การสร้างโมเดลและการจำลองการทำงานของระบบเครือข่ายไร้สายด้วยโปรแกรม OMNeT++

MODELING AND SIMULATION OF WIRELESS NETWORK USING OMNET++

โดย

เดชพงษ์ บุญญฤทธิ์

อานนท์ แซ่เบ๊

อาจารย์ที่ปรึกษา

อาจารย์ลภัส ประดิษฐ์ทัศนีย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคเรียนที่ 2 ปีการศึกษา 2550

PROPOSAL OF MODELING AND SIMULATION OF WIRELESS NETWORK USING OMNET++

DETCHPONG BOONYARIDH

ARNON SAEBAE

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF SCIENCE PROGRAM IN INFORMATION TECHNOLOGY

FACULTY OF INFORMATION TECNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2/2007

ใบรับรองปริญญานิพนธ์ ประจำปีการศึกษา 2550

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง

การสร้างโมเดลและการจำลองการทำงานของระบบเครือข่ายไร้สายด้วยโปรแกรม OMNeT++

MODELING AND SIMULATION OF WIRELESS NETWORK USING OMNET++

ผู้จัดทำ

1. นายอานนท์ แซ่เบ๊ รหัสประจำตัว 47070091
2. นายเดชพงษ์ บุญญฤทธิ์ รหัสประจำตัว 47070093

..... อาจารย์ที่ปรึกษา

(อาจารย์ลภัส ประดิษฐ์ทัศนีย์)

หัวข้อโครงงาน	การสร้างโมเดลและการจำลองการทำงานของระบบเครือข่ายไร้สายด้วยโปรแกรม OMNeT++
นักศึกษา	นายอานนท์ แซ่เบ๊ รหัสนักศึกษา 47070091
	นายเดชพงษ์ บุญญฤทธิ์ รหัสนักศึกษา 47070093
ปริญญา	วิทยาศาสตร์มหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
ปีการศึกษา	2550
อาจารย์ที่ปรึกษา	อาจารย์ลภัส ประดิษฐ์ทัศนีย์

บทคัดย่อ

โครงงานฉบับนี้เสนอการสร้างโมเดลและจำลองการทำงานของระบบเครือข่ายไร้สายด้วยโปรแกรม OMNeT++ โดยจะประกอบไปด้วยการศึกษาตัวโปรแกรม OMNeT++ ศึกษาตัวภาษา NED และการสร้างไฟล์กำหนดค่า (configuration file) แล้วจึงทำการทดสอบทำการสร้างแบบจำลองขึ้นมาเพื่อทดสอบการทำงานของระบบเครือข่ายแบบไร้สาย จากนั้นจึงนำผลการทดลองมาวิเคราะห์ เพื่อเข้าใจถึงการทำงานและประสิทธิภาพของระบบเครือข่าย

Title	Modeling and Simulation of Wireless Network using OMNeT++	
Student	Mr. Arnon Saebae	Student ID. 47070091
	Mr. Detchpong Boonyaridh	Student ID. 47070093
Degree	Bachelor of Science	
Programme	Information Technology	
Academic Year	2006	
Advisor	Mr. Lapas Pradittasanee	

ABSTRACT

This project has represented Model and simulation wireless network with OMNeT++.Including the study of program OMNeT++, NED language and configuration file for create a model. Then create a simulation for testing wireless network to get the result to process and analyze for understanding and measure performance wireless network.

กิตติกรรมประกาศ

ปริญญานิพนธ์เล่มนี้สามารถสำเร็จได้เพราะความกรุณาจากอาจารย์ที่ปรึกษาโครงงาน อาจารย์ลภัส ประดิษฐ์ทัศนีย์ ที่คอยให้ความช่วยเหลือและให้คำชี้แนะช่วยแก้ปัญหาตลอดจนให้ความรู้, ประสบการณ์ และทุ่มเทเวลาที่มีค่ายิ่งให้แก่ข้าพเจ้าทั้งหลาย

ขอขอบคุณกรรมการสอบ รศ.ดร. โชติพัชร์ ภรณวลัย, ผศ.ดร. จันทร์บูรณ์ สถิตวิริยวงศ์ และผศ. อัครินทร์ คุณกิตติ ที่กรุณาให้คำแนะนำ ตลอดจนให้ข้อชี้แนะจนในที่สุดทำให้ปริญญานิพนธ์เล่มนี้สำเร็จลงได้ด้วยดี

สุดท้ายนี้ขอขอบคุณเพื่อนๆในคณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกคนที่ได้ให้การช่วยเหลือทั้งความรู้ และกำลังใจ เพื่อให้วิทยานิพนธ์เล่มนี้สำเร็จโดยสมบูรณ์

อานนท์ แซ่เบ๊

เดชพงษ์ บุญญฤทธิ์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันการติดต่อสื่อสารภายในชีวิตประจำวันนั้นเป็นสิ่งที่มีความจำเป็นอย่างมาก และได้มีการพัฒนาเทคโนโลยีที่ใช้ในการสื่อสารเรื่อยมาจนเป็นการสื่อสารในลักษณะไร้สาย ซึ่งในระบบเครือข่ายไร้สายนั้นมีบทบาทเข้ามาในชีวิตประจำวันของชีวิตคนเรามากยิ่งขึ้น โดยจะสังเกตได้จากการสื่อสารไร้สายในลักษณะต่างๆเช่น โทรศัพท์มือถือหรืออินเตอร์เน็ตไร้สาย เป็นต้น ซึ่งในการพัฒนาระบบเครือข่ายไร้สายให้ได้มีประสิทธิภาพสูงขึ้นนั้นจำเป็นที่จะต้องใช้เทคโนโลยีและเงินลงทุนจำนวนมาก จึงจำเป็นที่จะต้องมีการสร้างแบบจำลอง และทำการทดสอบก่อนที่จะนำมาพัฒนาให้กับตัวอุปกรณ์ เพื่อนำไปใช้งานได้จริง ซึ่งการทำการทดสอบในทางทฤษฎีและการทำแบบจำลองเพื่อที่จะเป็นแนวทางให้กับการพัฒนาต่อไปมีโอกาสประสบความสำเร็จสูงขึ้นโดยที่ไม่สิ้นเปลืองเงินทุนและเวลา และสามารถเป็นแนวทางให้ผู้อื่นนำไปพัฒนาต่อยอดไปได้

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ศึกษาการทำงานของระบบเครือข่ายไร้สายและใช้โปรแกรม OMNeT++ เพื่อจำลองระบบเครือข่าย แล้วทำการทดลองตามสมมุติฐานต่างๆ และนำผลลัพธ์ที่ได้จากการทดลองมาเปรียบเทียบกับผลคำนวณทางคณิตศาสตร์

1.3 ทฤษฎีหรือแนวคิดที่ใช้ในการวิจัย

การจำลองและวิเคราะห์ประสิทธิภาพเครือข่ายไร้สายด้วย OMNeT++ จะเป็นการศึกษาเครือข่ายไร้สายตั้งแต่พื้นฐาน ไปจนถึงสามารถนำไปใช้งานได้ โดยรายงานเล่มนี้จะรวบรวมเนื้อหาความรู้จากการศึกษาทฤษฎีเครือข่ายไร้สายในส่วนที่เกี่ยวข้องกับจำลองและการวิเคราะห์ รวมไปถึงคำสั่งพื้นฐานต่างๆของ OMNeT++ ที่จำเป็นในการจำลอง ซึ่งรายงานฉบับนี้ได้แสดงถึงทฤษฎี, คำสั่งต่างๆ และภาษาที่ใช้ในการจำลองและวิเคราะห์

1.4 ขอบเขตการวิจัย

ศึกษาการใช้งานและจำลองระบบเครือข่ายไร้สายโดยใช้เครื่องมือช่วยจำลองระบบเครือข่ายที่มีชื่อว่า OMNeT++ โดยเริ่มศึกษาจากคู่มือการใช้งานของโปรแกรม และศึกษาจากตัวอย่างที่มีภายในโปรแกรมก่อนเพื่อเพิ่มความชำนาญกับโปรแกรมนี้มากยิ่งขึ้น หลังจากนั้นเริ่มจำลองเครือข่ายไร้สายขึ้นเองเพื่อที่จะทดสอบตามสมมุติฐานต่าง, ดูประสิทธิภาพและข้อจำกัดต่างๆ ของโปรแกรม

1.5 ขั้นตอนของการศึกษา

- ทำการศึกษาการใช้งานของโปรแกรม OMNeT++ เพื่อจำลองระบบเครือข่ายไร้สาย
- ทำการศึกษาทฤษฎีที่เกี่ยวข้องกับกระบวนการทำงานของระบบเครือข่ายไร้สาย
- จำลองการทำงานของระบบเครือข่ายไร้สาย แล้วทดสอบเพื่อดูข้อจำกัดของโปรแกรม OMNeT++
- จำลองการทำงานของระบบเครือข่ายไร้สาย แล้วเปรียบเทียบผลที่ได้กับการคำนวณทางคณิตศาสตร์
- ปรับปรุงและแก้ไขมอดูลการทำงานของ OMNeT++ ให้ทำงานได้ตามที่ต้องการ

บทที่ 2

OMNeT++

2.1 การวิเคราะห์ประสิทธิภาพของระบบเครือข่าย

รูปแบบของการวิเคราะห์สามารถทำได้ 3 วิธีคือ

- การวิเคราะห์ด้วยหลักการทางคณิตศาสตร์ (Analytical Model) เป็นวิธีที่สามารถทำขณะใดก็ได้ ไม่ว่าจะอยู่ในช่วงของการออกแบบ หรือหลังจากการติดตั้งระบบเครือข่ายไปแล้ว ซึ่งเป็นวิธีที่ง่ายและวิเคราะห์ได้อย่างรวดเร็วเพราะสามารถแทนค่าลงไปในสมการได้เลย แต่ผลการวิเคราะห์ที่ได้จากการคำนวณนั้นจะเป็นเพียงค่าในแง่ทฤษฎี ซึ่งอาจจะไม่แม่นยำมากพอ แต่ก็เป็นวิธีที่ใช้ค่าใช้จ่ายต่ำกว่าเมื่อเทียบกับวิธีอื่น
- การจำลอง (Simulation) จะมีลักษณะคล้ายกับวิธีการวิเคราะห์ด้วยหลักการทางคณิตศาสตร์ แต่จะมีความแม่นยำที่มากกว่า เนื่องจากการจำลองจะทำให้เห็นภาพได้ง่ายกว่า และสร้างแบบจำลองระบบขึ้นมาแทนระบบจริงได้ และมีคุณสมบัติที่ใกล้เคียงความเป็นจริง แต่ก็จะขึ้นอยู่กับความสามารถของแบบจำลองด้วย โดยการวิเคราะห์นั้นจะทำผ่านภาษาโปรแกรม ทำให้จำเป็นที่จะต้องมีความสามารถด้านนี้ด้วย
- การวัดจากระบบจริง (Measurement) จะเป็นวิธีที่มีความแม่นยำมากที่สุดเนื่องจากจะต้องทำหลังจากการติดตั้งระบบเครือข่ายแล้วเท่านั้น ซึ่งค่าที่ได้จากการวิเคราะห์คือค่าที่เกิดขึ้นจริง แต่วิธีนี้จะใช้ค่าใช้จ่ายสูงที่สุดเนื่องจากจะต้องติดตั้งระบบก่อน

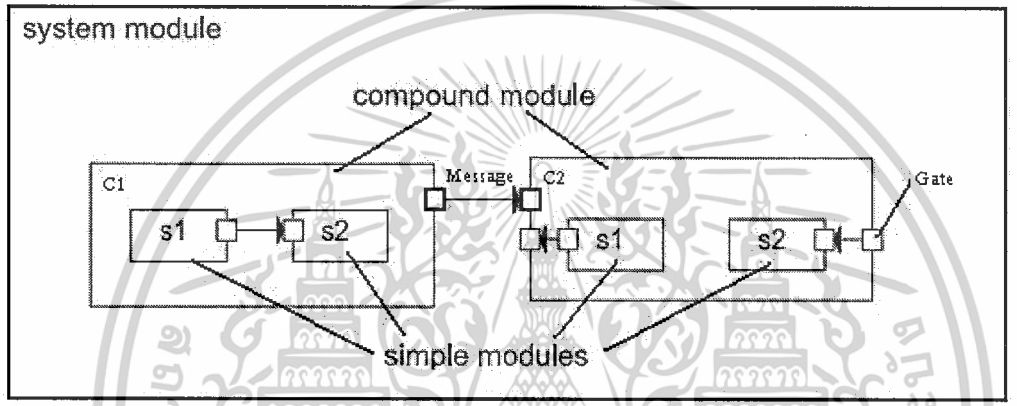
2.2 ประเภทของการจำลอง

การจำลองสามารถแบ่งได้หลายประเภทดังนี้

1. Emulation เป็นการจำลองที่ใช้ฮาร์ดแวร์เข้ามาช่วย
2. Monte Carlo Simulation เป็นการจำลองที่ไม่เกี่ยวกับเวลา จะทำการทดลองซ้ำๆเพื่อหาค่าทางสถิติ
3. Trace-driven Simulation เป็นการจำลองโดยใช้ข้อมูลที่ได้จากการทดลองจริงมาเป็นอินพุตของระบบที่จำลองขึ้นมา
4. Discrete-time Simulation เป็นการจำลองระบบที่ไม่ต่อเนื่องทางเวลา การจำลองแบบนี้จะสามารถตั้งเวลาจำลองเหตุการณ์ด้วยเวลาที่คงที่ (unit time approach) หรือการจำลองด้วยเวลาที่ควบคุมด้วยเหตุการณ์ที่เกิดขึ้น (event-driven approach)

2.3 โครงสร้างของ OMNeT++

แบบจำลอง OMNeT++ นั้นจะประกอบด้วยกลุ่มลำดับชั้นของมอดูล ซึ่งอนุญาตให้ผู้ใช้สามารถมองเห็นถึงโครงสร้างของมอดูลในระดับ logical ของระบบที่แท้จริงของโครงสร้างในแบบจำลอง การติดต่อสื่อสารกันระหว่างมอดูลจะติดต่อกันด้วยข้อความ ซึ่งแต่ละข้อความจะบรรจุด้วยโครงสร้างข้อมูลที่ซับซ้อนได้โดยไม่มีกฎเกณฑ์ ในแต่ละมอดูลสามารถส่งข้อความหลายๆ ข้อความไปยังปลายทางได้ทั้งแบบโดยตรงหรือแบบกำหนดเส้นทาง, ทางผ่าน (Through gates) และรูปแบบการเชื่อมต่อได้



รูปที่ 2.1 โครงสร้างของ OMNeT++

ในแต่ละมอดูลสามารถมีค่าตัวแปรต่างๆเป็นของตัวเองได้เอาไว้ใช้สำหรับปรับเปลี่ยนพฤติกรรมต่างๆของมอดูลและไว้กำหนดรูปแบบโครงสร้างของเครือข่าย ดังรูป

```
module WirelessAP
    gates:
        in: radioIn;
    submodules:
        notificationBoard: NotificationBoard;
            display: "p=79,74;i=block/control";
        wlan: Ieee80211NicAP; // see also Ieee80211NicAPSimplified
            display: "p=110,179;q=queue;i=block/ifcard";
        mobility: NullMobility;
            display: "p=144,70;i=block/cogwheel_s";
    connections nocheck:
        radioIn --> wlan.radioIn display "m=s";
endmodule
```

รูปที่ 2.2 แสดงตัวแปรต่างๆในมอดูล

จากด้านบน เราสามารถเปลี่ยนแปลงค่าของตัวแปร ทำให้เกิดการแสดงผลที่แตกต่างไปจากเดิมเช่น การกำหนดตำแหน่งการวางของ submodule เป็นต้น

2.4 ขอบเขตของ OMNeT++

OMNeT++ คือโปรแกรมที่ช่วยในการจำลองเหตุการณ์การทำงานต่างๆของระบบเครือข่าย โดยมองแต่ละมอดูลแยกออกจากกันเป็นวัตถุและโปรแกรมนี้สามารถใช้ได้กับ

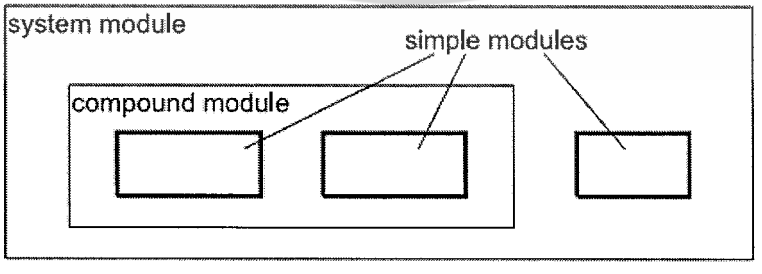
- การจำลองการสื่อสารภายในเครือข่าย
- การจำลองข้อตกลงที่ใช้ในการติดต่อสื่อสารภายในเครือข่าย
- การจำลองการจัดลำดับแถวคอยของการส่งข้อมูล (Queuing) ภายในเครือข่าย
- การจำลองหน่วยประมวลที่มากกว่าหนึ่งและระบบการกระจายฮาร์ดแวร์ไปยังที่อื่นๆ
- การใช้กับสถาปัตยกรรมของฮาร์ดแวร์ต่างๆที่ถูกต้องตามความเป็นจริง
- การประเมินประสิทธิภาพทางด้านความซับซ้อนของระบบซอฟท์แวร์
- การจำลองระบบอื่นๆที่ใกล้เคียงกับเหตุการณ์แบบไม่ต่อเนื่องได้อย่างเหมาะสม

2.5 แนวคิดของแบบจำลอง

OMNeT++ นั้นจัดหาเครื่องมือที่มีประสิทธิภาพสำหรับให้ผู้ใช้นำไปอธิบายถึงโครงสร้างของระบบที่เกิดขึ้นจริง ซึ่งมีจุดเด่นหลักๆดังนี้

2.5.1 การทำงานเป็นลำดับชั้นภายในมอดูล

แบบจำลอง OMNeT++ จะประกอบด้วยลำดับชั้นของมอดูลย่อยๆซ้อนกันอยู่ และจะติดต่อสื่อสารกันด้วยข้อความ โดยมีมอดูลย่อยที่อยู่ภายนอกสุดก็คือ system module ซึ่งจะเป็นตัวที่บรรจุมอดูลย่อยๆเรียกว่า compound module ซึ่งในแต่ละ compound module ก็สามารถบรรจุมอดูลซ้อนลงไปได้อีก (ดังรูป) โดยไม่จำกัดจำนวนชั้นที่ซ้อนกัน ทำให้ผู้ใช้ได้เห็นถึงโครงสร้างในระดับ logical ของระบบที่แท้จริงภายในโครงสร้างของแบบจำลอง



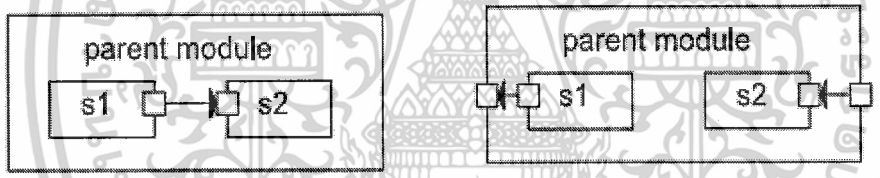
รูปที่ 2.3 การแบ่งลำดับชั้นภายในมอดูล

จากรูปนั้นบอกถึงมอดูลย่อยของ compound modules คือ simple modules ซึ่งเป็นมอดูลที่อยู่ในระดับล่างสุดของลำดับชั้นของรูปนี้ และเป็นที่เก็บแนวทางในการแก้ไขปัญหาต่างๆภายในแบบจำลอง ซึ่งผู้ใช้นั้นสามารถที่จะนำ simple module เหล่านี้ไปใช้งานได้โดยใช้ภาษา C++ และใช้ OMNeT++ จำลองเป็น class ขึ้นมา

2.5.2 การติดต่อสื่อสารผ่านทางข้อความ

การติดต่อสื่อสารระหว่างมอดูลนั้นจะติดต่อกันด้วยข้อความผ่านช่องทางการติดต่อสื่อสาร ข้อความที่ส่งจะเทียบได้กับ frame หรือ packet ของระบบเครือข่ายคอมพิวเตอร์ ซึ่งในแต่ละข้อความนั้นสามารถบรรจุโครงสร้างของข้อมูลที่ซับซ้อนได้อย่างไม่มีกฎเกณฑ์ simple module จะสามารถส่งข้อความไปยังมอดูลปลายทางได้โดยตรงหรือจะกำหนดเส้นทาง, ทางที่จะต้องผ่านออกและการเชื่อมต่อได้

การสร้างการเชื่อมต่อจะเป็นหน้าที่ของระดับล่างสุดของลำดับชั้นของมอดูล ซึ่งมีลักษณะการเชื่อมต่ออยู่ 2 แบบคือ การเชื่อมต่อกันระหว่างมอดูลย่อย และการเชื่อมต่อกันระหว่างมอดูลย่อยกับมอดูลที่ใหญ่กว่าดังรูป



การเชื่อมต่อกันระหว่างมอดูลย่อย การเชื่อมต่อระหว่างมอดูลย่อยกับมอดูลที่ใหญ่กว่า

รูปที่ 2.4 การเชื่อมต่อของมอดูล

2.5.3 การจำลองการส่ง packet

ในการเชื่อมต่อนั้นจะต้องมีการกำหนดค่าตัวแปร 3 ตัวที่เป็นตัวที่ช่วยอำนวยความสะดวกในการจำลองการติดต่อสื่อสารภายในเครือข่ายและสามารถใช้ในแบบจำลองอื่นๆได้อีกด้วย ซึ่งตัวแปรที่กล่าวถึงก็จะมี

- Propagation delay คือช่วงเวลาที่ข้อความใช้ในการเดินทางภายในช่องทางการสื่อสาร
- Bit error rate คือการระบุถึงความเป็นไปได้ของการแลกเปลี่ยนข้อมูลที่ผิดพลาดและการยินยอมให้มีการรบกวนจากภายนอกเพียงเล็กน้อยในช่องทางการติดต่อสื่อสารที่จำลองขึ้น
- Data rate คือการระบุถึงจำนวนข้อมูลที่สามารถส่งได้เป็นบิตต่อระยะเวลา 1 วินาทีและเป็นตัวแปรที่ใช้ในการคำนวณเวลาที่ใช้ในการส่ง packet

2.5.4 ค่าตัวแปรต่างๆของมอดูลนั้นๆสามารถยืดหยุ่นและเปลี่ยนแปลงได้

มอดูลแต่ละตัวจะมีตัวแปรที่สามารถกำหนดค่าต่างๆในไฟล์ NED หรือปรับแต่งค่าจะไฟล์ omnetpp.ini ค่าของตัวแปรนั้นจะใช้สำหรับปรับเปลี่ยนพฤติกรรมของมอดูลและใช้สำหรับกำหนดแบบจำลองของโครงสร้างเครือข่าย ค่าตัวแปรนั้นสามารถใส่ได้ทั้งลักษณะตัวอักษร ตัวเลข หรือ ค่าถูกผิด ดังรูป

```
# cli
**.cli.reqLength = 1000
**.cli.respLength = 0
**.cli.destAddress = "20:00:00:00:00:00"
**.cli.destStation = ""
**.cli.waitTime = 0.0005
**.writeScalars = true
```

รูปที่ 2.5 การกำหนดค่าตัวแปรในไฟล์ omnetpp.ini

2.6 การเขียนโปรแกรมเพื่อแก้ไขปัญหา

มอดูลเบื้องต้นของแบบจำลองนั้นจะถูกบรรจุอยู่ในฟังก์ชั่นของภาษาซีซึ่งเป็นภาษาที่มีความยืดหยุ่นสูง ในการจำลองผู้เขียนสามารถเลือกได้ว่าจะเขียนในลักษณะการควบคุมเหตุการณ์ที่เกิดขึ้นหรือเขียนอธิบายรูปแบบของขบวนการทำงาน สามารถใช้แนวคิดในเชิงวัตถุได้อย่างอิสระ (การมีได้หลายรูปแบบ, การสืบทอด ฯลฯ) อีกทั้งยังสามารถออกแบบรูปแบบที่ช่วยในการเพิ่มความสามารถให้กับตัวช่วยสร้างแบบจำลองได้อีกด้วย

การจำลองวัตถุ (ข้อความ, มอดูล, แถวคอย ฯลฯ) สามารถแทนได้ด้วยคลาสของภาษาซี ที่ถูกออกแบบเอาไว้ให้สามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ ตัวอย่างของคลาสที่เป็นส่วนหนึ่งของแหล่งรวบรวมคลาสของการจำลองเช่น

- มอดูล, ช่องทางเข้าออกของข้อมูล, การเชื่อมต่อ ฯลฯ
- คลาสของค่าตัวแปรต่างๆ
- ข้อความ
- คลาสที่มีข้อมูลเป็นชุดๆ (array, queue)
- คลาสที่ไว้ใช้เก็บข้อมูล
- คลาสที่เกี่ยวกับสถิติ
- คลาสที่เก็บความแม่นยำของผลลัพธ์ที่พบ

คลาสเหล่านี้จะเป็นเครื่องมือที่ช่วยให้วัตถุในแบบจำลองสามารถทำงานได้และแสดงถึงข้อมูลของวัตถุเหล่านั้น เช่น ชื่อ, ชื่อคลาส, สถานะของตัวแปรหรือเนื้อหา จุดเด่นเหล่านี้ทำให้

เป็นไปได้ที่จะสร้างแบบจำลองในส่วนติดต่อกับผู้ใช้ซึ่งเป็นส่วนที่แสดงแบบจำลองออกทางหน้าจอ

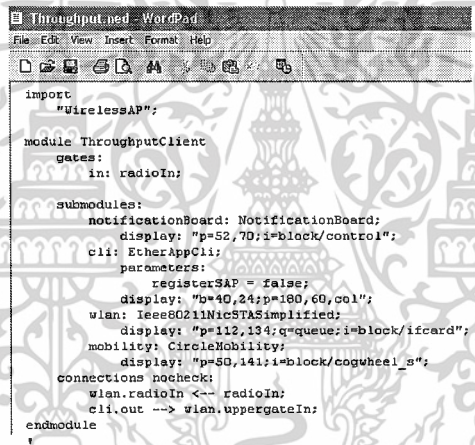
2.7 การใช้ OMNeT++

2.7.1 การสร้างและการจำลอง

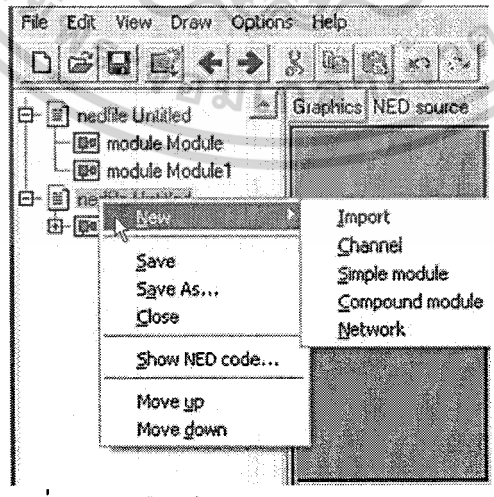
ในส่วนนี้จะมองถึงการทำงานด้วย OMNeT++ แบบพื้นฐาน ซึ่งจำแนกออกมาเป็น ไฟล์ของแบบจำลอง, การแปลงเป็นภาษาเครื่อง และพูดถึงการสร้างแบบจำลอง

แบบจำลองของ OMNeT++ จะประกอบด้วยส่วนต่างๆดังนี้

- ไฟล์ .ned คือไฟล์ที่อธิบายถึงโครงสร้างของมอดูลซึ่งจะถูกกำหนดด้วยค่าตัวแปรต่างๆ และยังสามารถเขียนได้ด้วยโปรแกรม text editor หรือ GNED graphical editor ดังรูป



รูปที่ 2.6 การแก้ไขด้วยโปรแกรม Text editor



รูปที่ 2.7 แก้ไขด้วย GNED graphical editor

- ไฟล์ .msg คือไฟล์ที่เก็บรายละเอียดของข้อความ ซึ่งเราสามารถกำหนดประเภทของข้อความได้หลายแบบ และสามารถเพิ่มขอบเขตของข้อมูลให้กับไฟล์ได้ ซึ่ง OMNeT++ จะแปลงรายละเอียดของข้อความไปเป็นคลาสของภาษาซี

- แหล่งกำเนิดของ Simple มอดูลซึ่งเป็นไฟล์ภาษาซี++จำพวกไฟล์ที่มีนามสกุลเป็น .h หรือไฟล์ที่มีนามสกุลเป็น .cc

ระบบการจำลองจะต้องประกอบด้วยส่วนประกอบดังนี้

- แกนหลักของแบบจำลอง (Simulation kernel) จะเป็นส่วนที่บรรจุชุดของคำสั่ง(code)ที่ใช้จัดการแบบจำลองและแหล่งเก็บคลาสที่เกี่ยวกับแบบจำลอง ซึ่งจะเขียนด้วยภาษาซี++ จะถูกแปลงเป็นภาษาเครื่องและไปใส่ไว้ในแหล่งเก็บ เช่นไฟล์ที่มีนามสกุลเป็น .a หรือ .lib

- ส่วนติดต่อกับผู้ใช้ (user interfaces) ของ OMNeT++ จะใช้ในขั้นตอนการดำเนินการจำลองเพื่ออำนวยความสะดวกในการแก้ไขข้อผิดพลาด, ง่ายต่อการแสดงให้เห็นจริง หรือสะดวกต่อการจำลองการดำเนินการแบบเป็นกลุ่มๆ ซึ่งมีส่วนติดต่อกับผู้ใช้หลายๆอันที่ถูกเขียนขึ้นโดยภาษาซี++ และถูกแปลงเป็นภาษาเครื่อง สุดท้ายก็จะนำไปเก็บไว้รวมกันในแหล่งเก็บ(libraries) เช่นไฟล์ที่มีนามสกุลเป็น .a หรือ .lib

โปรแกรมที่ช่วยในการจำลองถูกสร้างมาจากส่วนประกอบที่กล่าวมาข้างต้น โดยเริ่มจากไฟล์ .msg จะถูกแปลงไปเป็นชุดคำสั่งของภาษาซี++ โดยโปรแกรม opp_msgc หลังจากนั้นจะทำการแปลงคำสั่งของภาษาซี++ทั้งหมดไปเป็นภาษาเครื่องและเชื่อมต่อไปยังแกนหลักของแบบจำลอง (Simulation kernel) และแหล่งเก็บส่วนติดต่อกับผู้ใช้ เพื่อสร้างรูปแบบจำลองขึ้นมา ส่วนไฟล์ NED ก็สามารถแปลงไปอยู่ในรูปแบบของภาษาซี++โดยใช้ nedtool และเชื่อมต่อหรือโหลดแบบ dynamic ไปเป็นรูปแบบตัวอักษรปกติเมื่อเริ่มโปรแกรมช่วยจำลองเริ่มทำงาน

2.7.2 การเริ่มการจำลองและการวิเคราะห์ผลลัพธ์

เครื่องมือสร้างแบบจำลองนี้เป็นโปรแกรม standalone จึงสามารถทำงานบนเครื่องอื่นๆได้โดนไม่จำเป็นต้องมี OMNeT++ หรือไฟล์ของแบบจำลองที่มีก่อนหน้านี้ เมื่อโปรแกรมเริ่มการทำงาน มันจะอ่านค่าต่างๆที่กำหนดอยู่ในไฟล์ omnetpp.ini ซึ่งเป็นไฟล์ที่รวบรวมค่าการติดตั้งที่เอาไว้ควบคุมการทำงานของแบบจำลอง, ค่าตัวแปรต่างๆของแบบจำลอง ฯลฯ ไฟล์นี้จะเป็นตัวออกคำสั่งให้เริ่มการจำลอง

ผลลัพธ์ที่ได้จากการจำลองระบบจะถูกบันทึกลงในไฟล์ข้อมูลเช่น ไฟล์ที่เก็บค่าเวกเตอร์, ไฟล์ที่เก็บเป็นค่าสเกล่าร์ หรือจะเก็บเป็นไฟล์เฉพาะของผู้ใช้เองก็ได้ โปรแกรมOMNeT++ได้เตรียมอุปกรณ์ช่วยแสดงผลที่มีชื่อว่า Plove ไว้สำหรับดูและกำหนดจุดบนกราฟจากผลลัพธ์ที่อยู่ในไฟล์เวกเตอร์

2.7.3 ส่วนต่อประสานกับผู้ใช้

จุดประสงค์หลักของส่วนติดต่อของผู้ใช้คือเพื่อให้ผู้ใช้ได้เห็นและควบคุมส่วนแบบจำลองที่สร้างขึ้น และอนุญาตให้ผู้ใช้เข้าไปแก้ไขตัวแปรและวัตถุภายในแบบจำลอง ซึ่งสิ่งเหล่านี้จำเป็นอย่างมากในช่วงการพัฒนาและการแก้ไขข้อผิดพลาดภายในโครงการสร้างแบบจำลอง โดยผู้ใช้จะทดสอบและแก้ไขข้อผิดพลาดของแบบจำลองได้ด้วยส่วนติดต่อกับผู้ใช้แบบกราฟิก

2.7.4 แหล่งเก็บข้อมูลของส่วนประกอบ (component libraries)

Module types สามารถเก็บอยู่ในไฟล์ได้หลายๆที่ที่มีการใช้งานจริง ทำให้ผู้ใช้สามารถรวมกลุ่มของ module types และสร้างแหล่งเก็บข้อมูลของส่วนประกอบ (component libraries)

2.8 INET Framework

INET Framework จะเป็นส่วนที่สนับสนุนการจำลองในเครือข่ายไร้สายและอุปกรณ์ไร้สายได้เป็นอย่างดี อีกทั้งยังสนับสนุนการติดต่อสื่อสารแบบไร้สายซึ่งมีต้นกำเนิดจาก Mobility Framework ซึ่งใน Framework นี้จุดเด่นตรงที่มีมาตรฐาน IEEE 802.11 รวมอยู่ด้วยซึ่งเป็นมาตรฐานที่สนับสนุนทั้งแบบ ad-hoc และแบบ infrastructure ซึ่งภายใน INET Framework จะมีโปรโตคอล IPv4, IPv6, TCP, UDP บรรจุอยู่

2.8.1 สถาปัตยกรรมของ INET Framework

INET Framework นั้นถูกสร้างขึ้นบน OMNeT++ และใช้แนวคิดเดียวกันนั่นก็คือการติดต่อสื่อสารระหว่างมอดูลด้วยการส่งข้อความ

2.8.2 มอดูลและโปรโตคอล

โปรโตคอลใน OMNeT++ จะถูกแทนด้วย Simple modules ซึ่งส่วนที่ติดต่อภายนอกของ Simple modules นั้นจะถูกเขียนอธิบายอยู่ในไฟล์ NED และการดำเนินการต่างๆจะถูกบรรจุอยู่ในคลาสของภาษาซี++ ด้วยชื่อเดิมเช่น TCP, IP

มอดูลเหล่านี้สามารถรวมเข้ากับ host และอุปกรณ์จากเครือข่ายอื่นๆได้อย่างอิสระด้วยภาษา NED ตัวอย่างเช่น host, router, switch, access point ฯลฯ

อย่างไรก็ตามบางมอดูลก็ไม่จำเป็นจะต้องมีโปรโตคอลก็ได้ เช่นมอดูลต่างๆดังนี้

- มอดูลที่ทำหน้าที่ครอบครองข้อมูลอย่างเดียวเช่น Routing Table
- มอดูลที่อำนวยความสะดวกในการติดต่อสื่อสารเช่น NotificationBoard
- มอดูลที่ช่วยกำหนดค่าต่างๆให้กับเครือข่ายแบบอัตโนมัติเช่น FlatNetworkConfigurator
- มอดูลที่ทำหน้าที่เคลื่อนย้ายอุปกรณ์ไร้สาย(node)ไปรอบๆเช่น ConstSpeedMobility

ส่วนหัวของโปรโตคอลและรูปแบบของ Packet จะอธิบายอยู่ใน message definition files (msg files) ซึ่งเป็นไฟล์ที่ถูกแปลไปเป็นคลาสของภาษาซี++โดยอุปกรณ์ที่ชื่อว่า opp_msgc ของ OMNeT++

2.9 NED Overview

โครงสร้างของแบบจำลอง จะถูกกำหนดโดยภาษา NED และตัวภาษา NED จะช่วยทำให้การแยกส่วนลักษณะของระบบ เครือข่าย หมายความว่าลักษณะของระบบเครือข่ายสามารถที่จะอยู่ในรูปของส่วนประกอบต่างๆ เช่น ช่องสัญญาณ, simple และ compound module type แล้วในการแยกส่วนต่างๆ จึงทำให้สามารถนำมาใช้ใหม่ได้

ในตัวไฟล์ที่เก็บข้อมูลเกี่ยวกับลักษณะของระบบ จะถูกเก็บในไฟล์สกุล .ned ไฟล์ ned สามารถถูกโหลดแบบไดนามิก เพื่อนำไปใช้ในโปรแกรมการจำลอง หรือ ถูกเปลี่ยนรูปแบบ ไปยัง C++ โดยตัว NED compiler และถูกเชื่อมโยงไปยังการประมวลผลการจำลอง

2.9.1 ส่วนประกอบของข้อมูลใน NED

ข้อมูลใน NED จะสามารถเก็บ ส่วนประกอบต่อไปนี้ ในรูปแบบของตัวเลขหรือคำสั่ง

Import directives

Chanel definitions

Simple and compound module definitions

Network definitions

2.9.2 คำที่สำรองไว้ใช้ในคำสั่ง

ผู้ที่เขียนตัวภาษา NED ต้องคำนึงถึงคำที่ถูกเก็บไว้เป็นคำสั่งโดยต้องระมัดระวัง ห้ามไปใช้เป็นชื่อใดๆ โดยคำที่ถูกสำรองไว้ก็คือ

Import, channel, endchannel, simple, endsimple, module, endmodule, error, delay, datarate, const, parameters, gates, submodule, connections, gatesize, if, for, do, endfor, network, endnetwork, nocheck, ref, ancestor, true, false, like, input, numeric, string, bool, char, xml, xmldoc

2.9.3 ตัวระบุ (identifiers)

ตัวระบุคือชื่อของมอดูล, ช่องสัญญาณ, เครือข่าย, มอดูลย่อย, พารามิเตอร์, gate, channel attribute และ function โดยตัวระบุจะต้องสามารถเขียนตัวอักษรภาษาอังกฤษทั้งตัวใหญ่, ตัวเล็ก, ตัวเลข และ underscore ("_") โดยตามหลักสากลแล้ว ตัวระบุ หากเป็นการที่รวมคำแล้วจะต้องเขียนโดยเริ่มต้นด้วยตัวพิมพ์ใหญ่ของทุกคำ ส่วนหากเป็นมอดูล, ช่องสัญญาณ และเครือข่าย จะต้อง

เขียนตัวอักษรแรกของคำด้วยตัวใหญ่ทั้งหมด และหากเป็นพารามิเตอร์, gate และมอดูลย่อย จะต้องเขียนด้วยตัวอักษรเล็กทั้งหมด

2.9.4 ข้อควรระวัง

ข้อมูลลักษณะของระบบ และ ตัวระบุ เป็นกรณีที่ควรระวัง เพราะตัวอักษรเดียวกันแต่ขนาดต่างกันก็จะถือว่าเป็นชื่อที่ต่างกันเช่น TCP กับ Tcp นับเป็นชื่อที่ต่างกัน

2.9.5 การแสดงข้อคิดเห็น

ทำได้โดยใช้ // โดยจะทำการให้บรรทัดนั้นกลายเป็นข้อคิดเห็นทั้งบรรทัด โดยจะไปสิ้นสุดที่ จุดสิ้นสุดบรรทัด

2.10 การรับคำสั่งจากภายนอก (import directive)

Import directive เป็นการนำเข้าคำสั่งจาก network description อื่นๆ หลังจากที่ได้ทำการ import แล้วจะสามรถทำให้ใช้ components ต่างๆที่ประกาศอยู่ใน network description นั้นๆ

เมื่อไฟล์ถูก import มีแค่ข้อมูลที่ประกาศไว้ถูกใช้เท่านั้น เช่นเดียวกัน การที่ตัว parent ของ NED file ที่ import เข้ามาทำการ compile จะไม่ได้เป็นเหตุให้ตัว file ที่ import มานั้น ถูก compile ไปด้วย ต้องทำการ compile ทุกตัวไม่ใช่ตัวที่เป็น top-level เพียงตัวเดียว ตัวอย่างเช่น

```
Import "ethernet";          // import ethernet.ned
```

2.11 การกำหนดช่องทาง channel definitions

การกำหนดช่องทางเป็นการกำหนดรูปแบบการเชื่อมต่อ และให้คุณลักษณะแก่รูปแบบการเชื่อมต่อนั้นๆ โดย ชื่อช่องทางสามารถใช้ใน ned description เพื่อการสร้างการเชื่อมต่อด้วยค่าต่างๆ

```
The syntax:
channel ChannelName
        //…
Endchannel
```

ทั้ง 3 คุณลักษณะสามารถถูกกำหนดในช่วงของการประกาศช่องทาง ทั้ง 3 อย่างนั้นเป็น optional ได้แก่ delay, error และ datarate delay คือค่า propagation delay หน่วยเป็นวินาที error คือค่าความน่าจะเป็นของการเกิด bit error ในการส่ง และ datarate คือค่า bandwidth ของช่องทาง หน่วยเป็น bit/sec ทั้ง 3 ตัวแปรนี้ใช้ในการคำนวณระยะเวลาในการส่ง packet ค่าต่างๆ นั้นควรเป็นค่าคงที่ ตัวอย่างเช่น

```
channel LeasedLine
        delay    0.0018  // sec
        error    1e-8
        datarate 12800   // bit/sec
endchannel
```

2.12 การกำหนดมอดูลอย่างง่าย simple module definitions

การสร้างมอดูลอย่างง่ายเป็นพื้นฐานที่จะนำไปสู่การสร้าง component ต่างๆ สำหรับ module อื่นๆ ชนิดมอดูลอย่างง่ายสามารถถูกระบุได้โดยชื่อ โดยสากลแล้วชื่อมอดูลจะใช้ตัวอักษรใหญ่นำหน้าเช่น SimpleModule และมอดูลอย่างง่ายสามารถถูกกำหนดโดยการประกาศพารามิเตอร์ และ gate

```
Syntax:
simple SimpleModuleName
        parameters :
                //...
        gates :
                //...
endsimple
```

2.12.1 ค่าพารามิเตอร์ของมอดูลอย่างง่าย (simple module parameters)

พารามิเตอร์คือตัวแปรซึ่งเป็นของมอดูล พารามิเตอร์ของมอดูลอย่างง่ายนั้นสามรถที่จะถูกดึงหรือใช้โดย อัลกอริธึมของมอดูล เช่น มอดูล TrafficGen อาจมีพารามิเตอร์ชื่อ numofMessage ซึ่งเป็นการประมาณการณ์ว่ามี message เท่าไหร่ที่ควรสร้างขึ้นมา

พารามิเตอร์สามารถถูกกำหนดขึ้นมาโดยการประกาศชื่อ อยู่ในส่วนของ parameter: แล้วชนิดของพารามิเตอร์กำหนดเป็น numeric, numeric const, bool, string หรือ xml โดยหากไม่ได้กำหนดจะมีค่าปกติคือ numeric ตัวอย่างเช่น

```
simple TrafficGen
        parameters:
                interarrivalTime,
                numOfMessage : const,
                address : string;
```

```
        gates : //…
endsimple
```

พารามิเตอร์จะถูกกำหนดจาก NED หรือจาก file config omnetpp.ini เมื่อต้องการสร้างเป็นส่วนที่จะนำไปใช้ใหม่ได้ จึงทำการกำหนดค่าจาก config file แทน

2.12.1.1 การสุ่มค่าพารามิเตอร์และค่าคงที่

ค่าพารามิเตอร์ที่เป็นตัวเลขสามารถคืนค่าเป็นเลขสุ่มได้ โดยสุ่มแบบกระจายไม่มีรูปแบบแน่นอน หรือกระจายแบบทั่วถึง เช่นตัวอย่าง ตั้งค่าพารามิเตอร์เป็น truncnormal(2,0.8) ก็จะให้ค่าที่เป็นการสุ่มแบบ truncated normal โดย มีค่า mean อยู่ที่ 2 และ standard deviation (sd) อยู่ที่ 0.8 ในทุกครั้งที่ค่าพารามิเตอร์อ่านจากมอดูลอย่างง่าย (จาก code c++) สำหรับตัวอย่างการนำไปใช้คือการที่มีประโยชน์มากในการระบุค่า interarrival times สำหรับการสร้าง packet หรือ jobs

เราอาจต้องการกำหนดค่าพารามิเตอร์เป็นการสุ่ม แต่ไม่มีการปรับเปลี่ยนหลังจากสุ่มขึ้นมาแล้ว สามารถทำได้โดยการประกาศค่าพารามิเตอร์เป็น const โดยพารามิเตอร์ const นั้นสามารถที่จะประเมินค่าที่จะใช้ได้ครั้งเดียวในตอนที่จะเริ่มทำการจำลอง แล้วจึงทำการกำหนดค่าคงที

ในการสุ่มค่าตัวแปรภายใน OMNeT++ นั้นสามารถทำให้อยู่ในลักษณะการกระจายที่แตกต่างกันได้หลายรูปแบบโดยสามารถกำหนดได้ดังตารางนี้

ตารางที่ 2.1 ตารางฟังก์ชั่นการสุ่มค่าด้วยการแจกแจงแบบต่างๆ

ฟังก์ชันหรือรูปแบบในการกำหนด	คำอธิบาย
การกระจายแบบต่อเนื่อง(Continuous distributions)	
Uniform(a,b)	การแจกแจงเอกรูปในช่วง [a,b)
exponential(mean)	การแจกแจงแบบเลขชี้กำลัง โดยการกำหนดค่าเฉลี่ย
normal(mean, stddev)	การแจกแจงปรกติโดยการกำหนดค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐาน
chi_square(k)	การแจกแจงไคกำลังสอง k คือระดับขั้นความเสรี
การกระจายแบบไม่ต่อเนื่อง(Discrete distributions)	
intuniform(a, b)	จำนวนเต็มแบบเอกรูปจาก a..b
binomial(n, p)	การแจกแจงทวินามด้วยค่าตัวแปร n>=0 และ 0<=p<=1
geometric(p)	การแจกแจงเรขาคณิตด้วยค่าตัวแปร 0<=p<=1
negbinomial(n, p)	การแจกแจงทวินามด้วยค่าตัวแปร n>0 และ 0<=p<=1

2.12.1.2 พารามิเตอร์ XML

ในบางครั้งมอดูลต้องการ input ที่มีความซับซ้อน มากกว่าที่พารามิเตอร์มอดูลอย่างง่ายจะสามารถให้ค่าได้ จึงทำให้ต้องใส่พารามิเตอร์ลงยัง file ที่เก็บค่า ภายนอก แล้วให้ตัวมอดูลอ่านค่าและทำการประมวลผลไฟล์โดยทั่วไปจะทำโดยการส่งชื่อไฟล์ไปยังมอดูลในรูปแบบ string พารามิเตอร์

ในทุกวันนี้การใช้ XML ได้เพิ่มขึ้นเป็นอย่างมาก และได้กลายมาเป็นแบบมาตรฐานสำหรับ file ที่ใช้ในการเก็บค่าพารามิเตอร์ ดังนั้นสามารถที่จะทำให้ file ที่ใช้อยู่รูปของ XML ในโปรแกรม OMNeT++ ตั้งแต่ version 3.0 เป็นต้นไปได้มีตัวสนับสนุนสำหรับ XML file

OMNeT++ ห่อ XML ให้อยู่ในรูปของไวยากรณ์ (LibXML, Expat, etc.) อ่าน และ DTD-validates file (ถ้าเอกสาร XML บรรจุ DOCTYPE) เก็บ file (หากได้ทำการอ้างอิงไปยังหลายมอดูลแล้วก็ยังคงทำการโหลดเพียงครั้งเดียว) ปล่อยให้เลือกส่วนของเอกสารผ่าน XPath-Subset notation และแสดงเนื้อหาโดยผ่าน DOM-like object tree

ในรูปแบบกลไกนี้สามารถเข้าถึงผ่าน NED พารามิเตอร์ ในรูปแบบ xml และ ตัวดำเนินการ xmldoc() คุณสามารถระบุ xml-type มอดูล พารามิเตอร์ เพื่อที่จะทำการกำหนด XML file (หรือทำการระบุด้วยองค์ประกอบ (element(tag)) ภายใน XML) ผ่านตัวดำเนินการ xmldoc() เราสามารถกำหนดค่า xml พารามิเตอร์ ทั้งทาง NED และทาง omenetpp.ini

2.12.2 ประตูของมอดูลอย่างง่าย (simple module gate)

Gate คือจุดเชื่อมของมอดูล และจุดเริ่มต้นและจุดสุดท้ายของการเชื่อมต่อระหว่างมอดูล OMNeT++ สนับสนุนการเชื่อมต่อแบบทางเดียว ดังนั้นจึงเป็นประตูทางเข้าและทางออก ข้อความถูกส่งผ่านประตูทางออก และข้อความจะถูกรับเข้าทางประตูทางเข้า

Gate จะถูกระบุจากชื่อของตัวเอง โดยโดยข้อตกลงของการระบุชื่อคือเป็นตัวเล็กทั้งหมด

Gate vector ได้ถูก support ในโปรแกรม OMNeT++ gate vector นั้นจะบรรจุไปด้วยจำนวน จำนวนหนึ่งภายใน gate เดียว

Gate ประกาศโดยการ ชื่อภายในมอดูลในส่วนของ gate: และ gatevector นั้น จะระบุโดยใส่วงเล็บ [ ] เพื่อกำหนดค่าจำนวนภายใน gate โดยองค์ประกอบมีได้ตั้งแต่ 0 ขึ้นไปตัวอย่างเช่น

```
simple NetworkInterface
        parameter : //…
        gates :
                in: fromPort, fromHigherLayer ;
                out: toPort, toHigherLayer ;
endsimple
```

```
simple RoutingUnit
        parameters : //…
        gate :
                in : output [ ] ;
                out : input [ ] ;
endsimple
```

ขนาดของ gate vector จะถูกกำหนดต่อไปเมื่อมอดูลถูกใช้ในการที่จะถูกนำไปใช้ในกรณีที่เป็นมอดูลที่เรียกมอดูลอย่างง่าย ดังนั้นทุกกรณีของมอดูลสามารถที่จะมี vector ที่มีขนาดต่างๆ กัน

2.13 การกำหนดมอดูลแบบประกอบ (Compound module definition)

มอดูลประกอบ คือมอดูลที่ประกอบไปด้วยมอดูลตั้งแต่ 1 ขึ้นไปเข้าด้วยกันจะแต่ล่ะมอดูลว่ามอดูลย่อย โดยจะเป็นมอดูลอย่างง่ายหรือมอดูลประกอบ ที่จะใช้งานเป็นมอดูลย่อยภายในมอดูลประกอบ และส่วนประกอบของมอดูลประกอบนั้นก็เหมือนกับมอดูลย่อยคือมีค่าพารามิเตอร์และประตู เหมือนกัน

จะเป็นการง่ายที่มองว่ามอดูลประกอบเป็นตัวช่วยที่ทำให้ลดความซับซ้อนของการทำงานลง เพื่อที่จะช่วยให้จัดการโครงสร้างได้ง่ายและมีประสิทธิภาพมากขึ้นโดยสามารถที่จะนำไปใช้เป็นแบบจำลองหรือเป็นส่วนในการสร้างมอดูลประกอบอื่นๆ ต่อไป

มอดูลย่อยอาจใช้งานค่าพารามิเตอร์ของมอดูลประกอบจึงอาจต้องสร้างการเชื่อมต่อกันและกันและหรือเชื่อมต่อกับตัวมอดูลประกอบเอง ตัวอย่างเช่น

```
module CompoundModule
        parameters :
                //…
        gates :
                //…
        submodules :
                //…
        connections :
                //…
Endmodule
```

ทุกส่วนของมอดูลถึงจะไม่ได้กำหนดก็สามารถทำงานได้ไม่จำเป็นต้องมีครบทุกส่วน (parameters, gates, submodules, connections)

2.13.1 พารามิเตอร์และ gate ของมอดูลประกอบ

พารามิเตอร์และ gate จะมีการทำงานแบบเดียวกับมอดูลอย่างง่าย โดยตามแบบแผนแล้วพารามิเตอร์ของมอดูลประกอบนั้นจะส่งผ่านไปยังมอดูลย่อยและใช้สำหรับกำหนดค่าเริ่มต้นของพารามิเตอร์

พารามิเตอร์สามารถที่จะถูกนำไปใช้กำหนดโครงสร้างภายในของมอดูลประกอบ โดยจำนวนของมอดูลย่อยและตัวขนาดของ gate vectors ถูกกำหนดได้ด้วยพารามิเตอร์ และยังสามารถกำหนดทางเชื่อมต่อภายในมอดูลประกอบ ตัวอย่าง สามารถที่จะสร้าง มอดูลประกอบชื่อ Router โดยมีค่าตัวแปรของ ports ระบุด้วยพารามิเตอร์ numofPorts

พารามิเตอร์ที่ส่งผลกระทบต่อโครงสร้างภายในของมอดูลประกอบควรที่จะต้องทำการกำหนดค่าเป็นค่าคงที่ (const) เพื่อที่จะสามรถเข้าถึงค่าที่เหมือนกัน อย่างไรก็ตามถ้าค่าพารามิเตอร์ถูกกำหนดให้เป็นค่าสุ่ม อาจทำให้ส่วนหนึ่งได้ค่าที่ต่างๆกันไป ซึ่งจะทำให้เกิดข้อผิดพลาดโดยได้ค่าที่ไม่ได้ต้องการให้เป็น ตัวอย่างเช่น

```
module Router
        parameters :
                packetsPerSecound : numeric,
                bufferSize : numeric,
                numofPorts : const ;
        gates :
                in : inputPort[ ] ;
                out : outputPort[ ] ;
        submodule : //…
        connections : //…
endmodule
```

2.13.2 มอดูลย่อย

ตัวมอดูลย่อยจะถูกกำหนดในส่วนของ submodules: ภายในมอดูลประกอบ ตัวมอดูลย่อยนั้นจะถูกระบุโดยชื่อ และตามหลักทั่วไปแล้วจะตัวอักษรเล็ก

มอดูลย่อยจะมีตัวชนิดของมอดูลจะเป็นได้ทั้งมอดูลอย่างง่ายหรือมอดูลประกอบ แต่ตัวชนิดมอดูลนั้นจะต้องมีตัว NED compiler รู้จักถึงจะทำงานได้ซึ่งจะต้องมีอยู่ใน NED file เดียวกันหรือถูก import เข้ามา

มีความเป็นไปได้ที่จะกำหนด vectors ของมอดูลย่อยและขนาดของ vector อาจมาได้จากค่าพารามิเตอร์

เมื่อทำการกำหนดมอดูลย่อย สามารถที่จะทำการกำหนดค่าพารามิเตอร์ต่างๆ ได้ และถ้ามีความสอดคล้องของชนิดมอดูลมี gate vectors จะสามรถที่จะกำหนดขนาดได้ ตัวอย่างเช่น

```
module CompoundModule
        //…
        submodules :
                submodule1 : ModuleType1
                        parameters :
                                //…
                        gatessizes :
                                //…
                        submodule2 : ModuleType2
                        patameters :
                                //…
                        gatesizes :
                                //…
endmodule
```

Module Vectors

มีความเป็นไปได้ที่จะสร้าง array ของมอดูลย่อย (module vectors) โดยทำการสร้างได้โดยมี การกระทำในวงเล็บ[] หลัง ค่าชื่อชนิดของมอดูล โดยการกระทำสามรถอ้างถึงพารามิเตอร์ของมอดูล และค่าการนับของมอดูลเป็นศูนย์สามรถทำได้ ตัวอย่างเช่น

```
module CompoundModule
        parameters :
                size : const ;
        submodules :
                submod1 : Node[3]
                        //…
                submod2 : Node[size]
```

```
                //…
        submod3 : Node[2*size+1]
                //…
Endmodule
```

2.13.3 การกำหนดค่าพารามิเตอร์ของมอดูลย่อย

ถ้าชนิดมอดูลใช้เป็นมอดูลย่อยที่มีพารามิเตอร์ สามารถที่จะทำการกำหนดค่าให้แกพารามิเตอร์ในส่วนของ parameters: ในส่วนของมอดูลย่อย โดยสามรถที่จะเป็นค่าคงที่ต่างๆ หรือพารามิเตอร์อื่นๆ หรือใช้เป็นการกระทำ

ไม่จำเป็นที่จะต้องทำการกล่าวถึงหรือกำหนดพารามิเตอร์ทุกตัว ตัวที่ไม่ได้รับการกำหนดสามารถที่จะได้ค่าในขณะที่อยู่ระหว่างการทำงาน คล้ายกับการทีได้รัยจากตัว configuration file (omnetpp.ini) หรือหากยังไม่มีการกำหนดอีก ในขณะทำการจำลองก็จะมีให้ทำการใส่ค่าลงไป การทำแบบนี้เพื่อที่จะเป็นการทำให้เกิดความยืดหยุ่นมากขึ้นของตัวโปรแกรม ตัวอย่างเช่น

```
Module CompoundModule
        parameters :
                param1 : numeric,
                param2 : numeric,
                useParam1 : bool ;
        submodules :
                submodule : node
                        parameters :
                                p1 = 10,
                                p2 = param1+param2,
                                p3 = usrParam1==true ? param1 : param2;
                //…
endmodule
```

เมื่อพารามิเตอร์ไม่ได้รับค่าจากไฟล์ NED หรือไฟล์ configuration ผู้ใช้จะต้องกำหนดค่าพารามิเตอร์ในตอนก่อนที่จะเริ่มทำการจำลอง ถ้าวางแผนไว้ว่าจะใช้การตอบสนองจากผู้ใช้จะสามารถระบุข้อความและค่าปกติ

```
Parameters :
        numCPUs = input(10,"Number of processors?"), //default value ,prompt
```

```
processingTime = input(10ms), // promt text
cacheSize = input;
```

2.13.4 การเชื่อมต่อ (Connections)

การเชื่อมต่อเป็นการเชื่อต่อระหว่างตัวมอดูลย่อยที่ตัว gate ของแต่ล่ะมอดูลในส่วนการเชื่อมต่อจะเป็นการที่ gate ของแต่ล่ะมอดูลนั้นเชื่อมต่อกันอย่างไร

การเชื่อมต่อสามารถที่จะเชื่อมต่อกับมอดูลย่อยหรือตัวมอดูลประกอบข้างเคียง นั่นหมายความว่าตัว NED นั้นจะไม่อนุญาตให้ทำการเชื่อมต่อกับหลายระดับของลำดับขั้น และทิศทางการเชื่อมต่อของ gate จะถูกตรวจสอบโดยจะไม่สามารถทำการเชื่อมต่อ gate in กับ in ด้วยกัน หรือ out กับ out ด้วยกันได้

การเชื่อมต่อนั้นสนับสนุนแบบหนึ่งต่อหนึ่งเท่านั้น ดังนั้น gate ที่ไม่ธรรมดาจึ่งอาจถูกใช้ในการเชื่อมต่อแบบ one-to-many และแบบ many-to-one โดยการเชื่อมต่อสามรถทำได้โดยการใช้มอดูลอย่างง่ายที่ทำการใช้ ข้อความร่วมกันหรือรวมการส่งข้อความเข้าด้วยกัน เพราะในที่ใดๆ แบบนี้ fan-in หรือ fan-out ปรากฏอยู่ในแบบจำลอง และส่วนใหญ่จะเกี่ยวข้องกับการประมวลผลจึงทำให้จำเป็นที่จะต้องใช้มอดูลอย่างง่ายในการทำ

การเชื่อมต่อถูกกำหนดภายในมอดูลประกอบ ในส่วนของ conections: โดยแต่ล่ะการเชื่อมต่อจะถูกขึ้นด้วยเครื่องหมาย ; ตัวอย่างเช่น

```
module CompoundModule
        parameters : //…
        gates : //…
        submodules : //…
        connections :
                sode1.output --> node2.input ;
                sode1.input --> node2.output ;
endmodule
```

ตัว gate ที่เป็นตัวตั้งต้นสามารถเป็น gate ที่ทำหน้าที่ส่งออกของมอดูลย่อย หรือเป็นตัวรับข้อมูลของตัวมอดูลประกอบ และตัว gate ปลายทางก็สามารถเป็นตัวส่งออกของมอดูลย่อย หรือเป็นเป็นตัวรับข้อมูลของมอดูลประกอบเช่นเดียวกัน ตัวลูกศรสามารถที่จะชี้จากซ้ายไปขวา หรือจากขวาไปทางซ้าย ได้

สัญลักษณ์ gate++ จะให้สามรถทำการขยาย gate vector ด้วย gate ตัวใหม่ โดยไม่ต้องมีการประกาศขนาดของ vector ล่วงหน้าด้วย gatesizes ในการที่มีวิธีการแบบนี้ทำให้สะดวกเป็นอย่างมากสำหรับหารเชื่อมต่อ node ของระบบเครือข่าย ตัวอย่างเช่น

```
simple Node
        gates :
                in : in[ ] ;
                out : out[ ] ;
endsimple
module smallNet
        submodules :
                node: Node [6] ;
        connections :
                node[0].out++ --> node[1].in++;
                node[0].in++ <-- node[1].out++;
                node[1].out++ --> node[2].in++;
                node[1].in++ <-- node[2].out++;
                node[1].out++ --> node[4].in++;
                node[1].in++ <-- node[4].out++;
                node[3].out++ --> node[4].in++;
                node[3].in++ <-- node[4].out++;
                node[4].out++ --> node[5].in++;
                node[4].in++ <-- node[5].out++;
endmodule
```

การเชื่อมต่อ :

อาจมีค่าคุณลักษณะเช่น (delay, bit error rate หรือ data rate) หรือใช้ตั้งชื่อช่องสัญญาณ

อาจมีการใช้ for-loop ในการสร้าง connection จำนวนมาก

อาจมีการกำหนดทางเลือกได้

แบบการเชื่อมต่อลักษณะต่างๆ

- การเชื่อมต่อแบบเดี่ยว และ ช่องสัญญาณ

ถ้าไม่ได้ทำการกำหนดช่องสัญญาณ การเชื่อมต่อจะไม่มีการล่าช้าของการเดินทางของข้อมูล (propagation delay) ไม่มีการล่าเชื่องเนื่องจากการส่ง (transmission delay) และ bit error

```
node1.outGate --> node2.inGate;
```

การกำหนดช่องสัญญาณทำได้โดยใส่ชื่อของช่องสัญญาณลงระหว่าง gate ทั้งสอง

```
node1.outGate --> Fiber  -->  node2.inGate;
```

ในกรณีนี้ ตัว NED ที่เป็นต้นกำเนิดจะต้องบรรจุด้วยคำจำกัดความของช่องสัญญาณต่างๆ

อีกวิธีคือการกำหนดค่าระหว่าง gate ทั้งสอง

```
node1.outGate --> error 1e-9 delay 0.001  -->  node2.inGate;
```

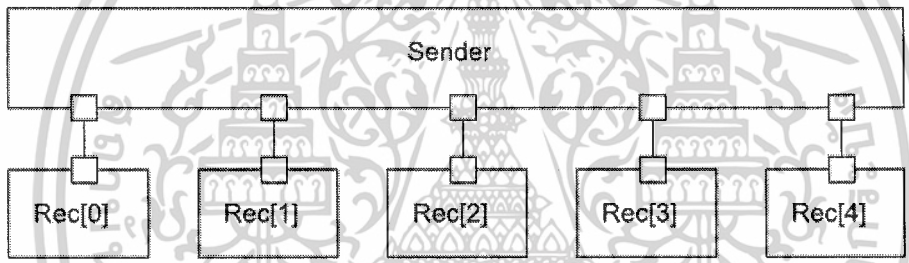
เช่นเดียวกับค่าพารามิเตอร์สามรถที่จะไม่กำหนดและสามรถที่จะเป็นจำนวนเท่าใดก็ได้ตามที่ต้องการ

- การเชื่อมต่อแบบวน (loop connection)

ถ้ามอดูลย่อยหรือ gate vector ถูกใช้งาน มีความเป็นไปได้ที่จะถูกสร้างมากกว่า 1 connection ด้วยการประกาศสั้นๆ จึงจัดอยู่ประเภทการเชื่อมต่อแบบหลายหรือวน

โดยการสร้างการเชื่อมต่อประเภทนี้จะใช้คำสั่ง for ในการช่วงสร้าง

```
for i=0..4 do
        node1.outGate[i]-->  node2[i].inGate;
endfor;
```



รูปที่ 2.8 ผลที่ได้จากการสร้างการเชื่อมต่อแบบวน

สามารถที่จะทำการวนให้ซับซ้อนมากขึ้นได้ด้วยการเพิ่มตัวแปรใน loop ได้เช่น

```
for i=0..4, j=0…4do
        //…
endfor ;
for i=0..4, j=i+1..4do
        //…
endfor ;
```

- การเชื่อมต่อแบบมีเงื่อนไข

การสร้างการเชื่อมต่อสามรถที่จะเพิ่มเงื่อนไขลงไปได้โดยใช้คำสั่ง if

```
for i = 0..n do
        node1.outGate[i]-->  node2[i].inGate if i % 2 = 0;
endfor ;
```

เงื่อนไข if จะทำการเรียบเทียบกับแต่ละการเชื่อมต่อ ว่าสร้างหรือไม่สร้าง เช่นตัวอย่างจะเปรียบเทียบว่าถ้า ค่า i เป็นเลขคู่จะทำการสร้างการเชื่อมต่อ และเงื่อนไขสามารถที่จะเป็นค่าสุ่มได้

การใช้(The nocheck modifier)

โดยปกติ NED จะต้องการให้ทุก gate มีการเชื่อมต่อ แต่การตรวจสอบนี้อาจทำให้เกิดความไม่สะดวกในบางครั้ง จึงสามารถปิดได้โดยการใช้ nocheck modifier

ตัวอย่างจะสร้างกราฟย่อยแบบสุ่มของกราฟเต็มรูปแบบ

```
module RandomConnections
        parameters : //…
        gates : //…
        submodules : //…
        connections nocheck:
                for i = 0..n-i, j = 0..n-i do
                        Node[i].out[j] --> node[i].in[j] if uniform (0,1) < 0.3 ;
                endfor ;
  endmodule
```

เมื่อใช้ nocehck ตัวมอดูลอย่างง่ายจะรับผิดชอบในการไม่ส่งข้อความไปยัง gate ที่ไม่ได้มีการสร้างการเชื่อมต่อไว้

2.14 การกำหนดระบบเครือข่าย (Network Definition)

การประกาศมอดูลไม่ว่าจะเป็นแบบอย่างง่ายหรือแบบประกอบก็เป็นการกำหนดชนิดของมอดูลเท่านั้น การสร้างแบบจำลองจริงๆ ที่สามารถทำงานได้จำเป็นที่จะต้องมีการกำหนดค่าระบบเครือข่าย (network definition)

การกำหนดระบบเครือข่ายเป็นการประกาศแบบจำลองเป็นตัว instance ของมอดูลที่ถูกกำหนดไปแล้วซึ่งในการทำส่วนนี้ ส่วนใหญ่จะเป็นการใช้งานมอดูลแบบประกอบ แม้ว่ามันสามารถที่จะมี มอดูลอย่างง่ายอยู่ภายในและทำตัวเองเป็นอย่าง ระบบเครือข่าย

อาจมีบ้างส่วนของระบบเครือข่ายที่กำหนดอยู่ใน NED เพียง file เดียวหรืออยู่ในหลายๆ file ก็ได้ โปรแกรมจำลองใช้ NED file ในการ run ทั้งหมด หรืออาจเลือกที่ต้องการใน configuration file

ตัวไวยากรณ์ที่ใช้ในการกำหนดระบบเครือข่ายส่วนใหญ่จะคล้ายกับการประกาศมอดูลย่อย

```
Network wirelessLAN : WirelessLAN
        parameters :
                numUsers = 10,
                httpTraffic = true,
                ftpTraffic = true,
                distanceFormHub = truncnormal (100,60) ;
endnetwork
```

WirelessLAN เป็นชื่อของมอดูลประกอบที่เราได้ทำการกำหนดไปแล้วซึ่งประกอบไปด้วยมอดูลประกอบอื่นๆ อีก เช่น WirelessHost, WirelessHub, etc

โดยธรรมชาติแล้วชนิดมอดูลที่ไม่มี gate สามารถใช้ในการกำหนดระบบเครือข่ายได้เหมือนกับในตัวมอดูลย่อยไม่จำเป็นที่จะต้องทำการกำหนดค่าทุกพารามิเตอร์ ค่าสามรถได้มาจากตัว configuration file หรือได้มากจากการรับข้อมูลเมื่อตอนเริ่มการจำลอง

2.15 การกระทำ (Expression)

2.15.1 ค่าคงที่ (Constant)

- ค่าคงที่จะมีทั้งหมด 2 แบบ คือตัวเลขและสตริง
- ค่าคงที่แบบตัวเลขยอมรับในรูปของเลขฐานสิบและสัญลักษณ์ทางวิทยาศาสตร์
- ค่าคงที่แบบสตริงใช้ double quotes (อ้างอิง)
- ค่าคงที่แบบเวลา ในทุกครั้งที่ใช้ตัวเลขในการกำหนดเวลาจะหมายความว่าหน่วยเป็นวินาที แต่สามรถที่จะกำหนดได้ว่าใช้หน่วยอื่นๆ ได้เช่น

```
Parameters:
        propagationDelay = 560 ms, // 0.560 sec
        connectionTimeout = 6 m 30 s 500 ms, // 390.5 s
        recoveryIntv1 = 0.5 h; // 30 min
```

หน่วยของเวลาที่ใช้ได้มีดังนี้

Ns	= nanoseconds	m	= minutes (60 s)
Us	= microseconds	h	= hours (3600 s)
Ms	= milliseconds	d	= days (86400 s)
S	= seconds		

2.15.2 ตัวกระทำ (Operators)

ตัวกระทำในภาษา NED นั่นจะคล้ายคลึงกับตัวกระทำในภาษา C++ ความแต่ตากที่มีคือเครื่องหมาย ^ ถูกใช้สำหรับการยกกำลัง (ใน C จะถูกใช้ในการ bit wise XOR)## ถูกใช้สำหรับ XOR (อย่างเดียวกับ != ในความหมายทางตรรกะ) และ # ใช้ในการ bit wise XORตัวกระทำ bit wise ที่เหนือกว่า (&, |, #) ได้ถูกทำให้เป็นความสัมพันธ์ที่แน่นมากว่าเดิม ทำให้มีความสะดวกในการใช้มากกว่าใน C/C++ ทุกค่าถูกทำให้อยู่ในรูปของ doubles สำหรับ bit wise operator ค่า doubles จะถูกแปลงให้เป็น unsigned long โดยใช้ C/C++ การแปลงภายใน(type cast) ตัว operator ทำตามหน้าที่แล้วก็จะนำผลมาแปลงค่ากลับเป็น double ในทำนองเดียวกันตัว &&, || และ ## ก็จะมีการทำงานที่คล้ายกันคือนำมาแปลงเป็น bool แล้วทำงานแล้วแปลงค่ากลับมาเป็น double ส่วนตัว modulus (%) จะทำการแปลงเป็นค่า long ก่อน

ตารางที่ 2.2 ตารางสัญลักษณ์ของตัวกระทำ (Operators)

Operator	Meaning
-, !, ~	unary minus, negation, bitwise complement
^	power-of
*, /, %	multiply, divide, modulus
+, -	add, subtract
«, »	bitwise shift
&, |, #	bitwise and, or, xor
== != >, >= <, <=	equal not equal greater, greater or equal less, less or equal
&&, ||, ##	logical operators and, or, xor
?:	the C/C++ "inline if"

บทที่ 3
การสร้างและแก้ไข OMNeT++ ให้สอดคล้องกับปัจจัยที่มีผลกระทบต่อการส่งข้อมูลแบบไร้สาย

3.1 การสร้างแบบจำลอง

ในการสร้างแบบจำลองใน OMNeT++ นั้นจำเป็นต้องมีไฟล์สำคัญอยู่ 2 ไฟล์คือ ไฟล์ Omnetpp.ini และไฟล์ที่มีนามสกุล ned ซึ่งแต่ละไฟล์จะมีการสร้างและการกำหนดคุณสมบัติได้ดังนี้

3.2 การสร้างและการกำหนดคุณสมบัติของไฟล์ omnetpp.ini

ภายในไฟล์ omnetpp.ini นั้นจะต้องมีการแบ่งกำหนดค่าพารามิเตอร์ออกเป็นส่วนต่างๆ ดังนี้

3.2.1 การแทรกไฟล์

OMNeT++ จะรองรับการแทรกไฟล์ ini อื่นโดยใช้คำว่า include เป็นคีย์เวิร์ดในการเรียกใช้ ดังตัวอย่างทางด้านล่าง

```
# omnetpp.ini
. . .
Include parameters.ini
Include per-run-pars.ini
. . .
```

เราสามารถแทรกไฟล์จากไดเร็กทอรี่ (directory) อื่นๆได้โดยการระบุที่อยู่ของไฟล์ (pathname) นั้นด้วย การแทรกไฟล์อื่นๆเข้ามานั้นจะช่วยในการดึงค่าคุณสมบัติมาเพิ่มให้กับแบบจำลองของเราได้

3.2.2 ส่วนของการทำงาน (Sections)

ภายในไฟล์ omnetpp.ini จะแบ่งส่วนของการทำงานเอาไว้เป็นหัวข้อซึ่งแต่ละหัวข้อมีหน้าที่การทำงานดังต่อไปนี้

ตารางที่ 3.1 แสดงส่วนต่างๆภายในไฟล์ omnetpp.ini

Section	Description
[General]	คือส่วนที่เก็บการตั้งค่าทั่วไปที่จะประยุกต์มาเป็นแบบจำลองสามารถทำงานและส่วนติดต่อกับผู้ใช้ทั้งหมด
[Run 1], [Run 2], ...	คือส่วนที่ใส่หัวข้อหรือเงื่อนไขให้เลือกในการเริ่มต้นการทำงาน
[Cmdenv]	คือส่วนที่เก็บการตั้งค่าของ Cmdenv โดยเฉพาะ
[Tkenv]	คือส่วนที่เก็บการตั้งค่าของ Tkenv โดยเฉพาะ
[Parameters]	คือส่วนที่เก็บค่าสำหรับพารามิเตอร์ของมอดูลที่ไม่ต้องใส่ค่าหรืออินพุตใดๆในไฟล์ NED
[OutVectors]	คือการกำหนดส่วนที่ใช้ในการบันทึกค่าผลลัพธ์ที่ได้ออกมาเป็นเวคเตอร์

ส่วนเสริมหลักๆภายในส่วน [General] จะมีดังนี้

- network เป็นส่วนเลือกโมเดลที่จะใช้ในการติดตั้งและทำงาน
- sim-time-limit และ cpu-time-limit คือความยาวนานในการทดสอบแบบจำลอง สามารถระบุหน่วยของเวลาได้ทั้ง มิลลิวินาที, วินาที, นาที, ชั่วโมง ฯลฯ
- output-vector-file, output-scalar-file และ snapshot-file เป็นการระบุชื่อของไฟล์ที่ใช้เก็บผลลัพธ์ที่ได้จากการทดลอง
- Preload-ned-files คือการดึงข้อมูลจากไฟล์ NED

3.2.3 Dynamic NED loading

Dynamic NED loading หมายถึงเมื่อโปรแกรมสร้างแบบจำลองเริ่มทำงานจะทำการโหลดไฟล์ NED มาโดยอัตโนมัติ ทำให้เกิดความยืดหยุ่นและสามารถลดเวลาในการพัฒนาแบบจำลองได้ คำสั่งที่ใช้ในการดึงไฟล์ NED ก็คือ preload-ned-files ซึ่งเป็นหนึ่งในส่วนเสริมของส่วน [General] ของไฟล์ omnetpp.ini ซึ่งในส่วนเสริมนี้ควรจะใส่ชื่อของไฟล์ NED ที่ต้องการจะโหลดเมื่อโปรแกรมเริ่มต้นทำงาน ดังตัวอย่าง

```
[General]
Preload-ned-files = host.ned  router.ned  networks/testnetwork1.ned
```

ถ้าต้องการโหลดไฟล์ NED ทีละหลายๆไฟล์ภายในโฟลเดอร์สามารถโหลดได้โดยใช้เครื่องหมาย * ดังตัวอย่าง

```
[General]
preload-ned-files = *.ned networks/ *.ned
```

เราสามารถโหลดใช้ไฟล์แบบบัญชีรายชื่อในการโหลดไฟล์ NED ซึ่งภายในบัญชีรายชื่อนั้นจะเป็นที่รวมไฟล์ NED เอาไว้อยู่หลายๆไฟล์ เราสามารถโหลดได้โดยใช้เครื่องหมาย @ ดังตัวอย่าง

```
[General]
Preload-ned-files = *.ned @../nedfiles.lst
```

3.3 การตั้งค่าพารามิเตอร์ให้กับมอดูลภายในไฟล์ omnetpp.ini

แบบจำลองจะได้รับอินพุตจากค่าพารามิเตอร์ของมอดูลที่สามารถให้ค่าได้โดยการเขียนลงในไฟล์ NED หรือในไฟล์ omnetpp.ini แต่ถ้าเรากำหนดค่าพารามิเตอร์ภายในไฟล์ NED แล้วจะไม่สามารถเขียนซ้ำลงไปในไฟล์ omnetpp.ini ได้อีก ทำให้ง่ายและมีความยืดหยุ่นในการแก้ไขหรือจัดการค่าพารามิเตอร์ต่างๆของมอดูลภายในไฟล์ omnetpp.ini เพียงไฟล์เดียว

ตัวอย่างของไฟล์ omnetpp.ini ที่มีการตั้งค่าพารามิเตอร์ต่างๆ

```
[Parameters]
net.numHosts = 15
net.server.transactionsPerSecond = 100
```

3.3.1 การทำงานแบบเฉพาะและส่วนการทำงานแบบทั่วไป

ค่าพารามิเตอร์ของแต่ละมอดูลสามารถกำหนดได้ในไฟล์ omnetpp.ini ในส่วนของ [Parameters] หรือ ส่วนของการทำงานแบบเฉพาะเช่น [Run 1], [Run 2] ฯลฯ ซึ่งการทำงานแบบเฉพาะสามารถกำหนดค่าใหม่แทนค่าที่เคยกำหนดอยู่ในส่วน [Parameters] ได้ดังตัวอย่าง (ข้างหลังเครื่องหมาย # คือคำอธิบาย)

```
[Parameters]
net.numHosts = 15
net.server.transactionsPerSecond = 100
[Run 1]
description="general settings"
# ใช้การตั้งค่าจากส่วน [Parameters]
[Run 2]
description="higher transaction rate"
net.server.transactionsPerSecond = 150 # เกิดการเขียนค่าทับในส่วน [Parameters]
 [Run 3]
description="more hosts and higher transaction rate"
```

```
# เขียนทับทั้งหมดภายในส่วน [Parameters]
net.numHosts = 20
net.server.transactionsPerSecond = 150
```

3.3.2 การประยุกต์ค่าเริ่มต้น

การประยุกต์ใช้ค่าเริ่มต้นนั้นสามารถทำได้โดยการเพิ่มคำว่า use-default = yes ไว้หลังชื่อพารามิเตอร์ที่ต้องการกำหนดค่าเริ่มต้นให้ ดังตัวอย่างเช่น

```
[Parameters]
**.hostA.ip.ttl = 5
**.ip.ttl.use-default = yes
```

จากตัวอย่างข้างต้นหมายถึงการกำหนดค่า ttl (time to live) ให้มีค่าเท่ากับ 5 และใช้ค่านี้เป็นค่าเริ่มต้นของทุกๆตัว และถ้าต้องการตั้งค่าทั้งหมดในไฟล์ NED ให้ใช้ค่าเริ่มต้นเราสามารถทำได้โดยเขียนในไฟล์ omnetpp.ini ตามตัวอย่างทางด้านล่าง

```
[Parameters]
**.use-default = yes
```

3.4 การกำหนดค่าของผลลัพธ์แบบเวคเตอร์

ใน OMNeT++ เราสามารถความคุมการเก็บบันทึกของข้อมูลลงในหน่วยความจำของเครื่องคอมพิวเตอร์เราได้ และสามารถเลือกได้ว่าจะเก็บหรือไม่เก็บข้อมูลเหล่านั้น ซึ่งการกำหนดค่าของผลลัพธ์แบบเวคเตอร์คือการให้ค่าในส่วน [Out-Vectors] ภายในไฟล์ omnetpp.ini หรือกำหนดในทำงานแบบเฉพาะในส่วน [Run 1], [Run 2] ฯลฯ ซึ่งค่าเริ่มต้นนั้นจะเก็บค่าของผลลัพธ์แบบเวคเตอร์โดยอัตโนมัติ

ผลลัพธ์แบบเวคเตอร์สามารถกำหนดค่าได้ด้วยคำสั่งดังต่อไปนี้

```
module-pathname.objectname.enabled = yes/no
module-pathname.objectname.interval = start..stop
module-pathname.objectname.interval = ..stop
module-pathname.objectname.interval = start..
```

3.5 Cmdenv: ส่วนติดต่อกับผู้ใช้แบบ command-line

ส่วนติดต่อกับผู้ใช้แบบ command-line เป็นส่วนที่มีขนาดเล็ก ใช้งานได้รวดเร็วและสามารถนำไปใช้ได้บนทุกแพลตฟอร์ม ซึ่ง Cmdenv จะใช้ทำการจำลองระบบที่มีการอธิบายไว้ในไฟล์ที่เรา

ทำการกำหนดค่าเอาไว้แล้ว ถ้ามีการจำลองใดหยุดทำงานเพราะมีข้อความที่มีข้อผิดพลาด ข้อความต่อไปก็ยังคงกระทำการต่อ การดำเนินงานการกระทำการสามารถส่งผ่านไปยังอาร์กิวเมนต์ของ command-line หรือส่งไปในไฟล์ omnetpp.ini

3.5.1 ตัวเลือกของ Cmdenv ภายในไฟล์ omnetpp.ini

Cmdenv สามารถเลือกทำงานได้ 2 แบบคือ Normal (non-express) mode และ Express mode โดยเราสามารถเลือกโหมดได้โดยกำหนดภายในส่วน express-mode ในไฟล์ omnetpp.ini

- Normal (non-express) mode คือโหมดสำหรับแก้ไขข้อผิดพลาด รายละเอียดของข้อมูลผลลัพธ์จะถูกเขียนในรูปแบบ สัญลักษณ์เหตุการณ์ ผลลัพธ์ของมอดูล ฯลฯ
- Express mode สามารถใช้สำหรับทำการจำลองในระยะเวลานานๆ จะมีการแสดงผลแค่ส่วนของความก้าวหน้าของการสร้างแบบจำลองเท่านั้น

รายชื่อของตัวเลือกภายในส่วนของ [Cmdenv] ของไฟล์ omnetpp.ini มีดังต่อไปนี้

ตารางที่ 3.2 แสดงตัวเลือกภายของส่วน [Cmdenv]

รายชื่อและค่าเริ่มต้น	คำอธิบาย
[Cmdenv]	
express-mode=yes/no (default: no)	เลือก normal mode หรือ express mode
module-messages=yes/no (default:yes)	เฉพาะ normal mode : สำหรับพิมพ์หรือไม่พิมพ์เหตุการณ์ออกมา
event-banners=yes/no (default: yes)	เฉพาะ normal mode : สำหรับพิมพ์หรือไม่พิมพ์สัญลักษณ์ของเหตุการณ์ขึ้นมา
status-frequency=<integer> (de-fault: 50000)	เฉพาะ express mode : สำหรับพิมพ์ค่าสถานะที่อัพเดตทุกๆระยะเวลาที่กำหนด
performance-display=yes/no (de-fault: yes)	เฉพาะ express mode : สำหรับพิมพ์รายละเอียดประสิทธิภาพของระบบ (ดูคำอธิบายเพิ่มเติมที่หัวข้อ 3.4.2)

3.5.2 การแปล Cmdenv ที่ส่งออกมา

เมื่อการจำลองดำเนินงานในแบบ express mode ด้วยการกำหนดรายละเอียด performance-display = yes ตัว Cmdenv จะส่งสถานะที่เกี่ยวกับความก้าวหน้าของการจำลองออกมา 3 บรรทัด ซึ่งผลที่ส่งออกมาจะมีลักษณะดังนี้

```
...
** Event #250000 T=123.74354 ( 2m 3s) Elapsed: 0m 12s
Speed: ev/sec=19731.6 simsec/sec=9.80713 ev/simsec=2011.97
Messages: created: 55532 present: 6553 in FES: 8
** Event #300000 T=148.55496 ( 2m 28s) Elapsed: 0m 15s
Speed: ev/sec=19584.8 simsec/sec=9.64698 ev/simsec=2030.15
Messages: created: 66605 present: 7815 in FES: 7
...
```

บรรทัดแรกของการแสดงผล (เริ่มจาก **) จะบ่งบอกถึง

- ระบบได้ประมวลผลเป็นเหตุการณ์ที่เท่าไหร่
- เวลาปัจจุบันของการจำลอง
- เวลาที่ล่วงผ่านตั้งแต่เริ่มดำเนินการจำลอง

บรรทัดที่ 2 จะแสดงผลข้อมูลที่เกี่ยวกับประสิทธิภาพของการจำลอง

- ev/sec จะบ่งบอกถึงประสิทธิภาพว่ามีจำนวนเหตุการณ์เกิดขึ้นเท่าไหร่ภายใน 1 วินาที
- simsec/sec แสดงถึง ความเร็วในการจำลอง เป็นการเปรียบเทียบระหว่างความก้าวหน้าของการจำลองเทียบกับเวลาจริง ดังนั้นค่าเหล่านี้จะขึ้นอยู่กับหลายๆสิ่งเช่น ประสิทธิภาพของเครื่องคอมพิวเตอร์ที่ใช้ในการจำลอง, ขนาดของแบบจำลอง ฯลฯ
- ev/simsec คือความหนาแน่นของเหตุการณ์ที่เกิด เป็นการเปรียบเทียบระหว่างจำนวนเหตุการณ์ที่เกิดขึ้นกับระยะเวลาในการจำลอง

บรรทัดที่ 3 จะแสดงถึงจำนวนของข้อความ ซึ่งเป็นส่วนที่สำคัญเพราะสามารถบ่งบอกถึงความสมบูรณ์ของแบบจำลอง

- Created: แสดงถึงจำนวนของข้อความที่สร้างขึ้นตั้งแต่เริ่มดำเนินการจำลอง แต่ไม่ได้หมายถึงข้อความที่มีอยู่จริงทั้งหมด เพราะบางข้อความอาจจะถูกลบไป
- Present: แสดงถึงจำนวนของข้อความที่นำเสนออยู่ ณ เวลานั้นๆภายในแบบจำลอง นั่นก็คือจำนวนข้อความที่ถูกสร้าง (ข้อความจากการ Created) หักลบด้วยจำนวนข้อความที่ถูกลบไป ซึ่งข้อความในปัจจุบันจะรวมถึงข้อความใน FES ด้วย
- In FES: แสดงถึงจำนวนข้อความปัจจุบันที่ถูกกำหนดไว้ในเป็นกลุ่มของเหตุการณ์ที่จะเกิดขึ้นในอนาคต

3.6 Tkenv: ส่วนต่อประสานกราฟิกกับผู้ใช้

Tkenv คือส่วนต่อประสานกราฟิกกับผู้ใช้ ที่รองรับการโต้ตอบของการจำลอง, การติดตามผลและการแก้จุดบกพร่อง ซึ่งจะแสดงรายละเอียดรูปภาพสถานะของการจำลองที่จุดต่างๆของการกระทำการและติดตามผลที่เกิดขึ้นภายในเครือข่าย ซึ่งจุดเด่นที่สำคัญของ Tkenv คือ

- ภาพเคลื่อนไหวขณะส่งข้อความ
- การแสดงผลแบบกราฟิกของผลทางสถิติเช่นฮิสโทแกรม และค่าเวกเตอร์ที่ออกมาระหว่างการจำลอง
- สามารถแยกหน้าต่างการแสดงผลที่ออกมากของแต่ละมอดูล
- สามารถดูทีละเหตุการณ์ที่เกิดหรือเร่งการจำลองได้
- สามารถเริ่มการจำลองใหม่ได้
- สามารถหยุดดูรายละเอียดที่รายงานออกมาได้เป็นช่วงๆ เช่นรายละเอียดของมอดูล, ค่าตัวแปร ฯลฯ

Tkenv ทำให้เราสามารถดูผลของการจำลองได้ทางเวกเตอร์ที่ออกมา ระหว่างการจำลอง ซึ่งผลลัพธ์นั้นสามารถแสดงออกมาเป็นฮิสโทแกรมและแผนภูมิที่อ้างอิงตามระยะเวลา (time-series diagrams)

3.6.1 การตั้งค่า Tkenv ภายในไฟล์ omnetpp.ini

Tkenv ยอมให้มีการตั้งค่าในส่วน [Tkenv] ของไฟล์ omnetpp.ini

ตารางที่ 3.3 แสดงตัวเลือกภายของส่วน [Tkenv]

รายชื่อและค่าเริ่มต้น	คำอธิบาย
[Tkenv]	
use-mainwindow = yes	เปิด/ปิดการเขียนเหตุการณ์ออกไปยังหน้าต่างหลัก Tkenv
print-banners = yes	เปิด/ปิด การพิมพ์แถบประกาศของแต่ละเหตุการณ์
breakpoints-enabled = yes	ระบุว่าการจำลองควรจะหยุดเมื่อมีการเรียก breakpoint()
update-freq-fast = 10	จำนวนของเหตุการณ์ที่กระทำระหว่าง 2 หน้าจอแสดงผลที่มีการอัพเดตเมื่ออยู่ในการกระทำการแบบ Fast

ตารางที่ 3.3 (ต่อ) แสดงตัวเลือกภายของส่วน [Tkenv]

รายชื่อและค่าเริ่มต้น	คำอธิบาย
update-freq-express = 500	จำนวนของเหตุการณ์ที่กระทำระหว่าง 2 หน้าจอแสดงผลที่มีการอัพเดตเมื่ออยู่ในการกระทำการแบบ Express
animation-delay = 0.3s	ค่าหน่วงเวลาระหว่างแต่ละขั้นตอนการทำงาน
animation-enabled = yes	เปิด/ปิดข้อความบนภาพเคลื่อนไหว
animation-msgnames = yes	เปิด/ปิดจอแสดงผลชื่อของข้อความระหว่างเวลาที่ข้อความส่งไปมาแบบภาพเคลื่อนไหว
animation-msgcolors = yes	เปิด/ปิดการใช้สีที่แตกต่างกันสำหรับข้อความ
animation-speed = 1.0	ระบุความเร็วของข้อความที่ส่งไปมาแบบภาพเคลื่อนไหว
methodcalls-delay =	ตั้งค่าหนวงเวลาหลังจากวิธีเรียกภาพเคลื่อนไหว

3.6.2 การใช้สภาพแวดล้อมแบบกราฟิก

Tkenv มีรูปแบบในการดำเนินการจำลองดังนี้

รูปที่ 3.1 ปุ่มเลือกรูปแบบการดำเนินการจำลอง

- Step mode สามารถกระทำการจำลองแบบเหตุการณ์ต่อเหตุการณ์ได้
- Run mode เป็นการดำเนินการจำลองไปเรื่อยๆ จะมีข้อความภาพเคลื่อนไหวทำงานและในแต่ละเหตุการณ์ที่เกิดขึ้นจะมีหน้าต่างอัพเดตขึ้นมา เราสามารถหยุดทำการจำลองได้ด้วยการกดปุ่ม Stop บนแถบเครื่องมือ
- Fast mode ในโหมดนี้จะไม่มีการแสดงภาพเคลื่อนไหวแต่จะแสดงข้อความออกมาทางหน้าต่างทุกๆ 10 เหตุการณ์ที่เกิดขึ้น
- Express mode ความเร็วในการดำเนินการจะเท่ากับที่เราตั้งค่าไว้ในส่วนของ [Cmdenv] จะไม่มีการบันทึกผลที่ได้จากมอดูล เราสามารถโต้ตอบกับการจำลองได้แค่ช่วงครบรอบการทำงาน (มีการตั้งค่าเริ่มต้นไว้เมื่อครบ 1000 เหตุการณ์) ด้วยเหตุนี้ทำให้ใช้เวลาในการดำเนินงานน้อย

3.7 ปัจจัยที่มีผลต่อการส่งข้อมูลไร้สาย

การติดต่อสื่อสารแบบไร้สายนั้นจะใช้คลื่นวิทยุเป็นสื่อกลางในการรับส่งข้อมูล ซึ่งคุณสมบัติของคลื่นวิทยุนั้นจะมีคุณสมบัติพื้นฐานคือจากการสะท้อน, หักเหและเลี้ยวเบน ซึ่งเป็นคุณสมบัติที่มีผลกระทบต่อการกระจายคลื่นวิทยุโดยมี 2 ปรากฏการณ์หลักๆที่เกี่ยวข้องคือ

3.7.1 การจางหายของสัญญาณแบบระยะสั้น

การจางหายของสัญญาณแบบระยะสั้นจะเกิดขึ้นจาก 2 สาเหตุย่อยๆคือ

3.7.1.1 การจางหายของสัญญาณระหว่างทาง (Path loss)

การจางหายของสัญญาณระหว่างทางเป็นการสูญหายของสัญญาณระหว่างทางโดยขึ้นอยู่กับระยะทางเป็นหลัก

3.7.1.2 การจางหายแบบมัลติพาท (Multi Path)

สามารถใช้หลักการทางสถิติและความน่าจะเป็นเพื่ออธิบายเรื่องของแบบจำลองของการจางหาย (Fading models) ซึ่งเป็นตัวแปรที่สามารถกำหนดโดยใช้ผลลัพธ์ที่ได้จากการวัดผลทดลองของการกระจายคลื่นวิทยุ

เราสามารถจำลองช่องสัญญาณ (Channel models) โดยใช้สาเหตุเหล่านี้ไปใช้ในการออกแบบและประเมินค่าเบื้องต้นของระบบการสื่อสารแบบไร้สาย ในเมื่อเราคาดหวังประสิทธิภาพและความน่าเชื่อถือของระบบไร้สายให้กลายเป็นที่ต้องการมากขึ้น ดังนั้นความสำคัญของแบบจำลองช่องทางการสื่อสารที่ความแม่นยำในการออกแบบ การประเมินผลและการทำงานก็ยังต้องพัฒนากันต่อไป

3.7.2 การจางหายของสัญญาณแบบระยะยาว

การจางหายแบบระยะยาวเกิดขึ้นจากการที่สัญญาณถูกบดบังโดยสิ่งต่างๆเช่น อาคาร ตึก ต้นไม้ ผู้คน ภูเขา ฯลฯ โดยจะพบในการส่งสัญญาณแบบระยะไกลเช่น สัญญาณโทรศัพท์มือถือเป็นต้น แต่ในระบบเครือข่ายไร้สายนั้น คลื่นวิทยุโดยส่วนใหญ่จะได้รับผลกระทบจากการจางหายแบบระยะสั้นเท่านั้นเนื่องจากระบบเครือข่ายไร้สายจะส่งสัญญาณกันภายในพื้นที่ที่มีขนาดไม่กว้างมาก จึงได้รับผลกระทบจากการจางหายแบบระยะยาวน้อยมาก หรืออาจจะไม่มีผลเลยในบางกรณี ดังนั้นเราจะสนใจในเรื่องของการจางหายแบบระยะสั้นเท่านั้น

เนื่องจากมีการจางหายไปตามสภาพแวดล้อม (Fading environment) และการกระจายทางสถิติในแบบ ต่างๆนั้นเป็นจุดประสงค์หลักสำหรับการสร้างแบบจำลองของช่องทางการสื่อสารแบบไร้สายภายใต้เงื่อนไขการจางหายแบบระยะสั้น ซึ่งแบบจำลองการจางหายแบบระยะสั้นที่สำคัญได้แก่การกระจายแบบ Rayleigh, Weibull, Rice และ Nakagami-m

การจางหายแบบหลายเส้นทาง (Multipath Fading) คือการทำลายการรวมกันของการสุ่มความหน่วง, การสะท้อน, การกระจัดกระจาย และการเลี้ยวเบนของสัญญาณ ซึ่งประเภทของการจางหายนี้เกิดขึ้นค่อนข้างเร็วและเป็นต้นเหตุของการเปลี่ยนแปลงของสัญญาณในระยะสั้น ขึ้นอยู่กับสภาพแวดล้อมของการกระจายคลื่นวิทยุ ซึ่งในแต่ละแบบจำลองจะอธิบายถึงพฤติกรรมทางสถิติของการจางหายแบบหลายเส้นทาง โดยเราจะพูดถึงการจางหายแบบ Nakagami-m การมอดูเลตโดยใช้สัญญาณเคออติกถูกออกแบบมาเพื่อให้ใช้ในระบบเครือข่ายไร้สาย ซึ่งเป็นระบบที่มีการจางหายของช่องสัญญาณแบบ Nakagami-m

3.8 การจางหายแบบ Nakagami-m

แบบจำลองฟังก์ชั่นความหนาแน่นของความน่าจะเป็น (PDF) นี้เป็นการใช้จำลองบนช่องสัญญาณการจางหายแบบ Nakagami-m อยู่ในคุณสมบัติสำคัญของการกระจายแบบ chi-square กำหนดโดยสมการดังนี้

$$p_\alpha(\alpha) = \frac{2m^m \alpha^{2m-1}}{\Omega^m \Gamma(m)} \exp\left(-\frac{m\alpha^2}{\Omega}\right), \qquad \alpha \geq 0 \tag{3.1}$$

เมื่อ m เป็นค่าตัวแปรเสริมเฟดดิง

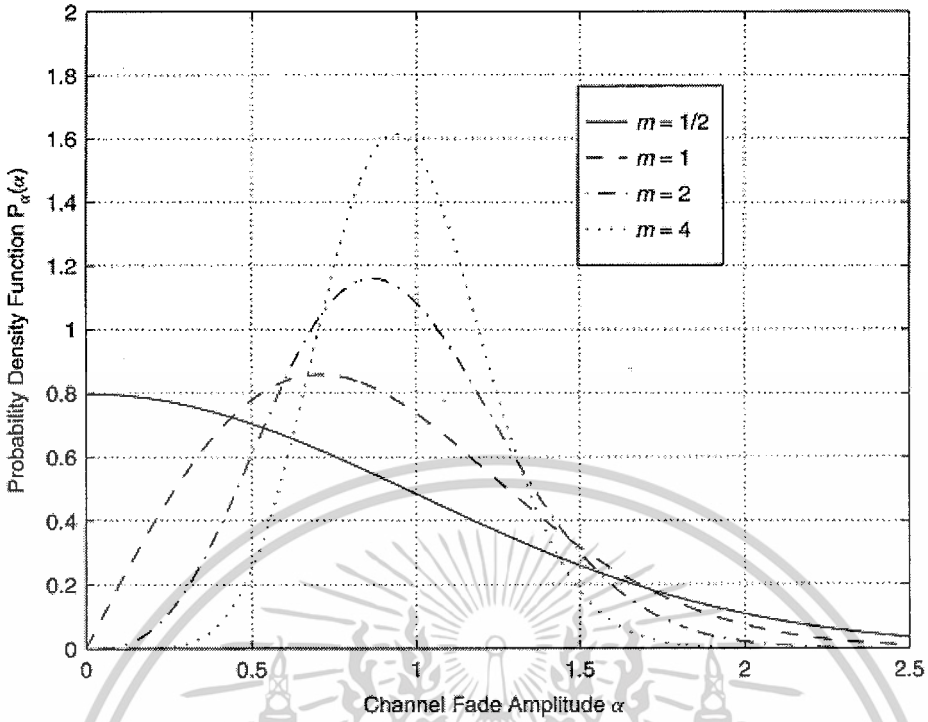
Ω เป็นโมเมนต์อันดับที่สอง (Second moment)

α เป็นค่าอัตราการลดทอนทางแอมปลิจูด ซึ่งเป็นตัวแปรสุ่มไม่เจาะจง

Γ(m) เป็นค่าของฟังก์ชันแกมมา (Gamma function)

โดย m คือค่าพารามิเตอร์ของการจางหายแบบ Nakagami-m ซึ่งมีค่าอยู่ในช่วง ½ ถึง ∞ ดังรูป 3.2 ซึ่งแสดงถึงฟังก์ชั่นความน่าจะเป็นของความหนาแน่น (PDF) แบบ Nakagami-m โดยมีค่า Ω = 1 กับค่าตัวแปร m หลายๆค่า และสามารถแสดงค่าอัตราส่วนสัญญาณต่อสัญญาณรบกวน (SNR) ต่อ 1 สัญลักษณ์ซึ่งค่า γ เป็นค่าของการแจกแจงขึ้นอยู่กับการแจกแจงของแกมมาซึ่งแสดงอยู่ในสมการดังนี้

$$p_\gamma(\gamma) = \frac{m^m \gamma^{m-1}}{\overline{\gamma}^m \Gamma(m)} \exp\left(-\frac{m\gamma}{\overline{\gamma}}\right), \qquad \gamma \geq 0 \tag{3.2}$$



รูปที่ 3.2 แสดงค่า pdf ของ Nakagami-m เมื่อ $\Omega = 1$ และเมื่อมีการเปลี่ยนค่าตัวแปรเสริมเฟดดิง (m = 0.5, 1, 2, 4)

เพราะฉะนั้นการกระจายแบบ Nakagami-m จะขยายออกโดยตัวแปร m เป็นช่วงกว้างที่สุดของจำนวนของการจางหาย (Amount of Fadings) (มีค่าตั้งแต่ 0 ถึง 2) ในขอบเขต m= ½ → +∞ การจางหายภายในช่องสัญญาณแบบ Nakagami-m จะบรรจบเข้ากับ การรบกวนในช่องสัญญาณแบบขาว AWGN (Additive White Gaussian Noise) หมายถึงยิ่งตัวแปร m มีค่ามากขึ้นก็จะทำให้การจางหายภายในช่องสัญญาณแบบ Nakagami-m มีผลน้อยลงไปเนื่องจากจำนวนของการจางหายจะมีการคำนวณดังนี้ $AF_m = 1/m$ โดยที่ m >= ½

3.9 การแก้ไขการทำงานของมอดูลใน OMNeT++

3.9.1 การเพิ่มการจางหายภายแบบ Nakagami-m ให้มอดูลภายใน OMNeT++

โดยปกติแล้วในการส่งข้อมูลแบบไร้สายภายใน OMNeT++ จะมีเพียงแค่ผลกระทบจากการสูญหายของสัญญาณระหว่างทาง (Path loss) เท่านั้นจึงต้องมีการเพิ่มในส่วนของผลกระทบจากการจางหายภายในช่องสัญญาณแบบ Nakagami-m เพื่อที่จะทำให้เราได้ผลลัพธ์จากกาทดลองที่แม่นยำยิ่งขึ้นตามทฤษฎีในการส่งข้อมูลแบบไร้สาย ซึ่งการแก้ไขมอดูลภายใน OMNeT++ โดยเพิ่มการจางหายภายในช่องสัญญาณแบบ Nakagami-m เข้าไปนั้นจะต้องเข้าไปทำการแก้ไขในมอดูล

Ieee80211Radio ซึ่งเป็นมอดูลที่เก็บตัวแปรต่างๆที่เกี่ยวกับสัญญาณที่ส่งแบบไร้สาย โดยเราสามารถแก้ไขมอดูลด้วยวิธีดังนี้

- เพิ่มการประกาศตัวแปร nakagami ลงในไฟล์ AbstractRadio.h ซึ่งเป็นไฟล์ที่เก็บการประกาศตัวแปรต่างๆใน OMNeT++

```
/** State: if not -1, we have to switch to that channel once we finished transmitting */
        int newChannel;
/** State: if not -1, we have to switch to that bitrate once we finished transmitting */
        double newBitrate;
/** State: the current noise level of the channel.*/
        double noiseLevel;
/** Include Nakagami-m Multipath fading */
        double nakagami;
```

- เข้าไปแก้ไขไฟล์ AbstractRadio.cc โดยในไฟล์นี้จะมีการแก้ไขอยู่ 2 ส่วนคือ

ส่วนที่ 1 เพิ่มตัวแปร nakagami เข้าไปเพื่ออ่านค่าตัวแปรที่เรากำหนดในไฟล์ omnetpp.ini เพื่อแปลงค่าจากหน่วย dBm เป็นหน่วยมิลลิวัตต์

```
// read parameters
transmitterPower = par("transmitterPower");
        if (transmitterPower > (double) (cc->par("pMax")))
        error("transmitterPower cannot be bigger than pMax in
        ChannelControl!");
        rs.setBitrate(par("bitrate"));
        rs.setChannelNumber(par("channelNumber"));
        thermalNoise = FWMath::dBm2mW(par("thermalNoise"));
        carrierFrequency = cc->par("carrierFrequency");
        sensitivity = FWMath::dBm2mW(par("sensitivity"));
        nakagami = FWMath::dBm2mW(par("nakagami"));
```

ส่วนที่ 2 นำค่าตัวแปร nakagami ที่ได้หลังการแปลงหน่วยเป็นมิลลิวัตต์แล้วมารวมเข้ากับ thermalNoise เพื่อคำนวณเป็นค่าระดับของสัญญาณรบกวนในแต่ละช่วงเวลา

```
        const Coord& myPos = myPosition();
        const Coord& framePos = airframe->getSenderPos();
        double distance = myPos.distance(framePos);
```

```
// calculate receive power
        double rcvdPower = receptionModel-
                >calculateReceivedPower(airframe->getPSend(), carrierFrequency,distance);
        thermalNoise =  FWMath::dBm2mW(normal(thermalNoise_dBm,
        thermalNoise_SD));
        noiseLevel = thermalNoise+nakagami;
```

- แก้ไขในไฟล์ Ieee80211Radio.ned โดยเพิ่มตัวรับค่าตัวแปรจากภายนอก

```
simple Ieee80211Radio
parameters:
channelNumber: numeric const, // channel identifier
bitrate: numeric const, // (in bits/s)
thermalNoise: numeric const, // base noise level (dBm)
nakagami: numeric const, // Nakagami-m Multipath Fading (dBm)
snirThreshold: numeric const, // if signal-noise ratio is below
this threshold, frame is considered noise (in dB)
sensitivity: numeric const; // received signals with power below
sensitivity are ignored
```

- เมื่อแก้ไขในไฟล์ต่างๆเรียบร้อยแล้วเราก็ต้องกำหนดค่าเริ่มต้นของตัวแปร nakagami-m ในไฟล์ omnetpp.ini อีกด้วย

```
**.radio.sensitivity = -85
**.radio.nakagami = -110
**.radio.pathLossAlpha = 2
**.radio.snirThreshold = 8  # in dB
```

3.9.2 การแก้ไขมอดูลเพื่อสุ่มค่าของสัญญาณรบกวน

ในการติดต่อสื่อสารแบบไร้สายนั้นย่อมได้รับผลกระทบจากสัญญาณรบกวนต่างๆ ซึ่งในแต่ละช่วงเวลาหรือในแต่ละพื้นที่ก็ย่อมมีระดับสัญญาณรบกวนที่แตกต่างกันออกไป โดยปกติเราสามารถแบ่งสัญญาณรบกวนจากภายนอกออกเป็นลักษณะต่างๆกันเช่น อุณหภูมิ, สภาพอากาศ ฯลฯ แต่ในการจำลองระบบเครือข่ายไร้สายใน OMNeT++ จะรวมสัญญาณรบกวนเหล่านี้เป็น thermal noise เพียงตัวเดียว ดังนั้นในการจำลองระบบเครือข่ายไร้สายควรมีการสุ่มค่าระดับสัญญาณรบกวนดังนี้

3.9.2.1 การสุ่มค่าระดับสัญญาณรบกวนเริ่มต้น

การสุ่มค่าระดับสัญญาณรบกวนเริ่มต้นเราจะแก้ไขมอดูลให้มีการใช้ค่าจากฟังก์ชั่นการแจกแจงแบบปรกติมาทำการสุ่มตัวเลขให้มีค่าแตกต่างกันออกไปเนื่องจากในการวางตัวของเครื่องลูกข่ายแต่ละเครื่องจะอยู่กระจายกันออกไป ซึ่งในแต่ละพื้นที่นั่นก็ย่อมได้รับผลกระทบจากสัญญาณรบกวนที่แตกต่างกันออกไป เช่นอุณหภูมิแตกต่างกัน เป็นต้น โดยการเลือกใช้ฟังก์ชั่นการแจกแจงแบบมาตรฐานจะช่วยเพิ่มความยืดหยุ่นในการจำลองระบบเครือข่ายไร้สายมากขึ้น เช่นหากระบบที่เราทำการจำลองมีสัญญาณรบกวนมากหรือน้อยเราก็สามารถกำหนดได้จากส่วนเบี่ยงเบนมาตรฐานให้มีค่ามากขึ้นหรือน้อยลง โดยต้องทำการแก้ไขที่ไฟล์ AbstractRadio.cc ในช่วงเริ่มต้นของการรับค่าตัวแปรให้รับค่าตัวแปรมา 2 ค่าคือรับค่า thermalNoise จะเป็นค่าเฉลี่ยของระดับสัญญาณรบกวนและรับค่า thermalNoise_SD เพื่อกำหนดส่วนเบี่ยงเบนมาตรฐานว่าสัญญาณรบกวนมีการเปลี่ยนแปลงมากน้อยแค่ไหน โดยมีลักษณะการแก้ไขในมอดูลดังนี้

```
void AbstractRadio::initialize(int stage)
{
        // read parameters
thermalNoise_dBm = par("thermalNoise");          // รับค่าเฉลี่ย
thermalNoise_SD = par("thermalNoise_SD");        // รับค่าส่วนเบี่ยงเบนมาตรฐาน
thermalNoise = FWMath::dBm2mW(normal(thermalNoise_dBm,thermalNoise_SD));
//สุ่มค่าสัญญาณรบกวนภายนอก
        // initialize noiseLevel
noiseLevel = thermalNoise;
```

3.9.2.2 การสุ่มค่าระดับสัญญาณรบกวนในแต่ละช่วงเวลา

ในการรับส่งข้อมูลในแต่ละช่วงเวลาก็ย่อมจะได้รับผลจากสัญญาณรบกวนที่แตกต่างกันออกไปเนื่องจากในแต่ละช่วงเวลาก็ย่อมมีการเปลี่ยนแปลงของอุณหภูมิหรือสภาพอากาศ ดังนั้นในขณะที่รับส่งแต่ละชุดข้อมูลก็ย่อมได้รับผลกระทบจากสัญญาณรบกวนที่แตกต่างกันออกไปด้วย โดยเราสามารถกำหนดให้แต่ละชุดข้อมูลที่ลูกข่ายจะได้รับมีค่าที่แตกต่างกันได้จากการสุ่มค่าของสัญญาณรบกวนด้วยการแจกแจงแบบปกติ โดยมีค่าเฉลี่ยค่าหนึ่งและมีส่วนเบี่ยงเบนมาตรฐานที่แตกต่างกันออกไปแล้วแต่ว่าในแต่ละระบบเครือข่ายที่ทำการจำลองจะมีระดับสัญญาณรบกวนที่มากน้อยแค่ไหนและต้องรวมกับผลกระทบจากการจางหายของ Nakagami-m ที่เกิดขึ้นระหว่างเส้นทางการรับส่งข้อมูลโดยมีลักษณะการแก้ไขที่ไฟล์ AbstractRadio.cc โดยทำการสุ่มค่าของสัญญาณรบกวนในช่วงระหว่างการรับข้อมูลที่เครื่องของลูกข่ายก่อนจะทำการเปรียบค่ากำลังสัญญาณที่ได้รับกับค่าความไวต่อสัญญาณที่ได้รับ (Seneitivity) เพื่อดูว่าชุดข้อมูลที่ได้รับมานั้นมีค่า

กำลังสัญญาณอยู่ในระดับที่รับค่าได้หรือไม่ ถ้าค่าสัญญาณที่ได้รับมีค่ามากกว่าก็จะนำข้อมูลที่ได้รับไปเก็บไว้ในบัฟเฟอร์ แต่ถ้าค่าสัญญาณที่ได้รับมีค่าน้อยกว่าค่าความไวต่อสัญญาณที่ได้รับ ก็จะถือว่าสัญญาณที่ได้รับนั้นเป็นสัญญาณรบกวนและนำค่าสัญญาณนั้นไปรวมกับค่าของระดับสัญญาณรบกวน โดยมีแก้ไขให้เป็นไปตามที่ต้องการดังนี้

```
void AbstractRadio::handleLowerMsgStart(AirFrame * airframe)
        {
                // Calculate the receive power of the message
                // calculate receive power
        double rcvdPower = receptionModel->calculateReceivedPower(airframe->
getPSend(), carrierFrequency, distance);
        thermalNoise =  FWMath::dBm2mW(normal(thermalNoise_dBm,thermalNoise_SD));
// ทำการสุ่มค่า thermal noise ขึ้นมาเพื่อใช้ในการคำนวณระดับสัญญาณรบกวน
        noiseLevel = thermalNoise+nakagami; // นำค่า thermal noise ที่ได้จากการสุ่มมารวม
กับค่าที่เกิดจากผลกระทบของการจางหายแบบ Nakagimi
        // store the receive power in the recvBuff
        recvBuff[airframe] = rcvdPower;
        // if receive power is bigger than sensitivity and if not sending and currently not
        receiving another message and the message has arrived in time
        // NOTE: a message may have arrival time in the past here when we are processing
        ongoing transmissions during a channel change
        if (airframe->arrivalTime() == simTime() && rcvdPower >= sensitivity &&
        rs.getState() != RadioState::TRANSMIT && snrInfo.ptr == NULL)
        {
        EV << "receiving frame " << airframe->name() << endl;
         // Put frame and related SnrList in receive buffer
        SnrList snrList;
        snrInfo.ptr = airframe;
        snrInfo.rcvdPower = rcvdPower;
        snrInfo.sList = snrList;
         // add initial snr value
        addNewSnr();
                if (rs.getState() != RadioState::RECV)
```

```
            {
            // publish new RadioState
            EV << "publish new RadioState:RECV\n";
            setRadioState(RadioState::RECV);
            }
        }
        // receive power is too low or another message is being sent or received
        else
        {
        EV << "frame " << airframe->name() << " is just noise\n";
        //add receive power to the noise level
        noiseLevel += rcvdPower;
```

3.9.3 การแก้ไขมอดูลเพื่อคำนวณค่าปริมาณงาน (Throughput) ของแต่ละเครื่อง

การแก้ไขมอดูลเพื่อให้ตัวมอดูลสามารถคำนวณค่าปริมาณงานซึ่งนอกเหนือจากการต่อเชื่อมตัวมอดูล ThruputMeter กับ มอดูล Ieee80211NicAPSimplified เพื่อวัดค่าปริมาณงาน ที่ตัว AP นั้นยังไม่เพียงพอจึงต้องการวัดค่าปริมาณงานในตัวของผู้รับบริการด้วย แต่เนื่องจากมอดูล ThruputMeter ไม่สามารถนำมาต่อเชื่อมกับระหว่างมอดูล Ieee80211NicSTASimplified กับมอดูล NetworkLayer ได้เนื่องจากจำนวน Gate ที่มีไม่เพียงพอ และไม่สนับสนุนการต่อเชื่อมในรูปแบบแทรกกลางระหว่างมอดูลการแก้ไขมอดูลเป็นวิธีการที่จะช่วยให้ได้ลักษณะการทำงานของตัวมอดูล ThruputMeter และยังคงลักษณะของมอดูลหลักได้

ในการแก้ไขมอดูลเพื่อให้ได้ส่วนการคำนวณค่าปริมาณงานนั้น ได้ทำการแก้ไขส่วนการทำงานของมอดูล Ieee80211MgmtSTASimplifiedในตัวมอดูล Ieee80211NicSTASimplified แต่เนื่องจากตัวมอดูลที่ต้องการแก้ไขได้ทำการสืบทอดมาจากมอดูล Ieee80211MgmtBase จึงเข้าไปทำการแก้ไขในมอดูลนั้นๆ แทน โดยมีหลักการแก้ไขที่สำคัญได้แก่ การประกาศตัวแปรและกำหนดค่าเริ่มที่ใช้ในการคำนวณ การเพิ่มการนับจำนวนบิต ก่อนที่จะทำการส่งข้อความออกไป และสร้างเมท็อดเพื่อทำการบันทึกค่าต่างๆ ได้มีไฟล์ที่ได้รับการแก้ไขดังนี้

- ไฟล์ Ieee80211MgmtBase.h ทำการประกาศค่าตัวแปร StartTime โดยกำหนดชนิดของข้อมูล เป็น simtime_t ซึ่งเป็นข้อมูลชนิดเวลา จากนั้นจึงกำหนดค่า numPackets numBits throughput และ packetPersec เป็นข้อมูลชนิด unsigned long เนื่องจากค่าที่ต้องใช้มีเพียงจำนวนบวกเท่านั้น

```
//declear for add thruput parameter
```

```
protected:
simtime_t startTime;
unsigned long numPackets ;
unsigned long numBits ;
unsigned long throughput ;
unsigned long packetPerSec ;
```

จากนั้นจึงทำการประกาศเมท็อด finnish สำหรับการcall finish() หลังจากทำการทดลองเสร็จในแต่ละครั้ง เพื่อทำการบันทึกค่าลงไฟล์ omnetpp.sca

```
protected: virtual void finish();
```

- ในตัวไฟล์ Ieee80211Mgmtbase.ccจะมีส่วนที่ต้องทำการแก้ไข 3 เมท็อด โดยในเมท็อด Ieee80211MgmtBase::initialize เป็นส่วนสำหรับทำการกำหนดค่าเริ่มต้นให้แก่มอดูลนี้ โดยจะต้องทำการกำหนดค่าที่ได้ประกาศไว้ใน Ieee80211MgmtBase.h ให้มีค่าเริ่มต้น

```
//initial for calculate throughtput
numPackets = 0;
numBits = 0;
throughput = 0;
packetPerSec = 0;
startTime = 0;
WATCH(numPackets);
WATCH(numBits);
WATCH(throughput);
WATCH(packetPerSec);
```

ต่อมาในเมท็อด Ieee80211MgmtBase::sendUp เป็นส่วนสำหรับการส่งข้อความจากมอดูลไปถึงมอดูลที่ทำการต่อเชื่อมอยู่ ซึ่งข้อมูลออกไปทาง gate uppergateOut ซึ่งได้ทำการเพิ่มส่วนการนับจำนวนแพ็คเก็ต และ จำนวนบิตจากค่าความยาวของข้อความที่จะทำการส่งข้อความออกไป

```
//process of thruput meter
numPackets++;
numBits += msg->length();
send(msg, "uppergateOut");
```

สุดท้ายในเมท็อด Ieee80211MgmtBase::finish เป็นเมท็อดที่ใช้สำหรับการคำนวณค่าทรูพุท และแพ็คเก็ตต่อวินาที หลังจากนั้นจึงทำการบันทึกค่าทั้งหมดลงตัวไฟล์ omnetpp.sca

```
void Ieee80211MgmtBase::finish() //write to scalar for thruput meter
{
        simtime_t duration = simTime() - startTime;
        throughput = numBits / simTime();
        packetPerSec = numPackets / simTime();
        recordScalar("numPackets", numPackets);
        recordScalar("numBits", numBits);
        recordScalar("avg throughput (bit/s)", throughput);
        recordScalar("packetPerSec", packetPerSec);
}
```

บทที่ 4
การจำลองระบบเครือข่ายไร้สายโดยใช้ OMNeT++

4.1 บทนำ

ในปัจจุบันการติดต่อสื่อสารแบบไร้สายได้ถูกพัฒนาจนมีระดับความเร็วที่สูงขึ้น จึงได้มีการพัฒนาเทคโนโลยีในการให้ข้อมูลภาพเคลื่อนไหวและเสียงพร้อมกันหรือที่เรียกว่า"วีดีโอสตรีมมิ่ง"ผ่านทางระบบเครือข่ายไร้สายโดยใช้โปรโตคอล UDP ในการส่งข้อมูลเนื่องจากโปรโตคอล UDP นั้นไม่จำเป็นต้องอาศัยการสร้างช่องทางเชื่อมต่อกันระหว่างแม่ข่ายกับลูกข่าย อีกทั้งไม่มีการตรวจสอบความถูกต้องของการรับส่งข้อมูลนั้น ๆ ด้วย เนื่องจากโปรโตคอล UDP ไม่มีสัญญาณสอบทานข้อมูล (acknowledgement) ในการส่งข้อมูลแต่ละครั้งและไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล ทำให้การส่งข้อมูลวีดีโอสตรีมมิ่งเป็นไปอย่างต่อเนื่อง แต่ในการติดต่อสื่อสารแบบไร้สายนั้นจะได้รับผลกระทบจากสัญญาณรบกวน และการจางหายในช่องสัญญาณต่างๆรวมไปถึงจำนวนลูกข่ายที่เข้าใช้บริการ

โครงการนี้จึงนำเสนอการจำลองระบบเครือข่ายไร้สายเพื่อหารูปแบบของการส่งข้อมูลแบบสตรีมมิ่งที่เหมาะสมกับระบบเครือข่ายไร้สายที่ได้รับผลกระทบจากสัญญาณรบกวนและการจางหายในช่องสัญญาณ โดยทดลองส่งข้อมูลที่มีขนาดความยาวของชุดข้อมูล (Payload) ในขนาดต่างๆกัน และทดลองส่งด้วยอัตราการส่งข้อมูลที่ความเร็วต่างๆตามมาตรฐาน IEEE 802.11b

4.2 จุดประสงค์ของการทดลอง

- เพื่อดูผลกระทบที่ได้รับจากสัญญาณรบกวนและการจางหายในช่องสัญญาณ
- เพื่อหาความยาวของชุดข้อมูลที่เหมาะสมในการส่งข้อมูลแบบวีดีโอ สตรีมมิ่ง (Video Streaming) ภายในระบบเครือข่ายไร้สาย โดยเปรียบเทียบจากการส่งข้อมูลด้วยขนาดความยาวของข้อมูลที่ต่างกัน
- เพื่อดูค่าปริมาณงานที่เครื่องลูกข่ายแต่ละเครื่องจะได้รับ
- เพื่อทดลองส่งข้อมูลไปยังเครื่องลูกข่ายที่มีจำนวนต่างๆกัน

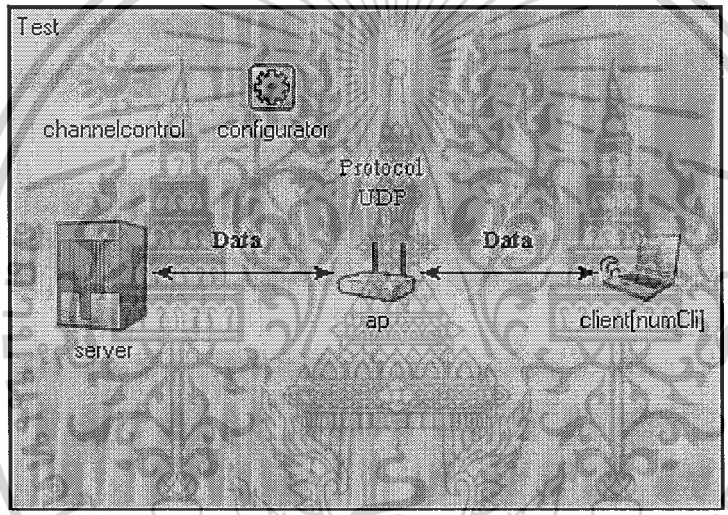
4.3 การสร้างแบบทดสอบการจำลอง

การสร้างแบบทดสอบการจำลอง สามารถทำได้โดยการวางรูปแบบของแบบจำลอง แล้วจึงทำการสร้างแบบจำลองขึ้น และทำการสร้างไฟล์ซึ่งเก็บค่าพารามิเตอร์ต่างๆ ของมอดูลทั้งหมด

หลังจากนั้นถึงจะกระทำการดำเนินการจำลองได้ ซึ่งจะได้เห็นวิธีการส่งผ่านข้อมูลของระบบเครือข่ายไร้สาย เมื่อทำการทดลองจนได้ถึงเวลาที่กำหนดจึงได้ทำการเรียกให้ทุกมอดูลทำการสั่งสิ้นสุดการทำงาน เมื่อทำการจำลองเสร็จก็จะได้ไฟล์ที่เก็บค่าทั้งแบบสเกลล่าร์และกราฟ

4.3.1 รูปแบบของแบบจำลอง

โดยในการทดสอบนี้ได้ออกแบบให้เป็นการทดสอบการทำงานของระบบเครือข่ายไร้สาย เมื่อมีการส่งผ่านข้อมูลระหว่างผู้ให้บริการและลูกข่าย โดยให้บริการแบบวิดีโอสตรีมมิ่ง บนโปรโตคอล UDP โดยได้ทดลองเปลี่ยนแปลงค่าขนาดข้อมูล และได้ทดลองให้บริการจำนวนตั้งแต่ 1 5 10 และ 15 และได้ทดลองในสภาพแวดล้อมที่แตกต่างกัน โดยกำหนดจากค่าสัญญาณรบกวน เพื่อดูค่าปริมาณงานที่ทำได้



รูปที่ 4.1 แสดงโครงสร้างของเครือข่ายที่ใช้ทดลอง

จากภาพเป็นการส่งผ่านข้อมูลระหว่างผู้ให้บริการและลูกข่าย โดยในการทดลองนี้จะมีการกำหนดค่าขนาดข้อมูล สัญญาณรบกวน และจำนวนลูกข่ายที่รับบริการด้วย

4.4 การสร้างแบบจำลอง

จากรูปแบบการจำลองที่ทำขึ้นจึงได้ทำการสร้างแบบจำลองจากมอดูลต่างๆ ประกอบเข้าด้วยกัน ซึ่งจะต้องประกอบไปด้วยมอดูลต่างๆ ซึ่งมีมอดูลหลักชื่อ Test ซึ่งจะเป็นส่วนรวมทั้งหมดของมอดูลทั้งหมดที่ทำงานในระบบเครือข่ายที่ทำการทดสอบนี้ ซึ่งภายในมอดูล Test จะประกอบไปด้วยมอดูลต่างๆ ดังนี้

- ChannelControl
- FlatNetworkConfigurator

- WirelessAPWithThruputMeter
- WirelessHostSimplified

การรวมมอดูลต่างๆ เหล่านี้ให้ทำงานร่วมกัน จะต้องใช้ภาษา NED โดยเขียนเป็น sourcecode ได้ดังนี้

```
Import
"WirelessAPWithThruputMeter" ;
```

ในส่วนของการเพิ่มมอดูล (import) จากภายนอกระบบเข้ามาซึ่งเราจะเห็นได้ว่ามีมอดูล WirelessAPWithThruputMeter ที่ทำการเพิ่มเข้ามาเพื่อที่จะนำมาใช้เป็นตัว accesspoint ซึ่งในส่วนของรายละเอียดขอกล่าวถึงในภายหลัง ส่วนตัวมอดูลอื่นๆ ที่ไม่ได้ทำการ import จะทำการโหลดแบบไดนามิกเข้ามาในขณะทำการดำเนินการ (Run Time) ซึ่งตัวมอดูลเหล่านั้นเป็นมอดูลที่มีอยู่ในกรอบการทำงาน (INET Framework)

```
module Test
   parameters:
         numCli: numeric const,
         playgroundSizeX: numeric const,
         playgroundSizeY: numeric const;
          // ส่วนของ submodule
Endmoule
```

เป็นส่วนของการประกาศมอดูลชื่อ Test และทำการประกาศค่าพารามิเตอร์ของ มอดูลนี้ซึ่งประกอบไปด้วย

- numCli() เป็นค่าที่ประกาศเพื่อเป็นจำนวนเครื่องของ Host ที่จะใช้ในการจำลอง
- playgroundSizeX เป็นค่าขนาดของพื้นที่จำลองตามค่าแกน X
- playgroundSizeY เป็นค่าขนาดของพื้นที่จำลองตามค่าแกน Y

โดยที่ค่าพารามิเตอร์ทั้งหมดนั้นถูกกำหนดให้มีค่าแบบค่าคงที่เป็นตัวเลข

```
   submodules:
      client: WirelessHostSimplified[numCli];
         display: "p=243;i=device/wifilaptop;r=,,#707070";
      server: WirelessHostSimplified;
         display: "p=55,136;i=device/server_l;r=,,#707070";
      ap: WirelessAPWithThruputMeter;
         display: "p=160,142;i=device/accesspoint;r=,,#707070";
      channelcontrol: ChannelControl;
```

```
        parameters:
            playgroundSizeX = playgroundSizeX,
            playgroundSizeY = playgroundSizeY;
        display: "p=61,46;i=misc/sun";
    configurator: FlatNetworkConfigurator;
        parameters:
            moduleTypes = "WirelessHostSimplified",
            nonIPModuleTypes = "WirelessAPWithThruputMeter",
            networkAddress = "145.236.0.0",
            netmask = "255.255.0.0";
        display: "p=140,50;i=block/cogwheel_s";
connections nocheck:
display: "b=297,203";
```

ส่วนของการทำมอดูลย่อย (submodule) จะทำการเรียกให้มอดูลอื่นๆ เข้ามาประกอบเป็นมอดูล Test ซึ่งจะประกาศดังนี้

- client: WirelessHostSimplified[numCli];

มอดูล WirelessHostSimplified ซึ่งจะถูกเรียกใช้เป็นมอดูลชื่อ client ซึ่งในส่วนนี้มีการกำหนดเป็น Vectorจะเห็นได้ว่าภายในมีการประกาศจำนวน element ตามค่าพารามิเตอร์ numCli ซึ่งหมายความว่าในมอดูล test นี้จะประกอบไปดวยมอดูล client ตั้งแต่ client[0] จนถึง client[numcli] ซึ่งจะเป็นมอดูลแบบ WirelessHostSimplified

- server: WirelessHostSimplified;

เป็นการประกาศมอดูล server ให้เป็นมอดูลแบบ WirelessHostSimplified

- ap: WirelessAPWithThruputMeter;

เป็นการประกาศมอดูล ap ให้เป็นมอดูลแบบ WirelessAPWithThruputMeter

- channelcontrol: ChannelControl;

เป็นการประกาศมอดูล channelcontrol ให้เป็นมอดูลแบบ ChannelControl

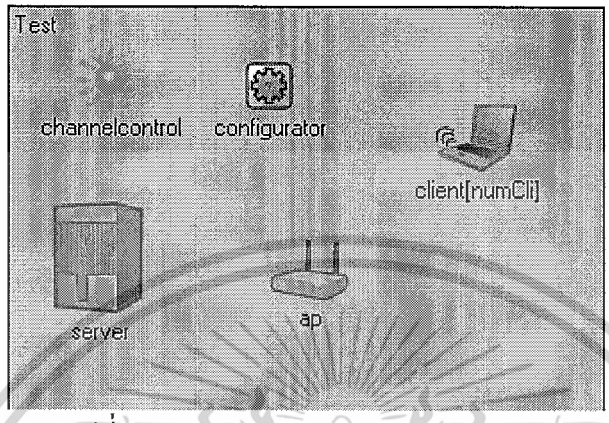
- configurator: FlatNetworkConfigurator

เป็นการประกาศมอดูล configurator เป็นมอดูลแบบ FlatNetwoekConfigurator

ในส่วนของ display คือการประกาศรูปแบบการแสดงผลในการจำลองให้มีรูปภาพอย่างไรและมีตำแหน่งใด และในส่วนของ parameter ภายใต้มอดูลย่อยจะเป็นการประกาศค่าพารามิเตอร์ของมอดูลนั้นๆ

```
network test : Test
endnetwork
```

ในส่วนการประกาศ network ให้ประกาศชื่อ test โดยใช้มอดูล Test



รูปที่ 4.2 แสดงมอดูลย่อยภายในมอดูล test

เมื่อเปิดตัวไฟล์ test.ned ด้วยโปรแกรม GNED จะได้ผลดังภาพ ซึ่งรูปต่างๆจะเป็นมอดูลย่อยที่ประกอบอยู่ในมอดูล Test โดยรายละเอียดของมอดูลย่อยต่างๆ มีดังนี้

4.4.1 มอดูล ChannelControl

มอดูล ChannelControl เป็นมอดูลที่จำเป็นที่ต้องมีอยู่ในการจำลองระบบเครือข่ายที่มีโหนดแบบไร้สายหรือแบบเคลื่อนที่ โดยมอดูล ChannelControl จะเก็บข้อมูลของตำแหน่งและการเคลื่อนที่ของโหนด และประเมินโหนดที่จะทำการติดต่อสื่อสาร หรือสิ่งที่เกิดเนื่องจากระยะทาง ด้วยข้อมูลเหล่านี้จะถูกใช้โดยส่วนติดต่อคลื่นวิทยุ (Radio Interface) ของโหนดที่ทำการส่งสัญญาณ

```
simple ChannelControl
  parameters:
    coreDebug: numeric const, // debug switch for core framework
    playgroundSizeX: numeric const, // x size of the playground (in meters)
    playgroundSizeY: numeric const, // y size of the playground (in meters)
    pMax: numeric const, // maximum sending power used for this network (in mW)
    sat: numeric const, // signal attenuation threshold (in dBm)
    alpha: numeric const, // path loss coefficient
    carrierFrequency: numeric const, // carrier frequency of the channel (in Hz)
    numChannels: numeric const; // number of radio channels (frequencies)
```

```
endsimple
```

4.4.2 มอดูล FlatNetworkConfigurator

เป็นมอดูลที่ทำการจัดการการตั้งค่า IP Addresses และ Routing Table แต่ในมอดูลนี้ได้ใช้แต่การตั้งค่า IP Addresses ซึ่งมอดูลนี้จะใช้สำหรับระบบเครือข่ายที่มี hosts และ router ที่มี network address เดียวกัน

โดยตัวมอดูลจะมีลักษณะที่ไม่มีการต่อเชื่อมกับมอดูลอื่นใด และควรที่จะต้องมีเพียงมอดูลเดียวในแบบจำลองใดๆ และมอดูลนี้จะมีการทำงานเพียงรอบเดียวในขณะที่ทำการเริ่มดำเนินการจำลอง

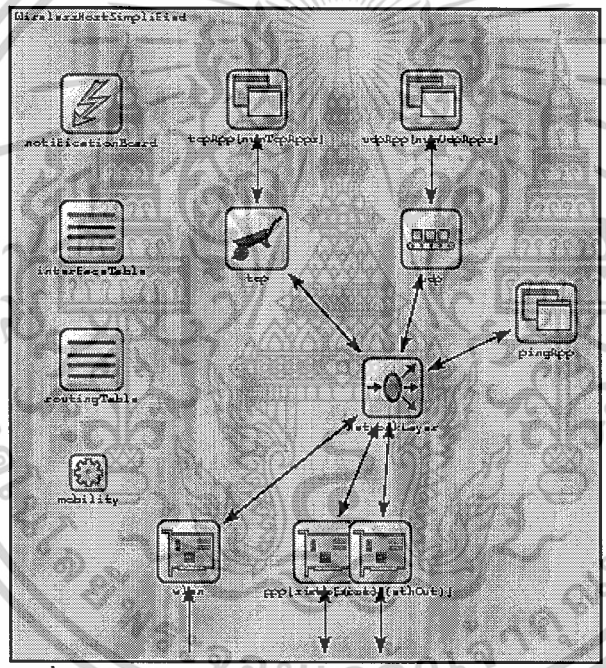
```
simple FlatNetworkConfigurator
  parameters:
    moduleTypes: string, // all module types to be considered part of the topology
    nonIPModuleTypes: string, // module types which don't need \IP configuration
                  // (e.g. Ethernet hub, switch or bus)
    networkAddress: string, // network part of the address (see netmask parameter)
    netmask: string; // host part of addresses are autoconfigured
endsimple
```

4.4.3 มอดูล WirelessHostSimplified

ตัวมอดูลมีลักษณะเป็นมอดูลประกอบ โดยจะประกอบไปด้วยส่วนเชื่อมต่อ(Interface) ที่มีมาตรฐานของ Ieee 802.11b แต่เป็นการใช้ตัวมอดูลที่เป็น Simplified ซึ่งไม่สนับสนุนการทำ handovers ตัวมอดูลจะประกอบไปด้วยมอดูลย่อยจำนวนมาก และมีหน้าที่ต่างๆ ดังนี้

- BasicMobility เป็นมอดูลที่ควบคุมเกี่ยวกับการเคลื่อนที่ของตัวมอดูลซึ่งในตัว BasicMobility ไม่ใช่เป็นตัวควบคุมจริง แต่เป็นตัวที่สร้างเพื่อรองรับรูปแบบการเคลื่อนที่ต่างๆ ที่จะเลือกใช้ในมอดูล
- EthernetInterface เป็นส่วนที่สอดคล้องกับแบบอย่างของ Network Interface มีส่วนประกอบของ EtherMac และ EtherEncap จากส่วนที่ได้จากคิวสำหรับการสนับสนุน Qos และ RED
- Ieee80211NicSTASimplified เป็นส่วนเชื่อมต่อกับระบบซึ่งเป็นตามมาตรฐาน Ieee 802.11b แต่ไม่สนับสนุนการทำ Handovers
- InterfaceTable เป็นที่เก็บตารางของส่วนเชื่อมต่อระบบเครือข่าย (Network Interface)
- NetworkLayer ลำดับชั้นระบบเครือข่าย(Network Layer)ของ IP โหนด

- NotificationBorad มอดูลสามรถทำการแจ้งเกี่ยวกับเหตุการณ์ต่างๆ
- PPPInterface ประกอบด้วยมอดูล PPP จากส่วนที่ได้จากคิวสำหรับการสนับสนุน Qos และ RED
- PingAPP สร้างการร้องขอการ ping และคำนวณเกี่ยวกับ packet ที่สูญเสีย และค่าพารามิเตอร์เกี่ยวกับรอบของการส่ง (round trip time) ของการส่งกลับ
- RoutingTable เก็บตารางการกำหนดเส้นทาง(Routing Table)
- TCP เป็นการนำโปรโตคอล TCP เพื่อใช้งาน
- TCPApp เป็นรูปแบบการทำงานของโปรแกรมที่ใช้โปรโตคอล TCP ในการทำงาน
- UDP เป็นการนำโปรโตคอล UDP เพื่อใช้งาน
- UDPApp เป็นรูปแบบการทำงานของโปรแกรมที่ใช้โปรโตคอล UDP ในการทำงาน



รูปที่ 4.3 ลักษณะของมอดูล WirelessHostSimplified

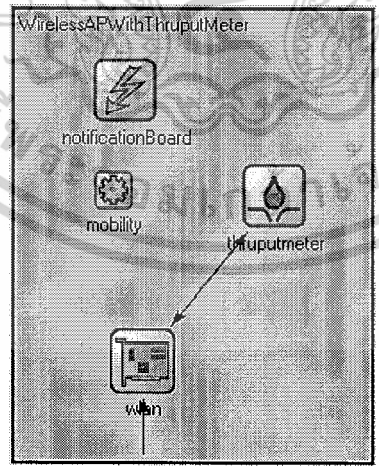
4.4.4 มอดูล WirelessAPWithThruputMeter

ในตัวมอดูลนี้เป็นการที่นำมอดูล WirelessApSimplified มาสร้างต่อโดยเพิ่มมอดูล ThruputMeter เข้าไปในมอดูลนั้นๆ เพื่อที่จะได้ตัวมอดูลที่ทำหน้าที่เป็น WirelessAP แบบ Simplified คือ Wireless Accesspoit แบบไม่สนับสนุนการทำ Handovers ที่มีตัววัดค่าทรูพุด (through put) ของแบบจำลอง ประกอบด้วยมอดูลต่างๆ ดังนี้

- ThruputMeter เป็นมอดูลที่ทำการเก็บค่าทรูพุด (through put)

- Ieee80211NicAPSimplified เป็นส่วนเชื่อมต่อกับระบบซึ่งเป็นตามมาตรฐาน Ieee 802.11b แต่ไม่สนับสนุนการทำ Handovers โดยเป็นส่วนเชื่อมต่อของอุปกรณ์ access point
- NotificationBoard มอดูลสามรถทำการแจ้งเกี่ยวกับเหตุการณ์ต่างๆ
- NullMobility ใช้สำหรับโหนดที่ไม่มีการเคลื่อนที่

```
submodules:
    Thruputmeter: ThruputMeter;
      display: "i=block/sink;p=169,124"
    notificationBoard: NotificationBoard;
      display: "p=79,74;i=block/control";
    wlan: Ieee80211NicAPSimplified;
      display: "p=110,179;q=queue;i=block/ifcard";
    mobility: NullMobility;
      display: "p=144,70;i=block/cogwheel_s";
  connections nocheck:
    radioIn --> wlan.radioIn display "m=s";
    wlan.uppergateOut --> thruputmeter.in;
    Wlan.uppergateIn <-- thruputmeter.out;
endmodule
```



รูปที่ 4.4 แบบจำลองของ WirelessApSimplified

4.5 การสร้างไฟล์กำหนดค่า (Configuration file)

ไฟล์กำหนดค่าเป็นไฟล์ที่เก็บค่าพารามิเตอร์ต่างๆ ที่ไม่ได้กำหนดภายในมอดูล การทำงานของการจำลองจะทำการตรวจสอบหาค่าพารามิเตอร์ก่อนการสร้างการจำลอง โดยจะค้นหาค่าเรียงลำดับจาก การประกาศภายในมอดูล ตามด้วยตัวไฟล์กำหนดค่า แล้วทำการขึ้นแสดงถามถึงค่าที่ยังไม่มีการกำหนดใดๆ ขณะที่กำลังมีการเริ่มดำเนินการจำลอง โดยถ้ามีพารามิเตอร์ตัวใดที่ไม่ได้ถูกกำหนดค่า และต้องทำการใส่ค่าก่อนที่จะเริ่มทำการจำลอง ไม่ควรที่จะมีการใส่จำนวนมากซึ่งอาจนำความสับสนเข้ามาได้ จึงควรที่จะทำการกำหนดค่าภายในไฟล์

โดยตัวไฟล์จะถูกเรียกใช้จากการดำเนินการจำลอง ต้องตั้งชื่อ “omnetpp.ini” เป็นไฟล์ที่มีนามสกุลเป็น ini ในการทดลองจะมีการเปลี่ยนแปลงเพียงค่าขนาดของข้อมูล สัญญาณรบกวน และจำนวนลูกข่ายดังนั้นแต่ละแบบจำลองจะมีการกำหนดดังนี้

4.5.1 ส่วน [General]

เป็นส่วนค่าทั่วไปของการจำลอง

```
[General]
output-file=omnetpp.log
preload-ned-files = *.ned @C:/INET-20061020/nedfiles.lst
;debug-on-errors = true
network = test
```

Output-file ไฟล์ที่เป็นการเก็บค่าการกระทำจะเก็บอยู่ที่ไฟล์ชื่อ “omnetpp.log”

preload-ned-files เป็นการโหลดตัวไฟล์ ned ที่มีใน INET framework ทั้งหมดโดยโหลดจากไฟล์ ned.files.lst

debug-on-errors การกำหนดให้มีการแก้ไขขอผิดพลาด

network จำลองระบบเครือข่ายชื่อ “test”

4.5.2 ส่วน [Cmdenv]

เป็นส่วนของ command line user interface

```
[Cmdenv]
express-mode = no
```

ในส่วนนี้ไม่ทำการกำหนดค่าให้ทำการใดๆ เกี่ยวกับ command line

4.5.3 ส่วน [Tkenv]

เป็นการกำหนดค่าเกี่ยวกับการติดต่อกับผู้ใช้โดยการใช้ภาพ (graphic user interface)

```
[Tkenv]
plugin-path=C:/INET-20061020/Etc/plugins
;default-run=1
```

plugin-path เป็นการกำหนดการดึงค่าเกี่ยวกับรูปต่างๆ ที่มีอยู่ในไฟล์ “plugins” เพื่อทำการแสดงผล

4.5.4 ส่วน [Run]

เป็นส่วนที่ทำการแยกรูปแบบการดำเนินการจำลองแต่ละรูปแบบออกจากกัน โดยในส่วนนี้หากไม่ได้อยู่ในส่วนการดำเนินการเดียวกันจะไม่มีการกำหนดค่านั้นๆ ลงไปในการจำลอง ซึ่งจะมีการเลือกเมื่อเริ่มต้นการจำลอง

```
[Run 1]
description = "1 client"
test.numCli = 1

[Run 2]
description = "5 clients"
test.numCli = 5

[Run 3]
description = "10 clients"
test.numCli = 10

[Run 4]
description = "15 clients"
test.numCli = 15
```

โดยในการจำลองนี้แต่ล่ะการจำลองจะมีการกำหนด 4 รูปแบบการดำเนินการ สิ่งที่ต่างกันคือจำนวนลูกข่ายที่ใช้บริการ เพื่อจะทดสอบการให้บริการของผู้ให้บริการ โดยกำหนดจำนวนตั้งแต่ 1, 5, 10 และ 15 เครื่อง

4.5.5 ส่วน [Parameters]

เป็นส่วนที่ทำการกำหนดค่าพารามิเตอร์ต่างๆ ในแต่ล่ะมอดูล

4.5.5.1 ค่าพารามิเตอร์ของการจำลองโปรโตคอล UDP

การจำลองโปรโตคอล UDP ได้ทำการการจำลองการทำงานการส่งผ่านข้อมูลวีดิโอ (videoStreaming) การกำหนดค่าทำโดยแยกส่วนของผู้ให้บริการและผู้ใช้บริการโดยจะมีค่าที่ต้องทำการกำหนดดังนี้

- การกำหนดค่าของผู้ให้บริการ (Server)

```
**.server.udpAppType = "UDPVideoStreamSvr"
**.server.udpApp[0].videoSize = 1e10
**.server.udpApp[0].serverPort = 3080
**.server.udpApp[0].waitInterval = .01
**.server.udpApp[0].packetLen = 2000
```

โดยทำการตั้งค่าให้ชนิดของการทำงานเป็น UDPVideoStreamSvr โดยการทำงานแบบนี้จะเป็นการตั้งการร้องของจากผู้ให้บริการส่งผ่านข้อมูลวีดีทัศน์ ในส่วนค่าพารามิเตอร์ที่เหลือ เป็นการกำหนดค่าของตัวการทำงานใดๆ ให้มีค่าของ

ตารางที่ 4.1 ตารางค่าพารามิเตอร์ของการตั้งค่าทั่วไปของผู้ให้บริการบนโปรโตคอล UDP

ชื่อ	ชนิดของค่า	คำอธิบาย
videoSize	Numeric const	ขนาดของการส่งวีดิทัศน์หน่วยเป็นไบต์(bytes)
serverPort	Numeric	พอร์ตที่ใช้ในการรับการร้องขอจากผู้ใช้บริการ
waitInterval	Numeric	ช่วงเวลาการรอ
packetLen	Numeric	ขนาดของแพ็คเก็ตที่ใช้

- การกำหนดค่าของลูกข่าย (Client)

```
**.client[*].udpAppType="UDPVideoStreamCli"
**.client[*].udpApp[0].serverAddress = "server"
**.client[*].udpApp[0].localPort = 9999
**.client[*].udpApp[0].serverPort = 3080
**.client[*].udpApp[0].startTime = 0
```

ทำการตั้งค่าให้ชนิดของการทำงานเป็น UDPVideoStreamCli เพื่อให้ตัวลูกข่ายทำงานตามลักษณะที่เป็นตามรูปแบบผู้ใช้บริการส่งผ่านข้อมูลวีดิทัศน์ ในส่วนค่าพารามิเตอร์ที่เหลือ เป็นการกำหนดค่าของตัวการทำงานใดๆ ให้มีค่าของ

ตารางที่ 4.2 ตารางค่าพารามิเตอร์ของการตั้งค่าทั่วไปของลูกข่ายบนโปรโตคอล UDP

ชื่อ	ชนิดของค่า	คำอธิบาย
serverAddress	String	ชื่อของมอดูลที่ทำหน้าที่เป็นผู้ให้บริการ
localPort	Numeric	พอร์ตที่ใช้ในการร้องขอข้อมูลจากผู้ให้บริการ(ตัวเอง)
serverPort	Numeric	พอร์ตที่ใช้ในการร้องติดต่อผู้ให้บริการ(ผู้ให้บริการ)
startTime	Numeric	เวลาเริ่มการทำงาน

- การกำหนดค่าของสัญญาณ (Radio)

```
**.radio.transmitterPower = 20.0  ;[mW]
**.radio.carrierFrequency = 2.4E+9
**.radio.sensitivity = -85
**.radio.pathLossAlpha = 2
**.radio.bitrate = 11e6
**.radio.thermalNoise = -110
**.radio.thermalNoise_SD = 10
**.radio.nakagami = -90
**.radio.snirThreshold = 8  # in dB
```

ทำการกำหนดค่าต่างๆของสัญญาณ เพื่อให้ค่าของสัญญาณมีลักษณะตามต้องการเพื่อเป็นการกำหนดสภาพแวดล้อม โดยค่าที่จะถูกเปลี่ยนแปลงคือค่า bitrate ,themalNoise_SD และ nakagami

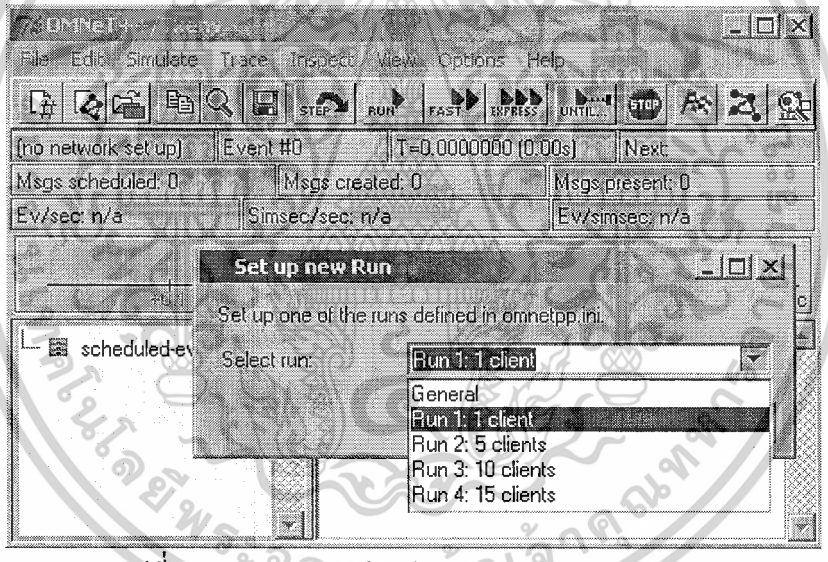
ตารางที่ 4.3 ตารางค่าพารามิเตอร์ของการตั้งค่าทั่วไปของสัญญาณ

ชื่อ	ชนิดของค่า	คำอธิบาย
transmitterPower	Numeric	ค่ากำลังส่งของสัญญาณ หน่วยเป็นมิลลิวัตต์
carrierFrequency	Numeric	ค่าความถี่ของสัญญาณตัวกลาง
sensitivity	Numeric	ค่าความไวต่อสัญญาณ
pathLossAlpha	Numeric	ค่าตัวแปรสำหรับคำนวณค่าสูญหายระหว่างการเดินทาง
bitrate	Numeric	ค่าอัตราการส่งข้อมูล
thermalNoise	Numeric	ค่าเฉลี่ยสัญญาณรบกวนแบบทั่วไป
thermalNoise_SD	Numeric	ค่าเบี่ยงเบนมาตรฐานของสัญญาณรบกวนแบบทั่วไป
nakagami	Numeric	ค่าการจางหายรูปแบบนากากามิ

4.6 การจำลอง (simulation)

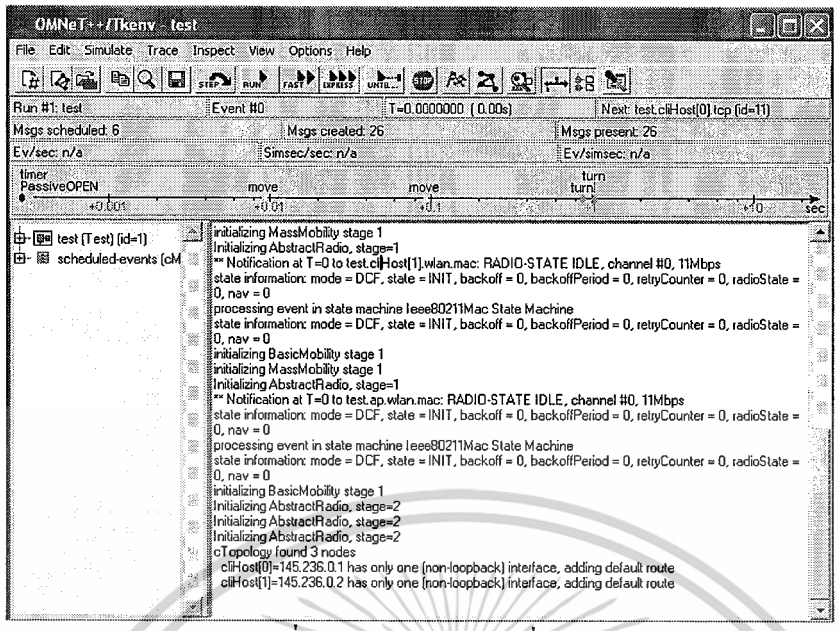
การจำลองจะเป็นการทอสอบค่าปริมาณงาน (Through put) ของการทำงานโดยจะมีการปรับเปลี่ยนค่าขนาดของมูลตั้งแต่ 250, 500, 1000, 1500 และ 2000 โดยมีสภาพของสัญญาณรบกวนที่มีลักษณะการรบกวนแบบทั่วโปโดยจะทำการสุ่มค่าแบบปกติโดยมีค่าเฉลี่ย -110 มีค่าเบี่ยงเบนมาตรฐานตั้งแต่ 5, 7 และ 10 และให้บริการจำนวนลูกข่ายที่ต่างๆ กันตั้งแต่ 1, 5, 10 และ 15 ซึ่งใช้โปรโตคอล UDP และลักษณะการให้บริการวิดีโอสตรีมมิ่ง โดยในรายงานฉบับนี้จะแสดงวิธีการจำลองแบบให้บริการลูกข่าย 1 เท่านั้น เพื่อแสดงให้เห็นลักษณะของการจำลองเท่านั้น ในส่วนของรูปแบบการจำลองของตัวโปรแกรม OMNeT++ นั้นจะมีรูปแบบการจำลองแบบเป็นเหตุการณ์ (event) ที่จะมีการคำนวณว่าแต่ล่ะเหตุการณ์มีการทำงานใดบ้างและมีเวลาเท่าใดขณะที่ทำเหตุการณ์นั้นๆ โดยจะเห็นแต่ล่ะเหตุการณ์ดังนี้

- เมื่อเริ่มทำการจำลองจะขึ้นให้เลือกรูปแบบการดำเนินการจำลอง

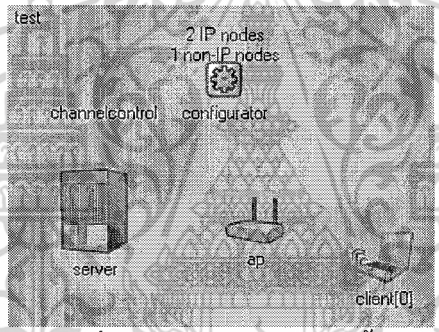


รูปที่ 4.5 การกำหนดค่ารูปแบบการดำเนินการจำลอง

- เมื่อทำการดำเนินการจำลองและทำการเลือกวิธีการดำเนินการแบบ 1 คู่แล้วโปรแกรมจะทำการกำหนดค่าเริ่มต้นของทุกโหนด และแสดงแบบจำลอง

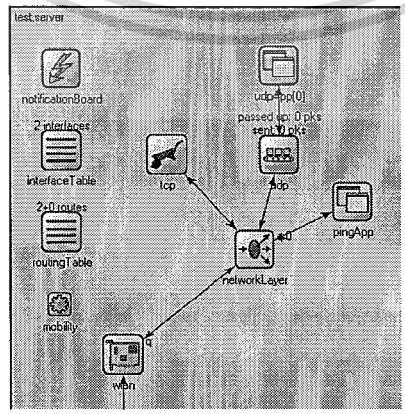


รูปที่ 4.6 การกำหนดค่าเริ่มต้น



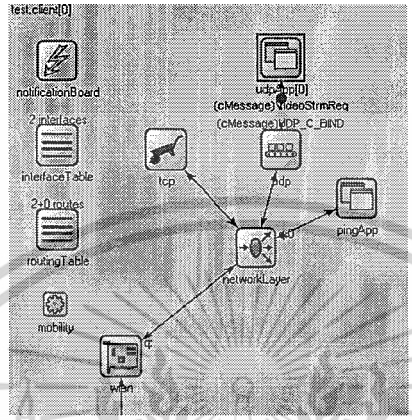
รูปที่ 4.7 แบบจำลองที่เกิดขึ้น

- ต่อมาตัวมอดูล Server จะทำการเปิดช่องทาง (Port) การติดต่อสื่อสาร และมีการแสดงค่าเหนือมอดูล udp ซึ่งจะเป็นค่าของกลุ่มข้อมูล (packet) ที่ได้ทำการส่งและการรับ



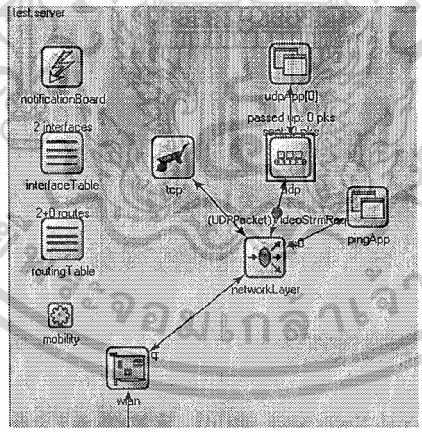
รูปที่ 4.8 แสดงมอดูลServer เมื่อทำการเปิดช่องทาง

- ภายในมอดูล client[0] ได้ทำการยึดเหนี่ยว (binding) หมายเลขช่องทาง (Port Number) ที่ 3088 กับ ช่องทางที่ 9999 ซึ่งเป็นช่องทางของตัวผู้ให้บริการ (Server) แล้วจึงทำการส่งสาร (Message) การร้องขอข้อมูลจากผู้ให้บริการ



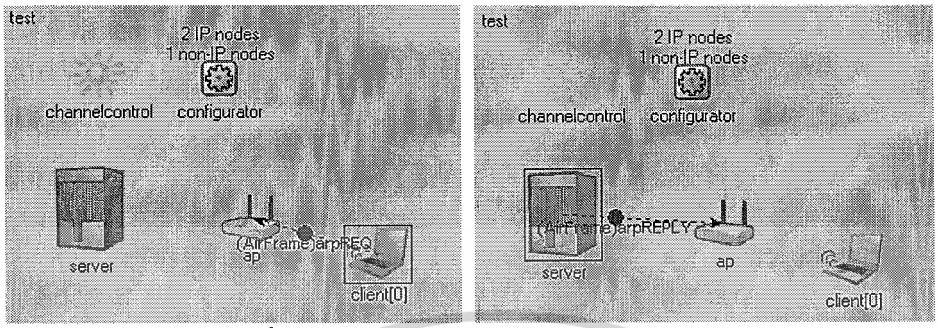
รูปที่ 4.9 แสดงการส่งสารเพื่อทำการยึดเหนี่ยวช่องทางและร้องขอข้อมูล

- ตัวมอดูล udp จะทำการยึดเหนี่ยวช่องทางแล้วจึงทำการส่งข้อมูลการร้องขอข้อมูลต่อไปให้มอดูล networkLayer เพื่อที่จะทำการส่งผ่านข้อมูลไปยังผู้ให้บริการ



รูปที่ 4.10 แสดงการส่งกลุ่มข้อมูลการร้องข้อให้มอดูล networkLayer

- ต่อมาจะเป็นการใช้โปรโตคอล ARP ในการส่ง arp request และ arp reply ซึ่งจะทำการส่งผ่านตัว access point เพื่อให้ได้ที่อยู่ปลายทาง

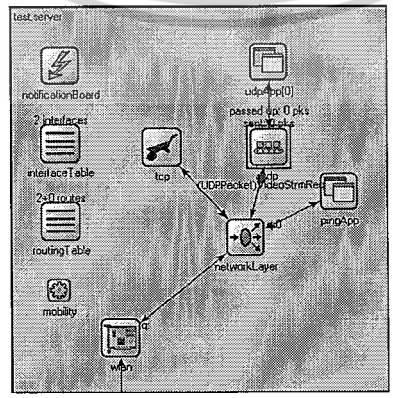


รูปที่ 4.11 การส่ง arp request และ arp reply

- ทำการส่ง AirFrame ข้อมูลการร้องขอไป Server

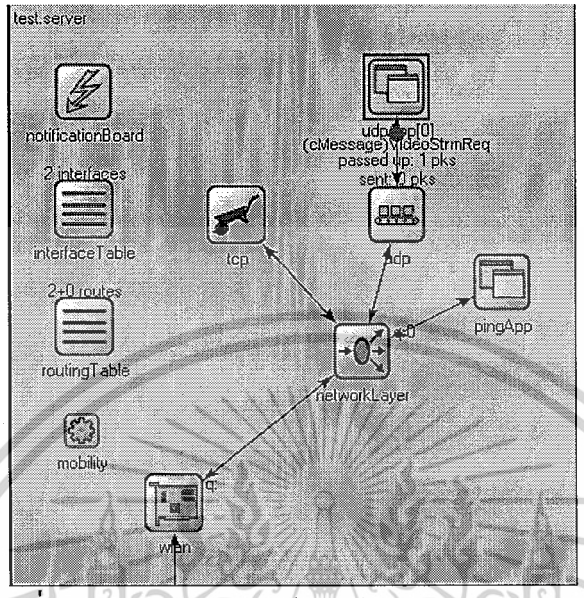
รูปที่ 4.12 แสดงการส่ง AirFrame ไปมอดูล Server

- เมื่อได้รับจึงทำการส่งกลุ่มข้อมูลการร้องขอข้อมูลถึงตัวมอดูล udp เพื่อทำการประมวลผลต่อไป



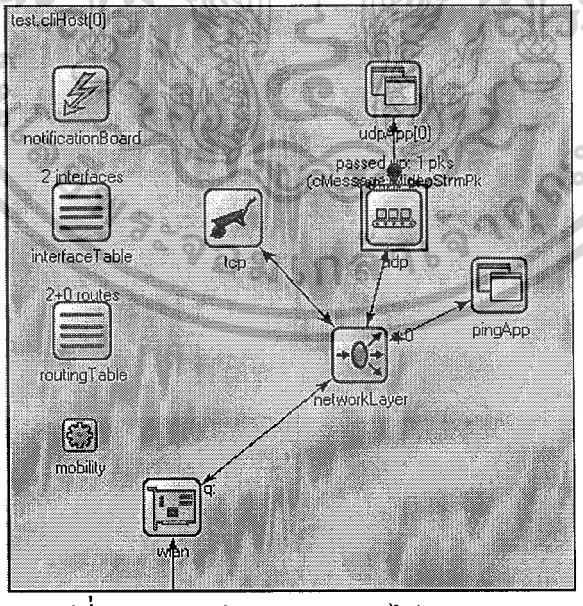
รูปที่ 4.13 การส่งกลุ่มข้อมูลร้องขอจากมอดูล networkLayer ไป udp

- เมื่อทำการเปรียบเทียบช่องทางที่ถูกต้องจึงทำการส่งสารไปถึงตัวมอดูล udpApp[0] เพื่อที่จะให้ตัวมอดูลนั้นทำการส่งข้อมูลกลับไป



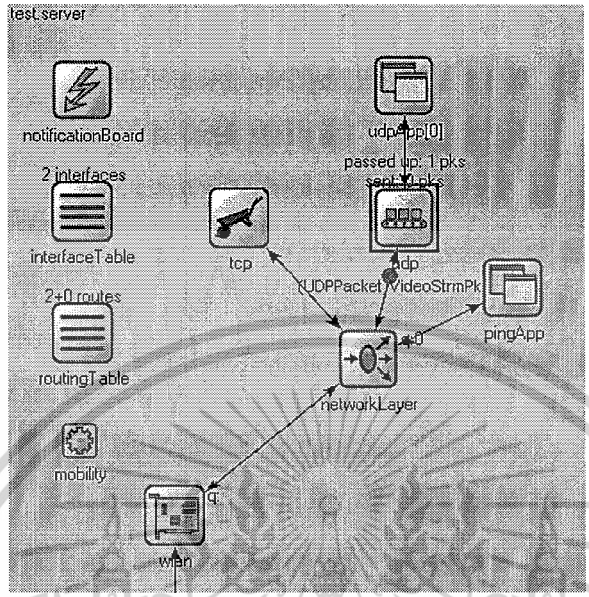
รูปที่ 4.14 การส่งสารการร้องขอไปมอดูล udpApp[0]

- เมื่อมอดูล udpApp[0] ได้รับการร้องขอจะเริ่มทำการส่งสารกลุ่มของข้อมูล stream ต่อไป



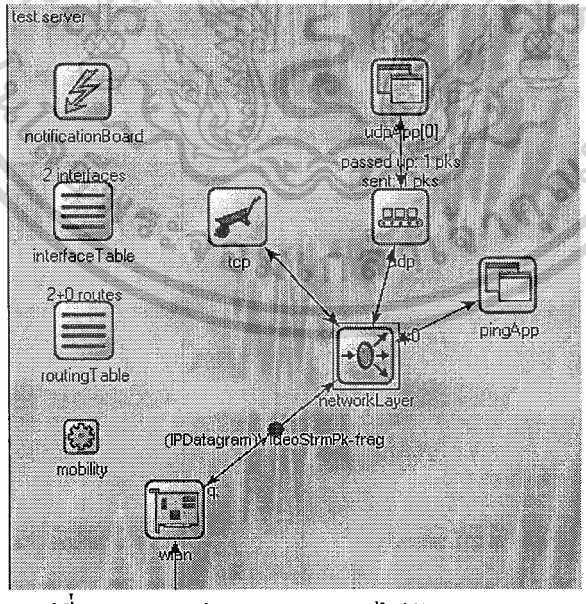
รูปที่ 4.15 การส่งสาร StrmPk ไปมอดูล udp

- ต่อไปมอดูล udp ทำการส่งกลุ่มข้อมูลที่ตัว client[0] ต้องการ ไปยังมอดูล networkLayer เพื่อทำงานในขั้นต่อไปเพื่อส่งข้อมูลที่ client[0] ร้องขอ



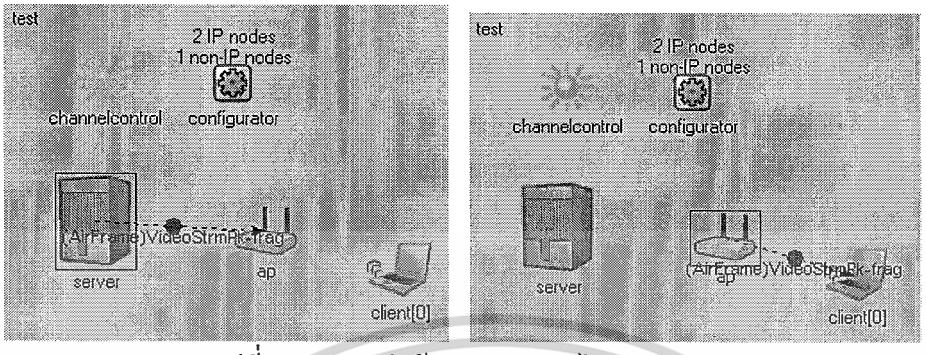
รูปที่ 4.16 การส่งแพ็คเก็ต UDP ข้อมูลไปยังมอดูล networkLayer

- จากนั้นข้อมูล IPDatagram ซึ่งถูกแบ่งส่วน(fragment) จะถูกส่งต่อไปยังมอดูล wlan ซึ่งทำหน้าที่เป็นส่วนเชื่อมต่อกับระบบเครือข่ายทำหน้าที่ส่งผ่านข้อมูลต่อไป



รูปที่ 4.17 การส่ง IPDatagram ไปยังมอดูล wlan

- เมื่อข้อมูลมาถึงมอดูล wlan แล้วก็จะทำการส่งข้อมูลออกไปยังระบบเครือข่ายโดยมีปลายทางคือลูกข่าย



รูปที่ 4.18 การส่งข้อมูล StrmPk ไปยังลูกข่าย

- และเมื่อส่งไปถึงปลายทางแล้วจึงทำการส่งส่วนต่อไปเรื่อยๆ จนกว่าจะครบ หรือตามเวลาที่ต้องการจำลอง

จากการจำลองจะได้เห็นการทำงานของโปรโตคอล UDP ที่จะมีความรวดเร็ว โดยไม่สนใจหากมีส่วนของข้อมูลหรือสูญหายจะทำการส่งข้อมูลต่อไปเรื่อยๆ จนครบ จึงมีความเหมาะสมกับการทำงานแบบวิดีโอสตรีมมิ่ง โดยการทำงานของโปรโตคอลจะดูได้จากภาพข้างต้น

4.7 ผลการทดลองและการวิเคราะห์ผลการทดลอง

การวิเคราะห์ผลการทดลองนั้นจะเริ่มจากทดสอบการส่งข้อมูลด้วยอัตราการส่งข้อมูลที่ความเร็ว 11 เมกะบิตต่อวินาที และสร้างแบบจำลองที่ไม่มีการจางหายเข้ามาเกี่ยวข้องก่อน จากนั้นจะนำผลจากการจางหายภายในช่องสัญญาณแบบ Nakagami-m ระดับต่างๆ กัน มาวิเคราะห์ผลการทดลองต่อๆ ไป โดยทำการทดลองเพื่อทดสอบขนาดชุดข้อมูล ที่ใช้งานได้ดีที่สุดในแต่ละสภาพแวดล้อมโดยสภาพแวดล้อมจำลองโดยการที่มีระดับสัญญาณรบกวนปกติ (Thermal Noise) และการจางหาย (Fading Channel) ต่างๆ กัน โดยเริ่มทดลองขนาดชุดข้อมูลที่ต่างๆ กันในช่องสัญญาณที่ไม่ได้รับผลกระทบจากการจางหาย แล้วจึงทดลองเพิ่มระกับการจางหาย แล้วจึงทดลองในอัตราการส่งข้อมูลที่ความเร็วต่างๆ กัน

4.7.1 การทดลองเมื่อมีอัตราการส่งข้อมูลที่ 11 เมกะบิตต่อวินาที

เริ่มทำกำหนดอัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที และทดลองที่ขนาดของชุดข้อมูล 500, 1000, 1500, 2000 ไบต์ ตามลำดับ

4.7.1.1 ส่วนที่ไม่มีผลกระทบจากการจางหาย

ทำการทดลองครั้งแรกโดยไม่ได้แก้ไขส่วนใดๆภายใน OMNeT++ จึงไม่ได้รับผลกระทบจากการจางหายที่เราเพิ่มเข้าไป

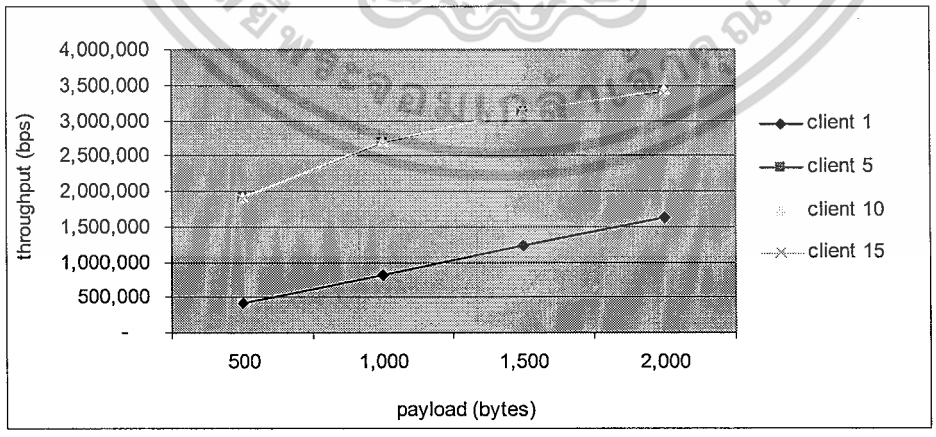
4.7.1.1.1 ผลการทดลอง

ตารางที่ 4.4 ตารางผลการทดลองแบบไม่มีผลกระทบจากการจางหายที่ความเร็ว 11 เมกะบิตต่อวินาที

จำนวนเครื่อง\ขนาดชุดข้อมูล	500	1,000	1,500	2,000
1	422,451	822,479	1,222,496	1,622,501
5	1,906,508	2,672,113	3,120,937	3,406,201
10	1,902,994	2,692,881	3,131,798	3,436,429
15	1,891,225	2,667,734	3,113,825	3,379,362

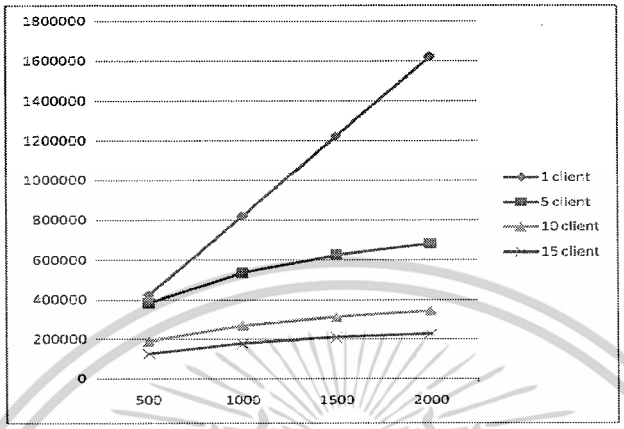
4.7.1.1.2 วิเคราะห์ผลการทดลอง

จากกราฟที่แสดงผลรวมของปริมาณงานในการทดลองที่มีขนาดชุดข้อมูลต่างๆ กัน จะสังเกตได้ว่า ในขณะที่ให้บริการลูกข่ายเพียงคนเดียวนั้น จะให้ค่าปริมาณงานที่ต่ำเนื่องจาก Access Point ยังทำงานไม่เต็มประสิทธิภาพ จึงมีค่าปริมาณงานที่ต่ำเมื่อเทียบกับการให้บริการลูกข่ายจำนวนที่มากกว่า และจะเห็นได้ว่าการให้บริการลูกข่ายตั้งแต่ 5 คนเป็นต้นไปนั้นจะให้ค่าปริมาณงานที่ไม่แตกต่างกันจึงเข้าใจได้ว่าระบบเครือข่ายได้ทำงานเต็มประสิทธิภาพแล้ว



รูปที่ 4.19 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลในการทดลองแบบไม่มีการจางหายที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

ส่วนกราฟแสดงค่าเฉลี่ยปริมาณงานกับขนาดชุดข้อมูลต่างๆ กันจะสังเกตได้ว่าหากมีการเพิ่มขึ้นของลูกข่ายจะทำให้ค่าปริมาณงานเฉลี่ยต่อ 1 ลูกข่ายจะลดลงเรื่อยๆ ไม่ว่าจะมีขนาดของชุดข้อมูลเท่าใดก็ตาม



รูปที่ 4.20 กราฟแสดงค่าปริมาณงานเฉลี่ยต่อขนาดชุดข้อมูลในการทดลองแบบไม่มีการจางหายที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

จากทั้งสองกราฟนี้จะเห็นว่าค่าขนาดชุดของข้อมูลยิ่งมีขนาดใหญ่แล้วจะทำให้ค่าปริมาณงานดีขึ้นซึ่งแสดงว่าในการใช้งานระบบเครือข่ายแบบไร้สาย ที่ไม่มีผลกระทบจากการจางหายนั้นควรที่จะใช้ค่าขนาดชุดของข้อมูลที่ใหญ่จะดีกว่าขนาดชุดข้อมูลที่เล็ก

4.7.1.2 ส่วนที่ได้รับผลกระทบจากการจางหาย

ในการทดลองที่กำหนดค่าเบี่ยงเบนมาตรฐานของสัญญาณรบกวนแบบทั่วไปเท่ากับ 5 จะเห็นผลกระทบจากการจางหายได้จากการเปรียบเทียบผลรวมปริมาณงาน ในขณะที่ค่ารบกวนจากการจางหายที่ -90 dBm และ -110 dBm

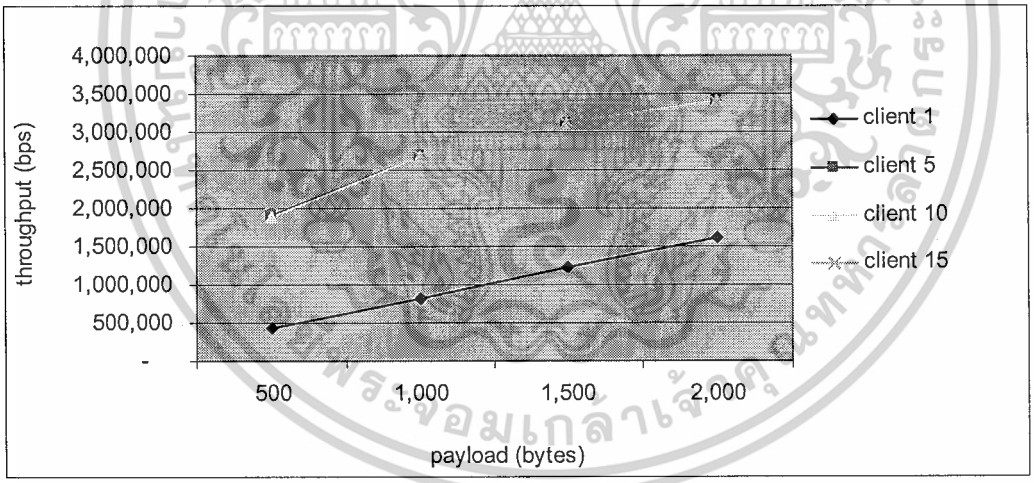
ตารางที่ 4.5 ค่าปริมาณงานเฉลี่ยแต่ละจำนวนลูกข่ายที่ได้รับผลจาก Nakagami-m ที่มีค่าเท่ากับ -110 dBm ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

	500.0	1,000.0	1,500.0	2,000.0
1	422,448.0	822,474.0	1,222,488.0	1,622,491.0
5	1,916,706.0	2,690,744.0	3,104,361.0	3,403,316.0
10	1,884,054.0	2,678,060.0	3,121,809.0	3,398,445.0
15	2,067,227.0	2,675,418.0	3,107,371.0	3,394,933.0

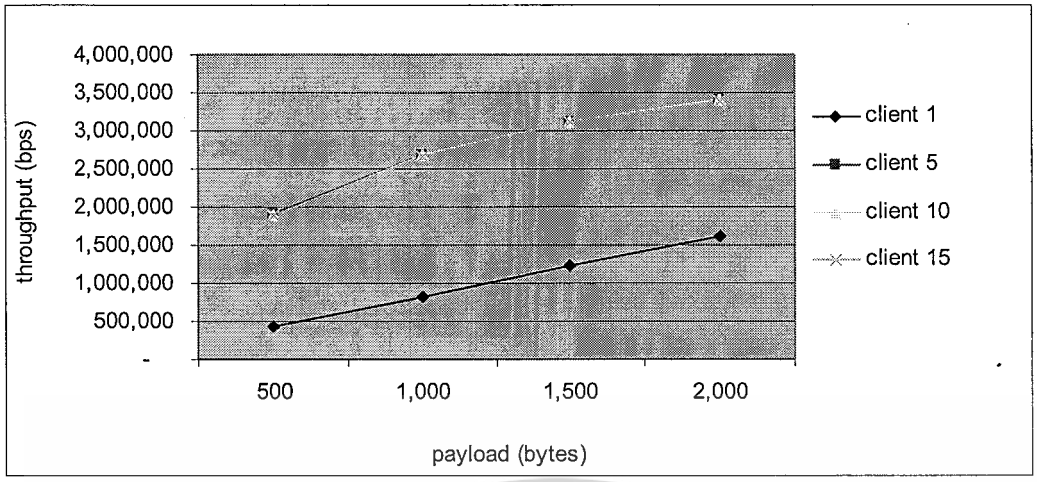
ตารางที่ 4.6 ค่าปริมาณงานเฉลี่ยแต่ละจำนวนลูกข่ายที่ได้รับผลจาก Nakagami-m ที่มีค่าเท่ากับ -90 dBm ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

	500.0	1,000.0	1,500.0	2,000.0
1	422,448.0	822,474.0	1,222,488.0	1,622,491.0
5	1,915,806.0	2,690,744.0	3,104,461.0	3,403,316.0
10	1,884,054.0	2,678,060.0	3,121,809.0	3,398,445.0
15	1,883,627.0	2,650,255.0	3,110,371.0	3,394,933.0

จากผลการทดลองเมื่อนำข้อมูลมาเขียนอยู่ในรูปกราฟจะเห็นได้ว่าที่ขนาดชุดข้อมูลที่ 250 และ 500 ไบต์นั้นจะได้รับผลกระทบจากการจางหายแบบ Nakagami-m ที่เพิ่มขึ้นจาก -110 dBm เป็น -90 dBm ส่งผลให้ค่าปริมาณงานลดลงเมื่อมีจำนวนลูกข่ายใช้ระบบมากกว่า 5 คนขึ้นไป ส่วนที่ขนาดชุดข้อมูล 1000 ไบต์ได้รับผลกระทบจากการจางหายเมื่อมีจำนวนลูกข่ายตั้งแต่ 10 คนขึ้นไป และที่ขนาดชุดข้อมูล 1500 และ 2000 ไบต์จะได้รับผลกระทบการการจางหายทำให้ค่าปริมาณงานมีค่าที่ไม่แตกต่างกัน

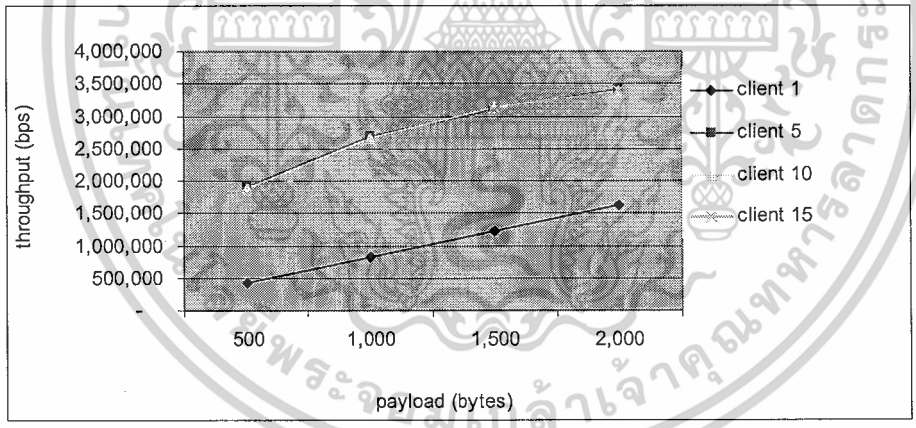


รูปที่ 4.21 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

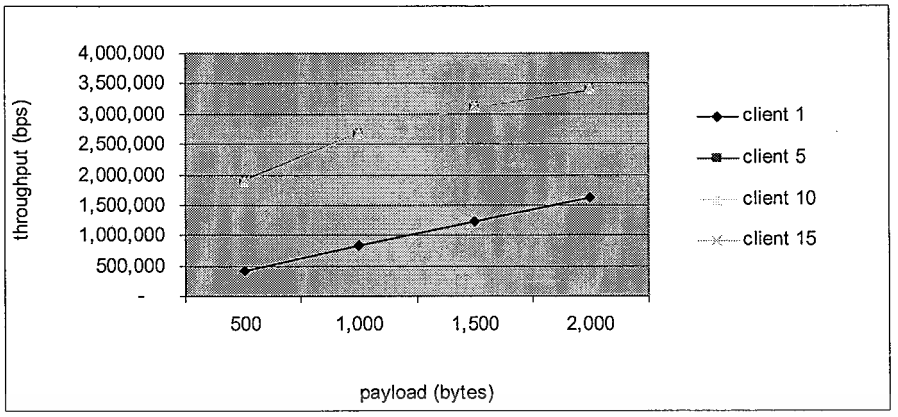


รูปที่ 4.22 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

เมื่อเปลี่ยนแปลงค่าเบี่ยงเบนมาตรฐานเป็น 7 เพื่อเพิ่มความแปรปรวนของระดับสัญญาณรบกวนพื้นฐานให้มีค่ามากขึ้น จะเห็นค่าผลรวมของปริมาณงานของขนาดชุดข้อมูล 250 , 500 และ 1000 ไบต์จะให้ค่าผลรวมของปริมาณงานที่น้อยกว่า เมื่อมีการใช้งานจากลูกข่ายตั้งแต่ 5 คนขึ้นไป

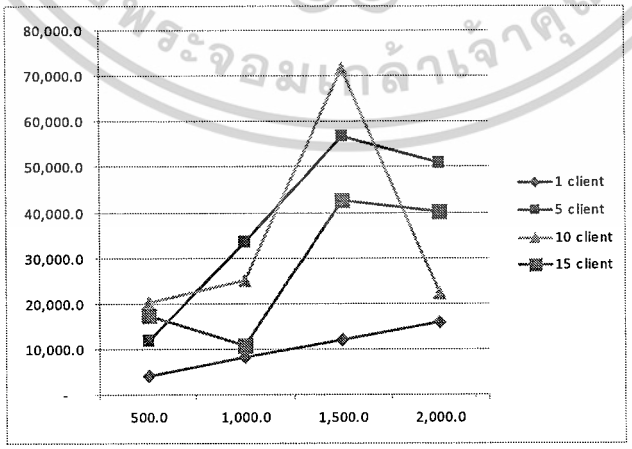


รูปที่ 4.23 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 7 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

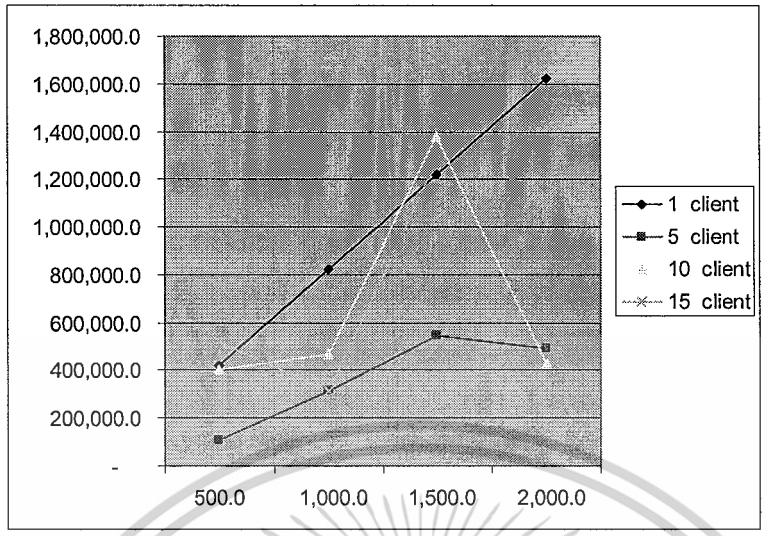


รูปที่ 4.24 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 7 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

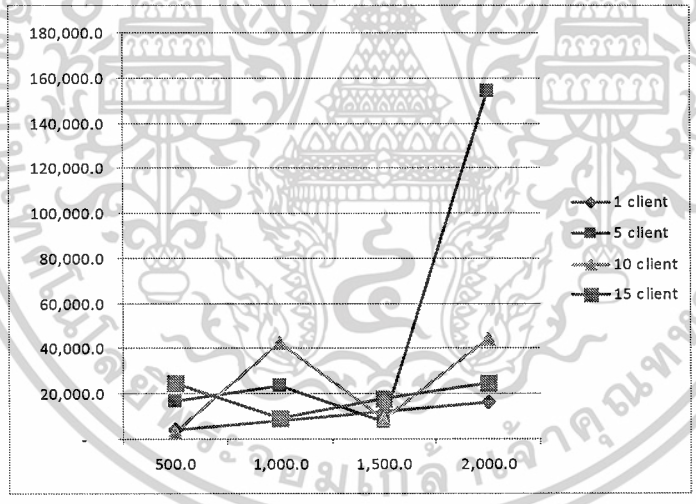
เมื่อเปลี่ยนแปลงค่าเบี่ยงเบนมาตรฐานของสัญญาณรบกวนเท่ากับ 10 จะทำให้มีค่าความแปรปรวนที่สูง และหากดูจากภาพการใช้งานขนาดชุดข้อมูล 1500 ไบต์และค่ารบกวนจากการจางหายที่มีความแรงของสัญญาณ -110 dBm จะให้ผลลัพธ์ทั้งค่าผลรวมและค่าเฉลี่ยของปริมาณงานที่ดีกว่าทั้งหมด แต่หากว่าค่ารบกวนการจางหาย -90 dBm แล้วจะเห็นได้ว่ามีการสลับของค่าปริมาณงานที่สูงและต่ำ ในขณะที่ขนาดชุดข้อมูลต่างๆ กัน ซึ่งจะเห็นได้จากทั้งสองกราฟ ว่าในขณะที่ให้บริการตั้งแต่ลูกข่าย 5 คนเป็นต้นไปนั้น จะเกิดความแปรปรวนของค่าทั้งผลรวมและค่าเฉลี่ยของปริมาณงานซึ่งทำให้กราฟมีลักษณะสลับกันขึ้นลง ซึ่งความแปรปรวนเกิดจากการสุ่มค่าซึ่งเมื่อมีการกระจายที่สูงทำให้ ช่วงของการสุ่มมากขึ้นทำให้ค่าของสัญญาณรบกวนมีค่าที่สูงหรือต่ำสลับกัน ซึ่งช่วงที่กราฟตกลงนั้นมาจากการสุ่มค่าที่เกิดอยู่ในช่วงที่มีค่าสูงหรือต่ำมากๆ ซึ่งจะส่งผลให้แก่ค่าปริมาณงานเป็นอย่างมากได้



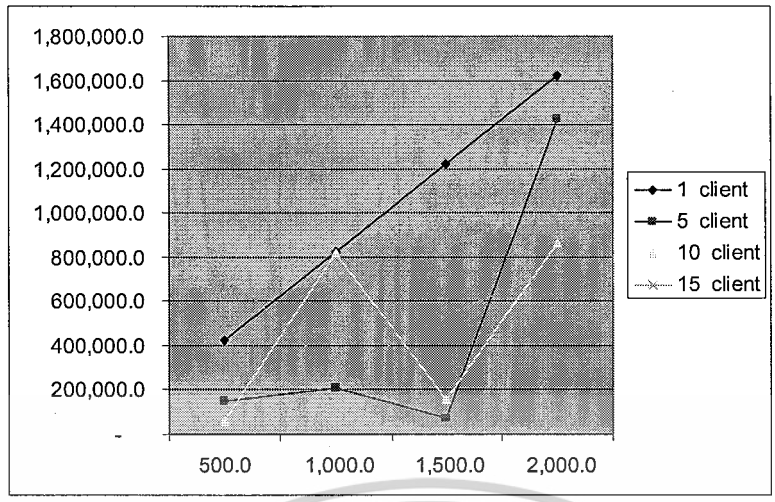
รูปที่ 4.25 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 10 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที



รูปที่ 4.26 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 10 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที



รูปที่ 4.27 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 10 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

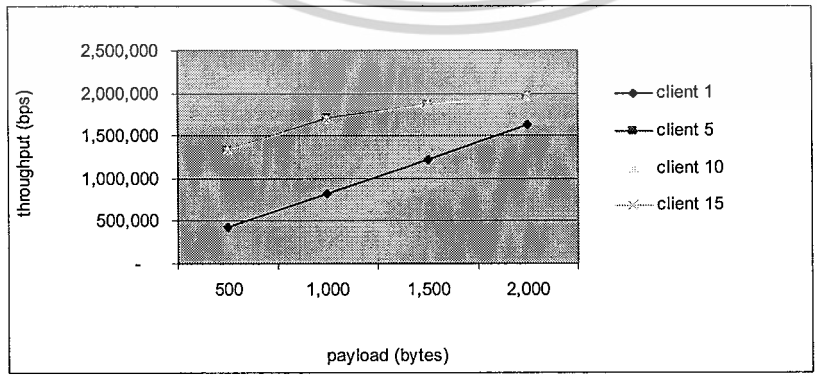


รูปที่ 4.28 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 10 ที่อัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที

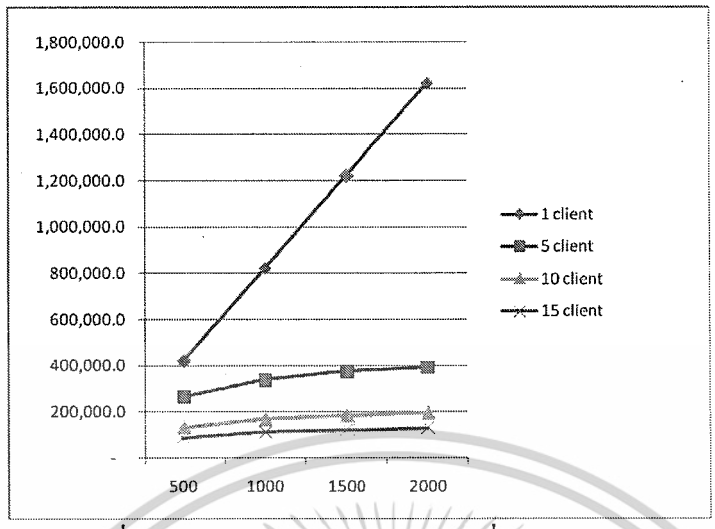
4.7.2 การทดลองเมื่อมีอัตราการส่งข้อมูลที่ 5.5 เมกะบิตต่อวินาที

4.7.2.1 ส่วนที่ไม่มีผลกระทบจากการจางหาย

ในส่วนนี้ในการทดลองเมื่อมีอัตราการส่งข้อมูลที่ต่ำลง จะเห็นได้ว่ามีค่าปริมาณงานที่ลดลงเมื่อเปรียบเทียบกับอัตราการส่งข้อมูลที่มากกว่า และลักษณะของกราฟแสดงให้เห็นคล้ายคลึงกับอัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที คือเมื่อมีขนาดของชุดข้อมูลที่ใหญ่ขึ้นตามลำดับ 500, 1000, 1500 และ 2000 ไบต์จะเห็นได้ว่ามีค่าปริมาณงานทั้งเฉลี่ย และผลรวมที่มากขึ้นด้วย แต่เมื่อเพิ่มจำนวนลูกข่ายเข้าไปในระบบตามลำดับ 1, 5, 10 และ 15 เครื่องจะมีผลค่าเฉลี่ยของปริมาณงานที่ลดลงตามลำดับ และในอัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที ค่าปริมาณงานก็จะเพิ่มตามลำดับลูกข่ายจน มีค่าไม่แตกต่างกันตั้งแต่ลูกข่าย 5 เครื่องขึ้นไป ซึ่งจะมีลักษณะเดียวกันกับอัตราการส่งข้อมูล 11 เมกะบิตต่อวินาที



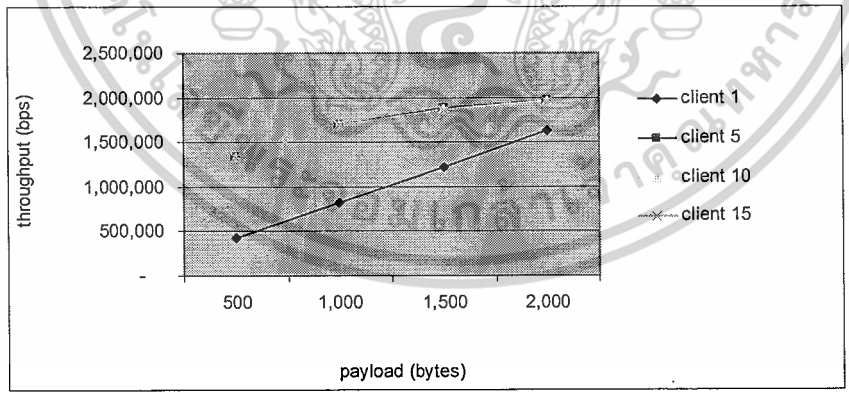
รูปที่ 4.29 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที



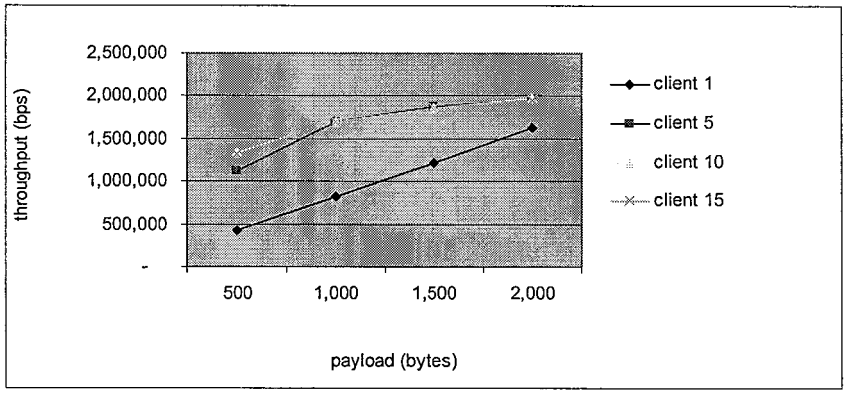
รูปที่ 4.30 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที

4.7.2.2 ส่วนที่ได้รับผลกระทบจากการจางหาย

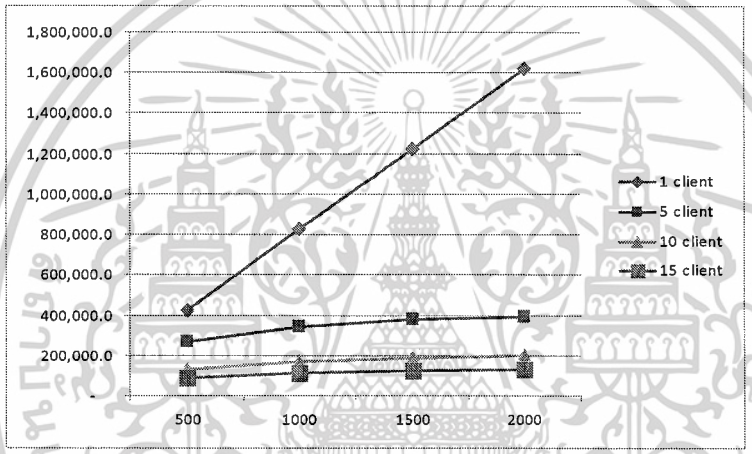
ในกรณีเปรียบเทียบการทดลองที่ได้รับผลกระทบจากสัญญาณรบกวนที่มีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 และ ผลกระทบจากการจางหาย -110 dBm กับ -90 dBm เมื่อทำการเปรียบเทียบค่าผลรวมของปริมาณงาน และค่าเฉลี่ยของปริมาณงานแล้วจะเห็นได้ว่าไม่มีความแตกต่างกันอย่างเด่นชัด เนื่องจากยังไม่มีสัญญาณรบกวนมากเพียงพอ



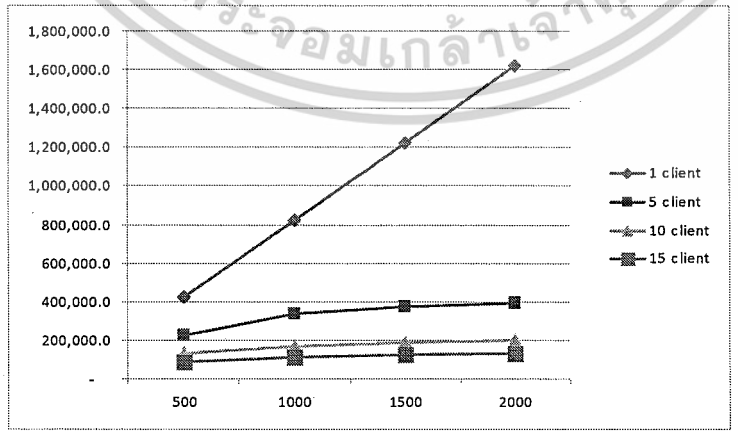
รูปที่ 4.31 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที



รูปที่ 4.32 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที

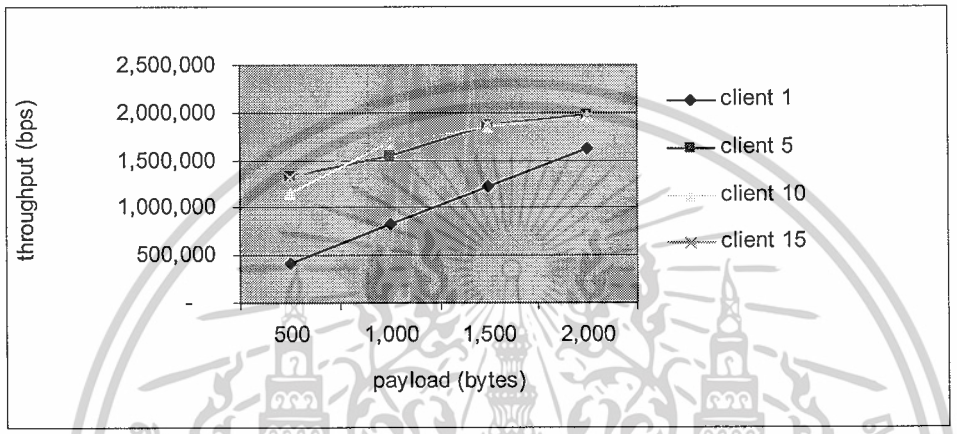


รูปที่ 4.33 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที

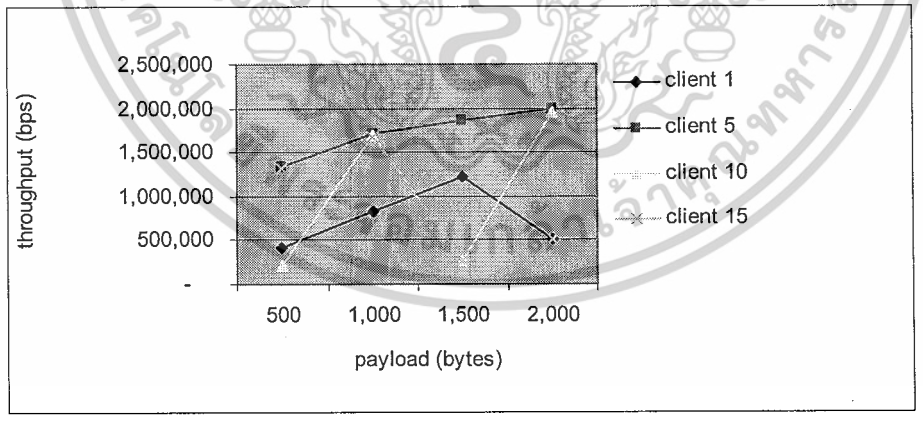


รูปที่ 4.34 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 5 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที

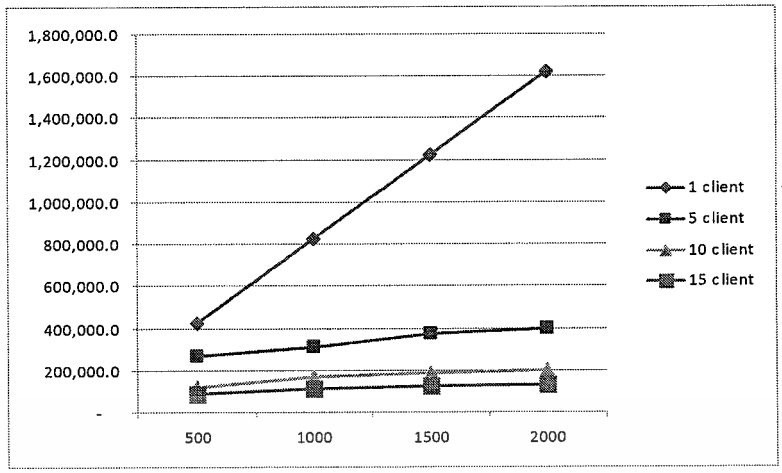
ในกรณีเปรียบเทียบการทดลองที่ได้รับผลกระทบจากสัญญาณรบกวนที่มีค่าเบี่ยงเบนมาตรฐาน 7 และ ผลกระทบจากการจางหาย -110 dBm กับ -90 dBm เมื่อทำการเปรียบเทียบค่าผลรวมของปริมาณงาน และค่าเฉลี่ยของปริมาณงานแล้วจะเห็นได้ว่าการที่เพิ่มค่าการจางหายจาก -110 dBm เป็น -90 dBmทำให้เกิดความแปรปรวนของข้อมูลโดยในค่าขนาดของข้อมูล 500 และ 1500โดยที่ปริมาณลูกข่าย 10 และ15 เครื่อง จะมีค่าผลรวมและค่าเฉลี่ย ที่ลดลงอย่างเห็นได้ชัดเมื่อเปรียบเทียบกับ การทดลองกรณีที่มีค่าการจางหาย -110 dBm



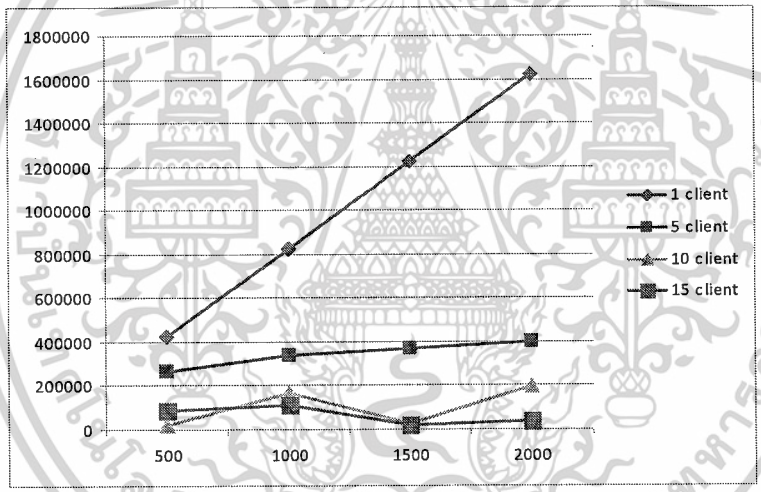
รูปที่ 4.35 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 7 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที



รูปที่ 4.36 กราฟแสดงผลรวมปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 7 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที



รูปที่ 4.37 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -110 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 7 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที



รูปที่ 4.38 กราฟแสดงค่าเฉลี่ยปริมาณงานต่อขนาดชุดข้อมูลขณะที่มีผลกระทบจากการจางหาย -90 dBm และมีค่าเบี่ยงเบนมาตรฐานเท่ากับ 7 ที่อัตราการส่งข้อมูล 5.5 เมกะบิตต่อวินาที

4.8 สรุปผลการทดลอง

จากการทดลองทั้งหมดนั้นจะสรุปผลได้ว่า การกำหนดค่าขนาดของชุดข้อมูล (payload) นั้นจะส่งผลต่อค่าปริมาณงาน โดยหากมีการกำหนดค่าขนาดของชุดข้อมูลที่มีขนาดใหญ่แล้วจะสามารถทำให้ค่าปริมาณงานนั้นมีค่าที่สูงขึ้นได้ แต่หากว่ามีสภาพแวดล้อมที่เปลี่ยนแปลงไปเช่นค่าสัญญาณรบกวน หรือการจางหาย และการปรับเปลี่ยนอัตราการส่งข้อมูล อาจทำให้ต้องทำการปรับค่าขนาดของชุดข้อมูลบ้างเพื่อให้ได้ ค่าปริมาณงานที่มีประสิทธิภาพสูงสุด และเมื่อมีการเพิ่มจำนวนของผู้ใช้บริการในระบบจะส่งผลให้ค่าผลรวมของปริมาณงานนั้นดีขึ้นจนเต็มประสิทธิภาพ

ของเครือข่าย แต่เมื่อดูค่าเฉลี่ยของปริมาณงานแล้วจะมีแนวโน้มที่ลดลง ซึ่งเกิดจากการที่ลูกข่ายบางตัวไม่สามารถใช้บริการได้ หรือใช้บริการได้น้อยมาก ซึ่งการปรับค่าขนาดของชุดของข้อมูลต้องคำนึงถึงปัจจัยต่างๆ เหล่านี้ด้วย เพื่อให้ได้ระบบเครือข่ายที่มีประสิทธิภาพสูงสุดในสภาพแวดล้อมที่แตกต่างกัน

บทที่ 5
สรุปผลการวิจัยและเสนอแนะ

ปัจจุบันการติดต่อสื่อสารในชีวิตประจำวันนั้นเป็นสิ่งที่มีความจำเป็นอย่างมากยิ่งขึ้น และได้มีการพัฒนาเทคโนโลยีที่ใช้ในการสื่อสารเรื่อยมาจนเป็นการสื่อสารแบบไร้สาย ซึ่งในระบบเครือข่ายไร้สายนั้นมีบทบาทเข้ามาในชีวิตประจำวันของชีวิตคนเรามากยิ่งขึ้น โดยจะสังเกตได้จากการสื่อสารไร้สายในลักษณะต่างๆเช่น โทรศัพท์มือถือ, อินเตอร์เน็ตไร้สาย เป็นต้น นับได้ว่าเทคโนโลยีการสื่อสารแบบไร้สารนั้นเป็นสิ่งที่มีความสำคัญมากในการพัฒนาธุรกิจต่างๆ และกำลังกลายเป็นสิ่งจำเป็นในชีวิตประจำวัน ซึ่งในการพัฒนาระบบเครือข่ายไร้สายให้ได้มีประสิทธิภาพสูงขึ้นนั้นจำเป็นที่จะต้องใช้เทคโนโลยีและเงินลงทุนจำนวนมาก

การจำลองระบบเครือข่ายแบบไร้สาย จึงเป็นทางเลือกทางหนึ่งที่สามารถใช้เป็นเครื่องมือในการวิเคราะห์ระบบเครือข่ายแบบไร้สาย ซึ่งข้อดีในการจำลองนั้นคือการที่สามารถทำการทดลองทฤษฎีต่างๆ ที่จะนำมาใช้ในการพัฒนาระบบเครือข่ายแบบไร้สาย ก่อนที่จะทำการทดลองกับอุปกรณ์จริง ซึ่งการจำลองนั้นมีความประหยัดทั้งด้านเวลาและเงินทุนกว่ามาก แต่การจำลองก็ยังมีข้อจำกัดของแบบจำลองต่างๆ โดยการสร้างแบบจำลองมีความสมควรที่จะต้องสามารถทำการจำลองแทนเทคโนโลยี อุปกรณ์ และเครือข่ายจริงได้โดยมีความใกล้เคียงกับของจริงมากที่สุด และควรมีความยืดหยุ่นให้สามารถปรับแต่งได้อย่างหลากหลายเพื่อให้มีความสะดวกต่อการทดลองต่างๆ

ในปัจจุบันโปรแกรมจำลองได้มีหลากหลาย ซึ่งตัวโปรแกรม OMNeT++ เป็นหนึ่งในโปรแกรมจำลองซึ่งมีกรอบการทำงานที่สนับสนุนการทำงานต่างๆ มากมาย อีกทั้งยังสนับสนุนการจำลองระบบเครือข่ายแบบไร้สาย และด้วยภาษา NED ซึ่งเป็นภาษาที่ใช้ในการสร้างแบบจำลอง มีความยืดหยุ่นโดยการสร้างเป็นมอดูลแล้วจึงทำการประกอบแต่ละมอดูลเข้าด้วยกัน ซึ่งมีประโยชน์อย่างมากในการทำงาน และมีเครื่องมือต่างๆ ที่ช่วยทำกราฟ หรือแสดงผลข้อมูล อีกทั้งยังมีส่วนต่อประสานกับผู้ใช้ที่ดีมาก และเนื่องจากยังไม่มีความแพร่หลายมากในประเทศไทยจึงควรที่จะต้องมีการศึกษาไว้เป็นทางเลือกในการใช้งาน

โปรแกรม OMNeT++ นั้นมีภาษา NED ที่ใช้ในการสร้างแบบจำลองและมีตัว ภาษา C++ ในการควบคุมการทำงาน อีกทั้งมีกรอบการทำงานที่ช่วยให้สามารถทำการจำลองแบบทั่วไปเพื่อการศึกษาได้ง่ายยิ่งขึ้น ซึ่งจำเป็นที่จะต้องมีพื้นฐานการใช้ภาษาระดับหนึ่ง รวมถึงความเข้าใจการทำงานต่างๆ และการกำหนดค่าต่างๆ เพื่อให้ได้ประสิทธิภาพสูงสุดของการจำลอง โดยการวิเคราะห์ข้อมูลมีโปรแกรม Plove ที่จะช่วยสนับสนุนการสร้างกราฟจากข้อมูลที่ได้ในการจำลอง โดยข้อมูลส่วนนี้อาจนำไปใช้โปรแกรมอื่นในการสร้างกราฟได้เช่น gnuplot เป็นต้น และโปรแกรม

scalar ที่ใช้ในการเปิดไฟล์ที่ทำการเก็บค่าพารามิเตอร์ ที่ถูกกำหนดไว้ให้ทำการเก็บในตัวภาษา C++ เพื่อที่จะนำข้อมูลมาใช้ในการวิเคราะห์ต่อไป

ในโครงงานนี้ได้นำเสนอความรู้พื้นฐานในการใช้งานโปรแกรม OMNeT++ โดยประกอบไปด้วยความรู้พื้นฐานของภาษา NED, กรอบการทำงาน INET, พื้นฐานวิธีการสร้างแบบจำลอง ปัจจัยที่มีผลต่อในการส่งข้อมูลไร้สายรวมถึงการแก้ไขตัวโปรแกรม OMNeT++ ให้สอดคล้องโดยการเพิ่มการจางหายในช่องสัญญาณ (fading channel) อีกทั้งยังทำการจำลองการทำงานแบบวีดีโอสตรีมมิ่ง เพื่อหาค่าขนาดของชุดข้อมูลที่จะทำให้เครือข่ายมีปริสิทธิภาพ ในสภาพแวดล้อมของสัญญารบกวนและการจางหายที่ต่างๆ กัน และนำผลที่ได้จากการจำลองมาวิเคราะห์

เนื้อหาของโครงงานนี้เป็นเพียงส่วนหนึ่งในการทดสอบตัวโปรแกรม OMNeT++ โดยแบบจำลองซึ่งมีภาษาที่ยืดหยุ่นทำให้มีความสามารถในการสร้างแบบจำลองที่มีการเพิ่มการทำงานอื่นๆเข้าไป หรือสร้างการทำงานที่สามารถนำมาใช้แทนที่การทำงานแบบเก่า ทั้งนี้ประสิทธิภาพการจำลองจะขึ้นอยู่กับแนวความคิด สมมติฐานของผู้ทำการจำลองเป็นสำคัญ โดยจะต้องออกแบบตัวแบบจำลอง และรูปแบบการจำลองให้ครอบคลุมตามความต้องการ และปัจจัยที่เข้ามามีผลกระทบต่อระบบที่ต้องการจะศึกษา โดยตัวโปรแกรม OMNeT++ เป็นเพียงเครื่องมือที่จะช่วยให้สามารถสร้างแบบจำลองได้ และทำการจำลองเพื่อผลลัพธ์เท่านั้น โดยผลลัพธ์ดังกล่าวจะถูกวิเคราะห์โดยผู้จำลอง เพื่อให้ได้เกิดประโยชน์สูงสุดในการพัฒนาระบบสื่อสารแบบไร้สายในขั้นต่อไป

ซึ่งในโครงงานฉบับต่อไปจะกล่าวถึงการแก้ไขปัญหาในการจัดการกับข้อมูลเวคเตอร์ และทำการปรับแต่งมอดูลให้มีการเก็บค่าที่จะนำมาใช้ในการวิเคราะห์ผลการทดลองให้ครบถ้วน และทดลองการเปลี่ยนการทำงานและเปรียบเทียบผลที่ได้จากการคำนวณทางคณิตศาสตร์กับผลการทดลองที่ได้จริง

บรรณานุกรม

Andras Varga. 2005. **OMNeT++ Discrete Event Simulation System Version 3.2 User Manual.** N.p. (no place of publishing).

INET Framework Documentation and Tutorials. [Online]. Available : http://www.omnetpp.org/staticpages/index.php?page=20041019113420757

OMNeT++ community site. [Online] Available: http://www.omnetpp.org

OMNeT++ Documentation and Tutorials. [Online] Available : http://www.omnetpp.org/staticpages/index.php?page=20041019105346896

Simon Marvin K.. **Digital communication over fading channels.** Hoboken, N.J.: John Wiley, 2005.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายอานนท์ แซ่เบ๊
วัน เดือน ปีเกิด	4 มกราคม 2528 จังหวัดกรุงเทพมหานคร
ที่อยู่	51/41 หมู่ 9 ถ.สุขุมวิท แขวง บางนา เขต บางนา กรุงเทพฯ 10260
อีเมล	zigzagzang@hotmail.com
ประวัติการศึกษา	
2550	วิทยาศาสตร์บัณฑิต คณะเทคโนโลยีสารสนเทศ สาขาเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อ-นามสกุล	นายเดชพงษ์ บุญญฤทธิ์
วัน เดือน ปีเกิด	6 พฤษภาคม 2529 จังหวัดกรุงเทพมหานคร
ที่อยู่	99 หมู่ 3 หมู่บ้านสัมมากร ซอย 5 ถนนรามคำแหง แขวงสะพานสูง เขตสะพานสูง กรุงเทพมหานคร 10260 โทรศัพท์ 02-398-1434
อีเมล	gapsoi5@hotmail.com
ประวัติการศึกษา	
2550	วิทยาศาสตร์บัณฑิต คณะเทคโนโลยีสารสนเทศ สาขาเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคผนวก ก.
ขั้นตอนการติดตั้งโปรแกรม OMNeT++

ขั้นตอนการติดตั้งโปรแกรม OMNeT++ เวอร์ชัน 3.3 และตัวอัพเดตไลบรารีเพื่อทำงานกับ Visual C++ 2005 บนระบบปฏิบัติการ window XP

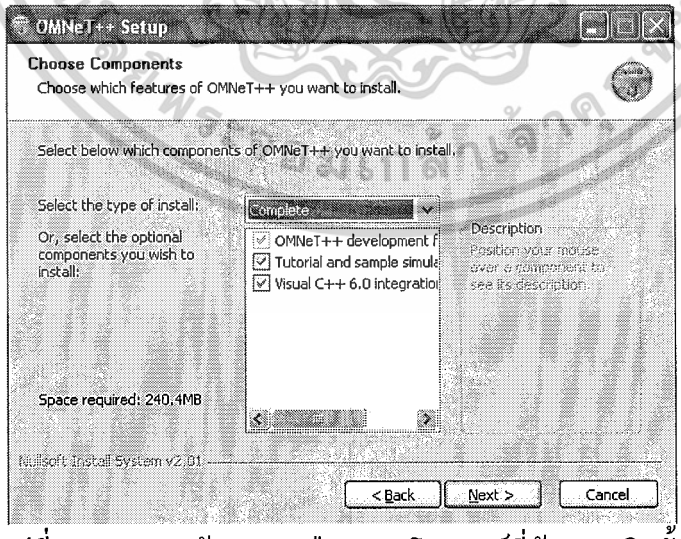
ขั้นตอนที่ 1 ในการคอมไพล์ OMNeT++ จำเป็นที่ต้องใช้ตัวคอมไพล์เลอร์ โดยสามารถใช้งานตัวฟรีเวอร์ชั่น IDE ของ Microsoft คือ Visual C++ Express Edition โดยสามารถที่จะทำการดาว์นโหลดจากเว็บไซต์ http://www.microsoft.com/express/download/

ขั้นตอนที่ 2 ทำการลงตัว Platform SDK ที่บรรจุตัวไฟล์ header ที่สำคัญ โดยดาวน์โหลดได้จากเว็บไซต์ http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en

ขั้นตอนที่ 3 (เป็นทางเลือกไม่จำเป็นต้องทำ) เพื่อที่จะใช้ตัวสนับสนุนรูปภาพใน Neddoc จำเป็นที่จะต้องทำการลง Ghostscript ก่อนที่จะทำการลงโปรแกรม OMNeT++ โดยสามารถที่จะสามารถหาได้จากเว็บไซต์ http://pages.cs.wisc.edu/~ghost/doc/AFPL/get853.htm

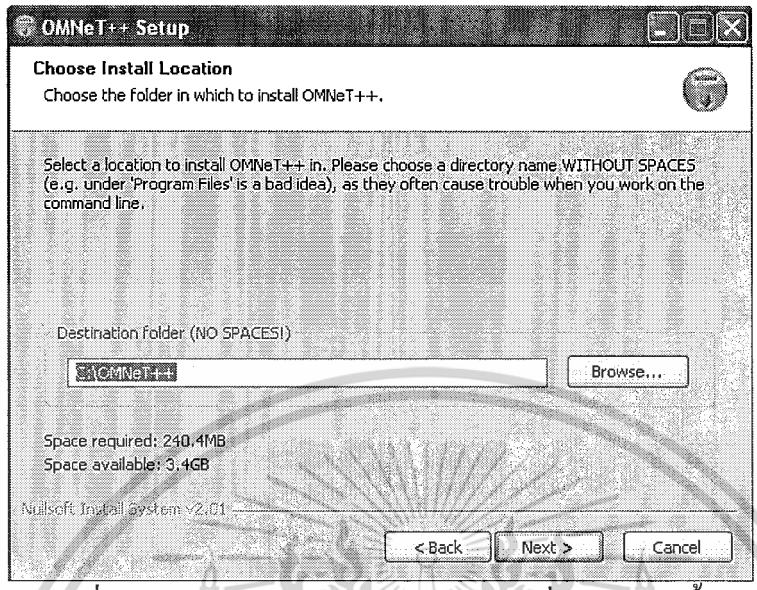
ขั้นตอนที่ 4 ดาวน์โหลดตัวโปรแกรม OMNeT++ เวอร์ชั่น 3.3 win32 binay(exe) ซึ่งหาได้จากเว็บไซต์ http://www.omnetpp.org/filemgmt/viewcat.php?cid=2

ขั้นตอนที่ 5 เริ่มทำการลงโปรแกรม OMNeT++ ทำการอ่านข้อมูลและข้อตกลงจากนั้นกดปุ่ม next จากนั้นทำการเลือกส่วนประกอบที่ต้องการติดตั้ง



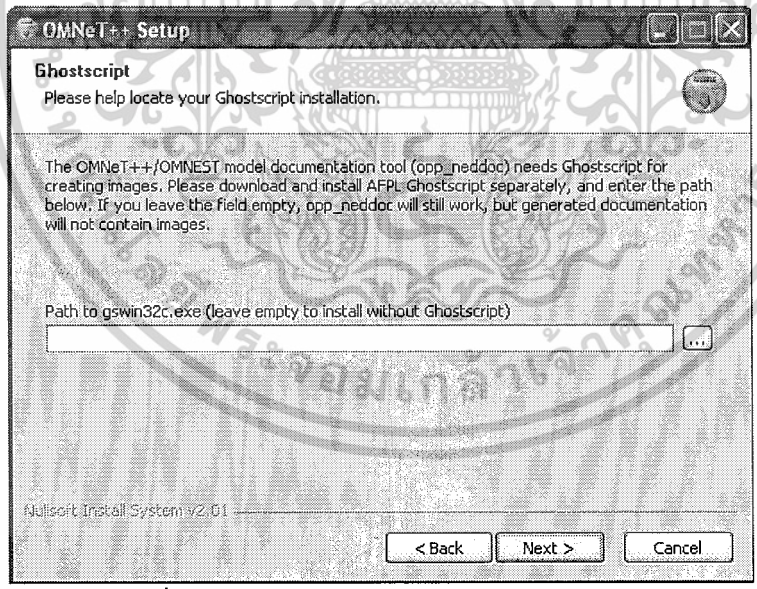
รูปที่ ก.1 แสดงหน้าจอการเลือกคอมโพเนนท์ที่ต้องการติดตั้ง

ขั้นตอนที่ 6 เลือกไดเร็กทอรีที่ต้องการติดตั้ง



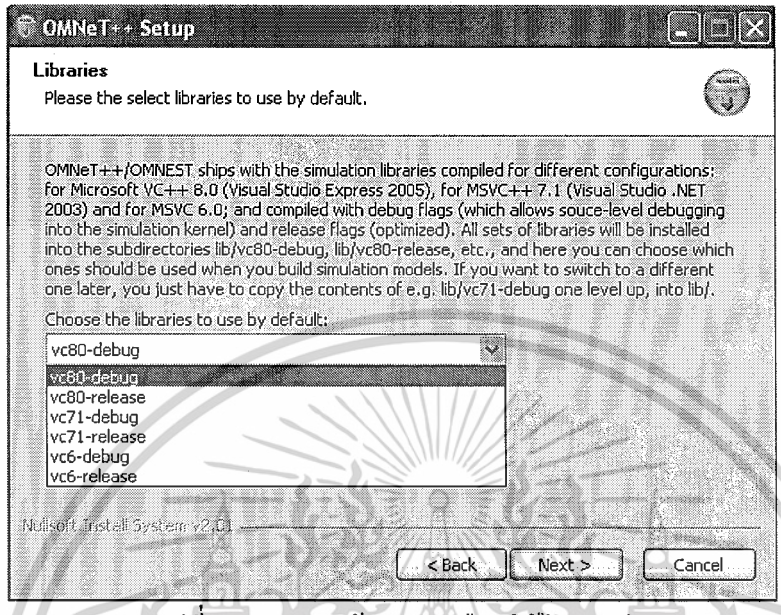
รูปที่ ก.2 แสดงหน้าจอการเลือกไดเร็กทอรีที่ต้องการติดตั้ง

ขั้นตอนที่ 7 ทำการใส่ค่า Ghostscript (ถ้ามี ไม่จำเป็นต้องใส่)



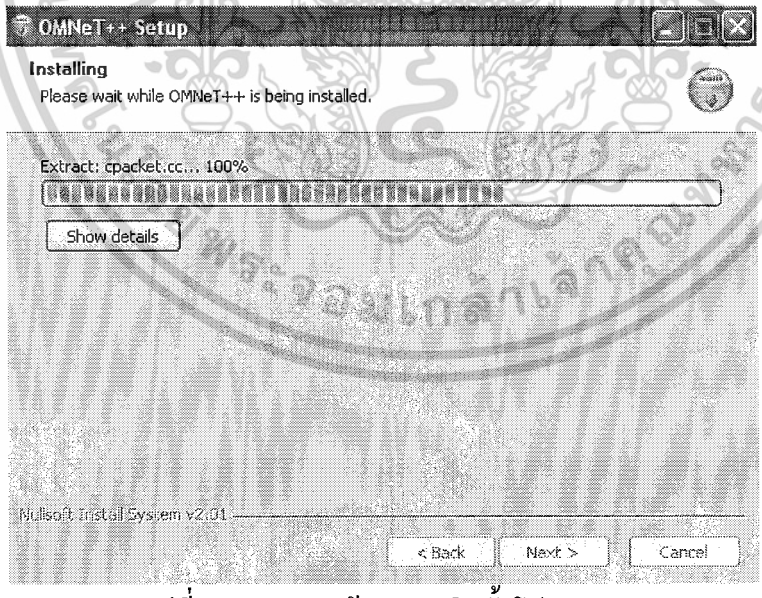
รูปที่ ก.3 แสดงหน้าจอการเลือกใช้ ghostscript

ขั้นตอนที่ 8 เลือกค่าไลบรารีที่ใช้งานโดยหากใช้ Visual Studio 2005 เลือกตัว vc80 โดยหากเลือกแบบ debug จะสามรถทำการคอมไพล์กับ debug flag ได้



รูปที่ ก.4 แสดงหน้าจอการเลือกใช้ไลบรารี

ขั้นตอนที่ 9 จากนั้นเป็นการเริ่มทำการติดตั้งโปรแกรม



รูปที่ ก.5 แสดงหน้าจอการติดตั้งโปรแกรม

ขั้นตอนที่ 10 ทำการติดตั้งตัวอัพเดตไลบรารีเพื่อทำงานร่วมกับ Visual Studio 2005 โดยสามารถดาวน์โหลดได้จากเว็บไซต์ http://www.omnetpp.org/filemgmt/viewcat.php?cid=2 โดยทำการโหลดตัว Update for OMNeT++ 3.3 Win32: Libs for Visual C++ 2005(Express)

ขั้นตอนที่ 11 ทำการสำเนาข้อมูลไฟล์ไปยังโปรแกรม OMNeT++ ที่ได้ทำการติดตั้งแล้ว (โดยนำไฟล์จาก lib\vc80-release และ lib\vc80-debug)

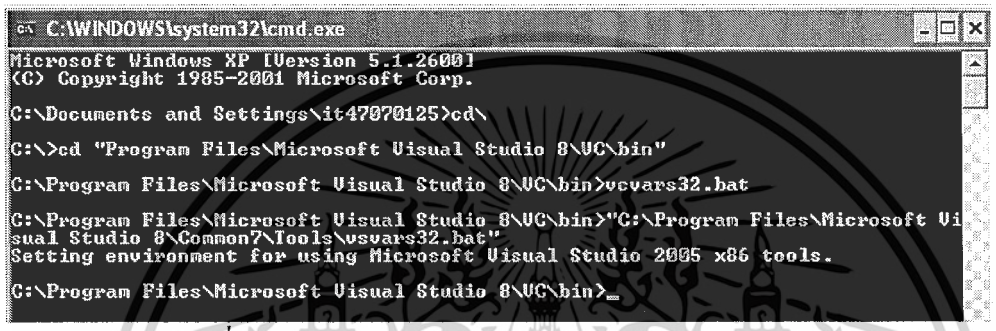
ขั้นตอนที่ 12 ทำการแก้ไขไฟล์ configuser.vc โดยทำการเพิ่ม /MTd ที่ CFLAGS_DEBUG, และ /MT ที่ CFLAGS_RELEASE, จากนั้นจึงทำการคอมไฟล์อีกครั้งโดยซ้ำคำสั่ง opp_nmakemake เพื่อสร้าง makefiles

ภาคผนวก ข.

ขั้นตอนการคอมไพล์

ขั้นตอนการคอมไพล์ไฟล์ในโปรแกรม OMNeT++ เพื่อใช้ในการแก้ไขมอดูล

ขั้นตอนที่ 1 ในทุกครั้งของการเปิดคอมพิวเตอร์ต้องทำการ run ไฟล์ vcvars32.bat ซึ่งอยู่ในไดเรกทอรี C:\Program Files\Microsoft Visual Studio 8\VC\bin

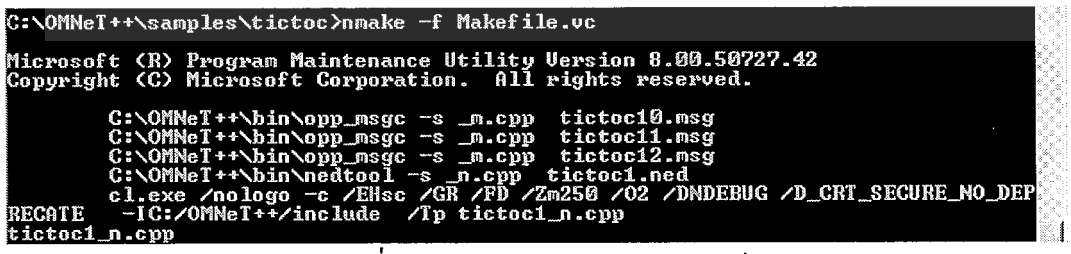


รูปที่ ข.1 แสดงหน้าจอการเรียกใช้งานไฟล์ vcvars32.bat

ขั้นตอนที่ 2 ไปยังไดเร็กทอรีที่ต้องการคอมไพล์ และพิมคำสั่ง opp_nmake ซ้ำเพื่อเป็นการสร้าง makefile

รูปที่ ข.2 แสดงหน้าจอการสร้าง Makefile

ขั้นตอนที่ 3 ใช้คำสั่ง nmake –f Makefile.vc จะเป็นการทำการคอมไพล์ไฟล์



รูปที่ ข.3 แสดงหน้าจอการคอมไพล์

ภาคผนวก ค.

ขั้นตอนการติดตั้ง INET Framework

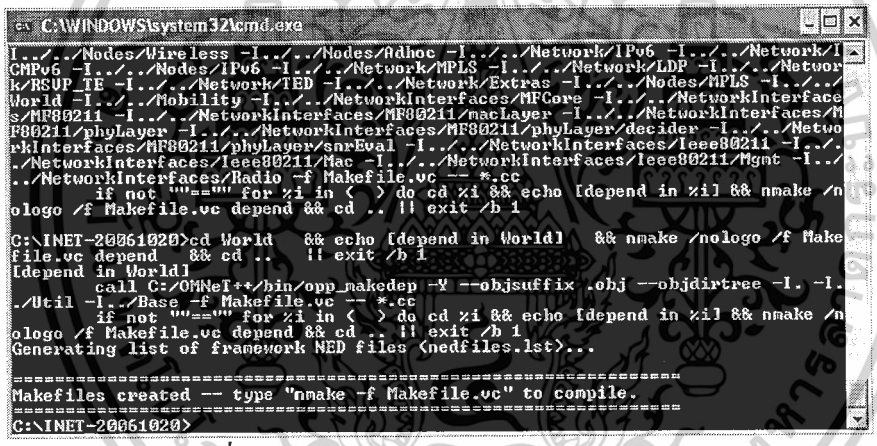
ขั้นตอนการติดตั้ง INET Framework เพื่อทำงานกับโปรแกรม OMNeT++

ขั้นตอนที่ 1 ดาวน์โหลดไฟล์ INET Framework เวอร์ชั่น 20061020 จากเว็บไซต์ http://www.omnetpp.org/filemgmt/viewcat.php?cid=3

ขั้นตอนที่ 2 ทำการแก้ซิปไฟล์สกุล rar จะได้ไฟล์สกุลจะได้โฟลเดอร์ INET-20061020 ทำการย้ายโฟลเดอร์ไปสู่ไดเรกทอรีที่ต้องการจะติดตั้ง

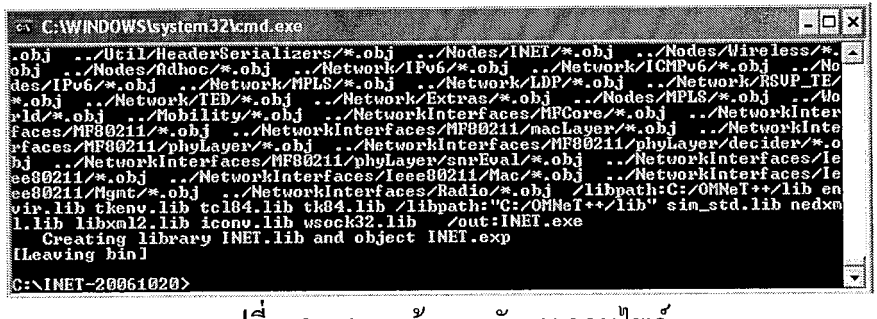
ขั้นตอนที่ 3 ทำการแก้ไขไฟล์ inetconfig.vc โดยในไดเรกทอรี จะมีตัวไฟล์ inetconfig.vc-SAMPLE ที่เป็นตัวอย่างการแก้ไข โดยจะต้องแก้ไข path ของตัวโปรแกรม OMNeT++ ที่ได้ติดตั้ง

ขั้นตอนที่ 4 เรียกใช้งาน makemake.cmd เพื่อทำการสร้าง Makefile



รูปที่ ค.1 แสดงหน้าจอหลังการสร้าง Makefile

ขั้นตอนที่ 5 เรียกใช้คำสั่ง nmake –f Makefile.vc ก็จะเริ่มทำการคอมไพล์ INET Framework เมื่อเสร็จก็จะสามารถใช้งาน Framework ได้ทันที



รูปที่ ค.2 แสดงหน้าจอหลังการคอมไพล์

คำเตือน ทุกครั้งที่เปิดคอมพิวเตอร์และต้องทำการคอมไพล์จะต้องเรียกไฟล์ vcvar32.bat เสมอ