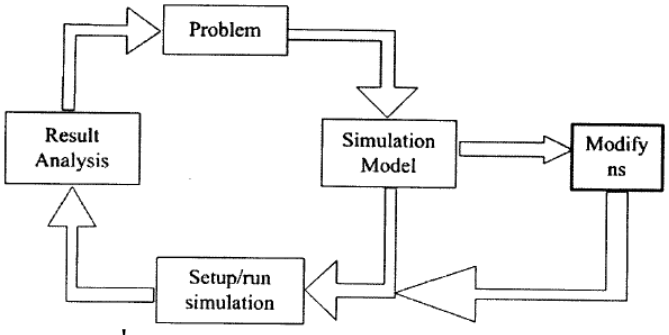
ภาคผนวก ก.
เครื่องมือในการสร้างแบบจำลอง

เนื่องจากเป็นการยากที่จะใช้ระบบจริงของโทรศัพท์เคลื่อนที่ยุคที่ 3 ในการทดสอบสมรรถนะการทำงานของโปรโตคอล นั้นดังทางเลือกที่เป็นไปได้ คือ การสร้างแบบจำลองที่สามารถจำลองทำงานของระบบได้ โดยใช้โปรแกรมสร้างแบบจำลองทางเครือข่ายคอมพิวเตอร์ที่มีให้เลือกมากมายในปัจจุบัน แต่สำหรับงานวิจัยนี้เลือกใช้โปรแกรมจำลอง Network Simulator หรือ ns ด้วยเหตุผลที่ โปรแกรมจำลอง ns เป็นโปรแกรมจำลองที่พัฒนาขึ้นแบบโอเพนซอร์สและใช้แนวคิดของการโปรแกรมเชิงอ็อบเจกต์ จึงเปิดโอกาสให้นักพัฒนาได้สร้างแบบจำลองของโปรโตคอลทางเครือข่ายคอมพิวเตอร์ แล้วนำโปรโตคอลเหล่านั้นมาทำงานเป็นระบบร่วมกับโปรโตคอลอื่น ๆ เพื่อการศึกษาและเปรียบเทียบสมรรถนะหรือข้อจำกัดของโปรโตคอลได้ในสภาพแวดล้อมที่กำหนดขึ้นได้

ในปัจจุบัน ns สามารถทำงานได้ทั้งบนระบบปฏิบัติการ Linux และ Windows และสนับสนุนแบบจำลองทางเครือข่ายจำนวนมาก เช่น แบบจำลองของลิงค์ ไม่ว่าจะเป็นแบบมีสายหรือไร้สาย แบบจำลองของโหนดทั้งที่เป็นคอมพิวเตอร์ เราเตอร์หรือสวิตซ์และโปรโตคอลต่าง ๆ ซึ่งสร้างขึ้นอย่างเป็นลำดับชั้น อาทิ ในชั้นทรานสปอร์ตก็จะประกอบด้วย UDP, TCP และในชั้นโปรแกรมประยุกต์ที่ประกอบด้วยโปรโตคอล Telnet, FTP และ HTTP เป็นต้น

ในกรณีของการจำลองเรื่องใด ๆ ที่โปรแกรม ns ได้มีการจัดเตรียมแบบจำลองไว้ให้แล้ว เช่น FTP สามารถทำการจำลองได้เลย แต่หากการจำลองของโปรโตคอลใดที่ ns ยังไม่ได้เตรียมแบบจำลองไว้ให้ เช่น โปรโตคอลของระบบโทรศัพท์เคลื่อนที่ยุคที่ 3 ที่จำเป็นต้องใช้ในงานวิจัยนี้ ก็จำเป็นที่จะต้องสร้างหรือปรับแต่ง ns ก่อนจึงจะทำการจำลองได้ ดังรูปที่ ก.1 ซึ่งได้แสดงถึงวัฏจักรของการจำลองด้วย ns โดยทั่วไปโมดูลที่แทนแบบจำลองของโปรโตคอลใน ns จะต้องสร้างด้วยภาษา C++ เนื่องจากต้องการความเร็วในการประมวลผล แต่ในส่วนของการกำหนดสภาวะสำหรับการจำลองจะเขียนด้วย OTcl สคริปต์เพราะสามารถเขียนโปรแกรมได้เร็วกว่า



รูปที่ ก.1 แสดงวัฏจักรของการจำลองด้วย ns

เนื่องจากโปรแกรม ns ยังไม่ได้มีการจัดเตรียมแบบจำลองสำหรับเครือข่าย UMTS ไว้ก่อน ดังนั้นจึงจำเป็นต้องมีการสร้างโมดูลเพิ่มเติมเพื่อปรับแต่งโปรแกรม ns ให้สามารถจำลองการทำงานของโปรโตคอลของระบบ UMTS ตามที่งานวิจัยต้องการได้ โดยโมดูลที่นำมาใช้จำลองระบบ UMTS นั้น เป็นโมดูลสำเร็จ เพราะโปรโตคอลของระบบ UMTS มีความซับซ้อนสูง หากสร้างเองต้องใช้เวลานาน และต้องทดสอบจนมั่นใจว่าโมดูลที่สร้างทำงานได้ถูกต้อง ครบถ้วน การเลือกใช้โมดูลสำเร็จจึงเป็นทางเลือกที่ดีกว่า เพราะโมดูลสำเร็จที่นำออกมาเผยแผ่สามารถการันตีความถูกต้องของการทำงานได้ดีระดับนึง โมดูลสำเร็จที่เลือกมาใช้จะกล่าวถึงในหัวข้อถัดไป

I. EURANE บน NS2

มีหลายกลุ่มวิจัยที่พยายามสร้างโมดูลจำลองการทำงานของโปรโตคอลของระบบ UMTS ได้แก่ โมดูลที่พัฒนาโดย Alfredo Todini และ FrancescoVacirca จากภาควิชา INFOCOM มหาวิทยาลัยแห่งโรม ประเทศอิตาลี [31], โมดูลที่พัฒนาโดย Pablo Martin และ Paula Ballester จาก Strathclyde University of Glasgow ประเทศสก็อตแลนด์ [32] และ EURANE หรือ Enhanced UMTS Radio Access Network ถูกพัฒนาภายใต้โครงการ SEACORN สำหรับ Ericsson Telecommunicate B.V. [33] โมดูลท้ายสุดถูกใช้กันอย่างหลากหลาย เนื่องจาก สามารถจำลองการทำงานของระบบ UMTS ได้ครอบคลุม พัฒนารองรับการทำงานทั้ง 3 โหนดด้วยกันคือ Radio Network Controller (RNC), Base station (BS) และ User Equipment (UE) สนับสนุนการทำงานของช่องสัญญาณขนส่งหรือ transport channels ต่างได ได้แก่ FACH, RACH, DCH และ HS-DSCH รองรับการส่งข้อมูลจากหลายโปรแกรมประยุกต์ ในขณะที่ 2 โมดูลแรกมีความซับซ้อนสูง และสามารถรองรับการส่งข้อมูลที่ส่งจาก UDP เท่านั้น

โมดูล EURANE เป็นตัวเพิ่มเติมสำหรับ ns-2 เวอร์ชั่น 2.28 การทำงานหลักพัฒนาเป็นไปตามรูปแบบของ RLC AM mode, UM mode, MAC-d/-c/sh ที่สนับสนุน RACH/FACH และ DCH และ MAC-hs ที่สนับสนุน HS-DSCH ตัวอย่างเช่น HSDPA เป็นต้น

การจำลอง UMTS ต้องการชนิดใหม่ของแพ็กเกตใน ns-2 และในฟิลด์ที่ 5 (pkt type) ของ trace output สามารถเป็นแพ็กเกตชนิดใดชนิดหนึ่งตามที่แสดงในตารางที่ ก.1

ตารางที่ ก.1 ชนิดของแพ็กเกต UMTS ที่ถูกเพิ่มเติม

ชนิดแพ็กเกต	คำอธิบาย
UM Data	UM Data PDU
AM Data	AM Data PDU
AM Ack	AM Positive acknowledgement
AM Bitmap Ack	AM Bitmap acknowledgement
AM Piggybacked Ack	AM Piggybacked Positive acknowledgement
AM Piggybacked Back	AM Piggybacked Bitmap acknowledgement

การเพิ่มเติมโมดูลลงใน ns-2 เป็นการเพิ่มเติมไลบรารี่ของ UMTS ที่เขียนด้วยภาษา C++ และ tcl ลงไป โดยไลบรารี่ภาษา tcl ของ EURANE ใช้ชื่อไฟล์ว่า *ns-umts.tcl* พร้อมกันนี้ยังมีการเพิ่มโคดลงไปในไฟล์เดิมของ ns-2 อีก 3 ไฟล์หลักคือ *ns-default.tcl*, *ns-packet.tcl* และ *packet.h* นอกจากนี้ยังได้มีการเปลี่ยนแปลงและสร้างไฟล์เพิ่มเติมเพื่อให้รองรับการทำงานของระบบ UMTS โดยรายละเอียดของไฟล์ และคำอธิบายไฟล์นั้น ๆ แสดงไว้ในตารางที่ ก.2

ตารางที่ ก.2 ไฟล์ที่แก้ไข/สร้างเพิ่มเติมสำหรับโมดูล EURANE

ไฟล์	คำอธิบาย
Makefile.in	ใส่ค่าสำหรับการ configure เพื่อให้รวมโคดของการทำงาน UMTS เข้าไปกับการทำงานเดิมด้วย
common/packet.h	เพิ่ม header ที่เป็นฟิลด์ของ UMTS
tcl/lib/ns-default.tcl	ใส่ค่ามาตรฐานของอ็อบเจกต์ UMTS
tcl/lib/ns-lib.tcl	เปลี่ยนแปลงเพื่อให้รวม ns-umts.tcl เข้าใช้งาน
tcl/lib/ns-packet.tcl	เพิ่มการกำหนด header ของ UMTS
tcl/lib/ns-umts.tcl	โคดสำหรับโหนด UMTS ใน ns
umts/am-hs.cc (.h)	การสร้าง RLC AM สำหรับโหมด enhanced UMTS
umts/am.cc (.h)	การสร้าง RLC AM สำหรับ UMTS
umts/classifier-sport.cc (.h)	ประเภทของพอร์ต UMTS
umts/demuxer.cc (.h)	Demuxer ของโหนด UMTS
umts/demuxerRtModule.cc(.h)	โมดูล Routing ของ UMTS

umts/dummy_drop_tail.h	การสร้าง Iub queue ที่ไม่มีการ drop
umts/hsdpalink.cc (.h)	การสร้าง MAC-hs (รวมถึงชั้น physical ด้วย)
umts/networkInterface.cc (.h)	UMTS Network Interface (NIF)
umts/nif-classifier.cc (.h)	UMTS NIF Classifier
umts/rlc.h	Superclass ของ RLC ทั่วไป
umts/tcs.cc (.h)	การสร้าง switching ช่องสัญญาณ UMTS
umts/um.cc (.h)	การสร้าง RLC UM สำหรับ UMTS
umts/um-hs.cc (.h)	การสร้าง RLC UM สำหรับโหมด enhanced UMTS
umts/umts-queue.cc (.h)	การสร้างคิวทั่วไปสำหรับ UMTS
umts/umts-timers.cc (.h)	การสร้างตัวจับเวลาทั่วไปสำหรับ UMTS
umts/umtslink.cc (.h)	การสร้างชั้น MAC และ physical ของ UMTS
umts/umtstrace.cc (.h)	การสร้างไฟล์ตามรอย
umts/virtual_umtsmac.cc (.h)	Class MAC ทั่วไป
umts/cqi.h	ตาราง lookup สำหรับ CQI กับขนาด MAC-hs PDU

การติดตั้งเข้ากับ ns-2 เวอร์ชั่น 2.28 ที่ถูกติดตั้งอยู่ก่อนแล้ว สามารถทำได้ 2 วิธี คือ วิธีแรกโดยการใช้ไฟล์ patch ของโมดูล EURANE ด้วยการเข้าไปยังไดเร็กเทอรี่ ns-2.28 แล้วเรียกใช้คำสั่ง *patch −p1 < [patch_file]* ที่ซึ่ง [patch_file] คือ ชื่อของไฟล์ patch นั้นเอง หรือวิธีที่ 2 ทำการแตกไฟล์ .tar ที่มีไฟล์ที่ข้างต้นที่สร้างและแก้ไขเรียบร้อยแล้วลงไปบันทึกแทนที่ไฟล์เดิม หลังจากนั้นเรียกใช้คำสั่ง *make* เพื่อทำการคอมไพล์โปรแกรมใหม่ทั้งหมด

และเนื่องจากในการทดสอบ จำเป็นต้องมีการเปลี่ยนแปลงค่าพารามิเตอร์ภายในของโปรโตคอล RLC ซึ่งเอกสารกำหนดมาตรฐานโปรโตคอล RLC ของ 3GPP นั้น [27] มิได้ระบุช่วงของค่าที่สามารถนำไปกำหนดได้ ดังนั้น ค่าของพารามิเตอร์ที่จะนำมาเปลี่ยนแปลงเพื่อทดสอบจะอ้างอิงตัวอย่างค่าจากเอกสารกรณีทดสอบของ 3GPP [36][37]

ภาคผนวก ข.
การทดลองเพิ่มเติมโดยใช้ไฟล์วิดีโอในการทดสอบ

จากคำแนะนำของคณะกรรมการสอบวิทยานิพนธ์ ที่ให้ทำการทดลองพิ่มเติมโดยใช้วิดีโอจริงในการทดสอบ เพื่อเปรียบเทียบกับผลการทดลองที่มาจากการทดสอบด้วยวิดีโอจำลองดังที่แสดงไปแล้วในบทที่ 5 ดังนั้นเนื้อหาส่วนนี้จะกล่าวถึงผลการทดลองที่ได้มาจากการทดสอบด้วยวิดีจริง ดังต่อไปนี้

I. รายละเอียดวิดีโอที่ใช้ทดลอง

ไฟล์วิดีโอที่ใช้ในการทดลองจัดอยู่ในกลุ่มวิดีโอที่ข้อมูลแต่ละเฟรมมีการเปลี่ยนแปลงต่ำ หรือมีการเคลื่อนไหวน้อย ลักษณะของไฟล์เป็นดังต่อไปนี้

ชื่อไฟล์วิดีโอ	Container_qcif.yuv
จำนวนความยาวเฟรม (เฟรม)	300
ขนาดเฟรม (pixels)	176 x 144
Key frame interval	24
ขนาดวิดีโอเฟรมสูงสุด (ไบท์)	9259
ขนาดวิดีโอเฟรมต่ำสุด (ไบท์)	200
ขนาดวิดีโอเฟรมเฉลี่ย (ไบท์)	1598
อัตราส่งเฟรม (fps)	15

รูปที่ ข.1 ตัวอย่างวิดีโอ Container_qcif

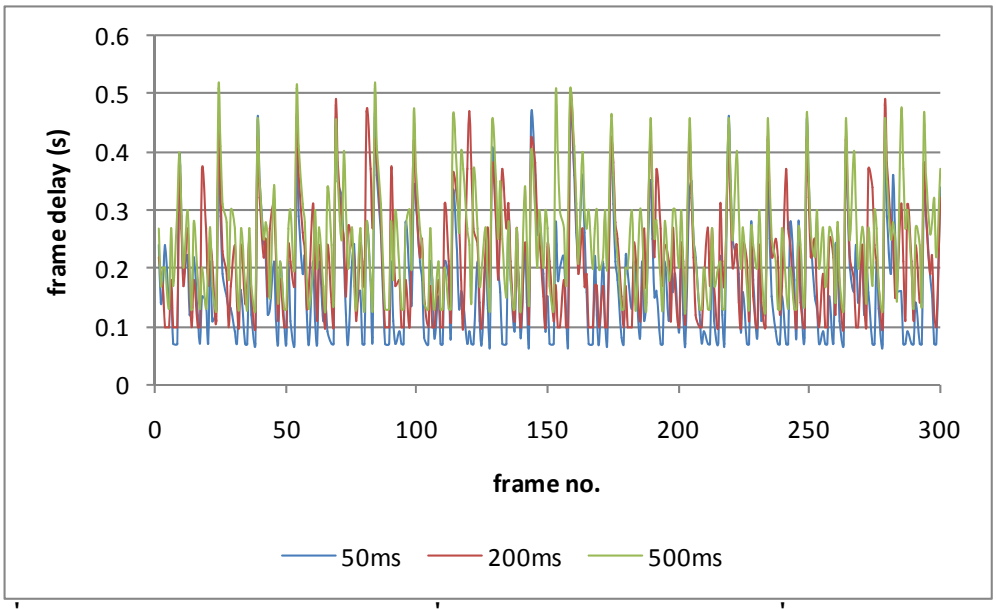
สำหรับรายละเอียดการติดตั้งแบบจำลอง โครงสร้างของระบบเครือข่าย และพารามิเตอร์ที่ใช้ในการทดลอง อ้างอิงตามหัวข้อในบทที่ 5

II. ผลการทดลองและวิเคราะห์ผลการทดลอง

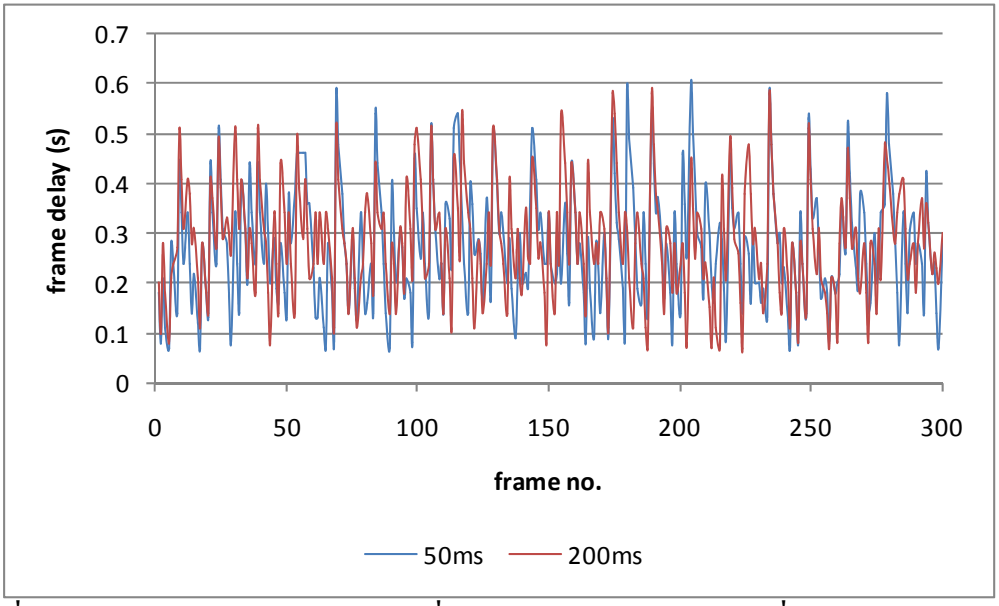
ก. เวลาหน่วงในการรับเฟรม (Received frame delay)

การทดลองจะเป็นไปในทางเดียวกันกับการทดลองที่กล่าวไว้ในบทที่ 5 โดยในส่วนแรกแสดงผลการทดลองให้เห็นถึงผลกระทบของ poll timer ที่มีต่อเวลาหน่วงในการรับเฟรมที่ UE เมื่อทดลองใช้ค่า poll timer ที่แตกต่างกันแต่มีอัตราความผิดพลาดในช่องสัญญาณ (link error rate) เท่ากัน ผลที่ได้เป็นดังรูปที่ ข.2 พบว่า poll timer ค่าต่ำกว่าจะให้ค่า frame delay ดีกว่าแม้ว่าอัตราความผิดพลาดในช่องสัญญาณจะเพิ่มขึ้นดังรูปที่ ข.3 ซึ่งให้ผลเช่นเดียวกับการใช้

วิดีโอจำลองในหัวข้อ 5.2.1



รูปที่ ข.2 เปรียบเทียบค่า frame delay เมื่อค่า poll timer แตกต่างกัน ที่ 10% link error rate

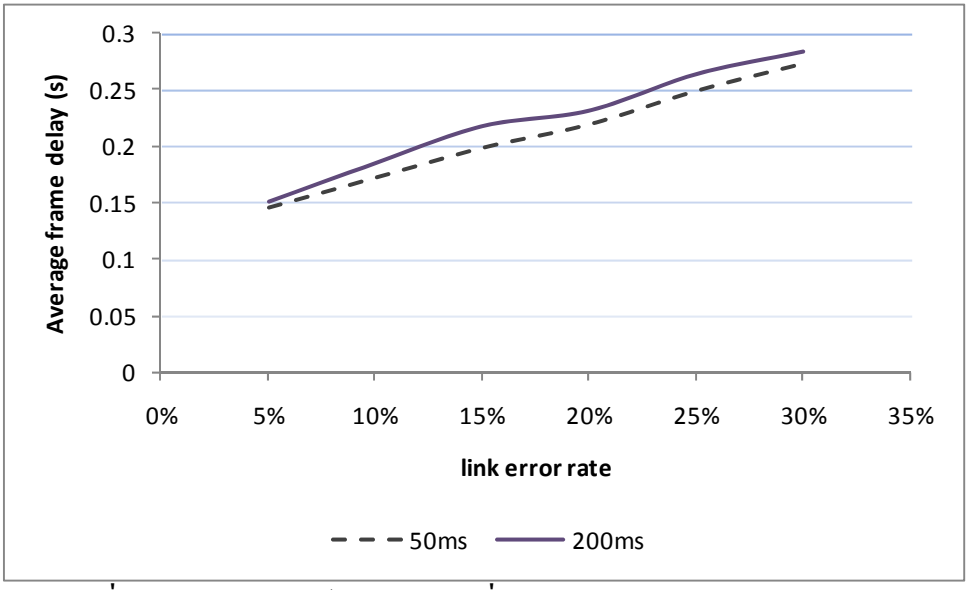


รูปที่ ข.3 เปรียบเทียบค่า frame delay เมื่อค่า poll timer แตกต่างกัน ที่ 30% link error rate

แต่สิ่งที่แตกต่างจากผลการทดลองในข้อ 5.2.1 คือ ยังไม่พบค่า threshold ของค่าเปอร์เซ็น like error rate ที่ poll timer ยังมีผลต่อการทำงานของระบบอยู่ ดังตารางที่ ข.1 และรูปที่ ข.4

ตารางที่ ข.1 เปรียบเทียบค่าเฉลี่ยเวลาหน่วงเฟรม

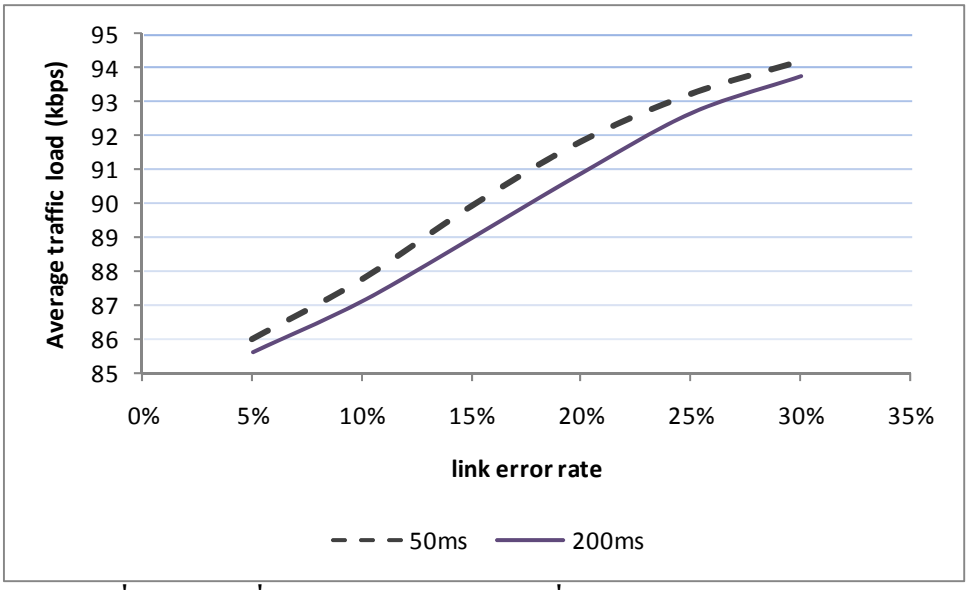
Poll Timer (ms)	Average Received Frame Delay (s)					
	Err5%	10%	15%	20%	25%	30%
50	0.145898	0.172465	0.198898	0.219732	0.249398	0.273598
200	0.150165	0.184665	0.217532	0.231398	0.263965	0.284065



รูปที่ ข.4 ความสัมพันธ์ระหว่างค่าเฉลี่ยเวลาหน่วงเฟรมกับ link error rate เมื่อให้ poll timer ต่างกัน

ข. การใช้งานช่องสัญญาณ

เมื่อทดลองเก็บข้อมูลการใช้งานของช่องสัญญาณที่อัตราความผิดพลาดค่าต่าง ๆ พบว่า ทั้ง poll ที่มีความถี่ 50 และ 200 มิลลิวินาที มีการใช้งานช่องสัญญาณเพิ่มสูงขึ้นตามอัตราความผิดพลาดที่เพิ่มขึ้น แต่ยังไม่พบว่าช่องสัญญาณมีการใช้งานเต็มแบนด์วิดท์แม้อัตราความผิดพลาดมีค่าสูงถึง 30% ดังรูปที่ ข.5 จึงเป็นเหตุให้ผลการทดลองในข้อก่อนหน้าไม่เจอค่า threshold นั่นเอง



รูปที่ ข.5 ค่าเฉลี่ยการใช้งานช่องสัญญาณที่อัตราความผิดพลาดค่าต่าง ๆ

ค. ประสิทธิภาพการทำงานของโปรโตคอล RLC

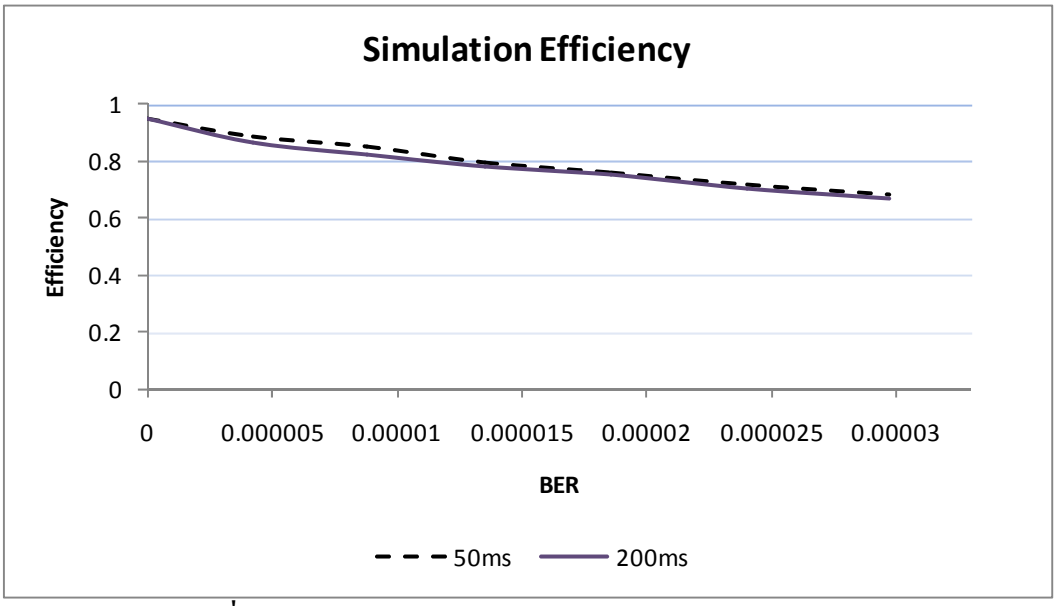
หากพิจารณาถึงประสิทธิภาพการทำงานของโปรโตคอล RLC จากผลการทดลองเมื่อเปลี่ยนข้อมูลที่ส่งผ่านเป็นวิดีโอจริง แสดงได้ดังตารางที่ ข.2 เมื่อ poll timer มีค่า 50 มิลลิวินาที และดังตารางที่ ข.3 เมื่อ poll timer มีค่า 200 มิลลิวินาที เมื่อนำมาเขียนกราฟแสดงความสัมพันธ์และทำการเปรียบเทียบ พบว่าให้ผลไปในทิศทางเดียวกันกับผลที่ได้จากการใช้วิดีโอจำลองเช่นกันดังรูปที่ ข.6

ตารางที่ ข.2 ประสิทธิภาพของโปรโตคอล RLC ที่ได้จากผลการทดลอง เมื่อใช้ 50ms poll timer

		no error	err 5%	err 10%	err 15%	err 20%	err 25%	err 30%
50 ms	header+data(kbyte/s)	10.25	9.57	9.17	8.59	8.17	7.71	7.36
	header (byte/s)	524.8	489.984	469.504	439.808	418.304	394.752	376.832
	data (kbyte/s)	9.7375	9.0915	8.7115	8.1605	7.7615	7.3245	6.992
	Eff	0.95	0.886976	0.849902	0.796146	0.75722	0.714585	0.682146

ตารางที่ ข.3 ประสิทธิภาพของโปรโตคอล RLC ที่ได้จากผลการทดลอง เมื่อใช้ 200ms poll timer

		no error	err 5%	err 10%	err 15%	err 20%	err 25%	err 30%
200 ms	header+data(kbyte/s)	10.25	9.32	8.85	8.41	8.08	7.59	7.21
	header (byte/s)	524.8	477.184	453.12	430.592	413.696	388.608	369.152
	data (kbyte/s)	9.7375	8.854	8.4075	7.9895	7.676	7.2105	6.8495
	Eff	0.95	0.863805	0.820244	0.779463	0.748878	0.703463	0.668244



รูปที่ ข.6 ประสิทธิภาพของโปรโตคอล RLC จากการทดลอง