้างการเชื่อมต่อ (Connection Establishment and Clearing)

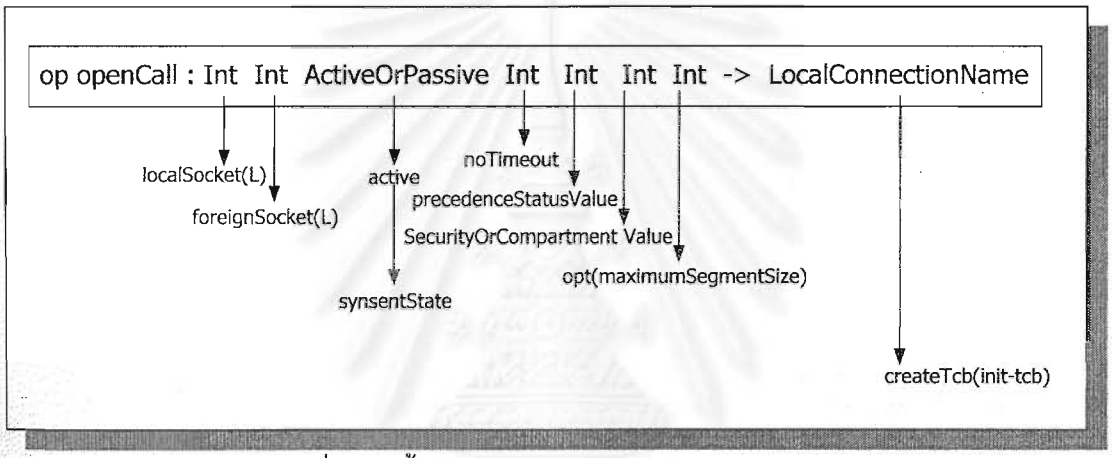
ทีซีพีระบุที่อยู่ของอินเทอร์เน็ตโดยผ่านทางตัวระบุช่องทาง เพื่อใช้สร้างซ็อกเก็ตที่ไม่ซ้ำกันในการรับส่งข้อมูลทั้งเครือข่าย การติดตั้งการเชื่อมต่อจะเรียกผ่านคำสั่งเรียกต่างๆ ดังนี้ ขั้นแรกทีซีพีจะเรียกผ่านคำสั่งเรียก "เปิด" เพื่อเปิดการติดต่อแบบ active (ดังรูปที่ 3.16) จับคู่กับการเรียกเปิดแบบ passive (ดังรูปที่ 3.17) มีการเก็บข้อมูลต่างๆ เอาไว้ รวมไปถึงชื่อที่ใช้อ้างถึงในการเชื่อมต่อ (ดังรูปที่ 3.15) เมื่อทำการติดตั้งการเชื่อมต่อเป็นที่เรียบร้อยแล้ว ทีซีพีจะส่งข้อมูลโดยผ่าน คำสั่งเรียก "ส่ง" (ดังรูปที่ 3.18) และทีซีพีผู้รับจะรับข้อมูลด้วยคำสั่งเรียก "รับ" (ดังรูปที่ 3.19) ท้ายสุดจะปิดการเชื่อมต่อด้วยคำสั่งเรียก "ปิด" (ดังรูปที่ 3.20) หรือการสั่งยกเลิกการเชื่อมต่อด้วยคำสั่งเรียก "ยกเลิก"

คำสั่งเรียก "เปิด" แบบ passive (ดังรูปที่ 3.17) จะทำการเปิดช่องทางการติดต่อด้วยตัวระบุช่องทาง localSocket(L) และ foreignSocket(L) และเมื่อการเรียกเปิดเป็นแบบ passive สถานะจะเปลี่ยนจาก closedState ไปยัง listenState เพื่อรอการเรียกเปิดแบบ active จากอีกทีซีพีหนึ่ง ทั้งนี้ข้อมูลที่ได้รับมาในการเรียกเปิดนั้นก็นำไปเก็บในระเบียนสถานภาพและเซกเมนต์ พร้อมทั้งสร้างชื่อในการเชื่อมต่อเฉพาะที่จากหมายเลขซ็อกเก็ตต้นทางและปลายทางที่ได้

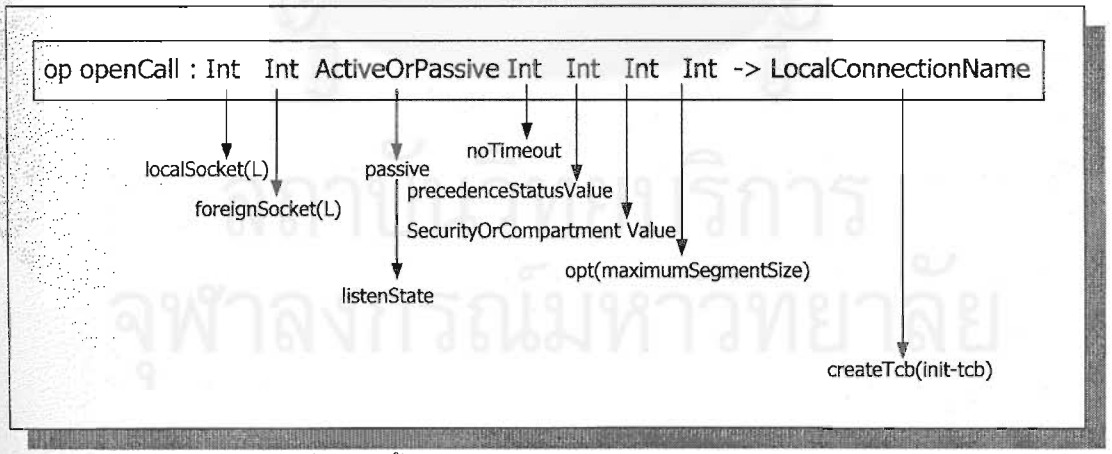
รับมาเพื่อใช้ในการอ้างถึงการติดต่อดังกล่าว (ดังรูปที่ 3.15) ท้ายสุดจะทำการสร้างระเบียนควบคุมการขนส่งข้อมูล



รูปที่ 3.15 ชื่อในการเชื่อมต่อเฉพาะที่



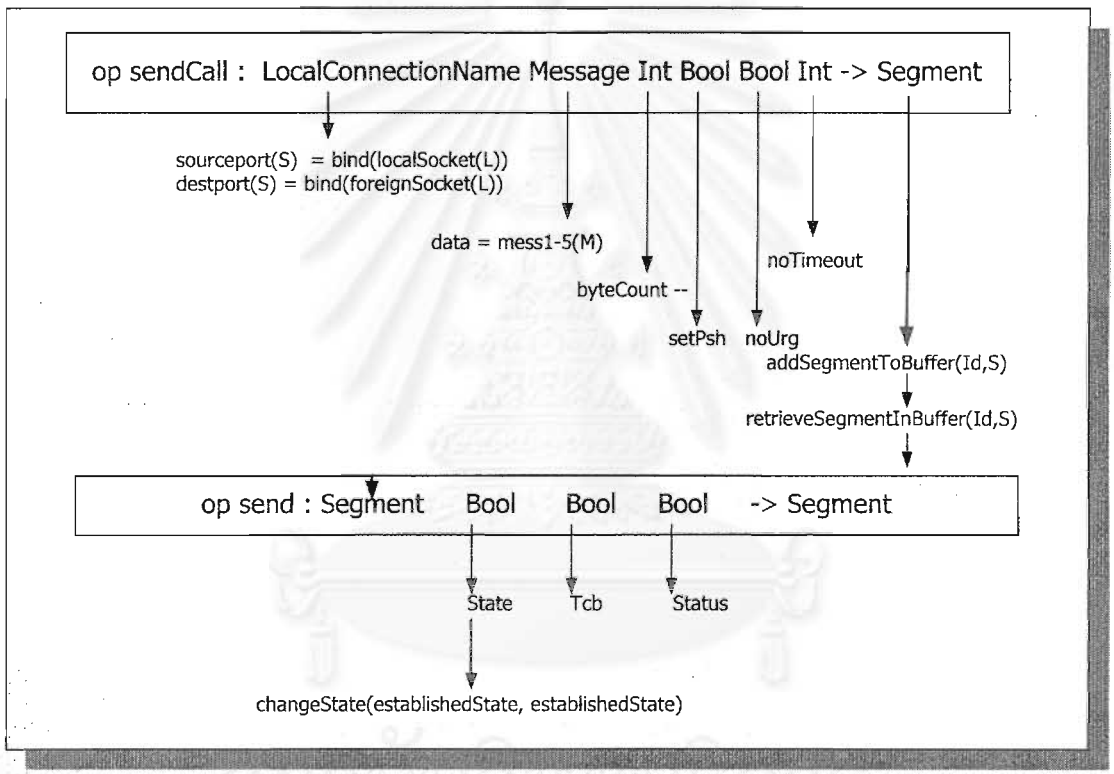
รูปที่ 3.16 ขั้นตอนการเปิดการติดต่อแบบ active



รูปที่ 3.17 ขั้นตอนการเปิดการติดต่อแบบ passive

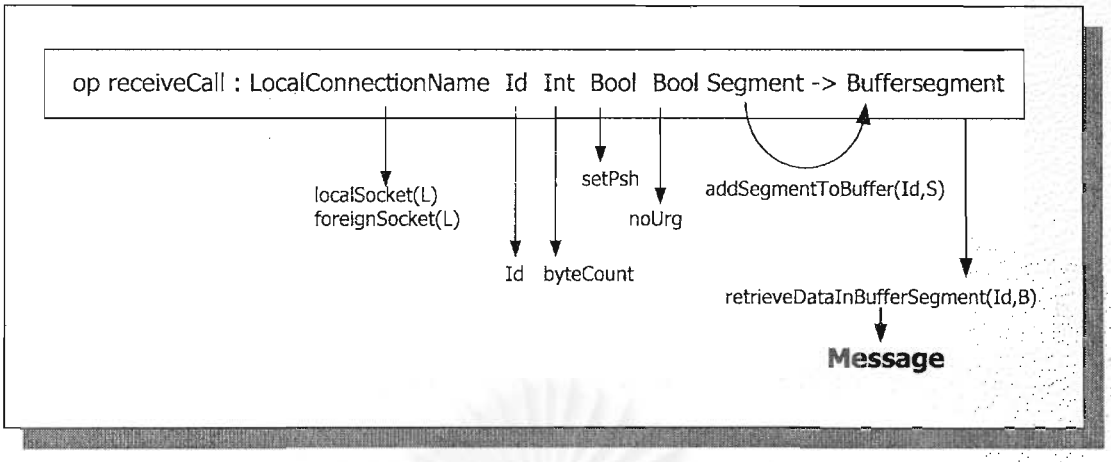
คำสั่งเรียก "เปิด" แบบ active (ดังรูปที่ 3.16) มีหน้าที่การทำงานคล้ายกับการเรียกเปิดแบบ passive แต่ต่างกันตรงที่สถานะจะเปลี่ยนไปเป็นสถานะ synsentState

หลังจากเปิดการติดต่อเรียบร้อยแล้ว ซีพีทีที่เรียกเปิดแบบ active จะทำการเรียกคำสั่งเรียก "ส่ง" (op sendCall ในรูปที่ 3.18) สถานะจะถูกเปลี่ยนไปเป็น establishedState เพื่อทำการส่งเซกเมนต์ไปยังเส้นทางที่ได้เปิดไว้ตามชื่อในการเชื่อมต่อเฉพาะที่ โดยในขั้นตอนแรกข้อมูลทุกอย่างที่ได้รับมาในขั้นตอนการเรียกเปิดรวมกับข้อมูลในคำสั่งเรียกส่งจะถูกบรรจุเป็นเซกเมนต์ แล้วจัดเก็บไว้ในบัฟเฟอร์ก่อนถูกส่งไปตามทีซีพีปลายทาง (ด้วยการดำเนินการ send ในรูปที่ 3.18) ทั้งนี้ก่อนที่เซกเมนต์ดังกล่าวจะถูกส่งไปยังทีซีพีปลายทาง (output ของ op send ในรูปที่ 3.18) ได้นั้น จะผ่านการตรวจสอบสถานะ ระเบียน ควบคุมการขนส่งข้อมูล และระเบียนสถานภาพ

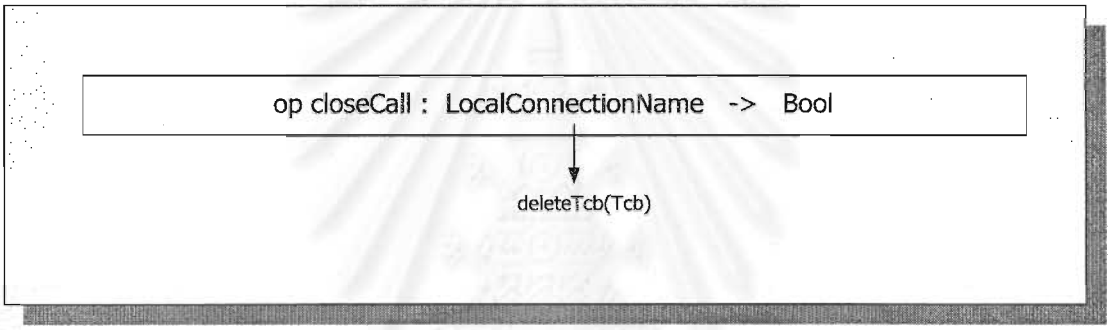


รูปที่ 3.18 ขั้นตอนการส่งข้อมูล

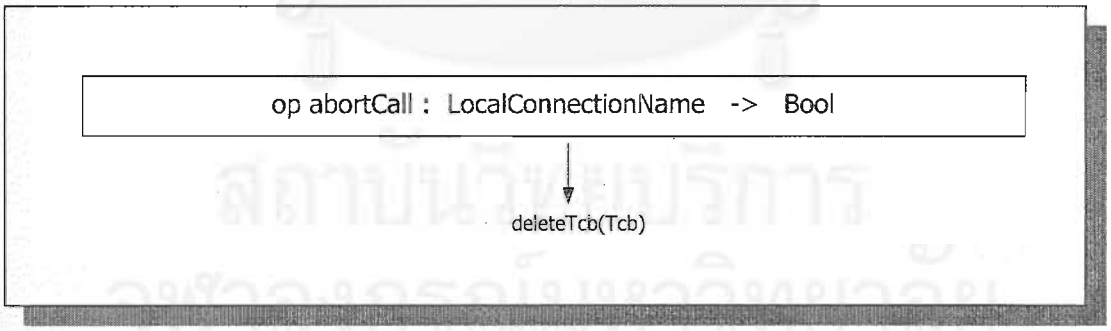
เซกเมนต์ที่ถูกส่งมานั้น ทีซีพีปลายทางจะรับข้อมูลด้วยคำสั่งเรียก "รับ" (ดังรูปที่ 3.8) ตามชื่อในการเชื่อมต่อเฉพาะที่ ทั้งนี้เมื่อทีซีพีได้รับข้อมูลในเซกเมนต์เรียบร้อยแล้วจะเก็บข้อมูลที่ได้รับมาไว้ในบัฟเฟอร์ (ด้วยการดำเนินการ addSegmentToBuffer ในรูปที่ 3.19) เพื่อนำข้อมูลไปใช้งานต่อไป (ด้วยการดำเนินการ retrieveDataInBufferSegment ในรูปที่ 3.19)



รูปที่ 3.19 ขั้นตอนการรับข้อมูล



รูปที่ 3.20 ขั้นตอนการปิดการติดต่อ



รูปที่ 3.21 ขั้นตอนการยกเลิกการติดต่อ

การปิดการเชื่อมต่อทำงานด้วยคำสั่งเรียก "ปิด" ซึ่งจะทำการลบล้างการเชื่อมต่อตามชื่อการเชื่อมต่อเฉพาะที่เพื่อคืนทรัพยากรที่เคยเรียกใช้งาน (ดังรูปที่ 3.20)

การยกเลิกการเชื่อมต่อนั้นทำงานด้วยคำสั่งเรียก "ยกเลิก" ซึ่งมีการทำงานคล้ายกับคำสั่งเรียก "ปิด" จะทำการลบล้างการเชื่อมต่อตามชื่อการเชื่อมต่อเฉพาะที่เพื่อคืนทรัพยากรที่เคยเรียกใช้งานเช่นกัน (ดังรูปที่ 3.21)

3.3.7 ข้อกำหนดฟังก์ชัน (Function Specification)

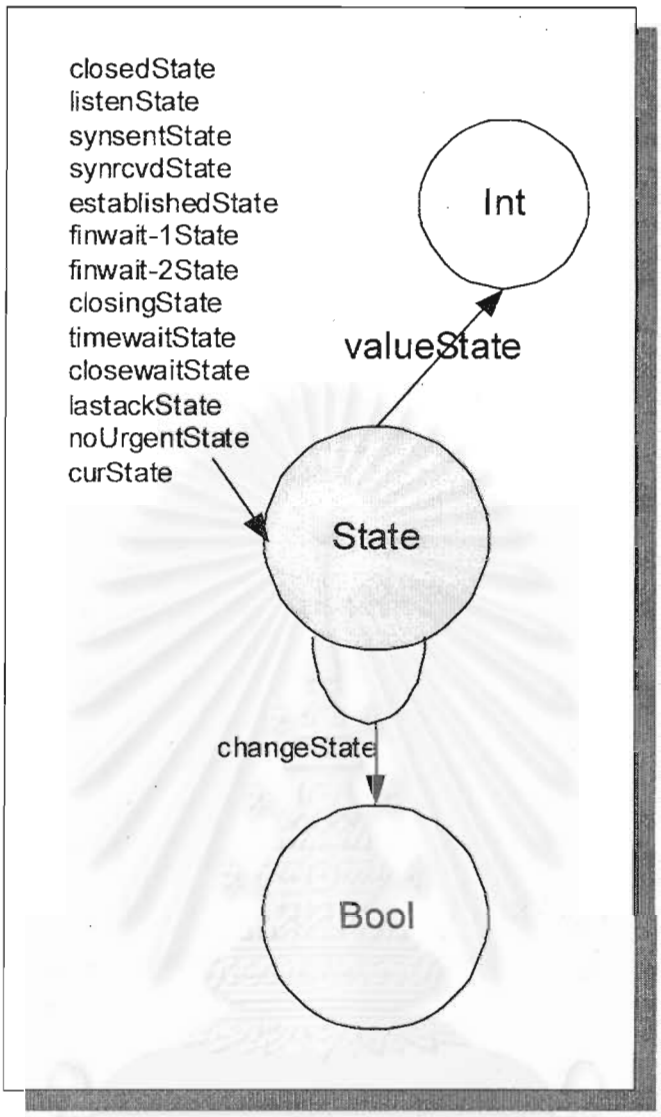
การเชื่อมต่อของทีซีพีเก็บรายละเอียดต่างๆ ที่ใช้ในการรับส่งข้อมูลในการเชื่อมต่อใน "กลุ่มระเบียนควบคุมการขนส่งข้อมูล" (รูปที่ 2.9) ซึ่งกระบวนการเชื่อมต่อดำเนินไปตามสถานะต่างๆ ตามข้อกำหนดเชิงหน้าที่ของทีซีพี [8]

งานวิจัยนี้จึงออกแบบระเบียนควบคุมการขนส่ง (ดังรูป 2.9) เพื่อเก็บรายละเอียดจากรูปที่ 2.10 และกำหนดชนิดข้อมูลของสถานะเป็น State เพื่อแสดงสถานะต่างๆ ตามข้อกำหนดเชิงหน้าที่ที่ระบุไว้ใน RFC793 [8] ดังรูปที่ 3.24 (ดูมอดูล STATE ในภาคผนวก ก)

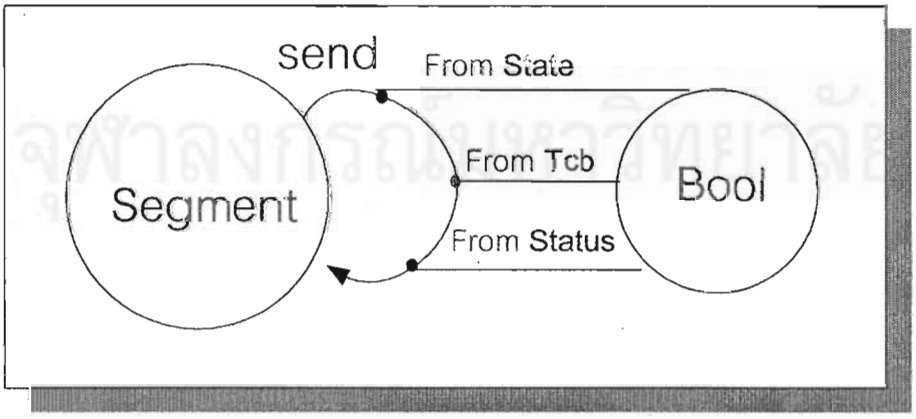
กระบวนการเชื่อมต่อจากทีซีพีหนึ่งไปยังอีกทีซีพีหนึ่งนั้นพิจารณาสถานะ (State) คำสั่งเรียก และเซกเมนต์ ดังนั้นในงานวิจัยนี้จึงออกแบบการรับส่งทีซีพีด้วย การดำเนินการ send ดังนี้

op send : Segment Bool Bool Bool -> Segment

โดยที่ Bool ลำดับที่ 1 2 และ 3 มาจากการตรวจสอบการทำงานของ State TCB และ Status ตามลำดับ เมื่อตรวจสอบค่าต่างๆ เรียบร้อยแล้วจะส่งค่ากลับมาเป็นจริงหรือเท็จก่อนส่งเซกเมนต์ไปยังทีซีพีปลายทางดังที่อยู่ที่ได้ระบุไว้ Segment (ดูรูป 3.23)



รูปที่ 3.22 แผนภาพเอดีเจของการเปลี่ยนสถานะ



รูปที่ 3.23 การดำเนินการส่ง

3.4 แผนภาพการเปลี่ยนสถานะของโพรโทคอลทีซีพี

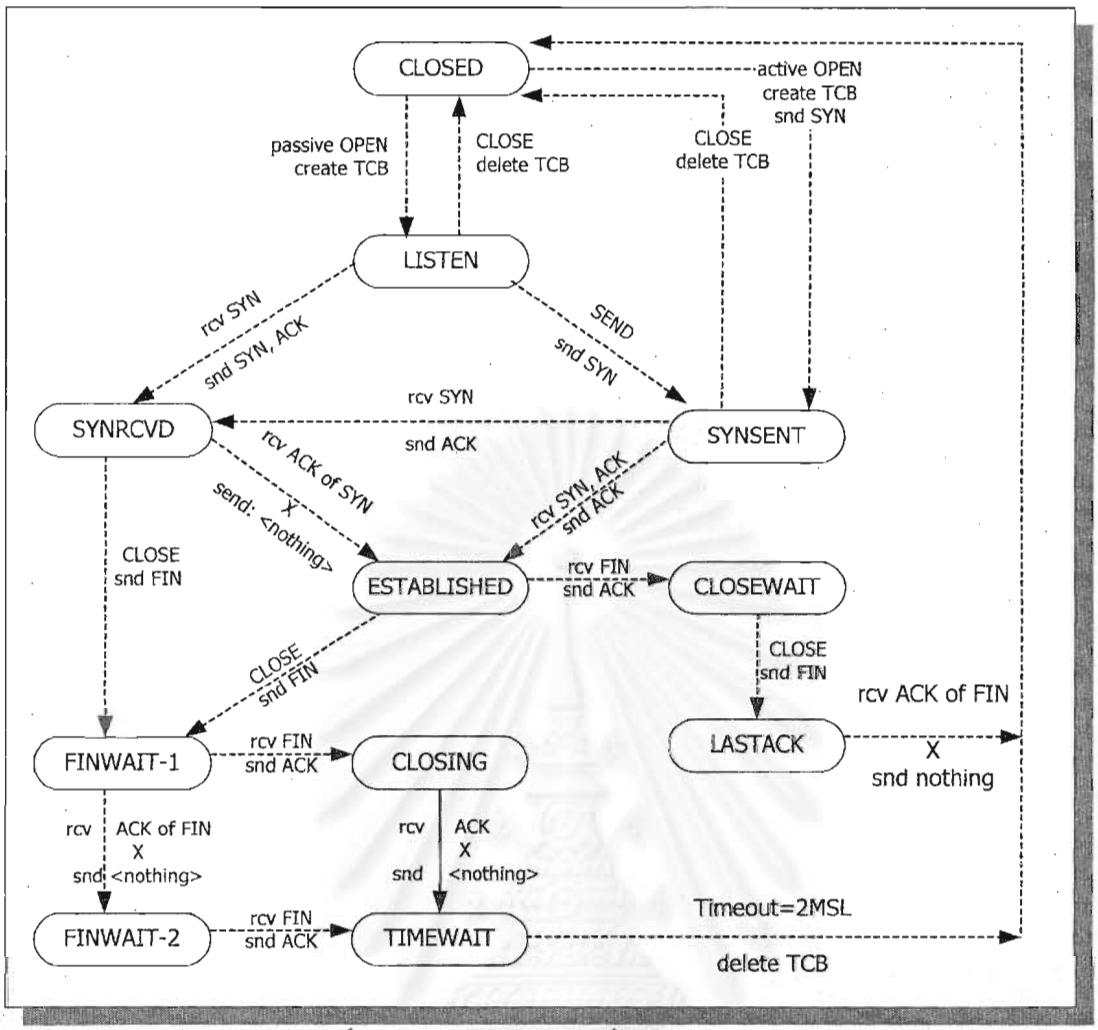
ทำหน้าที่อธิบายสถานะการทำงานต่างๆ ของทีซีพี [11] ดังรายละเอียดในหัวข้อ 2.2.7 โดยการทำงานของแผนภาพนี้โยงการทำงานไปยังมอดูล STATE CALLS TCB STATUS และ SEGMENT (รวมถึงมอดูล MESSAGE BUFFER และ INDEX ด้วย)

ในแผนภาพทีซีพี (รูปที่ 3.24) โยงการทำงานในมอดูล STATE ได้โดย การดำเนิน-การ changeState(T,T') มีสถานะนำเข้า (ต้นทางลูกศร=T) และสถานะส่งออก (ปลายลูกศร=T') (ดูรูปที่ 3.24)

ในแผนภาพทีซีพี (รูปที่ 3.24) มีรายละเอียดที่แสดงไว้ข้างๆ ลูกศรในการสร้าง หรือ ลบ TCB ซึ่งโยงการทำงานในมอดูล TCB (ดูรูปที่ 2.9) ได้โดยการดำเนินการ createTcb และ deleteTcb(Tcb) ตามลำดับ

ในแผนภาพทีซีพี (รูปที่ 3.24) มีรายละเอียดที่แสดงไว้ข้างๆ ลูกศรของคำสั่งเรียก OPEN (แบบ passive หรือแบบ active) SEND CLOSE ซึ่งโยงการทำงานในมอดูล OPENCALL SENDCALL CLOSECALL ด้วยการดำเนินการ openCall (ในหัวข้อ 3.3.1.1) sendCall ในหัวข้อ 3.2.1.2) และ closeCall (ในหัวข้อ 3.3.1.4)

ในแผนภาพทีซีพี (รูปที่ 3.24) มีรายละเอียดของเซกเมนต์ที่แสดงไว้ข้างๆ ลูกศร คือ syn ack fin หรือ data ซึ่งรายละเอียดดังกล่าวจะถูกระบุอยู่ใน Segment { syn, ack, fin, data} (รูปที่ 2.2) ที่ถูกรับส่งระหว่างทีซีพี



รูปที่ 3.24 แผนภาพการเปลี่ยนสถานะ [11]

3.5 กฎการเชื่อมแผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะ

การใช้งานข้อกำหนดโพรโทคลอทีซีพีนี้ ทำได้โดยเริ่มจากแผนภาพการเปลี่ยนสถานะ แล้วโยงรายละเอียดข้อมูลของทีซีพีไปยังแผนภาพเอดีเจได้ด้วยตารางที่ 3.1 ดังนี้

ตารางที่ 3.1 ตารางกฎการเชื่อมแผนภาพเอดีเจและแผนภาพการเปลี่ยนสถานะ

แผนภาพเอดีเจ	แผนภาพการเปลี่ยนสถานะ
openCall(I1,I2,passive,I4,I5,i6,I7), openCall(I1,I2,active,I4,I5,i6,I7), sendCall(), closeCall()	passive OPEN, active OPEN, SEND, CLOSE
op checkTcb(C) โดยที่ C คือ createTcb deleteTcb ตามลำดับ	TCB (create, delete)
T และ T' ใน op ChangeState(T,T')	State นำเข้า (T) State ส่งออก (T')

3.6 ตารางการตัดสินใจ

ขั้นตอนการทำงานที่สำคัญๆ ของข้อกำหนดโพรโทคอลทีซีพี ดังแสดงด้วยภาษาคาเฟ่โอบีเจในตารางที่ 3.2 ดังนี้

ตารางที่ 3.2 ตารางการตัดสินใจ

Rule Numbers	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Condition 1: B1 == changeState(T1,T2)	T		T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
Condition 2: B2 == checkTcb(createTcb)	T	T																		
Condition 3: B2 == checkTcb(deleteTcb(C))			T														T			
Condition4: B3 == checkState(U)	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
Action 1: T == closedState and T' == listenState	X																			
Action 2: T == closedState and T' == synsentState		X																		
Action 3: T == listenState and T' == closedState			X																	
Action 4: T == listenState and T' == synsentState				X																
Action 5: T == listenState and T' == synrcvdState					X															
Action 6: T == synsentState and T' == closedState						X														
Action 7: T == synsentState and T' == synrcvdState							X													
Action 8: T == synsentState and T' == establishedState								X												
Action 9: T == synrcvdState and T' == establishedState									X											
Action10: T == synrcvdState and T' == finwait-1State										X										
Action11: T == establishedState and T' == finwait-1State											X									
Action12: T == establishedState and T' == closewaitState												X								
Action13: T == finwait-1State and T' == closingState													X							
Action14: T == finwait-1State and T' == finwait-2State														X						
Action15: T == closingState and T' == timewaitState															X					
Action16: T == finwait-2State and T' == timewaitState																X				
Action17: T == timewaitState and T' == closedState																	X			
Action18: T == closewaitState and T' == lastackState																		X		
Action19: T == lastackState and T' == closedState																			X	
Action20: T == establishedState and T' == establishedState																				X

บทที่ 4

ข้อกำหนดรูปนัยโพรโทคอลทีซีพี

การจัดรูปนัยทีซีพีในงานวิจัยนี้ แบ่งอธิบายออกเป็นสองส่วนใหญ่ๆ ด้วยกัน ส่วนแรกคือ ชั้นทีซีพี (TCP Layer) ซึ่งเป็นการทำงานภายในของทีซีพีโดยละเอียด และส่วนที่สอง คือ ระดับบน (Top Level) ซึ่งเป็นส่วนการทำงานของการเรียกใช้งานจากผู้ใช้จริงๆ

4.1 ชั้นทีซีพี

การจัดรูปนัยทีซีพีเริ่มจากวิเคราะห์และออกแบบแผนภาพเอดีเจจาก RFC793 [8] ซึ่งแสดงรายละเอียดข้อมูลทั้งหมดของทีซีพี โดยพิจารณาสถานะการทำงานต่างๆ จากแผนภาพการเปลี่ยนสถานะ และแสดงการจัดรูปนัยการรับส่งข้อมูลตามแผนภาพเส้นกำหนดเวลา

ข้อกำหนดรูปนัยโพรโทคอลทีซีพี แบ่งออกเป็น 15 มอดูลด้วยกัน คือ มอดูล INDEX MESSAGE SEGMENT BUFFER STATE TCB STATUS STATUSCALL SEND OPENCALL SENDCALL RECEIVECALL CLOSECALL ABORTCALL และ TCP-SYSTEM โดยแสดงรายละเอียดการทำงานดังต่อไปนี้

4.1.1 มอดูล INDEX

ทำหน้าที่เป็นดรรชนี (Index) ของบัฟเฟอร์ ดรรชนีเริ่มที่ค่า 0 (eq1 ในรูป-ที่ 4.4.1) และค่าสูงสุดไม่เกินค่าของเลขจำนวนเต็ม (Integer) โดยมีการเพิ่ม (eq2 ในรูปที่ 4.4.1) และลดดรรชนี (eq3 ในรูปที่ 4.4.1) (ดู ADJ Diagram ในรูปที่ 3.1 ประกอบ)

```
eq val(init-index) = 0 .                                  ...(eq1)
eq val(inc X) = (val(X) + 1 ) .                           ...(eq2)
ceq val(dec X) = (val(X) - 1 ) if val(X) =/= 0 .          ...(eq3)
```

รูปที่ 4.1 สมการในมอดูล INDEX

4.1.2 มอดูล MESSAGE

ทำหน้าที่เก็บข้อมูลและตรวจสอบความยาวของข้อมูลในแต่ละเซกเมนต์ไม่ให้เกินขนาดเซกเมนต์ที่สามารถบรรจุได้คือ 24 บิต (ดังสมการ eq1-3 ในรูปที่ 4.2)

```
eq opt(maximumSegmentSize) = 429497296 .      -- 65536 * 65536 = Data 24 bit : Segment      ...(eq1)
ceq len (l) = l
        if l < opt(maximumSegmentSize) + 1 .   -- Message Length not over 1024              ...(eq2)
ceq len (l) =  ErrorLen
        if l > opt(maximumSegmentSize) .       -- Message Length errror if over 1024        ...(eq3)
```

รูปที่ 4.2 สมการในมอดูล MESSAGE

4.1.3 มอดูล SEGMENT

ทำหน้าที่เก็บรายละเอียดข้อมูลทั้งหมดของเซกเมนต์ (ตามรูปที่ 2.4 และ 3.6) ของทีซีพีที่พิจารณา กำหนดค่าแรกเริ่มของเซกเมนต์ (รูปที่ 4.3) เพื่อใช้เป็นค่าโดยปริยายของเซกเมนต์

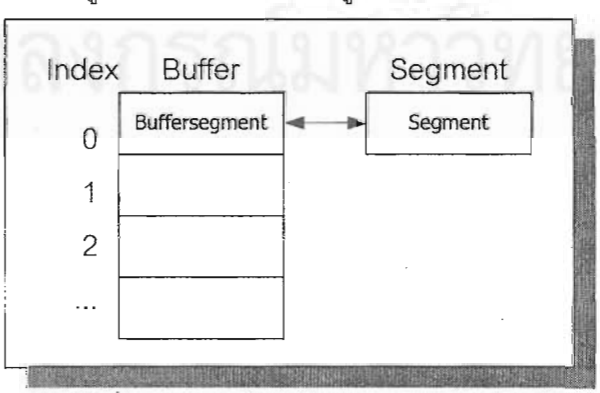
4.1.4 มอดูล BUFFER

ทำหน้าที่เป็นบัฟเฟอร์เก็บข้อมูล ซึ่งมองการเก็บข้อมูลของบัฟเฟอร์เช่นเดียวกับแถวลำดับของเซกเมนต์ (Array of Segment) (ดูรูปที่ 4.4) ใช้ดรรชนี (ในหัวข้อ 4.1.2.1) Index ในการอ้างถึงระเบียนหนึ่งๆ ของบัฟเฟอร์ดังกล่าวนี้ ซึ่งระเบียนบัฟเฟอร์นี้คือ "บัฟเฟอร์เซก-เมนต์" (Buffersegment) โดยที่บัฟเฟอร์เซกเมนต์มีส่วนประกอบภายในเหมือนกับเซกเมนต์

หน้าที่การทำงานของมอดูลบัฟเฟอร์ คือ กำหนดค่าแรกเริ่มของบัฟเฟอร์-เซกเมนต์ (eq1 ในรูปที่ 4.5) เพื่อใช้เป็นค่าโดยปริยายของบัฟเฟอร์เซกเมนต์ การใส่ข้อมูลเข้าไปยังบัฟเฟอร์ (addDataToBuffersegment ในรูปที่ 4.5) การนำข้อมูลออกมาจากบัฟเฟอร์ (retrieveDataInBufferSegment ในรูปที่ 4.5) การลบบัฟเฟอร์เซกเมนต์ (delBufferSegment ในรูปที่ 4.5) การเพิ่มบัฟเฟอร์เซกเมนต์ (addSegmentToBuffer ในรูปที่ 4.5) และการนำข้อมูลจากบัฟเฟอร์เซกเมนต์ใส่ไปยังเซกเมนต์ (retrieveSegmentInBuffer ในรูปที่ 4.5)

```
eq dataBegin = 0 .                          -- for "offset" bit
eq reservedValue = 0 .                      -- for "reserved" bit
eq noWindow = 0 .                           -- for "window" bit
eq setWindow = 1 .                          -- for initial "window" bit
eq correctChecksum = 0 .                    -- for "checksum" bit
eq errorChecksum = 1 .                      -- for "checksum" bit
eq opt(endOfOption) = 0 .                   -- for "options" bit
eq opt(noOperation) = 1 .                   -- for "options" bit
eq opt(maximumSegmentSize) = 2 .            -- for "options" bit
eq paddingValue = 0 .                       -- for "padding" bit
eq initIss = 0 .
eq initIrs = 0 .
eq init-segment = Segment { sourceport = initSouceport,
                            destport = initDestport,
                            seqnumber = initIss,
                            acknumber = initIrs,
                            offset = dataBegin,
                            reserved = reservedValue,
                            urg = resetUrg,
                            ack = resetAck,
                            psh = setPsh,
                            rst = resetRst,
                            syn = setSyn,
                            fin = resetFin,
                            window = setWindow,
                            checksum = correctChecksum,
                            urgentptr = noUrgentptr,
                            options = opt(maximumSegmentSize),
                            padding = paddingValue,
                            data = noMess } .
```

รูปที่ 4.3 สมการในมอดูล SEGMENT



รูปที่ 4.4 บัฟเฟอร์เซกเมนต์และเซกเมนต์

```
eq init-buffersegment = Buffersegment { sourceport = initSouceport,
                                        destport = initDestport,
                                        seqnumber = initIss,
                                        acknumber = initIrs,
                                        offset = dataBegin,
                                        reserved = reservedValue,
                                        urg = resetUrg,
                                        ack = resetAck,
                                        psh = setPsh,
                                        rst = resetRst,
                                        syn = setSyn,
                                        fin = resetFin,
                                        window = setWindow,
                                        checksum = correctChecksum,
                                        urgentptr = noUrgentptr,
                                        options = opt(maximumSegmentSize),
                                        padding = paddingValue,
                                        data = noMess } .                    ...(eq1)
ceq retrieveSegmentInBuffer(X,B) = no-segment if B == no-buffersegment .     ...(eq2)
ceq addSegmentToBuffer(X,S) = B if val(X) =/= -1 .                           ...(eq3)
ceq addSegmentToBuffer(X,S) = no-buffersegment if val(X) == -1 .             ...(eq4)
ceq addSegmentToBuffer(X,S) = B if retrieveSegmentInBuffer(X,B) == no-segment .  ...(eq5)
ceq retrieveDataInBufferSegment( X, addSegmentToBuffer(X',S) ) = data(S) if X == X' .  ...(eq6)
eq retrieveDataInBufferSegment(X, B) = data(B) .                             ...(eq7)
ceq addDataToBufferSegment(X,B,B') = B' if retrieveSegmentInBuffer(X,B) == no-segment . ...(eq8)
eq delBufferSegment(X,B) = addSegmentToBuffer(X,no-segment) .                ...(eq9)
ceq delBufferSegment(X,B) = no-buffersegment if retrieveSegmentInBuffer(X,B) == S .  ...(eq10)
ceq retrieveSegmentInBuffer(X,addSegmentToBuffer(X',S)) = S if X == X' .     ...(eq11)
ceq delBufferSegment(X,B) = addSegmentToBuffer(X,no-segment) if retrieveSegmentInBuffer(X,B) == S .
                                                                             ...(eq12)
```

รูปที่ 4.5 สมการในมอดูล BUFFER

4.1.5 มอดูล STATE

ทำหน้าที่เก็บสถานะเอาไว้ซึ่งทีซีพีมีการเปลี่ยนสถานะจากสถานะหนึ่งไปยังอีกสถานะหนึ่ง ข้อมูลในส่วนนี้ถูกเก็บไว้ในมอดูลดังกล่าว เพื่อนำไปวิเคราะห์ร่วมกับแผนภาพการเปลี่ยนสถานะ (ดูรูปที่ 2.10 และสมการในรูปที่ 4.6)

ceq changeState(T,T') = true if (T == closedState and (T' == listenState or T' == synsentState)) . ...(eq1)

ceq changeState(T,T') = false if (T == closedState and (T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T' == lastackState)) . ...(eq2)

ceq changeState(T,T') = true if (T == listenState and (T' == synsentState or T' == synrcvdState or T' == closedState)) . ...(eq3)

ceq changeState(T,T') = false if (T == listenState and (T' == listenState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T' == lastackState)) . ...(eq4)

ceq changeState(T,T') = true if (T == synsentState and (T' == closedState or T' == synrcvdState or T' == establishedState)) . ...(eq5)

ceq changeState(T,T') = false if (T == synsentState and (T' == listenState or T' == synsentState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T' == lastackState)) . ...(eq6)

ceq changeState(T,T') = true if (T == synrcvdState and (T' == establishedState or T' == finwait-1State or T' == closewaitState)) . ...(eq7)

ceq changeState(T,T') = false if (T == synrcvdState and (T' == closedState or T' == listenState or T' == synsentState or T' == synrcvdState or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == lastackState)) . ...(eq8)

ceq changeState(T,T') = true if (T == establishedState and (T' == finwait-1State or T' == closewaitState or T' == establishedState)) . ...(eq9)

ceq changeState(T,T') = false if (T == establishedState and (T' == closedState or T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == lastackState)) . ...(eq10)

ceq changeState(T,T') = true if (T == finwait-1State and (T' == closingState or T' == finwait-2State)) . ...(eq11)

ceq changeState(T,T') = false if (T == finwait-1State and (T' == closedState or T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == timewaitState or T' == closewaitState or T' == lastackState)) . ...(eq12)

ceq changeState(T,T') = true if (T == closingState and T' == timewaitState) . ...(eq13)

ceq changeState(T,T') = false if (T == closingState and (T' == closedState or T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == closewaitState or T' == lastackState)) . ...(eq14)

ceq changeState(T,T') = true if (T == finwait-2State and T' == timewaitState) . ...(eq15)

ceq changeState(T,T') = false if (T == finwait-2State and (T' == closedState or T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == closewaitState or T' == lastackState)) . ...(eq16)

ceq changeState(T,T') = true if (T == timewaitState and T' == closedState) . ...(eq17)

ceq changeState(T,T') = false if (T == timewaitState and (T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T' == lastackState)) . ...(eq18)

ceq changeState(T,T') = true if (T == closewaitState and T' == lastackState) . ...(eq19)

ceq changeState(T,T') = false if (T == closewaitState and (T' == closedState or T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState)) . ...(eq20)

ceq changeState(T,T') = true if (T == lastackState and T' == closedState) . ...(eq21)

ceq changeState(T,T') = false if (T == lastackState and (T' == listenState or T' == synsentState or T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T' == lastackState)) . ...(eq22)

รูปที่ 4.6 สมการในมอดูล STATE

4.1.6 มอดูล TCB

ทำหน้าที่เก็บระเบียนควบคุมการขนส่งข้อมูล (ตามรูปที่ 2.9 และ 3.7) โดยมีการกำหนดค่าแรกเริ่มของระเบียนควบคุมการขนส่งข้อมูล (eq1 ในรูปที่ 4.7) เพื่อใช้เป็นค่าโดยปริยายของระเบียนควบคุมการขนส่งข้อมูล อีกทั้งการสร้าง (eq1 ในรูปที่ 4.7) ลบ (eq2 ในรูป-ที่ 4.7) และตรวจสอบระเบียนควบคุมการขนส่ง (op checkTcb) ซึ่งจะตรวจสอบว่ามีความถูกต้องหรือไม่ แล้วส่งค่าการตรวจสอบนี้กลับออกมาเป็นจริงหรือเท็จ (eq3 และ eq4 ในรูปที่ 4.7)

```
eq precedenceStatusValue = 0 .
eq snd-una(C) = sndunaValue .
eq snd-nxt(C) = sndnxtValue .
eq snd-wnd(C) = sndwndValue .
eq snd-up(C) = noUrgentptr .
eq snd-wl1(C) = sndwl1Value .
eq snd-wl2(C) = sndwl2Value .
eq rcv-nxt(C) = val(Id) + 1 .
eq rcv-wnd(C) = val(Id) .
eq rcv-up(C) = noUrgentptr .
eq seg-seg(C) = seqnumber(S) .
eq seg-ack(C) = acknumber(S) .
eq seg-len(C) = opt(maximumSegmentSize) .
eq seg-wnd(C) = window(S) .
eq seg-up(C) = noUrgentptr .
eq seg-prc(C) = precedenceStatusValue .
eq init-tcb = Tcb {  snd-una = initIss, snd-nxt = initIss + 1, snd-wnd = sndwndValue,
snd-up = noUrgentptr,snd-wl1 = sndwl1Value, snd-wl2 = sndwl2Value, iss = initIss, rcv-nxt = val(init-index) + 1,
rcv-wnd = val(init-index), rcv-up = noUrgentptr, irs = initIrs, seg-seg = initIss, seg-ack = initIrs, seg-len =
opt(maximumSegmentSize), seg-wnd = window(init-segment), seg-up = noUrgentptr, seg-prc =
precedenceStatusValue } .
eq createTcb = init-tcb .                                          ...(eq1)
eq deleteTcb(C) = no-tcb .                                         ...(eq2)
ceq checkTcb(C) = false if iss(C) == -1 .                          ...(eq3)
ceq checkTcb(C) = true if iss(C) =/= -1 .                          ...(eq4)
```

รูปที่ 4.7 สมการในมอดูล TCB

4.1.7 มอดูล STATUS

ทำหน้าที่เก็บรายละเอียดสถานภาพต่างๆ ที่ใช้ในการติดต่อทีซีพี (eq1-eq13 ในรูปที่ 4.8) โดยกำหนดค่าแรกเริ่มของระเบียนควบคุมการขนส่งข้อมูล (eq14 ในรูปที่ 4.8) เพื่อใช้เป็นค่าโดยปริยายของระเบียนควบคุมการขนส่งข้อมูล

```
eq bind(I) = I .                                                    ...(eq1)
eq local-socket(U) = localSocket(L) .                               ...(eq2)
eq foreign-socket(U) = foreignSocket(L) .                           ...(eq3)
eq rcv-wnd(U) = rcv-wnd(C) .                                        ...(eq4)
eq snd-wnd(U) = snd-wnd(C) .                                        ...(eq5)
eq connection-state(U) = valueState(curState) .                     ...(eq6)
eq precedence-status(U) = seg-prc(C) .                              ...(eq7)
eq security-or-compartment(U) = securityOrCompartmentValue .        ...(eq8)
eq urg-state(U) = valueState(noUrgentState) .                       ...(eq9)
eq valueState(noUrgentState) = 0 .                                  ...(eq10)
eq securityOrCompartmentValue = 1 .                                 ...(eq11)
eq transTimeoutValue = noTimeout .                                  ...(eq12)
eq trans-timeout(U) = transTimeoutValue .                           ...(eq13)
eq init-status = Status {       local-socket = initLocalSocket,
                                foreign-socket = initForeignSocket,
                                rcv-wnd = rcv-wnd(init-tcb),
                                snd-wnd = snd-wnd(init-tcb),
                                connection-state = valueState(closedState),
                                num-buf-wait-ack = initNumBufWaitAck,
                                num-buf-pend-receipt = initNumBufPendReceipt,
                                urg-state = valueState(noUrgentState),
                                precedence-status = precedenceStatusValue,
                                security-or-compartment = securityOrCompartmentValue,
                                trans-timeout = transTimeoutValue } .   ...(eq14)
```

รูปที่ 4.8 สมการในมอดูล STATUS

4.1.8 มอดูล STATUSCALL

รายละเอียดแสดงในหัวข้อ 3.2.1.6

4.1.7 มอดูล STATUS

ทำหน้าที่เก็บรายละเอียดสถานภาพต่างๆ ที่ใช้ในการติดต่อทีซีพี (eq1-eq13 ในรูปที่ 4.8) โดยกำหนดค่าแรกเริ่มของระเบียนควบคุมการขนส่งข้อมูล (eq14 ในรูปที่ 4.8) เพื่อใช้เป็นค่าโดยปริยายของระเบียนควบคุมการขนส่งข้อมูล

```
eq bind(I) = I .                                                    ...(eq1)
eq local-socket(U) = localSocket(L) .                               ...(eq2)
eq foreign-socket(U) = foreignSocket(L) .                           ...(eq3)
eq rcv-wnd(U) = rcv-wnd(C) .                                        ...(eq4)
eq snd-wnd(U) = snd-wnd(C) .                                        ...(eq5)
eq connection-state(U) = valueState(curState) .                     ...(eq6)
eq precedence-status(U) = seg-prc(C) .                              ...(eq7)
eq security-or-compartment(U) = securityOrCompartmentValue .        ...(eq8)
eq urg-state(U) = valueState(noUrgentState) .                       ...(eq9)
eq valueState(noUrgentState) = 0 .                                  ...(eq10)
eq securityOrCompartmentValue = 1 .                                 ...(eq11)
eq transTimeoutValue = noTimeout .                                  ...(eq12)
eq trans-timeout(U) = transTimeoutValue .                           ...(eq13)
eq init-status = Status {       local-socket = initLocalSocket,
                                foreign-socket = initForeignSocket,
                                rcv-wnd = rcv-wnd(init-tcb),
                                snd-wnd = snd-wnd(init-tcb),
                                connection-state = valueState(closedState),
                                num-buf-wait-ack = initNumBufWaitAck,
                                num-buf-pend-receipt = initNumBufPendReceipt,
                                urg-state = valueState(noUrgentState),
                                precedence-status = precedenceStatusValue,
                                security-or-compartment = securityOrCompartmentValue,
                                trans-timeout = transTimeoutValue } .   ...(eq14)
```

รูปที่ 4.8 สมการในมอดูล STATUS

4.1.8 มอดูล STATUSCALL

รายละเอียดแสดงในหัวข้อ 3.2.1.6

4.1.9 มอดูล SEND

ภาพโดยรวมของการรับส่งข้อมูลระหว่างทีซีพีคู่หนึ่งๆ แทนการทำงานด้วยการดำเนินการ send ในมอดูล SEND ซึ่งการดำเนินการ send นี้ ครอบคลุมและเกี่ยวข้องกับการทำงานของ Segment State Tcb Status อีกต่อหนึ่ง

มอดูล SEND ทำหน้าที่ส่งข้อมูลเซกเมนต์จากทีซีพีหนึ่งไปยังอีกทีซีพีหนึ่ง (ดูรูปที่ 2.8 Segment (บน) -> Segment (ล่าง)) โดยพิจารณาถึงสถานะ ระเบียนควบคุมการขนส่งข้อมูลและระเบียนสถานภาพต่างๆ ร่วมด้วย (การดำเนินการ send ในรูปที่ 2.8 สมการในรูปที่ 4.9)

4.1.10 มอดูล OPENCALL

มอดูล OPENCALL จะทำการเปิดการติดต่อแล้วส่งผลออกมาเป็นชื่อในการเชื่อมต่อเฉพาะที่ดังรายละเอียดเอียดแสดงในหัวข้อ 3.3.1.1

4.1.11 มอดูล SENDCALL

มอดูล SENDCALL จะทำการส่งข้อมูลผ่านไปยังเส้นทางตามชื่อในการเชื่อมต่อเฉพาะที่ดังรายละเอียดแสดงในหัวข้อ 3.3.1.2

4.1.12 มอดูล RECEIVECALL

มอดูล RECEIVECALL จะทำการรับข้อมูลตามเส้นทางที่ระบุไว้ในชื่อในการเชื่อมต่อเฉพาะที่ ดังรายละเอียดแสดงในหัวข้อ 3.3.1.3

4.1.13 มอดูล CLOSECALL

มอดูล CLOSECALL จะทำการปิดการติดต่อตามชื่อในการเชื่อมต่อเฉพาะที่ ดังรายละเอียดแสดงในหัวข้อ ที่ 3.3.1.4

4.1.14 มอดูล ABORTCALL

มอดูล ABORTCALL จะทำการยกเลิกการติดต่อตามชื่อในการเชื่อมต่อเฉพาะที่ดังรายละเอียดแสดงในหัวข้อ 3.3.1.5

```
ceq send(S,B1,B2,B3) = no-segment
        if window(S) == noWindow and B1 == true and B2 == true and B3 == true .
ceq send(S,B1,B2,B3) = no-segment
        if psh(S) == resetPsh and B1 == true and B2 == true and B3 == true .
ceq send(S,B1,B2,B3) = no-segment
        if B1 == false or B2 == false and B3 == false .
-- send Ack, Fin : seqnumber = acknumber, acknumber = seqnumber + 1
ceq send(S,B1,B2,B3) =
Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = acknumber(S),
acknumber = seqnumber(S) + 1, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = resetAck, psh = setPsh,
rst = resetRst, syn = setSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr = noUrgentptr,
options = opt(maximumSegmentSize),          padding = paddingValue, data = data(S) }
        if window(S) =/= noWindow and psh(S) =/= resetPsh and ack(S) == setAck and fin(S) == setFin
        and B1 == true and B2 == true and B3 == true .
-- send Syn, Ack : acknumber = seqnumber + 1, seqnumber = acknumber
ceq send(S,B1,B2,B3) =
Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = acknumber(S),
acknumber = seqnumber(S) + 1 , offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = setAck,          psh =
setPsh, rst = resetRst, syn = setSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr =
noUrgentptr, options = opt(maximumSegmentSize),          padding = paddingValue, data = data(S) }
        if window(S) =/= noWindow and psh(S) =/= resetPsh and syn(S) == setSyn and ack(S) == setAck
        and B1 == true and B2 == true and B3 == true .
-- send Syn : acknumber = seqnumber + 1
ceq send(S,B1,B2,B3) =
Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = initIss,
acknumber = seqnumber(S) + 1, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = resetAck, psh = setPsh,
rst = resetRst, syn = setSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr = noUrgentptr,
options = opt(maximumSegmentSize),          padding = paddingValue, data = data(S) }
        if window(S) =/= noWindow and psh(S) =/= resetPsh and syn(S) == setSyn
        and B1 == true and B2 == true and B3 == true .
-- send Ack : seqnumber = acknumber
ceq send(S,B1,B2,B3) =
Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = acknumber(S),
acknumber = initIrs, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = setAck,
psh = setPsh, rst = resetRst, syn = resetSyn, fin = resetFin, window = setWindow, checksum = correctChecksum,  urgentptr =
noUrgentptr, options = opt(maximumSegmentSize),          padding = paddingValue, data = data(S) }
        if window(S) =/= noWindow and psh(S) =/= resetPsh and ack(S) == setAck
        and B1 == true and B2 == true and B3 == true .
-- send Rst : seqnumber = iss(C)
ceq send(S,B1,B2,B3) =
Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = iss(C),
acknumber = initIrs, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = resetAck,
psh = setPsh, rst = setRst, syn = resetSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr =
noUrgentptr, options = opt(maximumSegmentSize),          padding = paddingValue, data = data(S) }
        if window(S) =/= noWindow and psh(S) =/= resetPsh and rst(S) == setRst
        and B1 == true and B2 == true and B3 == true .
ceq send(S,B1,B2,B3) = S if B1 == true and B2 == true and B3 == true .
```

รูปที่ 4.9 สมการในมอดูล SEND

4.2 ชั้นทีซีพี

4.2.1 มอดูล TCP-SYSTEM

ทำหน้าที่เชื่อมการทำงานจากชั้นการประยุกต์ (Application Layer) ไปยังชั้นการขนส่ง (The Transport Layer) ซึ่งก็คือ การเชื่อมการทำงานจากผู้ใช้งานไปยังทีซีพี โดย ควบคุมการทำงานของสถานะ (eq1 ในรูปที่ 4.10) เซกเมนต์ (eq2-eq4 ในรูปที่ 4.10) บัฟเฟอร์ (eq5-eq7 ในรูปที่ 4.10) เลขที่อยู่ไอพี (eq8-eq9 ในรูปที่ 4.10)

```
-- State
        ceq state-sys(T) = curState if T == no-system .                    ...(eq1)
-- Segment
        ceq seg-sys(I,T) = no-segment if T == no-system .                  ...(eq2)
        ceq seg-sys(I,T) = Segment { sourceport = get-ip1(T), destport = get-ip2(T) }
                if T =/= init-system .                                     ...(eq3)
        ceq seg-sys(I,T) = init-segment
                if I == init-index and T == init-system .                  ...(eq4)
-- Buffersegment
        ceq buf-sys(I,T) = no-buffersegment if T == no-system .            ...(eq5)
        ceq buf-sys(I,T) = Buffersegment { sourceport = get-ip1(T), destport = get-ip2(T) }
                if T =/= init-system .                                     ...(eq6)
        ceq buf-sys(I,T) = init-buffersegment
                if I == init-index and T == init-system .                  ...(eq7)
-- IP
        eq get-ip1(sys(IPs,IPc)) = IPs .                                   ...(eq8)
        eq get-ip2(sys(IPs,IPc)) = IPc .                                   ...(eq9)
```

รูปที่ 4.10 สมการส่วนที่ 1 ในมอดูล TCP-SYSTEM

ทั้งนี้มอดูล TCP-SYSTEM มีการดำเนินการเรียกใช้งานคำสั่งเรียกต่างๆ จากผู้ใช้งานชั้นการประยุกต์ ดังต่อไปนี้ การเรียกใช้คำสั่งเรียกเปิดแบบ active (eq3-eq6 ในรูปที่ 4.11) การเรียกใช้คำสั่งเรียกเปิดแบบ passive (eq7-eq10 ในรูปที่ 4.11) หรือแม้แต่การตรวจสอบชื่อในการเชื่อมต่อเฉพาะที่ (eq1-eq2 ในรูปที่ 4.11) การเรียกใช้คำสั่งเรียกส่ง (รูปที่ 4.12) การเรียกใช้คำสั่งเรียกรับ (รูปที่ 4.13) การเรียกใช้คำสั่งเรียกส่งและรับ (รูปที่ 4.14) หรือแม้กระทั่งการเรียกใช้คำสั่งเรียกปิด (eq1-eq4 ในรูปที่ 4.15) หรือ ยกเลิก (eq5-eq8 ในรูปที่ 4.15)

```
-- Check Open
        ceq checkName(L) = openFail if L == errorLocalConnectionName .                    ...(eq1)
        ceq checkName(L) = openOk if L =/= errorLocalConnectionName .                     ...(eq2)
-- Open Call : Active
        ceq open-active-sys(T) = checkName(openCall(get-ip1(T), get-ip2(T), active, noTimeout,
precedenceStatusValue, securityOrCompartmentValue, opt(maximumSegmentSize) ) )
                if ( get-ip1(T) > 0 ) and ( get-ip2(T) > 0 ) and (get-ip1(T) =/= get-ip2(T) ) .-- no error IP   ...(eq3)
        ceq open-active-sys(T) = openFail
                if ( get-ip1(T) < 1 ) or ( get-ip2(T) < 1 ) or (get-ip1(T) == get-ip2(T) ) .    -- error IP      ...(eq4)
        ceq state-sys(T) = synsentState
                if open-active-sys(T) == openOk .                                         ...(eq5)
        ceq state-sys(T) = closedState
                if open-active-sys(T) == openFail .                                       ...(eq6)
-- Open Call : Passive
        ceq open-passive-sys(T) = checkName(openCall(get-ip1(T), get-ip2(T), passive, noTimeout,
precedenceStatusValue, securityOrCompartmentValue, opt(maximumSegmentSize) ) )
                if ( get-ip1(T) > 0 ) and ( get-ip2(T) > 0 ) and (get-ip1(T) =/= get-ip2(T) ) . -- no error IP   ...(eq7)
        ceq open-passive-sys(T) = openFail
                if ( get-ip1(T) < 1 ) or ( get-ip2(T) < 1 ) or (get-ip1(T) == get-ip2(T) ) .    -- error IP      ...(eq8)
        ceq state-sys(T) = listenState
                if open-passive-sys(T) == openOk .                                        ...(eq9)
        ceq state-sys(T) = closedState
                if open-passive-sys(T) == openFail .                                      ...(eq10)
```

รูปที่ 4.11 สมการส่วนที่ 2 ในมอดูล TCP-SYSTEM

```
-- Send Call
                ceq send-sys(T,M) = errorIp
                        if open-active-sys(T) == openFail or
                        get-ip1(T) < 1 or
                        get-ip2(T) < 1 or
                        ( get-ip1(T) == get-ip2(T) ) .                          ...(eq1)
                ceq send-sys(T,M) = noMessage
                        if M == noMessage .                                     ...(eq2)
                ceq send-sys(T,M) = Message {
        mess1 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M,
1, setPsh, noUrg, noTimeout) ),
        mess2 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M,
2, setPsh, noUrg, noTimeout) ),
        mess3 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M,
3, setPsh, noUrg, noTimeout) ),
        mess4 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M,
4, setPsh, noUrg, noTimeout) ),
        mess5 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M,
5, setPsh, noUrg, noTimeout) ) }
        if open-active-sys(T) == openOk and M =/= noMessage and ( get-ip1(T) > 0 ) and ( get-ip2(T) > 0 ). ...(eq3)
```

รูปที่ 4.12 สมการส่วนที่ 3 ในมอดูล TCP-SYSTEM

```
-- Receive Call
        ceq receive-sys(T,M) = noMessage
            if M == noMessage .                                          ...(eq1)
        ceq receive-sys(T,M) = errorIp
            if open-passive-sys(T) == openFail or
            get-ip1(T) < 1 or
            get-ip2(T) < 1 or
            get-ip1(T) == get-ip2(T) or
            M == errorIp .                                               ...(eq2)
        ceq receive-sys(T,M) = Message {
    mess1 = retrieveDataInBufferSegment(init-index, receiveCall(LocalConnectionName{ localSocket = get-
ip1(T), foreignSocket = get-ip2(T) }, init-index, 1, setPsh,noUrg, sendCall(LocalConnectionName { localSocket = get-
ip2(T), foreignSocket = get-ip1(T) },M, 1, setPsh, noUrg, noTimeout) ) ) ,
    mess2 = retrieveDataInBufferSegment(inc init-index, receiveCall(LocalConnectionName{ localSocket =
get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 2, setPsh,noUrg, sendCall(LocalConnectionName { localSocket =
get-ip2(T), foreignSocket = get-ip1(T) },M, 2, setPsh, noUrg, noTimeout) ) ) ,
    mess3 = retrieveDataInBufferSegment(inc inc init-index, receiveCall(LocalConnectionName{ localSocket =
get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 3, setPsh,noUrg, sendCall(LocalConnectionName { localSocket =
get-ip2(T), foreignSocket = get-ip1(T) },M, 3, setPsh, noUrg, noTimeout) ) ) ,
    mess4 = retrieveDataInBufferSegment(inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 4, setPsh,noUrg, sendCall(LocalConnectionName {
localSocket = get-ip2(T), foreignSocket = get-ip1(T) },M, 4, setPsh, noUrg, noTimeout) ) ) ,
    mess5 = retrieveDataInBufferSegment(inc inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 5, setPsh,noUrg, sendCall(LocalConnectionName {
localSocket = get-ip2(T), foreignSocket = get-ip1(T) },M, 5, setPsh, noUrg, noTimeout) ) ) }
            if open-passive-sys(T) == openOk and
            M =/= noMessage and
            ( get-ip1(T) > 0 ) and
            ( get-ip2(T) > 0 ) and
            get-ip1(T) =/= get-ip2(T) .                                  ...(eq3)
```

รูปที่ 4.13 สมการส่วนที่ 4 ในมอดูล TCP-SYSTEM

```
-- Send Call and Receive Call
        ceq receive-sys(T',send-sys(T,M)) = noMessage
            if M == noMessage .                                              ...(eq1)
        ceq receive-sys(T',send-sys(T,M)) = errorIp
            if get-ip1(T) < 1 or
            get-ip2(T) < 1 or
            get-ip1(T') < 1 or
            get-ip2(T') < 1 or
            get-ip1(T) == get-ip2(T) or
            get-ip1(T') == get-ip2(T') or
            get-ip1(T) =/= get-ip2(T') or
            get-ip1(T') =/= get-ip2(T) .                                     ...(eq2)
        ceq receive-sys(T',send-sys(T,M)) = Message {
    mess1 = retrieveDataInBufferSegment(init-index, receiveCall(LocalConnectionName{ localSocket = get-
ip1(T'), foreignSocket = get-ip2(T') }, init-index, 1, setPsh,noUrg, sendCall(LocalConnectionName { localSocket = get-
ip1(T), foreignSocket = get-ip2(T) },M, 1, setPsh, noUrg, noTimeout) ) ) ,
    mess2 = retrieveDataInBufferSegment(inc init-index, receiveCall(LocalConnectionName{ localSocket =
get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 2, setPsh,noUrg, sendCall(LocalConnectionName { localSocket =
get-ip1(T), foreignSocket = get-ip2(T) },M, 2, setPsh, noUrg, noTimeout) ) ) ,
    mess3 = retrieveDataInBufferSegment(inc inc init-index, receiveCall(LocalConnectionName{ localSocket =
get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 3, setPsh,noUrg, sendCall(LocalConnectionName { localSocket =
get-ip1(T), foreignSocket = get-ip2(T) },M, 3, setPsh, noUrg, noTimeout) ) ) ,
    mess4 = retrieveDataInBufferSegment(inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 4, setPsh,noUrg, sendCall(LocalConnectionName {
localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 4, setPsh, noUrg, noTimeout) ) ) ,
    mess5 = retrieveDataInBufferSegment(inc inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 5, setPsh,noUrg, sendCall(LocalConnectionName {
localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 5, setPsh, noUrg, noTimeout) ) ) }
            if M =/= noMessage and
            M =/= errorIp and
            open-active-sys(T) == openOk and
            open-passive-sys(T') == openOk and
            get-ip1(T) == get-ip2(T') and
            get-ip2(T) == get-ip1(T') and
            get-ip1(T) =/= get-ip2(T) and
            get-ip1(T') =/= get-ip2(T') .        -- correct IP and pair of port    ...(eq3)
```

รูปที่ 4.14 สมการส่วนที่ 5 ในมอดูล TCP-SYSTEM

```
-- Close Call
        ceq close-sys(T) = false
                        if T == no-system or
                        get-ip1(T) < 1 or
                        get-ip2(T) < 1 or
                        get-ip1(T) == get-ip2(T) .                              ...(eq1)
        ceq close-sys(T) = true
                        if T =/= no-system and
                        get-ip1(T) > 0 and
                        get-ip2(T) > 0 and
                        get-ip1(T) =/= get-ip2(T) .                             ...(eq2)
        ceq closeCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } ) = closeOk
                        if close-sys(T) == true .                               ...(eq3)
        ceq closeCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } ) = closeFail
                        if close-sys(T) == false .                              ...(eq4)
-- Abort Call
        ceq abort-sys(T) = false
                        if T == no-system or
                        get-ip1(T) < 1 or
                        get-ip2(T) < 1 or
                        get-ip1(T) == get-ip2(T) .                              ...(eq5)
        ceq abort-sys(T) = true
                        if T =/= no-system and
                        get-ip1(T) > 0 and
                        get-ip2(T) > 0 and
                        get-ip1(T) =/= get-ip2(T) .                             ...(eq6)
        ceq abortCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } ) = abortOk
                        if abort-sys(T) == true .                               ...(eq7)
        ceq abortCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } ) = abortFail
                        if abort-sys(T) == false .                              ...(eq8)
```

รูปที่ 4.15 สมการส่วนที่ 6 ในมอดูล TCP-SYSTEM

บทที่ 5

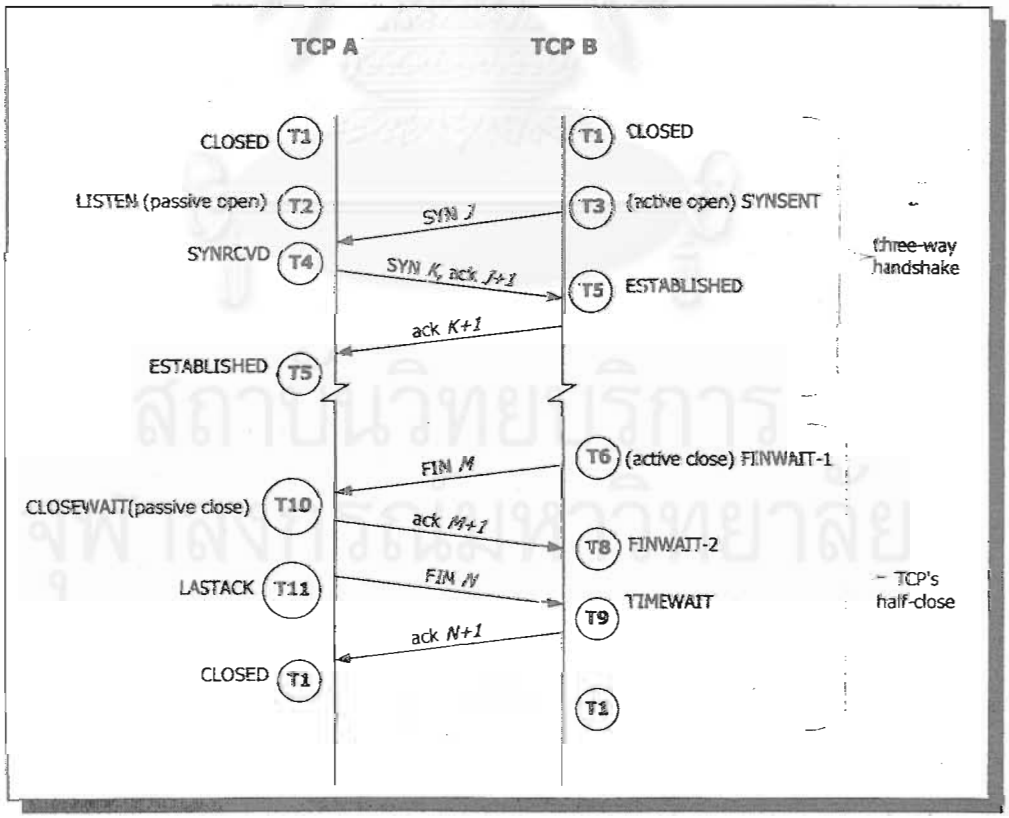
กรณีศึกษา

ข้อกำหนดโพรโทคอลทีซีพีนี้แบ่งกรณีศึกษาได้ 4 กรณี คือ การเริ่มการติดต่อ การรับส่งข้อมูล และยกเลิกการติดต่อ ซึ่งทั้งสามเหล่านี้ สามารถใช้แผนภาพเส้นกำหนดเวลาเพื่อแสดงการทำงาน และกรณีศึกษาลำดับสุดท้าย เป็นกรณีที่เรียกใช้งานจากชั้นบน (ผู้ใช้งาน) ผ่านคำสั่งเรียกต่างๆ ดังแสดงรายละเอียดของกรณีศึกษานี้ในหัวข้อ 5.6 ต่อไป

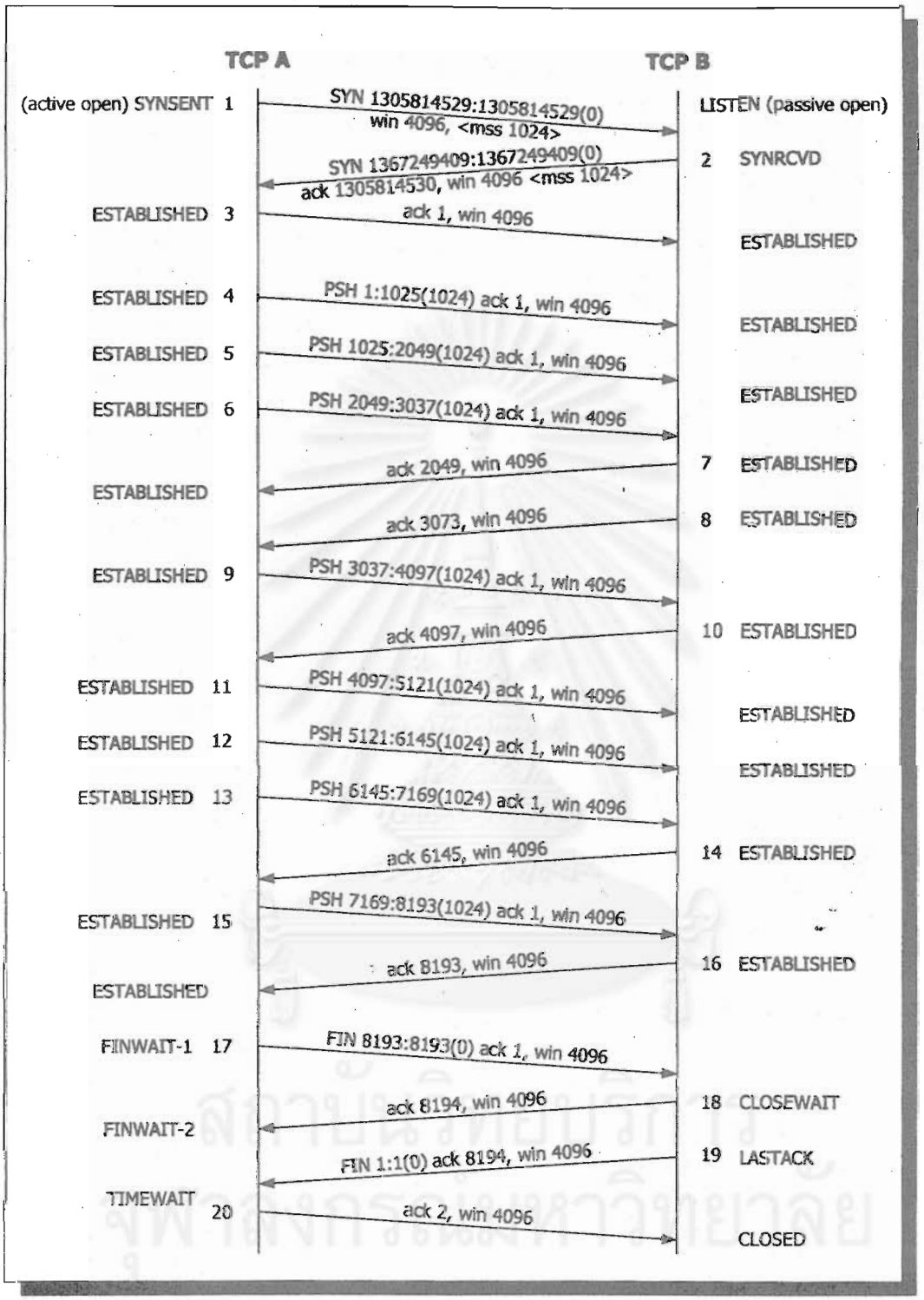
5.1 เส้นกำหนดเวลา

แผนภาพเส้นกำหนดเวลาบอกรายละเอียดข้อมูลที่รับส่งไปมาระหว่างทีซีพีคู่หนึ่งๆ ณ สถานะที่ทีซีพีนั้นๆ ทำงานอยู่ (ดังรูปที่ 5.1)

รูปที่ 5.1 แสดงการเริ่มการติดต่อและยกเลิกการติดต่อของทีซีพี และรูปที่ 5.5 แสดงการรับส่งข้อมูลจำนวน 8192 ไบต์



รูปที่ 5.1 แผนภาพเส้นกำหนดเวลาการเริ่มการติดต่อและการยกเลิกการติดต่อ



รูปที่ 5.2 แผนภาพเส้นกำหนดเวลาการส่งข้อมูลจำนวน 8192 ไบต์

5.2 กฎการเชื่อมแผนภาพเอดีเจ แผนภาพการเปลี่ยนสถานะ และแผนภาพเส้นกำหนดเวลา

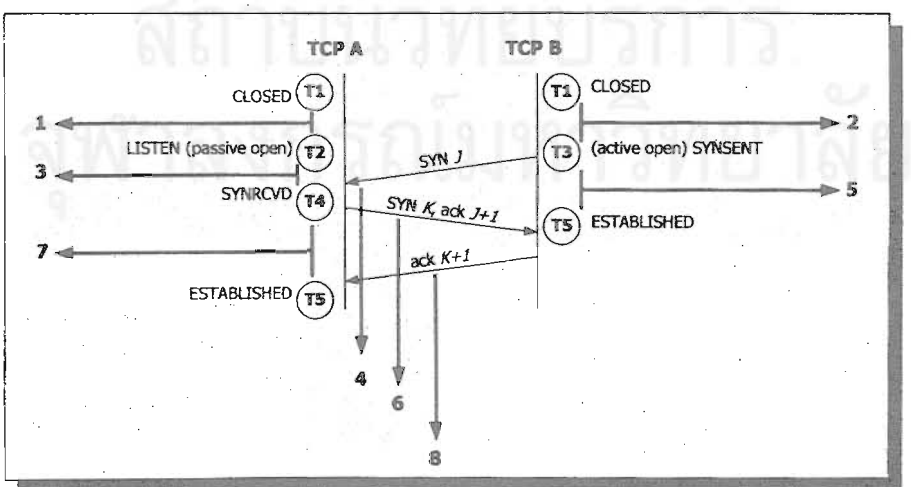
การใช้แผนภาพเส้นกำหนดเวลาในกรณีศึกษาที่จะกล่าวในหัวข้อต่อไปนั้น มีการเรียกการทำงานด้านกระบวนการและรายละเอียดข้อมูลของโพรโทคอลทีซีพีต่างๆ ดังนั้น จึงมีกฎการเชื่อมแผนภาพทั้งสามเข้าด้วยกัน ดังตารางที่ 5.1

ตารางที่ 5.1 ตารางกฎการเชื่อมแผนภาพเอดีเจ แผนภาพการเปลี่ยนสถานะ และแผนภาพเส้นกำหนดเวลา

แผนภาพเอดีเจ	แผนภาพการเปลี่ยนสถานะ	แผนภาพเส้นกำหนดเวลา
1. T และ T' ใน op ChangeState(T,T')	1. State นำเข้า (T), State ส่งออก (T')	1. State นำเข้า (T) State ส่งออก (T')
2. Segment {syn, ack, psh, fin, win, data}, changeState(T,T'), call, checkTcb(C) โดยกระทำการดำเนินการทุกอย่าง	2. พิจารณาตาม State ที่ทำงานอยู่และพิจารณา call และ TCB	2. ข้อมูล syn, ack, psh, fin, win, data บนเส้นลูกศร

5.3 ตัวอย่างการจัดรูปนัยการเริ่มติดต่อ

การเริ่มการติดต่อของทีซีพี (A และ B) มีขั้นตอนรับส่งข้อมูลไปมาระหว่างทีซีพีด้วยกัน 3 ขั้นตอน (หมายเลข4 6 และ 8 ในรูปที่ 5.3 ซึ่งเขียนได้ด้วยสมการที่ 4 6 และ 8 ในรูปที่ 5.4 ตามลำดับ) ซึ่งเรียกว่า การจับมือ 3 วิถี ทั้งนี้ก่อนส่งข้อมูลไปมาระหว่างกัน ทีซีพีมีการเปลี่ยนสถานะและกำหนดค่าแรกเริ่มก่อนการรับส่ง



รูปที่ 5.3 การจัดรูปนัยการติดต่อ

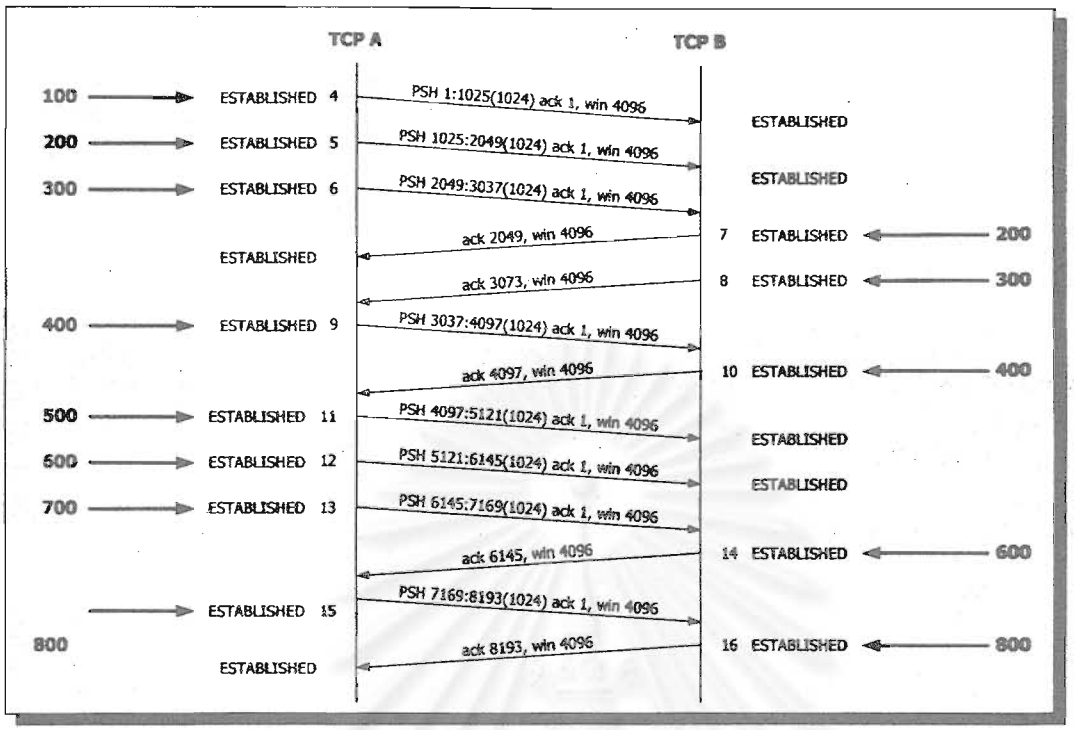
```
red send(no-segment, changeState(closedState,listenState), checkTcb(createTcb), checkStatus(init-status)) .
                                                                                    ...(eq1)
red send( no-segment, changeState(closedState,synsentState), checkTcb(createTcb), checkStatus(init-status)) .
                                                                                    ...(eq2)
red send( no-segment, changeState(listenState,synrcvdState), checkTcb(C), checkStatus(U)) .
                                                                                    ...(eq3)
red send( Segment { seqnumber = J, syn = setSyn, psh = setPsh, window = setWindow },
changeState(synsentState,synrcvdState), checkTcb(C), checkStatus(init-status)) .    ...(eq4)
red send( Segment { seqnumber = J, syn = setSyn, psh = setPsh, window = setWindow },
changeState(synsentState,establishedState), checkTcb(C), checkStatus(U)) .          ...(eq5)
red send( Segment { seqnumber = K, acknumber = J + 1, ack = setAck, syn = setSyn, psh = setPsh,
window = setWindow }, changeState(synrcvdState,establishedState), checkTcb(C), checkStatus(U)) . ...(eq6)
red send( S, changeState(synrcvdState,establishedState), checkTcb(C), checkStatus(U)) .
                                                                                    ...(eq7)
red send(Segment { seqnumber = K + 1, ack = setAck, psh = setPsh, window = setWindow },
changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)) .      ...(eq8)
```

รูปที่ 5.4 สมการการจัดรูปนัยการติดต่อ

5.4 ตัวอย่างการจัดรูปนัยการส่งข้อมูล

รูปที่ 5.2 แสดงรายละเอียดการติดต่อ (หมายเลข 1-3) การส่งข้อมูล(หมายเลข 4-16) และการยกเลิกติดต่อของทีซีพี (หมายเลข 17-20) เมื่อได้ทำการติดต่อระหว่างทีซีพีเป็นที่เรียบร้อยแล้ว มีการรับส่งข้อมูล (100-800) ในรูป 5.5 ดังรายละเอียดต่อไปนี้

- หมายเลข 4-6 (ดังสมการที่ eq1-3 ในรูปที่ 5.5) ทำการส่งข้อมูล 100 200 และ 300 ตามลำดับ
- หมายเลข 7-8 (ดังสมการที่ eq4-5 ในรูปที่ 4.5.5) ตอบรับกลับมาว่าได้รับข้อมูล100-200 และ 300 แล้ว
- หมายเลข 9 (ดังสมการที่ eq6 ในรูปที่ 5.5) ทำการส่งข้อมูล 400
- หมายเลข 10 (ดังสมการที่ eq7 ในรูปที่ 5.5) ตอบรับกลับมาว่าได้รับข้อมูล 400 แล้ว
- หมายเลข 11-13 (ดังสมการที่ eq8-10 ในรูปที่ 5.5) ทำการส่งข้อมูล 500 600 และ 700 ตามลำดับ
- หมายเลข 14 (ดังสมการที่ eq11 ในรูปที่ 5.5) ตอบรับกลับมาว่าได้รับข้อมูล 600 แล้ว
- หมายเลข 15 (ดังสมการที่ eq12 ในรูปที่ 5.5) ทำการส่งข้อมูล 800
- หมายเลข 16 (ดังสมการที่ eq13 ในรูปที่ 5.5) ตอบรับกลับมาว่าได้รับข้อมูล 800 แล้ว



รูปที่ 5.5 แผนภาพเส้นกำหนดเวลาการส่งข้อมูล

red retrieveDataInBufferSegment(init-index, addSegmentToBuffer(init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 1, ack = setAck, psh = setPsh, window = 4096, data = 100 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq1)

red retrieveDataInBufferSegment(inc init-index, addSegmentToBuffer(inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 1025, ack = setAck, psh = setPsh, window = 4096, data = 200 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq2)

red retrieveDataInBufferSegment(inc inc init-index, addSegmentToBuffer(inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 2049, ack = setAck, psh = setPsh, window = 4096, data = 300 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq3)

red retrieveDataInBufferSegment(init-index, addSegmentToBuffer(init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 2049, ack = setAck, psh = setPsh, window = 4096, data = 200 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq4)

red retrieveDataInBufferSegment(init-index, addSegmentToBuffer(inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 3073, ack = setAck, psh = setPsh, window = 4096, data = 300 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq5)

red retrieveDataInBufferSegment(inc inc inc init-index, addSegmentToBuffer(inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 3037, ack = setAck, psh = setPsh, window = 4096, data = 400 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq6)

red retrieveDataInBufferSegment(inc inc init-index, addSegmentToBuffer(inc inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 4097, ack = setAck, psh = setPsh, window = 4096, data = 400 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq7)

red retrieveDataInBufferSegment(inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 4097, ack = setAck, psh = setPsh, window = 4096, data = 500 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq8)

red retrieveDataInBufferSegment(inc inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 5121, ack = setAck, psh = setPsh, window = 4096, data = 600 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq9)

red retrieveDataInBufferSegment(inc inc inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 6145, ack = setAck, psh = setPsh, window = 4096, data = 700 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq10)

red retrieveDataInBufferSegment(inc inc inc init-index, addSegmentToBuffer(inc inc inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 6145, ack = setAck, psh = setPsh, window = 4096, data = 600 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq11)

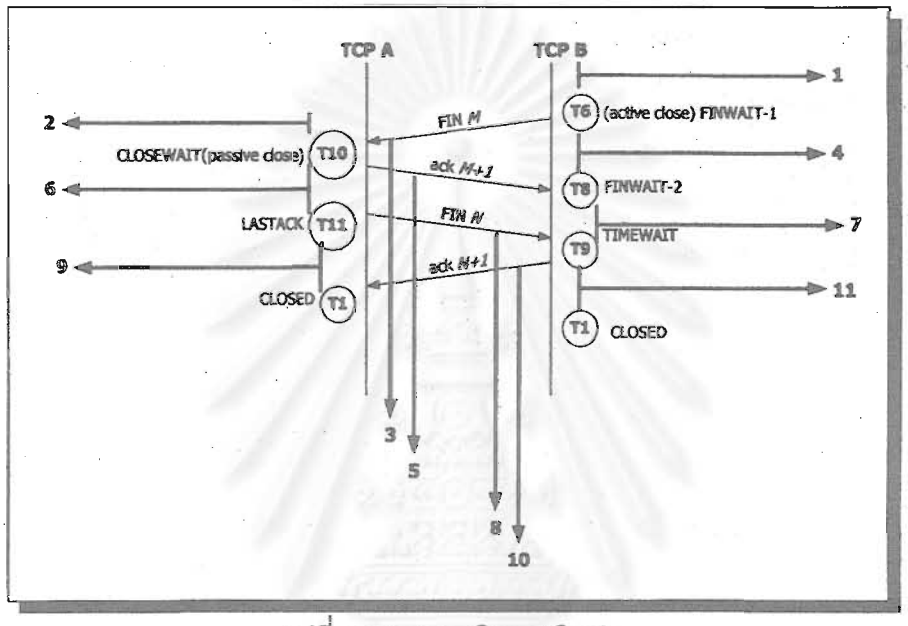
red retrieveDataInBufferSegment(inc inc inc inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 7169, ack = setAck, psh = setPsh, window = 4096, data = 800 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...eq12)

red retrieveDataInBufferSegment(inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 8193, ack = setAck, psh = setPsh, window = 4096, data = 800 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) . ...(eq13)

รูปที่ 4.6 สมการการจัดรูปนัยการส่งข้อมูล

5.5 ตัวอย่างการจัดรูปนัยการยกเลิกติดต่อ

การยกเลิกการติดต่อของทีซีพี (A และ B) มีขั้นตอนรับส่งไปมาระหว่างทีซีพีด้วยกัน 4 ขั้นตอน (สมการ 3 5 8 และ 10) ซึ่งเรียกว่า การปิดทีซีพีแบบครึ่ง (TCP's half close) ทั้งนี้ก่อนส่งข้อมูลไปมาระหว่างกัน ทีซีพีมีการเปลี่ยนสถานะและกำหนดค่าแรกเริ่มก่อนปิดการรับส่ง ดังนี้



รูปที่ 5.7 การยกเลิกการติดต่อ

red send(S, changeState(establishedState,finwait-1State), checkTcb(C), checkStatus(U)) . ...(eq1)

red send(S, changeState(establishedState,closewaitState), checkTcb(C), checkStatus(U)) . ...(eq2)

red send(Segment { seqnumber = M, fin = setFin, psh = setPsh, window = setWindow }, changeState(establishedState,finwait-1State), checkTcb(C), checkStatus(U)) . ...(eq3)

red send(S, changeState(establishedState,closewaitState), checkTcb(C), checkStatus(U)) . ...(eq4)

red send(Segment { seqnumber = M + 1, ack = setAck, psh = setPsh, window = setWindow }, changeState(timewaitState,finwait-2State), checkTcb(C), checkStatus(U)) . ...(eq5)

red send(S, changeState(closewaitState,lastackState), checkTcb(C), checkStatus(U)) . ...(eq6)

red send(S, changeState(finwait-2State,timewaitState), checkTcb(C), checkStatus(U)) . ...(eq7)

red send(Segment { seqnumber = N, fin = setFin, psh = setPsh, window = setWindow }, changeState(lastackState,timewaitState), checkTcb(C), checkStatus(U)) . ...(eq8)

red send(S, changeState(lastackState,closedState), checkTcb(C), checkStatus(U)) . ...(eq9)

red send(S, changeState(timewaitState,closedState), checkTcb(C), checkStatus(U)) . ...(eq10)

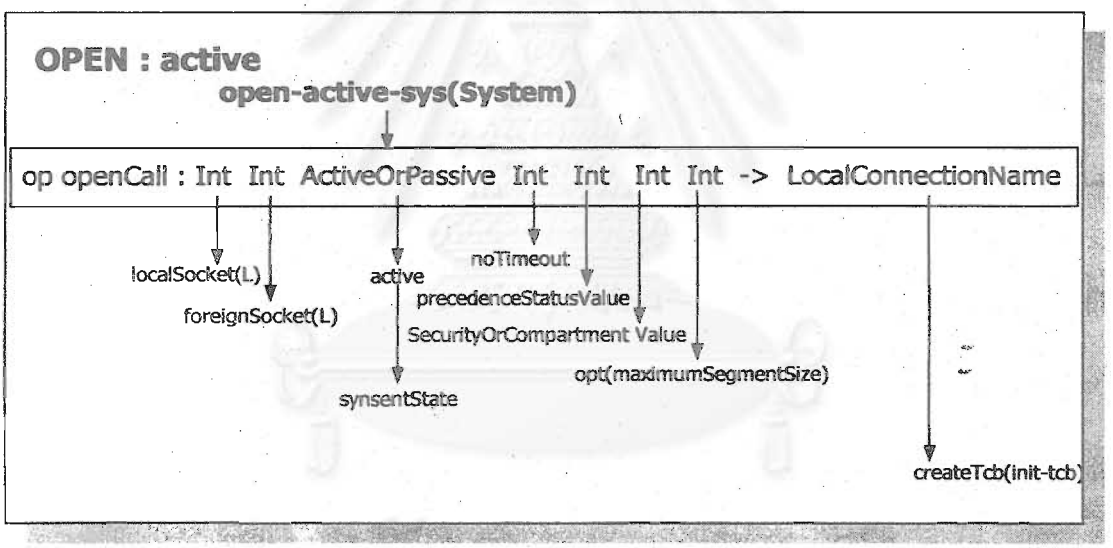
red send(Segment { seqnumber = N + 1, ack = setAck, psh = setPsh, window = setWindow }, changeState(timewaitState,closedState), checkTcb(C), checkStatus(U)) . ...(eq11)

รูปที่ 5.8 สมการการจัดรูปนัยการยกเลิกการติดต่อ

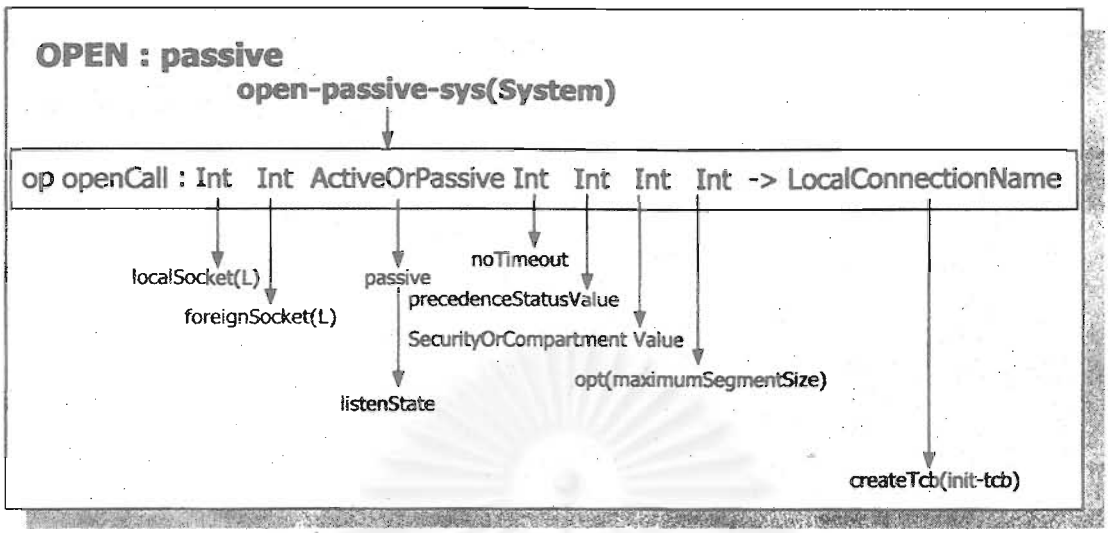
5.6 ตัวอย่างการจัดรูปนัยระดับบน

ระดับบนเป็นส่วนที่แสดงถึงขั้นตอนการทำงานจากผู้ใช้งานที่เรียกทำงานผ่านคำสั่งเรียกต่างๆ ไปควบคุมการทำงานในชั้นทีซีพี ขั้นตอนการทำงานต่างๆ นี้ จะทำงานโดยผ่านคำสั่งเรียก "เปิด" "ส่ง" "รับ" "ปิด" และ "ยกเลิก" โดยแสดงลำดับของกระบวนการและรายละเอียดขั้นตอนการทำงานดังต่อไปนี้

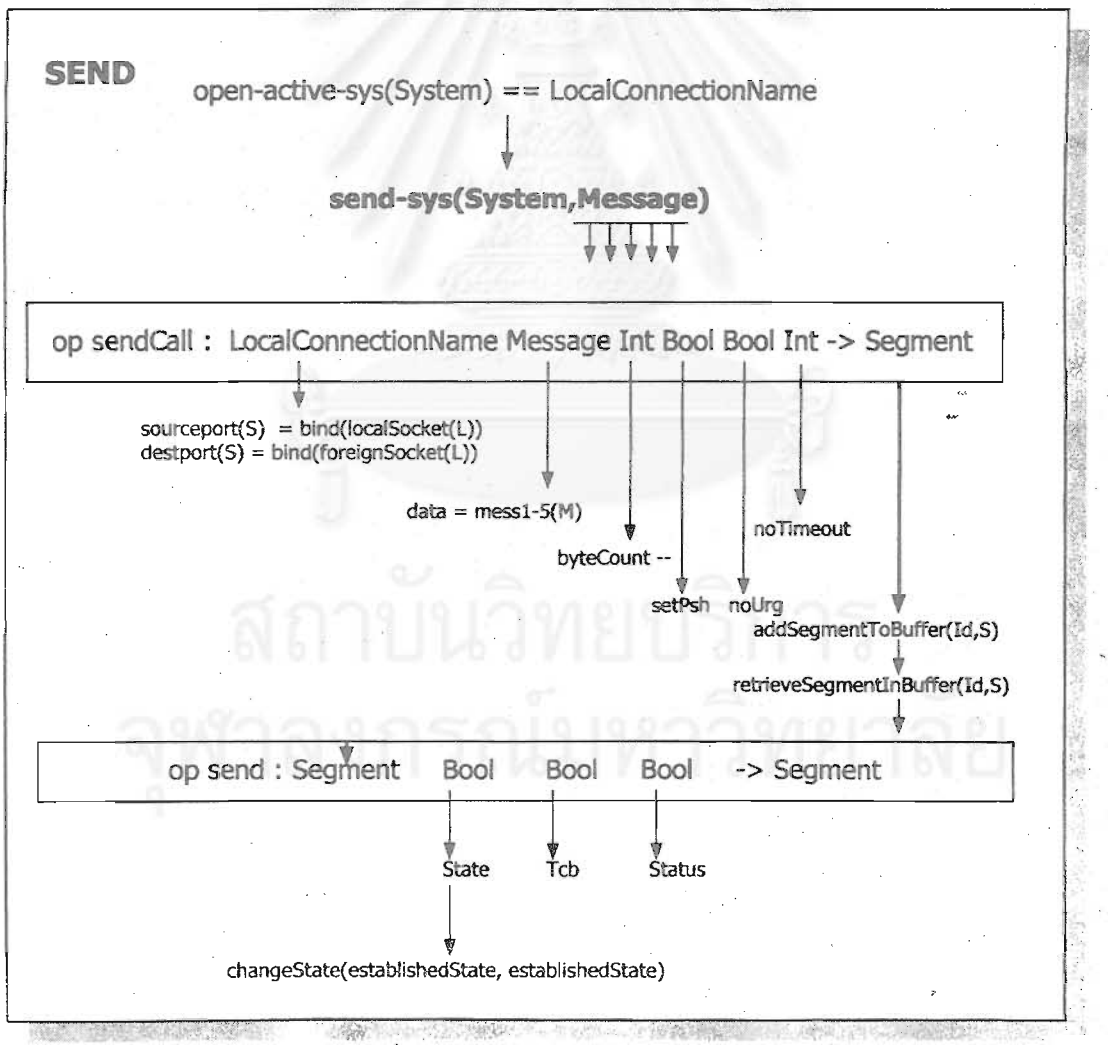
การส่งข้อมูลผ่านทีซีพีหนึ่งๆ ผู้ใช้จำเป็นต้องติดต่อเพื่อเปิดช่องทางในการรับส่งข้อมูลโดยผ่านคำสั่งเรียก "เปิด" (รูปที่ 5.9) เมื่อระบบสั่งเรียกเปิด (open-active-call(System)) ทีซีพีจะทำการเปิดแบบ active ด้วยการดำเนินการ openCall (รายละเอียดดังในรูปที่ 5.9) และเมื่อระบบ สั่งเรียกเปิด (open-passive-call(System)) ทีซีพีจะทำการเปิดแบบ passive (ด้วยการดำเนินการดังรูปที่ 5.10)



รูปที่ 5.9 การเรียกเปิดแบบ active จากผู้ใช้งาน

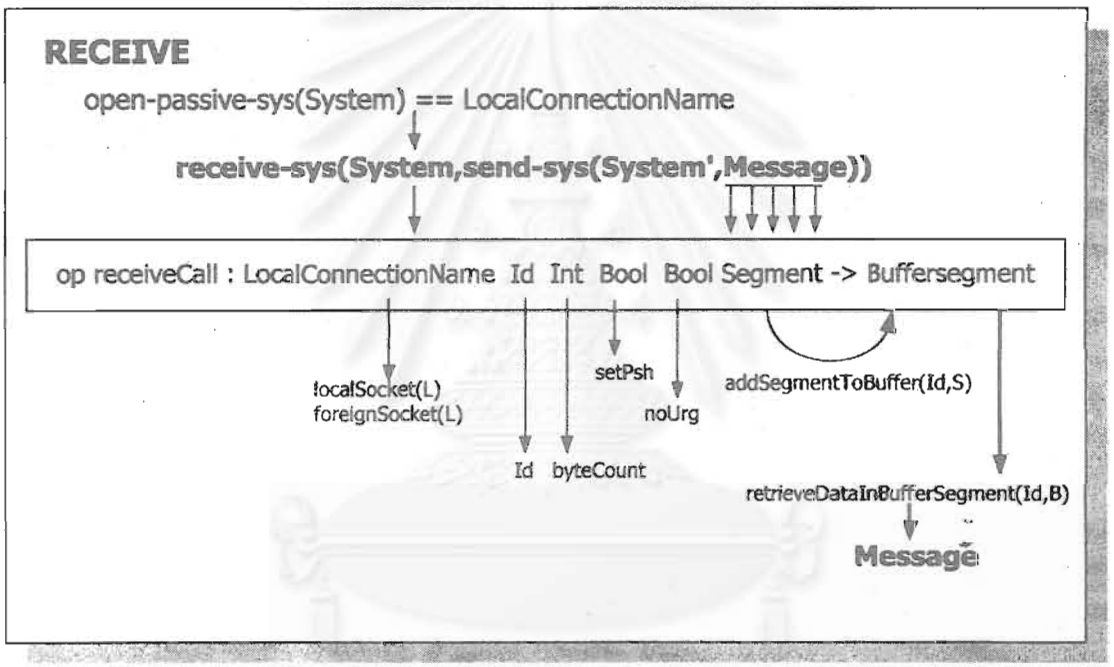


รูปที่ 5.10 การเรียกเปิดจากแบบ passive ผู้ใช้งาน

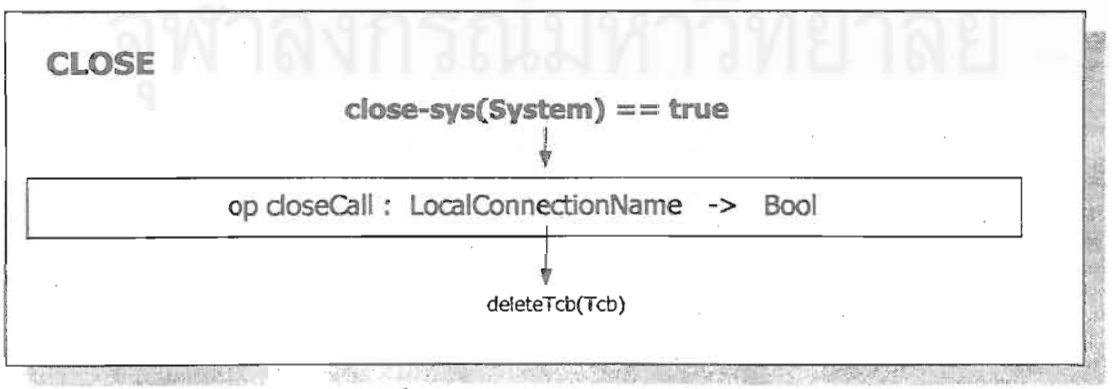


รูปที่ 5.11 การเรียกส่งจากผู้ใช้งาน

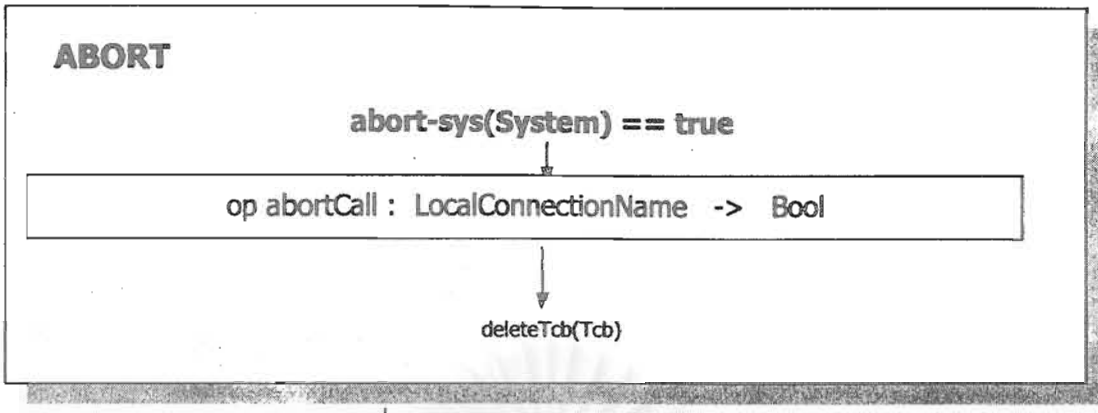
เมื่อระบบสั่งเรียกเปิดแบบ active สมบูรณ์แล้ว (open-active-call(System) == true) ขั้นตอนต่อไปจะเป็น ขั้นตอนที่ผู้ใช้งานทำการส่งข้อมูลโดยผ่านคำสั่งเรียก "ส่ง" เพื่อส่งข้อมูลไปยังอีกทีซีพี ระบบจะสั่งเรียกส่ง (send-sys(System,Message) ในรูปที่ 5.11) ทีซีพีจะทำการส่งข้อมูลด้วยการดำเนินการ sendCall (รายละเอียดดังในรูปที่ 5.11) เซกเมนต์ที่ต้องการส่งจะถูกส่งไปยังเป้าหมายตามช่องทางที่ได้ระบุเอาไว้แล้ว ด้านทีซีพีของผู้รับ (เมื่อระบบของผู้รับสั่งเรียกเปิดแบบ passive สมบูรณ์แล้ว) จะรับด้วยคำสั่งเรียก "รับ" (ระบบสั่งเรียกรับข้อมูล (receive-call(System,send-sys(System,Messgae)) ทีซีพีจะทำการรับข้อมูลด้วยการดำเนินการ receiveCall ดังรูปที่ 5.12)



รูปที่ 5.12 การเรียกรับจากผู้ใช้งาน



รูปที่ 5.13 การเรียกปิดจากผู้ใช้งาน



รูปที่ 5.14 การเรียกยกเลิกจากผู้ใช้งาน

เมื่อระบบทำการสั่งเรียกปิด (close-call(System) ดังรูปที่ 5.13) ทีซีพีจะทำการปิด การเชื่อมต่อด้วยการดำเนินการ closeCall พร้อมทั้งลบข้อมูลในระเบียนควบคุมการขนส่งที่อยู่ในหน่วยความจำ เป็นการคืนทรัพยากรให้แก่ผู้ใช้งาน

ทั้งนี้ระบบสามารถยกเลิกการติดต่อได้โดยผ่านคำสั่งเรียกยกเลิก (abort-call(System) ดังรูปที่ 5.14) ทีซีพีจะทำการยกเลิกการเชื่อมต่อด้วยการดำเนินการ abortCall พร้อมทั้งลบข้อมูลในระเบียนควบคุมการขนส่งในหน่วยความจำอีกด้วย

5.7 การทวนสอบ

การทวนสอบข้อกำหนดทีซีพีนี้ แบ่งการทวนสอบออกเป็นสองส่วนด้วยกัน คือ ทวนสอบมอดูลย่อยๆ ทุกๆ มอดูล (Unit Test) และการทวนสอบทั้งระบบ (System Test)

5.7.1 การทวนสอบมอดูลย่อยๆ

โดยการทวนสอบมีการตั้งสมมุติฐานด้วยหลักการเดียวกัน คือ มีการตั้งค่าแรกเริ่ม (Initial Value) ค่าตัวเลขตัวอย่าง (Integer) และค่าตัวแปร (Variable) (ดังรายละเอียดในรูปที่ 5.15) ผลการทวนสอบทั้งหมดถูกต้องดังแสดงรายละเอียดทั้งหมดในภาคผนวก ข

```
open INDEX
red val(init-index) .                    -- 0 : Zero
red val(inc init-index) .                -- 1 : NzNat
red val(dec init-index) .                -- val(dec init-index) : Int
red val(X) .                             -- val(X) : Int
close .
```

รูปที่ 5.15 ตัวอย่างการทวนสอบมอดูล INDEX

5.7.2 การทวนสอบระบบ

โดยการทวนสอบระบบจะมีการตั้งสมมุติฐานในการรับส่งข้อมูลไปมาจากทีซีพีหนึ่งไปยังอีกทีซีพีหนึ่ง และทดสอบการใช้คำสั่งเรียกเปิด รับ ส่ง ปิด และยกเลิก ทดสอบการรับส่งที่ไม่มีข้อมูล ทดสอบการรับส่งข้อมูลโดยหมายเลข IP ผิด (ดังรายละเอียดในรูปที่ 5.16) ดังตัวอย่างการทวนสอบในรูปที่ 5.16 ผลการทวนสอบระบบทั้งหมดถูกต้องดังแสดงรายละเอียดทั้งหมดในภาคผนวก ก

```
-- ———— Active Open ————
-- red open-active-sys(client) .                    -- openOk : Bool
-- red open-active-sys(errorSys) .                  -- openFail : Bool
-- red open-active-sys(self) .                      -- openFail : Bool
-- ———— Passive Open ————
-- red open-passive-sys(server) .                   -- openOk : Bool
-- red open-passive-sys(errorSys) .                 -- openFail : Bool
-- red open-passive-sys(self) .                     -- openFail : Bool
-- ———— Send and Receive Sys ————
-- red receive-sys(server, send-sys(client, m1)) .
   -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
   -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red receive-sys(server, send-sys(client, m2)) .
   -- Message { (mess1 = datasegment1 , mess2 = noMess , mess3 = noMess ,
   -- mess4 = noMess , mess5 = noMess)  } : Message
-- red receive-sys(client, send-sys(server, m1)) .
   -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
   -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red receive-sys(client, send-sys(server, m2)) .
   -- Message { (mess1 = datasegment1 , mess2 = noMess , mess3 = noMess ,
   -- mess4 = noMess , mess5 = noMess)  } : Message
-- ———— ErrorSys ————
-- red receive-sys(server, send-sys(errorSys, m1)) .    -- errorIp : Message
-- red receive-sys(errorSys, send-sys(client, m1)) .    -- errorIp : Message
-- red receive-sys(errorSys, send-sys(errorSys, m1)) .  -- errorIp : Message
-- red receive-sys(self, send-sys(server, m1)) .        -- errorIp : Message
-- red receive-sys(server, send-sys(self, m1)) .        -- errorIp : Message
-- red receive-sys(self, send-sys(self, m1)) .          -- errorIp : Message
-- ———— Close Sys ————
-- red close-sys(server) .                          -- true : Bool
-- red close-sys(errorSys) .                        -- false : Bool
-- red close-sys(self) .                            -- false : Bool
-- ———— Abort Sys ————
-- red abort-sys(client) .                          -- true : Bool
-- red abort-sys(errorSys) .                        -- false : Bool
-- red abort-sys(self) .                            -- false : Bool
```

รูปที่ 5.16 ตัวอย่างการทวนสอบระบบ TCP-SYSTEM

บทที่ 6

สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

งานวิจัยนี้ได้ทำการออกแบบแผนภาพเอดีเจในการกำหนดหลักนามธรรมข้อมูลของทีซีพี ใช้แผนภาพการเปลี่ยนสถานะในการกำหนดกระบวนการทำงานของทีซีพี ซึ่งมีกฎในการเชื่อมแผนภาพทั้งสองเข้าด้วยกัน ทำให้ข้อกำหนดทีซีพีซึ่งเป็นระบบที่ซับซ้อนนี้ สามารถระบุรายละเอียดข้อมูลและขั้นตอนกระบวนการทำงานของทีซีพีได้ชัดเจน แต่มิสามารถแสดงสมการจากแผนภาพทั้งสองนี้

งานวิจัยนี้เลือกการติดตั้งการเชื่อมต่อ การส่งข้อมูล และการยกเลิกการติดต่อของทีซีพี โดยใช้เส้นกำหนดเวลาเพื่อแสดงการรับส่งข้อมูลไปมาระหว่างทีซีพี รวมไปถึงระบบการเรียกใช้งานจากผู้ใช้ผ่านคำสั่งเรียกไปยังทีซีพีเหล่านี้เป็นกรณีศึกษา โดยข้อกำหนดดังกล่าวเขียนและทวนสอบด้วยภาษาคาเฟ่โอบีเจ ซึ่งผลลัพธ์ได้รับการตรวจสอบในส่วนวากยสัมพันธ์ และการพิสูจน์ทางคณิตศาสตร์ด้วยภาษาคาเฟ่โอบีเจ ว่ามีผลลัพธ์มีความถูกต้องและไม่คลุมเครือ

6.2 ข้อจำกัดของระบบ

ระบบที่พิจารณาตามข้อกำหนดนี้ จะไม่ครอบคลุมกรณีที่ผิดพลาด กรณีมีตัวบ่งชี้การเร่งและกรณีหมดเวลา โดยที่ขนาดของข้อมูลที่ใช้รับส่งต่อครั้ง ไม่เกินขนาดสูงสุดที่เซกเมนต์สามารถบรรจุได้ (Maximum Segment Size) เนื่องจากข้อจำกัดของระบบที่ออกแบบการเก็บข้อมูลของบัฟเฟอร์เป็นระเบียนเซกเมนต์ ประกอบกับข้อจำกัดของภาษาคาเฟ่โอบีเจที่ไม่สามารถจดจำค่าของข้อมูลในตัวแปร ระบบจึงไม่สามารถรับส่งข้อมูลที่มีขนาดใหญ่เกินกว่าที่เซกเมนต์จะบรรจุได้

6.3 ข้อเสนอแนะในการพัฒนาเพิ่ม

ข้อกำหนดนี้ใช้แผนภาพเอดีเจ แผนภาพการเปลี่ยนสถานะ และแผนภาพเส้นกำหนดเวลา ในการกำหนดร่วมด้วย ดังนั้น ผู้ใช้งานจึงต้องมีความรู้เกี่ยวกับแผนภาพดังกล่าว อีกทั้งจะมีกฎที่ใช้ในการเชื่อมแผนภาพดังกล่าวเข้าด้วยกัน ซึ่งหากผู้ใช้มีความชำนาญก็ยิ่งทำให้กำหนดข้อกำหนดได้รวดเร็วยิ่งขึ้น

จากวิธีการจัดรูปนัยและระบบที่ศึกษามานี้ เหมาะสำหรับกำหนดด้วยภาษาคาเฟ่โอบีเจเท่านั้น ดังนั้น ผู้ใช้งานจึงต้องมีความรู้และความเข้าใจในภาษานี้ เพราะจะต้องเขียนสมการขึ้นมาเอง จึงจะสามารถนำมาใช้งานและพัฒนาได้อย่างมีประสิทธิภาพยิ่งๆ ขึ้น

จากวิธีการจัดรูปนัยดังกล่าว อาจนำมาประยุกต์ใช้ในการจัดรูปนัยโพรโทคอลอื่นๆ หรือนำช่วยในการจำลองโปรแกรมที่เรียกใช้งานทีซีพี นอกจากนี้ยังสามารถเป็นข้อกำหนดเพื่อใช้ในการพัฒนา และออกแบบหรือพัฒนาทีซีพีเวอร์ชันๆ อื่นๆ

6.4 ผลงานที่เกี่ยวข้องกับการวิจัย

งานวิจัยนี้นอกจากได้จัดทำเป็นรูปเล่มวิทยานิพนธ์ฉบับนี้แล้ว งานวิจัยนี้ได้รับคัดเลือกและตีพิมพ์ใน Proceeding งานประชุมวิชาการ IconIT'2001 ในหัวข้อ "A Formal Specification Technique for the TCP Normal Connection Establishment and Termination" ระหว่างวันที่ 28-30 พฤษภาคม 2544 โดยรายละเอียดแสดงอยู่ในภาคผนวก ข

งานวิจัยนี้ยังได้รับคัดเลือกให้ถูกตีพิมพ์ใน Proceeding งานประชุมวิชาการ ANSCSE5 ในหัวข้อ "An Appropriate Formal Technique for Communication Protocols" เมื่อวันที่ 25 เมษายน 2544

รายการอ้างอิง

[1] Roger S. P. 1999. Software Engineering. Fifth Edition. Addison-Wesley Publish Company,

[2] Michael G.H. and Stephen A.J. 1993. Protocol Specification & verification Methods: An Overview. Proceeding of IEEE Singapore International Conference Vol. 2: 581-585.

[3] Layuan L. 1989. Formal Specification Technique for Communication Protocol. Proceeding of the 8th Annual Joint Conference of the IEEE Computer and Communications Societies Technology: Emerging or Converging Vol. 1: 74-81.

[4] Lai R. 1995. Formal specification and verification of a procedural protocol: case study. Software Engineering Journal Vol. 10: 97-104.

[5] Ataru T.N., Toshimi S. and Kokichi F. 1999. CafeOBJ User's Manual ver.1.4. [Online] Available from: http://www.ldl.jaist.ac.jp/cafeobj, [2001-Jan-3]

[6] Kokichi F. and Ataru N. 1997. An Overview of CAFE Specification Environment - an algebraic approach for creating verifying, and maintaining formal specifications over network. Proceeding of the First IEEE International Conference on Formal Engineering Methods: 170-181.

[7] Douglas E.C. and David L.S. 1996. Internetworking with TCP/IP Client-Server Programming and Application. Second Edition. Printed in the United States of America : Prentice-Hall International, inc.,

[8] Information Sciences Institute University of Southern California. 1981. Transmission Control Protocol Darpa Internet Program Protocol specification. RFC793 (Sep 1981): 1-85.

[9] Washburn K. and Evans J.T. 1993. . TCP/IP Running a Successful Network. Second Edition. Addison-Wesley Publishing Company,

[10] Richard S. 1995. TCP/IP Illustrated Volume I. Addison-Wesley,

[11] Information Sciences Institute University of Southern California. 1981. Transmission Control Protocol Darpa Internet Program Protocol specification. RFC793. California,

[12] Protocol Specification & verification Methods: An Overview. Proceeding of IEEE Singapore International Conference on Networks Vol. 2 (1993): 581-585.

ภาคผนวก

ภาคผนวก ก
ข้อกำหนดรูปนัยทีซีพี

```
-- ------------------------------------------------------------------------------------------
-- ------------------------------------ INDEX Module ----------------------------------------
-- ------------------------------------------------------------------------------------------
mod* INDEX{
        pr(INT)

        *[ Index ]*

        op init-index : -> Index      -- initial
        op dec_ : Index -> Index  -- method
        op inc_ : Index -> Index    -- method
        bop val : Index -> Int        -- attribute

        var X : Index

        eq val(init-index) = 0 .
        eq val(inc X) = (val(X) + 1 ) .
        ceq val(dec X) = (val(X) - 1 ) if val(X) =/= 0 .
}
-- ------------------------------------------
mod* ID{
        protecting(!NDEX *{ hsort Index -> Id })
}
-- ------------------------------------------------------------------------------------------
-- ------------------------------------ MESSAGE Module --------------------------------------
-- ------------------------------------------------------------------------------------------
mod* MESSAGE{
        pr(INT)
        *[ Data Ops ]*
        record Message{
                mess1 : Int
                mess2 : Int
                mess3 : Int
                mess4 : Int
                mess5 : Int
```

```
    }
    op mess1 : Message -> Int
    op mess2 : Message -> Int
    op mess3 : Message -> Int
    op mess4 : Message -> Int
    op mess5 : Message -> Int

    op message1 : -> Message
    op noMessage : -> Message
    op noMess : -> Int

    op len : Int -> Int
    op ErrorLen : -> Int

    op endOfOption : -> Ops
    op noOperation : -> Ops
    op maximumSegmentSize : -> Ops
    op opt : Ops -> Int

    var I : Int

    eq opt(maximumSegmentSize) = 429497296 .        -- 65536 * 65536 = Data 24 bit : Segment
    ceq len (I) = I
        if I < opt(maximumSegmentSize) + 1 .        -- Message Length not over 1024
    ceq len (I) =  ErrorLen
        if I > opt(maximumSegmentSize) .            -- Message Length error if over 1024
}
-- ------------------------------------------------------------------------------------
-- ------------------------------- SEGMENT Module ------------------------------------
-- ------------------------------------------------------------------------------------
mod* SEGMENT{
    pr(ID + MESSAGE)
    record Segment{
        sourceport : Int
        destport : Int
```

```
    seqnumber : Int
    acknumber : Int
    offset : Int            -- value = 0 ; where data Begin
    reserved : Int          -- value =0 ; for future use
    urg : Bool
    ack : Bool
    psh : Bool
    rst : Bool
    syn : Bool
    fin : Bool
    window : int            -- value = 0 ( for no Segment )
    checksum : Int          -- value = 0 ( checking data in correctly)
    urgentptr : Int         -- noUrgentprt
    options : Int           -- value = 0 (endOfOption), value = 1 (noOperation),
                            value = 2 (maximumSegmentSize)
    padding : Int           -- value = 0 ;Tcp header end and data begin on 32 bit boundary
    data : Int
}
op init-segment : -> Segment        -- initial
op no-segment : -> Segment          -- initial

op sourceport : Segment -> Int      -- projection
op destport : Segment -> Int        -- projection
op seqnumber : Segment -> Int       -- projection
op acknumber : Segment -> Int       -- projection
op options : Segment -> Int         -- projection
op urg : Segment -> Bool            -- projection
op psh : Segment -> Bool            -- projection
op syn : Segment -> Bool            -- projection
op ack : Segment -> Bool            -- projection
op fin : Segment -> Bool            -- projection
op data : Segment -> Int            -- projection

op initSouceport : -> Int                   -- method for "sourceport" bit
op initDestport : -> Int                    -- method for "destport" bit
```

```
op dataBegin : -> Int                    -- method for "offset" bit
op reservedValue : -> Int                -- method for "reserved" bit
op setUrg : -> Bool                      -- method for "urg" bit
op resetUrg : -> Bool                    -- method for "urg" bit
op setAck : -> Bool                      -- method for "ack" bit
op resetAck : -> Bool                    -- method for "ack" bit
op setPsh : -> Bool                      -- method for "psh" bit
op resetPsh : -> Bool                    -- method for "psh" bit
op setRst : -> Bool                      -- method for "rst" bit
op resetRst : -> Bool                    -- method for "rst" bit
op setSyn : -> Bool                      -- method for "syn" bit
op resetSyn : -> Bool                    -- method for "syn" bit
op setFin : -> Bool                      -- method for "fin" bit
op resetFin : -> Bool                    -- method for "fin" bit
op noWindow : -> Int                     -- method for "window" bit
op setWindow : -> Int                    -- method for "window" bit
op correctChecksum : -> Int              -- method for "checksum" bit
op errorChecksum : -> Int                -- method for "checksum" bit
op noUrgentptr : -> Int                  -- method for "urgentprt" bit
op paddingValue : -> Int                 -- method for "padding" bit

op initIss : -> Int                      -- method for "iss" bit
op initIrs : -> Int                      -- method for "irs" bit

eq dataBegin = 0 .                       -- for "offset" bit
eq reservedValue = 0 .                   -- for "reserved" bit
eq noWindow = 0 .                        -- for "window" bit
eq setWindow = 1 .                       -- for initial "window" bit
eq correctChecksum = 0 .                 -- for "checksum" bit
eq errorChecksum = 1 .                   -- for "checksum" bit
eq opt(endOfOption) = 0 .                -- for "options" bit
eq opt(noOperation) = 1 .                -- for "options" bit
eq opt(maximumSegmentSize) = 2 .         -- for "options" bit
eq paddingValue = 0 .                    -- for "padding" bit
eq initIss = 0 .
```

```
    eq initIrs = 0 .

    eq init-segment = Segment {        sourceport = initSouceport,
                                       destport = initDestport,
                                       seqnumber = initIss,
                                       acknumber = initIrs,
                                       offset = dataBegin,
                                       reserved = reservedValue,
                                       urg = resetUrg,
                                       ack = resetAck,
                                       psh = setPsh,
                                       rst = resetRst,
                                       syn = setSyn,
                                       fin = resetFin,
                                       window = setWindow,
                                       checksum = correctChecksum,
                                       urgentptr = noUrgentptr,
                                       options = opt(maximumSegmentSize),
                                       padding = paddingValue,
                                       data = noMess } .
}
-- ------------------------------------------------------------------------------------
-- ------------------------------- BUFFER Module ------------------------------------
-- ------------------------------------------------------------------------------------
mod* BUFFER{
    pr(ID + SEGMENT)
    record Buffersegment{
        sourceport : Int
        destport : Int
        seqnumber : Int
        acknumber : Int
        offset : Int            -- value = 0 ; where data Begin
        reserved : Int          -- value =0 ; for future use
        urg : Bool
        ack : Bool
```

```
        psh : Bool
        rst : Bool
        syn : Bool
        fin : Bool
        window : Int          -- value = 0 ( for no Segment )
        checksum : Int        -- value = 0 ( checking data in correctly)
        urgentptr : Int       -- noUrgentprt
        options : Int         -- value = 0 (endOfOption), value = 1 (noOperation),
                              value = 2 (maximumSegmentSize)
        padding : Int         -- value = 0 ;Tcp header end and data begin on 32 bit boundary
        data : Int
    }

    op init-buffersegment : -> Buffersegment      -- initial
    op no-buffersegment : -> Buffersegment        -- initial

    op sourceport : Buffersegment -> Int          -- projection
    op destport : Buffersegment -> Int            -- projection
    op options : Buffersegment -> Int             -- projection
    op urg : Buffersegment -> Bool                -- projection
    op psh : Buffersegment -> Bool                -- projection
    op syn : Buffersegment -> Bool                -- projection
    op ack : Buffersegment -> Bool                -- projection
    op fin : Buffersegment -> Bool                -- projection
    op data : Buffersegment -> Int                -- projection
-- Buffersegment(Old), Buffersegment( want to add ) - > Buffersegment(new)          -- method
    op addDataToBufferSegment : Id Buffersegment Buffersegment -> Buffersegment   -- method
    op delBufferSegment : Id Buffersegment -> Buffersegment                       -- method
    op retrieveSegmentInBuffer : Id Buffersegment -> Segment                      -- method
    op addSegmentToBuffer : Id Segment -> Buffersegment                           -- method
    op retrieveDataInBufferSegment : Id Buffersegment -> Int                      -- attribute

    vars X X' : Id
    vars B B' : Buffersegment
    var S : Segment
```

```
    var I : Int
    var D : Int

    eq init-buffersegment = Buffersegment {  sourceport = initSouceport,
                                             destport = initDestport,
                                             seqnumber = initIss,
                                             acknumber = initIrs,
                                             offset = dataBegin,
                                             reserved = reservedValue,
                                             urg = resetUrg,
                                             ack = resetAck,
                                             psh = setPsh,
                                             rst = resetRst,
                                             syn = setSyn,
                                             fin = resetFin,
                                             window = setWindow,
                                             checksum = correctChecksum,
                                             urgentptr = noUrgentptr,
                                             options = opt(maximumSegmentSize),
                                             padding = paddingValue,
                                             data = noMess } .
    eq retrieveSegmentInBuffer(X,B) = S .
    ceq retrieveSegmentInBuffer(X,B) = no-segment if B == no-buffersegment .
    ceq addSegmentToBuffer(X,S) = B if val(X) =/= -1 .
    ceq addSegmentToBuffer(X,S) = B if retrieveSegmentInBuffer(X,B) == no-segment .

    ceq retrieveDataInBufferSegment( X, addSegmentToBuffer(X',S) ) = data(S) if X == X' .
    eq retrieveDataInBufferSegment(X, B) = data(B) .
    ceq addDataToBufferSegment(X,B,B') = B' if retrieveSegmentInBuffer(X,B) == no-segment .

    eq delBufferSegment(X,B) = addSegmentToBuffer(X,no-segment) .
    ceq delBufferSegment(X,B) = no-buffersegment if retrieveSegmentInBuffer(X,B) == S .
    ceq retrieveSegmentInBuffer(X,addSegmentToBuffer(X',S)) = S if X == X' .
}
-- ----------------------------------------------------------------------------------------------------
```

```
-- ---------- STATE Module ----------
-- ------------------------------------------------------------------------------------------

mod* STATE{
    pr(INT)
    *[ State ]*

    op closedState : -> State              -- initial
    op listenState : -> State              -- initial
    op synsentState : -> State             -- initial
    op synrcvdState : -> State             -- initial
    op establishedState : -> State         -- initial
    op finwait-1State : -> State           -- initial
    op finwait-2State : -> State           -- initial
    op closingState : -> State             -- initial
    op timewaitState : -> State            -- initial
    op closewaitState : -> State           -- initial
    op lastackState : -> State             -- initial

    op changeState : State State -> Bool -- method
    op valueState : State -> Int           -- attribute
    op curState : -> State

    vars T T' : State
    var I : Int

ceq changeState(T,T') = true
    if ( T == closedState and ( T' == listenState or T' == synsentState ) ) .
ceq changeState(T,T') = false
    if ( T == closedState and ( T' == synrcvdState or T' == establishedState or
    T' == finwait-1State or T' == finwait-2State or T' == closingState or T' == timewaitState
    or T' == closewaitState or T' == lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == listenState and ( T' == synsentState or T' == synrcvdState or T' == closedState ) ) .
ceq changeState(T,T') = false
    if ( T == listenState and ( T' == listenState or T' == establishedState or T' == finwait-1State
```

```
    or T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T'
    == lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == synsentState and ( T' == closedState or T' == synrcvdState or T' == establishedState )
    ) .
ceq changeState(T,T') = false
    if ( T == synsentState and ( T' == listenState or T' == synsentState or T' == finwait-1State or
    T' == finwait-2State or T' == closingState or T' == timewaitState or T' == closewaitState or T' ==
    lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == synrcvdState and ( T' == establishedState or T' == finwait-1State or T' ==
    closewaitState ) ) .
ceq changeState(T,T') = false
    if ( T == synrcvdState and ( T' == closedState or T' == listenState or T' == synsentState or
    T' == synrcvdState or T' == finwait-2State or T' == closingState or T' == timewaitState or T' ==
    lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == establishedState and ( T' == finwait-1State or T' == closewaitState or T' ==
    establishedState ) ) .
ceq changeState(T,T') = false
    if ( T == establishedState and ( T' == closedState or T' == listenState or T' == synsentState or
    T' == synrcvdState or T' == establishedState or T' == finwait-2State or T' == closingState or
    T' == timewaitState or T' == lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == finwait-1State and ( T' == closingState or T' == finwait-2State ) ) .
ceq changeState(T,T') = false
    if ( T == finwait-1State and ( T' == closedState or T' == listenState or T' == synsentState or
    T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == timewaitState or
    T' == closewaitState or T' == lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == closingState and T' == timewaitState ) .
ceq changeState(T,T') = false
    if ( T == closingState and ( T' == closedState or T' == listenState or T' == synsentState or
    T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or
    T' == closingState or T' == closewaitState or T' == lastackState ) ) .
```

```
ceq changeState(T,T') = true
    if ( T == finwait-2State and T' == timewaitState ) .
ceq changeState(T,T') = false
    if ( T == finwait-2State and ( T' == closedState or T' == listenState or T' == synsentState or
    T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or
    T' == closingState or T' == closewaitState or T' == lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == timewaitState and T' == closedState ) .
ceq changeState(T,T') = false
    if ( T == timewaitState and ( T' == listenState or T' == synsentState or T' == synrcvdState or
    T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or
    T' == timewaitState or T' == closewaitState or T' == lastackState ) ) .
ceq changeState(T,T') = true
    if ( T == closewaitState and T' == lastackState ) .
ceq changeState(T,T') = false
    if ( T == closewaitState and ( T' == closedState or T' == listenState or T' == synsentState or
    T' == synrcvdState or T' == establishedState or T' == finwait-1State or T' == finwait-2State or
    T' == closingState or T' == timewaitState or T' == closewaitState ) ) .
ceq changeState(T,T') = true
    if ( T == lastackState and T' == closedState ) .
ceq changeState(T,T') = false
    if ( T == lastackState and ( T' == listenState or T' == synsentState or T' == synrcvdState or
    T' == establishedState or T' == finwait-1State or T' == finwait-2State or T' == closingState or
    T' == timewaitState or T' == closewaitState or T' == lastackState ) ) .


    ceq valueState(T) = 1 if T == closedState .
    ceq valueState(T) = 2 if T == listenState .
    ceq valueState(T) = 3 if T == synsentState .
    ceq valueState(T) = 4 if T == synrcvdState .
    ceq valueState(T) = 5 if T == establishedState .
    ceq valueState(T) = 6 if T == finwait-1State .
    ceq valueState(T) = 7 if T == closingState .
    ceq valueState(T) = 8 if T == finwait-2State .
    ceq valueState(T) = 9 if T == timewaitState .
    ceq valueState(T) = 10 if T == closewaitState .
```

```
        ceq valueState(T) = 11 if T == lastackState .
}
-- ------------------------------------------------------------------------------------------
-- ---------------------------------------- TCB Module ----------------------------------------
-- ------------------------------------------------------------------------------------------
mod* TCB{
        pr(INDEX + BUFFER)
        record Tcb{
-- send
                snd-una : Int           -- send unacknowledged
                snd-nxt : Int           -- send next
                snd-wnd : Int           -- send window
                snd-up : Int            -- send urgent pointer
                snd-wl1 : Int           -- send sequence number used for last window update
                snd-wl2 : Int           -- send acknowledgement number used for last window  update
                iss : Int               -- initial send sequence number
-- recv
                rcv-nxt : Int           -- receive next
                rcv-wnd : Int           -- receive window
                rcv-up : Int            -- receive urgent pointer
                irs : Int               -- initial receive sequence number
-- cur segment
                seg-seg : Int           -- segment sequence number
                seg-ack : Int           -- segment acknowledgement number
                seg-len : Int           -- segment length
                seg-wnd : Int           -- segment window
                seg-up : Int            -- segment urgent pointer
                seg-prc : Int           -- segment security and precedence value
        }
        op iss : Tcb -> Int                     -- projection
        op irs : Tcb -> Int                     -- projection
        op snd-una : Tcb -> Int                 -- projection
        op snd-nxt : Tcb -> Int                 -- projection
        op snd-wnd : Tcb -> Int                 -- projection
        op snd-up : Tcb -> Int                  -- projection
```

```
op snd-wl1 : Tcb -> Int                    -- projection
op snd-wl2 : Tcb -> Int                    -- projection
op rcv-nxt : Tcb -> Int                    -- projection
op rcv-wnd : Tcb -> Int                    -- projection
op rcv-up : Tcb -> Int                     -- projection
op seg-seg : Tcb -> Int                    -- projection
op seg-ack : Tcb -> Int                    -- projection
op seg-len : Tcb -> Int                    -- projection
op seg-wnd : Tcb -> Int                    -- projection
op seg-up : Tcb -> Int                     -- projection
op seg-prc : Tcb -> Int                    -- projection

op sndunaValue : -> Int                    -- method for "snd-una" bit
op sndnxtValue : -> Int                    -- method for "snd-nxt" bit
op sndwndValue : -> Int                    -- method for "snd-wnd" bit
op sndupValue : -> Int                     -- method for "snd-up" bit
op sndwl1Value : -> Int                    -- method for "snd-wl1" bit
op sndwl2Value : -> Int                    -- method for "snd-wl2" bit
op rcvnxtValue : -> Int                    -- method for "rcv-nxt" bit
op rcvwndValue : -> Int                    -- method for "rcv-wnd" bit
op rcvupValue : -> Int                     -- method for "rcv-up" bit
op segsegValue : -> Int                    -- method for "seg-seg" bit
op segackValue : -> Int                    -- method for "seg-ack" bit
op seglenValue : -> Int                    -- method for "seg-len" bit
op segwndValue : -> Int                    -- method for "seg-wnd" bit
op segupValue : -> Int                     -- method for "seg-up" bit
op segprcValue : -> Int                    -- method for "seg-prc" bit
op precedenceStatusValue : -> Int

op init-tcb : -> Tcb                       -- initial
op no-tcb : -> Tcb                         -- initial

op deleteTcb : Tcb -> Tcb                  -- method
op createTcb : -> Tcb                      -- method
```

```
    op checkTcb : Tcb -> Bool                    -- attribute

    vars C C' : Tcb
    var S : Segment
    var Id : Index

    eq precedenceStatusValue = 0 .
    eq snd-una(C) = sndunaValue .
    eq snd-nxt(C) = sndnxtValue .
    eq snd-wnd(C) = sndwndValue .
    eq snd-up(C) = noUrgentptr .
    eq snd-wl1(C) = sndwl1Value .
    eq snd-wl2(C) = sndwl2Value .
    eq rcv-nxt(C) = val(Id) + 1 .
    eq rcv-wnd(C) = val(Id) .
    eq rcv-up(C) = noUrgentptr .
    eq seg-seg(C) = seqnumber(S) .
    eq seg-ack(C) = acknumber(S) .
    eq seg-len(C) = opt(maximumSegmentSize) .
    eq seg-wnd(C) = window(S) .
    eq seg-up(C) = noUrgentptr .
    eq seg-prc(C) = precedenceStatusValue .
    eq init-tcb = Tcb {  snd-una = initIss, snd-nxt = initIss + 1, snd-wnd = sndwndValue, snd-up =
noUrgentptr,snd-wl1 = sndwl1Value, snd-wl2 = sndwl2Value, iss = initIss, rcv-nxt = val(init-index) +
1, rcv-wnd = val(init-index), rcv-up = noUrgentptr, irs = initIrs, seg-seg = initIss, seg-ack = initIrs,
seg-len = opt(maximumSegmentSize), seg-wnd = window(init-segment), seg-up = noUrgentptr, seg-
prc = precedenceStatusValue } .
    eq createTcb = init-tcb .
    eq deleteTcb(C) = no-tcb .
    ceq checkTcb(C) = false if iss(C) == -1 .
    ceq checkTcb(C) = true if iss(C) =/= -1 .
}
-- ------------------------------------------------------------------------------------------------------
-- ------------------------------------- STATUSCALL Module ------------------------------------
-- ------------------------------------------------------------------------------------------------------
```

```
mod* STATUSCALL{
      pr(STATUS)
      op statusCall : LocalConnectionName -> Status              -- method
      var L : LocalConnectionName
      var U : Status
      ceq checkStatus(statusCall(L)) = true if localSocket(L) > 0 and foreignSocket(L) > 0 .
      ceq checkStatus(statusCall(L)) = false if ( localSocket(L) < 1 ) or ( foreignSocket(L) < 1 ) .
      ceq checkStatus(U) = true if U == init-status .
      eq checkStatus(U) = true .
      ceq checkStatus(U) = true if checkStatus(statusCall(L)) == true  .
}
-- ------------------------------------------------------------------------------------------
-- ------------------------------------ SEND Module ------------------------------------
-- ------------------------------------------------------------------------------------------
mod* SEND{
      pr(BUFFER + TCB + STATE + STATUS + STATUSCALL)

-- State, Tcb, Status
      op send : Segment Bool Bool Bool -> Segment

      var S : Segment
      var C : Tcb
      vars B1 B2 B3 : Bool
      vars B1 B2 B3 : Bool
      var L : LocalConnectionName
      var U : Status

      ceq send(S,B1,B2,B3) = no-segment
            if window(S) == noWindow and B1 == true and B2 == true and B3 == true .
      ceq send(S,B1,B2,B3) = no-segment
            if psh(S) == resetPsh and B1 == true and B2 == true and B3 == true .
      ceq send(S,B1,B2,B3) = no-segment
            if B1 == false or B2 == false and B3 == false .
-- send Ack, Fin : seqnumber = acknumber, acknumber = seqnumber + 1
      ceq send(S,B1,B2,B3) =
```

Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = acknumber(S), acknumber = seqnumber(S) + 1, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = resetAck, psh = setPsh, rst = resetRst, syn = setSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr = noUrgentptr, options = opt(maximumSegmentSize), padding = paddingValue, data = data(S) }

if window(S) =/= noWindow and psh(S) =/= resetPsh and ack(S) == setAck and fin(S) == setFin and B1 == true and B2 == true and B3 == true .

-- send Syn, Ack : acknumber = seqnumber + 1, seqnumber = acknumber

ceq send(S,B1,B2,B3) =

Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = acknumber(S), acknumber = seqnumber(S) + 1 , offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = setAck, psh = setPsh, rst = resetRst, syn = setSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr = noUrgentptr, options = opt(maximumSegmentSize), padding = paddingValue, data = data(S) }

if window(S) =/= noWindow and psh(S) =/= resetPsh and syn(S) == setSyn and ack(S) == setAck and B1 == true and B2 == true and B3 == true .

-- send Syn : acknumber = seqnumber + 1

ceq send(S,B1,B2,B3) =

Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = initIss, acknumber = seqnumber(S) + 1, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = resetAck, psh = setPsh, rst = resetRst, syn = setSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr = noUrgentptr, options = opt(maximumSegmentSize), padding = paddingValue, data = data(S) }

if window(S) =/= noWindow and psh(S) =/= resetPsh and syn(S) == setSyn and B1 == true and B2 == true and B3 == true .

-- send Ack : seqnumber = acknumber

ceq send(S,B1,B2,B3) =

Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = acknumber(S), acknumber = initIrs, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack = setAck, psh = setPsh, rst = resetRst, syn = resetSyn, fin = resetFin, window = setWindow, checksum = correctChecksum, urgentptr = noUrgentptr, options = opt(maximumSegmentSize), padding = paddingValue, data = data(S) }

if window(S) =/= noWindow and psh(S) =/= resetPsh and ack(S) == setAck and B1 == true and B2 == true and B3 == true .

-- send Rst : seqnumber = iss(C)

```
    ceq send(S,B1,B2,B3) =
    Segment { sourceport = bind(sourceport(S)), destport = bind(destport(S)), seqnumber = iss(C),
    acknumber = initIrs, offset = dataBegin, reserved = reservedValue, urg = resetUrg, ack =
    resetAck, psh = setPsh, rst = setRst, syn = resetSyn, fin = resetFin, window = setWindow,
    checksum = correctChecksum, urgentptr = noUrgentptr, options = opt(maximumSegmentSize),
    padding = paddingValue, data = data(S) }
        if window(S) =/= noWindow and psh(S) =/= resetPsh and rst(S) == setRst
        and B1 == true and B2 == true and B3 == true .
    ceq send(S,B1,B2,B3) = S if B1 == true and B2 == true and B3 == true .
}
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- OPENCALL Module ------------------------------------
-- ------------------------------------------------------------------------------------------
mod* OPENCALL{
    pr(SEGMENT + STATE + TCB + STATUS + STATUSCALL)
    *[ ActiveOrPassive ]*
-- OPEN(1:local port, 2:foreign socket, 3:active (1) /passive(0) 4:[,timeout] 5:[,precedence]
6:[,security/compartment] 7:[,options])
    op openCall : Int Int ActiveOrPassive Int Int Int Int -> LocalConnectionName        -- method

    op active : -> ActiveOrPassive
    op passive : -> ActiveOrPassive
    op timeout : -> Int

    vars I1 I2 I4 I5 I6 I7 : Int
    var S : Segment
    var O : Ops
    var P : ActiveOrPassive
    var C : Tcb
    var L : LocalConnectionName
    ceq openCall( I1, I2, P, I4, I5, I6, I7) = LocalConnectionName { localSocket = I1, foreignSocket
= I2 }
        if ( I1 > 0 ) and ( I2 > 0 ) and
        ( I1 =/= I2 ) and
        ( P == active or P == passive ) and
```

```
            I4 == noTimeout and
            I5 == precedenceStatusValue and
            I6 == securityOrCompartmentValue and
            I7 == opt(maximumSegmentSize) .                    -- correct local + foreign port
      ceq openCall( I1, I2, P, I4, I5, I6, I7) = errorLocalConnectionName
            if ( I1 < 1 or I2 < 1 or  I1 == I2 ) and
            ( P == active or P == passive ) and
            I4 == noTimeout and
            I5 == precedenceStatusValue and
            I6 == securityOrCompartmentValue and
            I7 == opt(maximumSegmentSize) .                                -- error IP
      ceq curState = listenState
            if openCall( I1, I2, P, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize) )
            ==  LocalConnectionName { localSocket  = I1, foreignSocket = I2 } .
      ceq curState = synsentState
            if openCall( I1, I2, P, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize) )
            ==  LocalConnectionName { localSocket  = I1, foreignSocket = I2 } .
      ceq checkTcb(createTcb) = true
            if openCall(I1,I2,P,I4,I5,I6,I7) == LocalConnectionName { localSocket  = I1, foreignSocket
= I2 } .
}
-- ------------------------------------------------------------------------------------------------
-- ------------------------------------ SENDCALL Module ------------------------------------
-- ------------------------------------------------------------------------------------------------
mod* SENDCALL{
      pr(BUFFER + OPENCALL + SEND + MESSAGE + STATUS + TCB)
-- SEND(I1:local connection name, Id:buffer address= Message,
-- I3:byte count = num of segment, B4:PUSH flag, B5:URGENT flag, I6:[,timeout])
      op sendCall : LocalConnectionName Message Int Bool Bool Int -> Segment          -- method

      op byteCount : -> Int        -- for I3
      op noUrg : -> Bool
```

```
vars I1 I2 I3 I6 : Int
vars B1 B2 B3 B4 B5 : Bool
var C : Tcb
var S : Segment
var P : ActiveOrPassive
var U : Status
var I : Id
var B : Buffersegment
var L : LocalConnectionName
var M : Message

eq byteCount = 5 .                    -- number of Message segment

ceq sendCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 } ,M, I3, B4, B5, I6)
= no-segment
    if I1 < 1 or
    I2 < 1 or
    I1 == I2 or
    M == noMessage or
    ( I3 < 0 or I3 > byteCount ) or
    B4 =/= setPsh or
    B5 =/= noUrg or
    I6 =/= noTimeout .
ceq sendCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 },M, I3, B4, B5, I6)
= send(retrieveSegmentInBuffer(init-index ,addSegmentToBuffer(init-index ,
Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
urgentptr = B5, data = mess1(M) } )), changeState(establishedState,establishedState),
checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
foreignSocket = I2 })) )
    if I1 > 0 and
    I2 > 0 and
    I1 =/= I2 and
    M =/= noMessage and
    I3 == ( byteCount - 4 ) and
    B4 == setPsh and
```

B5 == noUrg and

I6 == noTimeout .

ceq sendCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 },M, I3, B4, B5, I6) = send(retrieveSegmentInBuffer(inc init-index ,addSegmentToBuffer(inc init-index , Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 , urgentptr = B5, data = mess2(M) })), changeState(establishedState,establishedState), checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 })))

if I1 > 0 and

I2 > 0 and

I1 =/= I2 and

M =/= noMessage and

I3 == (byteCount - 3) and

B4 == setPsh and

B5 == noUrg and

I6 == noTimeout .

ceq sendCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 },M, I3, B4, B5, I6) = send(retrieveSegmentInBuffer(inc inc init-index ,addSegmentToBuffer(inc inc init-index , Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 , urgentptr = B5, data = mess3(M) })), changeState(establishedState,establishedState), checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 })))

if I1 > 0 and

I2 > 0 and

I1 =/= I2 and

M =/= noMessage and

I3 == (byteCount - 2) and

B4 == setPsh and

B5 == noUrg and

I6 == noTimeout .

ceq sendCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 },M, I3, B4, B5, I6) = send(retrieveSegmentInBuffer(inc inc inc init-index ,addSegmentToBuffer(inc inc inc init-index , Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 , urgentptr = B5, data = mess4(M) })), changeState(establishedState,establishedState),

```
checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
foreignSocket = I2 })) )
    if I1 > 0 and
    I2 > 0 and
    I1 =/= I2 and
    M =/= noMessage and
    I3 == ( byteCount - 1 ) and
    B4 == setPsh and
    B5 == noUrg and
    I6 == noTimeout .
ceq sendCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 },M, I3, B4, B5, I6)
= send(retrieveSegmentInBuffer(inc inc inc inc init-index ,addSegmentToBuffer(inc inc inc inc
init-index , Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
urgentptr = B5, data = mess5(M) } )), changeState(establishedState,establishedState),
checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
foreignSocket = I2 })) )
    if I1 > 0 and
    I2 > 0 and
    I1 =/= I2 and
    M =/= noMessage and
    I3 == byteCount and
    B4 == setPsh and
    B5 == noUrg and
    I6 == noTimeout .
ceq curState = establishedState
    if sendCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 },M, I3, B4, B5,
    I6) == send(retrieveSegmentInBuffer(init-index ,addSegmentToBuffer(init-index ,
    Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
    urgentptr = B5, data = mess1(M) } )), changeState(establishedState,establishedState),
    checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
    foreignSocket = I2 })) ) or
    sendCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 },M, I3, B4, B5, I6)
    == send(retrieveSegmentInBuffer(inc init-index ,addSegmentToBuffer(inc init-index ,
    Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
    urgentptr = B5, data = mess2(M) } )), changeState(establishedState,establishedState),
```

```
        checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
        foreignSocket = I2 })) ) or sendCall(LocalConnectionName { localSocket  = I1,
        foreignSocket = I2 },M, I3, B4, B5, I6) ==
        send(retrieveSegmentInBuffer(inc inc init-index ,addSegmentToBuffer(inc inc init-index ,
        Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
        urgentptr = B5, data = mess3(M) } )), changeState(establishedState,establishedState),
        checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
        foreignSocket = I2 })) ) or
        sendCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 },M, I3, B4, B5, I6)
        == send(retrieveSegmentInBuffer(inc inc inc init-index ,addSegmentToBuffer(inc inc inc
        init-index , Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
        urgentptr = B5, data = mess4(M) } )), changeState(establishedState,establishedState),
        checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
        foreignSocket = I2 })) ) or  sendCall(LocalConnectionName { localSocket  = I1,
        foreignSocket = I2 },M, I3, B4, B5, I6) == send(retrieveSegmentInBuffer(inc inc inc inc init-
        index ,addSegmentToBuffer(inc inc inc inc init-index ,
        Segment { sourceport = bind(I1), destport = bind(I2), psh = B4 ,
        urgentptr = B5, data = mess5(M) } )), changeState(establishedState,establishedState),
        checkTcb(init-tcb), checkStatus(statusCall(LocalConnectionName { localSocket  = I1,
        foreignSocket = I2 })) ) .
}
-- ------------------------------------------------------------------------------------------
-- ------------------------------------ RECEIVECALL Module ----------------------------------
-- ------------------------------------------------------------------------------------------
mod* RECEIVECALL{
    pr(SENDCALL)
-- RECEIVE (I1:local connection name, Id:buffer address, I3:byte count, B4:PUSH flag, B5:URGENT
flag)
    op receiveCall : LocalConnectionName Id Int Bool Bool Segment -> Buffersegment  -- method

    vars I1 I2 I3 I4 I6 I8 : Int
    vars S1 S2 S3 S4 : Int
    vars B4 B5 B6 B7 : Bool
    var C : Tcb
    var I : Id
```

```
var M : Message
vars L1 L2 : LocalConnectionName
var S : Segment

ceq receiveCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 } ,I, I3, B4, B5,S)
= no-buffersegment
    if I1 < 1 or
    I2 < 1 or
    I1 == I2 or
    val(I) < 0 or
    ( I3 < 0 or I3 > byteCount ) or
    B4 =/= setPsh or
    B5 =/= noUrg or
    S == no-segment .
ceq receiveCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 }, I, I3, B4, B5, S
)
= addSegmentToBuffer(I, S)
    if I1 > 0 and
    I2 > 0 and
    I1 =/= I2 and
    val(I) >= 0 and
    I3 == byteCount - 4 and
    B4 == setPsh and
    B5 == noUrg and
    S =/= no-segment .
ceq receiveCall(LocalConnectionName { localSocket = I1, foreignSocket = I2 }, I, I3, B4, B5, S
)
= addSegmentToBuffer(inc I, S)
    if I1 > 0 and
    I2 > 0 and
    I1 =/= I2 and
    val(I) >= 0 and
    I3 == byteCount - 3 and
    B4 == setPsh and
    B5 == noUrg and
```

```
        S =/= no-segment .
    ceq receiveCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 }, I, I3, B4, B5, S
)
    = addSegmentToBuffer(inc inc I, S)
        if I1 > 0 and
        I2 > 0 and
        I1 =/= I2 and
        val(I) >= 0 and
        I3 == byteCount - 2 and
        B4 == setPsh and
        B5 == noUrg and
        S =/= no-segment .
    ceq receiveCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 }, I, I3, B4, B5, S
)
    = addSegmentToBuffer(inc inc inc I, S)
        if I1 > 0 and
        I2 > 0 and
        I1 =/= I2 and
        val(I) >= 0 and
        I3 == byteCount - 1 and
        B4 == setPsh and
        B5 == noUrg and
        S =/= no-segment .
    ceq receiveCall(LocalConnectionName { localSocket  = I1, foreignSocket = I2 }, I, I3, B4, B5, S
)
    = addSegmentToBuffer(inc inc inc inc I, S)
        if I1 > 0 and
        I2 > 0 and
        I1 =/= I2 and
        val(I) >= 0 and
        I3 == byteCount and
        B4 == setPsh and
        B5 == noUrg and
        S =/= no-segment .
}
```

```
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- CLOSECALL Module ------------------------------------
-- ------------------------------------------------------------------------------------------
mod* CLOSECALL{
      pr(RECEIVECALL)
-- CLOSE(l1:local connection name)
      op closeCall : LocalConnectionName -> Bool                -- method
      op closeOk : -> Bool
      op closeFail : -> Bool

      var C : Tcb
      var L : LocalConnectionName

      ceq closeCall(L) = closeOk if L =/= errorLocalConnectionName .
      ceq closeCall(L) = closeFail if L == errorLocalConnectionName .
      ceq deleteTcb(C) = no-tcb if closeCall(L) == closeOk .
}
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- ABORTCALL Module ------------------------------------
-- ------------------------------------------------------------------------------------------
mod* ABORTCALL{
      pr(TCB + OPENCALL)
-- ABORT(l1:local connection name)
-- ( l1 == local-connection-name(T))
      op abortCall : LocalConnectionName -> Bool                -- method
      op abortOk : -> Bool
      op abortFail : -> Bool

      var C : Tcb
      var L : LocalConnectionName

      ceq abortCall(L) = abortOk if L =/= errorLocalConnectionName .
      ceq abortCall(L) = abortFail if L == errorLocalConnectionName .
      ceq deleteTcb(C) = no-tcb if abortCall(L) == abortOk .
}
```

```
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- TCP-SYSTEM Module ----------------------------------
-- ------------------------------------------------------------------------------------------
mod* TCP-SYSTEM{
    pr(BUFFER + TCB + STATE + STATUS + OPENCALL + SENDCALL + RECEIVECALL +
CLOSECALL + ABORTCALL)
    *[ System ]*

    op init-system : -> System                          -- initial state
    op no-system : -> System                            -- no state
    op no-segment : -> Segment                          -- no Segment
    op no-buffersegment : -> Buffersegment              -- no Buffersegment
    op checkName : LocalConnectionName -> Bool          -- method
    op openFail : -> Bool
    op openOk : -> Bool
    op errorIp : -> Message

-- get State
    op state-sys : System -> State                      -- projection
-- get IP
    op get-ip1 : System -> Int                          -- attribute
    op get-ip2 : System -> Int                          -- attribute
-- Projection for System
    op sys : Int Int -> System                          -- projection ; Int = Ip
    op buf-sys : Id System -> Buffersegment             -- projection
    op seg-sys : Id System -> Segment                   -- projection
-- open-sys
    op open-active-sys : System -> Bool                 -- method
    op open-passive-sys : System -> Bool                -- method
-- send-sys
    op send-sys : System Message -> Message             -- method
-- receive-sys
    op receive-sys : System Message -> Message          -- method
-- close-sys
    op close-sys : System -> Bool                       -- method
```

```
-- abort-sys
    op abort-sys : System -> Bool                    -- method

    vars T T' : System
    vars IPs IPc IPs' IPc' : Int
    vars I I' : Id
    var B : Buffersegment
    vars B1 B2 B3 B4 B5 : Bool
    var S : Segment
    var D : Int -- data
    var C : Tcb
    vars M M' : Message
    var E : State
    var L : LocalConnectionName

-- State
    ceq state-sys(T) = curState if T == no-system .
-- Segment
    ceq seg-sys(I,T) = no-segment if T == no-system .
    ceq seg-sys(I,T) = Segment { sourceport = get-ip1(T), destport = get-ip2(T) }
        if T =/= init-system .
    ceq seg-sys(I,T) = init-segment
        if I == init-index and T == init-system .
-- Buffersegment
    ceq buf-sys(I,T) = no-buffersegment if T == no-system .
    ceq buf-sys(I,T) = Buffersegment { sourceport = get-ip1(T), destport = get-ip2(T) }
        if T =/= init-system .
    ceq buf-sys(I,T) = init-buffersegment
        if I == init-index and T == init-system .
-- IP
    eq get-ip1(sys(IPs,IPc)) = IPs .
    eq get-ip2(sys(IPs,IPc)) = IPc .
-- Check Open
    ceq checkName(L) = openFail if L == errorLocalConnectionName .
    ceq checkName(L) = openOk if L =/= errorLocalConnectionName .
```

```
-- Open Call : Active
    ceq open-active-sys(T) = checkName(openCall(get-ip1(T), get-ip2(T), active,
    noTimeout, precedenceStatusValue, securityOrCompartmentValue, opt(maximumSegmentSize)
) )
        if ( get-ip1(T) > 0 ) and ( get-ip2(T) > 0 ) and (get-ip1(T) =/= get-ip2(T) ) . -- no error IP
    ceq open-active-sys(T) = openFail
        if ( get-ip1(T) < 1 ) or ( get-ip2(T) < 1 ) or (get-ip1(T) == get-ip2(T) ) .    -- error IP
    ceq state-sys(T) = synsentState
        if open-active-sys(T) == openOk .
    ceq state-sys(T) = closedState
        if open-active-sys(T) == openFail .
-- Open Call : Passive
    ceq open-passive-sys(T) = checkName(openCall(get-ip1(T), get-ip2(T), passive,
    noTimeout, precedenceStatusValue, securityOrCompartmentValue, opt(maximumSegmentSize)
) )
        if ( get-ip1(T) > 0 ) and ( get-ip2(T) > 0 ) and (get-ip1(T) =/= get-ip2(T) ) . -- no error IP
    ceq open-passive-sys(T) = openFail
        if ( get-ip1(T) < 1 ) or ( get-ip2(T) < 1 ) or (get-ip1(T) == get-ip2(T) ) .    -- error IP
    ceq state-sys(T) = listenState
        if open-passive-sys(T) == openOk .
    ceq state-sys(T) = closedState
        if open-passive-sys(T) == openFail .

-- Send Call  and Receive Call
    ceq receive-sys(T',send-sys(T,M)) = noMessage
        if M == noMessage .

    ceq receive-sys(T',send-sys(T,M)) = errorip
        if get-ip1(T) < 1 or
        get-ip2(T) < 1 or
        get-ip1(T') < 1 or
        get-ip2(T') < 1 or
        get-ip1(T) == get-ip2(T) or
        get-ip1(T') == get-ip2(T') or
        get-ip1(T) =/= get-ip2(T') or
```

```
        get-ip1(T') =/= get-ip2(T) .

    ceq receive-sys(T',send-sys(T,M)) = Message {
mess1 = retrieveDataInBufferSegment(init-index, receiveCall(LocalConnectionName{ localSocket =
get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 1, setPsh,noUrg, sendCall(LocalConnectionName
{ localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 1, setPsh, noUrg, noTimeout) ) ) ,
mess2 = retrieveDataInBufferSegment(inc init-index, receiveCall(LocalConnectionName{ localSocket
= get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 2, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 2,
setPsh, noUrg, noTimeout) ) ) ,
mess3 = retrieveDataInBufferSegment(inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 3, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 3,
setPsh, noUrg, noTimeout) ) ) ,
mess4 = retrieveDataInBufferSegment(inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 4, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 4,
setPsh, noUrg, noTimeout) ) ) ,
mess5 = retrieveDataInBufferSegment(inc inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket = get-ip1(T'), foreignSocket = get-ip2(T') }, init-index, 5, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-ip2(T) },M, 5,
setPsh, noUrg, noTimeout) ) ) }
        if M =/= noMessage and
        M =/= errorIp and
        open-active-sys(T) == openOk and
        open-passive-sys(T') == openOk and
        get-ip1(T) == get-ip2(T') and
        get-ip2(T) == get-ip1(T') and
        get-ip1(T) =/= get-ip2(T) and
        get-ip1(T') =/= get-ip2(T') .                    -- correct IP and pair of port

    ceq send-sys(T,M) = errorIp
        if open-active-sys(T) == openFail or
        get-ip1(T) < 1 or
        get-ip2(T) < 1 or
```

```
        ( get-ip1(T) == get-ip2(T) ) .

    ceq send-sys(T,M) = noMessage
        if M == noMessage .

    ceq send-sys(T,M) = Message {
mess1 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-
ip2(T) },M, 1, setPsh, noUrg, noTimeout) ),
mess2 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-
ip2(T) },M, 2, setPsh, noUrg, noTimeout) ),
mess3 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-
ip2(T) },M, 3, setPsh, noUrg, noTimeout) ),
mess4 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-
ip2(T) },M, 4, setPsh, noUrg, noTimeout) ),
mess5 = data(sendCall(LocalConnectionName { localSocket = get-ip1(T), foreignSocket = get-
ip2(T) },M, 5, setPsh, noUrg, noTimeout) ) }
        if open-active-sys(T) == openOk and M =/= noMessage and ( get-ip1(T) > 0 ) and ( get-
        ip2(T) > 0 ) .

    ceq receive-sys(T,M) = noMessage
        if M == noMessage .

    ceq receive-sys(T,M) = errorIp
        if open-passive-sys(T) == openFail or
        get-ip1(T) < 1 or
        get-ip2(T) < 1 or
        get-ip1(T) == get-ip2(T) or
        M == errorIp .

    ceq receive-sys(T,M) = Message {
mess1 = retrieveDataInBufferSegment(init-index, receiveCall(LocalConnectionName{ localSocket =
get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 1, setPsh,noUrg, sendCall(LocalConnectionName
{ localSocket = get-ip2(T), foreignSocket = get-ip1(T) },M, 1, setPsh, noUrg, noTimeout) ) ) ,
mess2 = retrieveDataInBufferSegment(inc init-index, receiveCall(LocalConnectionName{ localSocket
= get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 2, setPsh,noUrg,
```

```
sendCall(LocalConnectionName { localSocket  = get-ip2(T), foreignSocket = get-ip1(T) },M, 2,
setPsh, noUrg, noTimeout) ) ) ,
mess3 = retrieveDataInBufferSegment(inc inc init-index, receiveCall(LocalConnectionName{
localSocket  = get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 3, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket  = get-ip2(T), foreignSocket = get-ip1(T) },M, 3,
setPsh, noUrg, noTimeout) ) ) ,
mess4 = retrieveDataInBufferSegment(inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket  = get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 4, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket  = get-ip2(T), foreignSocket = get-ip1(T) },M, 4,
setPsh, noUrg, noTimeout) ) ) ,
mess5 = retrieveDataInBufferSegment(inc inc inc inc init-index, receiveCall(LocalConnectionName{
localSocket  = get-ip1(T), foreignSocket = get-ip2(T) }, init-index, 5, setPsh,noUrg,
sendCall(LocalConnectionName { localSocket  = get-ip2(T), foreignSocket = get-ip1(T) },M, 5,
setPsh, noUrg, noTimeout) ) )  }
        if open-passive-sys(T) == openOk and
        M =/= noMessage and
        ( get-ip1(T) > 0 ) and
        ( get-ip2(T) > 0 ) and
         get-ip1(T) =/=  get-ip2(T) .

-- Close Call
    ceq close-sys(T) = false
        if T == no-system or
        get-ip1(T) < 1 or
        get-ip2(T) < 1 or
        get-ip1(T) == get-ip2(T) .
    ceq close-sys(T) = true
        if T =/= no-system and
        get-ip1(T) > 0 and
        get-ip2(T) > 0 and
        get-ip1(T) =/= get-ip2(T) .
    ceq closeCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } )
= closeOk
        if close-sys(T) == true .
```

```
    ceq closeCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } )
= closeFail
        if close-sys(T) == false .

-- Abort Call
    ceq abort-sys(T) = false
        if T == no-system or
        get-ip1(T) < 1 or
        get-ip2(T) < 1 or
        get-ip1(T) == get-ip2(T) .
    ceq abort-sys(T) = true
        if T =/= no-system and
        get-ip1(T) > 0 and
        get-ip2(T) > 0 and
        get-ip1(T) =/= get-ip2(T) .
    ceq abortCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } )
= abortOk
        if abort-sys(T) == true .
    ceq abortCall(LocalConnectionName { localSocket  = get-ip1(T), foreignSocket = get-ip2(T) } )
= abortFail
        if abort-sys(T) == false .
}
-- ------------------------------------------------
```

ภาคผนวก ข

การทวนสอบข้อกำหนดทีซีพี

```
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- INDEX Module ---------------------------------------
-- ------------------------------------------------------------------------------------------
-- red val(init-index) .                    -- 0 : Zero
-- red val(inc init-index) .                -- 1 : NzNat
-- red val(dec init-index) .                -- val(dec init-index) : Int
-- red val(X) .                             -- val(X) : Int
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- MESSAGE Module -------------------------------------
-- ------------------------------------------------------------------------------------------
-- red opt(maximumSegmentSize) .            -- 429497296 : NzNat
-- red len(1025) .                          -- 1025 : NzNat
-- red len(429497297) .                     -- ErrorLen : Int
-- red Message { mess1 = 1, mess2 = 2, mess3 = 3, mess4 = 4, mess5 = 5} .
        -- Message { (mess1 = 1 , mess2 = 2 , mess3 = 3 , mess4 = 4 , mess5 = 5) } : Message
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- SEGMENT Module -------------------------------------
-- ------------------------------------------------------------------------------------------
-- red init-segment .  -- Segment { (sourceport = initSouceport , destport = initDestport , seqnumber
= 1 , acknumber = initIrs , offset = 0 , reserved = 0 , urg = resetUrg , ack = resetAck , psh = setPsh ,
rst = resetRst , syn = setSyn , fin = resetFin , window = 1 , checksum = 0 , urgentptr = noUrgentptr ,
options = 2 , padding = 0 , data = noMess)  } : Segment
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- BUFFER Module --------------------------------------
-- ------------------------------------------------------------------------------------------
-- ----- Declaration Value of Buffer -----
-- red init-buffersegment .      -- Buffersegment { (  sourceport = initSouceport , destport =
initDestport , seqnumber = 1 ,
--     acknumber = initIrs , offset = 0 , reserved = 0 , urg = resetUrg ,
--     ack = resetAck , psh = setPsh , rst = resetRst , syn = setSyn , fin = resetFin ,
--     window = 1 , checksum = 0 , urgentptr = noUrgentptr , options = 2 ,
--     padding = 0 , data = noMess)  } : Buffersegment
-- ----- Delete Buffer : no Buffer -----
-- red delBufferSegment(init-index,init-buffersegment) .          -- : no- Buffersegment
```

-- red delBufferSegment(init-index, Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0,

-- reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,

-- checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 }) .

-- : no-: Buffersegment

-- red delBufferSegment(inc init-index, Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0,

-- reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,

-- checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 }) .

-- : no-: Buffersegment

-- red delBufferSegment(X,init-buffersegment) . -- : no- Buffersegment

-- red delBufferSegment(X, Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0, reserved = 0,

-- urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,checksum = 0, urgentptr = 0,

-- options = 0, padding = 0, data = 0 }) . -- : no-: Buffersegment

-- red delBufferSegment(inc X, Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0,

-- reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,

-- checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 }) .

-- : no-: Buffersegment

-- red retrieveDataInBufferSegment(init-index,delBufferSegment(init-index,init-buffersegment)) .

-- : -1 : NzInt

-- ----- Add Buffer if free : Buffer -----

-- red addDataToBufferSegment(init-index, no-buffersegment,Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 }) .

-- : Buffersegment

-- red addDataToBufferSegment(inc init-index, no-buffersegment,Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0, window = 0,checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 }) .

-- : Buffersegment

-- red addDataToBufferSegment(X, no-buffersegment,Buffersegment { sourceport = 5, destport = 1, seqnumber = 1, acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0, window = 0,checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 }) .

-- : Buffersegment

```
-- red addDataToBufferSegment(inc X, no-buffersegment,Buffersegment { sourceport = 5, destport =
1, seqnumber = 1, acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn =
0,fin = 0, window = 0,checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 } ) .
      -- : Buffersegment
-- ----- Add Data from Segment to Buffer : Buffer -----
-- red addSegmentToBuffer(init-index,init-segment) .              -- : Buffersegment
-- red addSegmentToBuffer(init-index,Segment { sourceport = 3, destport = 1, seqnumber = 1,
acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,
checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 } ) .
      -- : Buffersegment
-- red addSegmentToBuffer(inc init-index,Segment { sourceport = 3, destport = 1, seqnumber = 1,
acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,
checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 } ) .
      -- : Buffersegment
-- red addSegmentToBuffer(X,Segment { sourceport = 3, destport = 1, seqnumber = 1, acknumber =
2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,
checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 } ) .
      -- : Buffersegment
-- red addSegmentToBuffer(inc X,Segment { sourceport = 3, destport = 1, seqnumber = 1,
acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window =
0,checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 0 } ) .
      -- : Buffersegment
-- ----- Show Data from Buffer : Int -----
-- red retrieveDataInBufferSegment(init-index,init-buffersegment) .          -- 0 : Zero
-- red retrieveDataInBufferSegment(X,init-buffersegment) .                   -- 0 : Zero
-- red retrieveDataInBufferSegment(X,Buffersegment { sourceport = 3, destport = 1, seqnumber = 1,
acknumber = 2,offset = 0, reserved = 0, urg = 0,ack = 0,psh = 0,rst = 0,syn = 0,fin = 0,window = 0,
checksum = 0, urgentptr = 0, options = 0, padding = 0, data = 555 } ) .
         -- 555 : NzNat
-- ----- Show Data after sending Segment to  Buffer : Int -----
-- red retrieveDataInBufferSegment(init-index,   addSegmentToBuffer(init-index, init-segment ) ) .
         -- noMess : Int
-- red retrieveDataInBufferSegment(X,     addSegmentToBuffer(X, init-segment) ) .
         -- noMess : Int
-- red retrieveDataInBufferSegment(X,     addSegmentToBuffer(X, Segment{ data = 666 } ) ) .
```

```
        -- 666 : NzNat
-- ----- Add Segment to Buffersegment and then Delete it : no-Buffersegment -----
-- red delBufferSegment(init-index, addSegmentToBuffer(init-index,init-segment) ) .
        -- no-Buffersegment: Buffersegment
-- red delBufferSegment(X, addSegmentToBuffer(X,init-segment) ) .
        -- no-Buffersegment: Buffersegment
-- red delBufferSegment(X, addSegmentToBuffer(X,Segment{ data = 666}) ) .
        -- no-Buffersegment: Buffersegment
-- ----- Del Bufferegment and then retrieve it's data : -1 -----
-- red retrieveDataInBufferSegment(init-index, delBufferSegment(init-index,init-buffersegment) ) .
-- -1 : NzInt
-- red retrieveDataInBufferSegment(X, delBufferSegment(X,init-buffersegment) ) .
        -- -1 : NzInt
-- red retrieveDataInBufferSegment(X, delBufferSegment(X,Buffersegment{ data = 666 }) ) .
        -- -1 : NzInt
-- ----  Del Bufferegment and then retrieve it : no-segment -----
-- red retrieveSegmentInBuffer(init-index, delBufferSegment(init-index,init-buffersegment) ) .
        -- no-segment : Segment
-- red retrieveSegmentInBuffer(X, delBufferSegment(X,init-buffersegment) ) .
        -- no-segment : Segment
-- red retrieveSegmentInBuffer(X, delBufferSegment(X,Buffersegment{ data = 666 }) ) .
        -- no-segment : Segment
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- STATE Module -------------------------------------
-- ------------------------------------------------------------------------------------------
-- ----- Declaration Value of State : Must be 1 -----
-- red valueState(closedState) .                    -- 1 : NzNat
-- ----- Checking changing State : Must be true and false -----
-- red changeState(closedState,listenState) .       -- true : Bool
-- red changeState(closedState,lastackState) .      -- false : Bool
-- ----- Checking value of State : Must be 2 -----
-- red valueState(listenState) .                    -- 2 : NzNat
-- ------------------------------------------------------------------------------------------
-- ------------------------------------- TCB Module ---------------------------------------
-- ------------------------------------------------------------------------------------------
```

```
-- red init-tcb .
-- red iss(init-tcb) .                          -- initIss : Int
-- red createTcb .                              -- init-tcb : Tcb
-- red deleteTcb(init-tcb) .                    -- no-tcb : Tcb
-- red checkTcb(createTcb) .                    -- true : Bool
-- red checkTcb(deleteTcb(init-tcb)) .          -- true : Bool
-- red checkTcb(Tcb{ iss = -1 }) .              -- false : Bool
-- ------------------------------------------------------------------------------
-- ---------------------------------- STATUSCALL Module ----------------------------
-- ------------------------------------------------------------------------------
-- red checkStatus(init-status) .                       -- true : Bool
-- red checkStatus(statusCall(LocalConnectionName { localSocket = 1, foreignSocket = 2 })) .
     -- true : Bool
-- red checkStatus(statusCall(LocalConnectionName { localSocket = -1, foreignSocket = 2 })) .
     -- false : Bool
-- ------------------------------------------------------------------------------
-- ---------------------------------- SEND Module ----------------------------------
-- ------------------------------------------------------------------------------
-- red send(Segment { window = noWindow },true,true,true) .        -- no-segment : Segment
-- red send(Segment { psh = resetPsh },true,true,true) .           -- no-segment : Segment
-- red send(init-segment,true,true,true) .                         -- init-segment : Segment
-- ----- Checking value of send Syn -----
-- red send(Segment { seqnumber = 10, acknumber = 20, syn = setSyn, psh = setPsh, window =
setWindow },true,true,true) .
          -- Segment { (acknumber = 11) } : Segment
-- ----- Checking value of send Ack -----
-- red send(Segment { seqnumber = 10, acknumber = 20, ack = setAck, psh = setPsh, window =
setWindow },true,true,true) .
          -- Segment { (seqnumber = 20) } : Segment
-- ----- Checking value of send Ack, Fin -----
-- red send(Segment { seqnumber = 10, acknumber = 20, ack = setAck, fin = setFin, psh = setPsh,
window = setWindow },true,true,true) .
          -- Segment { (seqnumber = 20 , acknumber = 11) } : Segment
-- ----- Checking value of send Syn, Ack -----
```

```
-- red send(Segment { seqnumber = 10, acknumber = 20, ack = setAck, syn = setSyn, psh = setPsh,
window = setWindow },true,true,true) .
        -- Segment { (seqnumber = 20 , acknumber = 11) } : Segment
-- ----- Connection Established -----
-- red send(no-segment, changeState(closedState,listenState), checkTcb(createTcb),
checkStatus(init-status)) .
-- red send( no-segment, changeState(closedState,synsentState), checkTcb(createTcb),
checkStatus(init-status)) .
-- red send( no-segment, changeState(listenState,synrcvdState), checkTcb(C), checkStatus(U)) .
-- red send( Segment { seqnumber = J, syn = setSyn, psh = setPsh, window = setWindow },
changeState(synsentState,synrcvdState), checkTcb(C), checkStatus(U)) .
-- red send( S, changeState(synsentState,establishedState), checkTcb(C), checkStatus(U)) .
-- red send( Segment { seqnumber = K, acknumber = J + 1, ack = setAck, syn = setSyn, psh =
setPsh, window = setWindow }, changeState(synrcvdState,establishedState), checkTcb(C),
checkStatus(U)) .
-- red send( S, changeState(synrcvdState,establishedState), checkTcb(C), checkStatus(U)) .
-- red send(Segment { seqnumber = K + 1, ack = setAck, psh = setPsh, window = setWindow },
changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)) .
-- ----- Data Transfer -----
-- red retrieveDataInBufferSegment(init-index, addSegmentToBuffer(init-index, send(Segment {
sourceport = 1, destport = 2, seqnumber = 1, ack = setAck, psh = setPsh, window = 4096, data =
100 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .
-- red retrieveDataInBufferSegment(inc init-index, addSegmentToBuffer(inc init-index, send(Segment
{ sourceport = 1, destport = 2, seqnumber = 1025, ack = setAck, psh = setPsh, window = 4096, data
= 200 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .
-- red retrieveDataInBufferSegment(inc inc init-index, addSegmentToBuffer(inc inc init-index,
send(Segment { sourceport = 1, destport = 2, seqnumber = 2049, ack = setAck, psh = setPsh,
window = 4096, data = 300 }, changeState(establishedState,establishedState), checkTcb(C),
checkStatus(U)))) .
-- red retrieveDataInBufferSegment(init-index, addSegmentToBuffer(init-index, send(Segment {
sourceport = 2, destport = 1, seqnumber = 2049, ack = setAck, psh = setPsh, window = 4096, data
= 200 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .
-- red retrieveDataInBufferSegment(inc init-index, addSegmentToBuffer(inc init-index, send(Segment
{ sourceport = 2, destport = 1, seqnumber = 3073, ack = setAck, psh = setPsh, window = 4096, data
= 300 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .
```

-- red retrieveDataInBufferSegment(inc inc inc init-index, addSegmentToBuffer(inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 3037, ack = setAck, psh = setPsh, window = 4096, data = 400 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc init-index, addSegmentToBuffer(inc inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 4097, ack = setAck, psh = setPsh, window = 4096, data = 400 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 4097, ack = setAck, psh = setPsh, window = 4096, data = 500 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 5121, ack = setAck, psh = setPsh, window = 4096, data = 600 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 6145, ack = setAck, psh = setPsh, window = 4096, data = 700 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc inc init-index, addSegmentToBuffer(inc inc inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 6145, ack = setAck, psh = setPsh, window = 4096, data = 600 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc inc inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc inc inc inc init-index, send(Segment { sourceport = 1, destport = 2, seqnumber = 7169, ack = setAck, psh = setPsh, window = 4096, data = 800 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- red retrieveDataInBufferSegment(inc inc inc inc init-index, addSegmentToBuffer(inc inc inc inc init-index, send(Segment { sourceport = 2, destport = 1, seqnumber = 8193, ack = setAck, psh = setPsh, window = 4096, data = 800 }, changeState(establishedState,establishedState), checkTcb(C), checkStatus(U)))) .

-- ----- Termination -----

-- red send(S, changeState(establishedState,finwait-1State), checkTcb(C), checkStatus(U)) .

-- red send(S, changeState(establishedState,closewaitState), checkTcb(C), checkStatus(U)) .

-- red send(Segment { seqnumber = M, fin = setFin, psh = setPsh, window = setWindow },
changeState(establishedState,finwait-1State), checkTcb(C), checkStatus(U)) .

--red send(S, changeState(establishedState,closewaitState), checkTcb(C), checkStatus(U)) .

--red send(Segment { seqnumber = M + 1, ack = setAck, psh = setPsh, window = setWindow },
changeState(timewaitState,finwait-2State), checkTcb(C), checkStatus(U)) .

--red send(S, changeState(closewaitState,lastackState), checkTcb(C), checkStatus(U)) .

--red send(S, changeState(finwait-2State,timewaitState), checkTcb(C), checkStatus(U)) .

--red send(Segment { seqnumber = N, fin = setFin, psh = setPsh, window = setWindow },
changeState(lastackState,timewaitState), checkTcb(C), checkStatus(U)) .

--red send(S, changeState(lastackState,closedState), checkTcb(C), checkStatus(U)) .

--red send(S, changeState(timewaitState,closedState), checkTcb(C), checkStatus(U)) .

-- red send(Segment { seqnumber = N + 1, ack = setAck, psh = setPsh, window = setWindow },
changeState(timewaitState,closedState), checkTcb(C), checkStatus(U)) .

-- ----------

-- ---------- OPENCALL Module ----------

-- ----------

-- red openCall(-1, 2, active, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize)) .

-- errorLocalConnectionName : LocalConnectionName

-- red openCall(1, -2, active, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize)) .

-- errorLocalConnectionName : LocalConnectionName

-- red openCall(1, 1, active, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize)) .

-- errorLocalConnectionName : LocalConnectionName

-- red openCall(2, 3, active, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize)) .

-- LocalConnectionName { (localSocket = 2 , foreignSocket = 3) } : LocalConnectionName

-- red openCall(2, 3, passive, noTimeout, precedenceStatusValue, securityOrCompartmentValue,
opt(maximumSegmentSize)) .

-- LocalConnectionName { (localSocket = 2 , foreignSocket = 3) } : LocalConnectionName

-- red openCall(sourceport(Segment { sourceport = 2}), destport(Segment { destport = 3}), active,
noTimeout, precedenceStatusValue, securityOrCompartmentValue, opt(maximumSegmentSize)) .

-- LocalConnectionName { (localSocket = 2 , foreignSocket = 3) } : LocalConnectionName

```
-- red openCall(sourceport(Segment { sourceport = -2}), destport(Segment { destport = 3}), active,
noTimeout, precedenceStatusValue, securityOrCompartmentValue, opt(maximumSegmentSize) ) .
        -- errorLocalConnectionName : LocalConnectionName
-- ------------------------------------------------------------------------------------------
-- ---------------------------------- SENDCALL Module ----------------------------------
-- ------------------------------------------------------------------------------------------
-- red changeState(establishedState,establishedState) .
-- red checkTcb(init-tcb) .
-- red checkStatus(statusCall(LocalConnectionName { localSocket =
localSocket(LocalConnectionName {localSocket = 4, foreignSocket = 6 }),    foreignSocket =
foreignSocket(LocalConnectionName {    localSocket = 4, foreignSocket = 6 }) } ) ) .
-- ----- Checking value of <add segment to buffer> then <retrieve it> to <send> through another TCP
-- red send(retrieveSegmentInBuffer(init-index,addSegmentToBuffer(init-index, Segment { sourceport
= bind(localSocket(LocalConnectionName {    localSocket = 4, foreignSocket = 6 })), destport =
bind(foreignSocket(LocalConnectionName {    localSocket = 4, foreignSocket = 6 })), psh = setPsh,
urgentptr = noUrg, data = mess1(Message { mess1 = 3}) } )),
changeState(establishedState,establishedState), checkTcb(init-tcb),
checkStatus(statusCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 })) ) .
-- red send(retrieveSegmentInBuffer(init-index,addSegmentToBuffer(init-index, Segment { sourceport
= bind(localSocket(LocalConnectionName {    localSocket = 4, foreignSocket = 6 })), destport =
bind(foreignSocket(LocalConnectionName {    localSocket = 4, foreignSocket = 6 })), psh = setPsh,
urgentptr = noUrg, data = mess1(Message { mess1 = 3}) } )),
changeState(establishedState,establishedState), checkTcb(init-tcb),
checkStatus(statusCall(LocalConnectionN ame { localSocket = 4, foreignSocket = 6 })) ) .
    -- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 3)  } :
    Segment
-- ----- Checking value of sendCall -----
-- red sendCall(LocalConnectionName {   localSocket = 4, foreignSocket = 6 },Message { mess1 = 3
}, byteCount - 4, setPsh, noUrg, noTimeout) .
    -- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 3)  } :
    Segment
-- ----- Checking value of sendCall : Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 =
444, mess5 = 555 } -----
-- ----- Checking value of sendCall : Message Segment 1 -----
-- red sendCall(LocalConnectionName {   localSocket = 4, foreignSocket = 6 },
```

Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 1, setPsh, noUrg, noTimeout) .

-- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 111) } : Segment

-- ----- Checking value of sendCall : Message Segment 2 -----

-- red sendCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 2, setPsh, noUrg, noTimeout) .

-- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 222) } : Segment

-- ----- Checking value of sendCall : Message Segment 3 -----

-- red sendCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 3, setPsh, noUrg, noTimeout) .

-- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 333) } : Segment

-- ----- Checking value of sendCall : Message Segment 4 -----

-- red sendCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 4, setPsh, noUrg, noTimeout) .

-- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 444) } : Segment

-- ----- Checking value of sendCall : Message Segment 5 -----

-- red sendCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 5, setPsh, noUrg, noTimeout) .

-- Segment { (sourceport = 4 , destport = 6 , psh = setPsh , urgentptr = noUrg , data = 555) } : Segment

-- ----- Checking value of error socket -----

-- red sendCall(LocalConnectionName { localSocket = -4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 5, setPsh, noUrg, noTimeout) .

-- no-segment : Segment

-- ----------

-- ---------- RECEIVECALL Module ----------

-- ----------

mod* RECEIVECALL{

pr(SENDCALL)

-- RECEIVE (I1:local connection name, Id:buffer address, I3:byte count, B4:PUSH flag, B5:URGENT flag)

-- red receiveCall(LocalConnectionName { localSocket = 6, foreignSocket = 4 }, init-index,1,setPsh,noUrg, sendCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 }, noMessage , 1, setPsh, noUrg, noTimeout)) .

-- noMessage : Message

-- red receiveCall(LocalConnectionName{ localSocket = -6, foreignSocket = 4 }, init-index,1,setPsh,noUrg, sendCall(LocalConnectionName { localSocket = 4, foreignSocket = 6 },Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 1, setPsh, noUrg, noTimeout)) .

-- noMessage : Message

-- ----- Checking value of receiveCall : Buffersegment 1 -----

-- red receiveCall(LocalConnectionName{ localSocket = 6, foreignSocket = 4 }, init-index, 1, setPsh,noUrg, sendCall(LocalConnectionName{ localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 1, setPsh, noUrg, noTimeout)) .

-- init-index,data = 111 : Buffersegment

-- ----- Checking value of receiveCall : Buffersegment 2 -----

-- red receiveCall(LocalConnectionName{ localSocket = 6, foreignSocket = 4 }, init-index, 2, setPsh,noUrg, sendCall(LocalConnectionName{ localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 2, setPsh, noUrg, noTimeout)) .

-- inc init-index, data = 222 : Buffersegment

-- ----- Checking value of receiveCall : Buffersegment 3 -----

-- red receiveCall(LocalConnectionName{ localSocket = 6, foreignSocket = 4 }, init-index, 3, setPsh,noUrg, sendCall(LocalConnectionName{ localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 3, setPsh, noUrg, noTimeout)) .

-- inc (inc init-index), data = 333 : Buffersegment

-- ----- Checking value of receiveCall : Buffersegment 4 -----

-- red receiveCall(LocalConnectionName{ localSocket = 6, foreignSocket = 4 }, init-index, 4, setPsh,noUrg, sendCall(LocalConnectionName{ localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 3, setPsh, noUrg, noTimeout)) .

-- inc (inc (inc init-index)), data = 444 : Buffersegment

-- ----- Checking value of receiveCall : Buffersegment 5 -----

-- red receiveCall(LocalConnectionName{ localSocket = 6, foreignSocket = 4 }, init-index, 5, setPsh,noUrg, sendCall(LocalConnectionName{ localSocket = 4, foreignSocket = 6 }, Message { mess1 = 111, mess2 = 222, mess3 = 333, mess4 = 444, mess5 = 555 } , 5, setPsh, noUrg, noTimeout)) .

-- inc (inc (inc (inc init-index))),data = 555 : Buffersegment

-- ----------

-- ---------- CLOSECALL Module ----------

-- ----------

-- red closeCall(errorLocalConnectionName) .

-- red closeCall(L:LocalConnectionName) .

-- ----------

-- ---------- ABORTCALL Module ----------

-- ----------

-- red abortCall(errorLocalConnectionName) .

-- red abortCall(L:LocalConnectionName) .

-- ----------

-- ---------- TCP-SYSTEM Module ----------

-- ----------

-- ----- Checking for each component in TCP-SYSTEM -----

-- ----- Open Call -----

-- red open-active-sys(sys(-2,4)) . -- openFail : Bool

-- red open-active-sys(sys(2,-4)) . -- openFail : Bool

-- red open-active-sys(sys(2,4)) . -- openOk : Bool

-- red open-active-sys(sys(2,4)) . -- openOk : Bool

-- red open-passive-sys(sys(-2,4)) . -- openFail : Bool

-- red open-passive-sys(sys(2,-4)) . -- openFail : Bool

-- red open-passive-sys(sys(2,4)) . -- openOk : Bool

-- ----- seg-sys -----

-- red seg-sys(init-index,init-system) .

-- Segment { (sourceport = initSouceport , destport = initDestport , seqnumber = 1 , acknumber = initIrs , offset =0 , reserved =0 , urg =resetUrg , ack =resetAck , psh =setPsh , rst =resetRst , syn =setSyn , fin =resetFin , window =1 , checksum =0 , urgentptr =noUrgentptr , options =2 , padding =0 , data =noMess) } : Segment

-- red seg-sys(init-index,sys(2,4)) . -- Segment { (sourceport = 2 , destport = 4) } : Segment

-- red seg-sys(I,no-system) . -- no-segment : Segment

```
-- ----- buf-sys -----
-- red buf-sys(init-index,init-system) .
      -- Buffersegment { (sourceport = initSouceport , destport = initDestport , seqnumber = 1 ,
acknumber = initIrs , offset =0 , reserved =0 , urg =resetUrg , ack =resetAck , psh =setPsh , rst
=resetRst , syn =setSyn , fin =resetFin , window =1 , checksum =0 , urgentptr =noUrgentptr , options
=2 , padding =0 , data =noMess)  } : Buffersegment
-- red buf-sys(init-index,sys(2,4)) .-- Buffersegment { (sourceport = 2 , destport = 4) } : Buffersegment
-- red buf-sys(I,no-system) .                -- no-buffersegment : Buffersegment
-- ------------------------------------------------------
open TCP-SYSTEM

ops m1 m2 : -> Message .
ops datasegment1 datasegment2 datasegment3 datasegment4 datasegment5 : -> Int .
ops server client client1 errorSys self : -> System .
ops ips ipc ipc1 errorip : -> Int .

eq m1 = Message { mess1 = datasegment1, mess2 = datasegment2, mess3 = datasegment3,
mess4 = datasegment4, mess5 = datasegment5 } .
eq m2 = Message { mess1 = datasegment1, mess2 = noMess, mess3 = noMess,
mess4 = noMess, mess5 = noMess } .
eq ips = 111 .
eq ipc = 222 .
eq ipc1 = 333 .
eq errorip = -1 .
eq server = sys(ips,ipc) .
eq client = sys(ipc,ips) .
eq client1 = sys(ipc1,ips) .
eq errorSys = sys(errorip,errorip) .
eq self = sys(ips,ips) .
-- ---------- Active Open ----------
-- red open-active-sys(client) .          -- openOk : Bool
-- red open-active-sys(errorSys) .        -- openFail : Bool
-- red open-active-sys(self) .            -- openFail : Bool

-- ---------- Passive Open ----------
```

```
-- red open-passive-sys(server) .          -- openOk : Bool
-- red open-passive-sys(errorSys) .        -- openFail : Bool
-- red open-passive-sys(self) .            -- openFail : Bool

-- ---------- Send Sys ----------
-- red send-sys(server, m1) .
     -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
     -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red send-sys(client, m1) .
     -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
     -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red send-sys(errorSys, m1) .                -- errorIp : Message
-- red send-sys(self, m1) .                    -- errorIp : Message
-- red send-sys(client, noMessage) .           -- noMessage : Message
-- red send-sys(server, noMessage) .           -- noMessage : Message
-- red send-sys(self, noMessage) .             -- errorIp : Message
-- red send-sys(errorSys, noMessage) .         -- errorIp : Message

-- ---------- Receive Sys ----------
-- red receive-sys(server, m1) .
     -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
     -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red receive-sys(client, m1) .
     -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
     -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red receive-sys(errorSys, m1) .             -- errorIp : Message
-- red receive-sys(self, m1) .                 -- errorIp : Message
-- red receive-sys(client, noMessage) .        -- noMessage : Message
-- red receive-sys(server, noMessage) .        -- noMessage : Message
-- red receive-sys(errorSys, noMessage) .      -- errorIp : Message
-- red receive-sys(self, noMessage) .          -- noMessage : Message

-- ---------- Send and Receive Sys ----------
-- red receive-sys(server, send-sys(client, m1)) .
     -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
```

```
        -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red receive-sys(server, send-sys(client, m2)) .
        -- Message { (mess1 = datasegment1 , mess2 = noMess , mess3 = noMess ,
        -- mess4 = noMess , mess5 = noMess)  } : Message
-- red receive-sys(client, send-sys(server, m1)) .
        -- Message { (mess1 = datasegment1 , mess2 = datasegment2 , mess3 = datasegment3 ,
        -- mess4 = datasegment4 , mess5 = datasegment5)  } : Message
-- red receive-sys(client, send-sys(server, m2)) .
        -- Message { (mess1 = datasegment1 , mess2 = noMess , mess3 = noMess ,
        -- mess4 = noMess , mess5 = noMess)  } : Message
--  ---------- ErrorSys ----------
-- red receive-sys(server, send-sys(errorSys, m1)) .          -- errorIp : Message
-- red receive-sys(errorSys, send-sys(client, m1)) .          -- errorIp : Message
-- red receive-sys(errorSys, send-sys(errorSys, m1)) .        -- errorIp : Message
-- red receive-sys(self, send-sys(server, m1)) .              -- errorIp : Message
-- red receive-sys(server, send-sys(self, m1)) .              -- errorIp : Message
-- red receive-sys(self, send-sys(self, m1)) .                -- errorIp : Message

--  ---------- no Message ----------
-- red receive-sys(server, send-sys(client, noMessage)) .     -- noMessage : Message
-- red receive-sys(client, send-sys(server, noMessage)) .     -- noMessage : Message
-- red receive-sys(client, send-sys(client, noMessage)) .     -- noMessage : Message
-- red receive-sys(server, send-sys(server, noMessage)) .     -- noMessage : Message

-- ---------- Close Sys ----------
-- red close-sys(server) .                -- true : Bool
-- red close-sys(errorSys) .              -- false : Bool
-- red close-sys(self) .                  -- false : Bool

-- ---------- Abort Sys ----------
-- red abort-sys(client) .                -- true : Bool
-- red abort-sys(errorSys) .              -- false : Bool
-- red abort-sys(self) .                  -- false : Bool
close .
eof
```

ภาคผนวก ค
ผลงานตีพิมพ์ในงาน IconIT'2001

A FORMAL SPECIFICATION TECHNIQUE FOR THE TCP NORMAL CONNECTION ESTABLISHMENT AND TERMINATION

Suchada Junchan and Wanchai Rivepiboon

Department of Computer Engineering, Faculty of Engineering,
Chulalongkorn University, Bangkok 10330 Thailand
42706863@student.chula.ac.th and wanchai.r@chula.ac.th

ABSTRACT

In this paper, we propose a new technique for specifying and verifying the formal specification. This method synthesizes the TCP Normal Connection Establishment and Termination specification using the CafeOBJ specification language from the TCP State Transition diagram and the Time Line. As a feature of CafeOBJ, the specification is described as a behavioral abstraction. The procedure of this specification is analyzed by using the TCP State Transition diagram and the Time Line. In addition, the specification of the TCP encapsulation and decapsulation is provided. The result shows that the specification is both clear and precise; in addition it is valid for the satisfied operation. We conclude that this approach is effective and appropriate for the formal specification of the TCP Normal Connection Establishment and Termination.

INTRODUCTION

Formal software specifications are specifications expressed in a language in which vocabulary, syntax and semantic are formally defined. Their consistency and completeness are mathematically proven [Gaudel, 1994]. Moreover, they are also automatically processed and used as a guide to the tester of a component in identifying appropriate test cases [Pressman, 1997]. Due to the fact that user requirements need to be precise and unambiguous, the formal specification methods must be employed in the software design. For example, communication protocols are complex so specifying and verifying them require a clear and concise appropriate specification. Furthermore, the protocols specification should support the verification of protocol properties and proof of their correctness.

Over the last decade, there has been a growing research interest in the area of protocol specification and verification. The techniques for specification and verification of computer network protocols have progressed significantly. Several approaches have been presented for specifying and verifying data communication protocols [Lai, 1995], [Zhang, Takahashi, Shiratori and Noguchi, 1988], [Chu and Liu, 1988], [Hinchey and Jarvis, 1993], [Nakata, Higashino and Taniguchi, 1995], [Park and Miller, 1997]. In protocols [Lai, 1995], [Zhang, Takahashi, Shiratori and Noguchi, 1988], [Chu and Liu, 1988], the techniques based on the State Transition diagram were proposed; however, these methods are not sufficient for specifying the system procedure. Recently, [Hinchey and Jarvis, 1993], [Nakata, Higashino and Taniguchi, 1995], [Park and Miller, 1997] proposed to use the techniques based on time, but such methods cannot specify the system abstraction.

This paper presents a new formal technique based on the TCP State Transition diagram and the Time Line. This new method use the CafeOBJ specification

language, based on the behavioral abstraction, to specify the TCP Normal Connection Establishment and Termination. In addition, the specification of the TCP encapsulation and decapsulation is also provided.

The paper is organized as follows. First, the necessary background is given. Then, the review of literature is described. Finally, our methodology is introduced.

BACKGROUND

◆ CAFEOBJ OVERVIEW

CafeOBJ [Futatsugi and Nakagawa, 1997], [Nakajima and Futatsigi, 1997], [Nakagawa, Sawada and Futatsugi, 1999], [Diaconescu, Futatsugi and Iida, 1999] is a successor of the OBJ specification language adopting the algebraic specification paradigm. In brief, it has important new features to take into account concurrency and the behavioral specifications. Concurrency and behavior are respectively based on rewriting logic and behavioral algebra. The CafeOBJ methodology introduces a graphical notation extending the classical ADJ diagram notation for the data types to identify the behavioral specification.

The main features of CafeOBJ reflect its logical semantics as follows:

- *Equational specification and programming*: CafeOBJ is executable, which gives a refined declarative way of the functional programming.
- *Concurrent systems specification*: This is based on the rewriting logic specification framework for concurrent systems.
- *Behavioral specification*: This represents how objects and systems behave, not how they are implemented. Hidden sort represents the behavioral concept of satisfaction based on the idea of state indistinguishability.
- *Object orientation*: In CafeOBJ, there are two origins of object-orientation. Firstly, the rewriting logic represents implementation oriented. Secondly, the behavioral specification is truthful of the principle of state encapsulation.
- *Powerful module system*: The principles of the CafeOBJ module system are inherited from OBJ. CafeOBJ has several kinds of imports, the parameterized programmings, views and module expressions.
- *Powerful type system*: Order-sorted algebra is granted the system subtypes to increase expressiveness, to provide a rigorous framework for multiple data representations and to automate coercion among them.

◆ CAFEOBJ SYNTAX

In CafeOBJ, a module plays two different roles, depending on where it appears and what it contains [Nakagawa, Sawada and Futatsugi, 1999]. When a module is declared with *module**, it always denotes a class of models. When a module is declared with *module!* or *module*, it always denotes a unique model, such as the module *SEND*:

```
module SEND{
        protecting (INITIAL)
    op s5tos4 : SData -> STcp
    op s4tos3 : STcp -> SIp
    op s3tos2 : SIp -> SDlink
    op s2tos1 : SDlink -> SPhy
    op s1toc1 : SPhy -> CPhy
    op send : SData -> CData
    op c1toc2 : CPhy -> CDlink
    op c2toc3 : CDlink -> CIp
    op c3toc4 : CIp -> CTcp
    op c4toc5 : CTcp -> CData
    var S : SData
    var C : CData
--> ------ SEND
    eq c4toc5(c3toc4(c2toc3(c1toc2(s1toc1(s2tos1(s3tos2(s4tos3(s5tos4(S)))))))))) = C .
}
```

The detail of each part is described below.

- *Import declaration.* Module elements can refer to other modules. CafeOBJ has three kinds of import declarations, which are called *protecting, extending* and *using*, e.g.

protecting (INITIAL)

  - The command *protecting* is the strongest: it requires all the intended modules marked as *protecting* (here *INITIAL*) to be preserved as they are.
  - The command *extending* allows its module(s) to be inflated, but not to be collapsed.
  - The command *using* allows total destruction.

- *Sort declaration.* A sort is a set of classified elements. There are two kinds of sorts in CafeOBJ, which are called *visible sorts* and *hidden sorts*, e.g. *[NAT]* and **[Bool]** respectively.
- *Operation declaration.* CafeOBJ allows users to employ various notations in writing terms. There are infix, prefix and postfix operations (e.g., *op _+_ : Nat Nat -> Nat*). If any preference is not shown, get a standard notation for function application (e.g., *op synsent-synrcvd : CData -> SData).*
- *Behavioral operation declaration.* Behavioral operators use for specifying a special kind of operators, which are hidden sorts such as *bop check : -> Bool .*
- *Variable declaration.* In CafeOBJ, each variable belongs to a sort and represents an arbitrary term of that sort, e.g. *var C : CData .*
- *Equation declaration.* A plain equation is an axiom. Anticipating conditional cases are called as *unconditional*, such as

  eq c4toc5(c3toc4(c2toc3(c1toc2(s1toc1(s2tos1(s3tos2(s4tos3(s5tos4(S)))))))))) = C .

- PROTOCOL LAYERING

TCP/IP [Washburn and Evans, 1993], [Information Sciences Institute University of Southern California, 1981] is a five-layered model (figure 1). Layer 1 and 2, the Physical Layer and the Data Link Layer, are not actually defined by the TCP/IP RFCs, as TCP/IP was designed to be independent for the Physical media. Layer 3, the Internet Protocol Layer, provides a basic datagram service. Layer 4, the Transport Layer, provides the message sending and the connection closing to each node, this take time. And layer 5, the Application Layer, provides suitable services for the different types of application that wish to use the network.

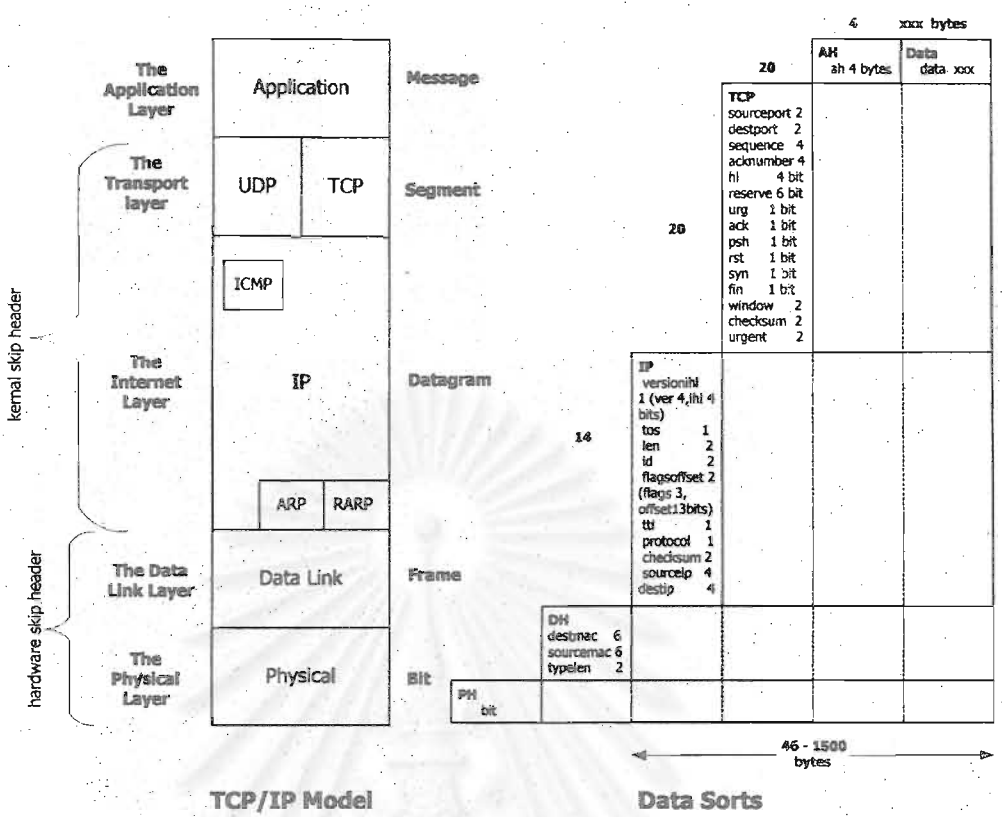


Figure 1. TCP/IP Model and Data Sorts of TCP

LITERATURE REVIEW

Lai [Lai, 1995] presented a case study using the State Transition technique, the numerical Petri nets (NPN) and the automated tool (PROTEAN) to specify and verify the procedural protocol. He showed that it was very useful for the formal specification and verification of state event protocols. Although this technique was very advantage, it was inadequate for specifying the procedural protocol such as Job Transfer Manipulating (JTM). This method could not hold the parameters associated with the primitives that required system processing.

Hinchey and Jarvis [Hinchey and Jarvis, 1993] determined the need of appropriate methods for specification and verification of the data communication protocols. Their approach was based on the combination of Communicating Sequential Processes (CSP) and Timed CSP process algebras. They claimed that any protocol (at any level in the OSI stack) could gain the benefit from their approach. The system consisted of processes and environment. This method combined the behavior of the sender and the receiver processes, but the system abstraction was ignored.

Layuan [Layuan, 1989] presented the new formal method for the communication protocol specification. His new method included Finite State Machine (FSM), CSP and Abstract Data Type (ADT) for designing the specification. The best features of these approaches for the protocol specification were determined. The result showed the effectiveness and suitableness of communication protocol specification. However, this method was limited in reachability analysis.

METHODOLOGY

In this section, the detail of the TCP Normal Connection Establishment and Termination [Comer and Stevens, 1996], [Stevens, 1995], [Stevens, 1991] using the TCP State Transition diagram and the Time Line is proposed. In fact, the object-oriented CafeOBJ methodology is based on the behavioral abstraction. Furthermore, the behavioral specification is more faithful principle of the state encapsulation. Hence, we represent the sort in CafeOBJ for encapsulating and decapsulating TCP/IP header as shown in figure 1.

In CafeOBJ, the record inheritance is used for identifying the frame building of TCP/IP header [Washburn and Evans, 1993]. We design records SData, STcp, SIp, SDlink and SPhy which represent the server data in the Application Layer, the Transport Layer, the Internet Layer, the Data Link Layer and the Physical Layer respectively (see figure 2). In the same manner, we design records CData, CTcp, CIp, CDlink and CPhy for the client data in the Application Layer, the Transport Layer, the Internet Layer, the Data Link Layer and the Physical Layer respectively.

The behavioral specification may be the most distinctive feature of CafeOBJ within a family of the algebraic specification languages. The basic behavioral specification is the simplest level of the behavioral specification operations. With this feature of CafeOBJ, we observe the behavior of the TCP/IP encapsulation and decapsulation by using the operation.

When each layer of TCP/IP encapsulates or decapsulates data, we represent operation, such as the server encapsulates in the Application layer to the Transport layer which is represented by *op s5tos4 : SData -> STcp* (see figure 2). This means that the server encapsulates (SData = AH + Data) from the Application layer to the Transport layer (STcp = TCP + AH + Data).

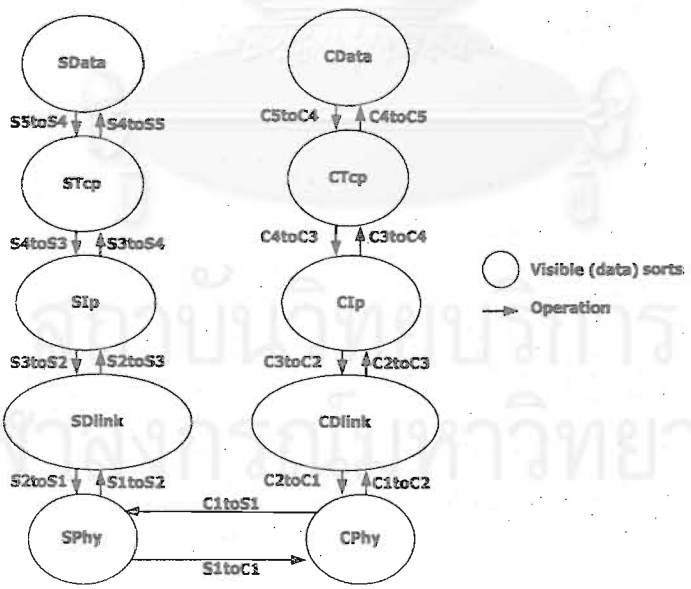


Figure 2. The TCP/IP Encapsulation and Decapsulation ADJ Diagram

When the data in each layer of the server and the client is encapsulated and decapsulated, we represent *send* and *recv* operations. The behavior of *send* operation encapsulates the data from the Application Layer to the Physical Layer of the server, sends through network and decapsulates the data from the Physical Layer to the Application Layer of the client (*op send : STcp -> CTcp*). We represent the *module SEND*

for sending the data from the server to the client through TCP/IP. In the same way, the behavior of *recv* operation decapsulates the data from the Application Layer to the Physical Layer of the client, sends through network and encapsulates the data from the Physical Layer to the Application Layer of the server (*op recv : CTcp -> STcp*). We represent the *module RECV* for receiving the data from the client to the server through TCP/IP. The following modules are *SEND* and *RECV* modules by using CafeOBJ,

```
module SEND{
        protecting (INITIAL)
        op s5tos4 : SData -> STcp
        op s4tos3 : STcp -> SIp
        op s3tos2 : SIp -> SDlink
        op s2tos1 : SDlink -> SPhy
        op s1toc1 : SPhy -> CPhy
        op send : SData -> CData
        op c1toc2 : CPhy -> CDlink
        op c2toc3 : CDlink -> CIp
        op c3toc4 : CIp -> CTcp
        op c4toc5 : CTcp -> CData
        var S : SData
        var C : CData
--> ------ SEND
        eq c4toc5(c3toc4(c2toc3(c1toc2(s1toc1(s2tos1(s3tos2(s4tos3(s5tos4(S)))))))))  = C .
}
module RECV{
        protecting (INITIAL)
        op c5toc4 : CData -> CTcp
        op c4toc3 : CTcp -> CIp
        op c3toc2 : CIp -> CDlink
        op c2toc1 : CDlink -> CPhy
        op c1tos1 : CPhy -> SPhy
        op s1tos2 : SPhy -> SDlink
        op s2tos3 : SDlink -> SIp
        op s3tos4 : SIp -> STcp
        op s4tos5 : STcp -> SData
        op recv : CData -> SData
        var S : SData
        var C : CData
--> ------ RECV
        eq s4tos5(s3tos4(s2tos3(s1tos2(c1tos1(c2toc1(c3toc2(c4toc3(c5toc4(C)))))))))  = S .
}
```

In previous articles [Lai, 1995], [Zhang, Takahashi, Shiratori and Noguchi, 1988], [Chu and Liu, 1988], the techniques based on the State Transition diagram were proposed; however, these methods are not sufficient for specifying the system procedure. And [Hinchey and Jarvis, 1993], [Nakata, Higashino and Taniguchi, 1995], [Park and Miller, 1997], proposed to use the techniques based on time, but such methods cannot specify the system abstraction. These formal specification techniques motivate the idea to consider the TCP State Transition diagram and the Time Line together.

From the TCP State Transition diagram for the server and the client (figures 6 and 5), the number 1-6 are represented the sequences of the TCP Normal Connection Establishment and Termination with considering the Time Line in figure 4. The characters *S* represent the record SData for the server and the characters *C* represent the record CData for the client.

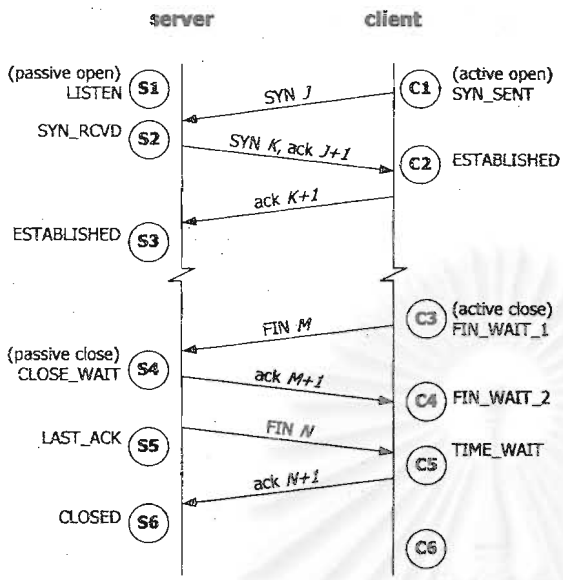


Figure 3. The TCP state corresponding to the TCP Normal Connection Establishment and Termination

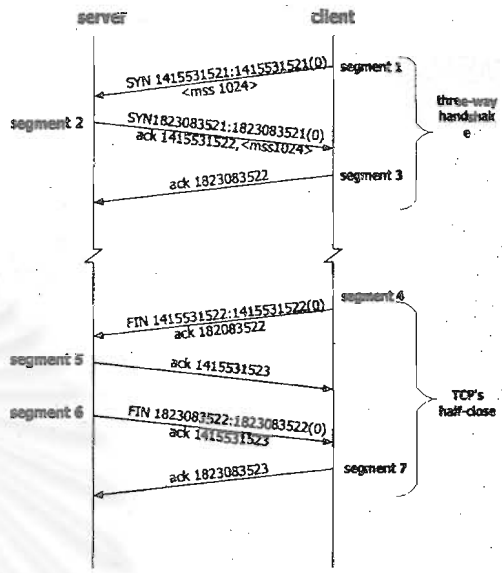


Figure 4. The Time Line of the TCP Normal Connection Establishment and Termination

In figure 4, the TCP Normal Connection Establishment and Termination is described [Washburn and Evans, 1993], [Stevens, 1995]. It shows three-way handshake and TCP's half-close.

Three-way handshake is shown for the completeness of connection establishment. In the first step (figures 3, 5 and 6) the client performs an active open (C1), the server performs a passive open (S1 When the client sends the first SYN J to the server S2). Second step, the server receives SYN J and sends the next SYN K which performs a passive open, and the server also SYN+1 to the client. Finally, the client must acknowledge this SYN from the server by ACK+1 to establish the connection (C2). The following is CafeOBJ specification which is analyzed by using the TCP State Transition diagram (figure 6 and 5) and the Time Line (figure 3).

```
open TCPSTATE
  ...
  eq closed-listen = init-sdata .                          -- S1
  eq closed-synsent = init-cdata .                         -- C1
  red CTcp { syn = J } .
  ceq listen-synrcvd(recv(C)) = S if check == Yes .        -- S2
  eq J + 1 = J1 .
  red STcp { syn = K, ack = J1 } .
  ceq synsent-established(send(S)) = C if check == Yes .   -- C2
  eq K + 1 = K1 .
  red CTcp { ack = K1 } .
  ceq synrcvd-established(recv(C)) = S if check == Yes .   -- S3
  ...
close
```

The details of three-way handshake are analyzed as follows;

S1: application passive open and send nothing
CafeOBJ: Define the record SData for the server

C1: application active open and send SYN J
CafeOBJ: Define *syn = J* in the record CData for the client

S2: send ACK J
CafeOBJ: Send the record CData to the server by using
op recv : CData -> SData
These operations also encapsulate and decapsulate through the TCP/IP.

C2: send ACK J+1 and SYN K
CafeOBJ: Define *syn = K* and *ack = J+1* in the record SData for the server and send the record SData to the client by using
op send : SData -> CData
These operations also encapsulate and decapsulate through the TCP/IP.

S3: send ACK K+1
CafeOBJ: Define *ack = K+1* in the record CData for the client and send the record CData to the server by using
op recv : CData -> Sdata
These operations also encapsulate and decapsulate through the TCP/IP.

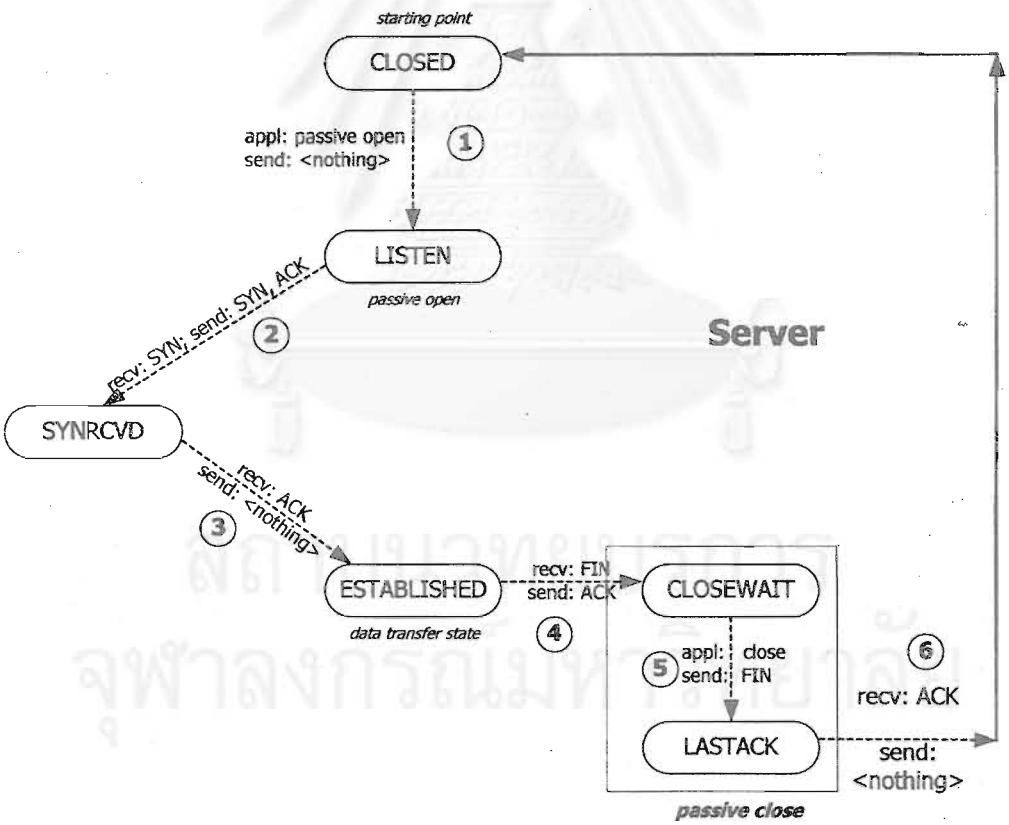


Figure 5. TCP state transition – Server

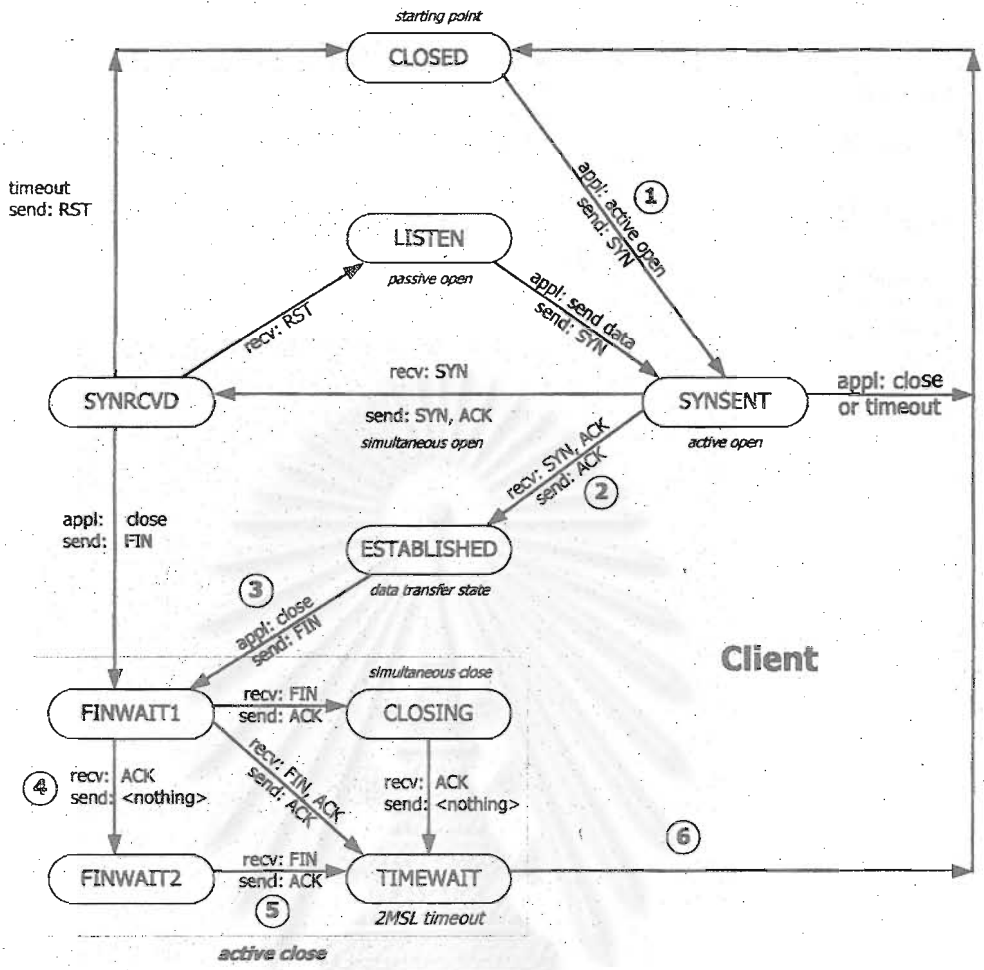


Figure 6. TCP state transition – Client

With the same concept of methodology, the TCP's half-close is analyzed.

The proof score validates that the TCP Normal Connection Establishment and Termination specification is correct.

CONCLUSION

The result of the TCP Normal Connection Establishment and Termination specification which is specified by considering the TCP State Transition diagram and the Time Line method is both clear and precise. This specification is especially valid for the correct operation.

As the feature of CafeOBJ, the specification is described as the behavioral abstraction. The procedure of the specification is analyzed by combining the TCP State Transition diagram and the Time Line and represented by using operation in CafeOBJ. Consequently, this specification is effective and appropriate for the system abstraction and the system procedure.

ACKNOWLEDGMENTS

We would like to thank Asso. Prof. Wiwat Vatanawood for his advice.

REFERENCES

Comer, D. E., and Stevens, D. L. 1996. "INTERNETWORKING WITH TCP/IP VOLUME III CLIENT-SERVER PROGRAMMING AND APPLICATIONS" Prentice-Hall.

Chu, P-Y.M., and Liu, M.T. 1988. "Protocol Synthesis in a State-Transition Model" In: Proc. of the Twelfth International Conference on Computer Software and Applications Conference, pp. 505-512.

Diaconescu, R., Futatsugi, K., and Iida, S. 1999. "Component-based Algebraic Specification and Verification in CafeOBJ" In: Proc. of World Congress of Formal Methods in the Development of Computing Systems.

Futatsugi, K., and Nakagawa, A. 1997. "An Overview of CAFE Specification Environment -an algebraic approach for creating verifying, and maintaining formal specifications over network-" In: Proc. of the First IEEE International Conference on Formal Engineering Methods, pp. 170-181.

Gaudel, M-C., 1994. "Formal Specification Techniques" In: Proc. of the Sixteenth Annual International Conference on Software Engineering, pp. 223-227.

Hinchey, M.G., and Jarvis, S.A. 1993. "Protocol Specification & verification Methods: An Overview" In: Proc. of IEEE Singapore International Conference on Networks, Vol. 2, pp. 581-585.

Information Sciences Institute University of Southern California. 1981. "Transmission Control Protocol Darpa Internet Program Protocol specification", RFC 793, 85 pages.

Layuan, L. 1989. "A Formal Specification Technique for Communication Protocol" In: Proc. of the 8th Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging, Vol. 1, pp. 74-81.

Lai, R. 1995. "Formal specification and verification of a procedural protocol: case study" In: Software Engineering Journal, Vol. 10, pp. 97-104.

Nakagawa, A.T., Sawada, T., and Futatsugi, K. 1999. "CAFEOBJ USER'S MANUAL -VER.1.4-".

Nakajima, S., and Futatsugi, K. 1997. "An Object-Oriented Modeling Method for algebraic Specification in CafeOBJ" In: Proc. of the International Conference on Software Engineering, pp. 34-44.

Nakata, A., Higashino, T., and Taniguchi, K. 1995. "Protocol Synthesis from Timed and Structured Specifications" In: Proc. of IEEE International Conference on Network Protocols, pp. 74-81.

Park, J-C., and Miller, R.E. 1997. "Synthesizing Protocol Specifications from Service Specfications in Timed Extended Finite State Machines" In: Proc. of the 17th International Conference on Distributed Computing Systems, pp. 253-260.

Pressman, R.S. 1997. "SOFTWARE ENGINEERING" Addison-Wesley Publish Company, Fifth Edition, 157-187.

Stevens, W.R. 1991. "UNIX NETWORK PROGRAMMING" Prentice-Hall.

Stevens, W.R. 1995. "TCP/IP ILLUSTRATED VOLUME I" Addison-Wesley.

Washburn, K., and Evans, J.T. 1993. "TCP/IP RUNNING A SUCCESSFUL NETWORK" Addison-Wesley.

Zhang, Y.X., Takahashi, K., Shiratori, N., and Noguchi, S. 1988. "An Interactive Protocol Synthesis Algorithm Using a Global State Transition Graph" In: Proc. of IEEE Transaction on Software Engineering, Vol. 4. No. 3, pp. 394-404.

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวสุชาดา จุลจันทร์ เกิดเมื่อวันที่ 13 พฤศจิกายน พ.ศ. 2519 สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต (วศ.บ.) สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี เมื่อปีการศึกษา 2542 และเข้าศึกษาต่อทันทีในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต (วศ.ม.) ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย