การพัฒนาแบบจำลองการเปิดการติดต่อแบบพร้อมกันในโปโตคอล DCCP

(Formal Model of the Procedures for Simultaneously Open)

โดย

นางสาวสาวิตรี	นาสวัสดิ์	B4906841
นางสาวจิตรา	จันทร์สมบูรณ์	B4909613

รายงานนี้เป็นส่วนหนึ่งของการศึกษาวิชา 427499 โครงงานวิศวกรรมโทรคมนาคม

หลักสูตรวิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมโทรคมนาคม หลักสูตรปรับปรุง พ.ศ. 2545

สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

ประจำภาคการศึกษาที่ 3 ปีการศึกษา 2552

โครงงาน	เรื่อง การพัฒนาแบบจำลองการเปิดการติดต่อแบบพร้อมกันในโปโตคอล DCCP
จัดทำโดย	1. นางสาวสาวิตรี นาสวัสดิ์ รหัสประจำตัว B4906841
	2. นางสาวจิตรา จันทร์สมบูรณ์ รหัสประจำตัว B4909613
อาจารย์ที่ปรึกษา	อาจารย์ ดร. สมศักดิ์ วาณิชอนันต์ชัย
สาขาวิชา	วิศวกรรมโทรคมนาคม
ภาคการศึกษา	3/2552

บทคัดย่อ

(Abstract)

โปรโตคอลในระดับชั้นทรานสปอร์ต Datagram Congestion Control Protocol (DCCP) เป็นโปรโตคอลที่กำลังได้รับการพัฒนาขึ้นมาเพื่อแก้ไขปัญหาความแออัดของ Traffic ใน Internet ในโครงงานนี้จึงต้องการศึกษาพัฒนาและวิเคราะห์แบบจำลองของ DCCP ในกรณี Simultaneous-Open ตามมาตรฐานที่ได้รับการอนุมัติจาก Internet Engineering Steering Group (IESG) โดยใช้ Coloured Petri Nets

กิตติกรรมประกาศ

(Acknowledgement)

จากการที่คณะผู้จัดทำได้รับมอบหมายให้ทำโครงงานเรื่อง การพัฒนาแบบจำลองการเปิดการติดต่อแบบพร้อมกันในโปโตคอล DCCP (Formal Model of the Procedures for Simultaneously Open) ทำให้คณะผู้จัดทำได้รับประโยชน์และความรู้มากขึ้น เกี่ยวกับการใช้โปรแกรม CPN Tools ตลอดจนมีความรู้ความเข้าใจในตัวโปรโตคอล DCCP บัดนี้โครงงานการพัฒนาแบบจำลองการเปิดการติดต่อแบบพร้อมกันในโปโตคอล DCCP สามารถสำเร็จลุล่วงไปด้วยดี เนื่องด้วยคณะผู้จัดทำได้รับคำแนะนำและความอนุเคราะห์ช่วยเหลือในด้านต่าง ๆ จนทำให้งานสามารถดำเนินงานลุล่วงไปด้วยดี จึงใคร่ขอขอบพระคุณบุคคลดังรายนามต่อไปนี้

อาจารย์ ดร. สมศักดิ์ วาณิชอนันต์ชัย อาจารย์ที่ปรึกษาโครงงานที่ให้คำแนะนำ ให้คำปรึกษา และดูแลการทำโครงงานอย่างใกล้ชิดตลอดการทำโครงงาน

คณาจารย์ทุกท่าน และเพื่อน ๆ ปริญญาตรีสาขาวิชาวิศวกรรมโทรคมนาคมทุกคนที่ให้ความช่วยเหลือมาโดยตลอด

โครงงานนี้จะไม่สามารถสำเร็จลุล่วงไปได้หากปราศจากแรงสนับสนุนจากบุคคลดังรายนามข้างต้น ทางคณะผู้จัดทำจึงขอขอบพระคุณทุก ๆ ท่านเป็นอย่างสูงมา ณ โอกาสนี้ หากมีข้อผิดพลาดประการใด คณะผู้จัดทำใคร่ขออภัยมา ณ ที่นี้ด้วย

นางสาวสาวิตรี นาสวัสดิ์

นางสาวจิตรา จันทร์สมบูรณ์

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ(ต่อ)

บทที่ 1
บทนำ

1.1 บทนำ

วิวัฒนาการของเทคโนโลยีการสื่อสารข้อมูลและเครือข่ายคอมพิวเตอร์นั้น ได้มีการพัฒนามาอย่างต่อเนื่องและรวดเร็ว เป็นผลให้ปัจจุบันเราสามารถส่งข้อมูลได้อย่างรวดเร็ว ทำให้การตัดสินใจต่างๆ สะดวกและมีความถูกต้องมากยิ่งขึ้น ดังนั้น เราจึงต้องศึกษาในเรื่องของเทคโนโลยีที่ช่วยในการสื่อสารข้อมูล เพื่อที่จะทำให้เราเข้าใจและสามารถนำไปใช้ประโยชน์ต่อไป

เนื่องจากการสื่อสารข้อมูลนั้นสามารถที่จะส่งได้ทั้งข้อความภาพ เสียง และวิดีโอ ได้อย่างรวดเร็ว ไม่ว่าจะอยู่ที่ใดในโลก ซึ่งถ้าสังเกตไปรอบๆ ตัวเรา จะพบว่าสิ่งเหล่านี้มีผลกับชีวิตประจำวันเราเป็นอย่างมาก และสิ่งที่จะนำเสนอในรายงานฉบับนี้ จะกล่าวถึง การสื่อสารข้อมูลของ โปรโตคอล DCCP ความรู้ความเข้าใจเกียวกับ Coloured Petri Net และการพัฒนา วิเคราะห์แบบจำลอง DCCP ตามมาตรฐาน RFC 4340 และ RFC 5596 โดยใช้ Coloured Petri Net

1.1.1 DCCP (Datagram Congestion Control Protocol)

DCCP (Datagram Congestion Control Protocol) เป็นโปรโตคอลที่อยู่ในชั้น transport ถูกออกแบบโดย IETF จากปี 2003 มาเป็นปี 2005 ซึ่งพัฒนามาจาก RFC 4340 ในเดือนมีนาคม 2006

DCCP เป็นโปรโตคอลที่พัฒนามาจากโปรโตคอล UDP และยังถูกพัฒนาขึ้นมาเพื่อช่วยแก้ปัญหาความแออัดของ Traffic ใน Internet โดยโปรโตคอล DCCP มีการสร้างการติดต่อแบบ Reliable ซึ่งมีการสร้างการติดต่อและยกเลิกการติดต่อแบบรับประกนความถูกต้อง และ มีการเลือกใช้ Congestion Control ได้ ขึ้นกับ Application ที่ใช้ และ DCCP ยังมีการใช้ Application บังคับเวลาให้มีการส่งข้อมูลที่เร็วกว่าข้อมูลเก่าที่สูญหาย ซึ่งเป็นจุดเด่นที่ TCP ,UDP และ SCTP ไม่มี DCCP จึงเหมาะกับการเพิ่มประสิทธิภาพการใช้งานแบบ Real Time และ Delay Sensitive

1.1.2 CPN (Coloured Petri Nets)

คัลเลอร์เพ็ทติ -เน็ต (CP-net หรือ CPNs) คือภาษาทางกราฟฟิกที่ใช้สำหรับสร้างโครงสร้างของโมเดลของระบบที่เกิดขึ้นพร้อม ๆ กัน และวิเคราะห์คุณสมบัติเหล่านั้น เพ็ทติ -เน็ต จะจัดหี้การสร้างกระบวนการโดยใช้เทคนิคทางกราฟฟิก และเหตุการณ์ที่มีการสื่อสารกัน

พื้นฐานแรกสำหรับการสร้างโมเดลของระบบที่เกิดเหตุการณ์ขึ้นพร้อม ๆ กันและเหตุการณ์ของระบบที่เกิดขึ้นในเวลาเดียวกัน คือการใช้ CPN model ซึ่งเป็นภาษาที่ใช้เขียนโปรแกรม ที่อยู่บนพื้นฐานของโปรแกรมภาษามาตรฐาน ซึ่งเป็นการพัฒนาแรกเริ่มสำหรับการกำหนดชนิดข้อมูล การอธิบายการจัดการของข้อมูล และสำหรับการสร้างและการกำหนดข้อตกลงของพารามิเตอร์ต่าง ๆ ของโปรโตคอล

1.2 หลักการและเหตุผล

Datagram Congestion Control Protocol (DCCP) เป็นโปรโตคอลในระดับชั้นทรานสปอร์ต ที่ได้รับการกำหนดเป็นมาตรฐาน RFC 4340 ใน ปี 2006 ตามมาตรฐาน RFC 4340 นั้น Client จะต้องเป็นผู้เริ่มต้นขอใช้บริการ (Initiates call) ก่อน โดยที่ Server ไม่สามารถเป็นผู้เริ่มต้นเรียกใช้บริการได้ ดังนั้นหาก Server อยู่ใน Private Network หรืออยู่หลัง Fire wall แล้ว Client จะไม่สามารถเรียก Server ได้ วิธีการแก้ปัญหาวิธีหนึ่งคือ กำหนดให้ Server สามารถเป็นผู้เริ่มต้นเรียกใช้ได้ เพื่อแก้ปัญหาดังกล่าวในเดือนมิถุนายน 2009 Internet Engineering Steering Group (IESG) ได้อนุมัติมาตรฐานใหม่คือ "DCCP Simultaneous-Open Technique to Facilitate NAT/Middlebox Traversal" RFC 5596

ในขณะนี้บริษัทผู้ผลิตต่างๆ กำลังพัฒนาซอร์แวร์ โปรโตคอล DCCP เพื่อนำไปใช้งานจริงใน Internet ถ้าหากหลังจากใช้งานแล้วพบว่า โปรโตคอลมีข้อผิดพลาด การแก้ไขจะทำได้ยาก เนื่องจากมีผู้ใช้ Internet เป็นจำนวนมาก ในโครงงานนี้จึงต้องการสร้างแบบจำลองของ DCCP ในกรณี Simultaneous-Open เพื่อตรวจสอบว่าโปรโตคอลมีข้อผิดพลาดหรือไม่

1.3 วัตถุประสงค์

1. ศึกษาวิธีการใช้งานโปรแกรม CPN Tools
2.ศึกษาโปรโตคอล DCCPในกรณี Simultaneous-Open
3.ต้องการสร้างและวิเคราะห์แบบจำลอง โปรโตคอล DCCPในกรณี Simultaneous- Open

1.4 ขอบเขตงาน

1. เนื่องจากDCCP มีรายละเอียดและความซับซ้อนมาก ขอบเขตของโครงงานนี้จะจำกัดอยู่ที่การจัดการการเชื่อมต่อในกรณี Simultaneous-Open
2. ในโครงงานนี้ใช้ Formal Method และ Coloured Petri Nets ในการสร้างแบบจำลองแล ะทดลอง

1.5 ผลที่คาดว่าจะได้รับ

1. แบบจำลองโปรโตคอล DCCP ในกรณี Simultaneous-Open
2. มีความรู้ความเข้าใจเกี่ยวกับโปรโตคอลใน Internet
3. มีความรู้ความเข้าใจเกี่ยวกับการวิเคราะห์ระบบที่ซับซ้อนมากๆ

บทที่2

Formal Methods และ Coloured Petri Nets

ในบทนี้ เราจะแนะนำระเบียบวิธีวิจัยที่ใช้ อันได้แก่ Formal Methods และ Coloured Petri Nets ซึ่งในโครงการวิจัยนี้ นำมาใช้สร้างและวิเคราะห์แบบจำลองของระบบอาณัติสัญญาณรถไฟ รวมทั้งตรวจทาน (Formal Verifications)

2.1 Formal Methods

Formal methods เป็นเทคนิคหนึ่งในสาขาวิศวกรรมซอร์ฟแวร์ ซึ่งนำคณิตศาสตร์ มาใช้กำหนดไวยากรณ์และความหมายของรายการจำเพาะ พัฒนา ตรวจทานซอร์ฟแวร์และฮาร์ดแวร์ของระบบที่ต้องการสร้าง Formal Specifications เป็นการนำคณิตศาสตร์ มาใช้กำหนดรายการจำเพาะ เพื่อกำจัดสิ่งที่กำกวมออกไป ทำให้รายการจำเพาะชัดเจน และไม่มีข้อผิดพลาดจากการตีความ Formal Verifications เป็นการใช้คณิตศาสตร์อีกเช่นกัน เพื่อพิสูจน์ว่าระบบมีคุณสมบัติถูกต้องตาม Formal Specifications หรือไม่ เมื่อนำ Formal methods มาใช้ในระหว่างการพัฒนาระบบ จะทำให้ค้นพบข้อผิดพลาด และความไม่สมบูรณ์ ในช่วงต้นๆ ของการพัฒนา ซึ่งถ้าหากพบข้อผิดพลาดในภายหลัง จะทำให้สิ้นเปลืองค่าใช้จ่ายในการแก้ไขข้อผิดพลาด Formal methods หลากหลายชนิดได้รับการพัฒนาขึ้นอยู่กับคณิตศาสตร์ที่ใช้ แต่ละชนิด แต่ละวิ ธีก็มีจุดเด่นจุดด้อยแกต่างกันออกไป เราสามารถแบ่งระเบียบวธีวิจัยของ Formal Verifications ออกเป็น 2 วิธีได้แก่ Theorem proving และ Model Checking

วิธีแรกเป็นการกำหนดคุณสมบัติที่ต้องการตรวจทานให้อยู่ในรูปของทฤษฎีบททาง คณิตศาสตร์ และทำการพิสูจน์โดยใช้เครื่องมือ อช่วยหรืออาจทำด้วยมือก็ได้ ในขณะที่วิธีที่สองสร้าง State space ของระบบที่ต้องการตรวจทานขึ้น กำหนดคุณสมบัติที่ต้องการตรวจทาน แล้วค้นหา States ที่มีคุณสมบัติตามที่ต้องการ ในโครงการวิจัยนี้เราจะใช้วิธี Model Checking

2.2 Petri Nets

Carl Adam Petri เป็นนักวิจัยรุ่นแรกๆที่เสนอ Theory of concurrency ในวิทยานิพนธ์เรื่อง “Kommunikation mit Automaten” ในปี 1962 ซึ่งต่อมาเป็นที่รู้จักกันแพร่หลายในยุโรป ในชื่อ Petri Nets หลังจากปี 1962 ได้มีนักวิจัยขยายและเพิ่มเติมทฤษฎีเกี่ยวกับ Petri Nets ออกไปอีกมากมาย อาทิเช่น Predicate Transition Nets, Algebraic Petri Nets, Fuzzy Petri Nets, Possibilistic Petri Nets, Stochastic Petri Nets, Coloured Petri Nets, Reference Net, Hybrid Petri Nets, Algebraic Higher Order Nets, Petri Nets without Tokens สำหรับทฤษฎีพื้นฐานของ Petri Nets อ้างถึง

2.3 ทำไมจึงเลือกใช้ Coloured Petri Nets

Coloured Petri Nets (CPN) ได้รับการพัฒนาโดย Kurt Jensen ที่ Aarhus University และได้มีการนำไปประยุกต์ใช้ในงานต่างๆ สาเหตุในการเลือก Coloured Petri Nets เป็นเครื่องมือสำคัญในการสร้างและวิเคราะห์แบบจำลองของระบบอาณัติสัญญาณ มีดังต่อไปนี้

2.3.1 ง่ายในการความเข้าใจ และการใช้งาน

ข้อได้เปรียบที่สำคัญของ CPN คือเรื่อง Readability แทนที่จะเป็นชุดคำสั่ง CPN ใช้รูปภาพ (Graphical notation) เป็นหลักทำให้เข้าใจง่าย และการจัดโครงสร้างแบบจำลองเป็นลำดับชั้น (Hierarchical structure) การแบ่งแบบจำลองออกเป็นโมดูล ทำให้ซ่อนหรือกระจายความซับซ้อน ทำให้อ่านแบบจำลองได้ง่ายขึ้น ความสามารถในการใช้ Datatype ที่ซับซ้อนทำให้ ความซับซ้อนของแบบจำลองลดน้อยลง สิ่งเหล่านี้ช่วยทำให้การพัฒน าแบบจองทำได้ง่ายขึ้น แทนที่จะเสียเวลาไปกับการใช้เครื่องมือ เพราะความง่ายจึงมีเวลาไปวิจัยระบบที่ต้องการศึกษามากขึ้น

2.3.2 ความสามารถในการวิเคราะห์แบบจำลอง

ความรู้เกี่ยวกับการตรวจทานระบบ (Formal verifications) ที่จำลองโดย Petri Nets นั้น มีการศึกษาวิจัย และสะสมความรู้มานานกว่าสี่สิบปี การวิเคราะห์ State space เป็นการสำรวจสถานะ

และเหตุการณ์ที่มีโอกาสเกิดขึ้นได้ทั้งหมด ทำให้ค้นพบบางกรณีที่อาจจะไม่สามารถค้นพบโดยวิธี Simulations

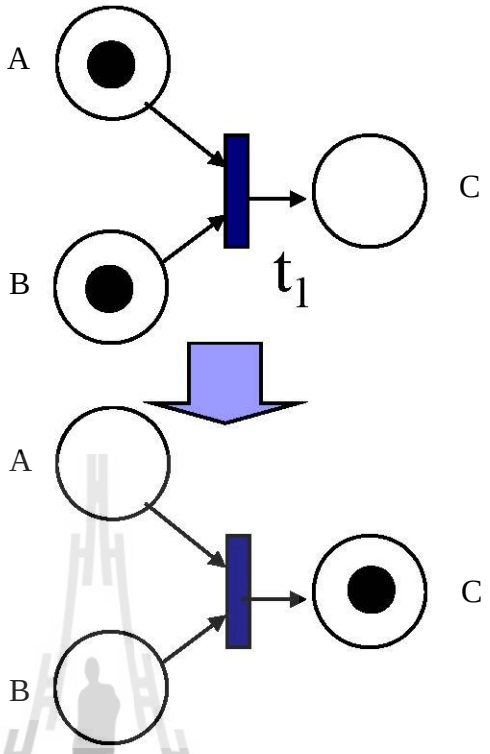
2.3.3 เครื่องมือสนับสนุนอื่นๆ

โครงการวิจัยนี้นำ CPN Tools ซึ่งเป็นซอร์ฟแวร์ที่พัฒนาโดย CPN group ที่ Aarhus University มาใช้ในการ Creating Editing Simulating Debugging และ Analyzing แบบจำลอง CPN Tools เป็นซอร์ฟแวร์ที่มีเสถียรภาพ และมีเครื่องมือสนับสนุนต่างๆ เช่น เครื่องมือช่วยวาด State space เครื่องมือวาด Time sequence chart ฟังก์ชัน Sweep-line และประการสำคัญ คือสามารถใช้ภาษา CPN-ML ซึ่งดัดแปลงมาจากภาษา SML (standard ML) CPN Tools เป็นซอร์ฟแวร์ที่ง่ายแก่การใช้งานและในขณะเดียวกันสามารถก็ขยายต่อและใช้งานอย่างซับซ้อนได้โดยใช้ ภาษาCPN-ML

2.4 แนะนำ Petri Nets

Petri Nets อันเป็นที่รู้จักกันอย่างแพร่หลายในปัจจุบัน มีลักษณะเป็นกราฟ ประกอบด้วย วงกลม สี่เหลี่ยม ลูกศร และจุดดำ ลูกศรนั้นเชื่อมต่อระหว่างวงกลมและสี่เหลี่ยม ไม่สามารถเชื่อมต่อระหว่างวงกลมและวงกลม หรือระหว่างสี่เหลี่ยมและสี่เหลี่ยมได้ สี่เหลี่ยมเรียกว่า Transition ใช้แทนเหตุการณ์ (Event) ที่สามารถเกิดขึ้นในระบบ วงกลมเรียกว่า Places ในแต่ละ Place จะมีชื่อกำกับอยู่ และในแต่ละ Places บรรจุจุดสีดำ ที่เรียกว่า Token หรือ เบี้ย ในแต่ละ Places สามารถบรรจุ Tokens มากกว่าหนึ่งชิ้น หรือไม่มีเลยก็ได้ (เรียกว่า empty)

Places ซึ่งอยู่ต้นทางลูกศรที่เชื่อมไปยัง Transitions เรียกว่า Input Places สำหรับลูกศรที่ชี้เข้าหา Transitions นี้เราเรียกว่า Input Arcs ส่วน Places ซึ่งอยู่ปลายลูกศรที่ออกจาก Transitions เรียกว่า Output Places และลูกศรที่ชี้ออกจาก Transitions เราเรียกว่า Output Arc



รูปที่ 2.1 แสดงตัวอย่างของ Petri Nets ที่มีคุณสมบัติ Synchronization

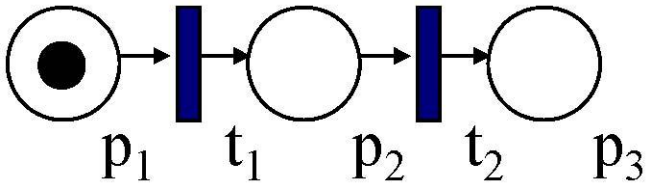
ลูกศรแต่ละเส้นอาจจะมีตัวเลขกำกับ (Arc Inscriptions) ตัวเลขที่อยู่บน Input Arcs หมายถึงจำนวน Tokens ใน Input Places ที่ต้องการใช้เพื่อให้เกิดเหตุการณ์ หรือเรียกว่า Transitions fires หลังจากที่ Transitions fires แล้ว Tokens ใน Input Places จะถูกกำจัดทิ้งไป แล้ว สร้าง Tokens ใหม่ใส่ลงใน Output Places เป็นจำนวนเท่ากับตัวเลขที่กำกับบน Output Arcs ถ้าไม่มีตัวเลขกำกับ หมายถึงต้องการใช้ Token เพียง 1 ตัว

รูปที่ 2.1 แสดงตัวอย่างของ Petri Nets ก่อนและหลังจากที่ Transition t fires Tokens ใน places ต่างๆ ใช้แทนสถานะ (State) ของระบบ เรียกว่า Markings ยกตัวอย่างเช่นก่อน Transition t ทำงาน Markings M1 = {1,1,0} หลังจาก Transition t ทำงาน Markings M2 = {0,0,1} เป็นต้น รูปที่ 2.1 ยังแสดงให้เห็นถึงระบบที่คุณสมบัติ Synchronization คือจะต้องมี Token อยู่ใน Place A และ B พร้อมๆกัน Transition t จึงสามารถทำงานได้ คุณสมบัติอื่นๆที่สำคัญได้แก่

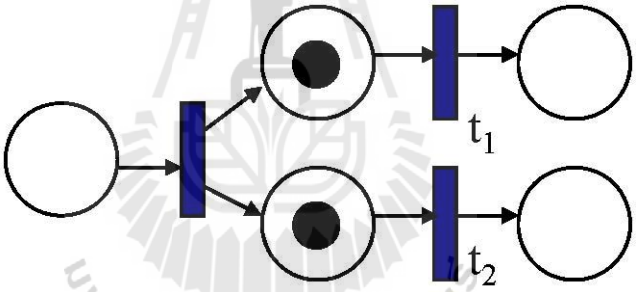
ก. คุณสมบัติ Sequential ดังแสดงในรูปที่ 2.2 Transition t2 จะทำงานได้ หลังจากที่ t1 ทำงานเท่านั้น

ข. คุณสมบัติ Concurrent ดังแสดงในรูปที่ 2.3 Transition t1 และ t2 สามารถทำงานได้พร้อมๆกัน ความหมายของ Concurrency คือไม่ว่า Transition t1 ทำงานก่อน แล้ว Transition t2 จึง

ทำงาน หรือว่า Transition t2 ทำงานก่อน แล้ว Transition t1 จึงทำงาน ผลลัพท์สุดท้ายจะได้ Markings ที่เหมือนกันทั้งสองกรณี

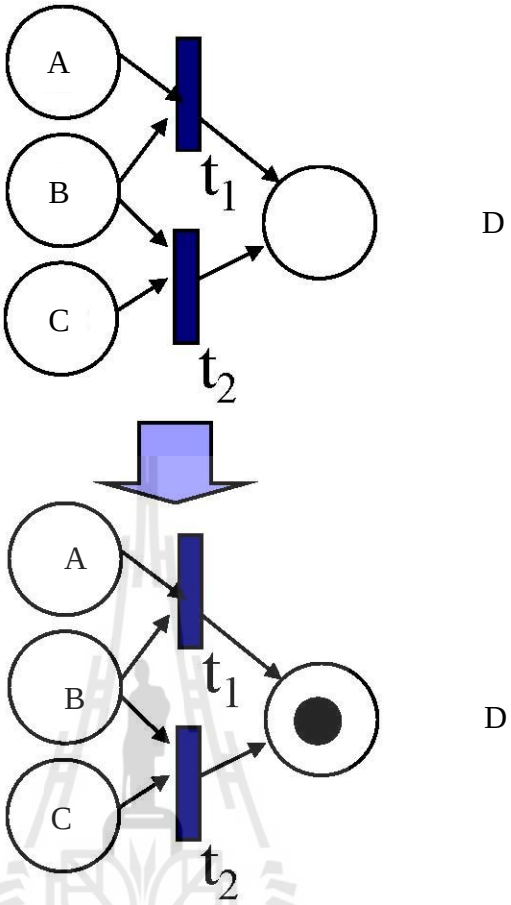


รูปที่ 2.2 แสดง Petri Nets ที่มีคุณสมบัติ Sequential



รูปที่ 2.3 แสดง Petri Nets ที่มีคุณสมบัติทำงานพร้อมๆกัน (Concurrent)

ค. คุณสมบัติขัดแย้งกัน (Conflict) ดังแสดงในรูปที่ 2.4 ใน Marking แรก ทั้ง Transition t1 หรือ t2 สามารถทำงานได้ แต่ถ้า t1 ทำงานแล้ว จะทำให้ t2 ไม่สามารถทำงานได้ เนื่องจากเมื่อได้ใช้ Token เพื่อให้ t1 ทำงานแล้ว t2 ไม่สามารถทำงานได้



รูปที่ 2.4 แสดง Petri Nets ที่มีคุณสมบัติขัดแย้งกัน (Conflict)

2.5 แนะนำ Coloured Petri Nets (CPN)

Coloured Petri Nets (CPN) นั้นแตกต่าง กับ Petri Nets ธรรมดาตรงที่ Tokens ใน CPNs ไม่ใช่เป็นเพียงจุดดำ แต่เป็นข้อมูลที่มีโครงสร้างข้อมูลอันซับซ้อนได้ นอกจากนี้ในแบบจำลอง CPNs ยังสามารถจัดโครงสร้างแบบ Hierarchy โดยสามารถแทน CPNs กลุ่มหนึ่งๆ ด้วย Substitution Transition ในหัวข้อนี้จะแนะนำ Coloured Petri Nets (CPN) โดยใช้รูปที่ 2.5 ประกอบการอธิบาย

2.5.1 <u>ส่วนประกอบต่างๆของ CPN diagram</u>

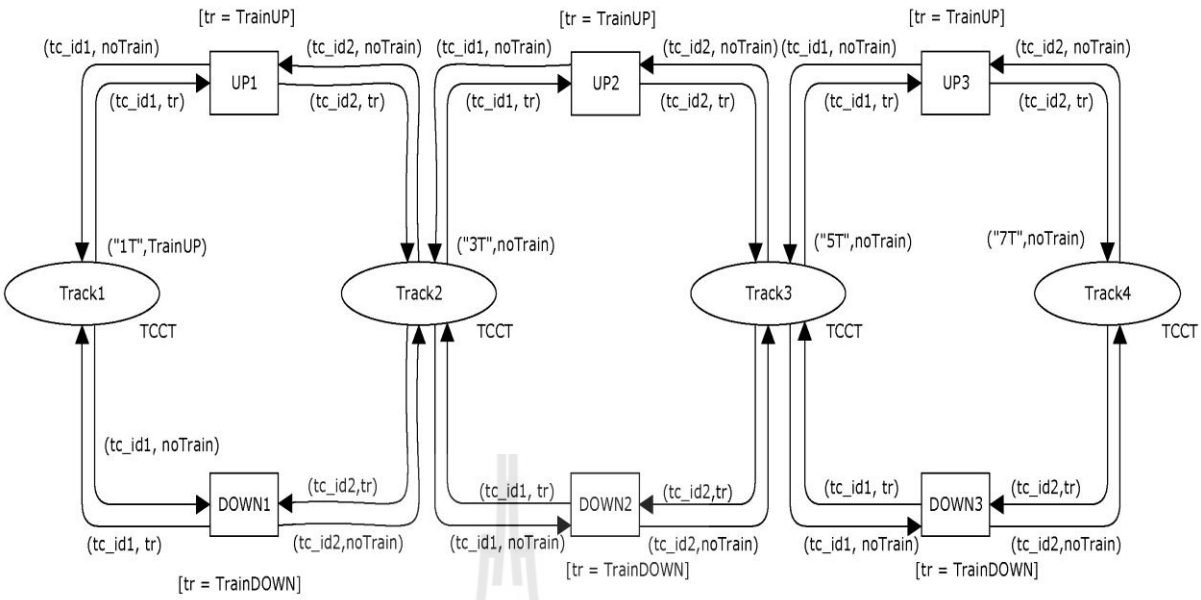
แบบจำลองวงจรไฟตอน 4 วงจร แสดงดังรูปที่ 2.5 ประกอบด้วย places จำนวน 4 ตัว และ transitions จำนวน 3 ตัวแต่ละ place มีคุณสมบัติได้แก่ 1) มีชื่อของตัวเอง 2) มี TYPE คือโครงสร้างของข้อมูลที่เป็น token บรรจุอยู่ใน place นั้น และ 3) ค่าของ token เริ่มต้น เรียกว่า Initial Markings

ระหว่าง places และ transitions เชื่อมต่อกันด้วย arc และมีนิพจน์ที่กำกับอยู่บน arc ซึ่งอาจจะเป็น expression หรือ function เรียกว่า arc inscriptions หมายความว่า ค่าและโครงสร้างข้อมูลของ tokens ที่นำมา enable transitions จะต้องตรงกับ ค่าและและโครงสร้างข้อมูลของของนิพจน์ที่กำกับอยู่

สำหรับ CPN ในรูปที่ 2.5 มีการประกาศโครงสร้างข้อมูลและตัวแปรดังนี้

```
colset TD = with noTrain | TrainUP | TrainDOWN;
var tr:TD;
colset TCCT = product STRING*TD;
var tc_id1, tc_id2:STRING;
```

ในรูปที่ 2.5 แต่ละ place (วงรี) ใช้จำลองการปรากฏตัวของรถไฟ โครงสร้างของ Token ในแต่ละ places เป็นผลคูณของหมายเลขของวงจรไฟตอน กับคำว่า TrainUP หรือ TrainDown หรือ noTrain ซึ่ง TrainUP ใช้แทนขบวนรถขาขึ้น TrainDOWN ใช้แทนขบวนรถขาลง และ noTrain หมายถึงไม่มีขบวนรถอยู่ในวงจรไฟตอน



รูปที่ 2.5 แสดงแบบจำลอง CPN แทนการเคลื่อนที่ของรถไฟบนวงจรไฟตอน 4 วงจร

ตัวอย่างความหมายของ Token ดังเช่น (“1T”, TrainUP) หมายความว่า มีขบวนรถขาขึ้นอยู่ในตอนหมายเลข 1T และ (“7T”, noTrain) หมายความว่า ไม่มีขบวนรถอยู่ในตอนหมายเลข 7T Transition “UP1” “UP2” และ “UP3” ใช้จำลองเหตุการณ์ การเคลื่อนที่ของขบวนรถจากซ้ายไปขวา (ขาขึ้น) ทำนองเดียวกัน Transition “DOWN1” “DOWN2” และ “DOWN3” ใช้จำลองเหตุการณ์ การเคลื่อนที่ของขบวนรถจากขวาไปซ้าย (ขาล่อง) ทุก Transitions ในรูปที่ 2.5 มี Guard (“[ ]”) ซึ่งนิพจน์ใน Guard ต้องเป็นจริง Transitions จึงสามารถทำงานได้ ตัวอย่างเช่น Transition “UP1” จะทำงานได้ก็ต่อเมื่อ ตัวแปร tr เท่ากับ TrainUP นั่นคือ มี token (“1T”, TrainUP) อยู่ใน place Track1

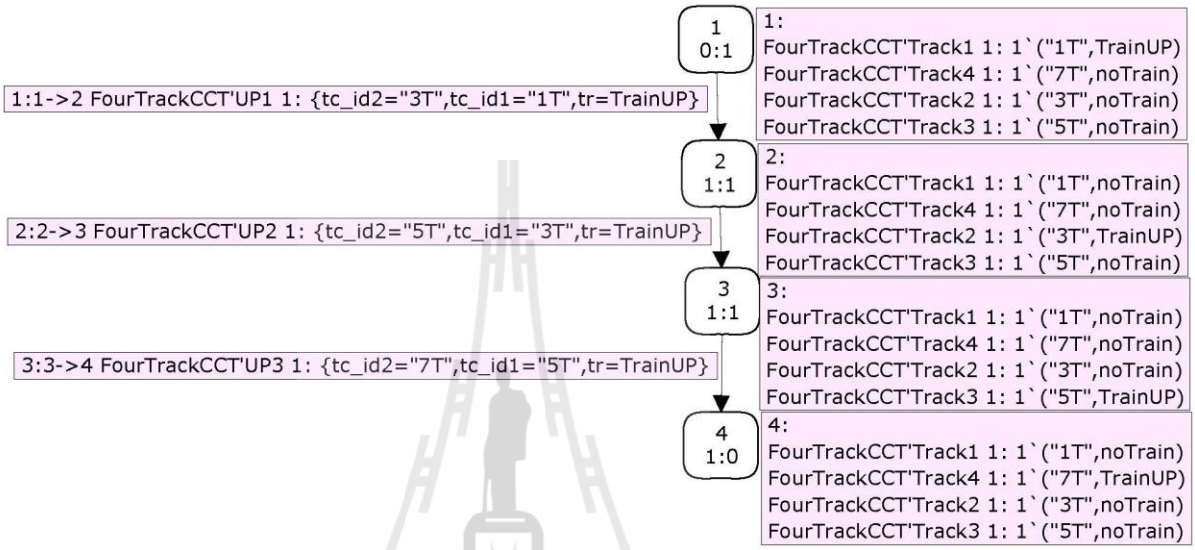
การที่ Transition UP1 จะทำงานได้ (Enable) จะต้องมีการกำหนดค่าให้ตัวแปรต่างๆ ที่เกี่ยวข้องกับ Transition UP1 อันได้แก่ tc_id1 = “1T”, tc_id2 = “3T” และ tr = TrainUP ขบวนการกำหนดค่าตัวแปรนี้เรียกว่า Binding ในตัวอย่างนี้แต่ละขั้นของการทำงานมี Binding เพียงชุดเดียว

2.5.2 State space diagram

เราตั้งชื่อ CPN diagram ในรูปที่ 2.5 ว่า “FourTrackCCT” และใช้ CPN Tools สร้างสถานะ (States) ของระบบทั้งหมดที่เป็นไปได้เริ่มต้นจาก Initial Markings ในรูปที่ 2.5 รูปที่ 2.6 แสดง State space diagram ของ CPN ในรูปที่ 2.5 ประกอบด้วยโนด แต่ละโนด มีหมายเลขประจำตัว

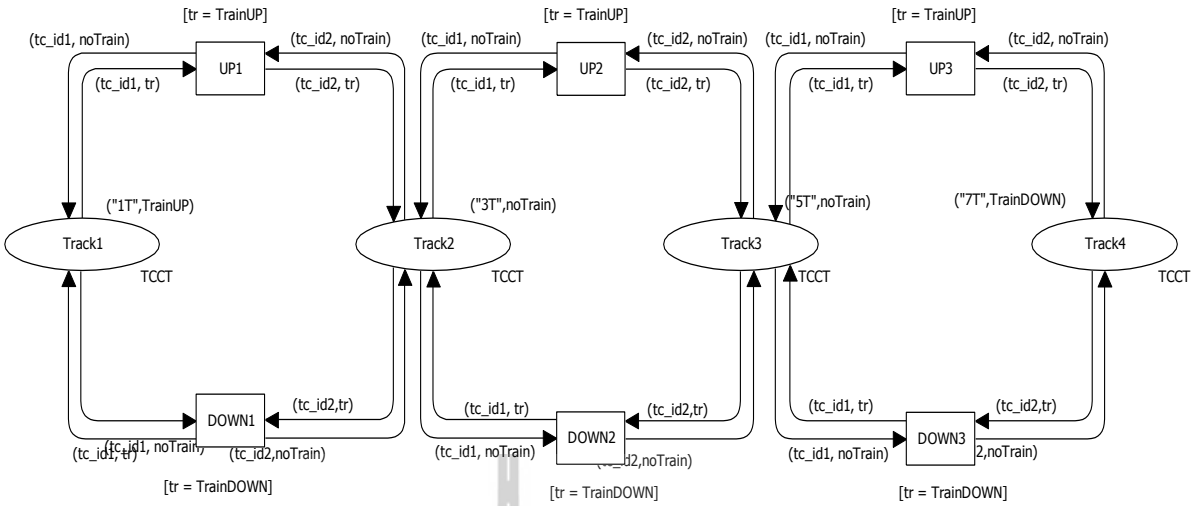
บรรทัดถัดจากหมายเลขประจำโนด แสดงจำนวนของ state ก่อนหน้านี้ และ state ถัดไป ในสี่เหลี่ยมข้างๆโนดแสดงค่าของ tokens ที่อยู่ใน place ต่างๆ จะเห็นได้ว่าค่าของ tokens ใน place ทั้งสี่ตัว เป็นตัวกำหนด State คือเรียกว่า Marking

ในสี่เหลี่ยมข้างๆ arc แสดง การ binding ค่าตัวแปรต่างๆที่ ใช้ในการ enable และ fire transition

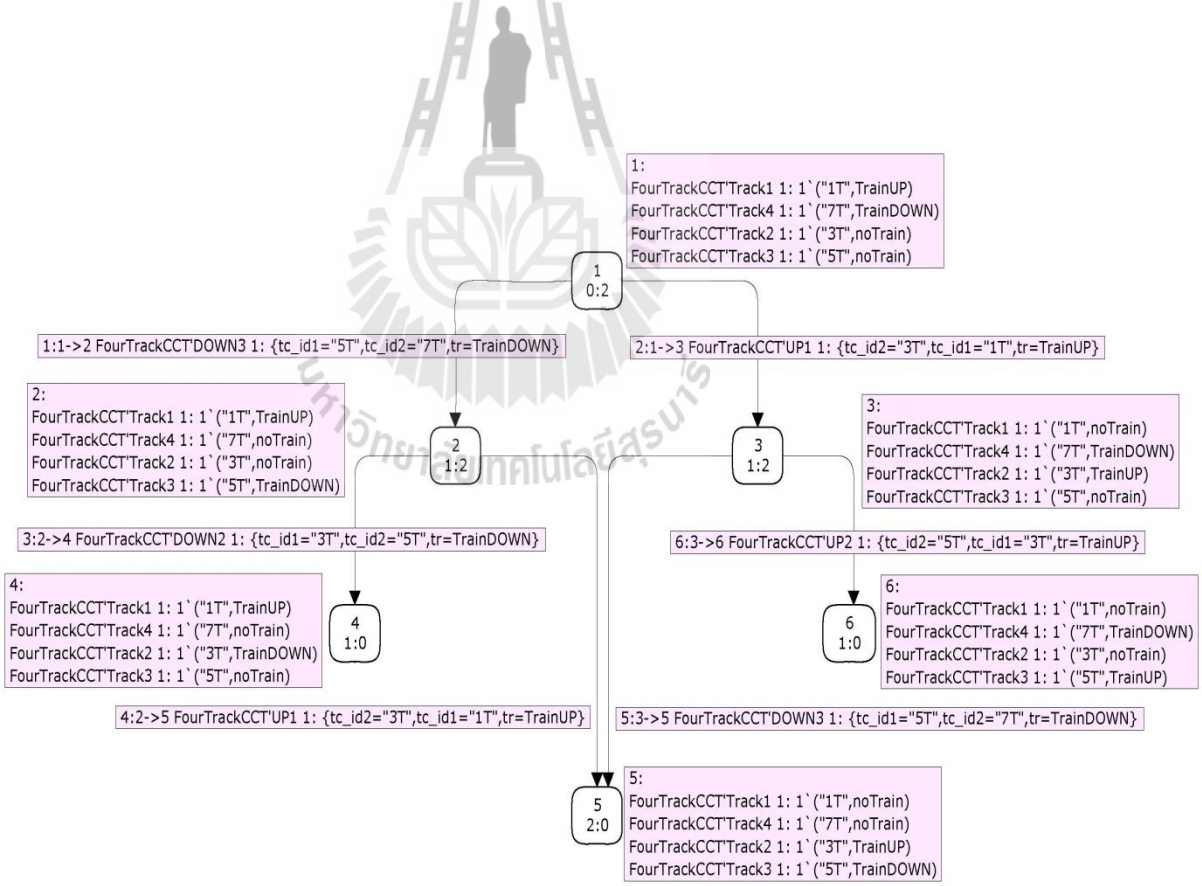


รูปที่ 2.6 State space diagram ของแบบจำลอง ในรูปที่ 2.5

เนื่องจากตัวอย่างในรูปที่ 2.5 เป็นระบบที่มีการทำงานที่แน่นอน (Deterministic) กล่าวคือ ในแต่ละ step มี binding ที่เป็นไปได้เพียงชุดเดียว ทำให้ไม่เห็นความสามารถของ Petri Nets สมมติว่าเราเปลี่ยน token ใน Track4 ให้เป็น (“7T”, TrainDOWN) ดังแสดงในรูปที่ 2.7 ระบบจะมีลักษณะการทำงานแบบไม่แน่นอน (non-deterministic) กล่าวคือ มีทางเลือกระหว่าง TrainUP เคลื่อนไปที่ Track2 หรือ TrainDOWN เคลื่อนไปที่ Track3 เมื่อสร้าง state space แล้วจะได้ดังรูปที่ 2.8 ในแต่ละ step ระบบจะมีทางเลือกและมี binding 2ชุด และระบบเมื่อทำงาน ไป จะมีผลลัพธ์สุดท้ายหรือที่เรียกว่า Terminal markings ที่เป็นไปได้ 3 แบบ



รูปที่ 2.7 แสดงแบบจำลอง CPN ของรูปที่ 2.5 แต่เปลี่ยนค่า token ใน place Track4



รูปที่ 2.8 แสดง State space diagram ของแบบจำลองในรูปที่ 2.7

บทที่ 3

Non-hierarchical Coloured Petri Nets

ในบทนี้ จะกล่าวถึงทฤษฎี และหลักการเบื้องต้น ของการ สร้างโมเดลพื้นฐานโดยใช้ Coloured Petri Nets ในการรันโปรแกรมตัวอย่างประกอบกับการตั้งค่าของโปรโตคอลที่ใช้ใน การสื่อสารอย่างง่าย โปรโตคอลที่ใช้เนื่องจากสะดวกในการอธิบายและเข้าใจ และเนื่องจากเกี่ยว กับ concurrency, non-determinism การสื่อสารและ synchronisation ซึ่งเป็นลักษณะสำคัญของระบบ

3.1 ตัวอย่างง่ายๆ ของโปรโตคอล

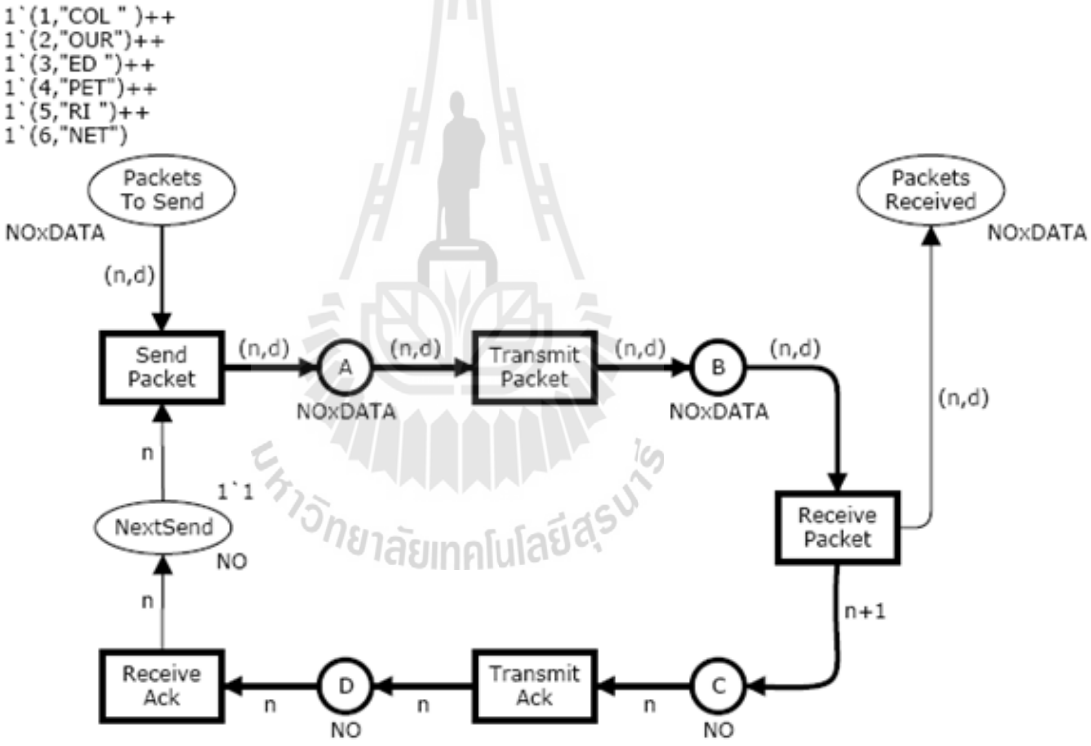
เราพิจารณาโปรโตคอลแบบง่ายๆจากชั้น transport layer ของ Open Systems Interconnection (OSI) ชั้น transport layer จะมีความน่าเชื่อถือได้ว่าการสื่อสารทั้งสองฝั่งจะติดต่อกันได้ดี โปรโตคอลคือความเรียบง่าย ตรงไปตรงมาแต่ซับซ้อนพอที่จะแสดง พื้นฐาน CPN constructs

โปรโตคอลแบบง่ายประกอบด้วยผู้ส่งโอนหมายเลขของแพ็ คเก็ตข้อมูลไปยังผู้รับ การสื่อสารจะผ่านทางเครือข่ายที่ไว้ใจไม่ได้ได้แก่แพ็กเกจอาจสูญหายและโปรโตคอลจะใช้หมายเลขลำดับ, acknowledgements, และ retransmissions เพื่อให้มั่นใจว่าแพ็คเก็ตข้อมูลจะถูกนำส่งครั้งเดียวและ ลำดับที่ปลายทางได้รับถูกต้อง โปรโตคอล deploys stop-and-wait strategyได้แก่ข้อมูลแพคเก็ตเดียวกันคือ retransmitted ซ้ำจนกว่า การรับตรงจะได้รับ แพ็คเก็ตข้อมูลที่ประกอบด้วยลำดับหมายเลขและข้อมูล payload. acknowledgement ประกอบด้วยหมายเลขลำดับ ระบุจำนวนของแพ็กเก็ตข้อมูลไปยังผู้รับ

เราเริ่มต้นด้วยการทดลองแบบง่ายมากของโปรโตคอลที่ retransmissions และเราไม่สามารถไว้ใจเครือข่ายได้ แบบคือกลั่นแล้วค่อยๆ เพื่อแนะนำด้านมากขึ้นรวมถึงการสูญเสียแพ็คเก็ตในเครือข่าย ค่อยๆปรับแต่งการท ดลองนี้ใช้เพื่อแสดงสถานที่ต่างๆในแบบจำลอง CPN การสร้างแบบจำลอง CPN มีข้อกำหนดเมื่อ โดยทั่วไปก็คือการปฏิบัติที่ดีในการเริ่มต้นด้วยการทำแบบง่ายๆ, omitting บางส่วนของระบบหรือทำให้สมมติฐานง่ายและทำการทดลองแบบ CPN เป็น แล้วค่อยๆปรับปรุงและขยายสมมติฐานและเพิ่มส่วนที่ถูกตัดออกของระบบ

3.2 โครงสร้างของเน็ทและ ข้อมูลที่เขียนลงในโครงสร้าง

แบบจำลอง CPN มีการสร้างเป็นภาพวาด กราฟฟิก ดังรูปที่ 3.1 เป็นแบบจำลองของโปรโตคอลแบบง่ายๆ ซึ่งซ้ายมือในแบบจำลองเป็นส่วนของ ผู้ส่ง ส่วนแบบจำลองในส่วนกลางเป็นเครือข่ายและแบบจำลองส่วนสุดท้ายเป็นส่วนของผู้รับ แบบจำลอง CPN จะบรรจุด้วย 7 เพลส (place) ซึ่งที่วาดเป็นวงรีหรือวงกลม จะมี 5 ทรานสิชั่น (transition) วาดเป็นกล่องรูปสี่เหลี่ยม และก็จำนวนเส้นทางของ อาร์ค (arcs) ที่ใช้เชื่อมต่อเพลสกับทรานสิชัน และสุดท้ายจะเกี่ยวกับข้อความใน อาร์ค , ทรานสิชัน และเพลส ข้อความที่เขียนลงในนั้น ก็คือ การเขียนภาษาใน CPN ML(Coloured Petri Nets Model)



รูปที่ 3.1 First CPN model of the simple protocol

เพลสทรานสิชันจะเรียกว่าโนด (Node) ซึ่งจะร่วมเส้นทางของอาร์ค ซึ่งจะประกอบกันเป็นโครงสร้างของเน็ท (Net) อาร์ปกติจะเชื่อมต่อจากเพลสไปยังทรานสิชัน หรือจากทรานสสิชันไปยังเพลส ซึ่งมันจะผิดปกติก็ต่อเมื่ออาร์คอยู่ระหว่าง 2 โนดที่เหมื่อนกัน

เพลสจะใช้แทนสเตจ (State) ของระบบโมเดล แต่ละเพลสจะสามารถสังเกตจากโทคเกน (token) ซึ่งอาจมีหนึ่งอันหรือมากกว่านั้น ซึ่งแต่ละโทคเ กนจะมีค่าของข้อมูลเชื่อมต่ออยู่ซึ่งค่าของข้อทูลจะเรียกว่า โทคเกน คัลเลอร์ (token colour) ซึ่งก็คื่อตัวเลขของโทคเกน และโทคเกนคัลเลอร์ที่อยู่บนเพลสเดี่ยวๆ ซึ่งแสดงร่วมอยู่ในสเตจของระบบ ซึ่งเรียกว่า มาร์คกิ้ง (Marking) ของคัลเลอร์เพทติ-เน็ท โมเดล

โดยเราจะเ ขียนชื่อของเพลสไว้ข้างในวงรี ซึ่งชื่อจะไม่มีความหมาย ซึ่งมันควรจะมีความเหมาะสมที่สำคัญที่ให้อ่าน โมเดลขนาดใหญ่ของคัลเลอร์ เพทติ-เน็ท โมเดล (ซึ่งจะคล้ายกับการใช้จำชื่อในการเขียนโปรแกรม) ถ้าสังเกตจะเห็นว่ามันคล้ายๆกับการประยุกต์ใช้ในการเขียนกราฟฟิกที่อยู่ในเพลส ตัวอย่างเช่น เส้นบาง , ขนาด, สี และตำแหน่ง โดยสเตจ ฝั่งส่งคือ โมเดลโดยมีเพลส Packet Received ส่วนสเตจของเครือข่ายก็คือ เพลส A, B, C และ D

ถัดมาก็คือ เพลส แต่ละอันที่เราจะหาข้อมูล ที่เขียนลงไป เพื่อที่จะกำหนดค่าของโทคเกนคัลเลอร์ (ค่าข้อมูล) ที่โทคเกนนั้นอยู่บนเพลสที่อนุญาตให้มีการตั้งค่าที่เป็นไปได้ของ โทคเกน คัล เลอร์ ซึ่งก็คือ ลักษณะเฉพาะโดยหมายถึงชนิดซึ่งจะเรียกว่า colour set ของเพลส colour set จะเขียนใต้เพลส เพลส NextSend, C และ D มี colour set NO colour set สามารถอธิบายโดยใช้ colset CPN ML และcolour set NO คือการตั้งค่าของจำนวนเต็ม (int)

```
colset NO = int;
```

นี่หมายถึง โทคเกน ที่อาศัยบนเพลส 3 เพลส คือ NextSend, C และ D จะมี จำนวนเต็มคล้ายกับโทคเกน คัลเลอร์ Colour Set NO ใช้กับโมเดลที่มีซีเคว้นนัมเบอร์ (Sequence number) ในโปรโตคอลยังมี 4 เพลสที่เหลืออยู่ก็จะมี Colour Set NOxDATA ซึ่งจะกำหนดให้เป็น การคูณของ NO และ DATA ซึ่ง ไทน์ (type) นี้จะบรรจุเป็นคู่ ที่ซึ่งอิลิเม้นท์ตัวแรกเป็นอินทีเจอร์ (integer) และอิลิเม้นท์ตัวที่สอง คือ สตริง (String) ซึ่งก็คือพวกข้อความ ซึ่งทั้งคู่จะเขียนโดยใช้วงเล็ บรอบๆ และใช้ Comma เป็นตัวแบ่ง Colour Set สามารถอธิบายได้ดังนี้

```
colset DATA = string;
colset NOxDATA = product NO * DATA;
```

คัลเลอร์เซ็ท DATA ใช้โมเดลในการบรรจุข้อมูลที่เป็นแพ็คเก็ต และ อธิบายถึง การกำหนดค่าทั้งหมดของข้อความที่เป็น สตริง colour set NOxDATA คือการ ใช้แบบจำลอง แพ็คเก็ตข้อมูลซึ่งประกอบไปด้วย ซีเคว้น นัมเบอร์และข้อมูลบางอย่าง

การกำหนดค่าที่อยู่ข้างบนทางด้านขวาเพลส NextSend ลักษณะเฉพาะนั่นก็คือ Initial Marking ของเพลสนี้ประกอบด้วย 1 โทคเกน กับค่าเท่ากับ 1 ซึ่งแสดงให้รู้ว่า เลขหนึ่งคือข้อมูล ที่เราต้องการส่งเป็นลำดับแรก การกำหนดค่าไว้ที่ด้านซ้ายบนของเพลส PacketsToSend placeคือ

```
1‘(1,"COL") ++
1‘(2,"OUR") ++
1‘(3,"ED ") ++
1‘(4,"PET") ++
1‘(5,"RI ") ++
1‘(6,"NET")
```

ลักษณะเฉพาะของ Initial Marking ของเพลสที่บรรจุ 6 โทคเกนกับค่าของข้อมูล

```
(1,"COL"),
(2,"OUR"),
(3,"ED "),
(4,"PET"),
(5,"RI "),
(6,"NET")
```

สัญลักษณ์ ++ และ ' คือการประกาศตัวแปรที่ยอมให้ใช้ในโครงสร้างของ multiset ประกอบด้วย 6 โทคเกน Colour Multiset คล้ายกับการตั้งค่า นอกจากนั้นค่าจะปรากฏมากกว่าหนึ่งครั้ง การประกาศตัวแปรโดยการใส่ ' ที่หน้าจำนวนเต็มบวกทางด้านซ้าย ซึ่งก็คือลักษณะเฉพาะที่เป็นข้อสรุปของตัวเลขที่เกิดขึ้นของ argument ที่จัดให้ initial marking ของ Packet To Send จะประกอบด้วย 6 โทคเกนซึ่งจะใช้แทนข้อมูลที่เร าต้องการที่จะส่ง initial marking โดยปกติเราจะเขียนไว้ของบนเพลส การไม่กำหนดค่าลักษณะเฉพาะ initial marking หมายถึงเพลสที่ไม่มีโทคเกนอยู่ ซึ่งในที่นี้ก็คือเพลส A, B, C, D และ Packets Received

5 ทรานสิชัน (เขียนเป็นสี่เหลี่ยม) แสดงให้เห็นถึง เหตุการณ์ที่สามารถ ใช้เพลสใน ระบบเช่นเดียวกับที่ชื่อของทรานสิชันภายในสี่เหลี่ยม ชื่อของทรานสิชัน จะไม่มีรูปแบบหรือความหมายที่ตายตัวแต่ก็จะมีความสำคัญมากสำหรับการอ่านโมเดล เมื่อเกิดทรานสิชัน จะย้ายไปโทคเกนจากอินพุทเพลส (input place) จากเพลสนั้นจะมีอาร์คที่นำไปถึงทรานสิชันและจะไปเพิ่มโทคเกน ที่เอาท์พุต เพลส (output place) จากเพลสนั้นจะมีอาร์คที่มาจากทรานสิชัน คัลเลอร์ของโทคเกนจะมี

การย้ายจาก อินพุท เพลส และเพิ่มเอาท์พุทเพลส เมื่อทรานสิชันเกิดขึ้น คือ การกำหนดโดยใช้ความหมายของอาร์ค เอ็กเพลสชั่น (arc expression) ซึ่งคือการกำหนดค่าของตำแหน่งถัดไปที่ อินดิวิดัล อาร์ค (individual arc)

อาร์ค เอ็กเพลสชั่น คือ การ เขียนในภาษาโปรแกรม CPN ML และ การสร้าง ตัวแปร constants, operators และ functions เมื่อตัวแปรทั้งหมด ขอบเขตของ ค่า สามารถประเมิน ได้ ตัวอย่างเช่นเราจะ พิจารณา 2 อาร์ค เอ็กเพลสชัน n และ (n, d) บน อาร์คทั้งสามที่ เชื่อมต่อ ไปที่ ทรานสิชัน SendPacket ซึ่งจะบรรจุตัวแปร n และ d ซึ่งประกาศไว้ดังนี้

var n : NO;

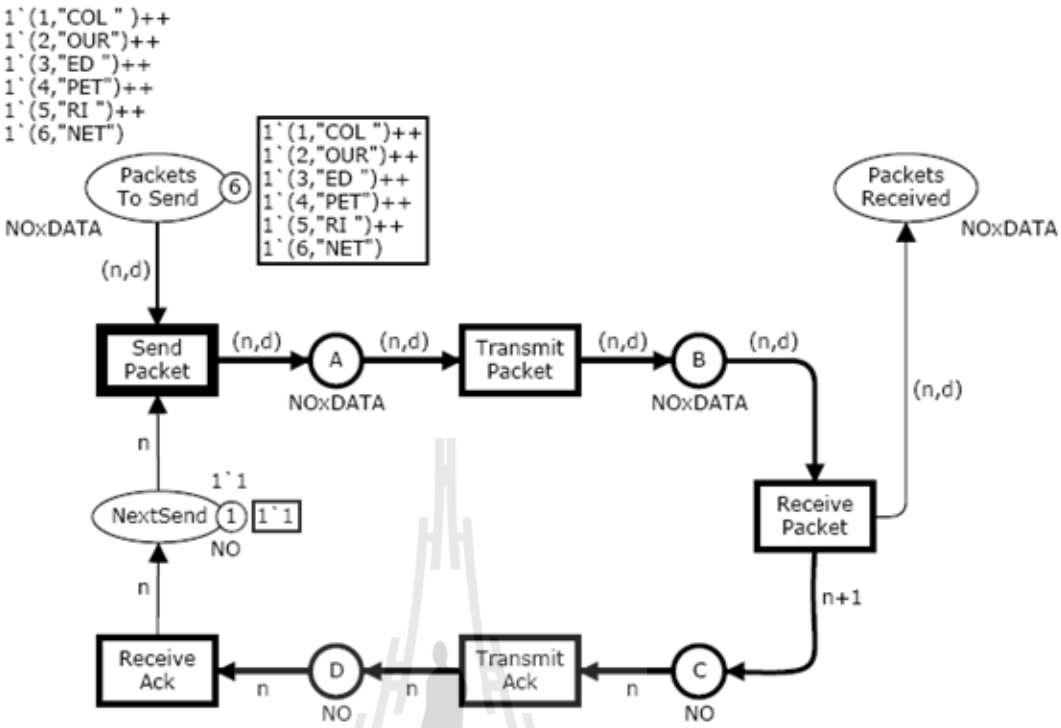
var d : DATA;

ซึ่งหมายความว่า ของ n จะต้องกำหนดขอบเขตค่าของไทน์ NO (ตัวอย่างเช่น จำนวนเต็ม) ในขณะที่ d ต้องกำหนดขอบเขตของค่าไทน์ DATA (ตัวอย่างเช่น text, string) ซึ่งตัวอย่างที่เราอาจจะพิจารณาเช่น บายกิ้ง (binging)

<n=3, d="CPN">

3.3 ความเป็นไปได้และการเกิดขึ้นของทรานสิชัน

ต่อไปพิจารณารูป 3.2 ซึ่งจะแสดงแบบ จำลอง โปรโตคอล กับ initial marking ของ M_0 marking ของแต่ละเพลส คือการแสดงให้เห็นว่า ถัดจากเพลสนี้ไป ไปที่เพลสตัวเลขของโทคเกนที่อยู่บนเพลสจะแสดงอยู่ในวงกลมเล็กๆ ในขณะที่รายละเอียดของโทคเกน คัลเลอร์จะแสดงในกล่องสี่เหลี่ยมที่อยู่ตำแหน่งถัดไปจากวงกลมเล็กๆ ซึ่งสามารถ อธิบายได้ง่าย คือ initial marking จะมี 6 โทคเกน บน PacketsToSend และมี 1 โทคเกน บน NextSend ซึ่งบนเพลสอื่น ๆทั้งหมดจะไม่มีโทคเกน



รูปที่ 3.2 Initial marking M_0

อาร์ค เอ็กเพลสชัน บน อินพุท อาร์ค ของทรานสิชันจะกำหนดว่า ทรานสิชันไหนจะ เอนนาเบิล (ennable) ได้จะต้องหาตัวแปรต่างๆมา binding ซึ่งตัวแปรเหล่านี้มาจาก อาร์ค อ็กเพลสชัน ที่อยู่รอบทรานสิชันนั้น ตวแปรอินพุทเพลส ที่เป็นเงื่อนไขจะต้องมี mutiset โทคเกน ตรงกับอาร์คเอ็กเพลสชัน เมื่อทรานสิชันเกิดเหตุการณ์ขึ้น ตัวเหตุการณ์นั้นก็ต้องดูว่าเป็น binding ชุดไหน binding จะเป็นตัวดึงโทคเกนออกจากอินพุทเพลส ตามจำนวนขนาด ที่ระบุไว้บน อาร์ค เอ็กเพรสชัน ทำนองเดียวกัน ทรานสิชันจะใส่โทคเกนบน เอาท์พุทเพลส ตามจำนวนชนิดที่ระบุไว้บน อาร์ค เอ็กเพรสชัน

ต่อไปจะพิจารณา ทรานสิชัน SendPacket ในรูปที่ 3.2 เราจะสังเกตทรานสิชัน SendPacket ซึ่งจะมีเส้นทึบหนาๆ ในขณะที่อีก 4 ทรานสิชันอื่นจะ ไม่มี ซึ่งแสดงให้เห็นว่า SendPacket เพียงอันเดียวเท่านั้นที่สามารถสร้าง binding ใน marking M_0 ได้ ส่วนทรานสิชันอื่นๆ ไม่สามารถทำให้เกิดขึ้นได้เพราะไม่มีโทคเกนบนอินพุทเพลส เหล่านั้น เมื่อทรานสิชันนี้เกิดขึ้น จะย้ายโทคเกนจากอินพุทเพลส Next Send และ PacketsToSend อาร์ค เอ็กเพลสชันของ 2 อินพุท อาร์คคือ n และ (n, d) อธิบายได้คือ

var n : NO;

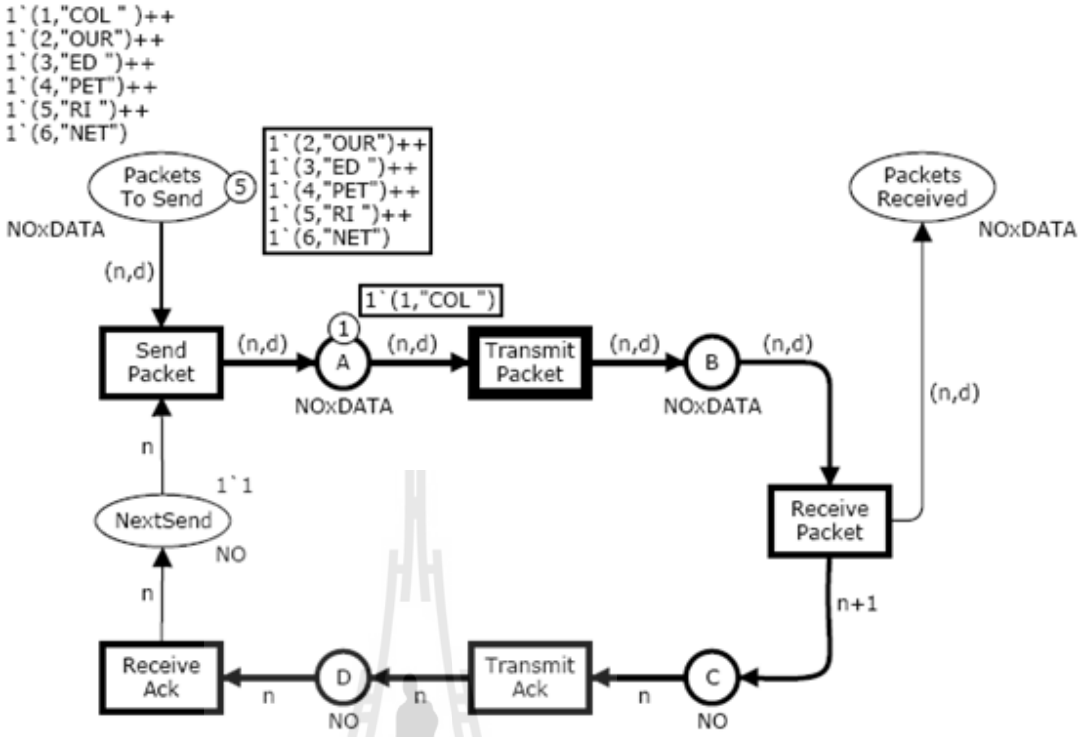
var d : DATA;

initial marking ของเพลส NextSend จะบรรจุด้วยโทคเกนเดี่ยวๆ กับคัลเลอร์ ซึ่งหมายความว่าจะต้องเชื่อมไปที่ เอ็กเพรสชัน บน อาร์ค จาก Next Send ควรจะประเมิน ค่าเ โทคเกน คัลเลอร์ ซึ่งไม่มีอยู่ที่ Next Send มีความหมายว่า ทรานสิชันนั้นไม่สามารถสร้าง binding ได้ต่อไป เราจะ พิจารณา อาร์ค เอ็กเพรส (n, d) บนอินพุทอาร์ค จาก Packets To Send เรามีคู่ของค่า n เท่ากับ 1 เรียบร้อยแล้ว และตอนนี้เรากำลังมองหา binding ของ d ดังนั้น เอ็กเพรสชัน (n, d) จะประเมินค่าเป็น 1 ของ 6 โทคเกน คัลเลอร์ นั้นคือจะแสดงบน Packets To Send เห็นได้ชัดว่าความเป็นไปได้เพียงอย่างเดียว คือ bind d เป็น String "COL" ดังนั้นเราสรุปว่า

<n=1, d="COL">

ซึ่งคือ ทางเดียวที่ทำให้ binding มัน เอนนาเบิลได้จาก SendPacket (ใน initial marking) การเกิดขึ้นขอ งเหตุการณ์ Send Packet กับ binding มันจะย้ายโทคเกน กับ คัลเลอร์ ของข้อมูลหมายเลขหนึ่ง จาก อินพุตเพลส Next Send ย้ายโทคเกนกับคัลเลอร์ (1, "COL") จากอินพุทเพลส Packets To Send จะเพิ่ม โทคเกน อันใหม่กับคัลเลอร์ (1, "COL") ไปที่เอาร์พุท เพลส A Intuitivelyนี้หมายถึงการส่งแพคเก็ตข้อมูลแรก (1, "COL") ที่จะส่ง ไปยังเครือข่าย ซึ่งเราจะบันทึกสิ่งนั้น ซึ่งมันก็คือโทคเกน บน Next Send ซึ่งจะอธิบายข้อมูลที่เป็น แพคเก็ต ที่ส่งออกไป แพ็คเก็ต (1, "COL") อยู่ที่เพลส A กำลังที่จะย้าย โดยเครือข่าย ก็คือ marking M_1 ที่แสดงดังรูป 2.3

ใน marking M_1 ของเพลส A มีโทคเกนกี่ยวกับคัลเลอร์ (1, "COL") ดังนั้นสามารถ สรุปได้ตรงๆ คือ < n = 1, d = " COL "> คือ binding อันเดียวที่สามารถทำให้เอนนาเบิลได้ทรานสิชัน Transmit Packet ใน M_1 เมื่อเกิดทรานสิชัน ขึ้นที่ binding นั้น จะย้ายโทคเกน (1, "COL") จาก A และ เพิ่มโทคเกน ใหม่ ด้วยโทคเกนคัลเลอร์ คล้ายๆกับเพลส B เพื่อส่งข้อมูลแพ็กเก็ตจำนวน 1 ผ่านเครือข่าย แพ็คเก็ตข้อมูลที่เป็นปัจจุบัน ณสถาน ะ B รอที่จะได้รับ marking ใหม่ M2 จะปรากฏในรูปที่ 3.4

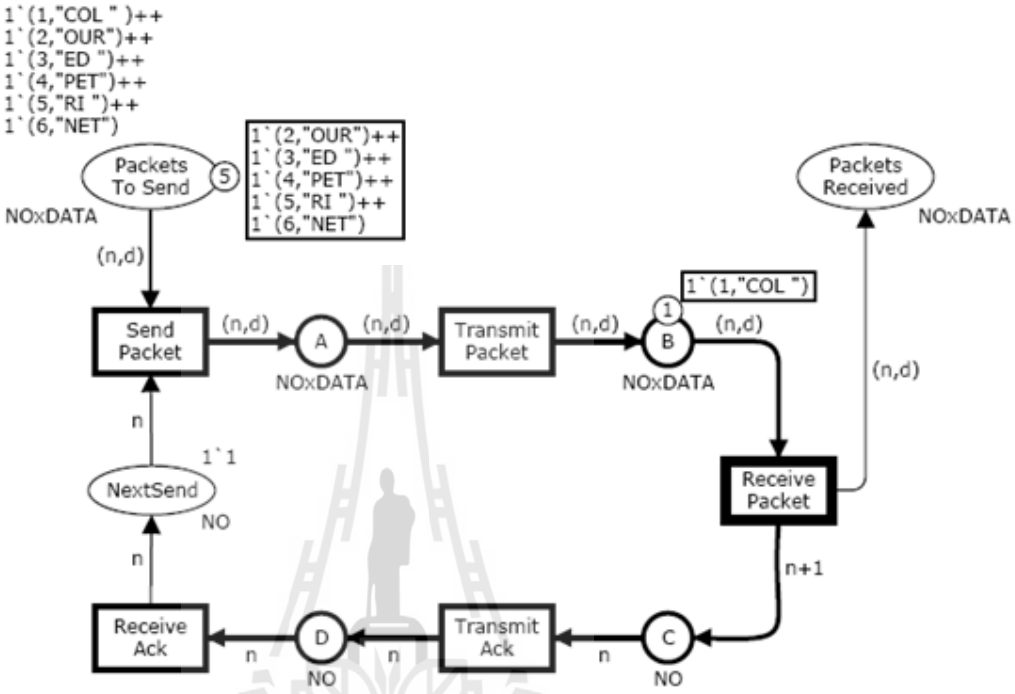


รูปที่ 3.3 Marking M_1 reached when SendPacket occurs in M_0

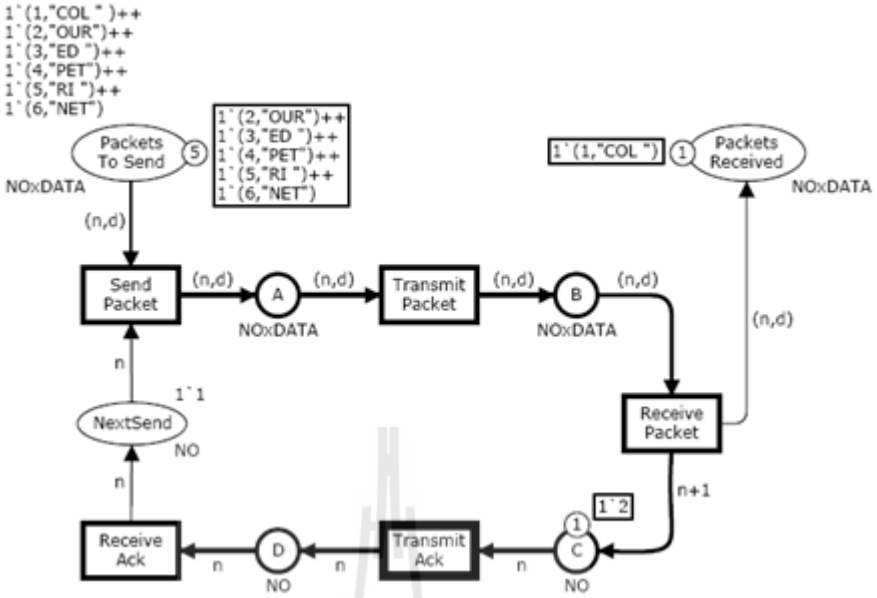
ใน marking M_2 เราจะมีทรานสิชันเดี่ยวๆ enabled คือ Receive Packet และ เราจะใช้ binding <n=1, d = "COL"> เกิดการเปลี่ยนแปลง ย้ายโทคเกน คัลเลอร์ (1, "COL") จากเพลส B และเพิ่ม โทคเกน คัลเลอร์ (1,"COL") ไปที่ Packets Received และเพิ่ม โทคเกนกับคัลเลอร์ ที่มีค่าเท่ากับ 2 ไปที่เพลส C โทคเกนคัลเลอร์ที่ C กลายเป็น 2 ตั้งแต่ อาร์ค เอ็กเพรสชัน n+1 บน อาร์คจาก ReceivePacket ไปยัง C ซึ่งทำให้มีค่าเท่ากับ 2 ใ นการ binding ด้านบนอย่าง Intuitively ระบุได้ว่า แพ็คเก็ตข้อมูลของข้อมูลหมายเลขหนึ่งอยู่ที่ฝ่ายรับแล้ว เมื่อรับแพคเก็ตจะถูกจัดเก็บในเพลส PacketsReceived โทคเกนบน C หมายถึงการรับ ส่งจากที่ผู้ส่งเพื่อยืนยันการรับจำนวน 1 แพ็คเก็ตข้อมูลและเพื่อขอหมายเลขแพคเก็ตข้อมูล ที่มีค่าเท่ากับ ซึ่งก็คือ marking อันใหม่ (marking M_3) ซึ่งแสดงในรูปที่ 3.5

ใน marking M_3 เรามีทรานสิชันที่เอนนาเบิลเดี่ยวๆ คือ Transmit Ack ซึ่งเราจะ ใช้ binding <n = 2> โดย Intuitively นี้หมายถึงการส่งผ่านเครือข่าย การแจ้งขอเลขที่แพคเก็ตข้อมูล 2 marking ใหม่ M_4 คือ ที่แสดงในรูป 3.6 ใน marking M_4 เรามีทรานสิชันที่เอนนาเบิลเดี่ยวๆ คือ ReceiveAck และเราจะใช้ binding <n = 2> ใน marking อันใหม่ M_5 จะแสดงในรูปที่ 3.7 marking นี้แสดงถึง

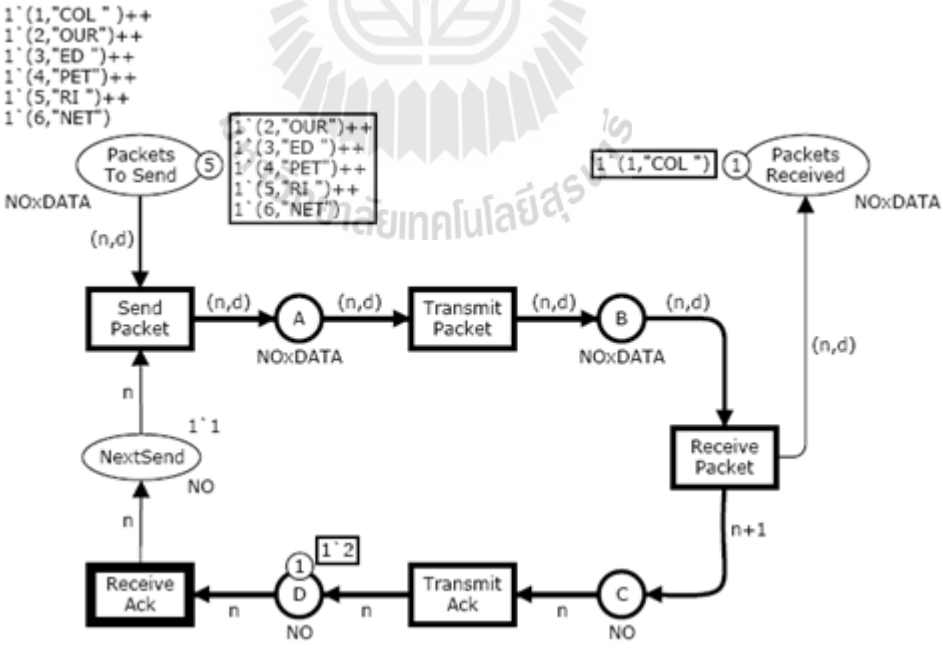
สภาวะที่ผู้ส่งพร้อมจะส่งจำนวนแพ็คเก็ตข้อมูล เท่ากับ 2 (ตั้งแต่แพ็คเก็ตข้อมูลแรก คือบอกให้รู้ว่าตอนนี้ได้รับข้อมูลเรียบร้อยแล้วสำเร็จ)



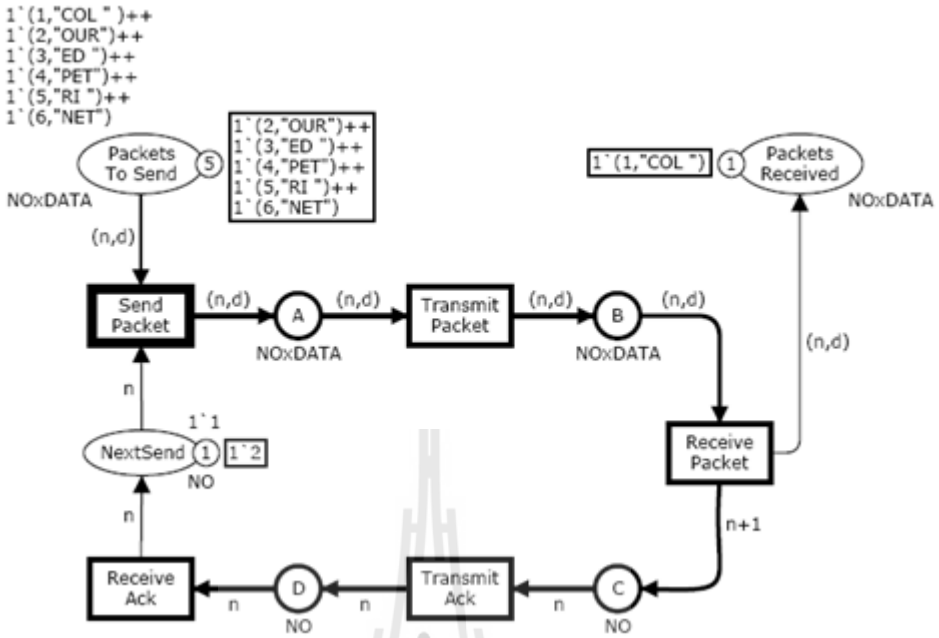
รูปที่ 3.4 Marking M_2 reached when TransmitPacket occurs in M_1



รูปที่ 3.5 Marking M_3 reached when ReceivePacket occurs in M_2



รูปที่ 3.6 Marking M_4 reached when TransmitAck occurs in M_3



รูปที่ 3.7 Marking M_5 reached when ReceiveAck occurs in M_4

ในข้างต้นเราได้อธิบายการส่ง การส่งและรับข้อมูลจำนวน 1 แพคเก็ค และรับทราบตรงกัน ในแบบจำลอง CPN นี้จะมี 5 ขั้นตอนซึ่งแต่ละขั้นตอนจะเกิดทรานสิชันใน binding มีอยู่ห้าขั้นตอนซึ่งแต่ละขั้นตอนเขียนเป็นคู่ๆ ของทรานสิชันและเกิดขึ้นของ binding ของทรานสิชันซึ่งแต่ละคู่เราจะเรียกว่า binding element

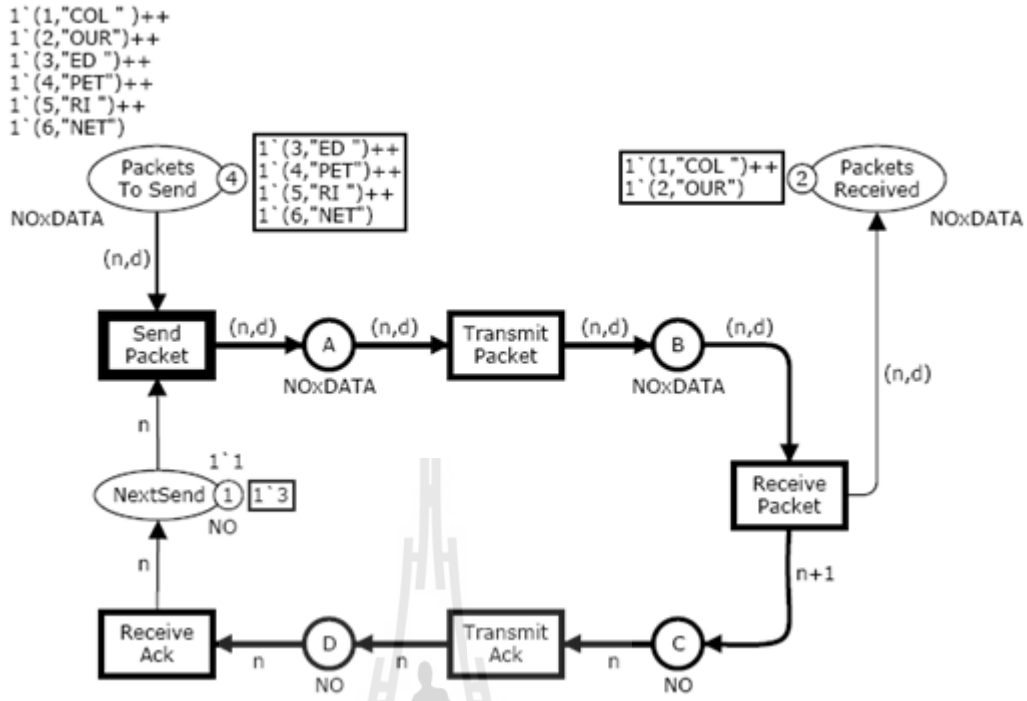
Step	Binding element
1	(SendPacket, ⟨n=1, d="COL"⟩)
2	(TransmitPacket, ⟨n=1, d="COL"⟩)
3	(ReceivePacket, ⟨n=1, d="COL"⟩)
4	(TransmitAck, ⟨n=2⟩)
5	(ReceiveAck, ⟨n=2⟩)

มันจะเป็นการง่ายที่จะเข้าใจใน 5 ขั้นตอนต่อไป ซึ่งมันจะคล้ายๆกับ 5 ขั้นตอนแรก ยกเว้นจะอธิบายอธิบายการส่ง การส่ง และรับข้อมูลจำนวน 2 แพ็คเก็ตและ acknowledgement

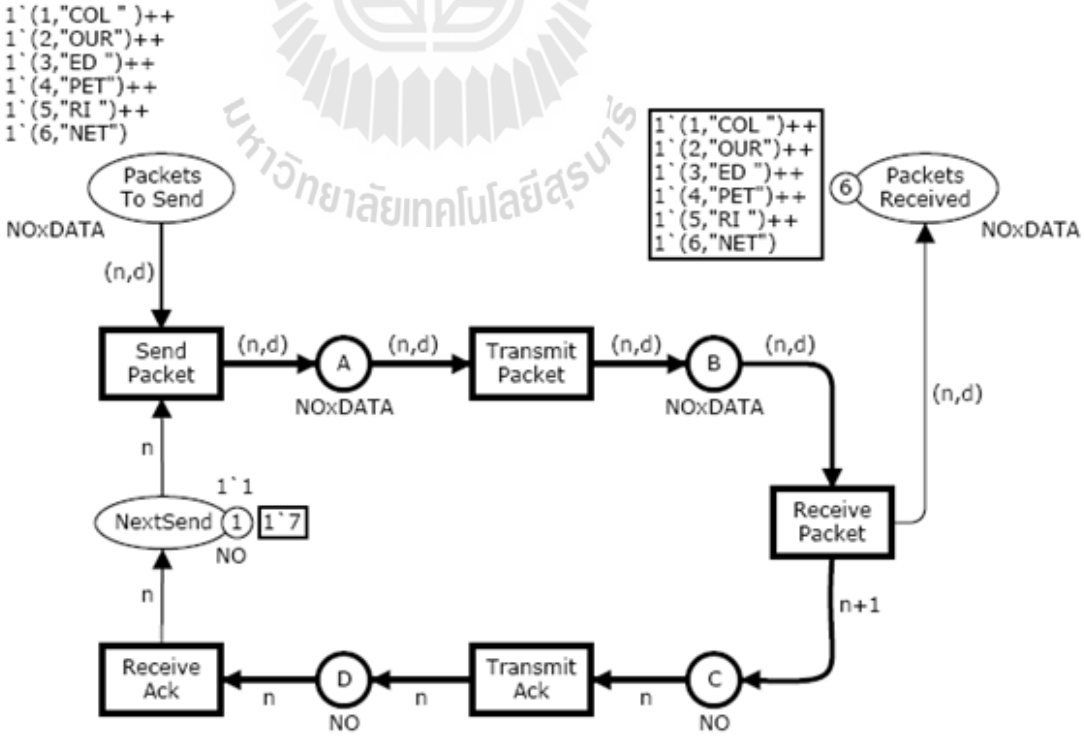
Step	Binding element
6	(SendPacket, ⟨n=2, d="OUR"⟩)
7	(TransmitPacket, ⟨n=2, d="OUR"⟩)
8	(ReceivePacket, ⟨n=2, d="OUR"⟩)
9	(TransmitAck, ⟨n=3⟩)
10	(ReceiveAck, ⟨n=3⟩)

หลังจากนี้อีก 5 ขั้นตอนเราเข้า ใจ marking M_{10} ซึ่งแสดงในรูป ที่ 2.8 ต่อไปเราจะมี 5 ขั้นตอน สำหรับ แพ็คเก็ต ข้อมูลที่มีค่าเท่ากับ 3 และ acknowledgement จากนั้นก็จะเป็น 5 ขั้นตอน สำหรับแพ็คเก็ต ข้อมูลที่มีค่าเท่ากับ 4,5 ขั้นตอน สำหรับแพ็คเก็ตข้อมูล ที่มีค่าเท่ากับ 5 และใน ตอนสุดท้าย 5 ขั้นตอนสำหรับแพ็คเก็ตข้อมูลที่มีค่าเท่ากับ 6 หลังจากนี้ คือ ขั้นตอน ที่เราจะทำ ความเข้าใจ marking M_{30} ซึ่งแสดงในรูปที่ 2.9 markingนี้หมายถึงสเตจของโปร โตคอลที่แพ็คเก็ต ข้อมูล ทั้งหมดได้รับ acknowledgements ทั้งหมด ผู้ส่งและ ไม่มีแพ็คเก็ตใดออกนอกเครือข่ายเ ลย marking นี้ไม่ใช่ทรานสิชัน ที่เอนนาเบิลเลย และดังนั้นเราจะเรียกว่า dead marking

นี้เป็นการ ดำเนินกา รแบบจำลอง CPN ง่ายมาก protocol แบบจำลอง นี้เป็น deterministic ซึ่งสามารถสรุปได้ว่า มีเพียงเส้นทางเดียวเท่านั้นที่เป็น ไปได้ของ marking M0, M1, M2,. . . , M30 และ 30 ขั้นตอนที่อธิบายข้างต้น



รูปที่ 3.8 Marking M_{10} reached after transmission of data packet number 2



รูปที่3.9 Dead marking M_{30} reached when all packet have been transmitted

บทที่4

DCCP และ NAT

4.1 แนะนำ DCCP

DCCP คือโปรโตคอลที่มีการติดต่อแบบจุดต่อจุดซึ่งอยู่ในชั้น Transport มีการติดต่อระหว่างสององค์กรคือ Client และ server โดยจะมีการส่งข้อมูลสำหรับการใช้งานที่มีความตรงต่อเวลาทำให้เกิดความน่าเชื่อถือ ซึ่งเวลาที่มีการกำหนดไว้จะไม่ไปขัดขวางการรับข้อมูลที่ส่งมา

DCCP มีการส่งข้อมูลได้มากมาย สำหรับตัวอย่างในเกมส์ออนไลน์ ข้อมูลจะส่งจาก server ไปยัง client ซึ่งจะเกิดความล้าช้าของภาพเคลื่อนไหวและเสียง แต่ข้อมูลที่ส่งจาก client ไป server จะมีการควบคุมซึ่งมีเพียง commands เท่านั้น เพราะว่า commands เป็นเพียงข้อความสั้น ๆ และสามารถสูญหายได้ มันจึงมีความง่ายในการควบคุมความแออัดโดยใช้ UDP ในทางกลับกันภาพเคลื่อนไหวและเสียงต้องการแบนด์วิดที่มีอยู่กับข้อความที่มีความยาวได้อย่างรวดเร็ว

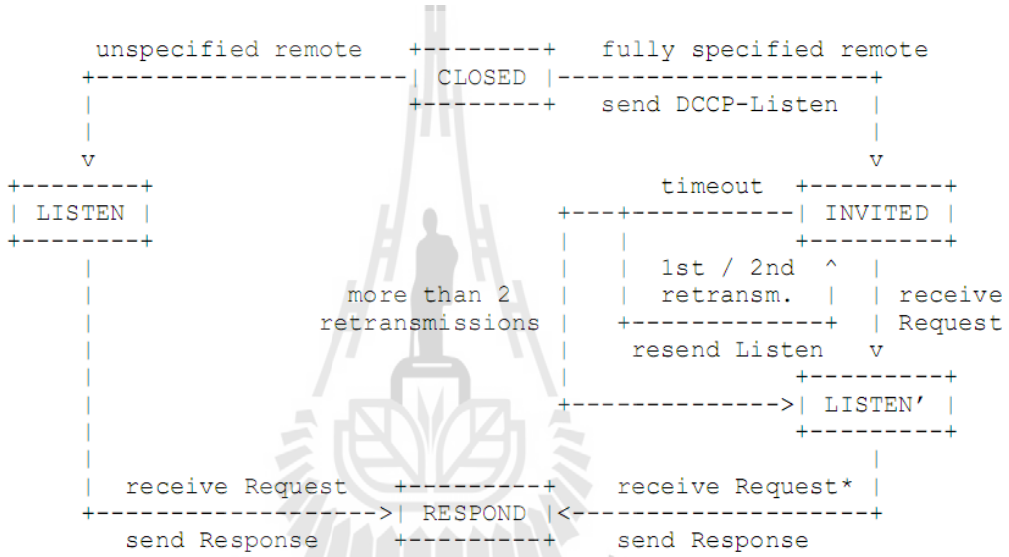
ดังนั้น DCCP จะแนะนำแนวคิดการติดต่อแบบครึ่ง แต่ละการติดต่อจะแบ่งการติดต่อแบบครึ่งออกเป็นสองส่วน การติดต่อแบบครึ่งส่วนแรกประกอบด้วยแพ็กเก็ตของข้อมูลโดยส่งจากด้านหนึ่ง และ corresponding acknowledgements ส่งจากทางด้านอื่น และการติดต่อแบบครึ่งสามารถควบคุมความแออัดของข้อมูลที่มีความแตกต่างกันได้ โดย DCCP ควบคุมความแออัดโดยใช้ Congestion Control Identification (CCID) ส่วนสำคัญคือสั้งสองต้องรู้จักและยอมรับใน CCID เดียวกัน สำหรับแต่ละครั้ งในการติดต่อ โดย RFC 4340 จะเป็นตัวกำหนดคุณลักษณะทั่วไปของ CCID บางอย่างจะระบุการรับข้อมูล คือ ใช้ข่าวสารที่มีความแออัดส่งกลับไปยังผู้ส่ง สำหรับกลไกนี้ ข้อมูลที่ถูกส่งกลับเรียกว่า Ack ซึ่งสามารถแสดงแพ็กเก็ตที่สูญหายได้ ข้อกำหนดของการควบคุมความแออัดและกลไกการส่งข้อมูลกลับสามารถหาข้อมูลที่เกี่ยวข้องได้ใน RFCS(e.g. RFC 4341 และ RFC 4342)

ในขณะที่ TCP จะให้ข้อมูลแบบ byte โดยที่ DCCP จะเป็นแบบ datagrame นั่นหมายความว่าข้อมูลอาจสูญหาย , เสียหาย , ล่าช้า, เกิดการซ้ำของข้อมูล หรือได้รับลำดับผิดไป โดยปกติแล้ว datagrame จะนำข้อมูลโดยใช้โปรโตคอลในการติดต่อ เช่น UDP,IP (Internet Protocol) และ ICMP (Internet Control Message Protocol) ในทางตรงกันข้าม DCCP มีการเชื่อมต่อแบบโปรโตคอลโดยเหตุผลนี้

DCCP ถูกออกแบบมาเพื่อควบคุมความแออัดเหนือ streaming media . การควบคุมความแออัดจำเป็นต้องทราบข้อมูลที่สูญหายไป เพราะ มันเกี่ยวข้องกับชั้นของการเชื่อมต่อในเครือข่าย การตรวจสอบรายงานและข้อมูลที่สูญหาย

DCCP จะใช้ลำดับของแพ็กแก็ตและหมายเลขของข้อมูลที่ส่งกลับ เ พื่อติดตามลำดับและหมายเลขข้อมูลที่ส่งกลับ DCCP จึงจำเป็นต้องมีค่าของตัวแปรทั้งสองที่เก็บค่าไว้ ตัวแปรทั้งสองต้อง Synchronised มิฉะนั้น DCCP อาจตีความข้อมูลที่สูญหายผิดไป ดังนั้น DCCP จึงต้องการกลไกในการตั้งค่า Synchronise และทำความสะอาดค่าที่เก็บไว้ในส่วนท้าย

DCCP สามารถถือได้ว่าเป็นเวอร์ชันที่มีการอัพเดทของ UDP เป็นการเชื่อมต่อในกระบวนการ Synchronisation มีการพูดถึงคุณลักษณะและกลไกการควบคุมความแออัด อย่างไรก็ตาม การเชื่อมต่อในกระบวนการ Synchronisation จะกล่าวใน RFC 4340



รูปภาพที่ 4.1 Updated state transition diagram for DCCP-Listen

จากส่วนของรูปทางด้านขวามือเป็นส่วนที่แสดงถึงการทำงานของโปรโตคอล DCCP ซึ่งเพิ่มเติมมาจากโปรโตคอล UDP โดยมี State เพิ่มขึ้นมา 2 State คือ INVITED State และ LISTEN 1 State โดยในส่วนนี้จะมีการส่ง DCCP-Listen ไปยัง INVITED State แล้วส่ง Request ไปยัง LISTEN 1 State

ซึ่งในส่วนนี้เราได้นำไปเพิ่มเติมในแบบจำลอง DCCP ในหน้าของ Server โดยมีทรานซิชั่นที่เพิ่มขึ้นจากต้นแบบอีก 6 ทรานซิชั่น คือ

1) ทรานซิชั่น send_L เป็นทรานซิชั่นที่ส่ง DCCP-Listen ที่รับมาจาก CLOSED State และส่งไปยัง INVITED State
2) ทรานซิชั่น re_Req เป็นทรานซิชั่นที่รับ Request มาจาก INVITED State และส่งไปยัง LISTEN 1 State

3) ทรานซิชั่น Send_r เป็นทรานซิชั่นที่รับ Request มาจาก LISTEN 1 State และส่งไปยัง RESPOND State
4) ทรานซิชั่น Reset_1
5) ทรานซิชั่น Reset_2
6) ทรานซิชั่น Reset_3

โดยที่ ทรานซิชั่น Reset_1, Reset_2 และ Reset_3 จะเกี่ยวข้องกับการส่งกลับซึ่งมี 3 กรณี โดยถ้าดูจากทรานซิชั่นทั้ง 3 แล้ว ค่าที่รับและส่งที่เป็นไปได้จะมีดังนี้

Lis_or_invit = InvitedState 0

Lis_or_invit = InvitedState 1

Lis_or_invit = IdleState LISTEN1

ในส่วนของ INVITED State และ LISTEN 1 State จะเรียกการทำงานนี้ว่า fully specified ก็คือ การรู้เบอร์ของฝั่งตรงข้าม รู้ว่าต้องการคุยกับใครโดยผ่าน NAT

4.2 NAT คืออะไร

Network Address Translation (NAT) คือวิธีการทางเครือข่ายที่จะเปลี่ยนค่า Network Address จากหมายเลขหนึ่งไปเป็นอีกหมายเลขหนึ่ง ซึ่งทำให้เกิดการเชื่อมต่อไปยังเครื่องปลายทางได้ โดยเครื่องต้นทางไม่จำเป็นต้องเปลี่ยนแปลงค่าทางเครือข่าย การทำ NAT ช่วยให้การใช้งานเครือข่ายทำได้อย่างมีประสิทธิภาพมากขึ้นกว่าที่เป็นอยู่ รวมทั้งมีส่วนในการรักษาความปลอดภัยในเครือข่ายได้ด้วย

4.2.1 จุดประสงค์ของการทำ NAT

ที่มาของการทำ NAT นั้นเกิดจากความคิดที่จะนำ Private IP ซึ่งเป็นหมายเลขไอพีแอดเดรสที่ใช้สำหรับเครือข่ายเฉพาะ ซึ่งไม่มีการใช้งานข้ามเครือข่ายได้ (ไม่มีการ route ไปยังเครือข่ายอื่นๆ) และนำมาใช้เพื่อแก้ปัญหาการขาดแคลนหมายเลขไอพีแอดเดรสในอนาคตด้วย ซึ่งภายหลัง การทำงานของ NAT สามารถเพิ่มความสามารถในการรักษาความปลอดภัย และนำมาใช้ในการแก้ปัญหาในกรณีที่หมายเลขไอพีแอดเดรสในองค์กรมีจำนวนจำกัด

4.2.2 Private IP Address

เนื่องจากการใช้งานระบบเครือข่ายนั้น ในบางครั้งก็ไม่จำเป็นที่จะต้องเชื่อมต่อกับเครือข่ายอื่นๆ เลย ยกตัวอย่างเช่นเครือข่ายภายในบริษัท ซึ่งจะติดต่อสื่อสารกันเฉพาะภายในบริษัทเท่านั้น ไม่จำเป็นต้องติดต่อกับบริษัทอื่นๆ หรือเครือข่ายอินเทอร์เน็ต แต่การติดต่อสื่อสารเพื่อใช้งานแอปพลิเคชั่นต่างๆ ก็ยังจำเป็นต้องใช้หมายเลขไอพีแอดเดรสเช่นเดียวกัน ปัญหาที่เกิดขึ้นก็คือถ้ามีการจัดแบ่งหมายเลขไอดีแอดเดรสที่มีอยู่ให้กับเครือข่ายในลักษณะนี้ จะทำให้เกิดปัญหาหมายเลขไอดีแอดเดรสไม่เพียงพอ การตรวจสอบและจัดสรรทำได้ยาก รวมถึงการรักษาความปลอดภัยในเครือข่ายจะทำได้ยากขึ้นด้วย

จากปัญหาดังกล่าวองค์กรที่มีชื่อว่า Internet Assigned Number Authority (IANA) ซึ่งเป็นผู้รับผิดชอบ ดูแลในการจัดสรรหมายเลขไอพีแอดเดรสให้กับ บผู้ใช้งานทั่วโลก ได้กำหนดช่วงของหมายเลขไอพีแอดเดรสที่ ทุกๆ คนสามารถนำไปใช้ได้โดยไม่จำเป็นต้องขึ้นทะเบียนก่อนเรียกว่าช่วง Private IP ซึ่งหมายเลขไอพีแอดเดรสในช่วงนี้จะไม่สามารถนำมาเชื่อมต่อกับเครือข่ายอื่นๆ ได้โดยตรง

ช่วงของหมายเลขไอพีแอดเดรสที่เป็น Private IP นั้น จะแบ่งเป็น 3 กลุ่มด้วยกันคือ

1. ช่วงหมายเลข 10.0.0.0 – 10.255.255.255 (10 / 8)
2. ช่วงหมายเลข 172.16.0.0 – 172.32.255.255 (172.16 / 12)
3. ช่วงหมายเลข 192.168.0.0 – 192.168. 255.255 (192.168 / 16)

4.2.3คุณสมบัติของอุปกรณ์ NAT

อุปกรณ์เครือข่าย หรือโปรแกรมที่ใช้ในการทำ NAT จะต้องมีความสามารถในการทำงานต่างๆ เหล่านี้คือ

1. สามารถกำหนดหมายเลขไอพีแอดเดรสได้ (Transparent address assignment)
2. สามารถส่งผ่านแพ็กเก็ตของข้อมูลที่มีการเปลี่ยนแปลงแอดเดรสได้ (Transparent address routing through address transition)
3. สามารถเปลี่ยนแปลงข้อมูลของ ICMP payload ได้ (ICMP error message payload translation)

สามารถกำหนดหมายเลขไอพีแอดเดรสได้ (Transparent address assignment)

อุปกรณ์ที่จะทำ NAT นั้นจะต้องสามารถเปลี่ยนค่าหมายเลขไอพีแอดเดรสของข้อมูลในเครือข่าย ซึ่งเป็นหมายเลขไอพีเอดเดรสในกลุ่มของ Private IP ให้กลายเป็นหมายเลขไอพีแอดเดรสที่ใช้ในเครือข่ายอินเทอร์เน็ต และสามารถเปลี่ยนหมายเลขไอพีแอดเดรสที่ใช้ในเครือข่ายอินเทอร์เน็ตให้กลายเป็นหมายเลขไอพีแอดเดรสในช่วย Private IP ได้อย่างถูกต้อง ซึ่งในบางกรณีอาจจำเป็นต้องเปลี่ยนแปลงค่าข้อมูลในชั้น Transport บางส่วนด้วยเช่นหมายเลขพอร์ตของ TCP และ UDP ในการเปลี่ยนแปลงค่าไอพีแ อดเดรสนั้นสามารถทำได้ 2 แบบคือแบบ Static และแบบ Dynamic

a. static address assignment

เป็นการเปลี่ยนแปลงค่าหมายเลขไอพีแอดเดรสโดยมีการจับคู่กันของหมายเลขไอพีแอดเดรสตลอดการทำงานของอุปกรณ์ ซึ่งจะเปลี่ยนแปลงค่าไอพีแอดเดรสจาก Private IP เป็นหมายเลขไอพีภายนอก และเปลี่ยนจากหมายเลขไอพีแอดเดรสภายนอกเป็น Private IP แบบหนึ่งต่อหนึ่งไปตลอด

b. dynamic address assignment

เป็นการเปลี่ยนแปลงค่าหมายเลขไอพีแอดเดรสโดยมีการจับคู่กันของหมายเลขไอพีแอดเดรสที่เป็น Private IP กับหมายเลขไอพีแอดเดรสภายนอกเพียงชั่วคราวเท่านั้น โดยอุปกรณ์ NAT จะจับคู่หมายเลขไอพีแอดเดรสในช่วงเวลาที่ session มีการเชื่อมต่อกันอยู่เท่านั้น หลังจากที่ใช้งาน session เสร็จเรียบร้อยแล้วจะไม่เก็บข้อมูลการจับคู่นั้นไว้อีก เมื่อมีการเชื่อมต่อกับเครือข่ายภายนอกอีกครั้ง อุปกรณ์ NAT จะเลือกหมายเลขไอพีแอดเดรสภายนอกใหม่อีกครั้งหนึ่ง ซึ่งไม่จำเป็นต้องซ้ำกับหมายเลขเดิม

สามารถส่งผ่านแพ็กเก็ตของข้อมูลที่มีการเปลี่ยนแปลงแอดเดรสได้ (Transparent address routing through address transition)

เนื่องจากอุปกรณ์ที่ทำ NAT นั้นจะอยู่ระหว่างระบบหมายเลขแอดเดรส 2 ระบบคือ Private Address และไอพีแอดเดรสที่จดทะเบียนอย่างถูกต้อง ดังนั้นสิ่งที่อุปกรณ์ NAT จะต้องคำนึงถึงก็คือการทำงานที่ไม่ขัดต่อการทำงานของระบบหมายเลขแอดเดรสทั้งสองระบบ และต้องไม่เป็นปัญหาในการหาเส้นทางและการรับส่งข้อมูลด้วย โดยข้อควรระวังข้อหนึ่งในการใช้อุปกรณ์ NAT คือการป้องกันการส่งข้อมูล routing information ข้ามเครือข่าย (จากเครือข่ายภายนอก ส่งมายังเครือข่ายภายใน หรือจากเครือข่ายภายในส่งไปยังเครือข่ายภายนอก) เนื่องจากจะทำให้ระบบโดยรวมมีปัญหาทันที กระบวนการในการเปลี่ยนหมายเลขไอพีแอดเดรสมีขั้นตอนทั้งหมด 3 ขั้นตอนหลักคือ

ขั้นตอนในการจับคู่หมายเลขไอพีแอดเดรส ขั้นตอนในการเปลี่ยนแปลงหมายเลขไอพีแอดเดรส ขณะที่มีการเชื่อมต่อกันแล้ว และกระบวนการเมื่อสิ้นสุดการทำงาน

a. การทำงานในการจับคู่หมายเลขไอพีแอดเดรส (address binding)

ขั้นตอนนี้เป็นขั้นตอนที่อุปกรณ์ NAT เปลี่ยนแปลงหมายเลขไอพีแอดเดรสจาก Private IP ให้กลายเป็นไอพีที่จดทะเบียนไว้แล้ว และเปลี่ยนแปลงหมายเลขไอพีแอดเดรสจากไอพีที่จดทะเบียนไว้แล้วให้กลายเป็น Privatte IP ซึ่งสามารถทำได้ทั้งแบบ Static และ Dynamic ซึ่งในการ binding นั้นจะมีการเปลี่ยนแปลงหมายเลขไอพีแอ ดเดรสคู่นั้นๆ ไปจนกว่าจะปิดการเชื่อมต่อ กระบวนการนี้เริ่มต้นเมื่อเริ่มต้นการเชื่อมต่อ (ซึ่งยังไม่มีการเชื่อมต่อกันมาก่อน) โดยเครื่องที่ส่งข้อมูลจะส่งข้อมูลผ่านอุปกรณ์ NAT ซึ่งอุปกรณ์ NAT จะมีการกำหนดหมายเลขไอพีแอดเดรสให้กับข้อมูลนั้นใหม่อีก

ครั้งหนึ่ง โดยการกำหนดหมายเลขไอพีแอดเดรสที่มีการจดทะเบียนให้แทน แล้วจะจำไว้ว่าได้มีการจับคู่หมายเลขไอพีแอดเดรสภายนอกอะไร กับหมายเลข Private IP อะไรบ้าง ตัวเลขคู่นี้จะกำหนดไปจนกว่าจะจบการเชื่อมต่อ สำหรับกรณีที่มีการเชื่อมต่อมากกว่า 1 การเชื่อมต่อในช่วงเวลาเดียวกันก็จะมีการจับคู่หมายเลขแอดเดรสแบบเดียวกัน

b. การทำงานขณะมีการเชื่อมต่อกันแล้ว (address lookup and translation)

หลังจากที่ session มีการเชื่อมต่อกันแล้ว เมื่อการส่งข้อมูลถัดๆ มาจะมีการเปลี่ยนแปลงหมายเลขไอพีแอดเดรสโดยใช้วิธีการค้นหาในหน่วยความจำว่าเคยจับคู่กับหมายเลขไอพีแอดเดรสอะไร

c. การทำงานเมื่อสิ้นสุดการเชื่อมต่อ (address unbinding)

เป็นกระบวนการที่เกิดขึ้นเมื่อสิ้นสุดการเชื่อมต่อแล้ว โดยอุปกรณ์ NAT จะมีกระบวนการในการตรวจ จับว่ามีการสิ้นสุด session ของคู่ไอพีแอดเดรสนั้นๆ หรือไม่ ซึ่งถ้ามีการสิ้นสุดแล้วจะลบข้อมูลการจับคู่ออกจากหน่วยความจำ

<u>สามารถเปลี่ยนแปลงข้อมูลของ ICMP payload ได้ (ICMP error message payload translation)</u>

การทำงานในเครือข่าย TCP/IP นั้น เมื่อมีการทำงานที่ผิดพลาดเกิดขึ้น จะมีการส่งรายละเอียดต่างๆ ไปกับแพ็กเก็ต ICMP ซึ่งในกรณีที่มีการใช้งาน NAT และเกิดการทำงานที่ผิดพลาดหรือผิดปกติเกิดขึ้นในเครือข่าย ตัวอุปกรณ์ NAT ต้องสามารถเปลี่ยนแปลงข้อมูลในแพ็ก

เก็ต ICMP ให้ถูกต้องด้วย เช่น Destination Unreachable , Source-Quench , Time-Exceed และ Parameter-Problem แต่ NAT ไม่ควรเปลี่ยนแปลงค่าข้อมูลใน Redirect Message การเปลี่ยนแปลงค่าในแพ็กเก็ต ICMP นั้นจะหมายรวมถึงค่าของหมายเลขไอพีแอดเดรสต้นทางใน ICMP payload ด้วย ซึ่งก็หมายความว่าต้องเปลี่ยนค่า checksum ทั้งใน ICMP header และ IP header ด้วยเช่นกัน

4.2.4 รูปแบบของการทำ NAT

เนื่องจากการทำงานของระบบและการใช้งานเครือข่ายมีหลากหลายรูปแบบ การทำ NAT จึงมีวิธีการหลายรูปแบบเพื่อให้เหมาะสมกับการทำงานแบบต่างๆ โดยการทำ NAT แบบต่างๆ มีดังนี้คือ

1. Traditional NAT (outbound NAT)

เป็นการทำ NAT แบบหนึ่งที่ออกแบบให้มีการเชื่อมต่อจากเครือข่ายภายใน ออกสู่เครือข่ายภายนอกเท่านั้น โดย outbound NAT แบ่งออกเป็น 2 แบบคือ Basic NAT และ Network Address Port Translation (NAPT)

1.1 basic NAT

เป็นการทำ NAT โดยเปลี่ยนแปลงข้อมูลของเครือข่ายภายในซึ่งเป็นเครือข่ายที่เริ่มการเชื่อมต่อ ให้กลายเป็นข้อมูลที่เหมาะสมในการเชื่อมต่อ โดยอุปกรณ์ NAT จะเปลี่ยนแปลงข้อมูลหมายเลขไอพีต้นทาง และข้อมูลที่เกี่ยวข้องอื่นๆ เช่น TCP , UDP , ICMP header checksum เป็นต้น ซึ่งหลังจากที่มีการเชื่อมต่อกันเรียบร้อยแล้ว ข้อมูลที่ตอบกลับมาจากเครือข่ายภายนอกก็จะถูกเปลี่ยนแปลงให้เหมาะสมในการเชื่อมต่อกับเครือข่ายภายในเช่นกัน

1.2 Network Address Port Translation (NAPT)

Network Address Port Translation (NAPT) คือกระบวนการที่คล้ายกับการทำ NAT แต่จะมีการเปลี่ยนแปลงข้อมูลในชั้น transport ด้วยเช่น TCP port , UDP port และ ICMP query identification เป็นต้น ซึ่งกระบวนการดังกล่าวจะช่วยให้สามารถทำ NAT โดยใช้หมายเลขไอพีแอดเดรสที่จดทะเบียนเพียงหมายเลขเดียวได้

2. Bi-Directional NAT (Two-way NAT)

เป็นการทำ NAT ที่สามารถเชื่อมต่อจากเครือข่ายภายนอกเข้ามายังเครือข่ายภายในได้ เช่นเดียวกับการเชื่อมต่อจากเครือข่ายภายในออกไปยังเครือข่ายภายนอก ในการจับคู่หมายเลขไอพีแอดเดรสสามารถทำได้ทั้งแบบ static และ dynamic สำหรับการเชื่อมต่อจากต้นทางไปยังปลายทางนั้นจำเป็นต้องใช้ DNS ในการบอกหมายเลขไอพีในการเชื่อมต่อด้วยโดยเฉพาะในการทำงานแบบ Dynamic

3. Twice-NAT

การทำงานของ Traditional NAT และ Bi-Directional NAT นั้นมีการเปลี่ยนแปลงเฉพาะค่าของหมายเลขไอพีแอดเดรสต้นทางหรือหมายเลขไอพีแอดเดรสปลายทาง อย่างใดอย่างหนึ่งเท่านั้น ซึ่งในการทำงานบางอย่างจำเป็นต้องมีการทำงานมากกว่านี้ เช่นในกรณีที่หมายเลขไอพีแอดเดรสภายในซ้ำกับหมายเลขไอพีแอดเดรสภายนอก (กรณีที่มีการเปลี่ยน ISP แต่ไม่ต้องการให้เปลี่ยนแปลง configuration ในองค์กร) ซึ่งปัญหาที่เกิดขึ้นก็คือไม่สามาร ถเชื่อมต่อไปยังเครือข่ายภายนอกได้ เพราะถือว่าเป็นการทำงานใน local เท่านั้น สำหรับปัญหานี้จำเป็นต้องมีการทำ NAT ที่มีการเปลี่ยนแปลงทั้งหมายเลขไอพีแอดเดรสต้นทางและปลายทางพร้อมๆ กัน ซึ่งต้องใช้การทำงานของ DNS มาช่วยในการเชื่อมต่อด้วย

ยกตัวอย่างเช่นในกรณีที่เ ครือข่ายภายในเป็นเครือข่าย 200.200.200.0/24 ต้องการเชื่อมต่อไปยังเครื่องในเครือข่ายภายนอกหมายเลขไอพีแอดเดรสคือ 200.200.200.100 จะมีการทำงานคือ

1. ต้องทำให้เครื่องต้นทางส่งข้อมูลไปยังหมายเลขไอพีแอดเดรสในเครือข่ายภายนอกให้ได้เพื่อให้แพ็กเก็ตผ่านอุปกรณ์ NAT ซึ่งต้องให้เป็นภาระการทำงานของ DNS และ
2. ต้องให้อุปกรณ์ NAT เปลี่ยนแปลงหมายเลขไอพีแอดเดรสปลายทางไปยังปลายทางที่แท้จริง และเปลี่ยนแปลงหมายเลขไอพีต้นทางเป็นหมายเลยไอพีแอดเดรสที่จดทะเบียนอย่างถูกต้อง

ในการทำงานข้อแรก เครื่องคอมพิวเตอร์ในเครือข่ายภายในร้องขอไปยัง DNS เมื่อ DNS รับการร้องขอแล้ว DNS จะส่งหมายเลขไอพีแอดเดรสปลอมซึ่งเป็นหมายเลขไอพีแอดเดรสของเครือข่ายภายนอกไปให้เครื่องคอมพิวเตอร์ที่ร้องขอ พร้อมกับส่งข้อมูลการร้องขอและหมายเลขไอพีแอดเดรสปลายทางที่แท้จริงไปให้อุปกรณ์ NAT

หลังจากนั้นเครื่องต้นทางจะส่งข้อมูลเพื่อร้องขอไปยังหมายเลขไอพีที่ได้ เมื่อแพ็กเก็ตส่งไปยังอุปกรณ์ NAT ได้แล้ว หลังจากนั้นอุปกรณ์ NAT จะเปลี่ยนหมายเลขไอพีแอดเดรส

ปลายทางให้กลายเป็นหมายเลขปลายทางที่แท้จริง และเปลี่ยนแปลงหมายเลขไอพีต้นทางให้เป็นหมายเลขไอพีที่จดทะเบียนอย่างถูกต้อง ซึ่งถ้าอุปกรณ์ NAT ได้รับข้อมูลตอบกลับมาแล้ว จะเปลี่ยนแปลงข้อมูลกลับไปเป็นแบบเดิมอีกครั้งหนึ่ง ซึ่งจะทำให้การเชื่อมต่อเป็นไปได้อย่างถูกต้องทั้งทางฝั่งผู้รับและผู้ส่ง

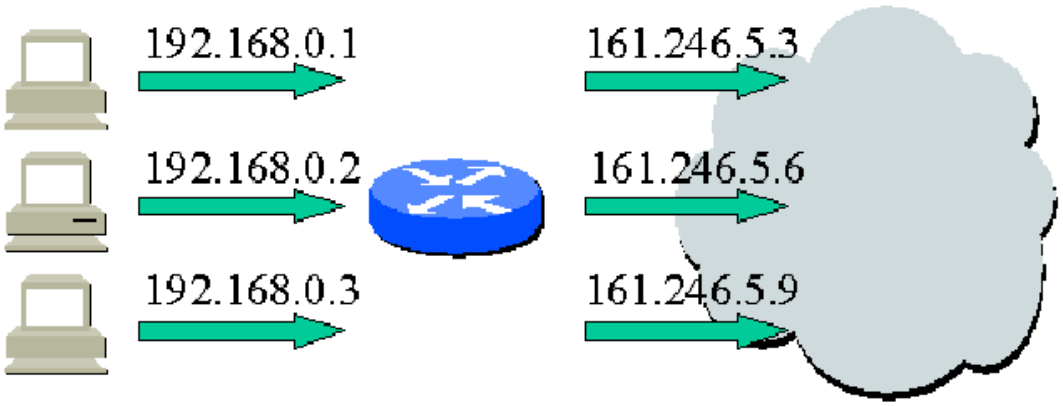
4. Multihomed NAT

จากการออกแบบเครือข่ายที่ทำให้ NAT เป็นเสมือนกับช่องทางเชื่อมต่อไปยังเครือข่ายภายนอกเพียงช่องทางเดียวซึ่งทำให้เป็นจุดอ่อนในระบบ (single point of failure) วิธีการแก้ปัญหานี้ก็สามารถทำได้โดยการออกแบบให้มีอุปกรณ์ NAT มากกว่าหนึ่งชิ้นในเครือข่าย ซึ่งอุปกรณ์ทั้งหมดต้องสามารถส่งข้อมูลสถานะการทำงานเช่นข้อมูลการจับคู่หมายเลขไอพีแอดเดรส และต้องมีความสามารถในการสวิทช์การทำงานไปยังอุปกรณ์ตัวอื่นๆ ได้ในกรณีที่มีอุปกรณ์หลักมีปัญหาได้

ลักษณะการทำงานของ NAT

1. Static NAT (static assignment and basic NAT)

เป็นการทำ NAT ที่ช่วยให้เครื่องคอมพิวเตอร์ที่มีหมายเลขไอพีแอดเดรสอยู่ในช่วง private IP หรือหมายเลขไอพีแอดเดรสที่ไม่ได้จดทะเบียนอย่างถูกต้อง สามารถติดต่อกับเครือข่ายอื่นๆ ได้ โดยการทำงานของ Static NAT นั้นจะจับคู่ระหว่างหมายเลขไอพีแอดเดรสภายในเครือข่าย กับหมายเลขไอพีแอดเดรสที่ได้รับการจดทะเบียนแบบหนึ่งต่อหนึ่ง ในการทำงานลักษณะนี้มีประโยชน์เพื่อความสะดวกในการจัดการหมายเลขไอพีแอดเดรสในเครือข่ายที่มักจะมีการปรับเปลี่ยนบ่อยๆ และทำให้เครื่องคอมพิวเตอร์ภายนอกเครือข่ายสามารถติดต่อเข้ามาในเครือข่ายได้ด้วย

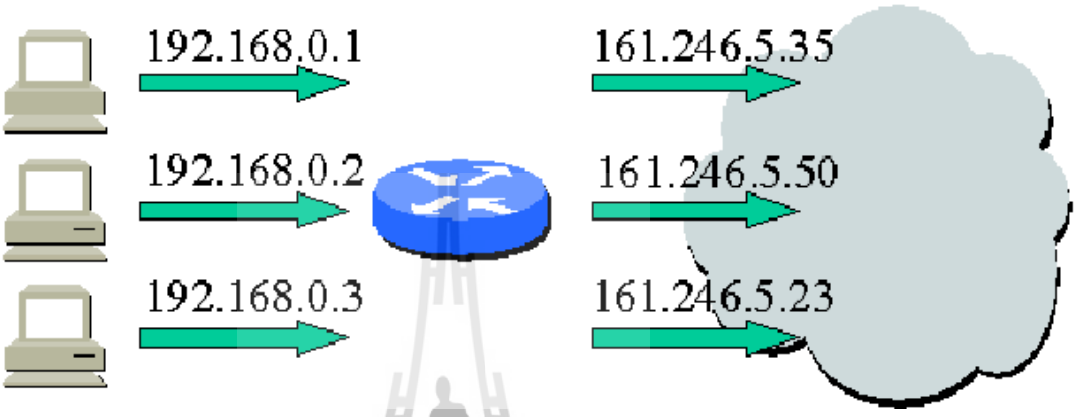


รูปภาพที่ 4.2 Static NAT

2. Dynamic NAT (dynamic assignment and basic NAT)

เป็นการทำ NAT ที่ใช้วิธีการเปลี่ยนแปลงหมายเลขไอพีแอดเดรสที่ใช้ในเครือข่าย ให้กลายเป็นหมายเลขไอพีแอดเดสรที่จดทะเบียนแล้ว โดยการสุ่มเลือกหมายเลขไอดีแอดเดรสซึ่งการทำงานลักษณะนี้จะช่วยให้เครือข่ายที่มีหมายเลขไอพีแอดเดรสในช่วง private IP หรือเป็นเครือข่ายที่มีการตั้งค่าหมายเลขไอพีแอดเดรสเองโดยไม่ได้จดทะเบียน สามารถติดต่อไปยังเครือข่ายอื่นๆ ได้ แต่การทำ Dynamic NAT นี้เครื่องคอมพิวเตอร์จากภายนอกเครือข่ายจะไม่สามารถติดต่อเข้ามายังเครื่องคอมพิวเตอร์ภายในเครือข่ายได้ เนื่องจากเครื่องคอมพิวเตอร์ภายนอกจะไม่สามารถทราบได้เลยว่าหมายเลขไอพีแอดเดรสของเครื่องที่จะเชื่อมต่อด้วยนั้นคือหมายเลขอะไร ซึ่งการทำ Dynamic NAT ก็สามารถนำมาใช้เพื่อรักษาความปลอดภัยในเครือข่ายได้

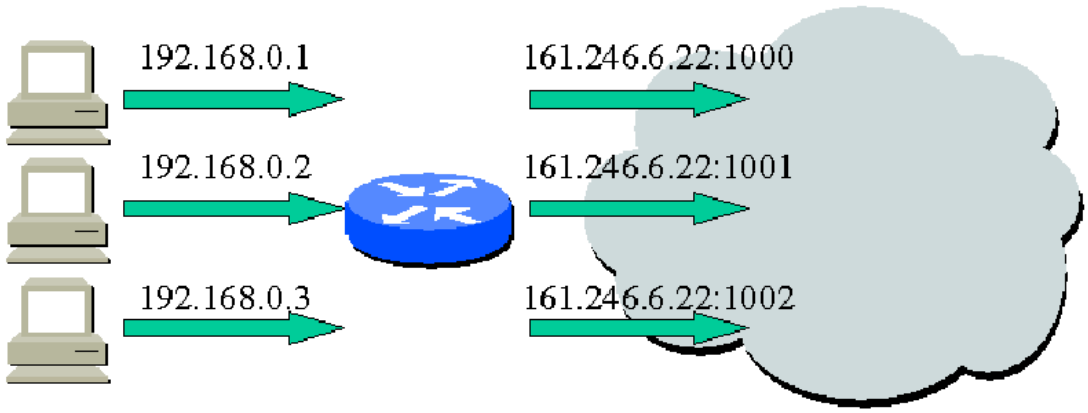
192.168.0.1	161.246.5.35
192.168.0.2	161.246.5.50
192.168.0.3	161.246.5.23



รูปภาพที่ 4.3 Dynamic NAT

3. Overloading (NAPT)

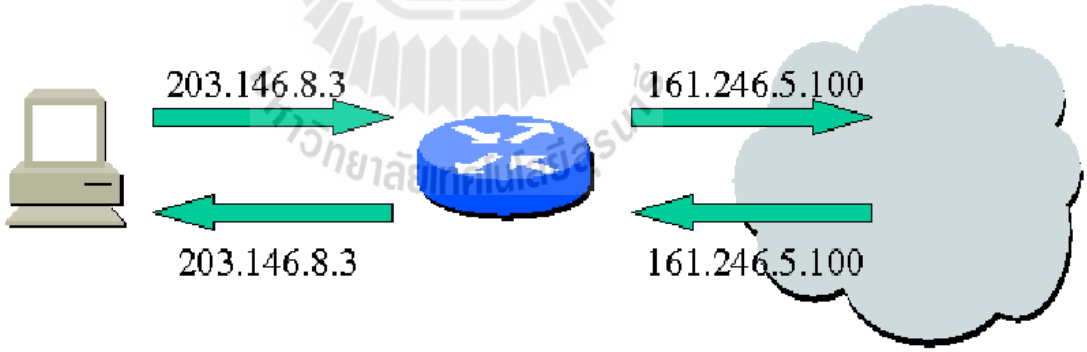
จากการทำงานของ Static NAT และ Dynamic NAT นั้นจะเห็นได้ว่าจำนวนของหมายเลขไอพีแอดเดรสที่จดทะเบียน จะต้องเท่ากับจำนวนหมายเลขไอพีแอดเดรสภายในเครือข่าย ซึ่งทำให้ยังจำเป็นต้องใช้จำนวนไอพีแอดเดรสจำนวนมากอยู่เช่นเดิม วิธีการหนึ่งที่ช่วยให้ประหยัดหมายเลขไอพีแอดเดรสคือการนำเอาวิธีการของ NAPT มาใช้ โดยเครื่องคอมพิวเตอร์ในเครือข่ายที่เป็น Private IP เมื่อติดต่อไปยังเครือข่ายอื่นๆ จะถูกเปลี่ยนเป็นหมายเลขไอพีแอดเด รสเพียงหมายเลขเดียวแต่มีการเปลี่ยนแปลงหมายเลขพอร์ตต้นทางในการเชื่อมต่อแทน เมื่อมีการตอบกลับจากเครื่องภายนอกเครือข่ายแล้ว ที่อุปกรณ์ NAT จะดูหมายเลขพอร์ตปลายทางในส่วนหัวของข้อมูลว่าเป็นหมายเลขอะไร แล้วจึงเปลี่ยนข้อมูลส่วนหัวให้ตรงกับเครื่องคอมพิวเตอร์ที่ทำการร้องขออีกครั้ง



รูปภาพที่ 4.4 Overloading

4. Overlapping (Twice-NAT)

ในกรณีที่หมายเลขไอพีแอดเดรสในเครือข่าย เป็นหมายเลขไอพีแอดเดรสซึ่งใช้งานอยู่ในเครือข่ายอื่นๆ หรือเป็นหมายเลขไอพีแอดเดรสที่เรานำมาใช้งานกันเอง โดยไม่ได้จดทะเบียนขอใช้งาน เมื่อมีการเชื่อมต่อกับเครือข่ายอื่นๆ จะทำให้เกิดปัญหาขึ้นในระบบเครือข่าย แต่การใช้งานในลักษณะนี้ก็ยังสามารถใช้งานได้ แต่ต้องทำ NAT ให้กลายเป็นหมายเลขไอพีที่จดทะเบียนถูกต้องเสียก่อน



รูปภาพที่ 4.5 Overlapping

ข้อจำกัดของการทำ NAT

1. โปรแกรมที่มีข้อมูลของหมายเลขไอพีแอดเดรสอยู่ในชั้น Application Layer

เนื่องจากการทำ NAT มีการเปลี่ยนแปลงข้อมูลของเฮดเดอร์ของแพ็กเก็ต จึงทำให้การทำงานของโปรแกรมบางโปรแกรมที่มีข้อมูลของหมายเลขไอพีแอดเดรสที่จะต้องติดต่อด้วยอยู่ในส่วนของข้อมูลทำงานไม่ได้เนื่องจากไม่สามารถทำงานตามการทำงานโดย

ปกติได้ เนื่องจากข้อมูลหมายเลขไอพีในเนื้อข้อมูลเป็นหมายเลขไอพีแอดเดรสของเครื่องในเครือข่ายภายใน ซึ่งเป็นหมายเลข private IP ถ้าการทำงานต้องมีการเชื่อมต่อไปยังหมายเลขไอพีแอดเดรสดังกล่าว โปรแกรมนั้นจะไม่สามารถทำงานได้เลย ยกตัวอย่างเช่น SNMP เป็นต้น

นอกจากนี้ในกรณีที่โปรแกรมต้องมีการแลกเปลี่ยนหมายเลขพอร์ตกันโดยใช้การทำงานในชั้น Application Layer หรือมีการรับส่งข้อมูลหมายเลขพอร์ตกันในเนื้อข้อมูลชั้นแอปพลิเคชั่น ก็จะทำให้การทำงานของโปรแกรมมีปัญหาได้ในกรณีที่มีการทำ NAPT เพราะจะได้หมายเลขพอร์ตที่ผิดไปได้

2. โปรแกรมที่มีความสัมพันธ์ระหว่าง control session กับ data session

การทำงานของอุปกรณ์ NAT นั้นอยู่บนสมมุติฐานที่ว่าแต่ละ session นั้นมีการทำงานแยกจากกันโดยอิสระ ไม่มีความสัมพันธ์กันระหว่าง session ใดๆ หมายถึง เมื่อมีการสร้าง session ใดๆ จะมีหมายเลขไอพีแอดเดรสต้นทาง , หมายเลขไอพีแอดเดรสปลายทาง, โพรโตคอล , หมายเลขพอร์ตต้นทาง และหมายเลขพอร์ตปลายทาง เป็นตัวเลขที่ไม่ขึ้นกับหมายเลข หรือข้อมูลใน session อื่นๆ ถ้ามีโปรแกรมที่มีการสร้าง session ใหม่โดยขึ้นอยู่กับการควบคุมของ session อื่นๆ จะทำงานไม่ได้ถ้ามีการทำ NAT เนื่องจากภายหลังจากการทำ NAT ข้อมูลของ session จะถูกเปลี่ยนไปทั้งหมดนั่นเอง

ตัวอย่างของโปรแกรมที่มีการทำงานที่มีความสัมพันธ์ระหว่าง control session และ data session เช่นโปรแกรมที่ใช้ H.323 ซึ่งโปรแกรมประเภทนี้จะใช้ control session ในการกำหนดลักษณะการทำงานของ session อื่นๆ โดยใช้ข้อมูลใน control session นั้น

3. การตรวจจบหาความผิดปกติต่างๆ ในระบบเครือข่าย

เนื่องจากการทำ NAT จะมีการเปลี่ยนแปลงข้อมูลจากหมายเลขไอพี แอดเดรสภายในเครือข่ายให้เป็นหมายเลขไอพี แอดเดรสที่ มีการจดทะเบียน โดยมการใช้ งานหมายเลขไอพแอดเดรสต่างๆ แบบสุ่ม และเปลี่ยนแปลงไปตลอดเวลา ทำ ให้การตรวจจับหาผู้กระทำ ผิดเช่นการส ง SPAM Mail หรือการโจมตี ไปยังเครือข่ายอื่นๆ ทำ ได้ยาก เนื่องจากข้อมูลมีการเปลี่ยนแปลงอยตลอดเวลา

4. การประมวลผลในอปกรณ NAT

เนื่องจากการทำ งานในอุ ปกรณ์ NAT จะต้องมีการคำ นวณหาคา checksum ของข้อมูลทุกๆ แพกเก็ต จึงทำให้การทำงานในเครือข่ายที่มีการทำ NAT ช้าลงได้

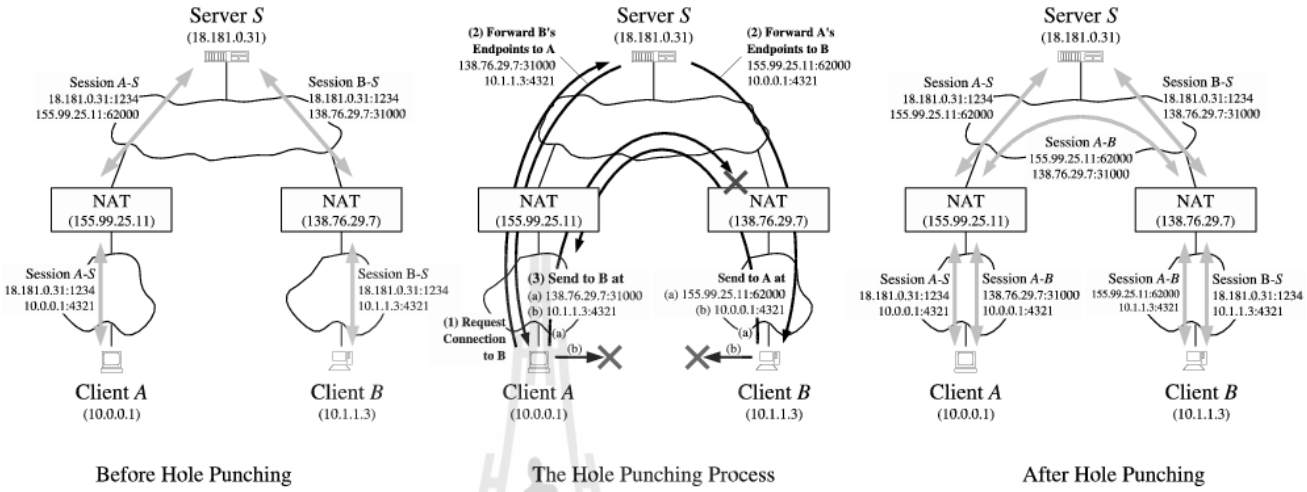
ตัวอย่างปัญหาอื่นๆ ที่เกิดขึ้นในการทำ NAT

- เป็นการปกปิดรายละเอียดของเครือข่ายภายในองค์กรแต่ก็ยังมีปัญหาในการทำงานอื่นๆ เช่นการทางานของโปรแกรมบางโปรแกรม
- มีปัญหากับการทางานของ DNS (“A” and “PTR” query) , SNMP , FTP (port command, PASV command), โปรแกรมที่มี content ที่เป็นหมายเลขไอพแอดเดรส , การทำ IPSec (ipsec tunnel ทำได้ถ้าให้ nat router เป็น tunnel end point)
- มีปัญหากับ App เช่น H.323 ที่ใช้ control หลาย session ซึ่งต้องใช้ การทำงานพิเศษเช่น payload interpretion gateway เข้าช่วย
- ไม่สนับสนุน ICMP , NetBIOS over TCP/IP , Real Audio , Video Live , IP multi cast
- มีปัญหากับ routing table update , DNS , Zone transfer , Bootp , Ntalk , talk

security consideration

- เพื่อไม่ให้ผู้บุกรุกเห็นว่ามการใช้ NAT device จึงไม่ควรมีข้อมูลของ private ip สงออกไปยังเครือข่าย internet
- ควรตรวจสอบทั้งหมายเลขไอพแอดเดรสต้นทาง , หมายเลขไอพแอดเดรสปลายทาง , พอร์ตต้นทาง และพอร์ตปลายทางที่ใช้ในการเชื่อมต่อด้วยเพื่อป้องกันการปลอมแปลงหมายเลขไอพแอดเดรสและพอร์ตได้ เพราะอาจมีผู้ไม่หวังดีทำการปลอมไอพีแอดเดรสให้เหมือนกับมาจากเครื่องคอมพิวเตอร์ที่เครือข่ายภายในแล้วเชื่อมต่อไปยังเครื่องของตน แล้วเข้ามาโจมตีเครือข่าย
- การใช้ multicast session อาจทำให้เกิดปัญหาความปลอดภัยใน basic NAT ได้เนื่องจากระบบจะไม่สามารถทราบได้เลยว่าข้อมูลทตอบกลับมานั้นเป็นข้อมูลที่ตอบกลับมาจากการ้องขอจากเครื่องคอมพิวเตอร์ภายในเครือข่ายภายในหรือจากผู้บุกรุก
- อุปกรณ์ NAT เป็นเป้าหมายในการโจมตี เช่นเดียวกับ server จึงควรมีการป้องกันในระดับเดียวกับการป้องกัน server

4.3 หลักการทำงานของ NAT ที่ใช้สร้างแบบจำลอง



รูปภาพที่ 4.6 Hole Punching

Clients A จะสามารถติดต่อกับ Clients B ได้โดย Clients A จะทำการส่ง Public IP address และ private IP address ของ Clients B ไปยัง Server แล้ว Server จะส่ง Public IP address และ private IP address ของ Clients B ไปยัง Clients A โดยผ่าน NAT A โดย NAT A จะสร้างตารางเก็บค่าไว้สามค่า คือ Public IP address, private IP address ของ A และ Server ในส่วนของ Clients B จะได้รับ Public IP address และ private IP address ของ Clients Aโดยผ่าน NAT B โดย NAT B จะสร้างตารางเก็บค่าไว้สามค่าเช่นกัน คือ Public IP address, private IP address ของ Clients B และ Server(แทนด้วย Y) จากนั้น Clients A จะทำการส่ง Source และ Destination ผ่าน NAT A ซึ่ง NAT A จะสร้างตารางใหม่เก็บค่าสามค่าคือ Public IP address , private IP address ของ Clients A และ Public IP address ของ Clients B (แทนด้วย Y) แต่ Clients A ก็ยังไม่สามารถติดต่อกับ Clients B ได้ เนื่องจาก NAT B มีการสร้างตารางเก็บค่า Y เป็น Server ค้างอยู่มันจึงทำการบล็อกไว้เพราะ Y มีค่าไม่ตรงกัน ดังนั้น Clients B จะพยายามติดต่อไปยัง Clients A (ในกรณีที่ Clients A และ Clients B พยายามติต่อกันทั้งสองฝั่ง เรียกว่า Simultaneous Open) โดยผ่าน NAT B ซึ่ง NAT B จะทำการสร้างตารางเก็บค่าใหม่สามค่าเป็น Public IP address , private IP address ของ Clients B และ Public IP address ของ Clients A และเมื่อมันผ่าน NAT B แล้ว จะได้ Source เป็น Public IP address ของ Clients B และ Destination เป็น Public IP address ของ Clients A ซึ่งในตอนนี้ตารางเก็บค่าที่

NAT A จะมีค่าของ Public IP address , private IP address ของ Clients A และ Public IP address ของ Clients B (Y) ซึ่งจะเห็นว่า Y มีค่าที่ตรงกันแล้วก็จะทำให้ Clients A และ Clients B ติดต่อกันได้

บทที่ 5

แบบจำลอง DCCP

5.1 บทนำ

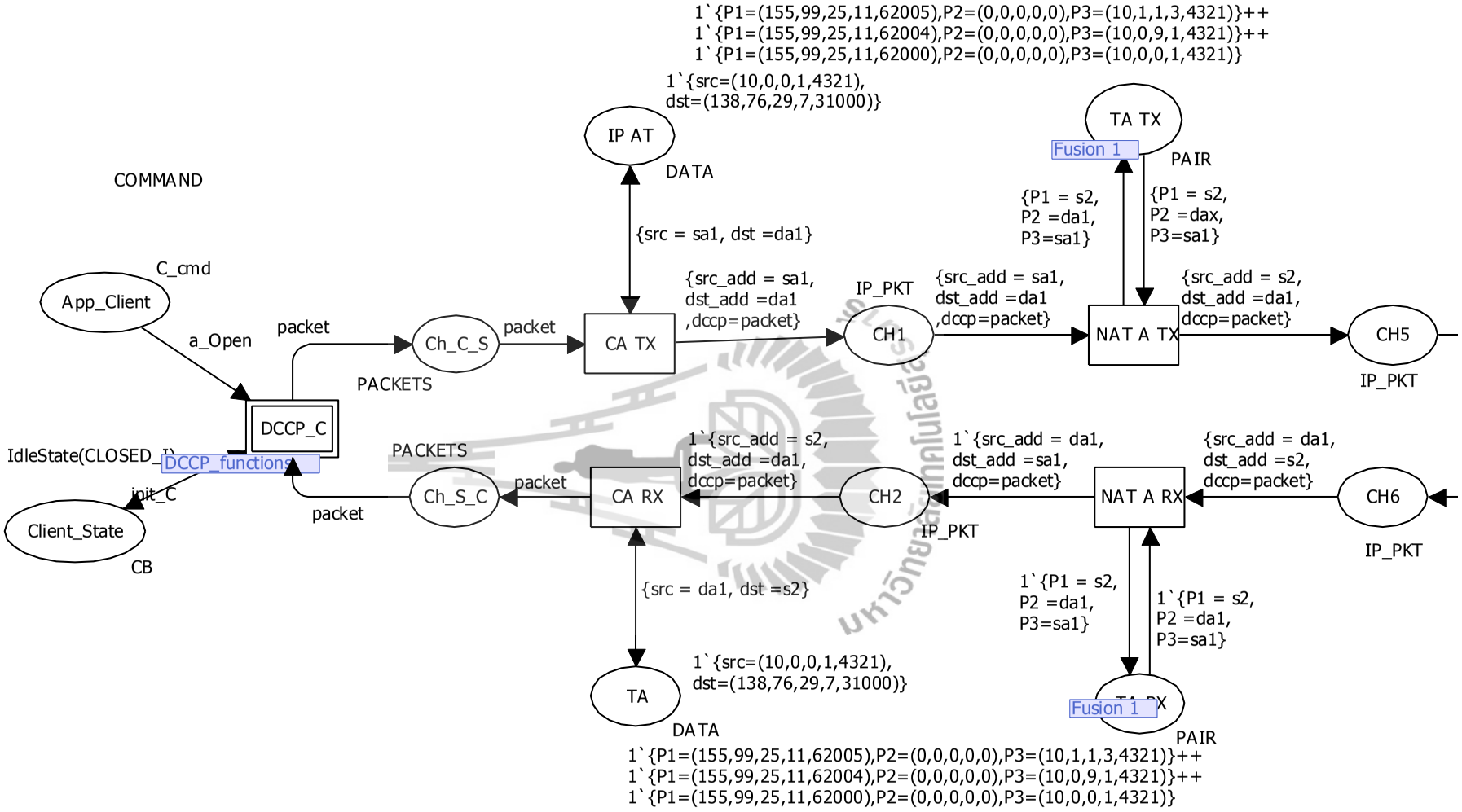
จากที่ได้ทำการศึกษาและทำความเข้าใจเกี่ยวกับทฤษฎีในบทที่ 2 และ 3 และศึกษาจากข้อกำหนด RFC 5596 แล้วทำให้สามารถสร้างแบบจำลอง DCCP และ NAT ซึ่งสามารถแสดงการทำงาน DCCP บน NAT และในบทนี้จะอธิบายรายละเอียดของแบบจำลอง DCCP ใน page ต่างๆ

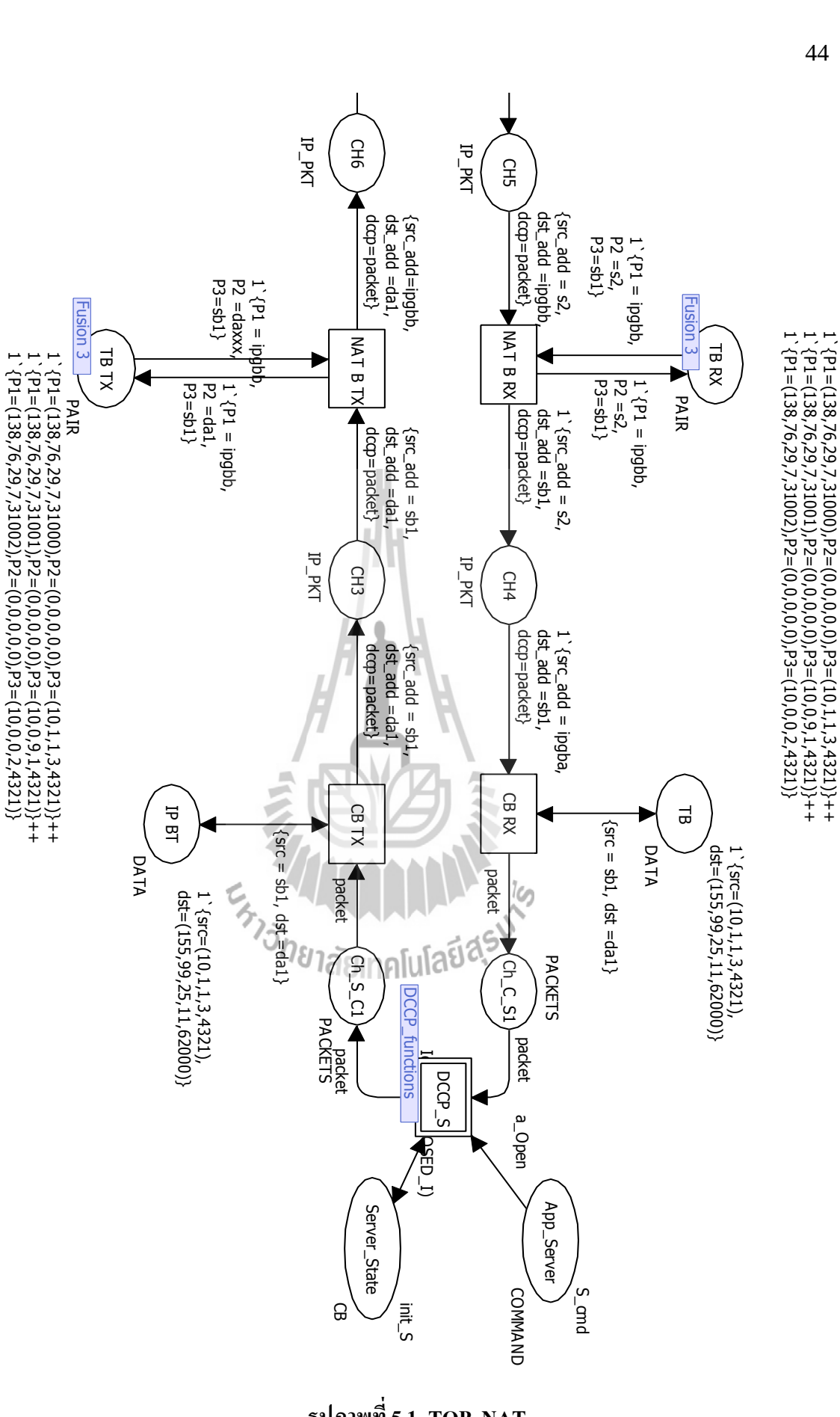
5.2 DCCP อธิบายแบบจำลองในแต่ละ page

5.2.1 TOP_NAT

ใน page นี้ประกอบด้วยทรานสิชัน DCCP_C, DCCP_S, CA TX, CA RX, NAT A TX, NAT A RX, CB TX, CB RX, NAT B TX และ NAT B RX และมี place ทั้งหมด 22 place คือ App_Client, Client_State, Ch_C_S, Ch_S_C, IP AT, TA, CH1, CH2, TA TX, TA RX, CH5, CH6, TB RX, TB TX, CH3, CH4, TB, IP BT,Ch_C_S1,Ch_S_C1,App_Server และ Server_State

ภายใน DCCP_C จะมีทรานสิชันย่อย ซึ่งเป็นส่วนที่เกี่ยวข้องกับ DCCP


1`{P1=(155,99,25,11,62005),P2=(0,0,0,0,0),P3=(10,1,1,3,4321)}++
1`{P1=(155,99,25,11,62004),P2=(0,0,0,0,0),P3=(10,0,9,1,4321)}++
1`{P1=(155,99,25,11,62000),P2=(0,0,0,0,0),P3=(10,0,0,1,4321)}
1`{src=(10,0,0,1,4321),
dst=(138,76,29,7,31000)}
IP AT
DATA
{src = sa1, dst =da1}
TA TX
Fusion 1
PAIR
{P1 = s2,
P2 =da1,
P3=sa1}
{P1 = s2,
P2 =dax,
P3=sa1}
COMMAND
C_cmd
App_Client
a_Open
packet
Ch_C_S
packet
CA TX
{src_add = sa1,
dst_add =da1
,dccp=packet}
IP_PKT
CH1
{src_add = sa1,
dst_add =da1
,dccp=packet}
NAT A TX
{src_add = s2,
dst_add =da1,
dccp=packet}
CH5
IP_PKT
PACKETS
DCCP_C
IdleState(CLOSED_I)
DCCP_functions
init_C
Client_State
CB
packet
Ch_S_C
PACKETS
packet
CA RX
1`{src_add = s2,
dst_add =da1,
dccp=packet}
CH2
IP_PKT
1`{src_add = da1,
dst_add =sa1,
dccp=packet}
NAT A RX
{src_add = da1,
dst_add =s2,
dccp=packet}
CH6
IP_PKT
{src = da1, dst =s2}
1`{P1 = s2,
P2 =da1,
P3=sa1}
1`{P1 = s2,
P2 =da1,
P3=sa1}
TA
1`{src=(10,0,0,1,4321),
dst=(138,76,29,7,31000)}
DATA
Fusion 1
PAIR
1`{P1=(155,99,25,11,62005),P2=(0,0,0,0,0),P3=(10,1,1,3,4321)}++
1`{P1=(155,99,25,11,62004),P2=(0,0,0,0,0),P3=(10,0,9,1,4321)}++
1`{P1=(155,99,25,11,62000),P2=(0,0,0,0,0),P3=(10,0,0,1,4321)}

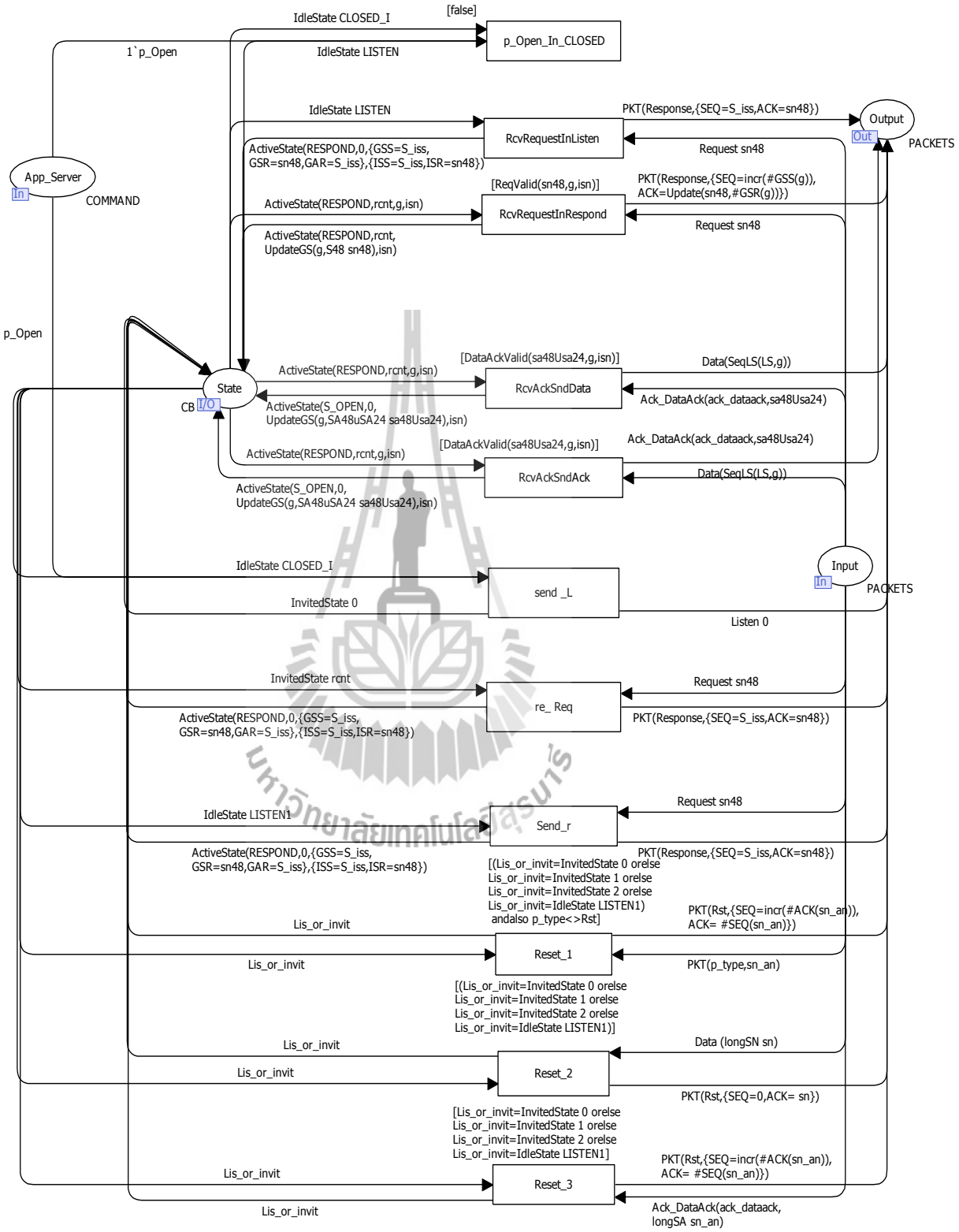


รูปภาพที่ 5.1 TOP_NAT

5.2.2 page Server

ใน page นี้จะประกอบไปด้วย 11 ทรานสิชันคือ p_Open_In_CLOSED, RcvRequestInListen, RcvRequestInRespond, RcvAckSndData, RcvAckSndAck, send_Lre_Req, Send_r, Reset_1, Reset_2 และ Reset_3 ซึ่งมีทั้งหมด place 4 place คือ App_Server, State, Output และ Input

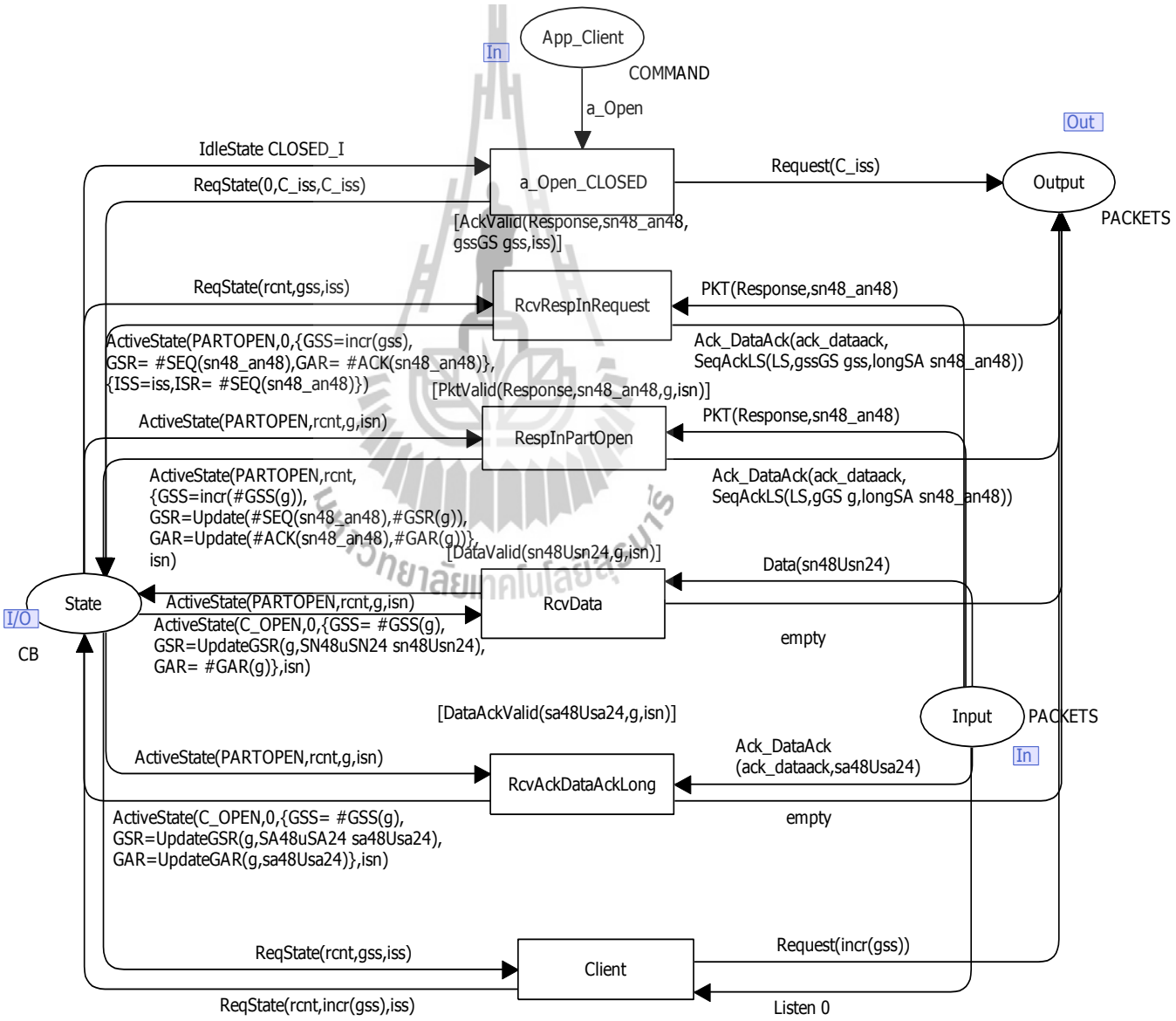
โดย ทรานสิชัน Send_r จะรับ Request sn48 จาก place Input และ ส่ง PKT(Response,{SEQ=S_iss,ACK=snco}) ไปยัง place Output นอกจากนี้ทรานสิชัน Send_r ยังรับ IdleState LISTEN1 มาจาก place State และส่งต่อ ActiveState(RESPOND,0,{GSS=S_iss, GSR=sn48,GAR=S_iss},{ISS=S_iss,ISR=sn48}) ไปยัง place State



รูปภาพที่ 5.2 Sever

5.2.3 page Client

ใน page นี้จะประกอบไปด้วย 6 ทรานสิชันคือ a_Open_CLOSED, RcvRespInRequest, RespInPartOpen, RcvData, RcvAckDataAckLong และ Client ซึ่งมีทั้งหมด place 4 place คือ App_Client, State, Input และ Output โดยทรานสิชัน a_Open_CLOSED จะรับ A_Open จาก place App_Client และรับ IdleState CLOSED_I จาก place State แล้วจะส่ง Request(C_iss) ไปยัง Output และส่ง ReqState(0,C_iss,C_iss) ไปยัง place State



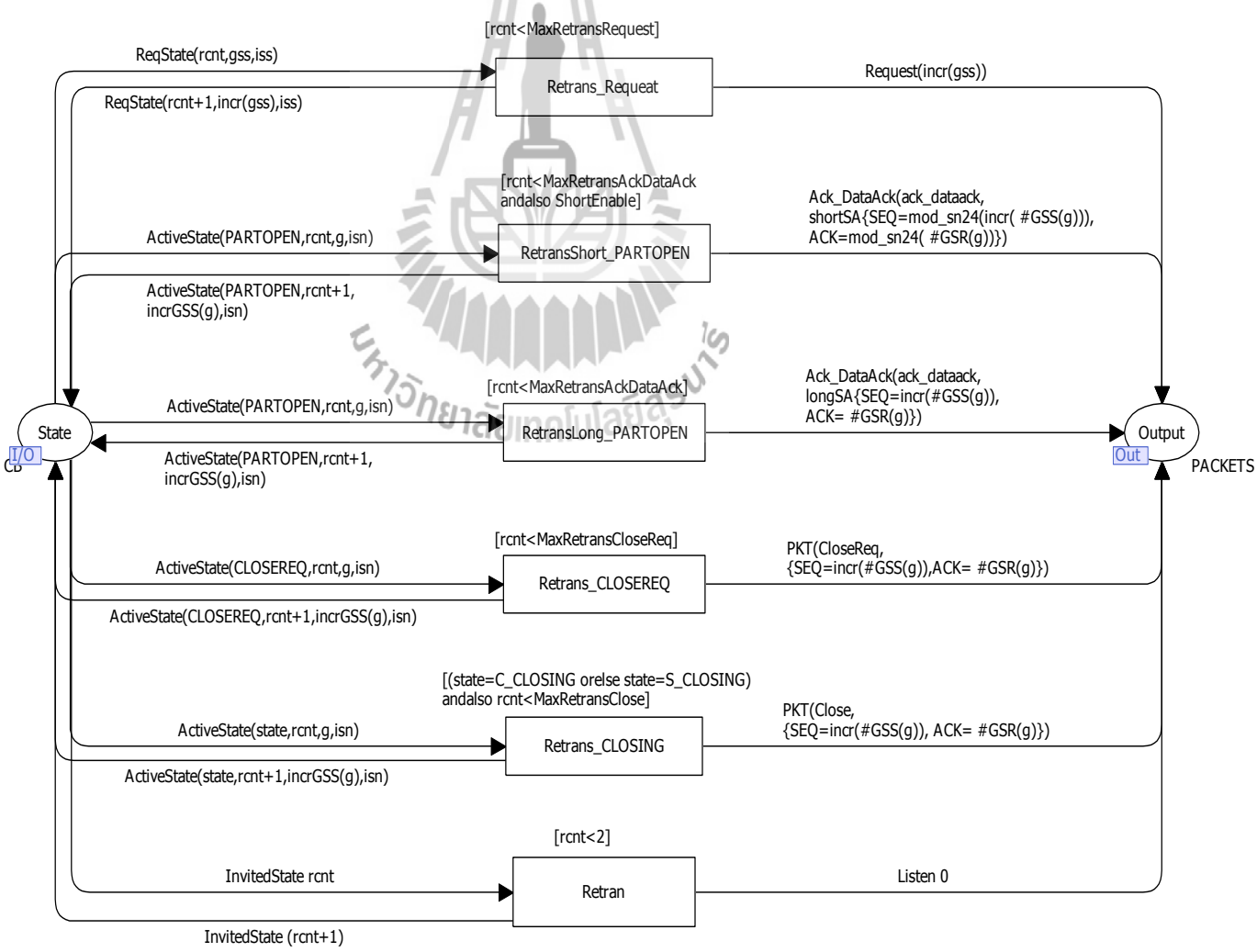
รูปภาพที่ 5.3 Client

5.2.4 page Retransmission

ใน page นี้จะประกอบไปด้วย 6 ทรานสิชันคือ Retrans_Requeat, RetransShort_PARTOPEN, RetransLong_PARTOPEN, Retrans_CLOSEREQ, Retrans_CLOSING, Retran ซึ่งมีทั้งหมด place 2 place คือ State และ Output

โดยที่ทรานสิชัน Retrans_Requeat จะรับ ReqState(rcnt,gss,iss) มาจาก place State และจะส่ง ReqState(rcnt+1,incr(gss),iss) กลับไปยัง place State พร้อมกับส่ง Request(incr(gss)) ไปยัง Output ด้วย โดยทรานสิชันนี้จะทำงานจนกว่าค่าของ rcnt<MaxRetransRequest มันจึงจะหยุดทำงาน

ทรานสิชัน Retranจะรับ InvitedState rcnt จาก place State แล้วส่งค่า InvitedState (rcnt+1) ไปยังplace State และส่ง Listen 0 ออกไป Output โดยที่ทรานสิชันนี้จะมีเงื่อนไขในการทำงานคือ rcnt<2

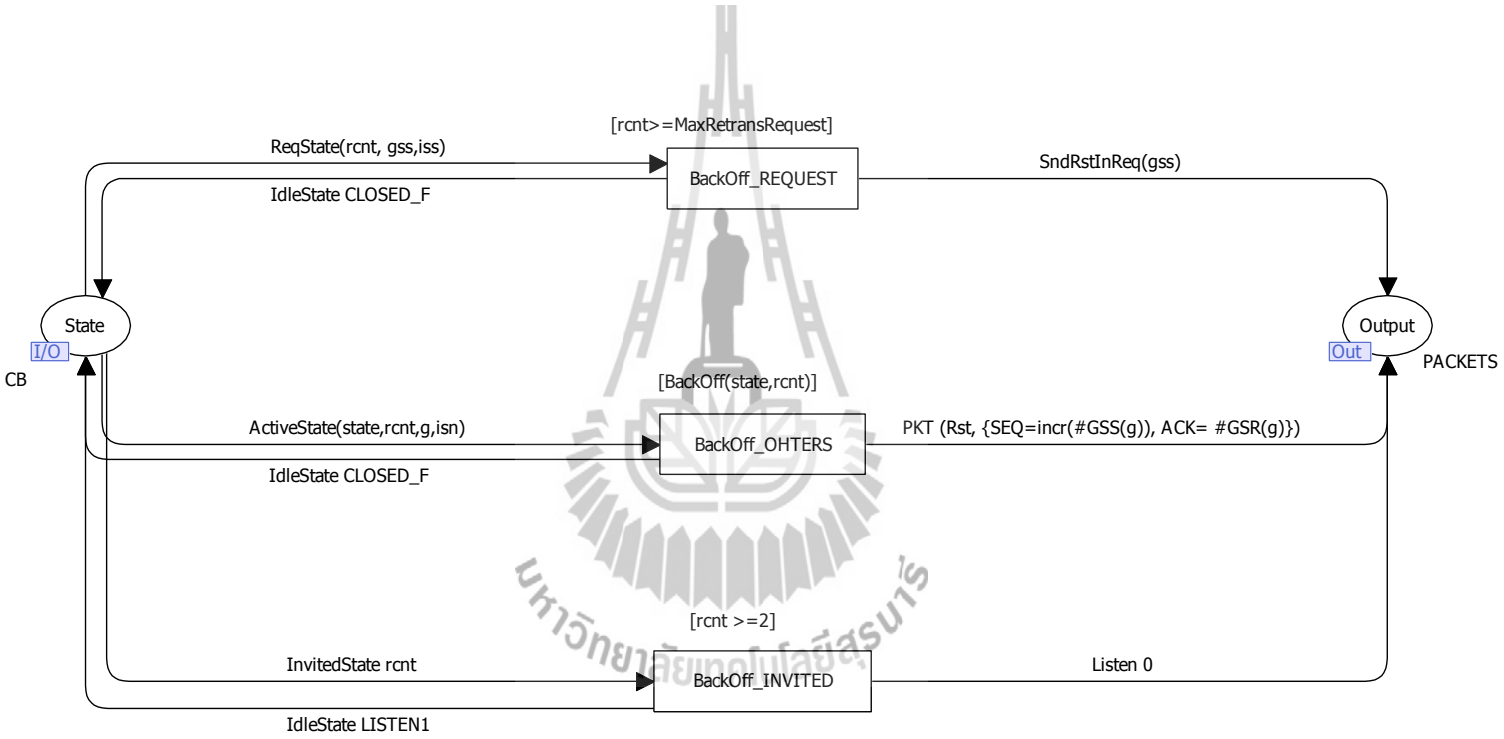


รูปภาพที่ 5.4 Retransmission

5.2.5 page BackOffFails

ใน page นี้จะประกอบไปด้วย 3 ทรานสิชันคือ BackOff_REQUEST, BackOff_OHTERS, BackOff_INVITED ซึ่งมีทั้งหมด place 2 place คือ State และ Output

การทำงานของ page นี้ ทรานสิชัน BackOff_REQUEST จะรับ ReqState(rcnt, gss,iss) มาจาก place State แล้วส่ง IdleState CLOSED_F กลับไปยัง place State และส่ง SndRstInReq(gss) ออกไปยัง Output โดยที่ทรานสิชันนี้จะหยุดทำงานก็ต่อเมื่อ rcnt>=MaxRetransRequest



รูปภาพที่ 5.5 BackOffFails

บทที่ 6

การจำลอง DCCP

6.1 บทนำ

ในบทนี้จะทำการวิเคราะห์แบบจำลอง DCCP (Datagram Congestion Control Protocol)ซึ่งผลที่ได้จะแสดงให้เห็นถึงหลักการทำงานของ NAT(Network Address Translation) โดยใช้โปรโตคอล DCCP โดยจะแสดงรายละเอียดดังต่อไปนี้

6.2 การจำลองการทำงานของ NAT เมื่อใช้งานกับโปรโตคอล DCCP

เราจำลองการทำงานของ NAT เมื่อใช้งานกับโปรโตคอล DCCP โดยใช้โปรแกรม CPN Tools รุ่น 2.2.0 ใช้คอมพิวเตอร์ Acer Aapire 4730z Ram 1GB สร้างแบบจำลอง DCCP-CPN

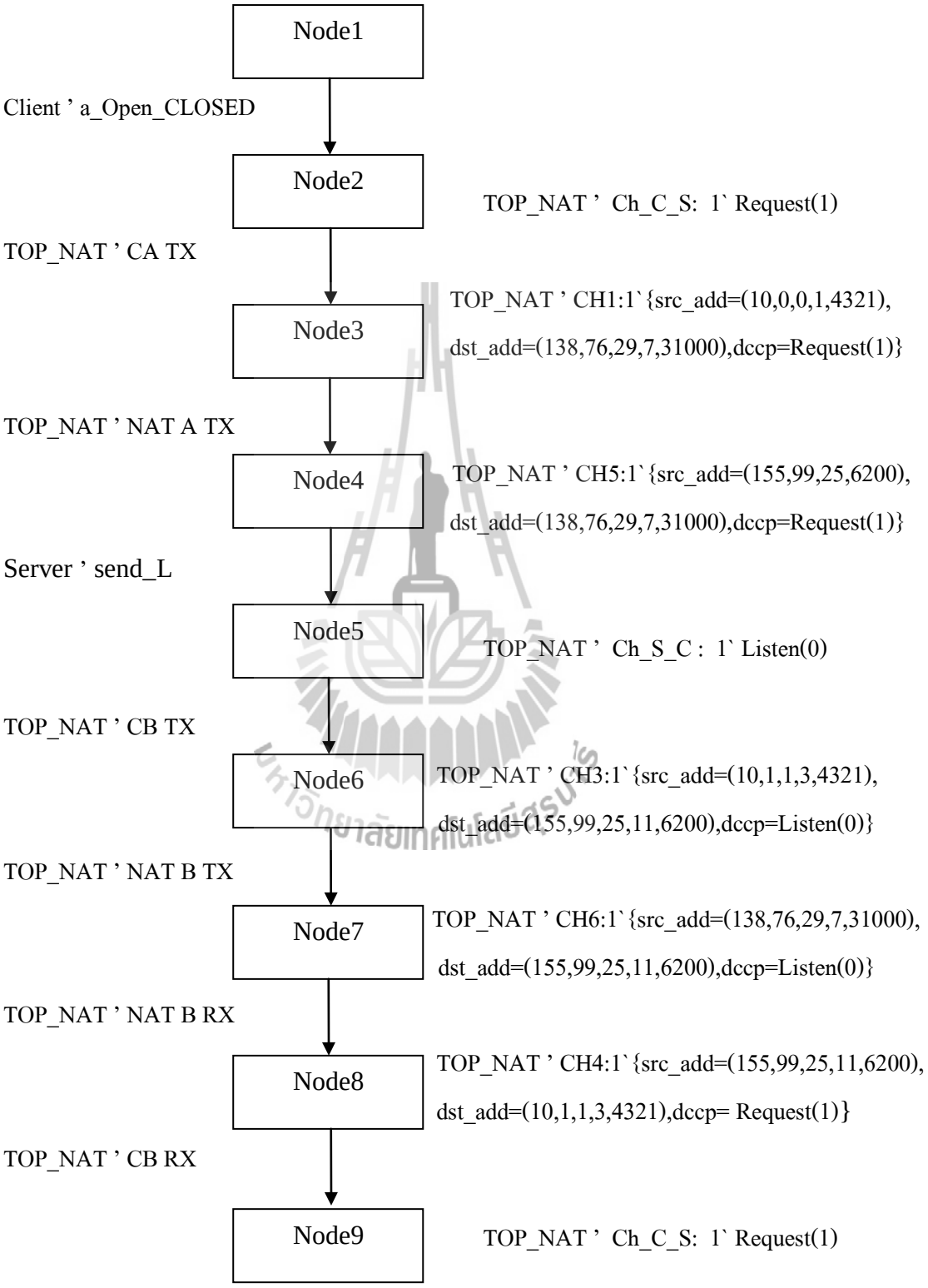
เราสามารถจำลองการทำงานได้ดังนี้

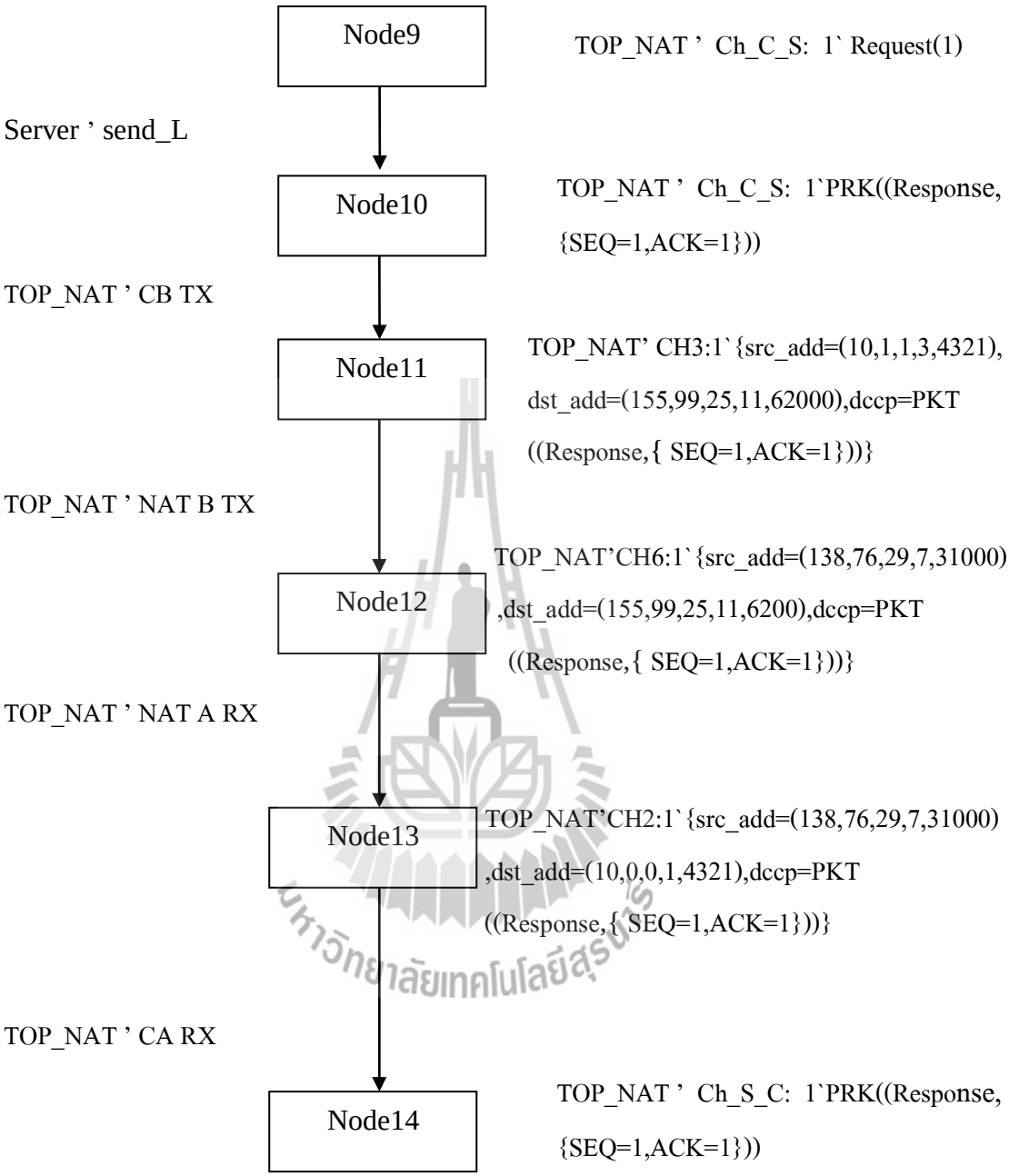
การทำงานของโปรแกรมจะประกอบด้วย 2 ส่วนด้วยกันคือ

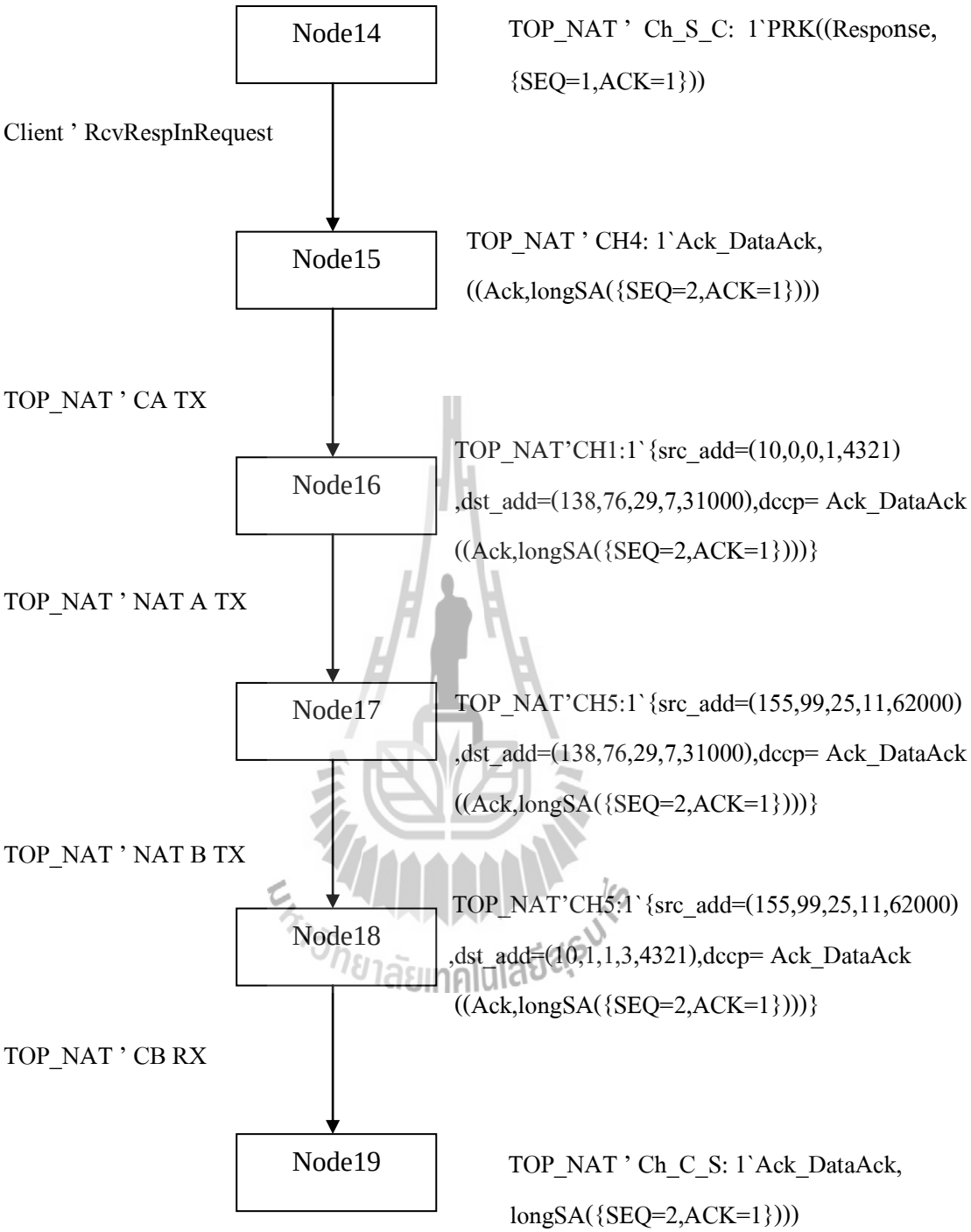
(1) ในส่วนการทำงานของ NAT ซึ่งเป็นการทำงานโดยใช้ IP

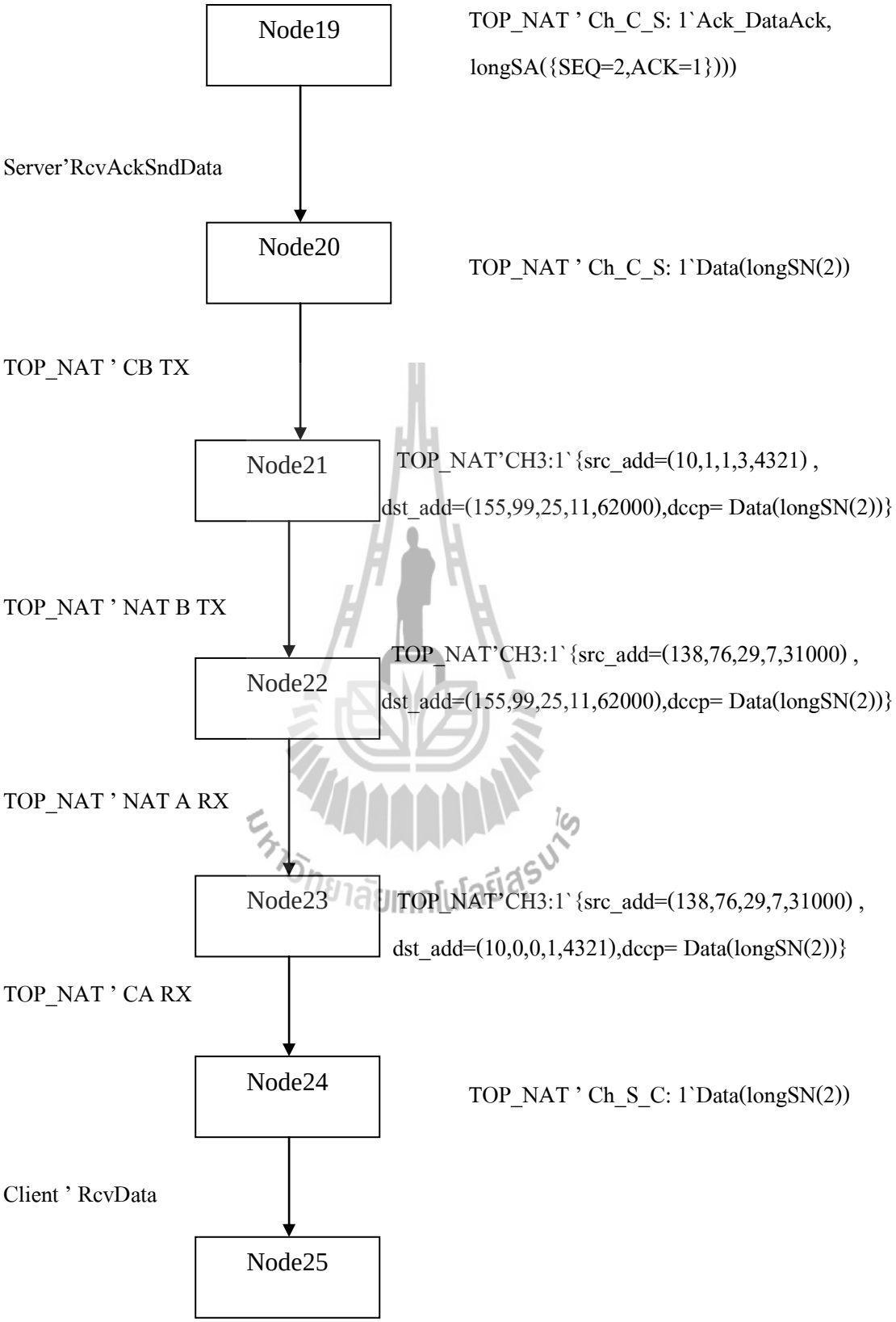
(2) การทำงานของโปรโตรคอล DCCP ซึ่งโปรโตคอล DCCPนี้จะอยู่ในชั้น Transport Layer

6.3 ผลการทดสอบ




Node9
TOP_NAT ’ Ch_C_S: 1` Request(1)
Server ’ send_L
Node10
TOP_NAT ’ Ch_C_S: 1`PRK((Response,
{SEQ=1,ACK=1}))
TOP_NAT ’ CB TX
Node11
TOP_NAT’ CH3:1`{src_add=(10,1,1,3,4321),
dst_add=(155,99,25,11,62000),dccp=PKT
((Response,{ SEQ=1,ACK=1}))}
TOP_NAT ’ NAT B TX
Node12
TOP_NAT’CH6:1`{src_add=(138,76,29,7,31000)
,dst_add=(155,99,25,11,6200),dccp=PKT
((Response,{ SEQ=1,ACK=1}))}
TOP_NAT ’ NAT A RX
Node13
TOP_NAT’CH2:1`{src_add=(138,76,29,7,31000)
,dst_add=(10,0,0,1,4321),dccp=PKT
((Response,{ SEQ=1,ACK=1}))}
TOP_NAT ’ CA RX
Node14
TOP_NAT ’ Ch_S_C: 1`PRK((Response,
{SEQ=1,ACK=1}))


Node14
TOP_NAT ’ Ch_S_C: 1`PRK((Response, {SEQ=1,ACK=1}))
Client ’ RcvRespInRequest
Node15
TOP_NAT ’ CH4: 1`Ack_DataAck, ((Ack,longSA({SEQ=2,ACK=1})))
TOP_NAT ’ CA TX
Node16
TOP_NAT’CH1:1`{src_add=(10,0,0,1,4321) ,dst_add=(138,76,29,7,31000),dccp= Ack_DataAck ((Ack,longSA({SEQ=2,ACK=1})))}
TOP_NAT ’ NAT A TX
Node17
TOP_NAT’CH5:1`{src_add=(155,99,25,11,62000) ,dst_add=(138,76,29,7,31000),dccp= Ack_DataAck ((Ack,longSA({SEQ=2,ACK=1})))}
TOP_NAT ’ NAT B TX
Node18
TOP_NAT’CH5:1`{src_add=(155,99,25,11,62000) ,dst_add=(10,1,1,3,4321),dccp= Ack_DataAck ((Ack,longSA({SEQ=2,ACK=1})))}
TOP_NAT ’ CB RX
Node19
TOP_NAT ’ Ch_C_S: 1`Ack_DataAck, longSA({SEQ=2,ACK=1}))


Node19
TOP_NAT ' Ch_C_S: 1`Ack_DataAck,
longSA({SEQ=2,ACK=1}))
Server'RcvAckSndData
Node20
TOP_NAT ' Ch_C_S: 1`Data(longSN(2))
TOP_NAT ' CB TX
Node21
TOP_NAT'CH3:1`{src_add=(10,1,1,3,4321) ,
dst_add=(155,99,25,11,62000),dccp= Data(longSN(2))}
TOP_NAT ' NAT B TX
Node22
TOP_NAT'CH3:1`{src_add=(138,76,29,7,31000) ,
dst_add=(155,99,25,11,62000),dccp= Data(longSN(2))}
TOP_NAT ' NAT A RX
Node23
TOP_NAT'CH3:1`{src_add=(138,76,29,7,31000) ,
dst_add=(10,0,0,1,4321),dccp= Data(longSN(2))}
TOP_NAT ' CA RX
Node24
TOP_NAT ' Ch_S_C: 1`Data(longSN(2))
Client ' RcvData
Node25

6.4 สรุป

1) มีความรู้ความเข้าใจเกี่ยวกับการใช้งานโปรแกรม CPN Tools
2) รู้จักการทำงานของโปรโตคอล DCCP ที่มีการทำงานแบบ Simultaneous Open
3) มีความรู้ความเข้าใจในเรื่องของ Hole Punching ของ NAT
4) รู้จักการทำงานแบบ SDP (Session Description Protocol)
5) สามารถจำลองการทำงานของ NAT เมื่อใช้งานกับโปรโตคอล DCCP เพื่อตรวจสอบแบบจำลองให้ทำงานได้อย่างถูกต้อง
6) จากขอบเขตการทำงานที่จำลองการทำงานของ NAT เมื่อใช้กับโปรโตรคอล DCCP ให้มีการติดต่อกันแบบ Simultaneous Open โดยใช้หลักการ Hole Punching เมื่อทำการทดสอบแบบจำลองแล้วยังไม่พบข้อผิดพลาด

บทที่ 7
สรุปผลและข้อเสนอแนะ

7.1 บทนำ

เนื้อหาในบทนี้จะเป็นการสรุปผลการดำเนินโครงการ รวมไปถึงปัญหาและแ นวทางแก้ไขปัญหาในระหว่างการดำเนินโครงการ และยังมีข้อเสนอแนะที่จะแนะนำแนวทา งในการพัฒนาเพื่อเป็นประโยชน์ในการศึกษาการสร้างและวิเคราะห์แบบจำลองต่อไป

7.2 สรุปผลการดำเนินโครงการ

โปรแกรม CPN Tools นี้สามารถทำง านได้ตามที่คาดหมาย แต่อาจจะมีในบางส่วนของโปรแกรมไม่สามารถทำงานได้ดีเท่าที่ควร เนื่องมาจากทางคณะผู้จัดทำยังขาด ทักษะ ความรู้ ความชำนาญในการใช้ฟังก์ชันต่างๆ ภายในตัวโปรแกรม CPN Tools จึงไม่สามารถทำโปรแกรมในส่วนนี้ได้อย่างรวดเร็ว แต่ผลโดยรวมถือว่าได้ผลสำเร็จตามเป้าหมาย

7.3 ปัญหาและแนวทางแก้ไข

ปัญหาที่พบส่วนใหญ่ในการทำงานนั้น จะเกิดขึ้นขณะทำแบบจำลอง DCCP โดยใช้โปรแกรม CPN Tools ซึ่งปัญหาที่พบบ่อยมีดังนี้

1. ในระหว่างการสร้างแบบจำลอง DCCP นั้น จะปรากหน้าต่างของ BETA Runtime Notification ขึ้น ทำให้โปรแกรม CPN Tools ที่ใช้อยู่มีปัญหาไม่สามารถใช้งานต่อได้ต้องทำการปิดโปรแกรม แล้วเปิดใหม่ ทำให้เสียเวลาในการทำงาน

2. เมื่อทำการสร้างแบบจำลอง DCCP ได้อย่างสมบูรณ์และถูกต้องแล้ว จากนั้นทำการรันโปรแกรม CPN Tools จะพบว่าเกิด error ขึ้น ต้องทำการปิดโปรแกรมแล้วเปิดใหม่ แล้วรันโปรแกรม CPN Tools อีกครั้งจะทำให้รันโปรแกรมสำเร็จและสมบูรณ์

จากปัญหาดังกล่าวน่าจะเกิดจาก Window ของเครื่องคอมพิวเตอร์ที่ใช้ในการทำงานไม่สามารถรองรับโปรแกรม CPN Tools ได้ หรือสามารถรองรับได้แต่ยังไม่สมบูรณ์ ทำให้เกิดปัญหาดังกล่าวในการสร้างแบบจำลอง DCCP

หมายเหตุ: ในการสร้างแบบจำลองนั้น จะใช้คอมพิวเตอร์ ASER Aspire 4730Z แรม 1 GB ระบบ Microsoft Window XP Proessional Version 2002 Service Pack 3

7.4 ข้อเสนอแนะ

เนื่องจากการทำงานของโปรแกรม CPN Tools นี้สามารถนำไปใช้งานกับระบบที่มีความซับซ้อนมากๆ ถ้ามีผู้สนใจและนำไปพัฒนาต่อไป อาจจะทำให้ระบบต่าง ๆ ที่ใช้งานขึ้นอยู่ในปัจจุบันหรือขั้นตอนการทำงานที่สามารถนำโปรแกรม CPN Tools ไปประยุกต์ใช้ได้นั้น มีความสะดวกสบายมากขึ้นเป็นไปได้และเป็นไปได้ในอนาคตอาจจะมีการใช้กันอย่างแพร่หลายอีกด้วย

ประวัติผู้เขียน

นางสาวจิตรา จันทร์สมบูรณ์ เกิดเมื่อวันที่ 15 กันยายน 2530 ภูมิลำเนาอยู่ที่ บ้านเลขที่ 100/2 หมู่ที่ 4 ตำบลท่าหิน อำเภอสวี จังหวัดชุมพร สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายโรงเรียนสวีวิทยา อำเภอสวี จังหวัดชุมพร เมื่อ ปีการศึกษา 2548 ปัจจุบันเป็นนักศึกษาชั้นปีที่ 4 สาขาวิชาวิศวกรรมโทรคมนาคม สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

นางสาวสาวิตรี นาสวัสดิ์ เกิดเมื่อวันที่ 22 เมษายน 2531 ภูมิลำเนาอยู่ที่ บ้านเลขที่ 75 หมู่ที่ 4 ตำบลหนองยายโต๊ะ อำเภอชัยบาดาล จังหวัดลพบุรี สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายโรงเรียนชัยบาดาลวิทยา อำเภอ ชัยบาดาล จังหวัดลพบุรี เมื่อ ปีการศึกษา 2548 ปัจจุบันเป็นนักศึกษาชั้นปีที่ 4 สาขาวิชาวิศวกรรมโทรคมนาคม สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

บรรณานุกรม

[1] **Internet-Draft DCCP Simultaneous-Open Technique** http://www.erg.abdn.ac.uk

[2] J.Billington and S.Vanit-Anunchai , **Modelling the Datagram Congestion Control Protocol**

[3] **Coloured Petri Nets and CPN Tools for modelling and validationof concurrent systems**, Kurt Jensen and Lars Michael Kristensen

[4] S.Vanit-Anunchai ,**An Investigation of the Datagram Congestion Control Protocol's Connection Management and Synchronisation Procedures**

[5] **Network Address Translation (NAT),** http://www.sans.org/infosecFAQ/firewall/net_add.htm

[6] **RFC 5596,** https://datatracker.ietf.org/doc/rfc5596/

[7]**RFC 4340**, http://www.faqs.org/rfcs/rfc4340.html

ภาคผนวก
Declaration ทั้งหมดที่ใช้ในการสร้างแบบจำลอง

```
(* Standard declarations *)
    colset UNIT = unit;
    colset INT = int;
    colset BOOL = bool;
    colset STRING = string;
colset COMMAND = with p_Open | a_Open | server_a_Close | a_Close;
var cmd:COMMAND;
val pr = 4;val ONE   = 1;  val ZERO = 0;
val MaxSeqNo48 = 32000;
val MaxSeqNo24 = 1600;
val max_seq_no24 = 1600;
val MaxSeqNo48plus1   = 32001;
val MaxSeqNo24plus1   = 1601;
val LARGE   = MaxSeqNo48;
colset SN = int;
colset Ack_DataAckPktTypes =  with DataAck | Ack;
val ack_dataack:Ack_DataAckPktTypes = Ack;
colset OtherPktTypes = with Sync | SyncAck | Response | CloseReq | Close | Rst;
var p_type:OtherPktTypes;
colset X = with LONG | SHORT;
colset SN48 = int with ZERO..MaxSeqNo48;
colset SN24 = int with 0..max_seq_no24;
colset SN48_AN48 = record SEQ:SN48*ACK:SN48;
colset SN24_AN24 = record SEQ:SN24*ACK:SN24;
colset LongSN_U_ShortSN = union longSN:SN48  + shortSN:SN24;
colset LongSA_U_ShortSA = union longSA:SN48_AN48 + shortSA:SN24_AN24;
```

```
var sn48Usn24:LongSN_U_ShortSN;
var sa48Usa24:LongSA_U_ShortSA;
colset Ack_DataAckPacket  = product Ack_DataAckPktTypes*LongSA_U_ShortSA;
colset OtherTypesPacket    = product OtherPktTypes*SN48_AN48;
colset PACKETS = union Request:SN48 + Listen:SN48 + Reset:SN48
 + Data:LongSN_U_ShortSN
 + Ack_DataAck:Ack_DataAckPacket
 + PKT:OtherTypesPacket
declare of_Request, of_Data, of_Ack_DataAck, of_PKT;
colset LPACKETS = list PACKETS;
var sn, sn48:SN48;
var sn_an, sn48_an48:SN48_AN48;
var sn24:SN24;
var sn24_an24:SN24_AN24;
var packet:PACKETS;
colset FN_ShortAllow = bool;
colset RCNT = int;(* Retransmit Counter *)
colset ACTIVE_STATE = with  RESPOND | PARTOPEN | S_OPEN | C_OPEN | CLOSEREQ |
C_CLOSING |S_CLOSING | RESPONDx;
colset IDLE = with CLOSED_I | LISTEN | TIMEWAIT | CLOSED_F | LISTEN1;
colset RCNTxGSSxISS = product RCNT*SN48*SN48; (* counter, gss, iss  *)
colset GS = record GSS:SN48*GSR:SN48*GAR:SN48;
colset ISN = record ISS:SN48*ISR:SN48;
colset ActiveStatexRCNTxGSxISN = product ACTIVE_STATE*RCNT*GS*ISN;
colset CB = union IdleState:IDLE+InvitedState:RCNT + ReqState:RCNTxGSSxISS +
ActiveState:ActiveStatexRCNTxGSxISN
declare of_IdleState, of_ReqState, of_ActiveState;
var Lis_or_invit:CB;
var rcnt:RCNT;
var state:ACTIVE_STATE;
```

```
var gss,gsr,gar,iss,isr:SN48;
var g:GS;
var isn:ISN;
var cb:CB;
var id_state:IDLE;
use "C:/x1.sml";
colset IP=product INT*INT*INT*INT*INT;
colset RG_TABLE=record ab:STRING * lc:IP* gb:IP;
colset PKT1=record src:IP*dst:IP*res:STRING*lc:IP*gb:IP;
colset PKTB=record src:IP*dst:IP*res:STRING;
colset PAIR=record P1:IP*P2:IP*P3:IP;
colset DATA=record src:IP*dst:IP;
colset DATA1=record src:IP*dst:IP*lc:IP*res:STRING;
colset IP_PKT=record src_add:IP*dst_add:IP*dccp:PACKETS;
var sa1,sb1,s2,s3,da1,da2,dax:IP;
var daxx,daxxx,sx,sxx,db1,db2:IP;
var ipgba,ipgbb,iplca,iplcb:IP;
var data1,data2,data3,data4:IP;
var abcd:STRING;
var name,nameA,name1,data:STRING;
val IP_GB_A:IP =(155,99,25,11,62000);
val IP_GB_B:IP =(138,76,29,7,31000);
```

use "C:/x1.sml";

```
datatype seq_ack = S48 of SN48
| SA48 of SN48_AN48
| S24 of SN24
| SA24 of SN24_AN24
| SN48uSN24 of LongSN_U_ShortSN
```

```
| SA48uSA24 of LongSA_U_ShortSA;

datatype state_variable = NoGS | gssGS of SN48 | gGS of GS;
fun incr(snl):SN48 = (* if sequence number equals 2^48-1 then go back to 0 *)
if (snl = MaxSeqNo48) then ZERO else (snl + ONE);

fun Wrap(seq_b:SN48, MaxValue:SN48):SN48 = (* MaxValue = MaxSeqNo24p1 *)
if (seq_b < ZERO) then (seq_b + MaxValue)
else (if (seq_b > (MaxValue - ONE) ) then (seq_b - MaxValue)
else seq_b);

val w = 100; val aw = 100; (* client's/server's sequence/ack window size *)
val center = Int.div(MaxSeqNo48,2); (* awh = center *)
val quater = w;
val swl = center+1-quater;
val three_quater = Int.div((w*3),4);
val swh = center+three_quater;
val awl = (center+1-aw);
val awh = center;

fun Update(new:SN48, old:SN48):SN48 = (* compare received sn/ack with GSR/GAR *)
(* if sn/ack > GSR/GAR then update GSR/GAR *)
let
val bias = (new - center); (* Rotate new --> center *)
val old_bias = Wrap((old - bias),MaxSeqNo48plus1);
in
if (center > old_bias) then new else old
end;
(* Input GSR, ShortSeq24bits : Output SeqNo48 bits *)
```

```
fun extend_seq(REF:SN48, sn24:SN24):SN48 =
let
val S = (sn24);
val center24 = 1600; (* awh = center *)
val REF_hi = Int.quot(REF,MaxSeqNo24plus1);
val REF_lo = Int.mod(REF,MaxSeqNo24plus1);
val bias = (REF_lo - center24);
val s_bias = Wrap(sn24,bias) - MaxSeqNo24plus1;
in

if (S < REF_lo) andalso (center24 < s_bias)
then ((Wrap((REF_hi + ONE),MaxSeqNo24plus1) * MaxSeqNo24plus1) + S)
else ( if (S > REF_lo) andalso (center24 > s_bias)
then ((Wrap((REF_hi - ONE),MaxSeqNo24plus1)
*MaxSeqNo24plus1) + S)
else ((REF_hi * MaxSeqNo24plus1) + S) )
end;

fun extendSA(g:GS,sn24_an24:SN24_AN24):SN48_AN48 =
{SEQ=extend_seq(#GSR(g),#SEQ(sn24_an24)),
ACK=extend_seq(#GSS(g),#ACK(sn24_an24))};

fun mod_sn24(sn48:SN48):SN24 = Int.mod(sn48, MaxSeqNo24plus1);

fun SeqAck(NoGS, SA48uSA24 (longSA sn48_an48)):SN48_AN48
= {SEQ=incr( #ACK(sn48_an48)), ACK= #SEQ(sn48_an48)}
| SeqAck(NoGS, SA48uSA24 (shortSA sn24_an24)):SN48_AN48
= {SEQ=Int.mod( (#ACK(sn24_an24)+1), MaxSeqNo24plus1),
ACK= ( #SEQ(sn24_an24))}
| SeqAck(NoGS, SN48uSN24 (longSN sn48)):SN48_AN48
```

```
= {SEQ= ZERO, ACK=sn48}
| SeqAck(NoGS, SN48uSN24 (shortSN sn24)):SN48_AN48
= {SEQ= ZERO, ACK= (sn24)}
| SeqAck(gGS g,S48 sn48) = {SEQ=incr(#GSS(g)),ACK=Update(sn48,#GSR(g))}
| SeqAck(gGS g,SA48 sn_an)
= {SEQ=incr(#GSS(g)),ACK=Update(#SEQ(sn_an),#GSR(g))};

fun SeqLS(LONG, g:GS) = longSN (incr(#GSS(g)))
| SeqLS(SHORT,g:GS) = shortSN (mod_sn24(incr(#GSS(g))));

fun SeqAckLS(LONG, gGS g, longSA sn48_an48) = longSA {SEQ=incr(
#GSS(g)),ACK=Update( #SEQ(sn48_an48), #GSR(g))}
| SeqAckLS(SHORT, gGS g, longSA sn48_an48) = shortSA {SEQ=mod_sn24(incr( #GSS(g))),
ACK=mod_sn24(Update(#SEQ(sn48_an48), #GSR(g))) }
|SeqAckLS(LONG, gGS g, shortSA sn24_an24) = longSA {SEQ=incr( #GSS(g)),
ACK=Update( #GSR(g),extend_seq( #GSR(g), #SEQ(sn24_an24)) )}
| SeqAckLS(SHORT, gGS g, shortSA sn24_an24) = shortSA {SEQ=mod_sn24(incr( #GSS(g))),
ACK=mod_sn24(Update( #GSR(g),extend_seq( #GSR(g), #SEQ(sn24_an24)) ))}
| SeqAckLS(LONG, gssGS gss, longSA sn48_an48) = longSA {SEQ=incr(gss),ACK=
#SEQ(sn48_an48)}
| SeqAckLS(SHORT, gssGS gss, longSA sn48_an48) = shortSA {SEQ = mod_sn24(incr(gss)),
ACK= mod_sn24( #SEQ(sn48_an48))};

fun SndRstInReq(gss:SN48) = 1` PKT (Rst, {SEQ=incr(gss),ACK=ZERO});
fun SyncSnd(g:GS,SA48 sn_an) = 1` PKT (Sync, {SEQ=incr(#GSS(g)),ACK= #SEQ(sn_an) } )
| SyncSnd(g:GS,S48 sn) = 1` PKT (Sync, {SEQ=incr( #GSS(g)),ACK=sn } )
| SyncSnd(g:GS,SA48uSA24 (longSA sn_an))
= 1` PKT (Sync, {SEQ=incr( #GSS(g)),ACK= #SEQ(sn_an) } )
| SyncSnd(g:GS,SN48uSN24 (longSN sn))
= 1` PKT (Sync, {SEQ=incr( #GSS(g)),ACK=sn } )
```

```
| SyncSnd(g:GS,SA48uSA24 (shortSA sn24_an24))
= 1` PKT (Sync, {SEQ=incr( #GSS(g)),ACK=extend_seq( #GSR(g), #SEQ(sn24_an24)) } )
| SyncSnd(g:GS,SN48uSN24 (shortSN sn24))
= 1` PKT (Sync, {SEQ=incr( #GSS(g)),ACK=extend_seq( #GSR(g),sn24) } );

(********** In CLOSED, LISTEN, REQUEST states, no Sync is sent out **********)

fun UpdateGS(g:GS, SA48 sn_an) = {GSS=incr(#GSS(g)),GSR=Update(#SEQ(sn_an),#GSR(g)),
GAR=Update(#ACK(sn_an),#GAR(g)) }
| UpdateGS(g:GS, S48 sn) = {GSS=incr(#GSS(g)),GSR=Update(sn,#GSR(g)),GAR= #GAR(g)}
| UpdateGS(g:GS, SA24 sn24_an24) = {GSS=incr( #GSS(g)),GSR=Update(
#GSR(g),extend_seq(#GSR(g),#SEQ(sn24_an24))),
GAR=Update(#GAR(g),extend_seq(#GSS(g),#ACK(sn24_an24)))}
| UpdateGS(g:GS, S24 sn24) =
{GSS=incr(#GSS(g)),GSR=Update(#GSR(g),extend_seq(#GSR(g),sn24) ),GAR= #GAR(g)}
| UpdateGS(g:GS, SA48uSA24 (longSA sn_an))
= {GSS=incr(#GSS(g)),GSR=Update(#SEQ(sn_an),#GSR(g)),
GAR=Update(#ACK(sn_an),#GAR(g)) }
| UpdateGS(g:GS, SA48uSA24 (shortSA sn24_an24))
= {GSS=incr(#GSS(g)),GSR=Update(#GSR(g),extend_seq(#GSR(g),#SEQ(sn24_an24))),
GAR=Update(#GAR(g),extend_seq(#GSS(g),#ACK(sn24_an24)) )}
| UpdateGS(g:GS, SN48uSN24(longSN sn))
= {GSS=incr(#GSS(g)),GSR=Update(sn,#GSR(g)),GAR= #GAR(g)}
| UpdateGS(g:GS, SN48uSN24(shortSN sn24))
= {GSS=incr(#GSS(g)),GSR=Update(#GSR(g),extend_seq(#GSR(g),sn24) ),GAR= #GAR(g)};


fun UpdateGSR(g:GS, SA48uSA24 (longSA sn48_an48))
= Update(#SEQ(sn48_an48),#GSR(g))
| UpdateGSR(g:GS, SA48uSA24 (shortSA sn24_an24))
```

```
=Update(#GSR(g),extend_seq(#GSR(g),#SEQ(sn24_an24)))
| UpdateGSR(g:GS, SN48uSN24 (longSN sn48))
=Update(sn48,#GSR(g))
| UpdateGSR(g:GS, SN48uSN24 (shortSN sn24))
=Update(#GSR(g),extend_seq(#GSR(g),sn24));
fun UpdateGAR(g:GS, longSA sn48_an48)
=Update(#ACK(sn48_an48),#GAR(g))
| UpdateGAR(g:GS, shortSA sn24_an24)
=Update(#GAR(g),extend_seq(#GSS(g),#ACK(sn24_an24)));

fun incrGSS(g:GS) = {GSS=incr(#GSS(g)),GSR= #GSR(g),GAR= #GAR(g) };

(* Initial markings, windows size for simulation purpose*)
(* val C_iss=IntInf.fromInt(16777233); val S_iss=IntInf.fromInt(16777233); *)

val C_iss= 1; val S_iss= 1;

(*
val MaxSeqNo47plus1 = pow(IntInf.fromInt(2),47);
*)

(*
val S_iss = (MaxSeqNo24plus1 - IntInf.fromInt(2));
val C_iss=IntInf.fromInt(1000);
*)

(*
val C_iss0 = (MaxSeqNo24plus1 - IntInf.fromInt(5));
val S_iss0 = (MaxSeqNo24plus1 - IntInf.fromInt(5));
val C_iss = (MaxSeqNo24plus1 + C_iss0);
```

```
val S_iss = (MaxSeqNo24plus1 + S_iss0);
*)
(*
val C_iss = (MaxSeqNo24plus1 + C_iss1);
val S_iss = (MaxSeqNo24plus1 + S_iss1);
*)

(*
val C_iss = MaxSeqNo24plus1;
val S_iss = MaxSeqNo24plus1;
*)
(*val C_gss0 = IntInf.fromInt(5); val C_gsr0 = IntInf.fromInt(4); val C_gar0 = IntInf.fromInt(3);
val S_gss0 = IntInf.fromInt(5); val S_gsr0 = IntInf.fromInt(4); val S_gar0 = IntInf.fromInt(3);
*)

val C_gss0 = (C_iss + 5); val C_gsr0 = (C_iss + 4);
val C_gar0 = (C_iss + 2);

val S_gss0 = (S_iss + (5)); val S_gsr0 = (S_iss + (4));
val S_gar0 = (S_iss + (2));

val SEQ0 = (5); val ACK0 = (4);

val MaxRetransRequest =1; val MaxRetransAckDataAck =1;
val MaxRetransCloseReq =0; val MaxRetransClose =0;

val TRespond = true;
val ShortEnable = false;
(*var LS:X;*)
val LS=LONG;
```

```
val C_cmd = 1`a_Open; val S_cmd = 1`p_Open;
(*val C_cmd = 1`a_Close; val S_cmd = 1`server_a_Close; *)
val init_C =1`IdleState CLOSED_I;
val init_S = 1`IdleState CLOSED_I; (* Client/Server initial state *)
(*var ack_dataack:ACK_DATAACK; *)

(*
val init_C = 1`ActiveState (C_OPEN,0,{GSS=C_gss0,GSR=C_gsr0,GAR=C_gar0},
{ISS=C_iss,ISR=S_iss});
*)
(*val init_Ch_C_S = 1` PKT2 (Ack, LONG, {SEQ=SEQ0,ACK=ACK0});*)
(*val init_Ch_C_S = 1`PKT1 (Data, LONG,C_gss0); *)
(*
val init_S = 1`ActiveState (S_OPEN,0,{GSS=S_gss0,GSR=S_gsr0,GAR=S_gar0},
{ISS=S_iss,ISR=C_iss});
*)

fun go_open(state) = if state = PARTOPEN then C_OPEN else S_OPEN;
fun BackOff(state,rcnt):bool =
case state of
RESPOND => TRespond
| PARTOPEN => (rcnt=MaxRetransAckDataAck)
| C_CLOSING => (rcnt=MaxRetransClose)
| CLOSEREQ => (rcnt=MaxRetransCloseReq)
| S_CLOSING => (rcnt=MaxRetransClose)
|_ => false;

(* function checking sequence number from DCCP draft-9*)
fun ReqValid(s:SN48, g:GS, isn:ISN) =
let
```

```
val bias = (#GSR(g) - center);
val seq_b = Wrap((s - bias),MaxSeqNo48plus1);
val isr_b = Wrap((#ISR(isn) - bias),MaxSeqNo48plus1);
val SWL = Int.max(swl,isr_b);
val SWH = swh;
in
(seq_b >= SWL) andalso (seq_b <= SWH)
end;
fun DataValid(longSN s, g:GS, isn:ISN) =
let
val bias = (#GSR(g) - center);
val seq_b = Wrap((s - bias),MaxSeqNo48plus1);
val isr_b = Wrap((#ISR(isn) - bias),MaxSeqNo48plus1);
val SWL = Int.max(swl,isr_b);
val SWH = swh;
in
(seq_b >= SWL) andalso (seq_b <= SWH)
end
| DataValid(shortSN s, g:GS, isn:ISN) =
let
val bias = (#GSR(g) - center);
val s1 = extend_seq(#GSR(g),s);
val seq_b = Wrap((s1 - bias),MaxSeqNo48plus1);
val isr_b = Wrap((#ISR(isn) - bias),MaxSeqNo48plus1);
val SWL = Int.max(swl,isr_b);
val SWH = swh;
in
if ShortEnable
then (seq_b >= SWL) andalso (seq_b <= SWH)
else false
```

```
end;

fun DataAckValid(longSA s, g:GS, isn:ISN) =
let
val bias1 = (#GSR(g) - center);
val seq_b = Wrap((#SEQ(s) - bias1),MaxSeqNo48plus1);
val isr_b = Wrap((#ISR(isn) - bias1),MaxSeqNo48plus1);
val SWL = Int.max(swl,isr_b);
val SWH = swh;

val bias2 = (#GSS(g) - center);
val ack_b = Wrap((#ACK(s) - bias2),MaxSeqNo48plus1);
val iss_b = Wrap((#ISS(isn) - bias2),MaxSeqNo48plus1);
val AWL = Int.max(awl,iss_b);
val AWH = center;

in
((seq_b >= SWL) andalso (seq_b <= SWH))
andalso ((ack_b >= AWL) andalso (ack_b <= AWH))

end
| DataAckValid(shortSA s, g:GS, isn:ISN) =
let
val bias1 = (#GSR(g) - center);
val seq = extend_seq(#GSR(g),#SEQ(s));
val seq_b = Wrap((seq - bias1),MaxSeqNo48plus1);
val isr_b = Wrap((#ISR(isn) - bias1),MaxSeqNo48plus1);
val SWL = Int.max(swl,isr_b);
val SWH = swh;
```

```
val bias2 = (#GSS(g) - center);
val ack = extend_seq(#GSS(g),#ACK(s));
val ack_b = Wrap((ack - bias2),MaxSeqNo48plus1);
val iss_b = Wrap((#ISS(isn) - bias2),MaxSeqNo48plus1);
val AWL = Int.max(awl,iss_b);
val AWH = center;
in
if ShortEnable
then ((seq_b >= SWL) andalso (seq_b <= SWH))
andalso ((ack_b >= AWL) andalso (ack_b <= AWH))
else false
end;
fun SeqValid(p_type:OtherPktTypes, s2:SN48_AN48, g:GS, isn:ISN) =
let
val bias = (#GSR(g) - center);
val seq_b = Wrap((#SEQ(s2) - bias),MaxSeqNo48plus1);
val isr_b = Wrap((#ISR(isn) - bias),MaxSeqNo48plus1);
val SWL = Int.max(swl,isr_b);
val SWH = swh;
in
case p_type of
Response => (seq_b >= SWL) andalso (seq_b <= SWH)
| CloseReq => (seq_b > center) andalso (seq_b <= SWH)
| Close => (seq_b > center) andalso (seq_b <= SWH)
| Rst => (seq_b > center) andalso (seq_b <= SWH)
| Sync => (seq_b >= SWL)
| SyncAck => (seq_b >= SWL)
end;

(* function checking acknowledgement number from DCCP draft-11*)
```

```
fun AckValid(p_type:OtherPktTypes, s2:SN48_AN48, gGS g, iss:SN48) =
let
val bias = (#GSS(g) - center);
val gar_b = Wrap((#GAR(g) - bias),MaxSeqNo48plus1);
val ack_b = Wrap((#ACK(s2) - bias),MaxSeqNo48plus1);
val iss_b = Wrap((iss - bias),MaxSeqNo48plus1);
val AWL = Int.max(awl,iss_b);
val AWH = center;
in
case p_type of
Response => (ack_b >= AWL) andalso (ack_b <= AWH)
| CloseReq => (ack_b >= gar_b) andalso (ack_b <= AWH)
| Close => (ack_b >= gar_b) andalso (ack_b <= AWH)
| Rst => (ack_b >= gar_b) andalso (ack_b<=AWH)
| Sync => (ack_b >= AWL) andalso (ack_b <= AWH)
| SyncAck => (ack_b >= AWL) andalso (ack_b <= AWH)
end
| AckValid(p_type:OtherPktTypes, s2:SN48_AN48, gssGS gss, iss:SN48) =
let
val bias = (gss - center);
(* val gar_b = Wrap((gar - bias),MaxSeqNo48plus1);*)
val ack_b = Wrap((#ACK(s2) - bias),MaxSeqNo48plus1);
val iss_b = Wrap((iss - bias),MaxSeqNo48plus1);
val AWL = Int.max(awl,iss_b);
val AWH = center;
in
case p_type of
Response => (ack_b >= AWL) andalso (ack_b <= AWH)
| Sync => (ack_b >= AWL) andalso (ack_b <= AWH)
| SyncAck => (ack_b >= AWL) andalso (ack_b <= AWH)
```

```
end;
(* function for checking both seq and ack *)
fun PktValid(p_type2:OtherPktTypes, s2:SN48_AN48, g:GS, isn:ISN) =
let
val check_seq = SeqValid(p_type2:OtherPktTypes,s2:SN48_AN48, g:GS, isn:ISN);
val check_ack = AckValid(p_type2:OtherPktTypes,s2:SN48_AN48, gGS g, #ISS(isn));
in
(check_seq) andalso (check_ack)
end;
fun RstValidinReqState(s2:SN48_AN48, gss:SN48, iss:SN48) =
let
val bias = (gss - center);
val ack_b = Wrap((#ACK(s2) - bias),MaxSeqNo48plus1);
val iss_b = Wrap((iss - bias),MaxSeqNo48plus1);
val AWL = Int.max(awl,iss_b);
val AWH = center;
in
(ack_b >= AWL) andalso (ack_b <= AWH)
end;
```