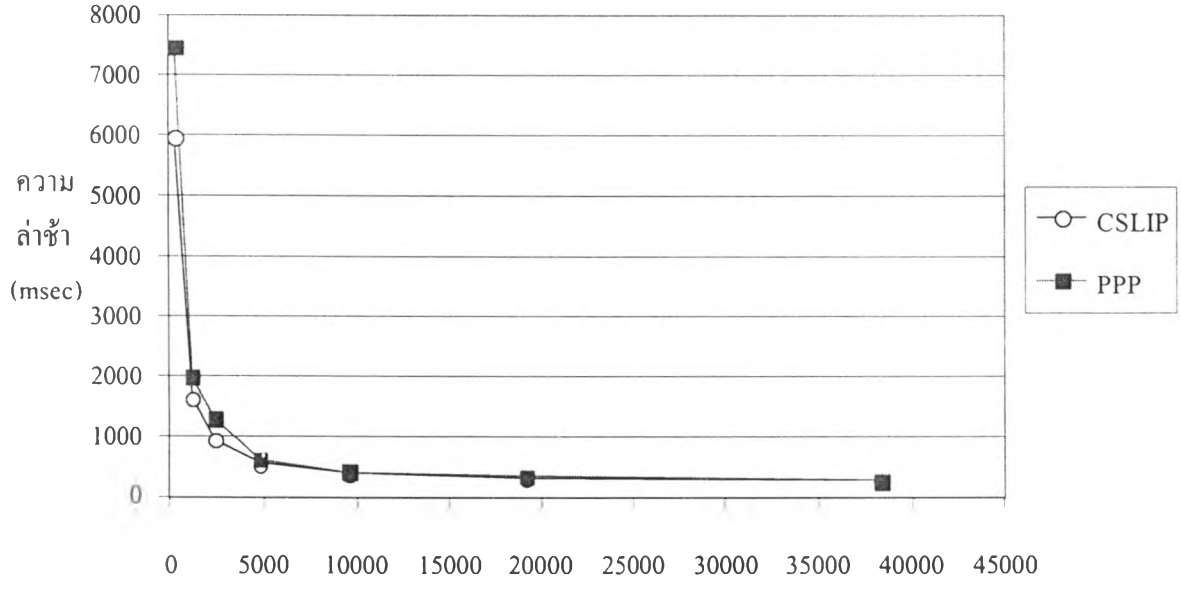
บทที่ 4

ผลการวิจัย

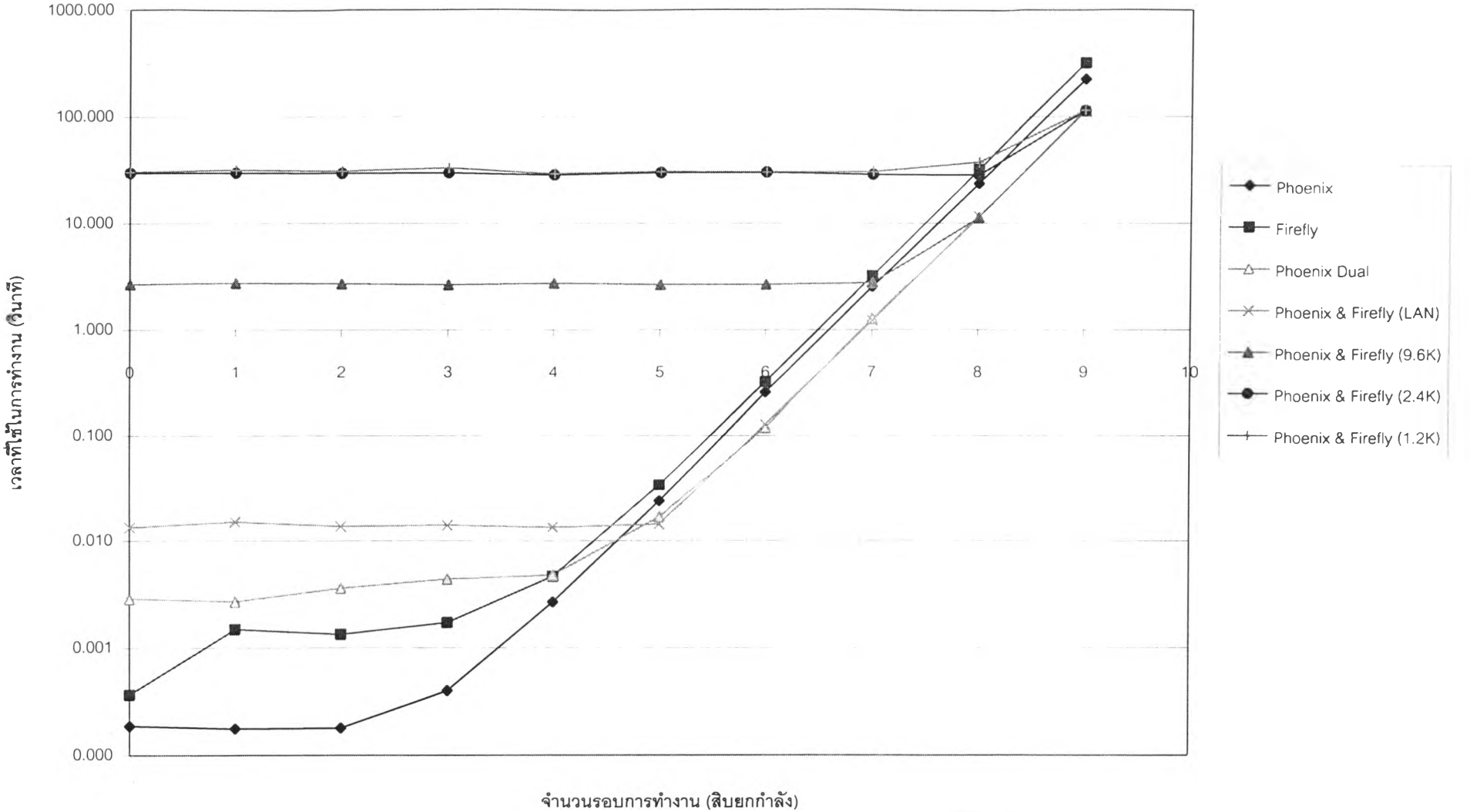
4.1 ผลการทดสอบโพรโตคอล CSLIP และ PPP

จากผลการทดสอบข้อ 3.4.1 เราสามารถแสดงได้เป็นกราฟดังนี้

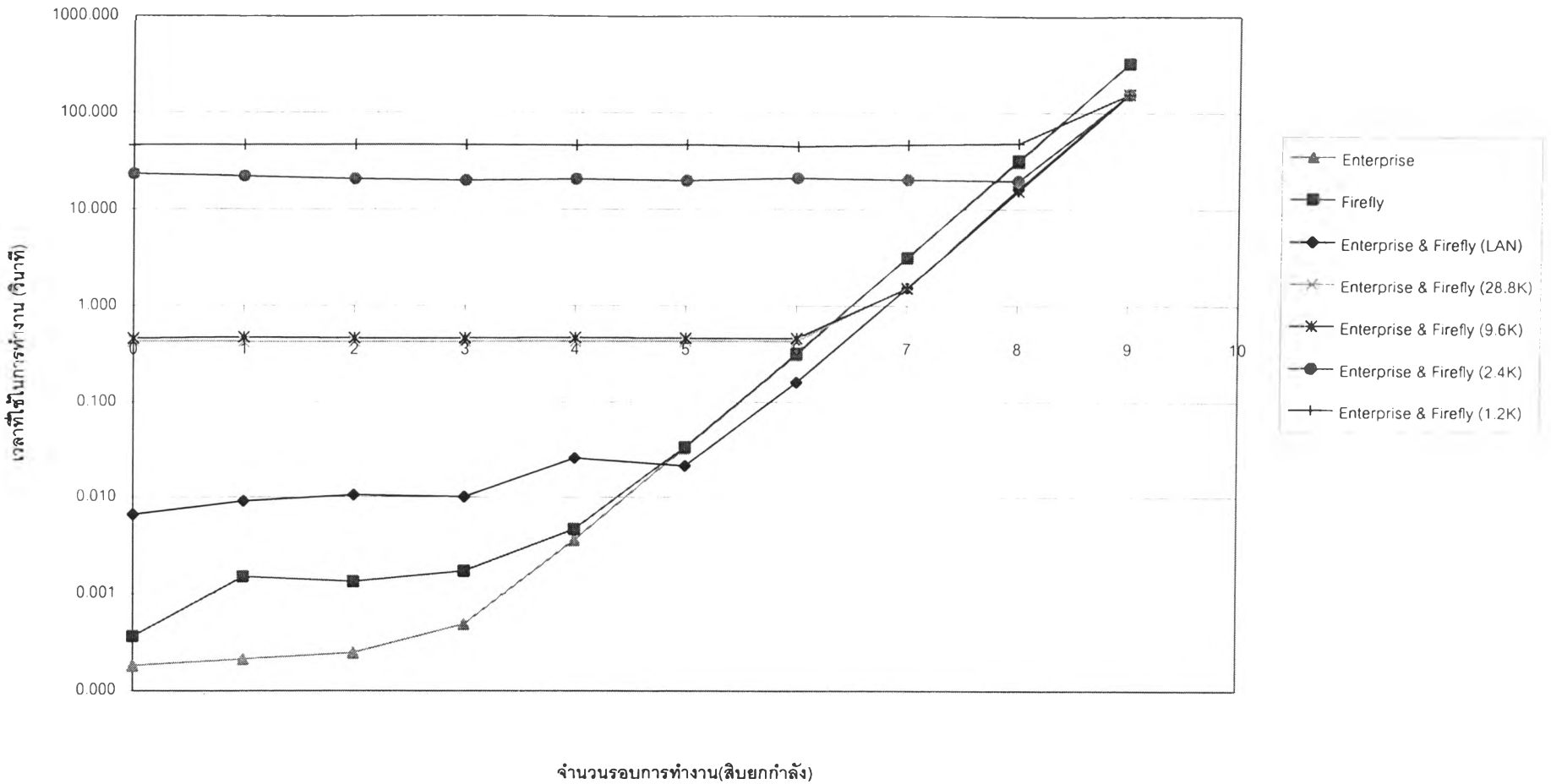


รูป 4.1 กราฟเปรียบเทียบความล่าช้าระหว่างโพรโตคอล CSLIP และ PPP

จากกราฟในรูปที่ 4.1 พบว่าประสิทธิภาพของการใช้งานโพรโตคอล CSLIP และ PPP มีความล่าช้าใกล้เคียงกัน โดยโพรโตคอล CSLIP สามารถทำงานอย่างมีประสิทธิภาพได้ดีที่ความ เร็วของการเชื่อมต่อต่ำมาก แต่เนื่องจากระบบเครือข่ายภายในมหาวิทยาลัยสนับสนุนการใช้โพรโตคอลชนิด PPP เพียงชนิดเดียวและผลต่างจากโพรโตคอล CSLIP มีค่าไม่มาก ทำให้ในวิทยานิพนธ์ฉบับนี้ทำการทดสอบใช้โพรโตคอล PPP เพียงชนิดเดียว



รูป 4.2 กราฟแสดงความสัมพันธ์ระหว่างจำนวนรอบการทำงานกับเวลาที่ใช้ในการทำงานระหว่างเครื่องฟินิกซ์และไฟร์ฟลาย



รูปที่ 4.3 กราฟแสดงความสัมพันธ์ระหว่างจำนวนรอบที่ใช้ในการทำงานกับเวลาที่ใช้ในการทำงานระหว่างเอนเตอร์ไพรซ์และไฟร์ฟลาย

4.2 ผลการทดสอบระบบเอ็มพีไอบนระบบเครือข่ายระยะไกล

จากการทดลองที่ 3.4.2 สามารถแสดงผลได้เป็นกราฟ 4.2 และ 4.3 โดยแสดงความสัมพันธ์ระหว่างการทำงานของจำนวนรอบการทำงานบนความเร็วในการเชื่อมต่อต่างกันเมื่อจำนวนรอบในการทำงานเพิ่มขึ้น เวลาที่ใช้ในการทำงานต่อจำนวนรอบมีแนวโน้มเพิ่มขึ้นโดยระยะห่างระหว่างเส้นกราฟเกิดจากความล่าช้าในการส่งข้อมูลด้วยความเร็วของการเชื่อมต่อที่แตกต่างกัน เมื่อจำนวนรอบในการทำงานเพิ่มมากขึ้น ผลของเวลาการส่งข้อมูลมีผลต่อเวลาใช้ในการทำงานโดยรวมลดน้อยลง

4.3 ผลกระทบของระบบเครือข่ายระยะไกลที่มีต่อระบบเอ็มพีไอ

จากการทดสอบของสตีเว่น[23] พบว่าแสดงถึงปัญหาในข้อ 3.4.3 ซึ่งเป็นปัญหาเกี่ยวกับระบบเครือข่ายมีดังนี้

1. แมสเซจซ้ำซ้อน (duplicate message)

แมสเซจภายในระบบเครือข่ายระยะไกล อาจเกิดการซ้ำซ้อนขึ้นเนื่องจากการทำงานผิดพลาดภายในตัวอุปกรณ์เอง หรือเนื่องจากการรบกวนระบบจากสิ่งแวดล้อมภายนอก

2. แมสเซจมาถึงล่าช้า (delay message)

ความกว้างของแถบสัญญาณ และระยะทางระหว่างต้นทางและปลายทาง ทำให้เกิดความล่าช้าของแพคเก็ตนั้นๆ เนื่องจากความเร็วในสื่อที่ใช้ระบบเครือข่ายตามจุดต่างๆ มีความแตกต่างกัน

3. แมสเซจไม่เรียงลำดับ (unsequence message)

เนื่องจากแมสเซจภายในระบบเครือข่ายระยะไกลต้องผ่านจุด (hop) หลายจุดในเส้นทาง ข้อมูลที่ส่งไปก่อนไปถึงปลายทางหลังข้อมูลที่ส่งไปทีหลัง ทำให้ข้อมูลที่มาถึงปลายทางไม่เรียงตามลำดับตามการส่ง

4. แมสเซจสูญหาย (lost message)

แมสเซจภายในเครือข่ายระยะไกลอาจมีการสูญหายเนื่องจากระบบเครือข่ายล่ม ระบบเครือข่ายคับคั่งเนื่องจากจำนวนของแพคเก็ตมาก หรือระบบเครือข่ายถูกรบกวนเนื่องจากปัจจัยภายนอก เช่นสนามแม่เหล็ก หรือมีการเสียหายทางกายภาพของสายสัญญาณเอง

4.4 สูตรที่ใช้คำนวนหาความเหมาะสมในการใช้งานระบบเอ็มพีไอบนระบบเครือข่ายระยะไกล

จากการศึกษาเราสามารถสรุปได้เป็นสูตรเพื่อใช้ในการคำนวนหาความเหมาะสมในการนำโปรแกรมมาใช้งานได้ดังนี้

1. $T_c = C_a + C_bD_C + C_cD_M + D_P$

เป็นสูตรเริ่มต้นจากสมการการทำงานทั่วไปบนระบบเครือข่ายเกิดจากเวลาจากเวลาที่ใช้ในการคำนวณ เวลาจากการเติมข้อมูลลงไปในสาย และเวลาเนื่องจากความล่าช้าเนื่องจากระยะทางรวมกัน จากระบบทดสอบเวลาที่มีผลต่อระบบคือเวลาที่เกิดเนื่องจากการทำงานของฝั่งที่ใช้เวลามากที่สุด จึงได้เป็นสูตรดังนี้

$T_c = C_a + \text{Max}\ (C_bD_C\ (i) + C_cD_M\ (i) + D_P(i))$

แทนค่า $D_C(i)$ ด้วยความล่าช้าในการทำงานต่อหนึ่งรอบคูณด้วยจำนวนรอบในการทำงาน

แทนค่า $D_M(i)$ด้วยจำนวนข้อมูลที่ส่งหารด้วยความเร็วในการส่ง

ดังนั้นจะได้สมการสุดท้ายคือ

2. $T_e = C_a + \text{Max}(C_b\ D_L(i) * L(i) + C_cS_D(i) / M_l\ (i) + D_P(i))$

โดยที่ตัวแปรต่างๆมีความหมายดังนี้

T_c = เวลาที่ใช้ในการทำงาน (วินาที)

C_a = สัมประสิทธิ์เนื่องจากเวลาที่ใช้ในการเริ่มต้นระบบ หาได้จากการนำโปรแกรมมาทำงานโดยมีเฉพาะในส่วนการเริ่มต้นระบบ (เพื่อให้ค่าเวลาเนื่องจากการทำงานเป็นศูนย์)

C_b = สัมประสิทธิ์เนื่องจากความหน่วงของการคำนวณ หาได้จากการนำโปรแกรมมาทำงานโดยมีจำนวนรอบเพียงรอบเดียวที่ความเร็วต่าง ๆ บนเครื่องคอมพิวเตอร์CPU มากกว่า 1 ตัว

C_c = สัมประสิทธิ์เนื่องจากความหน่วงของค่าส่งข้อมูลเข้าไปในสาย หาได้จากจำนวนของข้อมูลที่ใส่เข้าไปในสาย หารด้วยความเร็วของการใส่ข้อมูลลงไปในสายสำหรับข้อมูลขนาด 1 ไบต์

D_C (i)= ความล่าช้าของการทำงานต่อหนึ่งรอบของโพรเซสที่ I (วินาที)

D_M(i)= ความล่าช้าเนื่องจากการส่งข้อมูลไปในสายระหว่างโพรเซสที่ I กับโพรเซสแม่ (วินาที)

D_P(i)= ความล่าช้าเนื่องจากระยะทางระหว่างโพรเซสที่ I กับโพรเซสแม่(วินาที)

D_L(i) = ความล่าช้าเนื่องจากการทำงานหนึ่งรอบของโพรเซส (วินาที)

L(i)= จำนวนรอบของการทำงานของโพรเซสที่ I

$S_D(i)$= ขนาดของข้อมูลที่มีการส่งระหว่างโพรเซสที่ I กับโพรเซสแม่ (ไบต์)

$M_I(i)$= ความเร็วของสื่อที่ใช้ในการเชื่อมต่อระหว่างโพรเซสที่ I กับโพรเซสแม่ (ไบต์ต่อวินาที)

4.5 ตัวอย่างการหาค่าสัมประสิทธิ์ของการทดสอบ

จากผลการทดลองที่ได้ เนื่องจากใช้โพรเซสเซอร์เพียงโพรเซสเดียวทำให้เวลาที่ใช้เกิดเนื่องจากโพรเซสที่ไม่ได้ทำหน้าที่เป็นตัวเริ่มต้นของระบบ ดังนั้นเวลาจึงขึ้นอยู่กับโพรเซสบนเครื่องไฟร์ฟลายเป็นหลัก จากข้อสรุปข้างต้นสามารถหาค่าสัมประสิทธิ์ต่างๆได้ดังนี้

1. การหาค่า C_a

เมื่อพิจารณาจากผลการทดลองพบว่า หากโปรแกรมทำงานโดยไม่มีจำนวนรอบการทำงานและความเร็วภายในสายสัญญาณ ทำให้ค่านิพจน์ที่ 2 และ 3 จากสูตร 1 เป็นศูนย์ เราได้ค่าจากรูปที่ 4.2 ว่า ค่าของ $C_a = 0.00008$

2. การหาค่าสัมประสิทธ์ C_c

พิจารณาจากกราฟ 4.1 พบว่าค่าความล่าช้าของสัญญาณเนื่องจากการใช้โพรโตคอล PPP จากการส่งข้อมูล 64 ไบต์มีค่าเท่ากับ 0.60625 วินาที ที่ความเร็วของสื่อ 480 ไบต์ต่อวินาที และ 0.319100 วินาทีที่ความเร็ว 1920 ไบต์ต่อวินาที แก้สมการที่ 1 จะได้ค่าสัมประสิทธิ์ C_c เท่ากับ 2.8722

3. การหาค่า D_P

แทนค่า C_c ที่ได้จากสมการที่ 1 จะได้ค่า $D_P = 0.60625 - 2.8722 * .13334 = 0.22327$

4. การหาค่า C_b

พิจารณาการโปรแกรมที่ทำงาน 1 รอบและการทำงานโดยความเร็วในการเชื่อมต่อเป็น 960 ไบต์ต่อวินาที ค่าของ $S_D(i)$ มีค่าเท่ากับ 8 ไบต์ ความเร็วในการทำงานต่อหนึ่งรอบมีค่า 0.004918 แทนค่าในสมการ $1.56428 = 0.0008 + C_b (0.004918 * 1) + 2.8722 * 8/960 + 0.22327$ ดังนั้น สัมประสิทธิ์ C_b มีค่าเท่ากับ 267.8070

ดังนั้นเราจะได้สูตรเพื่อใช้ในการหาค่าเพื่อตัดสินใจในการใช้งานสำหรับระบบเอ็มพีไอบนระบบเครือข่ายระยะไกลจากแทนค่าในสมการที่ 2 ดังนี้

$T_c = 0.0008 + Max (267.807 * D_L(i) * Li + 2.8722 * S_D(i)/M_I(i) + 0.22327)$; สำหรับการทำงานระหว่างเครื่องฟินิกซ์และไฟร์ฟลายที่มีความเร็วของสายส่งที่ความเร็ว 960 ไบต์ต่อวินาที

4.6 การแก้ไขปัญหาเนื่องจากระบบเครือข่ายทางไกลที่มีผลต่อระบบเอ็มพีไอ

เราสามารถแก้ไขปัญหาต่าง ๆ ที่เกิดขึ้นจากแมสเซจ เพื่อให้สามารถทำงานได้บนระบบเครือข่ายระยะไกลได้โดยการใช้โพรโตคอลทีซีพี/ไอพี ที่มีความสามารถในการรองรับความผิดพลาดเนื่องจากการทำงานของระบบเครือข่ายระยะไกลได้ แต่ต้องมีการปรับแต่งค่าทีซีพี/ไอพีซึ่งเป็นโพรโตคอลหลักที่ใช้ในระบบเครือข่ายอินเตอร์เนตเพื่อให้สามารถแก้ปัญหาเนื่องจากเครือข่ายระยะไกลได้ดังนี้

1. ปัญหาเนื่องจากขนาดของหน้าต่างมีจำกัด

ควรมีการแก้ไขให้มีขนาดของหน้าต่างมากกว่า 2^{16} วิธีการนี้สามารถกำหนดได้โดยอ้างอิงในสเกลแฟกเตอร์ เพื่อใช้คูณกับขนาดของหน้าต่างเพื่อให้ได้ขนาดของหน้าต่างที่แท้จริง โพรโตคอลทีซีพีกำหนดให้มีตัวเลือกวินโด้สเกล (Window Scale) ในการทำงานนี้

2. ปัญหาเนื่องจากแมสเซจซึ่งสะสมเนื่องจากกำลังรอการตอบรับ

ปัญหานี้สามารถแก้ได้โดย ส่งการตอบรับเฉพาะแมสเซจที่ได้รับแล้ว ข้อมูลจากผู้รับสามารถยืนยันเซกเม้นท์ที่ได้รับทั้งหมด ดังนั้นผู้ส่งก็ส่งเพียงเซกเม้นท์ที่สูญหายได้

โพรโตคอลเช่น VTMP[1] NETBLT[2] และโพรโตคอล RDP[25] มีการพัฒนาซีเล็คทิฟแอคโนเลจเม้นท์ (selective acknowledgements) โดยที่โพรโตคอล RDP แสดงให้เห็นว่าการไม่มีซีเล็คทิฟแอคโนเลจเม้นท์ จะทำให้อัตราการส่งใหม่เพิ่มขึ้นอย่างมากในระบบเครือข่ายที่ มีอัตราการสูญเสียของแพคเก็ตสูงและมีอัตราการล่าช้าสูง[29] และการจำลองเพื่อศึกษาถึงการเพิ่มซีเล็คทิฟแอคโนเลจเม้นท์ลงใน ISO transport protocol IPv4[28] ก็แสดงถึงประสิทธิภาพที่เพิ่มขึ้นเนื่องจากตัวเลือกนี้

3. ปัญหาการคำนวณค่าราวทริปไทม์

ทีซีพีกำหนดให้ผู้ส่งสามารถแนบค่าเวลาส่งให้แก่ทุก ๆ เซกเม้นท์ที่ส่งออกไปได้ โดยผู้รับตอบรับค่านี้โดยการส่งเวลาที่ใช้ในการส่งกลับไปทำให้ผู้ส่งสามารถคำนวณค่าราวทริปไทม์ได้ในการรับแต่ละครั้ง เนื่องจากการนำมาใช้งานส่วนใหญ่สนับสนุนการตอบรับในทุก ๆ หน้าต่างเท่านั้น แต่ไม่สนับสนุนการตอบรับทุก ๆ เซกเม้นท์ โดยตัวเลือกไทม์สแตมป์นี้ใช้ได้ดีในกรณีหน้าต่างมีจำนวน 8 เซกเม้นท์ในการตอบรับ และเมื่อมีขนาดของหน้าต่างใหญ่ขึ้นก็ต้องใช้วิธีการอื่นเช่น อัลกอริทึมของคานส์ (Karn's Algorithm)[11,23] และตัวเลือกทีซีพีไทม์สแตมป์ (TCP timestamps option)[12]

อัลกอริทึมของคานส์กล่าวว่าเมื่อเกิดการหมดเวลา (timeout) และเกิดการส่งข้อมูลซ้ำ (retransmission) ทำให้ไม่สามารถเปลี่ยนแปลงค่าอาร์ทีทีได้จากการตอบรับของแพคเก็ตข้อมูลที่ส่ง

ซ้ำกลับมาถึง เพราะว่าไม่สามารถทราบได้ว่าการส่งซ้ำในครั้งนั้นเป็นของข้อมูลแพคเก็ตใดจากปลายทางที่ตอบกลับมา ดังนั้นทุกครั้งที่มีการส่งข้อมูลซ้ำไม่มีการคำนวณค่าของอาร์ทีทีใหม่จนกระทั่งได้รับค่าจากแพคเก็ตที่ไม่ได้มีการส่งข้อมูลใหม่จึงทำการคำนวณต่อไป

ตัวเลือกทีซีพีไทม์สแตมป์เป็นการเพิ่มส่วนขยายภายในส่วนหัวของทีซีพีโดยระเบียน TS Value (TSval) จะเป็นเวลาของการส่งปัจจุบันของไทม์สแตมป์คล้อก (timestamp clock) ระเบียน TS Echo Reply (TSecr) เป็นเวลาจากของการส่งจากผู้รับ ถ้า TSecr ไม่มีค่าของระเบียนนี้จะถูกตั้งค่าเป็นศูนย์

4. การตรวจสอบการคงอยู่ของแม่ข่ายและลูกข่าย[23]

การใช้ตัวเลือกทีซีพีคีฟอไลฟไทม์เมอร์ (Keepalive Timer) ภายในทีซีพีเพื่อเป็นการรับรองว่าแม่ข่าย (server) และลูกข่าย (client) นั้นยังทำงานและสามารถเข้าถึงได้ตามระยะเวลาที่กำหนด ค่าคีฟอไลฟนี้จะถูกกำหนดโดยแม่ข่าย โดยมาตรฐานตั้งไว้ไม่น้อยกว่า 2 ชั่วโมงและมีการตรวจสอบโดยการส่งซ้ำไป 10 ครั้ง เมื่อผ่านช่วงเวลาคีฟอไลฟโดยส่งทุก ๆ 75 วินาที โดยค่ามาตรฐานดังกล่าวนี้สามารถเพิ่มหรือลดได้

จากการแก้ปัญหาดังกล่าวสามารถสรุปปัญหาและวิธีการแก้ไขปัญหาได้ตารางที่ 4.1

ปัญหา	วิธีแก้ไขปัญหา
ขนาดของหน้าต่างมีจำกัด	ตัวเลือกวินโด้สเกล
แมสเซจสะสมเนื่องจากกำลังรอการตอบรับ	ตัวเลือกซีเล็คทิฟแอคโนเลจเม้นท์
การคำนวณค่าอาร์ทีที	อัลกอริทึมของคานส์ ตัวเลือกทีซีพีไทม์สแตมป์
การตรวจสอบการคงอยู่ของแม่ข่ายและลูกข่าย	ตัวเลือกทีซีพีคีฟอไลฟไทม์เมอร์

ตารางที่ 4.1 ปัญหาและวิธีแก้ปัญหาเนื่องจากระบบเครือข่ายระยะไกล

4.7 การออกแบบเพื่อเพิ่มเติมตัวเลือกในทีซีพี [11]

ในการออกแบบ ตัวเลือกสำหรับทีซีพีใหม่นี้ จะต้องคำนึงถึงผลกระทบต่อการออกแบบที่มีมาแล้ว ตัวเลือกสำหรับทีซีพีนี้จะเรียกว่าเป็น ตัวเลือกเริ่มต้น เช่น กำหนดในส่วน SYN segment ซึ่งอาจไม่ได้นำมาใช้งานในโปรแกรมหนึ่ง ๆ และไม่ควรมีผลกระทบในกรณีที่ลูกข่ายไม่รู้จัก

ตัวเลือกนั้น อีกสิ่งหนึ่งคือควรระวังในสิ่งที่คาดไม่ถึงเนื่องจากตัวเลือกที่ไม่ได้เริ่มทำงาน บางครั้งอาจทำให้ระบบทีซีพีบางส่วนหยุดการทำงานได้ นอกจากนี้ในการเพิ่มเติมตัวเลือกที่ยังไม่ได้เริ่มทำงานอาจส่งเพียงตัวเลือกเริ่มต้นที่ทั้งสองฝั่งรู้จัก การกระทำนี้ควรกำหนดให้ตัวโพรโตคอลสามารถตัดสินใจเมื่อเปิดการเชื่อมต่อว่าส่วนหัวของทีซีพีควรมีขนาดเท่าไหร่ก่อนการส่งเพื่อตั้งค่าของระบบ

4.8 ตัวเลือกทีซีพีบนระบบปฏิบัติการโซลาริส

ระบบปฏิบัติการโซลาริสรุ่น 2.4 และ 2.5 ยังไม่สนับสนุนการใช้งานตัวเลือกทีซีพีต่างๆที่กล่าวข้างต้น การสนับสนุนตัวเลือกวินโด้สเกลและดีฟอไลฟไทม์เมอร์พบในระบบโซลาริส 2.6 ขึ้นไป ขณะทำการวิจัย ผู้วิจัยไม่สามารถหาระบบดังกล่าวมาใช้ในการทดสอบได้ สำหรับตัวเลือกซีเล็คทิฟแอคโนเลจเม้นท์ และทีซีพีไทม์สแตมป์ ระบบปฏิบัติการโซลาริสที่มีในปัจจุบันไม่สนับสนุน

4.9 ข้อแนะนำในการใช้ระบบเอ็มพีไอบนเครือข่ายระยะไกล

1. ควรตรวจสอบการเริ่มต้นการทำงานของระบบเอ็มพีไอทุกครั้งเมื่อเริ่มทำงานโดยใช้คำสั่ง MPI_INITIALZED ในการทดสอบความสำเร็จของการทำงานคำสั่ง MPI_INIT และ MPI_ABORT ใช้ในการยกเลิกการทำงานในกรณีที่การทำงานเกิดความผิดพลาดขึ้น

ตัวอย่างในการใช้งาน

```
        CALL MPI_INIT(ierr)
        CALL MPI_Comm_rank( comm, rank, ierr )
        CALL MPI_Comm_size( comm, size, ierr )
! Checking for complete start MPI. If fail then abort MPI program
        CALL MPI_INITIALIZE( flag , ierr );
        IF ( .not. flag) THEN
            CALL MPI_ABORT ( comm, errorcode, ierr);
        END IF
< program code>
CALL MPI_FINALIZE(ierr);
END
```

2. ควรตรวจสอบการทำงานของระบบเอ็มพีไอในขั้นตอนการทำงานโดยใช้คำสั่งดังนี้

2.1 คำสั่ง MPI_TEST, MPI_TESTALL , MPI_TESTANY, MPI_TESTSOME ใช้ทดสอบความสำเร็จในการรับและส่งแบบนอน-บล๊อกกิ้งว่าสำเร็จหรือไม่

ตัวอย่างการใช้งาน

```
                CALL MPI_ISEND ( buffer, count, datatype, dest, tag,
$                                comm, request, ierr )
                <do another work>
10              CALL MPI_TEST( request, flag, status, ierr )
                IF (flag .EQ. FALSE) GOTO 10
```

2.2 คำสั่ง MPI_WAIT, MPI_WAITALL, MPI_WANITANY,MPI_WAITSOME ใช้หยุดรอการรับหรือส่งแบบ นอน-บล๊อกกิ้ง (non-blocking) ให้เสร็จก่อนจึงทำงานต่อไป

ตัวอย่างการใช้งาน

```
                CALL MPI_IRECV (..., comm, request(1), ierr )
                CALL MPI_IRECV (..., comm, request(2), ierr )
                CALL MPI_WAITALL(2,  request, flag, status, ierr )
<do work>
```

2.3 คำสั่ง MPI_IPROBE และ MPI_PROBE เพื่อตรวจสอบสถานะของแมสเซจว่าตรงกับที่กำหนดหรือไม่ก่อนใช้คำสั่ง MPI_RECV]

ตัวอย่างการใช้งาน

```
CALL MPI_COMM_RANK(comm, rank, ierr)
    IF (rank .EQ. 0) THEN
        CALL MPI_SEND(i, 1, MPI_INTEGER, 2, 0, comm, ierr)
    ELSE IF (rank .EQ. 2) THEN
        DO i=1, 2
```

```
! Checking type of receiving message wheter integer or real
              CALL MPI_PROBE(MPI_ANY_SOURCE, 0, comm, status, ierr)
              IF (status(MPI_SOURCE) = 0) THEN
100               CALL MPI_RECV(i, 1, MPI_INTEGER, MPI_ANY_SOURCE, 0 ,comm,
$                                 status,  ierr)
              ELSE
200               CALL MPI_RECV(i, 1, MPI_REAL, MPI_ANY_SOURCE, 0 ,comm,
$                                 status,  ierr)
              END IF
            END DO
         END IF
```

2.4 คำสั่ง MPI_CANCEL และ MPI_TEST_CANCELLED ใช้ในการยกเลิกคำสั่งรับและส่งแบบ non-blocking ใช้เพื่อป้องกันการรอที่ไม่สิ้นสุดของโพรเซส

ตัวอย่างการใช้งาน

```
CALL MPI_COMM_RANK(comm,  rank, ierr)
IF ( rank .EQ. 0) THEN
        CALL MPI_SEND(a, 1, MPI_CHAR, 1, tag, comm);
ELSE IF (rank .EQ. 1) THEN
        CALL MPI_IRECV(a, 1, MPI_CHAR, 0, tag, comm, request, ierr)
        CALL_CANCEL(request, ierr)
        CALL MPI_WAIT(request, status, ierr)
        CALL MPI_TEST_CANCELLED(status, flag, ierr)
        IF (flag .EQ. TRUE) THEN /* cancel succeeded – need to post new receive */
             CALL MPI_RECV(a, 1, MPI_CHAR, 0, tag, comm, request, ierr)
END IF
```

2.5 คำสั่ง MPI_SENDRECV และ MPI_SENDRECV_REPLACE ใช้สำหรับการส่งและรับข้อมูลต่อเนื่องกันและสามารถป้องกันการล๊อคตาย (dead lock) ได้

ตัวอย่างการใช้งาน

```
      SUBROUTINE exchng1 (a , nx, s ,e, comm1d, nbrbottom, nbrtop )
      INCLUDE "mpif.h"
      INTEGER nx, s, e
      DOUBLE PRECISION a(0:nx+1, s-1:e+1)
      INTEGER status(MPI_STATUS_SIZE), ierr
c
      CALL MPI_SENDRECV(
$           a(I,e), nx, MPI_DOUBLE_PRECISION, nbrtop, 0,
$           a(1, s-1), nx, MPI_DOUBLE_PRECISION, nbrbottom, 0,
$           comm1d, status, ierr)
      CALL MPI_SENDRECV(
$           a(1,s), nx, MPI_DOUBLE_PRECISION, nbrbottom, 1,
$           a(1, e+1), nx, MPI_DOUBLE_PRECISION, nbrtop, 1,
$           comm1d, status, ierr )
      RETURN
      END
```

2.6 ใช้คำสั่ง MPI_BARRIER เพื่อทำให้โพรเซสทั้งหมดหยุดรอ เป็นการรับประกันว่าโพรเซสทั้งหมดทำงานถึงจุดนี้แล้ว

ตัวอย่างการใช้งาน

```
c     This example use MPI_BARRIER for ensure that all process have reached the same
c     point in the code and are ready to proceed
CALL MPI_BARRIER( MPI_COMM_WORLD, ierr )
t1 = MPI_WTIME()
<do work>
CALL MPI_BARRIER( MPI_COMM_WORLD, ierr )
Total_time = MPI_WTIME() – t1
```

3. ควรใช้การทำงานแบบเพอร์ซิสเทนท์คอมมิวนิเคชั่นรีเควสท์ (Persistent Communication Requests) สำหรับการทำงานลักษณะเป็นการคำนวนแบบขนานภายใน (inner loop of a parallel computation) เพื่อลดโอเวอร์เฮด (overhead) ระหว่างโพรเซสและคอมมิวนิเคชั่นคอนโทรลเลอร์ (communication controller) โดยคำสั่งเหล่านี้ทำงานแบบนอน-บล๊อกกิ้งทั้งหมด คำสั่งเหล่านี้ได้แก่

3.1 MPI_SEND_INIT ใช้สร้างเพอร์ซิสเท้นท์คอมมิวนิเคชั่น (persistent communication) ของการส่งมาตรฐาน

3.2 MPI_RECV_INIT ใช้สร้างเพอร์ซิสแท้นคอมมิวนิเคชั่นของการรับมาตรฐาน

3.3 MPI_START , MPI_STARTALL ใช้เริ่มต้นการทำงานแบบเพอร์ซิสเท้นท์คอมมิวนิเคชั่น

3.4 ใช้คำสั่ง MPI_WAIT, MPI_WAITALL, MPI_TEST และ MPI_TESTALL ทดสอบสถานะของการส่งและรับ

ตัวอย่างการใช้งาน

```
! Create persistent requests
CALL MPI_SEND_INIT(B(1,1), n, MPI_REAL,  left, tag, comm, req(1), ierr)
CALL MPI_SEND_INIT(B(1,m), n, MPI_REAL,  right, tag, comm, req(2), ierr)
CALL MPI_SEND_INIT(A(1,0), n, MPI_REAL,  left, tag, comm, req(3), ierr)
CALL MPI_SEND_INIT(A(1,m+1), n, MPI_REAL,  right, tag, comm, req(4), ierr)
DO WHILE ( .NOT. coveraged)
    ! Compute boundary columns
    DO I = 1,n
        B(i, 1) = 0.25*(A(i-1, 1)+A(i+1, 1)+A(i, 0)+A(i,2))
        B(i, m) = 0.25*(A(i-1, m)+A(i+1, m)+A(i, m-1)+A(i,m+1))
    END DO
    ! Start commnication
    CALL MPI_STARTALL(4,req, ierr)
    ! Compute interior
    DO i=2, m-1
        DO j=1, n
```

```
                B(i, j) = 0.25*(A(i-1, j)+A(i+1, j)+A(i, j-1)+A(i,j+1))
            END DO
        END DO
        DO j=1,m
            DO i=1,n
                A(i, j) = B(i, j)
            END DO
        END DO
! Complete communication
        CALL MPI_WAITALL(4, req, status, ierr)
```