อัลกอริทึมเพื่อการกระจายงานสำหรับการประชุมด้วยภาพ

LOAD BALANCING ALGORITHM FOR MULTIPOINT VIDEO CONFERENCING CONTROL UNIT

จิรายุ เจริญวิริยะภาพ
JIRAYU CHAROENVIRIYAPHAP

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2550

อัลกอริทึมเพื่อการกระจายงานสำหรับการประชุมด้วยภาพ

LOAD BALANCING ALGORITHM FOR MULTIPOINT VIDEO CONFERENCING CONTROL UNIT

จิรายุ เจริญวิริยะภาพ

JIRAYU CHAROENVIRIYAPHAP

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2550

LOAD BALANCING ALGORITHM FOR MULTIPOINT VIDEO CONFERENCING CONTROL UNIT

JIRAYU CHAROENVIRIYAPHAP

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2007

หัวข้อวิทยานิพนธ์	อัลกอริทึมเพื่อการกระจายงานสำหรับการประชุมด้วยภาพ
นักศึกษา	นายจิรายุ เจริญวิริยะภาพ
รหัสนักศึกษา	47060806
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2550
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ.ดร.สุรินทร์ กิตติธรกุล

บทคัดย่อ

ปัจจุบันการการประชุมด้วยภาพผ่านทางอินเตอร์เน็ตได้รับความนิยมมากขึ้นเนื่องจากคุณภาพของอินเตอร์เน็ตมีการพัฒนาไปอย่างรวดเร็วและการติดต่อสื่อสารมีความจำเป็นมาก เนื่องจากมีผู้ใช้จำนวนมากขึ้นเซิร์ฟเวอร์ที่ให้บริการจึงมีภาระงานที่หนักมากจากการ เข้ารหัส, ถอดรหัส, ผสมวีดีโอและจัดการการบริหารห้องที่ใช้ในการติดต่อ ซึ่งงานเหล่านี้จะเพิ่มขึ้นเมื่อจำนวนผู้ใช้มากขึ้น จึงมีการเพิ่มเซิร์ฟเวอร์ให้บริการเข้ามาจากเดิมที่เป็นแบบรวมเข้าศูนย์กลาง (Centralize) เป็นแบบกระจาย(Distribute) เมื่อมีการกระจายเซิร์ฟเวอร์ออกก็เกิดปัญหาที่ว่างานในแต่ละเซิร์ฟเวอร์ไม่เท่ากัน จึงได้มีการเพิ่มในส่วนของการจัดการกระจายงานให้แก่เซิร์ฟเวอร์ โดยสิ่งสำคัญของส่วนนี้คืออัลกอริทึมในการกระจายงาน เพื่อให้ระบบมีประสิทธิภาพมากที่สุด เซิร์ฟเวอร์ทุกตัวมีการใช้งานอย่างคุ้มค่า และสามารถตอบสนองความต้องการของผู้ใช้ได้ โดยส่วนสำคัญของอัลกอริทึมเรานั้นจะเป็นการนำเอาเปอร์เซ็นต์การใช้งานของซีพียู,หน่วยความจำ และทรัพยากรทางเครือข่าย

Thesis Title	LOAD BALANCING ALGORITHM FOR MULTIPOINT VIDEO CONFERENCING CONTROL UNIT
Student	Mr. Jirayu Charoenviriyaphap
Student ID.	47060806
Degree	Master of Engineering
Program	Computer Engineering
Year	2007
Thesis Advisor	Asst.Prof.Dr. Surin Kittitornkun

ABSTRACT

Today videoconferencing is more popular because the Internet is more widespread and faster than it was in the past few years. Multipoint videoconferencing requires a multipoint control unit (MCU) to mix video/audio and manage the floor. MCU load increases as much as user's growth worldwide. Therefore, distributed MCUs can balance the load from all users. However, we need to equally balance the MCUs by introducing a load-balancer. The contribution of this paper is the balancing algorithm taking into account of %CPU used, %memory used and %bandwidth used.

กิตติกรรมประกาศ

คุณความดีอันใดที่บังเกิดจากวิทยานิพนธ์ฉบับนี้ ขอมอบแด่บิดา,มารดาและพี่สาวของผู้วิจัย ผู้ที่คอยห่วงใย เข้าใจและให้การสนับสนุนในการศึกษามาโดยตลอด

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงลงได้ด้วยดี โดยได้รับความกรุณาจาก ผศ.ดร สุรินทร์ กิตติธรกุล อาจารย์ผู้ควบคุมวิทยานิพนธ์ที่ได้ช่วยเหลือในการให้คำแนะนำ ความรู้ทางทฤษฎีและการทดลองต่างๆที่ใช้ และชี้แนะแนวทางในการแก้ปัญหาต่างๆอย่างทุ่มเทรวมทั้งฝึกฝนผู้วิจัยให้มีความสามารถในการทำวิจัยและพัฒนาได้อย่างมีประสิทธิภาพ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณกรรมการสอบวิทยานิพนธ์ทุกท่านที่ได้กรุณาให้คำแนะนำในทุกๆเรื่อง ทั้งวิธีแก้ปัญหาที่เกิดขึ้นในการทำวิทยานิพนธ์และมุมมองในเชิงวิศวกรรมอื่นๆซึ่งช่วยให้ผู้วิจัยมีวิสัยทัศน์ที่กว้างไกลขึ้น

ขอขอบคุณบัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ให้การสนับสนุนการทำวิทยานิพนธ์นี้

ขอบคุณพี่ๆเพื่อนๆและน้องๆนักศึกษาทุกคนในห้องวิจัย รวมทั้งเพื่อนๆหลายๆคนในอินเตอร์เน็ตที่ช่วยเหลือให้คำแนะนำต่างๆและให้กำลังใจแก่ผู้วิจัยตลอดมา

จิรายุ เจริญวิริยะภาพ

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

หน้า

บทที่1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันการการประชุมด้วยภาพผ่านทางอินเตอร์เน็ตได้รับความนิยมมากขึ้นเนื่องจากคุณภาพของอินเตอร์เน็ตมีการพัฒนาไปอย่างรวดเร็วและการติดต่อสื่อสารมีความจำเป็นมาก เนื่องจากมีผู้ใช้มากขึ้นเซิร์ฟเวอร์ที่ให้บริการจึงมีภาระงานที่หนักมากจากการ Encode, Decode, Mix วีดีโอและจัดการการบริหารห้องที่ใช้ในการติดต่อ ซึ่งงานเหล่านี้จะเพิ่มขึ้นเมื่อจำนวนผู้ใช้มากขึ้น ทำให้ภาระงานที่เซิร์ฟเวอร์มีมากขึ้น โดยมี Multipoint Control Unit (MCU) ทำหน้าที่เป็นมัลติมีเดียเซิร์ฟเวอร์ในการสนับสนุนการประชุมแบบหลายจุด (Multipoint Conference) ระหว่างเทอร์มินัล 3 เทอร์มินัลขึ้นไป และจัดการดูแลห้องในการประชุม การจัดการด้านการติดต่อ การ Encode และ Decode วีดีโอจากผู้ใช้ และการ Mix วีดีโอและทำการส่งให้แก่ผู้ใช้ จัดการการติดต่อระหว่างผู้ใช้ ทำให้เซิร์ฟเวอร์รับภาระมากขึ้น มีความจำเป็นต้องเพิ่มประสิทธิภาพของเซิร์ฟเวอร์และเพิ่มจำนวนเซิร์ฟเวอร์ให้มากขึ้นเพื่อให้สามารถรองรับการทำงานที่มีขนาดใหญ่ขึ้นได้ โดยการเปลี่ยนจากระบบ Centralization เป็น Distribution

เมื่อมีการกระจายงานของเซิร์ฟเวอร์ไปยังเซิร์ฟเวอร์หลายๆตัว ก็จะเกิดปัญหาในการกระจายงานที่มีประสิทธิภาพไม่เท่าที่ควร จึงมีการนำส่วนที่ทำหน้าที่ในการจัดการการกระจายงานเข้ามาแก้ปัญหาโดยองค์ประกอบสำคัญของการทำงานจะอยู่ที่อัลกอริทึม เพื่อให้ระบบที่เราสร้างขึ้นมา ใช้งานได้อย่างคุ้มค่า และสามารถตอบสนองความต้องการของผู้ใช้ได้

อัลกอริทึมในการกระจายงานในปัจจุบันมีหลายวิธี แต่ละวิธีก็จะเหมาะสมกับงานที่ต่างกัน ส่วนใหญ่เมื่อเรามองถึงภาระงานที่เกิดขึ้นกับคอมพิวเตอร์ 1 เครื่องเราจะมองกันที่ การทำงานของ CPU และ ปริมาณหน่วยความจำที่ใช้ แต่สำหรับ Multipoint Control Unit (MCU) แล้วนอกจากปัญหาในเรื่องของการประมวลผลแล้วนั้นเรายังต้องคำนึงถึงภาระงานในด้านเครือข่ายอีกด้วย เพราะ MCU จะต้องทำหน้าที่เป็นตัวกลางในการติดต่อสื่อสารกับเทอร์มินอลพร้อมๆกันหลายเครื่อง ทำให้การตอบสนองช้าลงโดยที่ไม่ได้เกิดมาจากการประมวลผลเพียงอย่างเดียว หากแต่เป็นเพราะความล่าช้าของเครือข่าย ทำให้เราสนใจที่จะนำเอาตัวแปรเหล่านี้เข้ามาใช้ในการพิจารณาว่าเมื่อมีเทอร์มินอลใหม่เข้ามาสมควรที่จะมอบให้ MCU ตัวใดเป็นตัวจัดการ และส่วนสำคัญที่สุดในการกระจายงานนั่นคืออัลกอริทึมในการตัดสินใจนั่นเอง

1.2 วัตถุประสงค์ของการศึกษา

1. ศึกษาอัลกอริทึมการกระจายงาน
2. ศึกษาองค์ประกอบที่มีผลกระทบต่อประสิทธิภาพการทำงานของ Multipoint Control Unit (MCU)
3. นำเสนออัลกอริทึมการกระจายงานสำหรับ Multipoint Control Unit (MCU)
4. นำอัลกอริทึมการกระจายงานที่คิดขึ้นมาทดลองใช้งานและดูการกระจายงานที่ได้
5. เปรียบเทียบอัลกอริทึมของเรากับอัลกอริทึมอื่นๆ

1.3 สมมุติฐานของการศึกษา

เพื่อให้บรรลุวัตถุประสงค์ของการศึกษา เราได้นำเสนอหลักการตามทฤษฎีที่เกี่ยวข้องเพื่อทำการกระจายงานภายในระบบการประชุมด้วยภาพ โดยหลักการที่นำเสนอก็คือ การนำเอาองค์ประกอบที่มีผลต่อการทำงานของ MCU เข้ามาเป็นตัวแปรที่ใช้ในอัลกอริทึมการกระจายงาน ได้แก่ การทำงานของซีพียู, หน่วยความจำที่เหลืออยู่, แบนด์วิดท์ที่เหลืออยู่ และ ค่าดีเลย์ระหว่าง MCU และ เทอร์มินอล ซึ่งตามทฤษฎีแล้ว เมื่อนำอัลกอริทึมนี้เข้าไปใช้ในระบบ MCU ทุกตัวจะทำงานได้อย่างเหมาะสมกับประสิทธิภาพของเครื่องมากที่สุด และมีการกระจายงานไปยัง MCU ต่างๆได้อย่างเหมาะสม

1.4 ขอบเขตของงานวิจัย

วิทยานิพนธ์ชิ้นนี้ได้ศึกษา และนำเสนออัลกอริทึมการกระจายงานสำหรับ Multipoint Control Unit (MCU) โดยนำเอา องค์ประกอบที่มีผลต่อการทำงานของ MCU เข้ามาเป็นตัวแปรที่ใช้ในอัลกอริทึมการกระจายงาน ได้แก่ การทำงานของซีพียู, หน่วยความจำที่เหลืออยู่, แบนด์วิดท์ที่เหลืออยู่ และ ค่าดีเลย์ระหว่าง MCU และ เทอร์มินอล โดยใช้ MPEG4-MCU เป็น MCU ที่ใช้ในการทดสอบประสิทธิภาพของอัลกอริทึม และใช้เซิร์ฟเวอร์ที่มีคุณสมบัติและสิ่งแวดล้อมที่ต่างกัน 4 เครื่องมาประมวลผลโปรแกรม MPEG4-MCU เพื่อใช้ในการทดสอบประสิทธิภาพของอัลกอริทึม

1.5 ขั้นตอนการศึกษา

1. ศึกษาอัลกอริทึมการกระจายงานทั้งแบบ Static และ Dynamic
2. ศึกษาการทำงานของ MPEG4-MCU เพื่อหาพารามิเตอร์สำคัญที่มีผลต่อการทำงานของ Multipoint Control Unit (MCU)
3. ศึกษาระบบ Distributed System เพื่อทำการพิจารณาการทำงานร่วมกันของคอมพิวเตอร์ที่มีคุณสมบัติและสิ่งแวดล้อมที่แตกต่างกัน
4. วัดประสิทธิภาพการทำงานของระบบ Multipoint Conference ที่ไม่มีตัวกระจายงาน
5. วัดประสิทธิภาพการทำงานของระบบ Multipoint Conference ที่มีตัวกระจายงานที่ใช้อัลกอริทึมของเรา
6. เปรียบเทียบผลการเพิ่มประสิทธิภาพของระบบที่ใช้อัลกอริทึมของเรากับอัลกอริทึมอื่นๆ

1.6 รายละเอียดของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้เป็นการนำเสนออัลกอริทึมการกระจายงานในระบบการประชุมด้วยภาพ โดยมีพารามิเตอร์ที่ใช้ในการพิจารณาไม่เพียงแต่ ซีพียูโหลดเท่านั้น แต่ยังรวมไปถึงการใช้งานหน่วยความจำ แบนด์วิดท์ และ ดีเลย์ระหว่าง ไคลเอนท์กับ MCU เพื่อให้ MCU ทุกๆ เครื่องในระบบการประชุมด้วยภาพได้รับการกระจายงานอย่างเหมาะสม เพื่อให้สามารถให้บริการผู้ใช้ได้อย่างดี เพราะ MCU ที่มีภาระงานมากจะส่งผลกระทบต่อการตอบสนองต่อผู้ใช้บริการ และใช้ทรัพยการที่ระบบมีได้อย่างคุ้มค่ามากที่สุด โดยรายละเอียดต่างๆภายในวิทยานิพนธ์เล่มนี้ได้จัดแบ่งในส่วนเนื้อหาออกเป็น 5 บท ซึ่งแต่ละบทมีหัวข้อและเนื้อหาดังต่อไปนี้

บทที่ 1 "บทนำ" อธิบายถึงวัตถุประสงค์ สมมุติฐานของการศึกษา ขอบเขตและขั้นตอนการศึกษา รวมไปถึงรายละเอียดเนื้อหาโดยสรุปของแต่ละบท

บทที่ 2 "โปรโตคอล H.323 และ MPEG4MCU" โดยจะอธิบายหลักการทำงาน และฟังก์ชั่นเบื้องต้น

บทที่ 3 "อัลกอริทึมการกระจายงาน" โดยจะกล่าวถึงหลักการทำงานเบื้องต้นของอัลกอริทึมการกระจายงาน, อัลกอริทึมต่างๆที่ได้รับการนำเสนอมาก่อนหน้านี้

บทที่ 4 "Load Balancing Algorithms ที่นำเสนอ" การออกแบบอัลกอริทึมการกระจายงานที่นำเสนอ

บทที่ 5 “การเปรียบเทียบและวิเคราะห์ผลอัลกอริทึมการกระจายงานในระบบการประชุมด้วยภาพ” กล่าวถึงการทดลองและผลการทดลอง การเปรียบเทียบภาระงานของ MPEG4MCU ของระบบการประชุมด้วยภาพที่ใช้และไม่ใช้อัลกอริทึมการระจายงานของเรา ในแง่ของ ซีพียูโหลด, จำนวนหน่วยความจำที่ใช้งาน และเน็ตเวิร์คแบนวิดท์

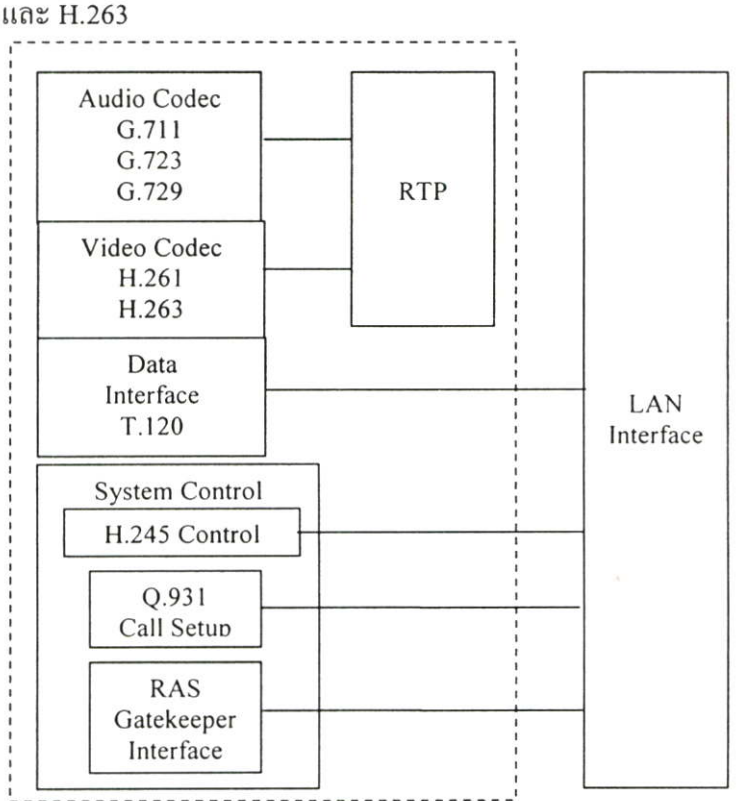
บทที่ 6 “บทสรุป” เป็นการสรุปและวิจารณ์ผลที่ได้จากการวิเคราะห์และการทดลอง พร้อมทั้งปัญหาที่เกิดขึ้น ตลอดจนข้อเสนอแนะสำหรับนำงานวิจัยในวิทยานิพนธ์นี้ไปพัฒนาใช้ประโยชน์ต่อไป

บทที่ 2

โปรโตคอล H.323 และ MPEG4MCU

2.1 โปรโตคอล H.323 (H.323 Protocol)

H.323 เป็นมาตรฐานหรือโปรโตคอลสำหรับการสื่อสารแบบพหุสื่อ (multimedia communication) แบบเวลาจริง บนเครือข่าย IP โปรโตคอล H.323 ได้ให้รายละเอียดสำหรับขั้นตอนในการสร้างการเรียก (call setup) เอนทิตี้ภายในเครือข่าย H.323 และการทำงานร่วมกันระหว่างเอนทิตี้ภายในเครือข่าย H.323 ถูกพัฒนาโดย ITU-T โดยเป็นส่วนหนึ่งของมาตรฐาน H.32x ที่เป็นมาตรฐานสำหรับการประชุมแบบพหุสื่อ (multimedia conference) บนเครือข่ายต่างๆ เช่น H.320 สำหรับเครือข่าย ISDN (Integrated Service Digital Networks) H.324 สำหรับเครือข่าย PSTN (Public Switching Telephone Networks) H.323 จะครอบคลุมโปรโตคอลอื่นไว้ คือ H.225.0 สำหรับ call signaling และการจัดรูปแบบแพ็กเกตมีเดีย (media packet format) H.245 สำหรับการแลกเปลี่ยนข้อมูลความสามารถเกี่ยวกับมีเดีย (media capability exchange) และการควบคุมช่องสัญญาณมีเดีย (media channel control) H.450.x เป็นขั้นตอนสำหรับสร้างการบริการเพิ่มเติม (supplementary service) และ H.235 เป็นมาตรฐานเกี่ยวกับความปลอดภัย เป็นต้น รวมทั้งยังได้อ้างอิงถึงมาตรฐานในการเข้ารหัสสำหรับสัญญาณเสียง เช่น G.711 G723.1 G.729 และสัญญาณวีดีโอเช่น H.261 และ H.263



รูปที่ 2.1 H.323 terminal architecture

ขอบเขตของ H.323 ดังแสดงในรูปที่ 2.1 จะจำกัดอยู่ที่มาตรฐานในการบีบอัดข้อมูล (compression) รูปแบบแพ็กเกตมีเดีย (media packet format) การส่งสัญญาณ (signaling) และ การควบคุมการรับส่งข้อมูล (flow control) ซึ่งจะมีรายละเอียดดังนี้

2.1.1 สถาปัตยกรรมของ H.323 (H.323 Architecture)

โปรโตคอล H.323 ครอบคลุมและอ้างอิงถึงโปรโตคอลอื่นๆ เช่น H.225.0/Q.931, H.245 และH.225.0/RAS เพื่อช่วยในการทำงานของโปรโตคอล H.323 โดยสถาปัตยกรรมของโปรโตคอล H.323 สำหรับเอนด์พอยน์ (endpoint) หรือเทอร์มินัล (terminal) จะเป็นดังรูปที่ 2.1

- การเข้ารหัส/ถอดรหัสสัญญาณวิดีโอ (video codec) ทำหน้าที่ในการเข้ารหัสสัญญาณวีดีโอสำหรับการส่ง และถอดรหัสสัญญาณวีดีโอที่ได้รับซึ่งจะถูกนำไปแสดงผลต่อไป มาตรฐานการเข้ารหัส/ถอดรหัสที่เอนด์พอยน์จำเป็นต้องเข้า/ถอดรหัสได้คือ H.261 ที่ระดับความละเอียด QCIF (Quarter Common Intermediate Format) ส่วน H.263 ซึ่งให้คุณภาพที่ดีกว่าเป็นตัวเลือกที่อาจจะรองรับหรือไม่ก็ได้ รายละเอียดเกี่ยวกับการเข้า/ถอดรหัสสัญญาณวีดีโอจะตกลงกันระหว่างเอนด์พอยน์ในช่วงของการแลกเปลี่ยนความสามารถ (capability exchange) โดยการใช้โปรโตคอล H.245 มาตรฐานการเข้ารหัส/ถอดรหัสที่จะใช้จะต้องรองรับโดยทุกๆเอนด์พอยน์ที่เข้าร่วมในการสื่อสาร
- การเข้ารหัสสัญญาณเสียง (audio codec) ทำหน้าที่ในการเข้ารหัสเสียงจากไมโครโฟน หรือแหล่งกำเนิดอื่นสำหรับการส่ง และถอดรหัสสัญญาณที่ถูกเข้ารหัสที่รับได้ซึ่งจะถูกส่งต่อไปยังลำโพง มาตรฐานที่เอนด์พอยน์จำเป็นต้องรองรับ คือ G.711 สำหรับ G.722, G.728, G.729, MPEG-1 และ G.723.1 เป็นตัวเลือกที่อาจจะรองรับหรือไม่ก็ได้ มาตรฐานการเข้ารหัส/ถอดรหัสที่ใช้จะต้องรองรับโดยทุกเอนด์พอยน์ซึ่งจะทำการตกลงกันโดยใช้โปรโตคอล H.245
- ช่องสัญญาณส่งข้อมูล (data channel) มาตรฐานที่ใช้ คือ T.120 สำหรับการสร้างการประชุมข้อมูล (data conferencing) รายละเอียดหรือพารามิเตอร์อาจจะตกลงกันโดยใช้โปรโตคอล H.245
- RTP (real time transport protocol) ทั้งสัญญาณเสียงและวีดีโอจะถูกส่งโดยบรรจุในแพ็กเกต RTP ซึ่งเป็นโปรโตคอลที่ใช้สำหรับการส่งข้อมูลแบบเวลาจริง บนเครือข่าย IP โดย RTP จะทำงานร่วมกับ RTCP ซึ่งทำหน้าที่ในการควบคุมการส่งข้อมูลโดย RTP ดังที่ได้กล่าวมาแล้ว

System Control Unit เป็นส่วนที่ทำหน้าที่เกี่ยวกับ การส่งสัญญาณ และ flow control ประกอบด้วยส่วนต่างๆ ดังนี้

- H.245 เป็นโปรโตคอลควบคุมมีเดีย (media control protocol) ทำหน้าที่ในการแลกเปลี่ยนความสามารถ (capability exchange) ตกลงรายละเอียดของช่องสัญญาณ (channel negotiation) เปลี่ยนโหมดของมีเดีย (switching of media mode) และการสร้างช่องสัญญาณทางตรรก (logical channel) สำหรับการส่งเสียงหรือวิดีโอ โปรโตคอลนี้จะใช้ TCP ในการส่งแมสเสจโดยใช้ช่องสัญญาณในการส่งของตัวเอง
- H.225.0/Q.931 เป็นโปรโตคอล H.225.0 ในส่วนที่ทำหน้าที่สร้างการเชื่อมต่อ (connection establishment) ซึ่งถูกดัดแปลงมาจาก โปรโตคอล Q.931 โดยโปรโตคอลนี้จะใช้ TCP ในการส่งแมสเสจผ่านช่องสัญญาณของตัวเอง
- H.225.0 RAS เป็นโปรโตคอล H.225.0 ที่ทำหน้าที่ในการควบคุมการยอมรับ (admission control) การลงทะเบียน (registration) และการรายงานสถานะ โปรโตคอลนี้จะใช้ระหว่างเอนด์พอยน์และเกทคีปเปอร์ (gatekeeper) เพื่อใช้สำหรับการควบคุมดูแลโดยเกทคีปเปอร์ โดยแมสเสจในโปรโตคอลจะใช้ UDP
- H.225.0 layer โปรโตคอล H.225.0 เป็นโปรโตคอลสำหรับ call-signaling ซึ่งมีหน้าที่ดังที่กล่าวมาข้างต้น นอกจากนั้นยังทำหน้าที่ในการแปลงมีเดีย (วิดีโอ เสียง และข้อมูล) และข้อมูลสำหรับการควบคุม (control data) ที่จะถูกส่งให้อยู่ในรูปแพ็กเกตที่เหมาะสมเพื่อส่งต่อให้กับ network interface และทำหน้าที่รับข้อมูลทั้งหมด (media data และ control data) จาก network interface เพื่อส่งให้ส่วนอื่นต่อไป

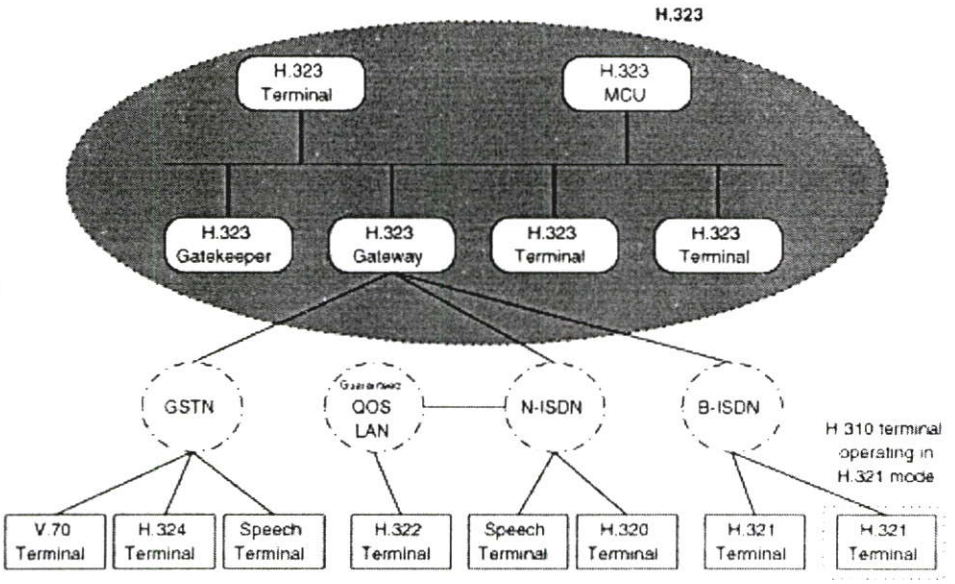
2.1.2 เอนทิตี้และฟังก์ชันของ H.323 (H.323 entities & functions)

ในมาตรฐาน H.323 ได้อธิบายถึงเอนทิตี้ที่เป็นองค์ประกอบเครือข่าย H.323ซึ่งได้แก่ เทอร์มินัล MCU (Multipoint Control Unit) เกทเวย์ และเกทคีปเปอร์ การเชื่อมต่อระหว่างเอนทิตี้ภายในเครือข่าย H.323 กับเครือข่ายอื่นดังแสดงในรูปที่ 2.3

รายละเอียดของแต่ละเอนทิตี้มีดังนี้

1) H.323 เทอร์มินัล (H.323 terminal)

เทอร์มินัลดังแสดงในรูปที่ 2.1 เป็นเอนด์พอยน์ของเครือข่ายซึ่งอาจจะเป็นคอมพิวเตอร์หรือชุดอุปกรณ์ที่สามารถใช้งานโปรโตคอล H.323 ได้ เทอร์มินัลต้องสนับสนุนการสื่อสารโดยใช้เสียง ส่วนสัญญาณวีดีโอและข้อมูลเป็นตัวเลือก ซึ่งฟังก์ชันหลักของเทอร์มินัลมีดังนี้



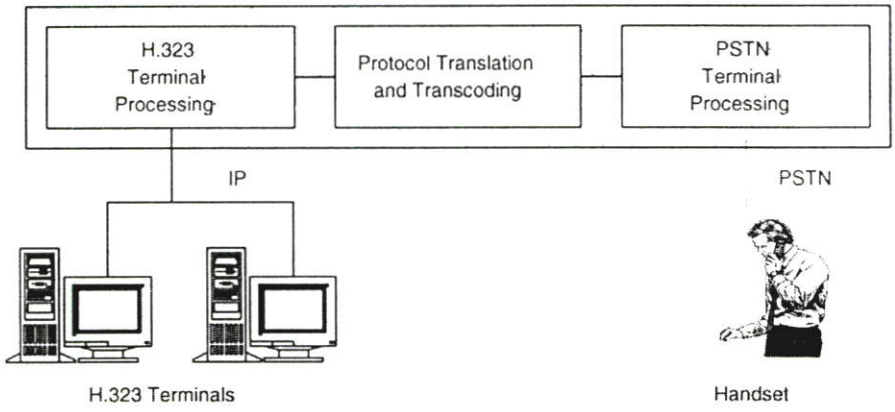
รูปที่ 2.2 เครือข่าย H.323 เอนทิตี้และฟังก์ชัน

- ทำหน้าที่ในการติดต่อกับผู้ใช้ โดยรับคำสั่งและแสดงผลให้กับผู้ใช้
- จัดการในการส่ง call signaling ให้กับ voice gateway
- ส่งหมายเลขโทรศัพท์ (มาตรฐาน E.164) และหมายเลข IP ของผู้ใช้ ให้กับเกทคีปเปอร์ ซึ่งเป็นหมายเลขที่ใช้อ้างอิงถึงในการเชื่อมต่อ ในการส่งหมายเลขดังกล่าวจะบรรจุอยู่ในแมสเสจ ARQ ของโปรโตคอล H.225.0/ RAS ซึ่งอาจจะมีหมายเลข alias address ส่งไปพร้อมกัน
- ทำการแปลงแพ็กเกตที่ได้รับจากเครือข่าย โดยผ่านกระบวนการของโปรโตคอลในชั้นต่างๆ ตามลำดับ (IP->UDP->RTP) เป็นเฟรมเสียง แล้วทำการถอดรหัส G.xxx ให้อยู่ในรูปของ PCM (Pulse Code Modulation) สตรีมเพื่อทำการส่งให้กับซาวน์การ์ดเพื่อแสดงผลต่อไป
- ทำการเข้ารหัส G.xxx ให้กับ PCM สตรีมจากซาวน์การ์ดและทำการรวมเป็นแพ็กเกต แล้วแปลงแพ็กเกตเป็นแพ็กเกตที่ส่งในเครือข่ายโดยผ่านกระบวนการของโปรโตคอลในชั้นต่างๆ (RTP->UDP->IP) แล้วจึงทำการส่งผ่านเครือข่ายในรูปของแพ็กเกต

2) H.323 เกทเวย์ (H.323 Gateway)

เกทเวย์เป็นเอนทิตี้ที่ทำหน้าที่ในการเชื่อมต่อระหว่างเครือข่าย H.323 กับเครือข่ายอื่นซึ่งอาจจะไม่จำเป็นต้องมีในกรณีที่ไม่มีการเชื่อมต่อกับเครือข่ายชนิดอื่นๆ การเชื่อมต่อเครือข่าย H.323

กับเครือข่ายอื่นโดยใช้เกทเวย์จะมีลักษณะดังรูปที่ 2.4 เกทเวย์ทำหน้าที่เสมือนเป็นเอนด์พอยน์ของเครือข่ายหนึ่งในการเชื่อมต่อระหว่างเครือข่าย โดยจะทำหน้าที่ในการเชื่อมต่อระหว่างเครือข่ายดังนี้



รูปที่ 2.3 H.323/PSTN Gateway

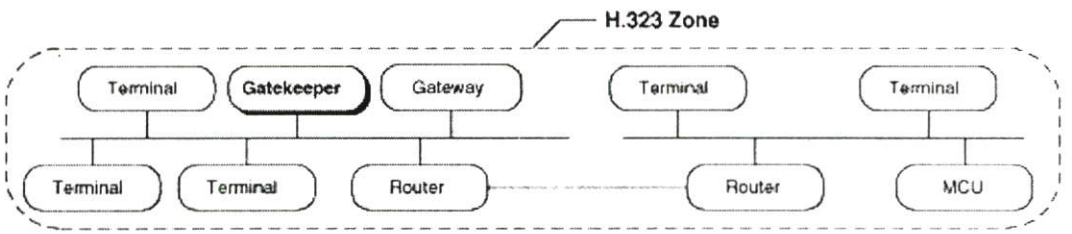
- สร้างการเชื่อมต่อกับเทอร์มินัล PSTN ในระบบแอนาลอก
- สร้างการเชื่อมต่อกับเทอร์มินัลที่รองรับมาตรฐาน H.320 บนเครือข่าย switched circuit ที่เป็น ISDN
- สร้างการเชื่อมต่อเทอร์มินัลที่รองรับมาตรฐาน H.324 บนเครือข่าย PSTN

เนื่องจากเกทเวย์สามารถให้การเชื่อมต่อระหว่าง H.323 กับเครือข่ายอื่น ดังนั้นฟังก์ชันของเกทเวย์ จึงเป็นฟังก์ชันในการแปลงข้อมูลระหว่าง 2 เครือข่ายคือ

- รับและประมวลผลการเรียกที่มาจากเทอร์มินัลในเครือข่ายอื่นไปยังเทอร์มินัล H.323 เกทเวย์จะต้องทำการแปลง การส่งสัญญาณและcontrol ต่างๆ จากเครือข่ายอื่นมาเป็นของ H.323 เช่น จากขั้นตอนในการสร้างการสื่อสารจาก H.242 เป็น H.245 รวมทั้งทำหน้าที่สร้างและสิ้นสุดการเรียก หรืออาจจะมองได้ว่าเกทเวย์จะทำหน้าที่แทนเทอร์มินัลในเครือข่ายอื่นโดยเสมือนกับเป็นเทอร์มินัลในเครือข่าย H.323
- รับและประมวลผลการเรียกจากเทอร์มินัล H.323 ไปยังเครือข่ายอื่น เกทเวย์จะต้องทำการแปลงการส่งสัญญาณ และ control ต่างๆ ตาม H.323 ให้เป็นมาตรฐานในเครือข่ายอื่น เช่น แปลงจากโปรโตคอล H.245 เป็น H.242 รวมทั้งสร้างและสิ้นสุดการเรียก หรืออาจจะมองว่าเกทเวย์จะทำหน้าที่แทนเทอร์มินัลในเครือข่าย H.323 เสมือนกับเป็นเทอร์มินัลในเครือข่ายอื่น
- ทำหน้าที่ในการดูแลกระบวนการการเรียก (call) และการส่งสัญญาณ (signaling) ว่ามีความผิดพลาดเกิดขึ้นหรือไม่ ซึ่งการทำงานจะอยู่ภายใต้การควบคุมของเกทคีปเปอร์

- ทำการเข้ารหัส G.xxx ให้กับสัญญาณ PCM จากเครือข่ายอื่น แล้วทำการรวมสัญญาณที่ถูกเข้ารหัสให้เป็นแพ็กเกตเพื่อส่งไปในเครือข่าย IP โดยผ่านกระบวนการแปลงของโปรโตคอลในชั้นต่างๆ เพื่อให้อยู่ในรูปแพ็กเกตที่สามารถส่งไปในเครือข่าย ดังในหัวข้อที่ 2
- ทำการแปลงแพ็กเกตจากเครือข่าย IP กลับแพ็กเกตเสียง แล้วทำการถอดรหัส G.xxx และแปลงสัญญาณ PCM เพื่อส่งให้กับเครือข่ายภายนอก

สำหรับฟังก์ชันเพิ่มเติมของเกทเวย์ไม่ได้มีการกำหนดไว้แน่นอนขึ้นอยู่กับผู้ออกแบบ เช่น จำนวนของเทอร์มินัลที่รองรับจำนวนการเชื่อมต่อกับเครือข่าย switched circuit และจำนวนการประชุมที่สนับสนุน เป็นต้น เมื่อเกทเวย์มีการพัฒนามากขึ้นจะทำให้เครือข่าย H.323 สามารถเชื่อมต่อกันกับเครือข่ายชนิดอื่นๆ ได้มากขึ้น



รูปที่ 2.4 H.323 Zone

3) H.323 เกทคีปเปอร์ (Gatekeeper)

เกทคีปเปอร์ทำหน้าที่ในการดูแลและให้บริการกับเอนทิตี้อื่นภายในโซนดังรูปที่ 2.5 โดยโซนจะประกอบไปด้วยเกทคีปเปอร์ 1 ตัวและเอนทิตี้อื่นๆ ทั้งหมดที่ลงทะเบียนกับเกทคีปเปอร์ ถึงแม้ว่าเกทคีปเปอร์เป็นเอนทิตี้ที่ไม่จำเป็นต้องมีในเครือข่าย H.323 แต่เกทคีปเปอร์ก็เป็นเอนทิตี้ที่สำคัญมาก ด้วยเหตุผลต่างๆ ดังนี้

- เครือข่ายขนาดใหญ่สามารถแบ่งเป็นหลายโซน ซึ่งแต่ละโซนจะอยู่ภายในการดูแลของเกทคีปเปอร์เพื่อความสะดวกในการดูแลรักษาเครือข่าย
- เกทคีปเปอร์สามารถให้ความปลอดภัยในการเข้าถึงเครือข่ายได้โดยการให้บริการ authentication สำหรับแต่ละการเรียก หรือแต่ละเอนทิตี้
- เกทคีปเปอร์เป็นศูนย์กลางในการ authentication authorization และ admission (เรียกรวมกันว่า AAA) ของโซน
- เกทคีปเปอร์สามารถจัดการควบคุมแบนด์วิดท์(bandwidth management) เช่นการจำกัดจำนวนของการเชื่อมต่อ

เพื่อเป็นการรักษาแบนวิดท์สำหรับการใช้งานอย่างอื่น เช่น อีเมล์ การโอนย้ายไฟล์ เกทคีปเปอร์ไม่สามารถเป็นเอนด์พอยน์ของการเชื่อมต่อได้ เมื่อมีเกทคีปเปอร์ทุกเอนทิตี้จะต้องทำการลงทะเบียนกับเกทคีปเปอร์ ดังนั้นเกทคีปเปอร์จึงทำหน้าที่เป็นศูนย์กลางของการเรียกทั้งหมดภายในโซนและอาจจะให้บริการเพิ่มเติมกับโซนได้ สำหรับฟังก์ชันหลักที่จำเป็นของเกทคีปเปอร์ตามมาตรฐาน H.323 มี 4 ฟังก์ชันดังนี้

- การแปลงแอดเดรส (Address translation) เกทคีปเปอร์จะทำหน้าที่ในการแปลง alias address ให้เป็น transport address เอนทิตี้จะทำการส่ง alias พร้อมกับการลงทะเบียนโดยใช้แมสเสจ RRQ ซึ่งอาจจะสามารถปรับเปลี่ยนในภายหลังได้
- การควบคุมการยอมรับ (Admission control) เมื่อเอนทิตี้ภายในโซนต้องการสร้างการเรียกจะต้องทำการขออนุญาตไปยังเกทคีปเปอร์ โดยใช้แมสเสจ ARQ เกทคีปเปอร์อาจจะอนุญาตหรือไม่ก็ได้โดยจะทำการตรวจสอบจากเงื่อนไขต่างๆ เช่น แบนด์วิดท์ แหล่งกำเนิดการเรียก (call) และ authentication เป็นต้น
- การควบคุมแบนด์วิดท์ (Bandwidth control) เกทคีปเปอร์สามารถรองรับการควบคุมแบนด์วิดท์ได้ เอนทิตี้จะทำการร้องขอแบนด์วิดท์ที่ต้องการ โดยใช้แมสเสจ BRQ และเกทคีปเปอร์จะทำการตรวจสอบค่าแบนด์วิดท์ที่ร้องขอกับเงื่อนไขที่กำหนดไว้สำหรับการจัดการแบนด์วิดท์ (bandwidth management) แล้วจึงจะอนุญาตหรือไม่อนุญาตด้วยการส่งแมสเสจ BCF หรือ BRJ ตามลำดับ
- การจัดการโซน (Zone management and Directory service) โซนจะประกอบด้วยเทอร์มินัล เกทเวย์และ MCU ทั้งหมดที่ลงทะเบียนกับ เกทคีปเปอร์ 1 ตัว เกทคีปเปอร์ทำหน้าที่ในการดูแลและจัดการให้กับทุกเอนทิตี้ที่อยู่ในภายในโซน โดยการใช้ฟังก์ชันข้างต้นและการให้บริการอื่นๆ รวมทั้งการให้บริการ directory service ของโซน

นอกจากฟังก์ชันหลักดังกล่าวแล้ว เกทคีปเปอร์อาจจะให้ฟังก์ชันเพิ่มเติมอื่นๆ มีดังนี้

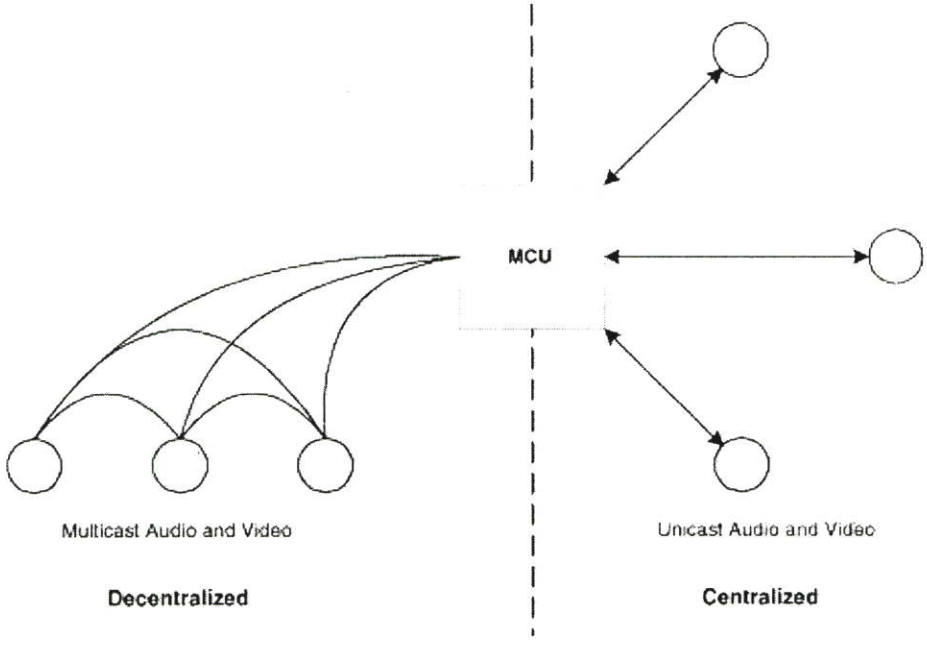
- การควบคุมการส่งสัญญาณ (call control signaling) เกทคีปเปอร์อาจจะช่วยในการประมวลผลแมสเสจ Q.931 ที่ส่งระหว่าง เทอร์มินัลได้ในระหว่างการสร้างการเรียก
- การตรวจสอบการเรียก (call authorization) เกทคีปเปอร์อาจจะปฏิเสธการสร้างการเรียกจากเทอร์มินัลด้วยเหตุผลบางอย่าง เช่น จำกัดการเข้าถึงจากเทอร์มินัลหรือเกทเวย์บางตัว จำกัดการเข้าถึงในบางช่วงเวลา เป็นต้น ซึ่งเงื่อนไขในการตรวจสอบจะอยู่นอกเหนือขอบเขตของ H.323
- การจัดการแบนด์วิดท์ เกทคีปเปอร์จะปฏิเสธการเรียกจากเทอร์มินัลในกรณีที่มีแบนด์วิดท์ไม่เพียงพอ รวมถึงในกรณีที่มีการร้องขอการเพิ่มแบนด์วิดท์ สำหรับเงื่อนไขจะอยู่นอกเหนือขอบเขตของ H.323

- การจัดการการเรียก (call management) เกทคีปเปอร์อาจจะทำการเก็บรักษารายการการเรียกที่เกิดขึ้นเพื่อใช้ในการระบุเทอร์มินัลที่ถูกเรียกว่าว่างหรือไม่ หรือเพื่อให้ข้อมูลกับฟังก์ชันในการจัดการแบนด์วิดท์
- การตรวจสอบผู้ใช้ (authenticating users) สามารถจำกัดการเข้าถึงของผู้ใช้ได้ตามเงื่อนไขที่กำหนดไว้
- การจัดการบริการ (managing services) เกทคีปเปอร์จะทำหน้าที่ในการจัดการให้บริการต่างๆ แก่ผู้ใช้
- การจัดการฐานข้อมูลของสมาชิก (managing subscriber databases) เกทคีปเปอร์ทำหน้าที่ดูแลและจัดการเกี่ยวกับฐานข้อมูลของสมาชิกที่ได้ลงทะเบียนไว้กับเกทคีปเปอร์
- การหาตำแหน่งของสมาชิก (locating subscribers) เกทคีปเปอร์ทำหน้าที่ในการหาตำแหน่งของสมาชิกได้โดยการค้นหาจากข้อมูลของสมาชิกที่ได้จากการลงทะเบียน
- การรวบรวมข้อมูลสำหรับการเก็บค่าบริการ (collecting charging information) เกทคีปเปอร์จะทำการเก็บข้อมูลที่จำเป็นสำหรับการคิดค่าบริการของการเรียก โดยที่การเรียก (call) ต้องถูกจัด เส้นทางผ่านเกทคีปเปอร์
- การควบคุมเกทเวย์ (managing gateway) เกทคีปเปอร์จะควบคุมการทำงานของเกทเวย์ เช่น ควบคุมการสร้างการเรียก ของเกทเวย์ระหว่างเครือข่าย
- การช่วยในการสร้างการเรียก (assisting in call setup) เมื่อการเรียกถูกจัดเส้นทางผ่านเกทคีปเปอร์ เกทคีปเปอร์จะช่วยในการสร้างการเรียก เช่นอาจจะทำการจัดเส้นทางให้กับการเรียก ไปยังเกทเวย์ที่เหมาะสม การติดต่อสื่อสารระหว่างเอนทิตี้กับเกทคีปเปอร์จะใช้โปรโตคอล H.225.0 /RAS ส่วนแมสเสจ call signaling (H.225.0/ Q.931) และ media control (H.245) อาจจะผ่านเกทคีปเปอร์หรือไม่ก็ได้ขึ้นอยู่กับการลงทะเบียนของเอนทิตี้และเงื่อนไขของเกทคีปเปอร์สำหรับ เกทคีปเปอร์อาจจะถูกรวมอยู่ในเกทเวย์และ MCU ได้ โดยที่ต้องแยกทางตรรก (logical) จากเอนด์พอยน์

4) Multipoint Control Unit (MCU)

MCU ทำหน้าที่ในการสนับสนุนการประชุมแบบหลายจุด (multipoint conference) ระหว่างเทอร์มินัล 3 เทอร์มินัลขึ้นไป MCU เป็นเอนทิตี้ที่จะมีหรือไม่ก็ได้ MCU ประกอบด้วย multipoint controller (MC) และ multipoint processor (MP) ในการประชุมจะต้องมี MC ส่วน MP อาจจะมีหรือไม่มีก็ได้ หรืออาจจะมีมากกว่าหนึ่งก็ได้ MC เป็นส่วนที่ทำหน้าที่ในจัดการเกี่ยวกับการส่งสัญญาณในการควบคุมมีเดีย (media control signaling) ให้กับแต่ละเทอร์มินัล โดยที่ทุกเทอร์มินัลต้องมีช่องสัญญาณ H.245 เชื่อมต่อกับ MC แบบจุดถึงจุด (point-to-point) ส่วน MP จะทำ

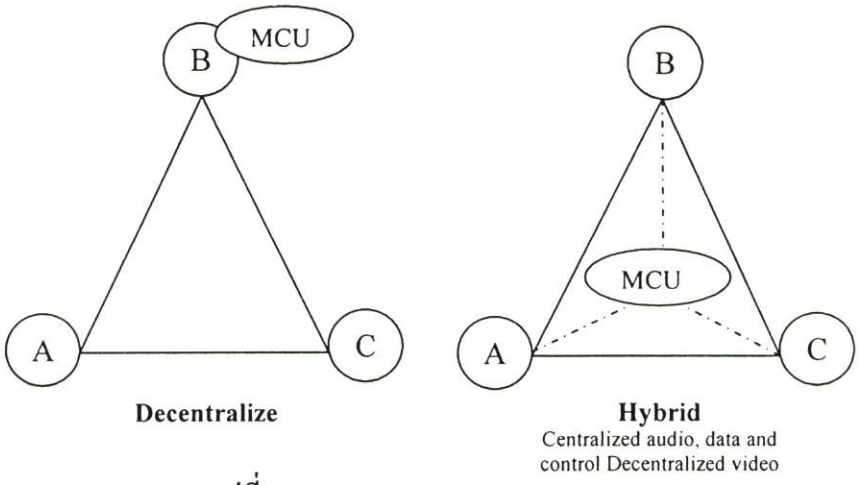
หน้าที่ในการจัดการกับมีเดียสตรีมโดยทำหน้าที่ในการผสม (mixing) สวิตซ์ (switching) และประมวลผลมีเดียที่ใช้การประชุมภายใต้การ ควบคุมของ MC



รูปที่ 2.5 Centralized และ Decentralized MCU

การประชุมแบบหลายจุด (Multipoint conference) คือการสื่อสารที่มีผู้เข้าร่วมมากกว่า 2 ซึ่งจำเป็นต้องมี MC อยู่เป็นอย่างน้อย สำหรับแบบจำลองที่ใช้มี 3 แบบ

- Centralize Model ดังรูปที่ 2.5 ในแบบจำลองนี้จำเป็นต้องมี MCU อยู่ ทุกๆเทอร์มินัลที่เข้าร่วมในการประชุมต้องมีช่องสัญญาณ H.245 เชื่อมต่อแบบจุดถึงจุด (point-to-point) กับ MCU ซึ่ง MC จะหน้าที่ควบคุมการประชุมโดยใช้ฟังก์ชันของ H.245 ส่วน MP จะทำหน้าที่รับมีเดียสตรีมจากทุกเทอร์มินัลทำการรวมสัญญาณเสียง เลือกสัญญาณวีดีโอที่ตรงกัน และประมวลผล แล้วทำการส่งกลับไปให้กลับเทอร์มินัลอื่นๆ ทุกเทอร์มินัล
- Decentralized Model ดังรูปที่ 2.5 และ 2.6 ในแบบจำลองนี้ เทอร์มินัลจะมัลติคาสท์สัญญาณเสียงและวีดีโอให้กับเทอร์มินัลอื่นๆ โดยไม่ผ่าน MCU แต่การควบคุมยังคงถูกควบคุมโดย MC ผ่านทางช่องสัญญาณ H.245 ที่เชื่อมต่อกับเทอร์มินัลแบบจุดถึงจุด (point-to-point) เทอร์มินัลที่ได้รับสัญญาณจะทำหน้าที่ในการประมวลผลสัญญาณเอง โดยอาจจะใช้ฟังก์ชัน MP ของแต่ละเทอร์มินัลช่วยทำหน้าที่ในการประมวลผลมีเดียสตรีม

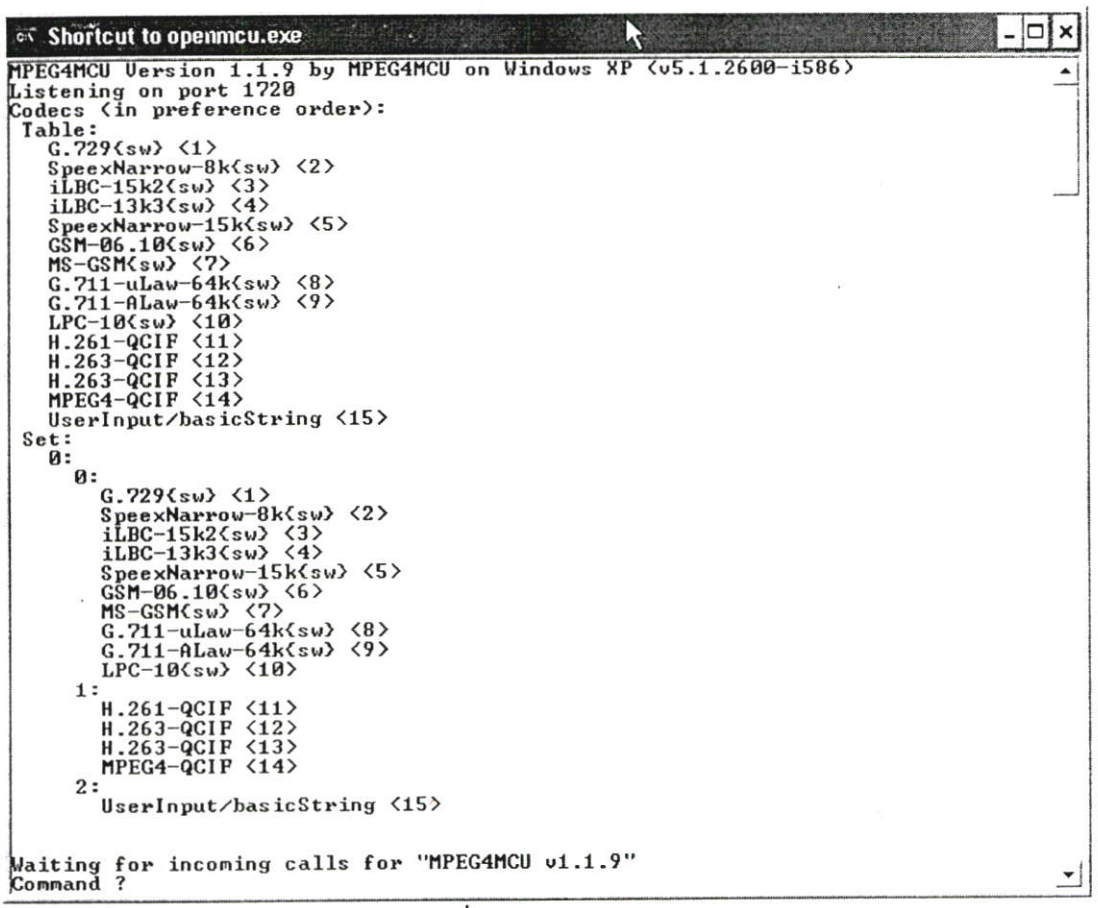


รูปที่ 2.6 Decentralized และ Hybrid MCU

- Hybrid Model ดังรูปที่ 2.6 แมสเสจ H.245 รวมทั้งสัญญาณเสียงหรือวีดีโอจะถูกส่งและประมวลผลผ่านMCUโดยใช้การเชื่อมต่อแบบจุดถึงจุด (point-to-point) ส่วนสัญญาณที่เหลือจะถูกส่งโดยเทอร์มินัลแบบมัลติคาสท์ให้กับเทอร์มินัลอื่นๆ

2.2 MPEG4MCU

MPEG4MCU[1] ดังรูปที่ 2.7 เป็นโอเพ่นซอร์ส H.323 MCU ซึ่งใช้ในงานวิจัยนี้ โดยทำงานบนไลบรารีหลัก 2 ตัวคือ PwLib.dll และ OpenH323.dll โดย PwLib.dll จะบรรจุคลาสต่างๆ สำหรับการเขียนโปรแกรมข้ามแพลตฟอร์ม ส่วน OpenH323.dll จะบรรจุคลาสต่างๆที่เกี่ยวกับ H.323 ทั้งหมดไม่ว่าจะเป็นการติดต่อ, การยกเลิกการติดต่อ, codec ต่างๆทั้งออดิโอและวิดีโอ



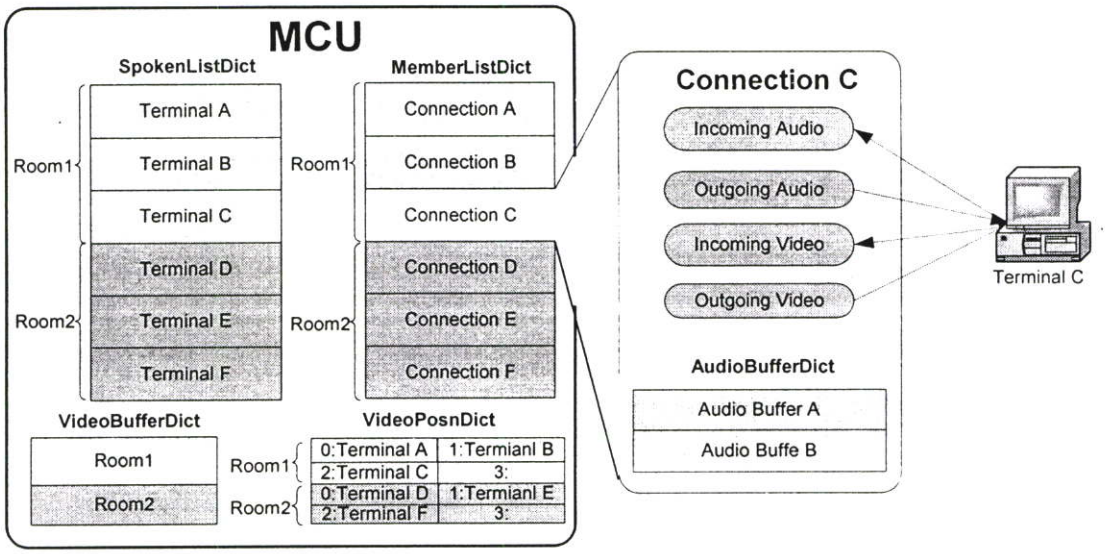
รูปที่ 2.7 โปรแกรม MPEG4MCU

2.2.1 โครงสร้างข้อมูลและการทำงานของ MPEG4MCU

MPEG4MCU มีส่วนประกอบหลักดังรูปที่ 2.8 โดยประกอบด้วยส่วนประกอบ 4 ส่วนคือ SpokenListDict, MemberListDict, VideoBufferDict และ VideoPosnDict เราจะเห็นว่าส่วนประกอบทั้งสี่มีคำว่า Dict ตามหลัง คำว่า Dict นี้มีความหมายว่าส่วนประกอบทั้ง 4 มีลักษณะคล้าย Dictionary คือเป็นที่เก็บออบเจ็กต์โดยใช้คีย์ซึ่งเป็นสตริงในการค้นหา

MemberListDict เป็น Dict ที่ใช้เก็บ MemberList โดยใช้คีย์เป็นชื่อของห้อง MemberList นี้เป็นลิสต์ที่เก็บออบเจ็กต์คอนเน็คชัน โดยแต่ละคอนเน็คชันจะทำหน้าที่ติดต่อกับเทอร์มินัล ออบเจ็กต์คอนเน็คชันประกอบด้วย Incoming Audio มีหน้าที่รับออดิโอสตรีมที่เข้ามา, Outgoing

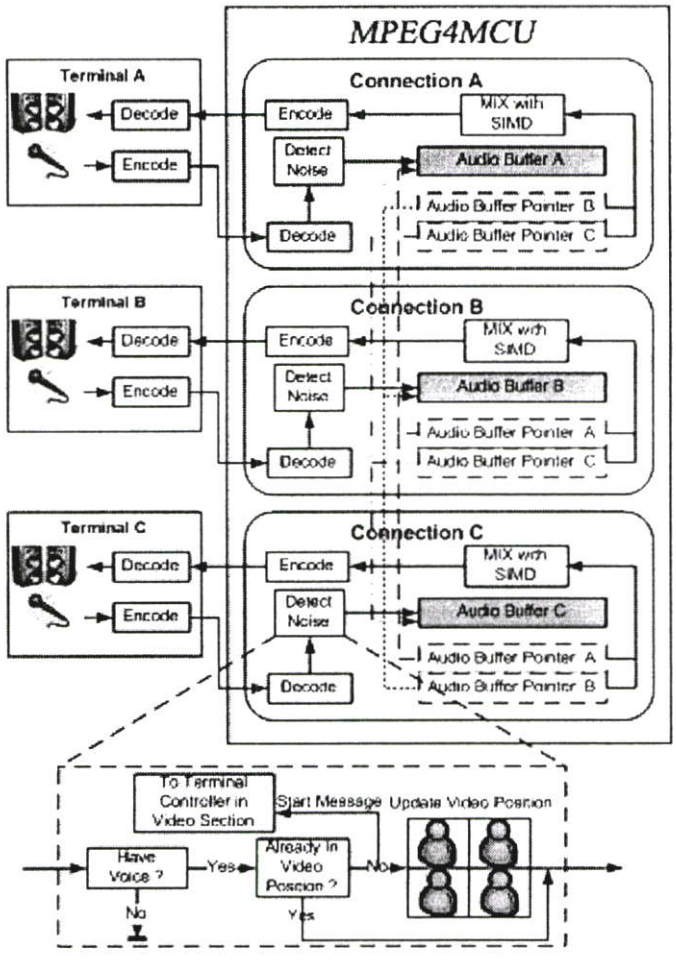
Audio มีหน้าที่ส่งออดิโอสตรีมไปสู่เทอร์มินัล, Incoming Video มีหน้าที่รับวิดีโอสตรีมที่เข้ามาและ Outgoing Video ทำหน้าที่ส่งวิดิโอสตรีมไปสู่เทอร์มินัล ภายในออบเจ็กต์คอนเน็คชันนี้ยังประกอบไปด้วย AudioBufferDict ซึ่งเป็น Dict ที่ใช้เก็บออดิโอบัฟเฟอร์ของทุกคอนเน็คชันยกเว้นของตัวมันเอง (ป้องกัน Echo) เพื่อใช้ประโยชน์ในการผสมออดิโอสตรีมที่เข้ามาแล้วส่งไปให้เทอร์มินัลต่างๆ โดยเราสามารถแบ่งการทำงานของ MPEG4MCU ออกเป็น 2 ส่วนคือ ส่วนออดิโอ และส่วนวิดีโอ ดังนี้



รูปที่ 2.8 ส่วนประกอบหลักของ MPEG4MCU

1) ส่วนออดิโอ

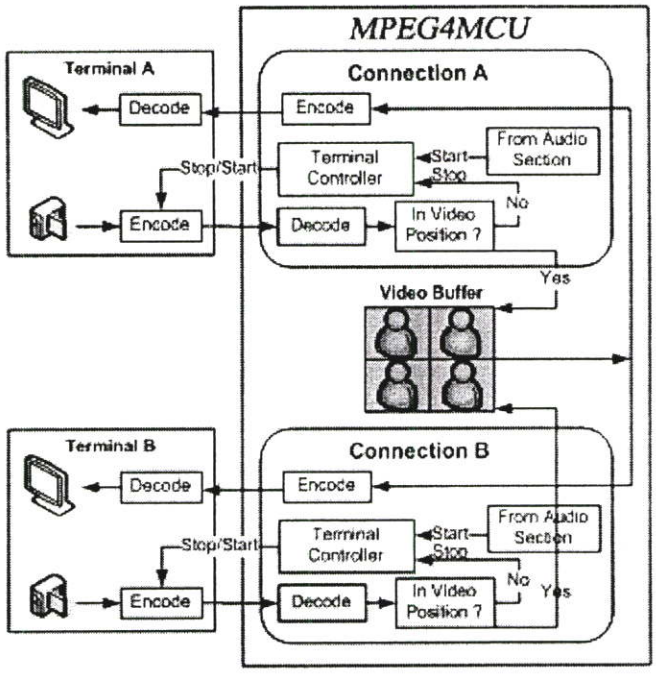
การทำงานของส่วนออดิโอสามารถแสดงได้ดังรูปที่ 2.9 โดยเทอร์มินัลต่างๆเชื่อมต่อกับ MPEG4MCU ผ่านทางคอนเน็คชันออบเจ็ก โดยเทอร์มินัลต่างๆจะบีบอัดออดิโอสตรีมที่ได้มาแล้วส่งต่อให้ MPEG4MCU หลังจากที่คอนเน็คชันออบเจ็กใดๆได้รับออดิโอสตรีมจากเทอร์มินัลแล้วจะทำการปรับปรุงขนาดของออดิโอบัฟเฟอร์ที่เข้ามาใหม่ในออบเจ็กต์ออดิโอบัฟเฟอร์พอยน์เตอร์ของทุกคอนเน็คชัน ก็จะทำการดีโค้ดแล้วใส่ในบัฟเฟอร์หลังจากนั้นจะนำบัฟเฟอร์นี้ไปยังฟังก์ชัน DetectNoise โดยฟังก์ชัน DetectNoise จะหาว่าผู้ใช้ที่เทอร์มินัลนี้พูดหรือไม่ ถ้าพูดก็จะทำการปรับปรุง SpokenList (หาได้จาก SpokenListDict โดยใช้คีย์เป็นชื่อห้อง) โดยใส่ชื่อของเทอร์มินัลนี้ที่ท้ายลิสต์ หลังจากนั้นก็จะทำการตรวจสอบว่าเทอร์มินัลนี้อยู่ใน 4 อันดับแรกของ VideoPosnList (หาได้จาก VideoPosnDict โดยใช้คีย์เป็นชื่อห้อง) หรือไม่ ถ้าไม่อยู่ก็จะแทนที่เทอร์มินัลนี้กับเทอร์มินัลที่พูดในอันดับแรกสุด



รูปที่ 2.9 Block Diagram การทำงานของ MPEG4MCU ในส่วนออดิโอ

รูปที่ 2.10 การผสมภาพเมื่อเชื่อม Microsoft Netmeeting กับ MPEG4MCU

2) การทำงานของส่วนวิดีโอ



รูปที่ 2.11 Block Diagram การทำงานของ MPEG4MCU ในส่วนวิดีโอ

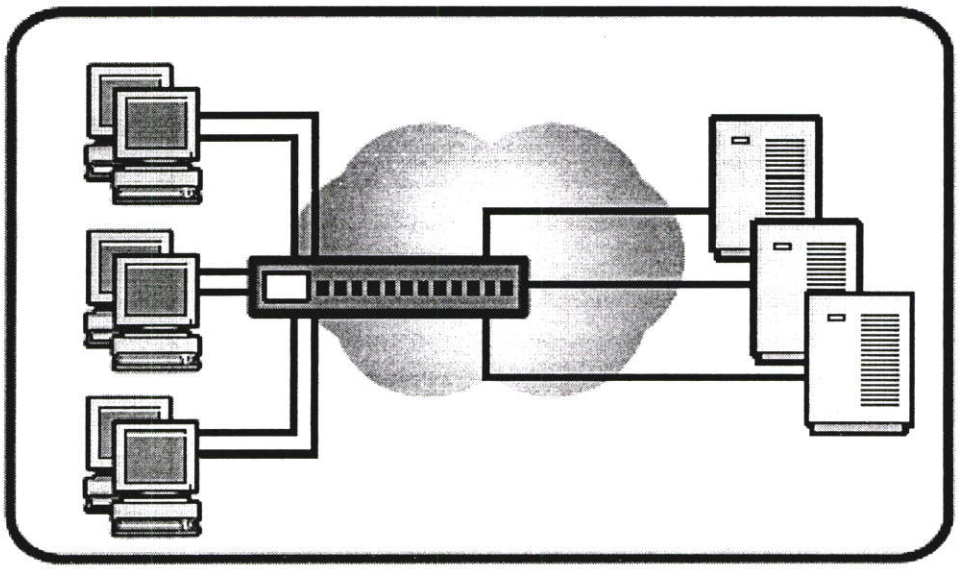
การทำงานของส่วนวิดีโอแสดงดังรูปที่ 2.11 เมื่อ MPEG4MCU รับวิดีโอสตรีมมาจากเทอร์มินัลแล้วจากนั้นดีโคด แล้วตรวจสอบว่าอยู่ใน 4 อันดับแรกของ VideoPosnList หรือไม่ ถ้าใช่ก็ทำการ Down Sampling ข้อมูลวิดีโอแล้วสำเนาลงในวิดีโอบัฟเฟอร์ในช่องที่เหมาะสม จากนั้นเมื่อถึงเวลาที่เหมาะสม (เวลา Playout Time) ก็จะนำข้อมูลวิดีโอจากบัฟเฟอร์ทั้ง 4 ช่องมาบีบอัดจากนั้นส่งกลับไปให้กับเทอร์มินัลต่างๆ ดังแสดงในรูปที่ 2.10 แต่ที่เวลาใดๆ MPEG4MCU จะแสดงผลเฉพาะข้อมูลวิดีโอของ 4 คนที่พูดล่าสุดเท่านั้นส่วนคนอื่นๆที่อยู่นอกเหนือ 4 คนนี้ MPEG4MCU ก็ยังคงจะต้องดีโคดแล้วทิ้งไป ซึ่งส่งผลเสียต่อซีพียูโหลด และอินคัมมิ่งแบนด์วิดท์ (Incoming Bandwidth) ของระบบเป็นอย่างสูง

บทที่ 3

Load Balancing Algorithms

การกระจายงานเป็นกระบวนการสำหรับระบบที่มีคอมพิวเตอร์หลายเครื่องช่วยกันทำงาน รูปที่ 3.1 แสดงถึงการกระจายงานในกลุ่มของเซิร์ฟเวอร์ โดย การกระจายงานจะมีความสำคัญกับระบบทางเครือข่ายมากขึ้นดังนี้

- เพิ่มขนาดของระบบให้รองรับได้มากขึ้น
- มีประสิทธิภาพสูง
- ความเชื่อถือได้และความสามารถในการกู้ระบบจากอุบัติภัยต่างๆสูง



รูปที่ 3.1 ระบบงานแบบกระจายที่มีการใช้อุปกรณ์กระจายงาน

คำร้องขอจากผู้ใช้บริการจะถูกส่งมายังอุปกรณ์ที่ใช้ในการกระจายงาน(Load Balancer - LB) เมื่อ LBตัดสินใจด้วยอัลกอริทึมแล้วว่าเซิร์ฟเวอร์ตัวใดเหมาะสมที่จะรองรับคำร้องขอนี้ ก็จะทำการส่งต่อไปยังเซิร์ฟเวอร์ นั้น โดย LB จะช่วยให้ระบบใช้ทรัพยากรอย่างคุ้มค่า มีการตอบสนองที่รวดเร็ว แต่มีข้อควรระวังไว้ว่า LB จะต้องสามารถรองรับจำนวนของเซิร์ฟเวอร์ได้ตามความต้องการของระบบ ไม่เช่นนั้นจะทำให้ส่วนของ LB เป็นคอขวดของระบบ โดยประสิทธิภาพของการกระจายงานนั้นคือการที่จะส่งงานไปยังเซิร์ฟเวอร์ต่างๆได้อย่างฉลาดและเหมาะสม โดยมีอัลกอริทึมเป็นกุญแจสำคัญในการทำงาน

3.1 ประเภทของอัลกอริทึมการกระจายงาน

อัลกอริทึมการกระจายงานสามารถแบ่งเป็น 2 แบบได้แก่

1. Static Load Balancing อัลกอริทึมแบบนี้จะทำการกระจายงานไว้ล่วงหน้าก่อนที่ระบบจะเริ่มต้นการทำงาน
2. Dynamic Load Balancing อัลกอริทึมแบบนี้จะทำการกระจายงานไปยังเซิร์ฟเวอร์อื่นๆ โดยอาศัยข้อมูลหลังจากที่ระบบเริ่มทำงานไปแล้ว โดย Dynamic Load-balancing ประกอบไปด้วยส่วนสำคัญ 2 ส่วน [2] ได้แก่ Information Policy และ Location Policy

Information policy:

เป็นส่วนที่ใช้ในการเก็บข้อมูลเพื่อนำไปให้ส่วนการตัดสินใจประมวลผล โดยInformation policy มีวิธีเก็บข้อมูลได้หลายแบบ เช่น เก็บข้อมูลไว้ที่แต่ละเครื่องเมื่อต้องมีการตัดสินใจจะทำการเรียกข้อมูลจากแต่ละเครื่องมาแล้วทำการตัดสินใจส่งงาน หรือ มีการส่งข้อมูลไปเก็บยัง LB ทุกๆ p วินาที เป็นต้น

Location Policy:

ส่วนนี้เป็นส่วนที่ทำการตัดสินใจว่าจะส่งงานชิ้นใหม่ที่ได้รับไปทำงานที่เซิร์ฟเวอร์ใด โดยเราจะนำข้อมูลของภาระงานที่เก็บไว้โดย Information policy มาประมวลผลโดยใช้อัลกอริทึมที่เลือกแล้ว เมื่อได้ผลลัพธ์แล้วว่าเซิร์ฟเวอร์เครื่องใดจะได้รับงานนี้ก็จะทำการส่งต่องานไปยังเซิร์ฟเวอร์นั้นๆทันที

3.2 ตัวอย่าง Load Balancing Algorithms

• **Global** – ทุกๆ p วินาที จะมีเซิร์ฟเวอร์ 1 ตัวที่ทำหน้าที่เป็น Load Information Center (LIC) ทำหน้าที่ในการรับโหลด จากเซิร์ฟเวอร์ตัวอื่น และจะทำการนำไปรวมกันใน Load Vector ที่จะทำการบรอดแคสต์ไปยังเซิร์ฟเวอร์ตัวอื่นๆ แต่ถ้าภาระงานของเซิร์ฟเวอร์ไม่มีการเปลี่ยนแปลงจากครั้งล่าสุด ก็ไม่ต้องส่งไปยัง LIC

• **Disted** - แทนที่จะรายงานภาระงานของตนเองไปยัง LIC เหมือนกับอัลกอริทึม GLOBAL แต่ละเซิร์ฟเวอร์จะทำการบรอดแคสต์ภาระงานของตนเองไปยังเซิร์ฟเวอร์ตัวอื่นเพื่อทำการ Update Load Vector ในแต่ละเครื่อง โดย Location Policy ของ อัลกอริทึมแบบ GLOBAL และ DISTED เป็นดังนี้เซิร์ฟเวอร์ที่รับผิดชอบงานอยู่จะทำการค้นหาเซิร์ฟเวอร์ที่มีภาระงานน้อยที่สุด ถ้าหากว่าเซิร์ฟเวอร์ที่ค้นหานั้นมีภาระงานต่ำกว่าเซิร์ฟเวอร์ปัจจุบันเกินค่า Threshold ที่ตั้งไว้เซิร์ฟเวอร์จะทำการส่งงานไปยังเซิร์ฟเวอร์นั้น

• **Central** - 2 อัลกอริทึมด้านบน placement decision ถูกสร้างโดยแต่ละเซิร์ฟเวอร์โดยใช้ load vector ที่อยู่บนเครื่องของตนเอง แต่ในอัลกอริทึมแบบ CENTRAL นั้น LIC จะทำหน้าที่เหมือนกับ

• **Central** - 2 อัลกอริทึมด้านบน placement decision ถูกสร้างโดยแต่ละเซิร์ฟเวอร์โดยใช้ load vector ที่อยู่บนเครื่องของตนเอง แต่ในอัลกอริทึมแบบ CENTRAL นั้น LIC จะทำหน้าที่เหมือนกับผู้จัดการส่วนกลางให้กับทุกเซิร์ฟเวอร์ เมื่อเซิร์ฟเวอร์ตัดสินใจว่าจะให้งานนี้เข้าไปสู่การกระจายงาน จะทำการส่ง request ไปยัง LIC พร้อมกับค่าภาระงานของตนเอง จากนั้นLIC จะเลือกเซิร์ฟเวอร์ที่มีคิวสั้นที่สุดและบอกไปยังเซิร์ฟเวอร์เริ่มต้นว่าจะส่งงานไปที่ใด ในขณะเดียวกันก็จะเพิ่มค่าภาระงานที่เซิร์ฟเวอร์ที่เราจะส่งงานเข้าไปด้วย 1

สำหรับ 3 อัลกอริทึมด้านบนจะถือว่า Placement decision maker จะรู้ภาระงานของเซิร์ฟเวอร์ทุกตัว ถึงแม้จะมี Delay บ้าง อัลกอริทึมด้านล่างต่อไปนี้ใช้ข้อมูลของระบบน้อยลง และจะมีโอเวอร์เฮดที่น้อยลง

• **Random** - อัลกอริทึมนี้ใช้เฉพาะข้อมูลภาระงานของระบบตัวเองเท่านั้น เมื่อพิจารณาแล้วว่าต้องกระจายงานนี้ก็จะกระจายงานออกไปยังเซิร์ฟเวอร์อื่นแบบ Random

• **Round robin** – เป็นอัลกอริทึมเบื้องต้นที่กระจายงานที่มีการร้องขอเข้ามาใหม่ไปยังเซิร์ฟเวอร์ที่มีทำงานอยู่เป็นเครื่องถัดไปโดยอัลกอริทึมแบบ round robin นี้จะเหมาะสมกับเซิร์ฟเวอร์ที่มีความสามารถในการประมวลผลที่เท่ากันเท่านั้น

• **Weighted round robin with response-time as weight** – เป็นการพัฒนาอัลกอริทึมแบบ Round robin โดยยการนำเวลาการตอบสนองของแต่ละเซิร์ฟเวอร์มาใช้ในการถ่วงน้ำหนักดูว่าเซิร์ฟเวอร์เครื่องใดเหมาะสมจะรับคำร้องขอนี้

• **Thrhld** - จำนวนของเซิร์ฟเวอร์ที่ Random เลือกขึ้นมาที่ไม่เกิน L_p จะถูกหามาเมื่อมีงานที่ต้องการการกระจาย และงานจะถูกส่งไปยังเซิร์ฟเวอร์ที่มี Load Threshold T_l ถ้าไม่มีเซิร์ฟเวอร์ที่เหมาะสมงานก็จะได้รับการประมวลผลที่เซิร์ฟเวอร์ปัจจุบัน

• **Lowest** - อัลกอรึทึมนี้จะคล้ายกับ Thrhld แต่ที่ต่างกันคือแทนที่จะใช้ Threshold ในการพิจารณาความเหมาะสม เซิร์ฟเวอร์ที่อยู่ใน L_p ที่ถูกเรียกมาทำงานน้อยที่สุดจะได้รับการจ่ายงาน

• **Average** - เมื่อมีงานเข้ามาจะดูว่าภาระงานของเซิร์ฟเวอร์ตัวเองที่มีอยู่มีค่ามากเกินค่างานเฉลี่ยของระบบหรือไม่ ถ้าไม่จะทำงานชิ้นใหม่นี้เอง แต่ถ้ามีค่ามากกว่าค่าเฉลี่ยจะทำการส่งงานนี้ไปยังเครื่องที่มีภาระงานน้อยที่สุดแทน

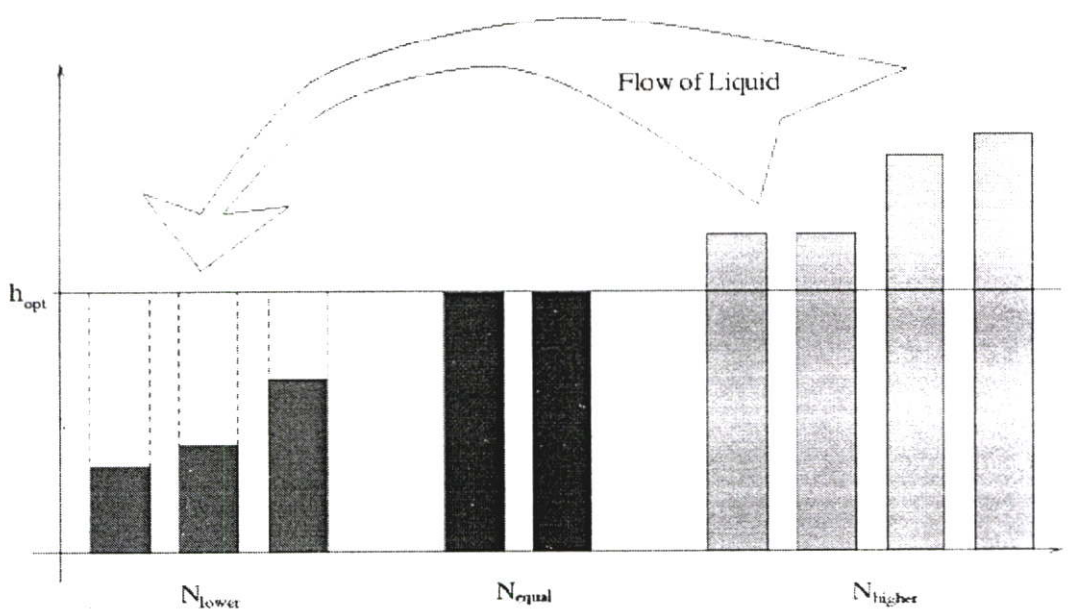
• **Fewest connections with limits** – อัลกอริทึมนี้จะรับคำร้องข้อมูลไว้ว่าขณะนี้เซิร์ฟเวอร์แต่ละเครื่องมีการติดต่อจำนวนเท่าไรที่กำลังประมวลผลอยู่ โดยเซิร์ฟเวอร์ที่มีจำนวนการติดต่อน้อยที่สุดจะเป็นเซิร์ฟเวอร์ที่จะทำงานการร้องขอครั้งถัดไป

• **Diffusive [11]** – อัลกอริทึมนี้เมื่อมีงานเข้ามาที่เซิร์ฟเวอร์ของตนจะทำการส่งค่าภาระงานของตนซึ่งก็คือจำนวนงานที่กำลังประมวลผลอยู่ แบบบรอดคาสต์ไปยังเซิร์ฟเวอร์ที่อยู่รอบข้าง จากนั้นเซิร์ฟเวอร์แรกที่มีค่าภาระงานน้อยกว่าตนเองจะรับงานนี้ไปประมวลผล โดยมีการทำงานเบื้องต้นดังรูปที่ 3.2

```
Algorithm Diffuse:
    (1)  for iteration ← 1 to ∞ begin
    (2)   All processors I parbegin
    (3)      load[i] ← number of tasks at i
    (4)      Broadcast load[i] to all neighbors
    (5)      For each j that is i's neighbor begin
    (6)          If load[i] > load[j] then
    (7)              Send Pij(load[i] – load[j]) task to j
    (8)      end
    (9)   parend
    (10) end
```

รูปที่ 3.2 การทำงานเบื้องต้นของอัลกอริทึมแบบ Diffusive

• **Hydrodynamic [10]** - อัลกอริทึมนี้จะทำการมองภาพโดยรวมของระบบแล้วทำการหาค่าภาระงานกลางที่เหมาะสมขึ้นมา จากนั้นเซิร์ฟเวอร์เครื่องใดที่มีภาระงานเกินค่านี้จะต้องทำการส่งงานที่ตนเองประมวลผลอยู่ไปยังเซิร์ฟเวอร์ที่มีภาระงานต่ำกว่าค่าภาระงานกลาง



รูปที่ 3.3 ภาพรวมของอัลกอริทึมแบบ Hydrodynamic

• **Credit-Base** - ทำการกำหนดค่าตัวเลข โดยการเรียกว่าเครดิตไปยังทุกๆเซิร์ฟเวอร์ โดยเครดิตจะบอกแนวโน้มว่าเซิร์ฟเวอร์นี้เหมาะกับงานชิ้นนี้หรือไม่โดยค่าเครดิตจะเพิ่มขึ้นด้วยปัจจัยต่อไปนี้

- เมื่อภาระงานของเซิร์ฟเวอร์ลดลง

- เมื่องานชิ้นนั้นมีความต้องการที่จะได้รับการประมวลผลด้วยเซิร์ฟเวอร์นี้เป็นพิเศษ เช่น หน่วยประมวลผลชนิดพิเศษ, อุปกรณ์ต่อพ่วงภายนอก หรือข้อมูลขนาดใหญ่ที่ต้องใช้อยู่ที่เซิร์ฟเวอร์นี้เป็นต้น

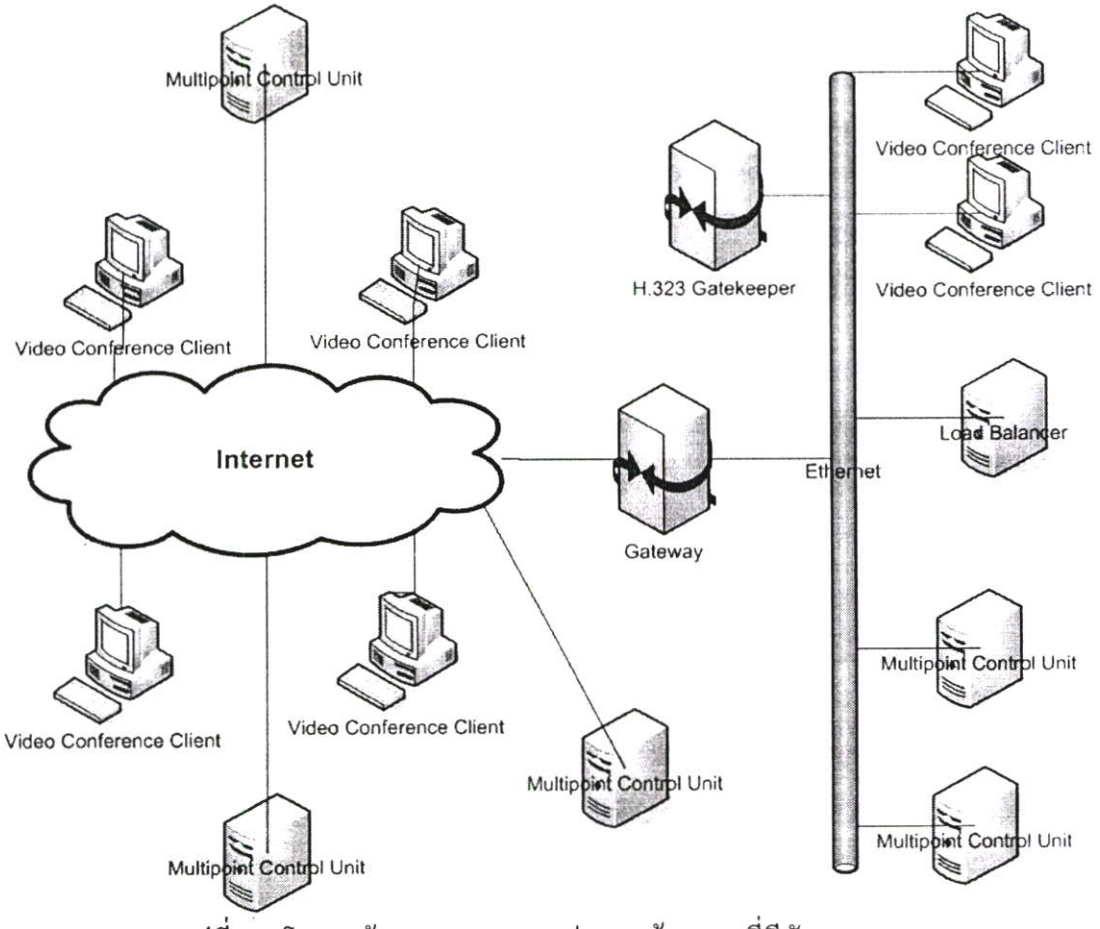
ในทางกลับกันค่าเครดิตจากลดลงจากปัจจัยดังต่อไปนี้

- เมื่อภาระงานของเซิร์ฟเวอร์เพิ่มขึ้น
- เมื่องานชิ้นนั้นมีความอิสระจากเซิร์ฟเวอร์เครื่องใดๆคือสามารถประมวลผลที่เครื่องใดๆ ก็ได้ และการย้ายเครื่องประมวลผลสามารถทำได้ง่าย

บทที่ 4

Load Balancing Algorithms ที่นำเสนอ

4.1 กล่าวนำ



รูปที่ 4.1 โครงสร้างของระบบการประชุมด้วยภาพที่มีตัวกระจายงาน

4.2 องค์ประกอบของอัลกอริทึม

เมื่อ Load Balancer (LB) ได้รับการร้องขอการสร้างห้องประชุมจากผู้ใช้จะต้องทำการพิจารณาหา MCU ที่เหมาะสมที่สุดในการบริการ จากบทที่ 3 อัลกอริทึมการกระจายงานจะประกอบไปด้วย 2 ส่วนหลักๆ นั่นคือ Information policy และ Placement policy โดยเราทำการนำเสนออัลกอริทึมของเราดังต่อไปนี้

4.2.1) Information policy:

ส่วนนี้จะเป็นส่วนที่คอยเก็บข้อมูลเกี่ยวกับภาระงานโดยเราทำการเก็บข้อมูลของภาระงานไว้ที่ LB โดยกำหนดให้ MCU ทุกตัวส่งค่าภาระงานมาที่ LB ตามเวลาที่กำหนด เพื่อให้เวลามีผู้ใช้ร้องขอ

การเปิดห้องประชุมใหม่ขึ้นมา LB จะได้สามารถตัดสินใจเลือก MCU มารองรับห้องประชุมได้ทันทีเพื่อลดเวลาการตอบสนองต่อผู้ใช้ในขั้นตอนการสร้างห้องประชุม เพราะข้อมูลของภาระงานมีขนาดที่เล็กมากเมื่อเปรียบเทียบกับขนาดของข้อมูลในการประชุมด้วยภาพ () การส่งค่าภาระงานจึงกระทบต่อทรัพยากรของระบบน้อย

หน้าที่ของ MCU คือการ encode, decode และ mix เสียงและภาพ และการจัดการบริหารห้องประชุม การจัดการเกี่ยวกับการติดต่อและส่งข้อมูล จะเห็นได้ว่าภาระงานของ MCU นั้นไม่เพียงมีแต่ภาระงานทางด้านการประมวลผลเท่านั้น หากแต่เรายังต้องคำนึงถึงทรัพยากรทางด้านเครือข่ายอีกด้วย

เราทำการกำหนดค่าขึ้นมา 1 ค่า เรียกว่า เครดิต เพื่อเป็นค่าที่จะใช้ในการตัดสินใจของ placement policy โดยเครดิตนั้นจะเป็นการรวมกันระหว่างค่าภาระงานของการประมวลผลและทรัพยากรของเครือข่ายของแต่ละ MCU โดยค่าเครดิตจะทำการคำนวณและเก็บไว้ที่ LB โดยค่าเครดิตจะเป็นตามสมการที่ 1

$$C_{MCU} = C_{CP} + C_{NW} \tag{1}$$

C_{MCU} = MCU Credit

C_{CP} = Computation Credit

C_{NW} = Communication Credit

A) Computation credit คือ การรวมกันของเปอร์เซ็นต์ที่เหลืออยู่ของหน่วยประมวลผลกลาง และหน่วยความจำหลักของ MCU เพราะหน้าที่ของ MCU คือการ encode, decode และ mix เสียงและภาพ ซึ่งงานส่วนนี้ของ MCU จะใช้ทรัพยากรของหน่วยความจำ และหน่วยประมวลผลกลาง เนื่องจากระบบการประชุมด้วยภาพนั้น MCU แต่ละตัวจะมีทรัพยากรและสิ่งแวดล้อมที่แตกต่างกัน เพราะฉะนั้นจึงต้องมีการถ่วงน้ำหนักให้กับ หน่วยประมวลผลกลางและหน่วยความจำ เพื่อให้ค่าเปอร์เซ็นต์ที่นำมาใช้ในการคำนวณเครดิตมีความสมดุลต่อความสามารถและความจุของ MCU แต่ละตัว โดยค่าถ่วงน้ำหนักของหน่วยประมวลผลกลางกลางหาได้ดังนี้

เราให้ MCU_j แทน MCU หนึ่งตัวในระบบ จากนั้นเรากำหนดค่าตัวแปร P เพื่อใช้ในการแทนจำนวน MCU ทั้งหมดในระบบ โดยค่าถ่วงน้ำหนักของหน่วยประมวลผลกลางจะมีความสัมพันธ์กับความเร็วของการประมวลผลของ MCU ที่มีความเร็วของหน่วยประมวลผลกลางสูงที่สุด โดยค่าถ่วงน้ำหนักของหน่วยประมวลผลจะมีค่าน้อยกว่าหรือเท่ากับ 1 โดยค่าถ่วงน้ำหนักของหน่วยประมวลผลกลางจะเป็นการนำความเร็วของหน่วยประมวลผลกลางโดยมีหน่วยการวัดคือล้านคำสั่งต่อวินาที (MIPS) ถ้า $V_{cpu}(j)$ เป็นความเร็วของหน่วยประมวลผลกลางในหน่วย MIPS โดย j = 1, 2, 3,..., P ค่าถ่วงน้ำหนักของหน่วยประมวลผลกลางจะเป็นดังสมการด้านล่าง

$$C_{CP} = f_{CP}(W_{cpu}A_{cpu}, W_{mem}A_{mem}) \quad (2)$$

A_{cpu} =Available Central Processor Unit (CPU) (%)

A_{mem} = Available Memory (%)

W_{cpu} = Weight number for CPU load

W_{mem} = Weight number for Memory used

f_{CP} = Computation Weight Function

$$W_{cpu}(j) = \frac{V_{cpu}(j)}{Max_{i=1}^{P} V_{cpu}(i)} \quad (3)$$

เช่นเดียวกับค่าถ่วงน้ำหนักของหน่วยความจำสามารถคำนวณได้จากการเปรียบเทียบขนาดของหน่วยความจำระหว่าง MCU ในระบบ ดังนี้

$$W_{mem}(j) = \frac{Mem_j}{Max_{i=1}^{P} Mem_i} \quad (4)$$

โดยที่ Mem_j คือจำนวนของเนื้อที่ที่เหลืออยู่ของหน่วยความจำในหน่วยความจำในหน่วย Megabytes บน MCU_j โดย j = 1, …, P.

B) Communication credit คือ แบนด์วิดท์ที่ยังหลืออยู่และดีเลย์ระหว่างผู้ใช้กับ MCU เพราะแต่ละ MCU มีสภาพแวดล้อมทางเครือข่ายที่ไม่เหมือนกัน และ เส้นทางระหว่างผู้ใช้กับ MCU แต่ละเครื่องก็ไม่เหมือนกันซึ่งส่งผลให้ดีเลย์ที่นำมาคิดไม่เท่ากันด้วย

$$C_{NW} = W_{bw}A_{bw} + T \quad (5)$$

A_{bw} = Bandwidth usage (%)

T = Percentage of delay time between MCU and client (%)

W_{bw} = Weight number for bandwidth used

เช่นเดียวกับหน่วยประมวลผลกลางและหน่วยความจำ แบนด์วิดท์ก็เช่นกันที่ต้องมีการถ่วงน้ำหนักโดยเทียบกับแบนด์วิดท์ที่มากที่สุดที่มีของ MCU ในระบบ

$$W_{bw}(j) = \frac{BW_j}{Max_{i=1}^{P} BW_i} \quad (6)$$

การประชุมด้วยภาพนั้นค่าดีเลย์ที่ยอมรับได้นั้นอยู่ระหว่าง 150 - 300 ms [8] ดังนั้นถ้าหากดีเลย์ระหว่าง MCU ไปยังผู้ใช้มีค่ามากกว่า 300 ms LB จะไม่ทำการเลือก MCU นั้นในการบริการ โดยการคิดค่าเปอร์เซ็นต์ดีเลย์นั้นจะคิดจากดีเลย์ที่สูงที่สุดในหน่วย ms

$$T = \begin{cases} (300 - delaytime^{*}) \times 0.33 & delaytime \le 300ms \\ -\infty & delaytime > 300ms \end{cases} \qquad (7)$$

* Size of test packet = 64 kilobyte (Maximum size of ping command)

4.2.2) Location Policy:

เราจะทำการกระจายภาระงานในกรณีที่มีการร้องขอการสร้างห้องประชุมใหม่เท่านั้น เพราะถ้าหากเราทำการตรวจสอบภาระงานของ MCU ทุกตัวในทุกๆ q วินาทีแล้วทำการกระจายงานใหม่จะส่งผลกระทบโดยตรงต่อผู้ใช้ที่อยู่ภายในห้องประชุมเนื่องจากการสื่อสารจะต้องชะงักไปช่วงเวลาหนึ่งอันเกิดจากการต้องทำการ redirect ไปยัง MCU ตัวอื่น และอาจจะเกิดการหลุดของผู้ใช้บางคนได้

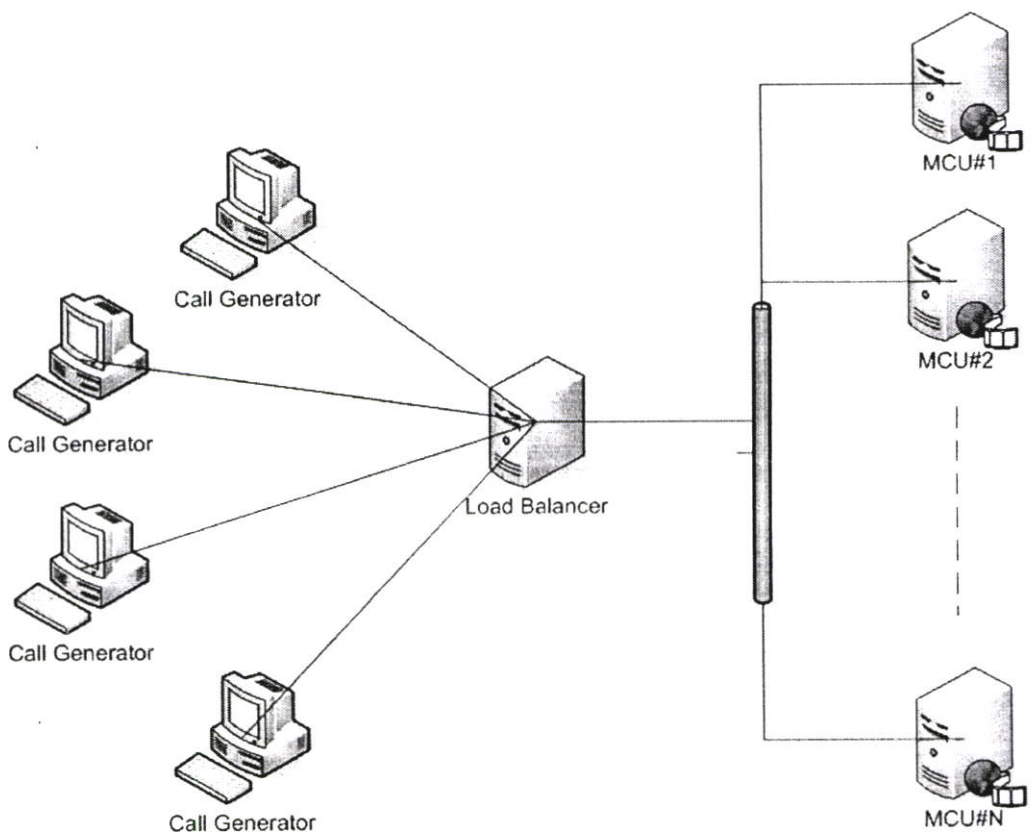
เมื่อมีการร้องขอการสร้างห้องประชุมใหม่ LB จะทำการคำนวณค่าเครดิตจากข้อมูลที่เก็บไว้โดย Information policy และทำการเลือก MCU ที่มีค่าเครดิตมากที่สุดเข้ามาจัดการห้องประชุมห้องนี้

บทที่ 5

การทดลองและผลการทดลอง

5.1 องค์ประกอบการทดลอง

การทดลองของเราจะประกอบไปด้วยส่วนต่างๆดังนี้ Load Balancer (LB), Multipoint Control Unit (MCU) และ Call Generator

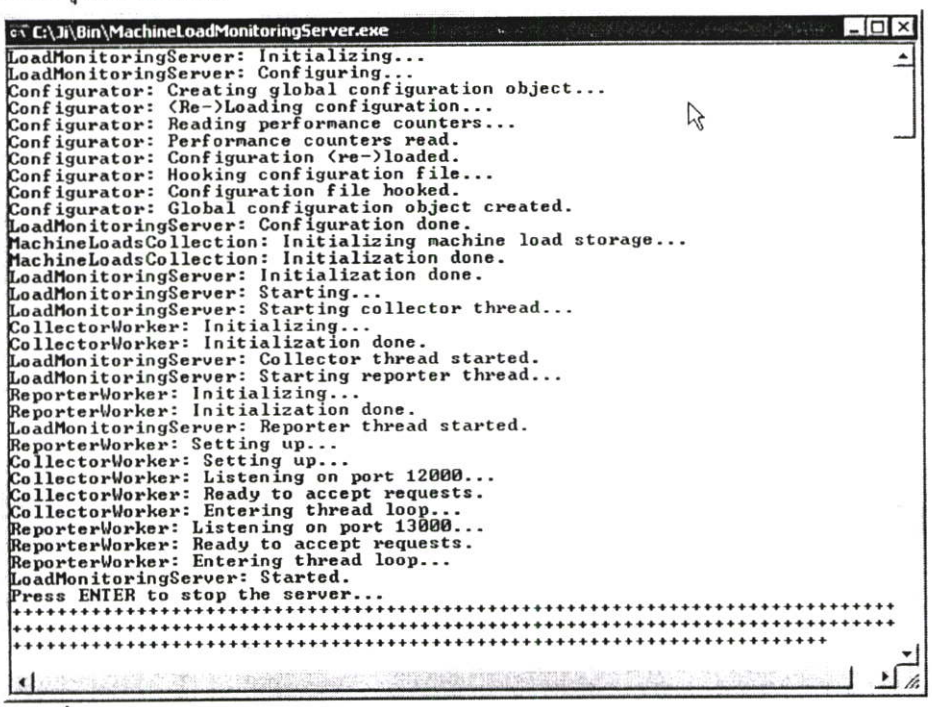


รูปที่ 5.1 โครงสร้างการทดลอง

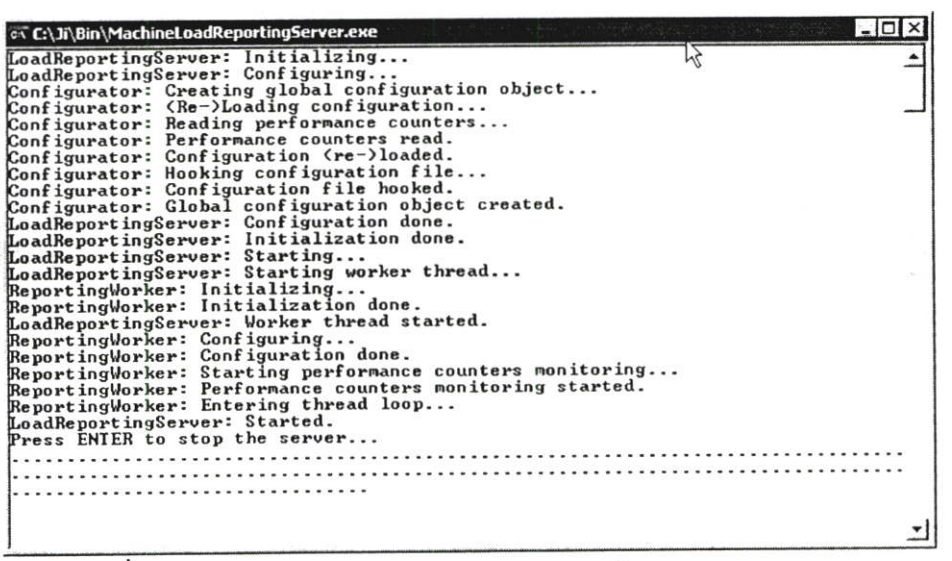
5.1.1)Load Balancer(LB)

ตัวกระจายงาน (LB) เป็นโอเพนซอร์สแอพพลิเคชั่น จาก www.codeproject.com โดยเราทำการเพิ่มความสามารถให้สามารถเก็บข้อมูลเกี่ยวกับหน่วยความจำ และแบนด์วิดท์ได้ โดยแอพพลิเคชั่นนี้จะทำการคำนวณเครดิตของการประมวลผลและการติดต่อสื่อสาร โดยเราทำการพัฒนา LB ด้วย Microsoft Visual C++. NET 2003 และ Winpcap library version 3.01 โดยโดยที่ MCU แต่ละตัวจะมี LoadReportingServer ทำการส่งข้อมูลเกี่ยวกับ การใช้ CPU, หน่วยความจำ และ แบนด์วิดท์มายัง LB จากนั้น LoadMonitoringServer ที่อยู่ที่ LB จะทำการเก็บข้อมูลไว้เตรียม

ใช้ในการตัดสินใจ ส่วนค่าดีเลย์ระหว่างผู้ใช้และ MCU นั้นจะสามารถหาได้ก่อเมื่อมีผู้ใช้ร้องขอการสร้างห้องประชุมแล้วเท่านั้น



รูปที่ 5.2 LoadMonitoringServer (อ่านค่าภาระงานและส่งไปยัง Load Balancer)



รูปที่ 5.3 LoadReportingServer (ส่งค่าภาระงานไปยัง Load Balancer)

5.1.2) Multipoint Conference Unit (MCU)

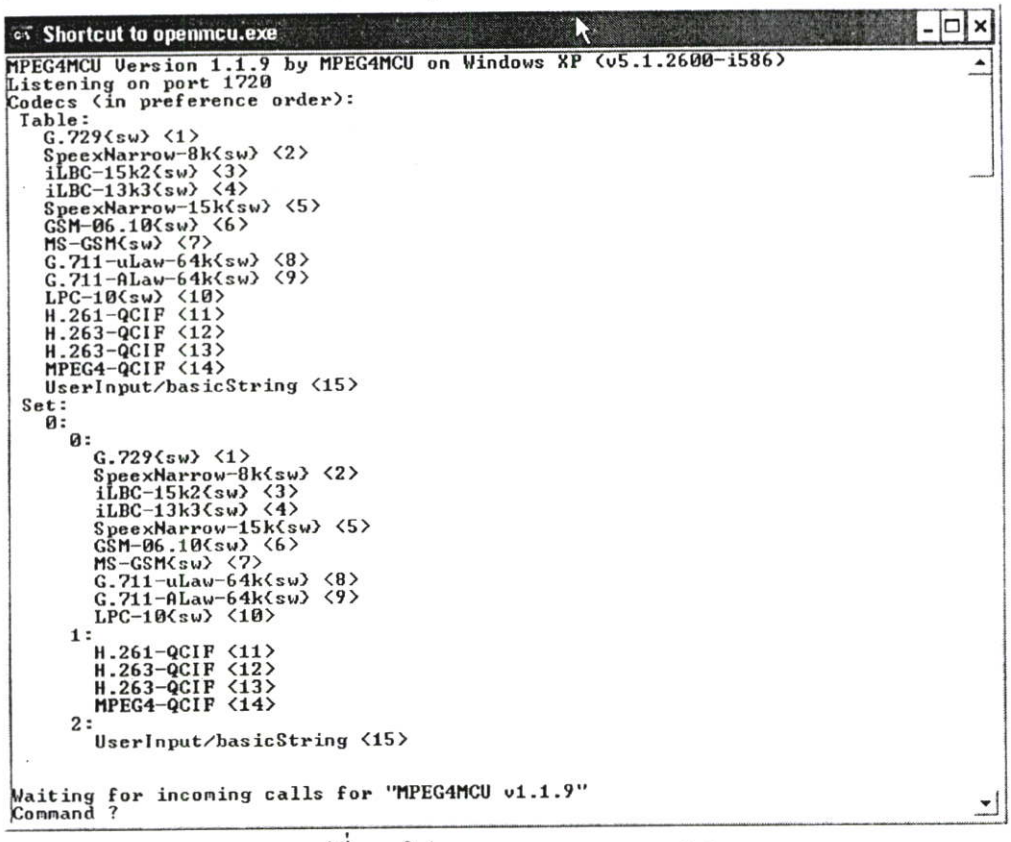
เราทำการลง MPEG4MCU จาก [1] โดยใช้ฟังก์ชั่นของ MPEG4MCU ที่ไม่ต้องมีเกทคีปเปอร์อยู่ในระบบ โดยลง MPEG4MCU บนเครื่อง 4 เครื่องดังต่อไปนี้:

1) MCU1 = Intel Pentium D CPU 3.0GHz, Memory = 512 MB และ Bandwidth = 100 Mbps

2) MCU2 = Intel Pentium 4 CPU 2.8 GHz, Memory = 512 MB และ Bandwidth = 10 Mbps

3) MCU3 = Intel Pentium M CPU 1.4 GHz, Memory = 768 MB และ Bandwidth = 100 Mbps

4) MCU4 = Intel Celeron 2.6 GHz, Memory = 384 MB และ Bandwidth = 100 Mbps



รูปที่ 5.4 โปรแกรม MPEG4MCU[1]

5.1.3) Call generator

เราใช้ Call generator จาก [1] โดย Call generator นี้สามารถรับและส่งได้ทั้งเสียงและภาพ โดยเรากำหนดตัวแปรของการทำงานไว้ดังนี้:

Video codec = MPEG-4

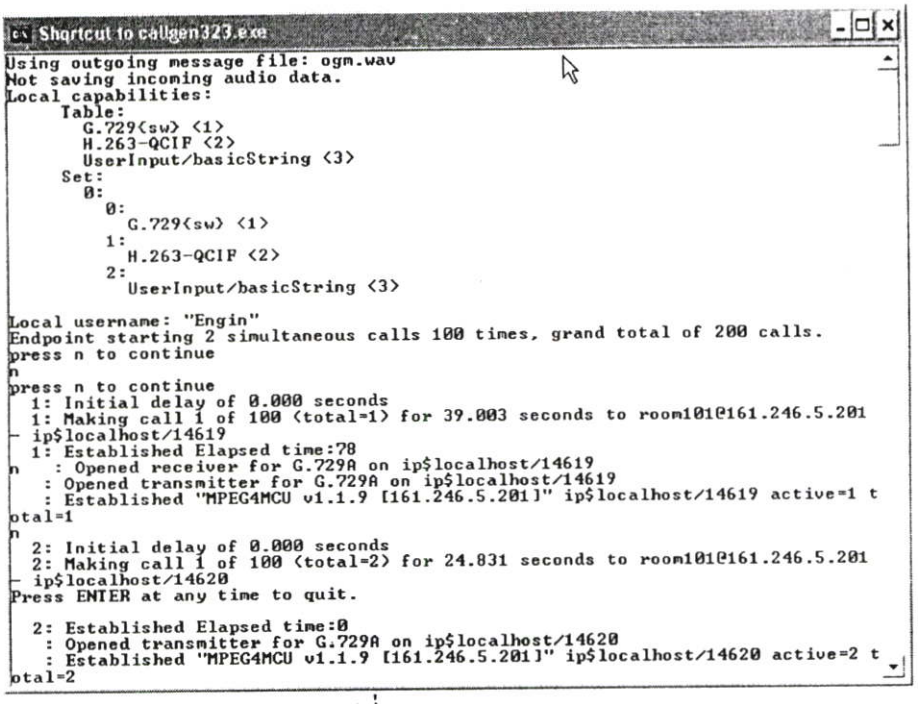
Frame rate = 10 Frames/sec

Audio codec = G.711-uLaw-64k (PCM)

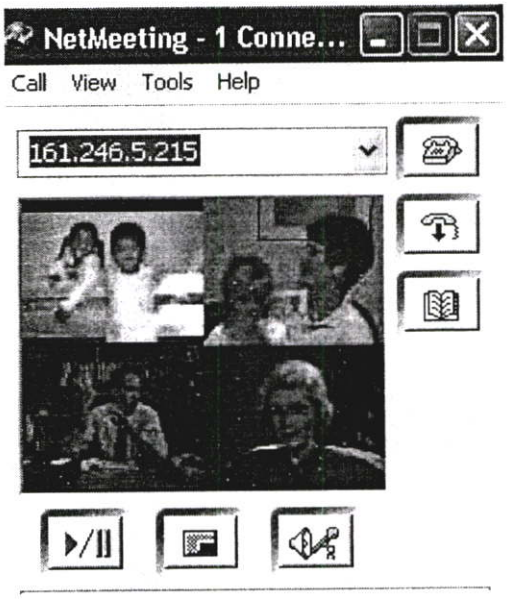
Audio Sampling rate = 8 kHz

Users per room = 4 users per room

โดย Call generator นี้สามารถทำงานร่วมกับ MPEG4MCU ได้จริงและเราสามารถเข้าไปสังเกตุการณ์ห้องประชุมได้โดยการใช้โปรแกรม Netmeeting เข้าไปร่วมการสนทนาได้ด้วยดังรูปที่ 5.6



รูปที่ 5.5 Call generator



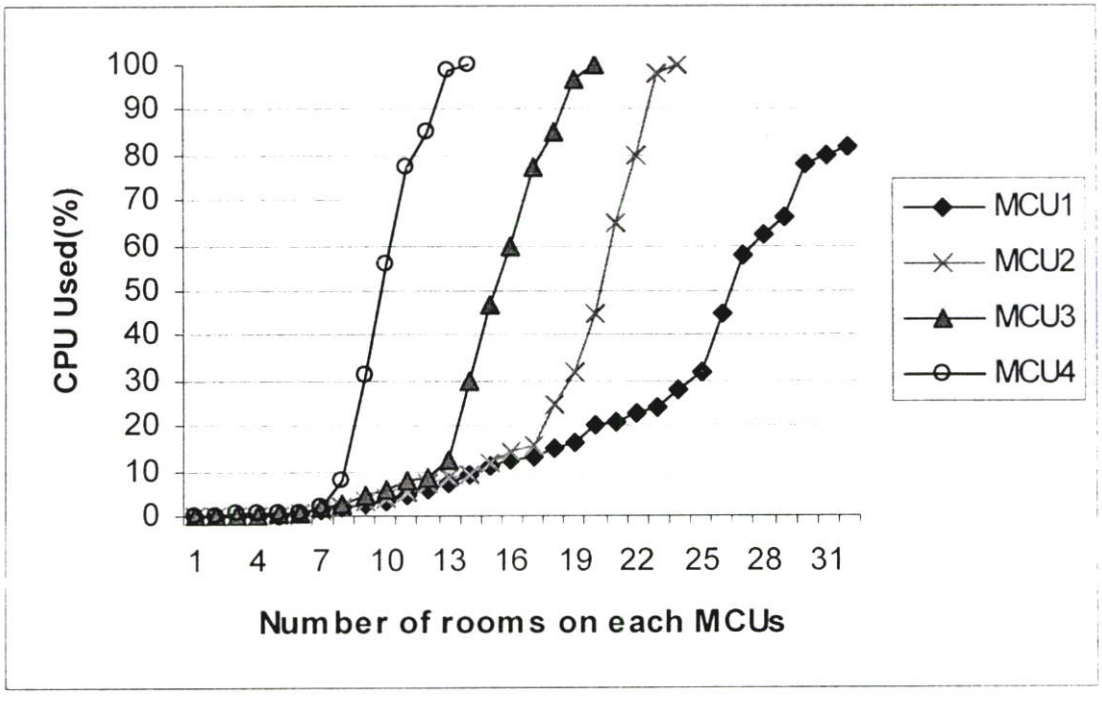
รูปที่ 5.6 การใช้งาน Call generator ร่วมกับ NetMeeting

5.2 ผลการทดลอง

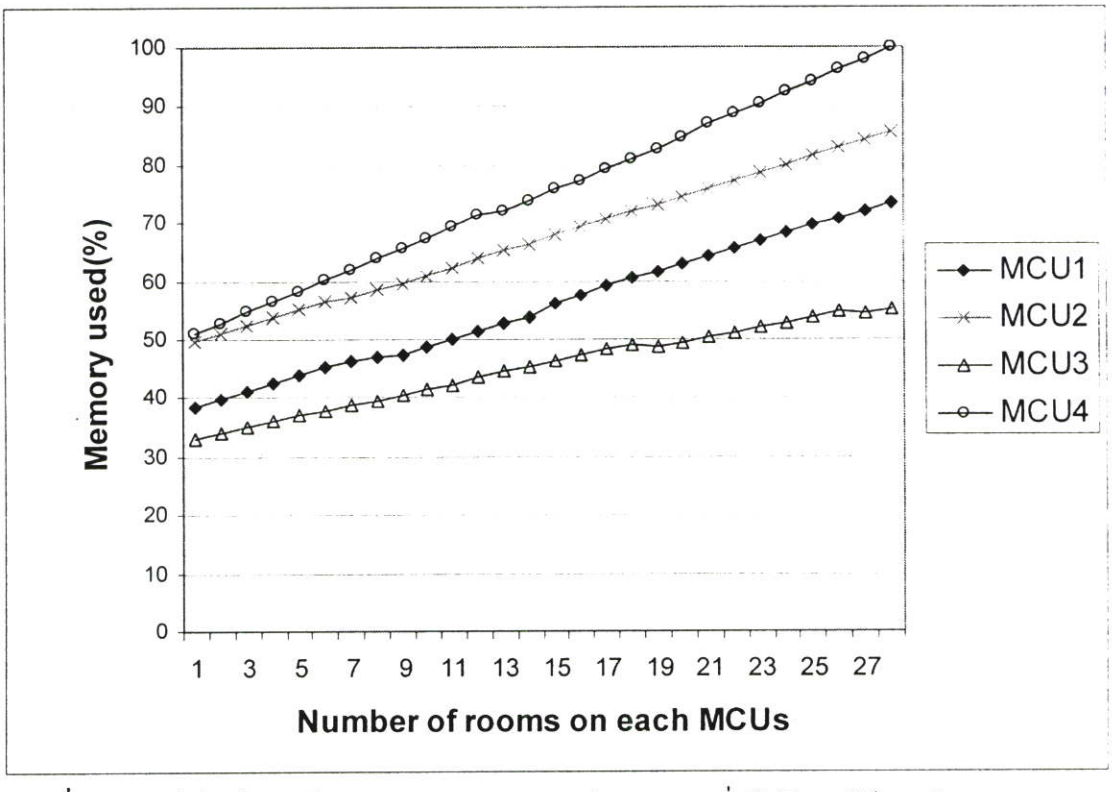
ในส่วนนี้จะแสดงถึงประสิทธิภาพของอัลกอริทึมของเราจากการทดลอง โดยจะแบ่งการทดลองออกเป็นส่วนย่อยๆ ดังต่อไปนี้

5.2.1) Without Load Balancer

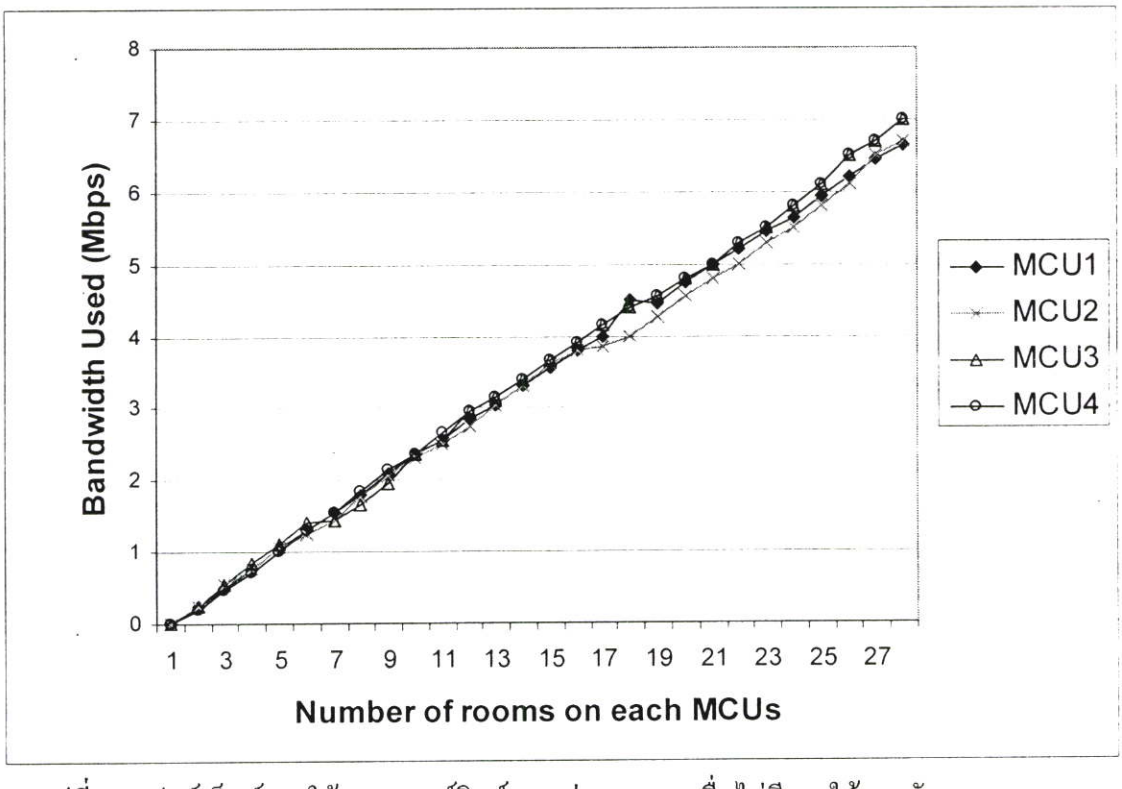
เราเริ่มต้นการทดลองโดยการไม่ใช้ LB เข้ามาเป็นการทดลองครั้งที่ 1 โดยเราทำการทดลองครั้งนี้เพื่อทำการดูแนวโน้มของการใช้ทรัพยากรของหน่วยประมวลผลกลาง, หน่วยความจำ และแบนด์วิดท์ของแต่ละ MCU โดยในรูปที่ 5.7 จะเห็นได้ว่าเมื่อจำนวนของห้องเพิ่มขึ้นในแต่ละ MCU โดยเราจะมาพิจารณาที่การใช้งานของหน่วยประมวลผลกลางก่อน จะสังเกตได้ว่าในช่วงแรกนั้นหน่วยประมวลผลกลางจะถูกใช้งานน้อยและมีแนวโน้มการเพิ่มขึ้นอย่างช้าๆ แต่เมื่อจำนวนห้องเพิ่มขึ้นมาจนถึงจุดวิกฤตของแต่ละ MCU จะเห็นได้ว่าแนวโน้มของการใช้หน่วยประมวลผลกลางจะเพิ่มขึ้นในเชิงของ exponential ในทางกลับกันจากรูปที่ 5.8 และ 5.9 แนวโน้มการเติบโตของการใช้งานหน่วยความจำและปริมาณแบนด์วิดท์ที่ถูกใช้ไปเป็นแบบ linear ดังนั้นการที่อัลกอริทึมของเราจะทำการคำนวณค่าเครดิตจากตัวแปรของ เปอร์เซ็นต์ของหน่วยประมวลผลกลางที่เหลืออยู่, ปริมาณหน่วยความจำ และ แบนด์วิดท์ที่เหลืออยู่ รวมถึงดีเลย์ระหว่างผู้ใช้กับ MCU จะไม่สามารถนำมารวมกันได้โดยตรง เพราะการเติบโตของการใช้ทรัพยากรต่างๆนั้นไม่เหมือนกันเราจึงต้องทำการสร้างฟังก์ชั่นการถ่วงน้ำหนักของตัวแปรด้านทรัพยากรขึ้น



รูปที่ 5.7 เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางบนแต่ละ MCU เมื่อไม่มีการใช้งานตัวกระจายงาน



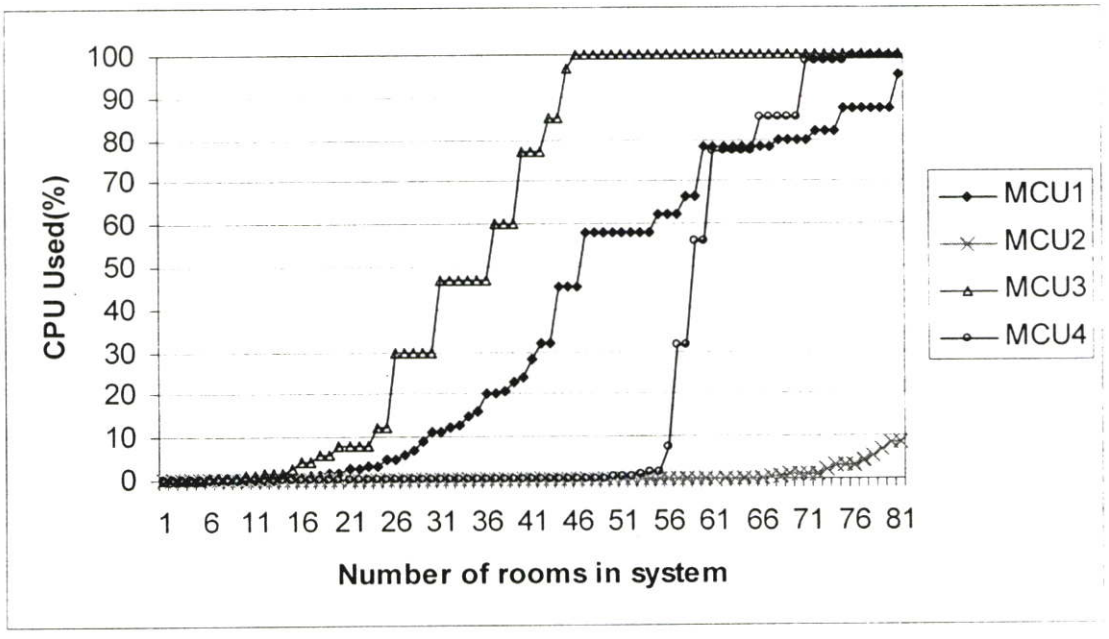
รูปที่ 5.8 เปอร์เซ็นต์การใช้งานหน่วยความจำบนแต่ละ MCU เมื่อไม่มีการใช้งานตัวกระจายงาน



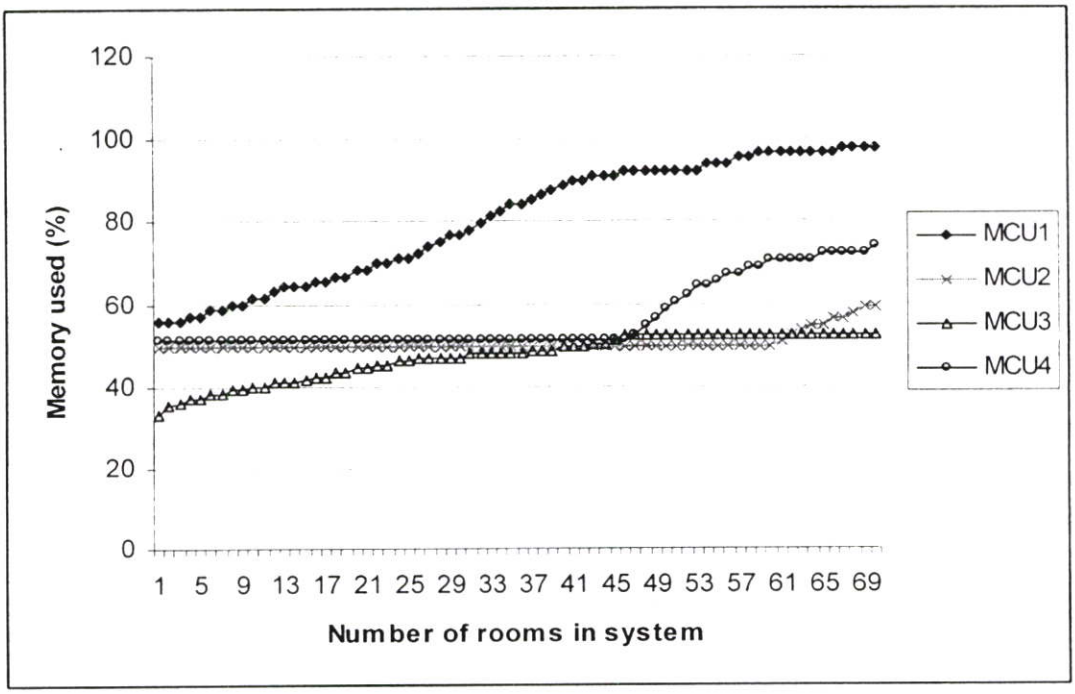
รูปที่ 5.9 เปอร์เซ็นต์การใช้งานแบนด์วิดท์บนแต่ละ MCU เมื่อไม่มีการใช้งานตัวกระจายงาน

5.2.2) With Load Balancer but no weighting function

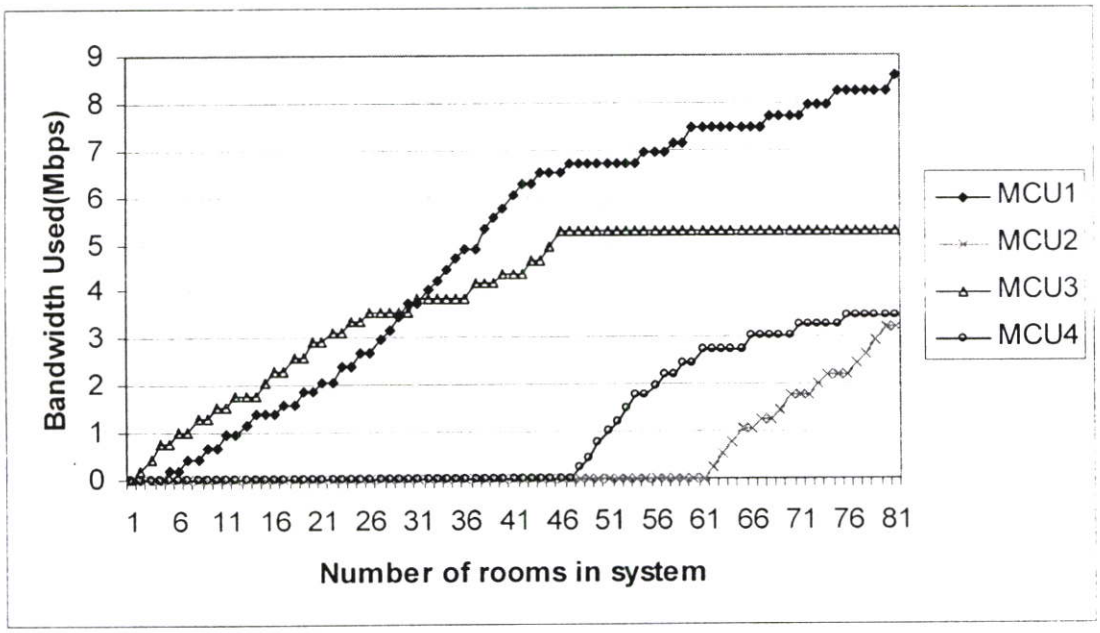
ขั้นที่2 เรานำเอา LB เข้ามาใส่ในการทดลองโดย LB ที่เราใช้นั้นจะไม่ใช้ weighting function แต่ว่าใช้การบวกกันโดยตรง โดยจะได้ผลดังรูปที่ 5.10 เมื่อเริ่มต้นการทดลอง LB จะทำการเลือก MCU1 และ MCU3 ขึ้นมาจัดการห้องประชุมที่เข้ามาในช่วงแรก เนื่องจาก MCU1 มีหน่วยประมวลผลกลางที่มีความเร็วสูงที่สุดและ MCU3 มีหน่วยความจำมากที่สุด MCU 2 ตัวนี้จึงถูกเลือกขึ้นมาให้บริการในช่วงแรก โดยจะเห็นได้ชัดจากรูปที่ 5.11 และ 5.12 MCU1 และ MCU3 จะรับห้องประชุมเข้ามาจัดการสลับกันไป จนกระทั่งมีห้องประชุมที่ 26 เข้ามาจะเห็นว่าเปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางของ MCU3 นั่นจะเพิ่มขึ้นอย่างเห็นได้ชัดแต่ LB ยังคงเลือก MCU3 ขึ้นมาใช้งานสลับกับ MCU1 อยู่เนื่องจากการที่ MCU3 มีปริมาณหน่วยความจำที่มากทำให้ค่าเปอร์เซ็นต์ของหน่วยความจำที่ยังเหลืออยู่เครดิตที่คำนวณมาได้ยังคงมากอยู่ และ MCU3 จะยังคงถูกเรียกใช้อยู่จนกระทั่งหน่วยประมวลผลกลางถูกใช้จนถึง 100%(เมื่อเปอร์เซ็นต์การใช้หน่วยประมวลผลกลางถึง 100 LB จะไม่เลือก MCU ตัวนั้นขึ้นมารับงานใหม่) จากนั้นเมื่อจำนวนห้องถึง 43 MCU4 จะเข้ามารับห้องประชุมใหม่เนื่องจากมีแบนด์วิดท์สูงกว่า MCU2 แต่ว่า MCU4 มีหน่วยประมวลผลกลางที่ช้าทำให้เปอร์เซ็นต์การใช้หน่วยประมวลผลกลางเติบโตขึ้นอย่างรวดเร็ว



รูปที่ 5.10 เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางบนแต่ละ MCU โดยระบบมีการใช้ LB แต่ไม่ใช้ฟังก์ชั่นถ่วงน้ำหนัก



รูปที่ 5.11 เปอร์เซ็นต์การใช้งานหน่วยความจำบนแต่ละ MCU โดยระบบมีการใช้ LB แต่ไม่ใช้ฟังก์ชั่นถ่วงน้ำหนัก



รูปที่ 5.12 เปอร์เซ็นต์การใช้งานแบนด์วิดท์บนแต่ละ MCU โดยระบบมีการใช้ LB แต่ไม่ใช้ฟังก์ชั่นถ่วงน้ำหนัก

5.2.3) With Load Balancer and weighting function

จากรูปที่ 5.10 จะเห็นว่าเมื่อมีห้องประชุมทั้งหมด 81 ห้อง เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางของ MCU1, MCU2 และ MCU3 เท่ากับ 100 แต่เปอร์เซ็นต์การใช้งานหน่วย

ประมวลผลกลางของ MCU2 นั้นมีเพียงแค่ 10 เท่านั้นซึ่งแสดงถึงความไม่สมดุลของการกระจายงานดังนั้นเราจึงต้องมีฟังก์ชั่นการถ่วงน้ำหนักเพิ่มเข้ามาใน LB โดยเราดูจากแนวโน้มการใช้ทรัพยากรที่ผ่านมาของ MCU จากการทดลองที่ผ่านมา เราจะเห็นว่าเปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางเพิ่มขึ้นเป็นเอ็กซ์โพเนนเชียลแต่เปอร์เซ็นต์การใช้งานหน่วยความจำและแบนด์วิดท์เพิ่มขึ้นแบบเส้นตรง เราจึงทำการยกกำลังสองให้กับ A_{cpu} เพื่อให้ ค่าเครดิตที่ LB คำนวณเหมาะสมกับความเป็นจริงที่เกิดขึ้นจากการทำงานจริงของ MCU โดยที่ (A_{cpu} = 100 - %CPU used).

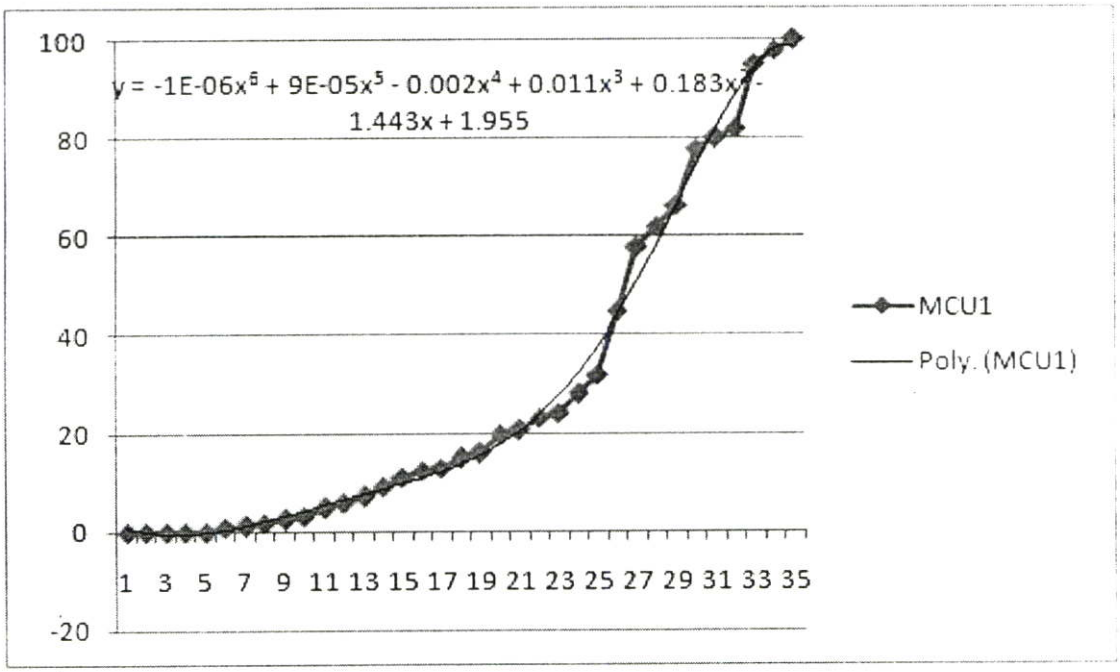
จากการทดลองที่ 5.2.1 เราได้ทำการใช้ fitting function เพื่อทำการหาสมการการใช้งานหน่วยประมวลผลกลาง ซึ่งได้ผลดังรูปที่....

$$MCU1: y = -1E^{-06}x^6 + 9E^{-05}x^5 - 0.002x^4 + 0.011x^3 + 0.183x^2 - 1.443x + 1.955 \quad (8)$$

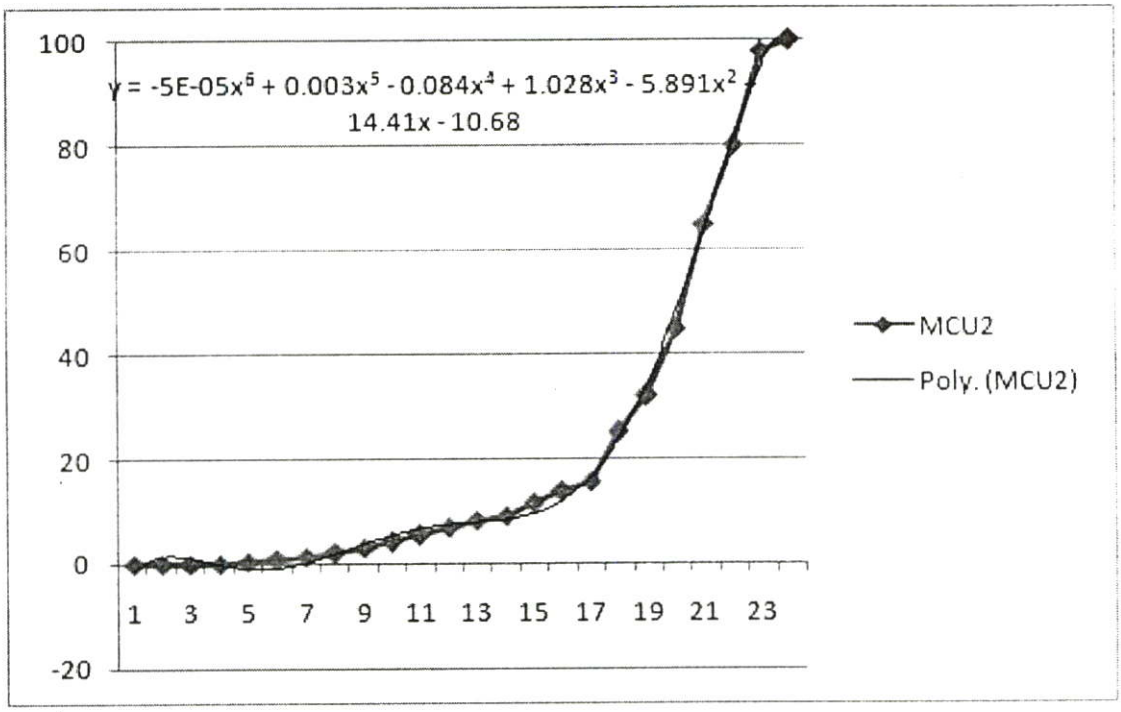
$$MCU2: y = -5E^{-05}x^6 + 0.003x^5 - 0.084x^4 + 1.028x^3 - 5.891x^2 + 14.41x - 10.68 \quad (9)$$

$$MCU3: y = 6E^{-09}x^6 - 0.000x^5 + 0.041x^4 - 0.646x^3 + 4.250x^2 - 11.05x + 8.311 \quad (10)$$

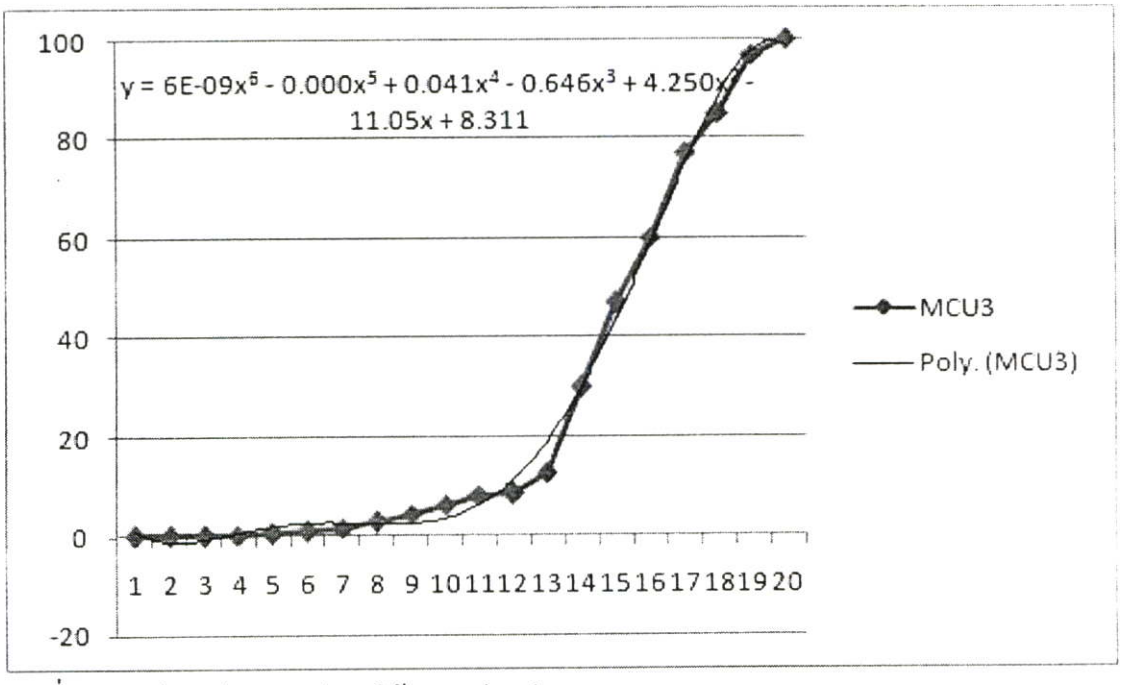
$$MCU4: y = 6E^{-09}x^6 - 0.000x^5 + 0.041x^4 - 0.646x^3 + 4.250x^2 - 11.05x + 8.311 \quad (11)$$



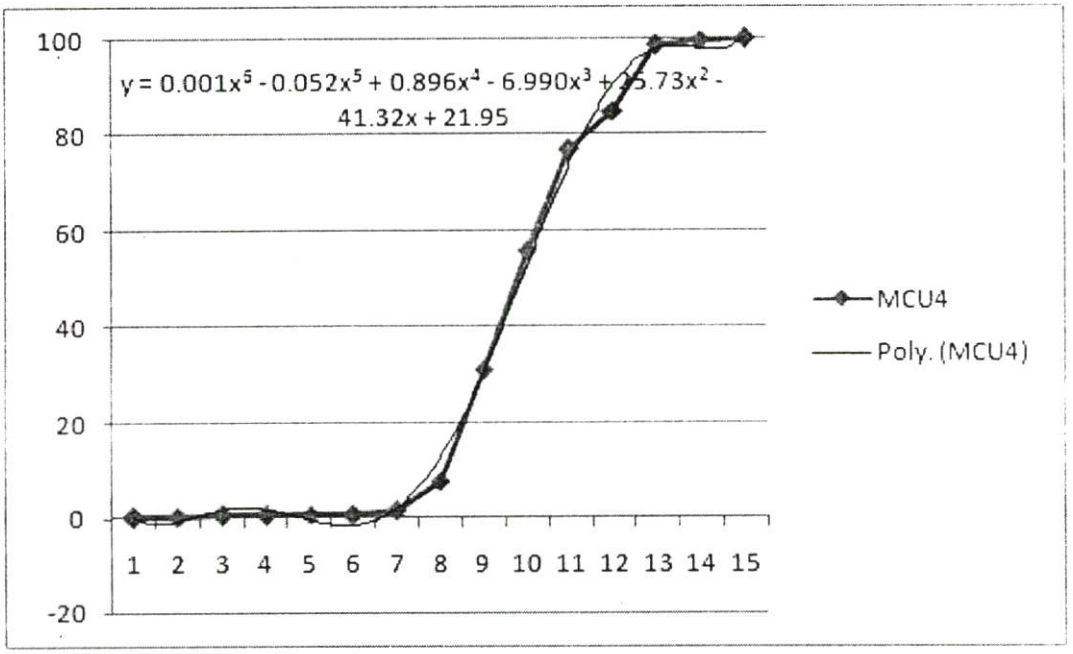
รูปที่ 5.13 เปรียบเทียบกราฟการใช้งานหน่วยประมวลผลกลางของ MCU1 จากการทดลอง 5.2.1 กับกราฟที่ได้จากสมการที่ 8



รูปที่ 5.14 เปรียบเทียบกราฟการใช้งานหน่วยประมวลผลกลางของ MCU2 จากการทดลอง 5.2.1 กับ กราฟที่ได้จากสมการที่ 9



รูปที่ 5.15 เปรียบเทียบกราฟการใช้งานหน่วยประมวลผลกลางของ MCU3 จากการทดลอง 5.2.1 กับ กราฟที่ได้จากสมการที่ 10



รูปที่ 5.16 เปรียบเทียบกราฟการใช้งานหน่วยประมวลผลกลางของ MCU4 จากการทดลอง 5.2.1 กับ กราฟที่ได้จากสมการที่ 11

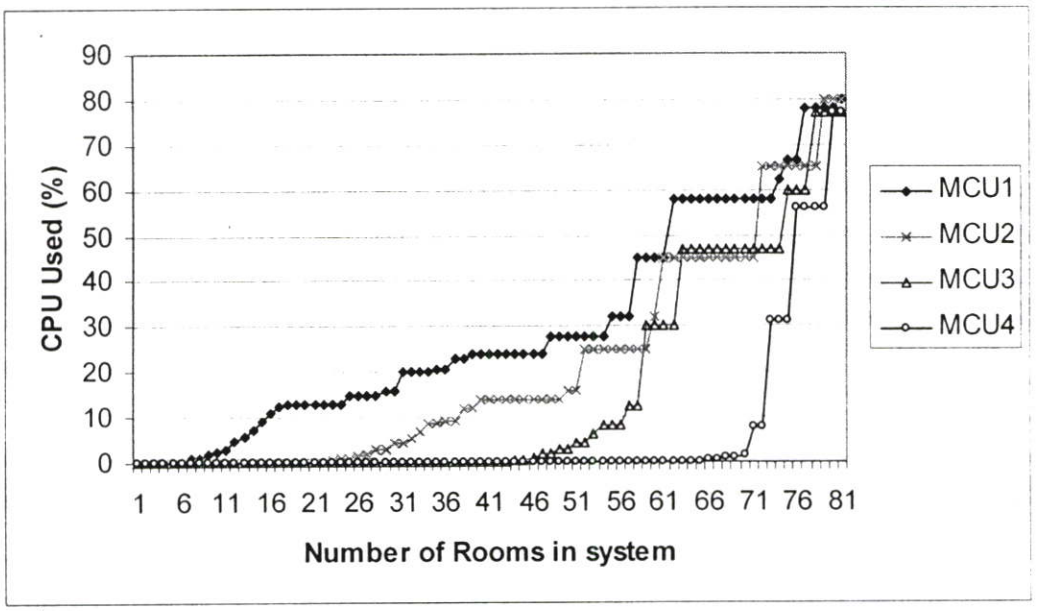
เมื่อพิจารณาจากสมการที่ 8 ถึง 11 จะพบว่าสัมประสิทธิ์ของพจน์กำลัง 6, 5, 4 และ 3 มีค่าน้อยมาก ทำให้เรากำหนดสมการถ่วงน้ำหนักดังสมการที่ 12

$$f_{CP}(W_{cpu}A_{cpu}, W_{mem}A_{mem}) = W_{cpu}A_{cpu}^2 + W_{mem}A_{mem} \quad (12)$$

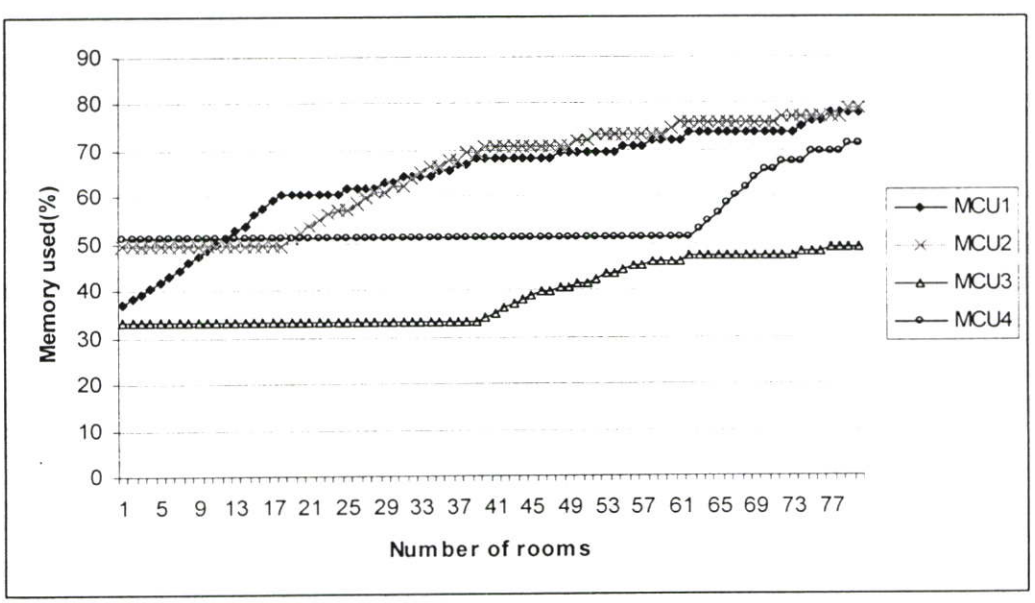
จากนั้นเราทำการนำเอาฟังก์ชั่นการถ่วงน้ำหนักตามสมการที่ 12 เข้าไปใช้ใน LB จากนั้นทำการทดลองอีกครั้งซึ่งได้ผลตามรูปที่ 5.17, 5.18 และ 5.19 เมื่อเราเริ่มทำการทดลองจนกระทั่งห้องประชุมมีจำนวนเท่ากับ 23 ห้องในระบบ จะเห็นได้ว่า LB ได้เลือกเพียงแค่ MCU1 ขึ้นมาทำงานเท่านั้นเนื่องจาก MCU1 นั้นมีหน่วยประมวลผลกลางที่มีความเร็วสูงที่สุดและฟังก์ชั่นถ่วงน้ำหนักของเราทำให้ LB ให้ความสำคัญกับหน่วยประมวลผลกลางมากกว่าตัวแปรอื่นๆ และในช่วงที่จำนวนห้องในระบบมีจำนวน 23 ถึง 41 ห้อง LB จะทำการเลือก MCU1 และ MCU2 ขึ้นมาสลับกันรับห้องประชุมไปบริหารเนื่องจาก MCU1 มีห้องประชุมในความดูแลอยู่จำนวนหนึ่งแล้วทำให้เปอร์เซ็นต์หน่วยประมวลผลกลางที่เหลืออยู่จะลดต่ำลง ทำให้เครดิตของ MCU2 ซึ่งมีความเร็วหน่วยประมวลผลกลางเร็วเป็นอันดับ2 และยังไม่มีห้องประชุมเลยมีค่าเครดิตมากที่สุดจึงได้รับเลือกขึ้นมารับผิดชอบห้องประชุม เมื่อมีห้องประชุมในระบบ 46 ห้อง เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางของ MCU1 และ MCU2 สูงขึ้นทำให้เครดิตของ 2 เครื่องนี้ลดลง และ MCU3 มีความเร็วหน่วยประมวลผลกลางปานกลางแต่มีปริมาณหน่วยความจำที่สูงมากทำให้ค่าเครดิตขณะนี้ MCU3 มีมากที่สุด LB จึงเลือก MCU3 ขึ้นมารับห้องประชุมไปบริหารร่วมด้วยกับ MCU1 และ MCU2 และเมื่อห้องประชุมที่ 61 ร้องขอเข้ามา MCU4 จะได้รับเลือกเข้ามาช่วยการทำงานแต่เรา

สังเกตได้ชัดว่า MCU4 ที่มีความเร็วหน่วยประมวลผลกลางต่ำที่สุดเปอร์เซ็นต์การใช้หน่วยประมวลผลกลางจะเติบโตขึ้นอย่างรวดเร็ว

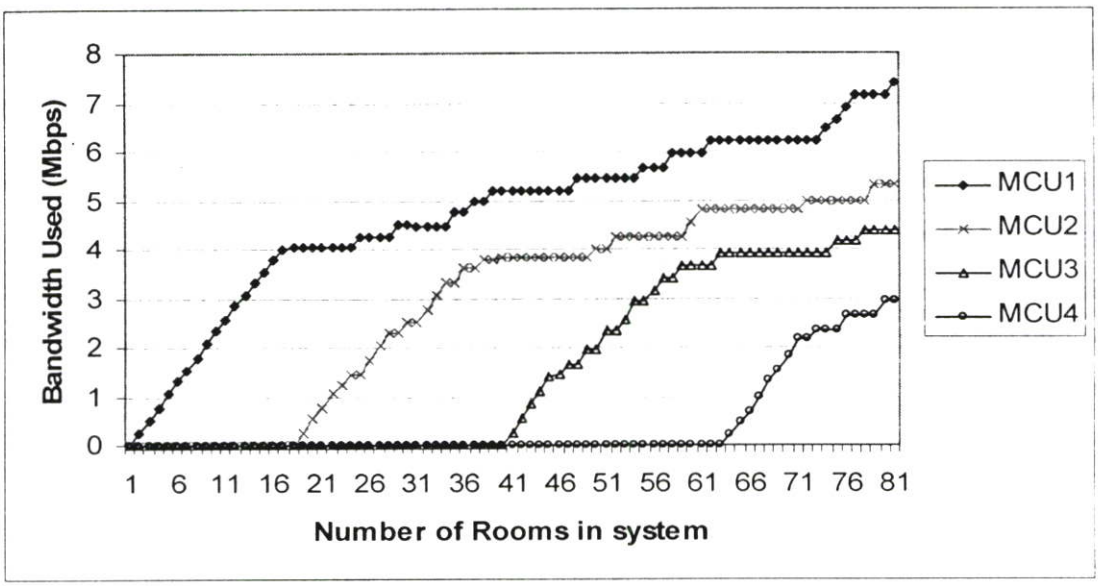
หลังจากที่เรานำเอาฟังก์ชั่นการถ่วงน้ำหนักมาใช้ เราจะสังเกตได้ว่าการใช้งานทรัพยากรต่างๆ จะเพิ่มขึ้นทีละน้อยไม่มีทรัพยากรชนิดใดของ MCU แต่ละตัวถูกใช้งานหนักกว่าตัวอื่นและไม่มี MCU ที่ทรัพยากรถูกใช้จนถึงขีดสุด เมื่อพิจารณาจากทั้ง 3 การทดลองปรากฏว่าขณะที่จำนวนห้องของระบบมีค่าเท่ากับ 81 ระบบที่ใช้ LB ที่มีการใช้ฟังก์ชั่นการถ่วงน้ำหนักไม่มี MCU ตัวใดที่มีค่าการใช้งานหน่วยประมวลผลกลางถึง 100% เลย



รูปที่ 5.17 เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางบนแต่ละ MCU โดยระบบมีการใช้ LB และฟังก์ชั่นถ่วงน้ำหนัก



รูปที่ 5.18 เปอร์เซ็นต์การใช้งานหน่วยความจำบนแต่ละ MCU โดยระบบมีการใช้ LB และฟังก์ชั่นถ่วงน้ำหนัก



รูปที่ 5.19 เปอร์เซ็นต์การใช้งานแบนด์วิดท์บนแต่ละ MCU โดยระบบมีการใช้ LB และฟังก์ชั่นถ่วงน้ำหนัก

5.3 เปรียบเทียบกับอัลกอริทึมอื่นๆ

ความแตกต่างระหว่างอัลกอริทึมของเรากับอัลกอริทึมอื่นๆนั้นคืออัลกอริทึมของเรามีการนำเอาค่าการใช้ทรัพยากรของเครือข่ายเข้ามาร่วมในการคำนวณและมีการใช้ฟังก์ชั่นถ่วงน้ำหนักเพื่อให้ความสำคัญกับหน่วยประมวลผลกลางมากกว่าตัวแปรตัวอื่นๆ โดยพิจารณาจากพฤติกรรมการใช้ทรัพยากรของ MCU ที่ได้จากการทดลอง โดยเราทำการทดลองเพื่อเปรียบเทียบอัลกอริทึมอื่นโดยเราจะทำการทดลองด้วยอัลกอริทึมอื่น 2 แบบ ได้แก่

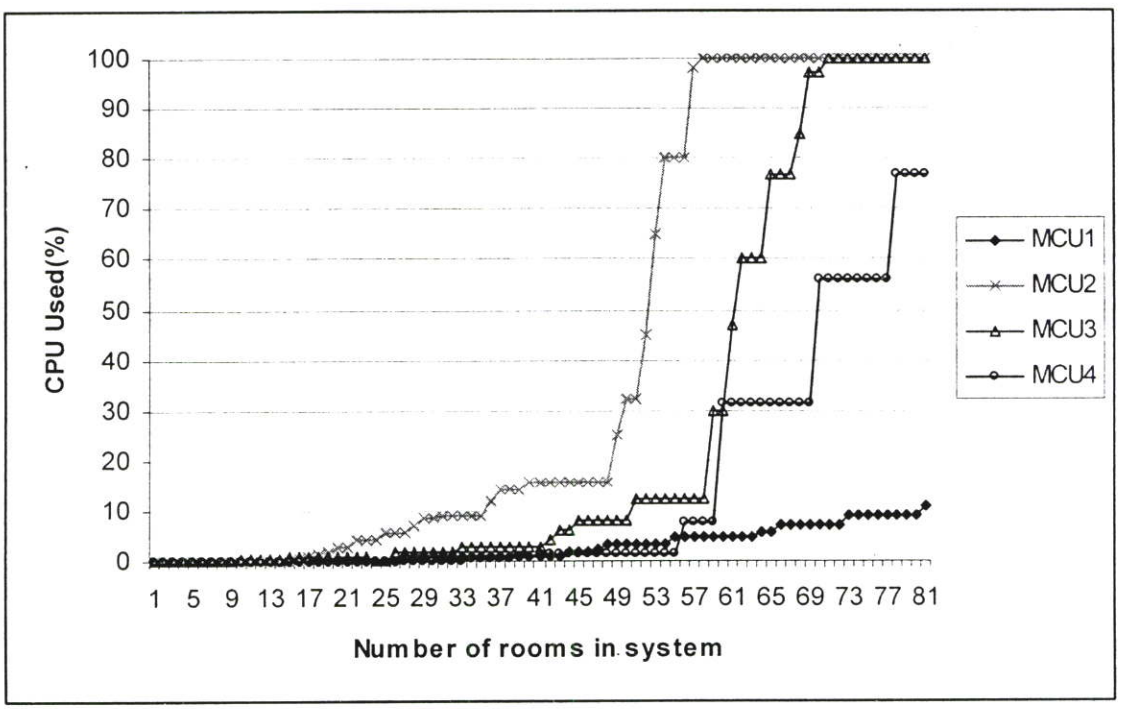
1. Random เหตุผลที่เราเลือกอัลกอริทึมนี้มาเปรียบเทียบนั้นเพราะว่าอัลกอริทึมนี้เป็น Static Load Balancing โดยอัลกอริทึมแบบ Static Load balancing ที่ได้รับความนิยมจะมี 2 อัลกอริทึม นั่นคือ Random และ Round - Robbin แต่อัลกอริทึมแบบ Round - Robbin นั้นไม่เหมาะกับระบบที่ทรัพยากรของแต่ละเครื่องมีความแตกต่างกัน(Heterogeneous) หรือระบบที่เซิร์ฟเวอร์แต่ละเครื่องมีประสิทธิภาพที่ต่างกัน ดังนั้นเราจึงเลือกอัลกอริทึม Random มาทำการเปรียบเทียบ
2. Lowest เหตุผลที่เราเลือกอัลกอริทึมนี้มาเปรียบเทียบนั้นเพราะว่าอัลกอริทึมนี้เป็น Dynamic Load Balancing เนื่องจากอัลกอริทึมแบบไดนามิคนั้นมีหลายอัลกอริทึมที่น่าสนใจเช่น Hydrodynamic หรือแบบ Diffusive แต่อัลกอริทึมที่ใช้กับการส่งงานเข้ามาที่ Load Balancer ก่อนแล้วค่อยส่งงานให้แก่เซิร์ฟเวอร์และน่าจะเหมาะสมกับงานของ MCU นั้น Lowest จะเป็นตัวเลือกที่เหมาะสมที่สุด

โดยใช้สภาพแวดล้อมเดียวกับการทดลองที่ 1, 2 และ 3

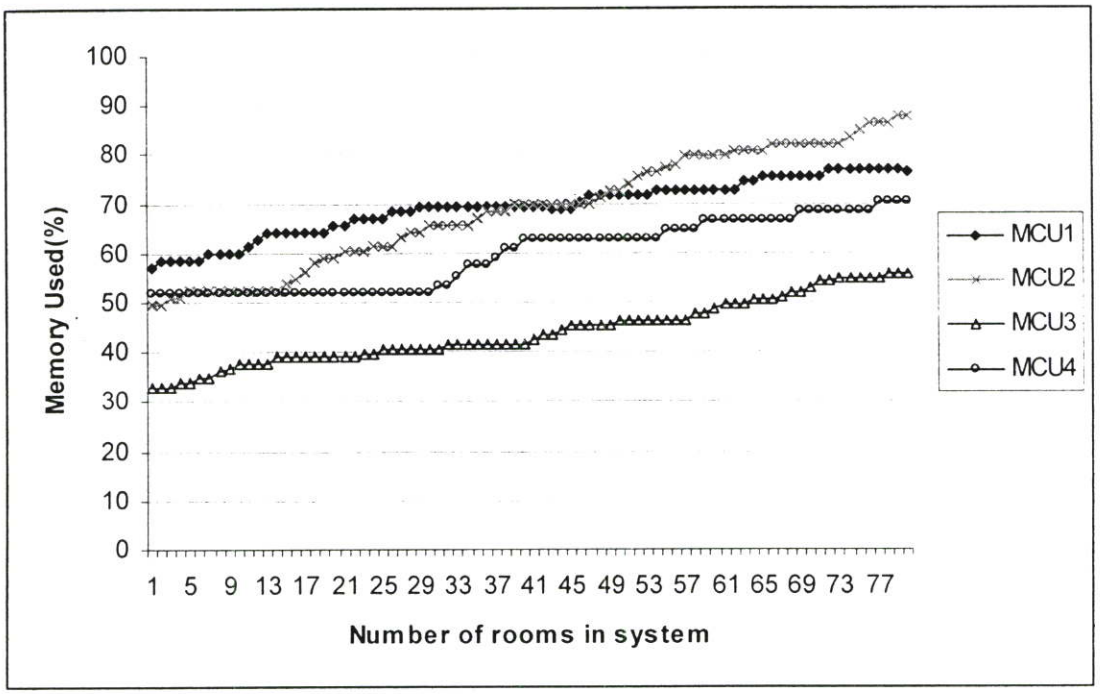
5.3.1) วัดการใช้ทรัพยากรของ MCU แต่ละตัวเมื่อใช้อัลกอริทึม Random

จากรูปที่ 5.20 ซึ่งแสดงเปอร์เซ็นต์การใช้งานของหน่วยประมวลผลกลางนั้น จะเห็นว่าเมื่อห้องที่ 57 เข้ามาสู่ระบบการใช้งานของ MCU2 ก็ถึง 100% แล้ว และจากการใช้อัลกอริทึมแบบสุ่มก็จะไม่สนใจว่าการใช้งานของหน่วยประมวลผลกลางนั้นถึง 100% แล้วหรือไม่ ทำให้ยังมีการส่งงานไปยัง MCU2 อยู่เรื่อยๆ ทำให้อาจจะเกิดการตอบสนองต่อผู้ใช้ในช้าลง เมื่อมีห้องประชุมจำนวน 73 ห้องในระบบ ปรากฏว่ามี MCU 2 ตัวมีการใช้งานหน่วยประมวลผลกลางถึง 100% แล้ว ในขณะที่ MCU1 มีการใช้งานหน่วยประมวลผลกลางประมาณ 10% เท่านั้น และจากรูปที่ 5.22 นั้นการใช้งานแบนด์วิดท์ของ MCU2 นั้นสูงมาก ถ้าเทียบกับแบนด์วิดท์ที่มีอยู่ (10 Mbps)

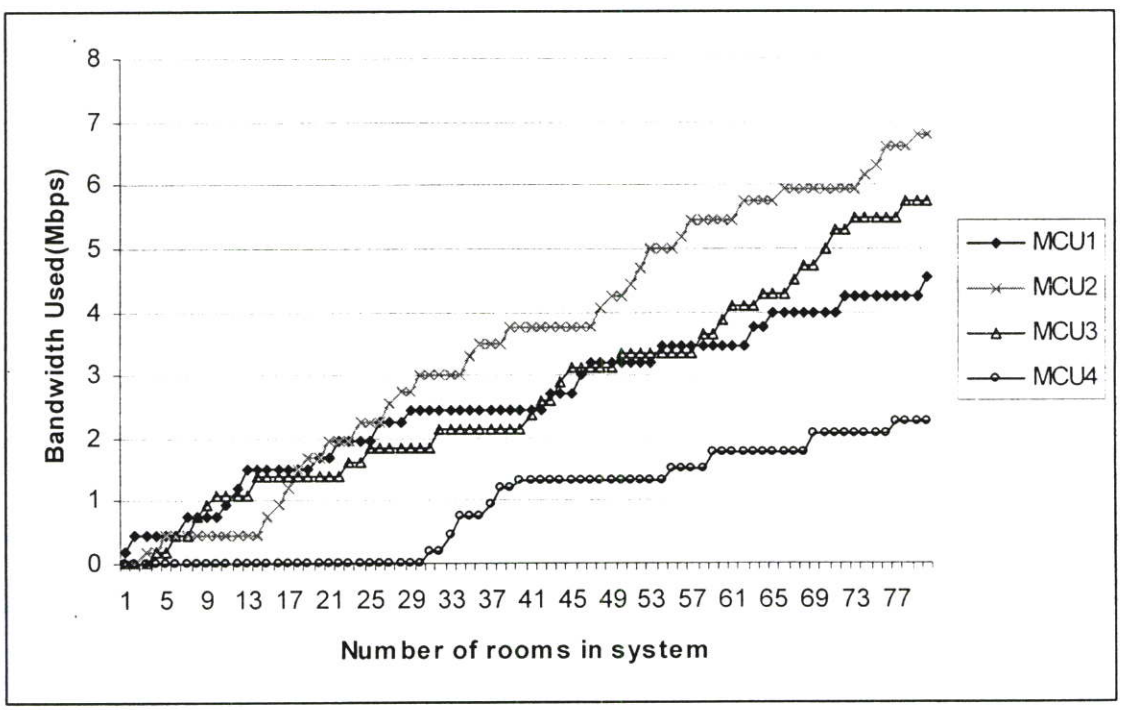
เมื่อเปรียบเทียบกับอัลกอริทึมของเราจะเห็นได้อย่างเด่นชัดว่าอัลกอริทึมของเราจะสามารถกระจายงานได้ดีกว่า



รูปที่ 5.20 เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางบนแต่ละ MCU โดยระบบมีการใช้ LB และอัลกอริทึมแบบ Random



รูปที่ 5.21 เปอร์เซ็นต์การใช้งานหน่วยความจำบนแต่ละ MCU โดยระบบมีการใช้ LB และอัลกอริทึมแบบ Random



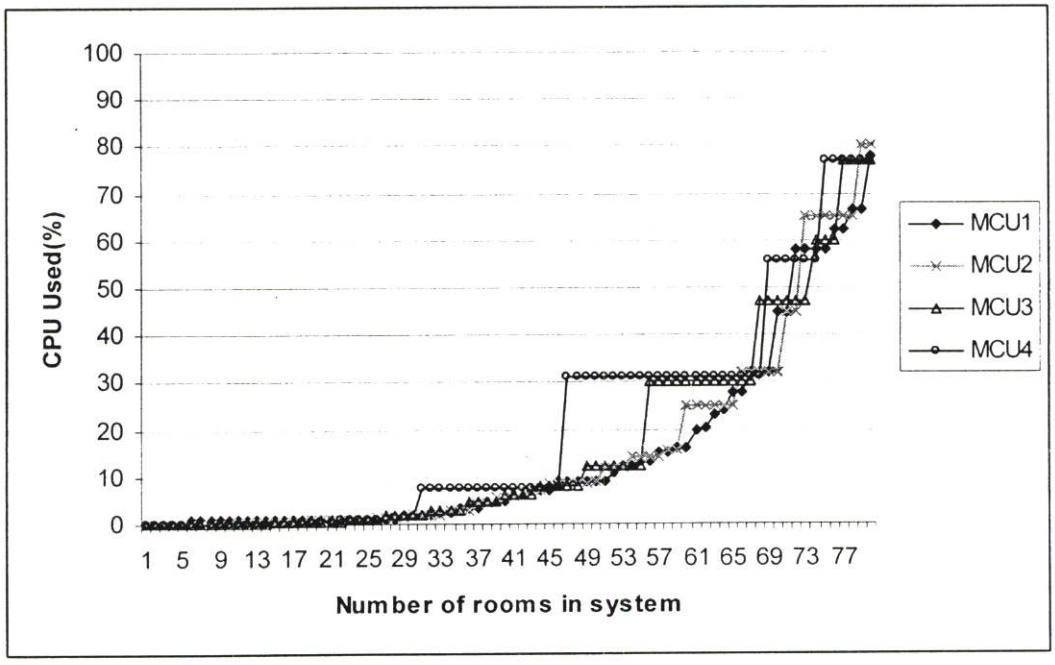
รูปที่ 5.22 เปอร์เซ็นต์การใช้งานแบนด์วิดท์บนแต่ละ MCU โดยระบบมีการใช้ LB และอัลกอริทึมแบบ Random

5.3.2) วัดการใช้ทรัพยากรของ MCU แต่ละตัวเมื่อใช้อัลกอริทึม Lowest

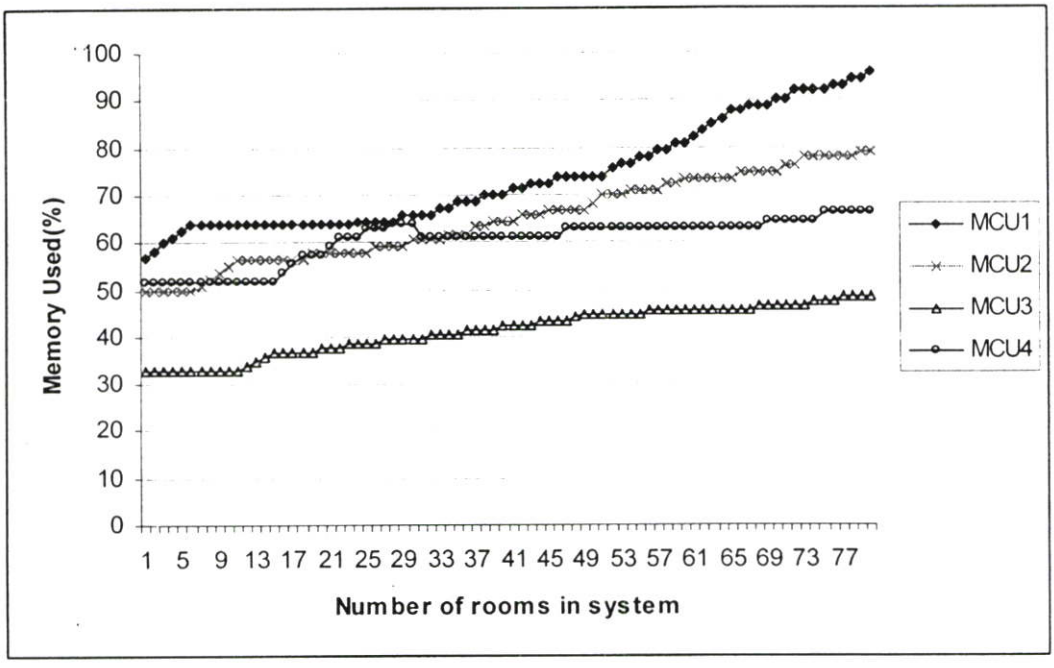
จากรูปที่ 5.23 จะเห็นว่า LB จะทำให้การใช้งานหน่วยประมวลผลกลางโดยรวมของระบบได้อย่างคุ้มค่าและเหมาะสมมากเพราะว่าอัลกอริทึมนี้จะนำเอาการใช้งานหน่วยประมวลผลกลางมาประมวลผลแล้วส่งห้องประชุมใหม่ไปยัง MCU ที่มีการใช้งานหน่วยประมวลผลกลางน้อยที่สุด ซึ่งเหมาะกับงานที่ใช้ทรัพยากรหน่วยประมวลผลกลางเป็นหลัก ซึ่งการทำงานของ MCU นั้นจากรูปที่ 5.7 ก็จะเห็นว่ามีการใช้งานหน่วยประมวลผลกลางมากและเติบโตแบบเอ๊กซ์โพเนนเชียลด้วย

เมื่อเปรียบเทียบกับอัลกอริทึมของเราจากรูปที่ 5.17, 5.18 และ 5.19 อัลกอริทึมแบบ Lowest ดูจะมีความสามารถในการกระจายงานมากกว่าเนื่องจาก MCU1 ที่มีความเร็วหน่วยประมวลผลกลางสูงที่สุดมีทรัพยากรทางเครือข่ายที่สูงและหน่วยความจำก็มีมากด้วยเช่นเดียวกันทำให้การที่ LB ส่งห้องประชุมส่วนมากไปที่ MCU1 นั้นเป็นการตัดสินใจที่ถูก

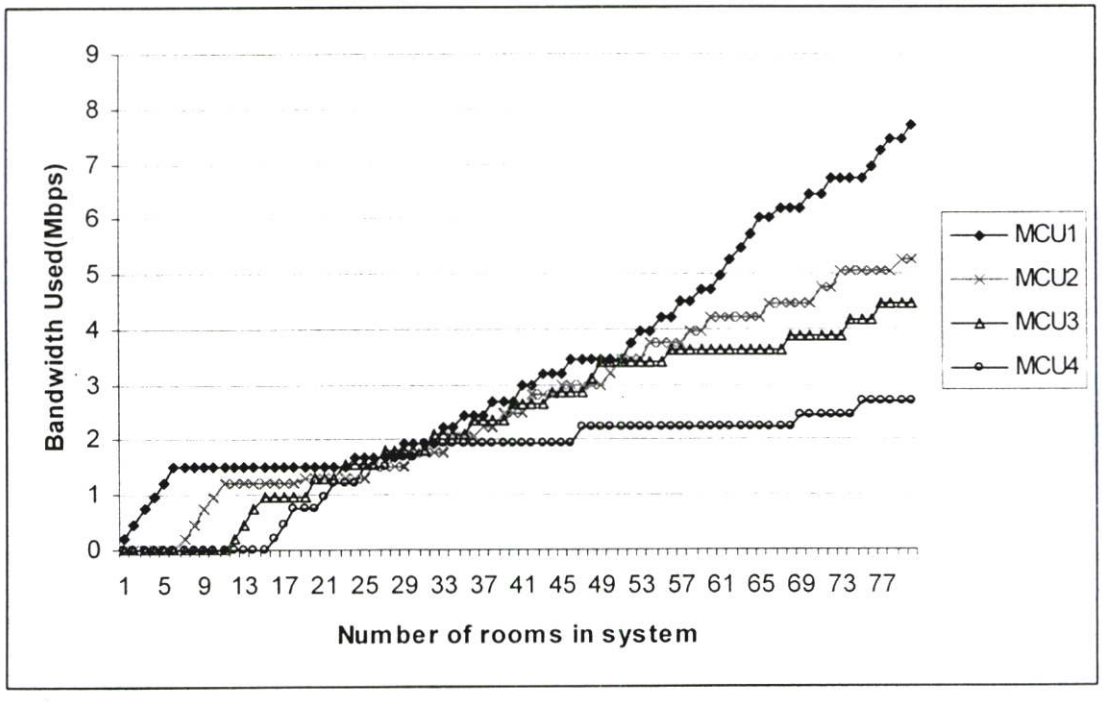
แต่ถ้าหากว่า MCU บางตัวมีทรัพยากรทางเครือข่ายน้อยแต่มีความเร็วหน่วยประมวลผลกลางมากจะทำให้การตอบสนองต่อผู้ใช้ช้าซึ่งระบบการประชุมด้วยภาพนั้นความสำคัญไม่ได้มีเพียงความสามารถ ในการประมวลผลเท่านั้น การตอบสนองต่อผู้ใช้ก็เป็นสิ่งสำคัญ หรือถ้า MCU มีหน่วยประมวลผลกลางมากแต่หน่วยความจำน้อยก็อาจจะเกิดการใช้หน่วยความจำเต็ม 100% ก็มีโอกาสเป็นไปได้ ดังนั้นสำหรับ ระบบการประชุมด้วยภาพนั้นอัลกอริทึมของเรานั้นเหมาะสมมากกว่า ดังตัวอย่างในตารางที่ 1, 2, 3 และ 4



รูปที่ 5.23 เปอร์เซ็นต์การใช้งานหน่วยประมวลผลกลางบนแต่ละ MCU โดยระบบมีการใช้ LB และอัลกอริทึมแบบ Lowest



รูปที่ 5.24 เปอร์เซ็นต์การใช้งานหน่วยความจำบนแต่ละ MCU โดยระบบมีการใช้ LB และอัลกอริทึมแบบ Lowest



รูปที่ 5.25 เปอร์เซ็นต์การใช้งานแบนด์วิดท์บนแต่ละ MCU โดยระบบมีการใช้ LB และอัลกอริทึมแบบ Lowest

ตารางที่ 5.1 ตัวอย่างการเลือก MCU ของอัลกอริทึมที่นำเสนอในกรณีที่มีการใช้หน่วยประมวลผลกลางใกล้เคียงกันแต่อัลกอริทึมที่นำเสนอจะพิจารณาที่ค่าดีเลย์ด้วย

	MCU 1							
	CPU	Weight CPU	Memory	Weight Memory	Bandwidth	Weight Bandwidth	Delay	Load
Room66	58.02	1	475	0.67	6.75	1	168	1992.412
Room72	62.1	1	482	0.67	7.05	1	82	1751.286
Room78	80	1	503	0.67	7.8	1	395	398.3777
Room79	80	1	503	0.67	7.8	1	19	774.3777
Room81	82	1	511	0.67	8	1	109	607.1309
Room84	95	1	517	0.67	8.25	1	107	309.0957
Room85	95	1	517	0.67	8.25	1	55	361.0957

	MCU 2							
	CPU	Weight CPU	Memory	Weight Memory	Bandwidth	Weight Bandwidth	Delay	Load
Room66	65	0.77	392	0.67	5	0.1	351	917.4531
Room72	65	0.77	392	0.67	5	0.1	96	1172.453
Room78	80	0.77	399	0.67	5.3	0.1	164	468.2571
Room79	80	0.77	399	0.67	5.3	0.1	143	489.2571
Room81	80	0.77	399	0.67	5.3	0.1	230	402.2571
Room84	98	0.77	407	0.67	5.5	0.1	419	-92.7298
Room85	98	0.77	407	0.67	5.5	0.1	124	202.2702

	MCU3							
	CPU	Weight CPU	Memory	Weight Memory	Bandwidth	Weight Bandwidth	Delay	Load
Room66	47	0.583	359	1	3.6	1	182	1916.147
Room72	60	0.583	365	1	3.85	1	420	972.45
Room78	77	0.583	372	1	4.15	1	330	437.057
Room79	77	0.583	372	1	4.15	1	413	354.057
Room81	85	0.583	380	1	4.35	1	159	429.825
Room84	97	0.583	386	1	4.6	1	310	152.047
Room85	97	0.583	386	1	4.6	1	16	446.047

	MCU4							
	CPU	Weight CPU	Memory	Weight Memory	Bandwidth	Weight Bandwidth	Delay	Load
Room66	0.78	0.251	223	0.5	0.95	1	450	2441.01
Room72	31.22	0.251	251	0.5	1.95	1	436	1166.77
Room78	55.9	0.251	259	0.5	2.2	1	138	764.2234
Room79	77	0.251	265	0.5	2.5	1	474	71.77379
Room81	77	0.251	265	0.5	2.5	1	176	369.7738
Room84	77	0.251	265	0.5	2.5	1	185	360.7738
Room85	85	0.251	272	0.5	2.7	1	204	264.3583

บทที่ 6
สรุปงานวิจัยที่นำเสนอ

วิทยานิพนธ์นี้ได้ทำการนำเสนออัลกอริทึมการกระจายงานสำหรับระบบการประชุมด้วยภาพเพื่อทำการบริหารจัดการ MCU ที่กระจายอยู่ในระบบการประชุมด้วยภาพให้ทำงานได้เหมาะสมกับทรัพยการแต่ละเครื่องให้มากที่สุด โดยอัลกอริทึมการกระจายงานของเรานั้นเป็นแบบ Dynamic Load Balancing คือจะนำข้อมูลของระบบ ณ เวลาปัจจุบันเข้ามาประมวลผลและเลือก MCU ที่เหมาะสมเข้ามารับผิดชอบห้องประชุม และเพราะสิ่งสำคัญในการทำงานของ MCU นั้นไม่เพียงอยู่ที่ภาระงานในการประมวลผลเท่านั้น(การทำงานของซีพียู และหน่วยความจำ) หากแต่ยังมีผลจากทรัพยากรทางด้านเครือข่ายระหว่าง MCU แต่ละตัวกับผู้ใช้(แบนด์วิดท์ของ MCU และ ดีเลย์ระหว่าง MCU กับผู้ใช้) ที่ได้ทำการร้องขอการสร้างห้องประชุมอีกด้วย เนื่องจากค่าภาระงานที่เราวัดมาจากแต่ละ MCU เราใช้เป็นหน่วยเปอร์เซ็นต์ ประสิทธิภาพของ MCU ที่ต่างกันนั้นเราไม่สามารถที่จะนำมาคำนวณได้เลย แต่เราต้องทำการถ่วงน้ำหนักตัวแปรชนิดเดียวกันก่อนที่จะนำมาคำนวณค่าเครดิตได้ จาการทดลองเราจะเห็นว่าการใช้ทรัพยากรแต่ละชนิดของ MCU นั้นมีแนวโน้มการใช้ที่ต่างกันเราจึงต้องมีฟังก์ชั่นถ่วงน้ำหนักเพื่อให้ค่าเครดิตที่คำนวณออกมานั้นมีความเหมาะสมที่จะใช้งานมากที่สุด เมื่อมีผู้ใช้ทำการร้องขอการสร้างห้องประชุมใหม่ตัวกระจายงานจะทำการประมวลผลค่าเครดิตแล้วทำการเลือก MCU ที่มีค่าเครดิตมากที่สุดแล้วมอบหมายห้องประชุมนี้ให้ MCU ตัวนั้นบริหารจัดการ จากการทดลองใช้อัลกอริทึมของเราจะพบว่ามีการใช้งาน MCU แต่ละเครื่องได้กระจายมากขึ้นและไม่มี MCU ตัวใดที่รับภาระงานหนักเกินที่จะสามารถรับได้ก่อน MCU ตัวอื่นๆ เมื่อการเลือกใช้ MCU เหมาะสมและการเลือกนั้นมีการนำเอาทรัพยากรทางด้านเครือข่ายเข้ามาร่วมด้วย การประมวลผล MCU ทำได้ดี การรับส่งข้อมูลไม่ดีเลย์มากเกินไปจะทำให้ระบบการประชุมด้วยภาพมีประสิทธิภาพที่จะตอบสนองต่อผู้ใช้ได้อย่างเต็มความสามารถ

บรรณานุกรม

[1] C. Traiperm and S. Kittitornkun, "High Performance Video Conference Multipoint Control Unit with MPEG-4 Codec", ECTI-Con 2005, May 12-13, 2005

[2] Ka-Po Chow and Yu-Kwong Kwok, "On Load Balancing for Distributed Multiagent," *IEEE Transactions on Computing Parallel and Distributed Systems*, 2002, pp. 787-801

[3] Songnian Zhou, " A Trace-Driven Simulation Study of Dynamic Load balancing," *IEEE Transactions* on *Software Engineering,* Volume 14, Issue 9, Sept. 1988

[4] Li Xiao, Xiadong Zhang and Yanxia Qu, "Effective load sharing on heterogeneous networks of workstations," *14th International Proceeding Parallel and Distributed Processing Symposium, 2000. IPDPS 2000.*

[5] T. Kawasaki and K. Okamura, "Evaluation on Scalability of Conference System using Request-Routing," *18th International Conference on Advanced Information Networking and Applications (AINA'04)*, 2004

[6] R. Ventakesha Prasad, H. N. Shankar and H. S. Jamadagni, S. Vijay, "Server Allocation Algorithm for VOIP Conferencing," *Distributed Frameworks for Multimedia Applications*, 2005.

[7] OpenH323 Project, http://www.openh323.org

[8] Prasad Calyam, Mukundan Sridharan, Weiping Mandrawal and Paul Schopis, "Performance measurement and analysis of H.323 traffic," *PAM 2004 : passive and active network measurement,* Antibes les Pin, Frances, 19-20 April 2004

[9] Mark Carson and Darrin Santay, NISTNET:a Linux-based network emulation tool, Computer Communication Review (ACM SIGCOMM),

[10] Chi-Chung Hui, Chanson, S.T, "Hydrodynamic load balancing," Parallel and Distributed Systems, IEEE Transactions on Volume 10, Issue 11, Nov. 1999 Page(s):1118-1137

[11] Cape May, New Jersey, "An analysis of diffusive load-balancing," ACM Symposium on Parallel Algorithms and Architectures archive Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures, United States Pages: 220 - 225 Year of Publication: 1994

ภาคผนวก

ภาคผนวก ก.

ตัวอย่างโค้ดส่วนการสร้างเทรดเพื่อรับค่าภาระงานที่ LoadReportingServer

```
void LoadReportingServer::Start ()
{
      Trace (S"Starting...");

      DEBUG_ASSERT (! (status == ServerStatus::Starting ||
                              (status == ServerStatus::Started)));
      if (status == ServerStatus::Starting || status ==
ServerStatus::Started)
            return;

      // subscribe the server to receive notification when the
configuration
      // changes (the underlying configuration file is modified)
      //
      configurator->add_Changed (
            new Configurator::ChangedEventHandler (
                  this,
&LoadReportingServer::OnConfigurationChanged));

      status = ServerStatus::Starting;

      // remove the const'ness of "this" to get away from C2664
      LoadReportingServer __gc* thisServer =
            const_cast<LoadReportingServer __gc*> (this);

      // set up the signalling event
      //
      eventWorkerThreadDone = new AutoResetEvent (false);

      Trace (S"Starting worker thread...");
      // launch the worker thread
      //
      ReportingWorker __gc* worker = new ReportingWorker (
            thisServer,
            new WorkerDoneEventHandler (
                  this,
                  &LoadReportingServer::OnWorkerThreadDone),
            ipAddress,
            port,
            reportInterval,
            performanceCounters);
      workerThread = new Thread (new ThreadStart (worker,
&ReportingWorker::Run));
      workerThread->Name = S"Load Reporting Server's ReportingWorker
Thread";
      workerThread->Start ();
      Trace (S"Worker thread started.");

      // wait for the thread to signal us, by calling out the
callback
      //
```

```
        eventWorkerThreadDone->WaitOne (Timeout::Infinite, false);

        // check the outcome of the thread result
        //
        if (workerThreadSucceeded)
        {
              status = ServerStatus::Started;
              Trace (S"Started.");
        }
        else
        {
              Stop ();
        }
}
```

ภาคผนวก ข.

ตัวอย่างโค้ดส่วนการสร้างเทรดเพื่อส่งค่าภาระงานที่

LoadMonitoringServer

```
void LoadMonitoringServer::Start ()
{
      Trace (S"Starting...");

      DEBUG_ASSERT (! (status == ServerStatus::Starting ||
                              (status == ServerStatus::Started)));
      if (status == ServerStatus::Starting || status ==
ServerStatus::Started)
            return;

      status = ServerStatus::Starting;

      // subscribe the server to receive notification when the
configuration
      // changes (the underlying configuration file is modified)
      //
      cfg->add_Changed (
            new Configurator::ChangedEventHandler (
                  this,
&LoadMonitoringServer::OnConfigurationChanged));

      // remove the const'ness of "this" to get away from C2664
      LoadMonitoringServer __gc* thisServer =
            const_cast<LoadMonitoringServer __gc*> (this);

      // set up the signalling events
      //
      eventCollectorThreadDone = new AutoResetEvent (false);
      eventReporterThreadDone = new AutoResetEvent (false);

      Trace (S"Starting collector thread...");
      // launch the collector worker thread
      //
      CollectorWorker __gc* collectorWorker = new CollectorWorker (
            thisServer,
            new WorkerDoneEventHandler (
                  this,
                  &LoadMonitoringServer::OnCollectorThreadDone),
            ipAddress,
            collectorPort,
            collectorBacklog,
            machineLoads);
      collectorThread = new Thread (
            new ThreadStart (collectorWorker,
&CollectorWorker::Run));
      collectorThread->Name = S"Load Monitoring Server's Collector
Thread";
      collectorThread->Start ();
      Trace (S"Collector thread started.");

      Trace (S"Starting reporter thread...");
      // launch the reporter worker thread
```

```
        //
        ReporterWorker __gc* reporterWorker = new ReporterWorker (
                thisServer,
                new WorkerDoneEventHandler (
                        this,
                        &LoadMonitoringServer::OnReporterThreadDone),
                reporterPort,
                reporterBacklog,
                machineLoads);
        reporterThread = new Thread (
                new ThreadStart (reporterWorker, &ReporterWorker::Run));
        reporterThread->Name = S"Load Monitoring Server's Reporter
Thread";
        reporterThread->Start ();
        Trace (S"Reporter thread started.");

        // wait for the threads to signal us, by calling out the
callbacks
        //
        eventCollectorThreadDone->WaitOne (Timeout::Infinite, false);
        eventReporterThreadDone->WaitOne (Timeout::Infinite, false);

        // check the outcome of the thread results
        //
        if (collectorThreadSucceeded && reporterThreadSucceeded)
        {
                status = ServerStatus::Started;

                // register the ServerLoadBalancer object with the
Remoting
                // runtime
                //
                registeredForRemoting = RegisterLoadBalancerForRemoting
();
                DEBUG_ASSERT (registeredForRemoting);
                if (!registeredForRemoting)
                {
                        Stop ();
                        throw (new InvalidOperationException (
                                S"Cannot register load balancer object "
                                S"with the Remoting runtime!"));
                }

                Trace (S"Started.");
        }
        else
        {
                Stop ();
        }
}
```

ภาคผนวก ค.

ฟังก์ชั่น Ping ที่ใช้ในการหาค่าดีเลย์ระหว่างผู้ใช้กับ MCU

```
int CPing::Ping(LPCTSTR pszHostName, CPingReply& pr, int nPings,
UCHAR nTTL, DWORD dwTimeout, int nPacketSize, UCHAR nTOS, BOOL
bDontFragment)
{
      pr.nError = -1;
      pr.minRTT = ULONG_MAX;
      pr.maxRTT = 0;
      pr.avgRTT = 0;

      //For correct operation of the T2A macro, see TN059
      USES_CONVERSION;

      //Use the High performace counter to get an accurate RTT
      LARGE_INTEGER Frequency;
      Frequency.QuadPart = 0;
      if (!QueryPerformanceFrequency(&Frequency))
      {
            TRACE(_T("Failed to get the high performance counter
frequency\n"));
            return -1;
      }
      __int64 nTimerFrequency = Frequency.QuadPart;

      //Resolve the address of the host to connect to
      sockaddr_in dest;
      memset(&dest,0,sizeof(dest));
      LPSTR lpszAscii = T2A((LPTSTR) pszHostName);
      unsigned long addr = inet_addr(lpszAscii);
      if (addr == INADDR_NONE)
      {
            //Not a dotted address, then do a lookup of the name
            hostent* hp = gethostbyname(lpszAscii);
            if (hp)
            {
                  memcpy(&(dest.sin_addr),hp->h_addr,hp->h_length);
                  dest.sin_family = hp->h_addrtype;
            }
            else
            {
                  TRACE(_T("CPing::PingUsingWinsock, Could not
resolve the host name %s\n"), pszHostName);
                  CString temp;
                  temp.Format("Could not resolve host name \"%s\" ",
pszHostName);
                  AfxMessageBox(temp);
                  return -1;
            }
      }
      else
      {
            dest.sin_addr.s_addr = addr;
            dest.sin_family = AF_INET;
      }
```

```
      //Create the raw socket
      SOCKET sockRaw = WSASocket(AF_INET, SOCK_RAW, IPPROTO_ICMP,
NULL, 0, 0);
      if (sockRaw == INVALID_SOCKET)
      {
            TRACE(_T("CPing::PingUsingWinsock, Failed to create a raw
socket\n"));
            return -1;
      }

      SOCKADDR_IN sockLocalAddress;
      ZeroMemory(&sockLocalAddress, sizeof(sockLocalAddress));
      sockLocalAddress.sin_family = AF_INET;
      sockLocalAddress.sin_port = htons((u_short)0);

      LPHOSTENT lphost;
      lphost = gethostbyname(NULL);
      if (lphost != NULL)
            sockLocalAddress.sin_addr.s_addr = ((LPIN_ADDR)lphost-
>h_addr)->s_addr;
      else
      {
            return -1;
      }

      if (bind(sockRaw, (sockaddr*) &sockLocalAddress,
sizeof(sockLocalAddress)) == SOCKET_ERROR)
      {
            TRACE(_T("CPing::PingUsingWinsock, Failed to bind to
specified address\n"));
            return -1;
      }

      //Set the TTL on the socket
      int nTempTTL = nTTL;
      if (setsockopt(sockRaw, IPPROTO_IP, IP_TTL, (char*) &nTempTTL,
sizeof(nTempTTL)) == SOCKET_ERROR)
      {
            TRACE(_T("CPing::PingUsingWinsock, Failed to set the TTL
value on the socket\n"));
            return -1;
      }

      //Set the TOS on the socket
      int nTempTos = nTOS;
      if (setsockopt(sockRaw, IPPROTO_IP, IP_TOS, (char*) &nTempTos,
sizeof(nTempTos)) == SOCKET_ERROR)
      {
            TRACE(_T("CPing::PingUsingWinsock, Failed to set the Tos
value on the socket\n"));
            return -1;
      }

      //Set the Don't Fragment flag on the socket
      if (bDontFragment)
      {
            if (setsockopt(sockRaw, IPPROTO_IP, IP_DONTFRAGMENT,
(char*) &bDontFragment, sizeof(bDontFragment)) == SOCKET_ERROR)
            {
                  TRACE(_T("CPing::PingUsingWinsock, Failed to set
the Don't Fragment value on the socket\n"));
```

```
                return -1;
            }
        }

        //Allocate the ICMP packet
        int nBufSize = nPacketSize + sizeof(ICMP_HEADER);
        char* pICMP = new char[nBufSize];
        FillIcmpData((LPICMP_HEADER) pICMP, nBufSize);

        //Get the tick count prior to sending the packet
        LARGE_INTEGER TimerTick;
        VERIFY(QueryPerformanceCounter(&TimerTick));
        __int64 nStartTick = TimerTick.QuadPart;

        for (int i=0; i<nPings; i++)
        {
            //Send of the packet
            int nWrote = sendto(sockRaw, pICMP, nBufSize, 0,
(sockaddr*)&dest, sizeof(dest));
            if (nWrote == SOCKET_ERROR)
            {
                TRACE(_T("CPing::PingUsingWinsock, sendto
failed\n"));

                delete [] pICMP;

                DWORD dwError = GetLastError();
                closesocket(sockRaw);
                SetLastError(dwError);

                return -1;
            }

            //allocate the recv buffer
            char* pRecvBuf = new char[MAX_ICMP_PACKET_SIZE];
            BOOL bReadable;
            sockaddr_in from;
            int nFromlen = sizeof(from);
            int nRead = 0;

            //Allow the specified timeout
            if (IsSocketReadible(sockRaw, dwTimeout, bReadable))
            {
                if (bReadable)
                {
                    //Receive the response
                    nRead = recvfrom(sockRaw, pRecvBuf,
MAX_ICMP_PACKET_SIZE, 0, (sockaddr*)&from, &nFromlen);
                }
                else
                {
                    TRACE(_T("CPing::PingUsingWinsock, timeout
occured while awaiting recvfrom\n"));
                    closesocket(sockRaw);

                    delete [] pICMP;
                    delete [] pRecvBuf;

                    //set the error to timed out
                    SetLastError(WSAETIMEDOUT);
```

```
                        return -1;
                    }
                }
                else
                {
                    TRACE(_T("CPing::PingUsingWinsock, IsReadible call
failed\n"));

                    delete [] pICMP;
                    delete [] pRecvBuf;

                    DWORD dwError = GetLastError();
                    closesocket(sockRaw);
                    SetLastError(dwError);

                    return -1;
                }

                //Get the current tick count
                QueryPerformanceCounter(&TimerTick);

                //Now check the return response from recvfrom
                if (nRead == SOCKET_ERROR)
                {
                    TRACE(_T("CPing::PingUsingWinsock, recvfrom call
failed\n"));

                    delete [] pICMP;
                    delete [] pRecvBuf;

                    DWORD dwError = GetLastError();
                    closesocket(sockRaw);
                    SetLastError(dwError);

                    return -1;
                }

                //Decode the response we got back
                pr.nError = DecodeResponse(pRecvBuf, nRead, &from);

                pr.Address = from.sin_addr;
                ULONG rtt = (ULONG) ((TimerTick.QuadPart - nStartTick) *
1000 / nTimerFrequency);
                pr.avgRTT += rtt;
                if (pr.maxRTT < rtt)
                    pr.maxRTT = rtt;
                if (pr.minRTT > rtt)
                    pr.minRTT = rtt;
                delete [] pRecvBuf;
            }
            pr.avgRTT /= nPings;
            //Don't forget to release out socket
            closesocket(sockRaw);

            //Free up the memory we allocated
            delete [] pICMP;

            //return the status
            return pr.nError;
}
```

```
BOOL CPing::IsSocketReadible(SOCKET socket, DWORD dwTimeout, BOOL&
bReadible)
{
  timeval timeout = {dwTimeout/1000, dwTimeout % 1000};
  fd_set fds;
  FD_ZERO(&fds);
  FD_SET(socket, &fds);
  int nStatus = select(0, &fds, NULL, NULL, &timeout);
  if (nStatus == SOCKET_ERROR)
  {
    return FALSE;
  }
  else
  {
    bReadible = !(nStatus == 0);
    return TRUE;
  }
}



//Decode the raw IP packet we get back
int CPing::DecodeResponse(char* pBuf, int nBytes, sockaddr_in* from)
{
      //Get the current tick count
      LARGE_INTEGER TimerTick;
      VERIFY(QueryPerformanceCounter(&TimerTick));


      LPIP_HEADER pIpHdr = (LPIP_HEADER) pBuf;
      int nIpHdrlen = pIpHdr->h_len * 4; //Number of 32-bit words*4 =
bytes

      //Not enough data recieved
      if (nBytes < nIpHdrlen + MIN_ICMP_PACKET_SIZE)
      {
            TRACE(_T("Received too few bytes from %sin ICMP
packet\n"), inet_ntoa(from->sin_addr));
            SetLastError(ERROR_UNEXP_NET_ERR);
            return -1;
      }

      //Check it is an ICMP_ECHOREPLY packet
      LPICMP_HEADER pIcmpHdr = (LPICMP_HEADER) (pBuf + nIpHdrlen);
      if (pIcmpHdr->i_type != 0) //type ICMP_ECHOREPLY is 0
      {
            return ((int)pIcmpHdr->i_type);
      }

      //Check it is the same id as we sent
      if (pIcmpHdr->i_id != (USHORT)GetCurrentProcessId())
      {
            TRACE(_T("Received someone else's ICMP packet!\n"));
            SetLastError(ERROR_UNEXP_NET_ERR);
            return -1;
      }
      return pIcmpHdr->i_type;
}

//generate an IP checksum based on a given data buffer
USHORT CPing::GenerateIPChecksum(USHORT* pBuffer, int nSize)
```

```
{
      unsigned long cksum = 0;

      while (nSize > 1)
      {
            cksum += *pBuffer++;
            nSize -= sizeof(USHORT);
      }

      if (nSize)
            cksum += *(UCHAR*)pBuffer;

      cksum = (cksum >> 16) + (cksum & 0xffff);
      cksum += (cksum >>16);
      return (USHORT)(~cksum);
}

//Fill up the ICMP packet with defined values
void CPing::FillIcmpData(LPICMP_HEADER pIcmp, int nData)
{
      pIcmp->i_type     = 8; //ICMP_ECHO type
      pIcmp->i_code     = 0;
      pIcmp->i_id       = (USHORT) GetCurrentProcessId();
      pIcmp->i_seq      = 0;
      pIcmp->i_cksum    = 0;
      pIcmp->timestamp = GetTickCount();

      //Set up the data which will be sent
      int nHdrSize = sizeof(ICMP_HEADER);
      char* pData = ((char*)pIcmp) + nHdrSize;
      memset(pData, 'E', nData - nHdrSize);

      //Generate the checksum
      pIcmp->i_cksum = GenerateIPChecksum((USHORT*)pIcmp, nData);
}
```

ภาคผนวก ง.
งานวิจัยที่ได้รับการตีพิมพ์เผยแพร่

Jirayu Charoenviriyaphap and Surin Kittitornkun, "ON LOAD BALANCING FOR MULTIPOINT VIDEO CONFERENCING CONTROL UNIT," The 4th International Joint Conference on Computer Science and Software Engineering, JCSSE2007, May 2nd – 4th ,2007 Department of Computer Engineering, Khon Kaen University, Khon Kaen, THAILAND


The 4th International Joint Conference
on Computer Science
and Software Engineering (JCSSE2007)
May 2nd - 4th, 2007
Department of Computer Engineering, Khon Kaen University
Khon Kaen, THAILAND
NECTEC
MFEC
SiPA
IRONPORT
TT&T
E-SAAN


International Joint Conference on
Computer Science & Software Engineering
JCSSE
2007

Computer Graphics

Computer Systems

International Joint Conference on Computer Science & Software Engineering (JCSSE2007)
May 2-4, 2007

ON LOAD BALANCING FOR MULTIPOINT VIDEO CONFERENCING CONTROL UNIT

Jirayu Charoenviriyaphap and Surin Kittitornkun
Dept. of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang,
Bangkok, 10520 Thailand
c_jirayu@msn.com, kksurin@kmitl.ac.th

***Abstract*-Today videoconferencing is more popular because the Internet is more widespread and faster than it was in the past few years. Multipoint videoconferencing requires a multipoint control unit (MCU) to mix video/audio and manage the floor. MCU load increases as much as user's growth worldwide. Therefore, distributed MCUs can balance the load from all users. However, we need to equally balance the MCUs by introducing a load-balancer. The contribution of this paper is the balancing algorithm taking into account of %CPU used, %memory used and %bandwidth used.**

I. INTRODUCTION

Videoconferencing is very popular because people do not need to spend the time for conference just 1-2 hours but take more time in traveling. And now internet is faster and cheaper than past e.g. ADSL, leased line and broadband. And internet price is cheaper. Videoconferencing is one choice to saving time and money. Multipoint videoconferencing requires a multipoint control unit (MCU) to manage the floor and connection, encode and decode video from users, mix audio and send to users. Encoding and decoding video and mixing audio consume a lot of resources. When we talk about resource computer, we almost think about CPU, memory and harddisk but for MCU another resource that is important is network resource. Except encoding and decoding video and mixing audio, MCU has a task to manage connection both signaling and data packet.

MCU load increases as much as user's growth. For supporting large users Centralized MCU is changed to distributed MCU and optimized MCU. When we distributed MCUs there are problems about balance of MCU load. Therefore we need to manage MCUs to work equally and accordingly. The main contribution of this paper is the balancing algorithm taking into account of %CPU used, %memory used and %bandwidth used.

This paper is organized as follows. Section II explains H.323 and general load balancing algorithms. We describe our own algorithm and show the experimental results in Section III and IV, respectively. Section V concludes this paper and suggests some future works.

II. H.323 AND LOAD BALANCING ALGORITHM

A. H.323 Multipoint Conference

H.323 protocol is developed by ITU-T (International Telecommunications Union- Telecommunications section) for real-time multimedia communication over IP.

Today H.323 conference system is one of the most famous conference systems. H.323 basically supports one-to-one remote conference, and H.323 also supports multipoint conference using MCU (Multipoint Control Unit).

MCU communicates with all hosts. MCU collects audio and video streams from all hosts, mixes them, and distributes the mixed stream to all hosts. In other words, MCU holds one-to-one conference with all hosts. Therefore, when the number of hosts increases, CPU and network load on MCU become heavier and conference system may become overloaded. To avoid this bottle neck about load on MCU, conference system should adopt the method of distributing the load on MCU. The existing method of load sharing is using multiple MCUs as Fig.1. It distributes the hosts of which MCU has to take care to two or more MCUs to reduce the load on each MCU. This method reduces the load on each host, but it has some problems. When the number of hosts increase, the number of MCUs which is required for load sharing also increases. And the number and location of hosts change dynamically in the conference. But, the number and location of MCUs should be static. Therefore if the number of hosts becomes enormous, MCUs may become overload after all. And if hosts crowd around somewhere, the MCU near the hosts should deal with them, and becomes overload after all.

B. MPEG4MCU [1]

MPEG4MCU [1] can be partitioned into 2 sections as follows

a) Audio Section

Each Connection object corresponds to each client. The client always sends encoded audio stream to the MPEG4MCU. The decoded audio is used by the DetectVoice function for detecting the speaking client.

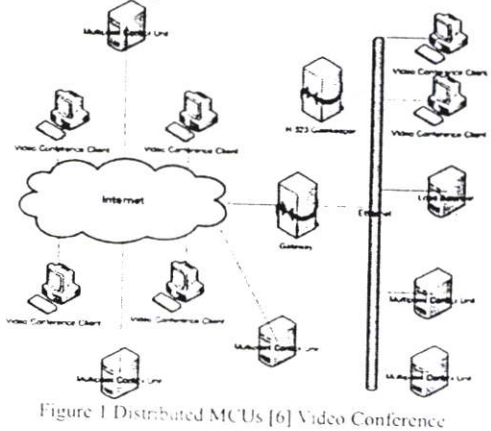


Figure 1 Distributed MCUs [6] Video Conference

b) Video Section

When the video stream from the client come to, the MPEG4MCU will decode and then detect whether this terminal is in the position for video displaying or not. If it is, it copies the video data to the center video buffer in the appropriate position. Finally, it reads the video data from the center video buffer, encode, and then send back to the terminal.

Note that, at any time, the MPEG4MCU will display only 4 users in the video position. So the video data of users outside the video position are still decoded and then leave these resulted in the waste of the network incoming traffic and the CPU Load of the MPEG4MCU.

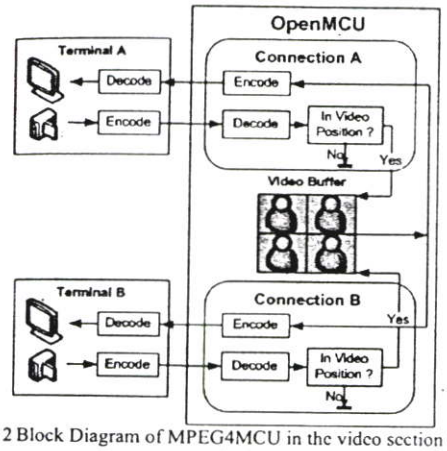


Figure 2 Block Diagram of MPEG4MCU in the video section with 2 terminals

C. Load balancing algorithms

In load-balancing schemes, the two most important policies are information policy and location policy. Information policy will collect load information of each host for location policy decision. When a new task location policy has will consider sending this task to proper host by load information from information policy.

GLOBAL: Every *P* seconds, *load information center* (LIC), receives load updates from all the other hosts and assembles them into *load vector,* which is then broadcast to all the other hosts. When the task arrives to host A, A will look at load vector to find the lowest load host and send the task to. [3]

RANDOM: This algorithm uses only local load information. When a job is found to be eligible for load balancing, it is sent to a randomly selected host. [3]

CENTRAL: In Global algorithm, placement decisions are made by each host using the load vector. In the [3] CENTRAL algorithm, the LIC acts as a central scheduler for all the hosts, in addition to receiving load information from the other hosts periodically. When a host decides that a job is eligible for load balancing, it sends a request to the LIC, together with the current value of its load. The LIC selects a host with the shortest queue length and informs the originating host to send the job there. It is assumed that the loads of all the hosts are known to the placement decision maker(s), with some delay. [3]

THRHLD: A number of randomly selected hosts, up to a limit *Lp* (probing limit), are polled when an eligible job arrives, and the job is transferred to the first host whose load is below load threshold *TI.* If no such host is found, the job is processed locally. [3]

LOWEST: This is similar to THRHLD except that, instead of using a threshold for the placement, a fixed number of hosts L_p are polled and the most lightly loaded host is selected. The probing stops if an empty host is found. [3]

Credit-based: The credit-based load-balancing model focuses on these two policies. We assign a numerical value, called credit, to every host. The credit indicates load of the host. The lower its credit has chance to take new task. The credit of each host changes in accordance with the behavior of the system and interaction [2]

III. OUR LOAD BALANCING ALGORITHM

There are two important policies, information policy and location policy in dynamic load balancing algorithm.

A) Information policy:

Load information from each host is very small size when compare with data packets in video conference. So we make decision to collect load from all MCUs to Load Balancer (LB) periodically.

The task of MCU is encoding and decoding video, mixing video and audio and managing floor and transfer data. Therefore load of MCUs is combining of computation load and communication load

$$C_{MCU} = C_{CP} + C_{NW} \quad (1)$$

C_{MCU} = MCU Credit

C_{CP} = Computation Credit

C_{NW} = Communication Credit

A.1) Computation credit is combining of central processing unit (CPU) and memory available because MCU task is encoding and decoding video, mixing video and audio that consume CPU resource and memory.

$$C_{CP} = f_{CP}(W_{cpu}A_{cpu}, W_{mem}A_{mem}) \quad (2)$$

A_{cpu} =Available Central Processor Unit (CPU) (%)

A_{mem} = Available Memory (%)

W_{cpu} = Weight number for CPU load

W_{mem} = Weight number for Memory used

f_{CP} = Computation Weight Function

MCU has variations of CPU powers, memory capacities and network performance. We use MCU_j to represent one of the MCUs in a heterogeneous network of MCUs [4]. We also use

variable P to represent the total number of MCUs in the system. The CPU weight of a workstation refers to its computing capability relative to the fastest workstation in a distributed system. The value of the CPU weight is less than or equal to 1. Since the CPU weight is a relative ratio, it can also be represented by the CPU speed of each node measured by millions of instructions per second (MIPS). If $V_{cpu}(j)$ is the speed of MCU_j in MIPS, j = 1, ..., the CPU weight can be expressed as follows:

$$W_{cpu}(j) = \frac{V_{cpu}(j)}{Max_{i=1}^{P} V_{cpu}(i)} \quad (3)$$

Similarly, the memory weight is calculated by comparing the memory sizes among the MCUs:

$$W_{mem}(j) = \frac{Mem_j}{Max_{i=1}^{P} Mem_i} \quad (4)$$

Where Mem_j is the amount of user available memory space in MegaBytes on MCU_j for j = 1, ..., P.

A.2) Communication credit is network bandwidth available and delay between host and client. Because each MCU is not on the same network bandwidth and speed between host and client will be different.

$$C_{NW} = W_{bw} A_{bw} + T \quad (5)$$

A_{bw} = Bandwidth usage (%)

T = Percentage of delay time between MCU and client (%)

W_{bw} = Weight number for bandwidth used

$$W_{bw}(j) = \frac{BW_j}{Max_{i=1}^{P} BW_i} \quad (6)$$

H.323 video conference acceptable delay is between 150 – 300 ms [8] so if MCU has delay from MCU and Client > 300 ms we will not choose it and we use 300 ms to convert delay time from ms to percent as below:

$$T = \begin{cases} (300 - delaytime^*) \times 0.33 & delaytime \le 300ms \\ -\infty & delaytime > 300ms \end{cases} \quad (7)$$

* Size of test packet = 64 kilobyte (Maximum size of ping command)

B) Location Policy:

When LB gets load information from each host, LB will convert received load to credit and collect in LB. Conversion method needs to weight the receiving load from each host.

Whenever a new conference room is created, LB will compute the collected credit of each MCU and select any MCU with the highest credit at that moment to serve the new room.

IV. EXPERIMENT AND RESULTS

Our experiment is consist of, Load balancer, MCU and Call generator as show in Fig 3.

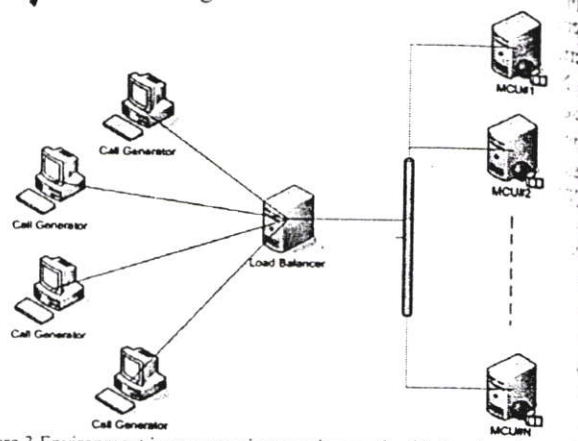


Figure 3 Environment in our experiment using our load balancing algorithm in Eq. (2)

A)Load Balancer(LB)

Load balancer (LB) is an opensoure application from www.codeproject.com that we enhance to collect memory and bandwidth used by the applications to calculate computation and communication credits. LB is developed using Microsoft Visual C++. NET 2003 with winpcap library version 3.01.

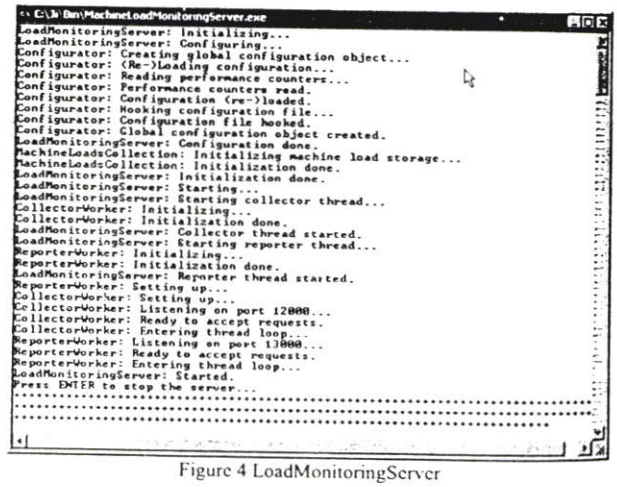


Figure 4 LoadMonitoringServer

B) MCU

We install MPEG4MCU from [1] with no Gatekeeper on several machines as follows:

1) MCU1 = Pentium D CPU 3.0GHz, Memory = 512 MB and Bandwidth = 100 Mbps
2) MCU2 = Pentium 4 CPU 2.8 GHz, Memory = 512 MB and Bandwidth = 10 Mbps
3) MCU3 = Pentium M CPU 1.4 GHz, Memory = 768 MB and Bandwidth = 100 Mbps
4) MCU4 = Celeron 2.6 GHz, Memory = 384 MB and Bandwidth = 100 Mbps

C) Call generator

We use call generator from [1] that can send and receive audio and video with following parameters:

Video codec = MPEG-4
Frame rate = 10 Frames/sec
Audio codec = G.711-uLaw-64k (PCM)
Audio Sampling rate = 8 kHz
Users per room = 4 users per room

Figure 5 Netmeeting with MPEG4MCU [1]

D) Results

In order to show the effectiveness of our load balancing algorithm, the results are divided into several subsections.

D1) Without Load Balancer

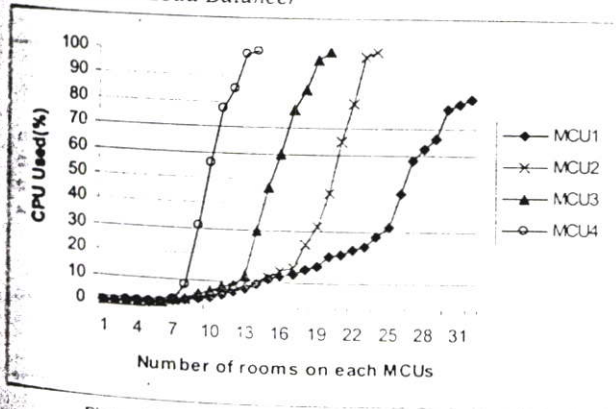


Figure 6 Percent CPU Used on each MCU with no LB

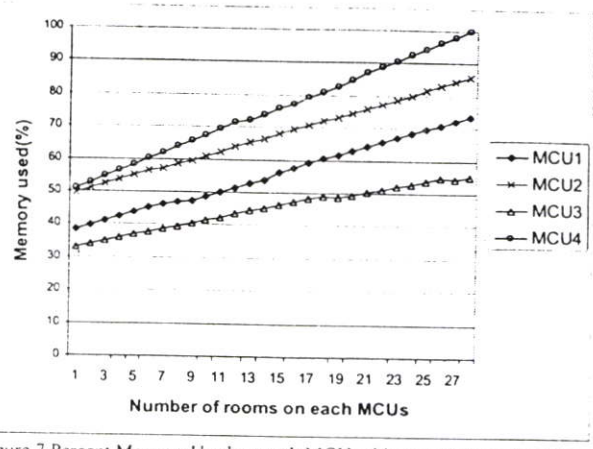


Figure 7 Percent Memory Used on each MCU with no weighting function

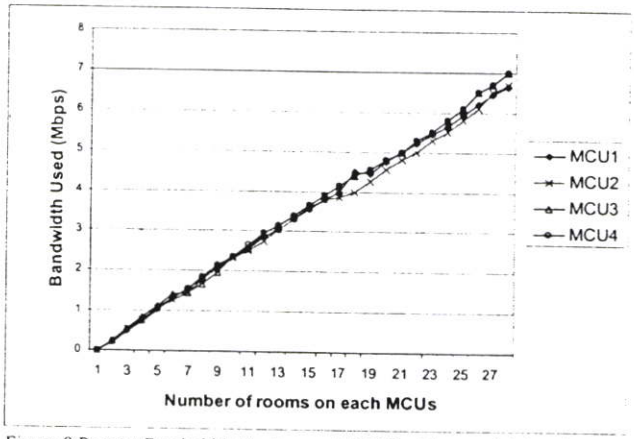


Figure 8 Percent Bandwidth Used on each MCU with no weighting function

Firstly, we do not use LB in the first experiment to view the trend of CPU, Memory and bandwidth used on each MCU. In Fig. 6, when the number of rooms increases on each MCU, %CPU used of each MCU grows slightly. But when the number of rooms reaches its critical point, %CPU used of each MCU grows exponentially. On the contrary, % memory used and bandwidth used in Fig. 7 and 8, respectively still grow linearly.

D2) With Load Balancer but no weighting function

Secondly, we use LB without any weighting function. Hence, the results are shown in Fig. 9, 10 and 11. Initially, both MCU1 and MCU3 are chosen by LB (without weighting function) because MCU1 has the fastest CPU and MCU3 has the biggest memory capacity. When the total number of rooms has reached 26, %CPU used of MCU3 grows exponentially but LB still chooses MCU3 because % memory used of MCU3 is still less than others. So MCU3 has been chosen until %CPU used reaches 100 (when %CPU used reaches 100, LB must not choose that MCU). When the number of rooms reaches 43, MCU4 will be chosen because bandwidth of MCU4 is more than MCU2 but CPU speed of MCU4 is slow that makes %CPU used grows very fast.

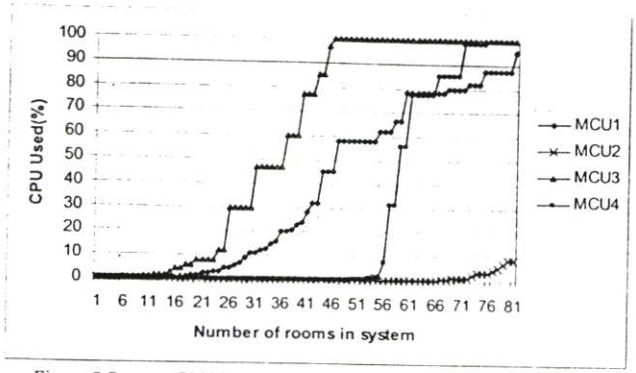


Figure 9 Percent CPU Used on each MCU with LB and no weighting function

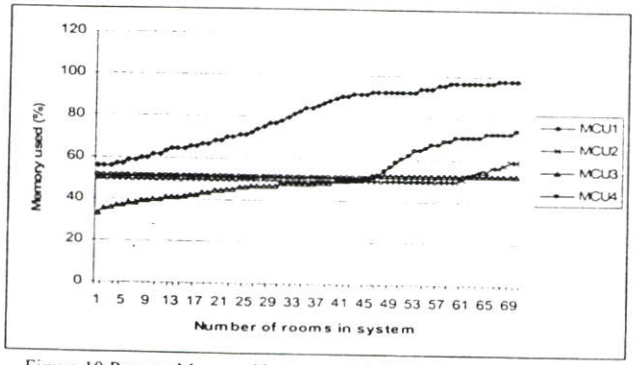


Figure 10 Percent Memory Used on each MCU with LB and no weighting function

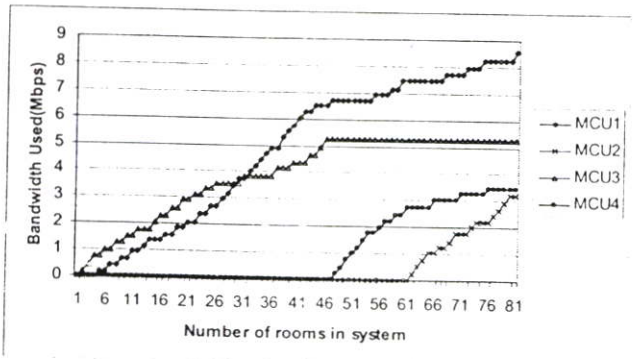


Figure 11 Bandwidth Used in Mbps on each MCU with LB and no weighting function

D3) With Load Balancer and weighting function

In Fig. 9, when the number of rooms reaches 81, %CPU used of MCU1, MCU2 and MCU3 reach 100 and %CPU used of MCU2 just only 10. That is not balanced, so we use weighting function in LB. As shown in Fig. 6 and 7, %CPU used grows exponentially but %Memory used grows linearly so we power A_{cpu} by 2 to raise the load credit suitable for %CPU used growth where (A_{cpu} = 100 - %CPU used).

$$f_{CP}(W_{cpu}A_{cpu}, W_{mem}A_{mem}) = W_{cpu}A_{cpu}^2 + W_{mem}A_{mem} \qquad (8)$$

Finally, we use weighting function (f_{CP}) according to Eq.(8). It can be noticed in Fig. 12, 13, and 14 when we start adding conference rooms until 23 rooms into the system, the LB chooses only MCU1 because MCU1 has the fastest CPU. Then, the LB chooses MCU2 together because of low % CPU used. At 46 rooms in the system, the LB just chooses MCU3 because MCU1 and MCU 2 are quite loaded. Eventually, when Room 70 is added, it is redirected to MCU4 which has the slowest CPU but the least %CPU used.

After we use LB with weighting function, CPU, memory and bandwidth used of all MCUs gradually increase together and no MCU has critical CPU, memory and bandwidth used at anytime.

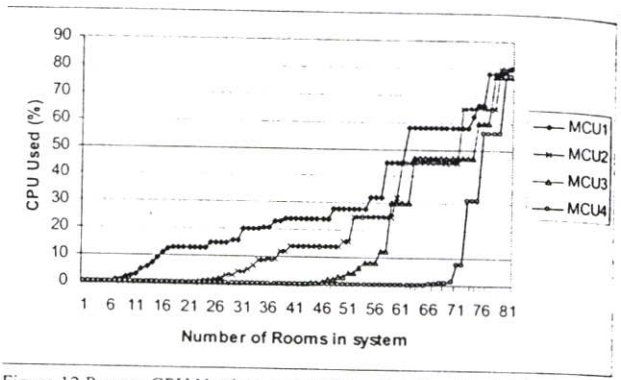


Figure 12 Percent CPU Used on each MCU with LB and weighting function

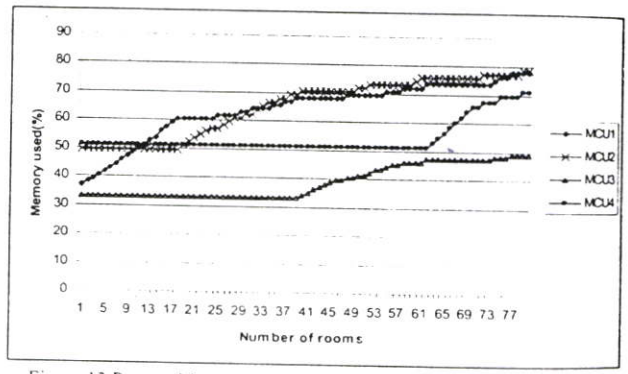


Figure 13 Percent Memory Used on each MCU with LB and weighting function

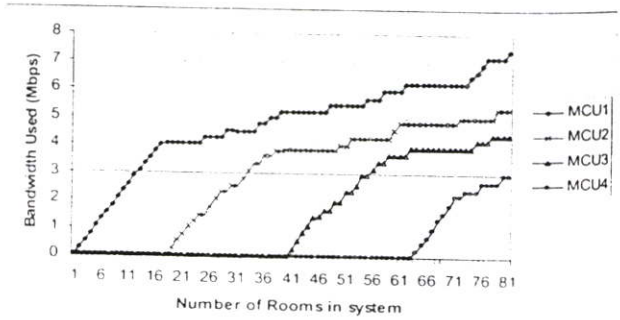


Figure 14 Bandwidth Used in Mbps on each MCU with LB and weighting function

VI. CONCLUSION

This paper presents the load balancing algorithm for videoconference multipoint control unit (MCU) to manage distributed MCUs in video conference system. The criterions are not only computation load (%CPU load and %memory used) on the MCUs but also the communication load between MCU and client. We take into account the computation and communication loads and weight them properly to make decision which MCU is the most suitable one to serve the new arriving videoconference room.

For future work, we will do more experiments by setting up variety of MCUs with different types of CPU, different memory sizes and emulated Internet using NISTNET [9] as shown in Fig. 15. Hence, we will measure CPU Load, memory used, and bandwidth used in order to compute the proper weighting function for different MCU specification and different network environments. Finally, we will compare our algorithm with others.

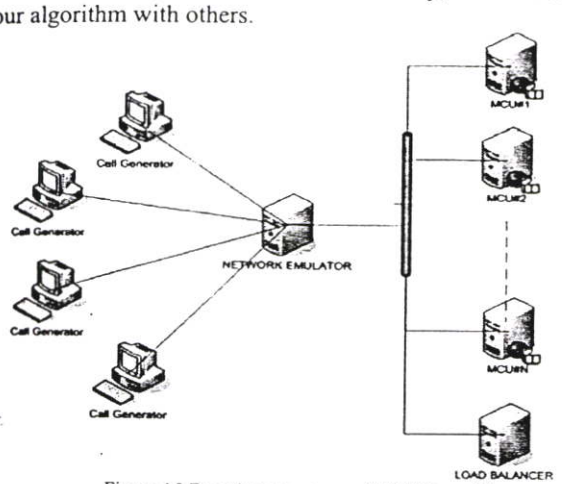


Figure 15 Experiment system with NIST net [9]

REFERENCES

[1] C. Traiperm and S. Kittitornkun, "High Performance Video Conference Multipoint Control Unit with MPEG-4 Codec", ECTI-Con 2005, May 12-13, 2005

[2] Ka-Po Chow and Yu-Kwong Kwok, "On Load Balancing for Distributed Multiagent," *IEEE Transactions on Computing Parallel and Distributed Systems*, 2002, pp. 787-801

[3] Songnian Zhou, " A Trace-Driven Simulation Study of Dynamic Load balancing," *IEEE Transactions* on *Software Engineering*, Volume 14, Issue 9, Sept. 1988

[4] Li Xiao, Xiadong Zhang and Yanxia Qu, "Effective load sharing on heterogeneous networks of workstations," *14th International Proceeding Parallel and Distributed Processing Symposium, 2000. IPDPS 2000*

[5] T. Kawasaki and K. Okamura, "Evaluation on Scalability of Conference System using Request-Routing," *18th International Conference on Advanced Information Networking and Applications (AINA'04)*, 2004

[6] R. Venkatesha Prasad, H. N. Shankar and H. S. Jamadagni, S. Vijay, "Server Allocation Algorithm for VOIP Conferencing," *Distributed Frameworks for Multimedia Applications*, 2005.

[7] OpenH323 Project, http://www.openh323.org

[8] Prasad Calyam, Mukundan Sridharan, Weiping Mandrawal and Paul Schopis, "Performance measurement and analysis of H.323 traffic," *PAM 2004 : passive and active network measurement*, Antibes les Pin, Frances, 19-20 April 2004

[9] Mark Carson and Darrin Santay, NIST Net: a Linux-based network emulation tool, Computer Communication Review (ACM SIGCOMM), 33(3):111-126, 2003.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายจิรายุ เจริญวิริยะภาพ
ประวัติการศึกษา	สำเร็จการศึกษาระดับปริญญาตรี หลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ จากคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2546
งานวิจัยที่สนใจ	Load balancing algorithm, Video conferencing, H323, Multipoint Control Unit