การสื่อสารทางเสียงผ่านเครือข่ายไอพี

VOICE OVER INTERNET PROTOCAL

โดย

ว่าที่ ร.ต. ธนพงศ์ ไชยชนะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

การสื่อสารทางเสียงผ่านเครือข่ายไอพี

VOICE OVER INTERNET PROTOCAL

โดย

ว่าที่ ร.ต. ธนพงค์ ไชยชนะ รหัส 46015178

อาจารย์ที่ปรึกษา

รศ.ดร. มนัส สังวรศิลป์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การสื่อสารทางเสียงผ่านเครือข่ายไอพี (Voice Over Internet Protocol)

ผู้จัดทำ ว่าที่ ร.ต. ธนพงศ์ ไชยชนะ รหัส 46015178

ลงชื่อ ..... อาจารย์ที่ปรึกษา

(รศ.ดร. มนัส สังวรศิลป์)

การสื่อสารทางเสียงผ่านเครือข่ายไอพี

ว่าที่ร้อยตรี ธนพงค์ ไชยชนะ รหัส 46015178
รศ.ดร. มนัส สังวรศิลป์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

โครงงานนี้เป็นการศึกษาการสื่อสารทางเสียงผ่านเครือข่ายอินเตอร์เน็ต (Internet Protocol) ได้มีการพัฒนาโปรแกรมสำหรับการสื่อสาร ให้มีความสามารถสื่อสารกันด้วยเสียง โดยใช้ฟังก์ชัน Winsock (Window Socket) และพัฒนาโปรแกรมโดยใช้โปรแกรม Microsoft Visual C++ Version 6.0 เป็นการเขียนโปรแกรมควบคุมการทำงานของการ์ดแลน (Lan Card) ให้มีการส่งข้อมูลติดต่อสื่อสารกันด้วยโปรโตคอลทีซีพีไอพี (TCP/IP Protocol) ระหว่างเครื่องคอมพิวเตอร์ทั้ง 2 ฝ่ายได้ โดยแบ่งออกเป็น ฝ่าย Server และ ฝ่าย Client

Voice Over Internet Protocol (VoIP)

Mr. Thanapong Chaichana ID. 46015178

Assoc.Prof.Dr. Manas Sangworasil Advisor

Educational Year 2005

ABSTRACT

This project study of communication voice over internet protocol (VoIP) and developed programs for communication voice chat with Winsock (Windows Socket) on Microsoft Visual C++ Version 6.0. Connection and transfer data on TCP/IP Protocol and project is control Lan Card. Test communication connections with Computer server transfer data swap Computer Client.

กิตติกรรมประกาศ

ขอขอบคุณ

อาจารย์ รศ.ดร. มนัส สังวรศิลป์ (อาจารย์ที่ปรึกษา) และ คุณนพรัตน์ พันธุ์เสนา นักวิจัยของสำนักนักวิจัยและบริการคอมพิวเตอร์ ที่ให้การอุปการะในการให้คำปรึกษาและแนะนำเกี่ยวกับโครงงานนี้ ให้ยืมใช้เครื่องมืออิเล็กทรอนิกส์ในการทดลองและหนังสือต่างต่างที่เกี่ยวข้อง และสั่งสอนให้ความรู้จนสามารถนำมาประยุกต์ใช้งานในการทำโครงงานครั้งนี้ คุณพ่อ คุณแม่ ที่คอยให้ความเป็นห่วง และให้กำลังใจพวกเราเสมอมา ทั้งในด้านการทำงานและการเรียน

สารบัญ

สารบัญตาราง

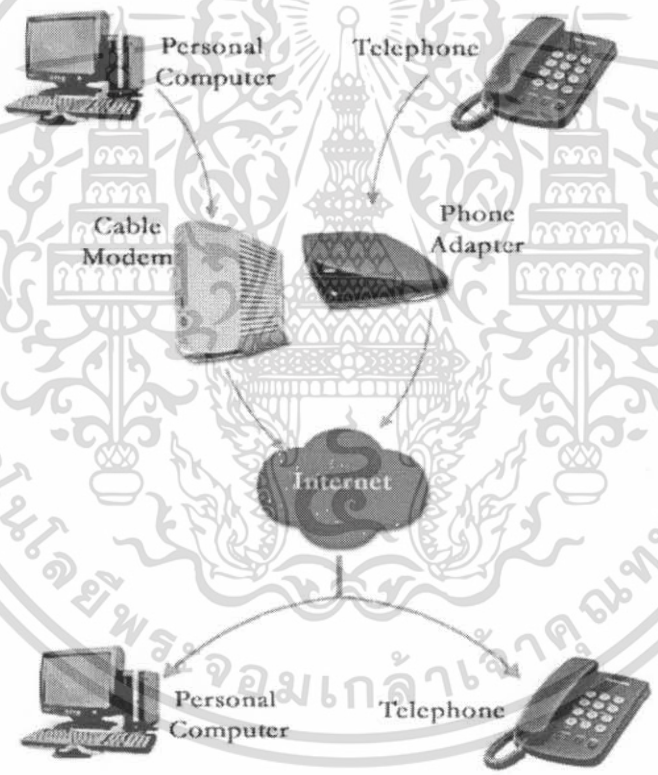
สารบัญรูป

หน้า

บทที่ 1 บทนำ

1.1 รายละเอียดโดยย่อของโครงงาน

โครงงานนี้เป็นการศึกษาการสื่อสารทางเสียงผ่านเครือข่ายไอพีโดยการส่งข้อมูลซึ่งเป็นเสียงไปบนโปรโตคอลทีซีพีไอพี ใช้คอมพิวเตอร์สองเครื่องในการทดสอบคอมพิวเตอร์ทั้งสองเครื่องจะต้องเชื่อมต่ออินเตอร์เน็ตอยู่ด้วยกันทั้งคู่และต้องทราบหมายเลขไอพีแอดเดสของทั้งสองเครื่องจากนั้นจะเริ่มทำการส่งข้อมูลที่เป็นข้อความและเสียงติดต่อกันระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่องแสดงดังรูปที่ 1.1 โดยใช้โปรแกรมที่พัฒนาขึ้นมาจากภาษาซี



รูปที่ 1.1 การติดต่อสื่อสารกันโดยผ่านเครือข่ายอินเตอร์เน็ต

โครงงานนี้ได้แบ่งขั้นตอนการดำเนินงานออกเป็น 2 ระยะ คือ

1. ระยะที่ 1 สร้างรูปแบบการติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่องจากการพัฒนาโปรแกรมโดยใช้ภาษาซีและทดสอบส่งข้อมูลที่เป็นข้อความระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่องที่เชื่อมต่ออินเตอร์เน็ตอยู่

2. ระยะที่ 2 พัฒนารูปแบบการติดต่อสื่อสารจากระยะที่ 1 โดยเพิ่มเติมให้มีความสามารถติดต่อสื่อสารกันด้วยเสียงพูดผ่านเครือข่ายอินเตอร์เน็ตได้แทนการใช้โทรศัพท์

1.2 วัตถุประสงค์ของโครงงาน

โครงงานนี้จัดทำขึ้นโดยมีวัตถุประสงค์เพื่อศึกษาการติดต่อสื่อสารทางเสียงผ่านเครือข่ายอินเตอร์เน็ตโดยใช้โปรโตคอลทีซีพีไอพีมีการพัฒนาโปรแกรมด้วยภาษาซีโดยใช้โปรแกรม Microsoft Visual C++ เวอร์ชั่น 6 แทนการสื่อสารโดยใช้โทรศัพท์ติดต่อกัน

1.3 ขอบเขตของการทำงาน

โครงงานนี้ได้เลือกที่จะศึกษาเทคโนโลยีการติดต่อสื่อสารด้วยการส่งข้อมูลข้อความและเสียงผ่านเครื่อข่ายอินเตอร์เน็ตโดยใช้คอมพิวเตอร์เพราะปัจจุบันนี้ได้มีการใช้งานอินเตอร์เน็ตอย่างกว้างขวางโครงงานได้ใช้งานอุปกรณ์คอมพิวเตอร์ต่าง ๆ ดังนี้ คือ ไมโครโฟน (Microphone) ลำโพง (Speaker) การ์ดเสียง (Sound Card) และการ์ดแลน (LAN Card) โดยมีการพัฒนาโปรแกรมการติดต่อสื่อสารทางเสียงโดยใช้ภาษาซีโดยใช้โปรแกรม Microsoft Visual C++ เวอร์ชั่น 6

1.4 ประโยชน์หรือผลลัพธ์ที่จะได้รับ

สิ่งที่จะได้รับจากการศึกษาการสื่อสารทางเสียงผ่านอินเตอร์เน็ตโปรโตคอล คือ

1. มีความรู้ความเข้าใจการส่งข้อมูลกันระหว่างเครื่องคอมพิวเตอร์ผ่านเครือข่ายอินเตอร์เน็ต
2. มีความรู้ความเข้าใจโปรโตคอล TCP/IP
3. มีความรู้ความเข้าใจโปรแกรมภาษาซีและฟังก์ชั่น Winsock
4. สามารถนำโปรแกรมที่พัฒนาแล้วไปใช้ติดต่อสื่อสารกันแทนการโทรศัพท์ทำให้ประหยัดค่าใช้จ่ายในการติดต่อสื่อสารกันได้
5. สามารถนำโปรแกรมไปพัฒนาต่อได้โดยการส่งข้อมูลที่เป็นภาพเพิ่มเติม

รายงานฉบับนี้ได้อธิบายขั้นตอน วิธีการในการออกแบบ และผลการทดลองของโครงงานการติดต่อสื่อสารทางเสียงผ่านเครือข่ายไอพี โดยจะมีเนื้อหาแบ่งออกเป็นบท ดังต่อไปนี้

บทที่ 2 ทฤษฎี โดยจะกล่าวถึงทฤษฎี และหลักการพื้นฐานของระบบโทรศัพท์ รูปแบบการติดต่อสื่อสารในเครือข่ายอินนเตอร์เน็ตโดยใช้โปรโตคอลทีซีพีไอพี (TCP/IP) อธิบายถึงองค์ประกอบพื้นฐานกับมาตราฐานการส่งข้อมูลติดต่อกันในเครือข่ายของ OSI Model

บทที่ 3 การออกแบบ โดยจะกล่าวถึงขั้นตอนในการออกแบบพัฒนาการติดต่อสื่อสารทางเสียงผ่านเครือข่ายไอพีโดยการใช้ภาษาซีใช้ฟังก์ชั่น Winsock โดยใช้โปรแกรม Microsoft Visual C++ เวอร์ชั่น 6 ให้โปรแกรมมีความสามารถส่งข้อเสียงระหว่างเครื่องคอมพิวเตอร์ที่เชื่องต่อกันอยู่กับเครื่อข่ายอินเตอร์เน็ตได้

บทที่ 4 ผลการทดลอง โดยจะกล่าวถึงผลการทดลอง ของการส่งข้อมูลเสียงผ่านเครือข่ายอินเอตร์เน็ต

บทที่ 5 สรุป และวิจารณ์

บทที่ 6 ภาคผนวก

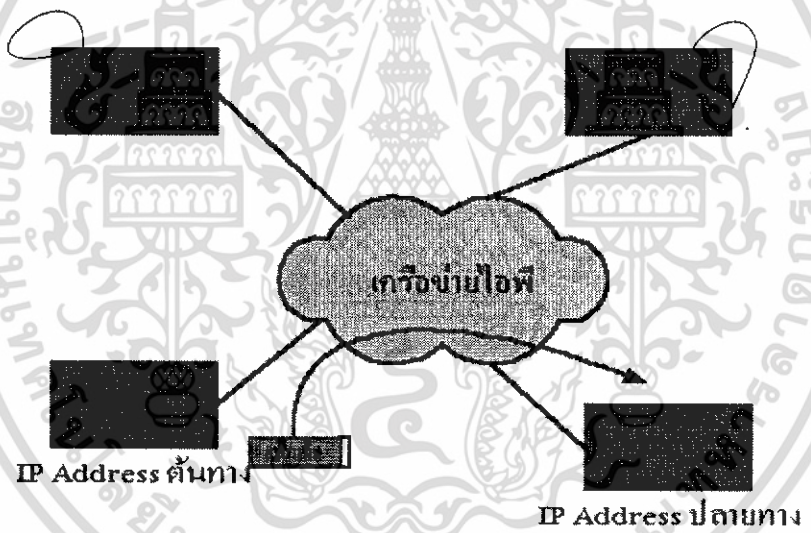
บทที่ 7 กิติกรรมประกาศ

บทที่ 8 หนังสืออ้างอิง

บทที่ 2 ทฤษฎี

2.1 หลักการพื้นฐานของเครือข่ายไอพี

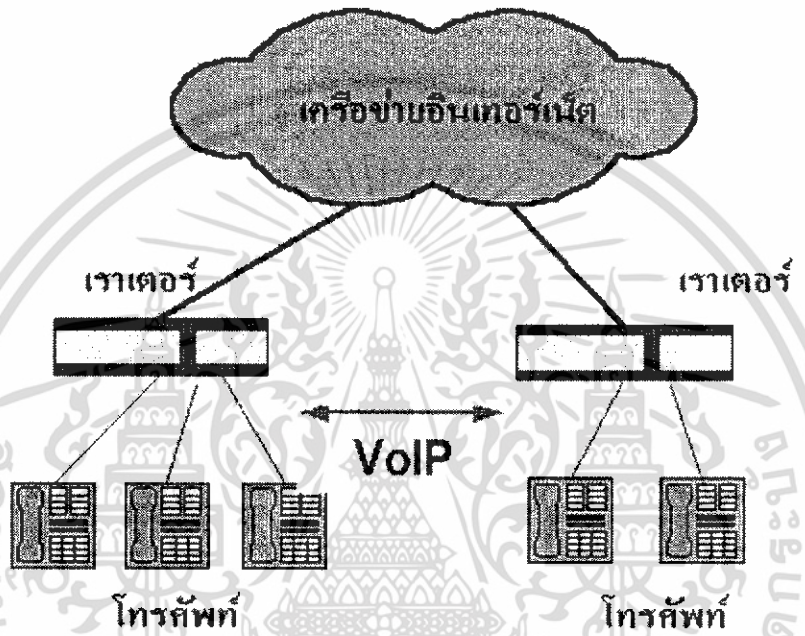
เครือข่ายไอพี (Internet Protocol) มีพัฒนามาจากรากฐานระบบการสื่อสารแบบแพ็กเก็ต โดยระบบมีการกำหนดแอดเดรส ที่เรียกว่า ไอพีแอดเดรส จากไอพีแอดเดรสหนึ่ง ถ้าต้องการส่งข่าวสารไปยังอีกไอพีแอดเดรสหนึ่ง ใช้หลักการบรรจุข้อมูลใส่ใน แพ็กเก็ต แล้วส่งไปในเครือข่าย ระบบการจัดส่งแพ็กเก็ตกระทำด้วยอุปกรณ์สื่อสารจำพวกเราเตอร์ มีหลักพื้นฐานการส่งแบบไปรษณีย์สมัยเก่า บางที่เราจึงเรียกการส่งแบบนี้ว่า ดาต้าแกรม การสื่อสารแบบไอพีแพ็กเก็ต จะเป็นการส่งแพ็กเก็ตเข้าไปในเครือข่าย โดยไม่มีการประกันว่า แพ็กเก็ตนั้นจะถึงปลายทาง เมื่อไร ดังนั้นรูปแบบของเครือข่ายไอพีจึงไม่เหมาะสมกับการสื่อสารแบบต่อเนื่อง เช่น ส่งเสียง หรือวิดีโอ



รูปที่ 2.1 การติดต่อสื่อสารกันโดยผ่านเครือข่ายไอพี

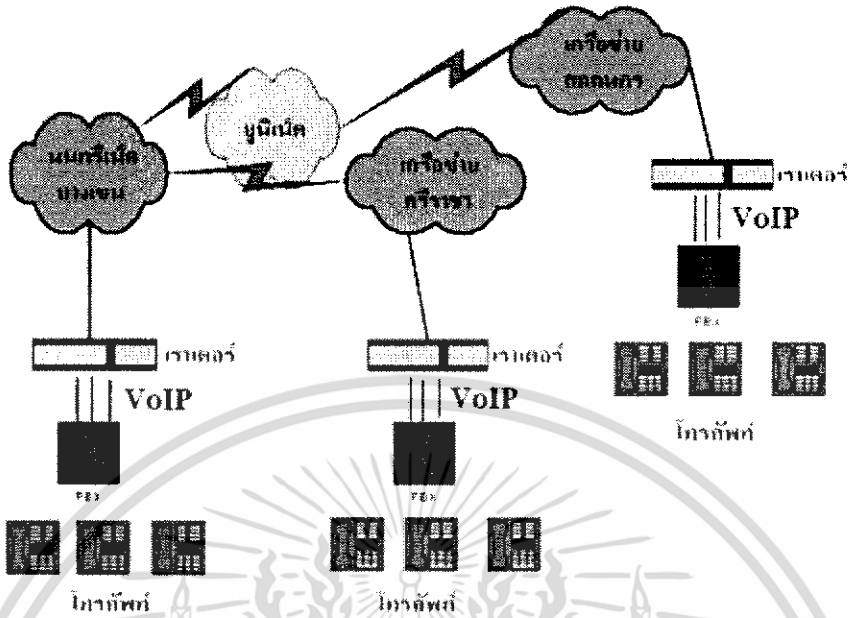
เมื่อจะส่งสัญญาณเสียง ครั้งเมื่อมีเครือข่ายไอพีกว้างขวางและเชื่อมโยงกันมากขึ้น ความต้องการส่งสัญญาณข้อมูลเสียงที่ได้คุณภาพก็เกิดขึ้น สิ่งที่สำคัญ คือระบบประกันคุณภาพการสื่อสาร โดยจัดลำดับความสำคัญ หรือจองช่องสัญญาณไว้ให้ก่อน ระบบการสื่อสารในรูปแบบใหม่นี้จะต้องกระทำโดยเราเตอร์ การสื่อสารทางเสียงผ่านเครือข่ายไอพี หรือเรียกว่า Voice Over Internet Protocal (VoIP) เป็นระบบที่นำสัญญาณข้อมูลเสียงมาบรรจุลงเป็นแพ็กเก็ต ไอพี แล้วส่งไปโดยที่เราเตอร์มีวิธีการปรับตัวเพื่อรับสัญญาณแพ็กเก็ต และยังแก้ปัญหาบางอย่างให้ เช่น การบีบอัดสัญญาณเสียง ให้มีขนาดเล็กลง การแก้ปัญหาเมื่อมีบางแพ็กเก็ตสูญหาย หรือได้มาล่าช้า

ระบบ VoIP เป็นระบบที่นำสัญญาณเสียงที่ผ่านการดิจิไตซ์ โดยหนึ่งช่องเสียงเมื่อแปลงเป็นข้อมูลจะมีขนาด 64 กิโลบิตต่อ วินาที การนำข้อมูลเสียงขนาด 64 Kbps นี้ ต้องนำมาบีบอัด โดยทั่วไปจะเหลือประมาณ 10 Kbps ต่อช่องสัญญาณเสียงแล้วจึง บรรจุลงในไอพีแพ็กเก็ต เพื่อส่งผ่านทางเครือข่ายไอพี



รูปที่ 2.2 การติดต่อสื่อสารกันโดยใช้โทรศัพท์ผ่านเครือข่ายไอพี

การสื่อสารผ่านทางเครือข่ายไอพีต้องมีเราเตอร์ที่ทำหน้าที่พิเศษเพื่อประกันคุณภาพช่องสัญญาณไอพีนี้ เพื่อให้ข้อมูลไปถึง ปลายทางหรือกลับมาได้อย่างถูกต้อง และอาจมีการให้สิทธิพิเศษก่อนแพ็กเก็ตไอพีอื่น เพื่อการให้บริการที่ทำให้เสียงมีคุณภาพ จากระบบดังกล่าวนี้เองจึงสามารถนำมาประยุกต์ใช้กับระบบเชื่อมโยงเครือข่ายโทรศัพท์ระหว่างองค์กรโดยองค์กรสามารถใช้ระบบสื่อสารทางโทรศัพท์ผ่านทางเครือข่ายไอพีด้วยวิธีการสื่อสารแบบ VoIP จึงทำให้ระบบโทรศัพท์ที่เป็นตู้ชุมสายภายในขององค์กร สามารถเชื่อมถึงกันผ่านทางเครือข่าย ไอพี การสื่อสารแบบนี้ทำให้สามารถใช้โทรศัพท์ข้ามถึงกันได้ในลักษณะ ตู้สาขาโทรศัพท์ (PABX) กับ PABX และทำให้ประหยัดค่าใช้จ่ายได้มาก



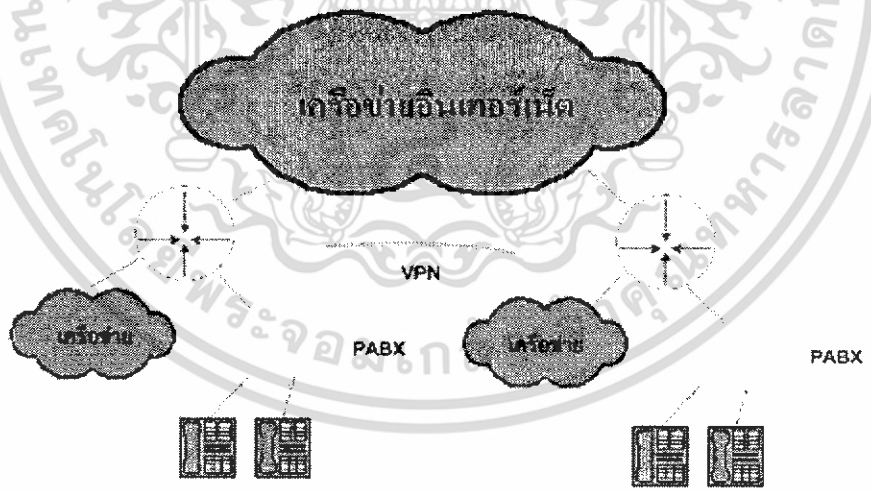
รูปที่ 2.3 การโทรศัพท์ติดต่อกันโดยใช้ตู้สาขาโทรศัพท์ผ่านเครือข่ายไอพี

2.2 การรวมกันของอินเทอร์เน็ตกับเครือข่ายโทรศัพท์

เป้าหมายของเทคโนโลยีไอที คือ การขยายฐานของผู้ใช้ให้มากที่สุดเท่าที่จะมากได้ ถ้าหากพิจารณาจากฐานผู้ใช้วันนี้ที่ร้อยล้านคน จะทำอย่างไรให้มีผู้ใช้เพิ่มเป็นร้อยพันล้านคน (ประชากรโลกมีเพียงหกพันล้านคน) นั่นหมายถึงทุกคนในโลกต้องเป็นผู้ใช้ไอทีทั้งหมด อย่าลืมว่าความแตกต่างของระดับผู้ใช้ไอทีและความรู้พื้นฐานที่จำเป็นยังเป็นเรื่องสำคัญไอทีจะก้าวเข้ามาเป็นส่วนหนึ่งของชีวิตมนุษย์ได้ต้องมีฐานให้กลมกลืนกับธรรมชาติ ทำอย่างไรจึงจะทำให้ไอทีกลมกลืนจนเป็นส่วนหนึ่งของชีวิตและเป็นธรรมชาติเหมือนสิ่งที่จำเป็นในการดำรงชีวิต สิ่งนี้เป็นสิ่งที่ท้าทายเทคโนโลยีและจะทำให้เทคโนโลยีผูกพันกับชีวิตเทคโนโลยีใหม่หลายอย่างจึงต้องเน้นในเรื่องการผสมกลมกลืนเข้าสู่วิถีธรรมชาติ เช่น การเข้าใจภาษามนุษย์ การช่วยทำงานในเรื่องเอกสารแบบมีความเข้าใจเชิงภาษามากขึ้น จนถึงระบบการแปลงคำพูดให้เป็นข้อความและข้อความกลับมาเป็นเสียงพูด การเชื่อมโยงความคิดแบบหลากหลายภาษา การแปลภาษาก็เป็นเทคโนโลยีบนพื้นฐานความท้าทาย เพื่อทำให้บุคคลในโลกสามารถสื่อสารกันได้โดยปราศจากกำแพงทางภาษามาขีดกั้นกลาง วิถีของไอทีต้องช่วยทำให้คนทั้งโลกเป็นสังคมที่เชื่อมโยงกัน โดยดำเนินชีวิตร่วมกันได้อย่างแนบแน่นหากจะพิจารณาเทคโนโลยีพีซีที่ใกล้ตัวในปัจจุบันเห็นได้ชัดว่า พีซีมีขีดความสามารถในเรื่องมัลติมีเดียเพิ่มขึ้นจากเดิมมาก พีซีมีขีดความสามารถแสดงผลเป็นกราฟิก รูปภาพ วิดีโอ และเสียง แนวโน้มที่สำคัญจึงอยู่ที่การทำให้พีซีเชื่อมโยงกับมนุษย์ในลักษณะธรรมชาติมากขึ้น การรับรู้

เสียงพูดจึงเป็นเทคโนโลยีลำดับต่อไปที่ทำให้คอมพิวเตอร์มีความสมบูรณ์เพิ่มมากขึ้นคำว่าธรรมชาติ จึงเป็นคำที่น่านึกคิดต่อไปว่า คอมพิวเตอร์ที่ใช้งานได้ง่าย เรียนรู้ได้เร็ว และมีประโยชน์สูงสุดที่เกี่ยวพันกับวิถีชีวิตและการดำเนินการ

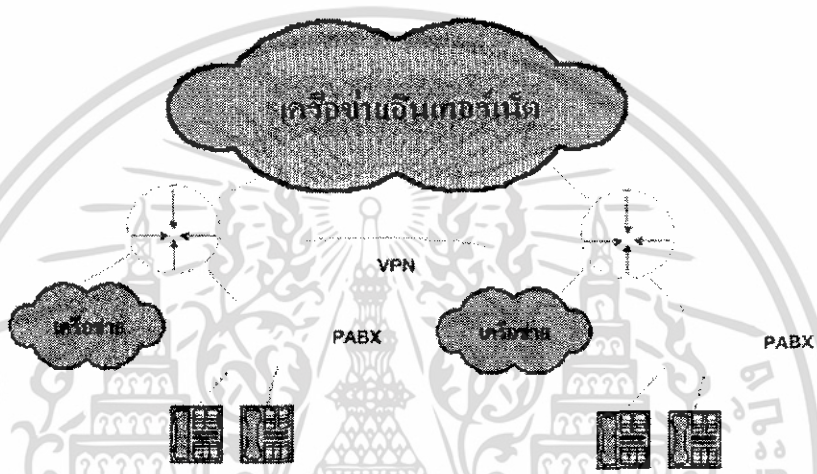
จากการที่อินเทอร์เน็ตมีพัฒนาการด้วยหลักการสื่อสารแบบแพ็กเกต หรือที่เรียกว่าไอพีแพ็กเกต ทำให้แนวโน้มของการสื่อสารเปลี่ยนแปลงจากแบบเซอร์กิต มาเป็นแบบแพ็กเกต เครือข่ายโทรศัพท์ที่ใช้เป็นเครือข่ายแบบเซอร์กิต การเชื่อมวงจรระหว่างคู่สายใช้อุปกรณ์เซอร์กิตสวิตชิง แต่เมื่อโทรศัพท์พัฒนามากขึ้น การใช้งานจึงหันมาใช้รูปแบบดิจิตอล การสวิตช์แบบดิจิตอลสำหรับโทรศัพท์ก็เป็นการใช้แบบเซอร์กิต แต่มีกลไกการใช้งานแบบแบ่งเวลา (time devision - TDM) แต่เมื่ออินเทอร์เน็ตแพร่หลาย การสื่อสารผ่านอินเทอร์เน็ตมีค่าใช้จ่ายต่ำกว่าแบบวงจรเซอร์กิตมาก เพราะการสื่อสารแบบแพ็กเกตสามารถแบ่งกันใช้ได้ ดังนั้นจึงมีผู้พัฒนาระบบโทรศัพท์ผ่านเครือข่ายอินเทอร์เน็ต ที่เรียกว่า VoIP และยังสามารถสร้างระบบการสวิตชิงแบบแพ็กเกตไอพีเพื่อใช้กับโทรศัพท์และเรียกว่า ไอพีโฟน องค์กรหลาย ๆ องค์กรจึงเริ่มใช้ระบบอินเทอร์เน็ตเชื่อมโยงเพื่อใช้โทรศัพท์ แนวคิดมีได้ตั้งแต่การสร้างวงจรเฉพาะที่เรียกว่า VPN ผ่านอินเทอร์เน็ตและใช้ระบบ VoIP ดังรูปที่ 2.4 เครือข่าย VPN เป็นเครือข่ายเฉพาะขององค์กร โดยเสมือนการเชื่อมโยงวงจรแบบ



รูปที่ 2.4 การใช้เครือข่าย VPN เชื่อมโยงเป็นอินทราเน็ตและใช้ VoIP เชื่อมโยง

วงจรเช่าผ่านทางเครือข่ายอินเทอร์เน็ต และมีสาขาอยู่ต่างประเทศที่ต่ออินเทอร์เน็ตด้วย ถ้านำเครือข่ายต่อเชื่อมถึงกัน ก็จะเสมือนมีวงจรเชื่อมผ่านอินเทอร์เน็ตเพื่อรวมสองเครือข่ายนี้เป็นเครือข่ายอินทราเน็ตขององค์กร ดังนั้นหากใช้ระบบ VoIP ก็จะเชื่อมโยงไปยังเครือข่ายเหล่านี้ได้เพราะ

เสมือนเป็นเครือข่ายเดียวกัน การพัฒนายังทำให้ระบบแพ็กเกตไอพีเป็นตัวนำข้อมูลเสียงของโทรศัพท์ โดยระบบโทรศัพท์ที่วิ่งบนอินเทอร์เน็ตนี้ เราเรียกว่า ไอพีโฟน โทรศัพท์นี้จะมีพอร์ตต่อกับฮับหรือสวิตช์ เช่น อีเทอร์เน็ต ตัวเครื่องโทรศัพท์ที่ต่อเชื่อมจึงต้องมีลักษณะพิเศษแตกต่างจากโทรศัพท์ปัจจุบันแนวโน้มที่สำคัญคือ การรวมกันของโทรศัพท์แบบแพ็กเกตที่เรียกว่า ไอพีโฟน ให้เข้ากับโทรศัพท์แบบเดิม ไดอะแกรมรูปภาพแสดงดังรูปที่ 2.5

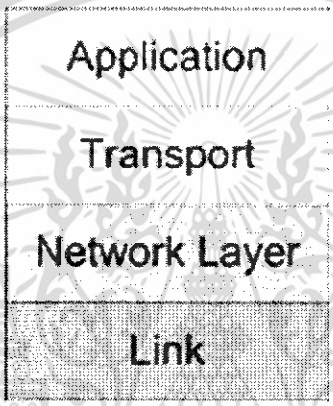


รูปที่ 2.5 การใช้เครือข่าย VPN เชื่อมโยงเป็นอินทราเน็ตและใช้ VoIP เชื่อมโยง

ไอพีโฟนจึงเป็นแนวโน้มที่สำคัญที่ทำให้การใช้โทรศัพท์และอินเทอร์เน็ตวิ่งอยู่บนเครือข่ายเดียวกัน นอกจากโทรศัพท์ในระบบใช้สายแล้ว โทรศัพท์แบบเซลลูลาร์ หรือโทรศัพท์เคลื่อนที่ ก็เป็นหนทางที่มีความนิยมสูง และมีแนวโน้มที่สำคัญ กล่าวกันว่าภายในอีกสองปีข้างหน้าทั่วโลกจะมีผู้ใช้โทรศัพท์ระบบจีเอสเอ็ม (GSM-Global System for Mobilization) มากกว่า 750 ล้านคน เฉพาะในสหรัฐอเมริกา เมื่อสิ้นปี 1999 มีผู้ใช้โทรศัพท์มือถือรวมกันทั้งสิ้นกว่า 70 ล้านเครื่อง พัฒนาการของโทรศัพท์เคลื่อนที่กำลังก้าวเข้าหาอินเทอร์เน็ต ระบบไร้สายที่เชื่อมโยงสู่อินเทอร์เน็ตและใช้งานร่วมกันกับข้อมูลเป็นสิ่งที่เป็นไปได้ ขณะนี้มีมาตรฐาน WAP (Wireless Application Protocol) ทำให้โทรศัพท์มือถือเชื่อมต่อและรับข้อมูลข่าวสารจากอินเทอร์เน็ต

2.3 เทคโนโลยี TCP/IP

TCP/IP (Transmitsion Control Protocol/InternetProtocol) เป็นชุดของโปรโตคอลที่ถูกใช้ในการสื่อสารผ่านเครือข่ายอินเทอร์เน็ต ได้รับการพัฒนามาตั้งแต่ปี 1960 ซึ่งถูกใช้เป็นครั้งแรกในเครือข่าย ARPANET ซึ่งต่อมาได้ขยายการเชื่อมต่อไปทั่วโลกเป็นเครือข่ายอินเตอร์เน็ต ทำให้ TCP/IP เป็นที่ยอมรับอย่างกว้างขวางจนถึงปัจจุบันและได้มีการแบ่งชั้น TCP/IP แบ่งออกเป็น 4 เลเยอร์ แสดงดังรูปที่ 2.6 โปรโตคอลทีซีพีไอพีมีการจัดกลไกการทำงานเป็นชั้น (Layer) เรียงต่อกันโดยชั้น



รูปที่ 2.6 การแบ่งชั้นของ TCP/IP

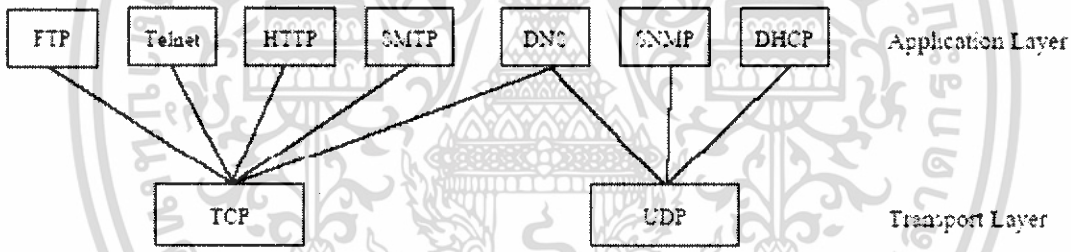
ในแต่ละชั้นจะมีการทำงานเทียบได้กับโอเอสไอโมเดลมาตรฐาน (OSImodel) แต่บางชั้นของโปรโตคอลทีซีพีไอพีจะทำงานเทียบกับ OSI หลายๆชั้นปนกันซึ่งในแต่ละชั้นจะประกอบด้วย

1. Application layer
2. Transport layer
3. Internet layer
4. Network Access layer

เมื่อได้เทียบกับมาตรฐานโอเอสไอโมเดลแล้วจะเห็นดังรูป 2.2 ซึ่งเราจะเห็นได้ว่าบางกลไกของโปรโตคอลทีซีพีไอพีเทียบได้กับมาตรฐานโอเอสไอโมเดลสองชั้นหรือบางกลไกจะทำงานคล้ายกันกับระหว่างบางชั้นของโอเอสไอโมเดลตัวอย่างเช่น กลไกการทำงานของโปรโตคอลทีซีพีไอพีในชั้นเนตเวิร์คแอคเซส (Network Access layer) เมื่อเทียบกับมาตรฐานโอเอสไอโมเดลจะเทียบได้กับชั้นดาต้าลิ้ง (Data link layer) และชั้นฟิสิกส์คอล (Physical layer) 2 ชั้นรวมกัน เป็นต้น รายละเอียดในแต่ละชั้นของโปรโตกอลทีซีพีไอพีมีดังต่อไปนี้

2.3.1 Application Layer

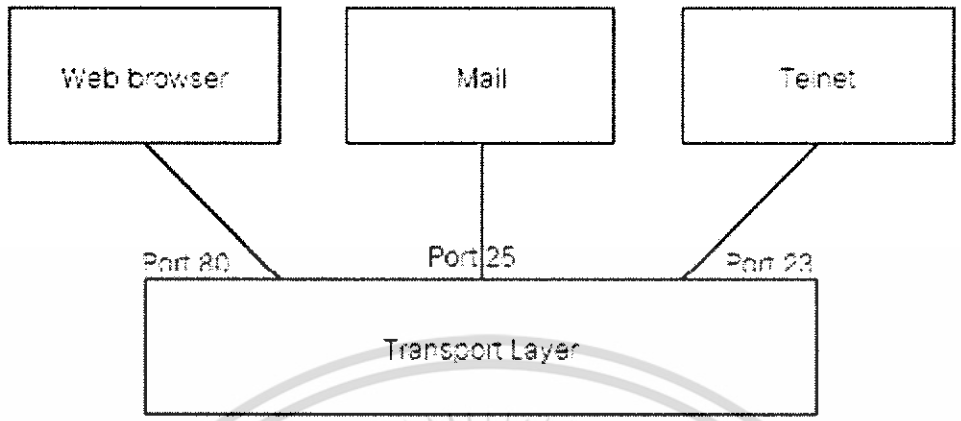
Application layer จะอยู่ชั้นบนสุดของโปรโตคอลทีซีพีไอพีซึ่งมีการทำงาน 2 หน้าที่ เมื่อเปรียบเทียบกับมาตรฐานโอเอสไอโมเดลคือ ชั้นแอพพลิเคชั่น (application layer) และ ชั้นพรีเซนเตชั่น (Presentation layer) ในชั้นนี้จะรองกับการทำงานของแอพพลิเคชั่นต่างๆที่ทำงานเป็นกระบวนการอยู่ในเครื่องผู้ให้บริการหรือเซิร์ฟเวอร์ (Server) ให้บริการและเครื่องที่ขอใช้บริการหรือไคลแอนต์ (client) ซึ่งจะติดต่อกันผ่านโปรโตคอลเฉพาะแอปพลิเคชั่นอีกทีหนึ่ง ตัวอย่างเช่นเมื่อผู้ใช้อินเตอร์เน็ตต้องการโอนถ่ายไฟล์ข้อมูลจากเครื่องเซิร์ฟเวอร์ที่ให้บริการ โดยอาจจะเรียกใช้โปรแกรม File Transfer Protocol (FTP) ทั่วไปการทำงานของแอพพลิเคชั่นต่างๆจะอยู่ที่ชั้นแอพพลิเคชั่นนี้และจะมีการติดต่อกันตามแต่ละโปรโตคอลเฉพาะแล้วแต่แอพพลิเคชั่นที่ใช้งานอยู่จากการที่ชั้นแอพพลิเคชั่นของทีซีพีไอพีรองรับให้โปรโตคอลอื่นๆทำงานได้หลายกระบวนการและหลายโปรโตคอลได้พร้อมกันนั้น ทำให้ผู้ใช้สามารถเปิดโปรแกรมใช้งานได้หลายๆอย่างพร้อมกัน เช่น เปิดโปรแกรม Internet Explorer เพื่อเรียกดูเว็บ (Web) พร้อมกับใช้งานโปรแกรม OutlookExpress



รูปที่ 2.7 การติดต่อระหว่างชั้นแอพพลิเคชั่นกับชั้นทรานสพอร์ท

2.3.2 Transport Layer

การทำงานที่ชั้นทรานสพอร์ทนี้จะมีหน้าที่จัดการต่อจากชั้นแอพพลิเคชั่น การทำงานของชั้นทรานสพอร์ท นี้จะมีการสร้างการเชื่อมต่อกันระหว่างชั้นแอปพลิเคชั่นกับชั้นทรานสพอร์ท โดยจุดที่เชื่อมกันเพื่อรับส่งข้อมูลนี้เรียกว่า พอร์ท (Port) หรือ ซ็อคเก็ต (socket) และในแต่ละแอปพลิเคชั่นก็จะสร้างการเชื่อมต่อผ่านพอร์ทได้พร้อมกันหลายแอปพลิเคชั่น ซึ่งการใช้งานพอร์ทของแต่ละแอปพลิเคชั่นที่อยู่ในชั้นทรานสพอร์ทจะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโปรโตคอลจะมีการใช้งานพอร์ทหมายเลขต่างๆไม่ซ้ำกัน ตามรูปที่ 2.8

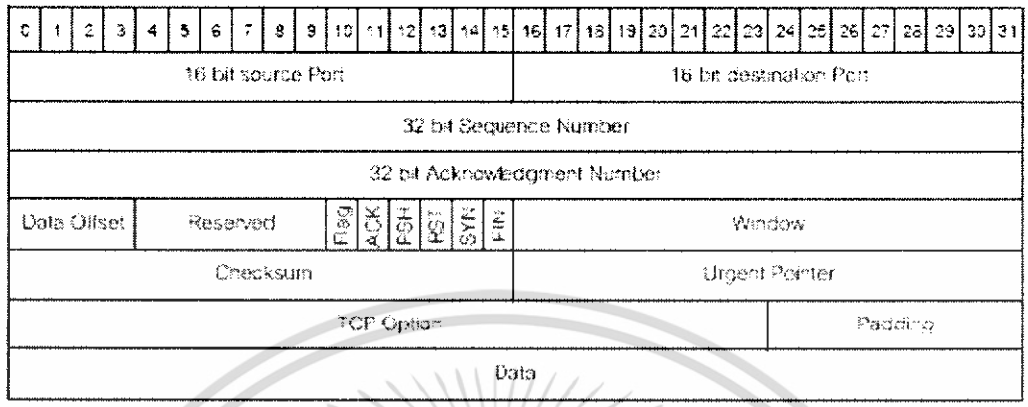


รูปที่ 2.8 การติดต่อระหว่างชั้นแอพพลิเคชั่นชั้นทรานสพอร์ทในแต่ละพอร์ท

เมื่อแอปพลิเคชั่นทำงานผ่านโปรโตคอลในชั้นแอปพลิเคชั่นจะมีการส่งผ่านข้อมูลไปยังชั้นทรานสพอร์ทที่ชั้นนี้จะมีการเชื่อมต่อผ่านพอร์ทที่กำหนดทำให้การรับส่งข้อมูลในแต่ละโปรโตคอลทำได้ถูกต้องถึงแม้ว่าในเครื่องเซิร์ฟเวอร์ที่ให้บริการจะมีการทำงานอยู่หลายโปรเซสที่แตกต่างกันก็ตาม หรือมีผู้ใช้บริการเข้ามาใช้งานพร้อมกันจำนวนมากและหลายแอปพลิเคชั่นในเวลาเดียวกัน ในชั้นทรานสพอร์ทของทีซีพีไอพีนี้ จะมีโปรโตคอลทำงานอยู่ 2 โปรโตคอลที่แตกต่างกันคือ โปรโตคอลทีซีพี (Transmission Control Protocol: TCP) และโปรโตคอลยูดีพี (User Datagram Protocol: UDP) ในการส่งผ่านข้อมูลลงไปในชั้นถัดๆไป เราจะเห็นว่าโปรโตคอลทีซีพีและยูดีพี จะผนึกเข้าไปโปรโตคอลไอพี (IP Protocol) อีกทีหนึ่งและจะส่งผ่านไปเครือข่ายอินเตอร์เน็ตต่อไปซึ่งจะแสดงรายละเอียดของโปรโตคอลทีซีพี และโปรโตคอลยูดีพี จะกล่าวดังต่อไปนี้

โปรโตคอลทีซีพี

โปรโตคอลทีซีพีเป็นโปรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่า การรับส่งข้อมูลจะไม่คำนึงถึงข้อมูลที่จะส่งไป แต่จะแบ่งข้อมูลเป็นส่วนย่อยๆก่อนและจึงส่งไป ยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งหายไป ก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูลดาต้าแกรม (datagram) ใหม่ให้ต่อเนื่องกันและประกอบกับเป็นข้อมูลทั้งหมดได้ ซึ่งจะแยกข้อมูลที่ไม่ถูกต้องออก ดังนั้นแอปพลิเคชั่นหรือโปรโตคอลใดที่อาศัยการส่งผ่านข้อมูลด้วยโปรโตคอลทีซีพี จำต้องให้หน่วยความจำและขนาดของช่องสัญญาณ (bandwidth) มากกว่าโปรโตคอลยูดีพีซึ่งหัวข้อมูลโปรโตคอลทีซีพี (Head TCP Protocol) จะแสดงดังรูปที่ 2.9



รูปที่ 2.9 หัวข้อมูลโปรโตคอลทีซีพี (Head TCP Protocol)

รายละเอียดของหัวข้อมูลโปรโตคอลทีซีพีมีดังนี้

Source Port หมายถึง พอร์ตที่โฮสต์ (Host) ต้นทางใช้ในการสื่อสารกันของส่วนนี้ และจะใช้พอร์ตนั้นไปตลอดตราบใดที่การสื่อสารในส่วนนี้ยังไม่ยุติลง โดยทั่วไปพอร์ตนี้จะเรียกว่าไคลเอนต์พอร์ต (Client port) พอร์ตที่ไคลเอนต์เปิดขึ้นมาเพื่อรอการตอบรับจากเซิร์ฟเวอร์(พิจารณาจากทิศทางของแพ็กเก็ตที่ส่งมาจากไคลเอนต์ไปยังเซิร์ฟเวอร์) ไคลเอนต์พอร์ต จะมีหมายเลขไม่แน่นอนและเปลี่ยนไปทุกครั้งที่มีการเริ่มการเชื่อมต่อใหม่ เป็นพอร์ตที่ถูกเปิดไว้ในระยะเวลาสั้น ๆ ค่าที่เป็นไปได้ของพอร์ตนี้ขึ้นอยู่กับการจัดสรรของระบบปฏิบัติการ ในการกำหนดขอบเขตของพอร์ตเหล่านี้ส่วนใหญ่จะมีค่า อยู่ในช่วง 1024 – 5000

Destination Port หมายถึง หมายเลขพอร์ตบนโฮสต์ปลายทางที่โฮสต์ต้นทางต้องการติดต่อด้วย โดยนัยแล้วจะ หมายถึงแอพพลิเคชั่นที่ให้บริการอยู่พอร์ตนั้นที่โฮสต์ปลายทางนั่นเองพอร์ตนั้นจะเรียกอีกอย่างหนึ่งว่า เซิร์ฟเวอร์พอร์ต (Server port) หมายเลขพอร์ตที่เปิดไว้จะขึ้นอยู่กับแอพพลิเคชั่นที่ให้บริการ โดยทั่วไปแอพพลิเคชั่นแต่ละประเภทจะมีหมายเลขพอร์ต เป็นมาตรฐานสำหรับให้ไคลเอนต์ได้เรียกใช้บริการ

Sequence Number เป็นส่วนที่ระบุถึงหมายเลขลำดับที่ใช้อ้างอิงในการสื่อสารข้อมูลแต่ละครั้ง เพื่อให้ทั้ง 2 ฝ่าย จะได้รับทราบตรงกันว่าเป็นข้อมูลของชุดใด การนำไปใช้งานจะได้ไม่ปะปนกันและมีลำดับที่ถูกต้อง เนื่องจากการสื่อสารข้อมูลผ่านโปรโตคอลทีซีพีนั้นจังหวะและลำดับเป็นส่วนสำคัญของโปรโตคอลไม่ยิ่งหย่อนไปกว่าข้อมูลใน TCP Header รวมไปถึง การที่ข้อมูลในแต่ละทีซีพีเซกเมนต์ (TCP Segment) อาจจะถูกทำการแฟรกเมนต์ (Fragment) ในชั้นถัดลงไป ทำให้ข้อมูลถูกแบ่งออกและส่งไปในลำดับที่ไม่เรียงกัน หากไม่มีจุดอ้างอิงของข้อมูลก็จะไม่สามารถอ่านข้อมูล

กลับใหม่ได้อย่างสมบูรณ์และถูกต้อง การส่งข้อมูลและการตอบรับจะใช้ส่วนนี้เป็นตัวยืนยันระหว่างกันเสมอ

Acknowledge Number ทำหน้าที่เช่นเดียวกับ Sequence Number ต่างกันตรงที่เป็น Sequence Number ซึ่งในการตอบรับ กล่าวคือ เนื่องจาก Sequence Number ที่ใช้ในการอ้างอิงนั้นผู้ที่เริ่มส่งข้อมูลจะเป็นผู้กำหนดเลขขึ้นมาและส่งไปพร้อมกับการสร้างการเชื่อมต่อครั้งใหม่แต่สำหรับฝ่ายที่ถูกติดต่อก็จำเป็นต้องกำหนดหมายเลขสำหรับใช้อ้างอิง ในการตอบรับเช่นกัน ค่าที่อยู่ใน Acknowledge Number ก็คือหมายเลขที่ใช้อ้างอิงในการตอบรับนี้

Header Length โดยปกติความยาวของ TCP Header จะเท่ากับ 20 ไบต์ แต่ถ้าหากมีการใช้ออปชั่น (Option) อาจจะทำให้ ขนาดของเฮดเดอร์ยาวขึ้นตามข้อมูลที่ต้องเพิ่มมาจากออปชั่นนั้นแต่ทั้งหมดแล้วจะไม่เกิน 60 ไบต์ เป็นข้อมูลในระดับบิตที่ใช้เป็นตัวบอกคุณสมบัติของทีซีพีเซกเมนต์ที่กำลังส่งอยู่นั้น และใช้เป็นตัวควบคุมจังหวะการรับส่งข้อมูลด้วย ซึ่งแฟรกทั้งหมดมีอยู่ 6 บิต แต่ละบิตมีชื่อและมีความหมายดังนี้

URG ใช้บอกความหมายว่าเป็นข้อมูลด่วน และมีข้อมูลพิเศษมาด้วย

ACK แสดงว่าข้อมูลในฟิลด์ Acknowledge Number นำมาใช้งานได้

DSH แจ้งให้ผู้รับข้อมูลทราบว่าควรส่งข้อมูลเซกเมนท์นี้ไปยังกระบวนการที่กำลังรออยู่ทันที

RST ใช้ในกรณีที่เกิดการสับสนขึ้นด้วยเหตุผลต่างๆ ให้เริ่มต้นสื่อสารกันใหม่

SYN ใช้ในการเริ่มต้นขอติดต่อกับปลายทาง

FIN ใช้ส่งเพื่อแจ้งให้ปลายทางทราบว่ายุติการติดต่อ

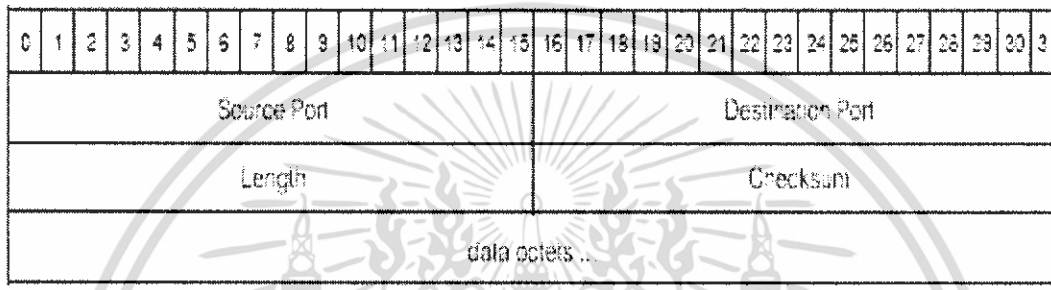
Window เป็นขนาดของการรับส่งข้อมูลในแต่ละครั้งที่ทางฝ่ายผู้รับจะสามารถรับได้เนื่องจากในการรับข้อมูลนั้น ทางผู้รับจะต้องจัดเตรียมหน่วยความจำในการพักข้อมูลที่มากจาก ทีซีพี และทำการดีมัลติเพลกซ์ (Demultiplex) ออกมา หากไม่มีการตกลง ถึงขนาดที่ทางฝ่ายรับสามารถรับได้ ก็จะทำให้การสื่อสารข้อมูลไม่สมดุล และฝ่ายรับอาจจะประมวลผลทัน ซึ่งจะส่งผลให้ต้องส่งข้อมูลซ้ำหลายครั้ง

Checksum ใช้ในการตรวจสอบความถูกต้องของข้อมูลในทีซีพีเซกเมนต์ใช้ระบุหมายเลข Sequence Number ทีซีพีเซกเมนต์ล่าสุดที่อยู่ในโหมด Urgent

Urgent Pointer ข้อมูลเพิ่มเติมซึ่งจะอยู่ใน TCP Header เมื่อมีการตั้งค่าออปชั่นบางอย่างที่ต้องการข้อมูลเพิ่มเติมซึ่งไม่มีใน TCP Header เช่น MSS, Strict Route

โปรโตคอลยูดีพี

UDP เป็นโปรโตคอลที่ถูกออกแบบมาให้ทำหน้าที่รับส่งข้อมูลโดยมีขั้นตอนการทำงานไม่ซับซ้อนและทำงานได้รวดเร็ว แต่มีจุดด้อยคือไม่มีความน่าเชื่อถือ (unreliable) และเป็นการสื่อสารแบบไม่ต่อเนื่อง (connectionless) โปรโตคอลยูดีพีทำงานในชั้นทรานสพอร์ท ซึ่งจะต้องพึ่งพาโปรโตคอลไอพีในการรับส่งข้อมูล



รูปที่ 2.10 หัวข้อมูลโปรโตคอลยูดีพี (Head UDP Protocol)

รายละเอียดของหัวข้อมูลโปรโตคอลยูดีพีมีดังนี้

Source port หมายเลขพอร์ตต้นทางที่ส่งดาต้าแกรมนี้ มีความยาว 16 บิต

Destination port หมายเลขพอร์ตปลายทางที่จะเป็นผู้รับดาต้าแกรม มีความยาว 16 บิต

UDP length ความยาวของดาต้าแกรมทั้งส่วนหัวข้อมูลและข้อมูล นั่นหมายความว่า ค่าที่น้อยที่สุดในส่วนนี้คือ 8 ซึ่งเป็นขนาดของหัวข้อมูล

Checksum เป็นตัวตรวจสอบความถูกต้องของยูดีพีดาต้าแกรม (UDP datagram) และจะนำข้อมูลบางส่วนใน IP Header มาคำนวนด้วย

ตัวอย่างขั้นตอนกลๆไกการทำงานโดยใช้โปรโตคอลยูดีพีมีดังต่อไปนี้

1. ในชั้นแอพพลิเคชั่นเมื่อโปรแกรมควบคุมอุปกรณ์เครือข่ายเช่น โปรแกรม Network Management ต้องการรับส่งข้อมูลไปยังอุปกรณ์ที่ต้องการ แอปพลิเคชั่นนั้นจะติดต่อโปรโตคอล SNMP ในชั้นแอพพลิเคชั่น

2. โปรโตคอล SNMP จะติดต่อกับโปรโตคอลยูดีพีในชั้นถัดไปเพื่อข้อติดต่อผ่านพอร์ทที่กำหนด

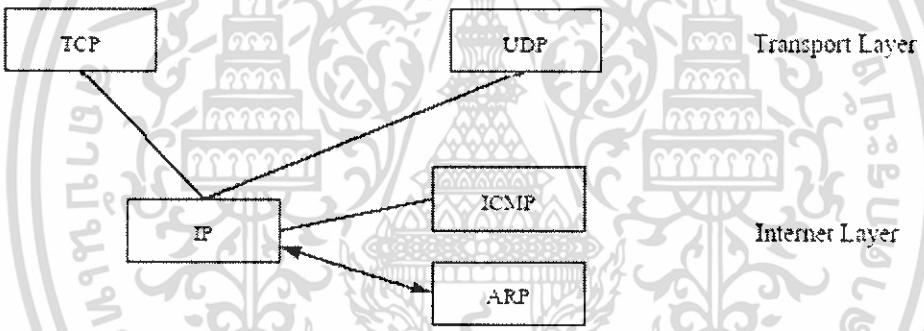
3. โปรโตคอล SNMP เตรียมข้อมูลที่จะส่งรวมทั้งที่อยู่ปลายทาง

4. โปรโตคอล SNMP ส่งผ่านข้อมูลให้โปรโตคอลยูดีพีที่อยู่ในชั้นทรานสพอร์ท

5. โปรโตคอลยูดีพี ทำหน้าที่ผนึกข้อมูลหรือดาต้าแกรมนั้นไปกับโปรโตคอลไอพีในชั้นถัดลงไปเพื่อส่งข้อมูลออกจากเครื่อง ซึ่งจะเห็นว่ามีกลไกต่างจากการส่งข้อมูลด้วยโปรโตคอลทีซีพีซึ่งจะต้องมีการติดต่อกันก่อน และทั้งสองฝ่ายรับทราบการรับส่งข้อมูลของช่องการส่งข้อมูลนั้น

2.2.3 Internet Layer

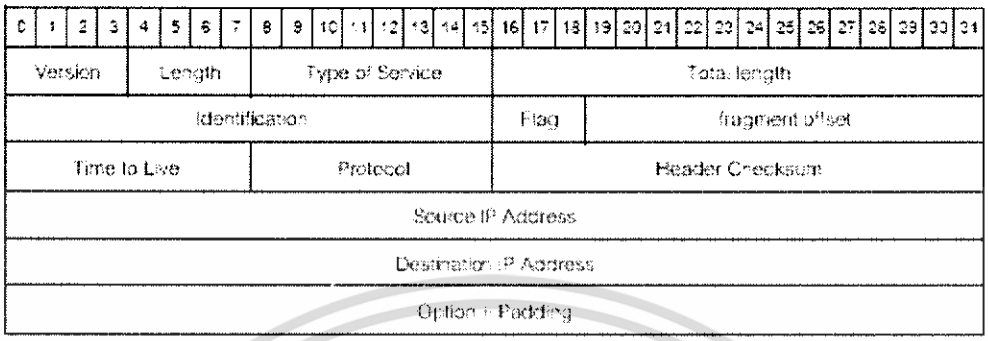
ในระดับล่างต่อมาในชั้นอินเตอร์เนต (Internet Layer) มีหน้าที่ส่งผ่านข้อมูลในระหว่างเครือข่าย โดยมีโปรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใดๆบนอินเตอร์เน็ต คือ โปรโตคอลไอพี นอกจากนี้ในชั้นอินเตอร์เนตยังมีโปรโตคอลทำงานอีกด้วยอีก 2 ชนิดคือ โปรโตคอล Internet Control Message Protocol (ICMP) และโปรโตคอล Address Resolution Protocol (ARP)



รูปที่ 2.11 แสดงการติดต่อระหว่างชั้นทรานสพอร์ทกับชั้นอินเตอร์เนต

โปรโตคอล IP

โปรโตคอลไอพี ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาชั้นทรานสพอร์ท เพื่อส่งข้ามไปยังเครือข่ายใดๆได้อย่างถูกต้อง แม้จะมีเครือข่ายเชื่อมต่อกันในอินเตอร์เน็ตเป็นล้านๆเครือข่ายก็ตามเนื่องจากโปรโตคอลไอพีมีข้อมูลตำแหน่งไอพี (IP) ปลายทางที่จะส่งข้อมูลไปให้ โดยทำงานรวมกับอุปกรณ์เราท์เตอร์ (Router) เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้ ตัวโปรโตคอลไอพีจะทำงานแบบ packet switching คือมีการส่งข้อมูลผ่านสวิทช์ (Switch) ไปยังปลายทาง โดยข้อมูลจะเดินทางไปยังเครือข่ายต่างๆผ่านสวิทช์นี้ไปเรื่อยๆจะกว่าจะถึงปลายทาง ตัววงจรผ่านหรือสวิทช์นี้อาจเป็นเกทเวย์ (Gateway) หรือเราท์เตอร์ (Router) ในระบบเครือข่ายก็ได้



รูปที่ 2.12 หัวข้อมูลโปรโตคอลไอพี (Header IP Protocol)

รายละเอียดของหัวข้อมูลโปรโตคอลไอพีมีดังนี้

Version แสดงเวอร์ชั่น (Version) ของ IP ปัจจุบันค่านี้ถูกกำหนดให้เป็น 4

Length แสดงค่าความยาวของ Header

Type of Service ปัจจุบันไม่นิยมใช้

Total length แสดงค่าจำนวนไบต์ทั้งหมดของ IP Datagram

Identification ใช้ในกรณีที่มีการแบ่งดาต้าแกรมออกเป็นแฟรกเมนต์ เมื่อนำกลับมารวมกันใหม่ จะได้รู้ว่ามาจากดาต้าแกรมเดียวกัน

Flag ใช้ในกรณีที่มีการแบ่งข้อมูลออกเป็นแฟรกเมนต์ มีความหมายดังนี้

บิต 0: reserved เป็น 0 เสมอ

บิต 1 (DF) 0 = อาจจะมีแฟรกเมนต์, 1 = ไม่มีแฟรกเมนต์

บิต 2 (MF) 0 = มีแฟรกเมนต์ตัวสุดท้าย, 1 = มีแฟรกเมนต์จำนวนมาก

Fragment offset เป็นส่วนระบุข้อมูลที่ใช้แยกรวมข้อมูล เพื่อให้ข้อมูลที่ถูกแยกออกเป็นแฟรกเมนต์กลับมารวมกันได้อย่างถูกต้องตามลำดับ

Time to live เป็นจำนวนครั้งสูงสุดที่ดาต้าแกรมนี้จะถูกส่งผ่านเครือข่ายไปยังปลายทางได้ เพื่อป้องกันไม่ให้ดาต้าแกรมถูกส่งไปเรื่อยๆอย่างไม่สิ้นสุด ปกติค่านี้จะเริ่มต้นที่ 32 และจะถูกลดค่าลงทีละ 1 เมื่อมีการเราต์ (Route) จนค่านี้มีค่าเป็น 0

Protocol เป็นข้อมูลที่ระบุโปรโตคอลที่ส่งดาต้าแกรมนี้มา

Header Checksum เป็นส่วนตรวจสอบความถูกต้องของข้อมูลใน Header

Source IP Address IP Address ของผู้ส่งดาต้าแกรม

Destination IP Address IP Address ของผู้รับดาต้าแกรม

Option มีขนาดข้อมูลไม่แน่นอน ใช้สำหรับกำหนดค่าพารามิเตอร์ปลีกย่อย

Padding มีข้อมูลว่างเปล่า ใช้เป็นส่วนเติมเต็มของฟิลด์ Option ให้ครบ 32 ไบต์

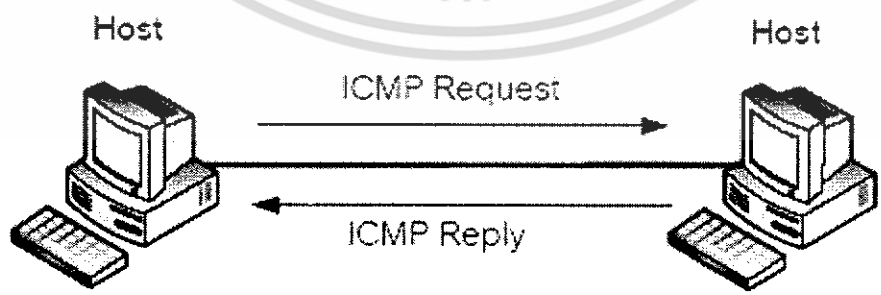
โปรโตคอล ICMP

หน้าที่หลักของโปรโตคอล ICMP (Internet Control Message Protocol) คือ การแจ้งหรือแสดงข้อความจากระบบ เพื่อบอกให้ผู้ใช้ทราบว่าเกิดอะไรขึ้นในการส่งผ่านข้อมูลนั้น ที่ปัญหาส่วนใหญ่ที่พบคือส่งไปไม่ได้ เป็นต้น นอกจากนี้โปรโตคอล ICMP ยังถูกใช้งานจากเครื่องเซิร์ฟเวอร์และเราท์เตอร์อีกด้วย เพื่อแลกเปลี่ยนข้อมูลที่ใช้ควบคุม ส่วนกับโปรโตคอล IP ในระดับเดียวกันและข้อความต่างๆที่แจ้งให้ทราบจะถูกผนึกอยู่ภายในส่วนของ IP อีกทีหนึ่ง ข้อความที่โปรโตคอล ICMP ส่งนั้นแบ่งออกได้ 2 แบบคือ ICMP error message หรือข้อความแจ้งข้อผิดพลาดและ ICMP query หรือข้อความเรียกข้อมูลเพิ่ม ตัวอย่างกลไกการทำงานของโปรโตคอล ICMP เช่น เมื่อมีการส่งผ่านข้อมูลจากผู้ใช้ไปยังปลายทางที่ไม่ถูกต้อง ขณะนั้นเครื่องปลายทางเกิดปัญหาจนไม่สามารถรับข้อมูลได้ที่เราท์เตอร์จะส่งข้อความแจ้งเป็น ICMP message ที่ชื่อ destination unreachable ให้กับผู้ส่งข้อมูล นอกจากนี้ตัวข้อมูลที่แจ้งข้อความก็จะมีส่วนของข้อมูล IP datagram ที่เกิดปัญหาด้วย ดังนั้นเมื่อผู้ส่งข้อมูลได้รับข้อความแจ้งแล้วก็จะทราบได้ว่าจุดที่เกิดปัญหานั้นอยู่ที่ใดดังนั้นโปโตคอล ICMP จึงกลายมาเป็นเครื่องมืออย่างหนึ่งในการช่วยทดสอบเครือข่าย เช่น คำสั่ง ping ที่เรามักใช้ทดสอบเครื่องเซิร์ฟเวอร์ที่ให้บริการหรืออุปกรณ์ที่ต่ออยู่ในเครือข่ายอินเตอร์เน็ตนั้นยังทำงานเป็นปกติหรือไม่ แล้วคำสั่ง ping ที่เราเรียกใช้งานโปรโตคอล ICMP แจ้งเป็นข้อความให้ทราบอีกต่อหนึ่ง

การใช้งาน ICMP

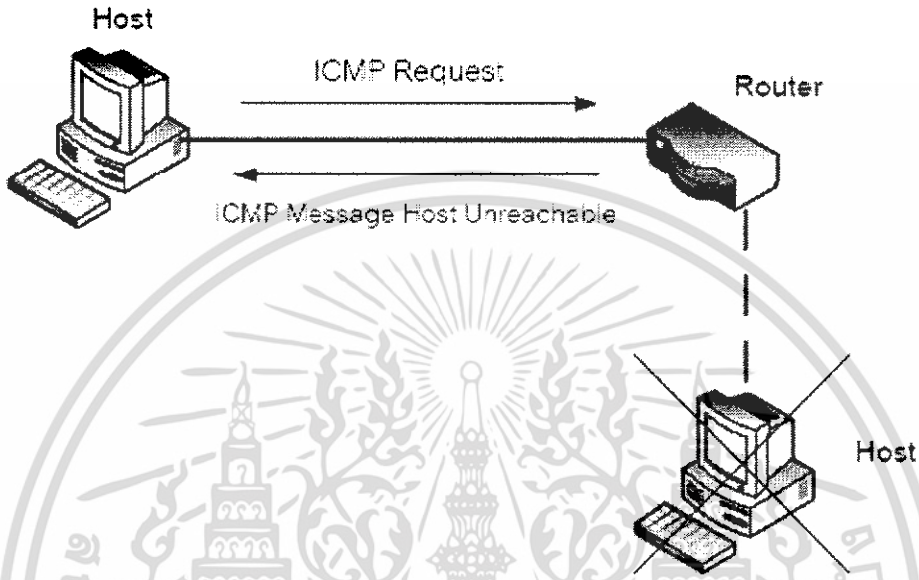
โดยทั่วไป ICMP มีการใช้งานในสองลักษณะคือ

1. Query ใช้สอบถามสถานะระหว่างกัน แสดงดังรูปที่ 2.13 เป็นการส่ง Echo request เพื่อถามสถานะของปลายทาง ซึ่งโฮสปลายทางอยู่ในสถานะปกติ สามารถทำการสื่อสารได้จะส่ง Echo Reply กลับมา



รูปที่ 2.13 การใช้งานโปรโตคอล ICMP เพื่อสอบถามสถานะระหว่างกัน

2. Error Report ใช้รายงานข้อผิดพลาดที่เกิดขึ้น เช่น หากไม่สามารถส่งดาต้าแกรมไปถึงปลายทางได้ เราเตอร์จะส่ง ICMP Message Host Unreachable กลับมารายงานโฮสต้นทาง แสดงดังรูปที่ 2.14



รูปที่ 2.14 การใช้งานโปรโตคอล ICMP เพื่อรายงานข้อผิดพลาดที่เกิดขึ้น

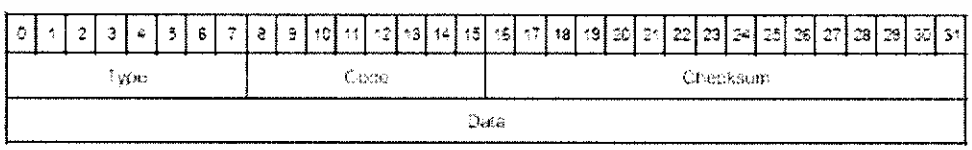
การรายงานข้อผิดพลาดของ ICMP นั้น จะ ไม่รายงานข้อผิดพลาดในบางกรณีคือ

1. เกิดความผิดพลาดในการส่ง ICMP เสียเอง

2. การส่งข้อมูลเป็นการส่งข้อมูลแบบ Multicast หรือ Broadcast

3. มีการแบ่งดาต้าแกรมออกเป็นส่วนย่อยๆ เรียกว่าแฟรกเมนต์ ในกรณีนี้จะส่ง ICMP Message (แสดงดังรูปที่ 2.14) สำหรับแฟรกเมนต์แรกเพียงครั้งเดียว ไม่ต้องแจ้งกลับทุกแฟรกเมนต์ เพราะถือว่าเป็นดาต้าแกรมเดียวกัน

4. แอดเดรส (Address) ต้นทางไม่ได้เจาะจงโฮส เช่น แอดเดรสที่เป็น 0 ทั้งหมด แอดเดรสที่เป็น loop back หรือ แอดเดรสที่เป็นแบบมัลติแคส (Multicast) หรือบรอดแคส (Broadcast)



รูปที่ 2.14 แสดงรูปร่างของ ICMP Message

รายละเอียดของ Header IP Protocol มีดังนี้

Type ประเภทของ ICMP Message

Code รหัสของ ICMP Message

Checksum เป็นตัวตรวจสอบความถูกต้องของ ICMP Massage ทั้งหมด

Data ข้อมูลภายใน ICMP Message ขึ้นอยู่กับ Type

ตำแหน่งต่างๆของข้อมูลภายใน Data จะขึ้นอยู่กับชนิดของ ICMP นั้นๆ ซึ่งมีรายละเอียดค่อนข้างมาก สามารถศึกษาโครงสร้างของ Data ของ ICMP แต่ละชนิดได้จาก RFC 792

ICMP Message Type มีความหมายดังนี้

0 Echo Reply

3 Destination Unreachable

4 Source Quench

5 Redirect Message

8 Echo Request

11 Time Exceeded

12 Parameter Problem

13 Timestamp Request

14 Timestamp Reply

15 Information Request (No Longer Used)

16 Information Reply (No Longer Used)

17 Address Mask Request

18 Address Mask Reply

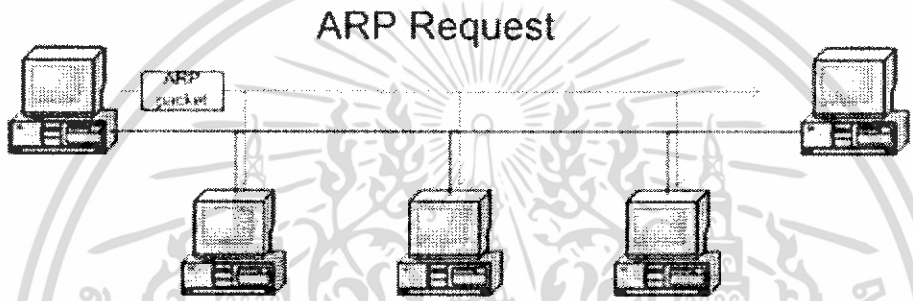
โปรโตคอล ARP

โปรโตคอล ARP (Address Resolution Protocol) ถูกเรียกใช้งานโดยโปรโตคอลไอพี เพื่อช่วยแปลงหมายเลขไอพีไปเป็นหมายเลข (Hardware) ฮาร์ดแวร์ปลายทาง ตัวอย่างเช่น เซิร์ฟเวอร์เครื่องหนึ่งเชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ต ในการเชื่อมต่อนี้ต้องอาศัย Network Interface card หรือ LAN card ติดตั้งอยู่ที่ LAN card เองจะมีหมายเลขเฉพาะประจำฮาร์ดแวร์ที่ไม่ซ้ำกับใคร เพื่อใช้อ้างอิงการส่งข้อมูลในเครือข่าย แต่เมื่อมาใช้งานในโปรโตคอลทีซีพีไอพีจะต้องมีการกำหนดหมายเลขไอพีแอดเดรสประจำตัวเพื่อใช้อ้างอิงกัน และโปรโตคอล ARP จะทำหน้าที่แปลงค่าหมายเลขไอพีให้เป็นหมายเลขฮาร์ดแวร์จริงในระดับการทำงานชั้นอินเตอร์เนตนี้ซึ่งกลไกการแปลงนี้เรียกว่า address resolution

กลไกการทำงานของ ARP

การทำงานของ ARP มี 2 ขั้นตอนคือ

1. เครื่องที่ต้องการสอบถามแม็คแอดเดรส (MAC Address) ส่ง ARP packet เรียกว่า ARP Request ซึ่งบรรจุไอพี, แม็คแอดเดรสของตนเอง และไอพีแอดเดรสของเครื่องที่ต้องการทราบแม็คแอดเดรสส่วนแม็คแอดเดรสปลายทางนั้น จะถูกกำหนดเป็น FF:FF:FF:FF:FF:FF ซึ่งเป็น Broadcast Address เพื่อให้ ARP packet ถูกส่งไปยังเครื่องทุกเครื่องที่อยู่ในเครือข่ายเดียวกันรูปที่ 2.15 ARP Request จะถูกส่งไปยังเครื่องทุกเครื่องในเน็ตเวิร์ค



รูปที่ 2.15 ARP Request จะถูกส่งไปยังเครื่องทุกเครื่องในเน็ตเวิร์ค

ARP Cache

กระบวนการ ARP จะเกิดขึ้นทุกครั้งที่มีการส่งไอพีดาต้าแกรมและกระบวนการ ARP ก็กินเวลารับส่งข้อมูลและทรัพยากรในเครือข่ายพอสมควร โดยเฉพาะในจุดที่ต้องมีการ Broadcast ARP Request ซึ่งหากเป็นเช่นนั้น ช่องสัญญาณอันมีค่าของเครือข่ายจะหมดไปกับ ARP Packet ที่วิ่งในสายเคเบิ้ล จึงมีการออกแบบบัฟเฟอร์ (Buffer) เป็นตารางจับคู่ ระหว่าง ARP กับไอพีแอดเดรสเพื่อไม่ต้องส่ง ARP Request / Reply ทุกครั้งที่จะทำการส่ง IP datagram แต่ไอพีแอดเดรสนั้นเป็นสิ่งที่ผู้ใช้กำหนดขึ้น เป็น Logical ซึ่งสามารถเปลี่ยนแปลงได้ ดังนั้น ข้อมูลในตารางนี้จึงต้องมีอายุการใช้งาน โดยทั่วไป กำหนดให้เป็นเวลา 20 นาที เมื่อหมดเวลาแล้ว หากจะส่งไอพีดาต้าแกรมครั้งต่อไปจะต้องทำการส่ง ARP Request ใหม่ ท่านสามารถเรียกดู ARP cache ในระบบปฏิบัติการ Linux ได้ด้วยคำสั่ง

#ARP [Enter]

```
[root@Sandy root]# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
172.17.0.1               ether   00:09:E9:95:D9:40   C                     eth0
172.17.3.28              ether   00:01:03:41:52:D5   C                     eth0
[root@Sandy root]#
```

รูปที่ 2.16 ผลของคำสั่ง arp แสดงตาราง arp cache สำหรับระบบปฏิบัติการ Linux

ARP Packet Format

รายละเอียดของ ARP Packet Format มีดังนี้

Ethernet Destination Address

มีขนาด 6 ไบต์ ส่วนนี้อยู่ในหัวข้อมูลของ Ethernet Frame ทั่วไป มีความหมายเป็นแอดเดรสปลายทาง ในกรณีของ ARP Request ข้อมูลในฟิลด์นี้จะเป็น FF:FF:FF:FF:FF:FF

Ethernet Source Address

มีขนาด 6 ไบต์ ส่วนนี้อยู่ใน Header ของ Ethernet Frame ทั่วไป มีความหมายเป็นแอดเดรสต้นทาง

Ethernet Frame Type

มีขนาด 2 ไบต์ ระบุถึงโปรโตคอลที่ถูกห่อหุ้มอยู่ใน Ethernet Frame นี้ สำหรับ ARP จะต้องเป็น 0x0806

Hard Type

มีขนาด 2 ไบต์ ระบุถึงประเภทของ Hardware ที่โปรโตคอล ARP ถามอยู่ สำหรับกรณีนี้คือ

Ethernet Address จะต้องเป็น 1

Prot Type

มีขนาด 2 ไบต์ ระบุชนิดของโปรโตคอลที่ถาม ว่าเป็น Hardware Address ของโปรโตคอลแบบใด ในที่นี้คือ ไอพี

Hard Size

มีขนาด 1 ไบต์ ระบุขนาดของ Hardware ที่โปรโตคอล ARP ถาม ซึ่งมีค่าเป็น 1 สำหรับ

Ethernet Address

Prot Size

มีขนาด 1 ไบต์ ระบุขนาดของแอดเดรสในโปรโตคอลที่ถาม ซึ่งมีค่าเป็น 4 สำหรับ IP

OP Field

มีขนาด 2 ไบต์ เป็นการระบุว่าเป็น ARP ชนิดใดซึ่งมีค่าดังนี้

1 = ARP Request 2 = ARP Reply

18

3 = RARP Request 4 = RARP Reply

Sender Ethernet Address

มีขนาด 6 ไบต์ Ethernet Address ของผู้ส่ง มีค่าซ้ำกับ Ethernet Source Address

Sender IP Address

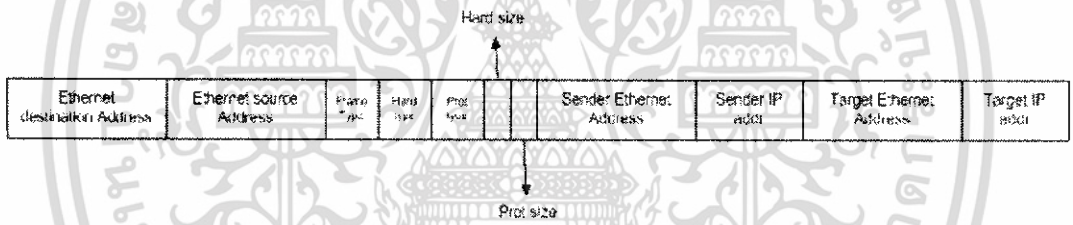
มีขนาด 4 ไบต์ IP Address ของผู้ส่ง

Target Ethernet Address

มีขนาด 6 ไบต์ Ethernet Address ของผู้รับ ค่านี้จะว่างไว้ในกรณีของ ARP Request

Target IP Address

มีขนาด 4 ไบต์ ไอพีแอดเดรสของผู้รับ



รูปที่ 2.17 ARP Packet Format

IP Addressing

อินเตอร์เฟซ (Interface) ที่ต่ออยู่บนอินเตอร์เน็ตจะต้องมีหมายเลขประจำตัวเพื่อใช้ในการสื่อสารข้อมูล เรียกว่า Internet Address หรือเรียกย่อๆว่า ไอพีแอดเดรส โดยค่าไอพีแอดเดรสนี้จะเป็นหมายเลขจำนวน 32 บิต แต่แทนที่จะกำหนดให้เลขทั้ง 32 บิตนั้นถูกนับต่อเนื่องกันไป ก็จะใช้วิธีการแบ่งหมายเลขดังกล่าวออกเป็นกลุ่มของเลขขนาด 8 บิตจำนวน 4 ชุด และคั่นแต่ละชุดด้วยจุด ตัวอย่างเช่น 172.17.3.12 นอกจากนี้ในไอพีแอดเดรสนั้นยังถูกแบ่งออกเป็น 2 ส่วนคือ ส่วนที่เป็นแอดเดรสของเครือข่าย (Network ID) และส่วนที่เป็นแอดเดรสของโฮสต์ (Host ID) ซึ่งข้อมูลในส่วนนี้จะถูกใช้สำหรับ ค้นหาเส้นทางของไอพี ในการที่จะขนส่งข้อมูลจากต้นทางให้ถึงปลายทางอย่างถูกต้อง เพื่อเป็นการกำหนดขนาดของเครือข่าย สำหรับไอพีแอดเดรสต่างๆดังนั้นจึงมีการจัดไอพีแอดเดรสในแต่ละช่วงออกเป็นคลาส (class) ต่างๆกันจาก A ถึง E เพื่อจะได้ทำการจัดสรรไอพีแอดเดรส ได้อย่างเหมาะสมกับขนาดของเครือข่ายจากข้อกำหนดในการแบ่งคลาสของไอพีแอด

เดรสหากลองนำบิตที่อยู่ในตอนต้นของไอพีแอดเดรสในแต่ละคลาสมาแปลงเป็นไอพีแอดเดรสในเลขฐานสิบ จะเห็นว่าแต่ละคลาสครอบคลุมไอพีแอดเดรสช่วงต่างๆ ดังตารางที่ 2.1

Class	IP Range
A	10.0.0.0 - 127.255.255.255
B	128.0.0.0 - 191.255.255.255
C	192.0.0.0 - 223.255.255.255
D	224.0.0.0 - 239.255.255.255
E	240.0.0.0 - 255.255.255.255

ตารางที่ 2.1 แสดงค่าช่วงของไอพีแอดเดรสในแต่ละคลาส

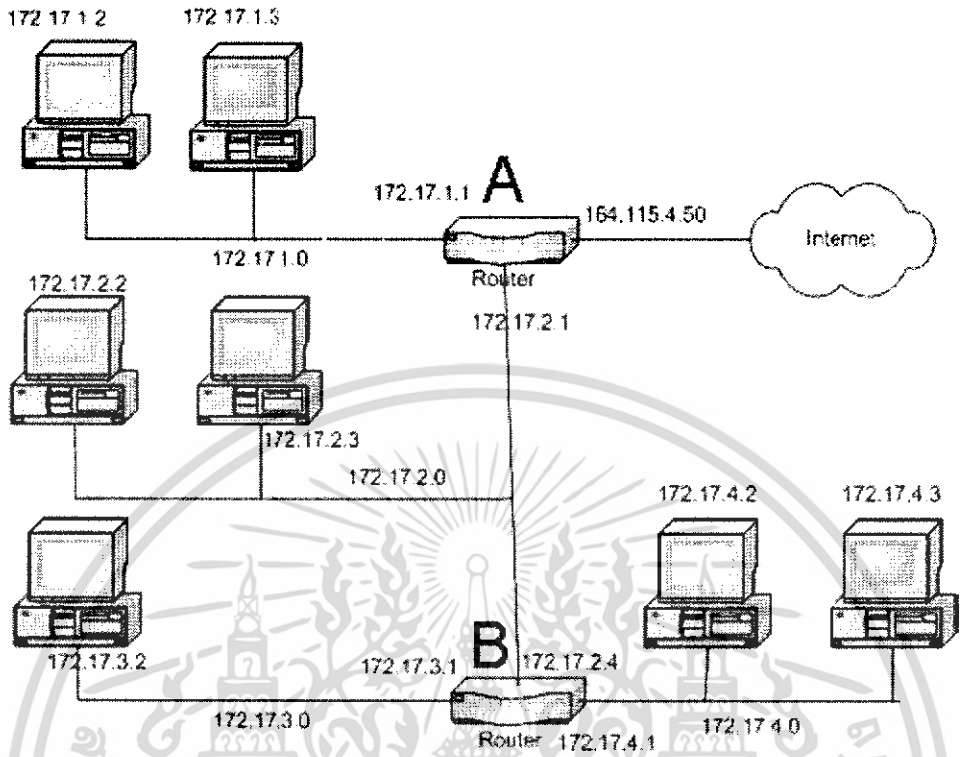
IP Routing

ไอพีเราท์ติ่ง (IP Routing) เป็นกระบวนการค้นหาเส้นทางในการส่งผ่านข้อมูลจากต้นทางไปยังปลายทางโดยผ่านการส่งต่อข้อมูลไปจนกว่าจะถึงปลายทาง นับเป็นกลไกสำคัญที่ทำให้ไอพีเป็นโปรโตคอลที่สามารถส่งข้อมูลจากโฮสต์หนึ่งไปอีกโฮสต์หนึ่งได้แม้ว่าจะอยู่ไกลแสนไกล ผมขอเริ่มต้นทฤษฎีไอพีเราท์ติ่ง ด้วยการทำความรู้จักกับส่วนประกอบต่างๆ ของเน็ตเวิร์ค ในแง่ของไอพีเราท์ติ่งกันก่อน

โฮสต์ (Host) เป็นอุปกรณ์ที่ทำหน้าที่ให้กำเนิดข้อมูลในกรณีเป็นผู้ส่ง หรือทำหน้าที่รับข้อมูลไปใช้งานในกรณีเป็นผู้รับ การสื่อสาร ข้อมูลใดๆจะต้องเป็นการสื่อสารจากโฮสต์ไปยังโฮสต์เสมอ สำหรับ IP Packet แล้วข้อมูลในเฮดเดอร์ที่ปรากฏอยู่ในฟิลด์ Source Address และDestination Address ซึ่งเรียกว่า ไอพีแอดเดรส จะเป็นหมายเลขระบุตำแหน่งของโฮสต์ต้นทางและโฮสต์ ปลายทางเท่านั้น

เน็ตเวิร์ก (Network) เน็ตเวิร์คเป็นเครือข่ายที่มีการเชื่อมต่อกันของโฮส 2 ตัวขึ้นไป โฮสต์แต่ละตัวในเน็ตเวิร์คเดียวกันสามารถเชื่อมต่อถึงกันได้โดยตรง

เราเตอร์ (Router) ทำหน้าที่ในการ ส่งผ่านข้อมูลจากเน็ตเวิร์กหนึ่งไปยังอีกเน็ตเวิร์กหนึ่ ตำแหน่งของเราเตอร์จะอยู่ในจุดที่เชื่อมต่อระหว่างสองเน็ตเวิร์กเข้าด้วยกัน ด้วยข้อกำหนดของไอพี ข้อมูลจะส่งไปถึงกันโดยตรงข้ามเน็ตเวิร์กไม่ได้ จะต้องอาศัยเราเตอร์เป็นผู้ทำหน้าที่ส่งผ่านข้อมูลไปให้เราเตอร์จะมีเราท์ติ่งเทเบิล Routing Table สำหรับเก็บข้อมูล เพื่อใช้ในการพิจารณาเลือกเส้นทางในการส่งดาต้าแกรม ซึ่งจะอธิบายกลไกการทำงานในหัวข้อถัดไปในการอธิบายกระบวนการเราท์ติ่ง (Routing) ให้เป็นที่เข้าใจในเบื้องต้น จะขออธิบายจากเครือข่ายตัวอย่างที่แสดงในรูปที่ 2.18



รูปที่ 2.18 เนตเวิร์คตัวอย่าง

การเราท์ติ่งจะเป็นไปตามขั้นตอนดังนี้

1. ถ้าโฮสต์ต้นทางและปลายทางต่อเชื่อมร่วมอยู่ในเครือข่ายเดียวกัน มีการเชื่อมต่อถึงกันโดยตรง เช่น อีเธอร์เน็ต (Ethernet) หรือโทเค็นริง (Token ring)

2. หากโฮสต์ต้นทางและปลายทางไม่ได้อยู่ในเครือข่ายเดียวกัน ไอพีดาต้าแกรมจะถูกส่งไปยังดีฟอลต์เราท์เตอร์ (default router)

3. เมื่อเราเตอร์ได้รับไอพีดาต้าแกรมจากข้อ 2 แล้วตรวจสอบดู หากพบว่าโฮสต์ปลายทางต่อร่วมอยู่บนเครือข่ายเดียวกันกับเราเตอร์ ให้ทำการส่งดาต้าแกรมไปที่โฮสต์นั้น เช่น หาก 172.17.3.2 ต้องการส่งดาต้าแกรมไปยัง 172.17.4.2 จะต้องส่งดต้าแกรมไปที่ เราท์เตอร์ B และเราท์เตอร์ B จะส่งดาต้าแกรมต่อไปยังโฮสปลายทาง

4. หากไม่ได้ต่อร่วมกันก็ส่งดาต้าแกรมไปที่เราเตอร์ตัวต่อไป โดยเราท์เตอร์จะเป็นผู้เลือกเส้นทางซึ่งมีอยู่ 2 กรณีคือ

4.1. ถ้ามีข้อมูลของโฮสปลายทางอยู่ในเราท์ติ่งเทเบิลเราท์เตอร์จะส่งดาต้าแกรมไปยังเราท์เตอร์ตัวที่ระบุไว้ในเราท์ติ่งเทเบิล

4.2. ถ้าไม่มีข้อมูลของโฮสปลายทางอยู่ในเราท์ติ่งเทเบิล เราท์เตอร์จะส่งดาต้าแกรมไปยังดีฟอลต์

เราท์เตอร์และกลับไปที่ขั้นตอนในข้อ 3 ใหม่ จนกว่าไอพีแอดเดรสจะเดินทางถึงปลายทางหรือหมดเวลาในการส่ง (TTL=0)

สมมติว่าเครื่อง 172.17.1.3 ต้องการติดต่อกับ 172.17.4.3 จะต้องส่งไอพีดาต้าแกรมไปยังเราท์เตอร์ A หากเราท์เตอร์ A มีข้อมูลเกี่ยวกับ 172.17.4.3 อยู่ ก็จะรู้ว่าต้องส่งดาต้าแกรมไปยังเราท์เตอร์ B คือ 172.17.2.4 และเราท์เตอร์ B ก็จะส่งไอพีดาต้าแกรมไปยังโฮสปลายทางได้สำเร็จ

Subnet Addressing / Subnet Mask

ในการใช้งานโปรโตคอลทีซีพีไอพีในอินเตอร์เนตนั้นการไอพีแอดเดรสออกเป็นแอดเดรสของเน็ตเวิร์ก และแอดเดรสของโฮสต์ ตามที่ระบุของแต่ละคลาสค่อนข้างจะขาดประสิทธิภาพ คือในเครือข่ายคลาส A และ B แต่ละเครือข่ายนั้น สามารถมีจำนวนโฮสต์ได้มาก ซึ่งการที่จะนำไอพีแอดเดรส มาใช้อย่างทั่วถึงนั้นมีโอกาสเป็นไปได้ยากมากทั้งคลาส A และคลาส B เพราะมีโอกาสน้อยมากที่จะมีเครือข่ายใดในโลกมีจำนวนโฮสต์มากมายขนาดนั้นอยู่ภายในเครือข่ายเดียว ดังนั้นไอพีแอดเดรสที่จัดสรรให้ไปในแต่ละเครือข่ายของคลาสเหล่านี้จึงถูกใช้ไม่หมดและไม่สามารถนำไปใช้ประโยชน์ที่อื่นได้เลย ดังนั้นเพื่อให้การจัดสรรไอพีเป็นไปอย่างมีประสิทธิภาพ จึงมีการนำส่วนของแอดเดรสของโฮสต์ มาแบ่งย่อยเป็นสองส่วนคือ ซับเนตไอดี (subnet ID) และ โฮสต์ไอดี (host ID) ทำให้ได้เครือข่ายย่อยหลายๆเครือข่าย โดยในแต่ละเครือข่ายมีจำนวนโฮสไม่มากเกินไปและเพียงพอต่อการใช้งาน การแบ่งซับเนตใช้เทคนิคที่เรียกว่า ซับเนตมาร์ค (Subnet Mask) ซึ่งเป็นตัวเลขมีความยาว 32 บิต แบ่งออกเป็นสี่ชุดเช่นเดียวกับไอพี แต่ค่าของซับเนตมาร์คจะขึ้นอยู่กับความต้องการในการแบ่งซับเนตว่าต้องการจำนวนซับเนตเท่าใดและมีจำนวนโฮสเท่าใด หากนำซับเนตมาร์คมาเขียนเป็นเลขฐานสอง จะมีลักษณะพิเศษคือ ขึ้นต้นด้วยเลข 1 มีจำนวนกี่ตัวก็ได้ ตามแต่ความต้องการในการแบ่งซับเนตและตำแหน่งที่เหลือจะมีค่าเป็น 0 ความสัมพันธ์ระหว่างไอพีแอดเดรส, ซับเนตมาร์ค, โฮสต์ไอดี , ซับเนตไอดี, จำนวนซับเนตและจำนวนโฮสต์จะเป็นดังนี้

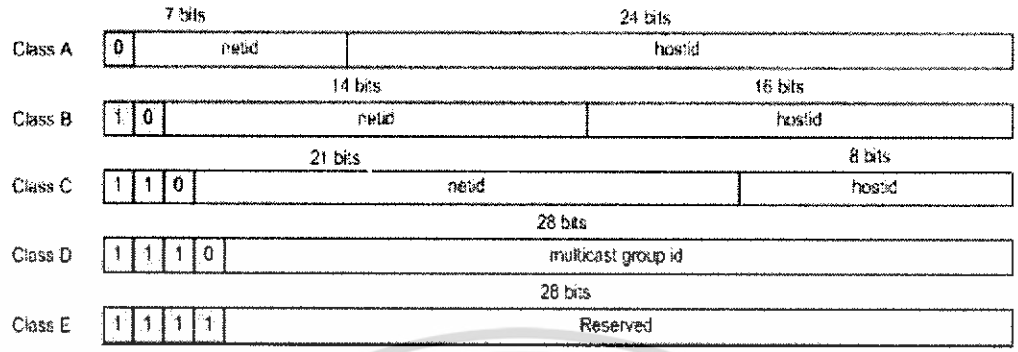
Host ID = (IP Address) AND ~ (subnet mask)

Net ID = (IP address) AND (subnet mask)

จำนวนโฮสต์ = ((2^จำนวนบิตที่เป็น 0 ของซับเนตมาร์ค)-2)

เนื่องจากไอพีแรกของซับเนตถูกใช้เป็นเนตไอพีและ ไอพีสุดท้ายของซับเนตถูกใช้เป็นบรอสแคสไอดี (broadcast ID)

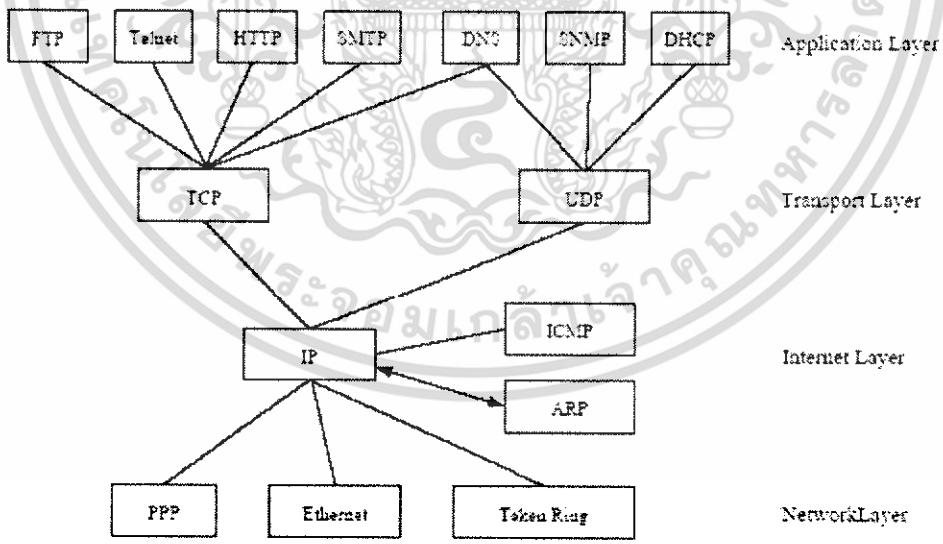
จำนวนซับเนต = (2^ (จำนวนบิตที่เป็น 1 ของซับเนตมาร์คในตำแหน่งที่เป็นโฮตส์ไอดีของไอพีแอดเดรส))



รูปที่ 2.19 ซับเนตแอดเดรสซิ่งและซับเนตมาร์ค

2.2.4 Network Access Layer

การทำงานระดับล่างสุดต่อจากชั้นอินเตอร์เนตจะเป็นการแปลงข้อมูลไอพีดาต้าแกรมให้อยู่ในรูปที่เหมาะสม และแปลงเป็นสัญญาณไฟฟ้าส่งไปยังเครือข่ายต่อไปซึ่งในชั้นเนตเวิร์คแอคเซสนี้เมื่อเทียบกับมาตรฐานโอเอสไอโมเดลแล้วจะเป็นการรวม 2 layer เข้าด้วยกันคือ ชั้นดาต้าลิงค์ และชั้นฟิสิกส์คอล กล่าวโดยสรุปคือการทำงานในชั้นต่างๆตามโครงสร้างของโปรโตคอลทีซีพีไอพีมีลักษณะแสดงดังรูปที่ 2.20



รูปที่ 2.20 แสดงการติดต่อระหว่างชั้นแอพพลิเคชั่นถึงชั้นเนตเวิร์ค

2.4 Server-Client และ Peer-To-Peer Architecture

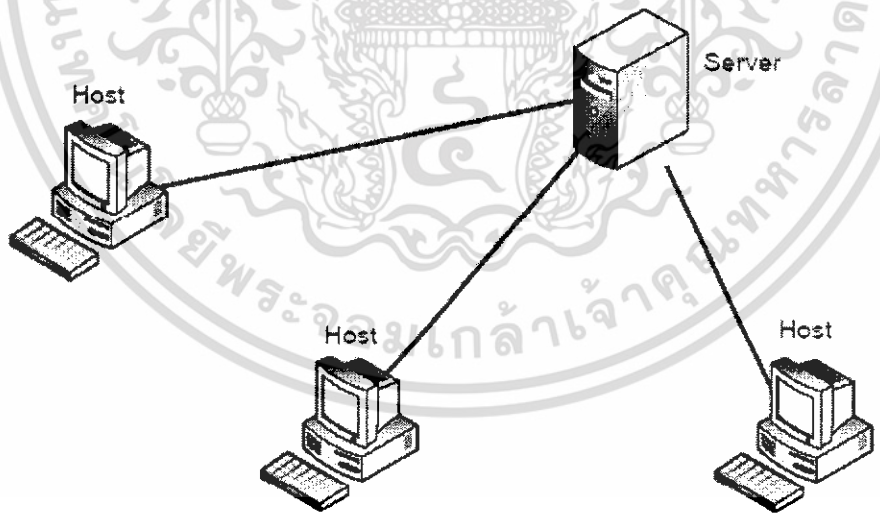
2.4.1 Peer-To-Peer Architecture

เป็นโครงสร้างแบบโฮสตัวหนึ่งติดต่อกับโฮสตัวอื่นๆโดยตรงโดยไม่ผ่าน และในอีกความหมายหนึ่งคือ เป็นการติดต่อระหว่างชั้นเดียวกันของโฮส 2 ตัวขึ้นไป

รูปที่ 2.21 สถาปัตยกรรม Peers-To-Peer

2.4.2 Server-Client Architecture

มีลักษณะคล้ายแบบ Peer-To-Peer แต่จะมีโฮสอย่างน้อย 1 ตัวเป็นกลางในการติดต่อสื่อสาร และมีโฮสอื่นๆ เป็นผู้ติดต่อเข้ามา เช่น Website เป็นต้น



รูปที่ 2.22 สถาปัตยกรรม Server-Client

2.5 คุณภาพเสียงและการบีบอัดเสียง

แซมพลิ้งเรท (Sampling rate) คือ อัตราการสุ่มข้อมูลให้อยู่ในรูปแบบของบิต (bit) และไบต์ (byte) ส่วนจำนวนข้อมูลที่ได้เรียกว่า แซมพลิ้งไซด์ (sampling size) คุณภาพของเสียงขึ้นอยู่แซมพลิ้งเรทและแซมพลิ้งไซด์ โดยทั่วไปแซมพลิ้งเรทและแซมพลิ้งไซด์ที่สูงกว่าจะส่งผลให้คุณภาพของเสียงดีกว่าและจะต้องมีเนื้อที่บนฮาร์ดดิสก์สำหรับรองรับอย่างเหมาะสม

การคำนวณขนาดของไฟล์เสียงโมโนโฟนิค monophonic

ขนาดไฟล์ = แซมพลิ้งเรท*เวลาในการบันทึก*(bit resolution/8)*1

เช่น เสียงที่ระดับความถี่ 22.05 kHz ความละเอียด 8 บิต บันทึกยาว 10 วินาที

จะมีขนาดไฟล์= 22050*10*(8/8)*1 ไบต์

= 220,500 ไบต์

= 0.2205 MB (220,500/1,000,000)

การคำนวณขนาดของไฟล์เสียงสเตอริโอ (stereo)

ขนาดไฟล์ = แซมพลิ้งเรท*เวลาในการบันทึก*(bit resolution/8)*2

เช่น เสียงที่ระดับความถี่ 22.05 kHz ความละเอียด 8 บิต บันทึกยาว 10 วินาที

จะมีขนาดไฟล์ = 22050*10*(8/8)*2 ไบต์

= 441,000 ไบต์ = 0.441 MB

ขนาดของไฟล์สำหรับเสียงแบบดิจิตอลในเวลา1 นาที

sampling rate (kHz)	แซมพลิ้งไซด์ (บิท)	สเตอริโอ / โมโน	ขนาดของไฟล์
44.1	16	สเตอริโอ	10.584 MB
44.1	16	โมโน	5.292 MB
44.1	8	โมโน	2.646 MB
22.05	16	สเตอริโอ	5.292 MB
22.05	8	สเตอริโอ	2.646 MB
11.025	16	โมโน	1.323 MB
11.025	8	โมโน	661.5 KB

ตารางที่ 2.2 แสดงความสัมพันธ์ระหว่างแซมพลิ้งเรท, แซมพลิ้งไซด์ และ ขนาดของไฟล์

2.5.1 การบันทึกข้อมูลเสียง

เป็นการนำเสียงจากการพูด การเล่นเครื่องดนตรี หรือจากแหล่งเสียงต่าง ๆ มาจัดเก็บลงในไฟล์ เพื่อนำไปใช้งานต่อไป

- synthesize sound เกิดจากตัววิเคราะห์เสียง โดยจะต้องมี synthesize chip หรือ wave table เพื่อทำการแยกเสียงว่าเป็นเสียงดนตรีชนิดใด

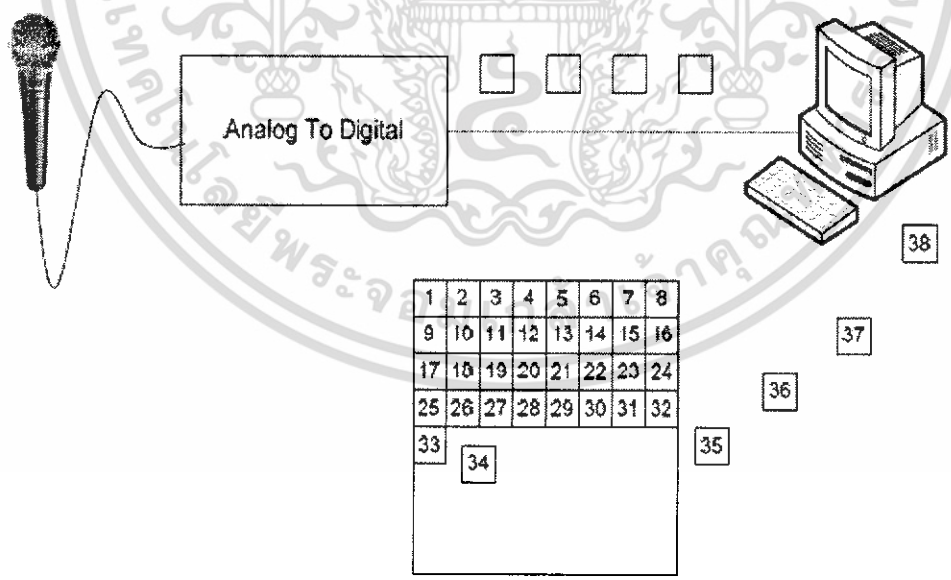
-sound data เป็นการแปลงสัญญาณอนาลอกเป็น สัญญาณดิจิตอล โดยจะเป็นการบันทึกคลื่นเสียง

- ไฟล์ .wav มีการใช้งานส่วนมากอยู่บนระบบเครือข่าย

บทที่ 3 หลักการออกแบบ

3.1 ส่วนบันทึกเสียง

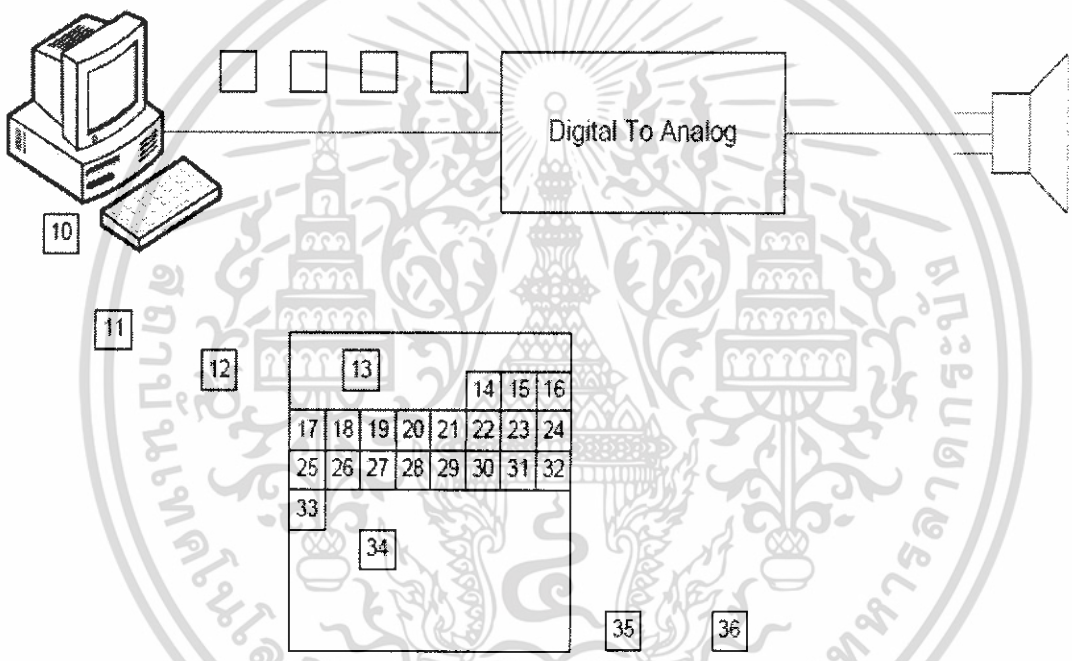
ในส่วนของการบันทึกเสียงจะเริ่มจาก สัญญาณเสียงผ่านซาวด์การ์ดซึ่งเป็นอุปกรณ์แปลงสัญญาณ อนาลอกเป็นสัญญาณดิจิตอล ขั้นตอนนี้จะต้องมีการระบุรูปแบบการเอ็นโค้ด (Encode) สัญญาณเสียง อัตราการสุ่ม (Sample Rate) ขนาดสัญญาณที่จะสุ่ม (sample Size in Bits) ชั่นแนล (channels) ขนาดของข้อมูล (frame Size) จำนวนเฟรมต่อวินาที (frame Rate) และขนาดบัฟเฟอร์ (bigEndian) ในที่นี้ได้กำหนดรูปแบบการเอ็นโค้ดเป็นแบบพีซีเอ็มซายด์ (signed linear PCM) อัตราการสุ่ม 8000 กิโลเฮิร์ต (kHz) ขนาดสัญญาณที่จะสุ่ม 8 บิต ชั่นแนลเป็น 2 หมายถึงเป็นแบบสเตอร์ริโอขนาดของข้อมูล 2 ไบต์ จำนวนเฟรมต่อวินาทีเท่ากับ 8000 บัฟเฟอร์มีขนาดใหญ่ จากนั้นจะกำหนดรูปแบบไฟล์ที่จะบันทึก ในที่นี้กำหนดเป็นแบบ WAVE แล้วเริ่มทำการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลตามรูปแบบที่กำหนด และการบันทึกสัญญาณดิจิตอลลงเป็นไฟล์ตามที่กำหนดกระบวนการนี้กระทำอย่างต่อเนื่องควบคู่ไปกับกระบวนการอื่นๆ เช่น กระบวนการส่งข้อมูลเสียงเป็นต้น ก่อนที่จะปิดโปรแกรมหรือหยุดการติดต่อแบบสัญญาณเสียง โปรแกรมจะสั่งหยุดรับสัญญาณอนาลอกแล้วจะจะปิดไฟล์เสียงนั้น



รูปที่ 3.1 แสดงสถาปัตยกรรมในส่วนการบันทึกเสียง

3.2 ส่วนเล่นเสียง

ในส่วนของการเล่นเสียงจะเริ่มจากการอ่านไฟล์บันทึกจากข้อมูลที่ส่งผ่านอินเตอร์เนตมายังเครื่องปลายทาง ตรวจสอบไฟล์นั้นว่าเป็นไฟล์เสียงรูปแบบใด กำหนดรูปแบบเสียงให้กับตัวแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก ในที่นี้ใช้ซาวด์การ์ด กำหนดขนาดบัฟเฟอร์ที่จะอ่านไฟล์นั้นแล้วจึงเริ่มอ่านไฟล์ข้อมูลลงในบัฟเฟอร์ แล้วนำข้อมูลในบัฟเฟอร์ไปแสดงผลเป็นเสียงอ่านไฟล์ไปเรื่อยๆจนกว่าจะพบจุดสิ้นสุดของไฟล์นั้น (End of file) จึงปิดการทำงานของตัวแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล



รูปที่ 3.2 แสดงสถาปัตยกรรมในส่วนการเล่นเสียง

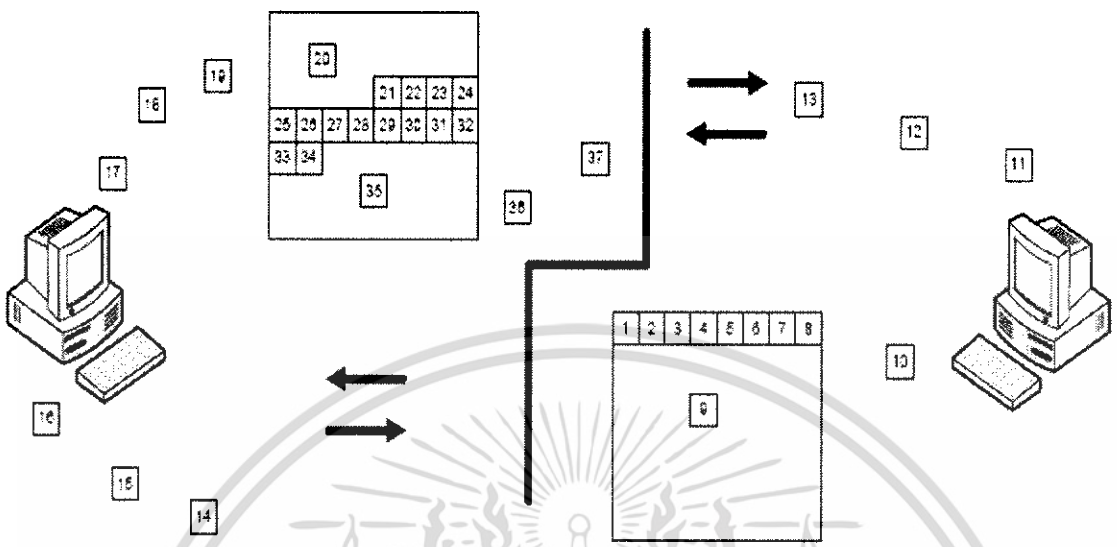
3.3 ส่วนติดต่อสื่อสาร

ส่วนการติดต่อสื่อสารจะแบ่งเป็น 5 เส้นทางได้แก่

1. เส้นทางสำหรับส่งคำสั่งบอกสถานะการทำงานของแต่ละฝ่ายใช้ควบคุมการติดต่อข้อมูลรูปแบบเสียง และควบคุมการติดต่อรูปแบบตัวอักษร

2. เส้นทางสำหรับส่งข้อมูลรูปแบบเสียง เส้นทางนี้ระบบปฏิบัติการ (Operation System) จะเป็นผู้เลือกว่าจะเปิดพอร์ทที่เรียกกันว่า ไคลแอนต์พอร์ท แล้วส่งข้อมูลออกไป

3. เส้นทางสำหรับรับข้อมูลรูปแบบเสียงเส้นทางนี้ผู้เขียนโปรแกรมเป็นผู้กำหนดช่องทางนี้



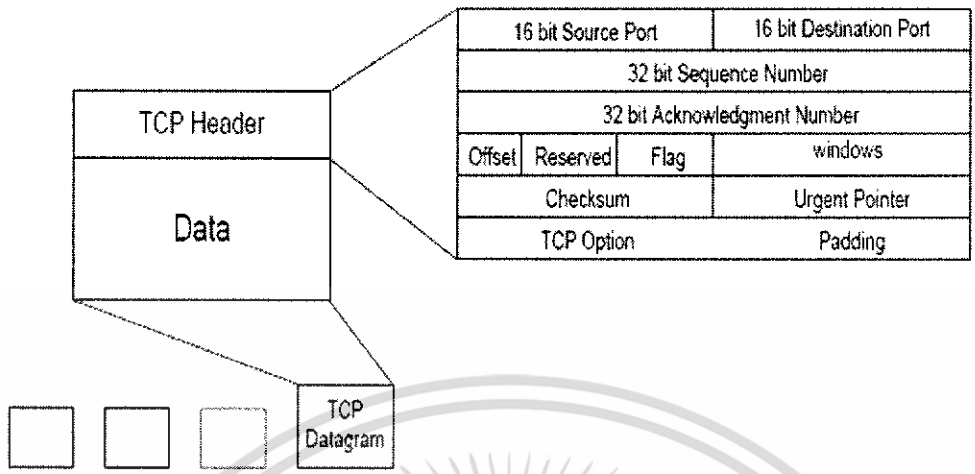
รูปที่ 3.3 แสดงสถาปัตยกรรมในส่วนของการติดต่อสื่อสารของโปรแกรม

3.3.1 ส่วนควบคุมการติดต่อสื่อสาร

ในส่วนนี้จะสร้างไว้สำหรับการควบคุมการติดต่อสื่อสารทั้งหมด เช่น การบอกไอพีแอดเดรสผู้ส่งและไอพีแอดเดรสผู้รับ โหมดในการติดต่อ (แบบเสียง/ตัวอักษร) การแสดงผลเป็นตัวอักษรหรือเสียง เป็นต้น เมื่อเริ่มโปรแกรมช่องทางนี้จะเปิดรอรับการติดต่อเข้ามา หากต้องการติดต่อออกไป ระบบปฏิบัติการก็จะเปิดไคลแอนด์พอร์ตเพื่อติดต่อไปยังเครื่องที่ผู้ใช้ต้องการ

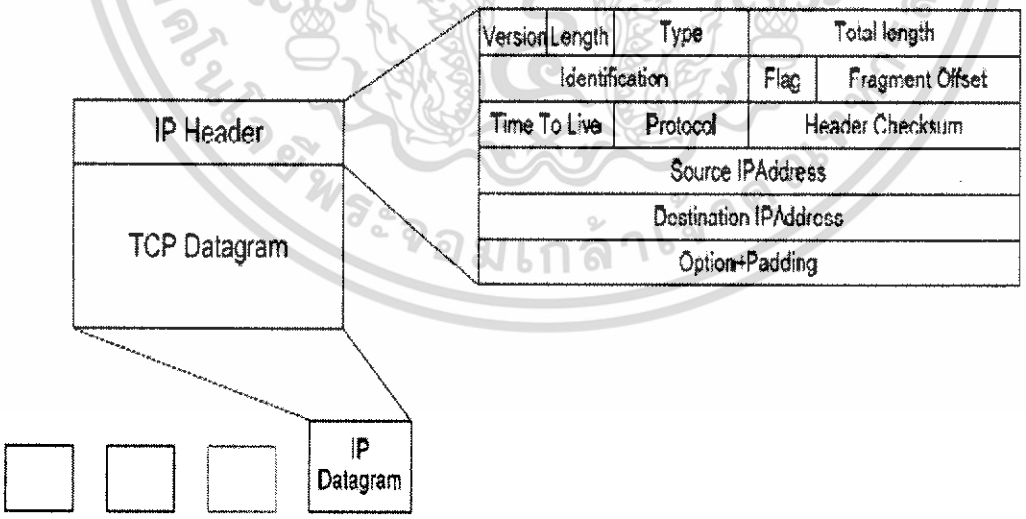
3.3.2 ส่วนส่งข้อมูลอักษร

ในส่วนนี้สร้างไว้สำหรับส่งข้อมูลแบบตัวอักษร กระบวนการส่งมีดังนี้ เมื่อมีการสร้างช่องทางไคลแอนด์พอร์ทแล้ว หลังจากนั้นจะอ่านข้อมูลจากช่องที่ไว้สำหรับเติมตัวอักษรนำมาเก็บไว้ในบัฟเฟอร์เป็นข้อมูลก้อนหนึ่ง แล้วเข้าสู่กระบวนการส่งในชั้นทรานสพอร์ท โดยจะข้อมูลเป็นชิ้นเล็ก ๆ ขนาด 640 บิท แล้วนำหัวข้อมูลซึ่งบรรจุเซิร์ฟเวอร์พอร์ทและไคลแอนด์พอร์ท ลำดับของชิ้นข้อมูล (Sequence Number) ลำดับในการตอบรับข้อมูล (Acknowledgment Number) ไบต์ตรวจสอบความผิดพลาดของข้อมูล และตัวแปรอื่นๆ ดังรูปที่ 3.4



รูปที่ 3.4 แสดงรายละเอียดของทีซีพีดาต้าแกรม

หัวข้อมูลชั้นทรานสพอร์ทรวมกับข้อมูลจะเรียกว่า ทีซีพีดาต้าแกรม แล้วส่งต่อไปยังชั้นอินเตอร์เนต เพื่อใส่หัวข้อมูลในชั้นนี้ ในหัวข้อมูลชั้นนี้จะบรรจุไอพีแอดเดรสต้นทางและไอพีแอดเดรสปลายทาง ความยาวของข้อมูล ลำดับของชิ้นข้อมูล และตัวแปรอื่นๆดังรูปที่ 3.5 ต่อจากชั้นอินเตอร์เนตจะถูกส่งไปยังชั้นเนตเวิร์คซึ่งจะแปลงข้อมูลเป็นสัญญาณดิจิตอลส่งออกไปยังสายนำสัญญาณไปยังเครื่องปลายทาง



รูปที่ 3.5 แสดงรายละเอียดของไอพีดาต้าแกรม

3.3.3 ส่วนรับข้อมูลอักษร

เมื่อมีสัญญาณดิจิตอลเข้ามาชั้นเนตเวิร์คจะแปลงสัญญาณและส่งต่อมายังชั้นอินเตอร์เนต ในชั้นนี้จะแยกหัวข้อมูลมาตรวจสอบความถูกต้องของหัวข้อมูล ตรวจสอบไอพีแอดเดรสต้นทางที่ส่งมาและไอพีแอดเดรสปลายทางว่าถูกต้องหรือไม่ หากไม่ถูกต้องจะทิ้งข้อมูลนั้นไป หากถูกต้องจะส่งต่อไปในชั้นทรานสพอร์ท ในระดับชั้นนี้ จะตรวจสอบหัวข้อมูลเพื่อจัดลำดับข้อมูลให้ถูกต้อง หากมาข้อมูลลำดับใดขาดหายไป จะส่งคำร้องขอข้อมูลส่วนที่ขาดไปนั้นกลับไปยังต้นทางที่ส่งข้อมูล เพื่อให้ส่งข้อมูลชั้นนั้นซ้ำอีกครั้ง เมื่อเรียงข้อมูลเสร็จก็จะเป็นข้อความในลำดับที่ถูกต้องและจะนำไปแสดงผลออกทางหน้าโปรแกรม

3.3.4 ส่วนส่งข้อมูลเสียง

ในส่วนนี้จะเริ่มจากการกำหนดรูปแบบไฟล์เสียงที่จะเปิดอ่าน เปิดอ่านไฟล์เสียงที่บันทึกไว้แบ่งข้อมูลเป็นชั้นย่อยๆใส่ไว้ในบัฟเฟอร์ขนาด 2048 ไบต์ แล้วส่งไปยังชั้นทรานสพอร์ทเพื่อทำการส่งข้อมูลตามกระบานการการส่งข้อมูลแบบตัวอักษร

3.3.5 ส่วนรับข้อมูลเสียง

ในส่วนนี้จะกระทำคล้ายกับกระบวนการรับข้อมูลแบบตัวอักษร แต่หลังจากได้ข้อมูลแล้วจะเก็บไว้ในบัฟเฟอร์ขนาด 2048 ไบต์ กำหนดรูปแบบไฟล์เสียงที่จะบันทึกข้อมูล แล้วทำการบันทึกลงไป

บทที่ 4 ผลการทดสอบ

ในการทดสอบ ได้ทำการทดสอบโดยใช้เครื่องคอมพิวเตอร์จำนวน 2 เครื่อง เชื่อมต่อกันเป็นระบบเครือข่าย

รายละเอียดของอุปการที่ใช้ในการทดสอบมีดังนี้

1. คอมพิวเตอร์เครื่องที่ 1

• ตัวประมวลผล (CPU) รุ่น Intel Pentium M Processor 1.60 GHz.

• แรม (RAM) ขนาด 760 MB.

• การ์ดเสียง (Sound Card) รุ่น SoundMAX Integrated Digital Audio

• การ์ดเครือข่าย (LAN Card) รุ่น Intel® Pro/1000 MT Mobile Connection

• ระบบปฏิบัติการ Windows XP Professional Service Pack 2

• ค่าอินเตอร์เน็ตโปรโตคอล 161.246.34.181

2. คอมพิวเตอร์เครื่องที่ 2

• ตัวประมวลผล (CPU) รุ่น Pentium4 1.7 GHz.

• แรม (RAM) ขนาด 256 MB.

• การ์ดเสียง (Sound Card) รุ่น CMI8738/C3DX PCI Audio Device

• การ์ดเครือข่าย (LAN Card) รุ่น SMC EZ Card 10/100 PCI (SMC1211TX)

• ระบบปฏิบัติการ Windows XP Professional Service Pack2

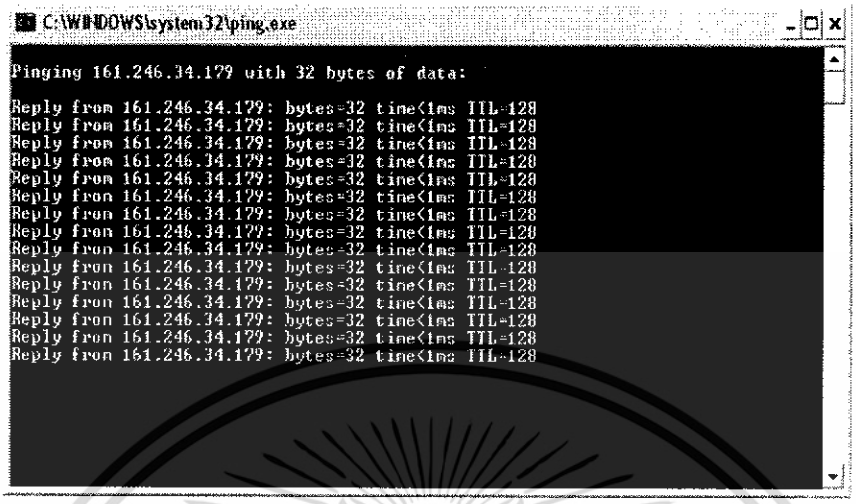
• ค่าอินเตอร์เน็ตโปรโตคอล 161.246.34.179

3. สายแลนเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์

4.1 การทดสอบความถูกต้องของช่องทางในการติดต่อสื่อสาร

ขั้นตอนนี้จะทำการตรวจสอบว่าเครื่องคอมพิวเตอร์ทั้งสองเชื่อมต่อกันเป็นระบบเครือข่ายและส่งข้อมูลไปกับระหว่างระบบเครือข่ายได้หรือไม่

1. ตรวจสอบเครื่องคอมพิวเตอร์เครื่องที่ 1 ส่งข้อมูลไปยังเครื่องที่ 2 โดยใช้คำสั่ง Ping161.246.34.179 –t แสดงดังรูปที่ 4.1



รูปที่ 4.1 ทดสอบการส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์เครื่องที่ 1 ไปยังเครื่องที่ 2

2. ตรวจสอบเครื่องคอมพิวเตอร์เครื่องที่ 2 ส่งข้อมูลไปยังเครื่องที่ 1 โดยใช้คำสั่ง Ping161.246.34.181 –t แสดงดังรูปที่ 4.2

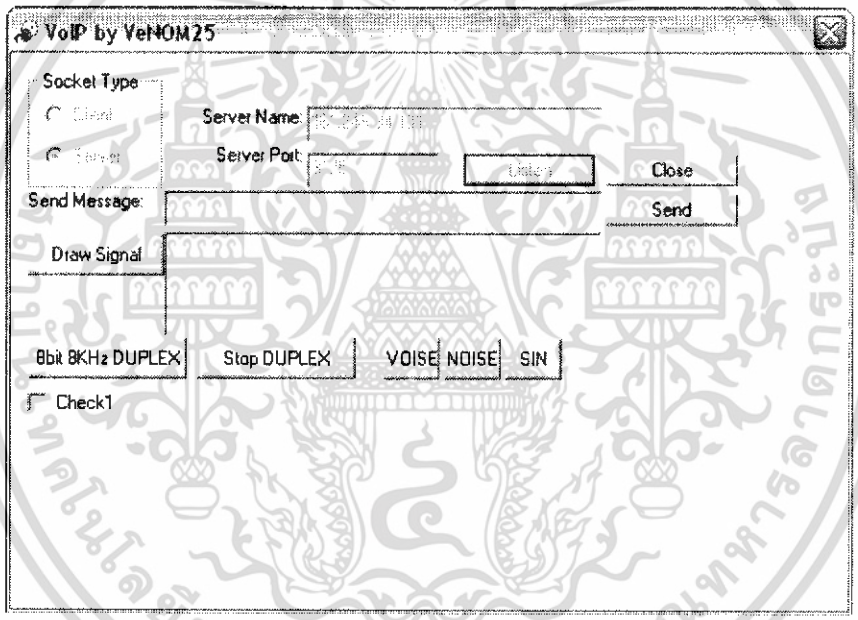
รูปที่ 4.2 ทดสอบการส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์เครื่องที่ 1 ไปยังเครื่องที่ 2

ผลการทดสอบสรุปได้ว่าเครื่องคอมพิวเตอร์ทั้งสองสามารถส่งข้อมูลไปกลับระหว่างกันได้ หากติดต่อส่งข้อมูลไม่ได้หลังจากเมื่อใช้คำสั่ง Ping แล้วจะได้ผลที่หน้าจอคอมพิวเตอร์แตกต่างจากรูปที่ 4.1 และรูปที่ 4.2

4.2 การทดสอบการรับส่งข้อมูลเสียง

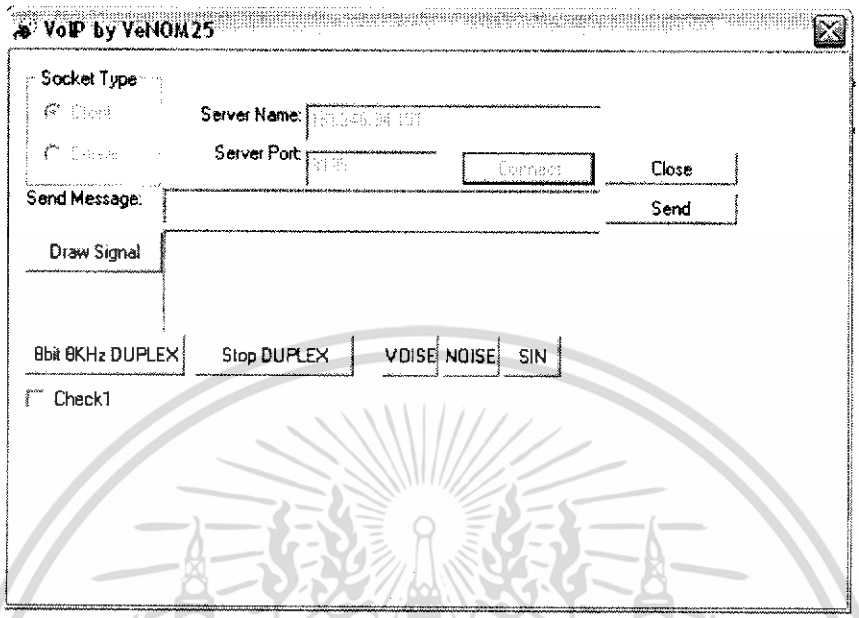
เมื่อตรวจสอบแน่ใจแล้วว่าเครื่องคอมพิวเตอร์เชื่อมต่อกันอยู่ในระบบเครือข่ายและสามารถส่งข้อมูลติดต่อสื่อสารระหว่างกันได้ จากนั้นก็จะให้โปรแกรมที่ได้ทำการพัฒนาขึ้นมาทำการส่งข้อมูลที่เป็นเสียงระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่องดังนี้

1. เปิดโปรแกรม VoIP By VeNOM25 ในเครื่องคอมพิวเตอร์เครื่องที่ 1 ขึ้นมาจากนั้นกำหนดค่าอินเตอร์เน็ตโปรโตคอลเป็น 161.246.34.181 ลงในช่อง Server Name กำหนดค่า 3125 ลงในช่อง Server Port ที่หัวข้อ Socket Type ให้ใช้เมาส์คลิกเลือกเป็น Server จากนั้น คลิกปุ่ม Listen แสดงดังรูปที่ 4.3



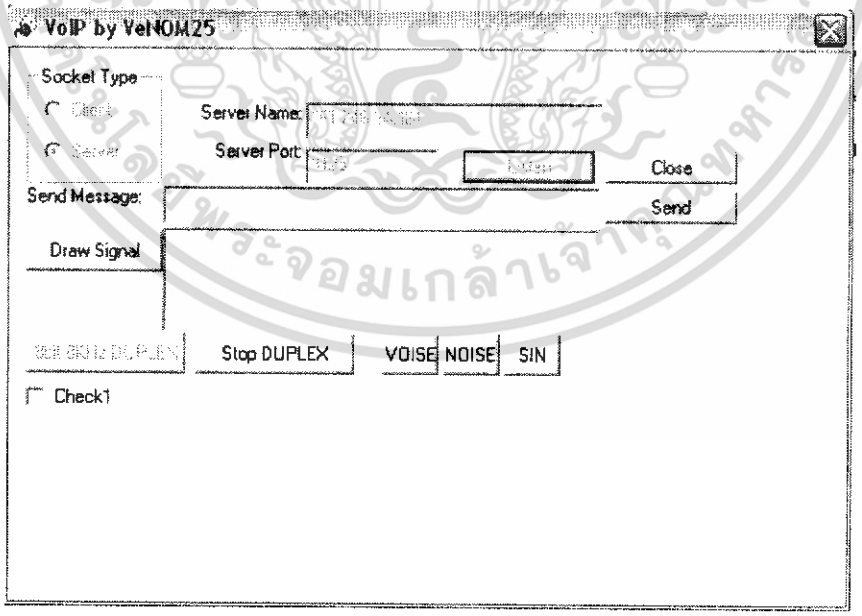
รูปที่ 4.3 เครื่องคอมพิวเตอร์เครื่องที่ 1 เปิดโปรแกรมเป็นฝ่าย Server

2. เปิดโปรแกรม VoIP By VeNOM25 ในเครื่องคอมพิวเตอร์เครื่องที่ 2 ขึ้นมาจากนั้นกำหนดค่าอินเตอร์เน็ตโปรโตคอลเป็น 161.246.34.181 ลงในช่อง Server Name กำหนดค่า 3125 ลงในช่อง Server Port ที่หัวข้อ Socket Type ให้ใช้เมาส์คลิกเลือกเป็น Client จากนั้น คลิกปุ่ม Connect แสดงดังรูปที่ 4.4



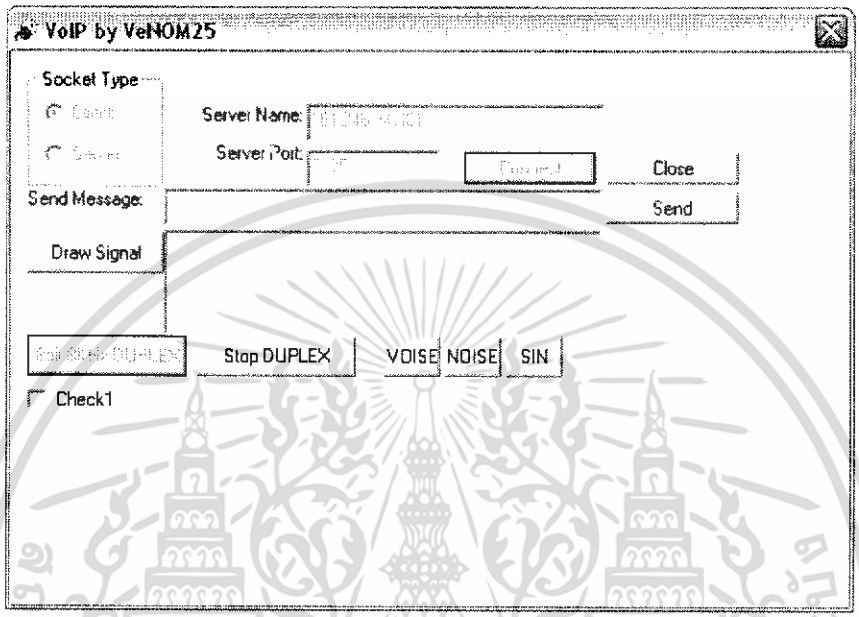
รูปที่ 4.4 เครื่องคอมพิวเตอร์เครื่องที่ 2 เปิดโปรแกรมเป็นฝ่าย Client

3. เครื่องคอมพิวเตอร์เครื่องที่ 1 ใช้เมาส์คลิกปุ่ม 8bit 8KHz DUPLEX เพื่อเริ่มต้นติดต่อสื่อสารทางเสียงผ่านเครือข่ายอินเตอร์เน็ตโปรดตคอล แสดงดังรูปที่ 4.5



รูปที่ 4.5 เครื่องคอมพิวเตอร์เครื่องที่ 1 เริ่มต้องการสื่อสารทางเสียง

4. เครื่องคอมพิวเตอร์เครื่องที่ 2 ใช้เมาส์คลิกปุ่ม 8bit 8KHz DUPLEX เพื่อเริ่มต้นติดต่อสื่อสราทางเสียงผ่านเครือข่ายอินเตอร์เน็ตโปรดตคอล แสดงดังรูปที่ 4.6

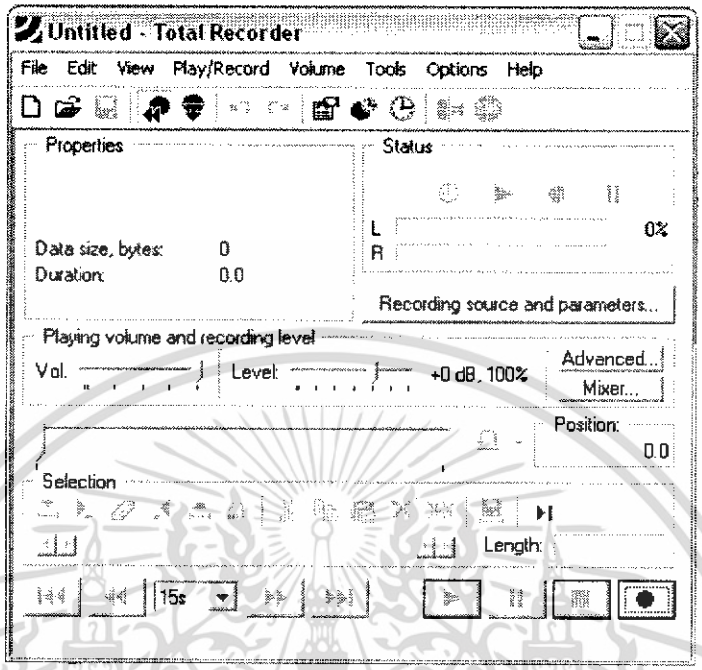


รูปที่ 4.6 เครื่องคอมพิวเตอร์เครื่องที่ 2 เริ่มต้องการสื่อสารทางเสียง

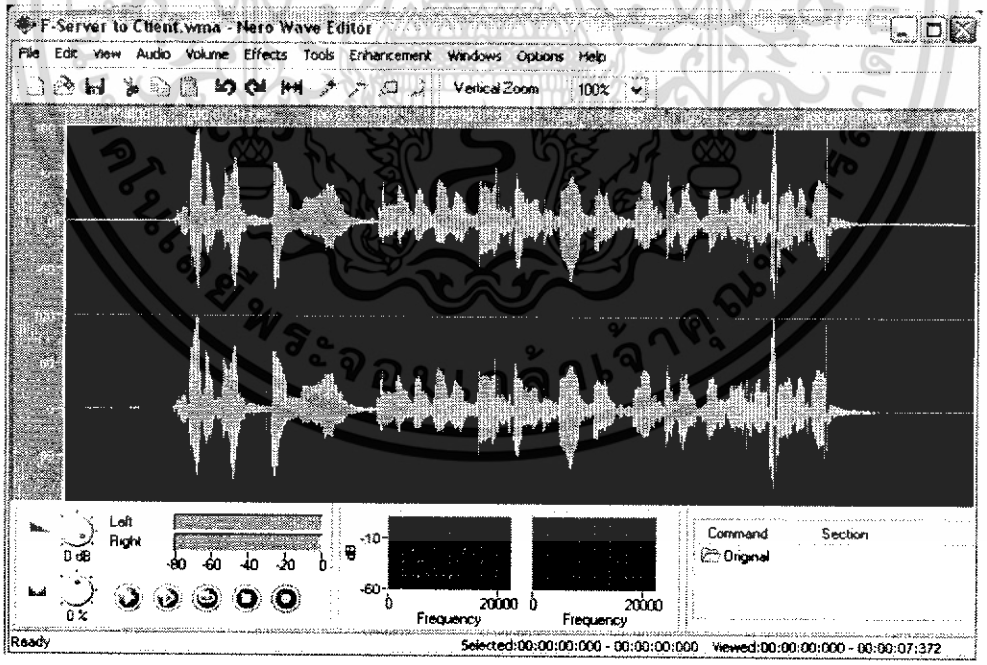
5. ใช้ไมโครโฟนสื่อสารกันระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่อง

4.3 การทดสอบไฟล์เสียง

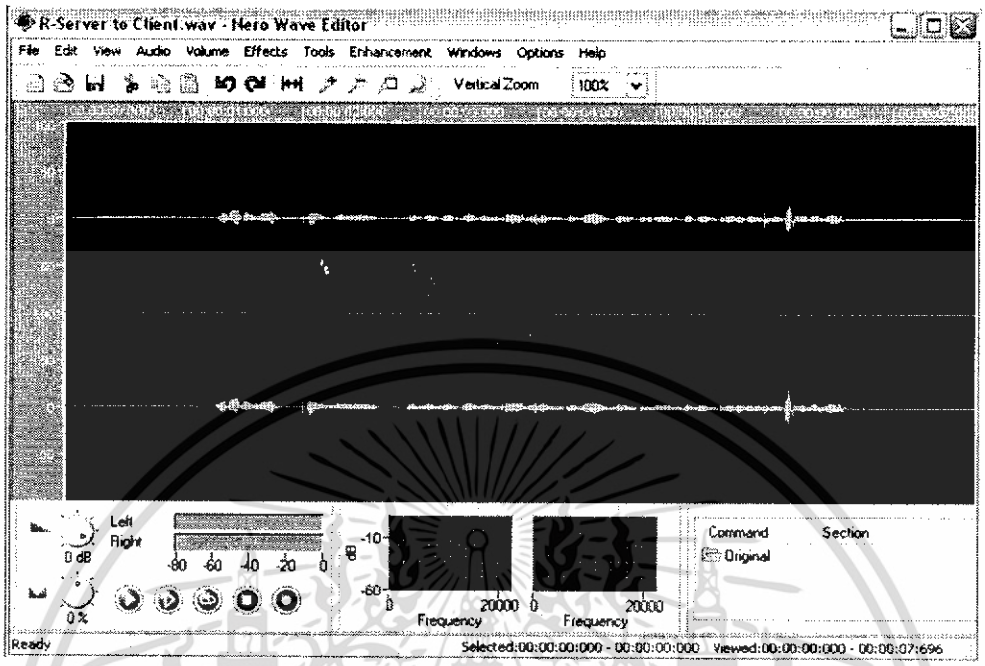
ในระหว่างการสื่อสารได้ใช้โปรแกรม ชื่อ Total Recorder แสดงดังรูปที่ 4.7 บันทึกเสียงระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่องไว้ จากนั้นนำไฟล์เสียงที่บันทึกไว้มาเปิดวิเคราะห์โดยใช้โปรแกรม Nero Wave Editor รูปวิเคราะห์ไฟล์เสียงของเครื่องคอมพิวเตอร์เครื่องที่ 1 แสดงดังรูปที่ 4.8 และรูปวิเคราะห์ไฟล์เสียงของเครื่องคอมพิวเตอร์เครื่องที่ 2 แสดงดังรูปที่ 4.9



รูปที่ 4.7 โปรแกรมบันทึกเสียง

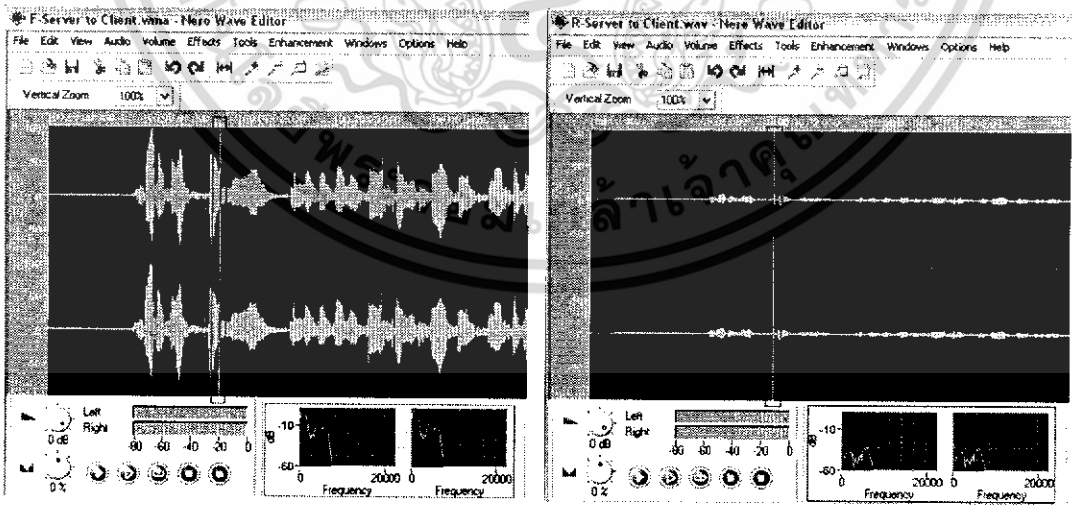


รูปที่ 4.8 ไฟล์เสียงของเครื่องคอมพิวเตอร์เครื่องที่ 1



รูปที่ 4.9 ไฟล์เสียงของเครื่องคอมพิวเตอร์เครื่องที่ 2

จากไฟล์เสียงที่ได้บันทึกไว้จะนำค่าอัตราขยายหน่วย dB ของทั้งสองไฟล์มาเปรียบเทียบกันจากภาพทั้งสองจะเห็นว่าช่วงความถี่ของสัญญาณเสียงทุกช่วงมีค่าเท่ากันแต่หากว่าความแรงของสัญญาณมีค่าลดลง แสดงการเปรียบเทียบดังรูปที่ 4.10



รูปที่ 4.10 เปรียบเทียบไฟล์เสียงของเครื่องคอมพิวเตอร์ทั้ง 2 เครื่อง

บทที่ 5 วิจารณ์ สรุป

จากผลการทดสอบที่หัวข้อ 4.1 จะเป็นการตรวจสอบการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่อง เพื่อต้องการทราบให้แน่ชัดว่าทั้งสองเครื่องสามารถติดต่อสื่อสารส่งข้อมูลระหว่างกันได้จริง

จากผลการทดสอบที่หัวข้อ 4.2 จะเป็นการทดสอบส่งข้อมูลเสียงติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์เครื่องที่ 1 (เครื่องเซิร์ฟเวอร์) กับเครื่องคอมพิวเตอร์เครื่องที่ 2 (เครื่องไคล์เอน) โดยใช้โปรแกรมที่พัฒนาขึ้นมาเองจากภาษา C โดยใช้ Visual C++ 6.0 (โปรแกรม VoIP By VeNOM25) จะพบว่าเครื่องคอมพิวเตอร์ทั้งสองเครื่องนั้นสามารถติดต่อสื่อสารกันโดยส่งข้อมูลเสียงได้จริงและเป็นแบบการสื่อสารสองทาง แต่ข้อมูลเสียงที่ส่งไปจากเครื่องคอมพิวเตอร์เครื่องที่ 1 ไปยังเครื่องคอมพิวเตอร์เครื่องที่ 2 นั้นจะมีสัญญาณรบกวนบ้างเล็กน้อยแต่ข้อมูลเสียงก็ยังมีอัตราความถี่เท่าเดิม เมื่อถอดรหัสข้อมูลเสียงออกมาแล้วก็จะแปลเป็นความหมายเดียวกัน

จากผลการทดสอบที่หัวข้อ 4.3 จะเป็นการบันทึกข้อมูลเสียงของเครื่องคอมพิวเตอร์ทั้งสองเครื่อง ในระหว่างที่สื่อสารกันแล้วนำตัวเลขอัตราขยายในหน่วยเดซิเบล (dB) มาเปรียบเทียบกัน ผลลัพธ์ที่ได้จะทำให้ทราบว่าค่าอัตราขยายในหน่วยเดวิเบล (เสียง) ของเครื่องคอมพิวเตอร์เครื่องที่ 1 จะมีค่ามากกว่าค่าอัตราขยายในหน่วยเดซิเบล (เสียง) ของเครื่องคอมพิวเตอร์เครื่องที่ 2 เพราะความผิดพลาดที่เกิดขึ้นอาจเกิดมาจากการเขียนโปรแกรมของผู้ทำการทดลองในเรื่องการจองพื้นที่หน่วยความจำของค่าเสียงที่จะบันทึก

จากผลการทดสอบในทุกหัวข้อที่กล่าวมานั้น ผลการทดลองที่ได้จากการทดสอบเป็นที่น่าพึงพอใจมากสำหรับการศึกษาในเรื่องการสื่อสารทางเสียงผ่านเครือข่ายไอพี เพราะสามารถติดต่อสื่อสารโดยส่งข้อมูลเสียงระหว่างเครื่องคอมพิวเตอร์ทั้งสองเครื่องได้จริง ซึ่งคุณภาพของข้อมูลเสียงที่ได้จากการบันทึกแล้วนำมาเปรียบเทียบกันจะได้ค่าความถี่เดียวกันต่างกันเพียงค่าอัตราขยายเท่านั้น

หนังสืออ้างอิง

ดร.สนั่น ศรีสุข. การเขียนโปรแกรมติดต่อสื่อสารบนระบบอินเตอร์เน็ตโดยใช้ Winsock [Online]. URL: Http://www.eng.mut.ac.th/upload_file/article/41.pdf

ยุธนา ลีลาศวัฒนกุล. คู่มือการเขียนโปรแกรมและใช้งาน Visual C++ 6.0 [Online]. URL: Http://www.infopress2000.com

รศ.สุรศักดิ์ สงวนพงษ์. Transmission ControlProtocol [Online]. URL:Http://www.cpe.ku.ac.th/~nguan/presentations/network/tcp1.pdf

Jarnes Peters. Voice over IP Fundamentals. 2000.

[Online]. URL: http://realdev.truehits.net/tcpip/charpter1.php

[Online]. URL: http://www.sut.ac.th/ccs/news/article/article004.asp

[Online]. URL: http://kanchanapisek.or.th/kp6/BOOK25/chapter6/t25-6-l1.htm

[Online]. URL: http://datacom.siamu.ac.th/index.html

[Online]. URL: http://www.ku.ac.th/magazine_online/voip.html

ภาคผนวก

โปรแกรมย่อยของการส่งข้อมูลเสียง

```
void CFULL_DUPLEXDlg::OnB8bit8khz()
{
        g_pDuplex->SetData();
        FillMemory(audio,8000,0);
        g_pDuplex->type = 1;

        GetDlgItem( IDC_B8BIT8KHZ )-> EnableWindow( FALSE );
        GetDlgItem( IDC_BSTOPDUPLEX )-> EnableWindow( TRUE );
        if( FAILED( g_pDuplex-> StartBuffers() ) )
        {
                MessageBox( "Can not DirectSound. "
                        , MB_OK | MB_ICONERROR );
                EndDialog( IDABORT );
        }
        else
        {
                g_bRecording   = true;
                AfxBeginThread( WaitThreadProc, this );
        }
}
```

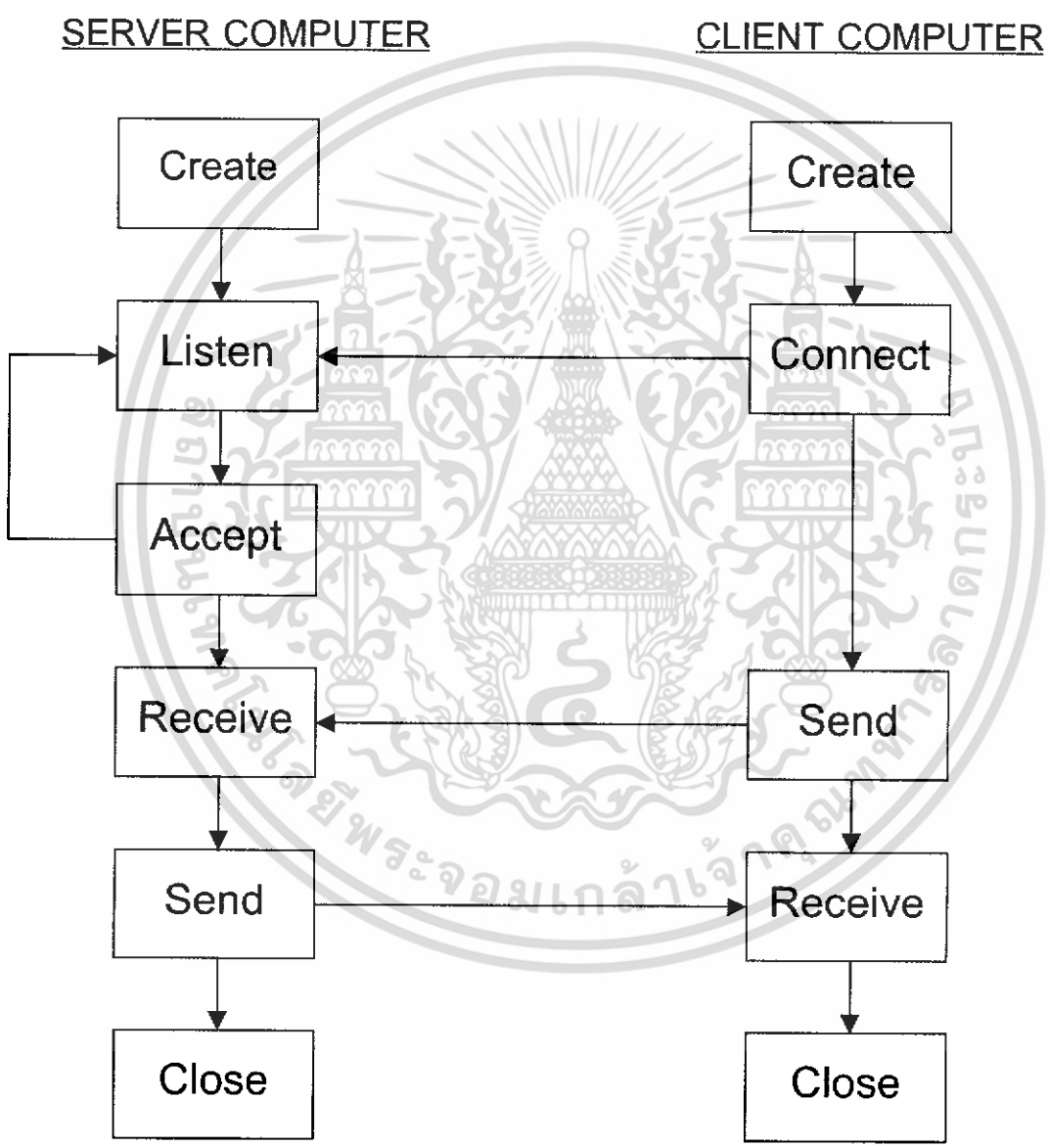
โปรแกรมย่อยของการยกเลิกส่งข้อมูลเสียง

```
void CFULL_DUPLEXDlg::OnBstopduplex()
{
        g_bRecording = false;
        if( g_pDuplex )
                delete   g_pDuplex;
        g_bRecording = false;

        GetDlgItem( IDC_B8BIT8KHZ )-> EnableWindow( TRUE );
        GetDlgItem( IDC_BSTOPDUPLEX )-> EnableWindow( FALSE );
}
```

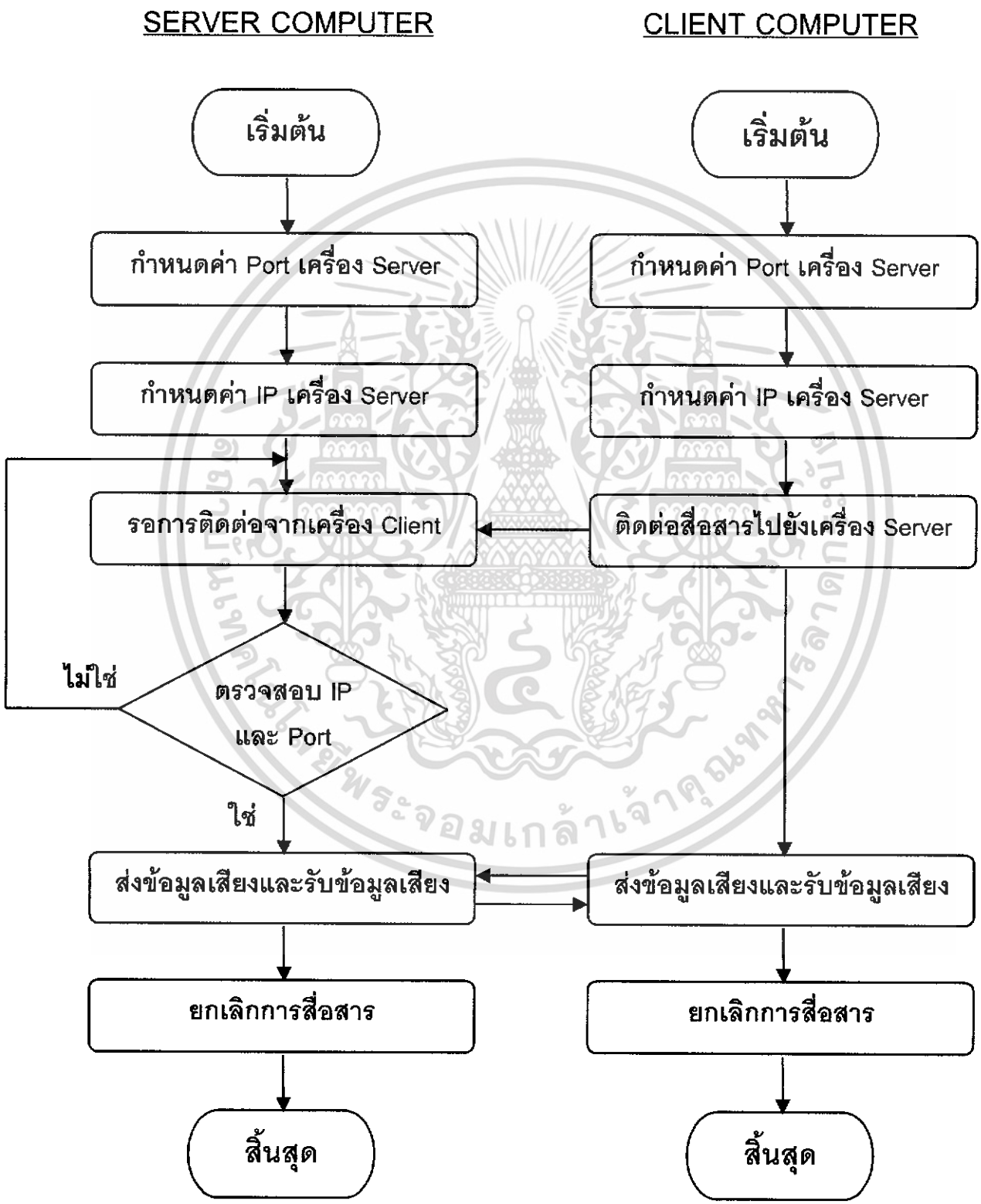
Voice Over Internet Protocol

Block Diagrams



Voice Over Internet Protocol

Flow Chart



```
//Duplex.cpp
//
//////////////////////////////////////////////////////////////////////
///

#include "stdafx.h"
#include "Duplex.h"
#include "math.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
///
//
//////////////////////////////////////////////////////////////////////
///

Duplex::Duplex( HWND hDlg )
{
        DSBUFFERDESC            dsbdesc;

        ZeroMemory( &g_aPosNotify, sizeof(DSBPOSITIONNOTIFY) *
(NUM_PLAY_NOTIFICATIONS + 1) );

        g_dwPlayBufferSize              = 0;
        g_dwCaptureBufferSize   = 0;
        g_dwNotifySize                  = 0;
        g_dwNextPlayOffset              = 0;
        g_bUseFilter                    = FALSE;


        g_pDS                                   = NULL;
        g_pDSCapture                    = NULL;
        g_pDSBPrimary                   = NULL;
        g_pDSBOutput                    = NULL;
        g_pDSBCapture                   = NULL;
        g_pDSNotify                             = NULL;
        g_pCaptureWaveFormat    = NULL;

        if( CoInitialize(NULL) )
                throw;

        if( FAILED( DirectSoundCreate( NULL, &g_pDS, NULL ) ) )
                throw;

        if( FAILED( g_pDS-> SetCooperativeLevel( hDlg,
DSSCL_PRIORITY ) ) )
                throw;

        ZeroMemory( &dsbdesc, sizeof(DSBUFFERDESC) );
        dsbdesc.dwSize  = sizeof(DSBUFFERDESC);
        dsbdesc.dwFlags = DSBCAPS_PRIMARYBUFFER;
```

```
        if( FAILED( g_pDS->CreateSoundBuffer( &dsbdesc,
&g_pDSBPrimary, NULL ) ) )
                throw;

        if( FAILED( DirectSoundCaptureCreate( NULL, &g_pDSCapture,
NULL ) ) )
                throw;

        g_hNotificationEvents[0] = CreateEvent( NULL, FALSE, FALSE,
NULL );
        g_hNotificationEvents[1] = CreateEvent( NULL, FALSE, FALSE,
NULL );
}

Duplex::~Duplex()
{

        if( g_pCaptureWaveFormat )
                delete  g_pCaptureWaveFormat;

        if( g_pDSNotify )
                g_pDSNotify-> Release();

        if( g_pDSBPrimary )
                g_pDSBPrimary-> Release();

        if( g_pDSBOutput )
                g_pDSBOutput-> Release();

        if( g_pDSBCapture )
                g_pDSBCapture-> Release();

        if( g_pDSCapture )
                g_pDSCapture-> Release();

        if( g_pDS )
                g_pDS-> Release();

        CoUninitialize();

        CloseHandle( g_hNotificationEvents[0] );
        CloseHandle( g_hNotificationEvents[1] );
}

//-----------------------------------------------------------------
-----------
//-----------------------------------------------------------------
-----------
HRESULT Duplex::ScanAvailableOutputFormats()
{
        WAVEFORMATEX    wfx;
        WAVEFORMATEX    wfxSet;
        HRESULT                 hr;
        HCURSOR                 hCursor;
```

```
                                Duplex
        hCursor = GetCursor();
        SetCursor( LoadCursor( NULL, IDC_WAIT ) );

        ZeroMemory( &wfxSet, sizeof(wfxSet) );
        wfxSet.wFormatTag = WAVE_FORMAT_PCM;

        ZeroMemory( &wfx, sizeof(wfx) );
        wfx.wFormatTag = WAVE_FORMAT_PCM;

        for( INT iIndex = 0; iIndex < 16; iIndex++ )
        {
                GetWaveFormatFromIndex( iIndex, &wfx );

                if( FAILED( hr = g_pDSBPrimary->SetFormat( &wfx ) )
)
                {
                        g_abOutputFormatSupported[ iIndex ] =
FALSE;
                }
                else
                {
                        if( FAILED( hr = g_pDSBPrimary->GetFormat(
&wfxSet, sizeof(wfxSet),
                                NULL ) ) )
                                return hr;

                        if( memcmp( &wfx, &wfxSet, sizeof(wfx) ) ==
0 )
                                g_abOutputFormatSupported[ iIndex ]
= TRUE;
                        else
                                g_abOutputFormatSupported[ iIndex ]
= FALSE;
                }
        }

        SetCursor( hCursor );

        return S_OK;
}

//-----------------------------------------------------------------
------------
//                      ScanAvailableInputFormats()
//-----------------------------------------------------------------
------------
HRESULT Duplex::ScanAvailableInputFormats()
{
        HRESULT                                 hr;
        WAVEFORMATEX                    wfx;
        HCURSOR                                 hCursor;
        DSCBUFFERDESC                   dscbd;
        LPDIRECTSOUNDCAPTUREBUFFER      pDSCaptureBuffer = NULL;
```

```
        hCursor = GetCursor();
        SetCursor( LoadCursor( NULL, IDC_WAIT ) );

        ZeroMemory( &wfx, sizeof(wfx) );
        wfx.wFormatTag = WAVE_FORMAT_PCM;

        ZeroMemory( &dscbd, sizeof(dscbd) );
        dscbd.dwSize = sizeof(dscbd);

        for( INT iIndex = 0; iIndex < 16; iIndex++ )
        {
                GetWaveFormatFromIndex( iIndex, &wfx );
                dscbd.dwBufferBytes = wfx.nAvgBytesPerSec;
                dscbd.lpwfxFormat = &wfx;

                if( FAILED( hr = g_pDSCapture->CreateCaptureBuffer(
&dscbd,

                                                &pDSCaptureBuffer,

                                                NULL ) ) )
                        g_abInputFormatSupported[ iIndex ] = FALSE;
                else
                        g_abInputFormatSupported[ iIndex ] = TRUE;

                if( pDSCaptureBuffer )
                        pDSCaptureBuffer->Release();
        }

        SetCursor( hCursor );

        return S_OK;
}

//-----------------------------------------------------------------
------------
//      GetWaveFormatFromIndex()
//-----------------------------------------------------------------
------------
void Duplex::GetWaveFormatFromIndex( int nIndex, WAVEFORMATEX* pwfx
)
{
        int iSampleRate = nIndex >> 2;
        int iType = nIndex % 4;

        switch( iSampleRate )
        {
                case 0: pwfx->nSamplesPerSec =  8000; break;
                case 1: pwfx->nSamplesPerSec = 11025; break;
                case 2: pwfx->nSamplesPerSec = 22050; break;
                case 3: pwfx->nSamplesPerSec = 44100; break;
        }

        switch( iType )
        {
                case 0: pwfx->wBitsPerSample =  8; pwfx->nChannels
```

```
= 1; break;
                case 1: pwfx->wBitsPerSample = 16; pwfx->nChannels
= 1; break;
                case 2: pwfx->wBitsPerSample =  8; pwfx->nChannels
= 2; break;
                case 3: pwfx->wBitsPerSample = 16; pwfx->nChannels
= 2; break;
        }

        pwfx->nBlockAlign = pwfx->nChannels * (
pwfx->wBitsPerSample >> 3 );
        pwfx->nAvgBytesPerSec = pwfx->nBlockAlign *
pwfx->nSamplesPerSec;
}

//-----------------------------------------------------------------
------------
//                      SetBufferFormats()
//-----------------------------------------------------------------
------------
HRESULT Duplex::SetBufferFormats( WAVEFORMATEX* pwfxInput,
WAVEFORMATEX* pwfxOutput )
{
        HRESULT hr;
        DSCBUFFERDESC dscbd;

        if( FAILED( hr = g_pDSBPrimary->SetFormat( pwfxOutput ) ) )
                return hr;

        if( pwfxInput->nAvgBytesPerSec > 32768 )
                g_dwNotifySize  = pwfxInput->nAvgBytesPerSec >> 3;
        else
                //This g_dwNotifySize sets the buffer lenght
                g_dwNotifySize  = 500;//4096;

        g_dwNotifySize -= g_dwNotifySize % pwfxInput->nBlockAlign;


        g_dwPlayBufferSize = NUM_BUFFERS * g_dwNotifySize / 2;
        g_dwCaptureBufferSize = g_dwNotifySize * NUM_BUFFERS;

        if( g_pDSBCapture )
                g_pDSBCapture->Release();

        ZeroMemory( &dscbd, sizeof(dscbd) );
        dscbd.dwSize            = sizeof(dscbd);
        dscbd.dwBufferBytes = g_dwCaptureBufferSize;
        dscbd.lpwfxFormat   = pwfxInput;

        if( FAILED( hr = g_pDSCapture->CreateCaptureBuffer( &dscbd,
                                                        &g_pDSBCapture,
NULL ) ) )
                return hr;

        return S_OK;
```

```
}

//-----------------------------------------------------------------------
//                      CreateOutputBuffer()
//-----------------------------------------------------------------------
HRESULT Duplex::CreateOutputBuffer()
{
        HRESULT                 hr;
        WAVEFORMATEX    wfxInput;

        ZeroMemory( &wfxInput, sizeof(wfxInput) );
        g_pDSBCapture->GetFormat( &wfxInput, sizeof(wfxInput), NULL
);

        // Set up the direct sound buffer, and only request the
flags needed
        // since each requires some overhead and limits if the
buffer can
        // be hardware accelerated

        DSBUFFERDESC dsbd;
        ZeroMemory( &dsbd, sizeof(DSBUFFERDESC) );
        dsbd.dwSize             = sizeof(DSBUFFERDESC);
        dsbd.dwFlags        = DSBCAPS_GLOBALFOCUS;
        dsbd.dwBufferBytes = g_dwPlayBufferSize;
        dsbd.lpwfxFormat    = &wfxInput;

        // DirectSound
        if( FAILED( hr = g_pDS->CreateSoundBuffer( &dsbd,
&g_pDSBOutput, NULL ) ) )
                return hr;

        if( FAILED( hr = g_pDSBCapture->QueryInterface(
IID_IDirectSoundNotify,

                                (VOID**)&g_pDSNotify ) ) )
                return hr;

        for( INT i = 0; i < NUM_PLAY_NOTIFICATIONS; i++ )
        {
                g_aPosNotify[i].dwOffset = (g_dwNotifySize * i) +
g_dwNotifySize - 1;
                g_aPosNotify[i].hEventNotify =
g_hNotificationEvents[0];
        }

        g_aPosNotify[i].dwOffset         = DSBPN_OFFSETSTOP;
        g_aPosNotify[i].hEventNotify = g_hNotificationEvents[1];

        if( FAILED( hr = g_pDSNotify->SetNotificationPositions(
NUM_PLAY_NOTIFICATIONS + 1,


g_aPosNotify ) ) )
```

```
                                    Duplex
                return hr;

        return S_OK;
}

//-----------------------------------------------------------------
-----------
//                      StartBuffers()
//-----------------------------------------------------------------
-----------
HRESULT Duplex::StartBuffers()
{
        LPVOID pbBuffer;
        DWORD dwBufferSize;
        WAVEFORMATEX wfxOutput;
        HRESULT hr;

        if( FAILED( hr = RestoreBuffers() ) )
                return hr;

        g_pDSBCapture->GetCurrentPosition( &g_dwNextCaptureOffset,
NULL );
        g_dwNextCaptureOffset -= g_dwNextCaptureOffset %
g_dwNotifySize;

        g_dwNextPlayOffset = g_dwNextCaptureOffset + ( 2 *
g_dwNotifySize );
        g_dwNextPlayOffset %= g_dwPlayBufferSize;  //

        g_pDSBCapture->Start( DSCBSTART_LOOPING );

        g_pDSBOutput->SetCurrentPosition( g_dwNextPlayOffset );

        // g_pCaptureWaveFormat
        if( g_pCaptureWaveFormat )
                delete  g_pCaptureWaveFormat;

        g_pCaptureWaveFormat = new WAVEFORMATEX;
        ZeroMemory( g_pCaptureWaveFormat, sizeof(WAVEFORMATEX) );
        g_pDSBCapture->GetFormat( g_pCaptureWaveFormat,
sizeof(WAVEFORMATEX), NULL );

        ZeroMemory( &wfxOutput, sizeof(wfxOutput) );
        g_pDSBOutput->GetFormat( &wfxOutput, sizeof(wfxOutput),
NULL );

        // HandleNotifications()
        if( FAILED( hr = g_pDSBOutput->Lock( 0, g_dwPlayBufferSize,


         &pbBuffer, &dwBufferSize,

         NULL, NULL, 0 ) ) )
                return hr;
        FillMemory( (BYTE*) pbBuffer, dwBufferSize,
                                (BYTE)( wfxOutput.wBitsPerSample ==
```

```
8 ? 128 : 0 ) );
        g_pDSBOutput->Unlock( pbBuffer, dwBufferSize, NULL, NULL );


        g_pDSBOutput->Play( 0, 0, DSBPLAY_LOOPING );

        return S_OK;
}

//-----------------------------------------------------------------------
------------
//                          StopBuffers()
//-----------------------------------------------------------------------
------------
HRESULT Duplex::StopBuffers()
{
        if( (g_pDSBCapture == NULL) || (g_pDSBOutput == NULL) )
                return S_OK;

        g_pDSBCapture-> Stop();
        g_pDSBOutput-> Stop();

        return S_OK;
}

//-----------------------------------------------------------------------
------------
//                          RestoreBuffers()
//-----------------------------------------------------------------------
------------
HRESULT Duplex::RestoreBuffers()
{
        HRESULT hr;

        if( NULL == g_pDSBCapture ||
                NULL == g_pDSBOutput )
                return S_OK;

        DWORD dwStatus;
        if( FAILED( hr = g_pDSBOutput->GetStatus( &dwStatus ) ) )
                return hr;

        if( dwStatus & DSBSTATUS_BUFFERLOST )
        {

                while( g_pDSBOutput->Restore() == DSERR_BUFFERLOST
)
                        Sleep( 10 );
        }

        return S_OK;
}

//-----------------------------------------------------------------------
------------
//                          HandleNotification()
```

```
//-----------------------------------------------------------------
------------
HRESULT Duplex::HandleNotification(BYTE* audio,BYTE*
receive_audio,int* lenght)
{
        DWORD   i;
        int             j;
        HRESULT hr;
        VOID*   pbCaptureData   = NULL;
        VOID*   pbPlayData        = NULL;
        DWORD   dwCaptureLength;
        DWORD   dwPlayLength;

        DWORD dwStatus;
        if( FAILED( hr = g_pDSBOutput->GetStatus( &dwStatus ) ) )
                return hr;

        if( dwStatus & DSBSTATUS_BUFFERLOST )
        {
                if( FAILED( hr = StartBuffers() ) )
                        return hr;

                return S_OK;
        }

        if( FAILED( hr = g_pDSBCapture->Lock(
g_dwNextCaptureOffset, g_dwNotifySize,

          &pbCaptureData, &dwCaptureLength,

          NULL, NULL, 0L ) ) )
                return hr;

        if( FAILED( hr = g_pDSBOutput->Lock( g_dwNextPlayOffset,
g_dwNotifySize,

         &pbPlayData, &dwPlayLength,

         NULL, NULL, 0L ) ) )
                return hr;

        if( g_bUseFilter )
        {
                if( g_pCaptureWaveFormat->wBitsPerSample == 8 )
                {
                        BYTE*   pbIn    = (BYTE*) pbCaptureData;
                        BYTE*   pbOut   = (BYTE*) pbPlayData;

                        for( i=0; i < dwPlayLength; i++, pbIn++,
pbOut++ )
                        {
                                j = *pbIn - nDelay;

                                if( j > 127 )
                                        *pbOut = 255;
                                else
```

```
                                        Duplex
                                                if( j < -128 )
                                                        *pbOut = 0;
                                                else
                                                        *pbOut = j + 128;

                                                nDelay  = *pbIn;
                                        }
                                }
                                else
                                {
                                        short*  pnIn    = (short*) pbCaptureData;
                                        short*  pnOut   = (short*) pbPlayData;

                                        for( i=0; i < dwPlayLength >> 1; i++,
pnIn++, pnOut++)
                                        {
                                                j = *pnIn - nDelay;

                                                if( j > 32767 )
                                                        *pnOut = 32767;
                                                else
                                                        if( j < -32768 )
                                                                *pnOut = -32768;
                                                        else
                                                                *pnOut = j;

                                                nDelay  = *pnIn;
                                        }
                                }
        }
//      else
/*      if(type==2)
        {//CopyMemory(data,pbCaptureData,dwCaptureLength);
        //      Sum(data,data,dwCaptureLength);
                CopyMemory( pbPlayData, data, dwPlayLength );
        }
        if(type==1)
//              CopyMemory( pbPlayData, pbCaptureData, dwPlayLength
);
        if(type==3)
                CopyMemory( pbPlayData, data2, dwPlayLength );
*/
        CopyMemory(audio,pbCaptureData,dwPlayLength);
        if(dwPlayLength==dwCaptureLength)
                CopyMemory( pbPlayData, receive_audio,
dwCaptureLength );

        *lenght = dwCaptureLength;
//      CopyMemory(pbPlayData,audio,dwPlayLength);
        g_pDSBOutput->Unlock( pbPlayData, dwPlayLength, NULL, 0 );

        g_pDSBCapture->Unlock( pbCaptureData, dwCaptureLength,
NULL, 0 );
        g_dwNextCaptureOffset += dwCaptureLength;
        g_dwNextCaptureOffset %= g_dwCaptureBufferSize;
```

```
                                        Duplex
        g_dwNextPlayOffset += dwPlayLength;
        g_dwNextPlayOffset %= g_dwPlayBufferSize;

        return S_OK;
}

//-----------------------------------------------------------------------
------------
//              SetFilter()
//-----------------------------------------------------------------------
------------
void Duplex::SetFilter( int bFilter )
{
        if( bFilter )
        {
                g_bUseFilter    = TRUE;

                if( g_pCaptureWaveFormat-> wBitsPerSample == 8 )
                        nDelay  = 128;
                else
                        nDelay  = 0;
        }
        else
                g_bUseFilter    = FALSE;
}

void Duplex::SetData()
{
        for(int i=0;i<100000;i++)
        {
                data[i] = rand()%250;
                data2[i] = sin(6.28*i*10/180)*2;
        }

}

void Duplex::ChangeType(int type1)
{
                type=type1;
}

void Duplex::Sum(byte *dest, byte *source, int lenght)
{
        for(int i=0;i<lenght;i++)
        {
                *(dest+i) = (*(source+i) + data2[i])/2;
        }

}
```

```
// FULL_DUPLEX.cpp : Defines the class behaviors for the
application.
//

#include "stdafx.h"
#include "FULL_DUPLEX.h"
#include "FULL_DUPLEXDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////
////////
// CFULL_DUPLEXApp

BEGIN_MESSAGE_MAP(CFULL_DUPLEXApp, CWinApp)
        //{{AFX_MSG_MAP(CFULL_DUPLEXApp)
                // NOTE - the ClassWizard will add and remove
mapping macros here.
                //    DO NOT EDIT what you see in these blocks of
generated code!
        //}}AFX_MSG
        ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////
////////
// CFULL_DUPLEXApp construction

CFULL_DUPLEXApp::CFULL_DUPLEXApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////
////////
// The one and only CFULL_DUPLEXApp object

CFULL_DUPLEXApp theApp;

/////////////////////////////////////////////////////////////////////////
////////
// CFULL_DUPLEXApp initialization

BOOL CFULL_DUPLEXApp::InitInstance()
{
        if (!AfxSocketInit())
        {
                AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                return FALSE;
        }
```

```
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce
the size
	//  of your final executable, you should remove from the
following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when
using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking
to MFC statically
#endif

	CFULL_DUPLEXDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog
is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog
is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that
we exit the
	//  application, rather than start the application's
message pump.
	return FALSE;
}
```

```
// MySocket.cpp : implementation file
//

#include "stdafx.h"
#include "FULL_DUPLEX.h"
#include "MySocket.h"
#include "FULL_DUPLEXDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMySocket

CMySocket::CMySocket()
{
}

CMySocket::~CMySocket()
{
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMySocket, CAsyncSocket)
	//{{AFX_MSG_MAP(CMySocket)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif  // 0

/////////////////////////////////////////////////////////////////////////////
// CMySocket member functions

void CMySocket::SetParent(CDialog *pWnd)
{
	m_pWnd = pWnd;
}

void CMySocket::OnAccept(int nErrorCode)
{
	if(nErrorCode==0)
		((CFULL_DUPLEXDlg*)m_pWnd)->OnAccept();
}

void CMySocket::OnConnect(int nErrorCode)
{
	if(nErrorCode==0)
		((CFULL_DUPLEXDlg*)m_pWnd)->OnConnect();
}
```

```
void CMySocket::OnClose(int nErrorCode)
{
        if(nErrorCode==0)
                ((CFULL_DUPLEXDlg*)m_pWnd)->OnClose();
}

void CMySocket::OnReceive(int nErrorCode)
{
        if(nErrorCode==0)
                ((CFULL_DUPLEXDlg*)m_pWnd)->OnReceive();
}

void CMySocket::OnSend(int nErrorCode)
{
        if(nErrorCode==0)
                ((CFULL_DUPLEXDlg*)m_pWnd)->OnSend();
}
```