สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การจำลองระบบการให้บริการบนเครือข่ายเคลื่อนที่แบบไร้สาย

SIMULATION OF MOBILE IP SERVICE SYSTEM

โดย

นายจีรศักดิ์ เครือเนตร

นางสาววรวรรณ อธิคมไตรรัตน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2546

เลขหมู่.....
เลขทะเบียน 55773
วัน.เดือน.ปี 25 พ.ค. 2548

b.....
i.....

SIMULATION OF MOBILE IP SERVICE SYSTEM

BY

MR. JEERASAK KRUANATE

MS. VORAVAN ATIKOMTRIRAT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENT FOR THE DEGREE OF

BACHELOR IN DEPARTMENT OF INFORMATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

หัวข้อปริญญานิพนธ์	การจำลองระบบการให้บริการบนเครือข่ายเคลื่อนที่แบบไร้สาย	
ชื่อนักศึกษา	นายจีรศักดิ์ เครือเนตร	รหัสประจำตัว 43010070
	นางสาววรวรรณ อธิคมไตรรัตน์	รหัสประจำตัว 43010373
อาจารย์ที่ปรึกษา	อาจารย์มนต์ชัย แช่มช้อย	
	อาจารย์พนารัตน์ ระวิวรรณ	
ระดับการศึกษา	ปริญญาตรี วิศวกรรมศาสตรบัณฑิต	
สาขา	วิศวกรรมสารสนเทศ	
ภาควิชา	วิศวกรรมสารสนเทศ	
ปีการศึกษา	2546	

ปริญญานิพนธ์ฉบับนี้ได้รับความเห็นชอบจากอาจารย์ที่ปรึกษาเป็นที่เรียบร้อยแล้ว

.....

(อาจารย์มนต์ชัย แช่มช้อย)

.....

(อาจารย์พนารัตน์ ระวิวรรณ)

อาจารย์ผู้ควบคุมปริญญานิพนธ์

หัวข้อปริญญานิพนธ์	การจำลองระบบการให้บริการบนเครือข่ายเคลื่อนที่แบบไร้สาย	
ชื่อนักศึกษา	นายจีรศักดิ์ เครือเนตร	รหัสประจำตัว 43010070
	นางสาววรวรรณ อธิคมไตรรัตน์	รหัสประจำตัว 43010373
อาจารย์ที่ปรึกษา	อาจารย์มนต์ชัย แช่มช้อย	
	อาจารย์พนารัตน์ ระวิวรรณ	
ระดับการศึกษา	ปริญญาตรี วิศวกรรมศาสตรบัณฑิต	
สาขา	วิศวกรรมสารสนเทศ	
ภาควิชา	วิศวกรรมสารสนเทศ	
ปีการศึกษา	2546	

บทคัดย่อ

ปัจจุบันการสื่อสารไร้สายมีความสำคัญต่อการดำเนินชีวิตของมนุษย์มากขึ้นเรื่อยๆ การสื่อสารแบบ Ad-Hoc เป็นหนึ่งในโครงข่ายที่อำนวยความสะดวกแก่ผู้ใช้ที่อยู่นอกขอบเขตการให้บริการของแอคเซสพ้อยต์ ซึ่งผู้ใช้อุปกรณ์ไร้สายสามารถติดต่อสื่อสารกันได้เองโดยปราศจากการควบคุมของแอคเซสพ้อยต์ ทั้งนี้ภายในหนึ่งเครือข่าย Ad-Hoc จะต้องมีหนึ่งอุปกรณ์ที่แสดงตัวเป็นมาสเตอร์ของเครือข่ายนั้นๆ และมีสลาฟซึ่งถูกดูแลโดยมาสเตอร์ โครงงานนี้ได้ทำการศึกษาระบบการให้บริการเคลื่อนที่แบบไร้สายในโครงข่าย Ad-Hoc โดยการจำลองการทำงานด้วยคอมพิวเตอร์ ซึ่งประกอบด้วยการค้นหาเส้นทาง (Routing Algorithm) ในโครงข่ายที่มีการเคลื่อนที่ตลอดเวลา ประสิทธิภาพของระบบที่ได้จะถูกวัดด้วยค่าพารามิเตอร์ เช่น Throughput และ Delay เป็นต้น ผลของการพัฒนาการค้นหาเส้นทางนี้จะทำให้อุปกรณ์ไร้สายสามารถค้นหาเส้นทางที่เหมาะสมในการสื่อสารได้อย่างมีประสิทธิภาพ

Thesis Title	Simulation of Mobile IP Service System	
Student	Mr. Jeerasak Kruanate	ID. 43010070
	Ms. Voravan Atikomtrirat	ID. 43010373
Advisor	Mr. Monchai Chamchoi	
	Ms. Panarat Rawiwan	
Graduate Level	Bachelor Degree of Information Engineering	
Department	Information Engineering	
Academic Year	2003	

ABSTRACT

Nowadays wireless communication system becomes more and more significant in human living. Ad hoc communication is one of the networks which facilitate the users who stay outside the service area of the access point. Each user can communicate with another users without access point controlling. Furthermore, in one ad hoc network should have one device function as master and other devices function as slaves, which being controlled by their master. The point of project is to study the simulation of service system in ad hoc network by simulate the system which consists of routing algorithm in network which has nomadic users. The network performance is indicated by throughput parater, delay parameter, etc. The result ogf the developed routing algorithm will make the wireless device can route efficiently for the suitable path.

กิตติกรรมประกาศ

การทำปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลายๆฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์มนต์ชัย แช่มช้อย และอาจารย์พนารัตน์ ระวิวรรณ อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ ช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างสูง ขอขอบคุณคณาจารย์ภาควิชาวิศวกรรมสารสนเทศทุกๆท่านที่ให้ความรู้และคำปรึกษามาตลอดระยะเวลา 4 ปี รวมทั้งขอขอบคุณรุ่นพี่และเหล่ามิตรสหายคณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่านที่คอยให้คำปรึกษา ช่วยเหลือและให้กำลังใจตลอดมา

สุดท้ายนี้ต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุกๆด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณและขอกราบขอบพระคุณมา ณ ที่นี้

นายจีรศักดิ์ เครือเนตร

นางสาววรวรรณ อธิคมไตรรัตน์

คณะผู้จัดทำ

สารบัญ

เรื่อง	หน้า
บทคัดย่อภาษาไทย	ก
บทคัดย่อภาษาอังกฤษ	ข
กิตติกรรมประกาศ	ค
สารบัญ	ง
สารบัญรูป	ซ
สารบัญตาราง	ฎ
บทที่ 1 บทนำ	1
1.1 บทนำ	1
1.2 วัตถุประสงค์ของโครงงาน	1
1.3 ขอบเขตของโครงงาน	2
1.4 ขั้นตอนการดำเนินงาน	2
บทที่ 2 เครือข่ายโมบายแอดฮอค	3
2.1 มาตรฐานของ Wireless LAN	4
2.2 ลักษณะของเครือข่ายไร้สายโมบายแอดฮอค	6
2.3 วิธีการค้นหาเส้นทาง	7
2.3.1 Proactive (table-driven)	8
2.3.2 Reactive (on-demand)	8
2.3.3 Hybrid	8
2.4 จุดอ่อนของเครือข่ายแอดฮอค	8
2.5 จุดแข็งของเครือข่ายแอดฮอค	9
2.6 โปรโตคอลและอัลกอริทึมของ QoS	9

สารบัญ (ต่อ)

เรื่อง	หน้า
บทที่ 3 โปรโตคอลการค้นหาเส้นทางของเครือข่ายแอดฮอค	11
3.1 Destination Sequenced Distance Vector (DSDV)	11
3.2 Wireless Routing Protocol (WRP)	12
3.3 Cluster Switch Gateway Routing (CSGR)	13
3.4 Ad Hoc On-Demand Distance Vector Routing (AODV)	13
3.5 Dynamic Source Routing (DSR)	15
3.6 Temporally Ordered Routing Algorithm (TORA)	17
3.7 Signal Stability Routing (SSR)	17
3.8 Zone Routing Protocol (ZRP)	18
3.9 Source Tree Adaptive Routing (STAR)	19
3.10 Relative Distance Microdiversity Routing (RDMAR)	19
บทที่ 4 Network Simulator 2	21
4.1 บทนำ	21
4.2 OTcl: The User Language	24
4.3 ตัวอย่างการจำลองอย่างง่าย	28
4.4 ตัวจัดลำดับเหตุการณ์	36
4.5 เน็ตเวิร์คคอมโพเนนต์ (Network Component)	39
4.6 โหนดและการค้นหาเส้นทาง	39
4.7 แพ็คเก็ต	40
บทที่ 5 การออกแบบโปรโตคอลการค้นหาเส้นทางของเครือข่ายแอดฮอค	42
5.1 สถานะของโหนด	43
5.2 เงื่อนไขของเส้นทางการรับส่งข้อมูล	43
5.3 กระบวนการค้นหาเส้นทาง	43
5.4 กระบวนการส่งข้อมูล	49
5.5 กระบวนการแจ้งความเสียหายของการเชื่อมต่อ	49

สารบัญ (ต่อ)

เรื่อง	หน้า
5.6 กระบวนการเข้าร่วมคลัสเตอร์ของโหนดไม่มีสถานะ	51
5.7 กระบวนการตอบรับด้วยข้อความ RREP	53
5.8 โหนดไร้สายใน Ns2	55
5.9 โมดูลที่ใช้ในการค้นหาเส้นทาง	58
5.10 รูปแบบของผลลัพธ์ที่ได้หลังจากการจำลองการทำงานของเครือข่าย (trace format)	58
5.11 โครงสร้างโฟลเดอร์ของ Ns2	59
5.12 คลาสของไทม์เมอร์	61
5.13 ฟังก์ชันที่สำคัญ	62
บทที่ 6 ผลการทดลอง	75
6.1 ค่าสภาพแวดล้อมที่ใช้ในการจำลองเครือข่าย และคำจำกัดความของค่าที่ใช้ในการวัดประสิทธิภาพของโปรโตคอลการค้นหาเส้นทาง	75
6.1.1 ค่าตัวแปรต่างๆที่กำหนดในการจำลองระบบเครือข่าย	75
6.1.2 คำจำกัดความของค่าที่ใช้ในการวัดประสิทธิภาพของโปรโตคอลการค้นหาเส้นทางที่ถูกพัฒนาในโครงงาน	75
6.2 การแสดงผลและวิเคราะห์ผลการทดลอง	75
6.3 การจำลองการจัดกลุ่มของโหนดในเครือข่าย	76
6.4 การจำลองเครือข่ายเพื่อใช้วัดเวลาที่ใช้ในการจัดกลุ่ม	97
6.5 การจำลองเครือข่ายเพื่อเปรียบเทียบความสัมพันธ์ระหว่างประสิทธิภาพและจำนวน ฮอป	97
6.6 การจำลองเครือข่ายเพื่อวัดประสิทธิภาพของโหนดคลัสเตอร์เฮด	99
6.7 การจำลองเครือข่ายเพื่อเปรียบเทียบประสิทธิภาพของทราฟฟิกแบบ CBR และตามกระบวนการป๋วซอง (Poisson Process)	101
6.8 การจำลองเพื่อวัดประสิทธิภาพโดยรวมของทราฟฟิกแบบ CBR	102
6.9 การจำลองเพื่อวัดประสิทธิภาพโดยรวมของทราฟฟิกที่เป็นไปตามกระบวนการป๋วซอง (Poisson Process)	106

สารบัญ (ต่อ)

เรื่อง	หน้า
บทที่ 7 สรุปผลการทดลอง	110
7.1 สรุปผลการทดลอง	110
7.2 ปัญหาที่เกิดขึ้นระหว่างทดลอง	110
7.3 แนวทางการพัฒนาโครงงาน	110
บรรณานุกรม	112

สารบัญรูป

รูป	หน้า
รูปที่ 2.1Homogeneous Mobile Host Network	3
รูปที่ 2.2 Heterogeneous Mobile Device Network	4
รูปที่ 2.3 QE - Quality/Efficiency product	10
รูปที่ 3.1 การแยกประเภทของการค้นหาเส้นทางบนเครือข่ายแอดฮอค	11
รูปที่ 3.2 แสดงการค้นหาเส้นทางของ AODV	14
รูปที่ 3.3 การค้นหาเส้นทางของ DSR	16
รูปที่ 3.4 ความเป็นลูกผสมของ ZRP	19
รูปที่ 4.1 Ns2 ในมุมมองของผู้ใช้	21
รูปที่ 4.2 การจับคู่กัน(Duality)ของ C++ และ OTcl	22
รูปที่ 4.3 สถาปัตยกรรมของ Ns2	23
รูปที่ 4.4 Network Animator (NAM)	24
รูปที่ 4.5 โทโพโลยี และเหตุการณ์ที่ใช้ในการจำลองการทำงานของเครือข่าย	29
รูปที่ 4.6 ผลลัพธ์ที่ได้จากการจำลองการทำงานของเครือข่าย	36
รูปที่ 4.7 การจัดลำดับเหตุการณ์แบบไม่ต่อเนื่อง	37
รูปที่ 4.8 ลำดับชั้นของคลาส(บางส่วน)	39
รูปที่ 4.9 โหนด (ยูนิแคส และ มัลติแคส)	40
รูปที่ 4.10 รูปแบบแพ็คเก็ตใน Ns2	41
รูปที่ 5.1 โมเดลกระบวนการค้นหาเส้นทาง	45
รูปที่ 5.2 โมเดลกระบวนการค้นหาเส้นทาง (ต่อ)	46
รูปที่ 5.3 โมเดลกระบวนการค้นหาเส้นทาง (ต่อ)	47
รูปที่ 5.4 โมเดลการส่งข้อมูล	48
รูปที่ 5.5 โมเดลกระบวนการแจ้งความเสียหายของการเชื่อมต่อ	50
รูปที่ 5.6 โมเดลขั้นตอนการเข้าร่วมคลัสเตอร์ของโหนดไม่มีสถานะ	52
รูปที่ 5.7 โมเดลกระบวนการตอบรับด้วยข้อความ RREP	54
รูปที่ 5.8 โครงสร้างของโหนดในเครือข่ายไร้สาย	56
รูปที่ 5.9 โครงสร้างของโหนดที่ใช้ในการค้นหาเส้นทางแบบ DSR	57

สารบัญรูป (ต่อ)

รูป	หน้า
รูปที่ 5.10 โครงสร้างโฟลเดอร์ของ Ns2 (บางส่วน)	59
รูปที่ 5.11 โฟลเดอร์ที่ทำการเพิ่มขึ้นมาใหม่และไฟล์ที่ต้องทำการแก้ไข	60
รูปที่ 5.12 ลำดับชั้นของคลาส	62
รูปที่ 5.13 Flowchart แสดงขั้นตอนการทางานของฟังก์ชัน recv( )	63
รูปที่ 5.14 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recv( )	64
รูปที่ 5.15 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvDOCR( )	65
รูปที่ 5.16 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvRequest( )	66
รูปที่ 5.17 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvRequest( ) (ต่อ)	67
รูปที่ 5.18 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvReply( )	68
รูปที่ 5.19 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvError( )	68
รูปที่ 5.20 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvHello( )	70
รูปที่ 5.21 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน reqHead( )	71
รูปที่ 5.22 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน reqHead( )	72
รูปที่ 5.23 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน rt_resolve( )	73
รูปที่ 5.24 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน forward( )	74
รูปที่ 6.1 สถานการณ์ที่ 1	76
รูปที่ 6.2 สถานการณ์ที่ 2	77
รูปที่ 6.3 สถานการณ์ที่ 3	78
รูปที่ 6.4 สถานการณ์ที่ 4	79
รูปที่ 6.5 สถานการณ์ที่ 4	80
รูปที่ 6.6 สถานการณ์ที่ 5	81
รูปที่ 6.7 สถานการณ์ที่ 5	82
รูปที่ 6.8 สถานการณ์ที่ 6	84
รูปที่ 6.9 สถานการณ์ที่ 6	87
รูปที่ 6.10 กราฟแสดงตำแหน่งของโหนดต่างๆเมื่อเวลาเริ่มต้น	88
รูปที่ 6.11 กราฟแสดงตำแหน่งของโหนดเมื่อเวลาสิ้นสุด	88
รูปที่ 6.12 สถานการณ์ที่ 7	89

สารบัญรูป (ต่อ)

รูป	หน้า
รูปที่ 6.13 กราฟแสดงตำแหน่งของโหนดเมื่อเวลาเริ่มต้น	90
รูปที่ 6.14 กราฟแสดงตำแหน่งของโหนดเมื่อเวลาสิ้นสุด	90
รูปที่ 6.15 สถานการณ์ที่ 7	96
รูปที่ 6.16 กราฟแสดงความสัมพันธ์ระหว่างจำนวนโหนดและเวลาที่ใช้ในการจัดกลุ่ม	97
รูปที่ 6.17 กราฟแสดงความสัมพันธ์ระหว่างอัตราการส่งข้อมูลและค่าทรูพุต เมื่อขนาดแพ็คเกจมีค่าคงที่	98
รูปที่ 6.18 กราฟแสดงความสัมพันธ์ระหว่างขนาดแพ็คเกจและค่าทรูพุต เมื่ออัตราการส่งข้อมูลมีค่าคงที่	98
รูปที่ 6.19 การจำลองการทำงานของโหนดคลัสเตอร์เฮด	100
รูปที่ 6.20 กราฟเปรียบเทียบประสิทธิภาพของโหนดคลัสเตอร์เฮด	100
รูปที่ 6.21 กราฟแสดงความสัมพันธ์ระหว่างความเร็วของโหนดกับประสิทธิภาพของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	101
รูปที่ 6.22 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับประสิทธิภาพของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	102
รูปที่ 6.23 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกจที่ส่งของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	103
รูปที่ 6.24 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกจที่รับของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	103
รูปที่ 6.25 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Routing Overhead ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	104
รูปที่ 6.26 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Normalize Routing Load ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	104
รูปที่ 6.27 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Packet Delivery Fraction ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR105	105
รูปที่ 6.28 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับเวลาดีเลย์เฉลี่ยระหว่างต้นทางและปลายทางของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR	105

สารบัญรูป (ต่อ)

รูป	หน้า
รูปที่ 6.29 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับประสิทธิภาพของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	106
รูปที่ 6.30 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกจที่ส่งของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	107
รูปที่ 6.31 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกจที่รับของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	107
รูปที่ 6.32 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Routing Overhead ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	108
รูปที่ 6.33 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Normalize Routing Load ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	108
รูปที่ 6.34 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Packet Delivery Fraction ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	109
รูปที่ 6.35 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับเวลาดีเลย์เฉลี่ยระหว่างต้นทางและปลายทางของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง	109

สารบัญตาราง

ตาราง	หน้า
ตารางที่ 2.1 มาตรฐาน Wireless LAN ประเภทต่างๆ	5

บทที่ 1
บทนำ

1.1 บทนำ

ปัจจุบันการสื่อสารไร้สายมีความสำคัญต่อการดำเนินชีวิตของมนุษย์มากขึ้นเรื่อย ๆ การสื่อสารไร้สายจึงเป็นปัจจัยหนึ่งที่ขาดไม่ได้สำหรับมนุษย์ ไม่ว่าจะใช้ในการติดต่อธุรกิจ การศึกษา การแพทย์ เป็นต้น ผู้ให้บริการการสื่อสารไร้สายต่างพัฒนาระบบของตนเองให้รองรับกับความต้องการของผู้ใช้มากขึ้นเรื่อย ๆ แต่พบว่าในบางครั้งความต้องการในการสื่อสารนั้นอยู่นอกขอบเขตการให้บริการของระบบ นั่นคือ โดยปกติแล้วการสื่อสารแบบไร้สายในแต่ละครั้งจะต้องทำการติดต่อจากตัวอุปกรณ์ไปยังสถานีฐาน หรือในบางเครือข่ายเรียกว่า แอคเซสพ้อยต์ ซึ่งในกรณีนี้สามารถนำเครือข่ายแอดฮอคมาใช้ให้บริการได้

การสื่อสารในเครือข่ายแอดฮอค เป็นหนึ่งในโครงข่ายที่อำนวยความสะดวกแก่ผู้ใช้ที่อยู่นอกขอบเขตการให้บริการของแอคเซสพ้อยต์ เช่น ในพื้นที่ทุรกันดารที่มีความต้องการใช้ระบบสื่อสารไร้สาย หรือการเรียนการสอนภายในห้องเรียนที่นักเรียนทุกคนมีคอมพิวเตอร์ส่วนตัว อาจารย์ผู้สอนสามารถติดต่อไปยังคอมพิวเตอร์ของนักเรียนแต่ละคนได้โดยตรง ซึ่งผู้ใช้อุปกรณ์ไร้สายสามารถติดต่อสื่อสารกันได้เองโดยปราศจากการควบคุมของแอคเซสพ้อยต์ ทั้งนี้ภายในหนึ่งเครือข่ายแอดฮอคจะมีหนึ่งอุปกรณ์ที่แสดงตัวเป็นมาสเตอร์ของเครือข่ายนั้น ๆ และมีสลาฟซึ่งถูกดูแลโดยมาสเตอร์

โครงงานนี้ ได้ทำการศึกษาระบบการให้บริการเคลื่อนที่แบบไร้สายในโครงข่ายแอดฮอค โดยการจำลองการทำงานด้วยคอมพิวเตอร์ ซึ่งประกอบด้วยอัลกอริธึมการค้นหาเส้นทาง (Routing Algorithm) ในโครงข่ายที่มีการเคลื่อนที่ตลอดเวลา ประสิทธิภาพของระบบที่ได้จะถูกวัดด้วยค่าพารามิเตอร์ เช่น ค่าทรูพุต ดีเลย์ เป็นต้น ผลของการพัฒนาการค้นหาเส้นทางนี้จะทำให้อุปกรณ์ไร้สายสามารถค้นหาเส้นทางที่เหมาะสมในการสื่อสารได้อย่างมีประสิทธิภาพ

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาโครงสร้างและการให้บริการเคลื่อนที่แบบไร้สาย
2. เพื่อศึกษาโครงสร้างและการทำงานของระบบเครือข่ายแอดฮอคระบบการสื่อสารไร้สาย
3. เพื่อศึกษาการทำงานของโปรโตคอลไอพี

4. เพื่อพัฒนาโปรโตคอล เพื่อรองรับการถ่ายโอนข้อมูลในโครงข่ายแบบไร้สายที่มีการเคลื่อนที่ด้วยความเร็วสูง

1.3 ขอบเขตของโครงงาน

1. โปรโตคอลมีความสามารถในการหาเส้นทางที่เหมาะสมได้อัตโนมัติ
2. มีการอัพเดทการจัดกลุ่มของเทอร์มินัลภายในช่วงระยะเวลาหนึ่ง ๆ ตามที่กำหนด ทำให้ได้ค่าทรูพุตที่เหมาะสม
3. สามารถจัดส่งแพ็คเก็จข้อมูลผ่าน Master Terminal ที่อยู่ติดกันได้โดยอาศัยการส่งแบบรีเลย์

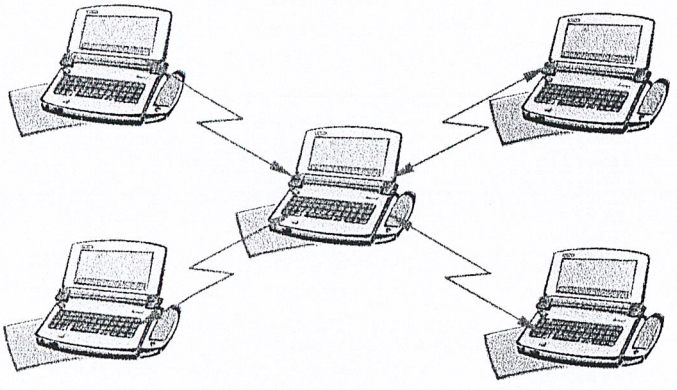
1.4 ขั้นตอนการดำเนินงาน

1. กำหนดขอบเขตและคุณสมบัติของโครงงาน
2. ศึกษาทฤษฎีพื้นฐานและโครงสร้างของเครือข่ายแอดฮอคโดยรวม เช่น สถาปัตยกรรมของเครือข่าย MAC Layer โปรโตคอลไอพี
3. ทำการศึกษาโปรโตคอลการหาเส้นทางแบบต่าง ๆ ที่ใช้ในเครือข่ายแอดฮอค เช่น Dynamic Source Routing (DSR), Ad-Hoc On-Demand Distance Vector (AODV), Cluster Switch Gateway Routing (CSGR) เป็นต้น
4. ศึกษาการทำงานและโครงสร้างของโปรแกรมจำลองการทำงานเครือข่าย (NS-2 : Network Simulator-2) รวมทั้งการใช้ภาษา Otcl (Objected-Tcl)
5. เปรียบเทียบคุณสมบัติของโปรโตคอลแบบต่าง ๆ เพื่อนำมาประยุกต์ใช้ในการพัฒนาโปรโตคอลต่อไป ซึ่งในที่นี้ได้พิจารณาจากโปรโตคอลที่ทำงานแบบ Master-Slave เป็นหลัก
6. พัฒนาโปรโตคอลการค้นหาเส้นทางในเครือข่ายแอดฮอคโดยได้จำลองการทำงานบน NS-2
7. วัดค่าประสิทธิภาพการทำงานของโปรโตคอลต่าง ๆ เช่น ค่าทรูพุต ค่าดีเลย์ เป็นต้น เพื่อนำไปใช้ในการปรับปรุงโปรโตคอลให้มีประสิทธิภาพที่ดียิ่งขึ้น
8. สรุปผลการทดลอง

บทที่ 2

เครือข่ายโมบายแอดฮอค

เครือข่ายโมบายแอดฮอค เป็นเครือข่ายที่สามารถจัดการตนเอง และปรับสภาพให้เข้ากับสภาพแวดล้อมได้ เครือข่ายนี้ประกอบด้วยอุปกรณ์สื่อสารไร้สายสองตัวขึ้นไป และเครือข่ายที่รองรับ โดยใช้การ์ดสำหรับการสื่อสารแบบไร้สายตามมาตรฐาน IEEE802.11 การสื่อสารในเครือข่ายแอดฮอคไม่ต้องการแอ็คเซสพอยต์ในการเชื่อมต่อการสื่อสาร แต่อุปกรณ์ลูกข่ายในเครือข่ายสามารถติดต่อสื่อสารกันโดยตรงได้เอง ทั้งที่อยู่ภายในระยะที่กำลังส่งของอุปกรณ์นั้นส่งถึง หรือแม้แต่นอกระยะที่กำลังส่งจะส่งถึงก็ได้ ซึ่งในกรณีหลังนี้ โหนดที่อยู่ระหว่างทางจะทำหน้าที่ในการรีเลย์ หรือส่งต่อแพ็คเก็ตที่ส่งจากต้นทางไปยังปลายทาง

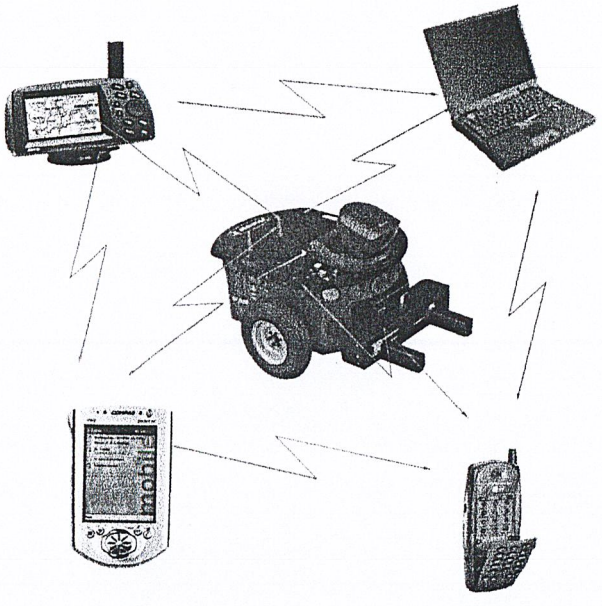


รูปที่ 2.1 Homogeneous Mobile Host Network

เนื่องจากอุปกรณ์ไร้สายแอดฮอคมีอยู่ในหลายรูปแบบ ทั้งพีดีเอ แลปท็อป อินเตอร์เน็ตโทรศัพท์เคลื่อนที่ เป็นต้น การคำนวณพื้นที่เก็บข้อมูล และความสามารถในการสื่อสารของอุปกรณ์ต่าง ๆ จึงมีความหลากหลายอย่างมาก และเนื่องจากเครือข่ายโมบายแอดฮอคไม่มีเอกลักษณ์ของโครง สร้างเครือข่ายที่แน่นอน ดังนั้นเครือข่ายนี้จึงเป็นเครือข่ายที่ไม่มีโครงสร้างพื้นฐาน ไม่มีความจำเป็นที่จะต้องมีสถานีฐาน ไม่ต้องมีสายเชื่อมโยง และไม่ต้องมีเราเตอร์ที่แน่นอน แต่ทุกโหนดในเครือข่าย จะมีฟังก์ชันที่ใช้ในการค้นหาเส้นทางที่เหมาะสมที่สุด กับสภาพแวดล้อมของเครือข่ายในขณะนั้น อย่างไรก็ตามอุปกรณ์ภายในเครือข่ายแอดฮอคยังคงมีการเคลื่อนที่ตลอดเวลา การเชื่อมต่อภายในเครือข่ายจึงเป็นแบบไดนามิค โดยจะเปลี่ยนแปลงไปตามการเคลื่อนที่ของโหนดต่างๆใน

เครือข่าย ดังนั้นข้อมูลการของเส้นทางที่ใช้ในการรับส่งข้อมูลจะต้องมีการเปลี่ยนแปลง เพื่อตอบสนองต่อการเปลี่ยนแปลงของการเชื่อมต่อ

เครือข่ายแอดฮอคนี้จึงเป็นเครือข่ายที่ถูกนำไปประยุกต์ใช้ในหลายๆด้าน ไม่ว่าจะเป็นทั้งทางด้านการทหาร การกู้ภัย การศึกษา และเครือข่ายภายในบ้าน



รูปที่ 2.2 Heterogeneous Mobile Device Network

2.1 มาตรฐานของ Wireless LAN

เครือข่ายโมบายแอดฮอค เป็นเครือข่ายอีกประเภทหนึ่ง ที่เป็นการสื่อสารที่ต้องพึ่งพา Wireless LAN เพื่อใช้ในการสื่อสาร ปัจจุบันมาตรฐานของ Wireless LAN ได้รับความนิยมอย่างมาก และถูกกำหนดมาตรฐานโดย IEEE โดยใช้มาตรฐาน IEEE802.11 ซึ่งยังแบ่งออกเป็นประเภทย่อยๆอีกตามช่วงความถี่ที่ใช้ กำลังงานในการส่ง อัตราการส่งข้อมูล และการมอดูเลชัน เป็นต้น

ตารางที่ 2.1 มาตรฐาน Wireless LAN ประเภทต่างๆ

มาตรฐาน	ช่วงความถี่ (GHz)	กำลังส่ง	Data Rate [claimed] (expected)	Modulation/ MAC protocol	Indep users	Avail-ability	Pros/Cons
802.11b	2.400-2.484	20-30dBm	[1-11] (4)	DSSS-FHSS/ CSMA	3	Now	Pros: cheap, ubiquitous Cons: slow, not robust to interference
HomeRF	2.400-2.484	20dBm	[1-2] (<1)	FHSS/ CSMA	6	Now	Pros: cheap, supports voice QoS Cons: low rate
Wi-LAN	2.400-2.484	15dBm	[30] (?)	DSSS-OFDM/ CSMA	3	Now	Pros: 25MHz channels, 30Mbps Cons:no market momentum
802.11a	5.15-5.35 5.75-5.85	16-29dBm	[6-54] (20)	OFDM-FDMA/ CSMA	12	Now	Pros: high rate, clean spectrum Cons: security, few real-time applications
HiperLAN2	5.15-5.35	23dBm	[6-54] (20)	OFDM-FDMA/ TDMA	8	After 802.11a	PHY nearly identical to 802.11a, optimized for real-time apps Cons: time to market, Europe

2.2 ลักษณะของเครือข่ายไร้สายโมบายแอดฮอค

เครือข่ายแอดฮอคเป็นเครือข่ายที่ไม่มีโครงสร้างพื้นฐาน อย่างที่เครือข่ายไร้สายทั่วไปต้องการ และลักษณะเป็นแบบไดนามิค นั่นคือ รูปร่างของเครือข่ายมีการเปลี่ยนแปลงได้ อันเกิดจากการสร้างและยุบการเชื่อมต่อระหว่างอุปกรณ์ไร้สาย ตามการเคลื่อนที่ของอุปกรณ์ไร้สายภายในเครือข่ายตลอดเวลา เราเตอร์ภายในเครือข่ายนี้ไม่ได้ถูกกำหนดไว้ตายตัว ทุกโหนดภายในเครือข่ายสามารถประพฤติตนเป็นเราเตอร์ และสามารถสามารถค้นหาและปรับปรุงเส้นทางไปยังโหนดอื่นๆ ภายในเครือข่ายได้

แนวคิดของระบบเครือข่ายแบบแอดฮอค ประกอบด้วยหลักการ 4 อย่าง คือ

- Spontaneous

อุปกรณ์ที่ทำงานในระบบเครือข่ายแบบแอดฮอคนั้น จะทำการติดต่อสื่อสารกับอุปกรณ์อื่นด้วยตนเอง โดยที่ไม่ต้องมีอุปกรณ์อื่นๆมากระตุ้นให้เริ่มทำงาน ซึ่งในการใช้งานนั้นอุปกรณ์จะทำการเชื่อมต่อด้วยตนเองโดยที่ไม่จำเป็นต้องรอการสั่งงานจากผู้ใช้ ซึ่งจะแตกต่างจากการทำงานในระบบเครือข่ายแบบเดิม เนื่องจากต้องรอคำสั่งจากผู้ใช้ถึงจะเริ่มต้นทำงาน

- Autonomous

อุปกรณ์ที่ทำงานในระบบเครือข่ายแบบแอดฮอคนั้นจะทำงานด้วยตนเองเป็นอิสระ ไม่ต้องถูกควบคุมจากส่วนกลาง ดังนั้นอุปกรณ์ที่ทำงานในระบบเครือข่ายแบบนี้นั้นต้องมีความสามารถในการจัดการการเชื่อมต่อและการปรับแต่งค่าต่างๆได้ด้วยตนเอง เมื่อใช้งานในเครือข่ายเรียบร้อยแล้วก็จะสามารถยกเลิกการเชื่อมต่อกับอุปกรณ์อื่นได้ด้วยตนเองเช่นกัน

- Dynamic

ระบบเครือข่ายแบบแอดฮอคเป็นระบบเครือข่ายที่มีความยืดหยุ่นสูง เนื่องจากอุปกรณ์ต่าง ๆ ไม่ได้มีการปรับแต่งค่าเพื่อถูกทำงานในระบบเครือข่ายใดๆมาก่อน แต่อุปกรณ์ต่าง ๆ สามารถสร้างการเชื่อมต่อกับอุปกรณ์อื่นเป็นระบบเครือข่ายแบบเฉพาะกิจขึ้นมา ณ เมื่อใด เวลาใดก็ได้ และสามารถยกเลิกการเชื่อมต่อทันทีเมื่อใช้งานเสร็จเรียบร้อยแล้ว

- Mobile

เนื่องจากความยืดหยุ่นของระบบเครือข่ายแบบแอดฮอค ที่สามารถสร้างการเชื่อมต่อ ณ เวลาใดที่ใดได้ ทำให้อุปกรณ์ที่ทำงานในเครือข่ายนี้สามารถที่จะเคลื่อนที่ไปยังที่ต่าง ๆ ได้ โดยที่ยังอยู่ภายในระยะเครือข่าย ซึ่งก่อให้เกิดการใช้งานที่สะดวกสบาย และเป็นธรรมชาติมากยิ่งขึ้น

จากแนวคิดของระบบเครือข่ายแบบแอดฮอคนั้น จะเห็นได้ว่า ต่างจากแนวคิดของระบบเครือข่ายแบบเดิม จากเดิมที่อุปกรณ์ต่างๆถูกจัดการจากส่วนกลาง ต้องถูกปรับแต่งค่าก่อนที่จะใช้งานในเครือข่าย และเมื่อต้องเปลี่ยนรูปแบบเครือข่ายนั้นต้องทำการปรับแต่งค่าใหม่ทุกครั้ง ทำให้อุปกรณ์ต่าง ๆ ถูกจำกัดการใช้งานเฉพาะในเครือข่ายที่กำหนดมาเท่านั้น ขาดความยืดหยุ่นในการใช้งาน ซึ่งต่างจากแนวคิดของระบบเครือข่ายแบบแอดฮอคที่ไม่ได้ถูกควบคุมจากส่วนกลาง ทำให้อุปกรณ์ต่าง ๆ ต้องมีความสามารถในการจัดการการเชื่อมต่อและปรับแต่งค่าต่าง ๆ ด้วยตนเอง ทำให้เกิดความยืดหยุ่นในการใช้งานที่สูงขึ้น

2.3 วิธีการค้นหาเส้นทาง

การทำให้อุปกรณ์สามารถทำงานในระบบเครือข่ายแบบแอดฮอคได้นั้น จะมีประเด็นที่ต้องพิจารณาเพิ่มขึ้นมาจากแนวความคิดของระบบเครือข่ายแบบเดิม ซึ่งประเด็นหนึ่งที่น่าสนใจ คือ การเลือกเส้นทางในติดต่อสื่อสารกับอุปกรณ์ต่าง ๆ เนื่องจากอุปกรณ์ที่ทำงานในระบบเครือข่ายแบบแอดฮอคนี้ มักเป็นอุปกรณ์ที่สามารถพกพา หรือเคลื่อนที่ขณะที่ใช้งานได้ ทำให้ต้องมีกระบวนการในการเลือกเส้นทางที่จะทำการติดต่อสื่อสารกับอุปกรณ์ตัวอื่น ๆ ซึ่งรวมไปถึงในกรณีที่อุปกรณ์ต่างๆเคลื่อนที่ออกจากเครือข่ายเดิม แล้วเข้าไปยังเครือข่ายใหม่ โปรโตคอลในการเลือกเส้นทางนี้ต้องสามารถทำการยกเลิกการเชื่อมต่อกับเครือข่ายเดิม แล้วทำการเชื่อมต่อกับเครือข่ายใหม่ที่กำลังเข้าไปใช้งาน

โปรโตคอลที่ใช้ในการค้นหาเส้นทางของเครือข่ายแอดฮอค โดยแบ่งตามการแลกเปลี่ยนข้อมูลของโมบายโฮสต์ แบ่งเป็น 3 ประเภท ได้แก่

1. Proactive (table-driven)
2. Reactive (on-demand)
3. Hybrid

2.3.1 Proactive (table-driven)

ข้อมูลเส้นทางถูกเก็บไว้ในตารางการค้นหาเส้นทางในแต่ละโหนด และจะถูกนำออกมาใช้เมื่อต้องการ โดยแต่ละโหนดจะบรอดแคสต์ข้อมูลเส้นทางไปยังทุก ๆ โหนดภายในเครือข่ายทุก ๆ ช่วงระยะเวลาที่กำหนด เพื่ออัพเดตข้อมูลเส้นทางอยู่เสมอ แต่จะทำให้เกิดการคับคั่งของข้อมูลภายในเครือข่ายขึ้น

2.3.2 Reactive (on-demand)

ข้อมูลเส้นทางยังคงถูกเก็บไว้ในแต่ละโหนด แต่จะมีการปรับปรุงข้อมูลเส้นทางก็ต่อเมื่อโหนดมีความต้องการใช้เส้นทางนั้น ๆ เมื่อโหนดใดมีความต้องการส่งข้อมูลไปยังปลายทาง ก็จะส่งแพคเก็ต HELLO เพื่อตรวจสอบว่ามีเส้นทางไปยังปลายทางหรือไม่ ถ้ามีก็สามารถส่งข้อมูลไปได้ทันที แต่ถ้าไม่มีก็จะทำการฟลัดดิงแมสเสจออกไป เพื่อหาเส้นทางไปยังปลายทางที่ต้องการ

2.3.3 Hybrid

เป็นการรวมคุณสมบัติของแบบ Proactive และ Reactive เข้าด้วยกัน โดยจะใช้กลวิธีของ Proactive เมื่ออยู่โหนดปลายทางอยู่ภายในเครือข่ายเดียวกัน และใช้กลวิธีของ Reactive เมื่อโหนดปลายทางอยู่คนละเครือข่ายกับโหนดต้นทาง

ซึ่งจะได้กล่าวถึงโปรโตคอลการค้นหาเส้นทางของเครือข่ายแอดฮอคอย่างละเอียด ในบทต่อไป

2.4 จุดอ่อนของเครือข่าย แอดฮอค

เช่นเดียวกันเครือข่ายอื่น ๆ เครือข่ายแอดฮอคเองก็มีจุดอ่อนเช่นกัน ได้แก่

- การจู่โจมจากภายนอกบนการเชื่อมต่อไร้สาย เช่น การดักฟัง การปฏิเสธการให้บริการ การปลอมแปลง การดัดแปลงข้อมูล
- การจู่โจมจากโหนดภายในเครือข่ายเดียวกัน
- เนื่องจากเป็นเครือข่ายแบบไดนามิค ดังนั้นระบบการรักษาความปลอดภัยที่ใช้กับระบบแบบสถิตจึงทำงานไม่ได้ผลดีเท่าที่ควร

- ความสามารถในการเพิ่มขนาด ทำให้เป็นปัญหาเมื่อมีจำนวนโหนดในเครือข่ายมากเกินไป

2.5 จุดแข็งของเครือข่าย แอดฮอค

- สามารถเพิ่มโมบิลิตี้และความยืดหยุ่น นั่นคือ เครือข่ายแอดฮอคสามารถสร้างและยุบได้ภายในระยะเวลาอันสั้น
- เครือข่ายแอดฮอคเป็นเครือข่ายที่มีความประหยัดในบางกรณี คือ มันสามารถกำจัดค่าใช้จ่ายในส่วนของโครงสร้างพื้นฐาน และลดการบริโภคพลังงานของตัวโมบายโหนด
- เครือข่ายแอดฮอคมีความแข็งแรงมากกว่าเครือข่ายไร้สายธรรมดาทั่วๆไป เพราะการควบคุมแบบกระจายที่ไม่เป็นลำดับชั้น และการจัดการกลไก
- เพราะความที่เป็นการเชื่อมต่อสื่อสารในระยะสั้น (โหนดไปยังโหนด แทนที่จะเป็นโหนดไปยังสถานีฐาน) การแพร่ของคลื่นวิทยุจึงถูกจำกัดที่ระดับต่ำ ซึ่งเป็นการเพิ่มการนำสเปคตรัมกลับมาใช้ใหม่ หรือ การใช้ย่านความถี่ที่ไม่ได้รับอนุญาต
- การรองรับมัลติฮอปของเครือข่ายแอดฮอค ทำให้มีความเป็นไปได้ในการสื่อสารแบบ Line of Sight (LoS) ที่คลื่นความถี่สูง

2.6 โปรโตคอล และอัลกอริธึมของ QoS

ReSerVation Protocol (RSVP) : เป็นโปรโตคอลที่ให้สัญญาณการอนุรักษ์ทรัพยากรเครือข่าย (network resource reservation) (สำหรับ Integrated Services)

Differentiated Services (DiffServ): แยกแยะและจัดลำดับความสำคัญแก่การจราจรข้อมูลบนเครือข่ายอย่างคร่าวๆ

Multi Protocol Labeling Switching (MPLS): จัดการแบนด์วิธสำหรับการ aggregate ผ่านการควบคุมเส้นทางบนเครือข่าย เพื่อระบุในเฮดเดอร์ของแพคเก็ต

Subnet Bandwidth Management (SBM):แยกแยะและจัดลำดับความสำคัญที่เลเยอร์ที่สองบนเครือข่าย IEEE802 แบบสวิทช์ และแบบแชร์(Shared and switched IEEE 802 networks)

	Top-down provisioning	Aggregate signaling	Per-flow signaling
No traffic handling	FIFO queuing	aggregate admission control	per-flow admission control
Aggregate traffic handling	top-down provisioned diffserv/802.1p	aggregate RSVP/diffserv aggregate RSVP/802.1p	RSVP/diffserv RSVP/802.1p
Per-flow traffic handling			RSVP/Intserv

improved QE product (increased overhead)

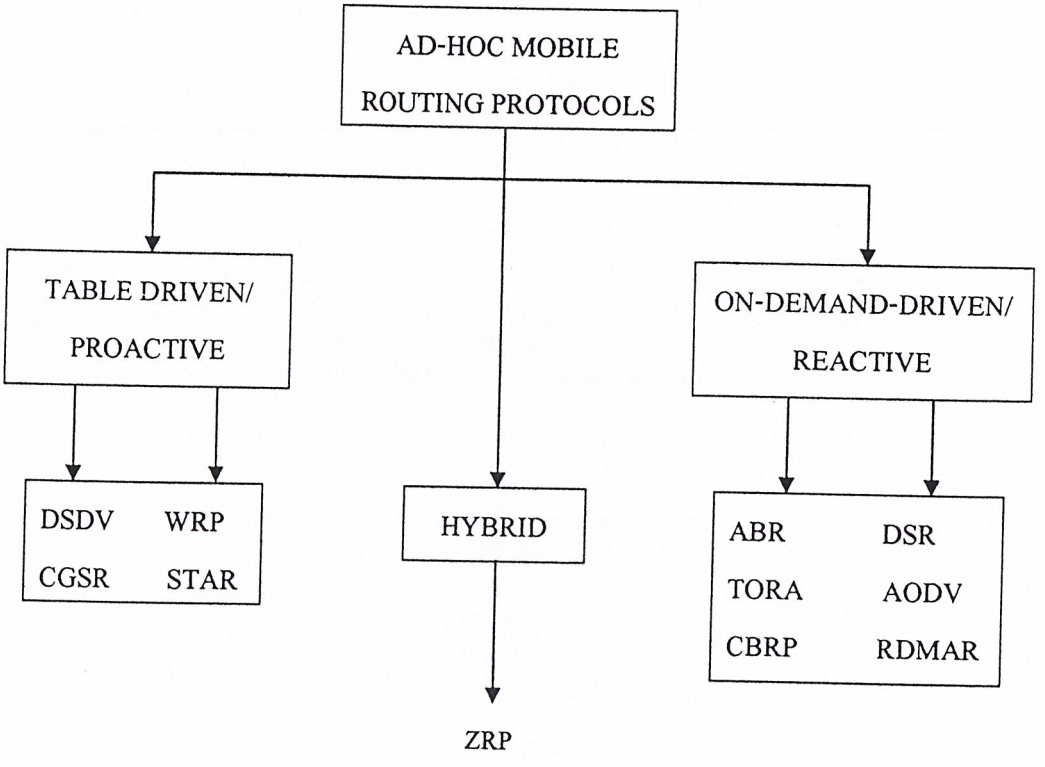
improved QE product (increased overhead)

รูปที่ 2.3 QE - Quality/Efficiency product

บทที่ 3

โปรโตคอลการค้นหาเส้นทางของเครือข่ายแอดฮอค

การค้นหาเส้นทางของเครือข่ายแอดฮอคต่างจากการค้นหาเส้นทางในโปรโตคอลที่มีการเชื่อมต่อตายตัว นั่นคือ ความเหมาะสมของเส้นทางไม่ได้เป็นสิ่งที่สำคัญที่สุดของโปรโตคอลเหล่านี้ แต่กลับเป็นคุณสมบัติของแต่ละเครือข่ายแอดฮอค เช่น การเปลี่ยนแปลงเส้นทางอย่างเฉียบพลัน ความสามารถในการตอบโต้สูง การรักษาแบนด์วิธ (Bandwidth Conservation) หรือ การจำกัดการดำเนินการที่ถูกร้องขอในแต่ละโหนด ที่ถูกมองว่ามีความสำคัญกว่า



รูปที่ 3.1 การแยกประเภทของการค้นหาเส้นทางบนเครือข่ายแอดฮอค

3.1 Destination Sequenced Distance Vector (DSDV)

เป็นโปรโตคอลค้นหาเส้นทางที่ยึดจากตารางการค้นหาเส้นทาง ซึ่งมีพื้นฐานมาจากอัลกอริธึมค้นหาเส้นทางของเบลล์แมน-ฟอร์ด โปรโตคอลแบบ DSDV นี้มีการแก้ไขปัญหาการค้นหาเส้นทางที่วนเป็นลูปภายในเครือข่าย ซึ่งภายในเครือข่ายประกอบด้วยโหนด โดยแต่ละโหนด

คือ อุปกรณ์สื่อสารไร้สาย ข้อมูลที่เก็บอยู่ในแต่ละโหนดประกอบด้วยตารางการค้นหาเส้นทาง (Routing Table) ซึ่งแสดงโหนดปลายทางที่เป็นไปได้ทั้งหมดภายในเครือข่ายเดียวกัน และจำนวนการกระโดดไปยังแต่ละโหนดปลายทาง ข้อมูลเหล่านี้จะต้องเป็นข้อมูลที่ใหม่ และพร้อมนำไปใช้เสมอ

ค่าตัวเลขซีเควนซ์ใช้ในการพิจารณาเส้นทาง โดยสามารถที่จะแยกแยะเส้นทางเก่าและเส้นทางใหม่ที่ดีกว่าจากกันได้ ดังนั้นจึงมีการอัพเดตตารางการค้นหาเส้นทางทุกๆช่วงระยะเวลาหนึ่งๆ ซึ่งทำให้เกิดความดับดั่งของข้อมูลในเครือข่าย และทำให้ใช้ทรัพยากรอย่างไม่เต็มประสิทธิภาพ ด้วยเหตุนี้ DSDV จึงมีการนำแพ็คเกตการอัพเดตข้อมูลเส้นทางสองแบบมาใช้ได้แก่ แบบ Full Dump และแบบ Incremental

ข้อมูลที่ใช้ในการค้นหาเส้นทางที่บรอดแคสต์ออกจากโหนดต้นทาง ประกอบด้วย แอดเดรสปลายทาง จำนวนฮอปไปยังปลายทาง และเลขซีเควนซ์ของข้อมูลที่ได้รับโดยเส้นทางใดๆ ก็ตามจะถูกใช้เมื่อ เส้นทางนั้นมีค่าเลขซีเควนซ์ที่ใหม่ที่สุด หรือถ้าเส้นทางนั้นมีเลขซีเควนซ์เหมือนกัน ก็จะพิจารณาจากเส้นทางที่มีจำนวนฮอปน้อยกว่า

3.2 Wireless Routing Protocol (WRP)

โปรโตคอลนี้ป้องกันการเกิดปัญหา count-to-infinity โดยการให้ทุกโหนดตรวจสอบการมีอยู่ของโหนดก่อนหน้า ซึ่งทำให้ขจัดการเกิดลูป และทำให้การสร้างเส้นทางใหม่ที่เร็วขึ้นเมื่อลิงค์เกิดความเสียหาย

ในโปรโตคอลนี้ แต่ละโหนดจะทราบการมีอยู่ของโหนดอื่นที่อยู่ข้างเคียงได้จากการได้รับข้อความ Ack และข้อความอื่นๆ ถ้าโหนดใดๆ ก็ตามยังไม่มีการส่งแพ็คเกต เมื่อผ่านไปช่วงระยะเวลาหนึ่ง โหนดนั้นจะต้องส่งข้อความ HELLO เพื่อให้แน่ใจว่ายังคงมีการเชื่อมต่ออยู่จากข้อความตอบรับ แต่ถ้าไม่มีการตอบรับ นั่นหมายถึงความเสียหายของการเชื่อมต่อ ณ จุดใดจุดหนึ่ง เมื่ออุปกรณ์ไร้สายได้รับข้อความ HELLO จากโหนดใหม่ ข้อมูลของโหนดใหม่นั้นจะถูกบันทึกลงใน ตารางการค้นหาเส้นทางของโมบายนั้นๆ หลังจากนั้นอุปกรณ์ไร้สายจะส่งสำเนาของข้อมูลในตารางการค้นหาเส้นทางของมันไปให้

โปรโตคอลแบบนี้ต้องประกอบด้วยสี่ตาราง ได้แก่

1. Distance Table ระบุจำนวนฮอประหว่างแต่ละโหนด และปลายทาง
2. Routing Table ระบุโหนดสำหรับฮอปครั้งต่อไป
3. Link-Cost Table ระบุดีเลย์ของการเชื่อมต่อ ณ จุดนั้นๆ
4. Massage Retransmission List (MRL) Table ประกอบด้วยเลขซีเควนซ์ของข้อความอัพเดต ตัวนับการส่งใหม่ (Retransmission Counter) เวคเตอร์แฟลคร้องขอการAcknowledge และลิสต์ของการอัพเดตที่ถูกส่งในข้อความอัพเดต

3.3 Cluster Switch Gateway Routing (CSGR)

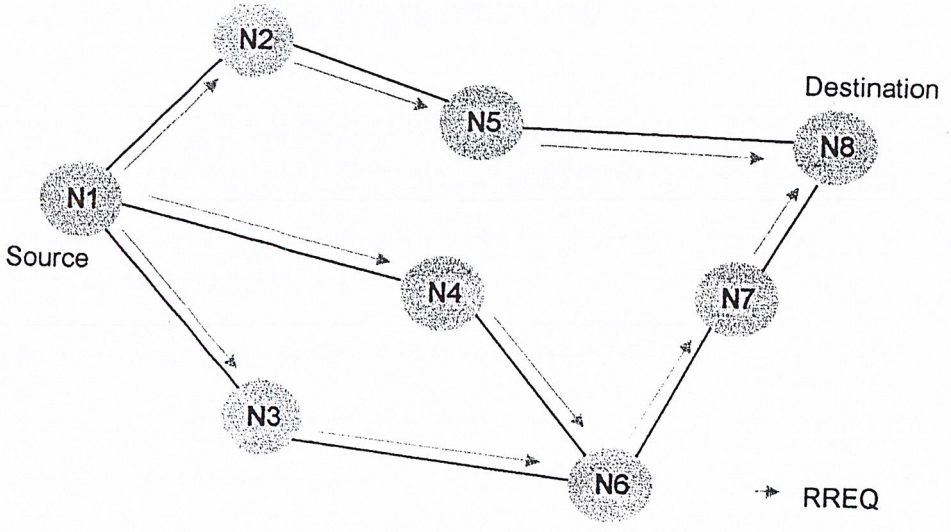
โหนดทุกโหนดจะถูกจัดให้อยู่เป็นกลุ่มๆ เรียกว่า "คลัสเตอร์" โดยแต่ละคลัสเตอร์จะต้องมีคลัสเตอร์เฮด ซึ่งสามารถควบคุมกลุ่มของโหนดในคลัสเตอร์ของตนเอง และจัดการโครงสร้างสำหรับการแยกใช้รหัสระหว่างคลัสเตอร์ Channel Access การค้นหาเส้นทาง และการจัดสรรแบนด์วิธ คลัสเตอร์เฮดถูกเลือกโดยอัลกอริธึมการเลือกคลัสเตอร์เฮดแบบกระจาย (Distributed Cluster Head Selection Algorithm) แต่เนื่องจากอัลกอริธึมแบบนี้ทำให้เกิดการเลือกคลัสเตอร์เฮดบ่อยครั้งจนเกินไป จึงได้เปลี่ยนมาใช้อัลกอริธึม Least Cluster Change (LCC) ซึ่งจะเลือกคลัสเตอร์เฮดใหม่ก็ต่อเมื่อมีคลัสเตอร์เฮดสองตัวมาเจอกัน หรือโหนดนั้นๆได้ออกจากการควบคุมของคลัสเตอร์เฮดทุกตัว

ตารางสมาชิกของคลัสเตอร์ (Cluster Member Table) อัพเดตโดยการบรอดแคสออกไปจากทุกๆ โหนดในทุกๆ ช่วงระยะเวลาหนึ่ง โดยใช้โปรโตคอล DSDV

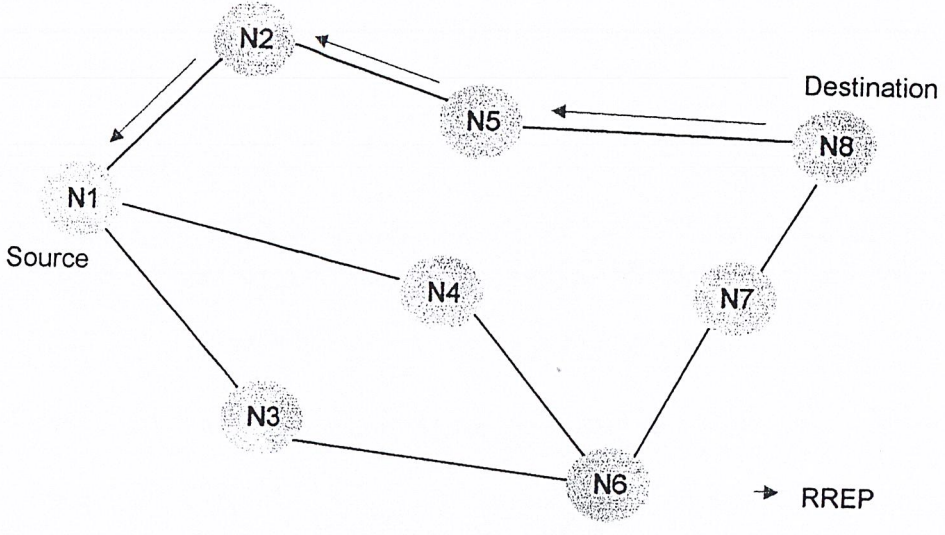
ในโปรโตคอลแบบ CSGR นี้ จะต้องมีการอัพเดตทั้งตารางเส้นทาง และตารางสมาชิกของคลัสเตอร์

3.4 Ad Hoc On-Demand Distance Vector Routing (AODV)

มีพื้นฐานอยู่บนอัลกอริธึม DSDV โดยเป็นการพัฒนามาจาก DSDV เพราะมีการลดปริมาณการบรอดแคสเหมือนกัน แต่ในกรณีของ AODV จะมีการสร้างเส้นทางก็ต่อเมื่อมีการร้องขอเท่านั้น ซึ่งตรงกันข้ามกับ DSDV ทางผู้คิดค้น AODV ได้จัดให้โปรโตคอลตัวนี้เป็นระบบ On-Demand Route Acquisition บริสุทธิ์ โดยแต่ละโหนดที่ไม่ได้อยู่บนเส้นทางที่เลือกจะไม่มีการอัพเดตข้อมูลเส้นทาง หรือมีส่วนร่วมในการแลกเปลี่ยนข้อมูลในตารางการค้นหาเส้นทาง



ก. เส้นทางของแพ็คเกต RREQ



ข. เส้นทางของแพ็คเกต RREP

รูปที่ 3.2 แสดงการค้นหาเส้นทางของ AODV

เมื่อโหนดต้นทางต้องการส่งข้อมูลไปยังโหนดปลายทาง แต่พบว่ายังไม่มีเส้นทางไปยังโหนดนั้นๆ โหนดต้นทางจะต้องสร้างกระบวนการ Path Discovery เพื่อหาตำแหน่งของโหนดอื่นๆ โดยบรอดแคสแพ็คเกต Route Request (RREQ) ออกไปยังโหนดข้างเคียง แล้วโหนดข้างเคียงนั้นๆ ก็จะส่งต่อไปยังโหนดข้างเคียงของตนเองเรื่อยๆจนกระทั่งส่งไปถึงโหนดปลายทาง หรือถึงโหนดระหว่างทางที่มีเส้นทางไปยังโหนดปลายทางที่ใหม่พอ AODV ใช้เลขซีเควนซ์ เพื่อทำให้แน่ใจว่าทุกเส้นทางไม่เกิดลูป และมีข้อมูลเส้นทางที่ใหม่ที่สุด โดยแต่ละโหนดจะมีการปรับปรุงเลขซี

เควนซ์ ของตนเอง เช่นเดียวกับ Broadcast ID ค่า Broadcast ID นี้จะมีค่าเพิ่มขึ้นทุกๆครั้งที่โหนดนั้นๆ ส่งแพ็คเกต RREQ ออกไป พร้อมๆ กับ IP Address ของโหนดนั้นๆ เพื่อระบุว่าเป็น RREQ ของโหนดใด โหนดที่อยู่ระหว่างทางสามารถตอบ RREQ ได้ก็ต่อเมื่อตัวมันมีเส้นทางไปยังโหนดปลายทางซึ่งมีเลขซีเควนซ์มากกว่าหรือเท่ากับที่มีอยู่ใน RREQ โดยโหนดระหว่างทางนั้นๆ สามารถตอบกลับโดยการยูนิแคส RREP (Route Reply) กลับไปยังโหนดที่ส่ง RREQ มาให้มัน RREP จะถูกส่งไปตามเส้นทางที่ถูกสร้างโดย RREQ ที่เป็นเช่นนี้ก็เพราะว่าโปรโตคอล AODV รองรับเฉพาะการเชื่อมต่อแบบสมมาตรเท่านั้น

ในกรณีของ AODV เมื่อโหนดต้นทางมีการเคลื่อนที่จะต้องเริ่มการทำงานของโปรโตคอลการค้นหาเส้นทางใหม่ เพื่อหาเส้นทางใหม่ไปยังปลายทาง และเมื่อโหนดที่อยู่ระหว่างทางมีการเคลื่อนที่ โหนดที่อยู่ก่อนหน้ามันบนเส้นทางเดียวกันจะกระจายข้อความ Linkfailure Notification ไปยังโหนดที่อยู่ก่อนหน้ามันอีกทีหนึ่งจนส่งไปถึงโหนดต้นทาง หลังจากนั้นโหนดต้นทางก็จะหาเส้นทางใหม่สำหรับโหนดปลายทางนั้น ถ้ายังมีความต้องการที่จะใช้เส้นทางนั้นอยู่

3.5 Dynamic Source Routing (DSR)

DSR อยู่บนพื้นฐานของ Source routing ประกอบด้วย 2 เฟสหลัก ได้แก่

1. การค้นหาเส้นทาง โดยการส่งแพ็คเกจ Route Request ซึ่งประกอบด้วย แอดเดรสปลายทาง แอดเดรสต้นทาง และ หมายเลข id
2. การบำรุงรักษาเส้นทาง

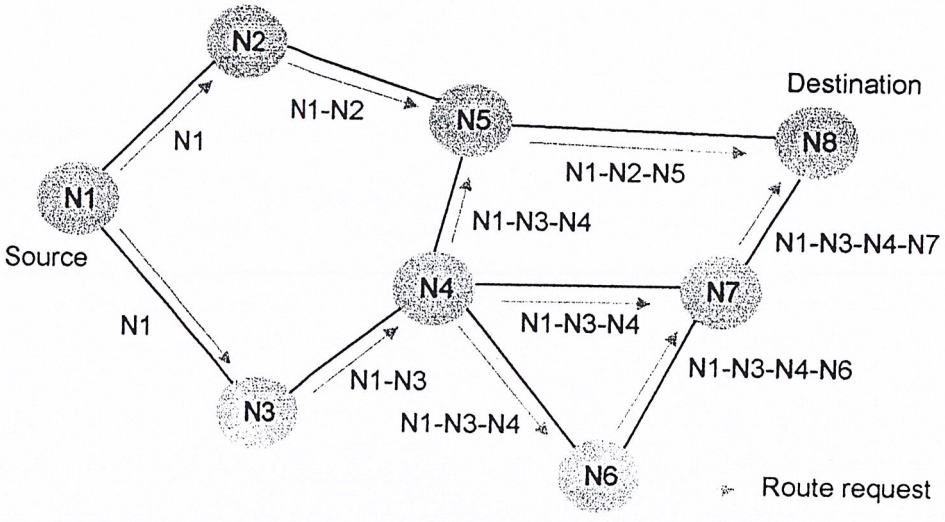
เมื่อมีโหนดที่ต้องการส่งข้อมูลไปยังโหนดหนึ่ง โหนดที่ต้องการส่งข้อมูลจะต้องตรวจใน Route Cache ของตนว่ามีเส้นทางที่ยังใช้งานได้ไปยังโหนดปลายทางหรือไม่ ถ้ามีก็จะใช้เส้นทางเดิม แต่ถ้าไม่มีก็จะทำการค้นหาเส้นทางใหม่ โดยการบรอดแคสแพ็คเกต Route Request ออกไป ซึ่ง Route Request นี้ ประกอบด้วยแอดเดรสของโหนดปลายทาง แอดเดรสของโหนดต้นทาง และหมายเลข id ตามที่ได้กล่าวไปแล้ว แต่ละโหนดที่ได้รับแพ็คเกตนี้ก็จะตรวจสอบว่าตัวมันรู้เส้นทางไปยังโหนดปลายทางหรือไม่ ถ้าไม่มีมันก็จะเพิ่มแอดเดรสของมันเข้าไปที่ Route Record ของแพ็คเกต แล้วส่งแพ็คเกตนั้นออกไป การจำกัดปริมาณ Route Request สามารถทำได้ โดยอุปกรณ์ไร้สายจะส่งแพ็ค เกตออกไปก็ต่อเมื่อ

1. อุปกรณ์ไร้สายนั้นๆไม่เคยได้รับ Request มาก่อน

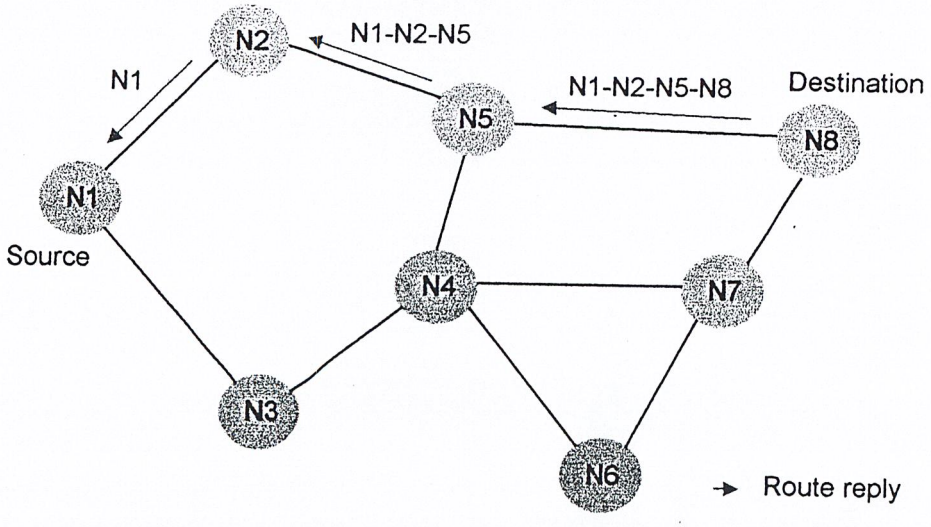
2. แอดเดรสของอุปกรณ์ไร้สายไม่ได้อยู่ใน Route Record

ส่วน Rooute Reply จะถูกสร้างเมื่อ

1. Route Request ส่งถึงปลายทางด้วยตัวของมันเอง
2. Route Request ส่งถึงโหนดระหว่างทางที่มีเส้นทางที่ยังใช้งานได้ไปยังโหนดปลาย ทางเก็บอยู่ใน Route Cache



ก. เส้นทางของแพ็คเกต Route request



ข. เส้นทางของแพ็คเกต Route reply

รูปที่ 3.3 การค้นหาเส้นทางของ DSR

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

3.6 Temporally Ordered Routing Algorithm (TORA)

มีความสามารถในการปรับตัวสูง ไม่เกิดลูป และใช้อัลกอริธึม Distributed routing ที่มีพื้นฐานอยู่บน Link reversal TORA ถูกนำไปใช้ในเครือข่ายที่มีสภาพแวดล้อมที่มีการเปลี่ยนแปลงบ่อย โปรโตคอลนี้ประกอบด้วย 3 ฟังก์ชัน ได้แก่ การสร้างเส้นทาง (Route creation) การบำรุงรักษาเส้นทาง (Route maintainence) และการลบเส้นทาง (Route erasure)

สำหรับเฟสของการค้นหาและปรับปรุงเส้นทาง โหนดจะใช้เมทริกความสูงในการสร้าง DAG (Directed Acyclic Graph) โดยมีรากอยู่ที่ปลายทาง หลังจากนั้น การเชื่อมต่อจะถูกกำหนดทิศทาง (Upstream หรือ Downstrem) โดยความสัมพันธ์กับเมทริกความสูงของโหนดข้างเคียง

3.7 Signal Stability Routing (SSR)

SSR มีพื้นฐานอยู่บน Associativity-Based Routing (ABR) โดยมีส่วนคล้าย ABR ตรงที่การเลือกเส้นทางจะพิจารณาจากความเข้มของสัญญาณระหว่างโหนด และความเสถียรของโหนด แต่ SSR จะพิจารณาเลือกที่ความแข็งแรงของการเชื่อมต่อด้วย

SSR สามารถแบ่งเป็นการทำงานร่วมกันของ 2 โปรโตคอล คือ

1. Dynamic Routing Protocol (DRP) รับผิดชอบในส่วนของการปรับปรุง Signal Stability Table (SST) ซึ่งบันทึกความเข้มของสัญญาณของโหนดข้างเคียง และ Routing Table (RT) โดยจะส่งแพ็คเกตต่อให้โปรโตคอลถัดไป (SRP) ดำเนินการต่อ
2. Static Routing Protocol (SRP) ดำเนินการกับแพ็คเกตโดยรับแพ็คเกตจาก DRP แล้วเก็บไว้แบบสแต็ค เมื่อตนเองเป็นโหนดปลายทาง หรือค้นหาเส้นทางไปยังปลายทางในตารางการค้นหาเส้นทางแล้วส่งมันต่อไป ถ้าตัวมันยังไม่ใช่โหนดปลายทาง

ถ้าไม่มีการบันทึกเส้นทางไปยังโหนดปลายทางใน ตารางการค้นหาเส้นทาง จะมีการทำ Route-search เพื่อหาเส้นทาง โดยการกระจาย Route Request ออกไปยังเครือข่าย หรือเพียงส่งต่อให้โหนดที่ถัดจากตัวมันเท่านั้น ในกรณีที่มันได้รับช่องสัญญาณที่แรงกว่า และไม่เคยดำเนินการมาก่อน เพื่อป้องกันการเกิดลูป

55773

โหนดปลายทางจะเลือกแพ็คเกต Route-Search ที่มาถึงแพ็คเกตแรก เพราะมีความเป็นไปได้มากที่สุดว่า เป็นแพ็คเกตของเส้นทางที่สั้น และมีความคับคั่งน้อยที่สุด หลังจากนั้น DRP จะเลือกเส้นทางเพื่อส่งแมสเสจ Route-reply กลับไปยังโหนดต้นทาง ระหว่างที่ส่งนั้น DRP ของโหนดที่อยู่ระหว่างทางก็จะอัพเดตตารางการค้นหาเส้นทางของตนเอง

เมื่อมีการพบความเสียหายของการเชื่อมต่อภายในเครือข่าย โหนดนั้นจะส่ง Error message ไปยังโหนดต้นทาง เพื่อระบุว่าเกิดความเสียหายขึ้นช่องในสัญญาณใด โหนดต้นทางจะดำเนินการ Route-search อีกครั้ง เพื่อหาเส้นทางใหม่ไปยังปลายทาง หลังจากนั้นโหนดต้นทางจะส่ง Erase message ไปยังทุกโหนดที่อยู่บนเส้นทางที่เกิดความเสียหาย เพราะฉะนั้นจึงกล่าวได้ว่า SSR มีความคล้ายคลึงกับ DSR และ AODV นั่นคือ โหนดต้นทางจะถูกรบกวนทุกครั้งที่เส้นทางถูกทำลาย

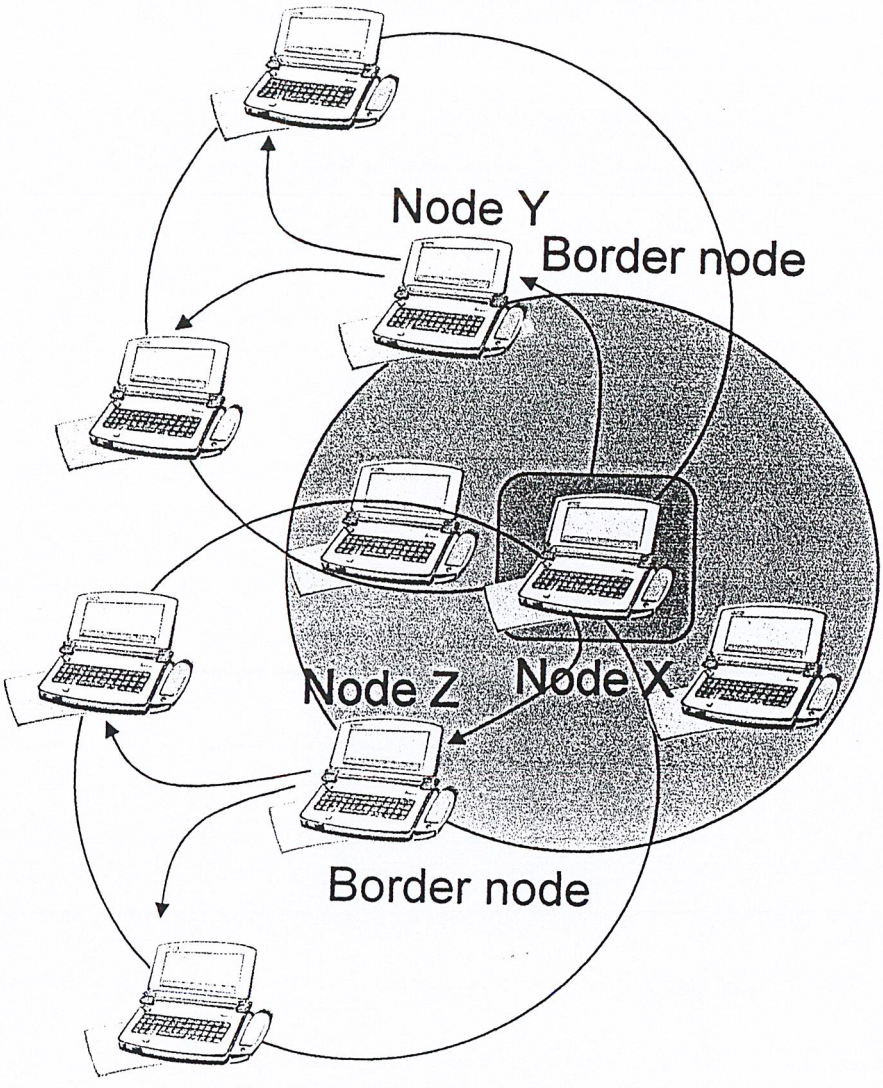
3.8 Zone Routing Protocol (ZRP)

ZRP เป็นโปรโตคอลลูกผสมระหว่างโปรโตคอลแบบ on-demand และโปรโตคอลแบบ Proactive routing การแบ่งโซนคล้ายกับคลัสเตอร์ เพียงแต่ทุกโหนดประพฤติตัวเป็นคลัสเตอร์เฮดและในขณะเดียวกันก็เป็นสมาชิกของคลัสเตอร์อื่น โซนสามารถซ้อนทับกันได้ แต่ละโหนดระบุรัศมีของโซนได้รูปแบบของฮอป ขนาดของโซนมีผลต่อประสิทธิภาพของการสื่อสารแอดฮอค

จากที่ได้กล่าวไปแล้วว่า ZRP เป็นโปรโตคอลลูกผสม โดยจะใช้โปรโตคอลแบบ Table-driven-based routing เมื่อดำเนินการภายในโซนเดียวกัน นั่นหมายถึงแต่ละโหนดจะเก็บเส้นทางไปยังทุกโหนดที่เหลือในโซนเดียวกัน ถ้าโหนดปลายทางอยู่คนละโซนกับโหนดต้นทาง ก็จะต้องใช้โปรโตคอลแบบ Proactive routing

ZRP ประกอบด้วยโปรโตคอลย่อย 3 โปรโตคอล

1. The proactive (table-driven) Intrazone Routing Protocol (IARP) สามารถใช้การค้นหาได้ทั้งแบบ Link-state และแบบ Distance-vector
2. The reactive Interzone Routing Protocol (IERP) อยู่บนบอร์เดอร์โหนด เพื่อรองรับการค้นหาแบบ On-demand ในการค้นหาเส้นทางไปยังโหนดที่อยู่ในโซนอื่น
3. The Bordercast Resolution Protocol (BRP)



รูปที่ 3.4 ความเป็นลูกผสมของ ZRP

3.9 Source Tree Adaptive Routing (STAR)

STAR เป็นโปรโตคอลแบบ Proactive routing ที่ไม่ต้องมีการอัพเดตเส้นทางทุกๆ ช่วงระยะเวลา หรือต้องปรับปรุงให้ได้เส้นทางไปยังปลายทางที่เหมาะสมที่สุด และ STAR ยังใช้ ORA table-driven-based routing ในการสร้าง source tree

3.10 Relative Distance Microdiversity Routing (RDMAR)

ประมาณค่าระยะทางระหว่างสองโหนดในรูปแบบของฮอปโดยใช้อัลกอริธึม relative distance estimation เนื่องจากการใช้การประมาณระยะทาง ทำให้สามารถลดปริมาณการฟลัดดิงได้

แต่ RDMAR นี้ก็มีข้อจำกัด ได้แก่

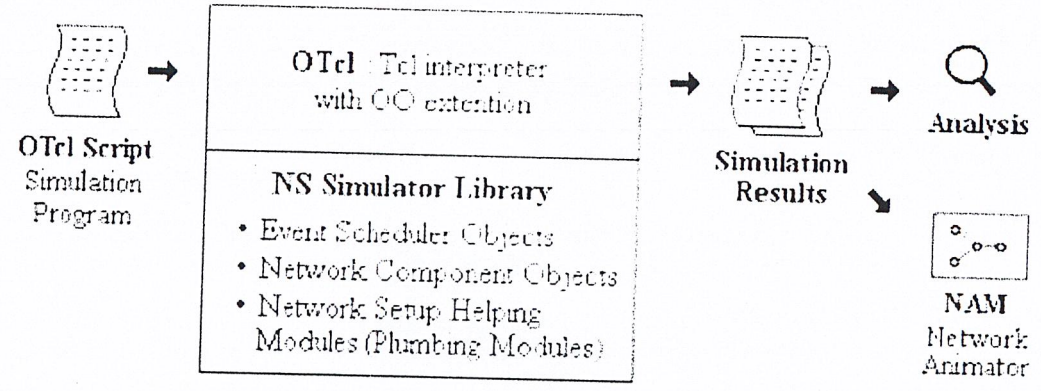
1. การประมาณระยะทางเป็นฟังก์ชั่นความสัมพันธ์ของระยะทางที่ถูกบันทึกไว้แล้ว
2. สมมติฐานบนความเร็วที่กำหนดไว้ตายตัวของโฮสต์ไร้สาย
3. ขอบเขตการรับส่งคลื่นวิทยุที่กำหนดไว้ตายตัว

บทที่ 4

Network Simulator 2

4.1 บทนำ

Ns2 เป็นโปรแกรมจำลองการทำงานของเครือข่าย (network simulator) ที่มีการทำงานแบบอีเวนท์-ไดร์เวน (even-driven) Ns ถูกพัฒนาขึ้นมาโดย UC Berkely เพื่อจำลองการทำงานของเครือข่ายไอพีแบบต่างๆ โดยใน Ns2 จะมีโปรโตคอลต่างๆที่ได้พัฒนาไว้เช่น TCP และ UDP ประเภทของตัวกำเนิดทราฟฟิค (traffic source) แบบต่างๆเช่น FTP, Telnet, Web, CBR และ VBR คิวของเราเตอร์ (router) แบบต่างๆเช่น Drop Tail, RED และ CBQ อัลกอริทึมในการค้นหาเส้นทางแบบต่างๆ เช่น Dijkstra และอื่นๆ ในปัจจุบัน Ns ได้กลายเป็นหนึ่งในโปรเจคของ VINT Project โดย Ns เวอร์ชันปัจจุบันคือเวอร์ชัน 2 เป็นเวอร์ชันที่ถูกพัฒนาขึ้นมาโดยใช้ภาษา C++ และ OTcl (ภาษาสคริปต์ Tcl ที่เพิ่มความสามารถด้านออปเจ็คเข้ามา ถูกพัฒนาขึ้นมาโดย MIT) โดยในบทนี้จะกล่าวถึงโครงสร้างพื้นฐานของ Ns2

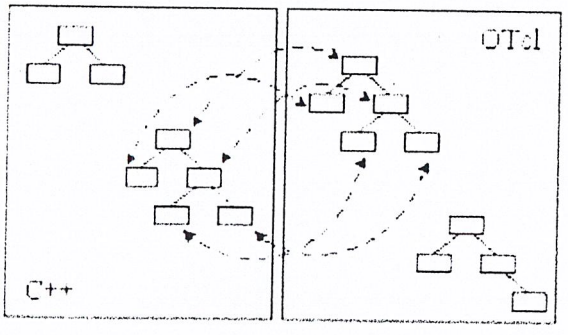


รูปที่ 4.1 Ns2 ในมุมมองของผู้ใช้

ในรูปที่ 4.1 เป็นการแสดงถึง Ns2 ในมุมมองของผู้ใช้ โดย Ns2 จะประกอบด้วย ตัวแปลภาษา Otcl และไลบรารี่ต่างๆที่ใช้ในการจำลองการทำงานของเครือข่าย หรืออาจกล่าวได้ว่า ในการที่จะใช้งาน Ns2 นั้น ผู้ใช้เพียงแค่เขียนโปรแกรมเป็นสคริปต์ภาษา OTcl เพียงเท่านั้น โดยในการที่จะจำลองการทำงานของเครือข่าย ผู้ใช้จะต้องเขียนสคริปต์ภาษา OTcl เพื่อที่จะกำหนดเหตุการณ์เริ่มต้น กำหนดโครงสร้างของเครือข่ายโดยใช้ออปเจ็คโหนดที่มีให้ หลังจากนั้นจึงทำการกำหนดแหล่งกำเนิดทราฟฟิคว่า จะทำการเริ่มส่งแพคเก็ตเมื่อใดและหยุดส่งเมื่อใด นอกจากนั้นถ้า

หากผู้ใช้ต้องการสร้างออปเจ็คโหนดขึ้นมาใหม่ก็สามารถทำได้โดยการพัฒนาออปเจ็คขึ้นมาใหม่ หรือ โดยการนำออปเจ็คที่มีอยู่แล้วมาประกอบกันแล้วสร้างเป็นออปเจ็คใหม่

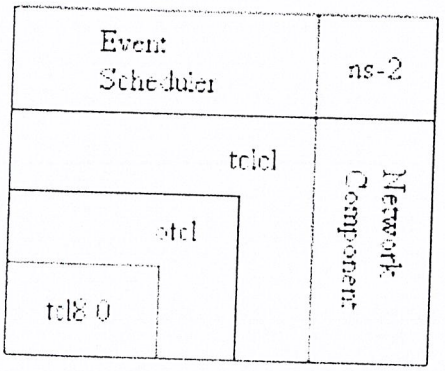
นอกจากออปเจ็คเน็ตเวอร์คแล้ว Ns2 ยังมีส่วนประกอบหลักอื่นๆอีกเช่น ตัวจัดลำดับของเหตุการณ์ (event scheduler) โดยเหตุการณ์ใน Ns2 ก็คือ ตัวระบุของแต่และแพคเก็ต (packet ID) โดยตัวระบุนี้จะแตกต่างกันในแต่ละแพคเก็ต นอกจากนั้นยังมีเวลาที่ถูกจัดลำดับการทำงาน (scheduled time) และพอยเตอร์ที่ทำการชี้ไปยังออปเจ็คที่ดูแลเหตุการณ์นั้นๆ ในการทำงานของ Ns2 ตัวจัดลำดับของเหตุการณ์จะคอยติดตามเวลาที่ใช้ในการจำลองการทำงาน และปล่อยให้เหตุการณ์ที่ถูกเก็บอยู่ในคิวของเหตุการณ์เกิดขึ้นเมื่อเวลาที่เหตุการณ์นั้นต้องเกิดดำเนินการมาถึง โดยทำการเรียกใช้เน็ตเวอร์คคอมโพเน้นต์ที่ทำการร้องขอให้เกิดเหตุการณ์นั้นๆ และปล่อยให้เน็ตเวอร์คคอมโพเน้นต์ตัวนั้นทำงานกับแพคเก็ตต่อไป อย่างไรก็ตามในการติดต่อกันระหว่างเน็ตเวอร์คคอมโพเนนต์และเน็ตเวอร์คอมโพเนนต์อีกตัวที่ทำการส่งแพ็คเก็ตถึงกันไม่ได้กินเวลาจริงที่ทำการจำลองการทำงาน หากเน็ตเวอร์คคอมโพเน้นใดต้องการที่จะใช้เวลาที่ใช้ในการจำลองการทำงานเพื่อจัดการกับแพคเก็ต (เช่น ต้องการดีเลย์) ให้ทำการเรียกใช้ตัวจัดลำดับการทำงานโดยการสร้างเหตุการณ์สำหรับแพคเก็ตนั้นให้เกิดการดีเลย์ขึ้น



รูปที่ 4.2 การจับคู่กัน(Duality)ของ C++ และ OTcl

Ns2 ถูกพัฒนาขึ้นมาโดยใช้ 2 ภาษารวมกันคือ Otcl และ C++ เพื่อต้องการให้เกิดประสิทธิภาพสูงสุด Ns2 ได้ทำการแยกส่วนที่ต้องทำงานเกี่ยวกับข้อมูลออกจากส่วนที่ทำงานเกี่ยวกับการควบคุม เพื่อที่จะทำการลดเวลาในการจัดการกับแพคเก็ตและเหตุการณ์ต่างๆ (ไม่ใช่เวลาที่ใช้ในการจำลองการทำงาน) โดยตัวจัดลำดับเหตุการณ์และออปเจ็คเบื้องต้นที่เกี่ยวกับเครือข่ายที่ต้องประมวลผลข้อมูลจะถูกพัฒนาและคอมไพล์โดยใช้ C++ ออปเจ็คที่ถูกคอมไพล์เหล่านี้จะถูกทำให้สามารถเรียกใช้ผ่านทางตัวแปลภาษา Otcl ได้โดยทำการใช้ Otcl linkage ที่จะทำการเชื่อมโยงและจับคู่ระหว่าง ออปเจ็คของOTcl และ ออปเจ็คของ C++ และทำให้ฟังค์ชันควบคุม

ต่างๆรวมทั้งตัวแปลต่างๆ ที่อยู่ในออปเจ็คของC++ ทำหน้าที่เหมือนเป็นฟังค์ชัน และตัวแปรของออปเจ็ค Otcl ที่คู่กัน ซึ่งการทำเช่นนี้จะทำให้เราสามารถควบคุมออปเจ็คของ C++ โดยผ่านทางออปเจ็คของ Otcl ที่เชื่อมโยงกันอยู่ ออปเจ็คของ C++ ที่ไม่จำเป็นต้องถูกควบคุมในการจำลองการทำงานของเครือข่าย หรือว่าถูกใช้โดยอีกออปเจ็คหนึ่ง เราก็ไม่จำเป็นต้องทำการเชื่อมโยงกับ OTcl เช่นเดียวกัน ออปเจ็คที่ไม่ได้ทำงานเกี่ยวข้องกับข้อมูลสามารถที่จะเขียนโดยใช้ Otcl ล้วนๆ ไม่จำเป็นต้องทำการเชื่อมโยงกับออปเจ็คของC++ ดังรูปที่ 4.2 แสดงลำดับชั้นของออปเจ็คใน C++ และ OTcl จากรูปจะสังเกตเห็นว่า ออปเจ็คของ C++ ที่มีการเชื่อมโยงกับ Otcl จะมีการเชื่อมโยงกันเป็นลำดับชั้นที่เป็นคู่กัน

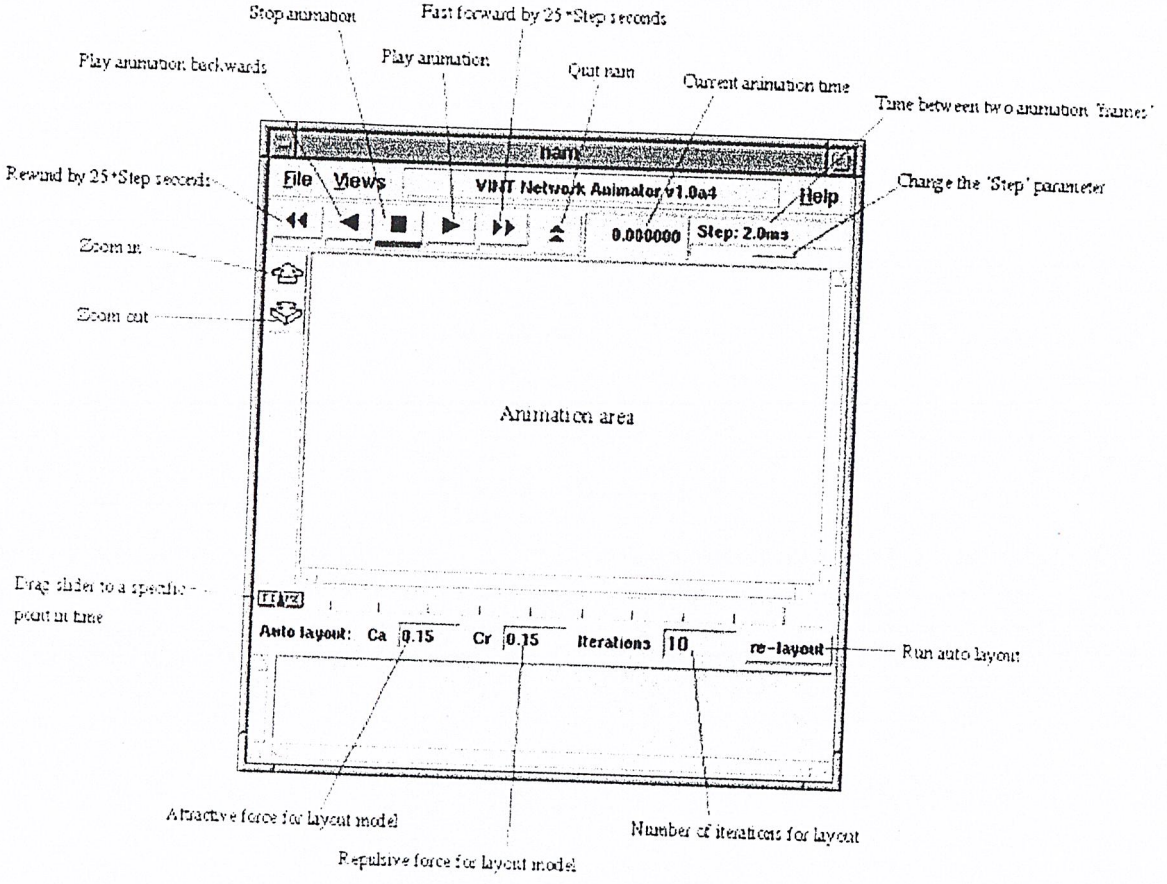


รูปที่ 4.3 สถาปัตยกรรมของ Ns2

รูปที่ 4.3 แสดงให้เห็นถึงสถาปัตยกรรมโดยทั่วไปของ Ns2 ในรูปของผู้ใช้ทั่วไป (ไม่ใช่ผู้ที่จะทำการพัฒนา Ns2) เริ่มพิจารณาจากมุมล่างซ้าย ทำการออกแบบและทำการจำลองการทำงานของเครือข่ายในภาษา Tcl โดยใช้ซิมูเลเตอร์ออปเจ็ค (simulator objects) ในไลบรารีของ OTcl โดยตัวจัดลำดับเหตุการณ์ และเน็ตเวอร์คคอมโพเนนต์ส่วนใหญ่ จะถูกพัฒนาโดยใช้ภาษา C++ และทำการเชื่อมโยงกับ OTcl โดยใช้ตัวเชื่อมโยงกับ OTcl (OTcl linkage) ที่ถูกพัฒนาขึ้นมาโดยใช้ tclcl และเมื่อทำการรวมทุกสิ่งทุกอย่างเข้าด้วยกันก็จะทำให้ได้ Ns2 ออกมา หรืออาจกล่าวได้ว่า Ns2 ก็คือ ตัวแปลภาษา OTcl ที่ประกอบด้วยไลบรารี่ของ Ns

ในส่วนที่ผ่านมาเราได้กล่าวอย่างย่อๆถึงโครงสร้างทั่วไป และสถาปัตยกรรมพื้นฐานของ Ns2 ต่อมาเราจะกล่าวถึงผลลัพธ์ที่ได้ภายหลังการจำลองการทำงานของเครือข่ายโดยใช้ Ns2 ดังที่ได้แสดงในรูปที่ 4.1 เมื่อการจำลองการทำงานจบลง Ns2 จะทำการสร้างผลลัพธ์ออกมาในรูปแบบที่เป็นตัวอักษร ที่จะประกอบไปด้วยรายละเอียดต่างๆในการจำลองการทำงาน โดยเราสามารถนำข้อมูลเหล่านี้มาทำการวิเคราะห์ หรือนำไปเป็นอินพุตให้กับเครื่องมือที่ใช้ในการแสดงผลแบบ

กราฟฟิคของการจำลองการทำงานของเครือข่าย ที่ชื่อว่า Network Animator (NAM) ซึ่ง NAM นั้นเป็นส่วนหนึ่งของ VINT Project โดย NAM จะมีส่วนติดต่อกับผู้ใช้ที่เป็นแบบกราฟฟิคที่มีหน้าตาคล้ายกับเครื่องเล่นซีดี (play, fast forward, rewind, pause และอื่นๆ) นอกจากนี้ยังมีตัวควบคุมเวลาที่ใช้ในการแสดงผล



รูปที่ 4.4 Network Animator (NAM)

4.2 OTcl: The User Language

ดังที่ได้กล่าวถึงในตอนต้น Ns2โดยพื้นฐานแล้วก็คือ ตัวแปลภาษา OTcl ที่ประกอบด้วยไลบรารี่ของออปเจ็คที่เป็นเน็ตเวอร์คซิมูเลเตอร์ ในการที่เราจะทำการจำลองการทำงานของเครือข่ายโดยใช้ Ns2 ได้นั้นเราจำเป็นต้องทำการเขียนโปรแกรมในภาษา OTcl เพื่อใช้งาน Ns2 โดยในส่วนนี้เราจะแสดงตัวอย่างการเขียนสคริปต์ภาษา Tcl และ OTcl เบื้องต้น โดยตัวอย่างที่นำมาแสดงนี้นำมาจาก VINT/NS Simulation Tutorial/Workshop (ครั้งที่ 5) โดยในการที่จะทำการรันโปรแกรมในตัวอย่างได้นั้นผู้ใช้ต้องทำการติดตั้ง Ns2 เสียก่อน

ตัวอย่างที่ 4.1 แสดงถึงลักษณะโดยทั่วไปของสคริปต์ภาษา Tcl ที่จะแสดงให้เห็นถึงวิธีการที่จะสร้างโพรซิเยอร์ (procedure) และวิธีการที่จะเรียกใช้โพรซิเยอร์นั้น วิธีการที่จะกำหนดค่าให้กับตัวแปร วิธีการที่จะสร้างลูป โดยภาษา OTcl นั้นเป็นภาษาที่พัฒนาขึ้นมาจากภาษา Tcl ดังนั้น คำสั่งที่มีใน Tcl ก็สามารถใช้ได้ใน OTcl เหมือนใน C และ C++ และวิธีการที่จะทำการรันสคริปต์สามารถทำได้โดย พิมพ์คำสั่ง "ns ex-tcl.tcl" ลงไปที่พร้อมพ์ของเชลล์ในระบบ ในตัวอย่างนี้สมมติให้ไฟล์สคริปต์ชื่อ ex-tcl.tcl คำสั่ง "ns" เป็นการเริ่มการทำงานของ Ns2 (ซึ่งก็คือตัวแปลภาษา OTcl)

```
# Writing a procedure called "test"
proc test {} {
   set a 43
   set b 27
   set c [expr $a + $b]
   set d [expr [expr $a - $b] * $c]
   for {set k 0} {$k < 10} {incr k} {
         if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
         } else {
            puts "k >= 5, mod = [expr $d % $k]"
         }
   }
}
# Calling the "test" procedure created above
test
```

ตัวอย่างที่ 4.1 สคริปต์ภาษา Tcl เบื้องต้น

ในภาษา Tcl คำหลัก proc ใช้ในการกำหนด โพรซิเยอร์ จากนั้นตามด้วยชื่อของโพรซิเยอร์ และ อากิวเมนต์ในวงเล็บปีกกา คำหลัก set ใช้ในการกำหนดค่าเริ่มต้นให้กับตัวแปร [expr ...] ใช้เพื่อให้ตัวแปลภาษาทำการคำนวณค่าของเอ็กเพรสชั่น (expression) ภายในวงเล็บหลังจากคำหลัก expr จะสังเกตเห็นว่าวิธีการที่จะเอาค่าที่ได้ทำการกำหนดให้ตัวแปลไปแล้วมาใช้ทำได้โดยการใช้

เครื่องหมาย $ ร่วมกับชื่อของตัวแปร คำหลัก puts ใช้ในการแสดงผลข้อความที่อยู่ภายในเครื่องหมายคำพูด โดยผลลัพธ์ที่ได้จากตัวอย่างที่ 4.1 จะเป็นดังนี้

k < 5, pow = 1.0

k < 5, pow = 1120.0

k < 5, pow = 1254400.0

k < 5, pow = 1404928000.0

k < 5, pow = 1573519360000.0

k >= 5, mod = 0

k >= 5, mod = 4

k >= 5, mod = 0

k >= 5, mod = 0

k >= 5, mod = 4

ในตัวอย่างถัดไป จะเป็นจะตัวอย่างการเขียนโปรแกรมแบบออปเจ็ค-โอเรียนเต็ด (object-oriented programming)ในภาษา OTcl โดยในตัวอย่างนี้เป็นตัวอย่างอย่างง่าย แต่แสดงให้เห็นถึงวิธีการที่จะสร้างและใช้งานออปเจ็ค ในภาษา OTcl

```
# Create a class call "mom" and
# add a member function call "greet"
Class mom
mom instproc greet {} {
   $self instvar age_
   puts "$age_ years old mom say:
   How are you doing?"
}

# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
   $self instvar age_
   puts "$age_ years old kid say:
   What's up, dude?"
}

# Create a mom and a kid object set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object
$a greet
$b greet
```

ตัวอย่างที่ 4.2 สคริปต์ภาษา OTcl เบื้องต้น

ตัวอย่างที่ 4.2 เป็นสคริปต์ภาษา OTcl ที่ทำการนิยามคลาส mom และ kid ที่ซึ่ง kid เป็นคลาสลูกของคลาส mom และมีฟังก์ชันสมาชิกชื่อ greet ในแต่ละคลาส หลังจากได้ทำการประกาศ

คลาส และประกาศอินสแตนส์ของออปเจ็ค ตัวแปลชื่อ age ของแต่ละอินสแตนส์ถูกกำหนดค่าให้เป็น 45 (สำหรับmom) และ 15 (สำหรับ kid) และสมาชิกฟังค์ชันที่ชื่อ greet ของแต่ละอินสแตนส์ได้ถูกเรียกใช้งาน คำหลัก Class ใช้ในการสร้างคลาส คำหลัก instproc ใช้ในการกำหนดฟังก์ชันสมาชิก คลาสสามารถทำการสืบทอด (inheritance) โดยการระบุคำหลัก –superclass ในการนิยามฟังก์ชันสมาชิก $self ทำหน้าที่เหมือนกับ พอยเตอร์ this ในภาษา C++ และคำหลัก instvar ตรวจสอบว่าชื่อตัวแปลที่ตามหลังคำหลักinstvar ได้ถูกนิยามไว้หรือยังในคลาสนั้นหรือซุปเปอร์คลาสของคลาสนั้น ถ้าตัวแปลนั้นได้ถูกประกาศไว้แล้ว ตัวแปลนั้นจะถูกอ้างอิง แต่ถ้ายังไม่มีการประกาศไว้ตัวแปรนั้นจะทำการประกาศ ในการที่จะสร้างอินสแตนท์ของออปเจ็คสามารถทำได้โดยใช้คำหลัก new ดังที่แสดงในตัวอย่าง เมื่อทำการรันไฟล์ตัวอย่างจะได้ผลดังนี้

45 year old mom say:

How are you doing?

15 year old kid say:

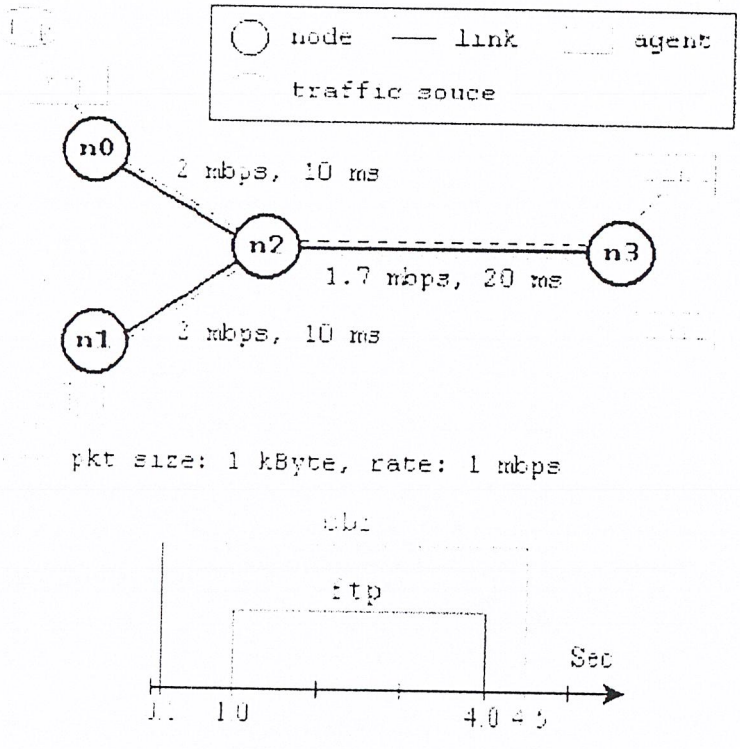
What' s up, dude?

4.3 ตัวอย่างการจำลองอย่างง่าย

ในส่วนนี้จะทำการแสดงให้เห็นถึงตัวอย่างการจำลองการทำงานของเครือข่ายโดยใช้ Ns2 และทำการอธิบายความหมายของคำสั่งที่ใช้ ในตัวอย่างที่ 4.3 นี้แสดงการใช้สคริปต์ OTcl เพื่อทำการเขียนโค้ดที่ใช้ในการกำหนดค่าต่างๆ ที่ใช้ในการจำลองการทำงานของเครือข่าย โดยสมมติให้ไฟล์ชื่อ ns-simple.tcl สามารถทำการรันโดยพิมพ์คำสั่ง “ns ns-simple.tcl” ที่พร้อมพ์ของเชลล์

เครือข่ายนี้ประกอบด้วย 4 โหนด (n0, n1, n2, n3) ดังที่แสดงในรูปข้างล่าง การเชื่อมต่อแบบดูเพล็ก(duplex) ระหว่าง n0 และ n2 รวมทั้ง n1 และ n2 มีขนาดแบนวิดธ์เท่ากับ 2 Mbps มีค่าดีเลย์เท่ากับ 10 ms การเชื่อมต่อแบบดูเพล็กระหว่าง n2 และ n3 มีแบนวิดธ์ขนาด 1.7 Mbps มีค่าดีเลย์เท่ากับ 20 ms แต่ละโหนดใช้คิวแบบ DropTail ที่มีขนาดคิวสูงสุดเท่ากับ 10 โดยทำการเชื่อมต่อ (attached) เอเจ้นท์ TCP เข้ากับโหนด n0 และทำการเชื่อมต่อ เอเจ้นท์ TCP แบบซิ้งค์ (sink) เข้ากับโหนด n3 โดยค่าปกติแล้ว ขนาดสูงสุดของแพคเก็ต ที่เอเจ้นท์ TCP สามารถสร้างคือ 1 KByte โดยเอเจ้นท์ TCP แบบซิ้งค์จะทำการสร้างและส่งแพคเก็ตชนิด ACK กลับไปยังผู้ส่ง (เอเจ้นท์ TCP) และทำการละทิ้งแพคเก็ตที่ได้รับ นอกจากนี้แล้ว เอเจ้นท์ UDP ยังได้ถูกทำการเชื่อมต่อเข้ากับโหนด n1 และโหนด n1 ก็ทำการเชื่อต่ออยู่กับโหนด n3 โดยที่โหนด n3 ทำการเชื่อมต่ออยู่

กับ เอเจ้นท์ชนิด Null โดยเอเจ้นท์ Null จะทำการระทิ้งแพคเก็ตที่ได้รับ ตัวกำเนิดทราฟฟิคแบบ ftp และ cbr ได้เชื่อมต่ออยู่กับเอเจ้นท์ TCP และเอเจ้นท์ UDP ตามลำดับ **โดย**ตัวกำเนิดทราฟฟิคชนิด cbr ได้ถูกกำหนดค่าให้ทำการสร้างแพคเก็ตขนาด 1 Kbyte ด้วยอัตรา 1 Mbps และ cbr จะเริ่มทำงานที่เวลา 0.1วินาที หยุดการทำงานที่เวลา 4.5 วินาที ftp จะเริ่มการทำงาน ที่เวลา 1.0 วินาที หยุดการทำงานที่เวลา 4.0 วินาที



รูปที่ 4.5 โทโพโลยี่ และ เหตุการณ์ที่ใช้ในการจำลองการทำงานของเครือข่าย

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

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

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

#Define a 'finish' procedure
proc finish {} {
```

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

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

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

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

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

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

#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
```

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

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

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

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

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
```

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

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

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

#Run the simulation
$ns run
```

ตัวอย่างที่ 4.3 สคริปต์จำลองการทำงานใน Ns2

โดยปกติสคริปต์ที่ใช้ควบคุมการทำงานของ Ns2 จะเริ่มต้นด้วยการสร้างอินสแตนท์ของออปเจ็คซิมูเลเตอร์ โดยข้อความข้างล่างเป็นคำอธิบายของสคริปต์ข้างบน

set ns [new Simulator]:ใช้ในการสร้างอินสแตนท์ของออปเจ็คซิมูเลเตอร์ และทำการกำหนดค่าให้กับตัวแปลชื่อ ns สิ่งที่บรรทัดนี้กระทำมีดังนี้

- กำหนดค่าเริ่มต้นให้กับแพคเก็ต
- สร้าง scheduler
- เลือกค่าปกติของรูปแบบที่อยู่ (address format)

ออปเจ็คซิมูเลเตอร์มีฟังก์ชันสมาชิกที่จะทำหน้าที่ดังนี้

- ทำการสร้างคอมพาวด์ออปเจ็ค เช่น โหนด และ ลิ้งค์
- ทำการเชื่อมต่อคอมโพเนนต์ของเครือข่ายที่ได้สร้างขึ้นมา (เช่น คำสั่ง attach-agent)
- ทำการกำหนดพารามิเตอร์ให้กับคอมโพเนนต์ของเครือข่าย
- สร้างการเชื่อมต่อระหว่างเอเจ้นท์ (เช่น ทำการเชื่อมต่อระหว่าง TCP และ SINK)
- กำหนดตัวเลือกในการแสดงผลของ NAM
- และอื่นๆ

การนิยามฟังก์ชันสมาชิกของออปเจ็คซิมมูเลเตอร์จะอยู่ที่ไฟล์ ns-2/tcl/lib/ns-lib.tcl

$ns color fid color: ใช้ในการกำหนดสีของแพคเก็ตในแต่ละโฟลว์ ที่ระบุโดย fid เมมเบอร์ฟังค์ชันนี้ซึ่งเป็นของออปเจ็คซิมูเลเตอร์ ใช้ในการกำหนดการแสดงผลของ NAM และไม่มีผลต่อการจำลองการทำงานของเครือข่าย

$ns namtrace-all file-descriptor: ฟังก์ชันสมาชิกนี้จะทำการบอกซิมูเลเตอร์ให้ทำการบันทึกผลของการจำลองการทำงานของเครือข่ายในรูปแบบของอินพุทไฟล์ของ NAM

proc finish{}: โพรซิเยอร์นี้จะต้องถูกเรียกใช้เมื่อการจำลองการทำงานของเครือข่ายได้เสร็จสิ้นลง โดยใช้คำสั่ง $ns at 5.0 "finish" โดยที่ภายในโพรซิเยอร์นี้จะมีขั้นตอนที่ต้องทำหลังจากการจำลองการทำงานของเครือข่ายเสร็จสิ้นลงระบุไว้

set n0 [$ns node]: ฟังก์ชันสมาชิกที่ชื่อ node จะทำการสร้างโหนด โดยโหนดใน Ns2 จะเป็นออปเจ็คชนิดคอมพาวด์ ที่ประกอบด้วย ตัวคลาสสิฟาย (classifier)โหนดและตัวคลาสสิฟายพอร์ท ผู้ใช้สามารถสร้างโหนดโดยทำการแยกตัวคลาสสิฟายพอร์ทและตัวคลาสสิฟายแอดเดรสออกจากกันแล้วจึงนำออปเจ็คทั้งสองมาเชื่อมต่อกัน แต่ถึงอย่างไรก็ตามการใช้ฟังก์ชันสมาชิกตัวนี้จะทำให้การทำงานง่ายขึ้น หากผู้ใช้ต้องการรู้ว่าโหนดถูกสร้างขึ้นมาได้อย่างไรสามารถดูได้ที่ "ns-2/tcl/libs/ns-lib.tcl" และ "ns-2/tcl/libs/ns-node.tcl"

$ns duplex-link node1 node2 bandwidth delay queue-type: ทำการสร้างลิ้งค์ชนิดซิมเพล็ก (simplex) จำนวน 2 ลิ้งค์ ที่มีค่าแบนด์วิธ และ ดีเลย์ตามที่ระบุ และทำการเชื่อมลิ้งค์กับโหนดทั้ง 2 โครงสร้างของNs2 เอาท์พุทคิวของโหนดจะถูกกำหนดให้เป็นส่วนหนึ่งของลิ้งค์ ดังนั้นผู้ใช้จะต้องทำการกำหนดชนิดของคิวเมื่อทำการสร้างลิ้งค์ โดยตัวอย่างโค้ดที่ใช้ในการจำลองการทำงานของเครือข่ายในตัวอย่างใช้คิวชนิดดร็อปเทล (DropTail) ไฟล์โค้ดที่ใช้ในการอิมพลีเมนต์ลิ้งค์สามารถดูได้ที่ "ns-2/tcl/libs/ns-lib.tcl" และ "ns-2/tcl/ns-link.tcl"

$ns queue-limit node1 node2 number : บรรทัดนี้จะทำการกำหนดค่าลิมิตของคิวของซิมเพล็กลิ้งค์ 2 ตัวที่เชื่อมต่อระหว่างโหนด 1 และ โหนด 2 โดยสามารถดูโค้ดเพิ่มเติมได้ที่ "ns-2/tcl/libs/ns-lib.tcl" และ "ns-2/tcl/libs/ns-link.tcl"

$ns duplex-link-op node1 node2 ... : บรรทัดเหล่านี้ใช้ในการกำหนดค่าที่ใช้ในการแสดงผลของ NAM หากต้องการดูผลของคำสั่งผู้ใช้สามารถทำการคอมเมนต์บรรทัดเหล่านี้ แล้วทำการจำลองการทำงานของเครือข่ายดูอีกครั้ง

เมื่อมาถึงขั้นนี้การทำการกำหนดค่าเบื้องต้นของเครือข่ายได้ทำเสร็จสิ้นแล้ว สิ่งต่อไปที่ต้องทำคือ ทำการกำหนดค่าเอเจ้นท์ของทราฟฟิคเช่น TCP และ UDP ทำการกำหนดค่าแหล่งกำเนิดทราฟฟิคเช่น FTP และ CBR

set tcp [new Agent/TCP] : บรรทัดนี้แสดงให้เห็นถึงวิธีการที่จะทำการสร้างเอเจ้นท์ TCP ซึ่งโดยปกติแล้วผู้ใช้สามารถทำการสร้างเอเจ้นท์ หรือตัวกำเนิดทราฟฟิคใดๆ ด้วยวิธีนี้ เอเจ้นท์และแหล่งกำเนิด ทราฟฟิคในความเป็นจริงแล้วเป็นออปเจ็คธรรมดา (ไม่ใช่คอมพาวด์ออปเจ็ค) โดยมากแล้วจะทำการอิมพลีเมนต์โดยใช้ C++ แล้วทำการเชื่อมโยงกับ OTcl การที่จะทำการสร้างเอเจ้นท์และตัวกำเนิดทราฟฟิคผู้ใช้จะต้องรู้จักชื่อคลาสของออปเจ็คเหล่านี้ (Agent/TCP, Agent/TCPSink, Application/FTP และอื่นๆ) ข้อมูลเหล่านี้สามารถดูได้ที่คู่มือของ Ns2 หรือสามารถดูอย่างคล่าวๆได้ที่ "ns-2/tcl/libs/ns-default.tcl" ไฟล์นี้จะเป็นไฟล์ที่เก็บค่าปกติของค่าพารามิเตอร์ต่างๆ ที่ใช้ในการกำหนดค่าออปเจ็คเครือข่าย ดังนั้นไฟล์นี้ตัวบ่งชี้ว่ามีออปเจ็คของเครือข่ายชนิดใดบ้างที่สามารถใช้ได้ใน Ns2 และออปเจ็คเหล่านี้มีค่าพารามิเตอร์อะไรบ้าง

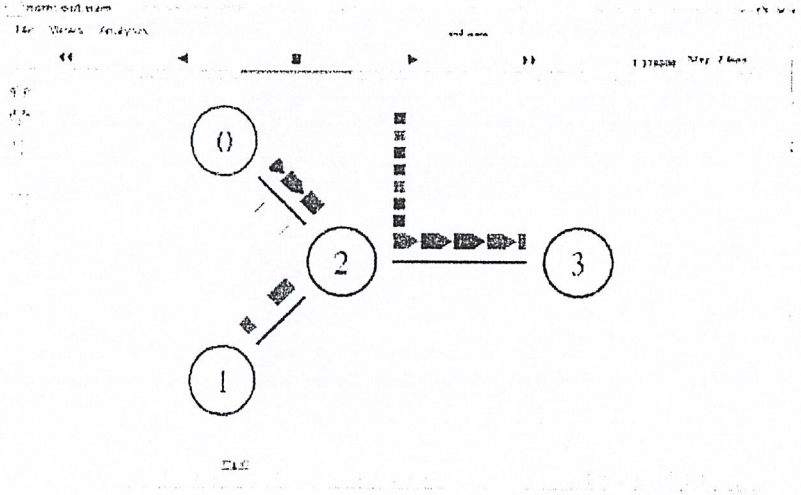
$ns attach-agent node agent : ฟังก์ชันสมาชิกที่ชื่อ attach-agent จะทำการเชื่อมต่อเอเจ้นท์เข้ากับโหนด แท้ที่จริงแล้วสิ่งที่ฟังค์ชันนี้ทำ คือการเรียกฟังก์ชันสมาชิกที่ชื่อ attach ของโหนดที่ระบุ ที่จะทำการเชื่อมเอเจ้นท์เข้ากับตนเอง ดังนั้นผู้ใช้สามารถทำสิ่งเดียวกันนี้โดยใช้คำสั่ง "$n0 attach $tcp" และเช่นเดียวกับที่กล่าวมาตัวเอเจ้นท์ก็มีฟังก์ชันสมาชิก attach-agent ที่จะทำการเชื่อมแหล่งกำเนิดทราฟฟิคเข้ากับตัวมันเอง

$ns connect agent1 agent2 : หลังจากที่เอเจ้นท์สองตัวที่จะทำการติดต่อกันได้ถูกสร้างขึ้น สิ่งต่อมาที่จะต้องทำคือทำการสร้างการเชื่อมโยงทางลอจิค (logical) ระหว่างทั้งคู่ สิ่งที่บรรทัดนี้ทำคือทำการสร้างการเชื่อมโยงระหว่างเครือข่ายโดยการกำหนดที่อยู่ปลายทางของแต่ละโหนด

เมื่อทำการกำหนดค่าเครือข่ายเบื้องต้นแล้ว สิ่งต่อมาที่จะต้องทำ คือทำการเขียนฉากที่จะทำการจำลอง ในออปเจ็คที่ชื่อซิมูเลเตอร์มีฟังก์ชันสมาชิกที่ทำงานเกี่ยวกับการจัดลำดับเหตุการณ์อยู่มากมาย ซึ่งฟังก์ชันสมาชิกข้างล่างนี้เป็นหนึ่งในฟังก์ชันสมาชิกที่ใช้บ่อยที่สุด

$ns at time “string” : ฟังก์ชันสมาชิกตัวนี้เป็นฟังก์ชันสมาชิกของออปเจ็คที่ชื่อซิมูเลเตอร์ที่ทำงานเกี่ยวกับการจัดลำดับเหตุการณ์ (ตัวแปล scheduler_ เป็นตัวแปลที่ทำการชี้ไปที่ออปเจ็คที่ชื่อ scheduler ที่ทำการสร้างโดยคำสั่ง [new Scheduler] ที่ส่วนต้นๆ ของสคริปต์ที่ใช้ในการจำลองการทำงานของเครือข่าย) ตัวอย่างเช่น คำสั่ง “$ns at 0.1 “$cbr start” จะทำให้ตัวจัดลำดับเหตุการณ์เรียกฟังก์ชันสมาชิกที่ชื่อ start ของตัวกำเนิดทราฟฟิค CBR ซึ่งจะทำให้ตัวกำเนิดทราฟฟิคทำการส่งข้อมูลออกมา ในการทำงานจริงๆ ของ Ns2 โดยปกติแล้วตัวกำเนิดทราฟฟิคจะไม่ทำการส่งข้อมูลออกมาจริงๆ แต่จะทำการแจ้งให้เอเจ้นท์ที่อยู่เบื้องล่างทราบว่ามันมีข้อมูลที่จะต้องทำการส่ง และเอเจ้นท์ก็เพียงแค่รู้ว่ามีข้อมูลจำนวนเท่าใดที่จะต้องทำการส่ง แล้วจึงทำการสร้างแพคเก็ตเหล่านั้นและทำการส่ง

หลังจากที่ได้ทำการกำหนดค่าเครือข่ายทั้งหมด และทำการกำหนดลำดับของเหตุการณ์รวมทั้งโพรซิเยอร์ที่จะต้องทำหลังจากการทำการจำลองการทำงานของเครือข่าย ก็เหลืออีกเพียงอย่างเดียวที่ต้องทำ คือทำการรัน ซึ่งสามารถทำได้โดยใช้คำสั่ง $ns run จะทำให้ได้ผลลัพธ์ดังนี้

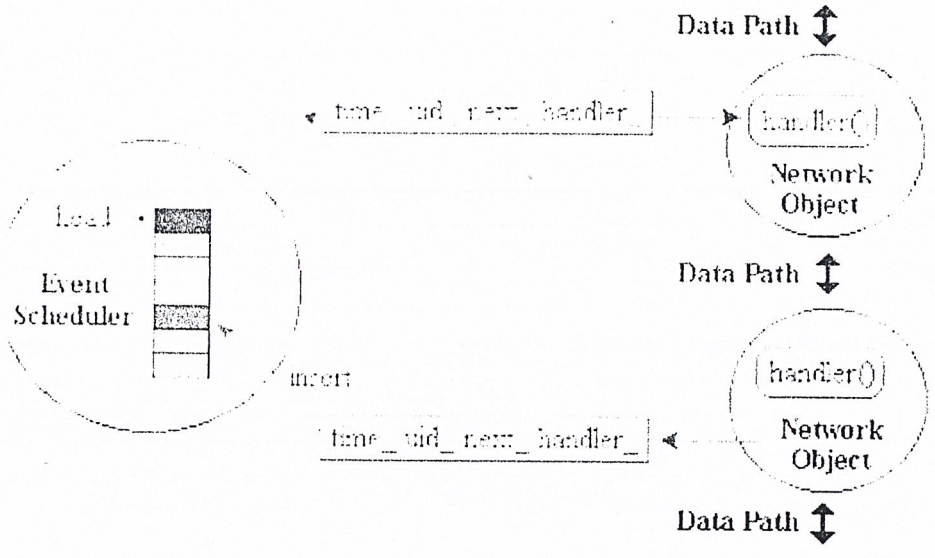


รูปที่ 4.4 ผลลัพธ์ที่ได้จากการจำลองการทำงานของเครือข่าย

4.4 ตัวจัดลำดับเหตุการณ์

ในส่วนนี้เราจะกล่าวถึงการจัดลำดับเหตุการณ์แบบไม่ต่อเนื่อง (discrete) ของ Ns2 ดังที่ได้กล่าวไปแล้วว่าผู้ใช้งานหลักของตัวจัดลำดับเหตุการณ์คือเน็ตเวอร์คคอมโพเน้นต์ต่างๆ ที่จะทำการ

จำลองการเกิดดีเลย์ของแพคเก็ต หรือในกรณีที่ต้องการใช้ไทม์เมอร์ (timer) รูปข้างล่างแสดงการใช้งานตัวจัดลำดับเหตุการณ์โดยออปเจ็คของเครือข่าย จะสังเกตเห็นว่าออปเจ็คของเครือข่ายที่ทำการสร้างเหตุการณ์จะต้องเป็นผู้ที่ทำการจัดการกับเหตุการณ์นั้นเองในภายหลังเมื่อเวลามาถึง นอกจากนี้แล้วจะสังเกตเห็นว่า เส้นทางของข้อมูลระหว่างออปเจ็คของเครือข่ายจะต่างจากเส้นทางของเหตุการณ์ โดยแท้จริงแล้ว แพคเก็ตจะถูกส่งจากออปเจ็คเครือข่ายตัวหนึ่งไปยังอีกตัวโดยใช้รูปแบบคำสั่งเป็น "send(Packet* p){target_ ->recv(p)}" สำหรับผู้ส่ง และ "recv(Packet*, Handler* h = 0)" สำหรับผู้รับ



รูปที่ 4.5 การจัดลำดับเหตุการณ์แบบไม่ต่อเนื่อง

Ns2 มีตัวจัดลำดับเหตุการณ์ 2 แบบคือตัวจัดลำดับเหตุการณ์แบบ เรียลไทม์ (real-time) และแบบ นอนเรียลไทม์ (non-real-time) โดยมีการอิมพลีเมนต์ไว้ 3 วิธีได้แก่ List, Heap และ Calendar โดยทั้ง 3 นี้มีการทำงานทางลอจิกเหมือนกัน เหตุผลที่ต้องมีถึง 3 แบบก็เพื่อให้เน็ตเวอร์คคอมโพเนนต์แบบเดิมที่พัฒนาโดยผู้ใช้เอง (ไม่ใช่ที่มีให้ในแพคเก็จของ Ns2) ที่อาจจะระบุชนิดของตัวจัดลำดับเหตุการณ์เองเอาไว้ โดยค่าปกติของตัวจัดลำดับเหตุการณ์จะเป็นแบบ นอนเรียลไทม์ชนิด Calendar ตัวอย่างข้างล่างเป็นตัวอย่างการระบุตัวจัดลำดับเหตุการณ์

```
...
set ns[new Simulator]
$ns use-scheduler Heap
...
```

นอกจากที่กล่าวมาแล้วตัวจัดลำดับเหตุการณ์ยังมีประโยชน์อีกคือ ใช้ในการจัดลำดับเหตุการณ์ที่ใช้ในการจำลองการทำงานของเครือข่ายเช่น เมื่อใดที่จะทำการเริ่มการทำงานของแอพพลิเคชัน FTP เมื่อใดที่การทำการจำลองการทำงานของเครือข่ายจะเสร็จสิ้นลง ตัวอย่างการจัดลำดับของเหตุการณ์ในตัวอย่างข้างล่างนี้เป็นเวอร์ชันเพิ่มเติมของตัวอย่างข้างบน

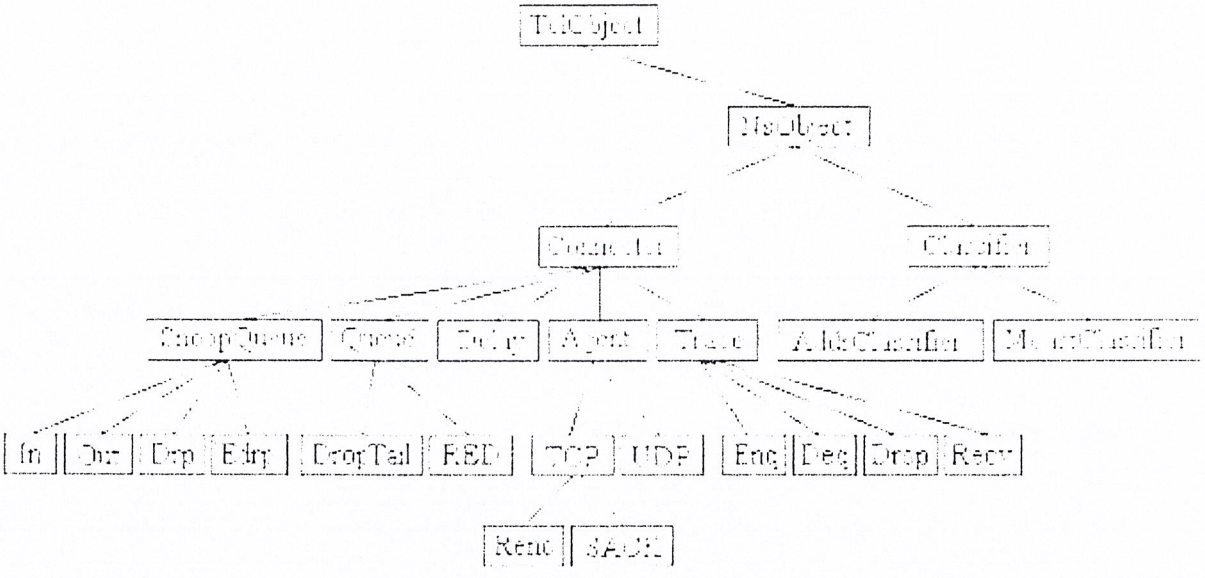
```
...
set ns[new Simulator]
$ns use-scheduler Heap
$ns at 300.5 "complete_sim"
...
proc complete_sim{}{
...
}
```

จากตัวอย่างข้างบนจะเห็นว่า at time "string" เป็นฟังค์ชันสมาชิกของออปเจ็คซิมมูเลเตอร์ (set ns [new Simulator]) โดยที่ออปเจ็คซิมมูเลเตอร์ทำหน้าที่สเมือนเป็นส่วนติดต่อกับผู้ใช้ ซึ่งออปเจ็คซิมมูเลเตอร์จะทำการเรียกฟังค์ชันสมาชิกของออปเจ็คเครือข่าย หรือออปเจ็ค scheduler เพื่อทำงานจริงๆ แทน ข้างล่างจะเป็นบางส่วนของฟังค์ชันสมาชิกของออปเจ็คซิมมูเลเตอร์ ที่ทำหน้าที่เชื่อมต่อกับฟังค์ชันสมาชิกของตัวจัดลำดับเหตุการณ์

```
Simulator instproc now          # คืนค่าเวลาปัจจุบันของตัวจัดลำดับเหตุการณ์
Simulator instproc at args      # กำหนดให้ตัวจัดลำดับเหตุการณ์ทำการประมวลผลคำสั่ง
                                # ที่เวลาที่ระบุ
Simulator instproc at-now args  # กำหนดให้ตัวจัดลำดับเหตุการณ์ทำการประมวลผลคำสั่งที่เวลา
                                # ปัจจุบัน
Simulator instproc after n args # กำหนดให้ตัวจัดลำดับเหตุการณ์ทำการประมวลผลคำสั่ง
ภายหลัง
                                # เวลา n วินาที
Simulator instproc run args     # เริ่มการทำงานของตัวจัดลำดับเหตุการณ์
Simulator instproc halt         # หยุดการทำงานของตัวจัดลำดับเหตุการณ์
```

4.5 เน็ตเวอร์คคอมโพเนนต์ (Network Component)

ในส่วนนี้จะพูดถึงคอมโพเนนต์ของ Ns2 ซึ่งโดยมากแล้วจะเป็นเน็ตเวิร์คคอมโพเนนต์ชนิดคอมพาวด์ รูปข้างล่างแสดงถึงบางส่วนของลำดับชั้นคลาสของ Otcl ใน Ns2 ซึ่งจะช่วยทำให้เข้าใจพื้นฐานของเน็ตเวอร์คอมโพเนนต์มากขึ้น ลำดับชั้นของคลาสที่สมบูรณ์สามารถดูได้ที่ http://www-sop.inria.fr/rodeo/personnel/Antoine.Clerget/ns



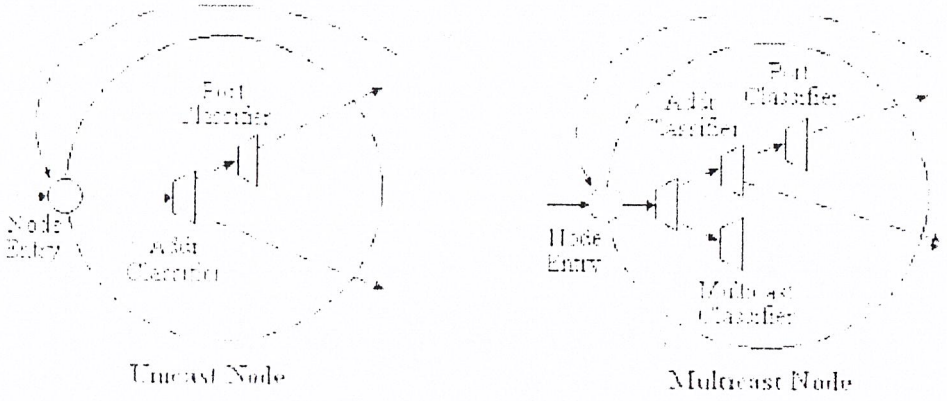
รูปที่ 4.6 ลำดับชั้นของคลาส(บางส่วน)

รากของลำดับชั้นทั้งหมดคือคลาส TclObject ซึ่งเป็นซุปเปอร์คลาส (superclass) ของออปเจ็คไลบรารี่ของOTclทั้งหมด (scheduler, เน็ตเวอร์คอมโพเนนต์, ไทเมอร์และออปเจ็คอื่นๆรวมทั้ง NAM) ในฐานะที่เป็นคลาสลูกของ TclObject คลาส NsObject จึงเป็นซุปเปอร์คลาสของออปเจ็คเน็ตเวอร์คคอมโพเนนต์ทั้งหมดที่จัดการเกี่ยวกับแพคเก็ต นอกจากนี้แล้วเน็ตเวอร์คอมโพเนนต์พื้นฐานยังแบ่งย่อยได้อีกเป็น 2 ชนิดคือ คอนเนคเตอร์ (connector) และคลาสซิฟายเออร์ (classifier) ขึ้นอยู่กับจำนวนของเอาท์พุท กล่าวคือออปเจ็คของเครือข่ายที่มีเพียงเอาท์พุทเดียวจะอยู่ภายใต้คลาสคอนเน็คเตอร์ และคลาสที่มีมากกว่า 1 เอาท์พุทจะอยู่ภายใต้คลาสคลาสซิฟายเออร์

4.6 โหนดและการค้นหาเส้นทาง

โหนดเป็นออปเจ็คชนิดคอมพาวด์ที่ประกอบด้วยออปเจ็ค node entry และ คลาสซิฟายเออร์ดังที่แสดงในรูปข้างล่าง โหนดใน Ns2 แบ่งออกได้เป็น 2 ชนิดคือ โหนดชนิดยูนิแคส (unicast) และมัลติแคส(multicast) โดยที่โหนดชนิดยูนิแคสจะประกอบด้วย คลาสซิฟายเออร์ของแอดเดรสที่

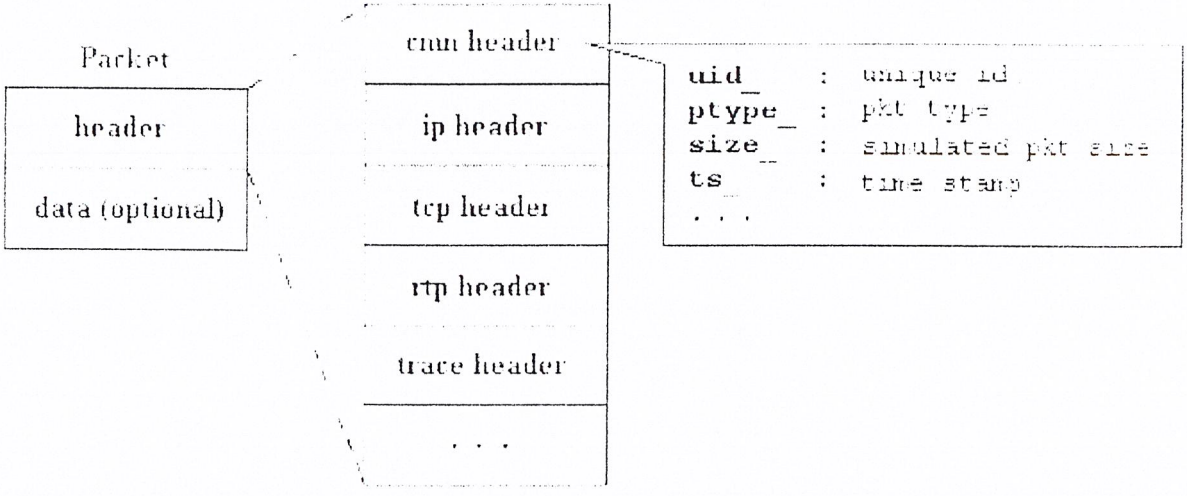
ใช้ในการค้นหาเส้นทางแบบยูนิแคสและตัวคลาสซิฟายเออร์ของพอร์ต โหนดชนิดมัลติแคสนอกเหนือจากคลาสซิฟายเออร์ที่มีในยูนิแคสโหนด แล้วยังมีคลาสซิฟายเออร์อีกตัวที่ทำการแยกแยะแพคเก็ตชนิดมัลติแคสออกจากแพ็คเก็ตชนิดยูนิแคส เพื่อใช้ในการค้นหาเส้นทางแบบมัลติแคส



รูปที่ 4.7 โหนด (ยูนิแคส และ มัลติแคส)

4.7 แพ็คเก็ต

แพคเก็ตใน Ns2 ประกอบด้วยสแตก (stack) ของเฮดเดอร์ (header) ในส่วนของข้อมูลนั้นอาจจะมีหรือไม่มีก็ได้ดังรูปข้างล่าง ดังที่ได้กล่าวไปแล้วในตัวอย่างสคริปต์ที่ใช้ในการจำลองการทำงานของเครือข่ายว่า เฮดเดอร์ของแพคเก็ตจะถูกกำหนดค่าเริ่มต้น เมื่อออปเจ็คที่ชื่อซิมูเลเตอร์ได้ถูกสร้างขึ้น ซึ่งสแตกของของเฮดเดอร์ที่ได้ทำการลงทะเบียน (register) ไว้เช่นเฮดเดอร์ทั่วๆไป ที่ถูกใช้โดยออปเจ็คต่างๆ เช่น เฮดเดอร์ IP เฮดเดอร์ TCP และเฮดเดอร์ Trace ได้ถูกนิยามและออฟเซ็ต (offset) ของแต่ละเฮดเดอร์ในสแตกได้ถูกทำการบันทึก ซึ่งที่ได้กล่าวมานี้หมายความว่า ไม่ว่าแต่ละเฮดเดอร์จะถูกใช้หรือไม่ สแตกของเฮดเดอร์ที่ได้ทำการลงทะเบียนไว้จะต้องถูกสร้างขึ้น และเมื่อแพคเก็ตถูกจับจองโดยเอเจ้นท์ออปเจ็คของเครือข่ายจะสามารถเข้าถึงเฮดเดอร์ใดๆ ของแพคเก็ตโดยผ่านทางค่าออฟเซ็ต



รูปที่ 4.8 รูปแบบแพ็คเก็ตใน Ns2

โดยปกติแล้วแพคเก็ตจะมีเพียงสแตกของเฮดเดอร์ (ในส่วนของข้อมูลจะเก็บค่าพ้อยเตอร์ที่เป็นนัลล์) ถึงแม้ว่าแพคเก็ตจะสามารถบรรจุข้อมูลจริงๆ เข้าไปได้ (จากแอพพลิเคชันใดๆ) โดยการจับจองเนื้อที่ในส่วนของข้อมูล แต่ในการอิมพลีเมนต์แอพพลิเคชันและเอเจ้นท์แทบจะไม่มีการรองรับคุณสมบัตินี้ เพราะมันไม่มีความหมายที่จะบรรจุข้อมูลในการจำลองการทำงาน ที่เป็นแบบนอนเรียล ไทม์ (non-real-time simulation) แต่ถึงอย่างไรก็ตาม ถ้าหากเราต้องการที่จะพัฒนาแอพพลิเคชันที่สามารถติดต่อกับอีกแอพพลิเคชันโดยผ่านทางเครือข่าย เราจำเป็นต้องใช้คุณสมบัตินี้โดยการแก้ไขโค้ดของเอเจ้นท์ หรืออาจจะใช้อีกวิธีหนึ่งคือการสร้างเฮดเดอร์ใหม่สำหรับแอพพลิเคชันนั้น แล้วทำการแก้ไขเอเจ้นท์ให้ทำการเขียนข้อมูลที่ได้รับจากแอพพลิเคชันนำไปใส่ในเฮดเดอร์ที่สร้างใหม่

บทที่ 5
การออกแบบโปรโตคอลการค้นหาเส้นทางของเครือข่ายแอดฮอค

ในการออกแบบโปรโตคอลการค้นหาเส้นทางของเครือข่ายแอดฮอคในโครงงานนี้ ไม่ได้เป็นการคิดค้นโปรโตคอลขึ้นมาใหม่ หากแต่เป็นการพิจารณาเปรียบเทียบถึงลักษณะเด่นของโปรโตคอลชนิดต่าง ๆ ที่ใช้กันอยู่ในปัจจุบัน และนำลักษณะเด่นเหล่านั้นมาพัฒนาการค้นหาเส้นทางในเครือข่ายแอดฮอค

ลำดับขั้นในการพิจารณาเลือกโปรโตคอลมีใช้มีขั้นตอน ดังนี้

ขั้นที่ 1 ทำการพิจารณาลักษณะการเลือกเส้นทางระหว่างอัลกอริธึมแบบ Distance Vector และ Link state เมื่อมองไปถึงปัญหาที่พบในอัลกอริธึมการเลือกเส้นทางแต่ละชนิด พบว่า ควรเลือกอัลกอริธึมการเลือกเส้นทางแบบ Distance Vector ถึงแม้ว่าอาจทำให้เกิดลูป ซึ่งทำให้เกิดการสิ้นเปลืองแบนด์วิธ และกำลังงาน แต่ก็เกิดโอเวอร์เฮดน้อยกว่าอัลกอริธึมแบบ Link State

ขั้นที่ 2 พิจารณาลักษณะของโปรโตคอลการค้นหาเส้นทางซึ่งมี 2 ประเภท ได้แก่ Table-Driven หรือ Proactive และ On-Demand Driven หรือ Reactive ดังที่ได้กล่าวไปแล้วในบทที่ 3 เมื่อเปรียบเทียบกันแล้วพบว่า โปรโตคอลประเภท On-Demand Driven มีความเหมาะสมกับเครือข่ายแอดฮอคมากกว่า เพราะโปรโตคอลประเภทนี้จะมีการสร้างเส้นทางก็ต่อเมื่อมีความต้องการใช้เท่านั้น ซึ่งเป็นการลดโอเวอร์เฮดในเครือข่ายให้ลดลง ในขณะที่โปรโตคอลประเภท Table-Driven จะต้องอัพเดตข้อมูลทุกๆช่วงระยะเวลา

ขั้นที่ 3 การใช้โปรโตคอลที่ลักษณะการทำงานเป็นแบบลำดับชั้น (Hierarchical Routing Protocol) เป็นการช่วยลดปริมาณของข้อมูลในตารางการค้นหาเส้นทางที่แต่ละโหนดจะต้องเก็บ

ขั้นที่ 4 เมื่อพิจารณาโปรโตคอลจากการแบ่งประเภทโปรโตคอลในลักษณะต่างๆแล้วพบว่า โปรโตคอลที่มีความเหมาะสมกับเครือข่ายแอดฮอคควรมีลักษณะดังนี้

- มีอัลกอริธึมการเลือกเส้นทางแบบ Distance Vector
- เป็นโปรโตคอลประเภท On-Demand Driven
- เป็นโปรโตคอลประเภทลำดับชั้น

ขั้นที่ 5 จากคุณสมบัติที่ควรมีของโปรโตคอลในเครือข่ายแอดฮอคในขั้นที่ 4 จึงทำให้ในโครงงานนี้ เลือกที่จะพัฒนาโปรโตคอล โดยปรับแต่งให้โปรโตคอลนี้ สามารถทำงานเป็นโปรโตคอลประเภท On-Demand Driven ได้ ซึ่งจะมีการกำหนดคุณสมบัติขององค์ประกอบต่างๆ ดังที่จะแสดงในหัวข้อต่อๆไป

5.1 สถานะของโหนด

ในโครงงานนี้สามารถแบ่งสถานะของโหนดได้เป็น 4 สถานะ ได้แก่

1. โหนดคลัสเตอร์เฮด หรือคลัสเตอร์เฮด เป็นโหนดที่ทำหน้าที่ควบคุมโหนดสมาชิกที่อยู่ภายในคลัสเตอร์เดียวกัน
2. โหนดเกตเวย์ หรือเกตเวย์ เป็นโหนดที่เป็นสมาชิกของคลัสเตอร์ตั้งแต่ 2 คลัสเตอร์ขึ้นไป มีหน้าที่เป็นตัวกลางในการเชื่อมต่อการรับส่งข้อมูลระหว่างคลัสเตอร์
3. โหนดสมาชิก ได้แก่ โหนดต่างๆที่อยู่ภายในคลัสเตอร์
4. โหนดไม่มีสถานะ (Initial Node) คือ โหนดใหม่ที่ยังไม่มีสถานะใดสภานะหนึ่งใน 3 สถานะข้างต้น และยังไม่สังกัดคลัสเตอร์ใดๆ

5.2 เงื่อนไขของเส้นทางการรับส่งข้อมูล

เมื่อมีโหนดภายในเครือข่ายต้องการส่งข้อมูล ถ้าโหนดปลายทางอยู่ในเครือข่าย พบว่ามีความเป็นไปได้ของการพบโหนดปลายทางสองลักษณะ ได้แก่

- พบโหนดปลายทางภายในการบรอดแคสต์ครั้งแรก
- ไม่พบโหนดปลายทางภายในการบรอดแคสต์ครั้งแรก

ซึ่งในโปรโตคอลนี้ ได้อ้างการรับส่งข้อมูลในลักษณะของคลัสเตอร์ที่มีคลัสเตอร์เฮดเป็นโหนดที่ควบคุมโหนดสมาชิกอื่นๆ ภายในคลัสเตอร์ของตน รวมทั้งการรับส่งข้อมูลด้วย โดยจะสามารถรับส่งข้อมูลผ่านทางคลัสเตอร์เฮด และโหนดเกตเวย์ได้เท่านั้น ดังนั้นการที่โหนดต้นทางต้องการส่งข้อมูลไปยังโหนดปลายทางจะต้องผ่านคลัสเตอร์เฮดของตนเสมอ อย่างไรก็ตาม ได้มีข้อยกเว้นในกรณีที่โหนดต้นทางและโหนดปลายทางอยู่ภายในระยะการบรอดแคสต์ครั้งแรก หรือระยะ 1 ฮอป สามารถส่งข้อมูลหากันโดยไม่ต้องผ่านคลัสเตอร์เฮดได้

5.3 กระบวนการค้นหาเส้นทาง

การค้นหาเส้นทางเกิดขึ้นเมื่อมีโหนดภายในเครือข่ายที่ต้องการส่งข้อมูล (ในที่นี้จะเรียกว่าโหนดต้นทาง) โหนดนั้นจะทำการตรวจดูในตารางการค้นหาเส้นทางของตนเองว่า มีข้อมูลเส้นทางไปยังโหนดปลายทางที่ต้องการส่งข้อมูลไปหรือไม่ ถ้ามีข้อมูลเส้นทางอยู่และเป็นข้อมูลที่ใหม่เพียงพอ โหนดต้นทางจะสามารถส่งข้อมูลไปยังโหนดปลายทางได้ทันที แต่ถ้าค้นหาดูในตารางการค้นหาเส้นทางแล้วพบว่า ไม่มีข้อมูลเส้นทางอยู่ โหนดต้นทางจะต้องค้นหาเส้นทางที่เหมาะสมที่สุด ซึ่งในโปรโตคอลนี้ใช้วิธี Distance Vector โดยการบรอดแคสต์ข้อความ RREQ ออกไปในเครือข่าย

ซึ่งมีความเป็นไปได้ที่จะพบโหนดปลายทางภายในการบรอดแคสต์ครั้งแรก ดังนั้น จึงสามารถแบ่งโหนดปลายของการบรอดแคสต์ครั้งแรกนี้ได้เป็นสองกรณี

กรณีที่ 1 ข้อความ RREQ นี้ไปถึงโหนดปลายทางได้ภายในการบรอดแคสต์ครั้งแรกเพียงครั้งเดียว (ระยะ 1 ฮอป) โหนดต้นทางสามารถที่จะส่งข้อมูลไปยังโหนดปลายทางได้ทันที โดยไม่ต้องผ่านคลัสเตอร์เฮด

กรณีที่ 2 ข้อความ RREQ ส่งไปไม่ถึงโหนดปลายทางภายในระยะ 1 ฮอป โหนดต้นทางจะต้องทำการบรอดแคสต์ครั้งต่อๆไป เพื่อหาโหนดปลายทาง

ในหัวข้อนี้ จะกล่าวถึงเฉพาะในกรณีที่ 2 เท่านั้น เนื่องจากในกรณีแรก ได้หยุดการค้นหาเส้นทางไปแล้ว หลังจากที่โหนดในระยะ 1 ฮอปได้รับข้อความ RREQ แล้ว จะต้องพิจารณาต่อไปอีกว่า โหนดที่ได้รับข้อความ RREQ นั้นมีสถานะอย่างไร ซึ่งจะแบ่งเป็นสองเงื่อนไข ได้แก่

เงื่อนไขที่ 1 เมื่อโหนดที่ได้รับข้อความ RREQ เป็นคลัสเตอร์เฮด หรือเกตเวย์ เงื่อนไขนี้จะถูกนำไปพิจารณาต่อไป

เงื่อนไขที่ 2 เมื่อโหนดที่ได้รับข้อความ RREQ ไม่ใช่คลัสเตอร์เฮด และเกตเวย์ ข้อความนี้จะถูกทิ้งไป ไม่นำไปพิจารณาหาเส้นทางอีก

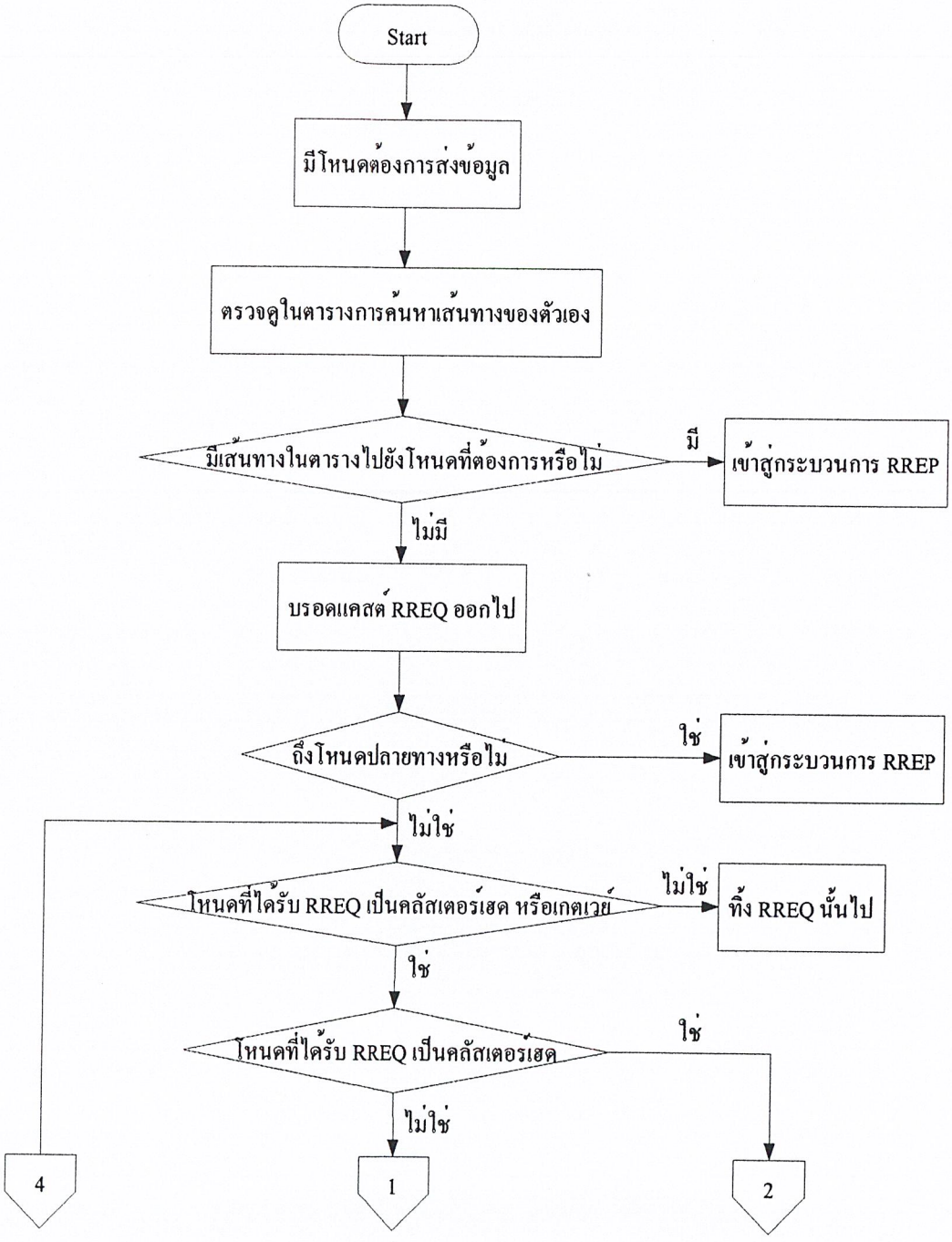
ในเงื่อนไขที่ว่า โหนดที่ได้รับ RREQ เป็นคลัสเตอร์เฮด หรือเกตเวย์ ในพิจารณาต่อไปอีกว่า โหนดนั้นเป็นคลัสเตอร์เฮด หรือเกตเวย์

เงื่อนไขที่ 1 เมื่อโหนดที่ได้รับ RREQ เป็นคลัสเตอร์เฮด คลัสเตอร์เฮดจะตรวจสอบในตารางสมาชิกของคลัสเตอร์ว่า มีโหนดปลายทางอยู่หรือไม่

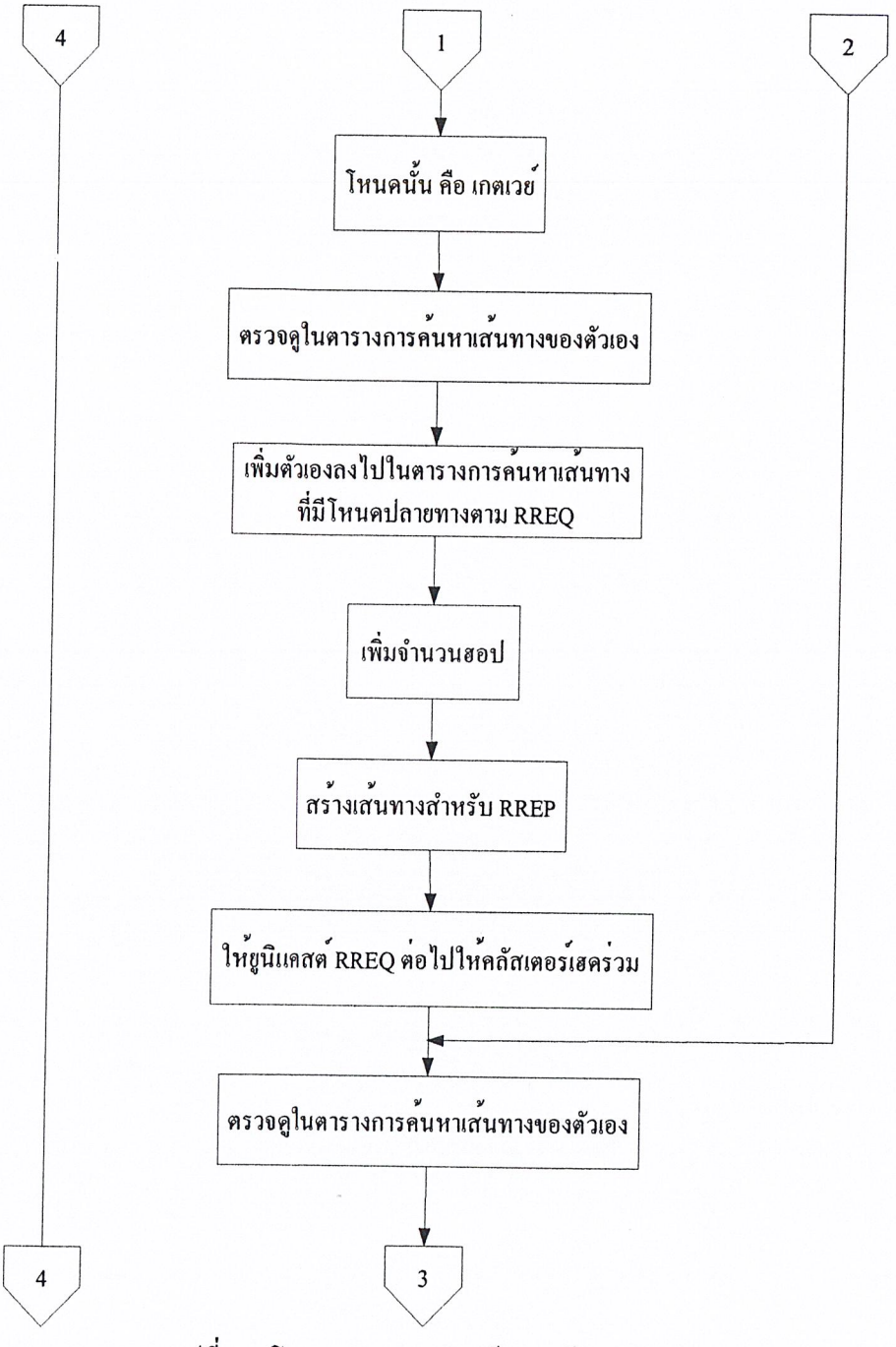
เงื่อนไขที่ 2 เมื่อโหนดที่ได้รับ RREQ เป็นเกตเวย์ เกตเวย์จะเพิ่มตัวเองลงในตารางการค้นหาเส้นทางที่มีโหนดปลายทางตามข้อความ RREQ เพิ่มจำนวนฮอปลงในข้อความ RREQ ระหว่างนั้นมันก็จะสร้างเส้นทางสำหรับข้อความ RREP ที่ใช้ในการตอบข้อความ RREQ หลังจากนั้นเกตเวย์ก็จะส่งข้อความ RREQ นั้นต่อให้ คลัสเตอร์เฮดตัวอื่นที่มันติดต่ออยู่โดยการบรอดแคสต์ข้อความ RREQ ไปให้

คลัสเตอร์เฮดที่ได้รับข้อความจากเกตเวย์นี้จะตรวจดูในตารางสมาชิกของคลัสเตอร์ของตนว่ามีโหนดปลายทางอยู่หรือไม่ ซึ่งขั้นตอนนี้จะเหมือนกับขั้นตอนในเงื่อนไขข้างต้นที่ว่า โหนดที่ได้รับ RREQ เป็นคลัสเตอร์เฮด ถ้าคลัสเตอร์เฮดพบว่ามีโหนดปลายทางอยู่ก็จะสิ้นสุดการค้นหาเส้นทาง เข้าสู่กระบวนการตอบข้อความ RREP และสามารถส่งข้อมูลไปยังโหนดปลายทางได้ แต่ถ้าคลัสเตอร์เฮดไม่พบโหนดปลายทาง คลัสเตอร์จะเพิ่มตัวเองลงในตารางการค้นหาเส้นทางที่มีโหนดปลายทางตามข้อความ RREQ เพิ่มจำนวนฮอปลงในข้อความ RREQระหว่างนั้นมันก็จะสร้างเส้นทางสำหรับข้อความ RREP ที่ใช้ในการตอบข้อความ RREQ และบรอดแคสต์ข้อความ RREQ

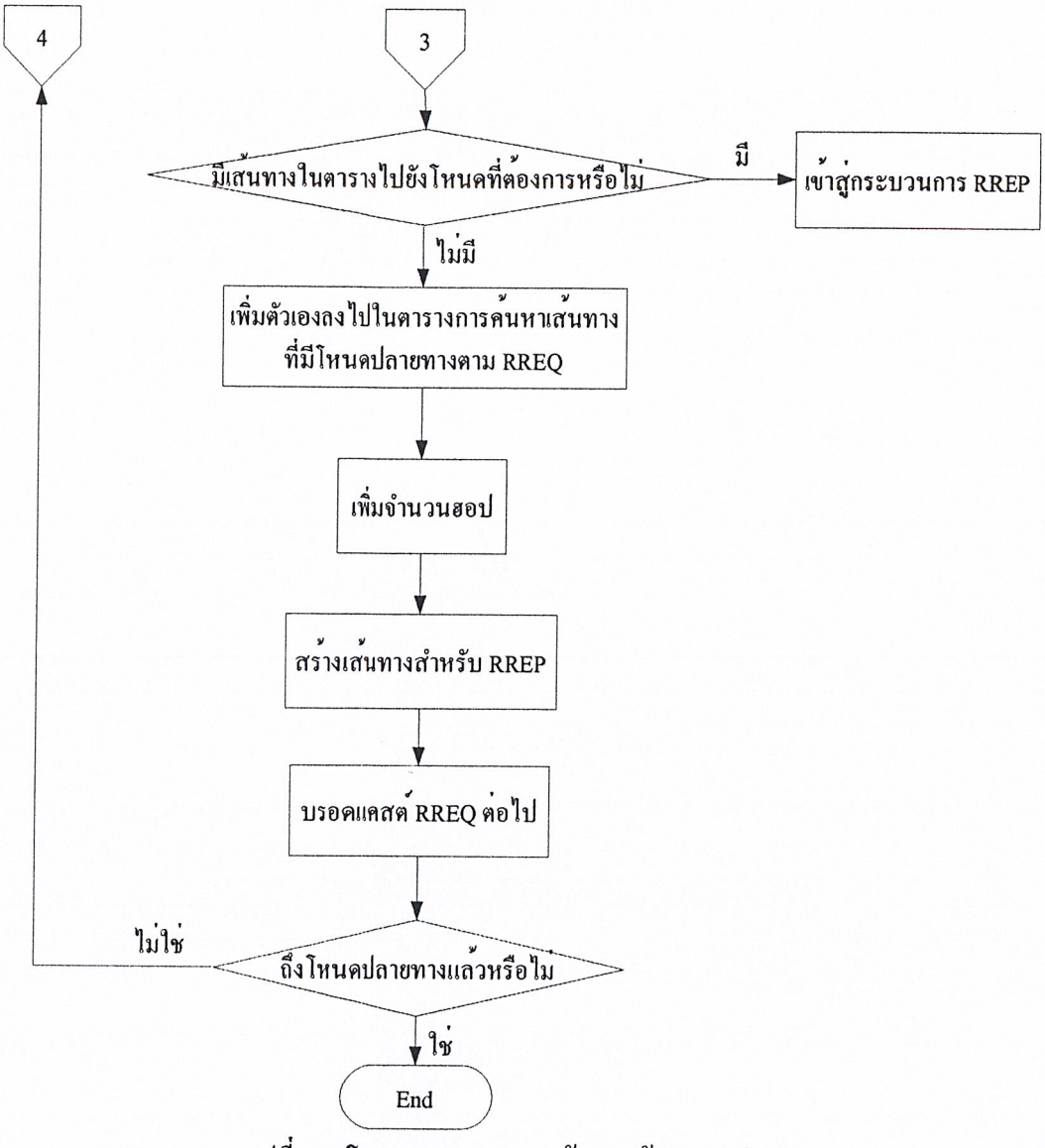
นี้ออกไปอีกครั้ง หลังจากนั้นก็จะทำการตรวจสอบว่าโหนดที่ได้รับเป็นคลัสเตอร์เฮด หรือเกตเวย์ เหมือนที่กล่าวไปแล้วข้างต้น เป็นเช่นนี้ไปเรื่อยๆจนกว่าจะพบโหนดปลายทาง



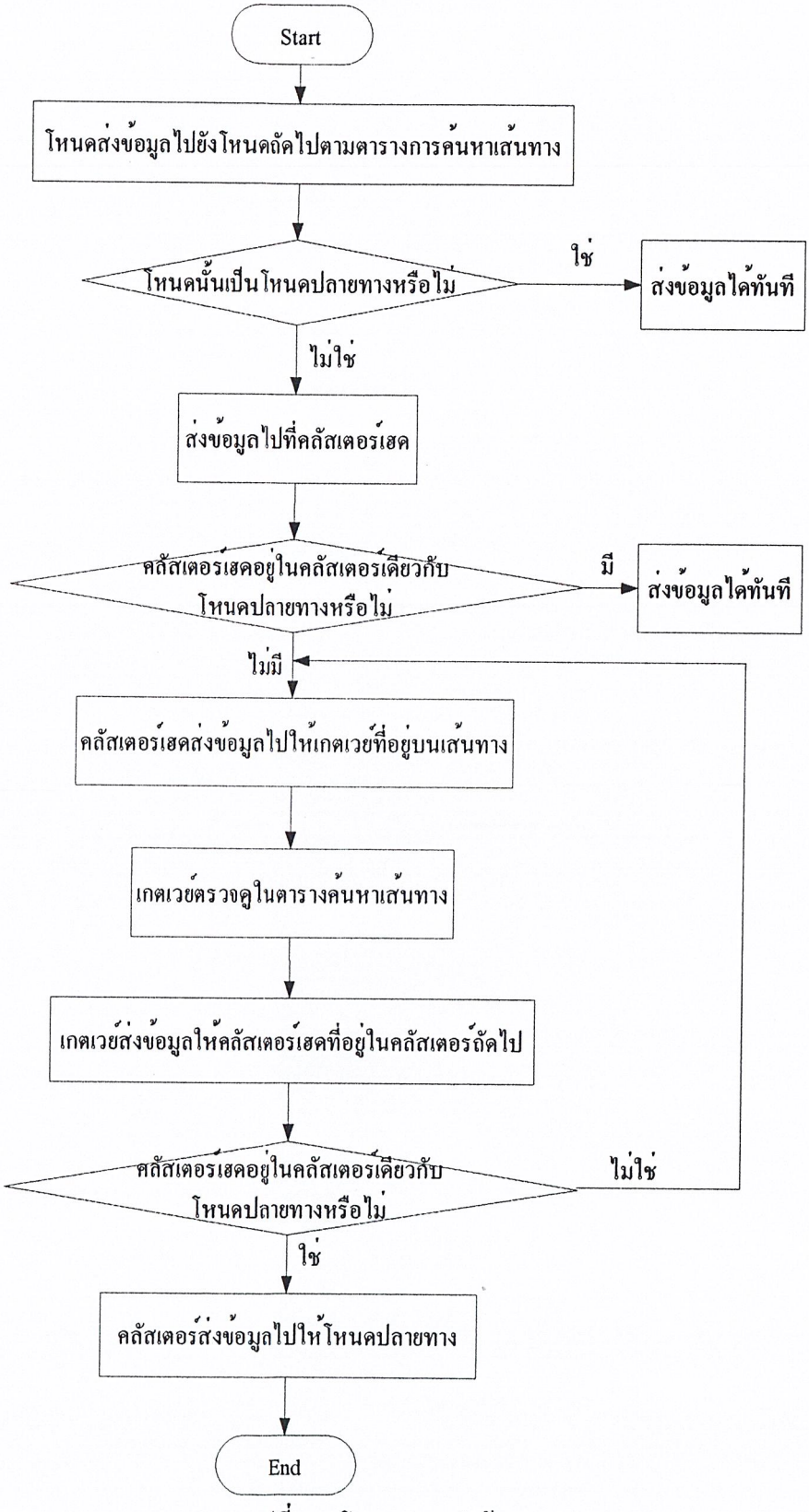
รูปที่ 5.1 โมเดลกระบวนการค้นหาเส้นทาง



รูปที่ 5.2 โมเดลกระบวนการค้นหาเส้นทาง (ต่อ)



รูปที่ 5.3 โมเดลกระบวนการค้นหาเส้นทาง (ต่อ)



รูปที่ 5.4 โมเดลการส่งข้อมูล

5.4 กระบวนการส่งข้อมูล

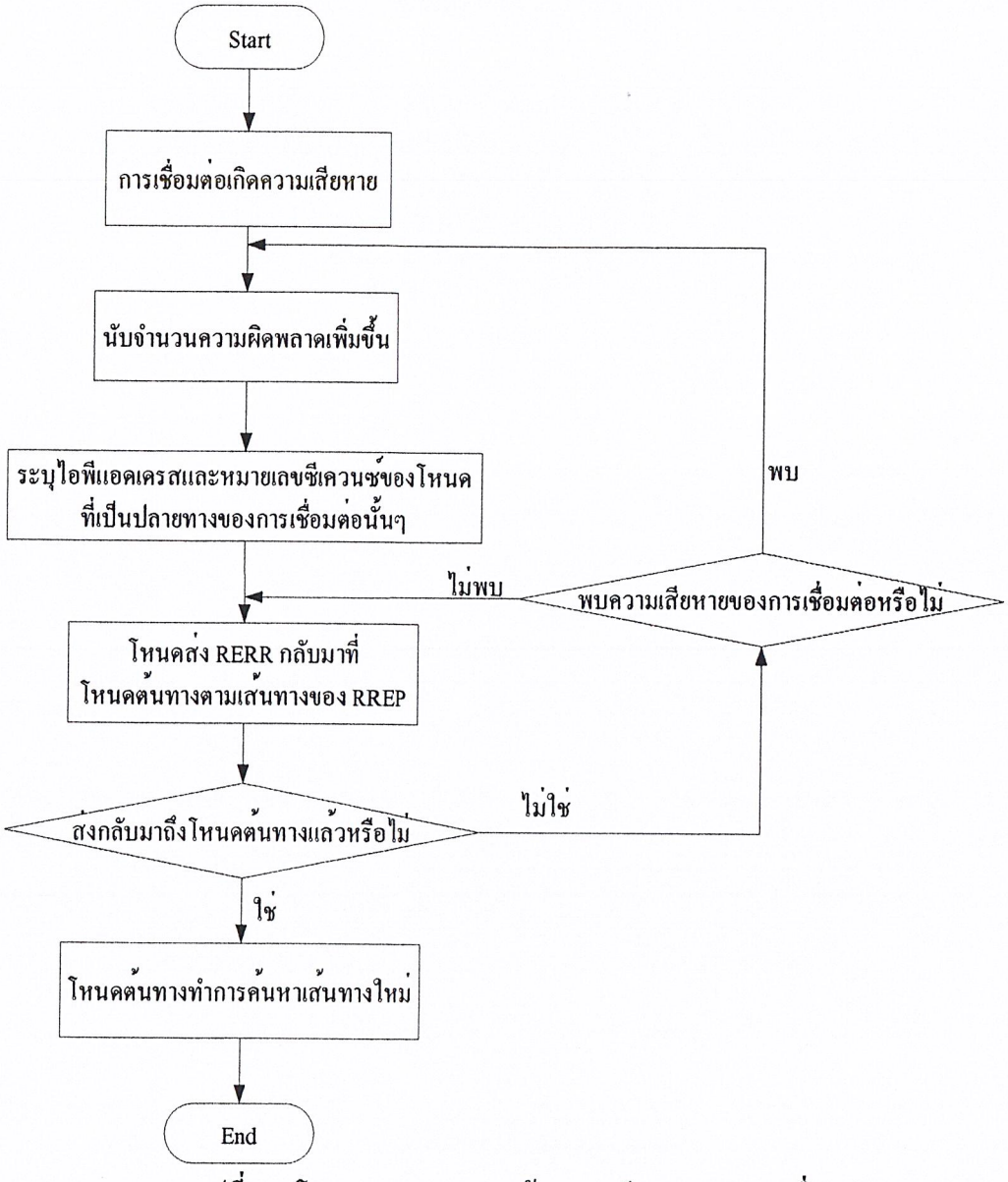
เมื่อโหนดทราบเส้นทางจากขั้นตอนการค้นหาเส้นทางแล้ว โหนดนั้นๆจะสามารถส่งข้อมูลไปได้ โดยอาศัยข้อมูลจากตารางการค้นหาเส้นทาง และตารางสมาชิกของคลัสเตอร์

ตามที่ได้กล่าวไปในหัวข้อที่ผ่านมาแล้วว่า ถ้าโหนดปลายทางอยู่ในระยะ 1 ฮอปจากโหนดต้นทาง โหนดต้นทางจะสามารถส่งข้อมูลไปยังโหนดปลายทางได้โดยตรง โดยไม่ต้องผ่านคลัสเตอร์เฮด แต่ถ้าโหนดปลายทางอยู่ในระยะที่มากกว่า 1 ฮอป โหนดต้นทางจะต้องส่งข้อมูลไปยังโหนดปลายทางโดยผ่านคลัสเตอร์เฮด คลัสเตอร์เฮดจะตรวจสอบในตารางสมาชิกของคลัสเตอร์ว่ามีโหนดปลายทางอยู่หรือไม่ ถ้ามีคลัสเตอร์เฮดก็จะทำการส่งข้อมูลไปยังโหนดปลายทาง แต่ถ้าไม่มีคลัสเตอร์เฮดจะส่งข้อมูลนี้ไปยังเกตเวย์ของตนเอง ซึ่งคลัสเตอร์เฮดจะทราบอยู่แล้วว่าจะส่งข้อมูลนี้ไปที่เกตเวย์ตัวใดจากข้อมูลที่ได้จากการค้นหาเส้นทาง หลังจากนั้นเกตเวย์จะส่งข้อมูลนี้ไปยังคลัสเตอร์เฮดที่อยู่บนคลัสเตอร์เส้นทางไปยังโหนดปลายทาง คลัสเตอร์เฮดโหนดนี้ก็จะทำการตรวจสอบแล้วส่งข้อมูลไปยังเกตเวย์ของตนไปเรื่อยๆ จนกว่าจะพบโหนดปลายทาง การส่งข้อมูลนี้จะทำการส่งเฉพาะบนคลัสเตอร์เฮด และเกตเวย์เท่านั้น เช่นเดียวกับการค้นหาเส้นทาง

5.5 กระบวนการแจ้งความเสียหายของการเชื่อมต่อ

การติดต่อสื่อสารต่างๆย่อมต้องการการเชื่อมต่อที่ดี ประสิทธิภาพสูง และไว้วางใจได้ ดังนั้นเมื่อเกิดการเชื่อมต่อเกิดความเสียหาย การแจ้งความเสียหายที่เกิดขึ้นกับผู้ที่มีความเกี่ยวข้องจึงมีความจำเป็นอย่างยิ่ง เพื่อที่จะแก้ไขและทำให้สามารถใช้งานอย่างเดิมให้ได้อย่างรวดเร็วที่สุด ในโปรโตคอลนี้ก็เช่นกัน ได้มีกำหนดให้มีการแจ้งความเสียหายของการเชื่อมต่อที่เกิดขึ้นส่งกลับไปยังโหนดต้นทาง เพื่อให้โหนดต้นทางสามารถค้นหาเส้นทางใหม่ได้

ขั้นตอนการแจ้งความเสียหายเริ่มจาก เมื่อมีความเสียหายเกิดขึ้นจะมีก็แต่เฉพาะโหนดที่ใช้การเชื่อมต่อที่เกิดความเสียหายเท่านั้นที่เดือดร้อน ดังนั้นโหนดที่ใช้การเชื่อมต่อที่อยู่ติดกับการเชื่อมต่อนั้นจะเป็นผู้แจ้งความเสียหาย โดยส่งเป็นข้อความ RERR กลับมาที่โหนดต้นทางที่ใช้เส้นทางนี้อยู่ โดยจะนับจำนวนของการเชื่อมต่อที่เกิดความเสียหาย และระบุถึงไอพีแอดเดรส และเลขซีเควนซ์ของโหนดปลายทางอีกด้วย ถ้าระหว่างทางมีจำนวนของการเชื่อมต่อที่เกิดความเสียหายหลายการเชื่อมต่อ ก็จะมีการระบุเพิ่มในข้อความ RERR ได้



รูปที่ 5.5 โมเดลกระบวนการแจ้งความเสียหายของการเชื่อมต่อ

5.6 กระบวนการเข้าร่วมคลัสเตอร์ของโหนดไม่มีสถานะ

โหนดทุกโหนดที่อยู่ในเครือข่ายจะต้องมีสถานะ หรือมีคลัสเตอร์อยู่ ดังนั้นเมื่อมีโหนดใหม่เข้ามาในเครือข่าย โหนดนั้นจะต้องบรอดแคสต์ข้อความ HELLO ออกไปแล้วรอการตอบรับและระบุสถานะของโหนดไม่มีสถานะนี้ ซึ่งคำตอบที่จะได้รับนี้มีอยู่ 4 กรณีที่มีผลต่อสถานะของโหนดไม่มีสถานะ ได้แก่

กรณีที่ 1 หลังจากผ่านไปช่วงระยะเวลาหนึ่งแล้วยังไม่มีการตอบรับกลับมา โหนดนั้นจะเปลี่ยนสถานะเป็นคลัสเตอร์เฮด

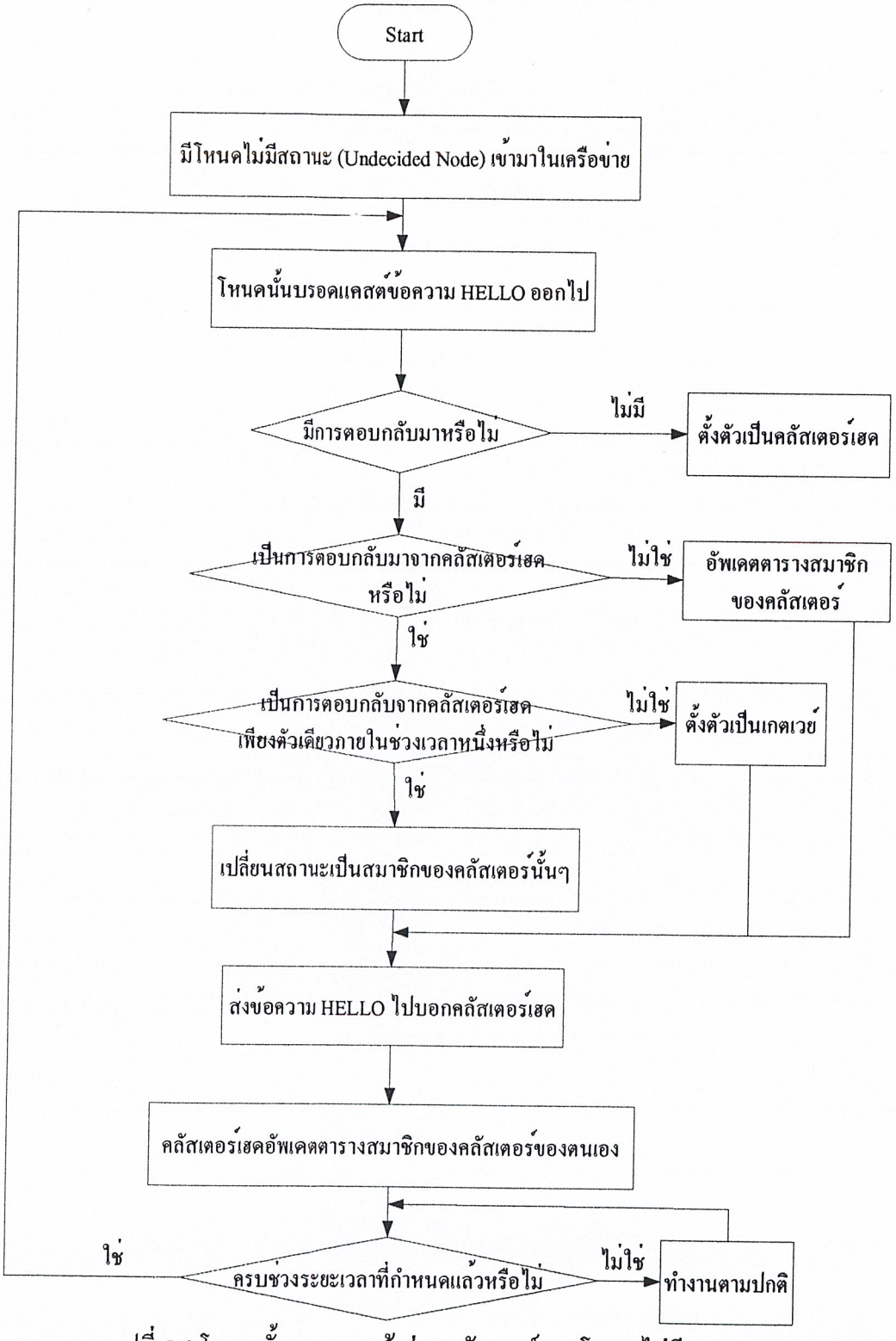
กรณีที่ 2 หลังจากผ่านไปช่วงระยะเวลาหนึ่งแล้ว มีการตอบรับกลับมาจากโหนดที่ไม่ใช่คลัสเตอร์เฮด โหนดนั้นจะอัพเดตตารางสมาชิกของคลัสเตอร์ของตนเอง

กรณีที่ 3 หลังจากผ่านไปช่วงระยะเวลาหนึ่งแล้ว มีการตอบรับกลับมาจากโหนดที่เป็นคลัสเตอร์เฮดเพียงตัวเดียว โหนดไม่มีสถานะจะเปลี่ยนสถานะเป็นสมาชิกของคลัสเตอร์เฮดนั้น

กรณีที่ 4 หลังจากผ่านไปช่วงระยะเวลาหนึ่งแล้ว มีการตอบรับกลับมาจากโหนดที่เป็นคลัสเตอร์เฮดมากกว่า 1 ตัว โหนดจะเปลี่ยนสถานะเป็นเกตเวย์ของคลัสเตอร์เฮดที่ตอบกลับมา

โดยทั้งในกรณีที่ 2, 3 และ 4 โหนดไม่มีสถานะจะต้องส่งข้อความ HELLO ไปบอกคลัสเตอร์เฮด ว่าตนเองได้ขอเข้ามาอยู่ในคลัสเตอร์ด้วย เพื่อให้คลัสเตอร์เฮดอัพเดตตารางสมาชิกของคลัสเตอร์

แต่ละโหนดต้องมีการบรอดแคสต์ข้อความ HELLO ออกไปทุกๆช่วงเวลา เพื่อตรวจสอบสถานะของตนเอง

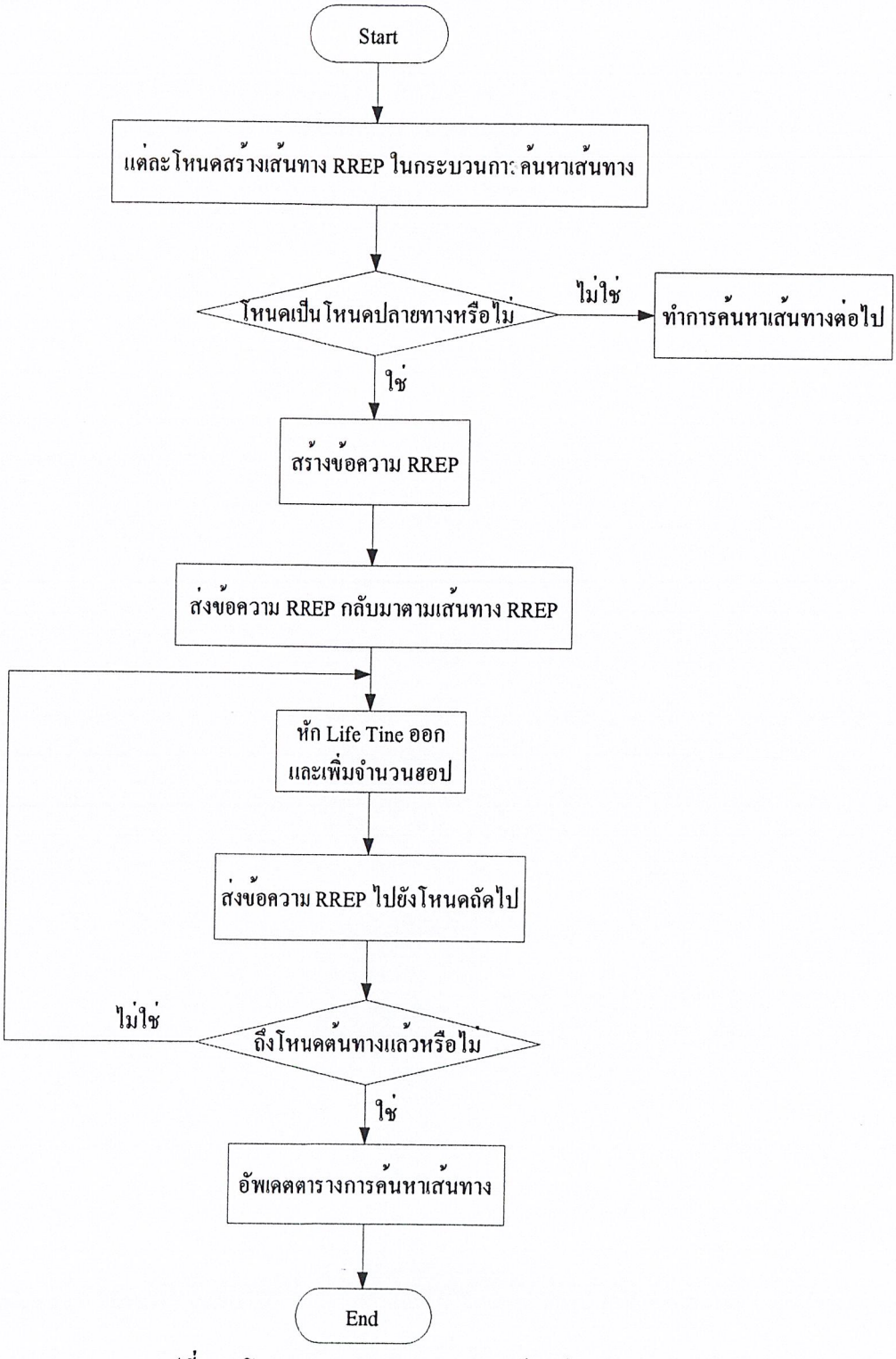


รูปที่ 5.6 โมเดลขั้นตอนการเข้าร่วมคลัสเตอร์ของโหนดไม่มีสถานะ

5.7 กระบวนการตอบรับด้วยข้อความ RREP

จากกระบวนการค้นหาเส้นทางที่ได้กล่าวไปแล้วข้างต้น พบว่า แต่ละโหนดได้สร้างเส้นทางสำหรับการตอบข้อความ RREP ไว้แล้ว ถ้าโหนดที่ข้อความ RREQ ไปถึงยังไม่ใช่โหนดปลายทาง กระบวนการค้นหาเส้นทางจะยังคงดำเนินต่อไป แต่ถ้าข้อความ RREQ ไปถึงโหนดปลายทางแล้ว ขั้นตอนต่อไปก็จะเป็นขั้นตอนของกระบวนการตอบรับด้วยข้อความ RREP

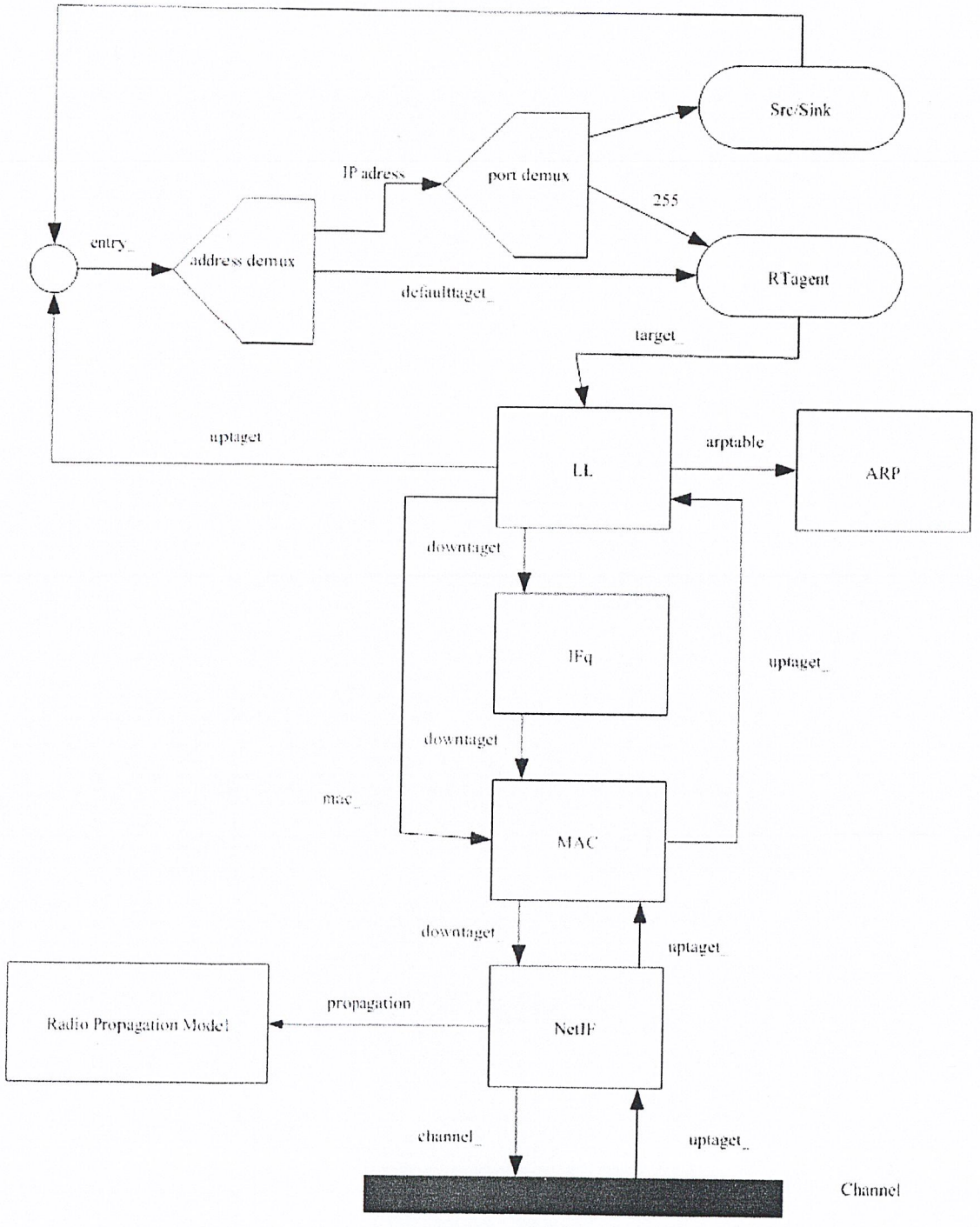
โหนดปลายทางสร้างข้อความ RREP โดยจะส่งกลับมาตามเส้นทาง RREP ที่ถูกสร้างไว้ในกระบวนการค้นหาเส้นทางแล้ว ภายในข้อความ RREP ประกอบด้วยข้อมูลต่างๆ เช่น จำนวน ฮอป ไอพีแอดเดรสของโหนดปลายทาง เลขซีเควนซ์ของโหนดปลายทาง และไลฟ์ไทม์ เป็นต้น โดยค่าไลฟ์ไทม์นี้จะถูกหักลบออกลงไปเรื่อยๆ เมื่อที่ข้อความ RREP นี้ถูกส่งกลับย้อนไปยังแต่ละโหนด ในขณะที่จำนวนฮอปเพิ่มขึ้นเรื่อยๆ จนกระทั่งข้อความ RREP มาถึงโหนดต้นทาง ซึ่งโหนดต้นทางจะทำการอัพเดตข้อมูลในตารางการค้นหาเส้นทาง เพื่อให้มีข้อมูลเส้นทางที่ทันสมัยที่สุด และได้เส้นทางที่มีความเหมาะสมที่สุดในขณะนั้น



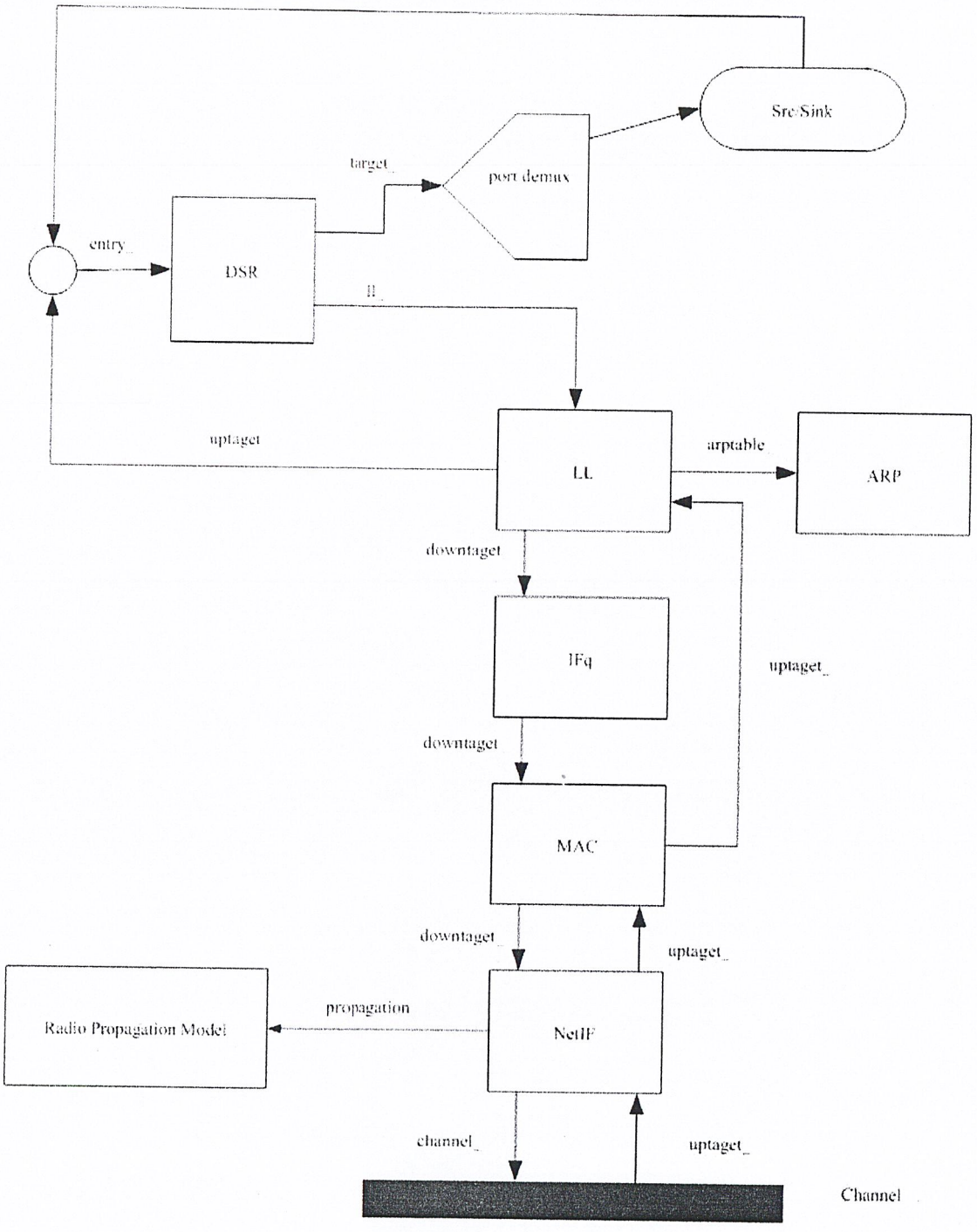
รูปที่ 5.7 โมเดลกระบวนการตอบรับด้วยข้อความ RREP

5.8 โหนดไร้สายใน Ns2

หลังจากทำการศึกษาการจำลองการทำงานของเครือข่ายไร้สายในNs2 จะพบว่ามีโหนดไร้สาย (mobile node) เป็นส่วนประกอบหลักนอกจากนั้นก็จะเป็นฟังก์ชันที่เสริมเข้ามาเพื่อเพิ่มเติมคุณสมบัติที่ใช้ในการจำลองการทำงานแบบไร้สายเช่น คุณสมบัติที่ใช้ในเครือข่ายแบบ multi-hop รวมทั้งคุณสมบัติทางด้านแลนไร้สาย คลาสของโหนดไร้สายใน Ns2 เป็นคลาสที่ทำการสืบทอดมาจากคลาสโหนดที่ใช้ในเครือข่ายแบบมีสาย โดยเพิ่มฟังก์ชันที่เกี่ยวข้องเข้ามาเช่นความสามารถในการส่งและรับสัญญาณในช่องสัญญาณแบบไร้สาย โครงสร้างของโหนดแบบไร้สายเป็นดังรูปข้างล่าง โหนดไร้สายใน Ns2 ประกอบด้วย ตัวแยกแยะแอดเดรสที่คอยแยกแยะแอดเดรสที่ เอนทรี่ (entry) ของโหนดรับเข้ามา ตัวแยกแยะพอร์ทมีไว้เพื่อทำการแยกแยะว่า แพ็คเก็ตที่ส่งมาเป็นของเอเจ้นท์ตัวใด โดยที่ถ้าหมายเลขพอร์ทเป็น 255 จะหมายถึงเราท์ติ้งเอเจ้นท์ โดยที่ในส่วนของเราท์ติ้งเอเจ้นท์นี้ เป็นเราท์ติ้งเอเจ้นท์ที่โครงงานนี้จะทำการพัฒนาขึ้นมาเพื่อทำการศึกษาการทำงานของเครือข่ายแอดฮอคที่มีการเราท์ติ้งเป็นแบบคลัสเตอร์ เราท์ติ้งเอเจ้นท์ที่ทำการพัฒนาขึ้นใหม่นี้เพื่อที่จะทำให้ง่ายในการกล่างถึงเราจะใช้คำว่า DOCR (Developed On-demand Cluster base Routing) ในการกล่าวถึงเราท์ติ้งเอเจ้นท์นี้ โดยโครงงานนี้จะทำการแทนที่เราท์ติ้งเอเจ้นท์ที่มีอยู่แล้วใน Ns2 ซึ่งได้แก่ DSDV (Destination Sequence Distance Vector) , DSR (Dynamic Source Routing) , AODV (Ad-hoc On-demand Distance Vector) , TORA (Temporally Ordered Algorithm) ด้วยเราท์ติ้งเอเจ้นท์ที่ทำการพัฒนาขึ้นมาใหม่เนื่องจากเราท์ติ้งเอเจ้นท์ที่มีใน Ns2 ยังไม่มีตัวใดที่รองรับความสามารถในการค้นหาเส้นทางแบบที่เป็นคลัสเตอร์ โดยเราท์ติ้งเอเจ้นท์ที่ทำการพัฒนาจะเชื่อมต่ออยู่กับลิ้งค์เลเยอร์ (LL) และลิ้งค์เลเยอร์ทำการเชื่อมต่ออยู่กับอินเทอร์เฟซคิว (IFq) อินเทอร์เฟซคิวก็ทำการเชื่อมต่ออยู่กับแมคเลเยอร์ (MAC) แมคเลเยอร์ก็ทำการเชื่อมต่อกับเน็ตเวอร์คอินเทอร์เฟซ (NetIF) โดยตัวเน็ตเวอร์คอินเทอร์เฟซนี้ก็ทำการเชื่อมต่ออยู่กับโมดูลเกี่ยวกับการแพร่กระจายคลื่น และเชื่อมต่ออยู่กับช่องสัญญาณที่ใช้ในการส่งสัญญาณอีกทีโดยมีเส้นทางการเชื่อต่อดังรูปข้างล่าง



รูปที่ 5.8 โครงสร้างของโหนดในเครือข่ายไร้สาย



รูปที่ 5.9 โครงสร้างของโหนดที่ใช้ในการค้นหาเส้นทางแบบ DSR

5.9 โมดูลที่ใช้ในการค้นหาเส้นทาง

ในโครงงานนี้ได้พัฒนาโมดูลที่ใช้ในการค้นหาเส้นทางใน Ns2 ซึ่งประกอบไปด้วย 2 ส่วนหลักๆ คือ

1) เอเจ้นท์ที่ใช้ในการค้นหาเส้นทาง

ใช้ในการแลกเปลี่ยนแพคเก็ตในการค้นหาเส้นทางกับโหนดข้างเคียง

2) ฟังก์ชันที่ใช้ในการค้นหาเส้นทาง

ในส่วนนี้จะใช้ข้อมูลที่ได้จากเราท์ติ้งเอเจ้นท์ในข้อ 1 เพื่อนำมาผ่านอัลกอริธึมที่ใช้ในการค้นหาเส้นทาง จากนั้นเมื่อได้เส้นทางที่ต้องการจะส่งแล้ว จึงทำการส่งข้อมูลออกไป

ในการพัฒนาโมดูลที่ใช้ในการค้นหาเส้นทางนั้น ผู้พัฒนาสามารถที่จะสร้างตัวคลาสซิฟาย (classifier) เองได้ดังที่แสดงในรูป 5.16 เป็นโมดูลที่ใช้ในการค้นหาเส้นทางแบบ DSR แต่ในโครงงานนี้ จะใช้ตัวคลาสซิฟาย (ตัวแยกแยะแอดเดรส) ที่ Ns2 มีให้อยู่แล้ว เนื่องจากเห็นว่าไม่มีความจำเป็น เพราะไม่มีความต้องการที่จะต้องใช้ความสามารถพิเศษนอกเหนือจากที่ Ns2 มีให้

5.10 รูปแบบของผลลัพธ์ที่ได้หลังจากการจำลองการทำงานของเครือข่าย (trace format)

ในการแสดงผลลัพธ์ที่ได้จากการจำลองการทำงานของเครือข่ายของ Ns2 นั้นจะได้ผลออกมาเป็นรูปแบบของตัวอักษร (text file) มีรูปแบบการแสดงผลที่ได้หลักๆ อยู่ 3 รูปแบบคือ

1) รูปแบบการแสดงผลแบบ Normal

2) รูปแบบการแสดงผลของการจำลองแบบ Wireless

3) รูปแบบการแสดงผลของ NAM (Network Animator)

โดยในการแสดงผลของการจำลองแบบ Wireless นั้นยังสามารถแบ่งย่อยออกไปอีกเป็นแบบเก่า และ แบบใหม่ ผู้ทำการจำลองการทำงานของเครือข่ายสามารถทำการเลือกได้ว่าจะใช้แบบใดในการแสดงผลลัพธ์ แต่ในโครงงานนี้จะเน้นการพัฒนาไปที่การแสดงผลในรูปแบบของ Wireless แบบใหม่ โดยมีรูปแบบเป็นดังนี้

1)ในกรณีที่เป็นแพคเก็ตชนิด Route Request

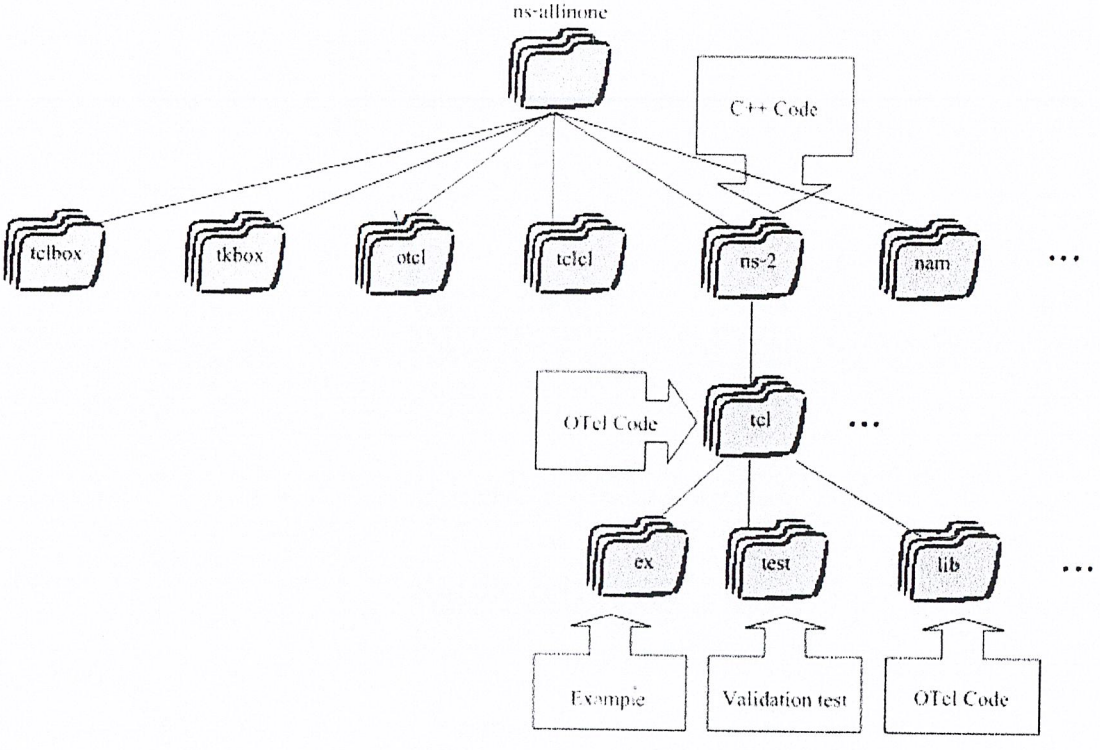
docr	rq type	rq hop count	rq beast id	rq dst	rq dst seqno	rq src	rq src seqno	REQUEST

2) ในกรณีที่เป็นแพ็คเก็ตชนิด Route Reply, Route Error, Hello

docr	rp type	rp hop count	rp dst	rp dst seqno	rp lifetime	REPLY,ERROR,HELLO

5.11 โครงสร้างโฟลเดอร์ของ Ns2

เมื่อทำการดาวน์โหลด Ns2 จะสามารถเลือกได้ 2 แบบคือ แบบที่ดาวน์โหลดครั้งเดียวแล้วได้ไฟล์มาทั้งหมด กับอีกแบบคือทำการดาวน์โหลดทีละส่วน โดยสามารถเลือกเฉพาะส่วนที่ต้องการ แต่ในโครงงานนี้จะใช้แบบที่ทำการดาวน์โหลดครั้งเดียวเนื่องจากมีความง่ายและสะดวกต่อการใช้งาน เมื่อทำการดาวน์โหลดและทำการติดตั้งลงบนเครื่องแล้วจะมีโครงสร้างโฟลเดอร์เป็นดังรูปข้างล่าง

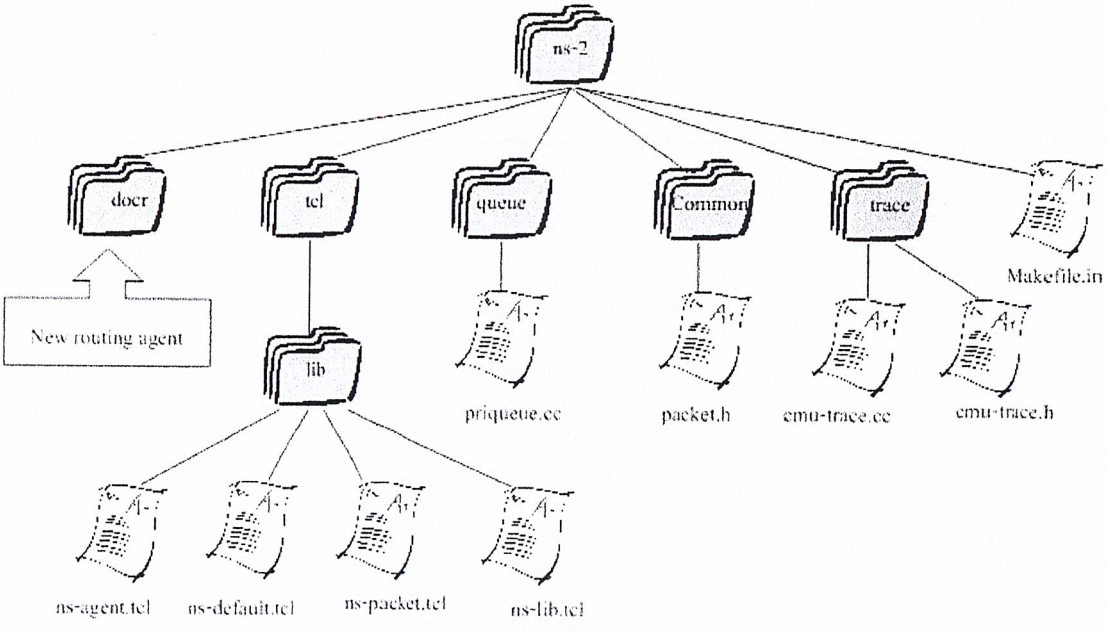


รูปที่ 5.10 โครงสร้างโฟลเดอร์ของ Ns2 (บางส่วน)

โฟลเดอร์ที่เป็นโฟลเดอร์รากจะชื่อ ns-allinone ถัดลงมาอีกชั้นจะเป็นโฟลเดอร์ของ NAM โฟลเดอร์ของ Ns2 รวมทั้งโฟลเดอร์ที่ใช้เก็บ tcl , tk, OTcl , tclcl และโฟลเดอร์อื่นๆ โดยโฟลเดอร์ที่ชื่อ Ns2 นี้จะเป็นโฟลเดอร์ที่ใช้เก็บโค้ดโปรแกรมของ Ns2 ที่เป็นภาษา C++ โฟลเดอร์ถัดลงมาอีกชั้นของโฟลเดอร์นี้จะประกอบไปด้วยโฟลเดอร์ต่างๆ แต่จะมีโฟลเดอร์ที่สำคัญอยู่โฟลเดอร์หนึ่งที่ชื่อ tcl เป็นโฟลเดอร์ที่ใช้เก็บโค้ดโปรแกรมของ Ns2 ในส่วนที่เป็นภาษา OTcl และโฟลเดอร์ที่อยู่

ในลำดับถัดลงมาจากโฟลเดอร์นี้ จะประกอบด้วยโฟลเดอร์ต่างๆ เช่น โฟลเดอร์ ex เป็นโฟลเดอร์ที่ใช้ในการเก็บไฟล์ตัวอย่างที่ใช้ในการจำลองการทำงานของเครือข่าย โฟลเดอร์ test เป็นโฟลเดอร์ที่ใช้ในการเก็บไฟล์ที่ใช้ในการทดสอบความถูกต้องในการทำงานของ Ns2 หลังจากที่ได้ทำการติดตั้งแล้ว และโฟลเดอร์ lib เป็นโฟลเดอร์ที่ใช้ในการเก็บไฟล์ไลบรารี่ของ Ns2 ที่เป็นภาษา OTcl

โครงงานนี้เป็นโครงงานที่ทำการพัฒนาโมดูลที่ใช้ในการค้นหาเส้นทาง ของเครือข่ายแอดฮอค ดังนั้นจึงต้องมีการแก้ไขไฟล์ซอร์ซโค้ดของ Ns2 เพื่อทำการเพิ่มโมดูลที่ใช้ในการค้นหาเส้นทางแบบใหม่เข้าไปให้ทำการค้นหาเส้นทางในรูปแบบที่ต้องการได้ โดยโฟลเดอร์ที่จะเพิ่มขึ้นมาและไฟล์ที่จะต้องทำการแก้ไข จะเห็นได้จากรูปข้างล่าง



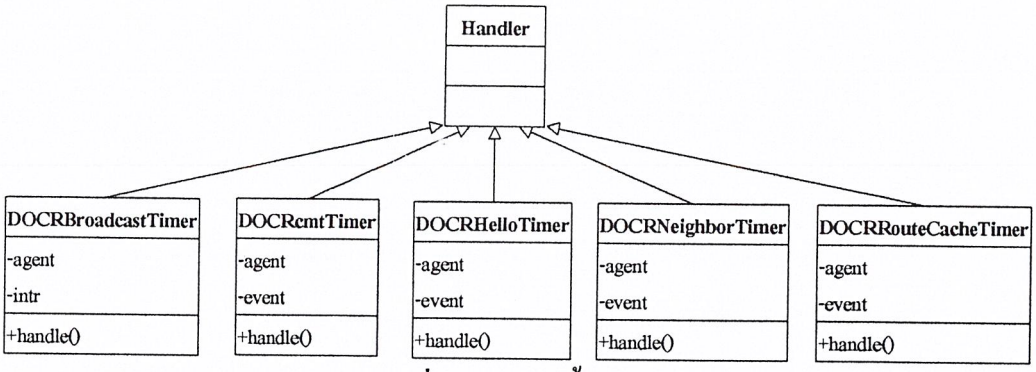
รูปที่ 5.11 โฟลเดอร์ที่ทำการเพิ่มขึ้นมาใหม่และไฟล์ที่ต้องทำการแก้ไข

โฟลเดอร์ที่ทำการเพิ่มขึ้นมาใหม่คือโฟลเดอร์ที่ชื่อ docr (Developed On-demand Cluster base Routing) เป็นโฟลเดอร์ที่ใช้ในการเก็บไฟล์ที่เพิ่มขึ้นมาในการสร้างโมดูลในการค้นหาเส้นทาง ซึ่งจะประกอบด้วย ไฟล์ที่ใช้ในการอิมพลีเมนต์ฟังก์ชันการค้นหาเส้นทาง , เราท์ติ้งคิว, ตารางการค้นหาเส้นทาง, ตารางสมาชิกของคลัสเตอร์, แพ็คเก็ตต่างๆ นอกจากโฟลเดอร์ที่จะต้องทำการเพิ่มขึ้นมาใหม่แล้ว ยังมีไฟล์ที่จะต้องทำการแก้ไขด้วยได้แก่

1) packet.h เป็นไฟล์ที่ใช้ในการนิยามเฮดเดอร์ของแพคเก็ตต่างๆ ซึ่งไฟล์นี้จะเก็บอยู่ในโฟลเดอร์ common ในโครงงานนี้ได้มีการสร้างเฮดเดอร์ของแพคเก็ตขึ้นมาใหม่ ดังนั้นจึงต้องมีการแก้ไขที่ไฟล์นี้ด้วย
2) priqueue.cc เป็นไฟล์ที่ใช้ในการนิยาม priority queue ที่ใช้ในการอิมพลีเมนต์อินเทอร์เฟซคิว ที่ทำงานอยู่ระหว่างลิงค์เลเยอร์กับแมคเลเยอร์ โดยในโครงงานนี้จะทำการแก้ไขตัวอินเทอร์เฟซคิวนี้ด้วยเพื่อให้แพคเก็ตที่เป็นแพ็คเก็ตในการค้นหาเส้นทางที่ใช้ในเราท์ติ้งเอเจ้นท์ที่พัฒนาขึ้นมาใหม่นี้ ได้รับค่าไพรออริตี้ (priority) สูงกว่าแพ็คเก็ตที่ใช้ในการส่งข้อมูลธรรมดา
3) cmu-trace.cc และ cmu-trace.h เป็นการที่ใช้ในการนิยามการแสดงผลของการจำลองการทำงานของเครือข่ายแบบไร้สายโดยเราจะทำการแก้ไขไฟล์นี้เพื่อให้ Ns2 สามารถแสดงผลลัพธ์ของการจำลองการทำงานของเครือข่ายแอดฮอคได้ เมื่อใช้เราท์ติ้งเอเจ้นท์ที่เราทำการพัฒนาขึ้นมาใหม่ โดยสามารถแสดงค่าที่เราสนใจดังที่แสดงในหัวข้อ 5.x
4) ns-agent.tcl เป็นไฟล์ที่ใช้ในการนิยามเอเจ้นท์ต่างๆ เนื่องจากในโครงงานนี้ต้องพัฒนาเราท์ติ้งเอเจ้นท์ขึ้นมาใหม่จึงต้องทำการแก้ไขไฟล์นี้
5) ns-default.tcl เป็นไฟล์ที่ใช้ในการกำหนดค่าปกติ (default) ของออปเจ็คต่างๆ ที่ใช้ในการจำลองการทำงานของเครือข่าย
6) ns-packet.tcl เป็นไฟล์ที่จัดการเกี่ยวกับแพคเก็ตต่างๆ ใน Ns2 จึงต้องทำการแก้ไขเพื่อให้รองรับแพคเก็ตชนิดใหม่ที่ได้ทำการพัฒนาขึ้นมาพร้อมกับเราท์ติ้งเอเจ้นท์ที่ได้พัฒนาขึ้นมาใหม่
7) ns-lib.tcl เป็นไฟล์ไลบรารี่ที่ใช้ในการกำหนดค่าต่างๆ ใน Ns2 จะต้องทำการแก้ไขเพื่อเพิ่มเราท์ติ้งเอเจ้นท์ที่พัฒนาขึ้นมาใหม่เข้าไป
8) Makefile.in จะต้องทำการแก้ไขเพื่อใช้ในการคอมไพล์ซอร์ซโค้ดที่เพิ่มขึ้นมาใหม่

5.12 คลาสของไทม์เมอร์

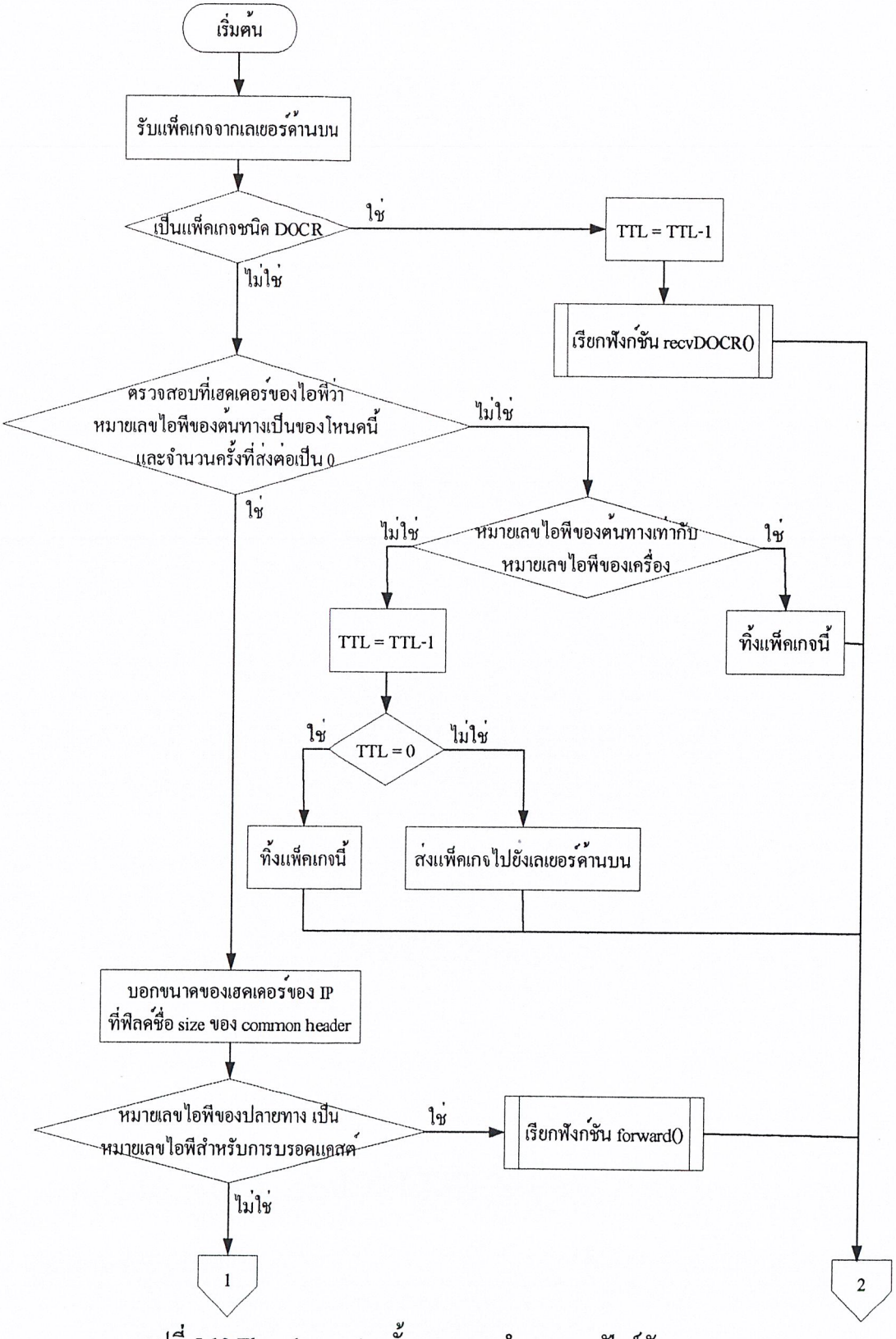
คลาสมีลักษณะเป็นลำดับขั้น โดยมีคลาสแฮนด์เลอร์อยู่ในลำดับขั้นที่สูงกว่าคลาสไทม์เมอร์ ในโปรโตคอลนี้ใช้ไทม์เมอร์หลัก 5 ไทม์เมอร์ ดังแสดงในรูป



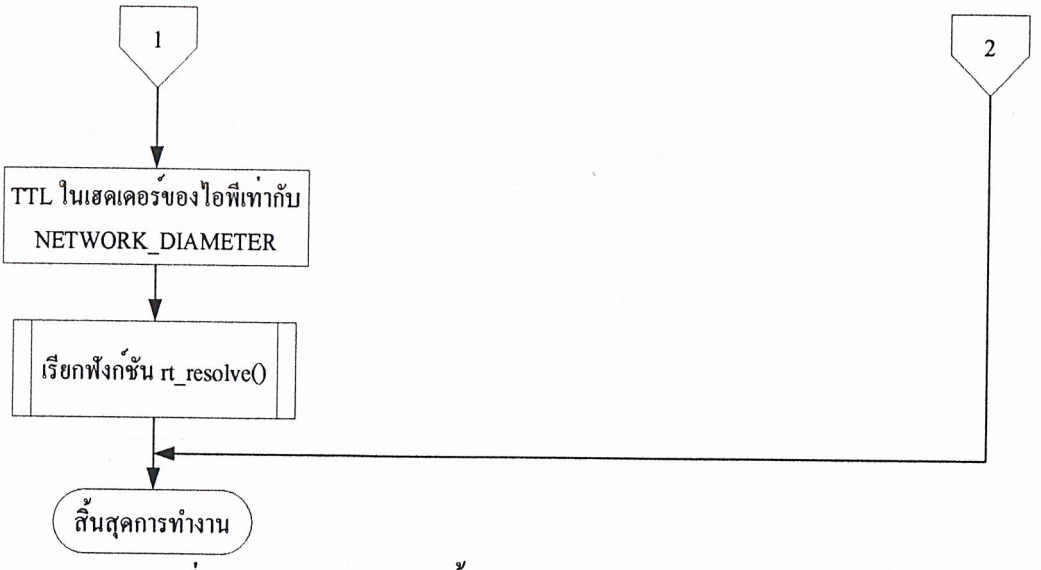
รูปที่ 5.12 ลำดับชั้นของคลาส

1. DOCR Broadcast Timer
2. DOCR cmt Timer เป็นไทม์เมอร์ที่จัดการในส่วนของตารางสมาชิกของคลัสเตอร์
3. DOCR Hello Timer
4. DOCR Neighbor Timer เป็นไทม์เมอร์ที่จัดการในส่วนของตารางโหนดข้างเคียง
5. DOCR Route Cache Timer

5.13 ฟังก์ชันที่สำคัญ



รูปที่ 5.13 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recv( )



รูปที่ 5.14 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recv( )

ฟังก์ชัน recv( )

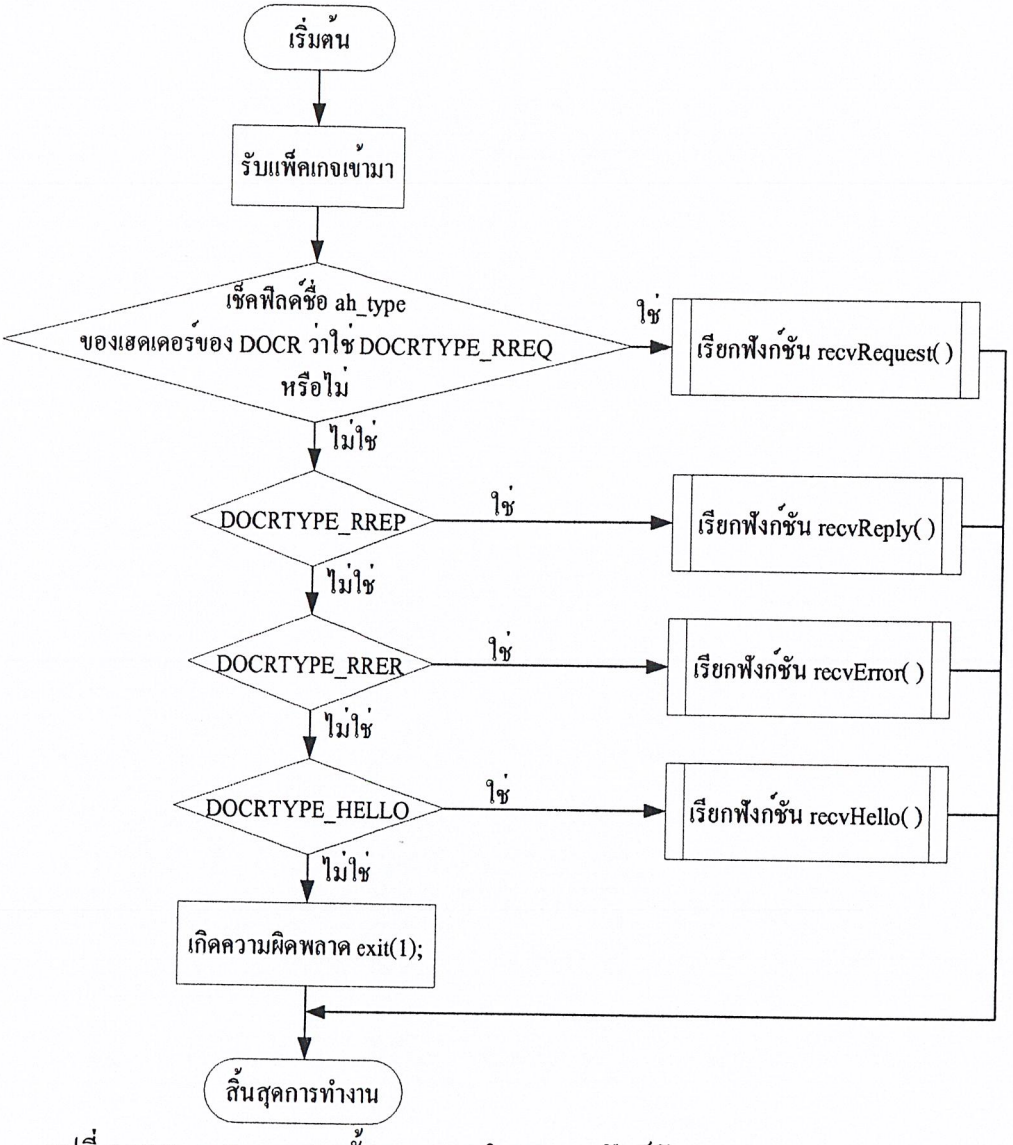
เป็นฟังก์ชันที่เรียกใช้เมื่อมีการรับแพ็คเกจจากเลเยอร์ชั้นบน นำมาพิจารณาว่าเป็นแพ็คเกจของโปรโตคอลอแบบ DOCR หรือไม่ ถ้าใช่จะมีการเรียกใช้ฟังก์ชันอื่นที่เกี่ยวข้องกับโปรโตคอลอแบบ DOCR ต่อไป ถ้าไม่ใช่ก็จะทำการพิจารณาต่อ ซึ่งแพ็คเกจที่รับมานั้นอาจจะต้องถูกส่งต่อไปยังเลเยอร์ชั้นบนหรือล่างหรือมีการส่งต่อไปยังฟังก์ชันอื่นเพื่อดำนินการต่อไป

ฟังก์ชัน recvDOCR( )

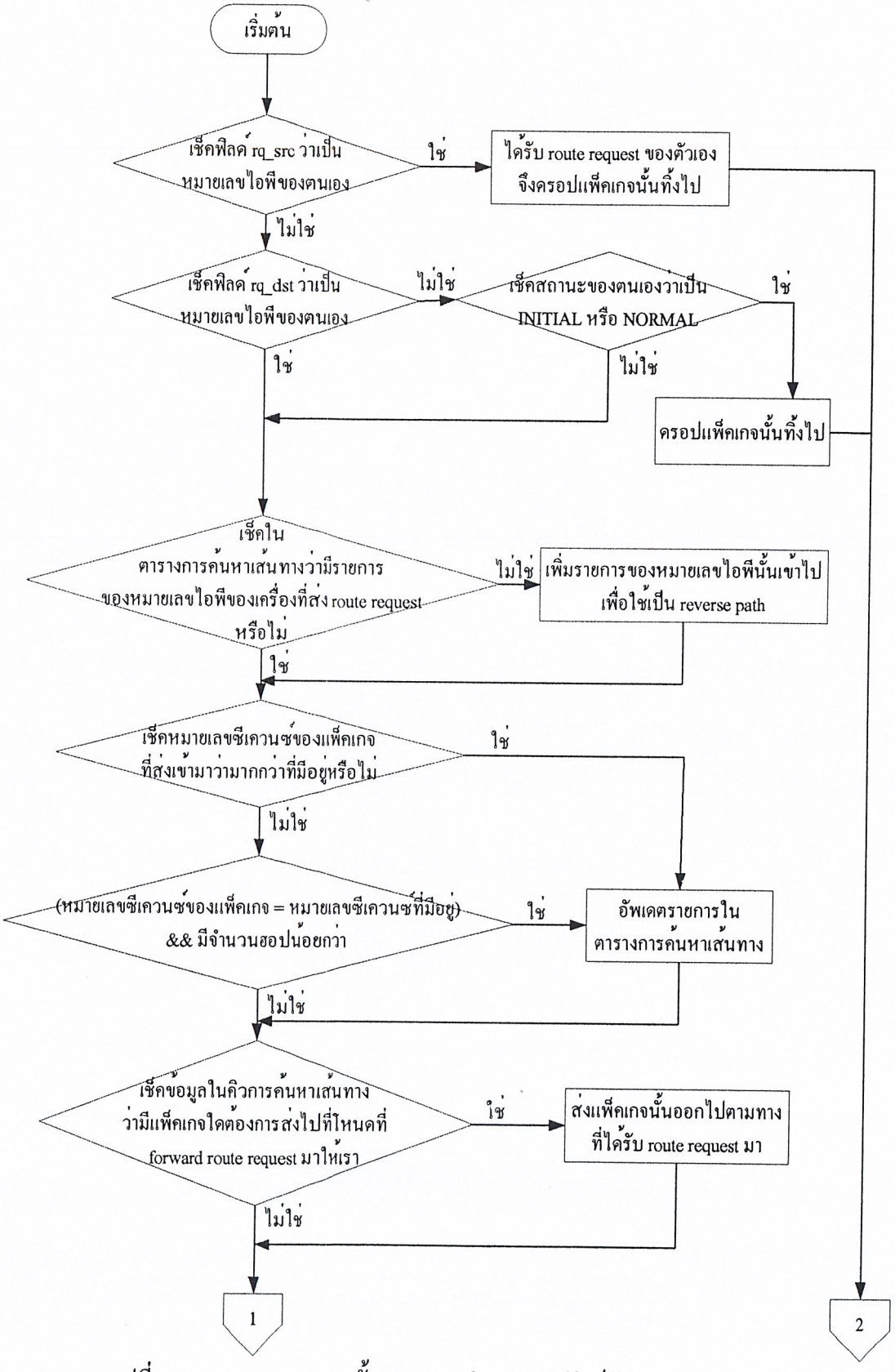
ฟังก์ชันนี้จะถูกเรียกใช้เมื่อแพ็คเกจที่โหนดนั้นๆรับเข้ามาเป็นแพ็คเกจ DOCR โดยฟังก์ชันนี้จะเป็นฟังก์ชันที่ใช้พิจารณาชนิดของแพ็คเกจ DOCR ว่าเป็นแพ็คเกจชนิดใดระหว่าง แพ็คเกจ RREQ แพ็คเกจ RREP แพ็คเกจ RERR หรือแพ็คเกจ HELLO แล้วจะทำการส่งแพ็คเกจเหล่านั้นไปยังฟังก์ชันที่มีการทำงานตรงตามชนิดของแพ็คเกจ DOCR เหล่านั้น แต่ถ้าพบว่าไม่ตรงกับแพ็คเกจชนิดใดๆเลย นั่นหมายความว่าเกิดความผิดพลาดขึ้น

ฟังก์ชัน recvRequest( )

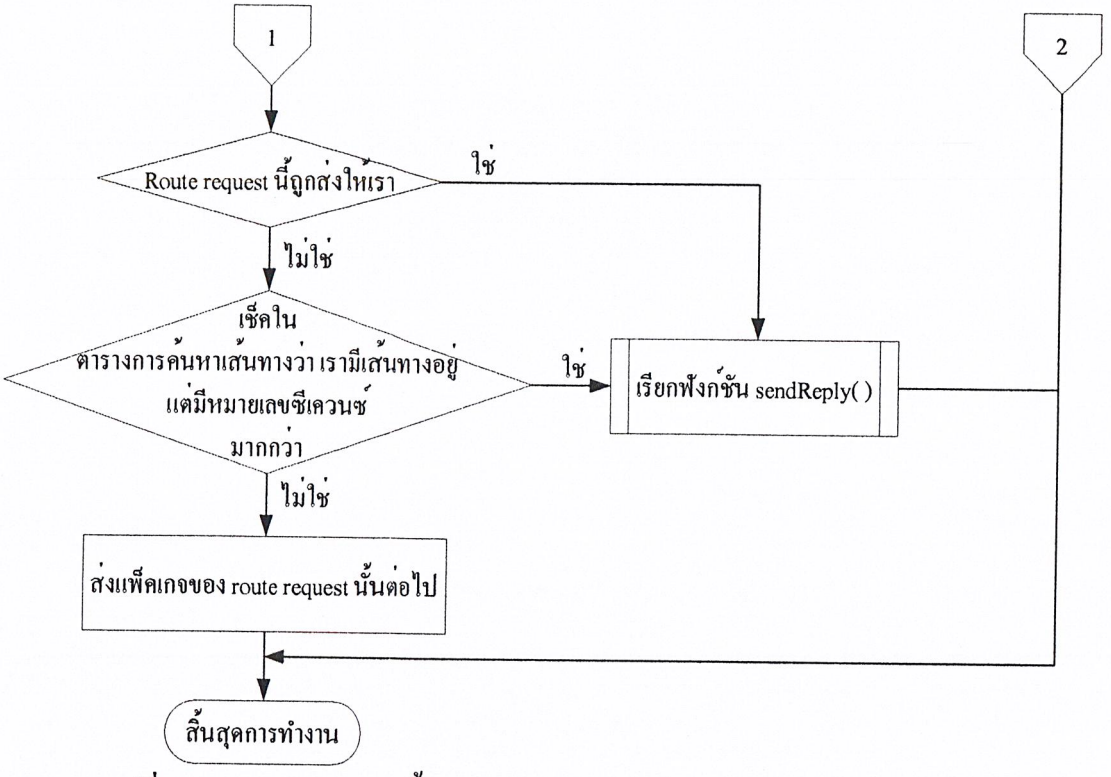
หลังจากที่ทราบว่าแพ็คเกจ DOCR นั้นเป็นแพ็คเกจ RREQ ฟังก์ชันนี้จะถูกเรียกใช้ เพื่อนำไปใช้ในการหาเส้นทาง ซึ่งจะมีการพิจารณาเงื่อนไขของเส้นทางออก เป็นเส้นทางเก่า และเส้นทางใหม่ เส้นทางที่เก่าและล้าหลังจะถูกลบออกไป แล้วแทนที่ด้วยเส้นทางใหม่ เพื่อให้เส้นทางที่เลือกใช้มีประสิทธิภาพมากที่สุด



รูปที่ 5.15 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvDOCR( )



รูปที่ 5.16 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvRequest( )



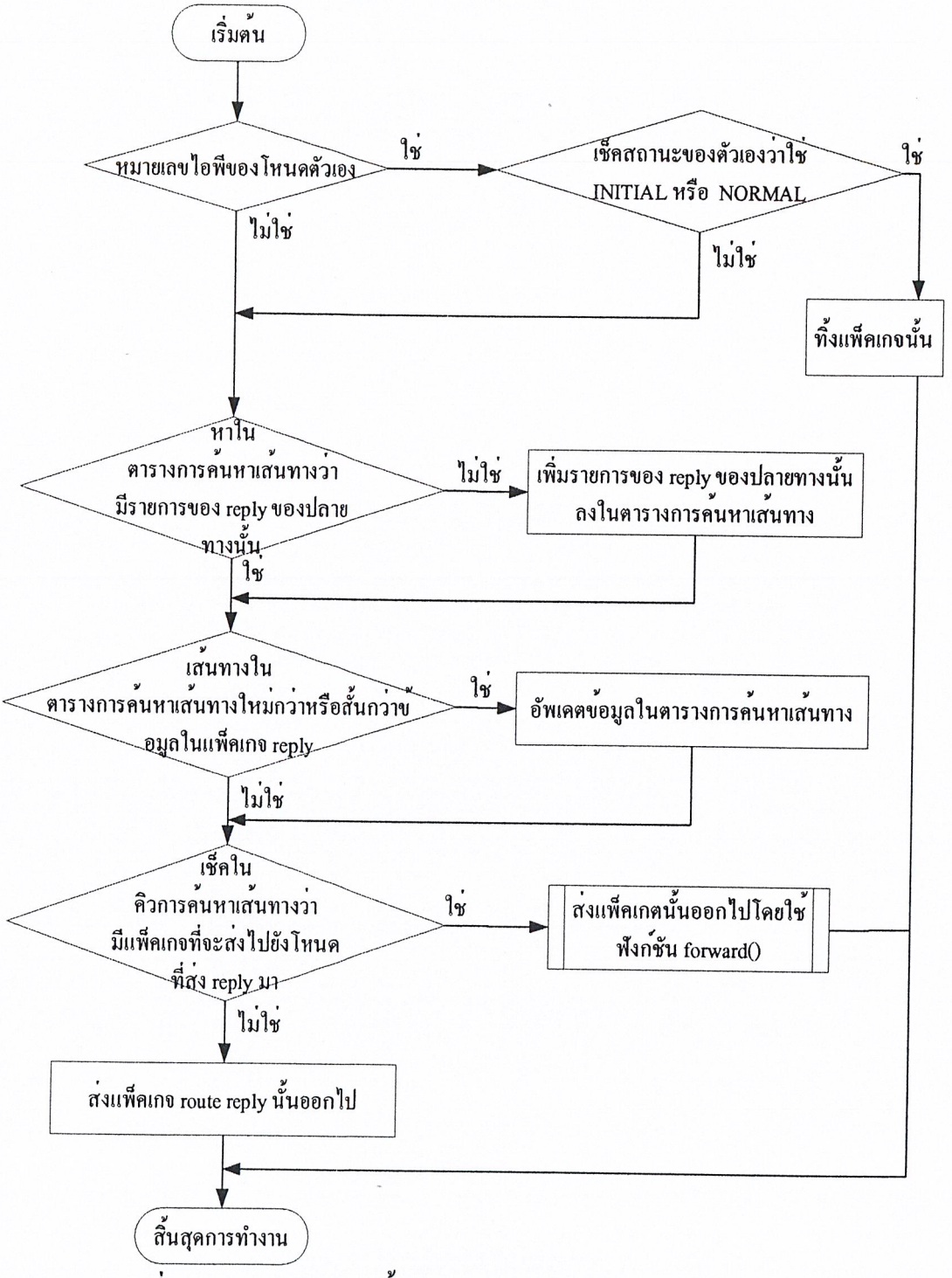
รูปที่ 5.17 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvRequest( ) (ต่อ)

ฟังก์ชัน recvReply( )

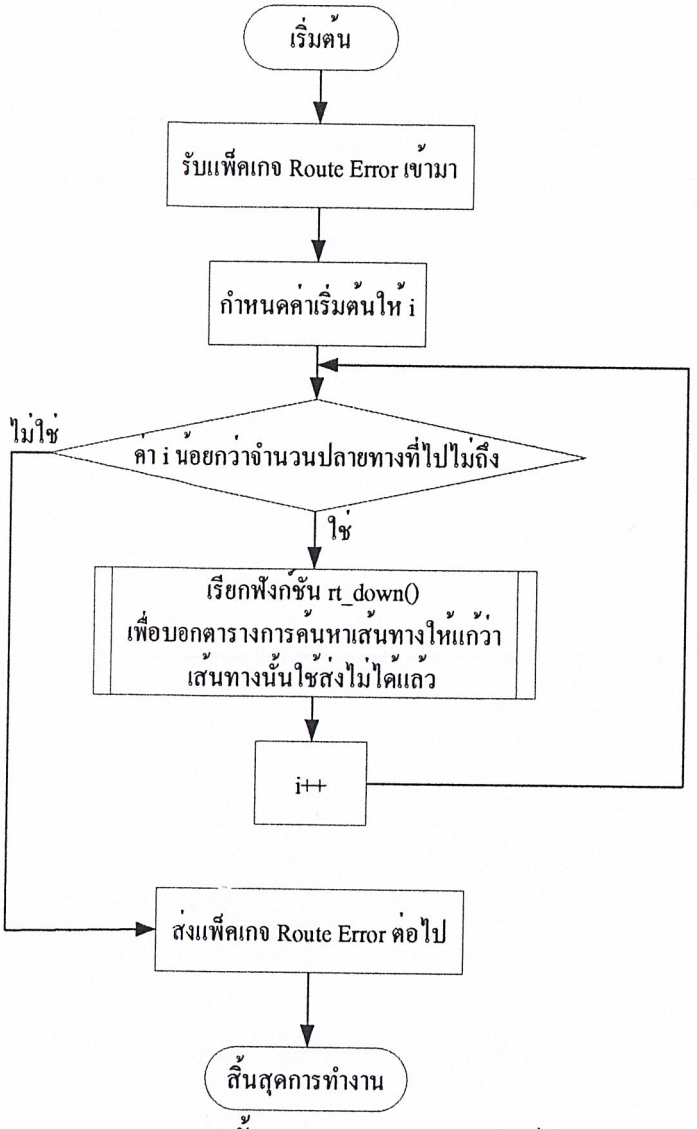
ฟังก์ชันนี้ถูกเรียกใช้เมื่อแพ็คเกจที่รับเข้ามาเป็นแพ็คเกจ RREP แพ็คเกจ มีขั้นตอนการทำงานคล้ายกับฟังก์ชัน RREQ แต่จะมีทิศทางตรงข้ามกัน คือ จะเป็นการส่งแพ็คเกจสวนทางกลับไปยังโหนดต้นทางที่ต้องการหาเส้นทาง โดยในระหว่างทางนั้น แต่ละโหนดที่ได้รับแพ็คเกจจะมีการอัพเดตตารางการค้นหาเส้นทางของตัวเองไปพร้อมๆกันด้วย

ฟังก์ชัน recvError( )

เป็นฟังก์ชันที่มีเงื่อนไขในการส่งแพ็คเกจ RRER ต่อไปยังโหนดอื่นๆ โดยกำหนดว่าจะต้องมีจำนวนของโหนดปลายทางที่แพ็คเกจถูกส่งไปไม่ถึงในปริมาณที่มากกว่าที่กำหนดไว้ จึงจะมีการส่งต่อแพ็คเกจ RRER อย่างไรก็ตาม แต่ละโหนดจะมีการเรียกใช้ฟังก์ชัน rt_down( ) เพื่อให้มีการแก้ไขตารางการค้นหาเส้นทางว่า ได้เกิดเส้นทางที่ใช้การไม่ได้เกิดขึ้น



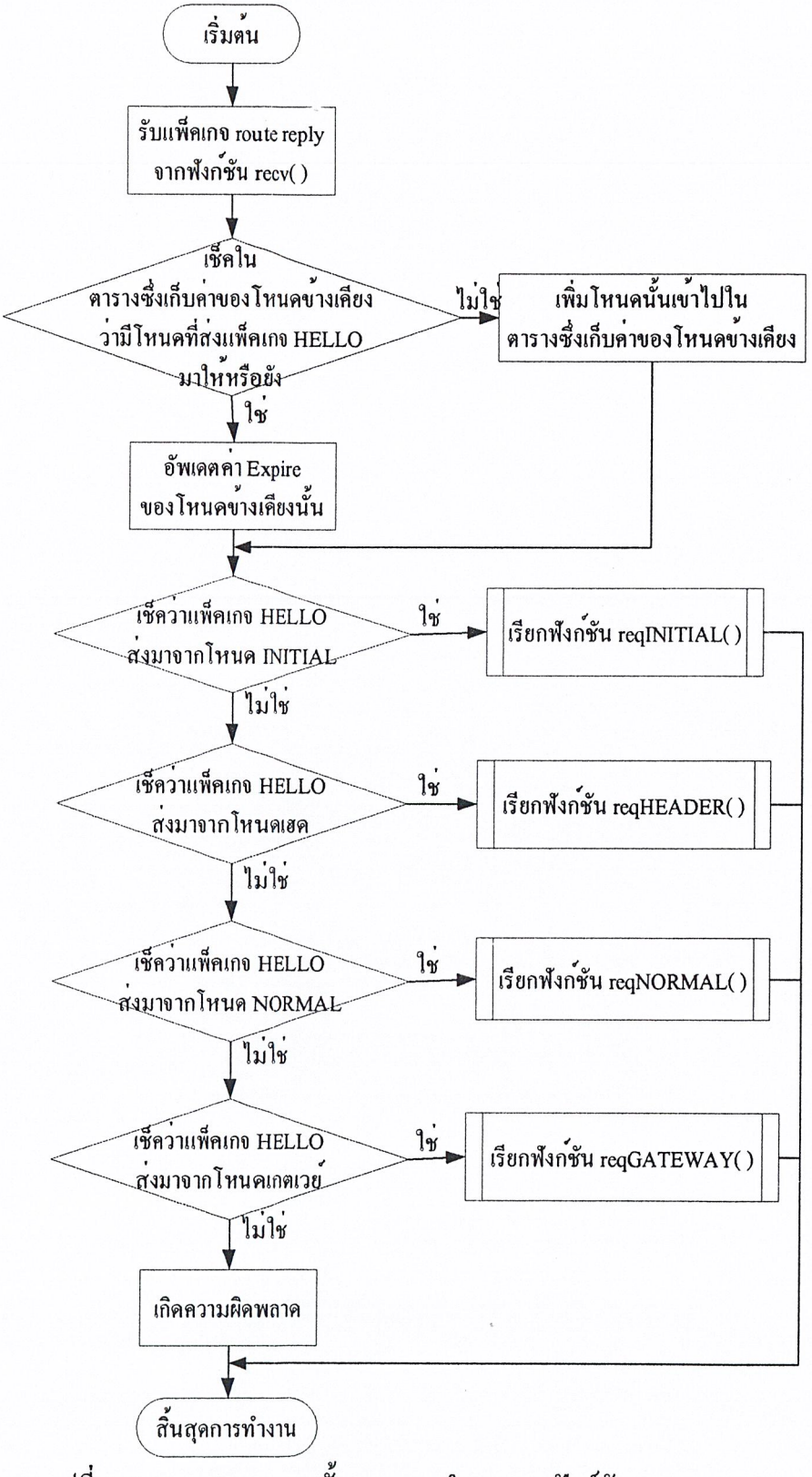
รูปที่ 5.18 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvReply( )



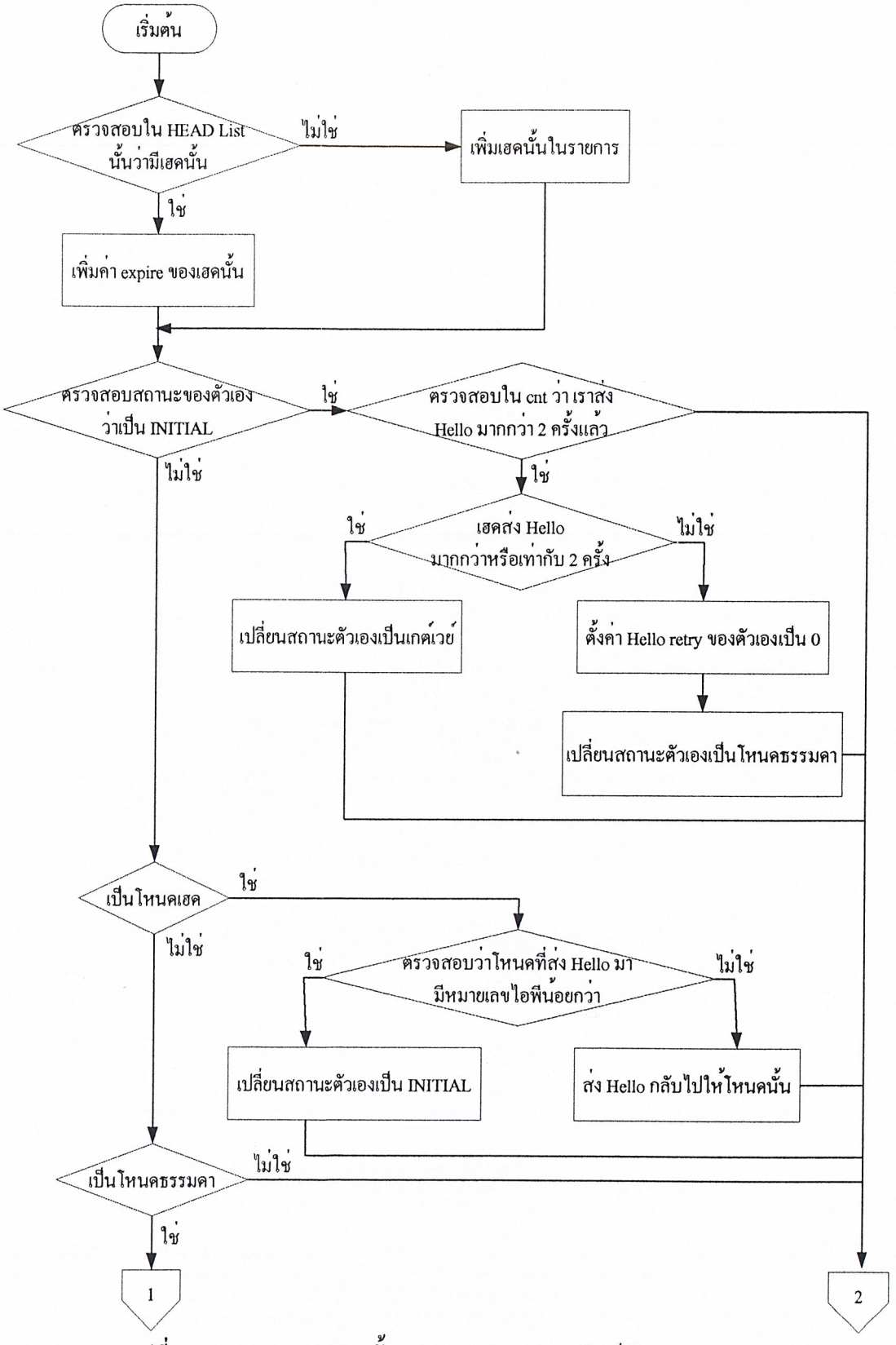
รูปที่ 5.19 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvError( )

ฟังก์ชัน recvHello( )

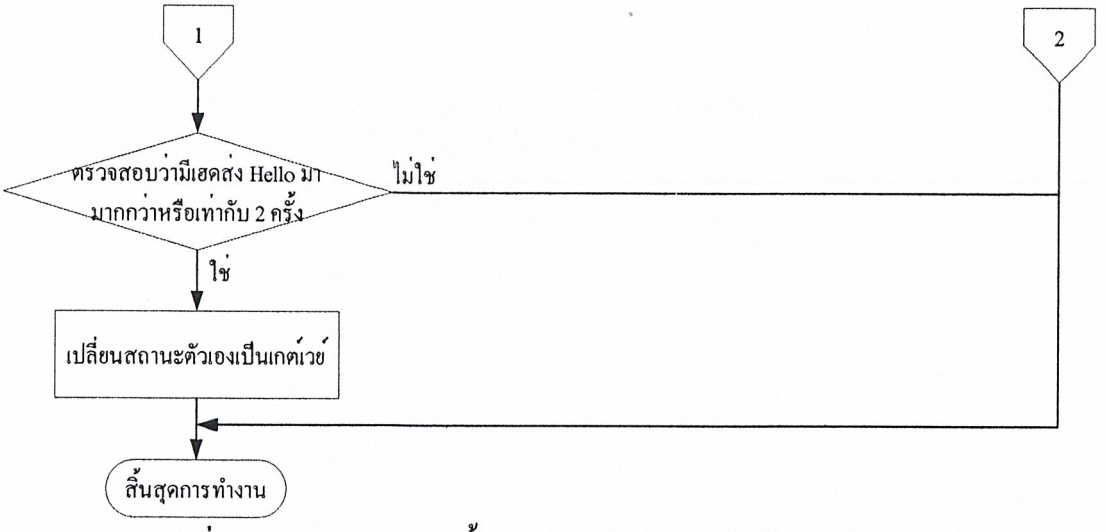
ฟังก์ชันจะถูกเรียกใช้เมื่อโหนดทราบว่าแพ็คเกจที่ได้รับมานั้นเป็นแพ็คเกจ HELLO โดยโหนดนั้นๆ จะทำการพิจารณาในตารางซึ่งเก็บค่าของโหนดข้างเคียงของตนว่ามีข้อมูลของโหนดที่ส่งแพ็คเกจ Hello นั้นมาให้แล้วหรือไม่ หลังจากนั้นจะพิจารณาต่อว่าโหนดที่ส่งมานั้นเป็นโหนดที่มีสถานะใด แล้วเรียกฟังก์ชันที่ทำงานตามสถานะของโหนดนั้นๆมาใช้ต่อไป



รูปที่ 5.20 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน recvHello( )



รูปที่ 5.21 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน reqHead( )



รูปที่ 5.22 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน reqHead( )

ฟังก์ชัน reqHead( )

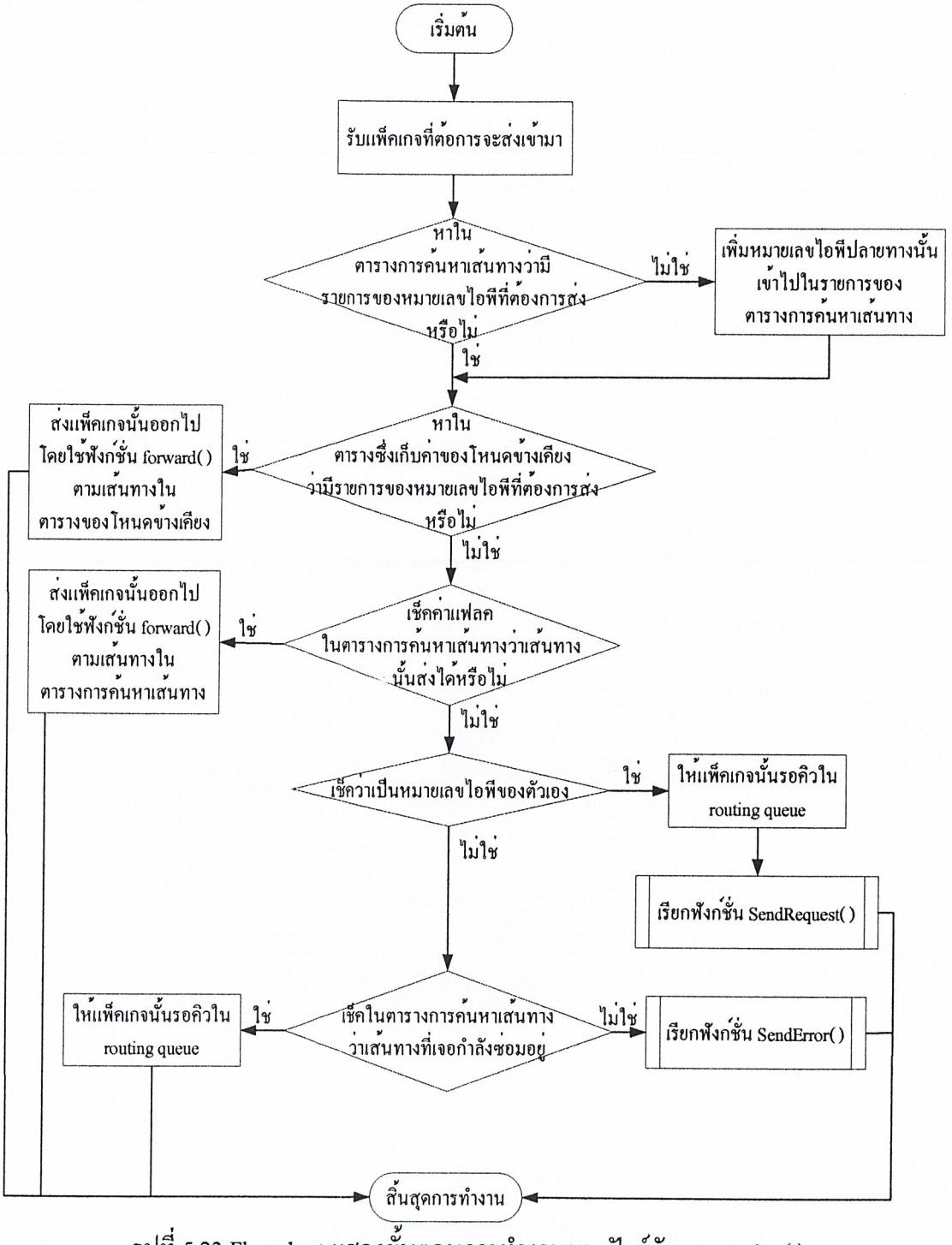
เมื่อโหนดใดๆ ได้รับแพ็คเกจ HELLO จากโหนดเฮด จะมีการเรียกใช้ฟังก์ชันนี้ ซึ่งฟังก์ชัน reqHead( ) นี้จะมีเงื่อนไขต่างๆที่ใช้พิจารณาในการเปลี่ยนสถานะของโหนด เมื่อได้รับแพ็คเกจ HELLO จากโหนดเฮดไม่ว่าจะเป็นการเปลี่ยนสถานะของตัวเองเป็นเกต์เวย์ เมื่อโหนดนั้นได้รับแพ็คเกจ HELLO จากโหนดเฮดมากกว่าหรือเท่ากับสองครั้ง หรือการเปลี่ยนสถานะของตัวเองเป็นโหนดธรรมดา หรือเป็นโหนดที่ไม่มีสถานะ (Initial node) ก็ตาม

ฟังก์ชัน rt_resolve( )

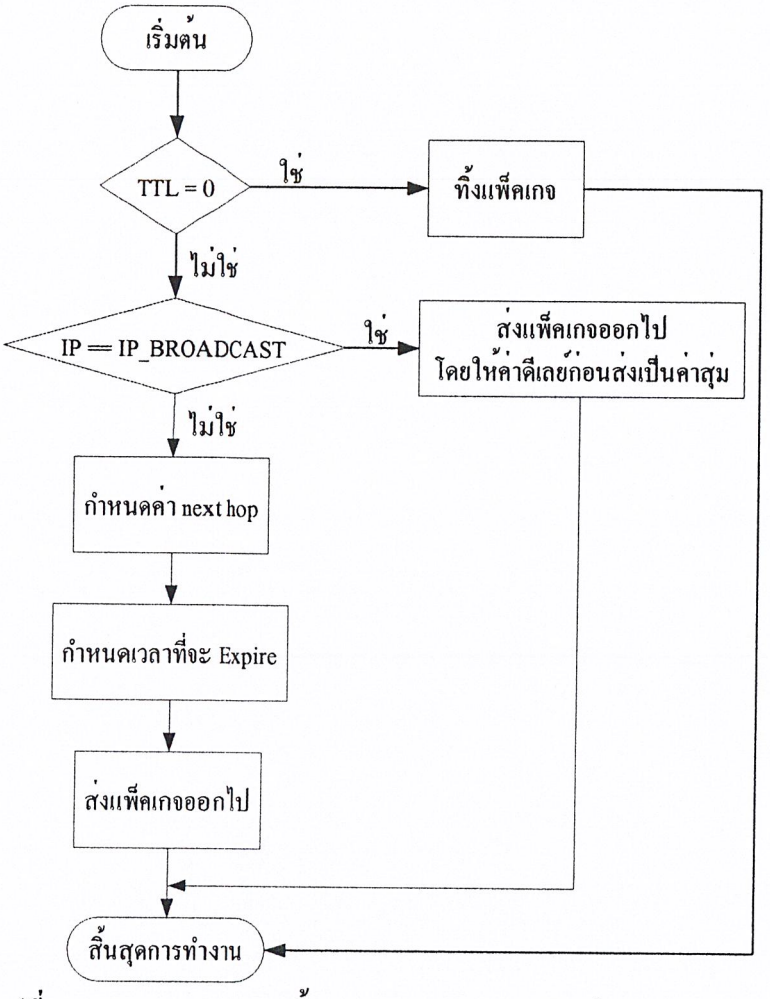
เป็นฟังก์ชันที่ถูกเรียกใช้เมื่อมีโหนดนั้นๆต้องการส่งแพ็คเกจออกไป โดยจะตรวจสอบทั้งเงื่อนไขของตารางการค้นหาเส้นทาง และตารางซึ่งเก็บค่าของโหนดข้างเคียง แล้วทำตามเงื่อนไขที่กำหนด เป็นที่สังเกตว่า ฟังก์ชันนี้จะถูกเรียกใช้เมื่อมีโหนดต้องการส่งแพ็คเกจออกไป ไม่ว่าจะมีเส้นทางที่จะส่งแล้วหรือไม่ก็ตาม

ฟังก์ชัน forward( )

ฟังก์ชันนี้จะถูกเรียกใช้ เมื่อโหนดต้องการส่งต่อข้อมูลออกไป โดยมีเงื่อนไขของเวลาเป็นตัวกำหนดอายุของแพ็คเกจ



รูปที่ 5.23 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน rt_resolve( )



รูปที่ 5.24 Flowchart แสดงขั้นตอนการทำงานของฟังก์ชัน forward( )

บทที่ 6
ผลการทดลอง

6.1 ค่าสภาพแวดล้อมที่ใช้ในการจำลองเครือข่าย และคำจำกัดความของค่าที่ใช้ในการวัดประสิทธิภาพของโปรโตคอลการค้นหาเส้นทาง

6.1.1 ค่าตัวแปรต่างๆที่กำหนดในการจำลองระบบเครือข่าย

- รัศมีของแต่ละโหนด กำหนดให้เป็นแบบสมมาตร (Symmetric) มีค่าเท่ากับ 250 เมตรเท่ากันทุกโหนด
- ช่องสัญญาณเป็นช่องสัญญาณแบบไร้สาย
- การแพร่กระจายคลื่นแบบ Two-ray
- ลักษณะทางกายภาพในการจำลองบน NS-2 เป็นแบบไร้สาย
- มาตรฐานของแมคเลเยอร์เป็นไปตามมาตรฐาน IEEE802.11
- คิวของอินเตอร์เฟสระหว่างเราท์ติ้งเลเยอร์และแมคเลเยอร์มีขนาดเท่ากับ 50

6.1.2 คำจำกัดความของค่าที่ใช้ในการวัดประสิทธิภาพของโปรโตคอลการค้นหาเส้นทางที่ถูกพัฒนาในโครงงาน

- ค่าทรูพุต (Throughput) สัดส่วนปริมาณข้อมูลที่รับได้ที่แอพพลิเคชันเลเยอร์กับเวลาที่ใช้ในการส่ง
- ค่าเวลาดีเลย์ (Delay time) คือ ค่าเวลาที่ใช้ในการส่งแพ็คเกตระหว่างด้านส่งและด้านรับ
- ปริมาณแพ็คเกตที่ใช้ในการค้นหาเส้นทาง
- ค่า Normalized ของปริมาณแพ็คเกตที่ใช้ในการค้นหาเส้นทาง สัดส่วนของปริมาณแพ็คเกตที่ใช้ในการค้นหาเส้นทางกับปริมาณแพ็คเกตที่ส่ง

6.2 การแสดงผลและวิเคราะห์ผลการทดลอง

ประกอบด้วย

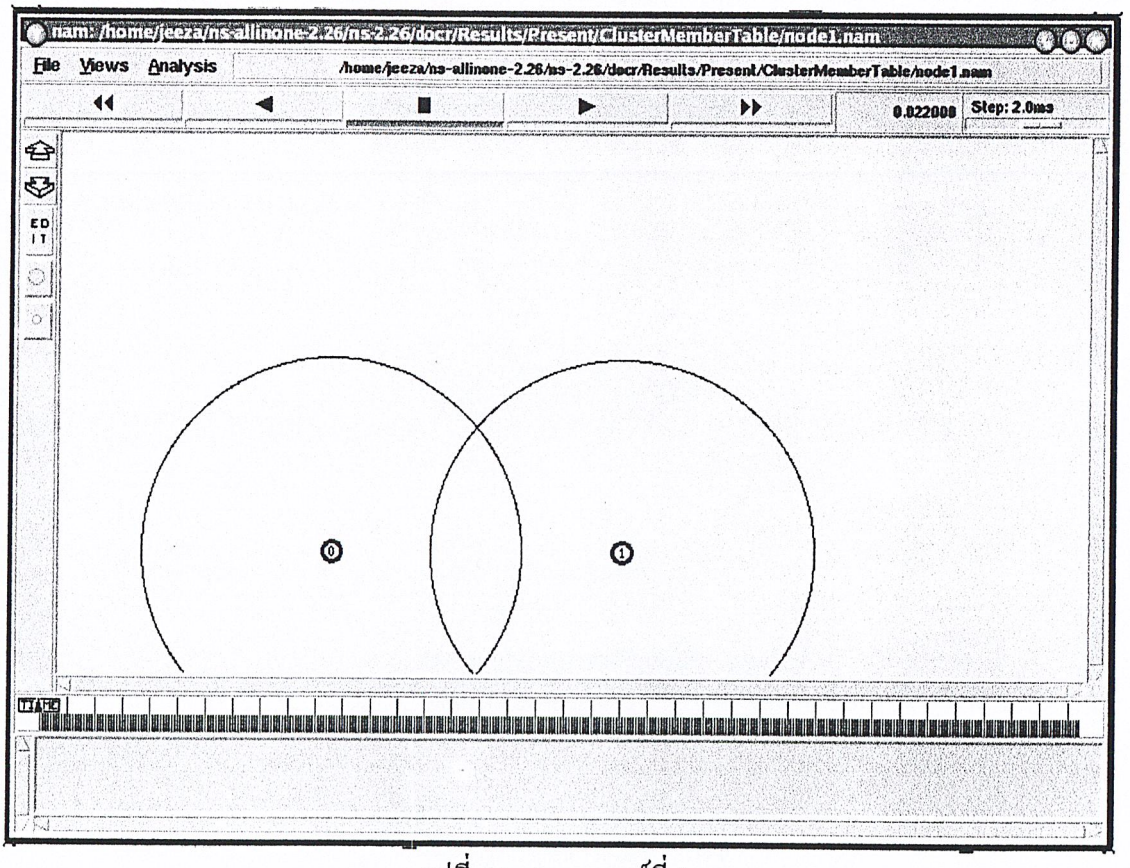
- รูปจากการจำลองระบบเครือข่าย โดยใช้ NS-2

- การทดลอง ผลการทดลอง และการวิเคราะห์ผลการทดลอง
- ไฟล์แสดงผลการจัดกลุ่มโหนด

6.3 การจำลองการจัดกลุ่มของโหนดในเครือข่าย

แบ่งการทดลองออกเป็น 8 สถานการณ์

สถานการณ์ที่ 1 เมื่อมีโหนดจำนวน 2 โหนดเคลื่อนที่อยู่ภายในเครือข่าย แต่ตัวโหนดไม่อยู่ในรัศมีของอีกโหนดหนึ่ง

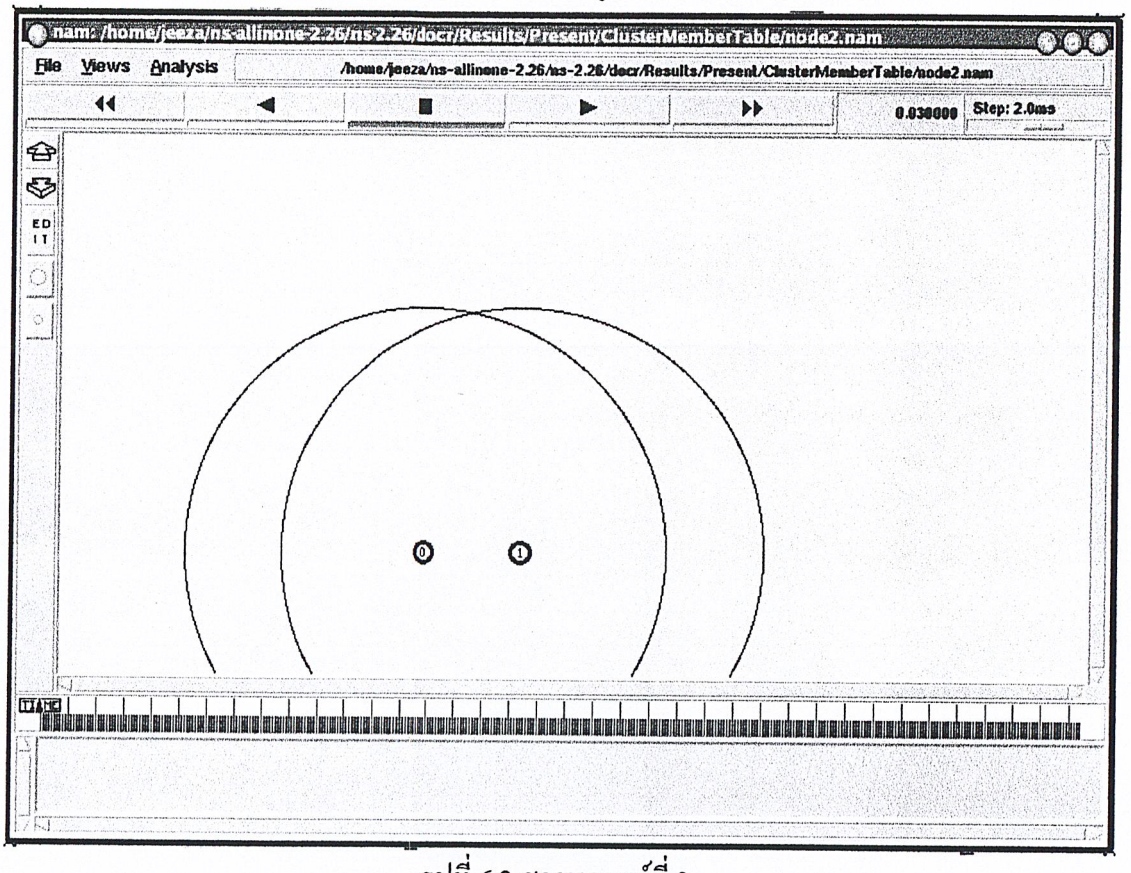


รูปที่ 6.1 สถานการณ์ที่ 1

โหนดทั้งสองจะส่งแพ็คเกต HELLO เพื่อตรวจสอบว่ามีโหนดคลัสเตอร์เฮดอยู่ภายในรัศมีของตนหรือไม่ ซึ่งหลังจากการทดลองแล้วพบว่า ทั้งสองโหนดไม่มีโหนดคลัสเตอร์เฮดอื่นอยู่ภายในรัศมีของตนเอง ดังนั้นโหนดทั้งสองจะเปลี่ยนสถานะของตนเองเป็นคลัสเตอร์เฮดทั้งคู่ มีไฟล์แสดงผลการจัดกลุ่มโหนด ดังนี้

- Set status of 0 to HEADER at 2.68
- Set status of 1 to HEADER at 3.11

สถานการณ์ที่ 2 เมื่อมีโหนดจำนวน 2 โหนดอยู่นิ่งในรัศมีของกันและกัน

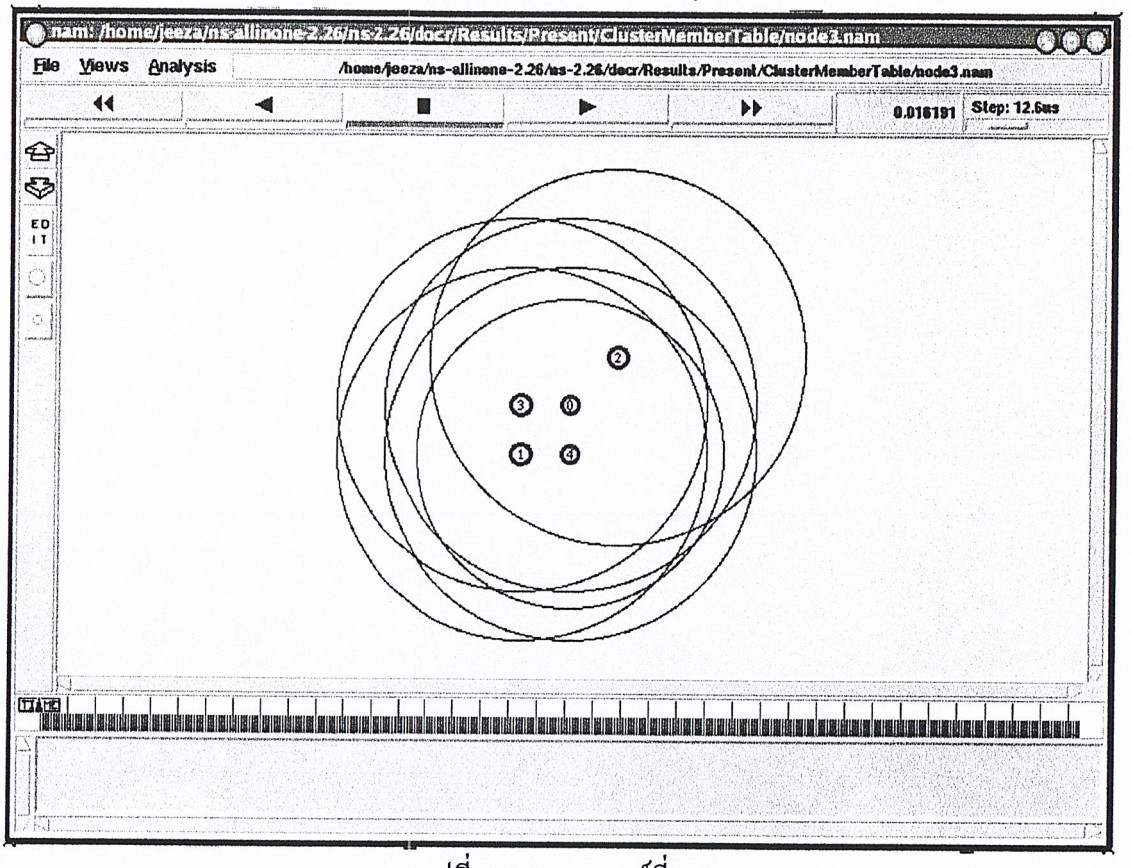


รูปที่ 6.2 สถานการณ์ที่ 2

เมื่อโหนดทั้งสองเคลื่อนที่เข้ามาอยู่ภายในรัศมีของกันและกัน จะมีเพียงโหนดเดียวเท่านั้นที่ยังคงสถานะเป็นเฮดได้ ส่วนอีกโหนดหนึ่งจะเปลี่ยนสถานะตัวเองเป็นโหนดธรรมดา ได้ไฟล์แสดงผลการจัดกลุ่มโหนด ดังนี้

- Set status of 0 to HEADER at 2.68
- Change status of 1 from INITIAL to NORMAL at 2.68

สถานการณ์ที่ 3 เมื่อมีโหนดจำนวน 5 โหนดเคลื่อนที่อยู่ภายในรัศมีของกันและกัน

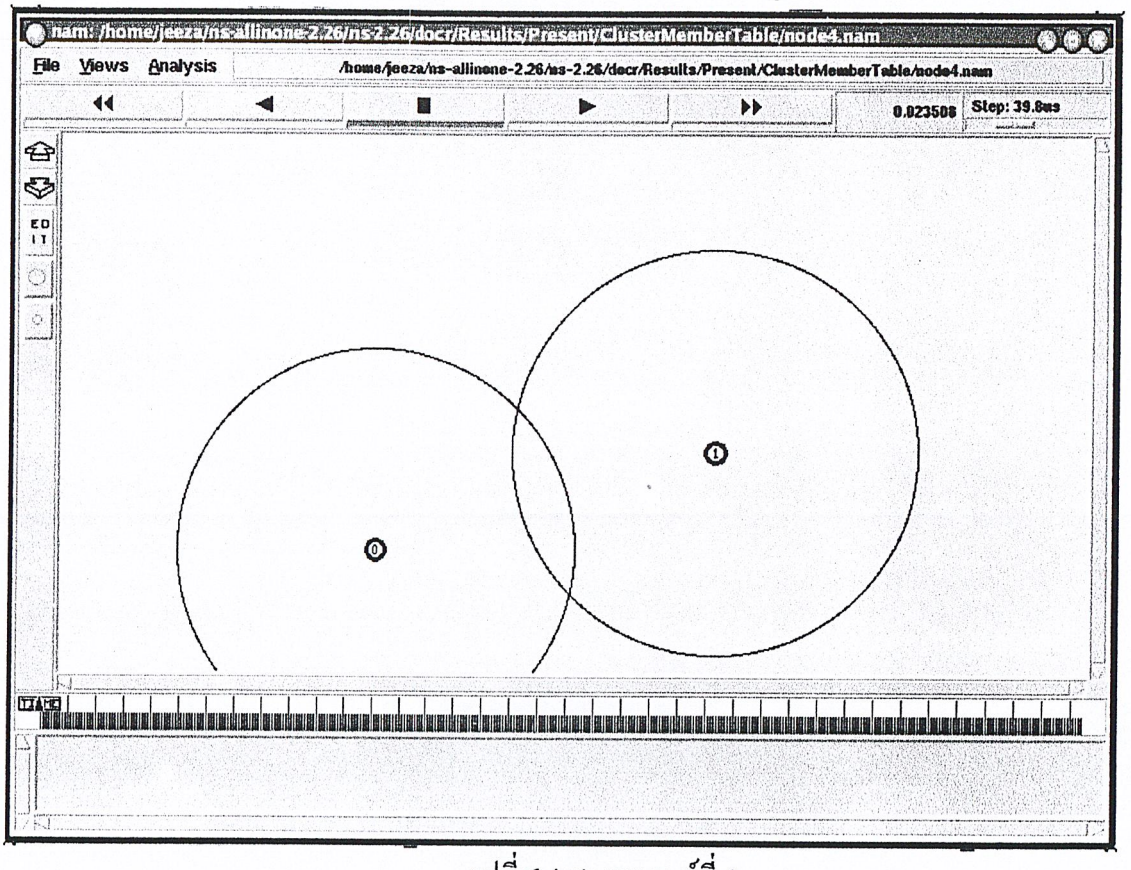


รูปที่ 6.3 สถานการณ์ที่ 3

ในสถานการณ์จำลองซึ่งมีโหนดจำนวน 5 โหนดเคลื่อนที่อยู่ภายในรัศมีของกันและกัน หลังจากเวลาผ่านไปไม่นาน จะมีหนึ่งโหนดที่เป็นคลัสเตอร์เฮด ส่วนโหนดที่เหลืออีก 4 โหนดมีสถานะเป็นโหนดธรรมดา ซึ่งจะขึ้นตรงกับโหนดที่เป็นคลัสเตอร์เฮดนั้น จะได้ผลไฟล์แสดงผลการจัดกลุ่มโหนด ดังแสดงด้านล่าง

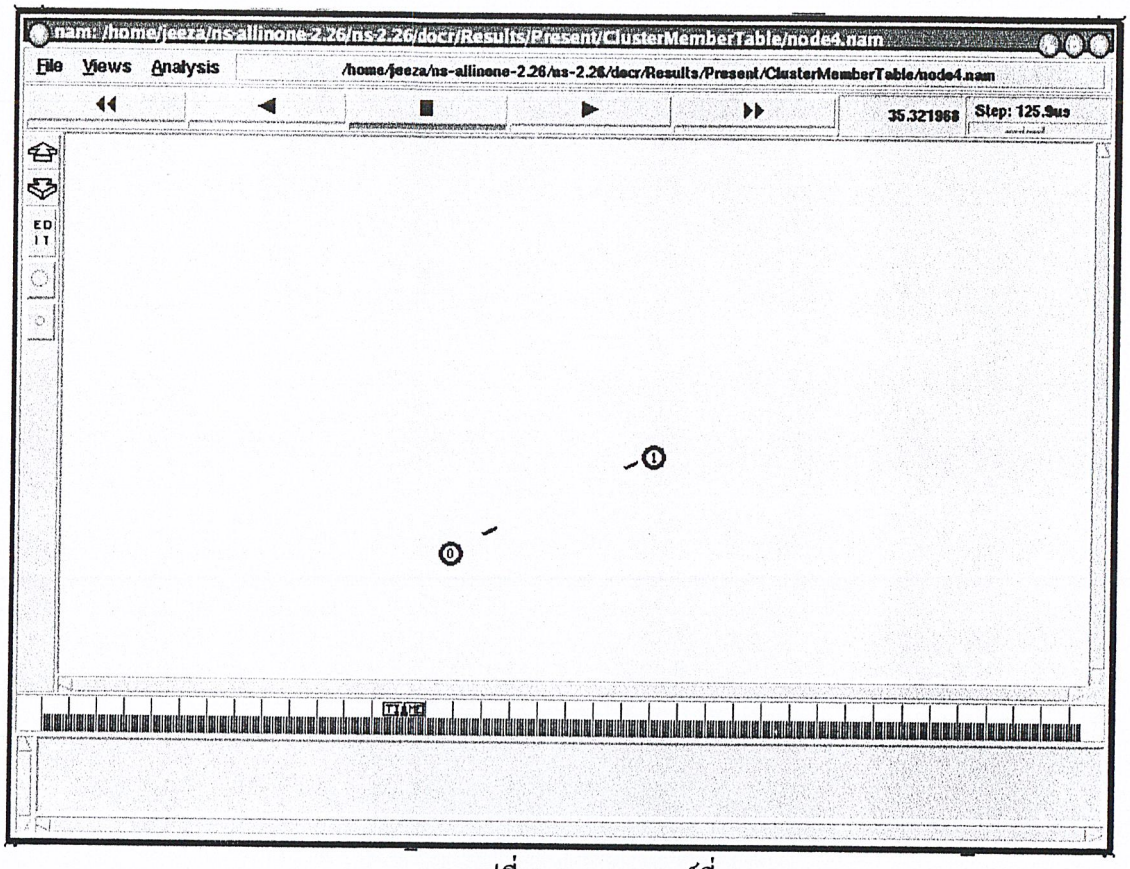
- Set status of 2 to HEADER at 2.59
- Change status of 0 from INITIAL to NORMAL at 2.59
- Change status of 4 from INITIAL to NORMAL at 2.59
- Change status of 3 from INITIAL to NORMAL at 2.59
- Change status of 1 from INITIAL to NORMAL at 2.59

สถานการณ์ที่ 4 เมื่อมีโหนดคลัสเตอร์เฮดจำนวน 2 โหนดซึ่งอยู่ห่างกัน แล้วเคลื่อนที่เข้าหากัน



รูปที่ 6.4 สถานการณ์ที่ 4

สถานการณ์นี้จะคล้ายกับการนำเอาสถานการณ์ที่ 1 และสถานการณ์ที่ 2 มารวมกัน นั่นคือ เมื่อเวลาเริ่มต้นของการจำลองสถานการณ์ โหนดคลัสเตอร์เฮดทั้งสองโหนดไม่อยู่ในรัศมีของกันและกัน หลังจากนั้นทั้งสองโหนดเคลื่อนที่เข้าหากัน หรือโหนดหนึ่งวิ่งเข้าหาอีกโหนดหนึ่ง จะทำให้โหนดหนึ่งต้องเปลี่ยนสถานะของตัวเองเป็นโหนดธรรมดาไป ในขณะที่อีกโหนดหนึ่งจะยังคงเป็นคลัสเตอร์เฮดต่อไป

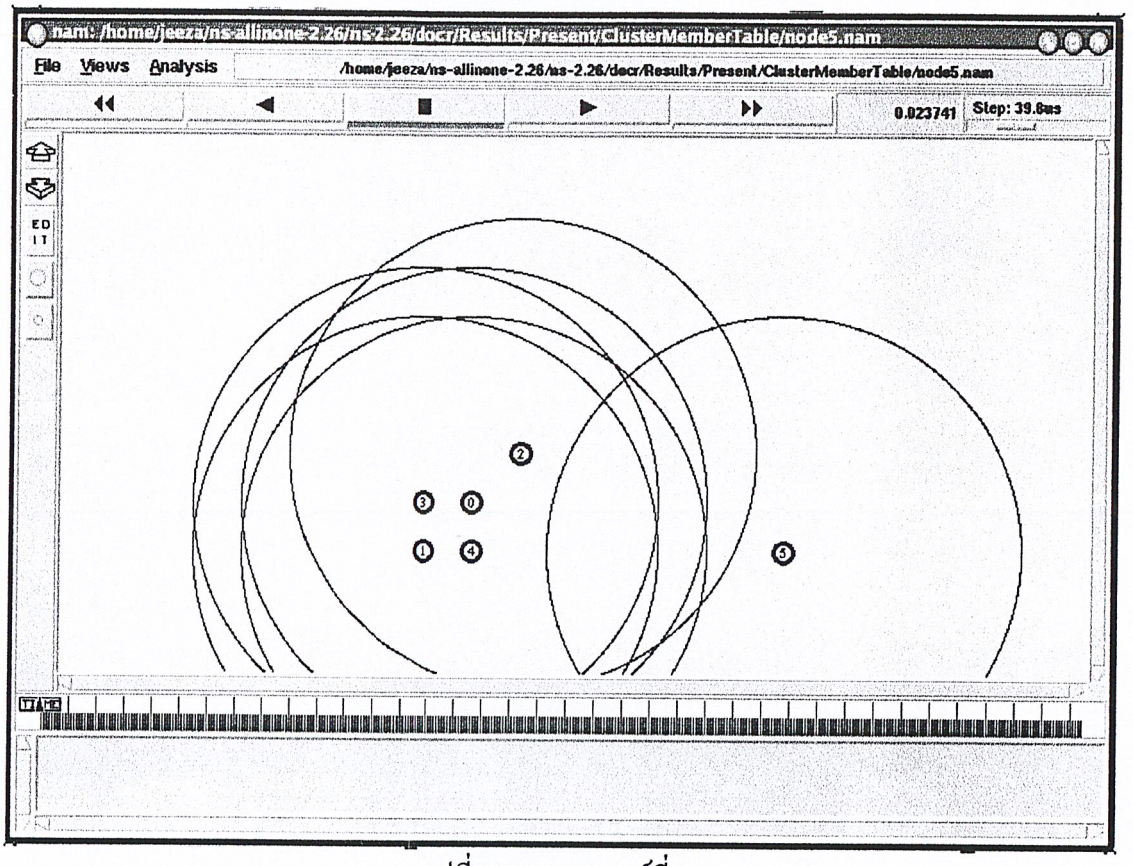


รูปที่ 6.5 สถานการณ์ที่ 4

เมื่อทำการทดลองแล้ว จะได้ไฟล์แสดงผลการจัดกลุ่มโหนด ดังนี้

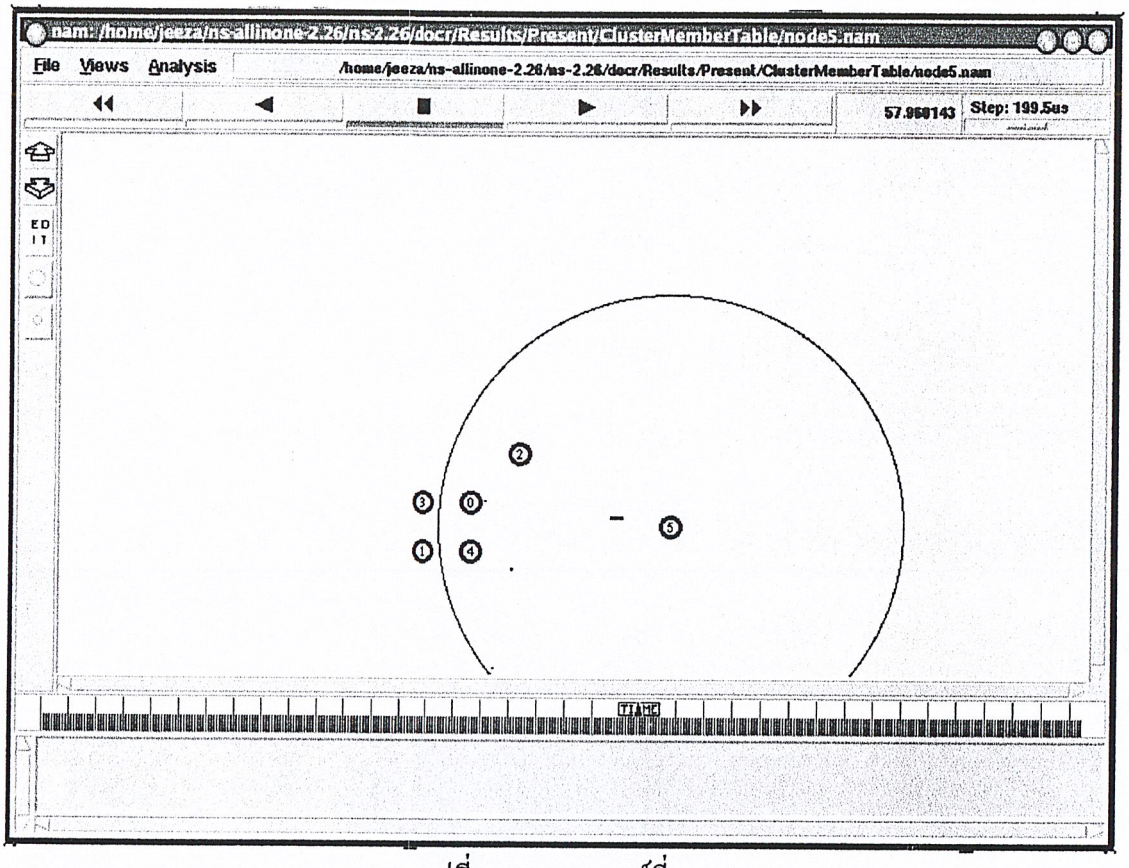
- Set status of 0 to HEADER at 2.68
- Set status of 1 to HEADER at 3.11
- Change status of 1 from HEADER to INITIAL at 34.83
- Change status of 1 from INITIAL to NORMAL at 37.23

สถานการณ์ที่ 5 เมื่อมีโหนดหนึ่งเคลื่อนที่เข้าหาคลัสเตอร์หนึ่งๆ



รูปที่ 6.6 สถานการณ์ที่ 5

เป็นที่ทราบกันแล้วว่า ในทุกๆคลัสเตอร์จะต้องมีโหนดหนึ่งที่เป็นคลัสเตอร์เฮด ซึ่งคอยควบคุมโหนดอื่นๆที่อยู่ภายในคลัสเตอร์นั้นๆ ในที่นี้ กำหนดให้โหนด 0 มีสถานะเป็นคลัสเตอร์เฮด ส่วนโหนดอื่นๆ อันได้แก่ โหนด 1 โหนด 2 โหนด 3 โหนด 4 ให้มีสถานะเป็นโหนดธรรมดา ภายใต้การจำลองสถานการณ์นี้ สถานะของโหนดต่างๆไม่เปลี่ยนแปลง จนกระทั่งมีโหนด 5 ซึ่งมีเป็นคลัสเตอร์เฮดเคลื่อนตัวเข้าหาคลัสเตอร์ ทำให้มีอาณาเขตบางส่วนของโหนดคลัสเตอร์เฮด ทั้งโหนด 0 และโหนด 5 เหลื่อมล้ำกัน และอาจมีโหนดบางตัวอยู่ในพื้นที่ดังกล่าว ในที่นี้ คือ โหนด 2 และโหนด 4 ทั้งสองโหนดนี้จะเปลี่ยนสถานะของตัวเองเป็นเกต์เวย์ ซึ่งจะทำหน้าที่เป็นตัวกลางในการติดต่อระหว่างคลัสเตอร์ที่มีโหนด 0 เป็นคลัสเตอร์เฮด และคลัสเตอร์ที่มีโหนด 5 เป็นคลัสเตอร์เฮด



รูปที่ 6.7 สถานการณ์ที่ 5

ถ้าโหนด 5 ยังคงเคลื่อนที่เข้าใกล้โหนด 0 ต่อไป จนตัวโหนด 5 อยู่ภายใต้อาณาเขตของโหนด0 ทั้งโหนด 5 และโหนด 0 มีเป็นคลัสเตอร์เฮดทั้งคู่ ซึ่งเป็นไปไม่ได้ที่คลัสเตอร์เฮด 2 ตัวอยู่ภายในคลัสเตอร์เดียวกัน ดังนั้นโหนด 5 (ในการจำลองสถานการณ์นี้โหนด 5 ถูกกำหนดให้มีหมายเลขไอพีมากกว่า โหนด 0) จะเปลี่ยนสถานะไปเป็นโหนดธรรมดาภายใต้คลัสเตอร์ที่มีโหนด 0 เป็นคลัสเตอร์เฮด ส่วนโหนด 2 และโหนด 4 ซึ่งเคยเป็นเกต์เวย์ระหว่างคลัสเตอร์ที่มีโหนด 0 เป็นคลัสเตอร์เฮด และคลัสเตอร์ที่มีโหนด 5 เป็นคลัสเตอร์เฮด ก็จะเปลี่ยนสถานะตัวเองเป็นโหนดธรรมดาภายใต้คลัสเตอร์ที่มีโหนด 0 เป็นคลัสเตอร์เฮดด้วยเช่นกัน

ได้ไฟล์แสดงผลการจัดกลุ่มโหนดดังนี้

- Set status of 0 to HEADER at 2.46
- Change status of 4 from INITIAL to NORMAL at 2.46
- Change status of 3 from INITIAL to NORMAL at 2.46

- Change status of 2 from INITIAL to NORMAL at 2.46
- Change status of 1 from INITIAL to NORMAL at 2.46
- Set status of 5 to HEADER at 2.87
- Change status of 2 from NORMAL to GATEWAY at 53.02
- Change status of 4 from NORMAL to GATEWAY at 54.98
- Change status of 5 from HEADER to INITIAL at 54.98
- Change status of 5 from INITIAL to NORMAL at 57.36
- Set status of 2 to NORMAL (purge) at 60.00
- Set status of 4 to NORMAL (purge) at 60.00

Purge เป็นฟังก์ชันที่จะถูกเรียกใช้เมื่อถึงเวลาที่ตั้งไว้ ในที่นี้ คือ การตรวจสอบในตารางสมาชิกของคลัสเตอร์ (Cluster member table) ว่ามีโหนดคลัสเตอร์เฮดอยู่เป็นจำนวนเท่าใด ถ้าพบว่ามีเพียงหนึ่งโหนด สมาชิกโหนดที่เป็นเกต์เวย์จะต้องเปลี่ยนสถานะเป็นโหนดธรรมดา

สถานการณ์ที่ 6 เมื่อสุ่มโหนดขึ้นมาจำนวน 10 โหนด

ในสถานการณ์ที่ 6 นี้ เป็นการจำลองสถานการณ์โดยการสุ่มโหนดขึ้นมาจำนวน 10 โหนด แต่ละโหนดเป็นอิสระจากกัน มีชุดคำสั่งซึ่งใช้ในการควบคุมการออกแบบ 2 ชุด ได้แก่

- **Node movement**

เป็นชุดคำสั่งที่ใช้กำหนดลักษณะการเคลื่อนที่ของโหนดต่างๆ มีรูปแบบการใช้ชุดคำสั่ง ดังนี้

setdest [-n nodes] [-p pause time] [-s maximum speed] [-t simulation time] [-x maximum x] [-y maximum y] >[outdir/movement_file]

ในการทดลองจริงใช้คำสั่งตามด้านล่างนี้

-nn 10 –p 10 –s 10 –t 100 -x 500 –y 500 >scene-10

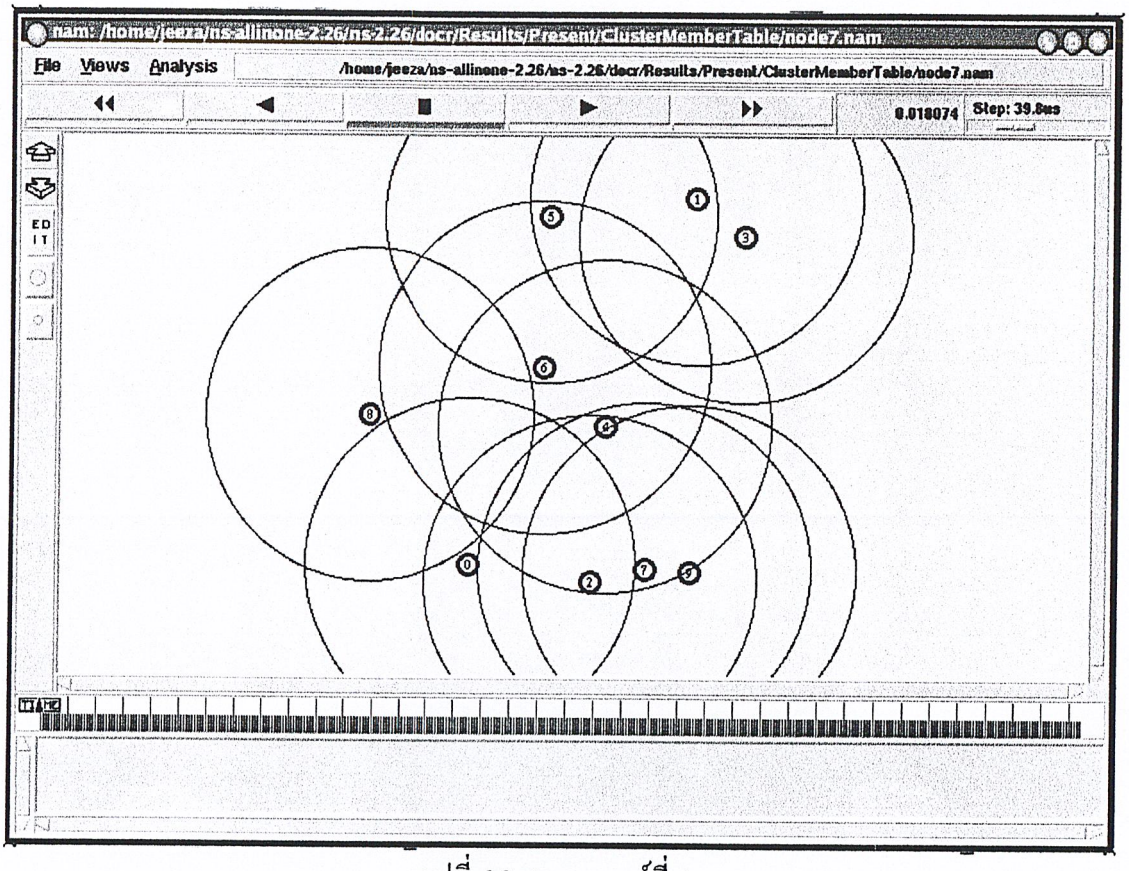
- **Traffic Pattern**

เป็นชุดคำสั่งที่ใช้ในการกำหนดรูปแบบทราฟฟิกบนเครือข่าย มีรูปแบบการใช้ชุดคำสั่ง ดังนี้

ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-seed seed] [-mc maximum connections] [-rate rate] >[outdir/traffic_file]

ในการทดลองจริงพิมพ์ชุดคำสั่ง ดังนี้

-type cbr –nn 10 –seed 0.1 –mc 7 –rate 8.0 >cbr-10-node



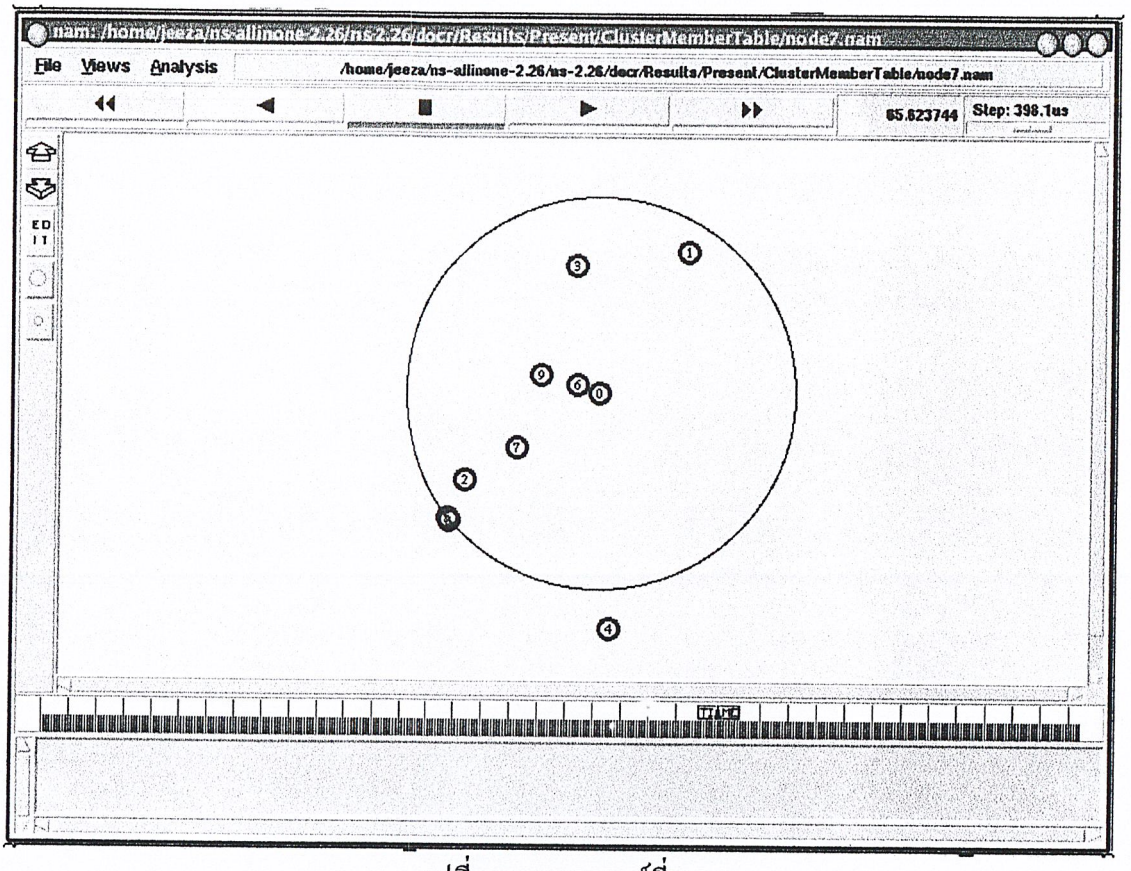
รูปที่ 6.8 สถานการณ์ที่ 6

หลังจากที่ทำการจำลองสถานการณ์ในเครือข่ายแล้ว ได้ไฟล์แสดงผลการจัดกลุ่มโหนด ดังนี้

- Set status of 2 to HEADER at 2.75
- Change status of 7 from INITIAL to NORMAL at 2.75
- Change status of 9 from INITIAL to NORMAL at 2.75
- Change status of 0 from INITIAL to NORMAL at 2.75
- Change status of 4 from INITIAL to NORMAL at 2.75
- Change status of 6 from INITIAL to NORMAL at 2.75
- Set status of 3 to HEADER at 2.80
- Change status of 1 from INITIAL to NORMAL at 2.80
- Change status of 5 from INITIAL to NORMAL at 2.80

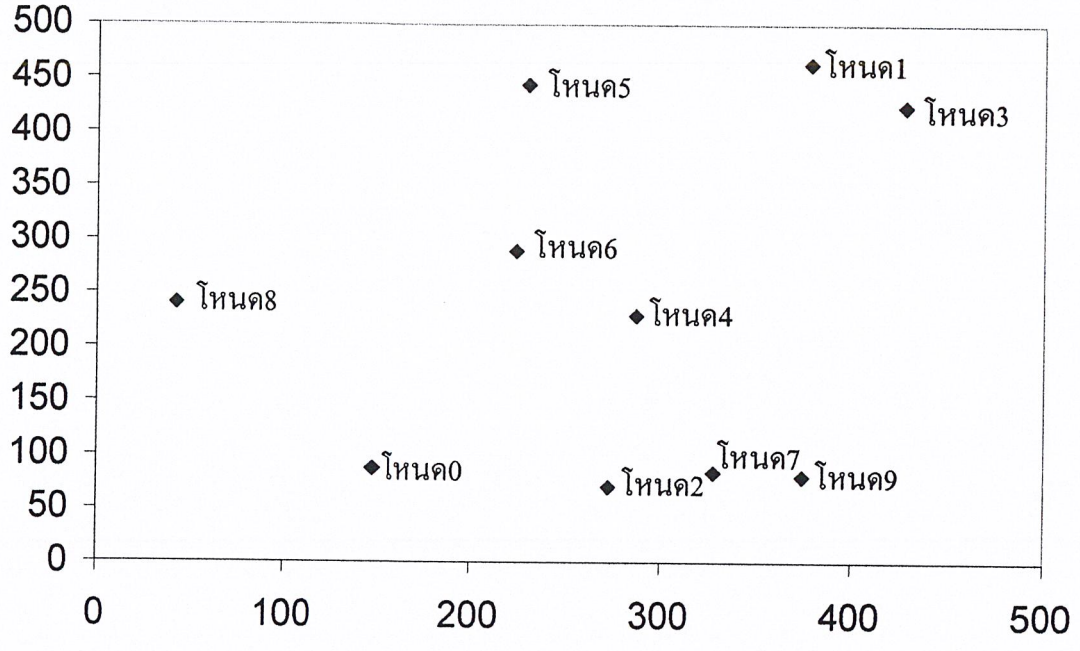
- Change status of 4 from NORMAL to GATEWAY at 2.80
- Change status of 6 from NORMAL to GATEWAY at 2.80
- Set status of 8 to HEADER at 3.09
- Change status of 0 from NORMAL to GATEWAY at 3.09
- Change status of 5 from NORMAL to GATEWAY at 13.60
- Change status of 8 from HEADER to INITIAL at 16.66
- Change status of 8 from INITIAL to NORMAL at 18.90
- Set status of 0 to NORMAL (purge) at 22.50
- Change status of 9 from NORMAL to GATEWAY at 28.30
- Set status of 4 to NORMAL (purge) at 33.00
- Change status of 0 from NORMAL to GATEWAY at 35.96
- Change status of 7 from NORMAL to GATEWAY at 45.27
- Change status of 3 from HEADER to INITIAL at 55.66
- Change status of 3 from INITIAL to NORMAL at 57.69
- Set status of 5 to NORMAL (purge) at 58.50
- Set status of 0 to NORMAL (purge) at 60.00
- Set status of 1 to INITIAL (purge) at 60.00
- Set status of 6 to NORMAL (purge) at 60.00
- Set status of 7 to NORMAL (purge) at 60.00
- Set status of 9 to NORMAL (purge) at 60.00
- Set status of 1 to HEADER at 63.63
- Change status of 6 from NORMAL to GATEWAY at 63.64
- Change status of 0 from NORMAL to GATEWAY at 63.64
- Change status of 9 from NORMAL to GATEWAY at 63.64
- Change status of 2 from HEADER to INITIAL at 83.36
- Change status of 2 from INITIAL to NORMAL at 86.15
- Set status of 0 to NORMAL (purge) at 88.50
- Set status of 3 to INITIAL (purge) at 88.50

- Set status of 4 to INITIAL (purge) at 88.50
- Set status of 5 to INITIAL (purge) at 88.50
- Set status of 6 to NORMAL (purge) at 88.50
- Set status of 7 to INITIAL (purge) at 88.50
- Set status of 9 to NORMAL (purge) at 88.50
- Change status of 3 from INITIAL to NORMAL at 90.20
- Set status of 5 to HEADER at 91.75
- Change status of 7 from INITIAL to NORMAL at 91.75
- Chang status of 9 from NORMAL to GATEWAY at 91.75
- Set status of 4 to HEADER at 91.77
- Change status of 7 from NORMAL to GATEWAY at 91.77
- Change status of 8 from NORMAL to GATEWAY at 91.77
- Change status of 6 from NORMAL to GATEWAY at 91.77
- Set status of 2 to INITIAL (purge) at 93.00
- Set status of 8 to NORMAL (purge) at 93.00
- Change status of 2 from INITIAL to NORMAL at 94.56

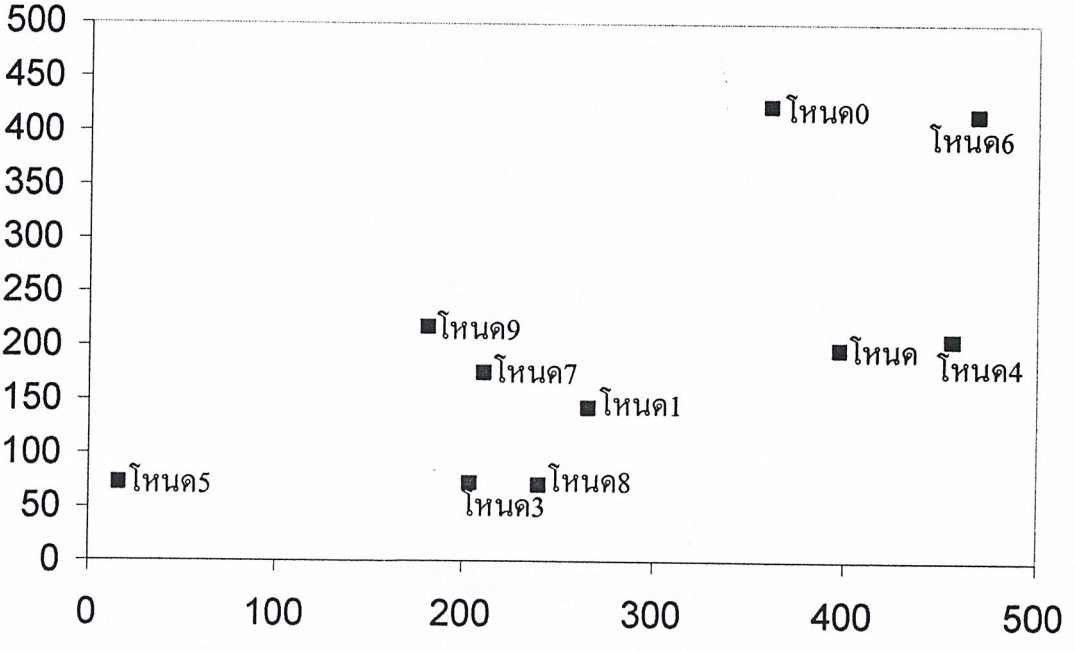


รูปที่ 6.9 สถานการณ์ที่ 6

รูปที่ 6.10 และรูปที่ 6.11 เป็นการพิกัดโหนดต่างๆลงบนพิกัด (x,y) เพื่อเป็นการเปรียบเทียบตำแหน่งของโหนดต่างๆ เมื่อเวลาเริ่มต้น และเมื่อเวลาสิ้นสุด

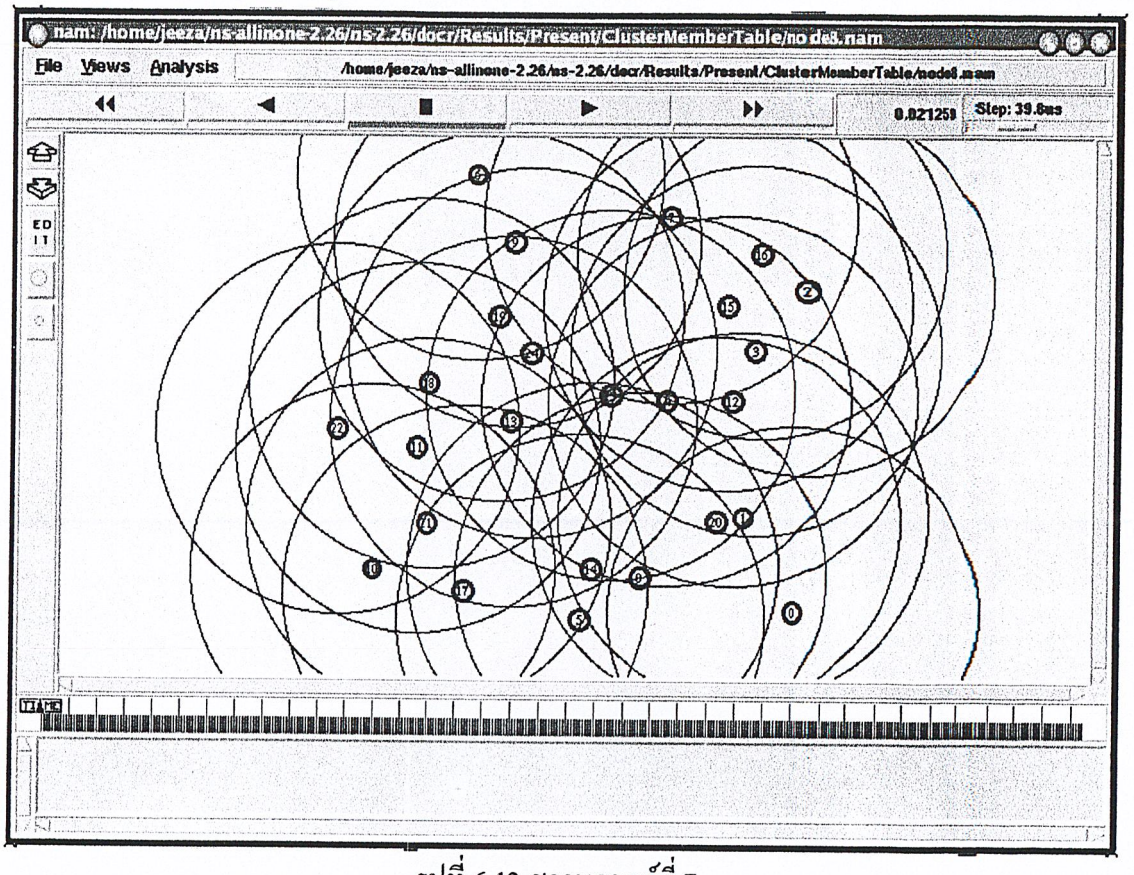


รูปที่ 6.10 กราฟแสดงตำแหน่งของโหนดต่างๆเมื่อเวลาเริ่มต้น



รูปที่ 6.11 กราฟแสดงตำแหน่งของโหนดเมื่อเวลาสิ้นสุด

สถานการณ์ที่ 7 เมื่อสุ่มโหนดขึ้นมาจำนวน 25 โหนด



รูปที่ 6.12 สถานการณ์ที่ 7

ชุดคำสั่งสองชุดด้านล่างเป็นการกำหนดการเคลื่อนที่ของโหนด และรูปแบบของทราฟฟิก ตามลำดับ โดยยังคงใช้ชุดคำสั่งเดียวกับการจำลองสถานการณ์ที่ 6 แต่เปลี่ยนเพียงบางค่าเท่านั้น มีชุดคำสั่งซึ่งใช้ในการควบคุมการออกแบบ ดังนี้

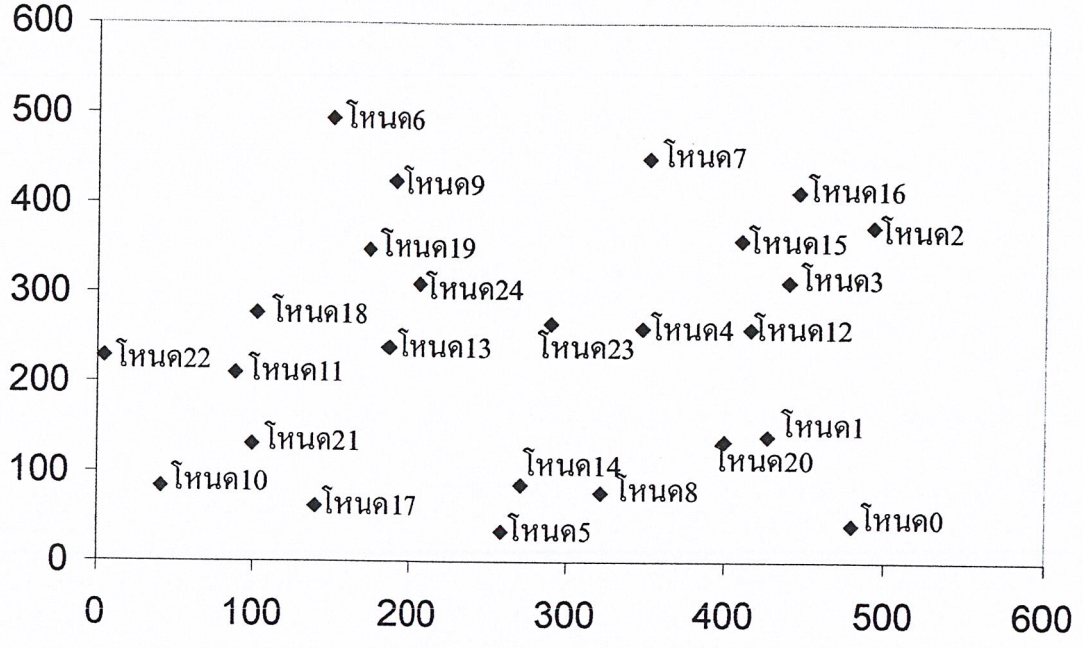
ชุดคำสั่งที่ใช้กำหนดลักษณะการเคลื่อนที่ของโหนดต่างๆ

-nn 25 –p 10 –s 10 –t 100 -x 500 –y 500 >scene-25

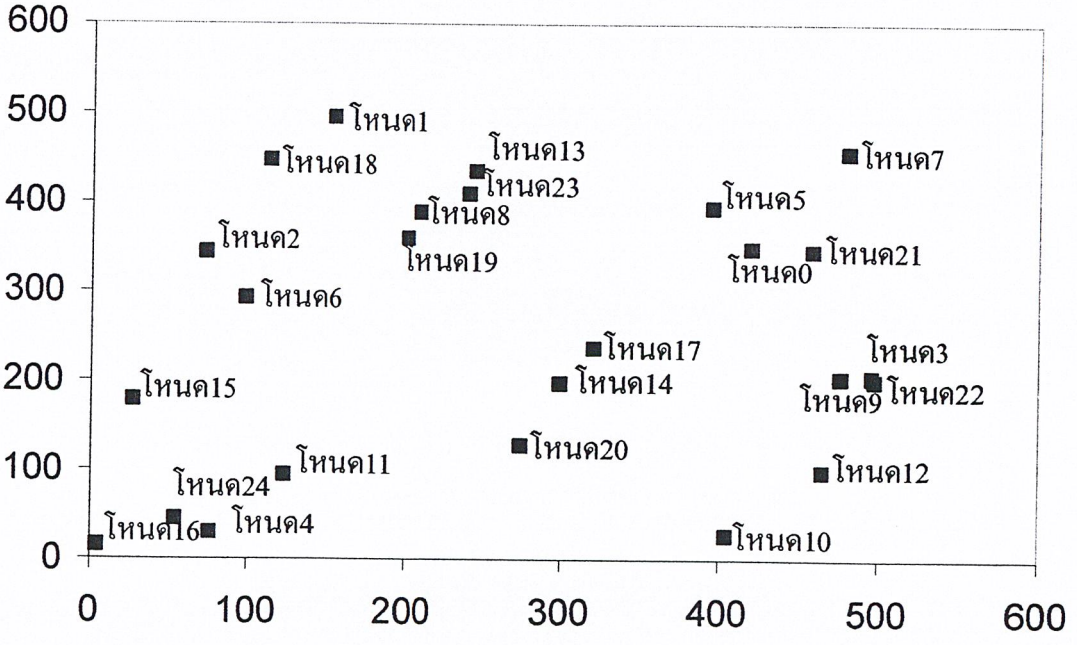
ชุดคำสั่งที่ใช้ในการกำหนดรูปแบบทราฟฟิกบนเครือข่าย

-type cbr –nn 25 –seed 0.1 –mc 15 –rate 8.0 >cbr-25-node

รูปที่ 6.13 และรูปที่ 6.14 เป็นการพิกัดโหนดต่างๆลงบนพิกัด (x,y) เพื่อเป็นการเปรียบเทียบตำแหน่งของโหนดต่างๆ เมื่อเวลาเริ่มต้น และเมื่อเวลาสิ้นสุด



รูปที่ 6.13 กราฟแสดงตำแหน่งของโหนดเมื่อเวลาเริ่มต้น



รูปที่ 6.14 กราฟแสดงตำแหน่งของโหนดเมื่อเวลาสิ้นสุด

เมื่อทำการทดลองแล้ว จะได้ไฟล์แสดงผลการจัดกลุ่มโหนด ดังแสดงด้านล่าง

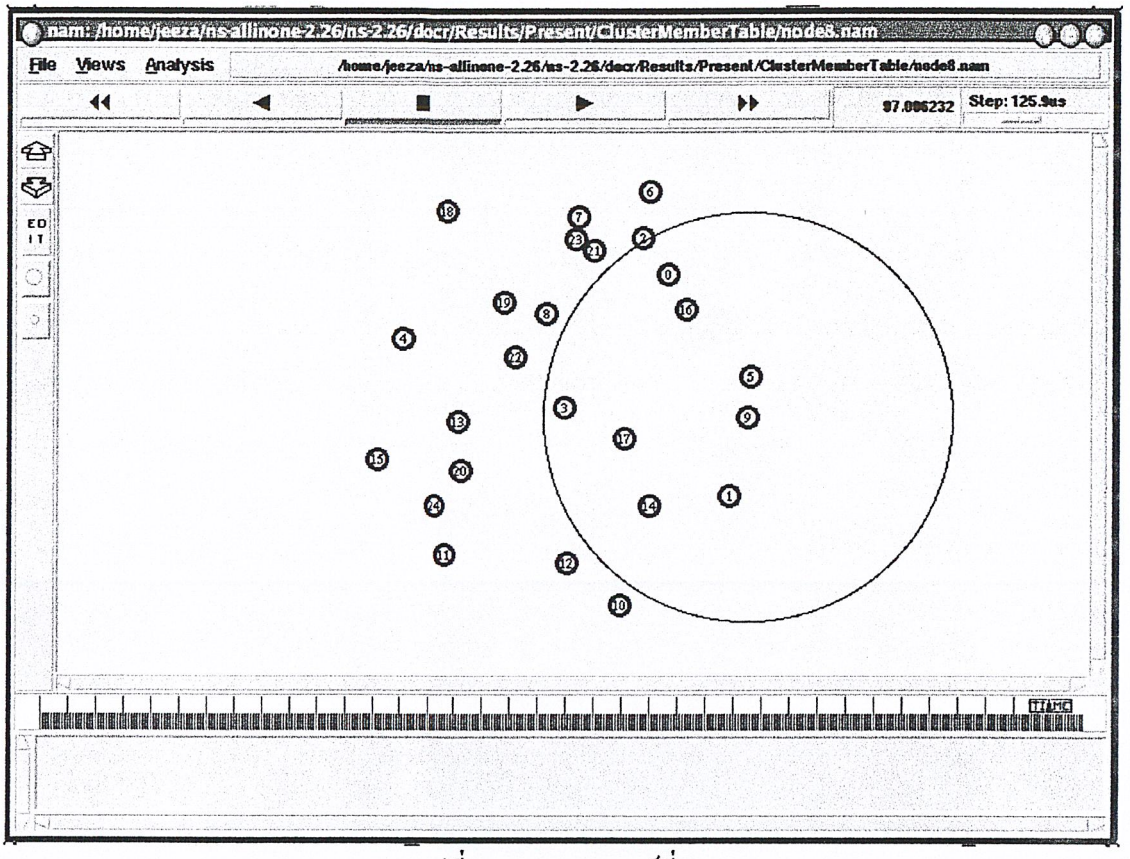
- Set status of 0 to HEADER at 2.29
- Change status of 1 from INITIAL to NORMAL at 2.29
- Change status of 20 from INITIAL to NORMAL at 2.29
- Change status of 8 from INITIAL to NORMAL at 2.29
- Change status of 14 from INITIAL to NORMAL at 2.29
- Change status of 5 from INITIAL to NORMAL at 2.29
- Change status of 12 from INITIAL to NORMAL at 2.29
- Set status of 13 to HEADER at 2.47
- Change status of 24 from INITIAL to NORMAL at 2.47
- Change status of 18 from INITIAL to NORMAL at 2.47
- Change status of 11 from INITIAL to NORMAL at 2.47
- Change status of 23 from INITIAL to NORMAL at 2.47
- Change status of 19 from INITIAL to NORMAL at 2.47
- Change status of 21 from INITIAL to NORMAL at 2.47
- Change status of 4 from INITIAL to NORMAL at 2.47
- Chang status of 14 from NORMAL to GATEWAY at 2.47
- Change status of 22 from INITIAL to NORMAL at 2.47
- Change status of 17 from INITIAL to NORMAL at 2.47
- Change status of 9 from INITIAL to NORMAL at 2.47
- Chang status of 8 from NORMAL to GATEWAY at 2.47
- Change status of 10 from INITIAL to NORMAL at 2.47
- Change status of 5 from NORMAL to GATEWAY at 2.47
- Change status of 12 from NORMAL to GATEWAY at 2.47
- Change status of 20 from NORMAL to GATEWAY at 2.47
- Set status of 6 to HEADER at 2.64
- Change status of 9 from NORMAL to GATEWAY at 2.64
- Change status of 19 from NORMAL to GATEWAY at 2.64

- Change status of 24 from NORMAL to GATEWAY at 2.64
- Change status of 7 from INITIAL to NORMAL at 2.64
- Chang status of 18 from NORMAL to GATEWAY at 2.64
- Set status of 2 to HEADER at 2.72
- Change status of 16 from INITIAL to NORMAL at 2.72
- Change status of 3 from INITIAL to NORMAL at 2.72
- Change status of 15 from INITIAL to NORMAL at 2.72
- Change status of 7 from NORMAL to GATEWAY at 2.72
- Change status of 4 from NORMAL to GATEWAY at 2.72
- Change status of 23 from NORMAL to GATEWAY at 2.72
- Change status of 1 from NORMAL to GATEWAY at 2.72
- Change status of 3 from NORMAL to GATEWAY at 12.59
- Change status of 15 from NORMAL to GATEWAY at 16.51
- Change status of 22 from NORMAL to GATEWAY at 16.51
- Change status of 17 from NORMAL to GATEWAY at 20.87
- Change status of 2 from HEADER to INITIAL at 20.87
- Change status of 16 from NORMAL to GATEWAY at 22.31
- Change status of 2 from INITIAL to NORMAL at 23.50
- Set status of 1 to NORMAL (purge) at 25.50
- Set status of 7 to NORMAL (purge) at 25.50
- Set status of 9 to NORMAL (purge) at 25.50
- Change status of 9 from NORMAL to GATEWAY at 29.25
- Set status of 5 to NORMAL (purge) at 30.00
- Change status of 11 from NORMAL to GATEWAY at 30.04
- Change status of 21 from NORMAL to GATEWAY at 31.48
- Set status of 14 to NORMAL (purge) at 31.50
- Set status of 19 to NORMAL (purge) at 33.00
- Set status of 22 to NORMAL (purge) at 34.50

- Change status of 19 from NORMAL to GATEWAY at 34.62
- Change status of 6 from HEADER to INITIAL at 37.45
- Set status of 8 to NORMAL (purge) at 39.00
- Set status of 17 to NORMAL (purge) at 39.00
- Set status of 21 to NORMAL (purge) at 39.00
- Change status of 6 from INITIAL to NORMAL at 39.28
- Set status of 20 to NORMAL (purge) at 40.50
- Set status of 2 to INITIAL (purge) at 42.00
- Set status of 3 to NORMAL (purge) at 42.00
- Set status of 4 to NORMAL (purge) at 42.00
- Set status of 7 to INITIAL (purge) at 42.00
- Set status of 9 to NORMAL (purge) at 42.00
- Set status of 12 to NORMAL (purge) at 42.00
- Set status of 14 to INITIAL (purge) at 42.00
- Set status of 15 to NORMAL (purge) at 42.00
- Set status of 16 to NORMAL (purge) at 42.00
- Set status of 18 to NORMAL (purge) at 42.00
- Set status of 19 to NORMAL (purge) at 42.00
- Set status of 21 to INITIAL (purge) at 42.00
- Set status of 22 to INITIAL (purge) at 42.00
- Set status of 23 to NORMAL (purge) at 42.00
- Set status of 24 to NORMAL (purge) at 42.00
- Change status of 20 from NORMAL to GATEWAY at 43.69
- Change status of 2 from INITIAL to NORMAL at 44.04
- Set status of 21 to HEADER at 45.38
- Change status of 22 from INITIAL to NORMAL at 45.38
- Change status of 4 from NORMAL to GATEWAY at 45.38
- Change status of 19 from NORMAL to GATEWAY at 45.38

- Change status of 7 from INITIAL to NORMAL at 45.38
- Change status of 3 from NORMAL to GATEWAY at 45.38
- Change status of 24 from NORMAL to GATEWAY at 45.38
- Change status of 23 from NORMAL to GATEWAY at 45.38
- Set status of 14 to HEADER at 46.20
- Change status of 1 from NORMAL to GATEWAY at 46.21
- Change status of 17 from NORMAL to GATEWAY at 46.21
- Change status of 8 from NORMAL to GATEWAY at 46.21
- Change status of 5 from NORMAL to GATEWAY at 46.21
- Change status of 12 from NORMAL to GATEWAY at 46.21
- Change status of 7 from NORMAL to GATEWAY at 48.69
- Change status of 10 from NORMAL to GATEWAY at 49.28
- Set status of 4 to NORMAL (purge) at 49.50
- Set status of 23 to NORMAL (purge) at 51.00
- Set status of 6 to INITIAL (purge) at 52.50
- Change status of 6 from INITIAL to NORMAL at 54.17
- Set status of 20 to NORMAL (purge) at 58.50
- Change status of 22 from NORMAL to GATEWAY at 58.76
- Change status of 15 from NORMAL to GATEWAY at 63.12
- Change status of 9 from NORMAL to GATEWAY at 64.08
- Set status of 24 to NORMAL (purge) at 64.50
- Set status of 1 to NORMAL (purge) at 69.00
- Change status of 23 from NORMAL to GATEWAY at 70.45
- Change status of 18 from NORMAL to GATEWAY at 73.60
- Change status of 24 from NORMAL to GATEWAY at 76.39
- Change status of 21 from HEADER to INITIAL at 76.88
- Change status of 21 from INITIAL to NORMAL at 78.89
- Set status of 7 to NORMAL (purge) at 79.50

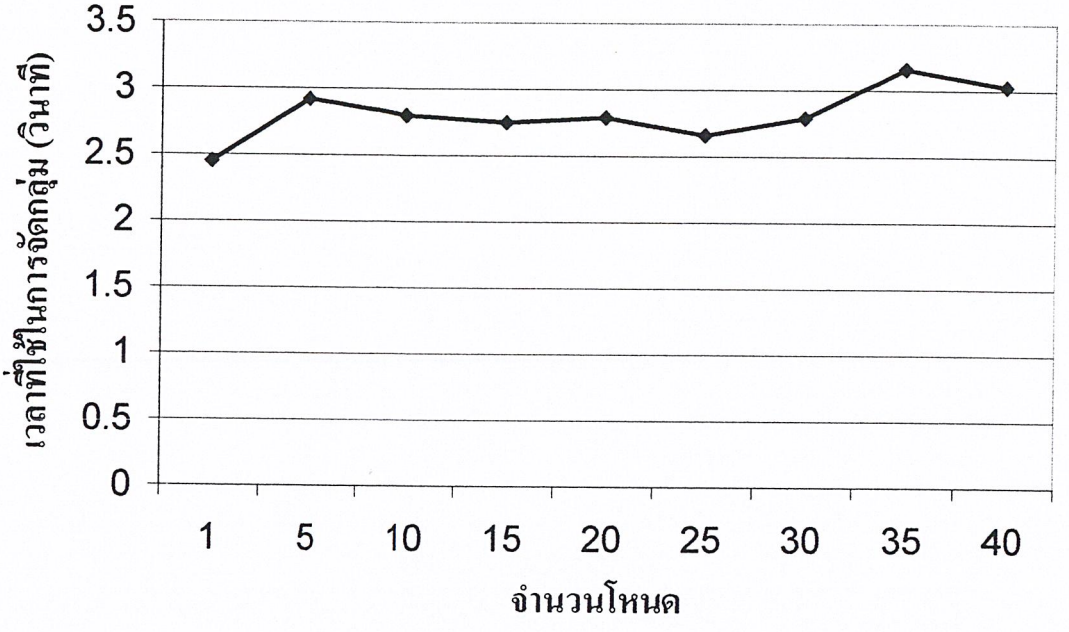
- Set status of 18 to NORMAL (purge) at 79.50
- Set status of 23 to NORMAL (purge) at 79.50
- Set status of 4 to INITIAL (purge) at 81.00
- Change status of 1 from NORMAL to GATEWAY at 81.74
- Change status of 16 from NORMAL to GATEWAY at 81.94
- Set status of 4 to NORMAL (purge) at 82.50
- Set status of 15 to NORMAL (purge) at 82.50
- Change status of 14 from HEADER to INITIAL at 85.31
- Change status of 14 from INITIAL to NORMAL at 87.74
- Chang status of 18 from NORMAL to GATEWAY at 88.88
- Set status of 1 to NORMAL (purge) at 90.00
- Set status of 5 to NORMAL (purge) at 90.00
- Set status of 9 to NORMAL (purge) at 90.00
- Set status of 10 to NORMAL (purge) at 90.00
- Set status of 11 to NORMAL (purge) at 90.00
- Set status of 12 to NORMAL (purge) at 90.00
- Set status of 16 to NORMAL (purge) at 90.00
- Set status of 24 to NORMAL (purge) at 90.00
- Change status of 23 from NORMAL to GATEWAY at 92.15
- Change status of 7 from NORMAL to GATEWAY at 95.10
- Change status of 16 from NORMAL to GATEWAY at 99.56



รูปที่ 6.15 สถานการณ์ที่ 7

6.4 การจำลองเครือข่ายเพื่อใช้วัดเวลาที่ใช้ในการจัดกลุ่ม

กราฟในรูปที่ 6.16 เป็นการแสดงความสัมพันธ์ระหว่างจำนวนโหนดและเวลาที่ใช้ในการจัดกลุ่ม พบว่า เมื่อจำนวนโหนดในเครือข่ายเพิ่มขึ้น แนวโน้มของเวลาที่ใช้ในการจัดกลุ่มจะเพิ่มขึ้นด้วย

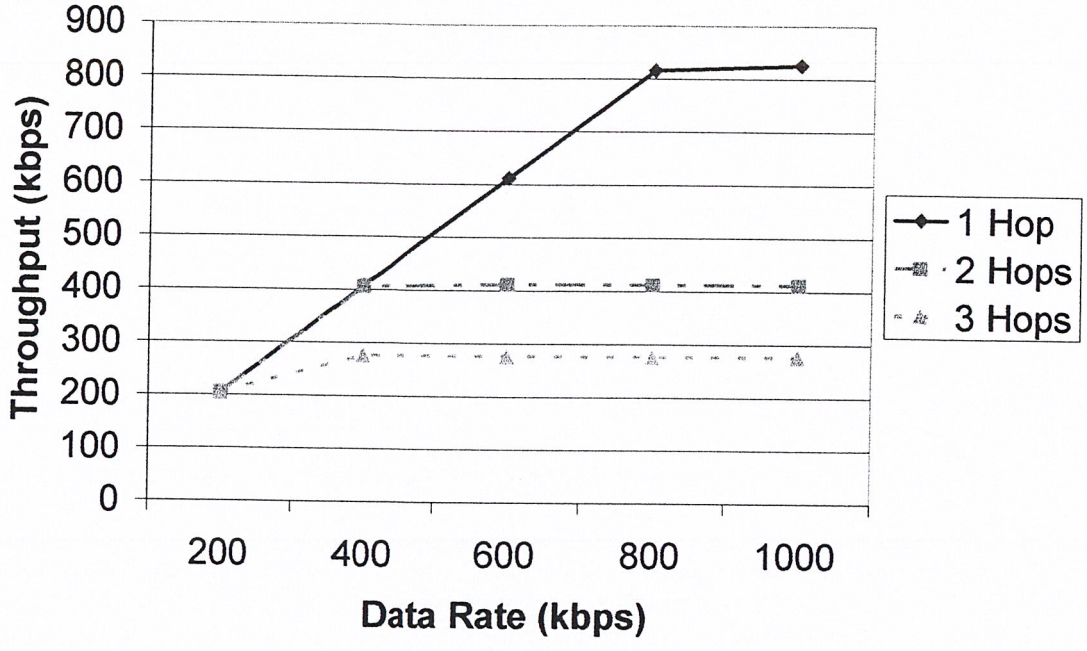


รูปที่ 6.16 กราฟแสดงความสัมพันธ์ระหว่างจำนวนโหนดและเวลาที่ใช้ในการจัดกลุ่ม

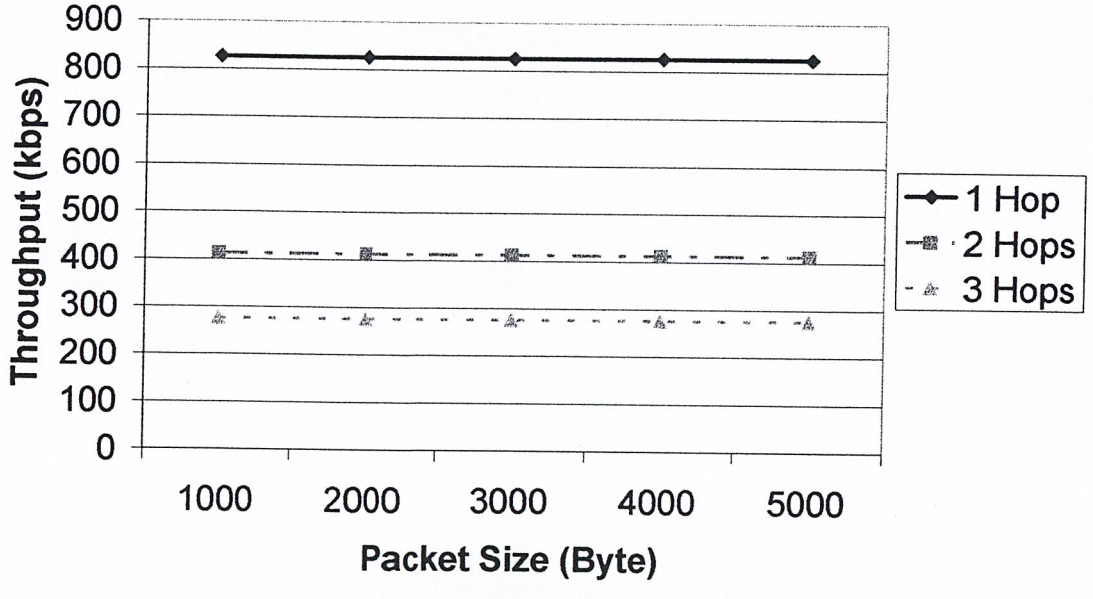
6.5 การจำลองเครือข่ายเพื่อเปรียบเทียบความสัมพันธ์ระหว่างประสิทธิภาพและจำนวนฮอป

กราฟในรูปที่ 6.17 และ 6.18 เป็นกราฟที่เกิดจากการจำลองเหตุการณ์ขึ้นมา เพื่อเปรียบเทียบค่าทรูพุตของระบบ เมื่อกำหนดให้จำนวนฮอปไม่เท่ากัน โดยแบ่งการทดลองออกเป็นสองกรณี

กรณีที่ 1 เมื่อกำหนดให้ขนาดแพ็คเกตที่ส่งมีขนาดคงที่ เท่ากับ 1000 ไบต์ ทราฟฟิกเป็นแบบ CBR (Constant Bit Rate) นั่นคือ มีอัตราการส่งที่มีขนาดแพ็คเกตคงที่ กำหนดให้อัตราการส่งข้อมูลเป็นตัวแปรต้น มีค่าตั้งแต่ 200 kbps ถึง 1000 kbps โดยทำการทดลองของการส่งแบบ 1 ฮอป 2 ฮอป และ 3 ฮอป สังเกตและบันทึกค่าทรูพุตที่ได้ลงกราฟ จะได้เป็นกราฟดังแสดงในรูปที่ 6.17



รูปที่ 6.17 กราฟแสดงความสัมพันธ์ระหว่างอัตราการส่งข้อมูลและค่าทรูพุต เมื่อขนาดแพ็คเกตมีค่าคงที่



รูปที่ 6.18 กราฟแสดงความสัมพันธ์ระหว่างขนาดแพ็คเกตและค่าทรูพุต เมื่ออัตราการส่งข้อมูลมีค่าคงที่

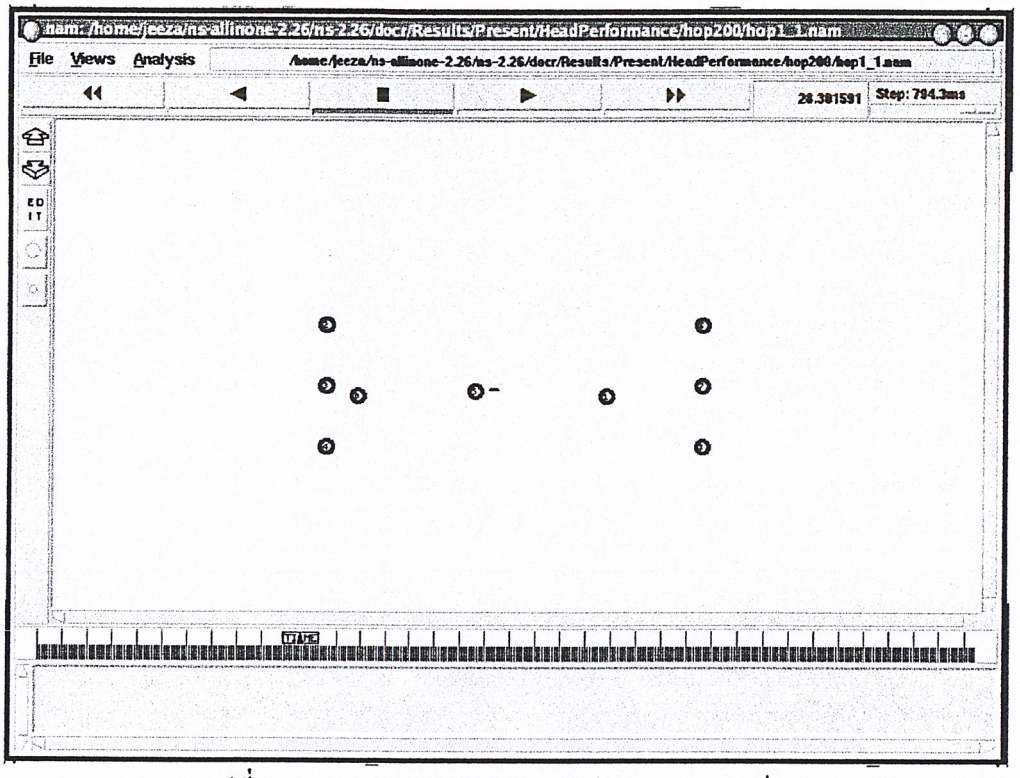
กรณีที่ 2 เมื่อกำหนดให้มีอัตราการส่งข้อมูลคงที่ เท่ากับ 1000 kbps ทราฟฟิกเป็นแบบ CBR (Constant Bit Rate) เช่นเดียวกับกรณีแรก กำหนดให้ขนาดแพ็คเกตเป็นตัวแปรต้น มีค่าตั้งแต่ 1000 ไบต์ ถึง 5000 ไบต์ ทำการทดลองของการส่งแบบ 1 ฮอป 2 ฮอป และ 3 ฮอป สังเกตและบันทึกค่าทรูพุตที่ได้ลงกราฟ จะได้เป็นกราฟดังแสดงในรูปที่ 6.18

6.6 การจำลองเครือข่ายเพื่อวัดประสิทธิภาพของโหนดคลัสเตอร์เฮด

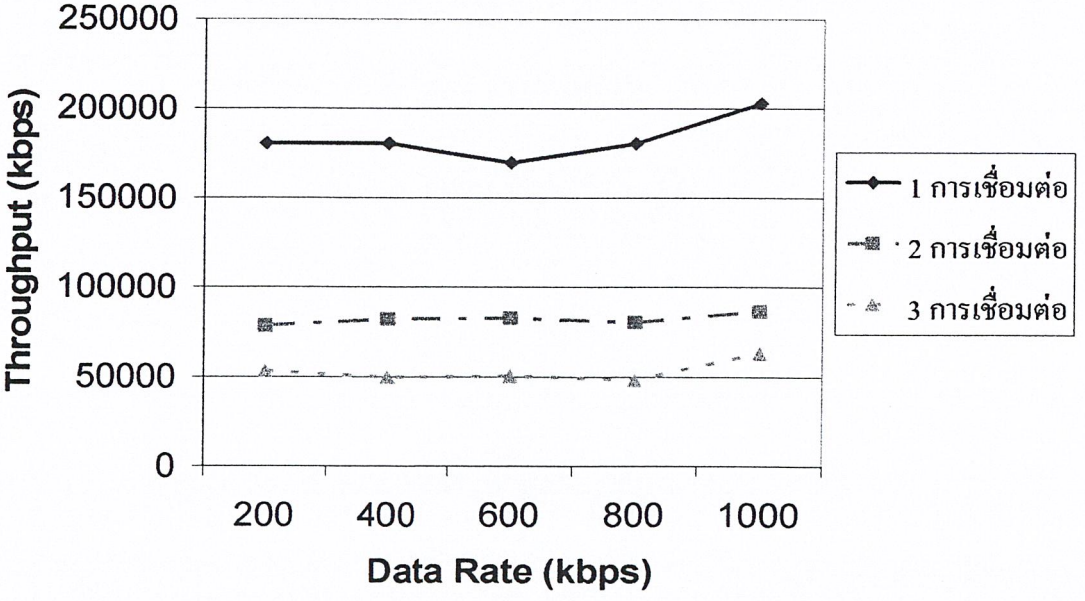
โหนดคลัสเตอร์เฮดเป็นโหนดที่มีอิทธิพลต่อการจัดกลุ่มในเครือข่าย เนื่องจากเป็นโหนดที่ทำหน้าที่ในการจัดสรรเส้นทาง และควบคุมการรับส่งข้อมูลให้กับโหนดสมาชิกที่อยู่ในคลัสเตอร์ที่ตนดูแลอยู่

การจำลองเครือข่ายเพื่อวัดประสิทธิภาพของโหนดคลัสเตอร์เฮด ทำได้โดยการแยกพิจารณาจากจำนวนการเชื่อมต่อของโหนดสมาชิกกับโหนดคลัสเตอร์เฮด ซึ่งในการจำลองในโครงงานนี้ ได้แยกพิจารณาเป็น หนึ่งการเชื่อมต่อ สองการเชื่อมต่อ และสามการเชื่อมต่อ แล้ววัดค่าทรูพุตที่ได้มาพิกัดจุดลงบนกราฟ โดยมีแกนนอนเป็นอัตราการส่งข้อมูล มีหน่วยเป็นเมตรต่อวินาที และแกนตั้งเป็นค่าทรูพุต มีหน่วยเป็น kbps

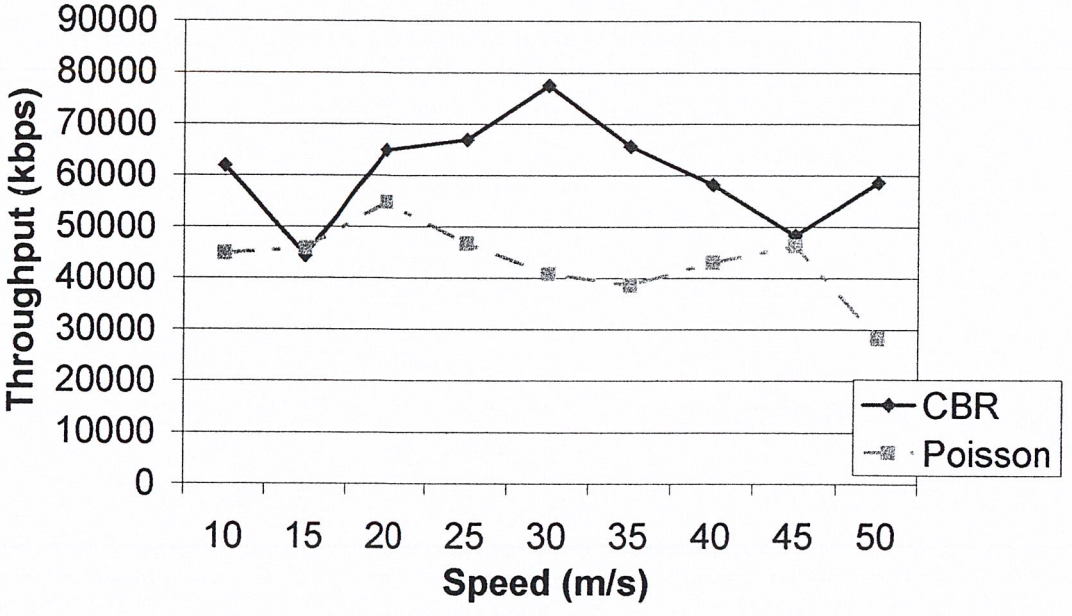
จากกราฟที่ได้ ดังรูปที่ 6.20 พบว่า จำนวนการเชื่อมต่อของโหนดสมาชิกกับโหนดคลัสเตอร์เฮดที่เท่ากับ 1 การเชื่อมต่อ ให้ค่าทรูพุตที่ดีที่สุด เพราะจำนวนการเชื่อมต่อที่มากขึ้นจะทำให้เกิดสภาวะคอขวดที่โหนดคลัสเตอร์เฮด ซึ่งโหนดคลัสเตอร์เฮดจะทำการดรอปข้อมูลทิ้ง เนื่องจากไม่สามารถส่งข้อมูลออกไปได้ทันกับข้อมูลที่รับเข้ามา



รูปที่ 6.19 การจำลองการทำงานของโหนดคลัสเตอร์เฮด



รูปที่ 6.20 กราฟเปรียบเทียบประสิทธิภาพของโหนดคลัสเตอร์เฮด



รูปที่ 6.21 กราฟแสดงความสัมพันธ์ระหว่างความเร็วของโหนดกับประสิทธิภาพของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR

6.7 การจำลองเครือข่ายเพื่อเปรียบเทียบประสิทธิภาพของทราฟฟิกแบบ CBR และตามกระบวนการปัวซอง (Poisson Process)

ทำการจำลองเครือข่ายภายใต้เงื่อนไข กำหนดให้มีจำนวนโหนดเท่ากับ 20 โหนด แต่ละโหนดมีความเร็วที่เท่ากัน ในแต่ละครั้งของการจำลอง โดยกำหนดให้มีทราฟฟิกทั้งแบบ CBR และตามกระบวนการปัวซอง แล้วนำค่าที่ได้จากการจำลองเครือข่ายมาวาดเป็นกราฟแสดงความสัมพันธ์ระหว่างความเร็วของโหนดกับค่าทรูพุต

เมื่อเปรียบเทียบค่าที่ได้จากทราฟฟิกทั้งแบบ CBR และตามกระบวนการปัวซองแล้ว พบว่าค่าทรูพุตที่ได้จากทราฟฟิกแบบ CBR มีค่าสูงกว่า เนื่องจากทราฟฟิกแบบ CBR มีการส่งข้อมูลด้วยอัตราคงที่ตลอดเวลา ในขณะที่ทราฟฟิกตามกระบวนการปัวซองเป็นการส่งข้อมูลแบบสุ่ม ดังนั้นอัตราการส่งจึงไม่คงที่ อาจไม่มีการส่งในบางช่วง ค่าทรูพุตที่ได้จึงมีค่าต่ำกว่าทราฟฟิกแบบ CBR กราฟแสดงในรูปที่ 6.21

6.8 การจำลองเพื่อวัดประสิทธิภาพโดยรวมของทราฟฟิกแบบ CBR

ผลของการจำลองสถานการณ์ เมื่อมีโหนดจำนวน 20 โหนดในเครือข่ายหนึ่งๆ ซึ่งมีทราฟฟิกเป็นแบบ CBR แล้วต้องการวัดค่าทรูพุตเฉลี่ยที่เกิดจากโหนดต่างๆในเครือข่ายที่มีความเร็วสูงสุดต่างๆกัน แล้วเปรียบเทียบกันระหว่างโปรโตคอลแบบต่างๆ ได้แก่ แบบ DOCR (Developed On-demand Cluster-based Routing, ซึ่งเป็นรูปแบบที่ถูกพัฒนาขึ้นในโปรโตคอลนี้) แบบ AODV และแบบ DSDV สามารถเขียนเป็นชุดคำสั่งที่ใช้กำหนดลักษณะการเคลื่อนที่ และการกำหนดรูปแบบทราฟฟิกบนเครือข่ายได้ดังนี้

ชุดคำสั่งที่ใช้กำหนดลักษณะการเคลื่อนที่ของโหนดต่างๆ

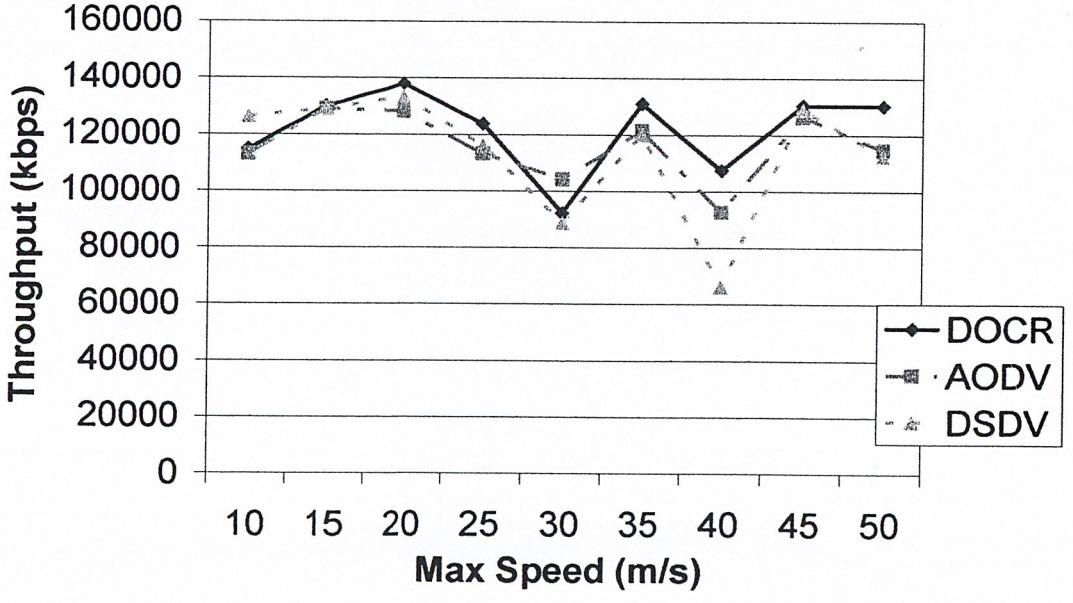
-n 20 –p 10 –s 10 –t 100 -x 500 –y 500

โดยค่าพารามิเตอร์ –s มีค่าเป็น 10, 20, 30, 40 และ 50 เมตรต่อวินาทีตามลำดับ

ชุดคำสั่งที่ใช้ในการกำหนดรูปแบบทราฟฟิกบนเครือข่าย

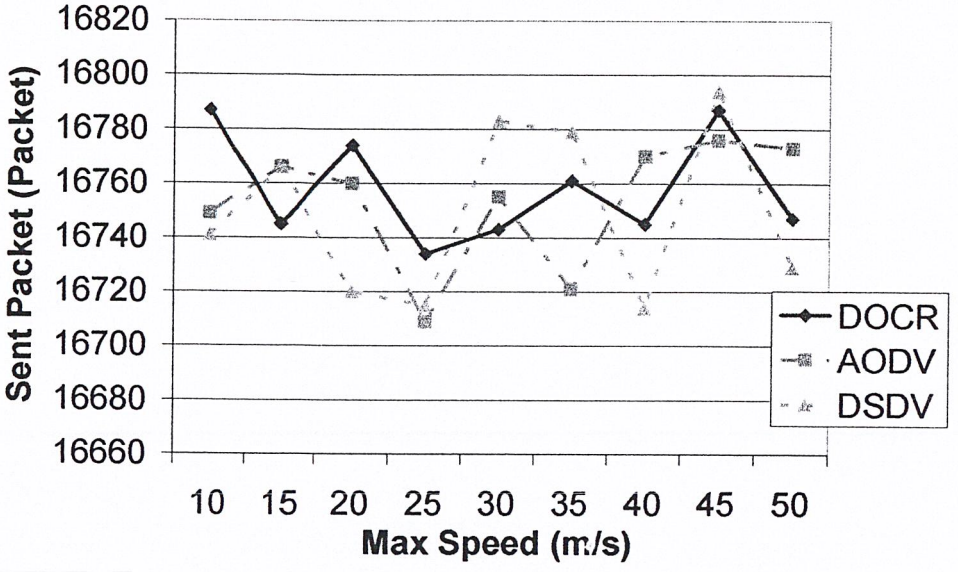
-type cbr –nn 20 –seed 0.1 –mc 15 –rate 50

นำผลที่ได้มาพิกัดจุดลงบนกราฟที่มีค่าในแกนนอนเป็นความเร็วสูงสุดของโหนด มีหน่วยเป็นเมตรต่อวินาที และค่าในแกนตั้งเป็นค่าทรูพุต มีหน่วยเป็น kbps

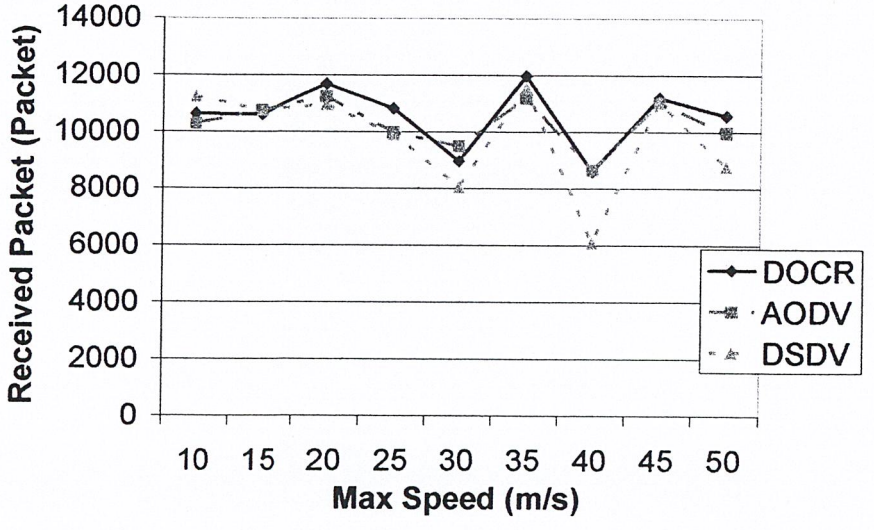


รูปที่ 6.22 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับประสิทธิภาพของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR

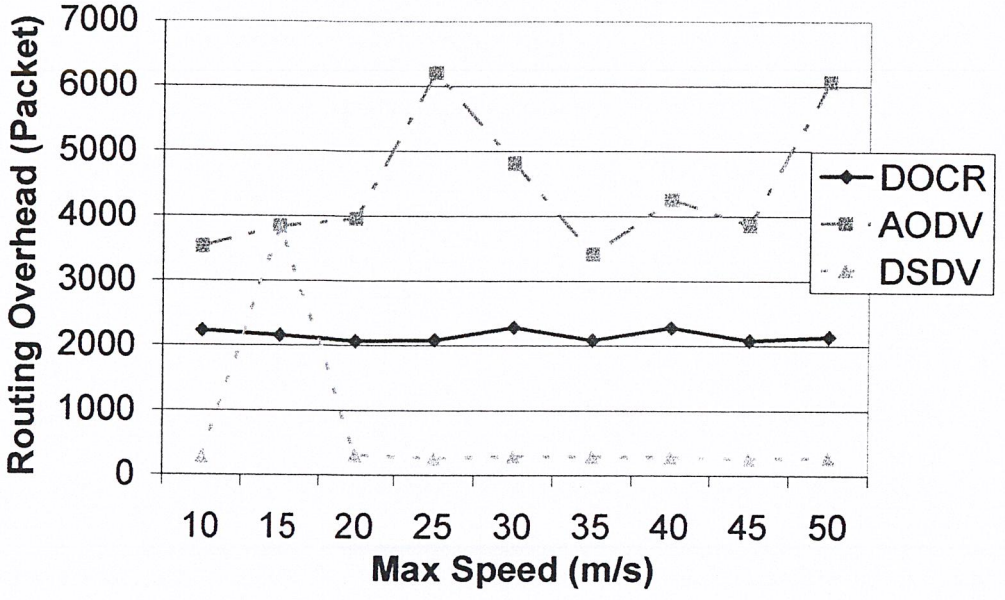
นอกจากนี้ยังพิจารณาถึงประสิทธิภาพในการรับส่งข้อมูลที่ความเร็วสูงสุดของโหนดต่างๆ กัน ซึ่งในการทดลองนี้ยังคงกำหนดให้มีความเร็วเป็น10, 20, 30, 40 และ 50 เมตรต่อวินาทีตามลำดับ



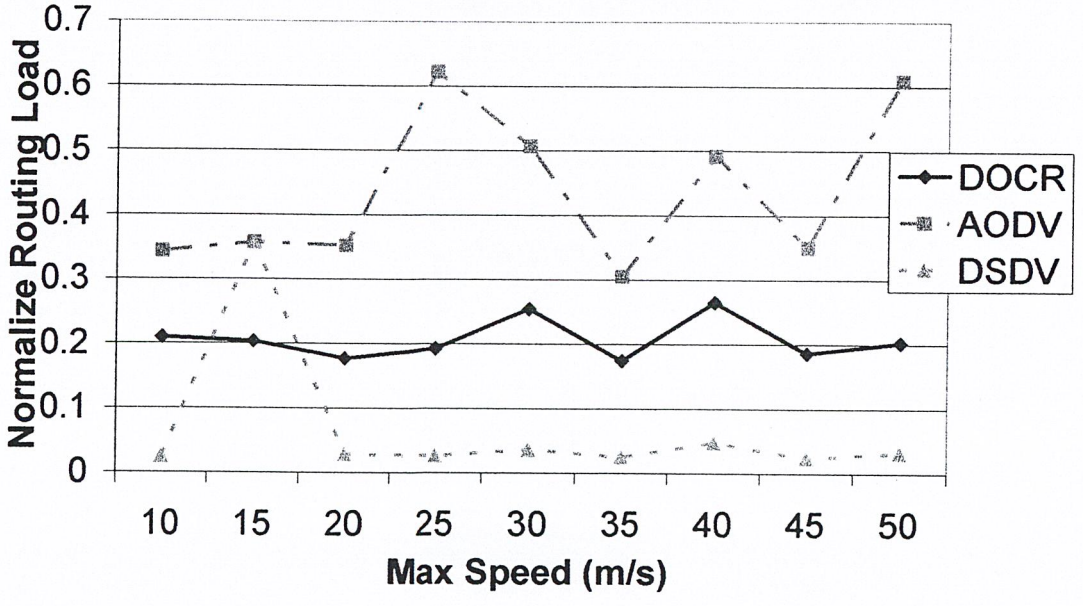
รูปที่ 6.23 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกตที่ส่งของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR



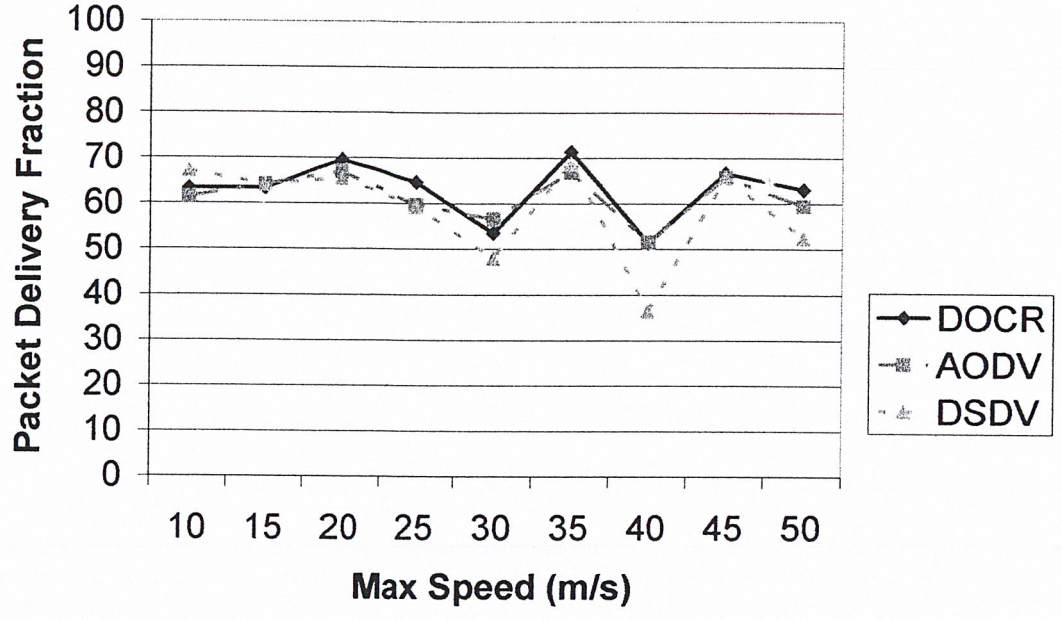
รูปที่ 6.24 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกตที่รับของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR



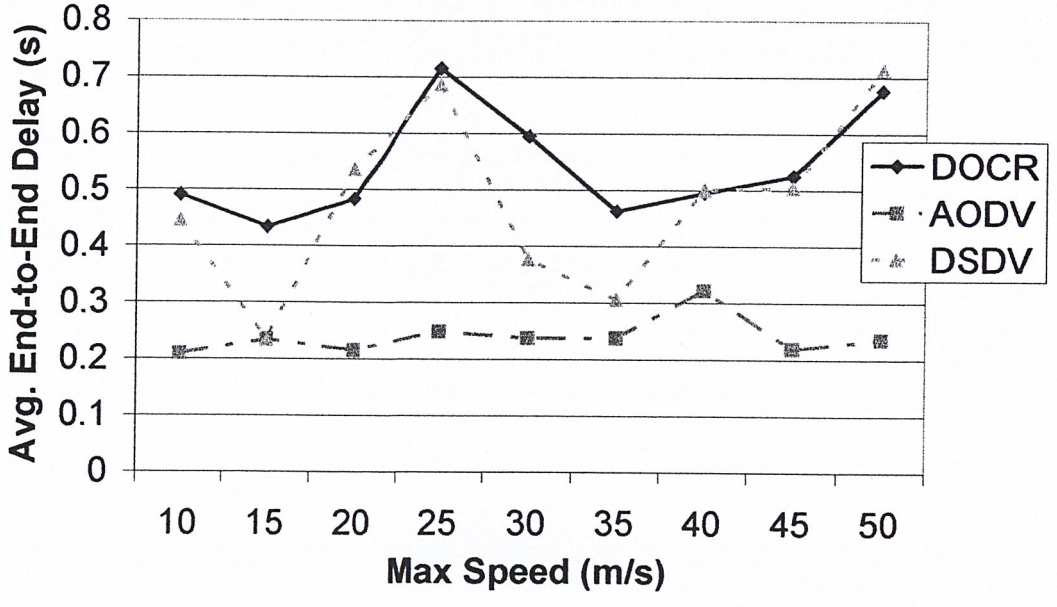
รูปที่ 6.25 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Routing Overhead ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR



รูปที่ 6.26 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Normalize Routing Load ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR



รูปที่ 6.27 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Packet Delivery Fraction ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR



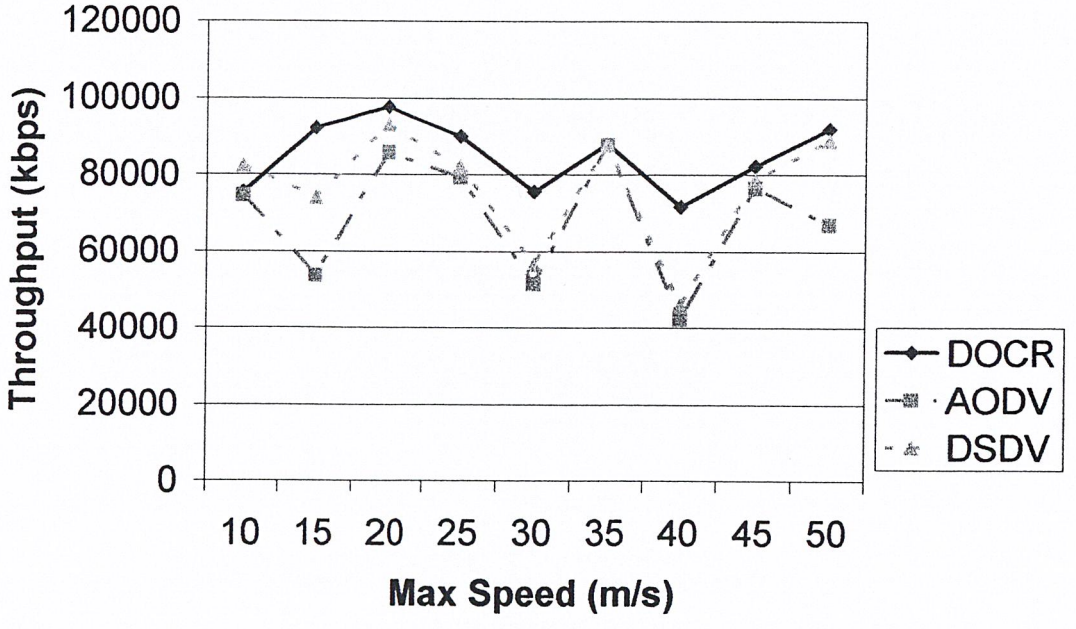
รูปที่ 6.28 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับเวลาดีเลย์เฉลี่ยระหว่างต้นทางและปลายทางของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบ CBR

6.9 การจำลองเพื่อวัดประสิทธิภาพโดยรวมของทราฟฟิกที่เป็นไปตามกระบวนการปัวซอง

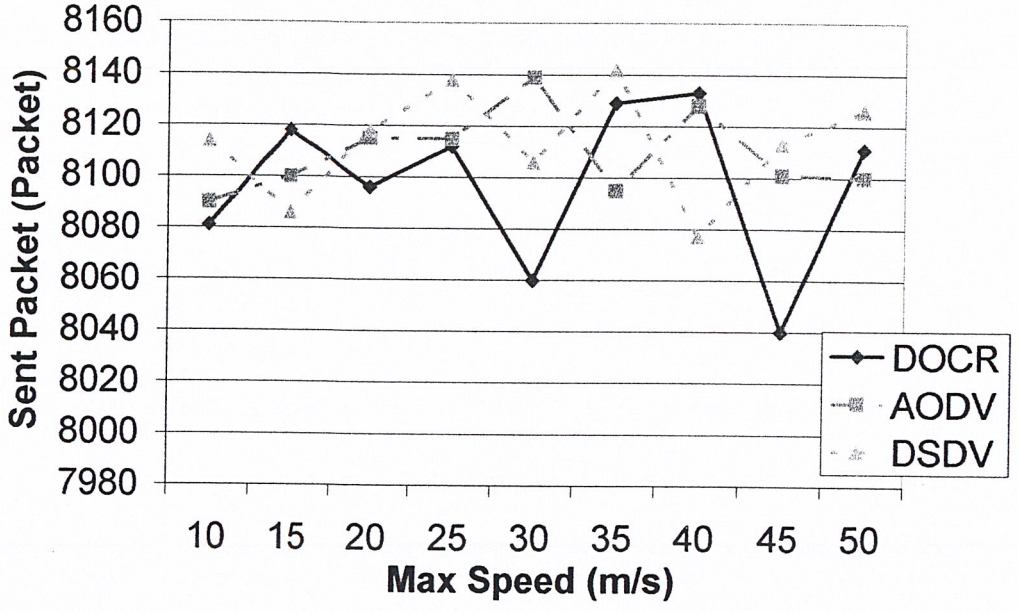
นอกจากทราฟฟิกบนเครือข่ายแบบ CBR แล้วยังได้ทำการจำลองทราฟฟิกบนเครือข่ายที่เป็นไปตามกระบวนการปัวซอง โดยยังคงกำหนดให้มีโหนดในเครือข่ายจำนวน 20 โหนด ซึ่งมีทราฟฟิกที่เป็นไปตามกระบวนการปัวซอง วัดค่าทรูพุตเฉลี่ยที่เกิดจากโหนดต่างๆในเครือข่ายที่มีความเร็วสูงสุดต่าง ๆ กัน แล้วเปรียบเทียบกันระหว่างโปรโตคอลแบบต่าง ๆ ได้แก่ แบบ DOCR แบบ AODV และแบบ DSDV

นำผลที่ได้มาพิกัดจุดลงบนกราฟที่มีค่าในแกนนอนเป็นความเร็วสูงสุดของโหนด มีหน่วยเป็นเมตรต่อวินาที และค่าในแกนตั้งเป็นค่าทรูพุต มีหน่วยเป็น kbps ดังแสดงในรูปที่ 6.29

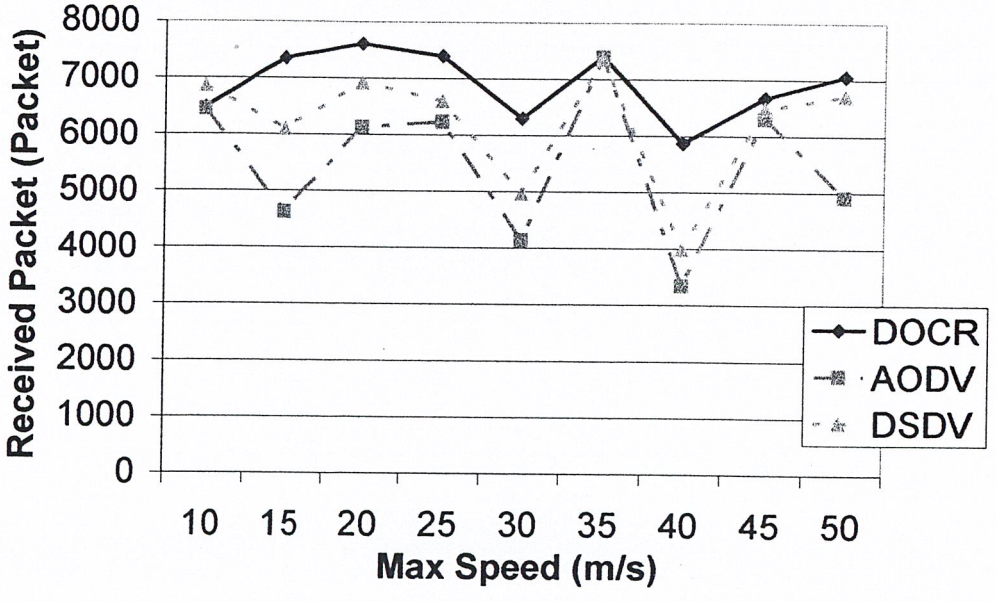
นอกจากนี้ยังพิจารณาถึงประสิทธิภาพในการรับส่งข้อมูลที่ความเร็วสูงสุดของโหนดต่างๆ กัน ซึ่งในการทดลองนี้ยังคงกำหนดให้มีความเร็วเป็น10, 20, 30, 40 และ 50 เมตรต่อวินาทีตามลำดับ แสดงเป็นกราฟดังแสดงในรูปที่ 6.30 รูปที่ 6.31 รูปที่ 6.32 รูปที่ 6.33 รูปที่ 6.34 และรูปที่ 6.35



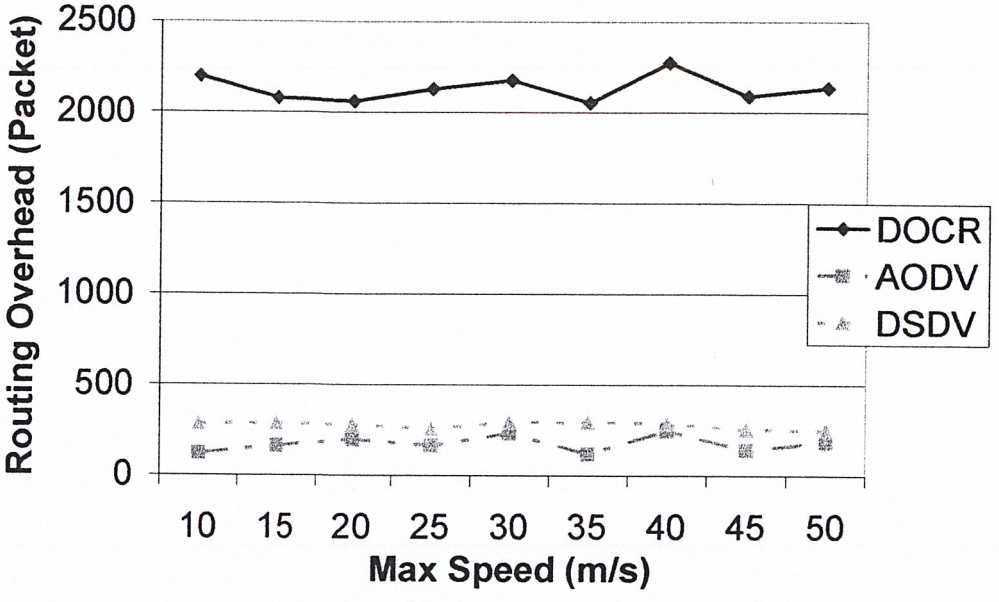
รูปที่ 6.29 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับประสิทธิภาพของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง



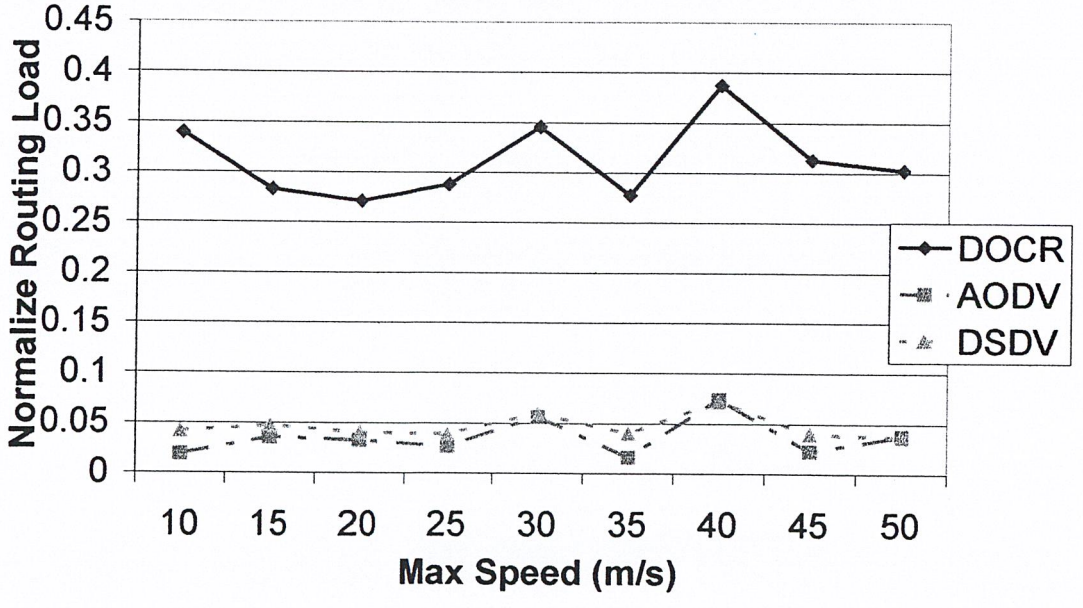
รูปที่ 6.30 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกตที่ส่งของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบบัวซอง



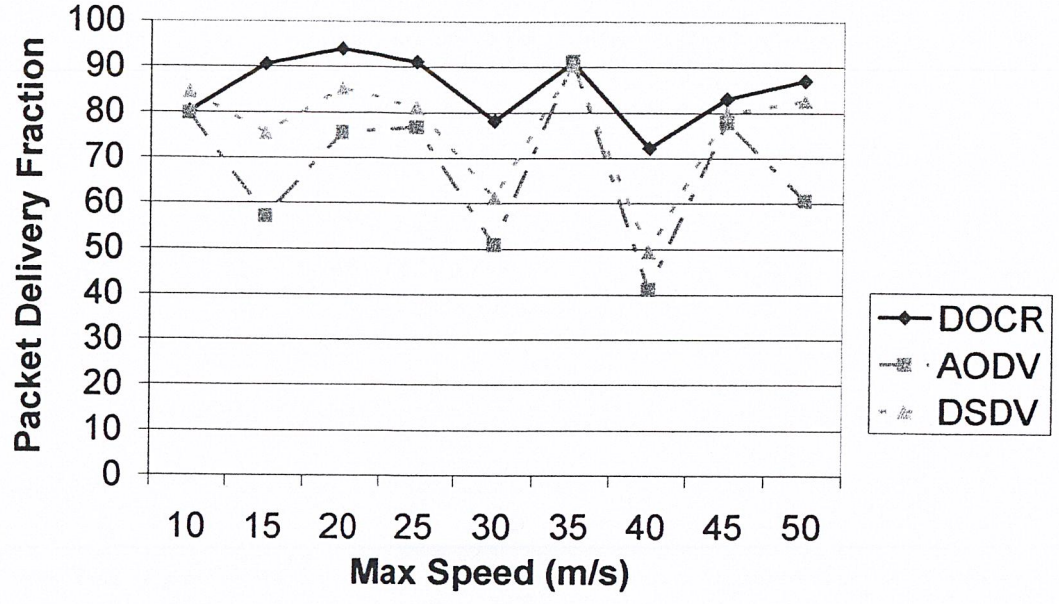
รูปที่ 6.31 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับปริมาณแพ็คเกตที่รับของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบบัวซอง



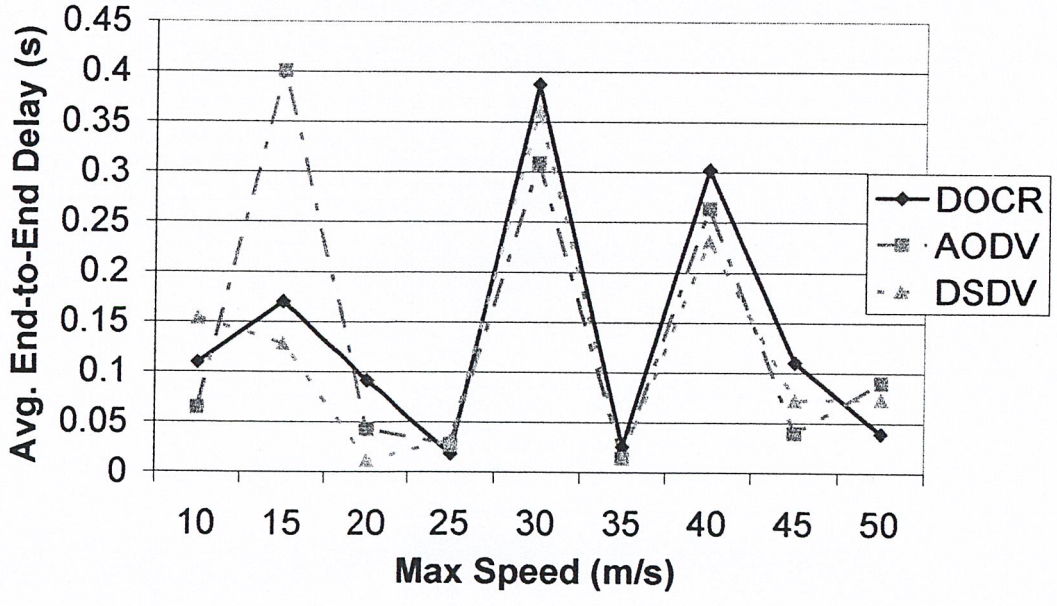
รูปที่ 6.32 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Routing Overhead ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง



รูปที่ 6.33 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Normalize Routing Load ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง



รูปที่ 6.34 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับ Packet Delivery Fraction ของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง



รูปที่ 6.35 กราฟแสดงความสัมพันธ์ระหว่างความเร็วสูงสุดของโหนดกับเวลาดีเลย์เฉลี่ยระหว่างต้นทางและปลายทางของโปรโตคอลต่างๆ เมื่อมีทราฟฟิกแบบปัวซอง

บทที่ 7

สรุปผลการทดลอง

5.1 สรุปผลการทดลอง

ระบบการให้บริการเคลื่อนที่แบบไร้สายในโครงข่ายโมบายที่ถูกพัฒนาโดยการจำลองการทำงานด้วยคอมพิวเตอร์ ในโครงงานนี้ มีความสามารถในการค้นหาเส้นทาง และส่งผ่านข้อมูลไปบนเครือข่ายแอดฮอคได้ โดยยังคงทำงานได้อย่างมีประสิทธิภาพ เมื่อแต่ละโหนดแคลื่อนที่ด้วยความเร็วสูง ซึ่งวัดได้จากค่าพารามิเตอร์ต่างๆ เช่น ค่าทรูพุต และค่าเวลาดีเลย์ เป็นต้น แต่ยังคงมีปัจจัยบางอย่างที่ทำให้ประสิทธิภาพลดลงบ้าง ได้แก่

- การส่งผ่านโหนดจำนวนมาก ซึ่งทำให้เกิดจำนวนฮอปที่มากขึ้นตามไปด้วย
- จำนวนโหนดที่มีการเชื่อมต่อกับโหนดคลัสเตอร์เฮดที่มากขึ้น ทำให้เกิดภาวะคอขวดที่โหนดคลัสเตอร์เฮด

5.2 ปัญหาที่เกิดขึ้นระหว่างทดลอง

- Ns2 เป็นแอพพลิเคชันที่เป็น Freeware ทำให้มีคู่มือการใช้งานไม่เพียงพอ และไม่ทันสมัย
- เครือข่ายแอดฮอคเป็นเครือข่ายใหม่ที่เพิ่งเกิดขึ้นมาได้ไม่นาน ทำให้หาข้อมูลได้ยาก มีหนังสือ และผู้ที่เข้าใจให้คำแนะนำได้น้อย ทำให้ต้องใช้เวลานานในการค้นคว้าข้อมูล
- ภาษา OTcl ซึ่งเป็นภาษาที่ใช้ในการพัฒนา เป็นภาษาที่ไม่ได้รับความนิยม และไม่เคยศึกษามาก่อน ทำให้ต้องใช้เวลานานในการศึกษาภาษานี้
- เครื่องที่ใช้ในการประมวลผลมีประสิทธิภาพไม่ดีพอ จึงต้องใช้เวลาในการรอผลพอสมควร
- โมดูลบางโมดูลของ Ns2 ยังคงมีความผิดพลาดอยู่ ซึ่งเป็นความผิดพลาดที่เกิดจากทางผู้พัฒนา Ns2 เอง

5.3 แนวทางการพัฒนาโครงงาน

- โครงงานนี้ทำการพัฒนาเฉพาะในส่วนของเน็ตเวิร์คเลเยอร์ ซึ่งหากมีความต้องการที่จะพัฒนาให้มีประสิทธิภาพมากขึ้น สามารถที่จะพัฒนาในส่วนของเลเยอร์ชั้นที่ต่ำลงมากได้ เช่น การจัดสรรช่องสัญญาณของโหนดที่เป็นคลัสเตอร์เฮดให้มีประสิทธิภาพดีขึ้น

- เนื่องจากโครงงานนี้เป็นการจำลองสถานการณ์ขึ้น ดังนั้นเมื่อพัฒนาไประดับหนึ่งแล้ว สามารถที่จะนำไปใช้งานได้จริงบนระบบปฏิบัติการลินุกซ์ได้
- อาจมีการพัฒนาคุณสมบัติในด้านอื่นๆเพิ่มขึ้นอีก เช่น ความปลอดภัยในการสื่อสาร เป็นต้น

บรรณานุกรม

1. M.Chamchoy, F. Kojima, H. Harada, "Stability Evaluation of Terminal Group for Inter-Vehicle Communication Network with an Autonomous Relay Access Scheme"

2. F. Kojima, H. Harada, M. Fujise, "System Configuration for Private Mobile Network using Slotted-Aloha Based Autonomous Packet Routing"

3. C.-K.Toh, "Ad Hoc Mobile Wireless Networks: Protocols and Systems", Printice Hall PTR, 2002

4. A.S. Tanenbaum, "Computer Networks", 4th ed., New Jersey: Prentice Hall, 2003

5. W. Richard Stevens, "TCP/IP Ilustrated Volume 1: The Protocols", Wesley

6. W. Richard Stevens, "TCP/IP Ilustrated Volume 2: The Implementation", Wesley

7. "http://www.isi.edu" : Network Simulator 2 Document