เครื่องควบคุมระยะไกลผ่านทางอินเทอร์เน็ต

นายณัฐพล ชมภูนุช
นายพูนสุข เลิศไพบูลย์ปัญญา

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

Electronic Device Controlled By Internet

MR.Nattapon Chompunuch

MR.Poonsuk Lertpiboonpunya

A Special Project Submitted in Partial Fulfillment of the Requirement for the Degree of

Bachelor of Science

Department of Applied Physics

Faculty of Science

King Mongkut's Institute of Technology Ladkrabang

Academic Year 2002

โครงงานพิเศษเรื่อง เครื่องควบคุมระยะไกลผ่านทางอินเทอร์เน็ต

นักศึกษา นาย ณัฐพล ชมภูนุช
นาย พูนสุข เลิศไพบูลย์ปัญญา

ภาควิชา ฟิสิกส์ประยุกต์

สาขาวิชา ฟิสิกส์ประยุกต์

อาจารย์ที่ปรึกษา รศ. วิชิต ศิริโชติ

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้ปัญหาพิเศษ/โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

คณะกรรมการตรวจสอบ	ลายมือชื่อ
ประธานกรรมการ ผศ.วิชาญ เตชิตธีระ	.....
กรรมการ รศ.วิชิต ศิริโชติ	.....
กรรมการ อ.สุรศักดิ์ พิพัฒน์ศาสตร์	.....
กรรมการ อ.ธนภรณ์ ลีลาวัฒนานนท์	.....

.....

(ผศ.วิชาญ เตชิตธีระ)

หัวหน้าภาควิชา

โครงงานพิเศษ เรื่อง เครื่องควบคุมระยะไกลผ่านทางอินเทอร์เน็ต

นักศึกษา นายณัฐพล ชมภูนุช
นายพูนสุข เลิศไพบูลย์ปัญญา

ภาควิชา ฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์
สาขาวิชา ฟิสิกส์ประยุกต์
ปีการศึกษา 2545
อาจารย์ที่ปรึกษา รศ.วิชิต ศิริโชติ

บทคัดย่อ

เครื่องควบคุมติดต่อทางอินเทอร์เน็ต ได้รับการออกแบบและสร้างขึ้น ตัวอุปกรณ์สร้างโดยไมโครคอนโทรลเลอร์ ตระกูล MCS-51 เชื่อมต่อกับแลนการ์ดชนิด 10BaseT ทางด้านเอาท์พุตของตัวอุปกรณ์สามารถต่อกับอุปกรณ์ไฟ AC และสามารถควบคุมการเปิดและปิดวงจรได้ ตัวโปรแกรมควบคุมเขียนด้วยภาษาแอสเซมบลี้ โปรโตคอลที่ใช้ได้คือ ARP ICMP และ UDP เราได้พัฒนาโปรแกรม GUI ที่สามารถรันบนเครื่องพีซีสำหรับส่งแพ็คเกต UDP ได้ทำการทดสอบเครื่องภายใต้ระบบเครือข่ายภายในมหาวิทยาลัย โดยใช้โปรแกรม ping โปรแกรมประยุกต์ได้ทดสอบระหว่าง Client กับเครื่องควบคุมระหว่างอาคาร

Special Project Title	Electronic Device Controlled By Internet
Name	MR.Nattapon Chompunuch
	MR.Poonsuk Lertpiboonpunya
Department	Applied Physics
Program	Applied Physics
Acadamic Year	2002
Special Project Advisor	Assoc.Prof.Wichit Sirichote

Abstract

A simple Internet enabled device has been designed and built. The device is based on using the microcontroller MCS51 interfaced with a 10BaseT LAN card. The output of the device is capable of providing the AC load switching on and off. The firmware was written using assembly coding. The protocols available are ARP, ICMP and UDP. We have developed the GUI software running on PC for sending the UDP packet. The device was tested under the campus network. Network reachable testing was done with PING and the application program was tested from client between the faculty's building.

กิตติกรรมประกาศ

โครงงานพิเศษนี้สำเร็จลุล่วงมาด้วยดีต้องขอขอบคุณ บิดา – มารดา และครอบครัวของข้าพเจ้าที่คอยถามข่าวคราวของโครงงานนี้และคอยให้คำปรึกษาให้ความช่วยเหลือตลอดมา ท่านอาจารย์วิชิต ศิริโชติ ที่ช่วยชี้แนะ ให้คำปรึกษาเกี่ยวกับโครงงานนี้ตลอดมา เพื่อน ๆ ที่ช่วยให้คำปรึกษาและคอยช่วยเหลือตลอดการทำโครงงานนี้ ภาควิชาฟิสิกส์ประยุกต์ที่ให้ที่ศึกษา ให้อุปกรณ์ในการศึกษาและทำงานโครงงานในครั้งนี้ทำให้โครงงานทำงานลุล่วงไปได้ ทั้งยังให้ความรู้ทางวิชาชีพต่างๆ และอบรมให้เป็นคนดี ทำให้เป็นคนดีมีประโยชน์ต่อสังคม และมีความรู้มีความสามารถออกไปทำงานหาอาชีพ ช่วยเหลือสังคมและเป็นคนดีต่อสังคมต่อไป

นายณัฐพล ชมภูนุช

นายพูนสุข เลิศไพบูลย์ปัญญา

สารบัญ

	หน้า
บทคัดย่อภาษาไทย	ก
บทคัดย่อภาษาอังกฤษ	ข
กิตติกรรมประกาศ	ค
สารบัญ	ง
สารบัญตาราง	จ
สารบัญรูป	ฉ
บทที่ 1 บทนำ	1
1.1 ความเป็นมาของโครงงานพิเศษ/ปัญหาพิเศษ	1
1.2 วัตถุประสงค์	1
1.3 ขอบเขตของการวิจัย	2
1.4 ผลที่คาดว่าจะได้รับ	2
บทที่ 2 ทฤษฎีและหลักการ	3
2.1 TCP/IP และ Internet	3
2.2 สถาปัตยกรรม TCP/IP	3
2.3 ARP (Address Resolution Protocol)	4
2.4 IP Address	6
2.5 ICMP (Internet Control Message Protocol)	12
2.6 UDP (User Datagram Protocol)	15
บทที่ 3 วิธีดำเนินการวิจัย	20
3.1 ส่วนควบคุมและสั่งการ	20
3.2 ส่วน server	21
3.3 ส่วน Relay	23
3.4 แสดง Flowchart การทำงานของตัว Server	23
บทที่ 4 ผลการทดลองและอภิปรายผล	25
4.1 ผลการทดลอง	25
4.1.1 การ Broadcast ของระบบ Network	25

สารบัญ (ต่อ)

	หน้า
4.1.2 การทดสอบระบบโดยใช้ Program Ping	26
4.1.3 วิธีการเปิด – ปิด หลอดไฟ	28
4.1.4 Checksum IP	30
4.1.5 Checksum ICMP	31
4.1.6 Checksum UDP	33
4.1.7 การพิจารณาข้อมูล UDP ที่ฝั่งผู้รับ	35
4.1.8 ระยะทางในการควบคุมไฟ	35
4.2 อภิปรายผล	35
บทที่ 5 สรุปผลการทดลอง	37
5.1 บทสรุป	37
5.2 การประยุกต์ใช้งาน	37
5.3 ข้อเสนอแนะ	37
เอกสารอ้างอิง	38
ภาคผนวก	39
Sourcecode Ethertest.asm	39
Sourcecode Delphi	57

สารบัญตาราง

หน้า

ตารางที่ 2.1 ประเภทความผิดพลาดของไอซีเอ็มพี 14

สารบัญรูป

	หน้า
รูปที่ 1.1 แสดงรูปแบบ Packet	2
รูปที่ 2.1 แสดง ARP Datagram	5
รูปที่ 2.2 แสดง IP Datagram	7
รูปที่ 2.3 แสดงการแบ่งคลาสเครือข่าย	9
รูปที่ 2.4 แสดงการแบ่งคลาส D และ E	9
รูปที่ 2.5 แสดงการแบ่ง subnet	12
รูปที่ 2.6 ข่าวสารของไอซีเอ็มพีบรรจุอยู่ในไอพี	12
รูปที่ 2.7 ฟอร์แมตของไอซีเอ็มพี	13
รูปที่ 2.8 เดทาแกรมไอซีเอ็มพีชนิด 0 และ 8	15
รูปที่ 2.9 ระดับชั้นของ TCP และ UDP	16
รูปที่ 2.10 การเอ็นแคปซูเลต UDP	16
รูปที่ 2.11 UDP Datagram	17
รูปที่ 2.12 UDP และ เฮดเดอร์เทียม	18
รูปที่ 3.1 แสดง Block Diagram การเชื่อมต่อของส่วนต่างๆ	20
รูปที่ 3.2 แสดงหน้าตาของโปรแกรมควบคุม Relay โดยใช้โปรแกรม Delphi เขียน	21
รูปที่ 3.3 แสดงการติดต่อระหว่างการ์ด LAN กับ บอร์ด EVB – 52	22
รูปที่ 3.4 แสดงภาพ Schematic ของ Slot ISA ต่อเชื่อมกับบอร์ด EVB – 52	22
รูปที่ 3.5 แสดงการเชื่อมต่อวงจร Relay	23
รูปที่ 3.6 แสดงภาพ Schematic ของวงจร Relay ที่เชื่อมต่อกับ บอร์ด EVB – 52	23
รูปที่ 3.7 แสดง Flowchart การทำงานของตัว Server	24
รูปที่ 4.1 แสดงการจับ Packet (ARP Request)	25
รูปที่ 4.2 แสดงการจับ Packet (ARP Reply)	26
รูปที่ 4.3 แสดงการใช้ Program Ping จาก Application ที่สร้างจาก Delphi	26

สารบัญรูป (ต่อ)

	หน้า
รูปที่ 4.4 แสดงการจับ Packet ICMP (Echo)	27
รูปที่ 4.5 แสดงการจับ Packet ICMP (Echo reply)	27
รูปที่ 4.6 แสดงการจับ Packet โดยการใช้โปรแกรม Sniffer ในกรณีการเปิดไฟ	28
รูปที่ 4.7 แสดงการจับ Packet โดยการใช้โปรแกรม Sniffer ในกรณีการปิดไฟ	29
รูปที่ 4.8 แสดง Sourcecode ในการตรวจสอบการเปิด – ปิด หลอดไฟ	29
รูปที่ 4.9 แสดงตำแหน่งของ Buffer ของ Chip Realtek	29
รูปที่ 4.10 แสดงการจับ Packet IP	30
รูปที่ 4.11 แสดงการจับ Packet ICMP	32
รูปที่ 4.12 แสดงการจับ Packet UDP	33
รูปที่ 4.13 แสดงภาพของตัว Server โดยใช้ Microcontroller เชื่อมกับ LAN card ชนิด 10BaseT	35
รูปที่ 4.14 แสดงภาพของตัว Server ในการเปิด – ปิด ไฟ	36

บทที่ 1
บทนำ

1.1 ความเป็นมาของโครงงานพิเศษ

ในปัจจุบันเทคโนโลยีทางด้าน Computer มีความก้าวหน้าเป็นอย่างมาก เราสามารถที่จะรับหรือ ส่งข้อมูลข่าวสารต่างๆ จากทั่วโลกผ่านทาง Internet โดยใช้ Computer เป็นสื่อในการควบคุมรับหรือส่งข้อมูลต่างๆ เราจึงสนใจจะนำ Computer และระบบ Internet มาประยุกต์ใช้ในการควบคุมอุปกรณ์ไฟฟ้าต่างๆ ตามสถานที่อื่นๆทั่วโลกผ่านทางระบบ Internet โดยที่เราไม่จำเป็นจะต้องอยู่ในที่บริเวณนั้นๆก็ได้ เราสามารถที่จะอยู่ห่างไกลจากสถานที่นั้นๆ และทำการสร้างอุปกรณ์ขนาดเล็กๆ ในการติดต่อผ่านระบบ Internet โดยประกอบเข้ากับการ์ด LAN ซึ่งจะเป็นตัวติดต่อระหว่างตัวอุปกรณ์ไฟฟ้าและตัวควบคุมโดยจะนำไปติดตั้งตามสถานที่ต่างๆ โดยที่สถานที่ที่เราต้องการจะติดตั้งนั้น จะนำตัวอุปกรณ์ที่เราสร้างประกอบเข้ากับตัวอุปกรณ์ไฟฟ้าในแต่ละแห่งซึ่งจะทำการเชื่อมต่อไปยังระบบ Network ผ่านทางสาย LAN ดังนั้นเราจะสามารถควบคุมอุปกรณ์ไฟฟ้าตามสถานที่ต่างๆ ได้ง่ายขึ้น ซึ่งต่อไปในภายภาคหน้าก็จะมีความสะดวกขึ้นในการควบคุมอุปกรณ์ไฟฟ้าต่างๆ นี่จึงเป็นที่มาของโครงงานพิเศษซึ่งจะประยุกต์ในการนำการ์ด LAN มาต่อเชื่อมกับบอร์ด EVB – 52 สร้างเป็นตัว Server ขนาดย่อมและเชื่อมต่อกับอุปกรณ์ไฟฟ้าต่างๆ จากนั้นก็ต่อสาย LAN เข้าไปยังระบบ Network และเชื่อมต่อเข้ากับเครื่อง Computer อื่นๆ ผ่านทางสาย LAN เช่นเดียวกัน ซึ่งจะทำให้สามารถติดต่อกันได้ทั่วโลกซึ่งจะง่ายในการควบคุมอุปกรณ์ตามสถานที่ต่างๆ มากขึ้น

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาการทำงานของระบบเครือข่าย Network

1.2.2 เพื่อศึกษาการสร้างตัว server โดยใช้การ์ด LAN ทำการ Interface กับบอร์ด EVB – 52

1.2.3 เพื่อศึกษาการเขียนโปรแกรมภาษา Assembly

1.2.4 เพื่อศึกษาการเขียนโปรแกรมภาษา Delphi

1.2.5 เพื่อสะดวกในการควบคุมอุปกรณ์ไฟฟ้ามากขึ้น

1.2.6 เพื่อเรียนฝึกทักษะของการคิดอย่างเป็นระบบ

1.2.7 เพื่อรู้จักการแก้ปัญหาและทำงานอย่างเป็นระบบ

1.3 ขอบเขตของการวิจัย

ในส่วนของการติดต่อระหว่าง Computer กับตัว server ที่เราสร้าง เราจะมีการส่ง packet (protocal) ไปยังตัว server ซึ่งรูปแบบของ packet เป็นดังนี้

DLC	ARP/IP	TCP,UDP	HTTP,SNMP

รูปที่ 1.1 แสดงรูปแบบ Packet

ขอบเขตของการศึกษาของเราศึกษาถึงระดับ UDP Protocal ในชั้น Transport ซึ่งจะใช้ในการติดต่อกับตัว server เพื่อไปควบคุมอุปกรณ์ไฟฟ้าปลายทาง โดยในโครงงานพิเศษนี้จะสร้างเป็นวงจร Relay ที่ใช้ในการควบคุมอุปกรณ์ไฟฟ้า คือ หลอดไฟ

1.4 ผลที่คาดว่าจะได้รับ

1.4.1 ได้เรียนรู้การทำงานของระบบ Network และสามารถประยุกต์ใช้งานได้

1.4.2 ได้รับความสะดวกสบายในการควบคุมอุปกรณ์ไฟฟ้ามากขึ้น

1.4.3 มีความเข้าใจในภาษา Assembly มากขึ้น

1.4.4 ได้ฝึกความเชี่ยวชาญทางด้าน Hardware ในการประกอบอุปกรณ์

1.4.5 รู้จักการทำงานอย่างเป็นระบบทำให้สามารถทำงานร่วมกับผู้อื่นได้

บทที่ 2
ทฤษฎีและหลักการ

2.1 TCP/IP และ Internet

TCP/IP เป็นโปรโตคอลที่ได้รับการออกแบบให้เป็นอิสระจากชนิดของคอมพิวเตอร์และระบบปฏิบัติการ

2.1.1 การเชื่อมโยงเครือข่าย

จุดประสงค์ของการเชื่อมโยงคอมพิวเตอร์เข้าเป็นระบบเครือข่าย คือ ต้องการให้คอมพิวเตอร์สามารถสื่อสารและแลกเปลี่ยนข้อมูลระหว่างกันได้

2.2.2 ความหมายของโปรโตคอล

การเชื่อมเครือข่ายต่างฮาร์ดแวร์จำเป็นต้องกำหนดข้อตกลงร่วม หรือ โปรโตคอล (Protocol) เพื่อให้คอมพิวเตอร์สื่อสารกันตามข้อกำหนด

2.2.3 โปรโตคอล TCP/IP

TCP/IP จะเชื่อมกลุ่มเครือข่ายย่อยเข้าด้วยกันเป็นเครือข่ายขนาดใหญ่ หรือ Internet โดย IP จะทำหน้าที่กำหนดแอดเดรส จัดแบ่งขนาดข้อมูลให้พอเหมาะ และเลือกเส้นทางส่งข้อมูล ส่วน TCP มีหน้าที่รับประกันความถูกต้องในการลำเลียงข้อมูล

2.2.4 TCP/IP และ Internet

คอมพิวเตอร์ใน Internet ทุกเครื่องจะใช้โปรโตคอล TCP/IP เพื่อสื่อสารกัน

2.2 สถาปัตยกรรม TCP/IP

2.2.1 IP Address

อินเทอร์เน็ตจะแยกแยะเครื่องโดยใช้ IP Address ประจำฮาร์ดแวร์ (การ์ดแลน) ที่ทำการอินเทอร์เฟสเชื่อมเข้ากับเครือข่าย

2.2.2 โปรโตคอลในชั้น TCP/IP

- IP (Internet Protocol) : IP เป็นโปรโตคอลแกนของ TCP/IP โดย IP จะทำหน้าที่กำหนดรูปแบบของแอดเดรสประจำเครื่อง เพื่อใช้ในการลำเลียงข้อมูลจากเครื่องต้นทางไปยังเครื่องปลายทาง นอกจากนี้ยังทำหน้าที่เลือกเส้นทางส่งข้อมูล ตลอดจนแบ่งขนาดข้อมูลให้เหมาะกับฮาร์ดแวร์ระดับล่าง

- ICMP (Internet Control Message Protocol) : ICMP เป็นโปรโตคอลซึ่ง

ใช้รายงานสถานะความผิดพลาดที่เกิดขึ้น เช่น ในกรณีที่เราเตอร์ไม่สามารถนำข้อมูลส่งไปถึงปลายทางได้ เราเตอร์จะใช้ ICMP แจ้งสาเหตุของปัญหากลับไปยังฝ่ายส่ง

- UDP (User Datagram Protocol) : UDP เป็นโปรโตคอลแบบ "connectionless" คือ ไม่ต้องสถาปนาการเชื่อมต่อระหว่างสถานีรับและสถานีส่ง UDP เป็นโปรโตคอลระดับชั้นเดียวกับ TCP แต่ว่าไม่มีกลไกรับประกันความเชื่อถือในการขนถ่ายข้อมูล หากข้อมูลสูญหาย ซ้ำซ้อน หรือลำดับไม่ถูกต้อง UDP จะปล่อยให้โปรโตคอลที่เรียกใช้งานดำเนินการกับปัญหาเหล่านี้เอง ลักษณะเด่นของ UDP คือ ใช้การประมวลผลต่ำกว่า TCP เนื่องจากเฮดเดอร์มีขนาดเล็กและไม่ต้องสถาปนาการเชื่อมต่อ แอพลิเคชันที่ทำงานโดยรับส่งคำถามและคำตอบเป็นรายการ (transaction) เช่น DNS

2.3 ARP (Address Resolution Protocol)

หน้าที่ของ ARP ในเครือข่าย TCP/IP คือ ช่วยแปลง IP Address ไปสู่ฮาร์ดแวร์แอดเดรส เพื่อให้สถานีสามารถนำฮาร์ดแวร์แอดเดรสไปสร้างเฟรมระดับ datalink ได้

2.3.1 โปรโตคอล ARP

การหาฮาร์ดแวร์แอดเดรสอาศัยการค้นจากตารางซึ่งเก็บคู่ของ IP Address และฮาร์ดแวร์แอดเดรส การสร้างตารางและการค้นหามีรูปแบบที่กำหนดด้วยโปรโตคอล ARP

2.3.2 หลักการทำงานของ ARP

ARP เป็นโปรโตคอลที่เริ่มนำมาใช้งานกับอีเทอร์เน็ตในคณะเดียวกันก็ผ่านการออกแบบให้สามารถใช้งานร่วมกับเครือข่ายอื่นๆ ที่สนับสนุนการบรอดคาสต์ได้ไม่ว่าจะเป็นโทเค็นริงหรือเอฟดีดีไอ แต่ในที่นี้จะอธิบายการทำงานของ ARP โดยอ้างอิงกับอีเทอร์เน็ตเป็นหลัก

สถานีต้นทางที่ต้องการส่งแพ็กเก็ตไปยังสถานีปลายทางจะเปิดตารางหาค่าฮาร์ดแวร์แอดเดรสที่ตรงกับ IP Address จาก ARP cache หากไม่พบค่าในแคช สถานีจะสร้างเฟรม ARP และ บรอดคาสต์ เดทาแกรมร้องขอ (ARP request) โดยใส่ IP address ของสถานีที่ต้องการถามหาฮาร์ดแวร์แอดเดรส

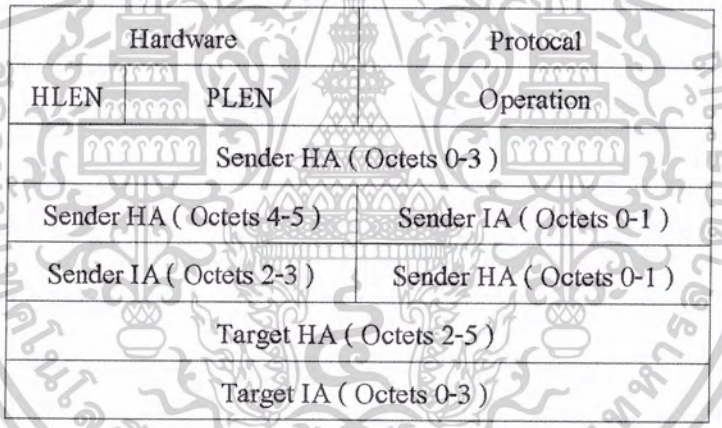
ทุกสถานีในเครือข่ายเมื่อได้รับเฟรมบรอดคาสต์จะตรวจสอบ IP Address ประจำตัวกับ IP Address ที่ร้องขอ หากว่าตรงกันสถานีนั้นก็จะส่งเดทาแกรมตอบรับ (ARP reply) โดยส่งฮาร์ดแวร์แอดเดรสกลับไปยังสถานีร้องขอ สถานีต้นทางที่ได้รับคำตอบก็จะนำฮาร์ดแวร์แอดเดรสไปใช้งานและเก็บเข้าตาราง ในขณะเดียวกันสถานีที่ตอบรับก็จะนำค่า IP และฮาร์ดแวร์แอดเดรสของสถานีที่ร้องขอเก็บเข้าตารางเช่นกันเพื่อใช้งานในโอกาสต่อไป

คู่แอดเดรสในตาราง ARP จะคงค่าอยู่เพียงระยะหนึ่งและจะถูกกำจัดออกเมื่อถึง

เวลาที่กำหนด เวลาที่กำหนดอายุค่าในตาราง ARP (ARP cache timeout) อาจแตกต่างไปตามแต่ละระบบปฏิบัติการ โดยปกติแล้วหากไม่มีการใช้ค่าใดค่าหนึ่งประมาณ 15 นาที ค่านั้นจะถูกลบทิ้งออกไป วิธีนี้ช่วยให้ค่าในตาราง ARP เปลี่ยนแบบไดนามิกตามสภาพแวดล้อม เช่น เมื่อเปลี่ยนการ์ดเครือข่ายใหม่ ฮาร์ดแวร์แอดเดรสจะเปลี่ยนแปลงตามไปด้วย ในคณะที่ IP Address ยังมีค่าเดิมได้ การเปลี่ยนแปลงค่าในตารางแบบไดนามิกของ ARP จึงช่วยลดภาระงานผู้ดูแลระบบในการดูแลตาราง

2.3.3 เออาร์พีเดทาแกรม

รูปที่ 2.1 แสดงถึงฟอร์แมตของเออาร์พีเดทาแกรม เนื่องจากเออาร์พีทำงานในระดับเดทาลิงค์ เดทาแกรมของเออาร์พีก็จะบรรจุอยู่ในเฟรมเดทาลิงค์อย่างเช่นอีเทอร์เน็ตโดยมีค่าประจำโปรโตคอลเท่ากับ 0x0806 แต่ละฟิลด์มีความหมายดังนี้



รูปที่ 2.1 แสดง ARP Datagram

- Hardware 16 บิต : กำหนดชนิดของฮาร์แวร์เครือข่ายที่เออาร์พี ทำงานอยู่ ค่าที่ใช้งานมีดังตัวอย่างต่อไปนี้

1	อีเทอร์เน็ต
4	โทเค็นริง
5	เคออส (Chaos)
6	เครือข่าย IEEE 802
7	อาร์คเน็ต
12	โลคัลทอล์ค

- protocol 16 บิต : ชนิดของโปรโตคอลที่ร้องของใช้เออาร์พี
- HLEN 8 บิต : ขนาดของฮาร์ดแวร์แอดเดรสเป็นจำนวนไบต์ ค่าปกติที่ใช้งานคือ 6 ซึ่งเท่ากับขนาด 6 ไบต์ของอีเทอร์เน็ตฮาร์แวร์แอดเดรส
- PLEN 8 บิต : ขนาดของแอดเดรสระดับเน็ตเวอร์คเป็นจำนวนไบต์ ค่าปกติที่ใช้งานคือ 4 ซึ่งเท่ากับขนาด 4 ไบต์ของไอพีแอดเดรส
- Operation 16 บิต : กำหนดรูปแบบการใช้เดทาแกรม ค่าในฟิลด์นี้ใช้กำหนดการทำงานของทั้งเออาร์พีและอาร์เออาร์พี ซึ่งมีสี่ค่าคือ

  ARP request (ค่าเท่ากับ 1)

  ARP reply (ค่าเท่ากับ 2)

  RARP request (ค่าเท่ากับ 3)

  RARP reply (ค่าเท่ากับ 4)

- Address : ฟิลด์แอดเดรสเรียงลำดับจากฮาร์แวร์และเน็ตเวอร์คแอดเดรสของสถานีที่ร้องขอ ตามด้วยฮาร์ดแวร์และเน็ตเวอร์คแอดเดรสของสถานีที่ตอบรับ

2.3.4 พร็อกซี่ ARP

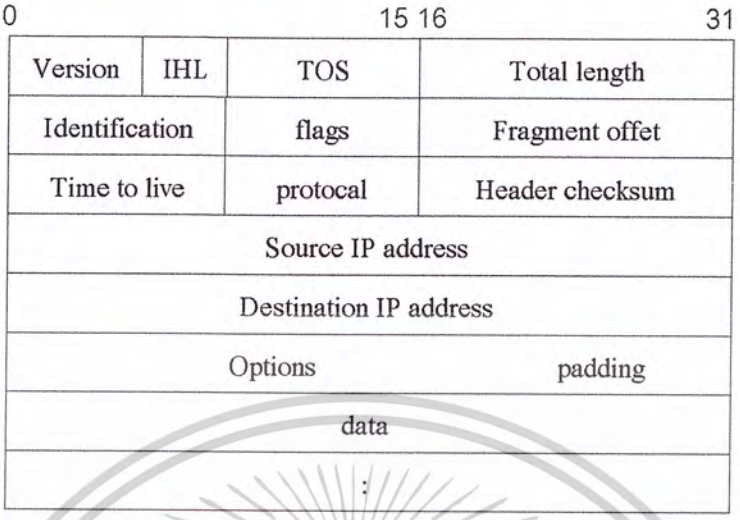
พร็อกซี่ ARP เป็นเทคนิคที่ใช้สำหรับการสื่อสารระหว่างสถานีต่างเครือข่ายที่ไม่สนับสนุนการใช้ซับเน็ต เช่น เครือข่ายถูกแบ่งออกเป็นสองเครือข่ายย่อยด้วยเราเตอร์แต่ซอฟต์แวร์ระบบที่ใช้ในสถานีของแต่ละเครือข่ายไม่สนับสนุนการใช้ซับเน็ต ซึ่งจะส่งผลให้สถานีไม่สามารถสื่อสารระหว่างกันได้

วิธีแก้ปัญหาดังกล่าวทำได้โดยปรับตั้งเราเตอร์ให้ทำหน้าที่เป็น พร็อกซี่ ARP (Proxy ARP) เมื่อสถานีในเครือข่ายหนึ่งบรอดคาสต์ ARP เพื่อถามหาฮาร์ดแวร์แอดเดรสของสถานีในอีกเครือข่าย เราเตอร์จะทำหน้าที่เป็นตัวกลางช่วยตอบเฟรม ARP นั้นแทนโดยส่งฮาร์ดแวร์แอดเดรสของเราเตอร์ไปให้ และเราเตอร์ก็จะรับหน้าที่เป็นตัวส่งเฟรมข้ามเครือข่ายให้

2.4 IP Address

2.4.1 IP Address

รุ่นที่ใช้กันอยู่นั้นมีขนาด 32 บิต โดยแอดเดรสขนาด 32 บิต นั้นจะประกอบด้วยหมายเลขสองส่วน คือ เลขเครือข่าย (network number) และ เลขโฮสต์ (host number) โดยเลขเครือข่ายใช้สำหรับจัดคลาสเครือข่าย ส่วนเลขโฮสต์ใช้ระบุหมายเลขโฮสต์ (หรืออีกนัยหนึ่งคืออินเตอร์เฟสของโฮสต์)ซึ่งไอพีจะมีฟอร์แมตเดทาแกรมดังรูป



รูปที่ 2.2 แสดง IP Datagram

เฟรมฟอร์แมตในทีซีพี/ไอพี นิยมเขียนแสดงเป็นแถวๆ ละ 32 บิต เรียงจากซ้ายไปขวาและจากบนลงล่าง

- version ขนาด 4 บิต : แสดงบางรุ่นของโปรโตคอล รุ่นที่ใช้งานขณะปัจจุบันมีค่า 4
- Internet Header Length (IHL) ขนาด 4 บิต : บอกความยาวเฉพาะเฮดเดอร์ของเดทาแกรมโดยนับจาก version จนถึงไบต์สุดท้ายก้อนที่จะส่งข้อมูล หน่วยนับความยาวจะบอกเป็นจำนวนเท่าของ 4 ไบต์ หรือ (32 บิตเวิร์ด) หาก IHL มีค่าเท่ากับ 5 จะหมายถึงส่วนหัวมีขนาด 20 ไบต์ซึ่งเป็นค่าที่บอกว่าไม่มี options และ padding อยู่ในเดทาแกรม
- Type of Service (TOS) ขนาด 8 บิต : ฟิลด์นี้ใช้กำหนดรูปแบบการทำงานตามลักษณะโปรโตคอลแอพพลิเคชัน
- Total length มีขนาด 16 บิต : บอกถึงความยาวทั้งหมดของเดทาแกรม (เฮดเดอร์และข้อมูล) โดยมีหน่วยนับเป็นไบต์ เนื่องจากฟิลด์นี้มีขนาด 16 บิต ไอพีเดทาแกรมจึงมีขนาดใหญ่สุดเท่ากับ 2^{16}-1 หรือ 65,535 ไบต์
- Identification ขนาด 16 บิต
- Flags ขนาด 3 บิต
- Fragment offset ขนาด 13 บิต

ใช้กับการแบ่งข้อมูลย่อยเพื่อให้เหมาะกับเฟรมระดับเดทาลิงค์

- Time to Live (TTL) ขนาด 8 บิต : ฟิลด์นี้ใช้กำหนดจำนวนเราเตอร์ที่เดทาแกรมจะเดินทางผ่านได้หรืออีกนัยหนึ่งคือกำหนดอายุของเดทาแกรมซึ่งมีค่าได้สูงสุดตามขนาดฟิลด์ คือ 2^8-1 หรือ 255 สถานีที่ส่งเดทาแกรมจะตั้งค่า TTL ไว้ที่ค่าใดค่าหนึ่ง เราเตอร์ที่รับเดทาแกรมจะปรับลดค่านี้ลงหนึ่งหน่วยหากลดลงเป็น 0 เราเตอร์จะทิ้งเดทาแกรมนั้นและรายงานกลับไปด้วยไอซีเอ็มพี วิธีนี้ช่วยป้องกันปัญหาเดทาแกรมวนรอบ (routing loop) สถานีต้นทางต้องเลือกใช้ค่านี้ให้เหมาะสม เนื่องจากหากมีค่าน้อยไปจะทำให้เดทาแกรมเดินทางไปไม่ถึงปลายทาง หรือหากตั้งไว้มากเกินไปก็จะสร้างภาระให้ระบบเมื่อมีความผิดปกติด้านการเลือกเส้นทาง ค่าโดยปกติที่ใช้คือ 64
- Protocal ขนาด 8 บิต : ฟิลด์บอกชนิดของโปรโตคอลระดับบนที่เอ็นแคปซูเลตในเดทาแกรม เพื่อให้สถานีปลายทางและสามารถส่งข้อมูลไปยังโปรโตคอลระดับบนได้ถูกต้องค่าที่ใช้ประจำโปรโตคอลมีดังเช่น

  1 ICMP
  6 TCP
  8 EGP
  17 UDP
  89 OSPF

- Header checksum ขนาด 16 บิต : ใช้ตรวจสอบความผิดพลาดเฉพาะเฮดเดอร์โดยไม่รวมส่วนข้อมูล การคำนวณผลรวมตรวจสอบจะเริ่มต้นด้วยการให้ฟิลด์ checksum มีค่าเป็น 0 จากนั้นจึงบวกเฮดเดอร์ครั้งละ 16 บิตแบบเติมเต็มหนึ่ง (1's complement) เมื่อได้ผลลัพธ์แล้ว จะนำใส่ในฟิลด์ checksum ไอพีปลายทางเมื่อได้รับเดทาแกรมแล้วก็เพียงแต่บวกเฮดเดอร์ทั้งหมดครั้งละ 16 บิตแบบเติมเต็มหนึ่ง หากค่าที่ได้ไม่เท่ากับ 0 แสดงว่ามีข้อผิดพลาดในเฮดเดอร์
- Source IP address ขนาด 32 บิต : กำหนดไอพีแอดเดรสต้นทาง
- Destination IP address ขนาด 32 บิต : กำหนดไอพีแอดเดรสปลายทาง
- Option ขนาดไม่คงที่ : ใช้สำหรับกำหนดข่าวสารเพิ่มเติมสำหรับเดทาแกรม ค่าที่ใช้ในปัจจุบันจะเกี่ยวข้องกับการรักษาความปลอดภัย และการบันทึกผลลัพธ์จากการทำงานของคำสั่ง ping
- Padding ขนาด 0 ถึง 3 ไบต์ :ใช้สำหรับผนวกเพิ่มเพื่อให้จำนวนไบต์ของ

option รวมกับ padding เป็นจำนวนเท่าของ 32 บิต ค่าในฟิลด์ padding จึงไม่มีความสำคัญใด

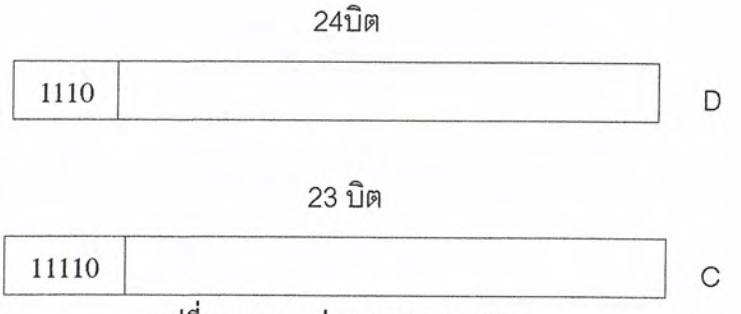
- Data ขนาดไม่คงที่ : ข้อมูลจากโปรโตคอลระดับบน

2.4.2 การจัดคลาสเครือข่าย

ไอพีแอดเดรสมีการจัดแบ่งออกเป็นกลุ่มหรือคลาส (class) เครือข่ายที่ใช้ในงานปัจจุบันมักสังกัดอยู่ในคลาสใดคลาสหนึ่งคือคลาส A,B หรือ C การแบ่งคลาสอาศัยจำนวนพรีฟิกซ์เครือข่ายที่แตกต่างกันตามรูปที่ 2.3 แต่ละคลาสจึงมีจำนวนเครือข่ายในสังกัดและจำนวนโฮสต์ต่อเครือข่ายไม่เท่ากัน

รูปที่ 2.3 การแบ่งคลาสเครือข่าย

การจัดคลาสตามรูปที่ 2.3 เป็นการจัดแบ่งตามการใช้งานเครือข่ายทั่วไป ในขณะที่ยังมีอีกสองคลาสซึ่งใช้เพื่อจุดประสงค์เฉพาะได้แก่ คลาส D และ E ดังรูปที่ 2.4 เครือข่ายคลาส D เป็นเครือข่ายแบบมัลติคาสต์ ส่วนคลาส E สงวนไว้ใช้งานหากมีความจำเป็นอื่นใดในอนาคต ทั้งสองคลาสนี้ไม่ได้แบ่งเลขโฮสต์จึงไม่กำหนดจำนวนโฮสต์ไว้



รูปที่ 2.4 การแบ่งคลาส D และ E

การจัดแบ่งคลาสโดยใช้พรีฟิกซ์เป็นการผนวกข้อมูล ใช้ในการเลือกเส้นทาง เช่น หากตรวจพบว่าพรีฟิกซ์ 2 บิตแรกมีค่าเป็น 10 แสดงว่าเป็นแอดเดรสในคลาส B ซึ่งมีค่า 16 บิตแรก กำหนดกลุ่มเครือข่ายและ 16 บิตถัดมาเป็นเลขโฮสต์

2.4.3 ลักษณะสำคัญของแต่ละคลาส

จำนวนเครือข่ายในแต่ละคลาสและจำนวนโฮสต์สูงสุดที่มีได้ สามารถคำนวณได้จากจำนวนบิตที่ใช้งานตามสูตร 2^n เมื่อ n คือจำนวนบิต ตัวอย่างเช่นในคลาส B มีเลขโฮสต์จำนวน 16 บิต จึงมีโฮสต์ได้ไม่เกิน 2^{16} ซึ่งเท่ากับ 65,536 แต่เลขโฮสต์ที่ทุกบิตเป็น "0" และ "1" จะสงวนไว้ใช้งานเฉพาะกรณี จำนวนโฮสต์จึงลดลงไป 2 โฮสต์ทุกเครือข่าย หรือมีโฮสต์ได้ไม่เกิน 2^{16}-2 = 65,534 สูตร 2^n-2 นี้จะใช้กับการคำนวณจำนวนเครือข่ายในคลาสและจำนวนโฮสต์ ทั้งคลาส A, B และ C ดังนี้

คลาส A

เครือข่ายในคลาส A มีบิตซ้ายสุดเป็น 0 และใช้ 7 บิตถัดมากำหนดเลขเครือข่าย ส่วนอีก 24 เป็นเลขโฮสต์ คลาส A จึงมีเลขเครือข่ายได้ 2^7 หรือ 128 ค่า แต่เครือข่าย 0.0.0.0 และ 127.0.0.0 สงวนไว้เป็นแอดเดรสเฉพาะงานคือ แอดเดรส 0.0.0.0 ใช้เป็นแอดเดรสกำหนดเส้นทางโดยปริยาย ส่วน 127.0.0.0 เป็นแอดเดรสลูปแบ็ค จำนวนเครือข่ายที่สังกัดในคลาส A จึงมีได้ 126 เครือข่ายคือเลขที่ขึ้นต้นด้วย 1.0.0.0 ถึง 126.0.0.0

แต่ละเครือข่ายในคลาส A มีแอดเดรสได้ 2^{24}-2 หรือเท่ากับ 16,777,214 คือตั้งแต่ 0.0.1 ถึง 255.255.254 เครือข่ายในคลาส A ใช้กับหน่วยงานขนาดใหญ่ที่ต้องการแอดเดรสเป็นจำนวนมาก เครือข่ายคลาสนี้จัดสรรให้กับหน่วยงานในยุคแรกเริ่มของอินเทอร์เน็ต แอดเดรสเครือข่ายที่เหลืออยู่ส่วนใหญ่จะสงวนไว้ ตัวอย่างของเครือข่ายในคลาสนี้มีเช่น

9.0.0.0 ibm.com

15.0.0.0 hp.com

20.0.0.0 csc.net

ขอให้สังเกตว่าในคลาส A นี้เมื่อกล่าวถึงเฉพาะเลขเครือข่ายก็จะเขียนเฉพาะที่แสดงเลขเครือข่ายที่ขนาด 8 บิต เท่านั้นเช่น 2 หรือ 26 ในทำนองเดียวกันเมื่อกล่าวถึงเฉพาะเลขโฮสต์ก็จะเขียนเฉพาะ 24 บิต เช่น 0.0.1 แต่ถ้าต้องการอ้างอิงโดยรวมทั้งเครือข่ายจะเขียนเฉพาะหมายเลขเครือข่ายโดยให้เลขโฮสต์เป็น "0" เช่น 2.0.0.0 รูปแบบการเขียนเช่นนี้ใช้กับคลาส B และ C เช่นกัน

คลาส B

เครือข่ายในคลาส B มีบิตแรกเริ่มเป็น 10 และใช้ 14 บิตถัดมากำหนดเลขเครือข่าย

จำนวนบิตที่กำหนดเลขโฮสต์มีขนาด 16 บิต คลาส B จึงมีสมาชิกเครือข่ายได้ 2^{14}-2 หรือ 16,382 คือ ตั้งแต่ 128.1.0.0 ถึง 191.254.0.0 แต่ละเครือข่ายมีเลขโฮสต์ได้ 2^{16}—2 หรือเท่ากับ 65,534 แอดเดรส หรือตั้งแต่ 0.1 ถึง 255.254

เครือข่ายในคลาส B มักจัดสรรให้กับหน่วยงานขนาดกลาง ในปัจจุบันมีเครือข่ายในคลาส B เหลือไม่มากนัก และมักไม่จัดสรรเครือข่ายในคลาสนี้ให้กับผู้ขอจดทะเบียนรายใหม่ หากไม่มีความจำเป็นอย่างแท้จริง ตัวอย่างของเครือข่ายนี้เช่น

129.123.0.0 usu.edu

130.149.0.0 tu-berlin.de

158.108.0.0 ku.ac.th

คลาส C

เครือข่ายในคลาส C มีพรีฟิกซ์ 110 และใช้ 21 บิตถัดมาเป็นเลขเครือข่าย จำนวนบิตที่เป็นเลขโฮสต์มีเพียง 8 บิต คลาส C จึงมีเลขเครือข่ายได้ตั้งแต่ 192.0.1.0 ถึง 223.255.254.0 รวม 2,097,150 เครือข่าย แต่ละเครือข่ายมีเลขโฮสต์ได้ตั้งแต่ 1 ถึง 254

จำนวนแอดเดรสได้จำกัดเพียง 254 แอดเดรสทำให้เครือข่ายจึงเหมาะสำหรับหน่วยงานขนาดเล็ก หากจำเป็นต้องใช้โฮสต์มากกว่านี้ต้องขอใช้เครือข่ายคลาส C หลายเครือข่าย ตัวอย่างของเครือข่ายคลาส C ได้แก่

198.6.250.0 isoc.org

198.137.240.0 whitehouse.gov

203.144.152.0 mcot.or.th

คลาส D และ E

เครือข่ายในคลาส D และ E ไม่มีการจัดแบ่งเลขเครือข่ายและเลขโฮสต์ คลาส D โดยมี 3 บิตแรกเป็น 111 จึงมีแอดเดรสตั้งแต่ 224.0.0.0 ถึง 239.255.255.255 แอดเดรสในคลาสนี้เรียกว่า มัลติคาสต์แอดเดรส (multicast address) เนื่องจากใช้ในเครือข่ายมัลติคาสต์

สำหรับคลาส E มีแอดเดรสจาก 240.0.0.0 ถึง 254.255.255.255 ซึ่งสำรองไว้เพื่อความจำเป็นเฉพาะงานในอนาคต

2.4.4 การแบ่งเครือขายย่อย (subnet)

การจัด subnet ใช้วิธีแบ่งบางส่วนของเลขโฮสต์มาใช้เป็น เลขซับเน็ต เพื่อกำหนดว่าเป็นเครือข่ายย่อยที่เท่าใด เช่น เครือข่าย 158.108.0.0 โดยจะใช้ 8 บิตแรกขอเลขโฮสต์เป็นเลขซับเน็ต และ 8 บิตที่เหลือใช้สำหรับเลขโฮสต์ ดังรูป 2.5

16 บิต	8 บิต	8 บิต
158.108	Subnetid	Hosted

รูปที่ 2.5 แสดงการแบ่ง subnet

2.5 ICMP (Internet Control Message Protocal)

ไอซีเอ็มพีเป็นโปรโตคอลที่ใช้แจ้งข้อผิดพลาด และปัญหาที่เกิดจากการลำเลียงไอพีเดทาแกรม โฮสต์และเราเตอร์จะต้องผ่านการออกแบบให้สามารถสร้างแพ็กเก็ตไอซีเอ็มพีรวมทั้งตอบสนองต่อแพ็กเก็ตไอซีเอ็มพีที่ได้รับ ภายในแพ็กเก็ตไอซีเอ็มพีจะบรรจุข่าวสารบ่งบอกถึงข้อผิดพลาดและปัญหาที่เกิดขึ้น

โปรโตคอลไอพีซึ่งกล่าวไว้เป็นโปรโตคอลที่มีความเรียบง่ายและมีโครงสร้างที่ได้รับการออกแบบมาเป็นอย่างดี แต่ไอพีมีคุณสมบัติแบบ "unreliable" และ "connectionless" จึงไม่มีกลไกการรับประกันในการลำเลียงเดทาแกรม หากมีปัญหาการลำเลียง เช่นเราเตอร์ทำงานผิดปกติหรือโฮสต์ไม่พร้อมรับเดทาแกรม ไอพีจะทิ้งเดทาแกรมนั้นไปโดยไม่ดำเนินการกับปัญหานั้น และใช้ไอซีเอ็มพีรายงานสาเหตุของปัญหากลับไปยังสถานีส่งต้นทาง

ไอซีเอ็มพีไม่ได้ช่วยรับประกันความถูกต้องในการส่งเดทาแกรมให้ไอพี เดทาแกรมยังคงมีโอกาสสูญหายหรือไปไม่ถึงจุดหมายเนื่องจากความผิดปกติใดๆการรับประกันและลำเลียงเดทาแกรมให้น่าเชื่อถือเป็นหน้าที่ของโปรโตคอลระดับบน ไอซีเอ็มพีเพียงช่วยเสริมการทำงานของไอพีให้มีประสิทธิภาพ และช่วยผู้ดูและระบบตรวจสอบปัญหาการทำงานในเครือข่าย

2.5.1 แพ็กเก็ตไอซีเอ็มพี

ไอซีเอ็มพีจะใช้ไอพีเดทาแกรมสำหรับแจ้งข่าวสาร ข้อมูลของไอพีเดทาแกรมก็คือข่าวสารที่บ่งบอกถึงปัญหาข้อผิดพลาดดังรูปที่ 2.6

ไอพีเดทาแกรม	
IP header	ICMP message

รูปที่ 2.6 ข่าวสารของไอซีเอ็มพีบรรจุอยู่ในไอพี

2.5.2 บริการในไอซีเอ็มพี

สาเหตุของปัญหาที่ทำให้ไม่สามารถส่งเดทาแกรมไปยังปลายทางมีได้มากมาย

เช่น ไม่สามารถติดต่อโฮสต์ปลายทางได้เนื่องจากสายสื่อสารขัดข้อง หรือโฮสต์ทำงานผิดปกติ หรือเราเตอร์ระหว่างทางอาจกำจัดเดทาแกรมทิ้งไปเนื่องจากหน่วยความจำไม่พอเพียงที่จะเก็บเดทาแกรม รวมทั้งปัญหาอื่นอีกหลายสาเหตุ

หากโฮสต์หรือเราเตอร์ต้องทิ้งเดทาแกรมเนื่องจากสาเหตุใดสาเหตุหนึ่ง ซอฟต์แวร์ไอซีเอ็มพีภายในโฮสต์หรือเราเตอร์จะแจ้งปัญหากลับไปยังต้นทางทันทีเพื่อให้ทราบถึงปัญหานอกเหนือไปจากการรายงานความผิดพลาดแล้ว ยังมีการใช้ไอซีเอ็มพีเพื่อการตรวจสอบและร้องขอข้อมูลจากสถานีปลายทางที่กำหนดได้เช่นกัน

ไอซีเอ็มพีจัดรูปแบบการรายงานข้อผิดพลาดโดยแบ่งฟิลด์ของข้อมูลในเดทาแกรมออกเป็น 4 ฟิลด์ ดังรูป 2.7 แต่ละส่วนมีความหมาย ดังนี้

รูปที่ 2.7 ฟอร์แมตของไอซีเอ็มพี

- Type ขนาด 8 บิต : กำหนดทั้งค่าความผิดพลาดและการรายงานสถานะ การใช้งานในปัจจุบันมีทั้งหมด 15 ประเภท ตามตาราง 2.1
- Code ขนาด 8 บิต : รหัสความผิดพลาดย่อย
- Checksum ขนาด 16 บิต : ค่าผลรวมตรวจสอบแบบ 1's complement สำหรับใช้ตรวจสอบความผิดพลาด โดยคำนวณผลรวมของ type , code และ contents
- Contents ขนาดไม่คงที่ : ฟิลด์นี้ใช้บรรจุข้อมูลข่าวสารเพิ่มเติมเพื่อแจ้งกลับซึ่งจะขึ้นอยู่กับค่า type และ code

Type	ความหมาย	การใช้งาน
0	Echo reply	แจ้งตอบกลับ
3	Destination unreachable	ไม่สามารถติดต่อปลายทางได้
4	Source quench	ให้ต้นทางลดระดับภาระงาน
5	Redirect	แจ้งเส้นทางที่เหมาะสมกว่า
8	Echo request	ร้องขอการตอบรับ
9	Router advertisement	เราเตอร์แจ้งประกาศตัวเอง
10	Router solicitation	โฮสต์ตรวจหาเราเตอร์
11	Time exceeded for datagram	เดทาแกรมใช้เวลาเกินกำหนด
12	Parameter problem on datagram	มีปัญหาพารามิเตอร์ในเดทาแกรม
13	Time stamp request	ร้องขอเวลาระบบ
14	Time stamp reply	แจ้งเวลาระบบกลับ
15	Information request	ร้องขอข่าวสาร
16	Information reply	แจ้งข่าวสารกลับ
17	Address mask request	ร้องขอแอดเดรสมาสค์
18	Address mask reply	แจ้งแอดเดรสมาสค์กลับ

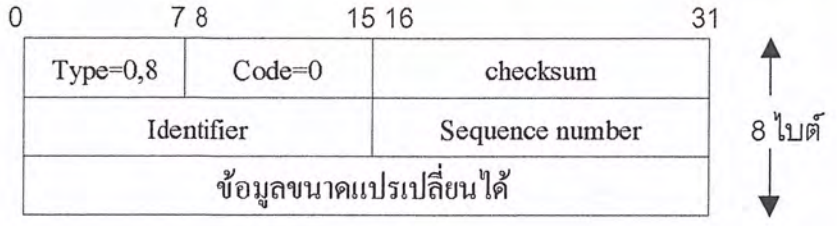
ตารางที่ 2.1 ประเภทความผิดพลาดของไอซีเอ็มพี

2.5.3 ไอซีเอ็มพี ชนิด 0 และ 8 : สะท้อนข้อมูล

ไอซีเอ็มพีชนิด 0 และ 8 ใช้ตรวจสอบว่าสถานีหรือเราเตอร์ปลายทางยังทำงานอยู่หรือไม่ ตัวอย่างโปรแกรมที่ใช้ คือ Ping

โดยโปรแกรม Ping จะส่งไอซีเอ็มพี 8 เพื่อสอบถาม (request) ไปยังสถานีหรือเราเตอร์ที่ต้องการตรวจสอบและรอการตอบกลับ หากโฮสต์หรือเราเตอร์เปิดให้บริการอยู่ก็จะตอบกลับด้วยไอซีเอ็มพีชนิด 0 (reply) ต่อจากนั้น Ping จึงนำผลการตอบกลับมาแสดงบนจอภาพ ฟอร์แมตของเดทาแกรมที่ใช้ร้องขอและตอบกลับมีโครงสร้างเช่นเดียวกับรูป 2.8

โปรแกรม Ping จะส่งเดทาแกรมต่อเนื่องกันไป การตอบกลับจะต้องตอบอ้างอิงกับทุกๆ เดทาแกรมด้วยค่า identifier และใช้ sequence number บอกลำดับเดทาแกรม



รูปที่ 2.8 เดทาแกรมไอซีเอ็มพีชนิด 0 และ 8

ฟิลด์ข้อมูลถัดมาเป็นข้อมูลตรวจสอบกล่าวคือ เมื่อส่งเดทาแกรมที่บรรจุข้อมูลหนึ่งๆ ออกไปแล้ว ทางฝ่ายรับจะต้องส่งข้อมูลที่ได้รับกลับมาด้วย โดยปกติแล้ว Ping จะใช้ 8 ไบต์แรกของข้อมูลเพื่อประทับเวลาสำหรับใช้คำนวณเวลาไปกลับ ขนดข้อมูลปกติที่ใช้คือ 56 ไบต์และสามารถเลือกค่าอื่นได้

2.5.4 ตัวอย่างการใช้ Ping

เป็นการใช้ Ping เพื่อส่งไอซีเอ็มพีชนิด 8 ไปยัง 161.246.10.21 โดยที่ 161.246.10.21 ตอบกลับมาด้วยไอซีเอ็มพีชนิด 0 ผลที่ได้มีความหมาย ดังนี้

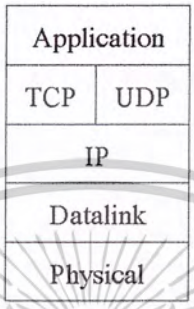
- ตัวเลข 72 ไบต์ จะเป็นขนาดของบัฟเฟอร์ที่ส่งไป
- ค่า sequence เป็นลำดับเดทาแกรมตาม โดยนำมาจากฟิลด์ sequence number เพื่อใช้ตรวจสอบว่าเดทาแกรมใดบ้างที่สูญหาย เพื่อนำมาคำนวณสถิติและเวลาเดินทางไปกลับโดยเฉลี่ยของเดทาแกรม
- ค่า ttl ใช้คำนวณหาจำนวนเราเตอร์ที่เดทาแกรมเดินทางผ่าน คำสั่ง Ping กำหนดค่า ttl เริ่มต้นเท่ากับ 255 และลดค่าลงครั้งละหนึ่งทุกครั้งที่ผ่านเราเตอร์ ตัวเลข 254 จึงหมายความว่าเดทาแกรมเดินทางผ่านเราเตอร์ 1 ตัว
- เวลาเดินทางไปกลับ (round-trip time : RTT) ซึ่งนับเริ่มต้นจากการส่งแพ็กเก็ตและได้รับแพ็กเก็ตตอบกลับในหน่วยวินาที Ping คำนวณเวลาเดินทางไปกลับโดยบรรจุเวลาไว้ในเดทาแกรมที่ส่งออกไป เมื่อได้รับเดทรแกรมตอบซึ่งบรรจุเวลานี้กลับมาดด้วยก็นำมาลบกับเวลาปัจจุบัน

2.6 UDP (User Datagram Protocol)

2.6.1 บทนำ

TCP/IP มีโปรโตคอลที่ให้บริการในระดับชั้นนี้สองโปรโตคอล คือ TCP และ UDP ดังรูป 2.9 TCP จัดเตรียมการส่งถ่ายข้อมูลโดยสถาปนาการเชื่อมต่อ และรักษาสภาพการเชื่อม

ต่อนั้นไว้ ตลอดจนยกเลิกสภาพเชื่อมต่อเมื่อสิ้นสุดการส่งถ่าย TCP มีกลไกรับประกันความถูกต้องของข้อมูล ส่วน UDP จะให้บริการที่เรียบง่ายกว่าโดยเพียงแต่ส่งข้อมูลออกไปโดยไม่ต้องสถาปนาการเชื่อมต่อระหว่างผู้ส่งและผู้รับ อีกทั้งไม่มีกลไกจัดการให้ความเชื่อถือในการลำเลียงข้อมูล UDP จะปล่อยให้เป็นหน้าที่ของโปรแกรมประยุกต์ดำเนินการเอง

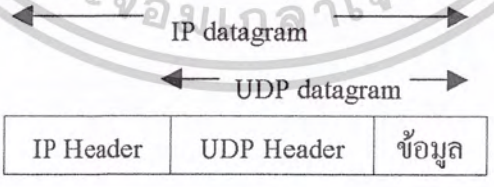


รูปที่ 2.9 ระดับชั้นของ TCP และ UDP

2.6.2 โปรโตคอล UDP

UDP เป็นโปรโตคอลระดับ transport ทำหน้าที่นำส่งข้อมูลจากโปรโตคอลประยุกต์ไปยัง IP ข้อมูลรวมกับ UDP เฮดเดอร์ เรียกว่า UDP Datagram หรือ User Datagram โดยมีรูปแบบการเอ็นแคบซูเลต ดังรูป 2.10

UDP ให้บริการแบบ "connectionless" กล่าวคือ ไม่สถาปนาการเชื่อมต่อระหว่างสถานีต้นทางและปลายทาง UDP ส่ง datagram โดยไม่ตรวจสอบว่าสถานีปลายทางพร้อมที่จะติดต่อหรือไม่ การสื่อสารลักษณะนี้อาจเทียบได้กับการส่งจดหมาย ผู้ส่งเพียงแต่มอบหมายให้ไปรษณีย์จัดส่งโดยไม่ต้องทราบว่าผู้รับปลายทางพร้อมรับหรือไม่



รูปที่ 2.10 การเอ็นแคบซูเลต UDP

หากมีปัญหาเกิดขึ้นกับ UDP datagram เช่น datagram สูญหาย หรือผิดลำดับหรือมี datagram ซ้ำกัน UDP จะไม่จัดการกับปัญหาเหล่านี้เนื่องจากไม่มีกลไกที่จะรับประกันความถูกต้องของ datagram โปรโตคอลประยุกต์ที่ใช้ UDP ต้องดำเนินการกับปัญหาเหล่านี้เอง

2.6.3 พอร์ต

มาถึงระดับชั้น transport ใน IP Header จะระบุชนิดของโปรโตคอลว่าต้องการเรียกใช้บริการ TCP หรือ UDP ต่อจากนั้น TCP และ UDP จะส่งข้อมูลไปยังโปรโตคอลระดับบนตามหมายเลยประจำโปรโตคอล หมายเลขที่ใช้ระบุโปรโตคอลประยุกต์เป็นตัวเลขขนาด 16 บิต และเรียกว่า พอร์ต (port) กล่าวอีกนัยหนึ่งแล้วพอร์ตเป็นเสมือนแอดเดรสประจำโปรโตคอลในชั้นประยุกต์

TCP/IP สงวนพอร์ตหมายเลข 1 ถึง 1023 ไว้ใช้ประจำโปรโตคอลประยุกต์โดยเรียกเลขพอร์ตนี้ว่า "well-known ports" ซึ่งใช้เป็นมาตรฐานทุกสถานีหรืออุปกรณ์เครือข่าย เช่น เซอร์ฟเวอร์ที่ให้เทลเน็ตใช้ TCP Port 23 server ที่ให้บริการ FTP ใช้ TCP Port 21 และ server ที่ให้บริการ TFTP ใช้ UDP Port 69 เป็นต้น

โปรโตคอลประยุกต์หนึ่งๆ อาจเรียกใช้บริการได้ทั้ง TCP และ UDP โดยใช้หมายเลขพอร์ตเดียวกันแต่เป็นอิสระต่อกัน เช่น echo มีหมายเลขพอร์ตเท่ากับ 7 สำหรับทั้ง UDP และ TCP หากมีการขอบริการ TCP และ UDP พร้อมกันที่พอร์ตนี้ process ที่เกิดขึ้นจะเป็น process อิสระต่อกัน

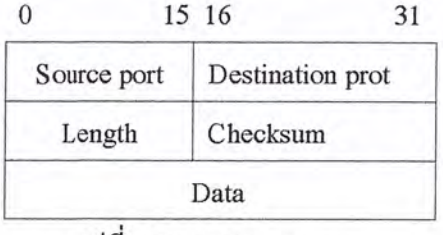
2.6.4 ซ็อกเก็ต

ซ็อกเก็ต (socket) หรือ ซ็อกเก็ตแอดเดรส (socket address) หมายถึง คู่ของ IP Address และเลขพอร์ต TCP และ UDP จะอาศัยซ็อกเก็ตเป็นตัวแยกแยะ Process ต้นทางและปลายทางที่ติดต่อระหว่างกัน หากพิจารณาถึงโปรโตคอลประยุกต์ซึ่งอนุญาตให้บริการได้หลายเซสชันพร้อมๆกัน และจำเป็นต้องแยกแยะเซสชันที่เกิดขึ้นให้ได้เพื่อให้ข้อมูลส่งถ่ายได้ถูกต้องตามคู่ต้นทางและปลายทาง

server ปลายทางจะใช้เลขพอร์ตตามพอร์ตมาตรฐานประจำ Application สำหรับสร้างซ็อกเก็ต แต่ client ต้นทางจะเลือกเลขพอร์ตนอกเหนือจากพอร์ตมาตรฐานมาใช้เป็นซ็อกเก็ตต้นทาง

2.6.5 UDP Datagram

รูปที่ 2.11 แสดงถึง UDP Datagram ซึ่งประกอบด้วยฟิลด์ดังต่อไปนี้



รูปที่ 2.11 UDP Datagram

- source ขนาด 16 บิต : พอร์ตสถานีต้นทาง
- destination port ขนาด 16 บิต : พอร์ตสถานีปลายทาง
- length ขนาด 16 บิต : บอกความยาวของ Datagram (ทั้ง Header และข้อมูล) เป็นจำนวนไบต์
- checksum ขนาด 16 บิต : ผลรวมตรวจสอบ คำนวณจากผลรวมของเฮดเดอร์และข้อมูล

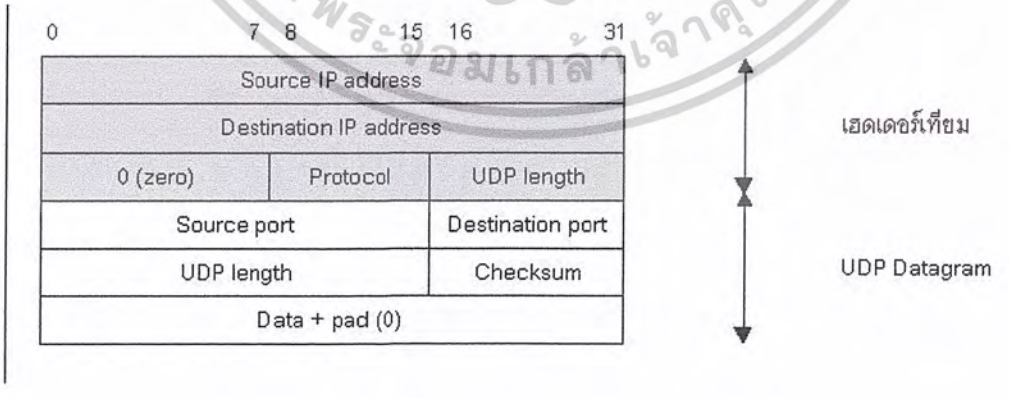
UDP Datagram มีเฮดเดอร์ที่เรียบง่ายและมีฟิลด์จำนวนน้อย ฟิลด์ที่สำคัญมีเพียงพอร์ตต้นทางและปลายทาง และค่าผลตวมตรวจสอบโดยไม่มีฟิลด์อื่นใดใช้ในการรับประกันความเชื่อถือการส่งข้อมูล

2.6.6 การคำนวณค่าผลรวมตรวจสอบ

UDP คำนวณผลรวมตรวจสอบจากเฮดเดอร์และข้อมูล ต่างจากผลรวมตรวจสอบของ IP ที่คำนวณจากเฮดเดอร์โดยไม่รวมข้อมูล ค่าผลรวมตรวจสอบของ UDP เป็นค่าเผื่อเลือก (optional) ที่จะใช้หรือไม่ก็ได้ ในขณะที่ TCP บังคับว่าต้องคำนวณผลรวมตรวจสอบเสมอ

ผลรวมตรวจสอบของ UDP คำนวณโดยบวกค่า 16 บิต แบบเลขเดิมเต็มหนึ่ง แต่ถ้าความยาวของ UDP Datagram เป็นเลขที่ให้แทรกค่า 0 อีก 1 ไบต์ เพื่อใช้เฉพาะขั้นตอนการคำนวณเท่านั้น (ไบต์ที่เพิ่มไม่ถูกส่งไปด้วย)

UDP และ TCP จะนำค่า เฮดเดอร์เทียม (psuedo – header) อีก 12 ไบต์เข้ามาร่วมคำนวณผลรวมตรวจสอบด้วย เฮดเดอร์เทียมนำมาจากบางฟิลด์ของ IP Datagram พื้นที่เข้มในรูปที่ 2.12 แสดงถึงเฮดเดอร์เทียมและ UDP Datagram



รูปที่ 2.12 UDP และ เฮดเดอร์เทียม

UDP ใช้เฮดเดอร์เทียมเพื่อคำนวณผลรวมตรวจสอบเท่านั้นและไม่ส่งเฮดเดอร์เทียมออกไป หากสถานีปลายทางได้รับ datagram และพบว่าผลรวมตรวจสอบไม่ถูกต้อง สถานีจะทิ้ง datagram นั้นไป แต่ละฟิลด์ในเฮดเดอร์เทียมมีความหมายดังนี้

- source address ขนาด 32 บิต : IP Address ต้นทาง
- destination address ขนาด 32 บิต : IP Address ปลายทาง
- zero ขนาด 8 บิต : เป็นศูนย์หมดทุกบิต
- protocol ขนาด 8 บิต : ชนิดของโปรโตคอล กรณีของ UDP จะมีค่าเท่ากับ 17
- length มีขนาด 16 บิต : ขนาดของเฮดเดอร์รวมกับข้อมูล (ไม่รวมเฮดเดอร์เทียม)

การเพิ่มเฮดเดอร์เทียมทำให้ผลรวมตรวจสอบจึงครอบคลุมซ็อกเก็ตเพื่อใช้ยืนยันว่าข้อมูลมาถึงปลายทางที่ถูกต้อง ขณะที่ผลรวมตรวจสอบเมื่อไม่มีเฮดเดอร์เทียมที่จะครอบคลุมเฉพาะหมายเลขพอร์ต

จากคุณสมบัติของเลขเติมเต็มหนึ่ง หากผลรวมตรวจสอบที่คำนวณได้มีค่าเท่ากับ 0 ยูดีพีจะใส่ค่าในฟิลด์นี้เป็นค่า –0 (ให้ทุกบิตมีค่าเป็น 1) เพราะหากใส่ค่าเป็น +0 ด้านปลายทางจะแปลความหมายได้ว่าฝ่ายส่งไม่นำค่าผลรวมตรวจสอบมาใช้ยืนยันความถูกต้อง

2.6.7การใช้บริการของยูดีพี

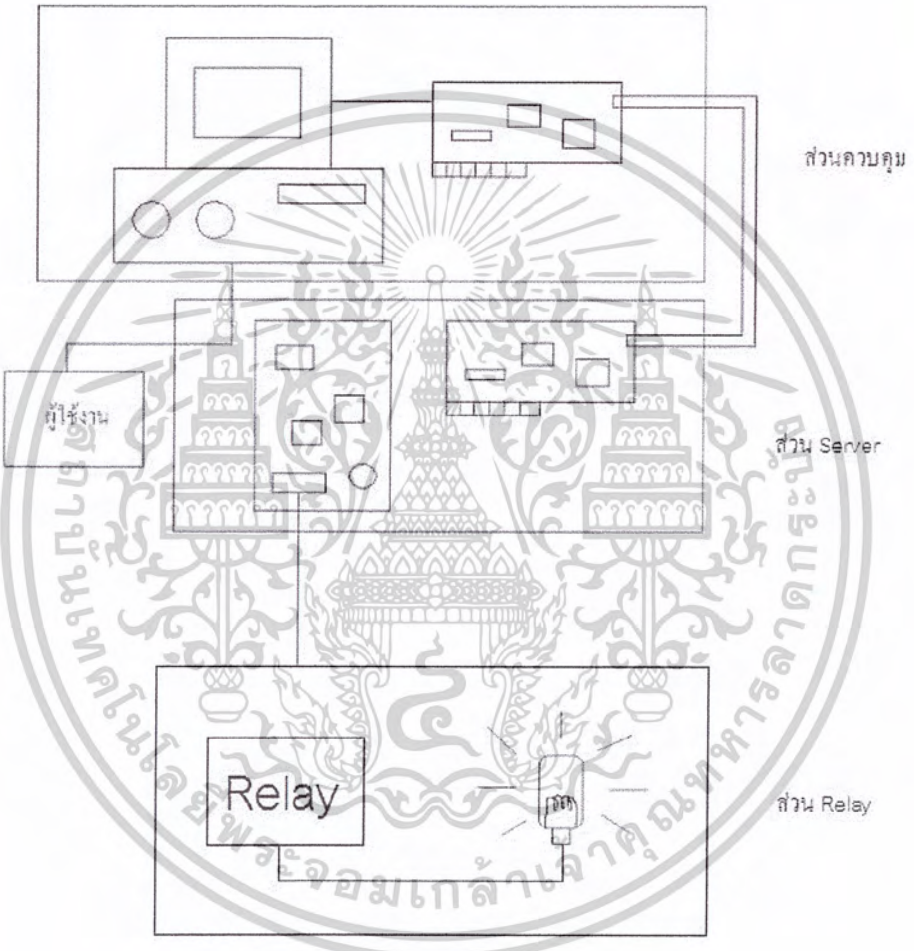
บริการของยูดีพีเหมาะสมกับโปรโตคอลประยุกต์หลายชนิดโดยเฉพาะโปรโตคอลที่ทำงานแบบไคลเอ็นต์-เซอร์ฟเวอร์ชนิดที่ใช้การร้องขอและตอบกลับ

ไคลเอ็นต์ทำหน้าที่ขอบริการและเซอร์ฟเวอร์ตอบกลับไปตามการร้องขอ การตอบกลับนี้เป็นรูปแบบหนึ่งของการตอบรับ (acknowledge) หากไคลเอ็นต์ไม่ได้รับคำตอบกลับภายในระยะเวลาที่กำหนดก็อาจขอบริการซ้ำใหม่ หากไคลเอ็นต์ร้องขอบริการซ้ำหลายครั้งเกินกำหนดโดยไม่ได้รับคำตอบ ไคลเอ็นต์จะถือว่าเซอร์ฟเวอร์ไม่อยู่ในสภาพที่ใบริการได้

ตัวอย่างของโปรโตคอลที่ใช้บริการยูดีพีแสดงไว้ บริการบางโปรโตคอลที่ขอยกตัวอย่างในที่นี้ได้แก่

- บริการ echo พอร์ต 7 : ทำหน้าที่สะท้อนเดทาแกรมที่ได้รับ
- บริการ discard พอร์ต 9 : รับเดทาแกรมมาแล้วทิ้งเดทาแกรมนั้นไป
- บริการ daytime พอร์ต 13 : ตอบวันเวลาของระบบของเซอร์ฟเวอร์ให้
- บริการ chargen ที่พอร์ต 19 : ตอบรหัสแอสกีกลับไปให้ทางไคลเอ็นต์
- บริการ domain พอร์ต 53 : ใช้ร้องขอบริการระบบชื่อโดเมน(DNS)
- บริการ bootps และ bootpc ที่พอร์ต 67 และ 68

บทที่ 3
วิธีดำเนินการวิจัย



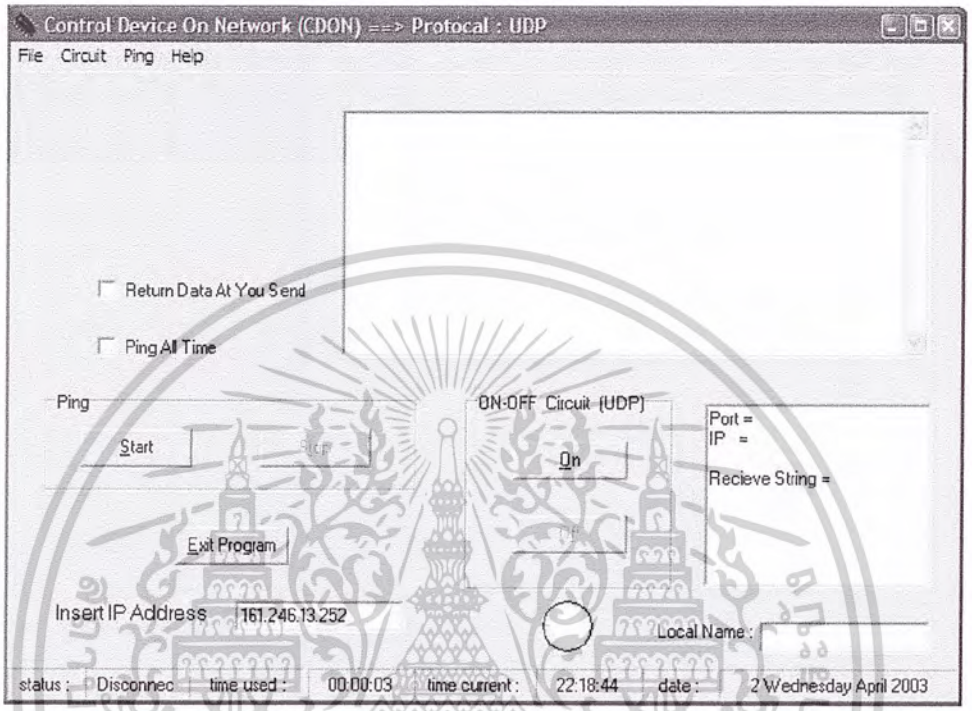
รูปที่ 3.1 แสดง Block Diagram การเชื่อมต่อของส่วนต่างๆ

ส่วนประกอบของโครงงาน ประกอบด้วย 3 ส่วน ดังนี้

3.1 ส่วนควบคุมและสั่งการ

จะเป็นส่วนที่ติดต่อกับ server ผ่านทางการ์ด LAN ไปยังระบบ Network โดยจะติดต่อผ่านทางสาย LAN เพื่อส่งคำสั่งและข้อมูลที่จำเป็นในการควบคุมอุปกรณ์ไฟฟ้าปลายทาง ซึ่งผู้ใช้งาน

(user) จะเป็นผู้ควบคุมส่วนนี้ผ่านทาง Computer ด้วยการสร้าง Application เป็นโปรแกรมควบคุมขึ้นมาโดยใช้ Delphi เขียนขึ้นมา



รูปที่ 3.2 แสดงหน้าตาของโปรแกรมควบคุม Relay โดยใช้โปรแกรม Delphi เขียน

3.2 ส่วน server ประกอบด้วย

3.2.1 ส่วนการ์ด LAN

มีไว้เพื่อติดต่อรับข้อมูลมาจาก Computer ตัวอื่นๆ ทั่วโลก และตัวกำหนด IP Address เพื่อที่จะให้เราสามารถระบุตำแหน่งที่อยู่ของ Server ที่ติดต่อกับอุปกรณ์ปลายทาง ทำให้ computer สามารถส่งคำสั่งและข้อมูลมายังส่วน Server นี้และส่งคำสั่งไปยังบอร์ด EVB – 52 ได้ เพื่อใช้ควบคุมอุปกรณ์ปลายทางต่อไป

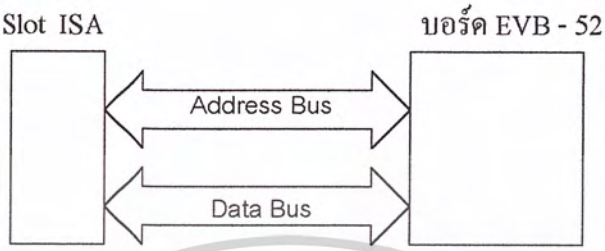
3.2.2 ส่วนบอร์ด EVB – 52

เป็นส่วนของ Microcontroller เพื่อใช้ในการควบคุมตัว Relay เพื่อไปเปิด – ปิด อุปกรณ์ไฟฟ้าต่างๆ ได้ (ในที่นี้จะใช้หลอดไฟ) ซึ่งจะมีการอัดโปรแกรมลงไปบนหน่วยความจำบนตัว Microcontroller

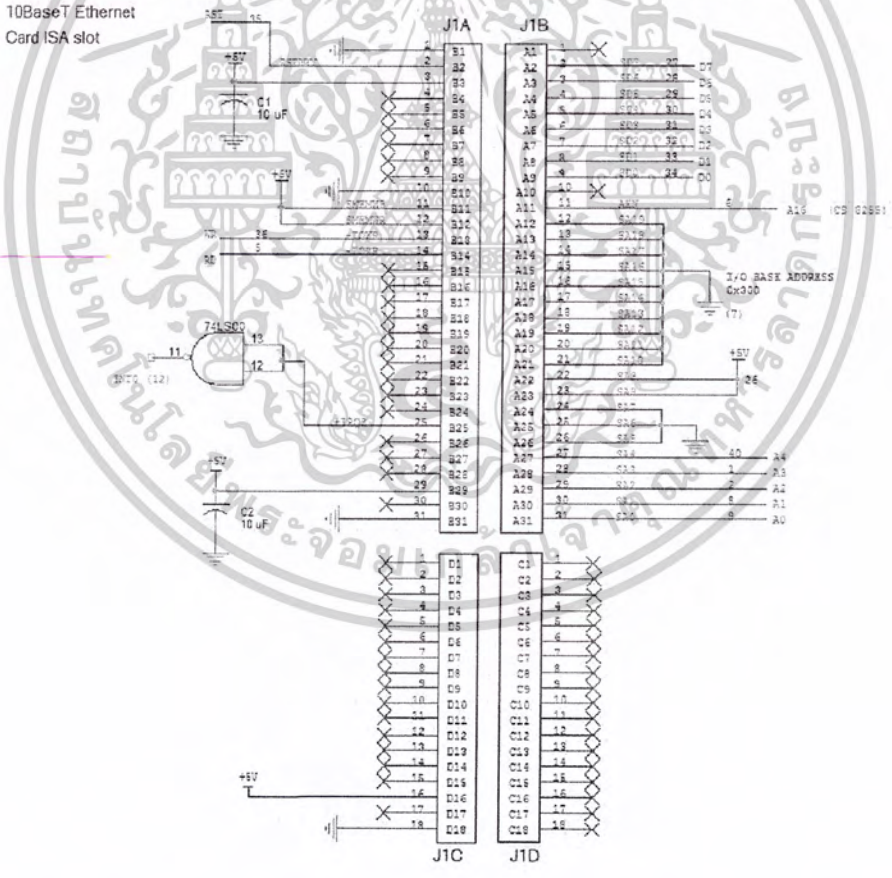
3.2.3 การติดต่อระหว่างการ์ด LAN กับ บอร์ด EVB – 52

เราจะต่อสาย Address Bus และสาย Data Bus จากบอร์ด EVB – 52 ไปยัง Slot ISA ที่จะต่อกับการ์ด

LAN และจะมีการต่อ Nand Gate เข้ากับ Slot ISA เพื่อทำให้เกิดการ Interrupt ได้ โดยเราสามารถที่จะกำหนด IRQ ได้ (ในที่นี้จะใช้ IRQ3)



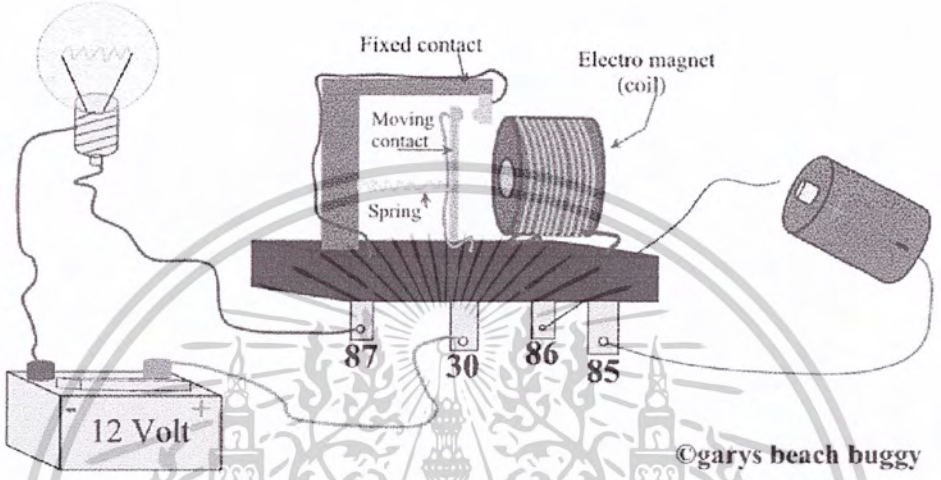
รูปที่ 3.3 แสดงการติดต่อระหว่างการ์ด LAN กับ บอร์ด EVB – 52



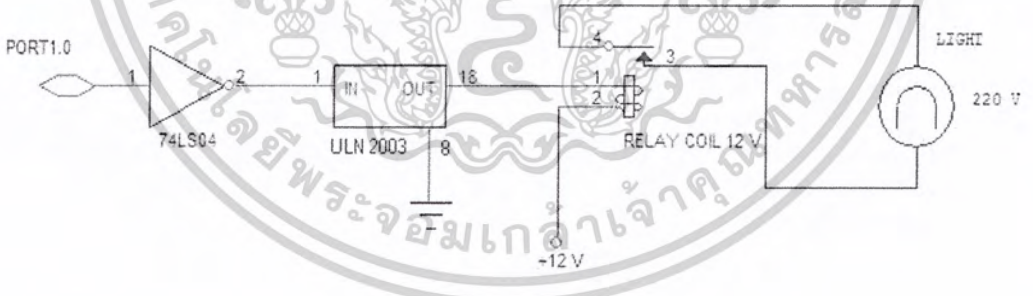
รูปที่ 3.4 แสดงภาพ Schematic ของ Slot ISA ต่อเชื่อมกับบอร์ด EVB – 52

3.3 ส่วน Relay

เป็นส่วนที่ใช้ควบคุมการเปิด – ปิด อุปกรณ์ไฟฟ้า เช่น หลอดไฟ ซึ่งจะรับข้อมูลการทำงานจากส่วน server และส่วน server นี้ จะไปควบคุมตัว Relay อีกที



รูปที่ 3.5 แสดงการเชื่อมต่อวงจร Relay

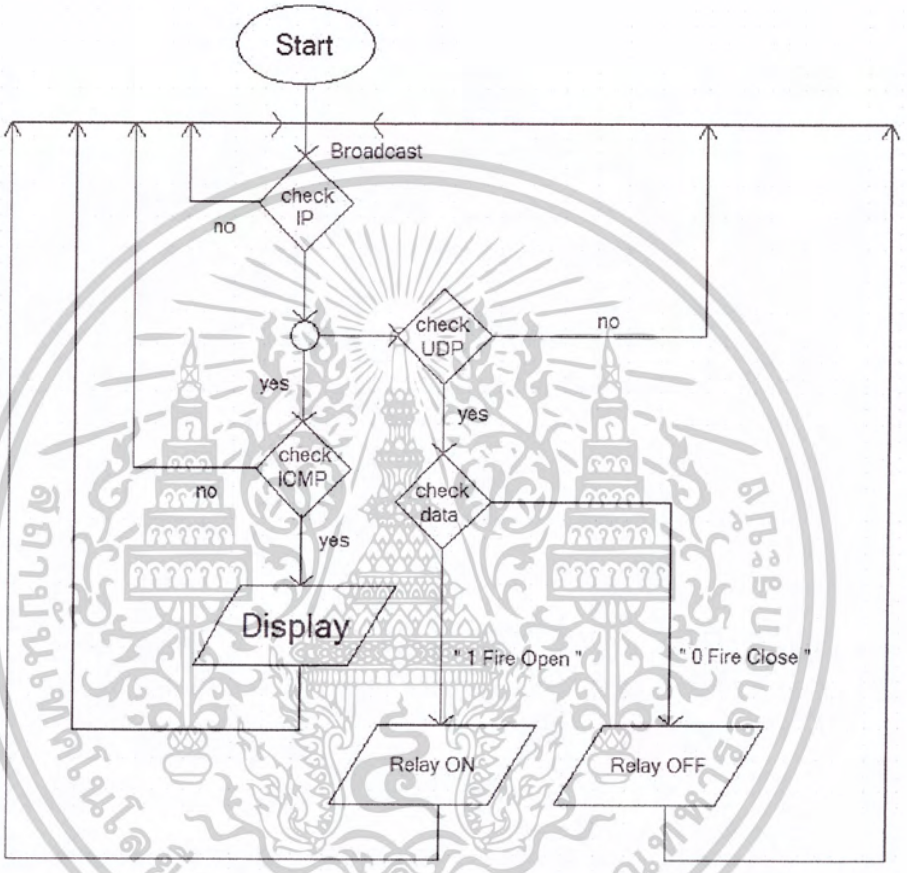


รูปที่ 3.6 แสดงภาพ Schematic ของวงจร Relay ที่เชื่อมต่อกับ บอร์ด EVB - 52

3.4 แสดง Flowchart การทำงานของตัว Server

จากรูปที่ 3.7 จะเป็นการทำงานของตัว Server ที่จะมีการรับการ Broadcast จาก client ตัวอื่นๆ ที่มีการเชื่อมต่ออินเทอร์เน็ตเข้าด้วยกัน โดย client จะถามหา MAC Address จากเครื่อง server ซึ่งจะมีการ check IP Address ถ้า IP Adderss นั้นเป็นของตัว server ก็จะเข้าสู่ขั้นตอนการ check ว่า

เป็น ICMP หรือ UDP เพื่อตรวจสอบการทำงานโดยจะรับ packet ข้อมูลมาแล้วเกิดการทำงานตามคำขอของ client



รูปที่ 3.7 แสดง Flowchart แสดงการทำงานของตัว Server

บทที่ 4
ผลการทดลองและอภิปรายผล

4.1 ผลการทดลอง

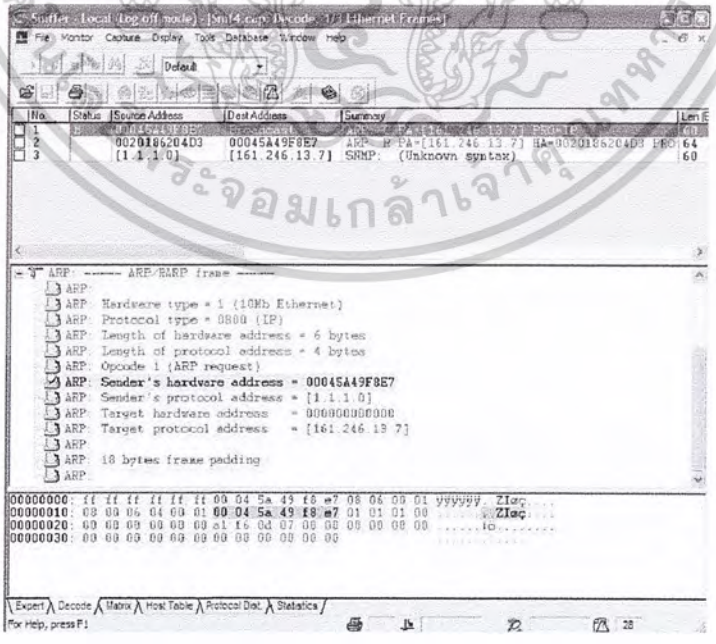
เราจะสรุปเป็น 2 กรณี ดังนี้

1. กรณีที่ต่อกับ Computer โดยตรง (ใช้สาย cross) ระบบ server ที่สร้างขึ้นนั้นไม่มีปัญหาใดๆ ตัว server สามารถทำงานได้ โดยสามารถปิด – เปิดไฟได้โดยไม่มีปัญหาอะไร

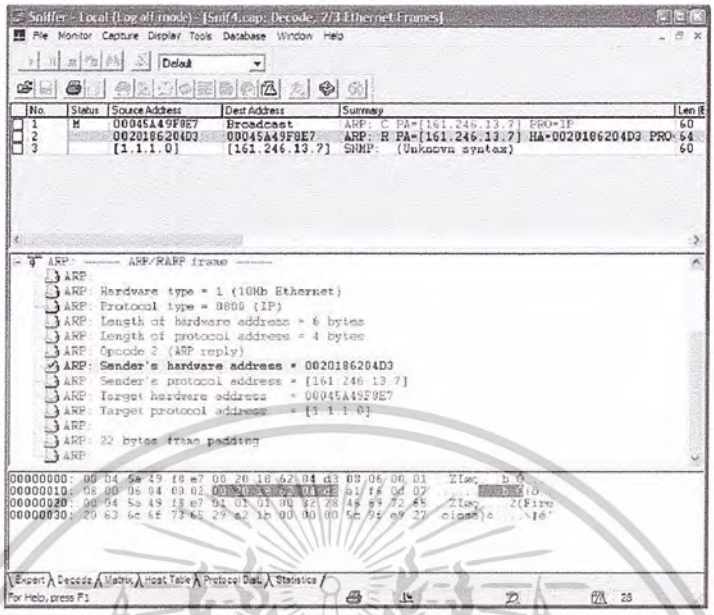
2. กรณีที่นำตัว server ไปต่อกับระบบ network ใหญ่ ตัวระบบ server ยังมีปัญหา คือ ตัวระบบ Server ที่สร้างขึ้นนั้นยังไม่สามารถที่จะทำงานติดต่อกันได้นาน ๆ เนื่องมีปัญหาทางด้าน software อยู่ แต่ระบบการปิด – เปิดไฟไม่มีปัญหาใดๆ

4.1.1 การ Broadcast ของระบบ Network

ในกรณีที่ทางตัว Client มีการส่ง Packet ใดๆ ไปยัง Server ปลายทางตามค่า IP Address ที่กำหนดไว้ ทางตัว Client ก็จะทำการ Broadcast กระจายไปทั่วระบบ Network เพื่อค้นหา MAC Address ของ Server ปลายทาง เมื่อเจอค่า MAC Address ของ Server ปลายทางแล้ว ทางตัว Server ปลายทางก็จะทำการตอบกลับมายังตัว Client ดังรูป 4.1 และ 4.2



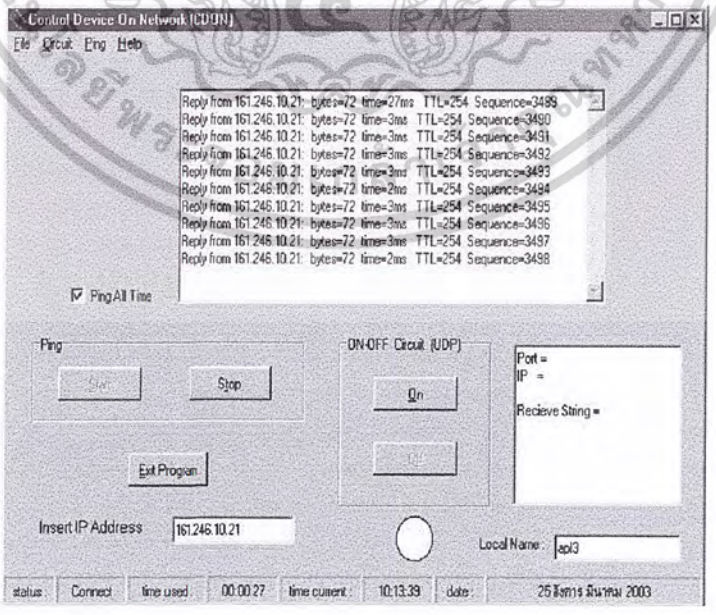
รูปที่ 4.1 แสดงการจับ Packet (ARP Request)



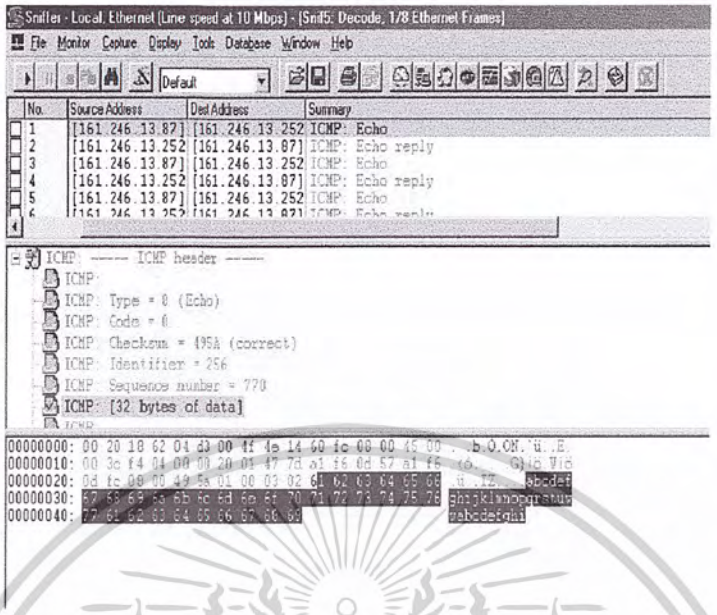
รูปที่ 4.2 แสดงการจับ Packet (ARP Reply)

4.1.2 การทดสอบระบบโดยใช้ Program Ping

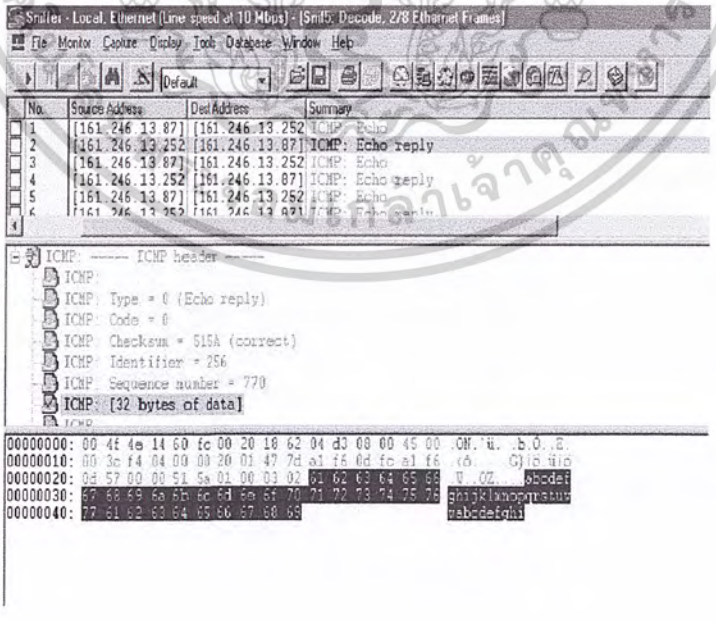
เราจะใช้ Application ที่สร้างขึ้นมาจาก Delphi ทำการทดสอบ Program Ping โดยจะ ping ไปยัง IP : 161.246.10.21 จะได้ผลการทดลองตามรูปที่ 4.3



รูปที่ 4.3 แสดงการใช้ Program Ping จาก Application ที่สร้างจาก Delphi



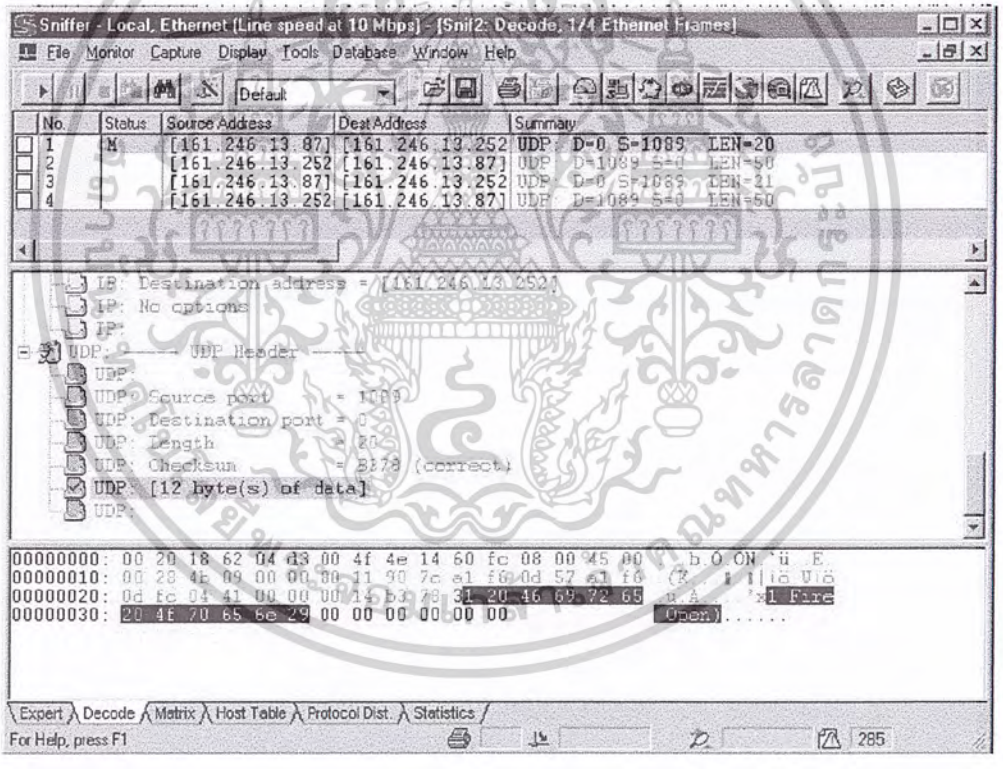
รูปที่ 4.4 แสดงการจับ Packet ICMP (Echo)



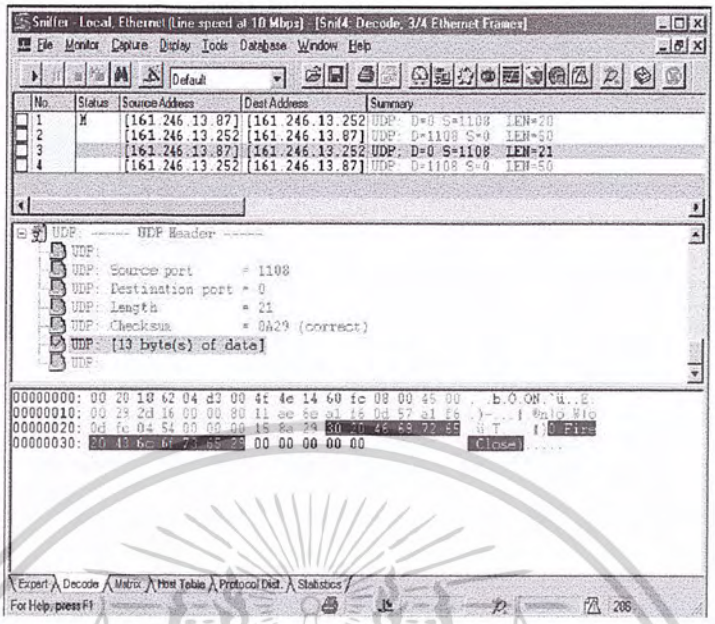
รูปที่ 4.5 แสดงการจับ Packet ICMP (Echo reply)

4.1.3 วิธีการเปิด – ปิด หลอดไฟ

เราจะทำการตรวจสอบข้อความ (String) ที่เราส่งไปจากโปรแกรม Application ที่เราสร้าง โดยในกรณีที่ทำการเปิด เราจะทำการส่งข้อความ 1(Fire Open) ซึ่งเราจะทำการตรวจเช็คตัวอักษร 2 ตัวอักษร นั่นก็คือ "1" และ "O" ซึ่งในโปรแกรมที่เราเขียนใส่ในตัว Controller จะนำตำแหน่งที่ตัวอักษรที่เราต้องการจะตรวจสอบ (ทราบโดยการใช้โปรแกรม Sniffer จับ) มาเก็บค่าไว้ในรีจิสเตอร์ในตัว Controller แล้วนำค่าในตำแหน่งนั้นๆมาตรวจสอบ เช่นเดียวกับการปิดไฟโดยเราจะส่งข้อความ (String) เป็น 0(Fire Close) โดยจะทำการตรวจสอบตัวอักษร "0"และ "C" หลังจากนั้นก็ทำการตรวจสอบเช่นเดียวกับที่กล่าวมา ดังแสดงดังรูป



รูปที่ 4.6 แสดงการจับ Packet โดยการใช้โปรแกรม Sniffer ในกรณีการเปิดไฟ



รูปที่ 4.7 แสดงการจับ Packet โดยการใช้โปรแกรม Sniffer ในกรณีการปิดไฟ

```
PDoD99: cjne a,#17,check_snmp   ;stop when get UDP

        mov   dpl, #2eh
        movx  a, @dptr
        cjne  a, #31h, check1
        mov   dpl, #35h
        movx  a,@dptr
        cjne  a, #4fh, check_snmp
        clr   p1.0
        lcall process_snmp
        ret

check1: cjne  a, #30h, check_snmp
        mov   dpl, #35h
        movx  a,@dptr
        cjne  a, #43h, check_snmp
        setb  p1.0
        lcall process_snmp
        ret
```

รูปที่ 4.8 แสดง Sourcecode ในการตรวจสอบการเปิด – ปิด หลอดไฟ

```
; ===========================================================================
; Receive Buffer.  This is where the received packet gets put to
RXBuffer        equ     0C000h          ; Location in memory to store received packet

        ; Frame header
FrameStatus     equ     RXBuffer ; Status of the received Frame
FrameNPP        equ     FrameStatus+1   ; Next Packet Pointer
FrameLength     equ     FrameNPP+1      ; Holds Length of frame
        ; Ethernet header
EDestMac        equ     FrameLength + 2 ; MAC Address that the packet is destined for
ESrcMac         equ     EDestMac + 6    ; MAC Address of device that packet came from
ETypeLenequ     ESrcMac + 6     ; Type of information within packet, i.e. ICMP, IP
EPacketData     equ     ETypeLen + 2    ; Data without Frame or ethernet header

HdrLength       equ     FrameStatus - EDestMac      ; length of the ethernet header
Buffer          equ     EDestMac            ; Buffer contains ethernet header as well as data
```

รูปที่ 4.9 แสดงตำแหน่งของ Buffer ของ Chip Realtek

4.1.4 Checksum IP

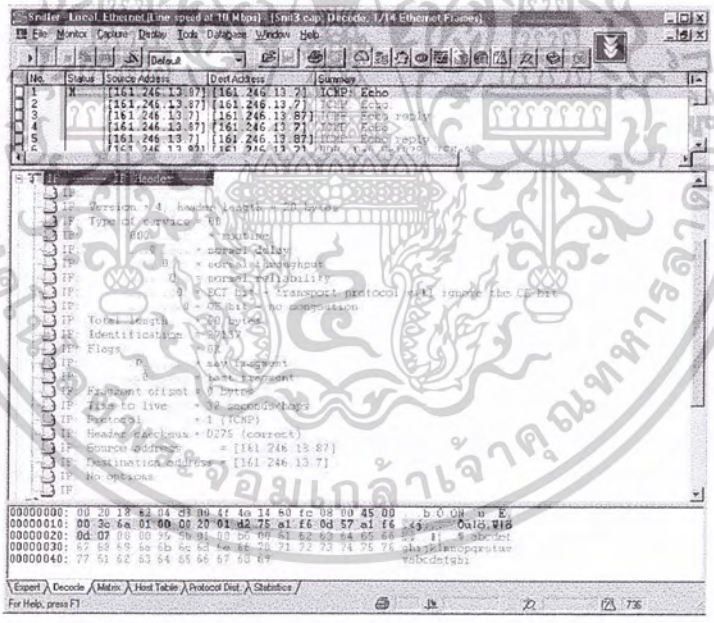
ค่า Checksum ของ IP Address นั้นจะเป็นการรวมค่าของ Version , Header length , Type of service , Total length , Identification , Fragment offset (จะรวม Flags เข้าไปด้วย) , Time to live , Protocal , Source address และ Destination address โดยจะเป็นการรวมแบบ 16 บิต 1's complement และถ้าเกิดในกรณีที่ทำการบวกกันแล้วเกิดการ overflow ก็ให้นำค่าบิตที่เกินนั้นมาทำการบวกกับ 16 บิตแรก เช่น

$$7AFD + FA8E = (1)\ 758B$$

นำบิตที่เกิดการ overflow มา นั่นคือ "1" มาบวกกับค่า 758B อีกครั้งจะได้เท่ากับ 758C หลังจากนั้นค่อยทำการ 1's complement

Example

ทำการ Checksum IP ตามรูปที่ 4.10



รูปที่ 4.10 แสดงการจับ Packet IP

จากรูปที่ 4.10 จะเห็นได้ว่ามีค่า

Version , Header length = 45

Type of service = 00

Total length = 00 3C

Identification	= 6A 01
Fragment offset	= 00 00
Time to live	= 20
Protocal	= 01
Source address	= A1 F6 0D 57
Destination address	= A1 F6 0D 07

เมื่อนำมาเรียง Packet จะได้เป็น

ค่า Checksum ที่ยังไม่ทราบค่า

45 00 00 3C 6A 01 00 00 20 01 xx xx A1 F6 0D 57 A1 F6 0D 07

จากนั้นทำการรวมแบบ 16 บิต จะได้

4500 + 003C + 6A01 + 0000 + 2001 + A1F6 + 0D57 + A1F6 + 0D07 = 2 2D88

นำบิตที่เกิดการ overflow มาบวกเพิ่มจะได้

2D88 + 2 = 2D8A

ทำการ 1's complement จะได้

2D8A —1's complement→ D275

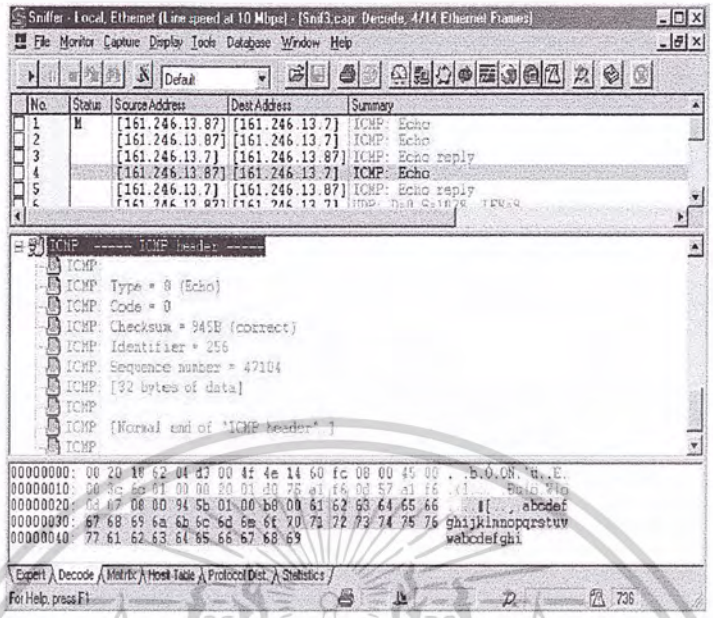
จะได้ค่า Checksum = D275

4.1.5 Checksum ICMP

ค่า Checksum ของ ICMP นั้นจะเป็นการรวมค่าของ Type , Code , Identifier , Sequence number และ Byte(s) of data โดยจะเป็นการรวมแบบ 16 บิต 1's complement และถ้าเกิดในกรณีที่ทำการบวกกันแล้วเกิดการ overflow ก็ให้นำค่าบิตที่เกินนั้นมาทำการบวกกับ 16 บิตแรก (เช่นเดียวกับการคิด Checksum IP)

Example

ทำการ Checksum ICMP ตามรูปที่ 4.11



รูปที่ 4.11 แสดงการจับ Packet ICMP

จากรูปที่ 4.11 จะเห็นได้ว่ามีค่า

Type = 08

Code = 00

Identifierh = 01 00

Sequence number = B8 00

Byte(s) of data = 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69

เมื่อนำมาเรียง Packet จะได้เป็น

ค่า Checksum ที่ยังไม่ทราบค่า

08 00 xx xx 01 00 B8 00 61 62 63 64 65 66 67 68 69 6A 6B 6C
6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69

จากนั้นทำการรวมแบบ 16 บิต จะได้

0800 + 0100 + B800 + 6162 + 6364 + 6566 + 6768 + 696A + 6B6C + 6D6E +
6F70 + 7172 + 7374 + 7576 + 7761 + 6263 + 6465 + 6667 + 6869 = 7 6B9D

นำบิตที่เกิดการ overflow มาบวกเพิ่มจะได้

6B9D + 7 = 6BA4

ทำการ 1's complement จะได้

6BA4 $\xrightarrow{\text{1's complement}}$ 945B

จะได้ค่า Checksum = 945B

4.1.6 Checksum UDP

ค่า Checksum ของ UDP นั้นจะเป็นการรวมค่าระหว่าง Header UDP ซึ่งประกอบไปด้วย

- Source port
- Destination port
- Length
- Byte(s) of data

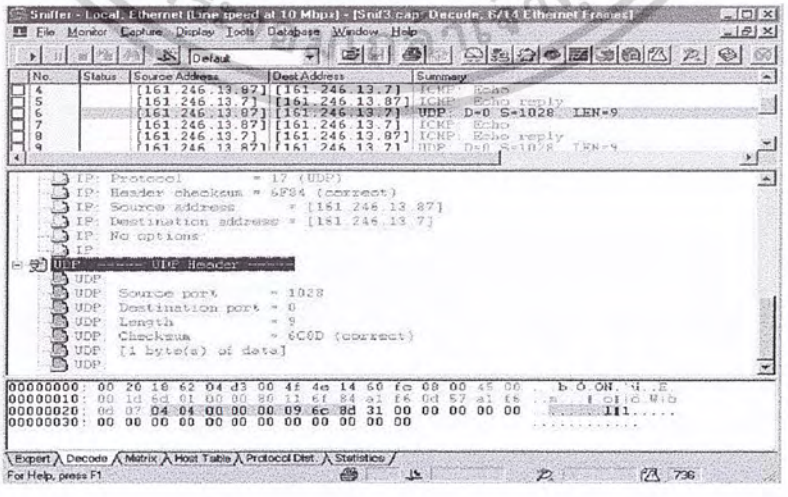
นำมารวมกับ Header เทียม ซึ่งประกอบไปด้วย

- Protocal
- Source address
- Destination address
- UDP length

โดยจะเป็นการรวมแบบ 16 บิต 1's complement และถ้าเกิดในกรณีที่ทำการบวกกันแล้วเกิดการ overflow ก็ให้นำค่าบิตที่เกินนั้นมาทำการบวกกับ 16 บิตแรก (เช่นเดียวกับการคิด Checksum IP หรือ Checksum ICMP)

Example

ทำการ Checksum UDP ตามรูปที่ 4.12



รูปที่ 4.12 แสดงการจับ Packet UDP

จากรูปที่ 4.12 จะเห็นได้ว่ามีค่า

- ในส่วน Header เทียม

Protocal = 00 11

Source address = A1 F6 0D 57

Destination address = A1 F6 0D 07

UDP length = 00 09

- ในส่วน UDP Header

Source port = 04 04

Destination port = 00 00

Length = 00 09

Byte(s) of data = 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

เมื่อนำมาเรียง Packet จะได้เป็น

ค่า Checksum ที่ยังไม่ทราบค่า

00 11 A1 F6 0D 57 A1 F6 0D 07 00 09 04 04 00 00 00 09 xx xx 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

จากนั้นทำการรวมแบบ 16 บิต จะได้

0011 + A1F6 + 0D57 + A1F6 + 0D07 + 0009 + 0404 + 0000 + 0009 + 3100 + 0000 + 0000 + 0000 + 0000 + 0000 + 0000 + 0000 + 0000 = 1 9371

นำบิตที่เกิดการ overflow มาบวกเพิ่มจะได้

9371 + 1 = 9372

ทำการ 1's complement จะได้

9372 —1's complement→ 6C8D

จะได้ค่า Checksum = 6C8D

4.1.7 การพิจารณาข้อมูล UDP ที่ฝั่งผู้รับ

เมื่อฝ่ายรับได้รับ UDP diagram มาแล้ว จะทำการตรวจสอบความถูกต้องของ UDP data โดยการนำ data มา add กับค่า checksum หากว่าผลรวมได้ค่าเป็น 1 ทุกบิต เช่น 1111 แสดงว่า data ที่ได้รับมาถูกต้อง และจะรับ UDP data นั้น แต่ถ้าผลรวมไม่ได้เป็น 1 ทุกบิต เช่น 1110 ฝ่ายรับจะปฏิเสธการรับ UDP data นั้น โดยที่ผู้ส่งก็ไม่รู้ว่าข้อมูลที่ส่งไปนั้นผู้รับได้รับหรือไม่ จึงเป็นข้อด้อยของ UDP ที่ไม่มีความน่าเชื่อถือในการส่งข้อมูล ไม่มีการรรับประกันว่าข้อมูลนั้นจะถึงมือผู้รับหรือไม่ หากข้อมูลไปไม่ถึงหรือถูกปฏิเสธจากผู้รับก็เสมือนกับข้อมูลส่วนนั้นสูญหายไปเลย

4.1.8 ระยะทางในการควบคุมไฟ

1. สามารถเปิด - ปิดจากภายในห้องเดียวกันได้
2. สามารถเปิด – ปิดข้ามห้องได้
3. สามารถเปิด – ปิดข้ามตึกได้

4.2 การอภิปรายผล

1. ผลของการทดลองยังไม่เป็นที่พอใจ เนื่องจากตัว server ยังไม่สามารถทำงานได้ในระบบใหญ่ๆ ได้นานพอ
2. ระบบการปิด – เปิดไฟ ยังต้องพัฒนาให้สามารถปิด – เปิด จากที่ต่างๆ ให้ง่ายกว่านี้ เช่น สามารถปิด – เปิดไฟ จากตัวเบราเซอร์ได้ เป็นต้น (ในที่นี้จะปิด – เปิด จากโปรแกรมที่สร้างขึ้นจาก Delphi)

รูปที่ 4.13 แสดงภาพของตัว Server โดยใช้ Microcontroller เชื่อมกับ LAN card ชนิด 10BaseT

รูปที่ 4.14 แสดงภาพของตัว Server ในการเปิด – ปิด ไฟ

บทที่ 5
บทสรุป การประยุกต์ใช้งาน และข้อเสนอแนะ

5.1 บทสรุป

การประยุกต์ระบบ เซิร์ฟเวอร์ร่วมกับไมโครคอนโทรเลอร์ MCS-51 เป็นประโยชน์อย่างยิ่งในการพัฒนาระบบเซิร์ฟเวอร์ในอนาคต โดยโครงงานนี้จะใช้งานในด้านการควบคุมอุปกรณ์ไฟฟ้าผ่านทางระบบอีเทอร์เนต และใช้การควบคุมโดย GUI ซึ่งง่ายต่อการควบคุม ไม่ว่าผู้ใช้จะอยู่ที่ใดก็ตามที่เครือข่ายอินเทอร์เนตสามารถเข้าถึง อย่างไรก็ตามโครงงานนี้ยังคงมีข้อบกพร่องหรือแนวทางในการปรับปรุงอีกมาก

5.2 การประยุกต์ใช้งาน

ระบบเซิร์ฟเวอร์นี้สามารถนำไปประยุกต์ใช้งานได้อย่างมากมาย ไม่ว่าจะเป็นอุปกรณ์ไฟฟ้าตามบ้านเรือน เช่น พัดลม หลอดไฟ กล้องดิจิตอล เป็นต้น อีกทั้งยังสามารถนำไปใช้ในงานควบคุมทางอุตสาหกรรม และภาคเกษตรกรรม ได้อีกด้วย

5.3 ข้อเสนอแนะ

แนวทางการปรับปรุง และการพัฒนาระบบเซิร์ฟเวอร์นี้ให้มีประสิทธิภาพ สามารถแบ่งเป็นการแก้ไขในส่วนระบบเครือข่าย และ ระบบการควบคุม เป็นหัวข้อต่างๆ ได้ดังนี้

1) พัฒนาให้มีความสามารถเข้าถึงระบบ SNMP โปรโตคอล
2) พัฒนาให้ผู้ใช้สามารถใช้งานในการควบคุมได้จากโฮมเพจ
3) พัฒนาให้ระบบเซิร์ฟเวอร์มีการติดต่อกับ TCP/IP โปรโตคอลได้นานขึ้น

เอกสารอ้างอิง

สุรศักดิ์ สงวนพงษ์. 2543. สถาปัตยกรรมและโปรโตคอลทีซีพี/ไอพี 1. กรุงเทพฯ : บริษัท เอช.เอ็น. กรุ๊ป จำกัด

ภาคผนวก

Sourcecode Ethertest.asm

```
;-------------------------------------------------------------------
; ETHTEST.ASM
;-------------------------------------------------------------------
$list
$include(dcl552r.txt)

CR        equ   0Dh
LF        equ   0Ah

IPAddr1 equ   161                ; IP Address of the NIC
IPAddr2 equ   246
IPAddr3 equ   13
IPAddr4 equ   252

BaseProg equ 8000H
     cseg at 8000h

     ljmp        Begin                    ; Start of code

     ORG         BaseProg+0003h           ; External Interrupt 0
     ajmp        Int8390

     org     BaseProg+0023h               ; Start of serial I/O routine
     jb      TI, ProcessTI           ; Jump if character sent
     clr     RI                           ; Clear any receive bit
exit:     reti
ProcessTI:     acall     OutSend
```

```
                ajmp    exit
begin:      mov     Sp, #50h            ; Move SP out of the way
            mov     tmod, #00           ; Set up timer 1
            orl     tmod, #20H
            mov     s0con, #52H         ; Mode 1
            orl     pcon, #128
            mov     th1, #250       ; Reload value for 19400 baud, 11.0592 MHZ XTAL

            setb    ea                  ; Enable interrupts
            setb    es0                 ; Enable serial interrupts
            setb    tr1                 ; Set timer 1 going
            setb    PS0                 ; Make the serial line high priority
                                        ; so can send messages in interrupt
            setb    EX0                 ; Enable External Interrupt 0
            setb    IT0                 ; Make edge triggered

            clr     SendingStr          ; Initally set as no data being sent
            clr     SendingChar
            clr     dmaing              ; set flag to not dmaing
                                        ;  Add in delay for NEx000 card to reset itself
lp0:        mov     R0, #40
lp1:        mov     R1, #0
            djnz    R1, $
            djnz    R0, lp1

            mov     DPTR, #HelloStr
            acall   SendStr
            jb      SendingStr, $

            acall   Probe8390           ; see if Card present
```

```
        jnc     FoundCard               ; if not carry, then all OK


        cjne    a, #NoCardFound, Init2  ; otherwise, say error
        mov     DPTR, #NotNeStr
        lcall   SendStr
        jb      SendingStr, $
        sjmp    begin
        sjmp    $                               ; halt processing

Init2:  cjne    a, #NoResetAck, Init3
        mov     DPTR, #NoResetAcks
        lcall   SendStr
        jb      SendingStr, $
        sjmp    begin
        sjmp    $                       ; Halt

Init3:  cjne    a, #NoIRQFound, Init4
        mov     DPTR, #IRQNotFound
        lcall   SendStr
        jb      SendingStr, $
        sjmp    begin
        sjmp    $                       ; halt

Init4:
FoundCard:
; ---------Got to here, so card probed OK
; Display Mac Address

        mov     DPTR, #MacAddress
```

```
        lcall   SendStr
        mov     DPTR, #SAPROM           ; Display Mac Address
        mov     R1, #6                  ; 6 bytes to display

Init7:  movx    a, @DPTR                ; Get byte
        acall   DisplayNum              ; Display it
        inc     DPTR                    ; point to next byte

        mov     a, #'-'
        acall   SendChar
        djnz    r1, Init7

        mov     DPTR, #EndMacAddr
        lcall   SendStr

loop:   lcall   NS8390_Init             ; Init card

Run0:   jnb     EI_Txing, Run1          ; if not Txing, jump
Run1:   jnb     EI_Rxd, Run0            ; if not received, jump
        acall   ProcessPacket           ; try and make sense of it
        lcall   EIGetPacket             ; get next packet?
        Cpl     p1.7
lop4:   lcall   turn_off

        mov r7,#0ffh
lop1:   djnz r7,lop1
        mov r7,#0ffh
lop2:   djnz r7,lop2
        mov r7,#0ffh
```

```
lop3:   djnz  r7,lop3

        mov     r1,#255
lop5:   mov     dptr,#Buffer
        clr     a
        movx    @dptr,a
        inc     dptr
        movx    @dptr,a
        inc     Buffer
        djnz    r1,lop5

        sjmp    loop                    ; let the interrupt do the work!

;---------------------------------------------------------------
;  Display Num - Display number currently in Acc - Displayed in Hex
;---------------------------------------------------------------
DisplayNum:  push    dph
             push    dpl
             push    acc
             push    acc

             mov     DPTR, #HexStr              ; point to Hex Str
             swap    a                          ; swap nibbles in acc
             anl     a, #15                     ; mask off upper bits
             movc    a, @a+DPTR                 ; get the Hex Code
             acall   SendChar

             pop     acc
             anl     a, #15
             movc    a, @a+dptr
```

```
                acall   SendChar

                pop     acc
                pop     dpl
                pop     dph
                ret

; Process Packet
; ---------------------------------------------------------------
ProcessPacket:  mov     DPTR, #EtypeLen     ; point to Type/Len field
                movx    a, @DPTR            ; get byte
                cjne    a, #8, PP99         ; if not type 0806, exit
                inc     DPTR
                movx    a, @DPTR

                cjne    a, #06, PP02        ; if not RARP packet, then exit

                acall   ProcessRARP         ; its a RARP, someone wants to talk
        to us
                sjmp    PP99
PP02:           cjne    a, #00, PP99        ; if not IP packet, jump
                acall   ProcessDoD
                sjmp    PP99
PP99:           ret

;Shiftbytes
; R1 and DPTR point to the correct buffer locations.  R2 contains number
; of bytes to be shifted.  Bytes are shifted from @DPTR to @R1
; ---------------------------------------------------------------
ShiftBytes:     movx    a, @DPTR
```

```
                movx    @r1, a
                inc     DPTR
                inc     r1
                djnz    r2, ShiftBytes
                ret

; SendMac
; puts out the mac address of the card to the array pointed to by R1
;=============================================================================SendM
ac:     push    dpl             ; save state of DPTR
                push    dph
                mov     DPTR, #SAPROM       ; point to MAC Address
                mov     r2, #6              ; Size bytes in MAC Address
SM1:            movx    a, @DPTR            ; get byte of MAC Address
                movx    @r1, a              ; store it
                inc     DPTR                ; point to next location
                inc     r1
                djnz    r2, SM1             ; repeat until all bytes transferred
                pop     dph                 ; get saved state back
                pop     dpl
                ret

; SendIP
; Puts our IP Address onto the array pointed to by R1
;-----------------------------------------------------------------------
SendIP:         mov     a, #IPAddr1
                movx    @r1, a
                inc     r1
                mov     a, #IPAddr2
                movx    @r1, a
```

```
        inc     r1
        mov     a, #IPAddr3
        movx    @r1, a
        Inc     r1
        mov     a, #IPAddr4
        movx    @r1, a
        inc     r1
        ret

; ProcessRARP
; Process an ARP request.  If it is our IP, then send a reply, else drop the
; packet
;-----------------------------------------------------------------------
ProcessRARP:  mov     a, dpl
        add     a, #25                  ; point to Target address
        mov     dpl, a
        movx    a, @DPTR                ; get high Byte
        cjne    a, #IPAddr1, PP99       ; if not required IP addr, then exit
        inc     DPTR
        movx    a, @DPTR
        cjne    a, #IPAddr2, PP99
        inc     DPTR
        movx    a, @DPTR
        cjne    a, #IPAddr3, PP99
        inc     DPTR
        movx    a, @DPTR
        cjne    a, #IPAddr4, PP99

; OK, someone has RARPd us, best send a reply
        clr     EA                      ; turn off interrupts so can play with P2
```

```
        mov     p2, #0C7H               ; XMitBfr   ; set up for move ; MSB of TXBuffer
        mov     dph, #0C0H              ; Set up second pointer; MSB of Header

; Create Destination address
        mov     R1, #00                 ; point to Destination Address
        mov     dpl, #10            ; point to Sender address
        mov     R2, #6                  ; six bytes to shift
        acall   ShiftBytes

; Add our MAC Address
        acall   SendMac

; now copy out the standard bits
        mov     dpl, #16            ; point to type/len field
        mov     r2, #9
        acall   shiftBytes
        mov     a, #2                   ; opcode byte 2
        movx    @r1, a
        inc     r1

; push out our Mac Address again
        acall   SendMac                 ; say our source address
        acall   SendIP                  ; with this IP
        mov     dpl, #26            ; point to Senders HW Address
        mov     r2, #10                 ; ten bytes
        acall   ShiftBytes              ; copy it across
; All right, finished packet
        setb    ea                      ; interrupts back on
        mov     R3, #00
        mov     R4, #40h
```

```
        mov     DPTR, #TXBuffer
        lcall   SendPacket              ; Send out the packet!
        ret

; Given the start of the packet pointed to by DPTR, calculates the checksum
; of the packet
; DPTR = start of packet for checksum to be computed
; R4/R5 = Length
;-------------------------------------------------------------------------
CalcCS: clr     a
        mov     r0, a                   ; clear out working registers
        mov     r1, a

        mov     a, r5                   ; half R5
        rr      a
        mov     r5, a

        inc     r4              ; Adjust for DJNZ
        clr     c

CS1:    movx    a, @DPTR        ; get first byte
        addc    a, r0           ; add to first byte
        mov     r0, a           ; store result
        inc     DPTR            ; get next byte

        movx    a, @DPTR
        addc    a, r1           ; add to r1
        mov     r1, a           ; store it
        inc     DPTR            ; point to next byte
        djnz    r5, CS1         ; repeat until finished
```

```
        djnz    r4, cs1

        cpl     a               ; complement result and exit
        mov     dpl, a
        mov     a, r0
        cpl     a
        mov     dph, a
        ret

; ProcessDoD
; Look into the packet, and see if it is an Echo request, if not exit
; ---------------------------------------------------------------------
ProcessDoD:     mov     dpl, #1bh       ; point to IP: Protocol
                movx    a, @DPTR        ; get byte
                cjne    a, #01, PDoD99  ; if not ICMP, exit

; Ok, looking at ICMP protocol, check for echo byte
        mov     dpl, #26h       ; point to type value
        movx    a, @DPTR
        cjne    a, #8, error

; Ok, looking at an ICMP Echo packet, build up and send back
        mov     dpl, #4         ; point to start of packet
        clr     EA              ; stop interrupts
        mov     p2, #0C7H       ; point to Transmit buffer  (XMitBfr)
        mov     r1, #0          ; (0)
        mov     r2, #050h       ; shift 255 bytes
        acall   ShiftBytes

; now tx buffer looks the same as the rx buffer.  Make required changes
```

```
mov     dpl, #0ah               ; point to source mac, and fill in
mov     r1, #0                  ; (0)
mov     r2, #06
acall   ShiftBytes
acall   SendMac                 ; output our mac address

mov     dpl, #1eh               ; point to Source address
mov     r1, #1eh        ; point to dest addr
mov     r2, #4
acall   ShiftBytes              ; copy

mov     r1, #1ah        ; point to source IP Addr
acall   SendIP                  ; put in our IP Address

clr     a
mov     r1, #24h        ; zero out Echo Checksum
movx    @r1, a
inc     r1
movx    @r1, a

mov     r1, #22h        ; change Type to 0 for echo reply
clr     a
movx    @r1, a

mov     DPTR, #TXBuffer         ; (TXBuffer)
mov     dpl, #22h               ; point to ICMP Header
mov     R4, #00
mov     r5, #28h        ; this many bytes
acall   CalcCS                  ; calculate checksum
mov     a, dpl
```

```
        mov     r2, a
        mov     a, dph

        mov     DPTR, #TXBuffer         ; (TXBuffer)
        mov     dpl, #24h               ; point to checksum field, show
        movx    @DPTR, a
        inc     DPTR
        mov     a, r2
        movx    @DPTR, a

       setb     EA                      ; enable interrupts

        mov     r3, #00h
        mov     r4, #4ah
        mov     DPTR, #TXBuffer                 ; (TXBuffer)

        lcall   SendPacket

error:  ret

check_snmp1: jmp check_snmp

PDoD99: cjne    a,#17,check_snmp   ; stop when get UDP
          mov   dpl, #2eh
          movx  a, @dptr
          cjne  a, #31h, check1
          mov   dpl, #35h
          movx  a,@dptr
          cjne  a, #4fh, check_snmp
          clr   p1.0
```

```
            lcall   process_snmp
            ret

check1:  cjne   a, #30h, check_snmp
            mov   dpl, #35h
            movx  a,@dptr
            cjne   a, #43h, check_snmp
            setb   p1.0
            lcall   process_snmp
            ret

check_snmp: nop
            ret

;=========================== process snmp ===========================

process_snmp:     mov    dpl, #4              ; point to start of packet
                  clr     EA                  ; stop interrupts
                  mov    p2, #0C7H            ; point to Transmit buffer  (XMitBfr)
                  mov    r1, #0               ; (0)
                  mov    r2, #60h             ; shift 256 bytes
                  acall    ShiftBytes

; now tx buffer looks the same as the rx buffer.  Make required changes
      mov    dpl, #0ah            ; prepare destination mac
      mov    r1, #0               ;
      mov    r2, #06
      acall   ShiftBytes
      acall   SendMac             ; followed with our mac address
      mov    dpl, #1eh              ; point to Source address
```

```
    mov    r1, #1eh              ; point to dest addr
    mov    r2, #4
    acall  ShiftBytes            ; copy

    mov    r1, #1Ah              ; point to source IP Addr
    acall  SendIP                ; put in our IP Address

;=============modify for snmp get-response ====================
    clr  a                  ; zero checksum
    mov  r1,#18h
    movx  @r1,a
    inc  r1
    movx  @r1,a

    mov   r1,#11h
    mov   a,#70
    movx   @r1,a

    mov   dptr,#0c70eh
    mov   r4,#0
    mov   r5,#20
    call  calccs

    mov   r1,#18h
    mov   a,dph
    movx  @r1,a
    inc   r1
    mov   a,dpl
    movx  @r1,a
    mov   r1,#37h
```

```
mov    a,#0a2h
movx   @r1,a              ; get-response

inc    r1                 ; a230

mov    a,#1bh
movx   @r1,a
mov    r1,#2bh
mov    a,#28h
movx   @r1,a

mov    r1,#27h
mov    a,#50              ; test with 50 bytes
movx   @r1,a

inc    r1

clr a
movx   @r1,a
inc    r1
movx   @r1,a              ; zero checksum

mov    r1,#22h            ; swap port location
movx   a,@r1
mov    r2,a
mov    r1,#24h
movx   a,@r1
xch    a,r2
movx   @r1,a
mov    a,r2
```

```
mov     r1,#22h
movx    @r1,a

mov     r1,#23h
movx    a,@r1
mov     r2,a
mov     r1,#25h
movx    a,@r1
xch     a,r2
movx    @r1,a
mov     a,r2
mov     r1,#23h
movx    @r1,a

mov     r1,#51h
mov     a,#43H          ; test with data
movx    @r1,a
inc     r1
mov     a,#1
movx    @r1,a
inc     r1
mov     r0,#80h
inc     @r0
mov     a,@r0
movx    @r1,a

setb    EA              ; enable interrupts
mov     r3, #00h
mov     r4, #84
mov     DPTR, #TXBuffer         ; (TXBuffer)
```

```
        lcall   SendPacket
        ret

HexStr:         db  '0123456789ABCDEF'          ; used for displaying hex
HelloStr:       db  'C52EVB Ethernet V1.0',CR,LF,'NEx000 ISA Card Program', CR, LF
                db  LF, 'Probing for card...$'
NotNeStr:       db  'Sorry, no NEx000 card found!$'
NoResetAckS:    db  'No reset Ack!$'
IRQNotFound:    db  'No IRQ response obtained from Card', CR, LF, '$'
MacAddress:     db  'Card Found', CR, LF,
                db  'MAC Address of card : $'
EndMacAddr:     db  8, ' ', CR, LF, '$'
UnknownInt:     db  Cr, LF, LF, ' UNKNOWN INT $'
UnknownError:   db  CR, LF, 'Unknown XMIT Error$'
TransmitErr:    db  CR, LF, 'TRANSMIT ERROR : $'
ENTSR_ABTStr:   db  ' Excess-Collisions ', CR, LF, '$'
ENTSR_NDStr:    db  ' Non-Deferral ', CR, LF, '$'
ENTSR_CRSStr:   db  ' Lost-Carrier ', CR, LF, '$'
ENTSR_FUStr:    db  ' FIFO-Underrun ', CR, LF, '$'
ENTSR_CDHStr:   db  ' Lost-Heartbeat ', CR, LF, '$'

$include(SRSND552.ASM)      ; Serial output routines
$include(8390.INC)          ; 8290 Ethernet chipset driver
end
```

Sourcecode Delphi

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, IdBaseComponent, IdComponent, IdRawBase, IdRawClient,
  IdIcmpClient, Sockets, ExtCtrls, ComCtrls, UThreadSniffer, IdUDPClient,
  IdUDPBase, IdUDPServer, IdDNSResolver, IdTCPConnection, IdTCPClient,
  IdEcho, IdTCPServer, IdEchoServer, IdDiscardServer, Menus, ActnList,
  Buttons, IdSNMP, IdHTTP, IdEchoUDP, MPlayer, OleCtrls, SHDocVw, HTTPApp,
  IdMessageClient, IdPOP3;


type
  TForm1 = class(TForm)
    Memo1: TMemo;
    btnexit: TButton;
    Timer1: TTimer;
    Edit1: TEdit;
    udp: TIdUDPClient;
    StatusBar1: TStatusBar;
    Timer2: TTimer;
    GroupBox1: TGroupBox;
    btnOn: TButton;
    btnOff: TButton;
    PopupMenu1: TPopupMenu;
    GroupBox3: TGroupBox;
    btnstart: TButton;
    btnstop: TButton;
```

```
MainMenu1: TMainMenu;
Exit1: TMenuItem;
OnCircuit1: TMenuItem;
Ping1: TMenuItem;
OffCircuit1: TMenuItem;
StopPing1: TMenuItem;
File1: TMenuItem;
Method1: TMenuItem;
OnCircuit2: TMenuItem;
OffCircuit2: TMenuItem;
Exit2: TMenuItem;
Ping2: TMenuItem;
Start1: TMenuItem;
Help1: TMenuItem;
About1: TMenuItem;
icmp: TIdIcmpClient;
Label1: TLabel;
Shape1: TShape;
CheckBox1: TCheckBox;
Memo2: TMemo;
Edit2: TEdit;
Label2: TLabel;
stop1: TMenuItem;
CheckBox2: TCheckBox;
procedure btnexitClick(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Timer2Timer(Sender: TObject);
procedure btnOnClick(Sender: TObject);
procedure btnOffClick(Sender: TObject);
```

```
    procedure btnstartClick(Sender: TObject);
    procedure btnstopClick(Sender: TObject);
    procedure OnCircuit1Click(Sender: TObject);
    procedure Ping1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure OffCircuit1Click(Sender: TObject);
    procedure StopPing1Click(Sender: TObject);
    procedure Exit2Click(Sender: TObject);
    procedure OnCircuit2Click(Sender: TObject);
    procedure OffCircuit2Click(Sender: TObject);
    procedure Start1Click(Sender: TObject);
    procedure icmpReply(ASender: TComponent;
      const AReplyStatus: TReplyStatus);
    procedure About1Click(Sender: TObject);
    procedure Edit1Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure sffasd1Click(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }
    ListThread: TSnifferThread;
  end;

var
  Form1: TForm1;
  time : Tdatetime;
implementation
```

```
{$R *.dfm}

procedure TForm1.btnOnClick(Sender: TObject);
var ip,s : string;
    port : integer;
begin
 udp.Host := Edit1.Text;
 statusbar1.Panels[1].Text := ' Connect';
 if edit1.Text = '' then  begin
 showmessage('You must insert IP Address');
 btnOn.Enabled := false;
 oncircuit1.Enabled := false;
 oncircuit2.Enabled := false;  end;
 if not(edit1.text = '') and (edit1.Text = '161.246.13.252')  then begin
 memo2.Clear;
 btnOff.Enabled := true;
 offcircuit1.Enabled := true;
 offcircuit2.Enabled := true;
 btnOn.Enabled := false;
 oncircuit1.Enabled := false;
 oncircuit2.Enabled := false;
 if (checkbox2.Checked = false) then begin
 udp.Send('1 Fire Open)');
 shape1.Brush.Color := clRed;
 sleep(500);
 udp.Send('1 Fire Open)');
 memo2.Lines.Add('No Data Return'); end;

 if (checkbox2.Checked = true) then begin
 udp.Send('1 Fire Open)');
```

```
 s := udp.ReceiveString(ip,port);
 shape1.Brush.Color := clRed;
 udp.Send('1 Fire Open)');
 sleep(1000);
 s := udp.ReceiveString(ip,port);
 sleep(200);
 memo2.Lines.Add(format('Port = %d',[port]));
 sleep(200);
 memo2.Lines.Add(format('IP    = %s',[ip]));
 sleep(200);
 memo2.Lines.Add('');
 sleep(200);
 memo2.Lines.Add('Recieve String = ');
 sleep(200);
 memo2.Lines.Add(format('.....%s.....',[s])); end;
 edit2.Text := udp.LocalName;  end;
 if not(edit1.Text = '161.246.13.252') then
 showmessage('Please Insert IP 161.246.13.252 to control device');
end;

procedure TForm1.btnOffClick(Sender: TObject);
var ip,a : string;
    port : integer;
begin
 memo2.Clear;
 statusbar1.Panels[1].Text := ' Disconnec';
 btnOn.Enabled := true;
 oncircuit1.Enabled := true;
 oncircuit2.Enabled := true;
 btnOff.Enabled := false;
```

```
 offcircuit1.Enabled := false;
 offcircuit2.Enabled := false;
 udp.Host := Edit1.Text;
 udp.Send('0 Fire Close)');
 shape1.Brush.Color := clWhite;
 if (checkbox2.Checked = false) then
 memo2.Lines.Add('No Data Return');

 if (checkbox2.Checked = true) then begin
 udp.Send('0 Fire Close)');
 sleep(500);
 udp.Send('0 Fire Close)');
 sleep(1000);
 a := udp.ReceiveString(ip,port);
 sleep(200);
 a := udp.ReceiveString(ip,port);
 sleep(500);
 memo2.Lines.Add(format('Port = %d',[port]));
 sleep(200);
 memo2.Lines.Add(format('IP    = %s',[ip]));
 sleep(200);
 memo2.Lines.Add('');
 sleep(200);
 memo2.Lines.Add('Recieve String = ');
 sleep(200);
 memo2.Lines.Add(format('.....%s.....',[a])); end;
 edit2.Text := udp.LocalName;
end;

procedure TForm1.btnstartClick(Sender: TObject);
```

```
begin
 icmp.Host := edit1.Text;
 if edit1.Text = '' then begin
 showmessage('You must insert IP Address');
 btnstart.Enabled := false;
 start1.Enabled := false;
 ping1.Enabled := false;  end;
 if not(edit1.text = '') then  begin
 edit2.Text := icmp.LocalName;
 if checkbox1.Checked = true then begin
 statusbar1.Panels[1].Text := 'Connect';
 btnstart.Enabled := false;
 start1.Enabled := false;
 ping1.Enabled := false;
 btnstop.Enabled := true;
 stop1.Enabled := true;
 stopping1.Enabled := true;
 Timer1.Enabled := true;  end;

 if checkbox1.Checked = false then begin
 statusbar1.Panels[1].Text := 'Connect';
 btnstart.Enabled := false;
 start1.Enabled := false;
 ping1.Enabled := false;
 btnstop.Enabled := true;
 stop1.Enabled := true;
 stopping1.Enabled := true;
 timer1.Enabled := false;
 icmp.Ping();
 sleep(1000);
```

```
 icmp.Ping();
 sleep(1000);
 icmp.Ping();
 sleep(1000);
 memo1.Lines.Add('');
 btnstart.Enabled := true;
 start1.Enabled := true;
 ping1.Enabled := true;
 btnstop.Enabled := false;
 stop1.Enabled := false;
 stopping1.Enabled := false;
 statusbar1.Panels[1].Text := 'Disconnect';  end;
 end;
end;

procedure TForm1.btnstopClick(Sender: TObject);
begin
 Memo1.Clear;
 Timer1.Enabled := false;
 btnstart.Enabled := true;
 start1.Enabled := true;
 ping1.Enabled := true;
 btnstop.Enabled := false;
 stop1.Enabled := false;
 stopping1.Enabled := false;
 statusbar1.Panels[1].Text := 'Disconnec';
end;

procedure TForm1.btnexitClick(Sender: TObject);
begin
```

```
 if (btnOff.Enabled = true) or (btnstop.Enabled = true) or
    (( btnOff.Enabled = true) and (btnstop.Enabled = true)) then
    showmessage('You must stop all process before exit');
 if ( btnOff.Enabled = false) and (btnstop.Enabled = false) then
    Application.Terminate;
end;

procedure TForm1.OnCircuit1Click(Sender: TObject);
var ip,s : string;
    port : integer;
begin
 udp.Host := Edit1.Text;
 statusbar1.Panels[1].Text := ' Connect';
 if edit1.Text = '' then begin
 showmessage('You must insert IP Address');
 btnOn.Enabled := false;
 oncircuit1.Enabled := false;
 oncircuit2.Enabled := false; end;
 if not(edit1.text = '') and (edit1.Text = '161.246.13.252') then begin
 memo2.Clear;
 btnOff.Enabled := true;
 offcircuit1.Enabled := true;
 offcircuit2.Enabled := true;
 btnOn.Enabled := false;
 oncircuit1.Enabled := false;
 oncircuit2.Enabled := false;
 if (checkbox2.Checked = false) then begin
 udp.Send('1 Fire Open)');
 shape1.Brush.Color := clRed;
 sleep(500);
```

```
udp.Send('1 Fire Open)');
s := udp.ReceiveString(ip,port);
memo2.Lines.Add('No Data Return'); end;

if (checkbox2.Checked = true) then begin
udp.Send('1 Fire Open)');
s := udp.ReceiveString(ip,port);
shape1.Brush.Color := clRed;
udp.Send('1 Fire Open)');
sleep(1000);
s := udp.ReceiveString(ip,port);
sleep(200);
memo2.Lines.Add(format('Port = %d',[port]));
sleep(200);
memo2.Lines.Add(format('IP   = %s',[ip]));
sleep(200);
memo2.Lines.Add('');
sleep(200);
memo2.Lines.Add('Recieve String = ');
sleep(200);
memo2.Lines.Add(format('.....%s.....',[s])); end;
edit2.Text := udp.LocalName;  end;
if not(edit1.Text = '161.246.13.252') then
showmessage('Please Insert IP 161.246.13.252 to control device');
end;

procedure TForm1.Ping1Click(Sender: TObject);
begin
icmp.Host := edit1.Text;
if edit1.Text = '' then begin
```

```
showmessage('You must insert IP Address');
btnstart.Enabled := false;
start1.Enabled := false;
ping1.Enabled := false;  end;
if not(edit1.text = '') then  begin
edit2.Text := icmp.LocalName;
if checkbox1.Checked = true then begin
statusbar1.Panels[1].Text := 'Connect';
btnstart.Enabled := false;
start1.Enabled := false;
ping1.Enabled := false;
btnstop.Enabled := true;
stop1.Enabled := true;
stopping1.Enabled := true;
Timer1.Enabled := true;  end;

if checkbox1.Checked = false then begin
statusbar1.Panels[1].Text := 'Connect';
btnstart.Enabled := false;
start1.Enabled := false;
ping1.Enabled := false;
btnstop.Enabled := true;
stop1.Enabled := true;
stopping1.Enabled := true;
timer1.Enabled := false;
icmp.Ping();
sleep(1000);
icmp.Ping();
sleep(1000);
icmp.Ping();
```

```
 sleep(1000);
 memo1.Lines.Add('');
 btnstart.Enabled := true;
 start1.Enabled := true;
 ping1.Enabled := true;
 btnstop.Enabled := false;
 stop1.Enabled := false;
 stopping1.Enabled := false;
 statusbar1.Panels[1].Text := 'Disconnect';  end;
 end;
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
 if (btnOff.Enabled = true) or (btnstop.Enabled = true) or
   (( btnOff.Enabled = true) and (btnstop.Enabled = true)) then
   showmessage('You must stop all process before exit');
 if (btnOff.Enabled = false) and (btnstop.Enabled = false) then
   Application.Terminate;
end;

procedure TForm1.OffCircuit1Click(Sender: TObject);
var ip,a : string;
    port : integer;
begin
 memo2.Clear;
 statusbar1.Panels[1].Text := ' Disconnec';
 btnOn.Enabled := true;
 oncircuit1.Enabled := true;
 oncircuit2.Enabled := true;
```

```
btnOff.Enabled := false;
offcircuit1.Enabled := false;
offcircuit2.Enabled := false;
udp.Host := Edit1.Text;
udp.Send('0 Fire Close)');
shape1.Brush.Color := clWhite;
a := udp.ReceiveString(ip,port);
if (checkbox2.Checked = false) then
memo2.Lines.Add('No Data Return');

if (checkbox2.Checked = true) then begin
udp.Send('0 Fire Close)');
sleep(500);
udp.Send('0 Fire Close)');
sleep(1000);
a := udp.ReceiveString(ip,port);
sleep(200);
a := udp.ReceiveString(ip,port);
sleep(500);
memo2.Lines.Add(format('Port = %d',[port]));
sleep(200);
memo2.Lines.Add(format('IP    = %s',[ip]));
sleep(200);
memo2.Lines.Add('');
sleep(200);
memo2.Lines.Add('Recieve String = ');
sleep(200);
memo2.Lines.Add(format('.....%s.....',[a])); end;
edit2.Text := udp.LocalName;
end;
```

```
procedure TForm1.StopPing1Click(Sender: TObject);
begin
 Memo1.Clear;
 Timer1.Enabled := false;
 btnstart.Enabled := true;
 start1.Enabled := true;
 ping1.Enabled := true;
 btnstop.Enabled := false;
 stop1.Enabled := false;
 stopping1.Enabled := false;
 statusbar1.Panels[1].Text := 'Disconnec';
end;

procedure TForm1.Exit2Click(Sender: TObject);
begin
 if (btnOff.Enabled = true) or (btnstop.Enabled = true) or
   (( btnOff.Enabled = true) and (btnstop.Enabled = true)) then
   showmessage('You must stop all process before exit');
 if ( btnOff.Enabled = false) and (btnstop.Enabled = false) then
   Application.Terminate;
end;

procedure TForm1.OnCircuit2Click(Sender: TObject);
var ip,s : string;
   port : integer;
begin
 udp.Host := Edit1.Text;
 statusbar1.Panels[1].Text := ' Connect';
 if edit1.Text = '' then begin
 showmessage('You must insert IP Address');
```

```
btnOn.Enabled := false;
oncircuit1.Enabled := false;
oncircuit2.Enabled := false;   end;
if not(edit1.text = '') and (edit1.Text = '161.246.13.252') then   begin
memo2.Clear;
btnOff.Enabled := true;
offcircuit1.Enabled := true;
offcircuit2.Enabled := true;
btnOn.Enabled := false;
oncircuit1.Enabled := false;
oncircuit2.Enabled := false;
if (checkbox2.Checked = false) then begin
udp.Send('1 Fire Open)');
shape1.Brush.Color := clRed;
sleep(500);
udp.Send('1 Fire Open)');
s := udp.ReceiveString(ip,port);
memo2.Lines.Add('No Data Return'); end;

if (checkbox2.Checked = true) then begin
udp.Send('1 Fire Open)');
s := udp.ReceiveString(ip,port);
shape1.Brush.Color := clRed;
udp.Send('1 Fire Open)');
sleep(1000);
s := udp.ReceiveString(ip,port);
sleep(200);
memo2.Lines.Add(format('Port = %d',[port]));
sleep(200);
memo2.Lines.Add(format('IP    = %s',[ip]));
```

```
 sleep(200);
 memo2.Lines.Add('');
 sleep(200);
 memo2.Lines.Add('Recieve String = ');
 sleep(200);
 memo2.Lines.Add(format('.....%s.....',[s])); end;
 edit2.Text := udp.LocalName;  end;
 if not(edit1.Text = '161.246.13.252') then
 showmessage('Please Insert IP 161.246.13.252 to control device');
end;

procedure TForm1.OffCircuit2Click(Sender: TObject);
var ip,a : string;
    port : integer;
begin
 memo2.Clear;
 statusbar1.Panels[1].Text := ' Disconnec';
 btnOn.Enabled := true;
 oncircuit1.Enabled := true;
 oncircuit2.Enabled := true;
 btnOff.Enabled := false;
 offcircuit1.Enabled := false;
 offcircuit2.Enabled := false;
 udp.Host := Edit1.Text;
 udp.Send('0 Fire Close)');
 shape1.Brush.Color := clWhite;
 a := udp.ReceiveString(ip,port);
 if (checkbox2.Checked = false) then
 memo2.Lines.Add('No Data Return');
 if (checkbox2.Checked = true) then begin
```

```
 udp.Send('0 Fire Close)');
 sleep(500);
 udp.Send('0 Fire Close)');
 sleep(1000);
 a := udp.ReceiveString(ip,port);
 sleep(200);
 a := udp.ReceiveString(ip,port);
 sleep(500);
 memo2.Lines.Add(format('Port = %d',[port]));
 sleep(200);
 memo2.Lines.Add(format('IP    = %s',[ip]));
 sleep(200);
 memo2.Lines.Add('');
 sleep(200);
 memo2.Lines.Add('Recieve String = ');
 sleep(200);
 memo2.Lines.Add(format('.....%s.....',[a])); end;
 edit2.Text := udp.LocalName;
end;

procedure TForm1.Start1Click(Sender: TObject);
begin
 icmp.Host := edit1.Text;
 if edit1.Text = '' then begin
 showmessage('You must insert IP Address');
 btnstart.Enabled := false;
 start1.Enabled := false;
 ping1.Enabled := false;  end;
 if not(edit1.text = '') then
 begin
```

```
edit2.Text := icmp.LocalName;
if checkbox1.Checked = true then begin
statusbar1.Panels[1].Text := 'Connect';
btnstart.Enabled := false;
start1.Enabled := false;
ping1.Enabled := false;
btnstop.Enabled := true;
stop1.Enabled := true;
stopping1.Enabled := true;
Timer1.Enabled := true;
end;

if checkbox1.Checked = false then begin
statusbar1.Panels[1].Text := 'Connect';
btnstart.Enabled := false;
start1.Enabled := false;
ping1.Enabled := false;
btnstop.Enabled := true;
stop1.Enabled := true;
stopping1.Enabled := true;
timer1.Enabled := false;
icmp.Ping();
sleep(1000);
icmp.Ping();
sleep(1000);
icmp.Ping();
sleep(1000);
memo1.Lines.Add('');
btnstart.Enabled := true;
start1.Enabled := true;
```

```
 ping1.Enabled := true;
 btnstop.Enabled := false;
 stop1.Enabled := false;
 stopping1.Enabled := false;
 statusbar1.Panels[1].Text := 'Disconnect';  end;
 end;
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
 icmp.Ping('');
end;

procedure TForm1.Timer2Timer(Sender: TObject);
begin
 statusbar1.Panels[3].Text := FormatDateTime('hh:nn:ss',now - time);
 statusbar1.Panels[5].Text := FormatDateTime('hh:nn:ss',now);
 statusbar1.Panels[7].Text := FormatDateTime('d dddd mmmm yyyy',now);
end;

procedure TForm1.icmpReply(ASender: TComponent;
  const AReplyStatus: TReplyStatus);
begin
 if AReplyStatus.ReplyStatusType = rsecho then
    Memo1.Lines.Add('Reply from ' + (AReplyStatus.FromIpAddress) + ':  ' + 'bytes='
             + inttostr(AReplyStatus.BytesReceived) + '  time='
             + inttostr(AReplyStatus.MsRoundTripTime) + 'ms '+ '  TTL='
             + inttostr(AReplyStatus.TimeToLive) + '  Sequence='
             + inttostr(AReplyStatus.SequenceId));
 if AReplyStatus.ReplyStatusType = rstimeout then
    Memo1.Lines.Add('Request timed out');
```

```
 if AReplyStatus.ReplyStatusType = rserror then
    Memo1.Lines.Add('Error');
 if AReplyStatus.ReplyStatusType = rsErrorUnreachable then
    Memo1.Lines.Add('address for ICMP message not available');
 if AReplyStatus.ReplyStatusType = rsErrorTTLExceeded then
    Memo1.Lines.Add('TTL exceeded for an ICMP response');
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 Memo1.Clear;
 statusbar1.Panels[1].Text := 'Disconnec';
 time := strtotime(formatDateTime('hh:nn:ss',now));
 memo2.Clear;
 memo2.Lines.Add('Port = ');
 memo2.Lines.Add('IP   = ');
 memo2.Lines.Add('');
 memo2.Lines.Add('Recieve String = ');
end;

procedure TForm1.About1Click(Sender: TObject);
begin
 showmessage('Control Device On Network (CDON)' + chr(13) + 'version 3.0');
end;

procedure TForm1.Edit1Click(Sender: TObject);
begin
 if (btnOff.Enabled = true) or (btnstop.Enabled = true) or
    (( btnOff.Enabled = true) and (btnstop.Enabled = true)) then     begin
    showmessage('You must stop all process before change IP Address');
```

```
 edit1.Enabled := false;     end;

 if (btnOff.Enabled = false) or (btnstop.Enabled = false) or
    (( btnOff.Enabled = false) and (btnstop.Enabled = false)) then
    edit1.Enabled := true;
end;

procedure TForm1.Edit1Change(Sender: TObject);
begin
 if (edit1.Modified = true) and not(edit1.text = '') then
 begin
 btnOn.Enabled := true;
 oncircuit1.Enabled := true;
 oncircuit2.Enabled := true;
 btnstart.Enabled := true;
 start1.Enabled := true;
 ping1.Enabled := true;
 end;
end;

procedure TForm1.sffasd1Click(Sender: TObject);
begin
 Memo1.Clear;
 Timer1.Enabled := false;
 btnstart.Enabled := true;
 start1.Enabled := true;
 ping1.Enabled := true;
 btnstop.Enabled := false;
 stop1.Enabled := false;
 stopping1.Enabled := false;
```

```
 statusbar1.Panels[1].Text := 'Disconnec';
end;


end.
```