อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

GPRS WIRELESS GATEWAY

นางสาวดวงฤดี ประมวลวุฒิรณ
นายปริวรรต วงษ์สำราญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

GPRS WIRELESS GATEWAY

โดย

นางสาวดวงฤดี ประมวลวุฒิรณ

นายปริวรรต วงษ์สำราญ

อาจารย์ที่ปรึกษา

ผศ.อภิเนตร อูนากูล

อ.ดุสิต นิยะโต

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

GPRS WIRELESS GATEWAY

ผู้จัดทำ

1. นางสาวดวงฤดี	ประมวลวุฒิรณ	รหัสประจำตัว	42010114
2. นายปริวรรต	วงษ์สำราญ	รหัสประจำตัว	42010194

..... อาจารย์ที่ปรึกษา

(ผศ.อภิเนตร อูนากูล)

..... อาจารย์ที่ปรึกษา

(อ.ดุสิต นิยะโต)

อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

นางสาวดวงฤดี ประมวลวุฒิรณ 42010114
นายปริวรรต วงษ์สำราญ 42010194
ผศ. อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
อาจารย์ ดุสิต นิยะโต อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

ในปัจจุบัน การเชื่อมต่อเครื่องคอมพิวเตอร์เข้ากับ ระบบเครือข่ายท้องถิ่นนั้น ถูกทดแทนด้วยระบบเครือข่ายท้องถิ่นไร้สาย โดยใช้อุปกรณ์แอกเซสพอยน์เป็นจุดเชื่อมต่อจากที่ใดใด เพื่อลดข้อจำกัดการใช้สายเคเบิล อย่างไรก็ตามการเชื่อมต่อเข้าระบบเครือข่ายอินเทอร์เน็ตยังถูกจำกัดอยู่เฉพาะการเชื่อมต่อผ่านโมเด็มเช่นเดียวกัน แต่ยังมีเทคโนโลยีใหม่ที่เริ่มเข้ามามีบทบาทสำคัญ เพื่อลดข้อจำกัดนี้ซึ่งก็คือเทคโนโลยีจีพีอาร์เอส (GPRS : General Radio Packet Service) ที่สามารถเชื่อมต่อเข้ากับระบบเครือข่ายอินเทอร์เน็ตจากที่ใดใดก็ได้

จากข้อจำกัด และ แนวทางแก้ไข ดังกล่าว ปริญญานิพนธ์นี้จึงมีวัตถุประสงค์เพื่อพัฒนาอุปกรณ์เกตเวย์ภายใต้ เทคโนโลยีระบบฝังตัว โดยมีคุณสมบัติดังต่อไปนี้ สามารถเชื่อมต่อระบบเครือข่ายอินเทอร์เน็ตผ่านเครือข่ายจีพีอาร์เอส ด้วยโปรโตคอล พีพีพี สามารถเชื่อมต่อกับระบบเครือข่ายท้องถิ่น ผ่านแอกเซสพอยน์ ด้วยมาตรฐาน IEEE 802.11b สามารถทำการแปลและแปลงไอพีแอดเดรสจากเครือข่ายส่วนตัวให้เป็นที่ยอมรับของเครือข่ายภายนอกได้ สามารถกำหนดค่าคุณสมบัติของอุปกรณ์เกตเวย์ผ่านทาง อาร์เอส - 232 โดยใช้โปรแกรม ไฮเปอร์เทอร์มินอล และผ่านทางโปรโตคอล เอชทีทีพี โดยใช้โปรแกรมเวบบราวเซอร์ได้ โดยเลือกพัฒนาบนชุดพัฒนาคอม 86 ซึ่งมีราคาถูกและมีคุณสมบัติเพียงพอที่จะพัฒนาเป็นอุปกรณ์เกตเวย์แบบไร้สายด้วยจีพีอาร์เอสได้

GPRS WIRELESS GATEWAY

Duangrudee Pramualwuttiron 42010114
Pariwat Wongsamran 42010194
Asst. Prof. Apinetr Unakul advisor
Mr. Dusit Niyato advisor

ABSTRACT

Nowadays, computer connection to Local Area Network (LAN) is replaced by Wireless LAN which user can connect computers to access point from everywhere without any cables. However, computer connection to Internet have a limit which must use modem connect to Internet. But even so, new technology which has become an importance for reducing that constrain, which is to say GPRS Technology (General Radio Packet Service). It can connect to Internet from everywhere.

From the constrain and the way of solving, this thesis has objective to develop gateway under embedded systems technology. Which have properties as follows : connect to Internet via GPRS network with PPP, connect to LAN with access point under standard IEEE 802.11b, Network Address Translation, can configure gateway via RS-232 with Hyper terminal program and via HTTP with web browser program. It has develop on COM 86 development kit which cheap and has property enough for development to GPRS wireless gateway.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลายๆฝ่ายด้วยกัน บุคคลกลุ่มแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์อภิเนตร อูนากูล และอาจารย์ดุสิต นิยะโต อาจารย์ที่ปรึกษาโครงงาน ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ขอขอบคุณอาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกๆท่านที่ให้ความรู้และคำปรึกษามาตลอดระยะเวลา 4 ปี

ขอขอบคุณบริษัท SIEMENS Ltd. ที่ได้อนุเคราะห์โมดูลเชื่อมต่อเครือข่ายจีพีอาร์เอส เอ็มซี 35 จึงทำให้การพัฒนาโครงงานนี้สำเร็จลุล่วงไปด้วยดี

ขอขอบคุณสมาชิกทุกท่านในห้องปฏิบัติการ Embedded Systems Lab ที่คอยให้ความช่วยเหลือเมื่อมีปัญหาในการทำงาน และห้องปฏิบัติการ Embedded Systems Lab ที่เอื้อเฟื้อสถานที่ในการดำเนินโครงงาน ขอขอบคุณ Mr. Jason Sarich สำหรับการให้ความช่วยเหลือ คอยให้คำปรึกษาผ่านทางอีเมล์ตลอดเวลาที่ต้องการความช่วยเหลือ และ พี่สุริยันต์ เลาหประภานนท์ บริษัท Netgadgets Co., Ltd ที่คอยให้คำปรึกษารวมทั้งให้ข้อมูลเกี่ยวกับ Embedded Systems Programming มาตลอด

ขอขอบคุณรุ่นพี่และเพื่อนๆ ทุกคนที่คอยให้คำปรึกษา ช่วยเหลือและให้กำลังใจมาตลอด และต้องขอขอบพระคุณ บุคคลสำคัญที่สุดที่ทำให้มีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุกๆด้าน อันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

นางสาวดวงฤดี ประมวลวุฒิรณ
นายปริวรรต วงษ์สำราญ

สารบัญ

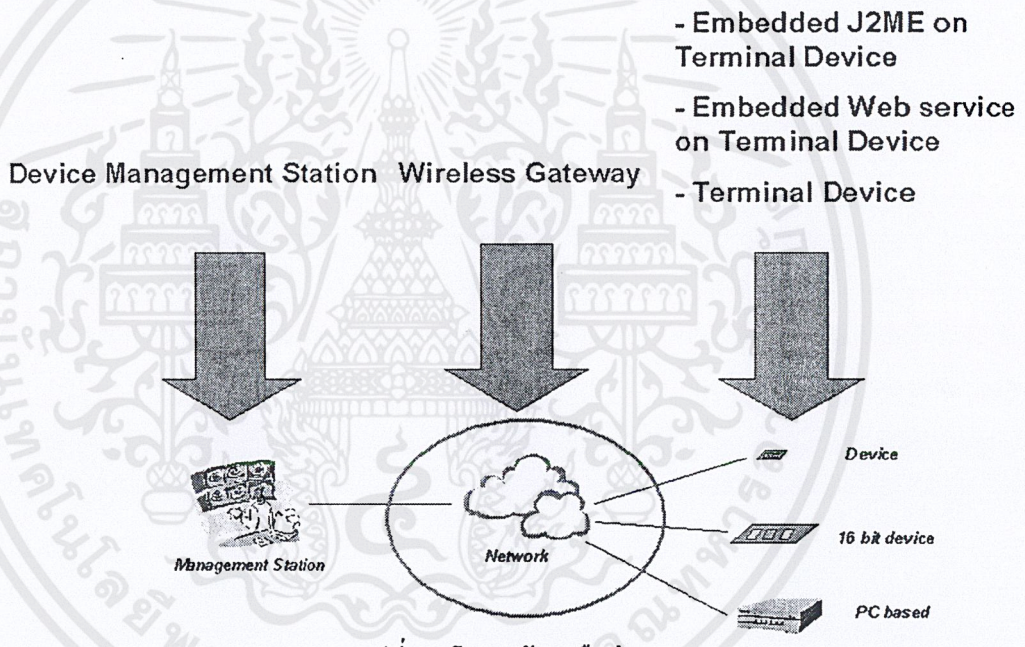
สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เมื่อพิจารณาแนวโน้มเทคโนโลยีในอนาคตที่มีอัตราการเติบโตสูงขึ้นอย่างก้าวกระโดด ของภาคอุตสาหกรรมแล้วจะพบว่า เทคโนโลยีระบบฝังตัว เป็นเทคโนโลยีหนึ่งที่กำลังเข้ามามีบทบาทต่อนักพัฒนาฮาร์ดแวร์และซอฟต์แวร์มากขึ้น ไม่เพียงแต่แนวโน้มของความนิยมการใช้งานอุปกรณ์มือถือเท่านั้น แต่ด้วยตัวของเทคโนโลยีที่ได้รับการพัฒนาอย่างต่อเนื่อง จึงเป็นการผลักดันให้เกิดนวัตกรรมใหม่ทั้งทางด้านระบบการจัดการ ระบบเครือข่าย และอุปกรณ์ปลายทาง และสิ่งที่น่าสนใจในปัจจุบันก็คือระบบเครือข่าย ดังรูปที่ 1-1



รูปที่ 1-1 โครงสร้างเครือข่าย

รวมทั้งในปัจจุบัน การเชื่อมต่อคอมพิวเตอร์หรืออุปกรณ์ต่างๆเข้ากับระบบเครือข่ายอินเทอร์เน็ตไม่ได้ถูกจำกัดอยู่เฉพาะการเชื่อมต่อผ่านโมเด็มเท่านั้น ยังมีเทคโนโลยีใหม่ที่เริ่มเข้ามีบทบาทสำคัญ ซึ่งก็คือเทคโนโลยี จีพีอาร์เอส (GPRS : General Radio Packet Service) ซึ่งเป็นการส่งข้อมูลต่างๆในรูปแบบแพ็กเกตโดยการส่งผ่านระบบเครือข่ายจีพีอาร์เอส ที่เชื่อมต่อกับระบบเครือข่ายอินเทอร์เน็ตอีกที ซึ่งมีข้อได้เปรียบจากการเชื่อมต่อด้วยโมเด็ม คือสามารถทำการเชื่อมต่อระบบเครือข่ายอินเทอร์เน็ตจากที่ใดก็ได้และคิดค่าบริการตามจำนวนข้อมูลที่รับ-ส่งข้อมูลเท่านั้น

ประกอบกับระบบเครือข่ายท้องถิ่นไร้สายได้เข้ามามีบทบาทต่อการเชื่อมต่อสื่อสารของคอมพิวเตอร์เป็นอย่างมาก เนื่องด้วยข้อดีก็คือผู้ใช้สามารถเชื่อมต่อเข้ากับระบบเครือข่ายท้องถิ่นไร้สายได้จากที่ใดๆ

โดยไม่จำเป็นต้องหาสายเคเบิลเหมือนในอดีต ระบบเครือข่ายท้องถิ่นไร้สาย (Wireless LAN) ที่ได้รับความนิยมเป็นอย่างมากได้แก่ IEEE 802.11b โดยโครงสร้างการเชื่อมต่อของระบบเครือข่ายท้องถิ่นไร้สายนั้น คอมพิวเตอร์ไคลเอ็นท์จะมีการ์ดที่เชื่อมต่อเข้ากับแอ็กเซสพอยน์ (Access Point) ซึ่งจะทำหน้าที่เป็นตัวกลางในการสื่อสารกับระบบเครือข่าย ท้องถิ่น (LAN) แบบใช้สายอีกทีหนึ่ง ซึ่งปัจจุบันได้มีผู้พัฒนาแอกเซสพอยน์นี้ให้มีโมเด็มเพื่อเชื่อมต่อไปยังผู้ให้บริการอินเทอร์เน็ต (ISP) อุปกรณ์นี้เหมาะสำหรับการใช้งานในบ้านที่มีคอมพิวเตอร์ 3-4 เครื่อง เนื่องจากผู้ใช้ไม่จำเป็นต้องตั้งเซิร์ฟเวอร์เพื่อทำหน้าที่เป็นเกตเวย์เอง และอุปกรณ์เหล่านี้ก็ได้รับการตั้งค่าให้สามารถทำงานได้ มาเรียบร้อยแล้ว

อย่างไรก็ตาม อุปกรณ์แอกเซสพอยน์ในท้องตลาดเหล่านี้ยังมีข้อจำกัดเนื่องจากการเชื่อมต่อยังต้องทำผ่านโมเด็มเท่านั้น ส่งผลให้ตัวแอกเซสพอยน์ต้องเชื่อมต่อกับสายเคเบิลของโทรศัพท์ ด้วยข้อจำกัดนี้จึงได้เกิดแนวความคิดของการเชื่อมต่อระบบเครือข่ายอินเทอร์เน็ตผ่านระบบเครือข่ายจีพีอาร์เอส (GPRS) ด้วยโทรศัพท์เคลื่อนที่ขึ้น ซึ่งสามารถอำนวยความสะดวกให้กับกลุ่มของผู้ใช้ที่ต้องการเชื่อมต่อกับอินเทอร์เน็ตด้วยเครือข่ายไร้สายอย่างแท้จริง

จากแนวคิดดังกล่าว จึงมีการพัฒนาอุปกรณ์เกตเวย์ภายใต้ เทคโนโลยีระบบฝังตัว ให้มีความสามารถเชื่อมต่อระบบเครือข่ายอินเทอร์เน็ตผ่านเครือข่ายจีพีอาร์เอสด้วยโทรศัพท์เคลื่อนที่ และเชื่อมต่อกับระบบเครือข่ายท้องถิ่นด้วยแอ็กเซสพอยน์ โดยเลือกพัฒนาบนชุดพัฒนาคอม86 ซึ่งมีราคาถูกและมีคุณสมบัติเพียงพอที่จะพัฒนาเป็นอุปกรณ์เกตเวย์แบบไร้สายด้วยจีพีอาร์เอสได้

1.2 วัตถุประสงค์ของงานวิจัย

1. เพื่อศึกษาการทำงานของระบบเครือข่ายอินเทอร์เน็ต ระบบเครือข่ายท้องถิ่น ระบบเครือข่ายจีพีอาร์เอส และระบบเครือข่ายท้องถิ่นไร้สาย
2. เพื่อเป็นการพัฒนาอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส ภายใต้เทคโนโลยีระบบฝังตัว
3. เพื่อพัฒนาการรวมเทคโนโลยีระบบเครือข่ายจีพีอาร์เอส และ ระบบเครือข่ายท้องถิ่นไร้สายต่างๆ ไว้ด้วยกัน
4. เพื่อให้เป็นการพัฒนาและลดข้อจำกัดของระบบเครือข่ายไร้สายในปัจจุบัน

1.3 ขอบเขตของงานวิจัย

การพัฒนางานวิจัยนี้จะคลอบคลุมงานต่างๆเพื่อให้ได้ผลลัพธ์ดังนี้

1. ต้นแบบอุปกรณ์เกตเวย์ สามารถเชื่อมต่อเครือข่ายอินเทอร์เน็ตผ่านเครือข่ายจีพีอาร์เอส
2. ต้นแบบอุปกรณ์เกตเวย์ สามารถเชื่อมต่อระบบเครือข่ายท้องถิ่นไร้สายได้
3. ต้นแบบอุปกรณ์เกตเวย์ สามารถกำหนดค่าคุณสมบัติผ่านทาคอนโซลพอร์ตอนุกรม
4. ต้นแบบอุปกรณ์เกตเวย์ สามารถกำหนดค่าคุณสมบัติผ่านเวบบราวเซอร์
5. ต้นแบบอุปกรณ์เกตเวย์ สามารถกำหนดการติดตั้งค่าแบบไดนามิกให้โฮสต์ในเครือข่าย (DHCP : Dynamic Host Configuration Protocol)

6. ต้นแบบอุปกรณ์เกตเวย์ สามารถแปลและแปลงไอพีแอดเดรสของเครือข่าย (Network Address Translation)

1.4 วิธีการดำเนินงาน

งานวิจัยในโครงงานนี้ จะเริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่างๆ ที่เกี่ยวข้องกับงานวิจัย ซึ่งก็มีเรื่องหลักๆอยู่ 4 เรื่อง ด้วยกัน คือ เทคโนโลยีไร้สาย โปรโตคอลที่เกี่ยวข้อง เทคนิคที่ใช้ในการพัฒนา และหลักการทำงานของอุปกรณ์เกตเวย์ และชุดพัฒนาคอม 86 ซึ่งจะมีรายละเอียดดังในบทที่ 2 , 3 , 4 และ 5

จากนั้นก็จะนำความรู้ที่ได้ศึกษามาทั้งหมดนั้น นำมาวิเคราะห์และออกแบบระบบพร้อมทั้งซอฟต์แวร์ โดยการทำการศึกษาวิศวกรรมย้อนกลับ (Reverse Engineering) โครงสร้างของอุปกรณ์เกตเวย์ และ ทำการออกแบบรายละเอียดและโครงสร้างของอุปกรณ์เกตเวย์ใหม่ เพื่อให้เหมาะสมกับการใช้งานในชุดพัฒนาคอม86 โดยในการออกแบบนั้น จะออกแบบด้วยยูเอ็มแอล (UML : Unified Modeling Language) ซึ่งเป็นโมเดลที่สนับสนุนการวิเคราะห์และออกแบบระบบเชิงวัตถุ (Methodology) จากนั้นจะเข้าสู่ขั้นตอนของการพัฒนาซอฟต์แวร์ องค์ประกอบต่างๆของซอฟต์แวร์ ซึ่งจะมีรายละเอียดอยู่ในบทที่ 6

หลังจากผ่านขั้นตอนในส่วนของการพัฒนาซอฟต์แวร์แล้ว ต่อไปก็จะเป็นขั้นตอนของการทดสอบซอฟต์แวร์ที่สร้างขึ้นมา ตรวจสอบความถูกต้องว่าสามารถทำงานได้ตรงตามมาตรฐานที่ได้กำหนดไว้ในตอนต้น โดยทดสอบในสภาพแวดล้อมจำลองบนสมมติฐานที่ว่า หลักการทำงานของระบบเครือข่ายท้องถิ่นไร้สาย มีหลักการทำงานเช่นเดียวกับระบบเครือข่ายท้องถิ่น ซึ่งจะมีรายละเอียดการทดสอบอยู่ในบทที่ 7

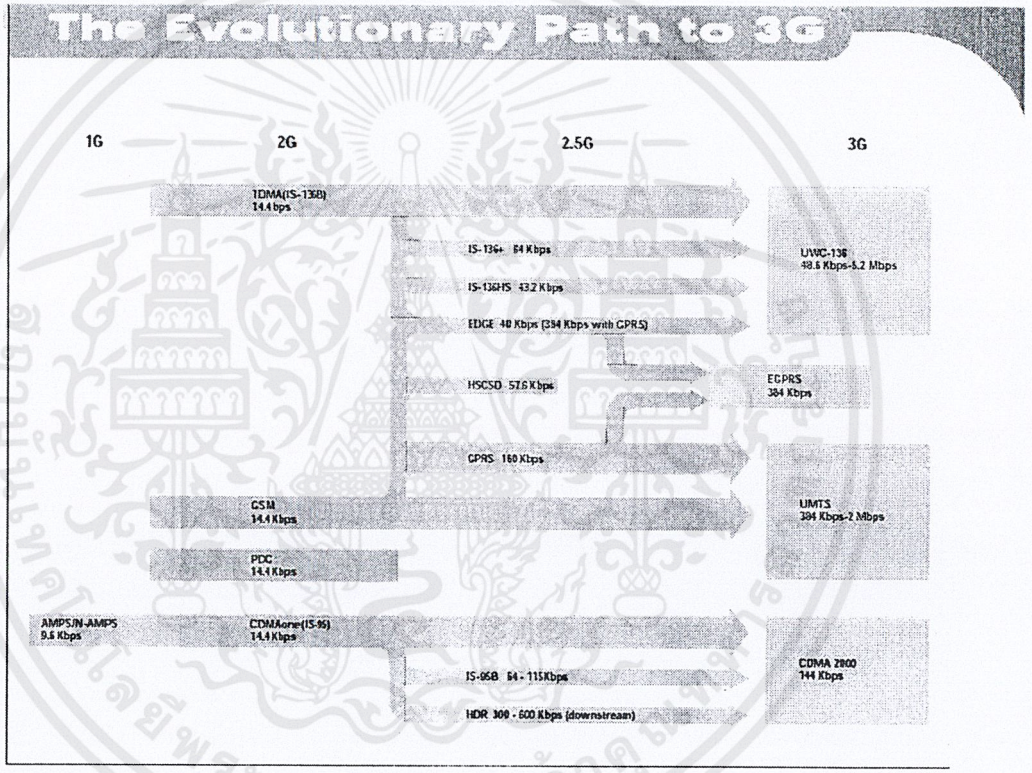
หลังจากนั้นก็จะนำผลลัพธ์ต่างๆที่ได้จากการทดสอบการทำงานของอุปกรณ์เกตเวย์ นำมาสรุปผลการทำงานซอฟต์แวร์ เป็นผลที่ได้รับจากงานวิจัยชิ้นนี้ และแนวทางในการพัฒนางานวิจัยนี้เพิ่มเติมและแนวทางในการนำไปประยุกต์ใช้ ซึ่งจะมีรายละเอียดอยู่ในบทที่ 8

สุดท้ายก็เป็นการรวบรวมข้อมูลข้อมูลที่ได้จากการวิจัยและพัฒนา มาทำเป็นเอกสารต่างๆ เช่น คู่มือการใช้งานอุปกรณ์เกตเวย์ คู่มือสำหรับการพัฒนาซอฟต์แวร์ของอุปกรณ์เกตเวย์

บทที่ 2
ทฤษฎีและหลักการของเทคโนโลยีไร้สาย

2.1 วิวัฒนาการของเทคโนโลยีไร้สาย

1. เทคโนโลยีการสื่อสารไร้สายในระดับขั้นการพัฒนา (1จี)
2. เทคโนโลยีการสื่อสารไร้สายในระดับขั้นการพัฒนา (2จี)
3. เทคโนโลยีการสื่อสารไร้สายในระดับขั้นการพัฒนา (2.5จี)
4. เทคโนโลยีการสื่อสารไร้สายในระดับขั้นการพัฒนา (3จี)



รูปที่ 2-1 วิวัฒนาการของเทคโนโลยีไร้สาย

1จี : ลักษณะของเครือข่าย ที่เป็น 1จี มีดังนี้

- ขึ้นอยู่กับการส่งสัญญาณ อนาล็อก
- ในอเมริกาเหนือเรียกว่า เอเอ็มพีเอส (AMPS : Analog Mobile Phone Systems)
- ในยุโรปและทั่วไปเรียกว่า ทีเอซีเอส (TACS : Total Access Communication Systems).
- เป็นเทคโนโลยีระบบอนาล็อกและเซอร์กิจ-สวิตซ์
- ออกแบบสำหรับการสื่อสารด้วยเสียง แต่ไม่ออกแบบสำหรับการสื่อสารด้วยข้อมูล

2จี : ลักษณะของเครือข่าย ที่เป็น 2จี มีดังนี้

- ขึ้นอยู่กับสัญญาณข้อมูลดิจิตอล ความถี่ต่ำ
- ส่วนใหญ่จะเรียกว่า จีเอสเอ็ม (GSM : Global Systems for Mobile Communications)
- เทคโนโลยี จีเอสเอ็มเป็นการเชื่อมต่อระหว่าง เอฟดีเอ็มเอ(FDMA : Frequency Division Multiple Access) และ ทีดีเอ็มเอ (TDMA : Time Division Multiple Access)
- เทคโนโลยีไร้สาย 2จี สามารถจัดการกับข้อมูลบางประเภทได้เช่นแฟกซ์ และ บริการส่งข้อความสั้นที่อัตราการส่ง ข้อมูลสูงถึง 9.6 กิโลบิตต่อวินาที แต่ไม่เหมาะสมสำหรับ แอปพลิเคชั่นประเภทมัลติมิเดีย

2.5จี : ลักษณะของเครือข่าย ที่เป็น 2.5จี มีดังนี้

- ขึ้นอยู่กับแพ็กเกต และเพิ่มความเร็วในการสื่อสารได้เร็วขึ้นถึง 384 กิโลบิตต่อวินาที
- ขึ้นอยู่กับเทคโนโลยีต่อไปนี้ : เอชเอสซีเอสดี (HSCSD : High Speed Circuit-Switched Data), จีพีอาร์เอส (GPRS : General Packet Radio Service) และ อีดีจีอี(EDGE : Enhanced Data Rates for Global Evolution)

3จี : ลักษณะของเครือข่าย ที่เป็น 3จี มีดังนี้

- สามารถรวม มาตรฐานเซลลูล่าร์ เช่น ซีดีเอ็มเอ จีเอสเอ็ม และ ทีดีเอ็มเอ ไว้ภายใต้มาตรฐานเดียวกันได้
- ประกอบด้วย 3 อินเทอร์เฟส คือ ซีดีเอ็มเอ ซีดีเอ็มเอ2000 และ ยูดับเบิ้ลยูซี-136 (Universal Wireless Communication)

2.2 เทคโนโลยีจีพีอาร์เอส

จีพีอาร์เอส (GPRS : General Packet Radio Service) เป็นเทคโนโลยีการสื่อสารไร้สายในระดับขั้นการพัฒนา 2.5จี หรือ 2จี+ ซึ่งขึ้นอยู่กับแพ็กเกต และถูกออกแบบให้สามารถทำงานขนานกับเทคโนโลยี 2จี ได้ คือ ระบบจีเอสเอ็ม ระบบพีดีซี และ ระบบทีดีเอ็มเอ ซึ่งใช้สำหรับการสื่อสารด้วยเสียง ที่มีรูปแบบการส่งสัญญาณเป็นสัญญาณดิจิตอลคล้ายในเทคโนโลยีการสื่อสารไร้สายในระดับขั้นการพัฒนา 2จี แต่มีการจัดส่งข้อมูลเป็นแพ็กเกต และมีอัตราเร็วในการรับส่งข้อมูลสูงถึง 115 กิโลบิตต่อวินาที

จีพีอาร์เอส ใช้หลักการส่งข้อมูลส่วนใหญ่มาจาก จีเอสเอ็ม แต่มีการปรับปรุงช่วงความถี่สัญญาณและจำนวนส่วนแบ่งช่วงเวลา (Timeslot) มากขึ้นต่อหนึ่งช่วงความถี่ และมีการจัดข้อมูลให้อยู่ในรูปแพ็กเกต แล้วส่งเข้าเครือข่ายส่วนกลางของระบบ (PLMN : Public Land Mobile Network) ซึ่งการส่งข้อมูลผ่านทางเครือข่ายจีพีอาร์เอส นี้จะรองรับการส่งสัญญาณที่เป็นข้อมูลมากขึ้นกว่าที่ระบบการสื่อสารไร้สายในปัจจุบันที่รองรับการส่งสัญญาณที่เป็นเสียงมากกว่าข้อมูล ทำให้ไม่เหมาะในการนำมาประยุกต์ใช้ในงานการสื่อสารที่ต้องมีการรับส่งข้อมูลมาก และต้องการความเร็วสูง เช่นการรับส่งไฟล์ข้อมูลต่างๆ เอสเอ็มทีพี/พีโอพี อีเมล์ เอฟทีพี และ เวบเซอร์วิส

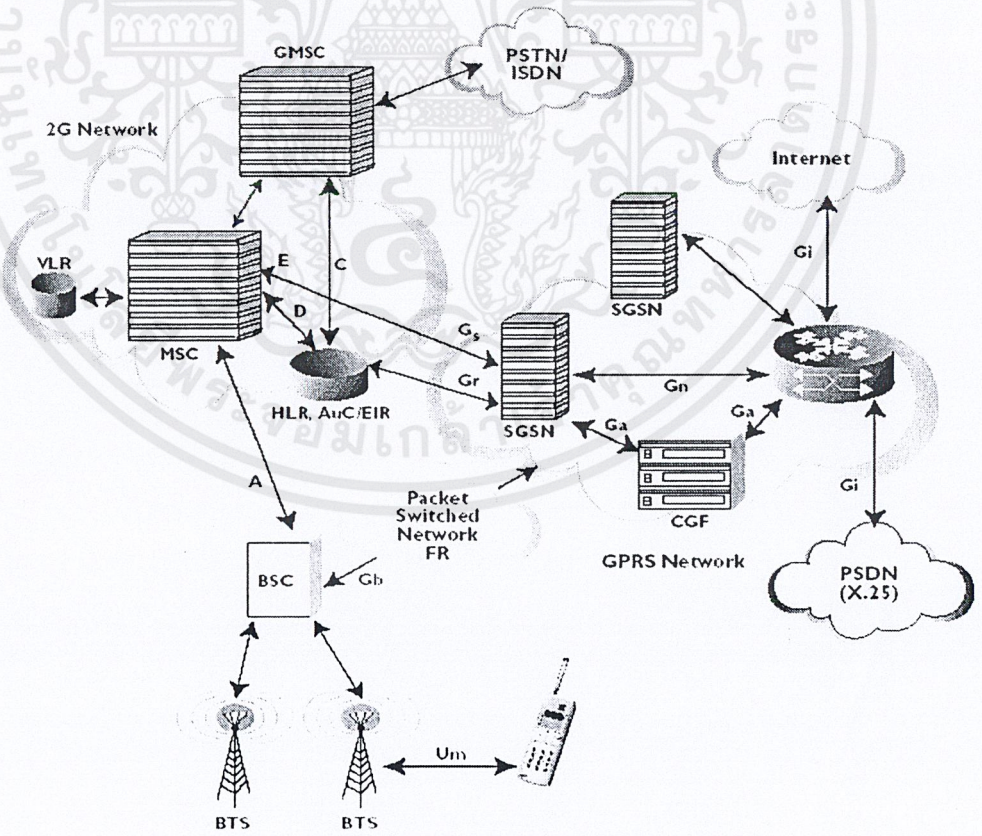
2.2.1 โครงสร้างของระบบเครือข่ายจีพีอาร์เอส

เครือข่ายจีพีอาร์เอส ประกอบด้วยส่วนหลักๆ ดังนี้

1. เครือข่ายส่วนกลางของระบบ (PLMN : IP –based Public Mobile Land Network)
2. สถานีบริการ (BSS : Base Station Service)
3. ศูนย์กลางการผลัดเปลี่ยน (MSC : Mobile Switching Center)
4. โทรศัพท์เคลื่อนที่ (MS : Mobile handsets)

และส่วนประกอบย่อยอื่นๆดังนี้

5. จุดสนับสนุนบริการจีพีอาร์เอส (SGSN : Service GPRS Support Nodes)
6. จุดสนับสนุนจีพีอาร์เอสเกตเวย์ (GGSN : Gateway GPRS Support Nodes)
7. ทะเบียนข้อมูลหลัก (HLR : Home Location Register)
8. ทำเบียนข้อมูลรอง (VLR : Visitor Location Register)
9. เครือข่ายข้อมูลแพ็กเกต (PDN : Packet Data Network)
10. สถานีทวนสัญญาณ (BTS : Base Transceiver Station)
11. สถานีควบคุม (BSC : Base Station Controller)



รูปที่ 2-2 โครงสร้างของเครือข่ายจีพีอาร์เอส

2.2.2 หลักการทำงานของระบบเครือข่ายจีพีอาร์เอส

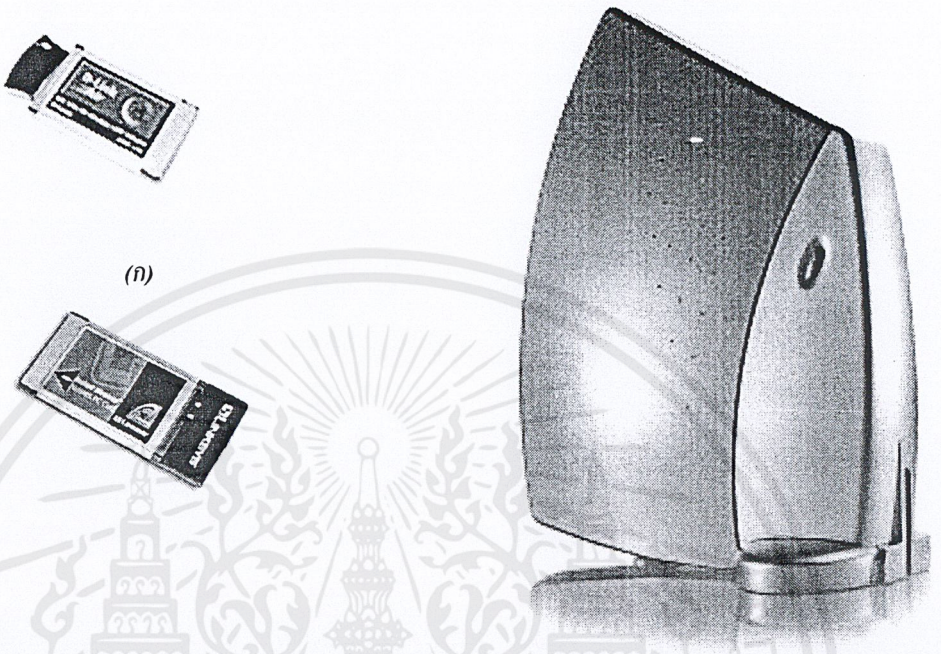
หลักการทำงานโดยรวมของเครือข่ายจีพีอาร์เอส เครือข่ายส่วนใหญ่โดยรวมจะคล้าย เครือข่ายของระบบ จีเอสเอ็ม แต่จะมีส่วนของสถานีควบคุม ที่เข้ามาช่วยในการแยกส่วนของสัญญาณที่เป็นข้อมูล และส่วนที่เป็นสัญญาณเสียงสำหรับการสื่อสารทางโทรศัพท์ทั่วไป แล้วส่งต่อไปยังส่วนของจุดสนับสนุนบริการจีพีอาร์เอส เพื่อส่งข้อมูลต่อออกไปทางเกตเวย์เข้าสู่เครือข่ายอินเตอร์เน็ตหรือเครือข่ายอื่นๆดังขั้นตอนต่อไปนี้

1. โทรศัพท์เคลื่อนที่ เริ่มกดเลือกบริการหรือหมายเลขผู้ให้บริการ
2. สถานีทวนสัญญาณ รับข้อมูลมาจากโทรศัพท์เคลื่อนที่ ส่งไปให้สถานีควบคุม
3. สถานีควบคุม ทำการเลือกเส้นทางการส่งข้อมูลถ้าเป็นข้อมูล จะส่งไปแบบแพ็กเกต ทางเครือข่ายส่งแพ็กเกต แต่ถ้าเป็น เสียง จะส่งไปที่ศูนย์กลางการผลัดเปลี่ยน
4. ทางด้าน ศูนย์กลางการผลัดเปลี่ยน จะส่งสัญญาณออกไปทาง เกตเวย์ศูนย์กลางการผลัดเปลี่ยนเพื่อออกไปยังชุมสายโทรศัพท์ทั่วไป
5. ส่วนทางด้านเครือข่ายส่งแพ็กเกต นั้นข้อมูลจะถูกส่งไปยังจุดสนับสนุนบริการจีพีอาร์เอสซึ่งจะดึง ข้อมูลผู้ใช้ จาก ทะเบียนข้อมูลหลักมาช่วยในการสถาปนาการเชื่อมต่อให้ง่ายขึ้น
6. จากนั้นจุดสนับสนุนบริการจีพีอาร์เอสจะส่งข้อมูลออกทาง จุดสนับสนุนจีพีอาร์เอสเกตเวย์เพื่อออกไปยังอินเทอร์เน็ต

2.3 ระบบเครือข่ายท้องถิ่นไร้สาย

ระบบเครือข่ายท้องถิ่นไร้สาย เป็นระบบเครือข่ายภายในที่มีขอบเขตจำกัดคล้ายกับเครือข่ายท้องถิ่น (LAN) ทั่วไป โดยมีมาตรฐานของ IEEE มากำหนดหลายมาตรฐาน อาทิเช่น IEEE802.11a, IEEE802.11b, IEEE802.11g เป็นต้น มาตรฐานทั้งหมดที่กล่าวมามีจุดเด่นละจุดที่ต้องปรับปรุงต่างกัน

ในเครือข่ายท้องถิ่นไร้สายนั้นมีส่วนประกอบที่สำคัญ 2 ส่วนคือ แอ็กเซสพอยน์ (access point) ซึ่งเป็นประตูเชื่อมเครือข่ายไร้สายเข้ากับระบบเคเบิลของอีเทอร์เน็ต ตัวแอ็กเซสพอยน์ทำหน้าที่แปลงทราฟฟิกบนสายเคเบิลให้กลายเป็นสัญญาณวิทยุส่งออกอากาศในย่านความถี่ 2.4 GHz (สำหรับผลิตภัณฑ์ 802.11b) หรือ 5 GHz (สำหรับผลิตภัณฑ์ 802.11a) และไม่ว่าไคล์เอ็นท์จะเป็นเดสก์ทอปหรือแลปทอป ก็ต้องใช้การ์ดอินเตอร์เฟซ สำหรับรับส่งสัญญาณในระบบเครือข่ายไร้สาย ซึ่งการ์ดนี้มีทั้งที่ติดตั้งตายตัวที่เครื่องไคล์เอ็นท์หรือถอดเปลี่ยนได้



(ก)

(ข)

รูปที่ 2-3 อุปกรณ์สำหรับเครือข่ายท้องถิ่นไร้สาย ได้แก่ การ์ดอินเตอร์เฟส 2-3(ก) แอ็กเซสพอยน์ 2-3(ข)

จากการทดสอบการใช้งานอุปกรณ์ที่ใช้ในระบบเครือข่ายท้องถิ่นไร้สาย พบว่าการทำงานของอุปกรณ์ดังกล่าว มีระยะรัศมีการสื่อสารประมาณ 200 – 500 ฟุต ในพื้นที่เปิดกว้าง และประมาณ 60 ฟุตในสำนักงานทั่วไปซึ่งมีการกั้นผนัง ซึ่งจะเห็นได้ว่าสามารถนำไปประยุกต์กับการใช้งานต่างๆ อาทิเช่นภายในห้องประชุมได้

บทที่ 3
ทฤษฎีและหลักการของโปรโตคอลที่เกี่ยวข้อง

สถาปัตยกรรมทีซีพี/ไอพี

3.1 แบบอ้างอิงทีซีพีไอพี

ระบบการสื่อสารข้อมูลในเครือข่ายคอมพิวเตอร์ประกอบด้วยทั้งฮาร์ดแวร์และซอฟแวร์ที่ซับซ้อน การมองภาพของระบบโดยรวมทั้งหมดเป็นหน่วยใหญ่ย่อมยากต่อการทำความเข้าใจ การใช้แบบอ้างอิงที่แบ่งระบบออกเป็นส่วนย่อยจะช่วยลดความซับซ้อนและสร้างความเข้าใจได้ง่ายกว่า

เครือข่ายคอมพิวเตอร์มีแบบอ้างอิงที่ใช้เป็นมาตรฐานคือ แบบอ้างอิงโอเอสไอ(OSI : Open Systems Interconnection Reference Model) ในขณะที่ทีซีพี/ไอพีเป็นโปรโตคอลที่กำเนิดก่อนโอเอสไอและมีแบบอ้างอิงเฉพาะตามรูปที่ 3-1 แบบอ้างอิงทีซีพี/ไอพีมีระดับชั้นจากล่างขึ้นบนและลักษณะสมบัติประจำชั้นต่างๆดังต่อไปนี้

ฟิสิคัล — ชั้นของการกำหนดคุณสมบัติฮาร์ดแวร์ เช่นคุณสมบัติทางกล (หัวต่อ และชนิดสายสื่อสาร) และคุณสมบัติทางไฟฟ้า (ลักษณะสัญญาณ และอัตราเร็ว) กล่าวโดยรวมแล้วระดับชั้นฟิสิคัลกำหนดวิธีการถ่ายโอนข้อมูลในระดับบิต ตัวอย่างของการเชื่อมต่อที่ตรงกับระดับชั้นฟิสิคัลได้แก่ อาร์เอส232 และ เอ็กซ์.21 เป็นต้น

เดทาลิงค์ — ชั้นของซอฟแวร์ (ดีไวซ์ไดร์เวอร์) และฮาร์ดแวร์ซึ่งทำงานด้วยการเชื่อมโยงเข้ากับสายสื่อสาร ตัวอย่างมาตรฐานในระดับชั้นนี้ได้แก่ อีเทอร์เน็ต และโทเค็นริงเป็นต้น

เน็ตเวอร์ค — ชั้นที่ทำหน้าที่เลือกเส้นทางเพื่อส่งข้อมูลระหว่างสถานีต้นทางและสถานีปลายทาง ตัวอย่างโปรโตคอลในชั้นนี้ได้แก่ ไอพี

ทรานสปอร์ต — ชั้นที่ทำหน้าที่จัดเตรียมการส่งข้อมูลระหว่างสถานีต้นทางและปลายทางโดยสถาปนาการเชื่อมต่อและรักษาสภาพการเชื่อมต่อ ตลอดจนยกเลิกการเชื่อมต่อเมื่อสิ้นสุดกระบวนการ และอาจมีหน้าที่เพิ่มเติมในการรับประกันความถูกต้องของข้อมูลที่จัดส่ง ทีซีพี/ไอพีมีโปรโตคอลประจำชั้นนี้จำนวนสองโปรโตคอลคือ ทีซีพีและยูดีพี

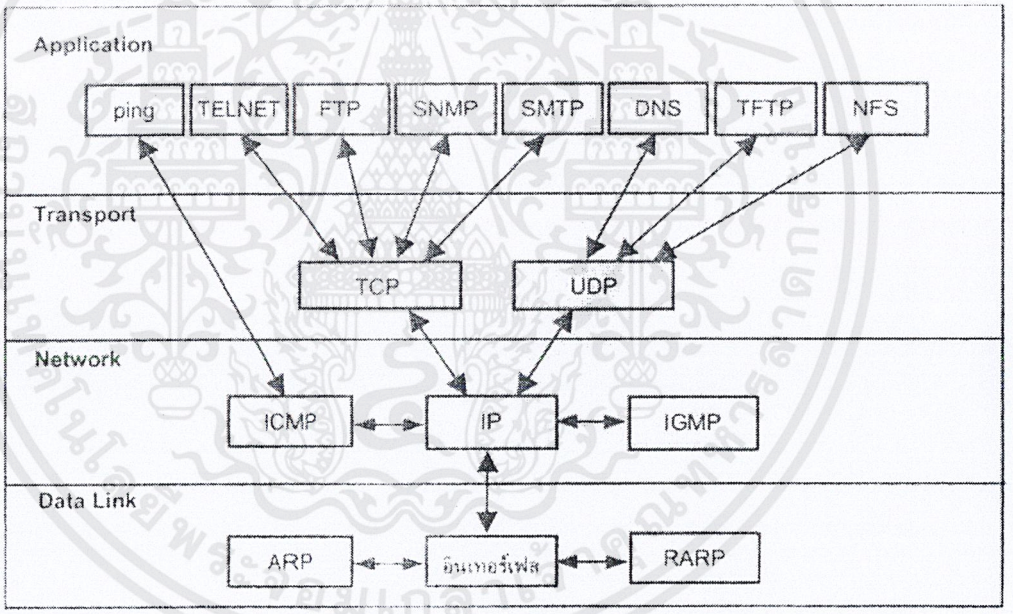
แอพลิเคชัน — ระดับชั้นนี้กำหนดการทำงานของโปรโตคอลประยุกต์ ตัวอย่างโปรโตคอลในระดับชั้นนี้ได้แก่ เอฟทีพี (FTP) เอสเอ็มทีพี (SMTP) หรือเทลเน็ต (TELNET) เป็นต้น

แอพลิเคชัน
ทรานสปอร์ต
เน็ตเวอร์ค
เดทาลิงค์
ฟิสิคัล

รูปที่ 3-1 แบบอ้างอิงทีซีพี/ไอพี

3.2 โปรโตคอลแสตค

การทำงานตามโปรแกรมประยุกต์หนึ่งๆไม่ได้ใช้โปรโตคอลพร้อมกันทั้งชุด หากแต่ใช้เพียงโปรโตคอลที่สัมพันธ์กันไปในแต่ละระดับชั้นของแบบอ้างอิง ตัวอย่างเช่นเทลเน็ตจะอาศัยทีซีพี และไอพีตามลำดับ การซ้อนทับของโปรโตคอลจากระดับชั้นบนไปยังชั้นล่างเรียกว่า โปรโตคอลแสตค (protocol stack) รูปที่ 3-2 แสดงโปรโตคอลแสตคของทีซีพี/ไอพี



รูปที่ 3-2 โปรโตคอลแสตคของทีซีพี/ไอพี

ไอพีซึ่งอยู่ในระดับชั้นเน็ตเวอร์คตามรูปที่ 3-2 เป็นแกนสำคัญของโปรโตคอลแสตคเนื่องจากทีซีพีและยูดีพีต้องใช้ไอพีเพื่อเลือกเส้นทางส่งแพ็กเกต ในระดับชั้นเน็ตเวอร์คยังมีไอซีเอ็มพีสนับสนุนการทำงานของไอพีเพื่อรายงานข้อผิดปกติที่เกิดขึ้นจากการส่งแพ็กเกตและมีไอจีเอ็มพีดูแลการจัดกลุ่มโฮสต์ในเครือข่ายมัลติคาสต์

ระดับชั้นทรานสปอร์ตมีสองโปรโตคอลสำคัญคือทีซีพีและยูดีพี เอพลิเคชันจะเลือกใช่ทีซีพีหรือยูดีพีตามลักษณะงาน เรื่องยูดีพีและทีซีพีจะกล่าวในหัวข้อ 3.2 และ 3.3ตามลำดับ

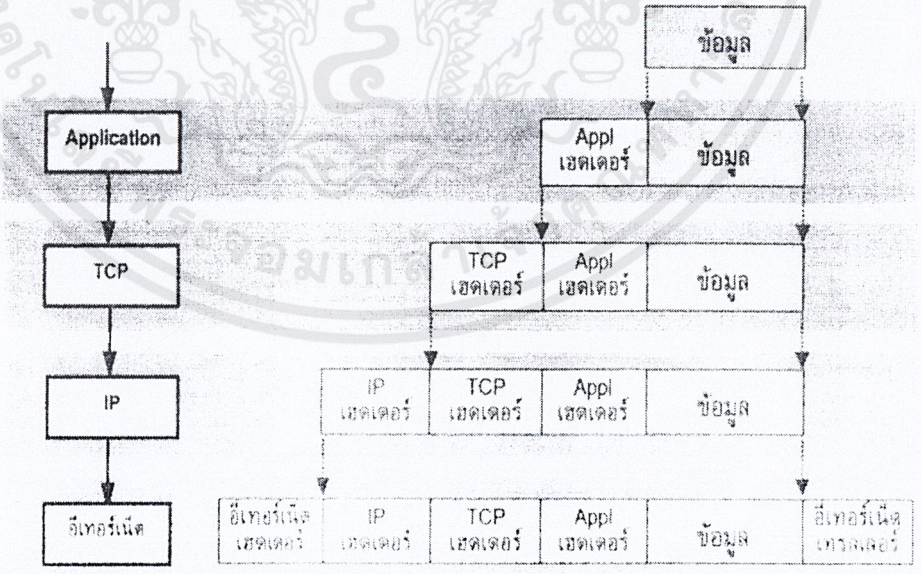
โปรโตคอลระดับล่างจากไอพีได้แก่ โปรโตคอลระดับเดทาลิงค์ซึ่งกำหนดการทำงานตามเทคโนโลยีเครือข่ายที่ใช้งานเช่น โปรโตคอล ซีเอสเอ็มเอ/ซีดี (CSMA/CD) ตามมาตรฐานอีเทอร์เน็ต ในระดับชั้นนี้โปรโตคอลในชุดของทีซีพี/ไอพีทำหน้าที่สนับสนุนการทำงานอยู่สองโปรโตคอลคือเออาร์พีและอาร์เออาร์พี ทั้งสองโปรโตคอลทำหน้าที่แปลงค่าระหว่างไอพีแอดเดรสกับฮาร์ดแวร์แอดเดรส

3.3 การส่งถ่ายข้อมูลระหว่างชั้น

โปรโตคอลในแต่ละชั้นล้วนมีหน้าที่เกี่ยวข้องในการส่งผ่านข้อมูลจากสถานีต้นทางไปยังสถานีปลายทาง ข้อมูลจะถูกส่งผ่านจากโปรโตคอลระดับบนสุดจากสถานีต้นทางไปยังระดับล่างจนข้อมูลถูกแปลงให้อยู่ในรูปสัญญาณไฟฟ้าแล้วเดินทางผ่านเครือข่ายไปยังสถานีปลายทาง โปรโตคอลระดับล่างสุดที่สถานีปลายทางจะรับสัญญาณและส่งผ่านขึ้นไปยังโปรโตคอลระดับบนต่อไป

เมื่อข้อมูลผ่านแต่ละระดับชั้น โปรโตคอลในชั้นนั้นจะผนวกข่างสารกำกับการทำงานประจำโปรโตคอลซึ่งเรียกว่า โปรโตคอลเฮดเดอร์ (protocol header) เข้ากับข้อมูล เฮดเดอร์และตัวข้อมูลจากระดับบนจะถูกส่งผ่านไปยังระดับล่าง โปรโตคอลระดับล่างจะมองเฮดเดอร์และตัวข้อมูลรวมเป็นเสมือนข้อมูลและเพิ่มเฮดเดอร์ประจำชั้นเข้าไป ข้อมูลเดิมจึงมีเฮดเดอร์หุ้มเป็นชั้นๆ กระบวนการนี้เรียกว่า การเอ็นแคปซูเลต (encapsulation) ตัวอย่างในรูปที่ 3-3 แสดงการเอ็นแคปซูเลตแพ็กเกตทีซีพี/ไอพีในอีเทอร์เน็ต

เมื่อสถานีปลายทางได้รับแพ็กเกตก็จะดำเนินการส่งไปตามลำดับชั้น โปรโตคอลประจำชั้นจะถอดเฮดเดอร์ออกและส่งส่วนที่เหลือไปยังชั้นถัดไป เฮดเดอร์จะถูกถอดออกเหลือเฉพาะข้อมูลเมื่อถึงชั้นบนสุด กระบวนการนี้เรียกว่า การดีแคปซูเลต (decapsulation)



รูปที่ 3-3 การห่อหุ้มข้อมูลตามลำดับโปรโตคอลแสตค

3.4 ไอพีแอดเดรส

อุปกรณ์ที่เชื่อมเข้าเครือข่ายและสามารถทำงานตามข้อกำหนดของทีซีพี/ไอพี จะต้องมีแอดเดรสประจำอุปกรณ์นั้น อุปกรณ์ที่กล่าวถึงนี้อาจเป็น โฮสต์ เราเตอร์ เครื่องพิมพ์หรือแม้กระทั่งอุปกรณ์สำนักงานยุคใหม่เช่นโทรศัพท์หรือเครื่องถ่ายเอกสาร ไอพีรุ่นสี่กำหนดให้ใช้ไอพีแอดเดรสขนาด 32 บิต อุปกรณ์ที่เชื่อมกับอินเทอร์เน็ตจะมีไอพีแอดเดรส 32 บิตประจำอินเทอร์เฟสที่ไม่ซ้ำกัน อุปกรณ์เช่นเราเตอร์มีหลายอินเทอร์เฟสจะมีไอพีแอดเดรสหลายค่าตามจำนวนอินเทอร์เฟสที่มีอยู่โดยไม่ซ้ำค่ากัน แต่คอมพิวเตอร์หรือโฮสต์ที่พบโดยทั่วไปมักมีเพียงอินเทอร์เฟสเดียวจึงมักเรียกว่าไอพีแอดเดรสเป็นแอดเดรสประจำโฮสต์

แอดเดรสขนาด 32 บิต ประกอบขึ้นมาจากหมายเลขสองส่วนคือ เลขเครือข่าย (network number หรือ network identifier หรือ netid) และ เลขโฮสต์ (host number หรือ host identifier หรือ hostid) เลขเครือข่ายสำหรับจัดคลาสเครือข่าย ส่วนเลขโฮสต์ใช้ระบุหมายเลขโอสต์ (หรืออีกนัยหนึ่งคืออินเทอร์เฟสของโฮสต์) ในเครือข่าย ไอพีแอดเดรสจึงแบ่งได้เป็นสองส่วนตามรูปที่ 3-4 จำนวนบิตที่ใช้สำหรับเลขเครือข่ายและเลขโฮสต์จะขึ้นอยู่กับคลาสที่สังกัด

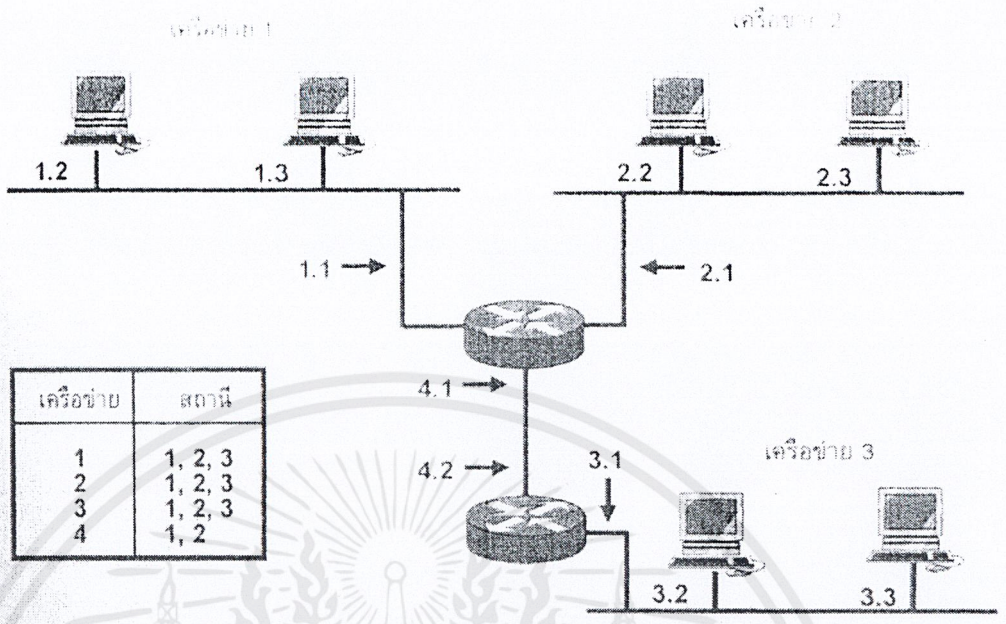
รูปที่ 3-4 รูปแบบของไอพีแอดเดรส

ในปัจจุบันฟิลด์กำหนดเลขเครือข่ายนิยมเรียกว่า พรีฟิกซ์เครือข่าย (network-prefix) เพราะทุกโฮสต์ในเครือข่ายจะต้องมีพรีฟิกซ์หรือบิตนำหน้าเหมือนกัน

1. ความสำคัญของเลขเครือข่ายและเลขโฮสต์

การจัดแบ่งไอพีแอดเดรสออกเป็นสองส่วนที่ประกอบด้วยเลขเครือข่ายและเลขโฮสต์ก็เพื่อประโยชน์ในการดูแลระบบ เราเตอร์จะอาศัยเลขเครือข่ายเพื่อเลือกเส้นทางส่งแพ็กเกตด้วยหลักการ โดยสังเขปดังต่อไปนี้

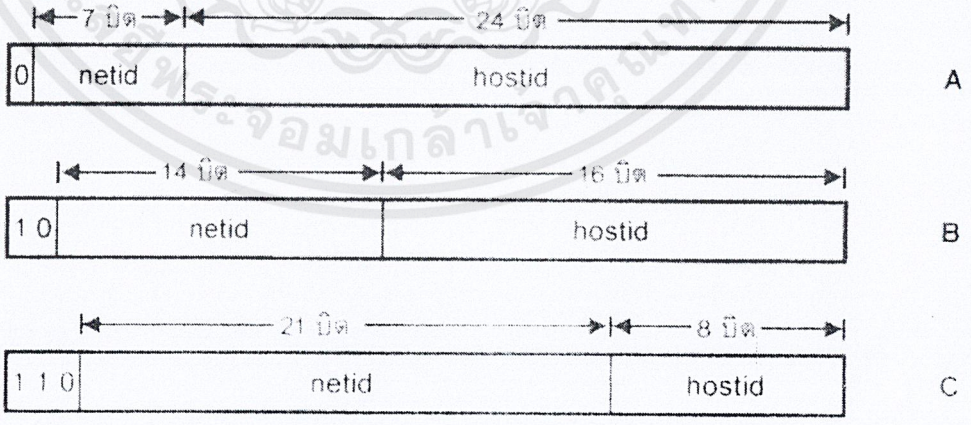
- โฮสต์ที่มีเลขเครือข่ายชุดเดียวกันย่อมอยู่ภายในเครือข่ายเดียวกัน และสามารถสื่อสารถึงกันโดยใช้เฟรมดาต้าลิงค์โดยไม่ต้องพึ่งพาเราเตอร์
- โฮสต์ที่มีเลขเครือข่ายต่างกันจะอยู่ต่างเครือข่ายกัน การสื่อสารระหว่างโฮสต์จะอาศัยเราเตอร์ที่เชื่อมเครือข่ายเป็นผู้นำส่งแพ็กเกต เราเตอร์อาจเชื่อมเครือข่ายที่อยู่ติดกันหรือส่งแพ็กเกตผ่านเราเตอร์อื่นไปยังปลายทางดังรูปที่ 3-5



รูปที่ 3-5 เราเตอร์เชื่อมโยงเครือข่ายที่มีเลขเครือข่ายต่างกัน

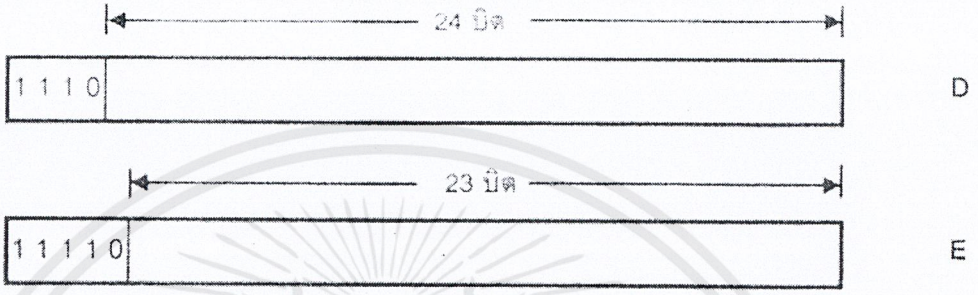
2. การจัดการคลาสเครือข่าย

ไอพีแอดเดรสมีการจัดแบ่งออกเป็นกลุ่มหรือคลาส (class) เครือข่ายที่ใช้งานในปัจจุบันมักสังกัดอยู่ในคลาสใดคลาสหนึ่งคือคลาส A, B หรือ C การแบ่งคลาสอาศัยจำนวนพรีฟิกซ์เครือข่ายที่แตกต่างกันตามรูปที่ 3-6 แต่ละคลาสจึงมีจำนวนเครือข่ายในสังกัดและจำนวนโฮสต์ต่อเครือข่ายไม่เท่ากัน



รูปที่ 3-6 การแบ่งคลาสเครือข่าย

การจัดคลาสตามรูปที่ 3-6 เป็นการจัดแบ่งตามการใช้งานเครือข่ายทั่วไป ในขณะที่ยังมีอีกสองคลาสซึ่งใช้เพื่อจุดประสงค์เฉพาะได้แก่ คลาส D และ E ดังรูปที่ 3-7 เครือข่ายคลาส D เป็นเครือข่ายแบบมัลติคลาส ส่วนคลาส E สงวนไว้ให้ใช้หากมีความจำเป็นอื่นใดในอนาคต ทั้งสองคลาสนี้ไม่ได้แบ่งเลขที่โฮสต์ จึงไม่กำหนดจำนวนโฮสตไว้



รูปที่ 3-7 การแบ่งคลาส D และ E

การแบ่งคลาสโดยใช้พรีฟิกซ์เป็นการผนวกข้อมูลเพื่อใช้ในการเลือกเส้นทาง เช่นหากตรวจพบว่าพรีฟิกซ์ 2 บิต แรกมีค่าเป็น 10 แสดงว่าเป็นแอดเดรสในคลาส B ซึ่งมีค่า 16 บิตแรกกำหนดกลุ่มเครือข่ายและ 16 บิตถัดมาเป็นเลขโฮสต์

3. ลักษณะสำคัญของแต่ละคลาส

คลาส A จะเริ่มจากเลข 0-127 หรือถ้าเป็นเลขฐานสองก็จะขึ้นต้นด้วย '0' ในส่วนไบต์แรกจะใช้ระบุหมายเลขที่อยู่เครือข่าย อีก 3 ไบต์หลังจะใช้ระบุหมายเลขที่อยู่โฮสต์ ในคลาส A นี้จะใช้กับเครือข่าย ที่ภายในประกอบด้วยโฮสต์จำนวนมาก ตัวอย่างของเครือข่ายในคลาสนี้เช่น

9.0.0.0	ibm.com
15.0.0.0	hp.com
20.0.0.0	csc.net

คลาส B จะเริ่มจากเลข 128-191 หรือถ้าเป็นเลขฐานสองก็จะขึ้นต้นด้วย '10' ในส่วนสองไบต์แรกจะใช้ระบุหมายเลขที่อยู่เครือข่าย อีก สองไบต์หลังจะใช้ระบุหมายเลขที่อยู่โฮสต์ มักใช้กับเครือข่ายขนาดกลางตัวอย่างของเครือข่ายในคลาสนี้เช่น

129.123.0.0	usu.edu
130.149.0.0	tu-berlin.de
158.108.0.0	ku.ac.th

คลาส C จะเริ่มจากเลข 192-223 หรือถ้าเป็นเลขฐานสองก็จะขึ้นต้นด้วย '110' ในส่วน

สามไบต์แรกจะใช้ระบุหมายเลขที่อยู่เครือข่าย อีก ไบต์สุดท้ายจะใช้ระบุหมายเลขที่อยู่โฮสต์ จึงใช้กับเครือข่ายที่มีขนาดเล็ก จำนวนเครื่องโฮสต์ไม่มาก ตัวอย่างของเครือข่ายในคลาสนี้เช่น

198.6.250.0 isoc.org

198.137.240.0 whitehouse.gov

203.144.152.0 mcot.or.th

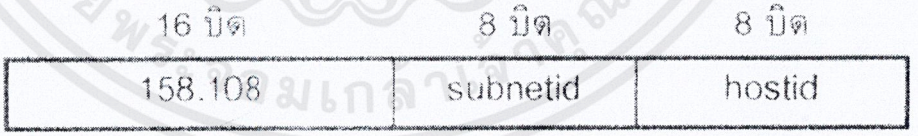
คลาส D จะเริ่มจากเลข 224-239 หรือถ้าเป็นเลขฐานสองก็จะขึ้นต้นด้วย '1110' ใช้กับแพ็กเกต ที่ต้องการส่งเป็นแบบหลายจุด

คลาส E จะเริ่มจากเลข 240-255 หรือถ้าเป็นเลขฐานสองก็จะขึ้นต้นด้วย '1111' สำรองไว้เพื่อความจำเป็นเฉพาะงานในอนาคต

4. การแบ่งเครือข่ายย่อย

เครือข่ายที่สังกัดในคลาส A และ B เป็นเครือข่ายที่มีจำนวนโฮสต์ได้เป็นจำนวนมากกล่าวคือ 16,777,214 และ 65,534 ตามลำดับ ในทางปฏิบัติแล้วเราไม่สามารถเชื่อมต่อโฮสต์ทั้งหมดในเครือข่ายเดี่ยวๆได้เพราะข้อจำกัดทางฮาร์ดแวร์ ผู้วางระบบจึงต้องจัดแบ่งเครือข่ายขนาดใหญ่ให้เล็กลงเป็นเครือข่ายย่อยหรือ ซับเน็ต (subnet) การแบ่งซับเน็ต นอกจากจะจัดจำนวนโฮสต์ให้เหมาะสมกับฮาร์ดแวร์ของเครือข่ายแล้วยังช่วยอำนวยความสะดวกในการบริหารเครือข่าย

การจัดซับเน็ต ใช้วิธีแบ่งบางส่วนของเลขโฮสต์มาใช้เป็น เลขซับเน็ต (subnetid) เพื่อกำหนดว่าเป็นเครือข่ายย่อยที่ใด ตัวอย่างเช่นเครือข่าย 158.108.0.0 ซึ่งอยู่ในคลาส B อาจใช้ 8 บิตแรกของเลขโฮสต์เป็นเลขซับเน็ต และ 8 บิตที่เหลือใช้สำหรับ เลขโฮสต์ ดังรูปที่ 3-8



รูปที่ 3-8 ตัวอย่างการแบ่งเครือข่ายย่อยของ 158.108

จำนวนบิตของเลขซับเน็ตเป็นตัวกำหนดจำนวนเครือข่ายย่อย ซับเน็ตขนาด 8 บิตสำหรับเครือข่าย 158.108.0.0 จะมี 254 ซับเน็ต แต่ละซับเน็ตมี 254 โฮสต์ดังรูปที่ 3-9 เลขซับเน็ตที่ทุกบิตเป็น'1' และ '0' จะสงวนไว้ใช้เฉพาะ ดังนั้นซับเน็ต 1580108.0.0 และ 158.108.255.0 จึงไม่นำมาใช้

ซับเน็ตที่	เครือข่ายย่อย	แอดเดรสเริ่มต้น	แอดเดรสสุดท้าย
1	158.108.1.0	158.108.1.1	158.108.1.254
2	158.108.2.0	158.108.2.1	158.108.2.254
3	158.108.3.0	158.108.3.1	158.108.3.254
:	:	:	:
:	:	:	:
252	158.108.252.0	158.108.252.1	158.108.252.254
253	158.108.253.0	158.108.253.1	158.108.253.254
254	158.108.254.0	158.108.254.1	158.108.254.254

รูปที่ 3-9 การจัดแบ่งเครือข่าย 158.108 ด้วยซับเน็ต 8 บิต

5. ซับเน็ตมาสค์

เมื่อผู้วางระบบเลือกขนาดซับเน็ตแล้วจะกำหนดพารามิเตอร์เพื่อใช้บอกให้โฮสต์และเราเตอร์ทราบว่าซับเน็ตที่ใช้งานมีขนาดกี่บิต ค่านี้เรียกว่า ซับเน็ตมาสค์ (subnet mask)

ซับเน็ตมาสค์มีขนาด 32 บิต ซึ่งจะมีบิตที่ตรงกับเลขเครือข่ายและเลขซับเน็ตเท่ากับ '1' ส่วนบิตที่ตรงกับเลขโฮสต์มีค่าเท่ากับ '0' การเลือกซับเน็ตมาสค์ควรใช้ค่าที่มีบิต '1' อยู่ติดกันจากซ้ายมือไปขวามือเสมอ

ตัวอย่างเช่นเครือข่าย 158.108.0.0 ซึ่งแบ่งให้มีเลขซับเน็ตและเลขโฮสต์อย่างละ 8 บิตจะมีค่าซับเน็ตมาสค์เท่ากับ 255.255.255.0 ค่านี้คำนวณได้จากการเขียนไอพีแอดเดรสทั้งสี่หลัก และใส่เลขฐานสองค่า '1' ให้ครบทุกบิตที่เป็นเลขเครือข่ายและเลขซับเน็ต จากนั้นให้ใส่ค่า ' 0' สำหรับเลขโฮสต์ แล้วจึงแปลงเลขฐานสองที่ได้เป็นเลขฐานสิบตามขั้นตอนต่อไปนี้ดังรูปที่ 3-10

	8 บิต	8 บิต	8 บิต	8 บิต
(1) ไอพีแอดเดรส	158	108	subnetid	hostid
(2) กำหนดบิต "1" และ "0"	11111111	11111111	11111111	00000000
(3) แปลงเป็นฐานสิบ	255	255	255	0

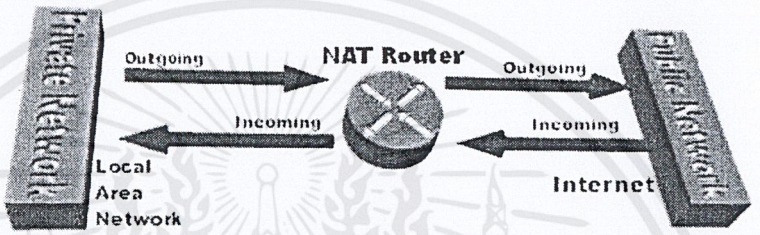
รูปที่ 3-10 การคำนวณหาค่าซับเน็ตมาสค์

เครือข่าย 158.108.0.0 ซึ่งใช้ซับเน็ตมาสค์เท่ากับ 255.255.255.0 เรียกว่ามีซับเน็ตมาสค์ 24 บิต เนื่องจากมีบิตที่มีค่า '1' จำนวน 24 บิต หรือเขียนตามรูปแบบที่นิยมใช้ในปัจจุบันคือ 158.108.0.0/24 โดยเรียกว่าเครือข่าย 158.108.0.0 มีพรีฟิกซ์ 24 บิต

บทที่ 4
ทฤษฎีและหลักการของเทคนิคที่เกี่ยวข้อง

4.1 การแปลและแปลงไอพีแอดเดรสของเครือข่าย

การแปลและแปลงไอพีแอดเดรสของเครือข่าย (NAT : Network Address Translation) เป็นวิธีการหนึ่งในการแปลงและแปล ไอพีแอดเดรส ของ เครือข่ายส่วนตัวให้เป็น ไอพีแอดเดรส ซึ่งเป็นที่ยอมรับและสื่อสารบนอินเทอร์เน็ต ดังรูปที่ 4-1

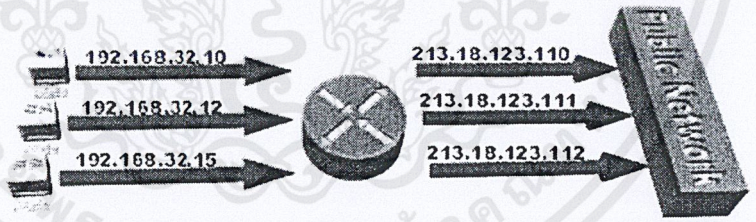


รูปที่ 4-1 วิธีการแปลและแปลงไอพีแอดเดรสของเครือข่าย

โดยมีรูปแบบการแปลและแปลงไอพีแอดเดรสของเครือข่ายอยู่ 4 รูปแบบคือ

1. การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบตายตัว

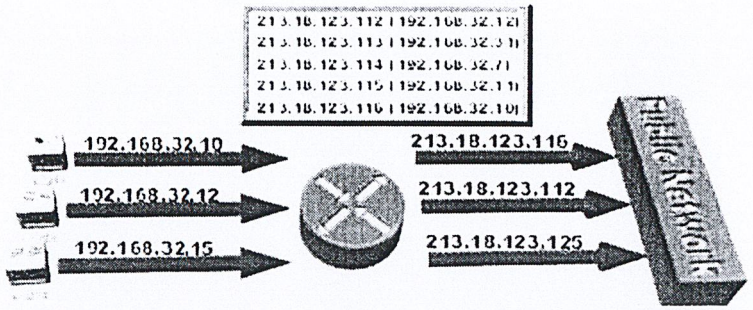
เป็นการแมปไอพีแอดเดรสที่ไม่ได้ลงทะเบียน กับ ไอพีแอดเดรสที่ลงทะเบียนแบบ หนึ่ง ต่อ หนึ่ง เมื่อต้องการออกสู่เครือข่ายสาธารณะ ดังรูปที่ 4-2



รูปที่ 4-2 การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบตายตัวโดยไอพีแอดเดรส 192.168.32.10 จะถูกแปลงเป็น 213.18.123.110 เสมอ

2. การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบไดนามิก

เป็นการแมปไอพีแอดเดรสที่ไม่ได้ลงทะเบียน กับ ไอพีแอดเดรสที่ลงทะเบียนจากกลุ่มของไอพีแอดเดรสที่ลงทะเบียน ดังรูปที่ 4-3

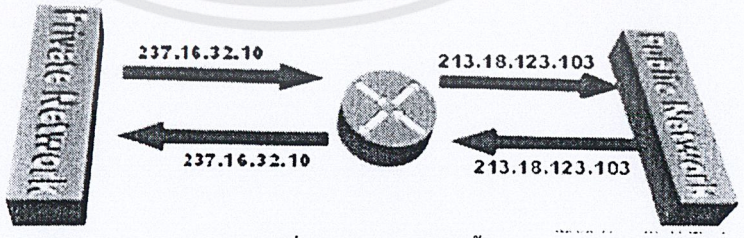


รูปที่ 4-3 การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบไดนามิกโดยไอพีแอดเดรส 192.168.32.10 จะถูกแปลงเป็นไอพีแอดเดรสในช่วงจาก 213.18.123.100 ถึง 213.18.123.150

3. โอเวอร์โหลดดิ้ง เป็นรูปแบบ การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบไดนามิกที่แมปไอพี แอดเดรสที่ไม่ลงทะเบียนหลายค่าเป็นไอพีแอดเดรสที่ลงทะเบียนค่าเดียวแต่ใช้ พอร์ตต่างกัน นิยมเรียกว่า พีเอที (PAT : Port Address Translation) ดังรูปที่ 4-4

รูปที่ 4-4 คอมพิวเตอร์แต่ละเครื่องในเครือข่ายส่วนตัวจะถูกแปลงเป็นไอพีแอดเดรสเดียวกัน (213.18.123.100) แต่หมายเลขพอร์ตต่างกัน

4. โอเวอร์แลปปิ้ง เมื่อไอพีแอดเดรสที่ใช้ในเครือข่ายภายใน เป็นไอพีแอดเดรสที่ถูกใช้กับเครือข่ายอื่น เราเตอร์ต้องทำการปรับปรุง ตารางลุคอัพ และทำการแปลงไอพีแอดเดรสที่ลงทะเบียนจากภายนอกเป็นไอพีแอดเดรสที่ไม่ลงทะเบียนจากภายใน เมื่อข้อมูลถูกส่งเข้าเครือข่ายภายในซึ่งสามารถใช้ได้ทั้ง การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบตายตัว และ การแปลและแปลงไอพีแอดเดรสของเครือข่ายแบบไดนามิก ดังรูปที่ 4-5



รูปที่ 4-5 โอเวอร์แลปปิ้ง

ข้อดีของการแปลและแปลงไอพีแอดเดรสของเครือข่าย

1. เป็นซ่อน ไอพีแอดเดรส เพื่อความปลอดภัย
2. ไอพีแอดเดรสเดียว สามารถใช้ได้กับหลายเครื่องเพราะสามารถแปลง ไอพีแอดเดรส ก่อนออกสู่เครือข่ายสาธารณะ

ข้อเสียของการแปลและแปลงไอพีแอดเดรสของเครือข่าย

1. ยากแก่การติดตามว่ามาจากไอพีแอดเดรสใด
2. เกิด เวลาดีเลย์ เมื่อมีการออกสู่ เครือข่ายภายนอก หลายๆเครื่องพร้อมกัน

4.2 มัลติทาสก์

เป็นโหมดการทำงานที่อนุญาตให้สามารถทำงานพร้อมๆกันได้ หรือ ประมวลผลแบบสอดแทรกกันของสองทาสก์ หรือมากกว่า คล้ายกับมัลติโปรแกรมมิ่งที่มีรูปแบบแตกต่างกัน

บทที่ 5
ทฤษฎีและหลักการทำงานของชุดพัฒนาคอม86 เบื้องต้น

หลักการทำงานของชุดพัฒนาคอม 86 เบื้องต้น

ชุดพัฒนาคอม86 เป็นชุดเครื่องมือสำหรับใช้ในการพัฒนาอุปกรณ์ทางด้านระบบฝังตัวที่อยู่บนพื้นฐานการพัฒนาโดยการใช้แพลตฟอร์มที่มีไมโครโพรเซสเซอร์ตระกูล x86 ซึ่งเป็นที่คุ้นเคยกับผู้ใช้งานในปัจจุบันเป็นส่วนประกอบหลักในการทำงาน

5.1 จุดเด่นของคอม86

5.1.1 ใช้ไมโครคอนโทรลเลอร์ที่มีความสามารถสูง

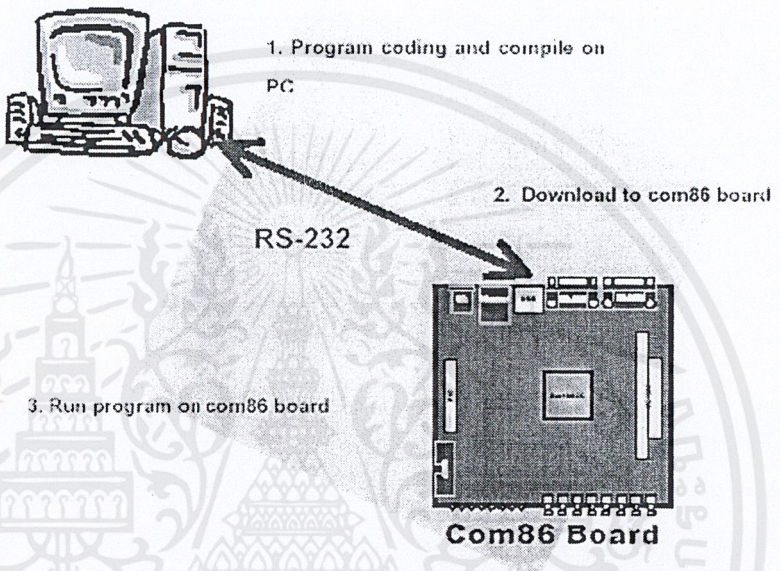
ภายในชุดพัฒนาคอม86 มีบอร์ดคอม86 ซึ่งได้เลือกใช้ไมโครคอนโทรลเลอร์ Am186CC ของบริษัทเอเอ็มดี (AMD) ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 16 บิตทำงานอยู่ที่ความเร็ว 24 เมกกะเฮิร์ต (MHz) ทำให้มีความสามารถในการประมวลผลต่างๆเพิ่มมากขึ้นเมื่อเทียบกับการพัฒนาโดยใช้ไมโครคอนโทรลเลอร์ขนาด 8 บิต ที่นิยมใช้กันอยู่ในปัจจุบัน นอกจากนี้ภายในตัวไมโครคอนโทรลเลอร์เองยังได้รวมเอาอุปกรณ์พื้นฐานต่างๆที่จำเป็นในการใช้งาน เช่น ดีแรมคอมโทรลเลอร์ (DRAM controller), อินเทอร์รัพคอนโทรลเลอร์ (Interrupt controller), ยูอาร์ต (UART) เข้าไว้ภายในตัวไมโครคอนโทรลเลอร์ซึ่งทำให้ง่ายต่อการออกแบบอุปกรณ์เพื่อทำเป็นสินค้าภายหลังการพัฒนาตัวอย่างเช่น การนำบอร์ดคอม86 ไปพัฒนาเป็นอุปกรณ์อย่างหนึ่งที่ต้องการควบคุมการทำงานผ่านทางยูเอสบีพอร์ต (USB port) จะสามารถทำได้โดยง่ายเนื่องจากไม่ต้องการออกแบบวงจรควบคุมการทำงานในส่วนยูเอสบี นี้เพิ่มอีก เพียงแต่ออกแบบวงจรการทำงานส่วนอื่นที่ต้องการเพิ่มเติมทำให้ทุ่นเวลาในการพัฒนาเป็นต้น นอกจากนี้ภายในไมโครคอนโทรลเลอร์ยังเพิ่มส่วนโมดูล (Module) ทางด้านการสื่อสารต่างๆ เข้าไปภายในตัว ทำให้เป็นไมโครคอนโทรลเลอร์ที่มีความสามารถเด่นทางด้านการสื่อสารเหมาะกับการพัฒนาอุปกรณ์ที่ต้องใช้คุณสมบัติการสื่อสารต่างๆ ซึ่งจะสามารถทำได้ง่ายเมื่อเทียบกับการใช้ไมโครคอนโทรลเลอร์ขนาดเล็กแบบเดิมซึ่งมีข้อจำกัดต่างๆ อยู่มากมาย

5.1.2 มีเครื่องมือในการพัฒนาโปรแกรมมาก

เนื่องจากการพัฒนาโปรแกรมโดยใช้ชุดพัฒนาคอม86 นั้นเป็นเสมือนกับการพัฒนาโปรแกรมบนของคอมพิวเตอร์ส่วนบุคคลทั่วไป ดังนั้นจึงสามารถนำโปรแกรมเครื่องมือต่างๆที่ใช้พัฒนาโปรแกรมสำหรับคอมพิวเตอร์ส่วนบุคคลมาใช้งานได้มากมาย และโปรแกรมเครื่องมือเหล่านี้ก็เป็นที่คุ้นเคยของนักพัฒนาจึงทำให้สามารถพัฒนาอุปกรณ์ได้อย่างรวดเร็ว ลดต้นทุนในการพัฒนาต่างๆได้เป็นจำนวนมากนอกจากนี้ภายในชุดพัฒนายังมีชุดซอฟต์แวร์เครื่องมือต่างๆที่จำเป็นสำหรับช่วยในการพัฒนาโปรแกรมด้วยอีกทางหนึ่ง

5.1.3 ง่ายในการเรียนรู้และพัฒนา

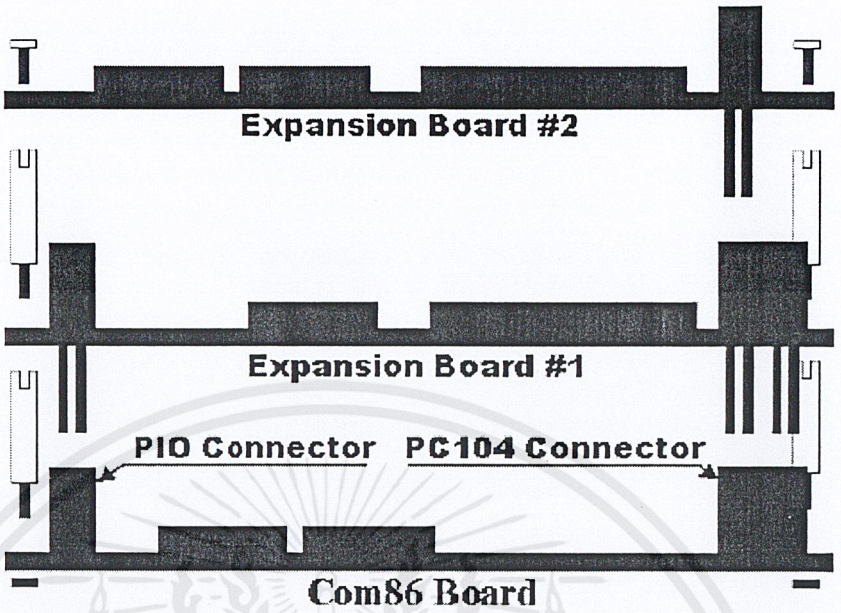
บอร์ดคอม86 ภายในชุดพัฒนาคอม86 นี้มีสถาปัตยกรรมของไมโครคอนโทรลเลอร์แบบ x86 ซึ่งเป็นสถาปัตยกรรมเดียวกับคอมพิวเตอร์ส่วนบุคคลทั่วไป โดยมีชุดคำสั่งที่เข้ากันได้กับชุดคำสั่งของ 80286 (Real mode) ทำให้ง่ายต่อการเรียนรู้ในการพัฒนาโปรแกรมต่างๆ และยังสามารถพัฒนาและทดสอบซอฟต์แวร์ต่างๆได้บนคอมพิวเตอร์ ก่อนที่จะทดสอบกับบอร์ดคอม86 ทำให้สามารถพัฒนาอุปกรณ์ทางด้านระบบฝังตัวต่างๆได้อย่างรวดเร็ว ซึ่งรูปแบบการพัฒนามีลักษณะเป็นไปดังภาพ



รูปที่ 5-1 แสดงถึงรูปแบบการพัฒนาโปรแกรมโดยใช้บอร์ดคอม86

5.1.4 มีความสามารถในการเพิ่มขยายได้

ในส่วนของการเพิ่มขยายของฮาร์ดแวร์ (Hardware) สามารถทำได้โดยผ่านทางคอนเนคเตอร์ (Connector) ของบอร์ดในรูปแบบของมาตรฐาน (พีซีหนึ่งศูนย์สี่) PC/104 ซึ่งจะมีการเชื่อมต่อแบบบัสอินเทอร์เฟส (BUS Interface) แบบไอเอสเอบัส (ISA BUS) ขนาด 8 บิต คล้ายกับไอเอสเอบัส ที่ มีอยู่ในคอมพิวเตอร์ส่วนบุคคลทั่วไป ซึ่งผู้พัฒนาสามารถออกแบบฮาร์ดแวร์ให้เป็นลักษณะของบอร์ดขยายและนำมาเชื่อมต่อกับบอร์ดคอม86 และเพียงเขียนซอฟต์แวร์ไดร์เวอร์ (Software Driver) สำหรับควบคุมการทำงานของส่วนเพิ่มเข้ามาใหม่ซึ่งจะทำงานร่วมกับระบบปฏิบัติการในการรองรับการทำงานของแอพพลิเคชันต่างๆที่เพิ่มเข้ามา นอกจากนี้ตัวบอร์ดคอม86 ยังสามารถเชื่อมต่อกับอุปกรณ์ภายนอกในลักษณะของพีไอโอ (PIO) หรือ โปรแกรมเมเบิลอินพุท/เอาท์พุท (Programmable Input/Output) เหมือนกันการใช้งานไมโครคอนโทลเลอร์รุ่นอื่นๆได้ด้วยเช่นกัน



รูปที่ 5-2 แสดงการเชื่อมต่อของบอร์ดขยายกับบอร์ดคอม86

5.2 แอพพลิเคชันเป้าหมายของชุดพัฒนา (Target Application)

จากความสามารถของไมโครคอนโทรลเลอร์ของบอร์ดคอม86 ทำให้การประยุกต์ใช้งานกับแอพลิเคชั่นต่างๆนั้น สามารถทำได้มากมาย โดยตัวอย่างแอพพลิเคชันต่างๆที่เหมาะแก่การพัฒนาโดยบอร์ดคอม86 มีดังนี้

5.2.1 เกี่ยวกับอุตสาหกรรม (Industrial control and Automation)

ตัวอย่างของอุปกรณ์ควบคุมภายในโรงงาน เช่น การควบคุมเครื่องจักรจากระยะไกล การตรวจวัดสภาวะต่างๆ ภายในโรงงานและส่งกลับไปยังศูนย์ควบคุม โดยการใช้บอร์ดคอม86 เป็นอุปกรณ์ที่ช่วยทำหน้าที่ในการสื่อสาร ควบคุมสภาวะแวดล้อม หรือเครื่องจักรต่างๆภายในโรงงานและรายงานกลับไปยังศูนย์กลางตามกำหนดเวลา เป็นต้น

5.2.2 อุปกรณ์ยูเอสบี (USB Peripheral)

อุปกรณ์ยูเอสบีต่างๆ สามารถพัฒนาได้โดยการใช้บอร์ดคอม86 ซึ่งภายในบอร์ดคอม86 นี้มียูเอสบีคอนโทรลเลอร์ภายในตัวไมโครคอนโทรลเลอร์ Am186CC สามารถเชื่อมต่อเข้ากับคอมพิวเตอร์ได้ที่ความเร็ว 12 เมกกะบิตต่อวินาที (Mbps) ซึ่งการพัฒนาเป็นอุปกรณ์ ยูเอสบี ต่างๆ โดยบอร์ดคอม86 นี้สามารถทำได้โดยไม่ต้องมีการต่อวงจรทางด้านยูเอสบีใดๆ เพิ่มเติมอีก

5.2.3 เว็บเซิร์ฟเวอร์ (Embedded web server)

เอ็มเบดเดดเว็บเซิร์ฟเวอร์ (Embedded web server) สามารถสร้างได้บนบอร์ดคอม86 โดยไม่จำเป็นต้องเชื่อมต่อวงจรอื่นๆ เพิ่มเติม ซึ่งบอร์ดคอม86 นี้สามารถสื่อสารกับภายนอกผ่านทางอีเทอร์เน็ตพอร์ตที่อยู่บนบอร์ดที่ความเร็ว 10 เมกกะบิตเปอร์เซ็ก ทำให้สามารถนำไปประยุกต์ใช้งานในกาออกแบบอุปกรณ์ควบคุมต่างๆ ผ่าน เอ็มเบดเดดเว็บเซิร์ฟเวอร์ ที่ทำงานอยู่บนบอร์ดคอม86 ได้โดยง่าย

5.2.4 โรโบติก (Robotics)

บอร์ดคอม86 มีไมโครคอนโทรลเลอร์ที่มีประสิทธิภาพสูงในการทำงาน การพัฒนาซอฟต์แวร์สามารถพัฒนาได้ซับซ้อนมากยิ่งขึ้น การเชื่อมต่อขยายทางด้านฮาร์ดแวร์ต่างๆสามารถทำได้โดยง่าย เหมาะสำหรับการพัฒนาหุ่นยนต์ที่มีการทำงานที่ซับซ้อน

5.2.5 การควบคุมบ้านหรือสิ่งก่อสร้าง (Home and building control)

อุปกรณ์สำหรับควบคุมระบบต่างๆภายในบ้าน หรืออาคารสามารถนำบอร์ดคอม86 ไปประยุกต์ใช้งานได้เช่นกัน โดยใช้เป็นส่วนในการควบคุมการทำงานของอุปกรณ์ต่างๆ หรือคอยตรวจสอบการทำงานของเซนเซอร์ (Senser) ต่างๆ ภายในบ้าน และแจ้งไปยังเจ้าของบ้านเมื่อเกิดเหตุร้ายขึ้น

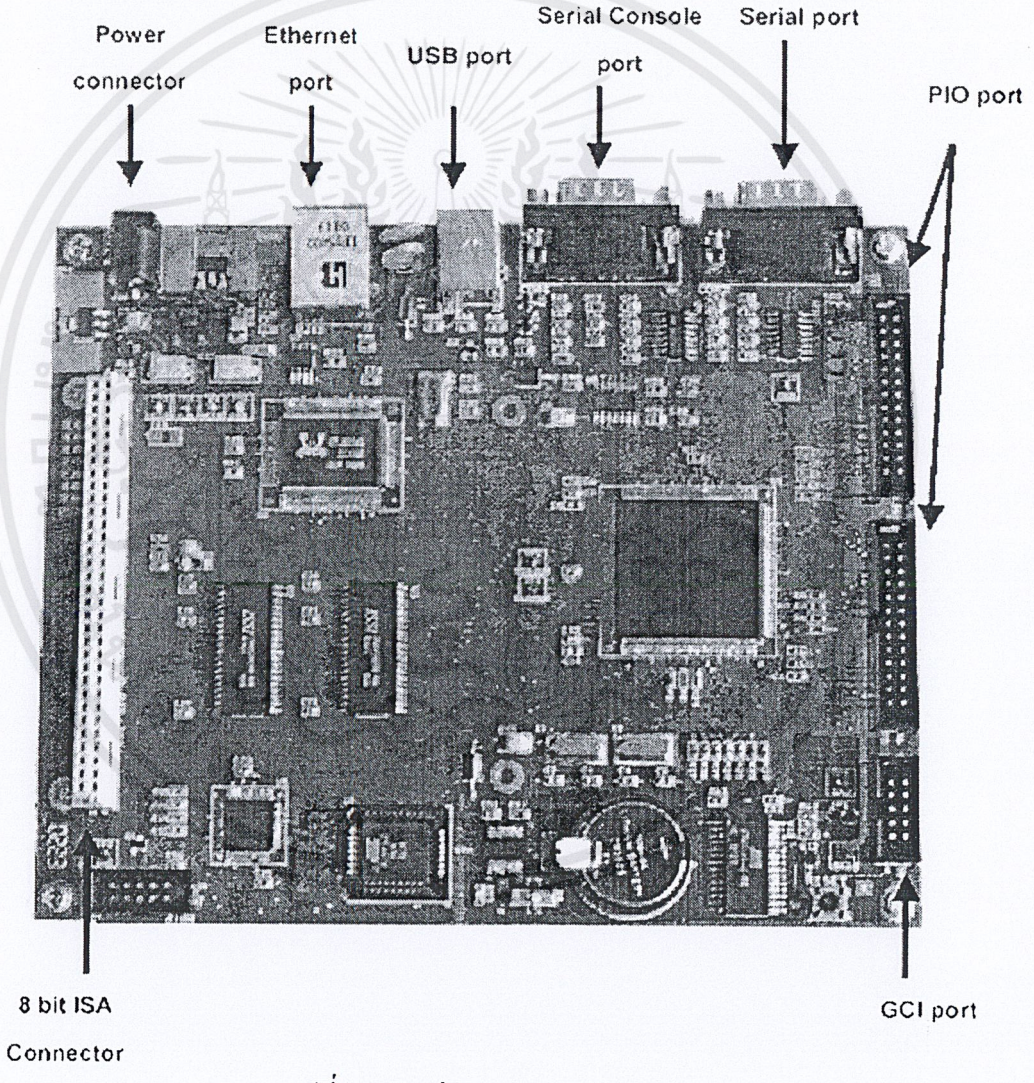
นอกจากนี้ตัวไมโครคอนโทรลเลอร์ยังรองรับการนำไปสร้างเป็นอุปกรณ์ต่างๆ ที่เกี่ยวข้องกับงานทางด้านการสื่อสารต่างๆเช่น ไอเอสดีเอ็นโมเด็มแอนด์เทอมินัลอแดบเตอร์ (ISDN Modem and Terminal Adaptor), โลเอนด์เราเตอร์ (Low-end Router), ไลน์การ์ดแอพพลิเคชั่น (Line card application), เอ็กซ์ดีเอสแอลแอพพลิเคชั่น (xDSL application), ดิจิตอลคอร์ดโฟน (Digital corded phone) เป็นต้น ซึ่งสามารถนำบอร์ดคอม86 ไปประยุกต์ใช้งานในการออกแบบพัฒนาได้เช่นกัน

5.3 คุณสมบัติของบอร์ดคอม86

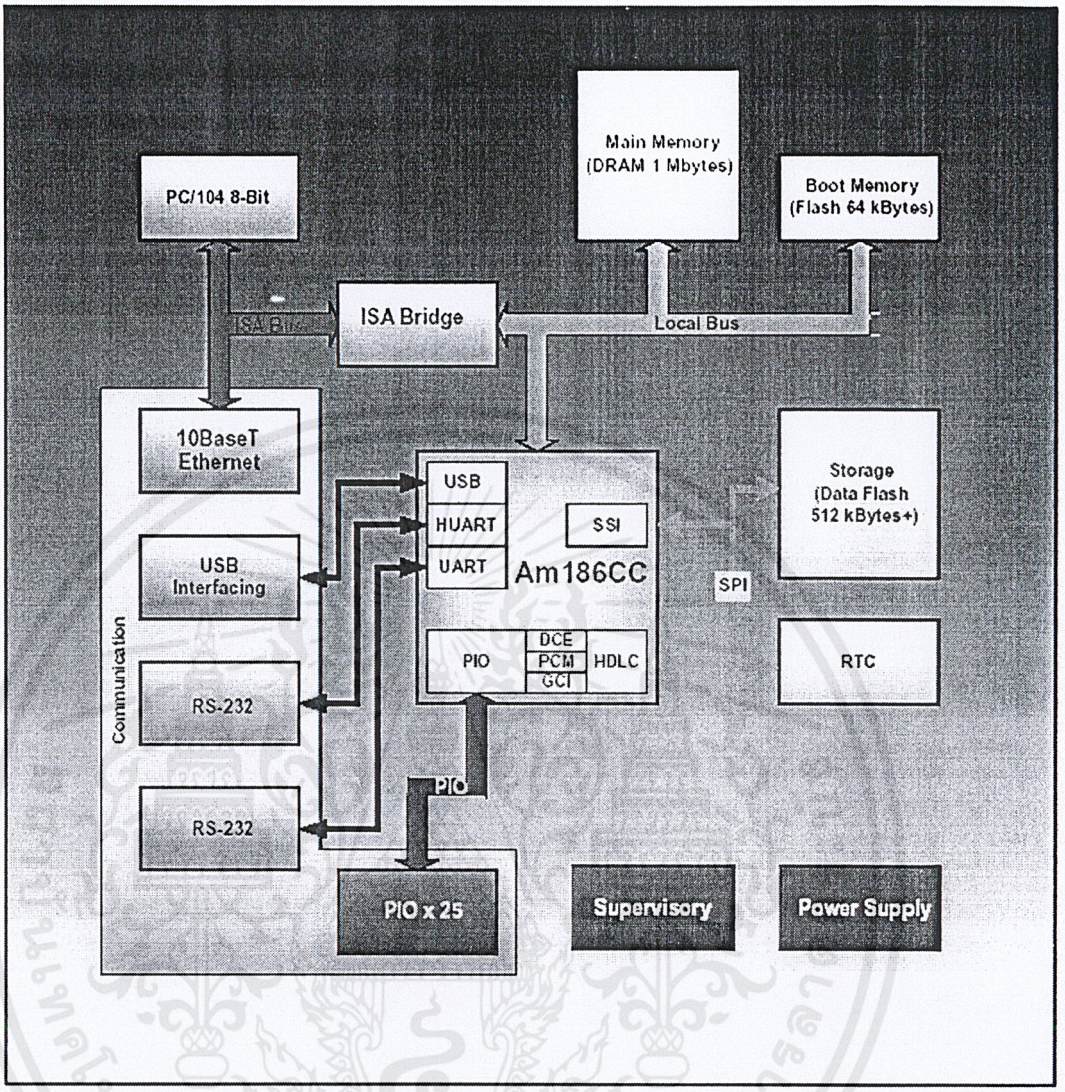
บอร์ดคอม86มีคุณสมบัติดังนี้

- ซีพียู เอเอ็มหนึ่งแปดหกซีซี (CPU Am186CC)
- ดีแรม 1 เมกกะไบต์ (1 MBytes DRAM)
- แฟลตไบออส 64 กิโลไบต์ (64 kBytes Flash BIOS)
- หน่วยความจำ 512 กิโลไบต์
- ฟูลดูเพล็ก IEEE 802.3 บนบอร์ดอีเทอร์เน็ตคอนโทรลเลอร์ด้วย RJ45 คอนเน็คเตอร์
- ยูเอสบี คอนเน็คเตอร์
- พอร์ตอนุกรมคู่ อาร์เอส232 ด้วย ดีบี-9 คอนเน็คเตอร์
- 4 ช่องสัญญาณเอชดีเอลซี (HDLC : High level Data Link Control)
- สี่ไทม์สล๊อต (TSAs : Time slot Assigner)
- จีซีไอ คอนโทรลเลอร์

- 3 ไทม์เมอร์ที่สามารถโปรแกรมได้
- นาฬิกา เรียลไทม์
- วอชด็อก ไทม์เมอร์ และ รีเซ็ท คอนโทรลเลอร์
- อินเทอร์รัพ คอนโทรลเลอร์
- 25 อินพุต/เอาท์พุต ที่สามารถโปรแกรมได้
- ไอซ่า บัส ขนาด 8 บิต
- เพาเวอร์ซัพพลาย 3.3 โวลต์ และ 5 โวลต์
- เอลอีดี แสดงสถานะของเพาเวอร์ซัพพลาย และ อีเทอร์เน็ต



รูปที่ 5-3 ตัวอย่างของภาพชุดพัฒนาคอม86



รูปที่ 5-4 บล็อกไดอะแกรมแสดงส่วนประกอบต่างๆของคอม86

5.4 ส่วนประกอบต่างของคอม86

5.4.1 พาวเวอร์ซัพพลาย (Power supply)

ระบบจ่ายไฟของบอร์ดคอม86 แบ่งได้ออกเป็น 2 ส่วนหลักด้วยกันคือ ในส่วนแรกจะเป็นระบบจ่ายไฟแรงดัน 5 โวลท์ (Volt) ซึ่งจะมีไอซีแอลดีโอ LDO (U14) ทำหน้าที่จ่ายกระแสไฟฟ้าสำหรับเลี้ยงอุปกรณ์บนบอร์ดที่ต้องการแรงดันไฟฟ้าในการทำงานที่ 5 โวลท์ เช่น ดีแรม, แฟลชไบออส, เรียลไทม์คล็อก เป็นต้น และสำหรับอุปกรณ์ที่ต้องการแรงดันไฟฟ้า 3.3 โวลท์ ในการทำงานนั้นจะมีไอซีแอลดีโอ LDO (U16) ทำหน้าที่จ่ายกระแสไฟฟ้าให้ซึ่งอุปกรณ์เหล่านี้ได้แก่ตัวไมโครคอนโทรลเลอร์, แฟลชดิสก์เป็นต้น ซึ่งอินพุทของภาคพาวเวอร์ซัพพลายของบอร์ดคอม86 นั้นจะรับเป็นไฟฟ้ากระแสตรงที่แรงดันประมาณ 8 โวลท์ และทำการลดระดับแรงดันลงเป็น 5 โวลท์และ 3.3 โวลท์ตามลำดับ โดยภายในชุดพัฒนาจะมีอแดปเตอร์ให้ 1 ตัวสำหรับเป็นส่วนจ่ายกระแสไฟฟ้าให้กับวงจรทั้งหมด

โดยทั่วไปส่วนวงจรจ่ายกระแสไฟฟ้าแรงดัน 5 โวลท์และ 3.3 โวลท์จะสามารถจ่ายกระแสไฟฟ้าให้กับวงจรที่ต่อเพิ่มเติมได้อีก 300 มิลลิแอมป์ (mA) หากมีการต่อวงจรส่วนขยายที่ต้องการใช้งานกระแสไฟฟ้ามากกว่านี้ แนะนำให้ควรมีการต่อวงจรพาวเวอร์ซัพพลายจากภายนอกเพิ่มเติมเพื่อป้องการเสียหายของส่วนวงจรภาคจ่ายไฟของบอร์ดคอม86

5.4.2 แอลอีดีอินดิเคเตอร์ (LED Indicator)

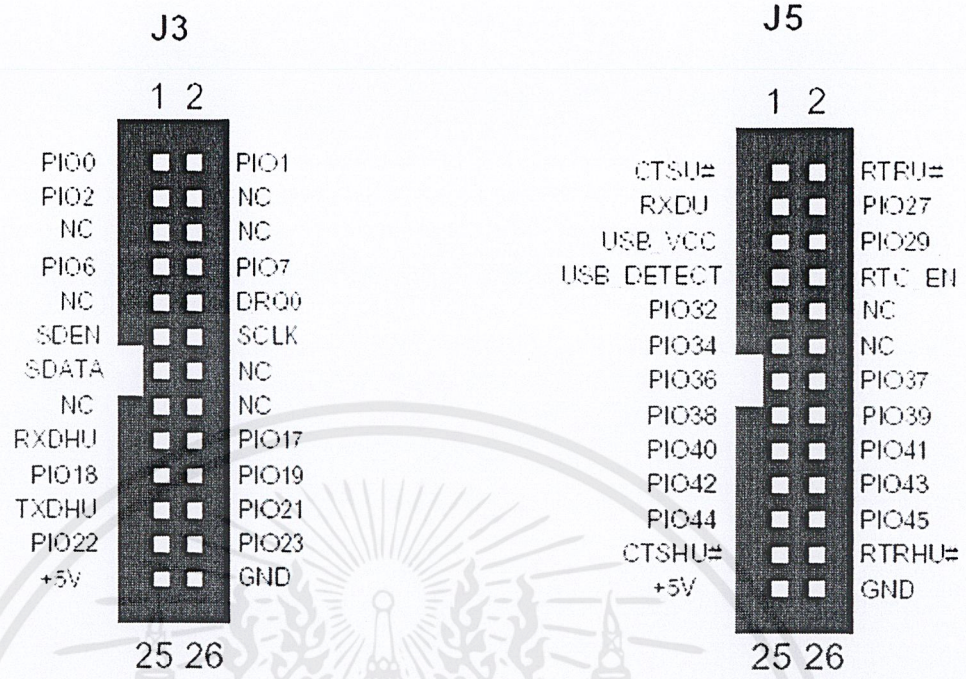
บนบอร์ดคอม86 จะมีแอลอีดี (LED) สำหรับแสดงสถานะต่างๆดังนี้คือ ดวงแรกเป็นแอลอีดีของพาวเวอร์ซัพพลาย (D4) มีสีแดงโดยจะติดสว่าง เมื่อบอร์คอม86 ได้รับการจ่ายกระแสไฟฟ้าจากภายนอกและวงจรพาวเวอร์ซัพพลาย 5 โวลต์ และ 3 โวลต์ สามารถทำงานได้อย่างถูกต้อง ดวงที่ 2 คือ LED สีเขียวของส่วนวงจรอีเทอร์เน็ต(D2) ซึ่งดวงนี้จะติดสว่างขึ้นเพื่อแสดงว่าการมีการเชื่อมต่อสายอีเทอร์เน็ตและพร้อมที่จะรับส่งข้อมูล ดวงที่ 3 คือ Lแอลอีดีดวงสีเขียวของส่วนวงจรอีเทอร์เน็ตที่เหลืออีกดวงหนึ่ง (D1) ซึ่งดวงนี้จะติดกระพริบเมื่อมีการรับส่งข้อมูลผ่านทางอีเทอร์เน็ตพอร์ต

5.4.3 เอ็กแพลนชั่นพอร์ต (Expansion Port)

ในการใช้งานชุดพัฒนาต่างๆสำหรับพัฒนาอุปกรณ์ทางด้านระบบฝังตัวนั้น เป็นธรรมดาที่นักพัฒนาย่อมอาจเกิดความต้องการที่จะออกแบบวงจรส่วนขยายต่ออุปกรณ์ต่างๆ เพิ่มเติมจากชุดพัฒนาพื้นฐานที่มีใช้งานอยู่ ซึ่งสำหรับบอร์ดคอม86 นี้ได้จัดเตรียมออกแบบการเชื่อมต่อใช้งานอุปกรณ์ภายนอกในรูปแบบต่างๆ ดังนี้

5.4.4 พีไอโอ (PIO)

ในส่วนของพีไอโอ นี้จะเป็นลักษณะการต่อใช้งานเหมือนกับที่ใช้งานภายในไมโครคอนโทรลเลอร์โดยทั่วไปซึ่งสามารถโปรแกรมบังคับควบคุมการใช้งานได้เป็นอิสระจากซึ่งกันและกันโดยบนบอร์ดคอม86 นี้ มีพีไอโอให้นักพัฒนาได้ใช้งานทั้งสิ้นจำนวน 25 แชนแนล (Channel) จากทั้งหมด 48 แชนแนลที่มีในไมโครคอนโทรลเลอร์เนื่องจากการที่สัญญาณพีไอโอบางสัญญาณถูกนำไปใช้งานในลักษณะของการมัลติเพล็กกับสัญญาณอื่นของบอร์ดคอม86 ซึ่งพีไอโอที่ใช้งานได้บนบอร์ดคอม86 และการเชื่อมต่อมายังคอนเน็คเตอร์ของพีไอโอต่างๆ เป็นไปดังภาพ

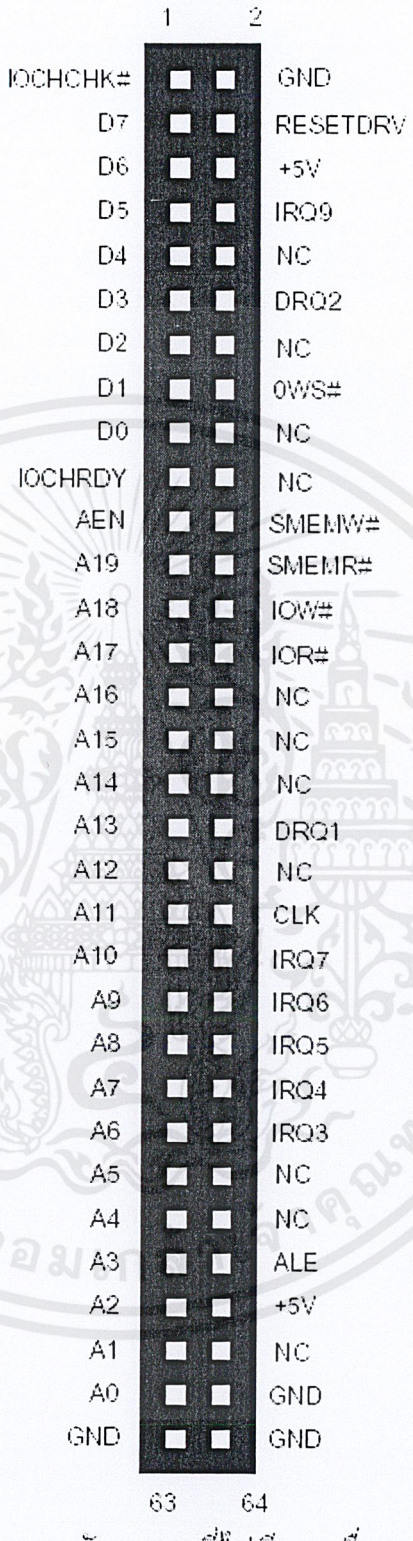


รูปที่ 5-5 แผนภาพการเชื่อมต่อ PIO ของบอร์ดคอม86

จากภาพในส่วนที่เป็นชื่อของสัญญาณอื่นที่มิใช่พีไอโอหมายเลขต่างๆ นั้นคือ พีไอโอ ที่ถูกใช้งานในรูปแบบของสัญญานอื่นที่มัลติเพล็กซ์ (Multiplex) อยู่และสามารถนำมาใช้งานเป็นพีไอโอได้ในกรณีที่ไม่ต้องการใช้งาน เช่นขา CTSHU# ถ้าไม่ต้องการใช้งานส่วนของ โฟลว์คอนโทรล (Flow Control) ของพอร์ต คอม1 ก็สามารถนำมาใช้งานเป็น พีไอโอ46 ได้ เอ็นซี คือขาพีไอโอที่ถูกนำไปใช้ในลักษณะของสัญญาณอื่นที่มัลติเพล็กซ์อยู่และไม่สามารถนำมาใช้งานได้ ซึ่งไม่มีการนำมาเชื่อมต่อให้ไว้ที่คอนเน็คเตอร์ ส่วนขาที่เหลือจะเป็นพีไอโอที่สามารถใช้งานได้

5.4.5 ไอซ่า (ISA)

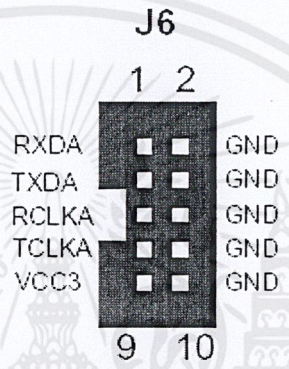
เป็นรูปแบบของการเชื่อมต่อกับอุปกรณ์แบบขนาน โดยได้ออกแบบให้มีลักษณะการเชื่อมต่อใช้งานคล้ายกับการเชื่อมต่อใช้งานของไอซ่าบัส (ISA Bus) ของคอมพิวเตอร์โดยทั่วไปแตกต่างกันเพียงรูปแบบการจัดวางอุปกรณ์ต่างๆ ที่พยามให้จัดวางในรูปลักษณะของมาตรฐาน พีซี/104 ซึ่งการออกแบบวงจรต่างๆ ที่จะนำมาเชื่อมต่อนั้นสามารถออกแบบในลักษณะเดียวกันกับการออกแบบการ์ดไอซ่า ของเครื่องคอมพิวเตอร์โดยทั่วไป จะมีคอนเนคเตอร์ เจ4 สำหรับใช้ในการเชื่อมต่อกับการ์ดขยายภายนอกที่จะนำมาเชื่อมต่อ ซึ่งการเชื่อมต่อของสัญญาณต่างๆ ของคอนเน็คเตอร์ เจ4 จะเป็นไปดังภาพ



รูปที่ 5-6 การเชื่อมต่อสัญญาณต่างๆของไอซ่าคอนเนคเตอร์ (ISA connector)

5.4.6 จีซีไอ (GCI)

จีซีไอ เป็นมาตรฐานการสื่อสารแบบอนุกรมสำหรับใช้ในการเชื่อมต่อวงจรทางด้านการสื่อสารต่างๆ ซึ่งมาตรฐานครอบคลุมถึงการเชื่อมต่อของไลน์การ์ด (Linecard), เอนที1 หรือ เทอร์มินัลอาร์คิเทกเจอร์ (Terminal Architecture) ของ ไอเอสดีเอน (ISDN) แอพพลิเคชั่นซึ่งบอร์ดคอม86 นี้สนับสนุนรองรับการทำงานในส่วนของมาตรฐานการเชื่อมต่อแบบเทอร์มินัลเวอร์ชั่น (Terminal Version) ของ จีซีไอ สแตนดาร์ด (GCI Standard) ซึ่งมีการเชื่อมต่อกับภายนอกผ่านทางพอร์ตจีซีไอ (J6 ของบอร์ดคอม86) โดยมีการเชื่อมต่อของสัญญาณต่างๆ ดังภาพ



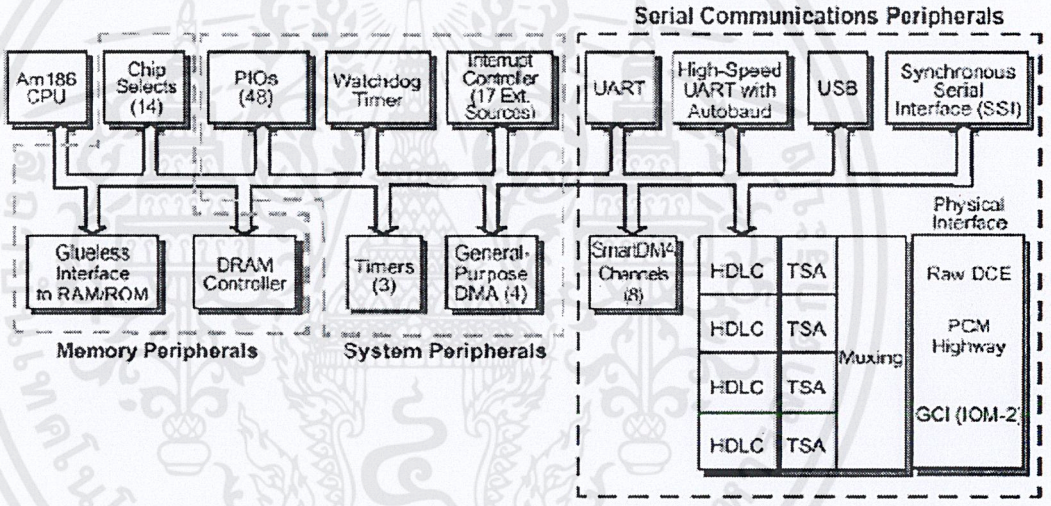
รูปที่ 5-7 แผนภาพการเชื่อมต่อในส่วนของจีซีไอ

5.5 ไมโครคอนโทรลเลอร์ Am186CC

ไมโครคอนโทรลเลอร์ AM186CC ถือว่าเป็นหัวใจหลักที่ควบคุมการทำงานส่วนต่างๆของบอร์ดคอม 86 ซึ่งไมโครคอนโทรลเลอร์ตัวนี้เป็นไมโครคอนโทรลเลอร์ขนาด 16 บิต ในตระกูล อี86 ของบริษัทเอเอ็มดีชื่อว่า Am186CC ซึ่งเป็นคอมมิวนิเคชันไมโครคอนโทรลเลอร์ (Communication Microcontroller) ที่ใช้แกนหลัก (core) ของ 80186 โดยได้เพิ่มความสามารถทางด้านการสื่อสาร และส่วนประกอบอื่นๆ โดยมีส่วนสำคัญที่ได้เพิ่มเข้าไปดังนี้

- ส่วนเอชดีแอลซี (HDLC) หรือ ไฮเลเวลดาต้าลิ้งค์คอนโทรล (High–level Data Link Control) สำหรับเป็นส่วนจัดการเรื่องการติดต่อสื่อสารแบบ เอชดีแอลซี ที่สามารถเชื่อมต่อกับ ดีซีอี, พีซีเอ็ม ไฮเวย์ และ จีซีไอ ผ่านทางพีไอโอโดยมีทีซ่า (TSAs) หรือ ไทม์สลอตแอสซายเนอร์ (Time Slot Assigners) สำหรับการทำงานในแบบมัลติเพลกซ์ (Multiplex)
- ยูเอสบีเพอร์ริพารอลคอนโทรล (USB Peripheral Controller) สำหรับการทำเป็นอุปกรณ์ยูเอสบี
- เอชยูอาร์ต (HUART) หรือ ไฮสปีดยูอาร์ต (High-speed UART) สำหรับใช้งานในการเชื่อมต่อแบบอนุกรมความเร็วสูง โดยสามารถสื่อสารได้ในอัตราเร็วสูงสุด 460 กิโลบิตต่อวินาที (Kbit /s) สามารถตรวจสอบอัตราเร็วในการเชื่อมต่อได้อัตโนมัติ (Automatic Baud Rate Detection) และมีฮาร์ดแวร์แฮนชาร์คกิ้ง(Handshaking)

- ยูอาร์ต (UART) หรือ ยูนิเวอร์ซัลดซิงโคนัสรีซีพเวอร์/ทรานส์มิทเทอร์ (Universal Asynchronous Receiver/Transmitter) สำหรับการเชื่อมต่อแบบอนุกรมโดยสามารถสื่อสารได้ในอัตราเร็วสูงสุด 115.2 กิโลบิตต่อวินาที และมีฮาร์ดแวร์แฮนชาร์คกิ้ง
- เอสเอสไอ (SSI) หรือ ซิงโคนัสซีเรียลอินเทอร์เฟส (Synchronous Serial Interface) สำหรับการเชื่อมต่อ สื่อสารแบบอนุกรมความเร็วสูง 25 เมกกะบิตต่อวินาที (Mbit/s) ซึ่งสามารถสื่อสารแบบเอสพีไอ (SPI) ได้
- โปรแกรมเมเบิลไทม์เมอร์ (Programmable Timer) ขนาด 16 บิต 3 ตัว
- ฮาร์ดแวร์วอชด็อกไทม์ (Hardware Watchdog Time)
- ดีแรมคอนโทรลเลอร์ (DRAM Controller)
- อินเทอร์รัพคอนโทรลเลอร์ (Interrupt Controller) 36 มาสก์เอเบิลอินเทอร์รัพ
- โปรแกรมเมเบิลอินพุทเอาท์พุทซิกแนล (Programmable I/O Signal) ซึ่งมีจำนวน 48 แชนแนล



รูปที่ 5-8 บล็อกไดอะแกรมแสดงองค์ประกอบของไมโครคอนโทรลเลอร์ AM186CC

ไมโครคอนโทรลเลอร์ Am186CC นี้ เป็นไมโครคอนโทรลเลอร์ที่ที่มีสถาปัตยกรรมแบบ x86 มีชุดคำสั่งที่เข้ากันได้กับชุดคำสั่งของ 80286 (Real Mode) สนับสนุนการอ้างอิงหน่วยความจำหลักได้ถึง1 เมกกะไบต์

5.6 หน่วยความจำ (Memory)

ภายในบอร์ดคอม86 ประกอบด้วยหน่วยความจำหลายส่วนประกอบเข้าด้วยกันซึ่งมีรายละเอียดต่างๆ ดังนี้

5.6.1 แรม (RAM)

หน่วยความจำส่วนนี้เป็นส่วนที่ใช้ในการทำงานของโปรแกรมต่างๆ ของบอร์ดคอม86 ซึ่งมีขนาด 1 เมกกะไบต์ อันเนื่องมาจากจากสถาปัตยกรรมของไมโครคอนโทรลเลอร์หน่วยความจำในส่วนนี้จะใช้เป็นพื้นที่ในการใช้งานสำหรับโปแกรมต่างๆทั้งระบบปฏิบัติการและโปรแกรมของนักพัฒนา โดยหน่วยความจำนี้เป็นหน่วยความจำชนิดอีดีโอ (EDO) ขนาด 1 เมกกะไบต์โดยมีการควบคุมการทำงานจากส่วนของดีแรมคอนโทรลเลอร์ภายในไมโครคอนโทรลเลอร์

5.6.2 แฟลชไบออส

หน่วยความจำส่วนนี้เป็นหน่วยความจำแบบแฟลชขนาด 64 กิโลไบต์สำหรับใช้ในการเก็บโปรแกรมไบออสสำหรับควบคุมการทำงานพื้นฐานของระบบ หน่วยความจำในส่วนนี้จะถูกเรียกใช้งานเป็นอันดับแรกเมื่อระบบเริ่มทำงาน

5.6.3 ซีเรียลดาต้าแฟลช (Serial DataFlash)

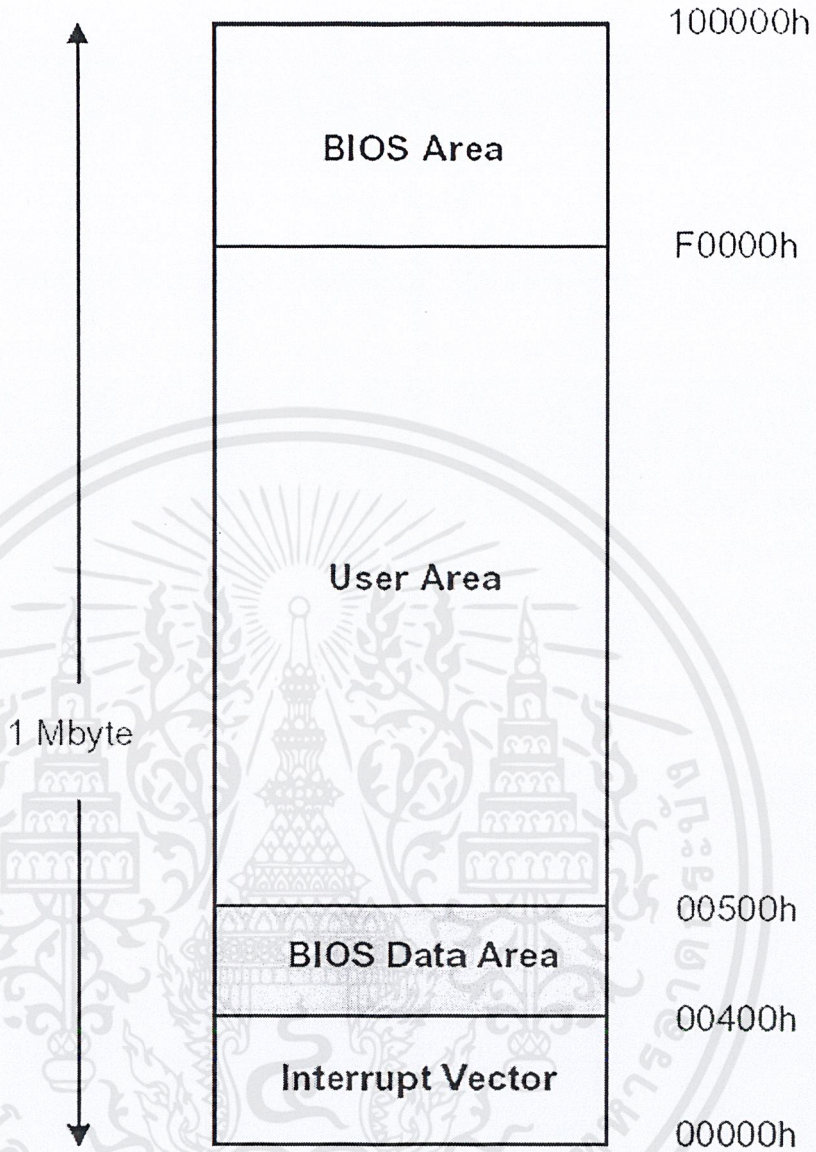
หน่วยความจำแบบแฟลชตัวนี้จะทำงานทำหน้าที่เป็นสตอเรท (Storage) สำหรับใช้ในการเก็บข้อมูลต่างๆ ของระบบที่ไม่ต้องการใช้เกิดการสูญหายในขณะที่บอร์ดคอม86 ขาดกระแสไฟฟ้าหล่อเลี้ยงเช่น ข้อมูลจากการตรวจวัดต่างๆ, โปรแกรมแอพพลิเคชั่นต่างๆ รวมไปถึงระบบปฏิบัติการด้วย ซึ่งบนบอร์ดคอม86 นี้ได้ติดตั้ง ดาต้าแฟลช (IC U12) เบอร์ เอที45ดีบี041ขนาด 512 กิโลไบต์สำหรับใช้งานในส่วนนี้ และได้ออกแบบให้สามารถขยายขนาดได้ตามความต้องการ ซึ่งในการใช้งานนักพัฒนาจะมองหน่วยความจำในส่วนนี้เปรียบเสมือนดิสก์ไดรฟ์อันหนึ่ง (ไดรฟ์ A) ของเครื่องคอมพิวเตอร์

5.6.4 แผนผังหน่วยความจำ

จากหน่วยจำทั้งหมดที่กล่าวมาในข้างต้นสามารถแสดงเป็นภาพการจัดวางของหน่วยความจำทั้งหมดได้ ดังนี้คือ

- **หน่วยความจำหลัก**

หน่วยความจำในส่วนนี้มีขนาด 1 เมกกะไบต์ สำหรับเป็นพื้นที่ใช้งานของโปรแกรมต่างๆซึ่งมีการจัดแบ่งดังนี้

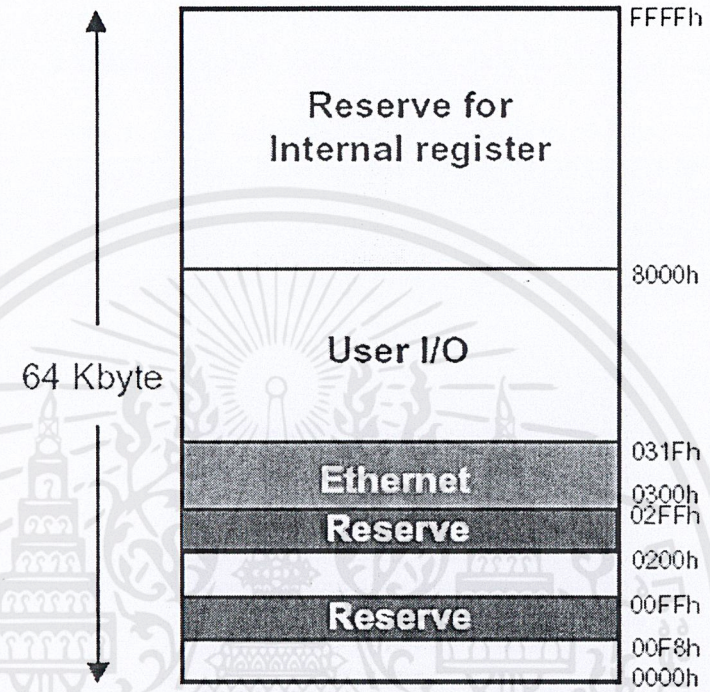


รูปที่ 5-9 แผนภาพการแบ่งการใช้งานของหน่วยความจำหลัก

จากภาพหน่วยความจำช่วงแรกคือจากตำแหน่ง 00000h จนถึงตำแหน่ง 003FFh เป็นส่วนของอินเทอร์รัพเวกเตอร์เทเบิล (Interrupt Vector Table) สำหรับรองรับการทำงานของการอินเทอร์รัพ (Interrupt) ที่เกิดขึ้นส่วนต่อมาคือช่วงตำแหน่ง 00400h ถึง 004FFh หน่วยความจำ ส่วนนี้ใช้สำหรับเก็บข้อมูลของไบออส (BIOS) ส่วนถัดมาคือช่วงตำแหน่ง 00500h ถึง F0000h ส่วนนี้จะเป็นพื้นที่ สำหรับใช้งาน โดยโปรแกรมทั่วไปทั้งระบบปฏิบัติการและโปรแกรมของนักพัฒนาที่สร้างขึ้นมา ส่วนที่เหลือในช่วงตำแหน่ง F0000h ถึง FFFFFh ส่วนนี้จะเป็นพื้นที่ที่ถูกสงวนไว้สำหรับไบออสของบอร์ดคอม86

- **หน่วยความจำที่เป็นส่วนอินพุท/เอาท์พุท (I/O)**

ส่วนนี้คือพื้นที่สำหรับใช้ในการติดต่อกับอุปกรณ์อินพุท/เอาท์พุทต่างๆ และใช้ในการติดต่อกับรีจิสเตอร์ภายในของไมโครคอนโทรลเลอร์ ซึ่งมีการจัดวางดังภาพ



รูปที่ 5-10 แผนภาพการจัดวางหน่วยความจำในส่วนของอินพุท/เอาท์พุท

จากภาพจะแบ่งตำแหน่งของหน่วยความจำได้ออกเป็น 2 ส่วนหลักคือส่วนบนตั้งแต่ตำแหน่งที่ 8000h ถึงตำแหน่ง FFFFh ส่วนนี้จะเป็นพื้นที่สำหรับการใช้งานโดยตัวบอร์ดคอม86 และเป็นส่วนที่ใช้ในการติดต่อรีจิสเตอร์ต่างๆของไมโครคอนโทรลเลอร์ซึ่งบริเวณนี้จะถูกสงวนไว้ไม่สามารถนำมาใช้งานได้ ส่วนที่สองคือส่วนของหน่วยความจำส่วนล่างคือ ตั้งแต่ตำแหน่ง 0000h ถึงตำแหน่ง7FFFh บริเวณนี้จะเป็นพื้นที่สำหรับให้นักพัฒนานำส่วนที่ว่างไปใช้งานโดยเว้นส่วนที่สงวนไว้ใช้งานสำหรับตัวไมโครคอนโทรลเลอร์ คือ ตำแหน่ง 00F8h ถึงตำแหน่ง 00FFh และส่วนที่สงวนไว้สำหรับการใช้งานของวงจรอีเทอร์เน็ตคือในตำแหน่ง 0300h จนถึง 031Fh บริเวณที่เหลือนอกจากที่กล่าวมานักพัฒนาสามารถนำไปใช้งานได้ตามต้องการ

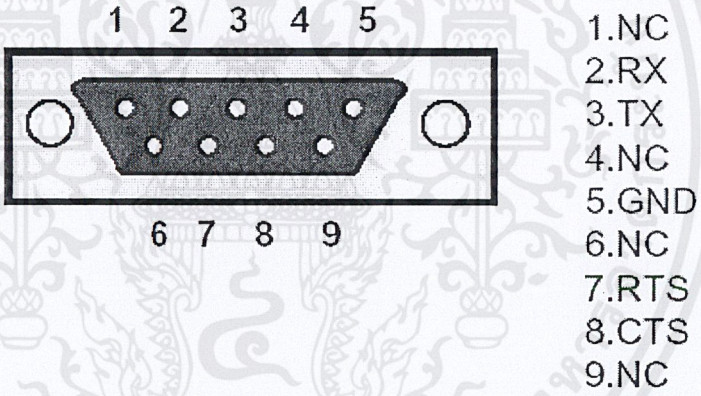
5.7 ยูเอสบี (USB)

บอร์ดคอม86 นั้น มีการสื่อสารกับภายนอกได้อีกรูปแบบหนึ่งคือในรูปแบบของยูเอสบี พอร์ตโดยภายในไมโครคอนโทรลเลอร์ AM186CC ของบอร์ดคอม86นั้นมีส่วนของยูเอสบีคอนโทรลเลอร์ (USB Controller) ที่เป็นยูเอสบีดีไวซ์คอนโทรลเลอร์ (USB device controller) ภายในไมโครคอนโทรลเลอร์และยูเอสบีทรานส์ซิพเวอร์ (USB Transceiver) อยู่ภายในบอร์ดซึ่งนักพัฒนาสามารถเขียนโปรแกรมจำลองตัว

บอร์ดคอม86 ให้สามารถทำตัวเองเป็นอุปกรณ์ยูเอสบี ชนิดหนึ่งได้ทันที โดยบนบอร์ดคอม86 จะมีคอนเนคเตอร์ยูเอสบี ชนิดไทพ์บี (Type B (J7)) ติดตั้งอยู่บนบอร์ดอยู่แล้วซึ่งนักพัฒนาเพียงแต่เขียนโปรแกรมเพิ่มเติมก็จะสามารถนำบอร์ดไปใช้งานเป็นอุปกรณ์ยูเอสบีได้ในทันทีซึ่งบนยูเอสบีคอนโทรลเลอร์ (USB Controller) บนบอร์ดคอม86 นี้สนับสนุนการทำงานในฟูลโหมดสปีด (Mode Full Speed) คือที่อัตราเร็ว 12 เมกกะบิตต่อวินาที (Mbit/s) ในการรับส่งข้อมูล

5.8 พอร์ตอนุกรม

บนบอร์ดคอม86 มีพอร์ตการสื่อสารแบบอนุกรมตามมาตรฐาน อาร์เอส-232 สำหรับการนำไปใช้งานจำนวน 2 พอร์ตด้วยกัน ซึ่งอยู่ในรูปแบบของคอนเนคเตอร์แบบ ดีบี9 บนบอร์ด (J8, J9) ซึ่งพอร์ตอนุกรมทั้ง 2 นี้เชื่อมต่อมาจากวงจรยูอาร์ต ที่อยู่ภายในไมโครคอนโทรลเลอร์ AM186CC และทำการปรับแรงดันจาก 5 โวลท์ให้เข้ากับมาตรฐาน อาร์เอส232 โดยชิพ อาร์เอส232 ไดร์เวอร์ (U9, U10) ซึ่งพอร์ตอนุกรมทั้งสองพอร์ตนี้ พอร์ต คอม1 จะถูกนำไปใช้งานสำหรับเป็นซีเรียลคอนโซล (Serial Console) สำหรับใช้ในการแสดงผลควบคุมการทำงานต่างๆ ของบอร์ดในขณะพัฒนาโปรแกรม ส่วนพอร์ต คอม2 นั้นสามารถนำไปใช้งานได้โดยทั่วไป ซึ่งการเชื่อมต่อสัญญาณต่างๆ ของพอร์ตอนุกรมทั้งสองนี้เป็นไปดังภาพ



รูปที่ 5-11 การจัดวางสัญญาณต่างๆของคอนเนคเตอร์ของพอร์ตอนุกรม

สำหรับในการพัฒนานั้น ภายในชุดพัฒนาคอม86 จะมีสายเชื่อมต่อระหว่างพอร์ต คอม1 ที่ทำหน้าที่เป็น คอนโซล (Console) กับคอมพิวเตอร์ซึ่งจะเป็นสายชนิดนูโมเด็ม (null modem) สำหรับใช้ในการแสดงผลและรับส่งข้อมูลต่างๆ ในระหว่างการพัฒนาโปรแกรม

5.9 ระบบรีเซ็ต

ภายในบอร์ดคอม86 จะมีวงจรสำหรับสร้างสัญญาณรีเซ็ตให้กับอุปกรณ์ต่างๆ บนบอร์ดโดยใช้ชิพ

เอลเอ็ม3722 (U15) เป็นตัวควบคุมการทำงานโดยวงจรจะสร้างสัญญาณรีเซ็ตเมื่อมีการกดสวิทช์รีเซ็ต (SW1) บนบอร์ดหรือในกรณีที่แรงดันไฟฟ้าบนบอร์ดในส่วนของแหล่งจ่ายแรงดัน 3.3 โวลต์มีแรงดันต่ำกว่า 3.08 โวลต์

5.10 อีเทอร์เน็ต

การเชื่อมต่อแบบอีเทอร์เนตกับภายนอกนั้นเป็นรูปแบบหนึ่งที่ชุดพัฒนาคอม86 สนับสนุนการทำงาน ซึ่งภายในบอร์ดคอม86 ได้มีการออกแบบติดตั้งทั้งตัวชิพอีเทอร์เน็ตคอนโทรลเลอร์ (U6) ซึ่งเป็นชิพอีเทอร์เน็ตที่สนับสนุนการเชื่อมต่อที่ความเร็วขนาด 10 เมกกะบิตต่อวินาที (Mbps) พร้อมทั้งติดตั้ง อาร์เจ45 คอนเน็คเตอร์ (J2) บนบอร์ดคอม86 ซึ่งมีการเชื่อมต่อสัญญาณตามมาตรฐานทั่วไปโดยนักพัฒนาสามารถใช้สายแลนชนิดแบบครอส (twisted pair cross cable) สำหรับเชื่อมต่อระหว่าง PC กับบอร์ดคอม86 โดยตรงหรือใช้สายแลนชนิดแบบสายตรง (straight-through twisted pair cable) ในกรณีที่ต่อผ่านฮับ (Hub) หรือเน็ตเวิร์คสวิทช์ (Network Switch) สำหรับให้นักพัฒนาเชื่อมต่อใช้งานได้ทันที ซึ่งตามมาตรฐาน IEEE 802.3 นั้นการเชื่อมระหว่างฮับกับบอร์ดคอม86 สายที่ใช้ควรมีระยะทางไม่เกิน 100 เมตร

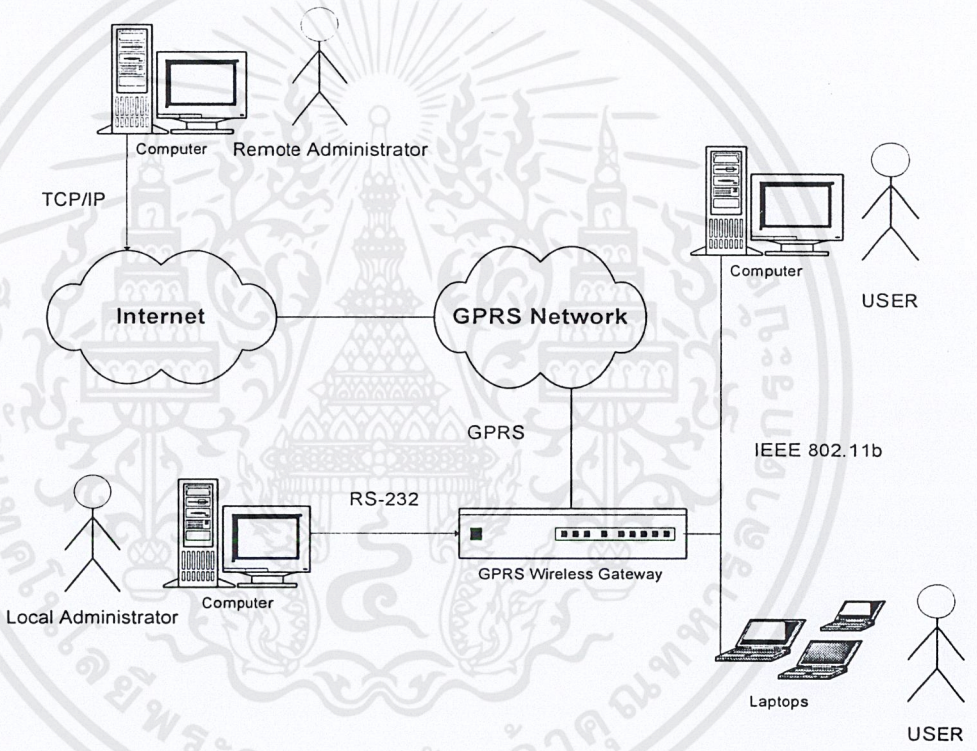
5.11 ยูอาร์ต UART (Universal Asynchronous Receiver - Transmitter)

เป็น ส่วนประกอบที่ใช้สำหรับการสื่อสารระหว่าง อุปกรณ์อินพุตและเอาท์พุต ที่มีการ ส่งข้อมูลเป็นแบบ อนุกรมซึ่ง ยูอาร์ต จะควบคุมอัตราการส่งข้อมูลที่ทั้งสองฝั่งของการรับส่ง เพื่อไม่ให้เกิดปัญหาเมื่ออัตราเร็วในการรับส่งข้อมูลทั้ง 2 ด้านไม่เท่ากัน

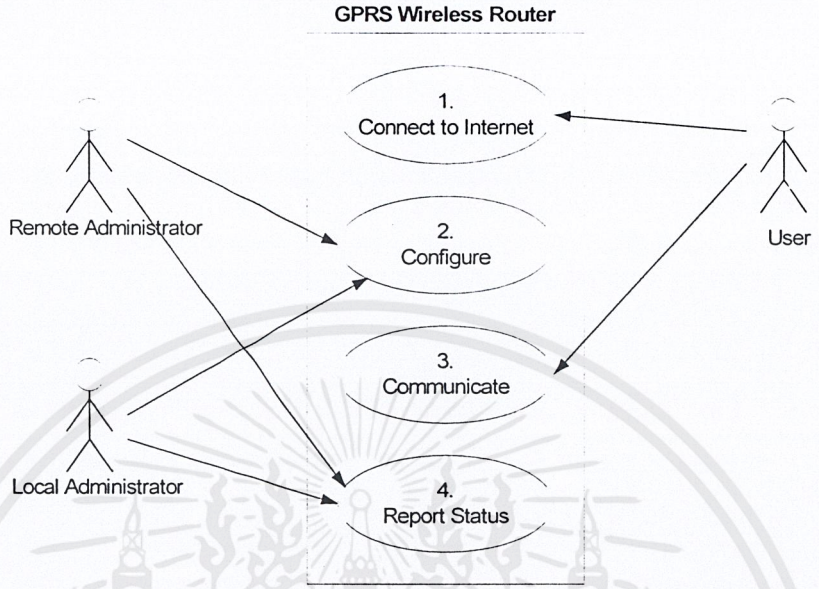
บทที่ 6
การสร้างและการออกแบบ

6.1 การออกแบบระดับสถาปัตยกรรมของอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

โครงสร้างและองค์ประกอบโดยภาพรวมของระบบเครือข่ายที่ใช้อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอสแสดงดังรูปที่ 6-1 และลักษณะการใช้งานอุปกรณ์แสดงดังรูปที่ 6-2



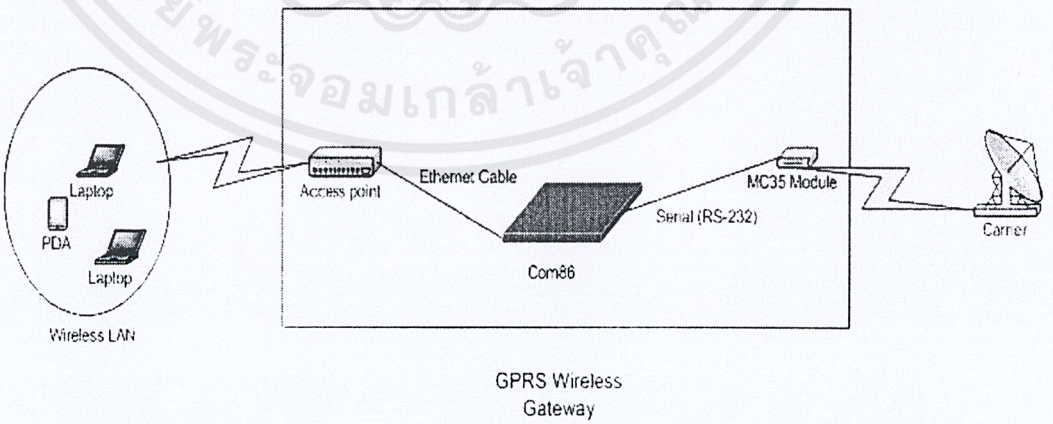
รูปที่ 6-1 แสดงลักษณะของการเชื่อมต่อเครือข่ายที่ใช้อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส



รูปที่ 6-2 แสดงการออกแบบลักษณะการใช้งานอุปกรณ์

ลักษณะของเครือข่ายที่จะใช้อุปกรณ์นี้จะเป็นเครือข่ายที่มีการเชื่อมต่อระหว่างระบบเครือข่ายท้องถิ่นและเครือข่ายอินเตอร์เน็ต โดยทางด้านของเครือข่ายท้องถิ่นจะเชื่อมต่อกับอุปกรณ์แอกเซสพอยน์ผ่านทางสายเคเบิลอาร์เจ45 (RJ-45 cable) และทางด้านของเครือข่ายอินเตอร์เน็ตจะเชื่อมต่อกับอุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอสผ่านทางสายซีเรียลอาร์เอส-232 โดยผู้ดูแลระบบสามารถเข้ามากำหนดค่าคุณสมบัติของอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอสนี้ ได้ทางเครือข่ายอินเตอร์เน็ตซึ่งจะผ่านทางเวบเพจหรือทางโปรแกรมไฮเปอร์เทอร์มินอลก็ได้

สำหรับองค์ประกอบต่างๆของอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส แสดงดังรูปที่ 6-3

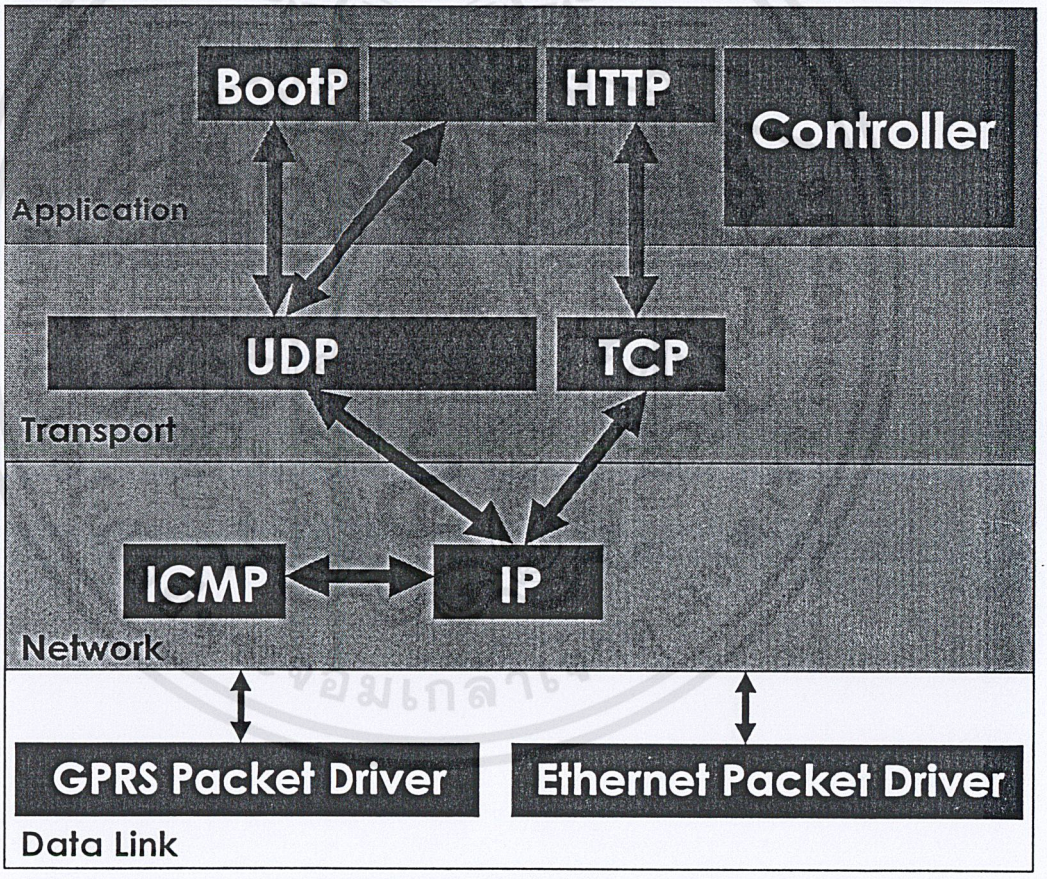


รูปที่ 6-3 แสดงอุปกรณ์ที่เป็นองค์ประกอบของอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

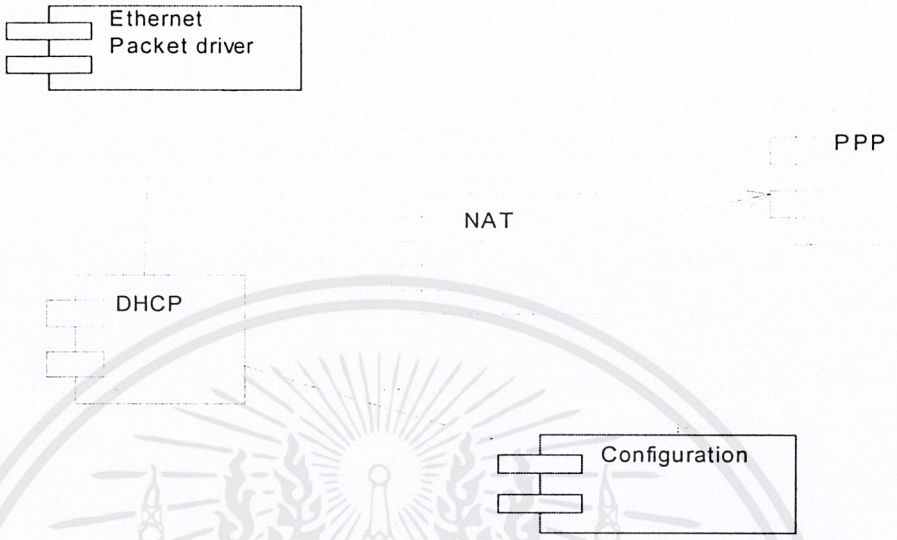
อุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอสจะประกอบด้วยอุปกรณ์หลักๆ 3 อย่างคือ

- อุปกรณ์แอกเซสพอยน์ที่จะเป็นอุปกรณ์ที่จะใช้เชื่อมต่อการสื่อสารที่เป็นเครือข่ายท้องถิ่นแบบไร้สาย
- บอร์ดชุดพัฒนาคอม86ที่จะเป็นส่วนควบคุมงานทุกอย่างเช่น การรับส่งแพ็กเกตข้อมูล การเชื่อมต่อการสื่อสาร การกำหนดค่าคุณสมบัติของเกตเวย์เป็นต้น
- อุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอส ที่จะเป็นอุปกรณ์ที่ทำให้สามารถเข้าสู่เครือข่ายจีพีอาร์เอสเพื่อใช้บริการอินเตอร์เน็ต

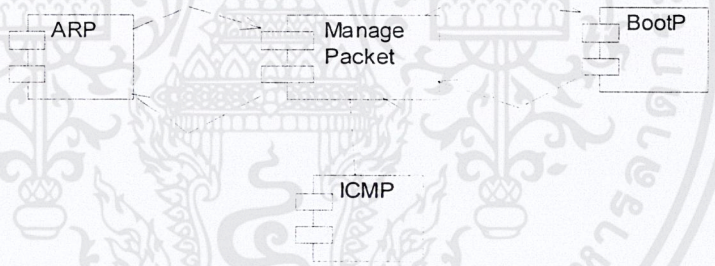
สำหรับการออกแบบซอฟแวร์ที่ใช้กับชุดพัฒนาคอม86นั้นจะต้องมีการออกแบบโปรโตคอลแสตคใหม่เนื่องจากไม่มี ทีซีพี/ไอพี โปรโตคอลแสตค สนับสนุนดังรูปที่ 6-4 จากนั้นจึงมีการออกแบบโครงสร้างแบ่งเป็นโมดูล ต่างๆ ดังรูปที่ 6-5 และ รูปที่6-6 โดยมีคลาสไดอะแกรม ดังรูปที่ 6-7



รูปที่ 6-4 แสดงแผนภาพทีซีพี/ไอพี โปรโตคอลแสตคที่พัฒนาขึ้น



รูปที่ 6-5 แสดงแผนภาพคอมโพเนนท์ของซอฟแวร์ที่สร้างขึ้นทั้งหมด



รูปที่ 6-6 แสดงแผนภาพคอมโพเนนท์ย่อยของโมดูล NAT

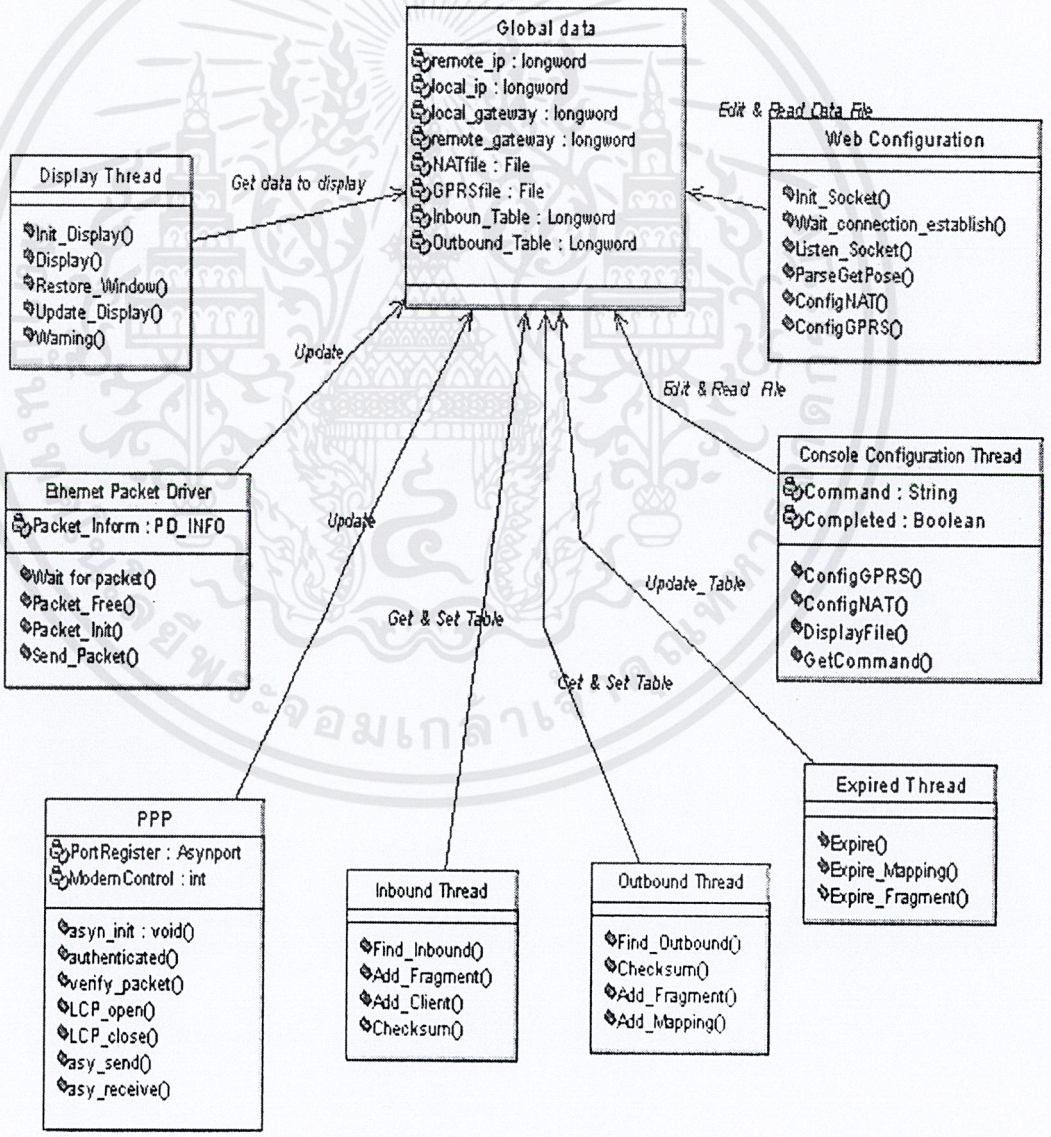
ซอฟแวร์ทุกโมดูลจะถูกโปรแกรมลงบนชุดพัฒนาคอม86 โดยการทำงานของแต่ละโมดูลนั้นจะสลับกันขึ้นมาทำงานเป็นมัลติทาสก์ ความสัมพันธ์และหน้าที่ของแต่ละโมดูลอธิบายได้ดังนี้

- โมดูลคอนฟิกูเรชันเป็นส่วนที่ทำการปรับแก้คุณสมบัติของโมดูลอื่นๆ โดยจะไปแก้ไฟล์คอนฟิกูเรชันของโมดูลเหล่านั้น เช่น โมดูลของการแปลและแปลงไอพีแอดเดรสของเครือข่าย (Network address translation) ส่วนของโมดูลคอนฟิกูเรชันจะไปทำการแก้ไอพีแอดเดรสของเกตเวย์เป็นต้น
- โมดูลของการแปลและแปลงไอพีแอดเดรสของเครือข่ายจะดึงแพ็กเกตที่อีเทอร์เน็ตแพ็กเกตไดร์เวอร์และพีพีพีแพ็กเกตไดร์เวอร์ดึงมาจากตัวอุปกรณ์มาทำการประมวลผล
- โมดูลดีเอชซีพีจะส่งข้อมูลที่ทำการกำหนดไอพีแอดเดรสของเครื่องไคล์เอนท์ให้กับโมดูลอีเทอร์เน็ตแพ็กเกตไดร์เวอร์เพื่อส่งต่อไปยังอุปกรณ์แอกเซสพอยน์และรอรับข้อมูลจากแอกเซสพอยน์ต่อไป

ในส่วนของโมดูลของการแปลและแปลงไอพีแอดเดรสของเครือข่ายยังมีโมดูลย่อยลงไปอีกโมดูลย่อย

ต่างๆจะทำงานดังต่อไปนี้

- โมดูลเออาร์พี (ARP) เป็นส่วนที่จัดการในการแปลงอีเทอร์เน็ตแอดเดรสเป็นไอพีแอดเดรส และ แปลงจากไอพีแอดเดรสกลับเป็นอีเทอร์เน็ตแอดเดรส
- โมดูลบูทพี เป็นส่วนที่ทำงานของบูทสแตรปโปรโตคอล คือรับไอพีแอดเดรสที่ได้รับมาจากผู้ให้บริการเครือข่ายจีพีอาร์เอส
- โมดูลไอซีเอ็มพี เป็นส่วนที่จัดการแพ็กเกตประเภทที่มาจากแอพพลิเคชัน PING ซึ่งแอพพลิเคชันประเภทนี้จะมีการซ่อนไอพีแอดเดรสเอาไว้ต้องใช้โมดูลพิเศษในการอ่านและตอบกลับแพ็กเกต
- โมดูลส่วนกลางที่ทำการแปลงไอพีแอดเดรส เช็คความถูกต้องของข้อมูล และจัดการกับแพ็คเกต



รูปที่ 6-7 แสดงแผนภาพยูทิลิตี้คลาส

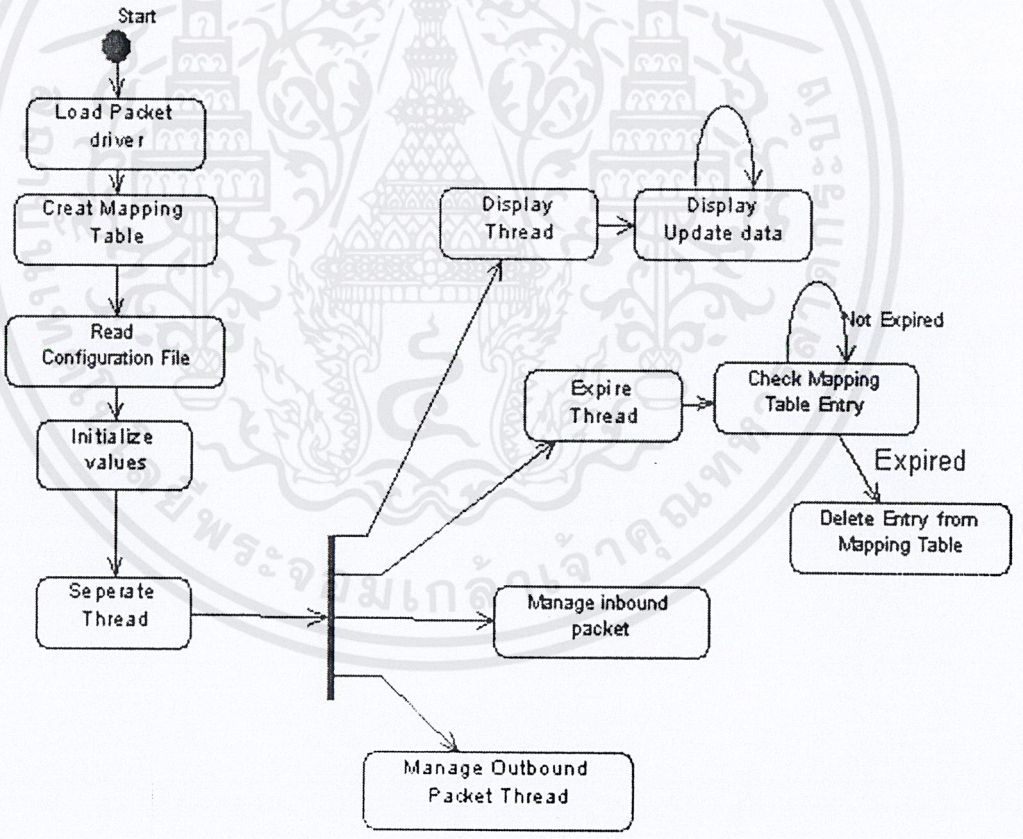
6.2 การออกแบบในระดับโครงสร้างการทำงานของแต่ละโมดูล

ซอฟแวร์ส่วนที่เป็นองค์ประกอบของอุปกรณ์นี้สามารถแบ่งออกเป็นกลุ่มได้แก่

- กลุ่มของการจัดการกับแพ็กเกตข้อมูลที่ผ่านเข้าออก อุปกรณ์ ซึ่งได้แก่โมดูลดีเอชซีพีและโมดูลเอ็นเอที
- กลุ่มของโปรแกรมที่ทำการติดต่อกับฮาร์ดแวร์ซึ่งได้แก่โมดูลของอีเทอร์เน็ตแพ็กเกตไดร์เวอร์และส่วนของพีพีพีแพ็กเกตไดร์เวอร์
- กลุ่มของโปรแกรมที่ทำการปรับแต่งคุณสมบัติต่างๆของอุปกรณ์ซึ่งได้แก่ โมดูลคอนฟิกูเรชัน

ในหัวข้อนี้จะอธิบายถึงหลักการการออกแบบและรายละเอียดการออกแบบของซอฟแวร์ในแต่ละกลุ่มข้างต้นส่วนวิธีการสร้างซอฟแวร์เหล่านี้จะอธิบายในหัวข้อต่อไป

การออกแบบในระดับโครงสร้างการทำงานของส่วนของการจัดการกับแพ็กเกตข้อมูลที่ผ่านเข้าออกอุปกรณ์เกตเวย์แสดงด้วยแผนภาพสถานะดังรูปที่ 6-8และรูปที่ 6-9



รูปที่ 6-8 แสดงแผนภาพสถานะการทำงานของโมดูลเอ็นเอที

สถานะการทำงานของส่วนที่ทำการแปลงไอพีแอดเดรสจะเริ่มจากการโหลดส่วนของแพ็กเกตไดร์เวอร์ทั้งทางด้านเชื่อมต่อกับระบบเครือข่ายท้องถิ่นและทางด้านเชื่อมต่อกับเครือข่ายอินเตอร์เน็ตเพื่อให้

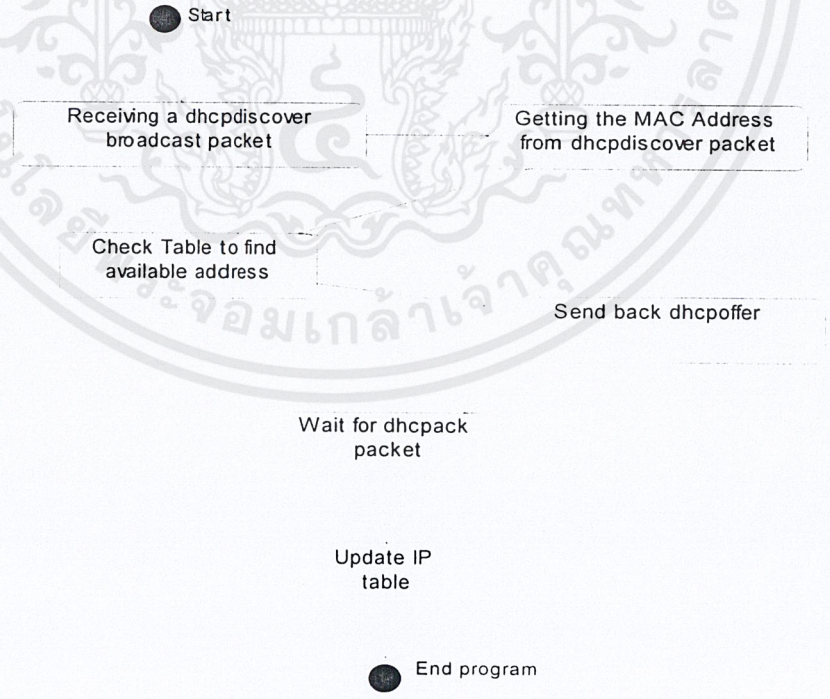
สามารถรับส่งแพ็กเกตข้อมูลได้ จากนั้นอ่านไฟล์ที่กำหนดคุณสมบัติของอุปกรณ์มาแล้วทำการกำหนดค่าเริ่มต้นต่างๆให้กับโปรแกรม โมดูลนี้จะทำมีการแยกการทำงานออกเป็นเทรดเพื่อให้สามารถจัดการงานหลายๆอย่างได้พร้อมๆกันโดยจะแยกงานออกเป็น 4 ส่วนคือ

(1) ส่วนที่จัดการกับแพ็กเกตที่เข้ามาจากภายนอกหรือจากเครือข่ายจีพีอาร์เอส ซึ่งจะทำการแปลงไอพีแอดเดรสของผู้ส่งเป็นของเครื่องไคล์เอ็นท์ภายในเครือข่ายแล้วตรวจสอบความถูกต้องของข้อมูลจากนั้นส่งเข้าไปยังเครือข่ายท้องถิ่น

(2) ส่วนที่จัดการกับแพ็กเกตที่เข้ามาจากทางเครือข่ายท้องถิ่น และต้องการส่งออกไปภายนอกเครือข่าย จะทำการแปลงไอพีแอดเดรสของผู้ส่งเป็นของเครื่องเกตเวย์ แล้วตรวจสอบความถูกต้องของข้อมูลจากนั้นส่งออกไปยังเครือข่ายจีพีอาร์เอส

(3) ส่วนที่ทำการตรวจสอบอายุของข้อมูลที่อยู่ในตารางที่ทำการเปรียบเทียบหมายเลขไอพี เพื่อป้องกันไม่ให้มีการจองหน่วยความจำมากเกินไป ถ้าข้อมูลใดที่มีอายุมากกว่าที่กำหนดไว้ก็จะทำการลบออกจากตาราง โดยอายุของข้อมูลจะถูกตั้งค่าใหม่ทุกครั้งที่มีการใช้งาน ทำให้ข้อมูลที่ถูกลบออกคือข้อมูลที่ไม่มีการใช้งานเป็นเวลานานที่สุด

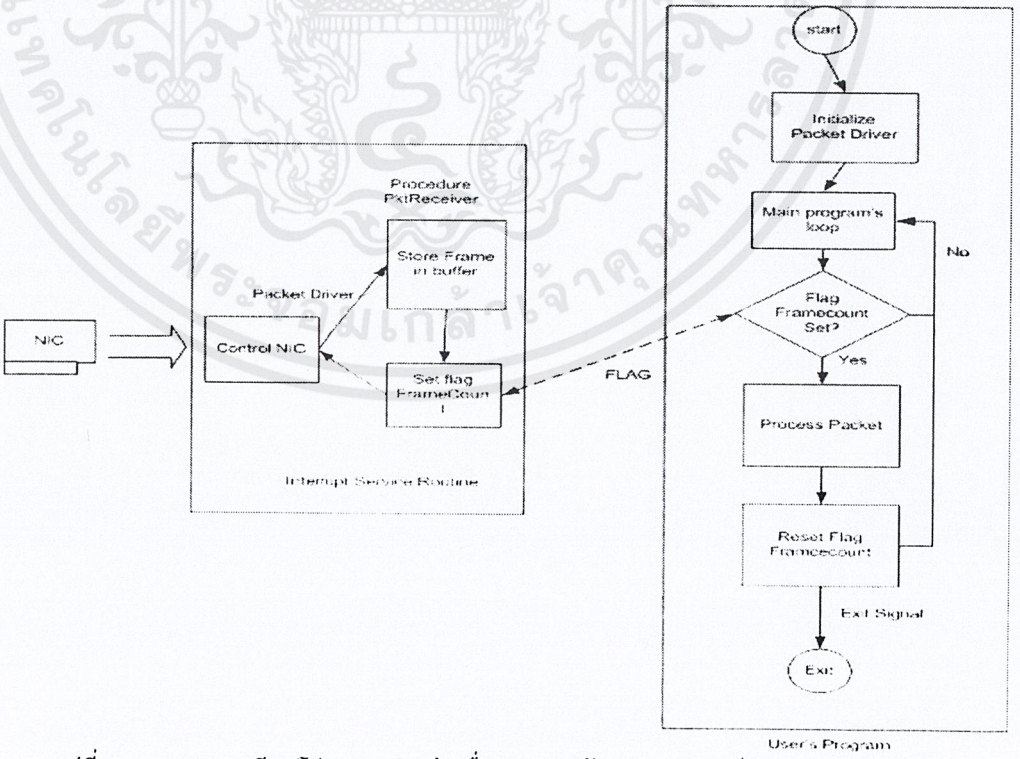
(4) ส่วนที่ทำการแสดงผลข้อมูลของแพ็กเกตที่ผ่านเข้าออกอุปกรณ์ โดยจะทำการแปลงค่าของไอพีแอดเดรสให้เป็นตัวอักษรที่สามารถแสดงผลได้ และคอยตรวจสอบการเปลี่ยนแปลงต่างๆเพื่อแสดงผล



รูปที่ 6-9 แสดงแผนภาพสถานะการทำงานของโมดูลดีเอชซีพี

การทำงานของโมดูลดีเอชซีพีจะเป็นขั้นตอนที่อุปกรณ์เกตเวย์ต้องแจกไอพีแอดเดรสให้กับเครื่องไคล์เอ็นท์ภายในระบบเครือข่ายท้องถิ่นไร้สาย เมื่อมีเครื่องไคล์เอ็นท์ใหม่เข้ามาเครื่องนั้นจะต้องส่งแพ็กเกตที่ขอหมายเลขอีเทอร์เน็ตของตัวเองเรียกว่าแพ็กเกตค้นหาดีเอชซีพี (dhcpdiscover) กระจายออกไปทั่วทั้งเครือข่าย ซึ่งเมื่อเครื่องเกตเวย์ได้รับแพ็กเกตนั้นจะทำตัวเองเป็นดีเอชซีพีเซิร์ฟเวอร์ ไปค้นหาไอพีแอดเดรสที่สามารถใช้งานได้นั่นคือยังไม่ได้ถูกใช้โดยเครื่องอื่นในเครือข่าย จากนั้นส่งแพ็กเกตที่เรียกว่าแพ็กเกตนำเสนอดีเอชซีพี (dhcpoffer) ซึ่งจะประกอบด้วยไอพีแอดเดรสที่จะให้กับเครื่องไคล์เอ็นท์และช่วงเวลาที่อนุญาตให้ใช้ไอพีแอดเดรสนั้น เมื่อเครื่องไคล์เอ็นท์ได้รับและยอมรับหมายเลขไอพีแอดเดรสนั้น จะส่งแพ็กเกตยอมรับกลับมา(dhcpack) เครื่องเกตเวย์ก็จะไปแก้ค่าในตารางไอพีแอดเดรสว่าไอพีนั้นถูกใช้แล้ว

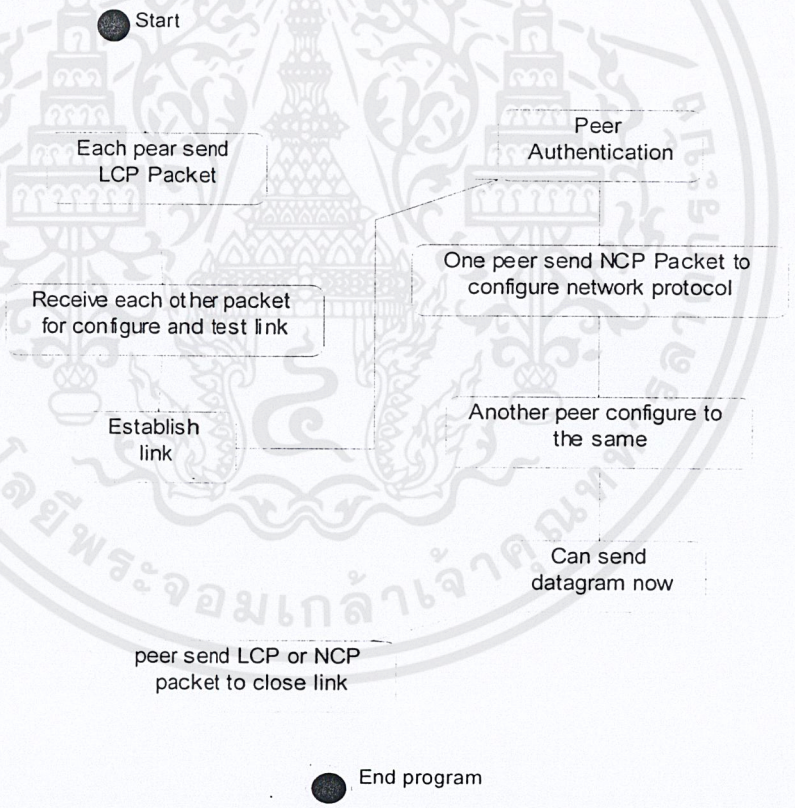
การออกแบบในระดับโครงสร้างการทำงานของส่วนที่ทำการติดต่อกับฮาร์ดแวร์จะมีสองส่วนคือส่วนที่ติดต่อกับแอ็กเซสพอยน์ และส่วนที่ติดต่อกับอุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอส สำหรับการออกแบบซอฟแวร์ในระดับขั้นการทำงานของส่วนที่ติดต่อกับแอกเซสพอยน์นั้นสามารถใช้อีเทอร์เน็ตแพ็กเกตไดร์เวอร์ที่มีอยู่บนชุดพัฒนาคอม86ได้ ซึ่งเป็นโปรแกรมชื่อว่าพีเอ็นพีพีดี(PNPPD) เนื่องจากการติดต่อกับอุปกรณ์แอกเซสพอยน์เป็นแบบเดียวกับการติดต่อกับอุปกรณ์ภายในเครือข่ายท้องถิ่นอื่นๆเช่นฮับหรือสวิตช์เป็นต้น อย่างไรก็ตามซอฟแวร์ในโมดูลอื่นๆที่ต้องมีการเขียนส่วนที่ทำการติดต่อกับแพ็กเกตไดร์เวอร์เพื่อรับเอาแพ็กเกตมาใช้งาน หลักการทำงานของโปรแกรมส่วนที่ทำการติดต่อกับอีเทอร์เน็ตแพ็คเกตไดร์เวอร์แสดงดังรูปที่ 6-10



รูปที่ 6-10 แสดงการเขียนโปรแกรมติดต่อเพื่อรับเฟรมข้อมูลจากอีเทอร์เน็ตแพ็กเกตไดร์เวอร์

การเขียนโปรแกรมในส่วนของการติดต่อเพื่อรับข้อมูลจากอีเทอร์เน็ตแพ็กเกตไดร์เวอร์ จะทำได้โดยการเรียกใช้ซอฟแวร์อินเตอร์รัพท์ โดยจะไปเรียกฟังก์ชันของการตั้งค่าเริ่มต้นให้กับแพ็กเกตไดร์เวอร์ ซึ่งต้องเขียนขึ้นเอง เมื่อได้รับเฟรมข้อมูลเข้ามาส่วนของฟังก์ชันที่รับเฟรมจะเซ็ตแฟลกเพื่อบอกแก่โปรแกรมที่ผู้ใช้เขียนขึ้นว่าได้รับเฟรมข้อมูลเข้ามา และทำการจัดเก็บลงในบัฟเฟอร์เพื่อรอโปรแกรมนำไปใช้

หลักการทำงานของส่วนที่ติดต่อกับอุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอส จะคล้ายกับการติดต่อกับโมเด็มผ่านพอร์ตอนุกรม โดยการเขียนโปรแกรมในส่วนนี้จะต้องทราบรีจิสเตอร์ที่ทำการติดต่อกับพอร์ตอนุกรมของชุดพัฒนาคอม86 แล้วทำการส่งคำสั่งเอทีคอมมานด์(ตามที่ได้กล่าวไปแล้วในบทก่อนหน้านี้) เพื่อสั่งให้อุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอสทำการเชื่อมต่อเข้าสู่เครือข่าย จากนั้นจะทำงานตามพีพีพีโปรโตคอล หลักการทำงานของโปรแกรมในส่วนนี้แสดงดังรูปที่ 6-11



รูปที่ 6-11 แผนภาพแสดงสถานะการทำงานของโปรแกรมในส่วนที่ทำงานตามพีพีพีโปรโตคอล

การออกแบบในระดับโครงสร้างการทำงานของส่วนที่ทำการปรับแต่งคุณสมบัติต่างๆของอุปกรณ์เกตเวย์หรือ โมดูลคอนฟิกูเรชัน ได้มีการออกแบบให้ผู้ดูแลระบบสามารถปรับแต่งค่าคุณสมบัติของอุปกรณ์เกตเวย์ได้ 2 ทางคือ ผ่านทางสายอนุกรมด้วยโปรแกรมเทอร์มินอลทั่วไป และผ่านทางโปรโตคอลเอชทีทีพี (HTTP) ด้วยเวบบราวเซอร์ ซึ่งหลักการของการปรับแต่งค่าของโมดูลอื่นๆใช้การเปิดไฟล์ที่เป็นตัวกำหนดคุณสมบัติของแต่ละโมดูลขึ้นมาแก้ไขและบันทึกเก็บไว้ เมื่อโมดูลที่ถูกทำการแก้ไขเข้ามาอ่านไฟล์เหล่านี้อีกครั้งจึงจะมีการปรับคุณสมบัติไปตามที่ผู้ดูระบบต้องการ

หลักการออกแบบในส่วนของการปรับแก้คุณสมบัตินี้ มีจุดหลักที่ต้องคำนึงถึงก็คือ ขณะที่ผู้ดูแลระบบทำการป้อนคำสั่งเพื่อปรับแก้ค่าต่างๆ อุปกรณ์เกตเวย์จะต้องยังคงสามารถทำงานหลักคือการรับส่งแพ็กเกตข้อมูลได้ตามปกติ โดยที่ไม่ต้องหยุดการทำงานและรอรับการป้อนคำสั่งจากผู้ใช้ ทำให้การออกแบบโปรแกรมที่ส่วนนี้ต้องใช้หลักการของการโปรแกรมแบบมัลติทาสก์เข้ามาช่วยเป็นอย่างมาก โดยเฉพาะในส่วนของการปรับแก้ค่าผ่านทางสายอนุกรม เพราะผู้ดูแลระบบต้องทำการปรับแก้ค่าด้วยการป้อนคำสั่ง ส่วนการปรับแก้ค่าผ่านทางเวบบราวเซอร์นั้น ไม่จำเป็นต้องคำนึงถึงส่วนนี้มากนักเพราะการส่งข้อมูลผ่านทางโปรโตคอลเอชทีทีพี จะเกิดขึ้นเมื่อผู้ดูแลระบบกดปุ่มยอมรับและส่งค่ามาให้ที่อุปกรณ์เกตเวย์เท่านั้น รูปที่6-12 เป็นการแสดงสถานะการทำงานของโปรแกรมในส่วนที่เป็นการปรับแก้ค่าของอุปกรณ์เกตเวย์

รูปที่6-12 แสดงแผนภาพสถานะ การทำงานของโปรแกรมส่วนของการปรับแก้ค่าคุณสมบัติ

6.3 การเขียนโปรแกรมเพื่อติดต่อกับแพ็กเกตไดร์เวอร์

6.3.1 การทำงานของแพ็กเกตไดร์เวอร์

การที่คอมพิวเตอร์จะสามารถเชื่อมต่อกันเป็นระบบเครือข่ายท้องถิ่นได้นั้น แต่ละเครื่องคอมพิวเตอร์จำเป็นจะต้องมีเน็ตเวิร์กการ์ดในการเชื่อมต่อกัน และการที่แอพพลิเคชันบนเครื่องคอมพิวเตอร์จะสามารถใช้งานระบบเครือข่ายได้นั้น แต่ละแอพพลิเคชันต้องทำความรู้จักกับตัวเน็ตเวิร์กการ์ดนั้นผ่านทางไดร์เวอร์ของเน็ตเวิร์กการ์ดนั้นๆ ซึ่งในปัจจุบันได้มีการพัฒนาให้มีมาตรฐานที่ตรงกันให้แต่ละแอพพลิเคชันสามารถใช้ไดร์เวอร์ร่วมกันและในเวลาเดียวกันได้ โดยไดร์เวอร์นี้เป็นรู้จักกันว่า แพ็กเกตไดร์เวอร์(Packet Driver)

ลักษณะการทำงานของแพ็กเกตไดร์เวอร์จะเข้าไปควบคุมเน็ตเวิร์กการ์ด โดยจะทำงานฝังอยู่ในหน่วยความจำ ทำหน้าที่จัดเตรียมแพ็กเกตให้กับโปรโตคอลในชั้นสูงขึ้นไป โดยเมื่อได้รับเฟรมข้อมูลเข้ามา แพ็กเกตไดร์เวอร์จะตรวจดูส่วนที่เป็นชนิดของเฟรมข้อมูลนั้นๆ ว่าตรงกับชนิดของโปรโตคอลที่มีการติดตั้งอยู่หรือไม่ ถ้าตรงก็จะส่งผ่านข้อมูลที่มีในเฟรมนั้นขึ้นไปให้ หากไม่ตรงกับโปรโตคอลใดเลย เฟรมนั้นจะถูกยกเลิกทิ้ง เพื่อเตรียมพร้อมสำหรับรับเฟรมต่อไป โดยชนิดของเฟรมที่อีเทอร์เน็ตแพ็กเกตไดร์เวอร์ทำงานได้คือ อีเทอร์เน็ต และ IEEE 802.3

6.3.2 การเรียกใช้โปรแกรมแพกเก็ตไดร์เวอร์

ขั้นตอนการใช้โปรแกรมแพ็กเกตไดร์เวอร์มีดังนี้

- เลือกตัวโปรแกรมแพกเก็ตไดร์เวอร์ ให้ตรงกับเน็ตเวิร์กการ์ดที่ใช้อยู่ ในที่นี้จะเลือกแพ็กเกตไดร์เวอร์ที่ตรงกับอีเทอร์เน็ตคอนโทรลเลอร์ที่มีอยู่บนชุดพัมนาคอม86 ได้แก่ พีเอ็นพีพีดี (PNPPD) ซึ่งเป็นซอฟแวร์ที่มีมาให้พร้อมกับชุดพัฒนา
- การเรียกใช้โปรแกรมแพกเก็ตไดร์เวอร์ ที่ได้มา โดยมีรูปแบบดังนี้

  A:/>pnppd packet_int_no

  Packet_int_no:

จะเป็นตำแหน่งอินเทอรร์รัพต์ของซอฟแวร์ที่แพ็กเกตไดร์เวอร์ จะใช้เพื่อติดต่อกับโปรแกรมที่ผู้ใช้สร้างขึ้น หมายเลขอินเตอร์รัพต์นี้ เลือกใช้ได้ตั้งแต่ 0x60 ถึง 0x7E แต่จะต้องไม่ไปทับซ้อนกับการทำงานของโปรแกรมอื่นๆ โดยทั่วไปแล้วค่าปกติเมื่อไม่ได้กำหนดจะ เป็น 0x60

แพ็กเกตไดร์เวอร์จะทำหน้าที่เป็น รูทีนบริการการอินเตอร์รัพต์ (Interrupt Service Routine) ของเน็ตเวิร์กการ์ด เพราะฉะนั้นการเรียกใช้งาน จะต้องไม่มีโปรแกรมใดที่ทำหน้าที่ในลักษณะเดียวกันนี้ ทำงานอยู่ก่อน อาทิเช่น ไดร์เวอร์ของเน็ตเวิร์กการ์ด

6.3.3 การเขียนโปรแกรม

การทำงานของแพ็กเกตไดร์เวอร์ แบ่งออกเป็น 3 ระดับ

(1) การทำงานในระดับขั้นพื้นฐาน (Basic Packet Driver) มีฟังก์ชันการทำงานให้เรียกใช้น้อยที่สุด แต่สามารถนำไปใช้ได้ง่าย อีกทั้งยังใช้ทรัพยากรของเครื่องน้อย การทำงานในระดับขั้นนี้คือ การส่งข้อมูลแบบกระจาย (Broadcast) และการรับแพ็กเกตข้อมูล

(2) การทำงานในระดับขั้นสูง (Extended Packet Driver) ครอบคลุมฟังก์ชันการทำงานของ Basic Packet Driver แต่ความสามารถในการส่งข้อมูลแบบกระจายเป็นกลุ่ม (Multicast) และเก็บรวบรวมสถิติการใช้งานแพกเก็ตไดร์เวอร์

(3) การทำงานในระดับประสิทธิภาพสูง(High Performance) มีวัตถุประสงค์เพื่อเพิ่มประสิทธิภาพการทำงานของแพกเก็ตไดร์เวอร์

การติดต่อกับแพกเก็ตไดร์เวอร์ ทำได้โดยการเรียกใช้ ซอฟท์แวร์อินเทอร์รัพต์ ในช่วง 0X60 ถึง 0X80 โดยที่จะเรียกใช้การทำงานใดนั้นขึ้นกับค่าที่ผ่านให้ทางรีจิสเตอร์ AH

ตัวอย่างการเรียกใช้ฟังก์ชันที่จะแสดงต่อไปนี้ เขียนอยู่ในรูปแบบของภาษาซีและภาษาแอสเซมบลี แพ็กเกตไดร์เวอร์จะเก็บค่าของรีจิสเตอร์ต่างๆที่ต้องนำไปใช้งาน ในขณะที่เข้าไปทำงานนั้นๆและจะคืนค่าดังกล่าวเมื่อทำงานเสร็จ ค่าของรีจิสเตอร์ที่ต้องเก็บคือ DS, SS, SP และแฟลก

ฟังก์ชันการทำงานต่างๆของแพ็กเก็ตไดร์เวอร์มีดังนี้

(1) Driver_info ( )

```
driver_info (handle)                    AH == 1, AL = 255
        int handle;                     BX      /* Optional */

error return:
        carry flag set
        error code                      DH
        possible errors:
                BAD_HANDLE                      /* Older drivers only */

non-error return:
        carry flag clear
        version                         CH
        class                           DX
        type                            CX
        number                          CL
        name                            DS : SI
```

functionality	AL
	1 == basic functions present.
	2 == basic and extended present.
	5 == basic and high-performance.
	6 == basic, high-performance, extend.
	255 == not installed.

ตารางที่ 6-1 แสดงการทำงานของฟังก์ชัน Driver_Info()

จะแสดงข้อมูลต่างๆที่เกี่ยวข้องกับแพ็คเกตไดร์เวอร์ที่มีการเรียกใช้ ค่าเวอร์ชันจะเป็นที่สมมติขึ้นเพื่อให้ทราบถึงความสามารถในการทำงานได้ ในเวอร์ชันก่อนหน้าเวอร์ชัน 1.07 การเรียกใช้ฟังก์ชันนี้จะต้องส่งค่าของส่วนจัดการอินเตอร์รัพท์(Interrupt handle) ให้ด้วย ซึ่งค่าของส่วนจัดการอินเตอร์รัพท์ นี้ได้มาจากการทำงานในฟังก์ชัน access_type

(2) Access_type()

int access_type(if_class,if_type,if_number,type,typelen,receiver)	AH=2
int if_class;	AL
int if_number;	BX
int if_number;	DL
char far *type;	DS : SI
unsigned typelen;	CX
int (far *receiver) ( );	ES : DI
error return:	
carry flag clear	
handle	AX
receicer call :	
(*receiver) (handle, flag, len [,buffer])	
int handle;	BX
int flag;	AX
unsigned len;	CX
if AX == 1	
char far *buffer	DX : SI

ตารางที่ 6-2 แสดงการทำงานของฟังก์ชัน Access_Type

ใช้เพื่อตรวจสอบข้อมูลแพ็คเกตว่าเป็นชนิดตามที่ตรวจสอบหรือไม่ ค่าของชนิดที่ส่งผ่านให้นั้นจะเป็นตัวที่ชี้ไปยังตำแหน่งส่วนที่แสดงชนิดของแพ็คเกตนั้นๆ ค่าขนาดของชนิดแพ็คเกตจะเป็นขนาดความยาวของแพ็คเกต ค่ารีซีฟเวอร์จะปีนตัวชี้ตำแหน่งของส่วนการทำงาน เมื่อได้รับแพ็คเกต

เมื่อได้รับแพ็คเกต ส่วนของรีซีฟเวอร์จะถูกเรียกทำงาน 2 ครั้ง ครั้งที่ 1 จะเรียกเพื่อให้แอพพลิเคชันเตรียมสำรองเนื้อที่บัฟเฟอร์ไว้ สำหรับแพ็คเกต สังเกตได้จากค่าในรีจิสเตอร์ AX = 0 แอพพลิเคชันจะคืนค่าตำแหน่งบัฟเฟอร์โดยใช้ ES:DI เป็นตัวชี้ ถ้าหารว่าแอพพลิเคชันไม่สามารถหาเนื้อที่ได้เพียงพอ ก็จะส่งค่า 00:00 ผ่านทาง ES:DI ซึ่งจะทำให้แพ็คเกตไดร์เวอร์ ทิ้งแพ็คเกตที่ได้รับมานั้น และจะไม่ทำการเรียกรีซีฟเวอร์ทำงานเป็นครั้งที่ 2

ค่าความยาวของแพ็คเกตไดร์เวอร์ที่เข้ามานั้น จะเก็บอยู่ใน CX เมื่อตอนเรียกให้ รีซีฟเวอร์ ครั้งแรก เพื่อให้แอพพลิเคชันทราบว่า จะต้องสำรองเนื้อที่ขนาดเท่าไร
ในการเรียก รีซีฟเวอร์ ครั้งที่ 2 จะเป็นการบอกให้ทราบว่า ได้ทำสำเนาแพ็คเกตลงในบัฟเฟอร์เรียบร้อยแล้ว สังเกตได้จากรีจิสเตอร์ AX จะมีค่าเท่ากับ 1

(3) Release_type ( )

int release_type (handle)	AH == 3
int handle;	BX
error return :	
carry flag set	
error code	DH
possibld errors :	
Bad_HANDLE	
non – error return :	
carry flag clear	

ตารางที่6-3 แสดงการทำงานของฟังก์ชัน Release_type()

ฟังก์ชันนี้จะเป็นการบอกเลิกค่า แฮนเดิล ที่ได้มาจากการทำงานในฟังก์ชัน access_type( )

(4) Send_pkt ( )

int send_pkt (buffer, length)	AH == 4

	char far *buffer;	DS : SI
	unsigned length;	CX
error return :		
	carry flag set	
	error code	DH
	possible errors :	
	CANT_SEND	
non – error return :		
	carry flag clear	

ตารางที่6-4 แสดงการทำงานของฟังก์ชัน Send_pkt()

เป็นฟังก์ชันสำหรับส่งแพ็คเกต แอพพลิเคชันที่เรียกใช้ฟังก์ชันนี้ จะต้องเตรียมข้อมูลแพ็คเกตให้พร้อมสรรพ รวมถึงในส่วนที่เป็นอีเทอร์เน็ตเฮดเดอร์ ที่แสดงค่าหมายเลขที่อยู่ต่างๆด้วย เพราะอยู่นอกเหนือการทำงานของแพ็คเกตไดร์เวอร์

(5) Terminate ( )

	terminate (handle)	AH == 5
	int handle;	BX
error return :		
	carry flag set	
	error code	DH
	possible errors :	
	BAD_HANDLE	
	CANT_TERMINATE	
non – error return :		
	carry flag clear	

ตารางที่6-5 แสดงการทำงานของฟังก์ชัน Terminate()

ทำงานสัมพันธ์กับค่าแฮนเดิล ซึ่งหากการทำงานไม่เกิดปัญหาใดๆ จะเป็นการคืนหน่วยความจำที่ถูกใช้โดยแพ็คเกตไดร์เวอร์ ให้กับดอส

(6) Get_address( )

get_address(handle, buf, lan)	AH == 6
int handle;	BX
char far *buf	ES : DI
int len;	CX
error return :	
carry flag set	
error code	DH
possible errors :	
BAD_HANDLE	
NO_SPACE	
non – error return :	
carry flag clear	
length	CX

ตารางที่ 6-6 แสดงการทำงานของฟังก์ชัน Get_Address()

เพื่อเก็บค่าหมายเลขที่อยู่ของเครือข่าย ลงในบัฟเฟอร์ โดยจะมีความยาวเ ท่ากับค่า len และจะส่งคืนค่าจำนวนไบต์ที่ได้รับการเก็บลงในบัฟเฟอร์ผ่านทางรีจิสเตอร์ CX ถ้าหารว่าเกิดความผิดพลาด NO_SPACE ขึ้น แสดงว่าเนื้อที่บัฟเฟอร์ ไม่เพียงพอในการจัดเก็บ

(7) Reset_interface ( )

reset_interface (handle)	AH == 7
int handle;	BX
error return :	
carry flag set	
error code	DH
possible errors :	
BAD_HANDLE	
CANT_RESET	
non – error return :	

```
        carry flag clear
```

ตารางที่ 6-7 แสดงการทำงานของฟังก์ชัน Reset_Interface()

เป็นฟังก์ชันการทำงานเพื่อรีเซตแพ็คเกตไดร์เวอร์ คือยกเลิกแพ็คเกตที่เตรียมส่ง เตรียมพร้อมให้เริ่มรับแพ็คเกตใหม่ กำหนดโหมดการรับแพ็คเกตไว้ที่โหมด 3 (รับเฉพาะแพ็คเกตของตนแบบกระจายทุกจุด)

(8) Get_parameters ( )

```
High – performance driver function
        Get_parameters ( )                              AH == 10

Error return :
        carry flag set
        error code                                      DH
        possible errors :
                BAD_COMMAND                             /* No high – performance support */
non – error return :
        carry flag clear
        struct param far *;                             ES : DI
struct param {
  unsigned char major_rev;                              /* Revision of Packet Driver spec */
  unsigned char minor_rev;                              /*   this driver conforms to.   */
  unsigned char length;                                 /* Length of structure in bytes */
  unsigned char addr_len;                               /* Length of a MAC – layer address */
  unsigned short mtu;                                   /* MTU. Including MAC headers */
  unsigned short multicast_aval;                        /* Buffer size for multicast addr */
  unsigned short rcv_bufs;                              /* (# of back–to–back MTU rcvs) – 1 */
  unsigned short xmt_bufs;                              /* (#of successive xmits) – 1 */
  unsigned sort int_num;                                /* Interrupt # to hook for post – EOI
                                                           processing. 0 == none */
};
```

ตารางที่ 6-8 แสดงการทำงานของฟังก์ชัน Get_parameters

ใช้เพื่อปรับเพิ่มประสิทธิภาพการทำงานของแอพพลิเคชันที่ใช้งานแพ็คเกตไดร์เวอร์ โดยฟังก์ชันนี้จะคืนค่าต่างๆที่จำเป็นต่อการปรับแต่งแอพพลิเคชันให้เหมาะสมกับการทำงานของแพ็คเกตไดร์เวอร์นั้นกลับมาทาง param ฟังก์ชันนี้ได้เริ่มมีในแพ็คเกตไดร์เวอร์ตั้งแต่เวอร์ชัน 1.09 (ตรวจสอบโดยใช้ driver_info ( ))

(9) As_send_pkt ( )

High performance driver function	
int as_send_packet (buffer, length, upcall)	AH == 11
char far *buffer;	DS : SI
unsigned length;	CX
int (far *upcall) ( );	ES : DI
error return :	
carry flag set	
error code	DH
possible errors :	
CANT_SEND	/* transmit error, re-entered, etc. */
BAD_COMMAND	/* Level 0 or 1 driver */
non – error return :	
carry flag clear	
buffer available upcall :	
(*upcall) (buffer, result)	
int result;	AX /* 0 for copy of */
char far * buffer;	ES : DI /* from as_send_pkt ( ) call */

ตารางที่ 6-9 แสดงการทำงานของฟังก์ชัน As_send_pkt

การทำงานคล้ายคลึงกับ send_pkt( ) แต่ต่างกันตรงที่บัฟเฟอร์ที่ใช้เก็บข้อมูลแพ็คเกตนั้น หากเป็น send_pkt( ) จะต้องรอจนกว่าจะทำงานในส่วนของ send_pkt( ) เสร็จเสียก่อน จึงจะสามารถนำไปใช้ได้ ส่วนหากเป็น as_send_pkt( ) สามารถนำไปใช้ได้ในทันที ที่เรียกใช้ฟังก์ชันนี้ ฟังก์ชันนี้ได้เริ่มมีในแพ็คเกตไดร์เวอร์ตั้งแต่เวอร์ชัน 1.09 (ตรวจสอบโดยใช้ driver_info ( ))

(10) Set_rcv_mode ( )

Extended driver function	
set_rcv_mode (handle, mode)	AH == 20

int handle;	BX
int mode;	CX
error return :	
carry flag set	
error code	DH
possible errors :	
BAD_HANDLE	
BAD_MODE	
non – error return :	
carry flag clear	

ตารางที่6-10 แสดงการทำงานของฟังก์ชัน Set_rcv_mode()

จะต้องผ่านค่าแฮนเดิลให้กับฟังก์ชันนี้เพื่อกำหนดโหมดในการรับแพ็คเกต โหมดในการรับแพ็คเกตมีดังนี้

1. ไม่รับแพ็คเกตใดๆ
2. รับเฉพาะที่ส่งมายังเครื่องนี้
3. โหมด 2 + แบบที่ส่งกระจายทุกจุด
4. โหมด 3 + แบบที่ส่งกระจายเป็นกลุ่มตามที่กำหนดไว้ฯ
5. โหมด 3 + แบบที่ส่งกระจายเป็นกลุ่มทุกๆกลุ่ม
6. รับทุกๆแพ็คเกต

แพ็คเกตไดร์เวอร์บางตัว ไม่สามารถทำการรับแพ็คเกตในบางโหมดได้ ในส่วนการรับแพ็คเกตแบบกระจายเป็นกลุ่ม สามารถกำหนดรายละเอียดได้ (ฟังก์ชัน get_multicast_lis( ) และ set_multicast_list) และหากไม่มีการกำหนดค่าโหมดการทำงานเป็นอื่น แพ็คเกตไดร์เวอร์จะกำหนดให้ค่าโหมด 3 เป็นค่าเริ่มต้น

(11) Get_rcv_mode ( )

Extended driver function	
get_rcv_mode (handle, mode)	AH == 21
int handle;	BX
error return :	
carry flag set	
error code	DH

possible errors :	
BAD_HANDLE	
non – error return :	
carry flag clear	
mode	AX

ตารางที่ 6-11 แสดงการทำงานของฟังก์ชัน Get_rcv_mode()

แสดงค่าโหมดการรับแพ็คเกต ที่ได้กำหนดๆไว้ ณ เวลาปัจจุบัน

(12) Set_multicast_list ( )

Extended driver function	
get_rcv_mode (handle, mode)	AH == 21
int handle;	BX
error return :	
carry flag set	
error code	DH
possible errors :	
BAD_HANDLE	
non – error return :	
carry flag clear	
mode	AX

ตารางที่ 6-12 แสดงการทำงานของฟังก์ชัน Set_multicast_list()

เป็นฟังก์ชันเพื่อกำหนดค่าของหมายเลขที่อยู่ ซึ่งจะใช้เป็นหมายเลขที่อยู่ในการทำงานแบบรับส่งแพ็คเกตเป็นกลุ่ม อาจจะใช้ร่วมกับ get_multicast_list ( ) เพื่อนำค่าที่กำหนดไว้ในปัจจุบัน มาปรับเปลี่ยนแก้ไข

(13) Get_multicast_list ( )

Extended driver function	
get_multicast_list ( )	AH == 23
error return :	
carry flag set	
error code	DH

```
        possible errors :
                NO_MULTICAST
                NO_SPACE
non – error return :
        carry flag clear
        char far *addrlst;                              ES : DI
        int len;                                        CX
```

ตารางที่ 6-13 แสดงการทำงานของฟังก์ชัน Get_multicast_list()

เป็นฟังก์ชันเพื่อแสดงค่าของมัลติคาสก์ลิสต์ ที่ใช้อยู่ในปัจจุบัน

(14) Get_statistics ( )

```
Extended driver function
        get_statistics ( handle )                       AH == 24
                int handle;                             BX

error return :
        carry flag set
        error code                                      DH
        possible errors :
                BAD_HANDLE

non – error return :
        carry flag clear
        char far *stats;                                DS : SI

struct statistics {
  unsigned long packet_in                   /* Totals across all handles */
  unsigned long packet_out;
  unsigned long bytes_in;                   /* Including MAC headers */
  unsigned long bytes_out;
  unsigned long errors_in;                  /* Totals across all error types */
  unsigned long errors_out;
  unsigned long packets_lost;               /* No buffer from receiver( ), card */
```

```
                                                    /* out of resources, etc */
};
```

ตารางที่ 6-14 แสดงการทำงานของฟังก์ชัน Get_statistics()

แสดงค่าสถิติของแพ็คเกตที่มีการรับส่งผ่านแพ็คเกตไดร์เวอร์

(15) Set_address ( )

```
Extended_driver function
        set_address ( addr, len )                       AH == 25
                char far *addr;                         EX : DI
                int len;                                CX

error return :
        carry flag set
        error code                                      DH
        possible errors :
                CANT_SET
                BAD_ADDRESS

non – error return :
        carry flag clear
        length                                          CX
```

ตารางที่ 6-15 แสดงการทำงานของฟังก์ชัน Set_address()

ใช้เพื่อกำหนดหมายเลขที่อยู่ให้มีรูปแบบ สัมพันธ์กับการทำงานของโพรโตคอลที่อยู่ในระดับชั้นบน

สรุปหมายเลขฟังก์ชันเพื่อใช้ติดต่อกับแพ็คเกตไดร์เวอร์

หมายเลขฟังก์ชันในรูปแบบเลขฐานสิบต่อไปนี้ จะใช้ติดต่อกับแพ็คเกตไดร์เวอร์ เพื่อให้ทำงานตามที่ต้องการ โดยหมายเลขดังกล่าวจะถูกส่งไปยังแพ็คเกตไดร์เวอร์ ผ่านทางรีจิสเตอร์ AH

driver_info	1
access_type	2
release_type	3
send_pkt	4
terminate	5

get_address	6
reset_interface	7
+get_parameters	10
+as_send_pkt	11
*set_rcv_mode	20
*get_rcv_mode	21
*set_multicast_list	22
*get_multicast_list	23
*get_statistics	24
*set_address	25

+ เป็นฟังก์ชันการทำงานในระดับประสิทธิภาพสูง

* เป็นฟังก์ชันการทำงานในระดับขั้นสูง

ค่าในช่วงของ 128 ถึง 255 (0X80 ถึง 0XFF) สำหรับให้ผู้ใช้กำหนดฟังก์ชันการทำงานเอง

สรุปหมายเลขแสดงข้อผิดพลาด

1. BAD_HANDLE	ค่าแฮนเดิลผิดพลาด
2. NO_CLASS	ไม่พบคลาสผิดพลาด
3. NO_TYPE	ไม่พบชนิดของแพ็คเกตดังกล่าว
4. NO_NUMBER	ไม่พบหมายเลขดังกล่าว
5. BAD_TYPE	ไม่รู้จักชนิดแพ็คเกตนั้น
6. NO_MULTICAST	ไม่สามารถทำงานแบบส่งกระจายเป็นกลุ่มได้
7. CANT_TERMINATE	ไม่สามารถออกจากโปรแกรมแพ็คเกตไดร์เวอร์
8. BAD_MODE	กำหนดโหมดที่ใช้ในการรับแพ็คเกตผิดพลาด
9. NO_SPACE	การทำงานผิดพลาด อันมีสาเหตุมาจากเนื้อที่ไม่เพียงพอ
10. TYPE_INUSE	ชนิดแพ็คเกตนั้น กำลังทำงานอยู่
11. BAD_COMMAND	ไม่รู้จักคำสั่งนั้นๆ
12. CANT_SEND	ไม่สามารถส่งแพ็คเกตออกไปได้ (มักมีสาเหตุมาจากฮาร์ดแวร์)
13. CANT_SET	ไม่สามารถแก้ไขค่าหมายเลขที่อยู่ของฮาร์ดแวร์ได้
14. BAD_ADDRESS	ค่าหมายเลขที่อยู่ผิดพลาด
15. CANT_RESET	ไม่สามารถทำการรีเซตได้

การเขียนโปรแกรมเพื่อรับเฟรมข้อมูล โดยทำงานติดต่อกับแพ็คเกตไดร์เวอร์ จะแบ่งการทำงานออกเป็น 2 ส่วนคือ

(1) ส่วนรูทีนบริการอินเทอร์รัพต์ (Interrupt Service Routine)

(2) ส่วนโปรแกรมของผู้ใช้ (Use's Main Program)

และทั้งสองส่วนนี้จะติดต่อกันโดยใช้ แฟลก (Flag)

ส่วนรูทีนบริการอินเทอร์รัพต์ (Interrupt Service Routine) แบ่งออกเป็น 2 ส่วนย่อยคือ

- ส่วนของแพ็คเกตไดร์เวอร์ (Packet Driver)

ทำหน้าที่เป็น *รูทีนบริการอินเทอร์รัพต์ของฮาร์ดแวร์* ควบคุมเน็ตเวิร์กการ์ด เมื่อมีอินเทอรัพต์เกิดขึ้นกับเน็ตเวอร์กการ์ด อาทิเช่น เมื่อตอนที่เน็ตเวอร์กการ์ดรับข้อมูลเข้ามา ก็จะเข้าไปทำในรูทีนดังกล่าวนี้

- โพรซีเดอร์ที่ทำหน้าที่ส่งสัญญาณผ่านแฟลก (Procedure pktReceiver)

ทำหน้าที่ติดต่อกับส่วนโปรแกรมหลักของผู้ใช้ โดยจะถูกเรียกใช้โดยแพ็คเกตไดร์เวอร์โพรซีเดอร์นี้ จะถูกเรียกใช้ 2 ครั้งโดยแพ็คเกตไดร์เวอร์ ทุกครั้งที่มีเฟรมเข้ามา 1 เฟรม ครั้งที่ 1. แพ็คเกตไดร์เวอร์จะเรียกใช้โพรซีเดอร์ครั้งแรกนี้ เพื่อให้ทำการ สำรองเนื้อที่บัฟเฟอร์ รอไว้ โดยตรวจสอบได้จากค่าที่ผ่านมาทางรีจิสเตอร์ AX จะเท่ากับศูนย์

AX = 0

CX = ขนาดของเฟรมที่รวมส่วนเฮดเดอร์ ประกอบด้วยฟิลด์ที่อยู่ปลายทาง, ต้นทางและชนิดแพ็คเกต นั่นหมายความว่าขนาดเฟรมจะเท่ากับขนาดข้อมูลในฟิลด์ข้อมูล บวก 18 ไบต์

คุณจะต้องส่งตัวชี้ (Pointer) ของตำแหน่งบัฟเฟอร์ที่ว่าง โดยผ่านทางรีจิสเตอร์ ES: DI ไปให้กับแพ็คเกตไดร์เวอร์ หากไม่ต้องการจะรับเฟรมข้อมูลนั้นให้ใส่ค่า 0000:0000 ที่รีจิสเตอร์ ES: DI

การสำรองเนื้อที่บัฟเฟอร์จะต้องทำตั้งแต่ตอนเรียกโปรแกรม โดยมักจะใช้ในรูปแบบของข้อมูลแบบอาร์เรย์(Array) ถ้าไม่เช่นนั้นแล้วจะไม่สามารถสำรองเนื้อที่ได้ เพราะไม่สามารถเรียกใช้การทำงานของดอสได้ (Int 21H) ขณะกำลังอยู่ในรูทีนบริการอินเทอร์รัพต์ (Interrupt Service Routine)

ครั้งที่ 2. แพ็คเกตไดร์เวอร์จะเรียกใช้โพรซีเดอร์นี้อีกครั้งหนึ่งเมื่อข้อมูลเฟรมนั้นๆ ได้จัดเก็บลงในบัฟเฟอร์ที่สำรองไว้ในตอนแรกเรียบร้อยแล้ว โดยตรวจสอบได้จากค่าที่ผ่านมาทางรีจิสเตอร์ AX จะเท่ากับหนึ่ง เป็นการบอกให้ทราบว่าเฟรมข้อมูลทั้งหมด โดยรวมส่วนเฮดเดอร์ (ขนาด 6+6+2 ไบต์) ได้รับการทำสำเนามาเก็บไว้ในบัฟเฟอร์ เพื่อให้เรานำไปใช้งานได้แล้วการทำงานในส่วนโพรซีเดอร์นี้เป็นส่วนหนึ่งของรูทีนบริการอินเทอร์รัพต์ จึงไม่ควรให้ทำงานอยู่นาน ไม่เช่นนั้นแล้วจะไม่สามารถรับเฟรมข้อมูลถัดไปได้ทัน เนื่องจากอินเทอร์รัพต์จะถูกปิดกั้นไว้ไม่ให้ไปทำงานในส่วนรูทีนบริการอินเทอร์รัพต์ซ้ำซ้อนกันจะสามารถทำงานในรูทีนอีกครั้งหนึ่งได้ ก็ต่อเมื่อออกจากรูทีนนั้นเสียก่อน

ส่วนโปรแกรมหลักของผู้ใช้ (Use's Main Program) แบ่งออกเป็น 3 ส่วนย่อย

- ส่วนการกำหนดค่าเริ่มต้นให้กับแพ็คเกตไดร์เวอร์ (Initialize Packet Driver)

ทำหน้าที่กำหนดรายละเอียดต่างๆ ในการทำงานของแพ็คเกตไดร์เวอร์ ที่สำคัญคือกำหนดให้ ตัวชี้

ตำแหน่งการทำงาน เมื่อมีแพ็คเกตเข้ามา ชี้ยังตำแหน่งโพรซีเดอร์ที่ทำหน้าที่ส่งสัญญาณแฟลก

- ส่วนการประมวลผลเฟรม (Procedure Process)

โดยจะเข้ามาทำงานในส่วนนี้ได้ก็ต่อเมื่อได้รับ สัญญาณแฟลก ซึ่งหมายถึงข้อมูลของฟรมที่ได้มาเก็บอยู่ในบัฟเฟอร์เรียบร้อยแล้ว แต่จะนำไปใช้งานอย่างไรบ้างขึ้นอยู่กับผู้ใช้ อาทิเช่นนำข้อมูลที่อยู่ในบัฟเฟอร์แสดงผลออกทางหน้าจอคอมพิวเตอร์

- ส่วนที่จะทำงานเมื่อออกจากโปรแกรม (Terminate)

ซึ่งจะคืนค่าต่างๆให้กับตัวแพ็คเกตไดร์เวอร์ ที่สำคัญคือจะคืนค่าของ *ตัวชี้* ที่ชี้มายังโพรซีเดอร์ที่ทำหน้าที่ส่งสัญญาณแฟลก

ลำดับขั้นตอนการทำงานจะเป็นดังนี้

(1) เรียก **แพกเก็ตไดร์เวอร์** ให้ฝังอยู่ในหน่วยความจำเสียก่อน ตัวแพ็คเกตไดร์เวอร์จะเข้าไปควบคุมการทำงานของเน็ตเวิร์กการ์ด

(2) เรียกใช้ **โปรแกรมหลักของผู้ใช้** โดยลำดับขั้นตอนในโปรแกรมหลักของผู้ใช้จะเป็นดังนี้

- ทำการ **กำหนดค่าเริ่มต้น** ของแพ็คเกตไดร์เวอร์ ซึ่งจะทำให้ ตัวชี้ตำแหน่งการทำงานเมื่อมีแพ็คเกตเข้ามา ชี้มาที่ส่วนโพรซีเดอร์ที่ทำหน้าที่ส่งสัญญาณแฟลก
- หลังจากนั้นตัวโปรแกรมหลักนี้ จะไป **หยุดรอ** ว่าเมื่อไรจะมีเฟรมของข้อมูลส่งมา
- เมื่อมีเฟรมของข้อมูลส่งมาแล้ว จึงไปทำการ **ประมวลผลเฟรมข้อมูล** ชุดนั้นๆ

(3) ในกรณีที่ **ไม่มีเฟรมข้อมูล** ส่งเข้ามา

- ส่วนของโพรซีเดอร์ที่ทำหน้าที่ส่งสัญญาณแฟลก จะไม่ทำงาน
- ส่วนของโปรแกรมหลักของผู้ใช้ จะไปหยุดรอจนกว่าจะมีเฟรมส่งเข้ามา

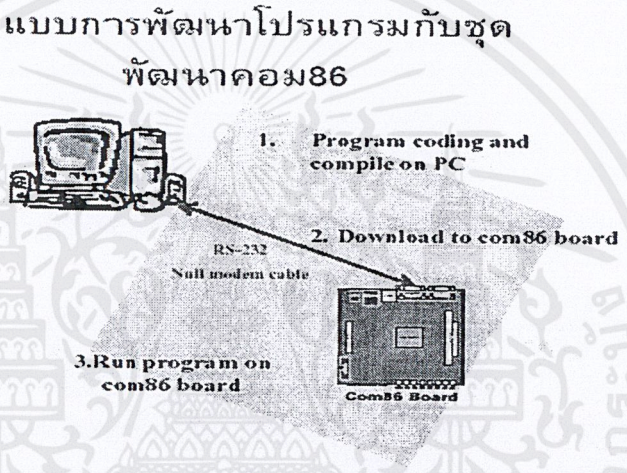
(4) ในกรณีที่ **มีเฟรมข้อมูล** ส่งเข้ามา

- ส่วนของ**รูทีนบริการอินเทอร์รัพต์**จะทำงาน โดยเริ่มจาก **รูทีนบริการอินเทอร์รัพต์ของฮาร์ดแวร์** *(แพ็คเกต ไดร์เวอร์)* ก่อน และจะเข้าไปทำ **โพรซีเดอร์ที่ทำหน้าที่ส่งสัญญาณแฟลก** *(โพรซีเดอร์ pktReceiver)* ซึ่งจะไปเซตค่า **แฟลก FrameCount** เพื่อไปบอกโปรแกรมหลักของผู้ใช้ให้เริ่มทำงานได้
- ส่วน *โปรแกรมหลักของผู้ใช้* เมื่อเห็นว่าค่า แฟลก FrameCount เซตแล้วซึ่งหมายความว่าข้อมูลในเฟรมได้มาเก็บอยู่ในบัฟเฟอร์เรียบร้อยแล้ว จะเข้าไปทำงานใน ส่วนประมวลผลเฟรม และเมื่อทำงานเสร็จแล้ว จะกลับไป **รีเซตค่าแฟลก** เพื่อให้พร้อมรับเฟรมข้อมูลตัวถัดไป

(5) ก่อนออกจากโปรแกรมหลักของผู้ใช้ จะทำการคืนค่าที่กำหนดได้กับแพ็คเกตไดร์เวอร์ เมื่อตอนเริ่มต้นโปรแกรม โดยการเรียกใช้ **ส่วนการออกจากโปรแกรม**

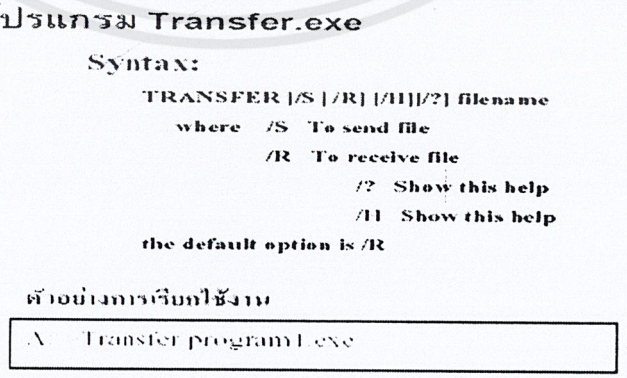
6.4 การพัฒนาโปรแกรมบนชุดพัฒนาคอม86

การพัฒนาโปรแกรมบนชุดพัฒนาคอม86 สามารถพัฒนาได้จากหลายภาษาด้วยกันอาทิเช่น ภาษาซี หรือซีพลัสพลัส ภาษาปาสคาล ภาษาแอสเซมบลี เป็นต้น โดยการพัฒนาโปรแกรมสามารถทำได้บนเครื่องคอมพิวเตอร์บุคคล (Personal computer) แล้วทำการส่งไฟล์ที่ผ่านการคอมไพล์และสร้างเป็นไฟล์ที่สามารถรันได้บนระบบปฏิบัติการดอส (executable file) ไปยังบอร์ดชุดพัฒนา แล้วทำการรัน รูปที่6-13 แสดงลำดับขั้นตอนพื้นฐานในการพัฒนาโปรแกรมทั่วไปบนชุดพัฒนาคอม86



รูปที่ 6-13 แสดงลำดับขั้นตอนพื้นฐานในการพัฒนาโปรแกรมทั่วไปบนชุดพัฒนาคอม86

การที่จะส่งไฟล์ไปไว้บนบอร์ดชุดพัฒนาได้นั้น จะส่งผ่านทางโปรแกรมเทอร์มินอลทั่วไป โดยในชุดพัฒนาคอม86 ได้มีโปรแกรมยูทิลิตี้ตัวหนึ่งในการรองรับการทำงานในส่วนนี้เป็นชื่อโปรแกรมว่า transfer.exe เป็นโปรแกรมที่ช่วยในการรับส่งไฟล์ต่างๆ ผ่านทางพอร์ตอนุกรม ซึ่งจะใช้โปรโตคอล เอ็กซ์-โมเด็ม (X-modem) ในการรับส่งไฟล์ที่ความเร็ว 57600 bps ซึ่งรูปแบบการใช้งานโปรแกรมแสดงดังรูปที่ 6-14



รูปที่ 6-14 แสดงรูปแบบการใช้โปรแกรมในการรับส่งไฟล์กับชุดพัฒนาคอม86

6.4.1 การใช้งานพอร์ตอนุกรมของชุดพัฒนาคอม86

สำหรับพอร์ตอนุกรมบนชุดพัฒนาคอม86 ในที่นี้หมายถึงพอร์ตคอม2(COM2) ซึ่งโดยมากจะใช้ไปควบคุมหรือติดต่อกับอุปกรณ์อื่นๆ ต่อไป โดยคุณสมบัติทั่วไปของพอร์ตอนุกรมนี้ได้แก่

- สามารถรับส่งข้อมูลได้ที่อัตราเร็วสูงสุด 115200 bps
- การส่งข้อมูลเป็นแบบสองทิศทางในเวลาเดียวกัน(Full-duplex)
- ใช้บิตที่เป็นสต็อปบิต 1 หรือ 2 บิตก็ได้

ในการใช้งานพอร์ตอนุกรมนี้ผู้เขียนโปรแกรมจำเป็นต้องทราบรายละเอียดของรีจิสเตอร์ตัวต่างๆของพอร์ตว่าต้องมีการส่งค่าใดไปให้บ้าง นอกจากนี้ยังจะต้องทราบอัตราเร็วของการรับส่งข้อมูลผ่านทางพอร์ตอนุกรมของอุปกรณ์ที่ชุดพัฒนาจะไปต่อเชื่อมด้วย สำหรับรายละเอียดของรีจิสเตอร์ต่างๆที่เกี่ยวข้องกับการรับส่งข้อมูลผ่านซีเรียลพอร์ต แสดงดังรูป 6-15

Register ที่เกี่ยวข้อง

SPCON0	Serial port control 0
SPCON1	Serial port control 1
SPSTAT	Serial port status
SPIMSK	Serial port Interrupt Mask
SPTXD	Serial port Transmit data
SPRXD	Serial port Receive data
SPRXDP	Serial port Receive data peak
SPBDV	Serial port Baud Rate Divisor

รูปที่ 6-15 แสดงรีจิสเตอร์ที่เกี่ยวข้องในการเขียนโปรแกรมเพื่อติดต่อผ่านทางพอร์ตอนุกรม

```
#include "am1866cc.h"
#include <dos.h>
#include <stdio.h>
#include <conio.h>

typedef unsigned char BYTE;
typedef unsigned int WORD;

void InitSerial(void)
```

```
{       WORD SysconRegister;
        WORD PIOMode1Register,PIODIR1Register;
        outport(SPCON0,0x60); // initial serial control register
        /* Parity and stop bit and baurate
        Parity = None,stop bit = 1
        */
        outport(SPBDV,0x1a);//   Baurate = 57600 bps
}
char TestIncomeData()
{       //check RDR bit
        if((inport(SPSTAT) & 0x0080 ) >0)
        {
                return(inport(SPRXD));
        }else
                return 0;
}
void SendChar(char sout)
{       WORD SPSTATRegister;
        //wait for ready to send
        do{
                SPSTATRegister = inport(SPSTAT);
                //check THRE bit
                SPSTATRegister = SPSTATRegister & 0x0040;
        }while (SPSTATRegister <= 0);
                outport(SPTXD,sout);
                //wait for sending complete
        do{
                SPSTATRegister = inport(SPSTAT);
                //check TEMT bit
                SPSTATRegister = SPSTATRegister & 0x0004;
        }while (SPSTATRegister <= 0);
}
```

```
void SendString(char *souts)
{
        int i;
        for(i = 0; i<strlen(souts);i++)
        {
                SendChar(souts[i]);
        }
}



void  main()
{       char ch = 0;
        InitSerial();
        SendString("\r\n Serial first program : press A to terminate");
        while(ch!=27)
        {
                ch = getche(); //wait for console input
                SendChar(ch); //send to modem
        }
    SendString("\r\nProgram terminated");
}
```

ตารางที่ 6-16 แสดงตัวอย่างโปรแกรมที่ทำการติดต่อกับพอร์อนุกรมอย่างง่าย

จากตารางที่ 6-5 เป็นโปรแกรมที่ทำการติดต่อกับอุปกรณ์ที่เชื่อมต่ออยู่กับชุดพัฒนาคอม86 ซึ่งในที่นี้ขอยกตัวอย่างโมเด็ม ซึ่งการทำงานของโปรแกรมก็คือ จะรอรับการป้อนค่าจากผู้ใช้ โดยผู้ใช้ป้อนค่าผ่านทางโปรแกรมเทอร์มินอล จากนั้นจะส่งตัวอักษรที่ผู้ใช้ป้อนเข้ามาไปให้กับโมเด็มทุกครั้งที่มีการกดคีย์บอร์ ถ้าหากผู้ใช้ต้องการให้โมเด็มสามารถทำงานต่างๆ ก็ทำโดยส่งคำสั่งเอทีคอมมานด์ไปให้กับโมเด็ม อาทิเช่น หากต้องการให้โมเด็มโทรออก ไปยังเบอร์ที่ต้องการก็กดคีย์บอร์ดเป็นคำสั่ง “ATDT [หมายเลขโทรศัพท์]” เป็นต้น และเมื่อผู้ใช้กดปุ่ม Esc โปรแกรมก็จะหยุดการทำงาน

6.4.2 การเขียนโปรแกรมเพื่อให้ชุดพัฒนาคอม86 สามารถเชื่อมต่อเข้าสู่เครือข่าย

การที่จะให้ชุดพัฒนาคอม86 สามารถเชื่อมต่อเข้าสู่ระบบเครือข่ายท้องถิ่น (Local Area Network) นั้น ในขั้นแรกจะต้องทำการต่อสายยูทีพีเข้ากับชุดพัฒนาคอม86 จากนั้นต่อสายซัพพลายเข้ากับชุดพัฒนาและทำการโหลดโปรแกรมแพ็กเกตไดร์เวอร์ โดยพิมพ์คำว่า "pnppd 0x60" ที่คอมมานไลน์ เมื่อสิ้นสุดขั้นตอนนี้ ชุดพัฒนาคอม86 จะสามารถเชื่อมต่อ เข้าสู่เครือข่ายได้ โดยสามารถทดสอบว่าชุดพัฒนาคอม86เชื่อมต่อเข้าสู่เครือข่ายได้ โดย โปรแกรม ping ที่อยู่ในไดเร็กทอรี a:/demo

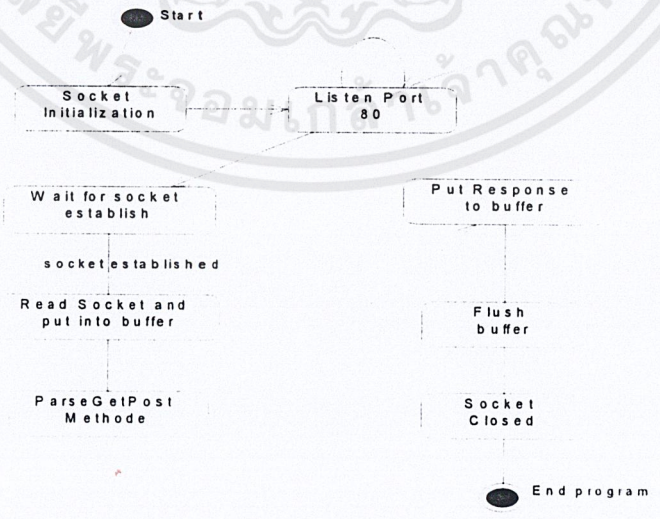
การใช้โปรแกรม ping มีรูปแบบของคำสั่งดังนี้ "ping [หมายเลขไอพีที่ต้องการทดสอบ]"ดังรูปที่6-16

```
A:\DEMO>ping 192.168.1.1
LXPING 2.5 - HP200LX TCP/IP Suite http://lxtcp.hplx.net/
Pinging [192.168.1.1]
sent PING # 1 , PING receipt # 1 : response time 0.00 seconds
Ping Statistics
Sent          : 1
Received      : 1
Success       : 100 %
Average RTT : 0.00 seconds
```

รูปที่6-16 แสดงตัวอย่างการทดสอบสถานะการเชื่อมต่อเข้าสู่เครือข่ายของชุดพัฒนาคอม86

สำหรับการเขียนโปรแกรมเพื่อติดต่อกับเครือข่ายเน็ตเวิร์กผ่านซ็อกเก็ต ด้วยภาษาซีจะใช้ไลบรารีชื่อว่า วัตทีซีพี(WATTCP) ซึ่งจะมาอำนวยความสะดวกในการเขียนโปรแกรมบนซ็อกเก็ตได้เป็นอย่างมาก โดยการหัวข้อนี้จะอธิบายถึงการนำไลบรารีนี้มาใช้กับอุปกรณ์เกตเวย์ และฟังก์ชันการทำงานบางส่วนของไลบรารี

ในส่วนของซอฟแวร์ที่มีการใช้วัตทีซีพีไลบรารีในตัวอุปกรณ์เกตเวย์นี้ เป็นโมดูลของการคอนฟิกูเรชันผ่านทางเวบบราวเซอร์ด้วยโปรโตคอลเอชทีทีพี ซึ่งขั้นตอนการทำงานคร่าวๆของโปรแกรมในโมดูลนี้แสดงดังรูป6-17



รูปที่ 6-17 แสดงขั้นตอนการทำงานของโปรแกรมส่วนของการคอนฟิกูเรชันผ่านทางเวบบราวเซอร์

ขั้นตอนการทำงานอย่างคร่าวๆของโปรแกรมที่รองรับการคอนฟิกูเรชันผ่านทางเวบบราวเซอร์ เริ่มจากการสร้างซ็อกเก็ต และเซ็ตค่าเริ่มต้นให้กับซ็อกเก็ตเหมือนกับการเขียนโปรแกรมที่ฝั่งเซิร์ฟเวอร์ทั่วๆไป โดยวัตทีซีไลบรารีมีฟังก์ชันสำหรับทำงานในส่วนนี้คือ void sock_init(void) จากนั้นเซิร์ฟเวอร์จะคอยฟังที่พอร์ต80 ว่ามีการร้องขอการเชื่อมต่อเข้ามาหรือไม่ โดยใช้ฟังก์ชัน word **tcp_listen(** void *tcpsocket, word listenport, longword remotelPaddr, word *remoteportnum, int (*signal_handler), word timeout) เมื่อมีการเชื่อมต่อเข้ามาที่พอร์ต 80 ก็จะทำการสร้างและติดตั้งช่องทางการเชื่อมต่อ ด้วยฟังก์ชัน *void* **sock_wait_established(** *void* * s, *word* seconds, *word (*optional_fn*)(),word* *optional_status_ptr); เมื่อสร้างช่องทางการเชื่อมต่อเสร็จเรียบร้อย ทางฝั่งเซิร์ฟเวอร์ก็จะอ่านข้อความที่ไคล์เอ็นท์ส่งมาเพื่อจะนำไปทำการแปลความหมายและปรับแก้ค่าตามที่ร้องขอ ฟังก์ชันที่ใช้ในการอ่านข้อมูลจากบัฟเฟอร์คือ *int* **sock_fastread(** *void* *s, *byte* *dp, *int* len) เมื่ออ่านข้อมูลเสร็จแล้วก็ส่งเข้าฟังก์ชันที่เขียนขึ้นเองเพื่อทำการแปลข้อความที่อ่านได้และนำไปประมวลผล(ในที่นี้คือฟังก์ชัน void ParseGetPost(buffer))

เมื่อทำการประมวลผลข้อความเรียบร้อยก็จะตอบรับกลับไปทางฝั่งไคล์เอ็นท์ด้วยการใส่ข้อความที่ต้องการลงไปในบัฟเฟอร์ด้วยคำสั่ง *int* **sock_puts(** *void* *s, *char* *text) แล้วทำการส่งออกทางซ็อกเก็ตด้วยฟังก์ชัน *void* **sock_flush(** *void* * s) แล้วรอการส่งข้อความร้องขอจากไคล์เอ็นท์ต่อไป ถ้ามีการปิดช่องทางการเชื่อมต่อจากทางฝั่งไคล์เอ็นท์ ทางฝั่งเซิร์ฟเวอร์ก็จะปิดการเชื่อมต่อนั้น เช่นกัน ตารางที่ 6-6 แสดงซอร์ดโค้ด อย่างคร่าวๆ ของการทำงานของโปรแกรมนี้

```
#include<tcp.h>
#include<string.h>
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

#define NEWLINE 0x0a          // Newline code //
#define SPACE  0x20           // Space code //
#define _MaxBuffer_ 600

void parseGETPOST(char * s) {
     .... //initial all local variable
       // start parse requested object
 while ((s[i]!=' ')&&(s[i]!='?')) {
  RequestedObject[a] = s[i] ;
  i++ ; a++ ;
 }
```

```
if (s[i]=='?') {
  // parse get method parameter
  i++ ;
  while (s[i]!=' ') {
   // parse Paramenter Name
   a = 0 ;
   while ((s[i]!='=')&&(s[i]!=' ')) {
         ParaName[a] = s[i] ;
         a++ ; i++ ;
   }
   // Parse Paramenter Value
   i++ ;
   a = 0 ;
   while ((s[i]!='&')&&(s[i]!=' ')) {
         ParaValue[a] = s[i] ;
         i++ ; a++ ;
   }
   if (s[i]=='&') {
    i++ ;
   }
  }
 } else { // no parameter
  // find POST Parameter
 while
(((s[i]!=0)&&(i<_MaxBuffer_))&&!(((s[i]=='\r')&&(s[i+1]=='\n')&&(s[i+2]=='\r')&&(s[i+3]=='\n')))) {
    i++ ;
  }
  if ((s[i]=='\r')&&(s[i+1]=='\n')&&(s[i+2]=='\r')&&(s[i+3]=='\n')) {
   i+=4 ;
   // parse POST Parameter
    while ((s[i]!=0)&&(s[i]!='\r')) {
         // parse Paramenter Name
         a = 0 ;
         while ((s[i]!='=')&&(s[i]!=0)&&(s[i]!='\r')) {
```

```
            ParaName[a] = s[i] ;
            a++ ; i++ ;
           }
           // Parse Paramenter Value
           i++ ;
           a = 0 ;
           while ((s[i]!='&')&&(s[i]!=0)&&(s[i]!='\r')) {
            ParaValue[a] = s[i] ;
            i++ ; a++ ;
           }
           if (s[i]=='&') {
            i++ ;
           }
//------------------------ RETURN PARAMETER ----------------------//
           printf("%s = %s : %d\n",ParaName, ParaValue, strlen(ParaValue)) ;
          ...//Formatting the parameter ready to use by each function
          configNAT(ip,sub1,sub2,sub3,sub4); //send these parameter to configuration function
         configGPRS(apn,tel);
}
void myhttp(void) {
          static tcp_Socket *s;
          clearBuffer() ;
          s = &telnetsock;
          tcp_listen( s, 80, 0L, 0, NULL, 0);
          sock_wait_established(s, 0, NULL, &status);
          sock_mode(s, TCP_MODE_BINARY) ;
          sock_wait_input(s, 0, NULL, &status) ;
          sock_fastread(s, buffer, 512) ;
          printf("%s", buffer) ;
          parseGETPOST(buffer) ;

          sock_puts(s, "HTTP/1.0 200 Ok\n\r") ;
          sock_puts(s, "Content-Type: text/html; charset=\"utf-8\"\n\r") ;
          sock_puts(s, "\n\r") ;
```

```
        sock_puts(s, "<h1>GPRS WIRELESS GATEWAY CONFIGURATION BY HTTP</h1>\n") ;
        sock_puts(s, "<FORM ACTION=\"A.html\" METHOD=\"POST\">") ;
        sock_puts(s, "TYPE AND APN          ") ;
        sock_puts(s, "<INPUT TYPE=\"TEXT\" SIZE = \"30\" NAME=\"apn\" /><Br>") ;
        sock_puts(s, "ISP TELEPHONE NUMBER   ") ;
        sock_puts(s, "<INPUT TYPE=\"TEXT\" SIZE = \"10\" NAME=\"tel\" /><Br>") ;
        sock_puts(s, "LOCAL IP ADDRESS      ") ;
        sock_puts(s, "<INPUT TYPE=\"TEXT\" SIZE = \"3\" NAME=\"ipa\" /> ") ;
        sock_puts(s, "<INPUT TYPE=\"TEXT\" SIZE = \"3\" NAME=\"ipb\" /><Br> ") ;
        sock_puts(s, "LOCAL SUBNET MASK      ") ;
        sock_puts(s, "<INPUT TYPE=\"TEXT\" SIZE = \"3\" NAME=\"suba\" /> ") ;
        sock_puts(s, "<INPUT TYPE=\"TEXT\" SIZE = \"3\" NAME=\"subb\" /><Br> ") ;
        sock_puts(s, "<INPUT TYPE=\"SUBMIT\" NAME=\"SUBMIT\" VALUE=\"SUBMIT\" />") ;
        sock_puts(s, "</FORM>") ;

        sock_flush(s) ;

        sock_close(s) ;
        sock_wait_closed(s, 0, NULL, &status) ;

        sock_err:
        switch (status) {
                case 1 :/*foreign host closed */
                        puts("User closed session");
                        return;
                case -1: /* timeout */
                        printf("\n\rConnection timed out!");
                        return;
                }
}
//------------------------------MAIN-----------------------
```

```
void main(void) {
        sock_init();
        do {
                myhttp() ;
        } while (!kbhit()) ;
}
```

ตารางที่ 6-16 แสดงตัวอย่างรองรับการปรับแก้ค่าผ่านทางเวบบราวเซอร์โปรแกรมอย่างคร่าวๆ

การคอมไพล์และสร้างไฟล์ที่สามารถรันได้ จะต้องทำการใส่ไลบรารีไฟล์เข้าไปในโปรเจ็คด้วย ไฟล์นั้นคือ WATTCPSM.LIB ในไดเร็กทอรีที่ชื่อ wattcp/lib โปรแกรมจึงจะสามารถใช้งานได้

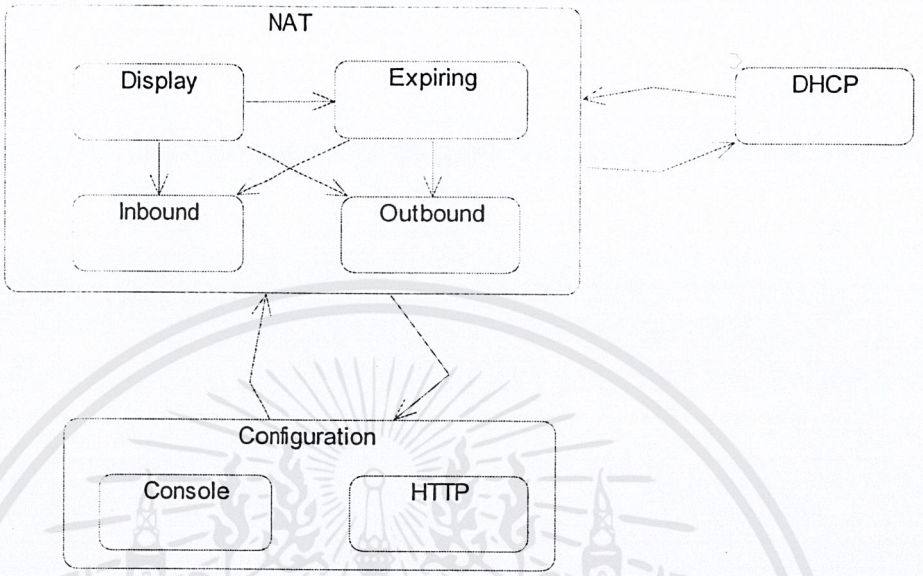
6.5 การเขียนโปรแกรมให้สามารถทำงานเป็นแบบมัลติทาสก์

ในการสร้างอุปกรณ์ประเภทที่ให้ความสะดวกในการเชื่อมต่อสื่อสารรับส่งข้อมูล จุดสำคัญที่ต้องพิจารณา คือประสิทธิภาพในการทำงาน อาทิเช่น ความรวดเร็วในการรับส่งข้อมูล ความถูกต้องของข้อมูล ความสามารถในการที่จะทำงานได้ตลอดเวลาที่ยังเปิดให้บริการอยู่ เป็นต้น ความรวดเร็วและความถูกต้องในการรับส่งข้อมูลนั้นสามารถเพิ่มหรือลดลงได้ตามความคับคั่งของช่องสัญญาณ หรือประสิทธิภาพของตัวกลางในการสื่อสาร แต่ความสามารถที่จะทำงานได้ตลอดเวลาที่เปิดให้บริการนั้นขึ้นอยู่กับความสามารถของอุปกรณ์ ด้วยเหตุนี้การเขียนโปรแกรมเพื่อให้สามารถทำงานหลักคือการให้บริการการรับส่งข้อมูลได้ตลอดเวลาแม้ว่าจะเกิดเหตุการภายนอกเข้ามาขัดขวางการทำงานของตัวอุปกรณ์ อาทิเช่นมีการเข้ามาขอทำการปรับแก้ค่าของอุปกรณ์ ทั้งทางพอร์ตอนุกรมและทางเวบบราวเซอร์

หลักการที่จะมาช่วยแก้ไขปัญหานี้ได้แก่การเขียนโปรแกรมให้สามารถทำงานหลายๆงานได้พร้อมกัน เรียกว่ามัลติทาสก์ ในระบบปฏิบัติการดอสมีข้อจำกัดคือไม่อนุญาตโปรแกรมหลายๆ โปรแกรมทำงานพร้อมๆกันต้องทำงานให้เสร็จทีละงานเป็นลำดับไป ดังนั้น งานในการจัดการให้แต่ละโปรแกรมหรือหลายๆงานทำไปพร้อมกันจึงเป็นของผู้สร้างโปรแกรม

ในปัจจุบันมีไลบรารีที่มาช่วยจัดการให้โปรแกรมหนึ่งๆ สามารถรันหลายๆฟังก์ชันสลับกันไปมาได้หลายตัวด้วยกัน อาทิเช่น uC/OS (อ่านว่า MicroC/OS) , Multi-C library เป็นต้น โดยในโครงงานนี้ผู้พัฒนาเลือกใช้ Multi-c library เนื่องจากเป็นไลบรารีที่ไม่ขึ้นกับชนิดของหน่วยประมวลผล ทำให้โปรแกรมที่พัฒนาบนคอมพิวเตอร์ส่วนบุคคลสามารถทำงานได้บนชุดพัฒนาคอม86 โดยที่ไม่ต้องแก้ไขซอร์ดโคดของไลบรารีให้ปรับไปตามหน่วยประมวลผล

สำหรับซอฟแวร์ที่พัฒนามาเพื่ออุปกรณ์เกตเวย์นี้มีโครงสร้างการแบ่งงานหรือทาสก์ออกเป็นหลายๆทาสก์ด้วยกัน ดังรูปที่ 6-18



รูปที่ 6- 18 แสดงแผนภาพของโครงสร้างการแบ่งทาสก์และความสัมพันธ์กันของแต่ละทาสก์

ซอฟแวร์ของอุปกรณ์เกตเวย์มีการแบ่งออกเป็น 3 ทาสก์ใหญ่ๆ คือส่วนที่ของการจัดการแปลและแปลงไอพีแอดเดรส ส่วนของการปรับแก้ค่าคุณสมบัติ และส่วนของดีเอชซีพี การทำงานของ 3 ทาสก์นี้ มีงานหลักคือส่วนของการจัดการแปลและแปลงไอพีแอดเดรส โดยเมื่อมีสัญญาณขอทำการปรับแก้ค่าคุณสมบัติ หรือ สัญญาณร้องขอไอพีแอดเดรสจากดีเอชซีพีไคล์เอ็นท์ ก็จะไปทำงานในส่วนนั้น ดังนี้

- การสลับกันทำงานระหว่างทาสก์จัดการแปลและแปลงไอพีและทาสก์คอนฟิกูเรชัน จะมีการสลับไปมาขึ้นกับการกระทำของผู้ใช้ คือ เมื่อมีการขอเข้ามาทำการปรับแก้ค่าในครั้งแรก โปรแกรมจะหยุดการทำงานของทาสก์ที่ทำอยู่มาทำงานในส่วนฟังก์ชันคอนฟิกูเรชัน และในขณะที่รอรับการป้อนคำสั่งจากผู้ใช้จะกลับไปทำงานในทาสก์เดิมที่ทำอยู่จนกระทั่งผู้ใช้กดปุ่ม Enter คือสั่งให้ทำงานตามคำสั่งจึงไปประมวลผล
- การสลับกันทำงานระหว่างทาสก์จัดการแปลและแปลงไอพีและทาสก์ดีเอชซีพี จะมีการสลับไปมาโดยเมื่อได้รับแพ็คเกตกระจายที่ร้องขอการแจกไอพีแอดเดรสจากเซิร์ฟเวอร์ โปรแกรมจะหยุดงานจัดการแปลและแปลงไอพีทั่วไปแล้วเข้ามารันในทาสก์ดีเอชซีพี เพื่อส่งแพ็กเกตนำเสนอดีเอชซีพี (dhcpoffer) จากนั้นในขณะที่รอแพ็กเกตยอมรับกลับมาจากเครื่องไคล์เอ็นท์จะกลับไปทำงานจัดการแพ็กเกตตามเดิมที่ทำอยู่

ในการเขียนโปรแกรมเพื่อทำการสลับการทำงานของแต่ละทาสก์ด้วยมัลติซีไลบรารีจะมีใช้หลักการการบังคับให้ทาสก์หนึ่งๆหยุดพักทำงานไปรันทาสก์อื่นที่ต่อคิวรออยู่ โดยใช้ฟังก์ชัน ดังต่อไปนี้

- Errorcode MtCCoroutine(ชื่อทาสก์) ใช้ในการสร้างทาสก์ใหม่ขึ้นมาในโปรแกรม

- Errorcode MtCYield() ใช้บังคับให้ทาสก์นั้นหยุดการทำงานแล้วไปรันงานอื่นที่ต่อ รอการรันอยู่
- Errorcode MtCStop(thread) ใช้ทำลายทาสก์ที่ระบุ

```
#include <stdio.h>
#include <string.h>
#include "c:\multi-c\mtc.h"

……..//declare global variable
void main()
{       for(;;)
        {
                if(bioskey(1))
                {       scan = bioskey(0);
                        console();
                }
                if(exitcode == 1)
                {       printf("exit from loop main\n");
                        break;
                }
        }
        printf("End main task");
}
void console()
{
        if(sendcommand == 1) //last command has not finished yet
        {
                MtCYield();

        }else{
                //last command is finish
                if(scan !=13) //get character into buffer
                {       command[i] = scan;
                        printf("%c",command[i]);
```

```
                        i++;
                        command[i] = '\0';
                    }else if(scan == 13){  //going to execute
                        //look up table
                        if(!strcmp(command,"aaa")) //if command = aaa
                        {       sendcommand =1;
                                MtCCoroutine(aaa());  //create aaa task
                        }else if(!strcmp(command,"bbb")){
                                sendcommand =1;
                                MtCCoroutine(bbb());
                        }else if(!strcmp(command,"ccc")){
                                sendcommand =1;
                                MtCCoroutine(ccc());
                        }else{ //if command is not match with any entry
                                printf("\nCannot recognized command:");
                                printf("%s",command);
                                printf("\n");
                                strcpy(command,"");
                                i=0;
                                clrscrn();
                        }
                    }else if(scan == 27){ //quit program
                                exitcode = 1;
                    }
                }
}
void aaa()
{               j=0;
        clrscrn();
        printf("aaa config\n");
        MtCStop(NULL);
}
void bbb()
{       clrscrn();
```

```
        printf("bbb config\n");
        printf("process this task depend on parameter \n");
        sendcommand =0;
        strcpy(command,"a");
        i =0;
        printf(">");
        MtCStop(NULL);

}
void ccc()
{
        while(1) //get user input
        {
                if(scan=! 13){
                        ccmd[k] = scan;
                        printf("%c",ccmd[k]);
                        k++;
                        ccmd[k] = '\0';
                }else if(scan == 13)
                {
                        printf("\ngoing to open file:%s\n",ccmd);
                        printf("Task complete\n");
                        k=0;
                  break;
                }else if(scan == 27){ //quit program
                        exitcode = 1;                   break;
                }
                MtCYield(); //if user not press any key ,switch to other task
        }
       //Execute command finish
        sendcommand = 0; //finised
        strcpy(command,"\0\0");
        strcpy(ccmd,"\0\0");
        i =0; k =0;
```

```
        printf(">");
        MtCStop(NULL);
}
```

ตารางที่ 6-17 แสดงตัวอย่างการใช้งานฟังก์ชันต่างๆของมัลติซีไลบรารี

โปรแกรมในตารางที่ 6-17 เป็นโปรแกรมจำลองการทำงานโดยสมมติว่าฟังก์ชัน main() เป็นส่วนที่ทำงานหลัก เมื่อมีการกดคีย์บอร์ดจากผู้ใช้เพื่อทำการใส่คำสั่งที่ต้องการคอนฟิกูเรชัน จะไปทำงานในฟังก์ชันที่ทำการรับค่าจากคีย์บอร์ดมาเก็บใส่บัฟเฟอร์ แล้วเมื่อมีการสั่งให้ทำคำสั่งก็จะสร้างทาสก์ของคำสั่งนั้นเพื่อทำงานต่อไป ซึ่งในขณะที่รอรับการกดคีย์บอร์ดจากผู้ใช้จะกลับมาทำงานในโปรแกรมหลักตลอดเวลา

นอกจากฟังก์ชันที่กล่าวมายังมีหลักการในการเขียนโปรแกรมมัลติทาสก์อื่นๆอาทิเช่น การใช้ซีมาฟอร์ (Semaphore) มาช่วยในการสลับการทำงาน โดยใช้ฟังก์ชันที่จัดการเกี่ยวกับซีมาฟอร์ ดังนี้

- MtCSemaCreate ใช้สร้างซีมาฟอร์
- MtCSemaDel ใช้ทำลายซีมาฟอร์
- MtCSemaGet ใช้เพื่อดูค่าของซีมาฟอร์ในขณะนั้น
- MtCSemaInverse ใช้เพื่อกลับค่าของซีมาฟอร์ให้เป็นตรงกันข้าม
- MtCSemaReset ใช้รีเซตค่าของซีมาฟอร์ให้เป็นฟอลซ์
- MtCSemaSet ใช้เซตค่าของซีมาฟอร์ให้เป็นทรู
- MtCSemaWait ใช้เพื่อหยุดการทำงานของทาสก์นั้นรอจนได้ค่าซีมาฟอร์ที่ต้องการจึงกลับมารันใหม่

บทที่ 7

การทดสอบและผลการทดลอง

7.1 การทดสอบในสภาพแวดล้อมต่างๆ

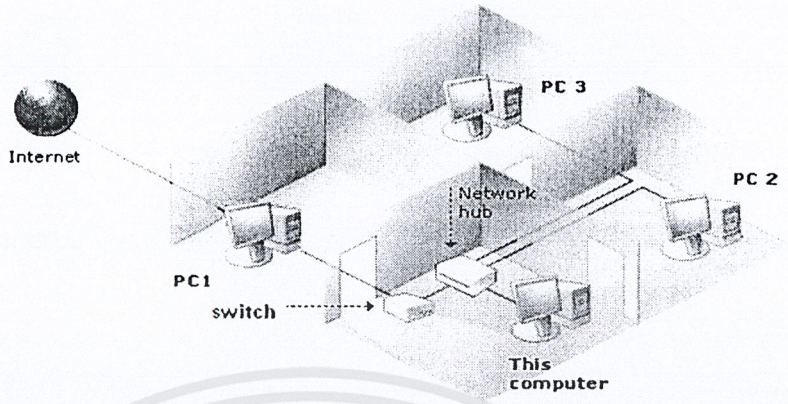
การทดสอบการทำงานของอุปกรณ์จะเน้นไปที่การทดสอบการทำงานของซอฟแวร์ โดยจะยังไม่ทดสอบที่สภาวะการทำงานจริงทันทีเนื่องจากข้อจำกัดทางด้านค่าใช้จ่าย และเวลาในการพัฒนา คือในสภาวะการทำงานจริงๆ นั้นอุปกรณ์จะต้องทำงานกับระบบเครือข่ายท้องถิ่นแบบไร้สายและเครือข่ายจีพีอาร์เอส ซึ่งการเชื่อมต่อกับเครือข่ายท้องถิ่นแบบไร้สายนั้น จำเป็นต้องมีอุปกรณ์แอกเซสพอยน์และการ์ดอินเตอร์เฟส ซึ่งมีราคาค่อนข้างสูงถ้าจะใช้เป็นจำนวนมาก ดังนั้นการทดสอบโปรแกรมจึงมีสภาพแวดล้อมต่างๆกันไปขึ้นกับความสะดวกในการจัดหาอุปกรณ์ และเป้าหมายของโปรแกรมแต่ละโปรแกรม สภาพแวดล้อมที่ทำการทดลองโปรแกรมมีดังนี้

- การทดลองโปรแกรมแปลงและแปลไอพีบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายท้องถิ่นอื่นๆที่มีการเชื่อมต่อเข้าสู่อินเตอร์เน็ต
- การทดลองโปรแกรมบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายอินเตอร์เน็ตผ่านทางโมเด็ม
- การทดลองโปรแกรมบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายจีพีอาร์เอสด้วยอุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอส
- การทดลองโปรแกรมบนชุดพัฒนาคอม86 โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายอินเตอร์เน็ตผ่านทางโมเด็ม
- การทดลองโปรแกรมบนชุดพัฒนาคอม86 โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายจีพีอาร์เอสด้วยอุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอส

จะเห็นว่าสภาพแวดล้อมในการทดสอบทั้งหมดจะใช้ระบบเครือข่ายท้องถิ่นแทนระบบเครือข่ายท้องถิ่นแบบไร้สาย เนื่องจากการทำงานของอุปกรณ์แอกเซสพอยน์สามารถที่จะทำงานร่วมกับเครือข่ายท้องถิ่นแบบมีสายได้ จึงไม่จำเป็นต้องเปลี่ยนหรือปรับแก้โปรแกรมเมื่อนำไปใช้งานจริงกับระบบเครือข่ายท้องถิ่นไร้สาย

7.1.1 สภาพแวดล้อมบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายท้องถิ่นอื่นๆที่มีการเชื่อมต่อเข้าสู่อินเตอร์เน็ต

ลักษณะของการเชื่อมต่อเครือข่ายในลักษณะนี้แสดงดังรูปที่7-1



รูปที่ 7-1 แสดงลักษณะการเชื่อมต่อคอมพิวเตอร์ที่ใช้ทดสอบโปรแกรมแปลและแปลงไอพีแอดเดรส

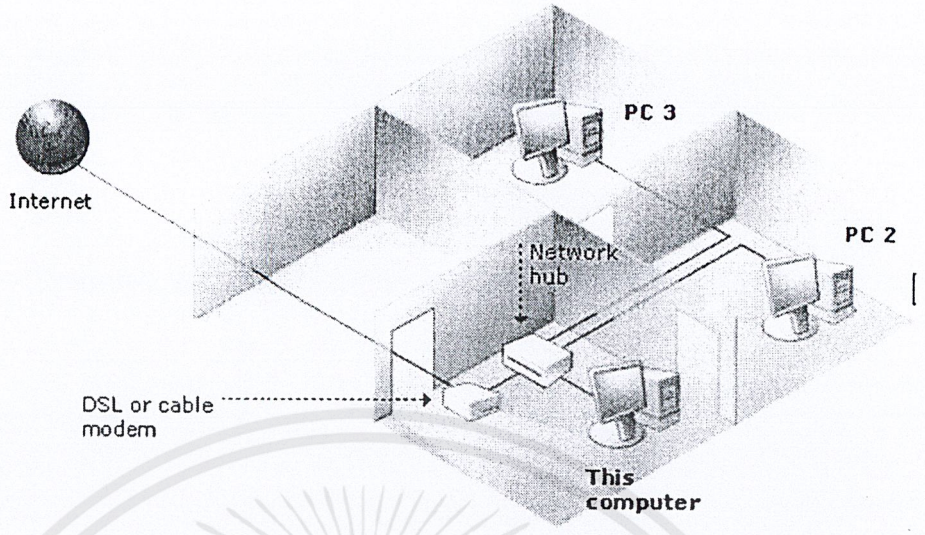
โปรแกรมจะมีส่วนการแสดงผลออกมาทางจอภาพเพื่อแสดงแพ็กเกตที่ผ่านเข้าออกเครือข่ายดังรูปที่ 7-2 โดยจะแสดงหมายเลขไอพีของเครื่องคอมพิวเตอร์ภายในซับเน็ต และหมายเลขปลายทางที่แพ็กเกตต้องการส่งไปรวมทั้งชนิดของแพ็กเกตเหล่านั้นและอายุของแพ็กเกต

Proto	Local IP:Port	Remote IP:Port	Mapping	Sent	Rcvd	Expire
TCP	10.64.1.1:247	129.210.120.40:21	5001	2	1	5:59:03
TCP	10.64.1.1:2305	198.34.2.35:80	5002	5	5	5:59:12
TCP	10.64.1.1:2306	198.34.2.35:80	5003	3	1	5:59:12
TCP	10.64.1.1:2307	198.34.2.35:80	5005	1	0	5:59:34
UDP	10.64.1.2:15443	207.189.4.16:6667	5006	2	2	0:01:25
ICMP	10.64.1.8:112	223.115.43.6:78	5143	1	1	0:00:43
TCP	10.64.1.2:2439	24.112.34.48:21	5147	45	65	6:00:00

รูปที่ 7-2 แสดงตัวอย่างของโปรแกรมที่ทำการแปลงไอพีแอดเดรส (Network Address Translation)

7.1.2 สภาพแวดล้อมบนเครื่องคอมพิวเตอร์ส่วนบุคคล โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นเครือข่ายอินเตอร์เน็ตผ่านทางโมเด็ม

ลักษณะการเชื่อมต่อของเครื่องคอมพิวเตอร์ที่รันโปรแกรมเป็นดังรูปที่ 7-3



รูปที่7-3 แสดงภาพการเชื่อมต่อเครือข่ายของเครื่องคอมพิวเตอร์ที่รันโปรแกรม

การทดสอบบนสภาพแวดล้อมนี้จะค่อนข้างซับซ้อนต่างจากแบบแรกเนื่องจากการที่จะใช้โมเด็มติดต่อไปยังผู้ให้บริการอินเตอร์เน็ต จะต้องทำตามพีพีพีโปรโตคอล ดังนั้นโปรแกรมจะต้องโมดูลของการเชื่อมต่อไปยังอินเตอร์เน็ตผ่านทางโมเด็ม ซึ่งการทำงานในขั้นนี้ได้ผลลัพธ์เหมือนกับแบบแรกแต่จะรันโปรแกรมได้ก็ต่อได้มีโหลดพีพีพีแพ็กเกตไดร์เวอร์ก่อน

7.1.3 สภาพแวดล้อมบนเครื่องคอมพิวเตอร์ส่วนบุคคล เชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายจีพีอาร์เอส

การทดสอบบนสภาพแวดล้อมนี้คล้ายกับการเชื่อมต่อโดยใช้โมเด็ม แต่ต่างกันที่โมดูลที่จัดการการเชื่อมต่อไปยังผู้ให้บริการ นั้นคือโมดูลพีพีพีโปรโตคอล ที่จะต้องทำการเปลี่ยนแปลงค่าในการเชื่อมต่อทางพอร์ตอนุกรมหรือที่เรียกว่า ยูอาร์ต ให้เหมาะสมกับการเชื่อมต่อ ผลลัพธ์การทดสอบคล้ายกับแบบแรก

7.1.4 การทดลองโปรแกรมบนชุดพัฒนาคอม86 โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายอินเตอร์เน็ตผ่านทางโมเด็ม

การทำการทดสอบโปรแกรมบนสภาพแวดล้อมที่ใช้เครื่องที่รันโปรแกรมเป็นชุดพัฒนาคอม86 แทนเครื่องคอมพิวเตอร์ส่วนบุคคล ต้องทำการแก้ไขโปรแกรมในส่วนที่ทำการแสดงผลของโปรแกรมแปลงไอพีแอดเดรสให้เป็นแบบเท็กซ์โหมดทั้งหมดและแก้ไขโปรแกรมส่วนของพีพีพีโปรโตคอลให้เข้ากับ ยูอาร์ตของชุดพัฒนาคอม86

7.1.5 การทดลองโปรแกรมบนชุดพัฒนาคอม86 โดยเชื่อมต่อระหว่างเครือข่ายท้องถิ่นและเครือข่ายจีพีอาร์เอส

การทดสอบในสภาพแวดล้อมแบบนี้ต้องทำการแก้ไขโปรแกรมในส่วนของพีพีพีให้สามารถเชื่อมต่อเข้าสู่เครือข่ายจีพีอาร์เอสได้ โดยแก้ที่ส่วนที่ควบคุมการส่งข้อมูลผ่านพอร์ตอนุกรมที่เรียกว่า ยูอาร์ต

การทดสอบว่าอุปกรณ์สามารถใช้งานได้หรือไม่ในขั้นแรกจะทำโดยการเซ็ตค่าไอพีแอดเดรสและค่าอื่นๆของเครื่องคอมพิวเตอร์ที่ต่ออยู่ภายในเครือข่าย(Static IP address)ดังนี้

หมายเลขไอพีแอดเดรส เป็นหมายเลขใดก็ได้ที่อยู่ภายในซับเน็ตเดียวกันกับอุปกรณ์

หมายเลขซับเน็ทมาสก์ ขึ้นกับจำนวนเครื่องไคล์เอ็นท์ที่อุปกรณ์สามารถรองรับได้ภายในเครือข่าย

หมายเลขไอพีแอดเดรสของเกตเวย์เป็นหมายเลขเดียวกันกับไอพีแอดเดรสของเกตเวย์เมื่อเซ็ตค่าเหล่านี้เสร็จแล้วลองใช้แอพพลิเคชันใดก็ได้ที่ต้องการเข้าใช้อินเตอร์เน็ต หากสามารถเข้าใช้งานได้แสดงว่าอุปกรณ์สามารถทำงานได้

ในขั้นต่อมาจะทดสอบการแจกไอพีของเซิร์ฟเวอร์ว่าทำงานได้เหรอไม่ด้วยการที่ไม่ต้องเซ็ตค่าไอพีแอดเดรสของเครื่องไคล์เอ็นท์ และเลือกให้ใช้ฟังก์ชันของการรับแจกไอพีโดยอัตโนมัติ (Obtain an IP address automatically) จากนั้นรีสตาร์ทเครื่องไคล์เอ็นท์และลองเข้าใช้แอพพลิเคชันที่ต้องเข้าใช้เครือข่ายอินเตอร์เน็ตหากสามารถใช้งานได้แสดงว่าอุปกรณ์สามารถทำงานได้ตามปกติ

7.2 ข้อกำหนดการใช้งานอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

เนื่องจากโครงงานนี้เป็นการพัฒนาซอฟแวร์บนชุดพัฒนาคอม86 ซึ่งมีข้อจำกัดในด้านทรัพยากรและด้านฮาร์ดแวร์ ทำให้ประสิทธิภาพการทำงานของอุปกรณ์ ไม่สูงเท่าการทำงานบนเครื่องคอมพิวเตอร์ส่วนบุคคล เช่นอัตราเร็วในการรับส่งข้อมูล ความสามารถในการรับรองจำนวนเครื่องไคล์เอ็นท์มีน้อยกว่า แต่ข้อดีของการใช้อุปกรณ์นี้คือ ความสามารถในการเคลื่อนย้ายอุปกรณ์ไปยังที่ใดก็ได้ และเป็นการสร้างผลงานที่เป็นระบบฝังตัว (Embedded System Device)อีกอย่างหนึ่งขึ้นมาในประเทศ

บทที่ 8
บทวิจารณ์และสรุป

ในปัจจุบันระบบฝังตัวได้เข้ามามีบทบาททางด้านอุตสาหกรรมเป็นอย่างมาก ซึ่งผลิตภัณฑ์ที่ใช้กันอยู่นั้นเป็นการนำเข้ามาจากต่างประเทศ จากการศึกษาและทำงานเกี่ยวกับระบบฝังตัว(Embedded Systems) ในแง่ของการพัฒนาและปรับปรุงในประเทศไทยนั้น พบว่ายังมีผู้ที่ศึกษาและพัฒนาระบบทางด้านนี้อยู่เป็นจำนวนน้อยมาก การพัฒนาอุปกรณ์สื่อสารที่ทันสมัยและใช้เทคโนโลยีใหม่ๆ เข้ามาเกี่ยวข้องบนระบบฝังตัว จึงทำได้ค่อนข้างยาก การค้นหาข้อมูลส่วนใหญ่จึงไม่ได้มาจากในตำรา แหล่งความรู้ส่วนใหญ่จะมาจากเวบไซต์ต่างๆ หรือบทความภาษาอังกฤษ ที่หาได้บนเครือข่ายอินเตอร์เน็ต

8.1 สรุปผลการดำเนินงาน

ผลการดำเนินงานของโครงการนี้ สามารถดำเนินไปได้ ด้วยดี ตามวัตถุประสงค์ที่ตั้งไว้ แต่อย่างไรก็ตามยังมีฟังก์ชันการทำงานบางส่วนของอุปกรณ์ ที่ยังไม่สามารถทำงานได้ดีเช่นในส่วนของการแจกไอพีแอดเดรสอัตโนมัติ นอกจากนี้ยังขาดการทดสอบในเรื่องของประสิทธิภาพเมื่อใช้งานจริง เนื่องจากการที่จะทำการทดสอบอุปกรณ์ทุกครั้ง ต้องใช้อุปกรณ์มาเกี่ยวข้องเยอะและใช้ค่าใช้จ่ายสูง การวัดผลการทำงานของอุปกรณ์จึงวัดเพียงแค่ สามารถทำงานได้หรือไม่ โดยที่ไม่ได้สนใจว่าทำงานได้ดีหรือไม่เพียงใด

เนื่องจากการโครงงานนี้เป็นโครงการพัฒนาอุปกรณ์ที่ใช้เทคโนโลยีหลายชนิดมารวมกัน ทำให้ในการขั้นตอนของการพัฒนานั้นจะมีงานหลักๆ คือการพิจารณาเลือกใช้เครื่องมือและเทคโนโลยีที่เหมาะสมกับโครงงานมากที่สุด เพื่อให้โครงงานบรรลุวัตถุประสงค์ อาทิเช่น การเลือกใช้ไลบรารีที่มาช่วยในการเขียนโปรแกรมที่ทำงานเป็นมัสติทาสก์ จะมีการนำเอาข้อดีข้อเสียของแต่ละไลบรารีมาเปรียบเทียบกัน และเลือกใช้ หรือการออกแบบซอฟแวร์ว่าจะให้มีการทำงานตามระดับชั้นโปรโตคอล(Protocol Layer) ในส่วนไหนบ้าง หรือส่วนใดสามารถนำสิ่งที่มีอยู่แล้วมาใช้เองบ้าง นอกจากนี้การเขียนโปรแกรมยังต้องคำนึงถึงรูปแบบที่เป็นมาตรฐานในการสื่อสารเพื่อให้สามารถนำไปพัฒนาต่อได้ หรือให้ผู้ที่สนใจนำเอาโมดูลที่เขียนขึ้นไปใช้ต่อได้

8.2 แนวทางในการนำไปใช้งาน

อุปกรณ์ที่พัฒนาขึ้นในโครงงานนี้ เป็นอุปกรณ์ช่วยในการสื่อสารที่ทันสมัย ซึ่งจากการทดลองใช้ในห้องปฏิบัติการ ทำให้มองเห็นแนวทางในการนำเอาอุปกรณ์ไปใช้ให้เกิดประโยชน์ได้จริง ยกตัวอย่างดังนี้

- นำไปใช้กับงานที่ต้องการความคล่องตัวในการสื่อสาร หรือมีการเคลื่อนย้ายไปในที่ที่ไม่สามารถลากสายเคเบิลไปได้ถึง อาทิเช่น งานก่อสร้างทาง หรือ ธรุกิจที่ต้องการความสะดวกในการเชื่อมต่อการสื่อสารรับส่งข้อมูล เป็นต้น
- นำไปใช้ในงานอุตสาหกรรมที่โดยนำไปเป็นตัวเชื่อมต่อการควบคุมอื่นๆ เช่น การควบคุมอุปกรณ์ปลายทางผ่านทางเครือข่ายอินเตอร์เน็ต

8.3 แนวทางในการพัฒนาต่อ

แนวทางในการนำเอาอุปกรณ์นี้ไปพัฒนาต่อมีได้หลายแนวทาง เนื่องจากเป็นอุปกรณ์ที่ออกแบบมาเพื่ออำนวยความสะดวกในการสื่อสารหลายรูปแบบ ตัวอย่างการนำเอาอุปกรณ์ไปพัฒนาต่อ ได้แก่

- การพัฒนาเพื่อเพิ่มความสามารถของอุปกรณ์ อาทิเช่น เพิ่มฟังก์ชันการค้นหาเส้นทางเพื่อแลกเปลี่ยนข้อมูลของแต่ละเครือข่ายท้องถิ่นแบบไร้สาย โดยที่ยังไม่ต้องส่งผ่านข้อมูลไปยังเครือข่ายอินเตอร์เน็ต (แลกเปลี่ยนกันภายในเครือข่ายจีพีอาร์เอส)
- การพัฒนาให้อุปกรณ์ไร้สาย สามารถสร้างเครือข่ายชั่วคราวระหว่างกัน (Ad hoc network) เป็นลักษณะคล้ายกันกับเครือข่ายระบบท้องถิ่นแบบไร้สาย
- การพัฒนาทางด้านไอพีแอดเดรส เช่น โมบายไอพีแอดเดรส (Mobile IP) หรือ IPv6 เป็นต้น
- นำไปพัฒนาให้มีความสามารถในการควบคุมอุปกรณ์ปลายทางเพื่อใช้ในโรงงานอุตสาหกรรม ถือเป็นการพัฒนาทางด้านระบบฝังตัวในประเทศไทยอีกทางหนึ่ง

ภาคผนวก

ภาคผนวก ก.
วิธีการติดตั้งและใช้งานอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

1. วิธีการติดตั้งอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

ขั้นตอนการติดตั้งอุปกรณ์เกตเวย์มีดังนี้

- เชื่อมต่อชุดพัฒนาคอม86 เข้ากับอุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอส ผ่านทางพอร์ตอนุกรมที่2 (COM2)
- เชื่อมต่อสาย ยูทีพี เข้ากับชุดพัฒนา และอีกปลายด้านหนึ่งของสายเชื่อมต่อกับฮับหรืออุปกรณ์แอกเซสพอยน์
- ถ้าต้องการดูค่าสถานะ การทำงานของอุปกรณ์ให้เชื่อมต่อสายอนุกรมเข้าที่พอร์ตอนุกรมที่1 (COM1) เข้ากับพอร์ตอนุกรมของเครื่องคอมพิวเตอร์บุคคลและดูผ่านทางโปรแกรมเทอร์มินอลบนเครื่องคอมพิวเตอร์ตัวนั้น
- ต่อปลั๊กของหม้อแปลงไฟฟ้าที่ให้มาด้วยเข้ากับบอร์ดคอม86 และต่อหม้อแปลงเข้ากับแหล่งจ่ายไฟของอาคาร
- จากนั้นอุปกรณ์เริ่มรันไฟล์ต่างๆ ตามลำดับโดยอัตโนมัติ ดังนี้
  - (1) pnppd ซึ่งจะทำการโหลดอีเทอร์เน็ตแพ็กเกตไดร์เวอร์
  - (2) Dialer ซึ่งจะทำการสั่งให้อุปกรณ์เชื่อมต่อเครือข่ายจีพีอาร์เอสเชื่อมต่อไปยังเครือข่าย
  - (3) รันโปรแกรมหลักของอุปกรณ์
- หากผู้ดูแลระบบต้องการปรับแก้ค่าคุณสมบัติต่างๆของอุปกรณ์ ทำได้โดยผ่านทางเวป หรือทางเทอร์มินอล หากทำการเชื่อมต่อผ่านเทอร์มินอลจะมีคอมมานไลน์ให้สำหรับใสคำสั่งต่างๆ ในการแก้ไข
- สำหรับเครื่องไคล์เอ็นท์ที่ต้องการเข้าใช้อุปกรณ์ ทำได้โดยการเซ็ตไอพีแอดเดรส ให้รับการแจกจากเครื่องเซิร์ฟเวอร์ (Obtain an IP address automatically) ซึ่งในบางอุปกรณ์อาทิเช่นโทรศัพท์มือถือที่สามารถใช้บริการอินเตอร์เน็ตได้จะมีฟังก์ชันนั้นให้เองอัตโนมัติ
- หากระบบเครือข่ายท้องถิ่นที่ใช้เป็นแบบไร้สาย เครื่องไคล์เอ็นท์ต้องติดตั้งการ์ดอินเตอร์เฟสสำหรับระบบเครือข่ายท้องถิ่นแบบไร้สายด้วย แต่ถ้าหากเป็นระบบเครือข่ายอีเทอร์เน็ตทั่วไป เครื่องไคล์เอ็นท์ต้องติดตั้งเน็ตเวิร์กการ์ด (LAN Card) และต่อสาย UTP เข้ากับฮับหรือสวิตซ์

2. วิธีการกำหนดค่าคุณสมบัติอุปกรณ์เกตเวย์แบบไร้สายด้วยเครือข่ายจีพีอาร์เอส

ค่าที่ต้องกำหนดคุณสมบัติมีด้วยกัน 3 ประเภทคือ

1. ค่าในไฟล์ chatscr.gpr เป็นไฟล์คุณสมบัติของการร้องขอบริการจีพีอาร์เอส ประกอบด้วย 2 ฟิลด์คือ
   - ฟิลด์เบอร์โทรศัพท์ผู้ให้บริการจีพีอาร์เอส เช่น *99***1#
   - ฟิลด์ เอพีเอ็น (APN : Access Point Name) เช่น www.dtac.co.th

2. ค่าในไฟล์ nat.ini เป็นคุณสมบัติของการใช้ เอ็นเอที (NAT) ประกอบด้วย 2 ฟิลด์คือ
   - ฟิลด์ไอพีแอดเดรสท้องถิ่น เช่น 161.246.5.25
   - ฟิลด์ซับเนต มาสก์ เช่น 255.255.255.0

3. ค่าในไฟล์ dhcp.gpr เป็นไฟล์คุณสมบัติของการเป็น ดีเอชซีพีเซิร์ฟเวอร์ ของอุปกรณ์เกตเวย์ ประกอบด้วย 2 ฟิลด์คือ
   - ฟิลด์ไอพีแอดเดรสเริ่มต้น เช่น 161.246.5.0
   - ฟิลด์ไอพีแอดเดรสสุดท้าย เช่น 161.246.5.254

2.1 วิธีการกำหนดคุณสมบัติผ่านทางคอนโซลพอร์ต

มีขั้นตอนกำหนดคุณสมบัติดังต่อไปนี้

1. เริ่มต้นที่ ดอส พรอม พิมพ์คำสั่ง config.exe จะได้ผลลัพธ์ดังนี้

Please select :	1.	Read file
	2	GPRS Configuration
	3	NAT Configuration
	4	DHCP Configuration
	5	Exit

2. เลือกหมายเลขที่ต้องการคือ

เลือก 1 เมื่อต้องการอ่านไฟล์	ดูข้อ 3
เลือก 2 เมื่อต้องการกำหนดค่าคุณสมบัติไฟล์ chatscr.gpr	ดูข้อ 4
เลือก 3 เมื่อต้องการกำหนดค่าคุณสมบัติไฟล์ nat.ini	ดูข้อ 5
เลือก 4 เมื่อต้องการกำหนดค่าคุณสมบัติไฟล์ dhcp.gpr	ดูข้อ 6
เลือก 5 เมื่อต้องการ ออกจากโปรแกรม	ดูข้อ 7

3. ถ้าเลือก 1 ให้ใส่ชื่อไฟล์ที่ต้องการอ่าน จะปรากฏรายละเอียดในไฟล์นั้น เช่น

Please insert file name : chatscr.gpr

จะปรากฏรายละเอียดดังนี้

ABORT ERROR ABORT BUSY ABORT 'NO DIALTONE'

ABORT 'NO CARRIER' ABORT RING

REPORT CONNECT

TIMEOUT 10

" ATZ

OK AT&F

OK AT+CGDCONT=2,"IP","www.dtac.co.th"

OK ATDT*99***2#

TIMEOUT 60

CONNECT

4. ถ้าเลือก 2 ให้ใส่คุณสมบัติ เป็น เอพีเอ็น และ หมายเลขโทรศัพท์ผู้ให้บริการ ตามลำดับ เช่น

APN : www.dtac.co.th

ISP Number Phone : *99***1#

5. ถ้าเลือก 3 ให้ใส่คุณสมบัติ เป็นไอพีแอดเดรส และซับเนตมาสก์ ที่ใช้ทำ เอ็นเอที ตามลำดับ เช่น

Local IP address : 161.246.5.25

Local subnet mask : 255.255.255.0

6. ถ้าเลือก 4 ให้ใส่คุณสมบัติ เป็น ไอพีแอดเดรสเริ่มต้น และ ไอพีแอดเดรสสุดท้ายของดีเอชซีพี ตามลำดับ เช่น

IP address from : 161.246.5.0

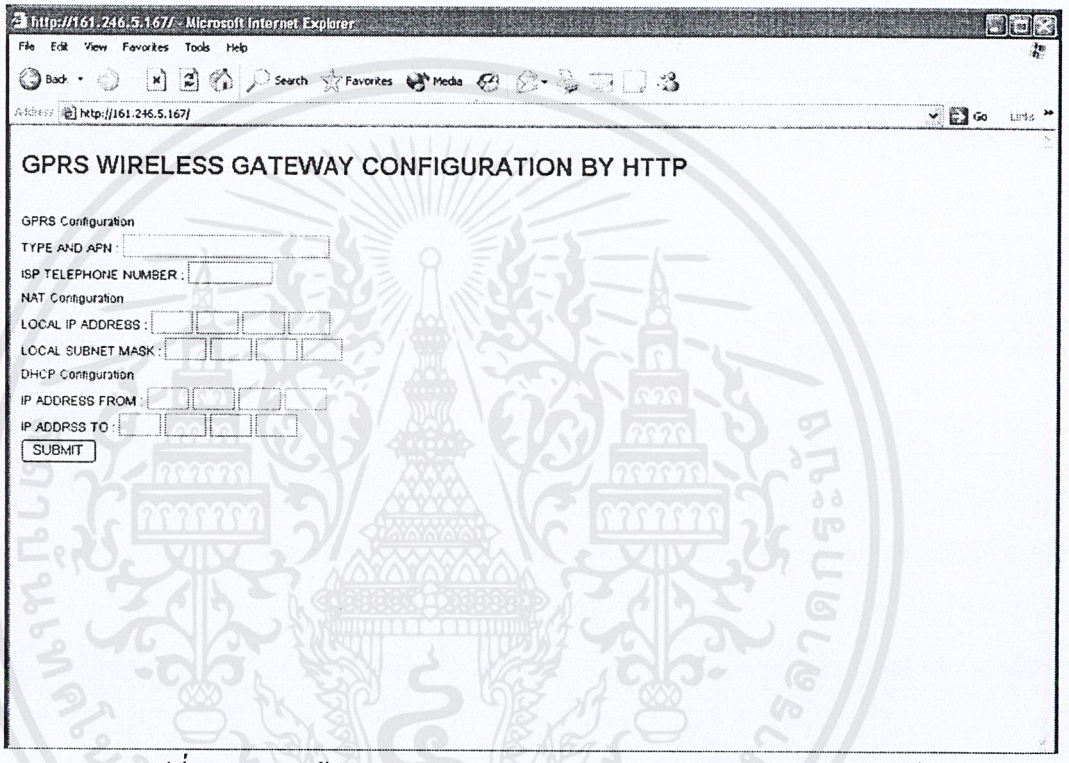
IP address to : 161.246.5.254

7. ถ้าเลือก 5 จะเป็นการออกจากโปรแกรมเข้าสู่ ดอส พรอม เพื่อที่รอรับคำสั่งต่อไป

2.2 ผ่านทางเวบบราวเซอร์

มีขั้นตอนกำหนดคุณสมบัติดังต่อไปนี้

1. เริ่มต้นที่อุปกรณ์เกตเวย์ รันไฟล์ pnppd ต่อจากนั้นก็รันโปรแกรม myweb.exe เพื่อรอคอนเน็กชันจากเวบบราวเซอร์ผ่านโปรโตคอล เอชทีทีพี ซึ่งเป็นการกำหนดค่าคุณสมบัติผ่านทางเวบบราวเซอร์
2. เมื่อต้องการกำหนดค่าคุณสมบัติให้เปิดเวบบราวเซอร์แล้วใส่แอดเดรสด้วย หมายเลขไอพีแอดเดรสของอุปกรณ์เกตเวย์ จะปรากฏหน้าเพจดังรูปที่ ก-1



รูปที่ ก-1 แสดงหน้าเวบเพจสำหรับการกำหนดค่าคุณสมบัติผ่านเวบบราวเซอร์

2. กรอกฟิลด์ต่างๆโดยมีความหมายดังนี้

TYPE AND APN	:	เอพีเอ็น
ISP TELEPHONE NUMBER	:	หมายเลขโทรศัพท์ผู้ให้บริการ
LOCAL IP ADDRESS	:	ไอพีแอดเดรส ที่ใช้ทำ เอ็นเอที
LOCAL SUBNET MASK	:	ซับเนตมาสก์ ที่ใช้ทำ เอ็นเอที
IP ADDRESS FROM	:	ไอพีแอดเดรสเริ่มต้น ที่ใช้ทำ ดีเอชซีพี
IP ADDRESS TO	:	ไอพีแอดเดรสสุดท้าย ที่ใช้ทำ ดีเอชซีพี

3. เมื่อกรอกฟิลด์ต่างๆเรียบร้อยแล้ว ให้กด submit เพื่อยืนยันการกำหนดค่าคุณสมบัติต่างๆ โดยจะมีซีจีไอ (CGI) ไปเรียกโปรแกรม สำหรับเขียนไฟล์อีกที เพื่อทำการเขียนไฟล์ต่างๆดังนี้ APN และ ISP Tel No. จะเขียนลง ไฟล์ chatscr.gpr และ Local IP address กับ Local subnet mask จะเขียนลงไฟล์ nat.ini ส่วน IP address from กับ IP address to จะเขียนลงไฟล์ dhcp.gpr จากนั้นก็จะกลับมาที่หน้าเพจเดิม เป็นการบอกว่าการ กำหนดค่าคุณสมบัติเรียบร้อยแล้ว

ภาคผนวก ข.

Waterloo TCP Programming Guide

ในส่วนนี้จะกล่าวถึงการใช้งานวัตทีซีไลบรารีในการเขียนโปรแกรมเพื่อให้สามารถเชื่อมต่อเครือข่ายได้โดยรายละเอียดของส่วนนี้จะเน้นไปที่ฟังก์ชันต่างๆที่ไลบรารีมีให้เรียกใช้และการทำงานของฟังก์ชันเหล่านี้ แต่จะไม่ได้ยกตัวอย่างซอร์ทโค้ดหรือวิธีการเขียนโปรแกรม

1. Important Concepts

1.1Sockets

Waterloo TCP uses what I call sockets. Enough other people call other things sockets that I don't mind adding yet another confusing definition. For the rest of this manual, what I call sockets will apply to my sockets, and not necessarily to any other programming environment. A socket may be defined to be of type *tcp_Socket* or *udp_Socket*. You will have to use one of the functions **tcp_Open**, **udp_Open** or **tcp_Listen** to open the socket, but subsequently any routine of type **sock...** will work. Note that I have made many functions non-blocking, this means you can do things between functions. That provides power but may sacrifice simplicity, so I have added blocking macros which make everything look a little more similar to UN*X or other systems. There are also blocking functions available for you to use. See **sock_...** functions for more details. An optional *signal_handler* may be passed during the open or listen functions. Please use NULL until such time as this service is properly announced.

1.2 ASCII vs. BINARY

'C' programmers are used to ASCII and binary oriented file i/o. To simplify some situations, I have added a mode flag to the socket structure and several functions act upon it. If you intend to use ASCII oriented instructions, call **sock_mode** with the socket in question and the flag **TCP_MODE_ASCII**. You can return to binary mode at any time by calling **sock_mode** again with the flag **TCP_MODE_BINARY**. *Sockets always open in binary mode, so you must explicitly indicate to use ASCII mode.*

void **sock_mode(** *void* ***s,** *word* **mode);**

eg. **if (!tcp_open(s, 0, host, port, NULL)) return(0);**

sock_mode(s, TCP_MODE_ASCII);

sock_wait_established(s, sock_delay, NULL, &status);

The mode flag affects **sock_dataready** and **sock_wait_input** because they normally respond as soon as any data is available. In ASCII mode, these functions act as if no data is available until a complete string is ready to be transferred. In ASCII mode, the operation of **sock_puts** changes slightly as it appends a '\n' to the sent data, normally no '\n' is added. Remember, if you intend to use **sock_gets** or **sock_scanf**, you *must* set ASCII mode. No other functions are affected.

1.3 Kernel Initialization

The kernel must be initialized before it is used, likewise it must be shut down before your application exits. Failure to shut down, for example, would cause you machine to hang the next time a packet arrived for your application. The easiest mechanism is to use **sock_init()**. It reads the configuration file, installs a control break handler, sets up the automatic shutdown facility using **atexit()**, checks the packet driver, loads the Ethernet address for ARP stuff, and clears tables. Note that **sock_init** removes the need for you to call a routine before you exit *if* you use control break, **exit()**, or abort, retry, ignore. *(the abort, retry, ignore description is not correct at the date of publishing. Such an exit leave the system in an unknown state) void* **sock_init(** *void* **);** For details on extending the configuration file, see the reference section under sock_init and usr_init.

1.4 Socket Functions

Opening Sockets

*Note, you cannot open a socket until after you have called **sock_init()**.* Sockets are formed using local and possibly remote IP addresses and sessions. All references hereafter will use:

lport - local port number, must be defined for **tcp_listen**

port - remote port number, must be defined for **tcp_open**

ina - remote host internet address, necessary for **tcp_open**

In the *tcp* realm of thinking, there is usually one host who listens for a caller and one who calls although *tcp* inherently supports two mutual callers. The listener must provide a local port on which he will be waiting and may or may not define the remote ip and/or remote port. Whenever the remote port or ina is not defined (i.e. 0), a

matchall value is used and incoming sessions are not screened by those parameters. Remember, the completion of all necessary parameters does not constitute a session, the call must still be received. For

udp, the remote port and/or internet address may be defined. If the remote host is zero, a received packet will complete the socket, if the remote host is non-zero, the socket will transmit to the defined node or network upon the next sock_ writing function. **udp_open** uses ARP to find the correct network address.

tcp_open actively attempts to connect to the other machine and uses ARP to find the machine address.

tcp_open and **udp_open**, if called with an lport of 0, will dynamically assign a generic port value.

tcp and *udp* sessions are unrelated, separate sockets exist for each of the two even if the local port numbers are identical.

All open functions return zero on error such as being unable to resolve the other machine's hardware address. They also accept *signal_handler* as described under the topic "sockets". tcp_Listen accepts a timeout value after which inbound socket attempts will fail.

word **tcp_open(** *void* *s, *word* lport, *longword* ina, *word* *port, *int (*signal_handler*)*);

word **tcp_listen(** *void* *s, *word* lport, *longword* ina, *word* *port, *int (*signal_handler*), word* **timeout);**

word **udp_open(** *void* *s, *word* lport, *longword* ina, *word* *port, *int (*signal_handler*)*);

After a session open call (**tcp_open** or **tcp_listen**), you must wait until the connection is established. There is a minimum of three packet transmissions, so the connection is hardly immediate. You may either poll the connection (**sock_established**) or have the library perform the waiting (**sock_wait_established**) and also the error handling if a timeout or reset occurs, see sock_... functions for more details on **sock_wait_established.**

word **sock_established(** *void* ***s);**

void **sock_wait_established(** *void* * s, *word* seconds, *word (*optional_fn*)(),word* *optional_status_ptr**);**

1.5 Closing Sockets

Once a socket has been opened, you will wish to read and write data. Upon completion of your data exchange you should use **sock_close** to close the connection. For tcp sockets, you must either use **sock_wait_closed** or poll the connection as described under **sock_wait_closed**. Ctrl brk or **exit()** will also close connections. *I do not think that abort() keep() or _exit() will close connections.*

void **sock_close(** *void* ***s);**

Waterloo TCP does not use keep-alives. They are highly discouraged by the internet community.A socket is assumed alive until closed, remotely reset, the program interrupted, or a user determined timeout occurs.

1.6 Reading Data

All reading functions return -1 when an error is encountered although I have not repeated it each time below. You should use signed len quantities. See

sock_wait_input() *for error handling.*

The two basic socket read functions are **sock_read** and **sock_fastread**. They both take the same parameters, but **sock_read** waits until the user buffer is full while **sock_fastread** returns immediately. **sock_dataready** returns the number of bytes ready to be read or -1 on error. It may be polled while waiting for data, and in ASCII mode (as set by sock_mode), it waits until a complete line is in the *tcp* buffer.

int **sock_read(** *void* *s, *byte* *dp, *int* len **);**

int **sock_fastread(** *void* *s, *byte* *dp, *int* len **);**

sock_gets reads a string and removes the lf or cr. The length of the new string is returned. If the new string is len or longer, the string is NULL terminated at len bytes and the remainder is discarded. If you intend to use **sock_gets**, you must first set ASCII mode using **sock_mode**. See the section on ASCII vs. Binary.

int **sock_gets(** *void* *s, *char* *text, *int* len **);**

sock_getc reads the next char if there is one, returns -1 on error, or 0 if there were no characters waiting. The limitation is that **sock_getc** cannot be used to read \0 or \1 unless you first check for data presence and connection stability which can be done with **sock_dataready()**:

```
if (sock_dataready( &socket )) {
        ch = sock_getc( &socket ); /* know connection stable and data present */
        printf("Got %u \n\r", ch );
}
```

char **sock_getc(** *void* *s **);**

sock_scanf busy waits for data and returns -1 on error or the number of fields completed. This function first performs an intrinsic **sock_gets**, so data must arrive on a single line. See the notes under the ASCII vs. Binary section.

int **sock_scanf(** *void* *s, *char* *format, ... **);**

To handle some tricky situations, sock_PreRead works exactly like sock_FastRead except it does not clear the data from the buffers. This technique may be used to prevent double buffering of received data.

int **sock_preread(** *void* *s, *byte* *dp, *int* len **);**

1.7 Writing Data

Writing to a socket places the bytes on the queue, but they may not necessarily be transmitted immediately (unless udp_Open was used), or the data may be lost in the transmission, or the other system may not immediately pass the data on to the user (unless udp_Open was used). Most of the write functions do not transmit until a certain number of bytes are ready to send, or until a certain period of time has passed. Those values are either system constants, or determined using algorithms dependant upon the particular connection. All write functions except **sock_fastwrite** block until sufficient buffer space exists for the kernel to hold the remainder of the data. **sock_fastwrite** simply returns the number of bytes it was able to write and the user must
loop through until all the data is written. The basic socket writing function is **sock_write**. I have provided two additional functions which flush data buffers and instruct the remote computer to pass the data to the user. **sock_flush** ensures the data already written is transferred, and **sock_flushnext** affects the next data to be written. **sock_flushnext** is preferred because it is much more efficient.

int **sock_write(** *void* * s, *byte* *data, *int* len **);**

int **sock_fastwrite(** *void* * s, *byte* *data, *int* len **);**

void **sock_flush(** *void* * s **);**

void **sock_flushnext(** *void* * s **);**

sock_puts will put a null terminated string. It performs an intrinsic flush (sock_flushnext). If in binary (default) mode, the string is not changed, but if ASCII mode has been set with **sock_mode**, the string and a subsequent \n are written. In either case, length of the original string is returned.*int* **sock_puts(** *void* *s, *char* *text **);** **sock_putc** places a character on the queue. If the character is a CR or LF, the stream is intrinsically flushed. There is no character expansion performed. The character written is returned.

int **sock_putc(** *void* *s, *char* ch **);**

sock_printf writes the formatted string using **sprintf** and **sock_puts** and returns length.

sock_printf(*void* *s, *char* *format, ... **);**

2. Buffer Information

While the input and output buffers are not truly available to you, there are several routines which allow you to inspect the size of the buffers (sock_tbsize, sock_rbsize), the number of bytes currently used in the buffer (sock_tbused, sock_rbused), and the number of bytes not used (sock_tbleft, sock_rbleft). All values are < 32767 and in bytes. **sock_t...** functions refer to transmit buffers, while **sock_r...** functions refer to receive buffers. *udp* sockets will return 0 for all transmit buffer functions since they do not maintain a transmit buffer. Closed sockets should return 0 for all such functions.

int **sock_tbsize(** *void* ***socket);**

int **sock_tbused(** *void* ***socket);**

int **sock_tbleft(** *void* ***socket);**

int **sock_rbsize(** *void* ***socket);**

int **sock_rbused(** *void* ***socket);**

int **sock_rbleft(** *void* ***socket);**

3. Kernel Waiting

User applications may do polling to wait for data, but unless you are expecting real-time data (keystrokes) or handling multiple sessions, you will want to have simple mechanisms to do your busywaiting.

I have already written a few macros/functions to perform this task for most simple situations, they are macros called **sock_wait_...** These macros have several things in common, they accept a timeout value for the current operation and they routinely give the kernel a time slice, do a user defined yield if **sock_yield** was used, check the keyboard for ^C (remember we safely close sessions), and call an optional short user defined function repeatedly. They also accept a status pointer and handle errors so your input and output routines do not have to.

sock_wait_established is used after an open to wait until the connection is complete. For *udp* connections it returns immediately. **sock_wait_input** waits until some input arrives. **sock_wait_closed** is

used after a **sock_close** operations, but is unnecessary after a **sock_abort**. **sock_tick** just performs a kernel slice, but replicates the error handling of the other **sock_**functions.

sock_wait_established(*void* * s, *word* seconds, *word (*optional_fn*)*, *word* *optional_status_ptr **);**

sock_wait_input(*void* * s, *word* seconds, *word (*optional_fn*)*, *word* *optional_status_ptr **);**

sock_wait_closed(*void* * s, *word* seconds, *word (*optional_fn*)*, *word* *optional_status_ptr **);**

sock_tick(*void* * s, *word* seconds, *word (*optional_fn*)*, *word* *optional_status_ptr **);**

Although not considered pretty, (what systems programming is?) the sock_... functions branch on an error by setting *optional_status_ptr if it is non-NULL and doing a **goto sock_err;** The result is much simpler user code. Performing the same with **breaks** and **returns** creates much larger modules. Using upcalls looses context sensitivity. Obviously the other two mentioned options exist, they are simply computer science imposed cleanliness. If you don't agree, look at the assembler code your compiler creates on a do while loop.

4. Name Server

The name server allows users to specify names rather than dotted ip address. Usually you will just need to use **resolve** and it will correctly handle names or dotted ip address formats.

longword **resolve(** *char* * **host_string);**

To save exe size, you may insist on dotted ip format only. Use inet_addr which returns an ip number in longword format, or 0 on error:

longword **inet_addr(** *char* ***dotted_ip_string);**

The kernel uses the following constants which are automatically filled in by sock_init. You may read def_nameserver and def2_nameserver, but writing to them is forbidden, particularly once the TSR version is released.

extern char ***def_domain;**

extern longword **def_nameserver, def2_nameserver;**

5. What You Have Learned

You have learned almost everything which you will need in order to program TCP/IP applications using Waterloo TCP. The remaining functions provide additional features, but are not really as important. You might like to look at some of the sample applications in the reference section or in the software distribution and see how they work. When you are resonably familiar with their basic layout, return to this section for a brief description of the rest of Waterloo TCP.

6. Nagle Algorithm

The performance and bandwidth costs of TCP can often be improved by using a simple and interesting algorithm commonly referred to as the Nagle Algorithm. The Nagle algorithm instructs us to not launch many small packets, but to store up new data until it can piggyback an acknowledgement, or it reaches a maximum size, or the previous ack timed out. For slow lines, Nagle reduces the number of headers transmitted which in turn reduces the ratio of headerbytes/databytes, which improves channel efficiency and will free up the channel for other uses. Without Nagle, Telnet keystrokes invoke a flurry of small, wasteful packets. With Nagle, characters seem to 'bunch up', but the actual performance is identical.

To assume that Nagle is only helpful over slow media is actually quite incorrect. TCP can be costly over any network media, even Ethernet or FDDI where bandwidth may not be a problem. Consider the cost of having your UNIX box receive many small packets from user TELNET sessions. If twenty secretaries are typing, say each at 80 words per minute. Those 133 packets per second are a nusance because they use up your limited ethernet buffers as well as lots of processing time for interrupts, acknowledgements, tcp protocol updates, etc. Over a local Ethernet, Nagle only has an influence when response times are slow, namely when the network is congested or the server is very busy. Clearly those are exactly the times we wish to reduce
redundant packets, so Nagle is again appreciated.

These two scenarios explain why Nagle is on by default, but Waterloo TCP allows you to disable it with a call. Some applications, such as X-Windows and real time data collection need responsiveness enough to warrent being traffic hogs. Neither application is suggested to be used over anything but local subnets if that gives you any idea of the importance.

sock_mode(*void* ***** **socket,** *const* **flag);**

eg. **sock_mode(s, TCP_MODE_NAGLE);**

sock_mode(s, TCP_MODE_NONAGLE);

You can turn it on or off as often as you like, thereby allowing you to gain the benefits of Nagle, while temporarily disabling it to permit truely urgent data to be handled immediately. *Note: The Nagle algorithm has no effect on UDP sockets.*

7. UDP Checksums

udp does not necessarily require checksums. According to the specifications, *udp* checksums should be enabled by default. Many systems will only support *udp* checksums on for all connections or off for all connections. Waterloo TCP defaults to checksums but allows you to override that decision on a socket by socket basis.

Incomming data is always checked if the remote computer uses checksums. Disabling local checksums speeds up both local and remote processing of datagrams.

A good reason to disable checksums is if your program implements its own checksum or you *know* that the protocol is being run over networks which implement checksums or cyclic redundancy checks. Remember, if the remote computer is on a different subnet, data may be passing through networks with no implicit checksum or other error detection.

sock_mode(*void* *** socket,** *const* **flag);**

eg. **sock_mode(s, UDP_MODE_CHK);**

sock_mode(s, UDP_MODE_NOCHK);

You can turn checksums on or off as often as you like, thereby allowing you to gain the benefits of checksums for reliable data while gaining speed over unimportant data.

Note: The checksum flag has no effect on TCP sockets.

8. UDP Record Services

As discussed earlier, Waterloo TCP can work with *udp* connections just as easily as *tcp* connections, in fact they both seem to act identically. Of course this appearence is actually quite ficticious. The basic sock read functions (**sock_read**, **sock_fastread**, etc.) are not adequate for all your *udp* needs. The most basic limitation in their is inability to treat *udp* as a record service.

By record service I mean you will receive distinct datagrams which should be kept and distinct. You will wish to know the length of the received datagram and the sender (if you opened in broadcast mode). You may also receive the datagrams very quickly, so you must have a mechanism to buffer them.

Rather than add clumsy appendages to the standard socket reading functions to handle these new requirement, I elected to add new functions specifically for record service.

Once a socket has been openned with **udp_open**, you can use **sock_recv_init** to initialize that socket for the following functions. Note that **sock_recv** and related functions are *incompatible* with **sock_read, sock_fastread, sock_gets, sock_getc** and **sock_scanf**. Once you have used **sock_recv_init**, you can no longer use the older style calls.

sock_recv_init installs a large buffer area which gets segmented into smaller buffers. Whenever a *udp* datagram arrives, Waterloo TCP stuffs that datagram into one of these new buffers. The new functions scan those buffers. You must select the size of the buffer you submit to sock_recv_init, you should make it as large as possible, say 4, 8 or 16 Kbytes. **sock_recv_init** returns 0 on success.

sock_recv scans the buffers for any datagram received by that socket. If there is a datagram, the length is returned and the user buffer is filled. If no datagrams were found, sock_recv returns 0.

sock_recv_from works identically to **sock_recv** with the addition that it places the source *ip* address in a user location pointed to by *fromptr*. This allows you to write a server using *udp*. To reply, you must open a *udp* socket with the *udp_open* function. There is no facility in Waterloo TCP to simply send **udp** datagrams without a socket.

int **sock_recv_init(***void* ***socket,***byte* ***buffer,***int* **bufferlen);**

int **sock_recv(** *void* ***socket,** *byte* ***buffer,** *int* **maxlength);**

int **sock_recv_from(** *void* ***socket,** *byte* ***buffer,** *int* **maxlength,** *longword** **fromptr);**

9. General User Functions

psocket will print a remote host and port to standard output

void **psocket(** *void* ***socket);**

sockerr returns an ASCII string containing the last error message found and believed to be the cause of the current problem.

char ***sockerr(** *void* ***socket);**

sockstate returns a brief ASCII string containing the current state of the connection. For *udp*sockets, the state is always "UDP Socket", since UDP is a connectionless protocol and has no state.

char ***sockstate(** *void* ***socket);**

getpeername will return the remote computer's IP address. **getsockname** returns the local port number. These are UNIX similar functions, see the reference section for more details and differences between these functions and their UNIX counterparts.

void **getpeername(** *void* ***socket,** *struct sock_addr* ***** **p,** *word* ***length);**

void **getsockname(** *void* ***socket,** *struct sock_addr* ***** **p,** *word* ***length);**

gethostid() returns the IP address of the machine, **sethostid()** sets the IP address.

longword **gethostid();**

longword **sethostid();**

getdomainname fills the user supplied string with the current domain name. A pointer to the string is also returned. **setdomainname()** sets the domain name to the passed string and returns the pointer to that string.

char ***getdomainname(** *char* ***user_buffer);**

char ***setdomainname(** *char* ***string);**

ntohs, htons, ntohl and **htonl** convert host to network byte ordering.

word **ntohs(** *word* **value);**

word **htons(** *word* **value);**

longword **ntohl(** *longword* **value);**

longword **htonl(** *longword* **value);**

rip will remove cr's and lf's from the string and return a pointer to the string.

char ***** **rip(** *char* ***** **string);**

To handle control breaks correctly, (remember we get passed interrupt calls) Waterloo TCP automatically installs a control break handler. You may change the mode of the control break handler with **tcp_cbreak** or you may write your own handler.

If you use **tcp_cbreak**, *mode* is a combination of 0x01 disallow ctrl breaks, and 0x10 display a message on ctrl breaks. **sock_init** calls this with mode 0x10 which displays a message and then aborts the program.

void **tcp_cbreak(** *word* **mode);**

If you use your own control break handler, remember to call **sock_exit()** with no parameters to release the packet driver and to close all sessions.

10. Kernel Accessible Routines

If you are doing your own delaying functions, you will need to be able to tick the kernel. **tcp_tick** returns almost immediately and will return 0 if the optional socket passed has been closed, timed out, or reset by the other host. The full operations of **tcp_tick** are described in the "indepth understanding" chapter.

word **tcp_tick(** *void* ***s);**

The macro **sock_tick** calls **tcp_tick** and then acts similarly to the **sock_wait_...** functions except that it does not perform any waiting. See the earlier section on "Kernel Waiting" . Relatively little in the way kernel debugging facilities is available in the basic system, but what is provided may be turned on using **set_debug_state**. However, TCPDBUG.EXE is a better form of debugger. See the chapter on debugging.

void **set_debug_state(** *word* **state);**

eg. **set_debug_state(0);** turns off debugging

set_debug_state(1); turns on basic debugging

If you are debugging your code and need to see some error termination message, call **sock_debugdump** with your socket. Also described in the debugging chapter.

void **sock_debugdump(** *void* ***s);**

eg. **sock_debugdump(&socket);**

A user-defined yield function will be called on any kernel waiting functions if you set the function with **sock_yield**. If parameter s is NULL, yield_fn will be inherited by all subsequent open or listen commands, otherwise s must point to a specific socket and yield_fn is applicable only to that socket. An obvious use for **sock_yield** is to add multitasking features. Hopefully, someone will release a public domain version of their tasker in this application. See the appendix on adding taskers.

void **sock_yield(** *void* ***s,** *void (**)***yield_fn));**

eg. 1 **fn() { printf("WAITING"); }**

```
main()
{
        tcp_Socket socket;
        sock_init();
        sock_yield( NULL, fn );
        if ( !tcp_open( &socket, ...
```

eg. 2 **finger(host, port)**

```
{

        tcp_Socket socket;
        sock_init();
        if ( !tcp_open( &socket, 0, host, port, NULL )) exit( 0 );
        sock_yield( &socket, fn );
        ...
```

Notes: **sock_yield(NULL,** ... does not affect functions which are already open

sock_yield(s, NULL) or **sock_yield(NULL, NULL)** can be used to reset the appropriate sockets so that no yield function will be called.

There are two user accessible timer routines. All work under the same premise, you supply a value in seconds and save the new time, later you make successive calls to determine if the timer has timed out. Day wraps are handled and timeouts up to about 18 hours may be selected.

The **sock_wait...** timers use the following with 0 meaning never timeout

void **ip_timer_init(** *void* ***s,** *word* **seconds);** to set the timer

word **ip_timer_expired(** *void* ***s);** to check the timer, 0 on not timedout

You may use as many user defined timers as you wish since you will hold the timer values. *Kludge: if you must, you can adjust the timeouts by adding or subtracting constants to the value returned from set_timeout. Add 1L for each 1/18 second. Eg. set_timeout(1) - 17 would specify a timeout of about 55 milliseconds.*

longword **set_timeout(** *word* **seconds);**

word **chk_timeout(** *longword* **timeouttime);**

```
eg. longword timeout;
timeout = set_timeout( 20 );
while ( ! chk_timeout( timeout ))
        printf("not timed out yet");
```

11. ICMP and PING

ICMP (Internet Control Message Protocol) is the standard method for Internet hosts to send control messages. Although the ICMP protocol is technically sent using *ip* (much like *tcp* and *udp*), it is considered a required part of IP rather than another layer. IP uses ICMP to complain, query and respond at a level lower than *tcp* or *udp*. Since ICMP is usually implemented as a part of IP, it usually runs at the same high priority as IP and is not typically bogged down even when the *tcp* and *udp* protocols are very busy. One excellent use of this fact is the 'PING' program.

PING is the standard name for the standard method of determining whether another computer is alive, and if so, for computing the round-trip-time (RTT) between the two computers. PING works by sending an ICMP echo request packet to the other host. The remote computer responds, usually as fast as possible, and the original computer records the difference in time.

Waterloo TCP supports ping with two functions: **_ping** which generates the request and **_chk_ping** which should be used periodically to check the ping cache for a response from that particular host. The ping cache has length one and is overwritten on each incoming ICMP echo response (which posted during **tcp_tick** or **sock_tick**) and nullified on each successful **_chk_ping** request. **_chk_ping** returns 0xffffffff if the cache does not contain a new entry for that host, or the number of *ticks* between the transmission and response receipt of the currently returned sequence number. Failure to call **tcp_tick** (or its counterpart **sock_tick**) or **_chk_ping** very
frequently will result in lost receipts due to the cache size.

word **_ping(** *longword* **host_ip,** *longword* **sequence_number);**

longword **_chk_ping(** *longword* **host_ip,** *longword* ***sequence_number);**

For more details on how to use the ping functions, look at the example program **PING.C**.

12. Undocumented Calls

Only the calls listed *in this manual* (and extensions in future revisions of the manual) are supported. Undocumented calls may have parameters change or functionality changes in future revisions. Likewise the *tcp_Socket* and *udp_Socket* structures will change. I intend to offer a service similar to setbuf to allow you to use larger *tcp* windows. In the future, the kernel may be moved to a TSR which will support multiple EXE's. This support will help with WINDOWS, DESQVIEW, and simply when user's shell out of TELNET and want to run LPR or something. All undocumented functions would be totally unavailable at that time.

13. Indepth Understanding

In this section, I simply went through my source code and listed what I saw. By no means is this chapter essential for programming with Waterloo TCP. However, if my descriptions are sparse elsewhere, this might give you a feel for what is going on.

The three open commands, **tcp_open**, **udp_open**, and **tcp_listen** start by zapping your buffer space and putting it to a linked list. In the case of the **..._open** commands, the destination address is checked for accessibility. If it is a local broadcast, the hardware broadcast is selected, otherwise local tables or subsequently *ARP* is used to resolve the address. A failure on the resolution of the address at this point will mean the open function fails. **tcp_open** also sends the first datagram in an attempt to open the tcp connection.

As you may recall, **sock_yield** is used to specify a global yield function. If **sock_yield** has been used, all subsequent open and listen calls inherit that yield function by default. An explicit call to **sock_yield** may be used to remove or replace the yield function for a specific socket. **sock_yield** itself merely updates the pointer in a particular socket or the pointer in a system variable which is used on future opens and listens.

The **sock_close** function identifies the socket type. In the case of tcp, a *fin* is attempted and subsequent calls to **sock_wait_closed** or a similar service must be made so the close can be completed or timedout. In any case, once a socket is truly closed, (immediate for udp) the function removes the socket from the appropriate list and sets a value in the buffers so future i/o should fail.

tcp_tick is the basic call to the tcp kernel. It performs everything which can be done at the current time, then returns immediately.

First it checks the receive queue for packets and demultiplexes the datagrams among the tcp, udp, icmp and arp protocols and the handler for the protocol performs the necessary operation on the data. For tcp, the connection state is updated, the statistics are updated for the retransmission strategy, and an acknowledgement datagram is sent immediately. The ack piggybacks any new, untransmitted data and include the new window size available for the remote tcp to fill. In the case of udp, the new data is appended to the buffer if space permits. Arp receipts update the local arp table and generate a response if we are being queried. ICMP messages are displayed. If the message indicates an error, a pointer to the message is placed in the socket structure and an attempt to correct the situation is made. User applications may read the message by issuing a call to **sockerr**.

After all incoming packets are processed, the retransmitter is called. **chk_timeout** is used to ensure retransmissions for any socket are made not more than once per second. For each open tcp connection with pending data, it checks the last transmission time and recommended retransmission time. If

appropriate, a new datagram is sent. If the user previously called sock_close on a tcp socket, the socket is checked for a close timeout.

The read and write functions can be grouped as to blocking and non-blocking functions. The nonblocking functions obviously return immediately. The blocking functions repeatedly call **tcp_tick** to handle incoming datagrams, and the user defined yield function if one had been provided with **sock_yield**. Non-blocking functions may or may not yield. All read functions transfer data then update the buffer to reflect the new space. In the case of tcp, reading data affects the available data window. Adjustments to the window size are placed in the socket structure but are not immediately transmitted. At the next sending of data, or after a small timeout period, the new window size will be automatically transmitted. *(At the date of publishing, the window size is immediately retransmitted which may lead to the well-documented sillywindow syndrome.)*

The write functions are dependant upon the local transmit buffer size available and not the tcp window. An attempt is made to immediately write the data if the buffer size is sufficiently large or the user has specified to do so with **sock_flushnext** or **sock_flush**.

sock_printf and **sock_scanf** significantly increase the size of executable and may cause problems in TSRs.

The functions **sock_rbsize**, **sock_rbused**, **sock_rbleft**, **sock_tbsize**, **sock_tbused** and **sock_tbleft** merely check the socket type and return the values from the structures. No other processing is performed.

The name resolution functions, available through **resolve** will return immediately if the passed argument is a numeric dotted ip address. Otherwise the primary name domain server will be queried. Failing resolution, the secondary nameserver will be consulted. **resolve** blocks until the name is resolved. Normal packet processing continues since **tcp_tick** is also called constantly. This is one instance of the default yield function being over-ridden, **resolve** does not honour yield functions declared with **sock_yield** because it uses a static buffer space.

The address resolution protocol handler caches a table of arp related data. Currently the cache size is 20. Calls to **_arp_resolve(** *longword* **ip,** *void* ***hardware)** attempt to get a hardware address for the destination ip address. First the cache is checked, then, if the ip address is on our subnet, an arp request is made. Otherwise the gateway is assumed and the address is found using a recursive call to **_arp_resolve** with the gateway's ip address. **_arp_resolve** returns with zero if it times out, or a non-zero value if the address is found. Typically arps are fast. During the response delay, **tcp_tick** is repeatedly called to handle incoming packets and manage other sessions. The optional yield function set with **sock_yield** is not called because **_arp_resolve** is too low in the kernel, arps are rare, and the response should be fast enough to not be a problem.

Many socket functions must determine if the passed structure is a valid open socket. They do so by calling **_chk_socket(** *void* ***s)**. The returned values are 1 for udp, 2 for tcp, and 0 for failure. A text string indicating the status of a connection may be found using the **sockstate** function.

The bottom two levels of code are the packet driver and Ethernet handlers. The "Ethernet" code simply encapsulates outgoing packets and removes headers on received packets, it does not manage the ip to hardware translation, that is performed in the arp code and managed by ip. The packet code performs sends, receives, initializes the packet driver, resets the packet driver and reads the hardware address. All outgoing frames are built in a single frame buffer as they are needed. Incoming frames are placed into one of the buffers during the interrupt and processed inside **tcp_tick**. *The current implementation is limited to 5 buffers, each of length 1500 bytes.*

set_debug_state can be used to enable or disable the internal tcp debugger code. Most of the messages are to help me solve kernel problems rather than user-level debugging. Whether you find the messages useful or not is another question. In **PING.EXE**, for example, the debugger will transmit the destination ethernet address. For debugging your code, use the TSR, **TCPDBG.EXE** (release date, March 15, 1991).

The initialization code in sock_ini.c calls pcconfig.c to open the configuration file, read the contents, parse the lines and process them. Application programmers, of course, may use the **usr_init** hook to extend the configuration file.

If the attempt to open the config file failed, or if BOOTP was selected, functions in pcbootp.c get called. They open up a simple udp session, defaulting to broadcast unless otherwise configured, and pump out bootp requests with random durations between. The randomness is intended to allow several machines to all boot up at the same time, but to not synchronize their requests. The randomness is partially dependant upon the millisecond clock and the ethernet address of the network card.

The **sock_recv** feature is located in pcrecv.c. The udp socket structure's data buffer is trampled upon since the two sock_recv and normal input functions are incompatible. Each time incoming data is found during a tcp_tick(), the udp_handler function processes the packet and decides whether to place the data in the usual small buffer, or to do an upcall to the udp socket's datahandler function which **sock_recv_init** changed to be its own **_recv_daemon**. **_recv_daemon** scans the available buffers and places the packet in a free buffer if one exists.

14. Using Multitaskers with Waterloo TCP

Oddly enough, while the MIT tcp kernel started with an idea and a multitasker, the Waterloo TCP kernel does not have a resident tasker. For other projects I have written my own tasker and used other peoples as well. For this particular situation, however, I decided to leave the tasker up to the user. If you

are familiar with Desqview or whatever, either check the distribution site or help writing whatever is necessary

All blocking read, write, and **sock_wait...** functions are programmed using loops which repeatedly call the kernel **tcp_tick** function. Also included in the loops are a call to the yield function for that particular socket. The yield function may be set on a per-socket basis, or on a system-wide basis with the **sock_yield** command.

void **sock_yield(** *void* ***socket,** *void (**yield_fn)*);

To simplify a multitasker environment, you may call **sock_yield** immediately after **sock_init** with the socket parameter set to NULL and the function parameter set to your yield function. All subsequent opens and listens will inherit that yield function automatically. Note that **sock_yield** may be called with socket set to NULL at any time, but it only affects future opens and listens, not currently opened or listening sockets.

You may set the yield function for a particular socket by opening the socket with either an open or listen command, then calling **sock_yield** with a pointer to that socket and to the yield function to be called. This is useful to turn off the tasker for a specific socket, such as ones which use static data structures rather than stack based structures. The **resolve** function falls into this category though it could certainly be re-written. *Note: if you wish to not perform yields, you should call **sock_yield(&socket, NULL);** immediately after the open function.*

If the yield function is set to NULL, the kernel is instructed to not call any yield function.

eg.
```
#include <tcp.h>
    void yield() { printf("WAITING,");}
    main(int argc, char **argv)
    {
        tcp_Socket socket;
        longword host;
        sock_init();
        sock_yield( NULL, yield );
        if (argc < 2 ) exit(3);
        host = resolve( argv[1] );
        if (! tcp_open( &socket, 0, host, 25, NULL)) {
            puts("unable to open socket");
            exit( 3 );
```

```
}
sock_yield( &socket, NULL );
...
```

The example above demonstrates how to set all sockets to use the function 'yield', then goes on to create a socket which will *not* use yield.

The kernel will not call the yield function at any critical areas as long as the tasker is not preemptive.

ภาคผนวก ค.

คู่มือการใช้งาน MC35 Siemens Cellular Engine

Date:	28.11.2001
Technical Support:	wm.support@mch.siemens.de
DocId:	GPRS-startup-00-V01.01
Status:	Released

General note

With respect to any damages arising in connection with the described product or this document, Siemens shall be liable according to the General Conditions on which the delivery of the described product and this document are based.

This product is not intended for use in life support appliances, devices or systems where a malfunction of the product can reasonably be expected to result in personal injury. Siemens AG customers using or selling this product for use in such applications do so at their own risk and agree to fully indemnify Siemens for any damages resulting from illegal use or resale.

Applications incorporating the described product must be designed to be in accordance with the technical specifications provided in these guidelines. Failure to comply with any of the required procedures can result in malfunctions or serious discrepancies in results.

Furthermore, all safety instructions regarding the use of mobile technical systems, including GSM products, which also apply to cellular phones must be followed.

Subject to change without notice at any time.

Copyright

This product is an original Siemens product protected by US, European and other patents.

2 Introduction

This document describes how to establish a PPP connection with Microsoft Windows NT4.0 using the prototype of the MC35 GSM/GPRS engine. Configurations and settings for other operating systems (Windows 95/98) may slightly differ, but the general proceeding is similar.

The current release of MC35 is able to connect to all networks which offer GPRS services (in Germany for example T-D1, D2, E-plus, Viag). Please note that not all of the GPRS related AT commands have been fully implemented yet.

The figures presented in this guide, show the settings required for the German network provider T-D1. Users of other networks must adapt their settings accordingly. Chapter /9.2/ references the settings for all German network providers.

2.1 References

/1/ MC35 AT Command specification (filename: mc35_atc_01_<version number>.pdf)

/2/ Request for Comments: 1661 - The Point-to-Point Protocol (PPP)

/3/ Request for Comments: 1994 - PPP Challenge Handshake Authentication Protocol (CHAP)

2.2 Abbreviations

APN	Access Point Name
CHAP	Challenge Handshake Authentication Protocol
CID	Context Identity
CSD	Circuit switched Data
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
DSB	Developer Support Box
GPRS	General Packet Radio Service
GSM	Global System of Mobile Communication
IP	Internet Protocol
NOM	Network Operation Mode
OS	Operating System
PDP	Packet Data Protocol
PIN	Personal Identification Number
PPP	Point to Point Protocol
PSD	Packet switched Data
QoS	Quality of Service
RAS	Remote Access Service
ROM	Read Only Memory
SIM	Subscriber Identity Module
TCP	Traffic Control Protocol

3 What you need

1. GPRS settings from your network provider

Before you can set up your device to access the GPRS network, contact your network provider to obtain the following information:

- APN (name of an access point that connects the mobile network to the Internet)
- Primary and secondary DNS
- Default QoS settings
- IP header compression
- IP address (DHCP or static)
- User name and password (may be optional)

2. Modem installation and configuration

You will need to set up the modem driver used for the Dial-Up connection. Follow the steps in chapter /4/.

Note:

Administration rights for your Operating System may be needed in order to install the modem driver. Contact your local system administrator for advice.

3. Dial-Up Network installation and configuration

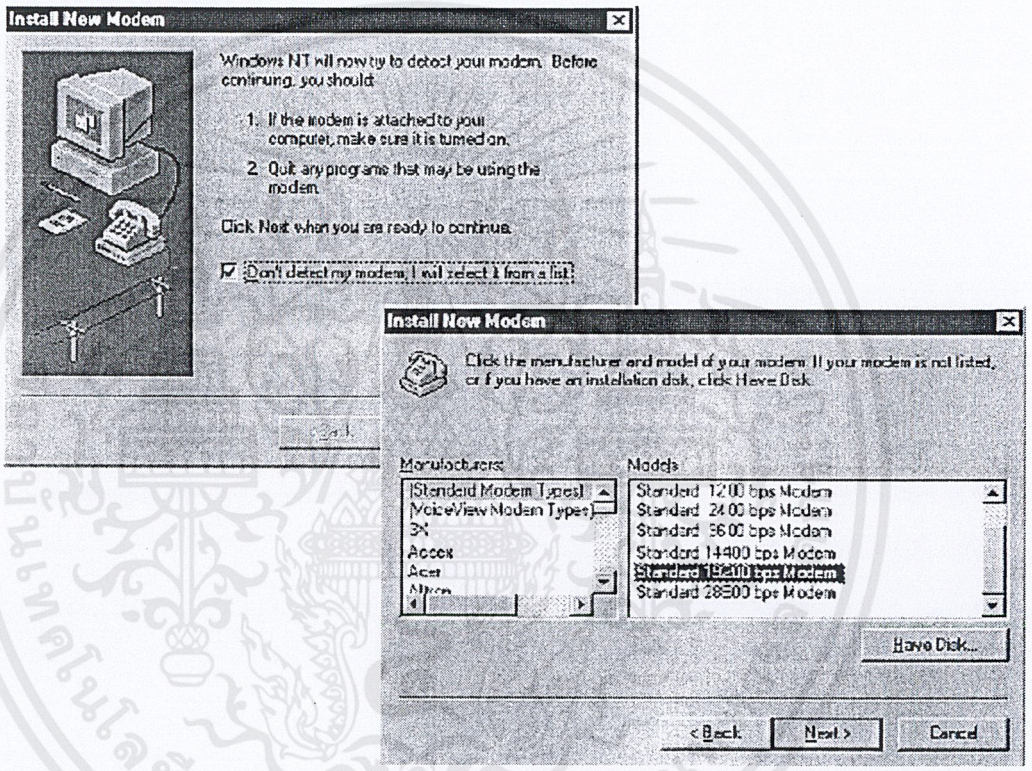
Furthermore, you will need to configure the Dial-Up Network itself. Follow the steps in chapter /5/.

4 Setting up a Windows modem driver

4.1 Adding a new modem

If no Standard 19200 bps Modem has been installed yet, add a new standard modem to the modem section of the control panel:

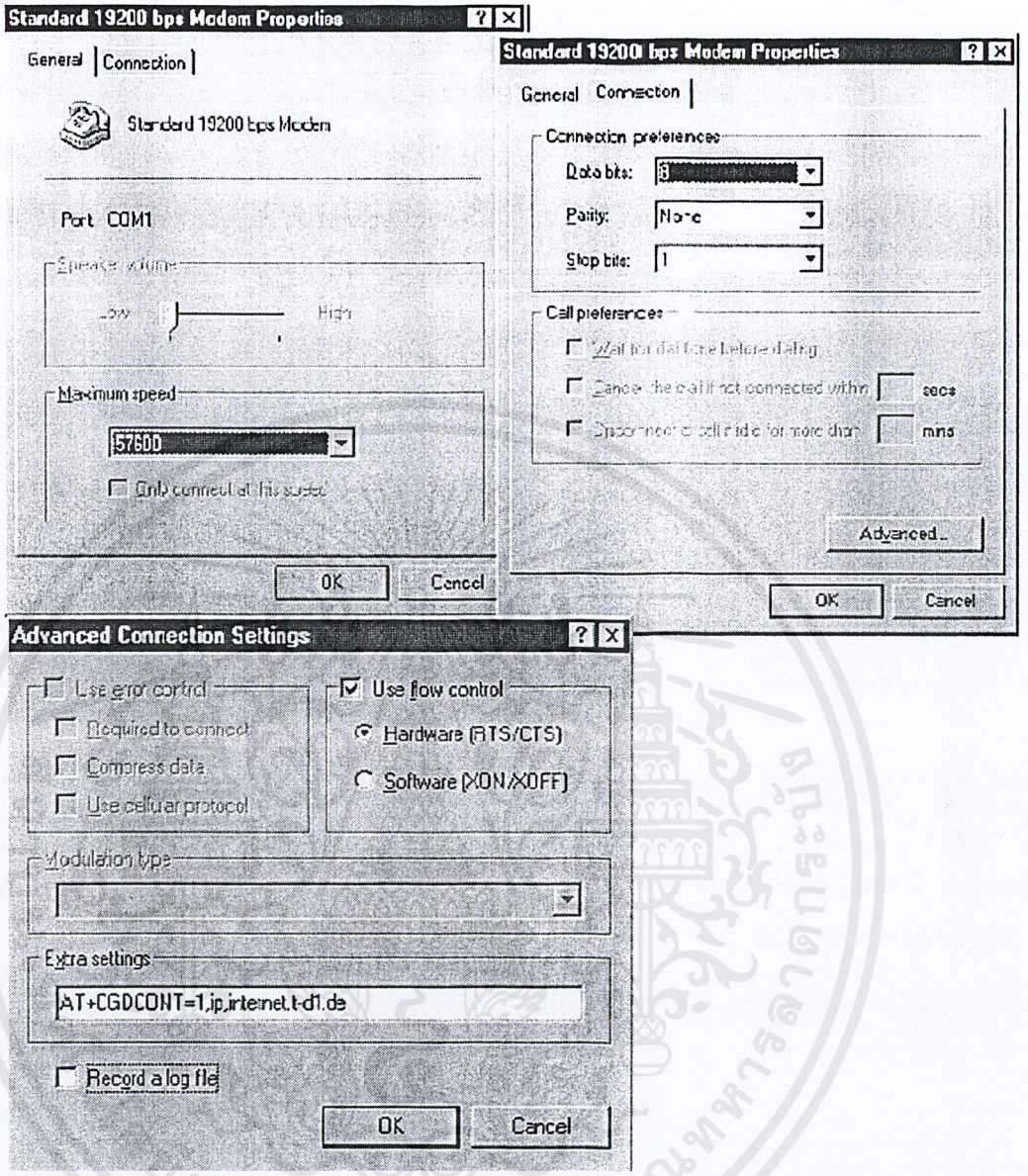
Tap **Start -> Properties -> Control panel -> Modems.**



Follow the instructions on the screen and click **Next** to finish the modem installation.

4.2 Configuring the modem driver

To configure the modem driver, select the baud rate from the **Maximum speed** pull down menu (recommended: 57600) and choose the **Connection preferences** as illustrated in the figures below.

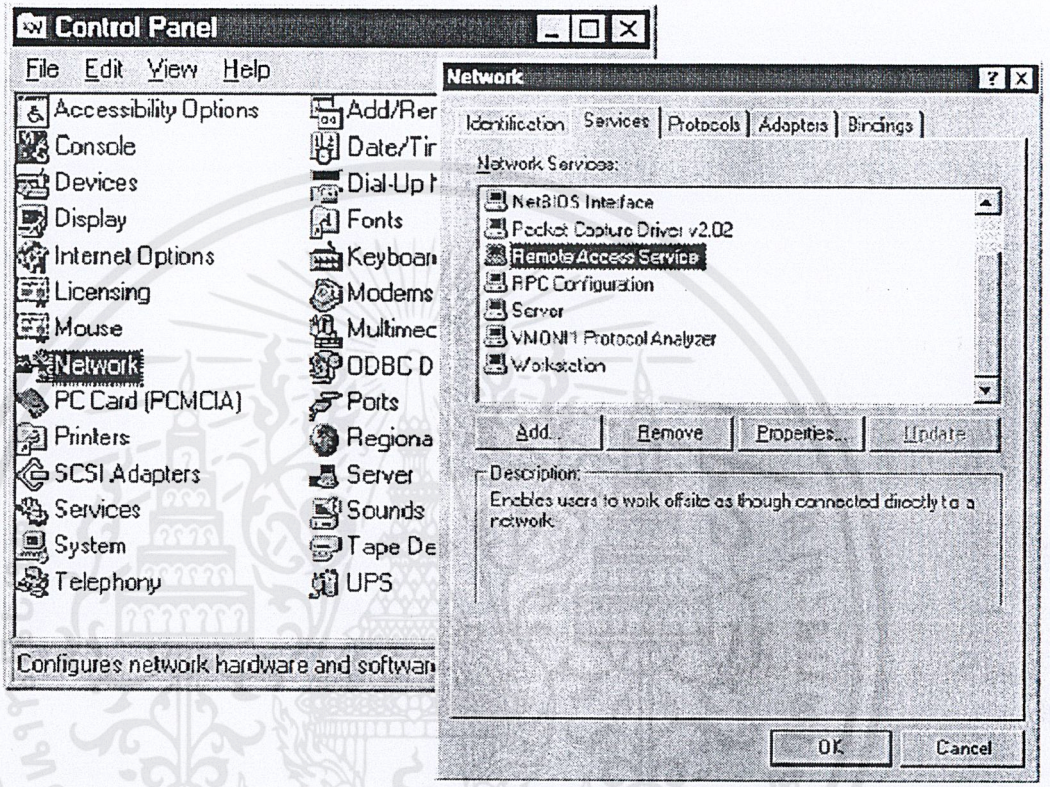


Now, tap the **Advanced** button to configure the following settings: Tick the **Use flow control** box. In some cases you may be required to turn off the **flow control.** The **Extra settings** can either be left blank or used to predefine the PDP context. See chapter /6.2/ for information on whether you need to define the PDP context and how to proceed. Chapter /5/ describes the steps to activate a predefined PDP context.

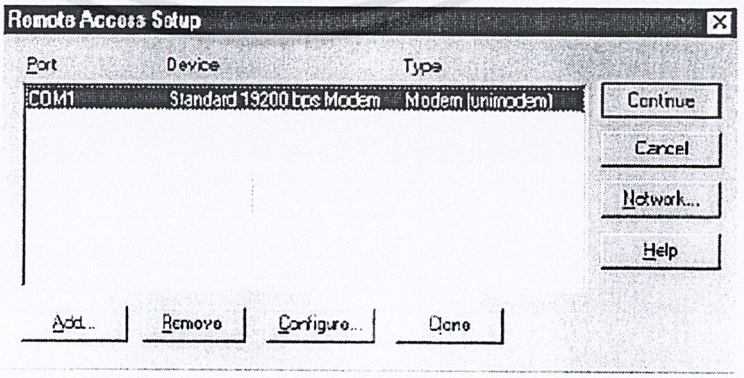
In the example above, the settings predefine a PDP context where CID = 1, PDP type = *ip*and APN = *internet.t-d1.de.* Note that the APN represents the German network provider T-D1 and needs to be replaced with the APN address provisioned by your operator.

4.3 Adding the modem driver to the Remote Access Service (RAS)

In some operating systems (e.g. Windows NT or Windows 2000) it is necessary to activate the modem driver by adding it to the Remote Access Service. The following figures show the Windows NT setup.



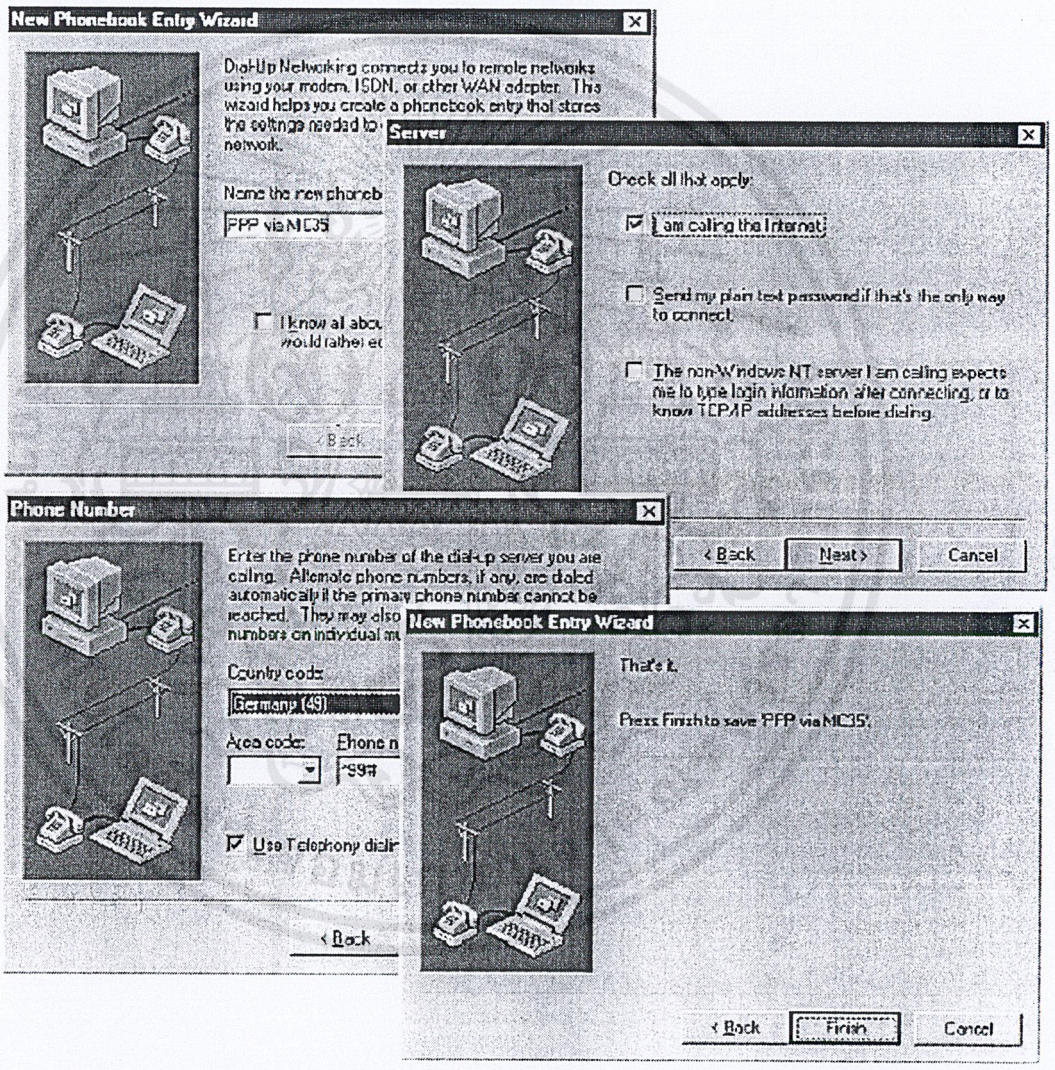
Open the **Control Panel** and tap **Network**. Choose the **Services menu** and tap **Remote Access Service**. If more than one modem driver has been installed, choose the correct one and tick the **Add** button in order to add the driver to the RAS. From now on, the chosen driver will be used when starting the dial up network.



5 Setting up the Dial-up network

5.1 Adding a new phonebook entry

The **Dial-Up Network** icon is located in the **My Computer** folder on your Desktop. Select **New** to open the wizard shown below. If the Dial-Up Network has not yet been set up you can follow the resulting screens to install it.



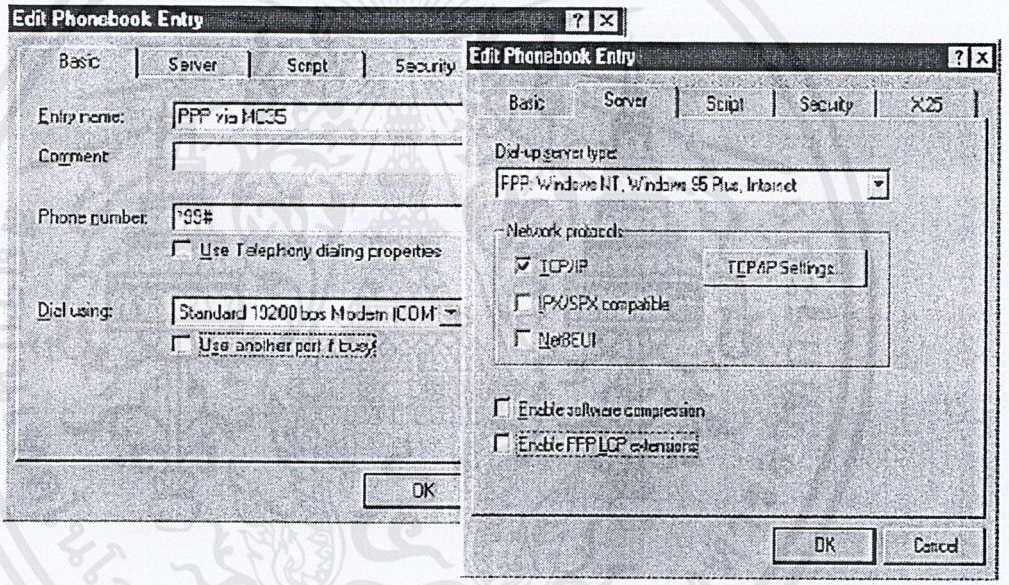
- Enter a suitable label for the phonebook entry (e.g. **PPP via MC35**) and tap **Next**.
- In the **Server** panel, select all entries which apply and tap **Next**

- Select the **Country code**. Use the **Phone number** field to enter ***99#** or ***99***1#**, depending on what approach you have chosen to handle the PDP context (see chapter /4.2/). Click **Next** and **Finish** to close the wizard.

5.2 Configuring the settings

Choose the newly created phonebook entry from the Dial-Up Network folder and enter the settings described below.

- The Entry name, the phone number and the modem type (Standard 19200 bps modem) should be those you have specified using the wizard (as described in chapter /5.1/).Whenever needed, you can use these dialog boxes to change the settings.

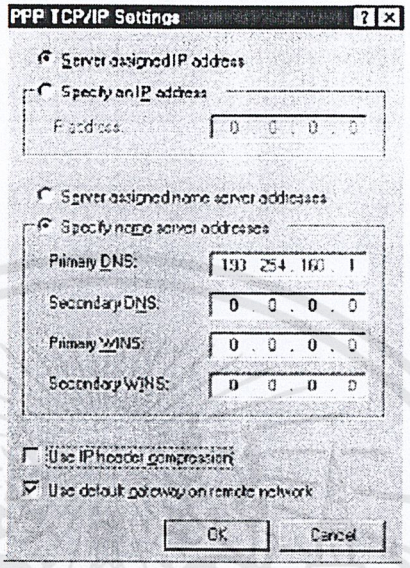


- The **Dial-up server type** is *PPP: Windows NT, Windows 95 Plus, Internet*. Choose *TCP/IP* to specify the allowed **Network protocol**. **Software compression** and **PPP LCP extensions** should be disabled. Tap the **TCP/IP settings...** button to proceed.

- Choose **Server assigned IP address** and enter the DNS server address used by your provider. The example shows the **primary DNS** address of the German network provider T-D1: 193.254.160.1. Settings of other providers are listed in chapter /9.2/ (only Germany) or must be requested from the local network provider. IP header compression should be disabled, but the default gateway is required.

Note:

The settings may vary with the local service provider. In some cases it may be necessary to tick IP header compression. This information has to be requested from the local provider.



6 Getting started

Make sure the MC35 connects properly to the application platform. For testing purposes, you can use for example the evaluation kit named DSB35 Support Box.

6.1 Registering to the GSM network

To be able to attach to the GPRS services your terminal or mobile must be registered to the GSM network, i.e. PIN authentication must have been done during the current session. You have two ways to do so:

If the used SIM card is unlocked, the terminal or mobile will automatically log on to the network without requesting SIM PIN1.

If the SIM card is locked you can use any terminal program to enter the PIN. The terminal program should be properly connected to MC35 and configured for 8 Databits, No Parity, 1 Stopbit, flow control and the recommended bit rate of 57600 bps. IMPORTANT: When PIN authentication has been successful, close the terminal program to free the communication port. To enter the PIN use the AT+CPIN command (see /1/ for detailed instructions).

Example:

AT+CPIN=1234

OK

Now, quit the terminal program.

6.2 Defining the PDP context

In order to activate a PDP context with the dial up network, a PDP context needs to be defined before. This means, that the mobile needs to inform the network about certain parameters, so that the context will be activated correctly. The PDP context definition can be done in several ways:

- If no entry has been made in the **Extra settings** of the modem properties and the dial up network is started with the phone number ***99#**, no individual PDP context values are known. In this case the module will use default values for the connection. The default values are the ones, which had been used for the last connection.

Note:

In some cases it may be required to predefine a PDP context. For example, if the local network provider requests an APN. In these cases one of the two following options mustbe chosen.

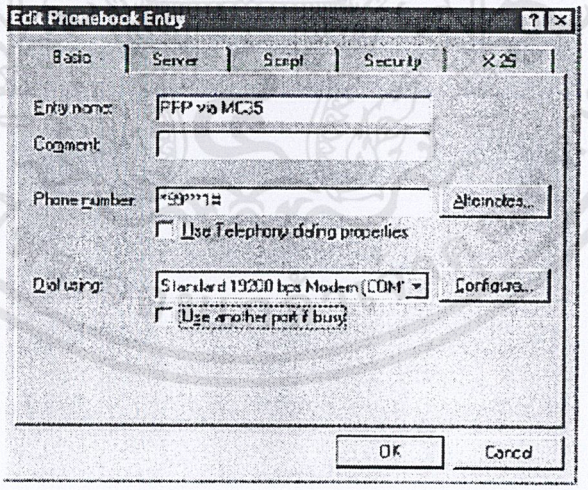
- If the **Extra settings** of the modem properties have been filled in, the CID is known. In order to activate the PDP context with the predefined values, ***99***<cid>#** instead of ***99#** must be chosen as the phone number entry (see figure below). However, if ***99#** is chosen, the module will ignore the predefined values and activate the PDP context with the default values.

- The PDP context may also be defined manually via a terminal program (57600 Baud, 8 Databits, No Parity, 1 Stopbit, flow control). Start the terminal program. Define the PDP context using the .AT+CGDCONT. command (see also /1/ for details):

Example: AT+CGDCONT=1,ip,internet.t-d1.de <RETURN>

Close the terminal program when finished to free the communication port.

Choose ***99***<cid>#** instead of ***99#** as the phone number entry in the dial up network, in order to activate the PDP context with the predefined values. Otherwise, the module will ignore the predefined values and activate the PDP context with the default values.



6.3 Defining the QoS profile

After the PDP context definition specific optional QoS parameters like **Delay Class** or **Mean throughput rate** can be set before activating the PDP context with the Dial-up network. This can be done via a terminal program (57600 Baud, 8 Databits, No Parity, 1 Stopbit, flow control).:

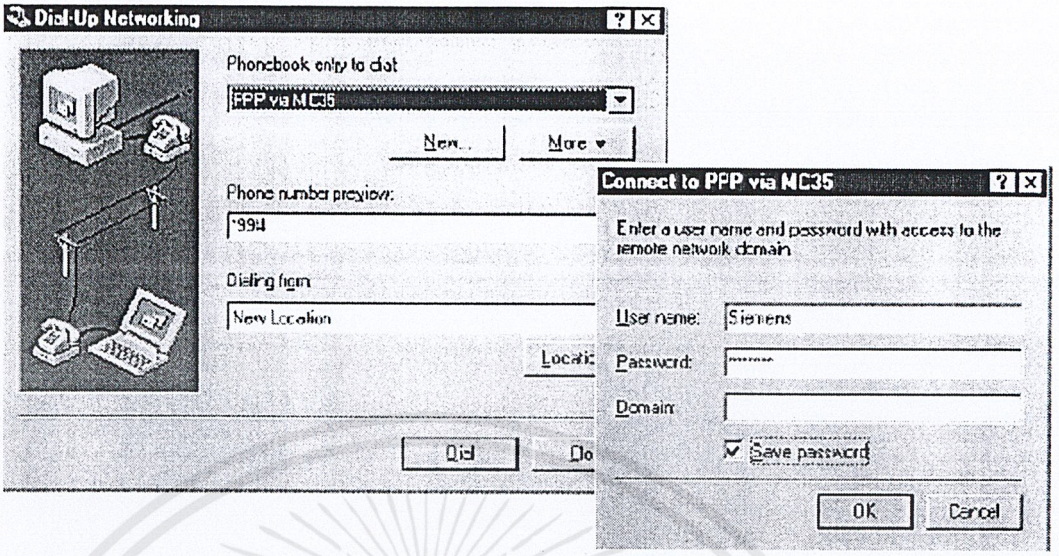
Start the terminal program. Define the Set the parameters using the .AT+CGQMIN. and/or .AT+CGQREQ. commands (see also /1/ for details). Close the terminal program when finished to free the communication port.

Example: AT+CGQREQ=<cid>,<precedence>,<delay>,<reliability>,<peak>,<mean> <RETURN>
AT+CGQMIN=<cid>,<precedence>,<delay>,<reliability>,<peak>,<mean> <RETURN>

Note:
The QoS AT commands can only be used when a PDP context has been defined! In order to activate the PDP context with the predefined QoS values, ***99***<cid>#** instead of *99# must be chosen as the phone number entry (see also chapter /6.2/). However, if ***99#** is chosen, the module will ignore the predefined values and activate the PDP context with the default QoS values.

6.4 Setting up a GPRS call

Go to the Dial-Up Network folder and select the connection you just created. The example illustrated below uses the phone number ***99#** to set a GPRS call. Depending on your configuration, this may need to be replaced with ***99***1#**. Enter the **User name** and **Password** and, if dialing from a corporate network, the domain. If you tick **Save password**, no panel requesting the password will open in the future.



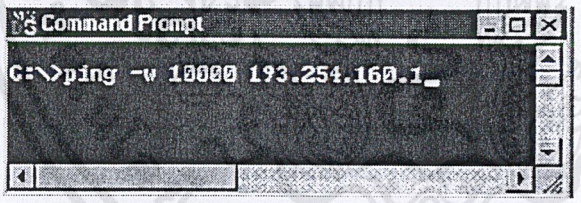
Note:

User name and password may in some cases be optional. Currently the **domain** field is ignored for GPRS network access. Address to your network operator for possible changes and more information.

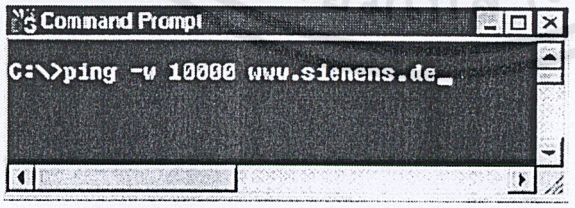
6.5 Quick test

To test the connection, simply initiate a ping at the command prompt.

1st test: ping without DNS name resolution:



2nd test: ping using DNS name resolution:



The "-w. parameter specifies the number of milliseconds to wait for the reply. To run the test type for example "ping .w 10000 www.siemens.de". The amount of time may vary with the network access. It is recommended to enter a value in the range from 10000 ms to 20000 ms.

6.6 Browser Settings

No proxy server needs to be configured in the browser.

If you have Internet Explorer installed you can select the phonebook entry created for the GPRS access as your favorite dial-up networking connection. To do so, go to the Tools menu and tap Internet Options / Connections. Select the phonebook entry and check one of the three dial-out option buttons. The sample phonebook entry used throughout this guide, would be listed as "PPP via MC35".

Please note that, currently, the GPRS service is not yet implemented to the full extent. This may slow down responses to requested URLs.

6.6 Terminating the GPRS call

For stopping a GPRS data connection, deactivating a PDP context and detaching the module, see chapter /7.4/.

7 GPRS AT commands

7.1 General

To transmit data via GPRS the application software uses a PPP stack. Most standard operating systems (e.g. Windows, Unix/Linux) use it as a part of a standard modem driver. In Windows based operating systems these drivers are part of the Dial-up network. For other operating systems these drivers must be provided by the user in order to enable TCP/IP connections.

However, application software which does not run on a standard OS platform or which does not use the Dial-up network, can establish a PDP context by executing a series of AT commands. Of course, you may also manually type these commands using any terminal program. Afterwards the PPP stack is activated.

Refer to the following chapters for instructions.

7.2 Discrete PDP context activation

- **Entering the PIN (AT+CPIN)**

  The connection of the MC35 to the terminal program and the registration to the GSM network are described in chapter /6.1/.

  Example: AT+CPIN=1234 <RETURN>
  OK

- **Defining a PDP context (AT+CGDCONT)**

  Chapter /6.2/ describes the procedure of how to define a PDP context.

  Example: AT+CGDCONT=1,ip,internet.t-d1.de <RETURN>

- **Defining a QoS profile (AT+CGQREQ, AT+CGQMIN)**

  Chapter /6.3/ describes the procedure of how to define a QoS profile.

  Example: AT+CGQREQ=1,3,4,3,0,0 <RETURN>
  AT+CGQMIN=1,3,3,3,0,0<RETURN>,

  where the numbers correspond to <PDP context identifier>, <Service precedence>, <delay class>, <reliability class>, <peak throughput rate>, <mean throughput rate >.

- **GPRS attach (AT+CGATT)**

  In order to use the GPRS service the MC35 must be GPRS attached. From that moment on the PLMN knows that the MC35 is GPRS capable. This means the MC35 can initiate a GPRS data call and Mobility Management routines apply.

  Example: AT+CGATT=1

  OK

  Note: The GPRS attach may already be done before the PDP context and the QoS profile definition.

- **PDP context activation (AT+CGACT)**

  Before data can be transmitted, a PDP context must be activated. One way to do this is the CGACT command.

  Example: AT+CGACT=1,<cid>

  OK

  Note: More than one context can be activated. If no GPRS attach has been done before, it will be done automatically by the AT+CGACT command.

- **Entering the GPRS data mode (AT+CGDATA)**

In order to set the MC35 in the GPRS data mode and activate the PPP stack, the CGDATA command may be used.

Example: AT+CGDATA=PPP,<cid>

CONNECT

~ }#+!}!}#} }9}"}&} }*} } }'}"}(}"}%}&µ-r#}#}%-#}%*,~~ }#

When the MC35 has answered with CONNECT, it is in PPP mode and no further ATcommands can be sent to the module until the PPP connection has terminated. Thecryptic letter combination displayed after the CONNECT is the terminal interpretation ofthe PPP traffic. For more details about the PPP handshake see also /2/ and /3/.

Note: The activation of the drivers necessary to make a TCP/IP connection has to beinitiated by the OS. It is the user's responsibility to adapt the software accordingly andprovide the appropriate drivers.

7.3 Modem compatible PDP context activation

7.3.1 No preceding PDP context activation

The series of discrete AT commands can be replaced by the ATD command as described in chapter /6.4/.

Example:
```
ATD*99#
CONNECT
~}#+!}!}#} }9}"}&} }*} } }'}"}(}"}%}&µ-r#}#}%-#}%*,~~ }#
```

7.3.2 Preceding PDP context activation

If setting up a call with defined QoS and PDP context parameters, the following AT commands can be used as described in chapter /7.2/.

Example:
```
AT+CGDCONT=1,IP,volume.d2gprs.de
OK
AT+CGQREQ=1,3,4,3,0,0
OK
ATD*99***1#
CONNECT
~}#+!}!}#} }9}"}&} }*} } }'}"}(}"}%}&µ-r#}#}%-#}%*,~~ }#
```

7.4 Shutting down the connection

The GPRS data connection can be shut down in the Dial-Up Network by taping the **Hang up** button. However the PDP context is still alive. It can be stopped with the AT command AT+CGACT.

Example:
```
AT+CGACT=0,<cid>
OK
```

When detaching with the AT+CGATT command, no GPRS data connection is possible before reattaching.

Example:
```
AT+CGATT=0,<cid>
OK
```

Note: The CGATT command also performs a PDP context deactivation, if the context had not been deactivated before.

Another alternative is the ATH command, which closes a data connection, deactivates the PDP context and detaches the module.

Example: ATH

Note: ATH closes all ongoing voice and data connections!

8 GPRS Features

8.1 Mobile Station Class B

The MC35 supports Mobile Station Class B. This means that a voice or data call can be accepted during an ongoing GPRS data transfer without loosing the GPRS PDP context. The implemented and tested behavior can be guaranteed for the NOM II.

When a voice or CSD call is ongoing (dedicated mode) no GPRS call can be activated, no PSD transfer is possible.

ATH ends all ongoing calls (V25.ter)! In order not to stop the GPRS call , the voice or CSD call must be ended with AT+CHLD=1. Use AT+CHLD=0 to reject voice and data calls. This can be done disregarding the SIM Card. Even if multiparty has not been activated, AT+CHLD=(0,1) is available.

Using ATO on multiplexer instances other than the data channel (DLCI=1) leads to an ERROR. No data connection can be made on channels 2, 3, you cannot switch back to data mode.

Examples for using Class B functionality via the serial interface:

1. GPRS call activated, set up voice call
   - GPRS call is activated
   - Use +++ to change to command mode
   - Setup a voice call with ATD....
   - End the voice call with AT+CHLD=1
   - Change back to GPRS data mode with ATO

2. GPRS call activated, set up CSD call
   - GPRS call is activated
   - Use +++ to change to command mode
   - Setup a CSD call with ATD....
   - End the CSD call with AT+CHLD=1
   - Change back to GPRS data mode with ATO

3. GPRS call activated, accept an incoming voice call
   - GPRS call is activated

- Hardware ring indicates incoming voice call
- Use +++ to change to command mode
- URC ring is displayed
- Accept the voice call with ATA or reject with AT+CHLD=0
- If the voice call was accepted, end it with AT+CHLD=1
- Change back to GPRS data mode with ATO

4. GPRS call activated, accept an incoming data call
   - GPRS call is activated
   - Hardware ring indicates incoming data call
   - Use +++ to change to command mode
   - URC ring is displayed
   - Accept the data call with ATA or reject with AT+CHLD=0
   - If the data call was accepted, change to command mode with +++
   - If the data call was accepted, end it with AT+CHLD=1
   - Change back to GPRS data mode with ATO

5. GPRS call activated, data call activated, go to command mode
   - GPRS call is activated
   - Set up data call as described in 2. and 4.
   - Use +++ to change to command mode
   - AT commands can be executed
   - Change back to CSD call with ATO

6. GPRS call activated, activate voice call, go to command mode
   - GPRS call is activated
   - Set up voice call as described in 1. and 3.
   - ATO switches back to GPRS data mode, but no packet transfer takes place; the voice call must be ended first, then the GPRS data transfer can resume

9 APPENDIX: Selected German service providers

9.1 Configuration example: D2-Vodafone

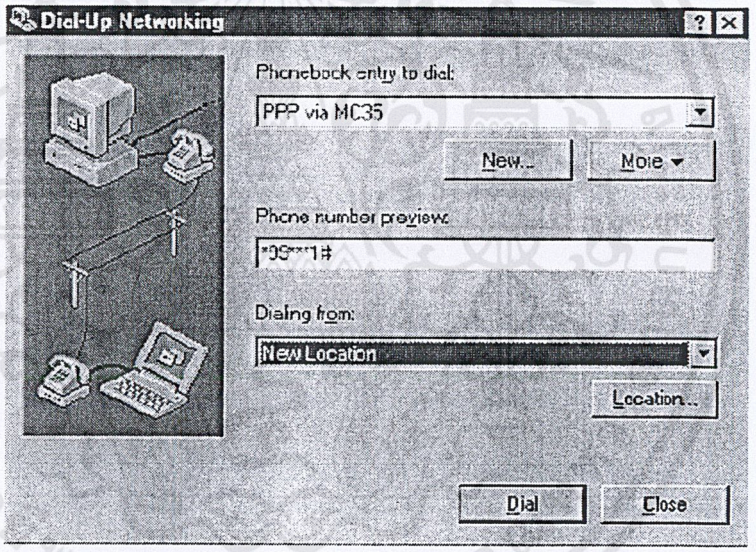
The procedure for connecting to the German D2 network is the same as described in chapter /6.4/. However, the following commands must be entered in the terminal program before a connection can be made over the dial-up network:

Example: AT+CGDCONT=1,PPP,volume.d2gprs.de <RETURN>

AT+CGQREQ=1,3,4,3,0,0 <RETURN>

Note:

Since this configuration uses a predefined context, the dialed phone number must be ***99***1#** (do not use ***99#**).



9.2 GPRS parameters of German service providers

The following table presents GPRS parameters of selected German service providers and operators.

Table 1: Service provider information, valid 16.10.2001

	T-D1	D2 Vodafone	E-Plus	VIAG
Primary DNS	193.254.160.1	139.7.30.125	212.23.97.2	195.182.96.28
Secondary DNS	free	139.7.30.126	212.23.97.3	195.182.96.61
IP address	Automatic	Automatic	Automatic	Automatic
APN	internet.t-d1.de	volume.d2gprs.de	internet.eplus.de	internet
IP header compression	no	no	no	no
Default QoS	Precedence 3; delay 4; reliability 3; peak 0; mean 0	Precedence 3; delay 4; reliability 3; peak 7; mean 31	Precedence 2; delay 4; reliability 3; peak 9; mean 31	Precedence 2; delay 4; reliability 3; peak 4; mean 31

Note:

These settings may be subject to regular changes by the network providers.

ภาคผนวก ง.
จีพีอาร์เอส เอที คอมมานด์ ที่ใช้กับ จีเอสเอ็ม 07.07

จีพีอาร์เอส เอที คอมมานด์ ที่ใช้กับ จีเอสเอ็ม 07.07

1.1 คำสั่งเฉพาะสำหรับ โมดูลไร้สายจีพีอาร์เอส

เป็นการกำหนดคำสั่งที่ให้ อุปกรณ์เทอร์มินอล (TE : Terminal Equipment) โดยจะเรียกว่า ทีอี ให้สามารถควบคุม โมดูลไร้สายจีพีอาร์เอส (GPRS MT : Mobile Termination, the Wireless Module) โดยจะเรียกว่า เอ็มที

ง.1.1 AT+CGATT แนบ และ ปลด บริการจีพีอาร์เอส	
คำสั่งสำหรับทดสอบ AT+CGATT=?	คำสั่งสำหรับทดสอบ ใช้สำหรับการร้องขอข้อมูลจากสถานะที่สนับสนุนบริการจีพีอาร์เอส ผลตอบสนอง **+CGATT: (<state>s)** **OK/ERROR/+CME ERROR** พารามิเตอร์ **<state>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับอ่าน AT+CGATT?	คำสั่งสำหรับอ่านจะคืนสถานะบริการจีพีอาร์เอส ปัจจุบัน ผลตอบสนอง **+CGACT: <state>** **OK/ERROR/+CME ERROR** พารามิเตอร์ **<state>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับเขียน AT+CGATT= [<state>]	คำสั่งประมวลผลจะทำให้ เอ็มที ทำการแนบ หรือ ปลดบริการจีพีอาร์เอส หลังจากคำสั่งสิ้นสุด เอ็มที ยังคงอยู่ในสถานะของคำสั่ง V.25ter (ตามมาตรฐาน V.25ter) ถ้าเอ็มทีอยู่ในสถานะร้องขอ เรียบร้อยแล้ว จะไม่สนใจคำสั่งใดใดแต่จะทำการส่งผลตอบสนอง “ตกลง” กลับมาโดยที่ แอคทีฟ พีดีพี จะเปลี่ยนเป็น ดีแอคทีฟ โดยอัตโนมัติ ในขณะที่สถานะแนบบริการ

	เปลี่ยนเป็น สถานะปลดบริการ ถ้าเอ็มทีไม่สามารถจะแนบบริการได้ภายใน 5 นาที คำสั่งจะส่งค่าผิดพลาด กลับมา ตลอดเวลาที่ เอ็มที พยายามที่จะแนบบริการ พารามิเตอร์ <state> แสดงสถานะแนบบริการจีพีอาร์เอส 0 . ปลด 1 . แนบ ผลตอบสนอง **OK/ERROR/+CME ERROR**
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-1 แสดงการใช้งานเอที คอมมานด์สำหรับแนบและปลดบริการจีพีอาร์เอส

ง.1.2 AT+CGACT พีดีพี คอนเท็กซ์ แอคทีฟ หรือ ดีแอคทีฟ	
คำสั่งสำหรับทดสอบ AT+CGACT=?	คำสั่งสำหรับทดสอบ ใช้สำหรับการร้องขอข้อมูลที่ให้การสนับสนุน สถานะพีดีพี คอนเท็กซ์ ผลตอบสนอง **+CGACT: (<state>s)** **OK/ERROR/+CME ERROR** พารามิเตอร์ <state> ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับอ่าน AT+CGACT?	คำสั่งสำหรับอ่านจะคืนค่าของสถานะ พีดีพี คอนเท็กซ์ ทั้งหมด ผลตอบสนอง **+CGACT: <cid>, <state> [<CR><LF>+CGACT: <cid>, <state>...]** **OK/ERROR/+CME ERROR** พารามิเตอร์

	<cid> ดู คำสั่งสำหรับเขียน **<state>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับเขียน AT+CGACT= [<state>[,<cid>[,<cid> [,...]]]]	คำสั่งนี้ถูกใช้เพื่อระบุว่าเป็น แอคทีฟ หรือ ดีแอคทีฟ ของพีดีพี คอนเท็กซ์ หลังจากคำสั่งสิ้นสุด เอ็มทียังคงอยู่ในสถานะคำสั่ง V.25ter ผลตอบสนอง **OK/ERROR/+CME ERROR** พารามิเตอร์ **<state>** แสดงสถานะของการกระตุ้น พีดีพี คอนเท็กซ์ 0 ไม่ถูกกระตุ้น ดีแอคทีฟ 1 ถูกกระตุ้น แอคทีฟ **<cid>** ตัวระบุ พีดีพี คอนเท็กซ์ เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ . หมายเหตุ : ช่วงที่ได้รับการสนับสนุน cid จะถูกคืนค่าโดยAT+CGDCONT=? ผลตอบสนอง **+CGACT: (<state>s)** **OK/ERROR/+CME ERROR**
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-2 แสดงเอที คอมมานด์สำหรับพีดีพี คอนเท็กซ์ แอคทีฟ หรือ ดีแอคทีฟ

ง.1.3 AT+CGDATA สถานะเข้าสู่ข้อมูล	
คำสั่งสำหรับทดสอบ AT+CGDATA=?	คำสั่งสำหรับทดสอบ ใช้สำหรับการร้องขอข้อมูลที่สนับสนุนโปรโตคอล เลเยอร์ 2 ที่ใช้ระหว่าง ทีอี และ เอ็มที ผลตอบสนอง **+CGDATA: (<L2P>s)** **OK/ERROR/+CME ERROR** พารามิเตอร์

	<L2P> ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับเขียน +CGDATA=[<L2P>, [<cid>[,<cid>,.]]]]	คำสั่งประมวลผลนี้จะส่งผลให้ เอ็มที ทำการสถาปนาการสื่อสารระหว่าง ทีอี และ เครือข่ายที่ใช้ ประเภทจีพีอาร์เอส พีดีพี 1 ประเภทหรือมากกว่า และอาจรวมถึงการแนบบริการจีพีอาร์เอส และกระตุ้น พีดีพี คอนเท็กซ์ โดยคำสั่ง +CGDATA ในเอที คอมมานด์ จะไม่ถูกประมวลผลโดย เอ็มที พารามิเตอร์ **<L2P>** โปรโตคอลเลเยอร์ 2 ที่ถูกใช้ระหว่าง ทีอี และ เอ็มที **<cid>** ตัวระบุ พีดีพี คอนเท็กซ์เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ ถ้าไม่มีคอนเท็กซ์ที่ถูกระบุ ภายในคอนเท็กซ์ 0 จะถูกกำหนดด้วย คุณภาพของบริการและเอพีเอ็น จาก EEPROM 1 2 ผลตอบสนอง ถ้าเป็นผลสำเร็จ เอ็มที จะได้รับ CONNECT และ เข้าไปอยู่ในสถานะ V.25ter ออนไลน์ ข้อมูล : **CONNECT** หลังจากข้อมูลถูกส่งเรียบร้อย และโปรโตคอลเลเยอร์ 2 จบการทำงาน อย่างสมบูรณ์แล้ว สถานะของคำสั่งคือ ทำการเข้าอีกครั้ง และเอ็มที จะคืนค่า : **OK** ถ้าค่าพารามิเตอร์**<L2P>** ไม่ถูกยอมรับจาก เอ็มที, เอ็มทีจะทำการคืนค่าผลตอบสนองเป็น ERROR หรือ +CME ERROR : **ERROR/+CME ERROR** ถ้าในกรณี จบการทำงานผิดพลาด หรือ การเริ่มต้นผิดพลาด สถานะคำสั่งจะเป็น ทำการเข้าใหม่อีกครั้ง และเอ็มที จะคืนค่า **NO CARRIER** หรือ

	+CME ERROR

ตารางที่ ง-3 แสดงเอที คอมมานด์สำหรับสถานะเข้าสู่ข้อมูล

ง.1.4 AT+CGDCONT กำหนด พีดีพี คอนเท็กซ์	
คำสั่งสำหรับทดสอบ AT+CGDCONT=?	คำสั่งสำหรับทดสอบ จะคืนค่าหลายค่า ผลตอบสนอง **+CGDCONT:(<cid>s),<PDP_type>,,,(<d_comp>s),(<h_comp>s)[<CR><LF>+CGDCONT: ...]** **OK/ERROR/+CME ERROR** พารามิเตอร์ **<cid>** ดู คำสั่งสำหรับเขียน **<PDP_type>** ดู คำสั่งสำหรับเขียน **<d_comp>** เป็นพารามิเตอร์เลขจำนวนที่ควบคุมการบีบอัดข้อมูล พีดีพี 0 off **<h_comp>** เป็นพารามิเตอร์เลขจำนวนที่ควบคุมการบีบอัดเฮดเดอร์ พีดีพี 0 off
คำสั่งสำหรับอ่าน AT+CGDCONT?	คำสั่งสำหรับอ่านจะคืนค่าที่กำหนดไว้ในปัจจุบันสำหรับแต่ละคอนเท็กซ์ที่ถูกระบุ ถ้าไม่มีคอนเท็กซ์ที่ถูกระบุ จะคืนค่า **OK** ผลตอบสนอง **+CGDCONT:<cid>,<PDP_type>,<APN>,<PDP_addr>,** **<data_comp>,<head_comp> [<CR><LF>+CGDCONT: ...]** **OK/ERROR/+CME ERROR** พารามิเตอร์ **<cid>** ดู คำสั่งสำหรับเขียน **<PDP_type>** ดู คำสั่งสำหรับเขียน **<APN>** ดู คำสั่งสำหรับเขียน

	<PDP_addr> ดู คำสั่งสำหรับเขียน **<d_comp>** ดู คำสั่งสำหรับทดสอบ **<h_comp>** ดู คำสั่งสำหรับทดสอบ
คำสั่งสำหรับเขียน AT+CGDCONT=[<cid>[,<PDP_type>[,<APN>[,<PDP_addr>]]]]	คำสั่งนี้จะใช้กำหนด ค่าพารามิเตอร์พีดีพี คอนเท็กซ์ สำหรับ พีดีพี คอนเท็กซ์ ที่ถูกระบุ โดย พารามิเตอร์ระบุ คอนเท็กซ์ <cid> รูปแบบพิเศษของชุดคำสั่ง +CGDCONT= <cid> ส่งผลให้ค่าของตัวระบุคอนเท็กซ์<cid>ไม่สามารถถูกกำหนดได้ AT&F และ ATZ จะไม่ถูกกำหนดในทุกคอนเท็กซ์ที่ไม่แอคทีฟ หรือ ไม่ออนไลน์ พารามิเตอร์ **<cid>** ตัวระบุ พีดีพี คอนเท็กซ์เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ ถ้าค่า cid ไม่มีจะไม่มีสิ่งใดเกิดขึ้น 1 2 **<PDP_type>** ประเภทของโปรโตคอลแพ็กเกตข้อมูล เป็นพารามิเตอร์สตริง ที่ระบุประเภทของโปรโตคอลแพ็กเกตข้อมูล : **IP** อินเทอร์เน็ตโปรโตคอล **<APN>** ชื่อแอ็กเซสพอยน์ เป็นพารามิเตอร์สตริง ซึ่งเป็นชื่อท้องถิ่นที่ถูกเลือกโดย จีจีเอสเอ็นหรือเครือข่ายแพ็กเกตข้อมูลด้านนอก ซึ่งไม่สามารถกำหนดค่าเป็น null หรือเว้นไว้ได้ **<PDP_addr>** พารามิเตอร์สตริงที่ระบุที่แอดเดรสของ เอ็มที ผลตอบสนอง **OK/ERROR/+CME ERROR**
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-4 แสดงเอที คอมมานด์สำหรับกำหนด พีดีพี คอนเท็กซ์

ง.1.5 AT+CGQMIN ข้อมูลของคุณภาพของบริการ (น้อยสุดที่สามรถรับได้)	
คำสั่งสำหรับทดสอบ AT+CGQMIN=?	คำสั่งสำหรับทดสอบ จะคืนค่าหลายค่า ผลตอบสนอง

	+CGQMIN:<PDP_type>, (<precedence>s), (<delay>s), (<reliability>s) , (<peak>s), (<mean>s) [<CR><LF>+CGQMIN: ...] **OK/ERROR/+CME ERROR** พารามิเตอร์ **<PDP_type>** ประเภทของโปรโตคอลแพ็กเกตข้อมูล IP **<precedence>** ดู คำสั่งสำหรับเขียน **<delay>** ดู คำสั่งสำหรับเขียน **<reliability>** ดู คำสั่งสำหรับเขียน **<peak>** ดู คำสั่งสำหรับเขียน **<mean>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับอ่าน AT+CGQMIN?	ผลตอบสนอง คำสั่งสำหรับอ่าน จะคืนค่าที่กำหนดปัจจุบันของแต่ละคอนเท็กซ์ที่ถูกระบุ ถ้าไม่มีรายระเอียดขั้นต่ำของคอนเท็กซ์ที่ถูกระบุอย่างชัดเจนจะคืนค่า **OK** แต่ค่าที่ใช้ในคอนเท็กซ์นั้นจะเป็นค่าที่ถูกกำหนดไว้เบื้องต้น +CGQMIN: <cid>, <precedence>, <delay>, <reliability>, <peak>, <mean> [<CR><LF>+CGQMIN: ...] **OK/ERROR/+CME ERROR** พารามิเตอร์ **<cid>** ดู คำสั่งสำหรับเขียน **<precedence>** ดู คำสั่งสำหรับเขียน **<delay>** ดู คำสั่งสำหรับเขียน **<reliability>** ดู คำสั่งสำหรับเขียน **<peak>** ดู คำสั่งสำหรับเขียน **<mean>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับเขียน AT+CGQMIN= [<cid>[,<precedence> [,<delay>[,<reliabilit	คำสั่งนี้จะอนุญาตให้ ทีอี ระบุรายละเอียดขั้นต่ำที่สามารถรับได้ซึ่งจะถูกตรวจสอบโดย เอ็มที ด้วยวิธีการเจรจารายละเอียดชุดคำสั่งนี้จะระบุรายละเอียดสำหรับ คอนเท็กซ์ที่ถูกระบุด้วย cid รูปแบบพิเศษของชุดคำสั่งคือ +CGQMIN= <cid> ส่งผลให้ cid ไม่ถูกกำหนด ในกรณีนี้จะไม่มีการ

y>[,<peak>[,<mean>]]]]]	ตรวจสอบด้วยวิธีการเจรจาราบละเอียด AT&F และ ATZ จะไม่ถูกกำหนดในทุกคอนเท็กซ์ที่ไม่แอคทีฟ หรือ ไม่ออนไลน์ พารามิเตอร์ <cid> > ตัวระบุ พีดีพี คอนเท็กซ์เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ ถ้าค่า cid ไม่มีจะไม่มีสิ่งใดเกิดขึ้น 1 2 <precedence> พารามิเตอร์ตัวเลขสำหรับคลาส พรีซิเดนซ์ 0 ค่าของเครือข่าย 1 สิทธิสูง ข้อตกลงของบริการจะทำการจัดการกับพีซิเดนซ์ คลาส 2 และ 3 2 สิทธิปกติ ข้อตกลงของบริการจะทำการจัดการกับพีซิเดนซ์ คลาส 3 3 สิทธิต่ำ ข้อตกลงของบริการจะทำการจัดการกับพีซิเดนซ์ คลาส 1 และ 2 <delay> พารามิเตอร์ตัวเลขสำหรับคลาส ดีเลย์ 0 ค่าของเครือข่าย ขนาด เอสดียู : 128 ออกเตท: ดีเลย์ คลาส ค่ากลางของการส่งข้อมูล ดีเลย์ 95 เปอร์เซ็นไทล์ ดีเลย์ 1 (Predictive) <0.5 <1.5 2 (Predictive) < 5 < 25 3 (Predictive) < 50 < 250 4 (Best Effort) Unspecified ขนาด SDU : 1024 ออกเตท: ดีเลย์ คลาส ค่ากลางของการส่งข้อมูล ดีเลย์ 95 เปอร์เซ็นไทล์ ดีเลย์ 1 (Predictive) <0.5 <1.5 2 (Predictive) < 5 < 25 3 (Predictive) < 50 < 250

	4 (Best Effort) Unspecified พารามิเตอร์ดีเลย์ จะกำหนดจาก การส่งค่าดีเลย์ แบบจุดต่อจุด ของการส่งค่าของ เอสดียู ตลอดทั้งเครือข่ายจีพีอาร์เอส <reliability> พารามิเตอร์ตัวเลขสำหรับคลาส ความน่าเชื่อถือ 0 ค่าของเครือข่าย 1 ไม่ เรียลไทม์ , รับรู้ได้เร็วเมื่อผิดพลาด, ไม่สามารถจัดการกับข้อมูลที่สูญหายได้ 2 ไม่ เรียลไทม์, รับรู้ได้เร็วเมื่อผิดพลาด, นานๆทีจะสามารถจัดการกับข้อมูลที่สูญหายได้ 3 ไม่ เรียลไทม์, รับรู้ได้เร็วเมื่อผิดพลาด, สามารถจัดการกับข้อมูลที่สูญหายได้ ด้วย จีเอ็มเอ็ม/เอสเอ็ม และ เอสเอ็มเอส 4 เรียลไทม์, รับรู้ได้เร็วเมื่อผิดพลาด, สามารถจัดการกับข้อมูลที่สูญหายได้ 5 เรียลไทม์, รับรู้ได้ช้าเมื่อผิดพลาด, สามารถจัดการกับข้อมูลที่สูญหายได้ <peak> พารามิเตอร์ตัวเลขสำหรับคลาส พีค ทรูพุต 0 ค่าของเครือข่าย พีค ทรูพุต คลาส พีค ทรูพุต (ในรูปของ ออกเตท ต่อวินาที) 1 สูงถึง 1 000 (8 กิโลบิตต่อวินาที) 2 สูงถึง 2 000 (16 กิโลบิตต่อวินาที) 3 สูงถึง 4 000 (32 กิโลบิตต่อวินาที) 4 สูงถึง 8 000 (64 กิโลบิตต่อวินาที) 5 สูงถึง 16 000 (128 กิโลบิตต่อวินาที) 6 สูงถึง 32 000 (256 กิโลบิตต่อวินาที) 7 สูงถึง 64 000 (512 กิโลบิตต่อวินาที) 8 สูงถึง 128 000 (1024 กิโลบิตต่อวินาที) 9 สูงถึง 256 000 (2 048 กิโลบิตต่อวินาที) <mean> พารามิเตอร์ตัวเลขสำหรับ คลาส ค่ากลางของ ทรูพุต 0 ค่าของเครือข่าย

	ค่ากลางของ ทรูพุต ค่ากลางของ ทรูพุต (ในรูปของ ออกเตท ต่อ ชั่วโมง) 1 100 (~0.22 บิตต่อวินาที) 2 200 (~0.44 บิตต่อวินาที) 3 500 (~1.11 บิตต่อวินาที) 4 1 000 (~2.2 บิตต่อวินาที) 5 2 000 (~4.4 บิตต่อวินาที) 6 5 000 (~11.1 บิตต่อวินาที) 7 10 000 (~22 บิตต่อวินาที) 8 20 000 (~44 บิตต่อวินาที) 9 50 000 (~111 บิตต่อวินาที) 10 100 000 (~0.22 กิโลบิตต่อวินาที) 11 200 000 (~0.44 กิโลบิตต่อวินาที) 12 500 000 (~1.11 กิโลบิตต่อวินาที) 13 1 000 000 (~2.2 กิโลบิตต่อวินาที) 14 2 000 000 (~4.4 กิโลบิตต่อวินาที) 15 5 000 000 (~11.1 กิโลบิตต่อวินาที) 16 10 000 000 (~22 กิโลบิตต่อวินาที) 17 20 000 000 (~44 กิโลบิตต่อวินาที) 18 50 000 000 (~111 กิโลบิตต่อวินาที) 31 best effort.
	at+cgqmin? **OK** **at+cgqmin=1,0** **OK** **at+cgqmin?** **+CGQMIN:1,0,0,0,0,0** **OK** **at+cgqmin=1,0,0,0,1** **OK** **at+cgqmin?** **+CGQMIN:1,0,0,0,1,0** **OK**

	at+cgqmin=1,1 **OK** **at+cgqmin?** **+CGQMIN:1,1,0,0,1,0** **OK** ผลตอบสนอง **OK/ERROR/+CME ERROR**
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-5 แสดงเอที คอมมานด์สำหรับกำหนดข้อมูลของคุณภาพของบริการ (น้อยสุดที่สามารถรับได้)

ง.1.6 AT+CGQREQ รายละเอียดคุณภาพของบริการ (ถูกร้องขอ)	
คำสั่งสำหรับทดสอบ AT+CGQREG=?	คำสั่งสำหรับทดสอบ คืนค่าหลายค่า ผลตอบสนอง **+CGQREG:<PDP_type>,(<precedence>s), (<delay>s), (<reliability>s), (<peak>s), (<mean>s) [<CR><LF>+CGQREG: ...]** **OK/ERROR/+CME ERROR** พารามิเตอร์ **<PDP_type>** ประเภทของโปรโตคอลแพ็กเกตข้อมูล IP **<precedence>** ดู คำสั่งสำหรับเขียน **<delay>** ดู คำสั่งสำหรับเขียน **<reliability>** ดู คำสั่งสำหรับเขียน **<peak>** ดู คำสั่งสำหรับเขียน **<mean>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับอ่าน AT+CGQREG?	คำสั่งสำหรับอ่าน จะคืนค่าที่กำหนดปัจจุบันของแต่ละคอนเท็กซ์ที่ถูกระบุ ถ้าไม่มีรายระเอียดขั้นต่ำของคอนเท็กซ์ที่ถูกระบุอย่างชัดเจนจะคืนค่า **OK** แต่ค่าที่ใช้ในคอนเท็กซ์นั้นจะเป็นค่าที่ถูกกำหนดไว้เบื้องต้น

	ผลตอบสนอง **+CGQREG: <cid>, <precedence>, <delay>, <reliability>, <peak>, <mean> [<CR><LF>+CGQREG: ...]** **OK/ERROR/+CME ERROR** **<cid>** ดู คำสั่งสำหรับเขียน **<precedence>** ดู คำสั่งสำหรับเขียน **<delay>** ดู คำสั่งสำหรับเขียน **<reliability>** ดู คำสั่งสำหรับเขียน **<peak>** ดู คำสั่งสำหรับเขียน **<mean>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับเขียน AT+CGQREG= [<cid>[,<precedence> ,<delay>[,<reliability>[,<peak>[,<mean>]]]]]]	คำสั่งนี้จะอนุญาตให้ ทีอี ระบุรายละเอียดคุณภาพของบริการ ที่ถูกใช้ขณะที่เอ็มที ส่ง และกระตุ้น พีดีพี คอนเท็กซ์ ข้อความร้องขอ ไปยังเครือข่าย ชุดคำสั่งนี้จะระบุรายละเอียดสำหรับ คอนเท็กซ์ที่ถูกระบุด้วย cid รูปแบบพิเศษของชุดคำสั่งคือ +CGQREQ= <cid> ส่งผลให้ cid ไม่ถูกกำหนด ในกรณีนี้จะไม่มีการตรวจสอบด้วยวิธีการเจรจาราบละเอียด AT&F และ ATZ จะไม่ถูกกำหนดในทุกคอนเท็กซ์ที่ไม่แอคทีฟ หรือ ไม่ออนไลน์ พารามิเตอร์ **<cid> >** ตัวระบุ พีดีพี คอนเท็กซ์เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ ถ้าค่า cid ไม่มีจะไม่มีสิ่งใดเกิดขึ้น 1 2 **<precedence>** พารามิเตอร์ตัวเลขสำหรับคลาส พรีซิเดนซ์ 0 ค่าของเครือข่าย 1 สิทธิสูง ข้อตกลงของบริการจะทำการจัดการกับพีซิเดนซ์ คลาส 2 และ 3 2 สิทธิปกติ ข้อตกลงของบริการจะทำการจัดการกับพีซิเดนซ์ คลาส 3 3 สิทธิต่ำ

ข้อตกลงของบริการจะทำการจัดการกับพีซิเดนซ์ คลาส 1 และ 2

<delay> พารามิเตอร์ตัวเลขสำหรับคลาส ดีเลย์
0 ค่าของเครือข่าย
ขนาด เอสดียู : 128 ออกเตท:

ดีเลย์ คลาส	ค่ากลางของการส่งข้อมูล ดีเลย์	95 เปอร์เซ็นไทล์ ดีเลย์
1 (Predictive)	<0.5	<1.5
2 (Predictive)	< 5	< 25
3 (Predictive)	< 50	< 250
4 (Best Effort)	Unspecified	

ขนาด เอสดียู : 1024 ออกเตท:

ดีเลย์ คลาส	ค่ากลางของการส่งข้อมูล ดีเลย์	95 เปอร์เซ็นไทล์ ดีเลย์
1 (Predictive)	<0.5	<1.5
2 (Predictive)	< 5	< 25
3 (Predictive)	< 50	< 250
4 (Best Effort)	Unspecified	

พารามิเตอร์ดีเลย์ จะกำหนดจาก การส่งค่าดีเลย์ แบบจุดต่อจุด ของการส่งค่าของเอสดียู ตลอดทั้งเครือข่ายจีพีอาร์เอส

<reliability> พารามิเตอร์ตัวเลขสำหรับคลาส ความน่าเชื่อถือ
0 ค่าของเครือข่าย
1 ไม่ เรียลไทม์ , รับรู้ได้เร็วเมื่อผิดพลาด, ไม่สามารถจัดการกับข้อมูลที่สูญหายได้
2 ไม่ เรียลไทม์, รับรู้ได้เร็วเมื่อผิดพลาด, นานๆทีจะสามารถจัดการกับข้อมูลที่สูญหายได้
3 ไม่ เรียลไทม์, รับรู้ได้เร็วเมื่อผิดพลาด, สามารถจัดการกับข้อมูลที่สูญหายได้ด้วยจีเอ็มเอ็ม/เอสเอ็มและเอสเอ็มเอส
4 เรียลไทม์, รับรู้ได้เร็วเมื่อผิดพลาด, สามารถจัดการกับ

	ข้อมูลที่สูญหายได้ 5 เรียลไทม์, รับรู้ได้ช้าเมื่อผิดพลาด, สามารถจัดการกับข้อมูลที่สูญหายได้
	<peak> พารามิเตอร์ตัวเลขสำหรับคลาส พีค ทรูพุต 0 ค่าของเครือข่าย พีค ทรูพุต คลาส พีค ทรูพุต (ในรูปของ ออกเตท ต่อวินาที) 1 สูงถึง 1 000 (8 กิโลบิตต่อวินาที). 2 สูงถึง 2 000 (16 กิโลบิตต่อวินาที). 3 สูงถึง 4 000 (32 กิโลบิตต่อวินาที). 4 สูงถึง 8 000 (64 กิโลบิตต่อวินาที). 5 สูงถึง 16 000 (128 กิโลบิตต่อวินาที). 6 สูงถึง 32 000 (256 กิโลบิตต่อวินาที). 7 สูงถึง 64 000 (512 กิโลบิตต่อวินาที). 8 สูงถึง 128 000 (1024 กิโลบิตต่อวินาที). 9 สูงถึง 256 000 (2 048 กิโลบิตต่อวินาที).
	<mean> พารามิเตอร์ตัวเลขสำหรับ คลาส ค่ากลางของ ทรูพุต 0 ค่าของเครือข่าย ค่ากลางของ ทรูพุต ค่ากลางของ ทรูพุต (ในรูปของ ออกเตท ต่อ ชั่วโมง) 1 100 (~0.22 บิตต่อวินาที) 2 200 (~0.44 บิตต่อวินาที) 3 500 (~1.11 บิตต่อวินาที) 4 1 000 (~2.2 บิตต่อวินาที) 5 2 000 (~4.4 บิตต่อวินาที) 6 5 000 (~11.1 บิตต่อวินาที) 7 10 000 (~22 บิตต่อวินาที) 8 20 000 (~44 บิตต่อวินาที) 9 50 000 (~111 บิตต่อวินาที) 10 100 000 (~0.22 กิโลบิตต่อวินาที) 11 200 000 (~0.44 กิโลบิตต่อวินาที) 12 500 000 (~1.11 กิโลบิตต่อวินาที) 13 1 000 000 (~2.2 กิโลบิตต่อวินาที)

	14 2 000 000 (~4.4 กิโลบิตต่อวินาที) 15 5 000 000 (~11.1 กิโลบิตต่อวินาที) 16 10 000 000 (~22 กิโลบิตต่อวินาที) 17 20 000 000 (~44 กิโลบิตต่อวินาที) 18 50 000 000 (~111 กิโลบิตต่อวินาที) 31 best effort. **at+cgqreq?** **OK** **at+cgqreq=1,0** **OK** **at+cgqreq?** **+CGQREQ:1,0,0,0,0,0** **OK** **at+cgqreq=1,0,0,1** **OK** **at+cgqreq?** **+CGQREQ:1,0,0,1,0,0** **OK** **at+cgqreq=1,1** **OK** **at+cgqreq?** **+CGQREQ:1,1,0,1,0,0** **OK** ผลตอบสนอง **OK/ERROR/+CME ERROR**
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-6 แสดงเอที คอมมานด์รายละเอียดคุณภาพของบริการ (ถูกร้องขอ)

ง.1.7 AT+CGSMS เลือกบริการสำหรับ ข้อความ เอ็มโอ เอสเอ็มเอส	
คำสั่งสำหรับทดสอบ AT+CGSMS=?	คำสั่งสำหรับทดสอบ รายการของบริการซึ่งถูกเลือกโดย AT+CGSMS คำสั่งสำหรับเขียน. ผลตอบสนอง **+CGSMS: (<service>s)** **OK** พารามิเตอร์ **<service>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับอ่าน AT+CGSMS?	คำสั่งสำหรับอ่าน จะคืนค่า บริการที่ถูกเลือก ผลตอบสนอง **+CGSMS: <service>** **OK/ERROR/+CME ERROR** **<service>** ดู คำสั่งสำหรับเขียน
คำสั่งสำหรับเขียน AT+CGSMS= [<service>]	คำสั่งสำหรับเขียน เพื่อระบุว่าสิ่งใดคือบริการที่ เอ็มที ควรใช้ ขณะกำลังส่งข้อความ เอ็มโอ เอสเอ็มเอส พารามิเตอร์ **<service>** พารามิเตอร์ตัวเลขที่กำหนดบริการที่ถูกใช้ 0 จีพีอาร์เอส 1 เซอร์กิต สวิสต์ 2 จีพีอาร์เอส พรีเฟอร์ (ใช้เซอร์กิต สวิสต์ ถ้าโทรศัพท์เคลื่อนที่ไม่สามารถแนบบริการจีพีอาร์เอสได้) 3 เซอร์กิต สวิสต์ พรีเฟอร์ (ใช้จีพีอาร์เอส เมื่อไม่สามารถใช้เซอร์กิต สวิสต์ได้) ผลตอบสนอง **OK/ERROR/+CME ERROR**
อ้างอิง	

จีเอสเอ็ม 07.07	

ตารางที่ ง-7 แสดงเอที คอมมานด์เลือกบริการสำหรับ ข้อความ เอ็มโอ เอสเอ็มเอส

ง.1.8 AT^SGAUTH กำหนดประเภทของการพิสูจน์ตนสำหรับการเชื่อมต่อด้วยพีพีพี	
คำสั่งสำหรับทดสอบ AT^SGAUTH=?	ผลตอบสนอง **^SGAUTH: (<auth>s)** **OK/ERROR/+CME ERROR** พารามิเตอร์ **<auth>** ระบุประเภทของการรับรองตัวตน 0 None 1 PAP 2 CHAP 3 PAP and CHAP
คำสั่งสำหรับอ่าน AT^SGAUTH?	ผลตอบสนอง **+CGACT: <auth>** **OK/ ERROR/ + CME ERROR** พารามิเตอร์ ดู คำสั่งสำหรับทดสอบ
คำสั่งสำหรับเขียน AT^SGAUTH= <auth>	ผลตอบสนอง **OK/ ERROR/ + CME ERROR** พารามิเตอร์ ดู คำสั่งสำหรับทดสอบ
อ้างอิง Siemens	

ตารางที่ ง-8 แสดงเอที คอมมานด์ กำหนดประเภทของการพิสูจน์ตนสำหรับการเชื่อมต่อด้วยพีพีพี

ง.2 คำสั่งโมเด็มที่เข้ากันกับโมดูลไร้สายจีพีอาร์เอส

ง.2.1 ATD *99# ร้องขอบริการจีพีอาร์เอส	
คำสั่งประมวลผล ATD*99[*[<called_address>] [*[<L2P>][*[<cid>]]]]#	คำสั่งนี้จะทำให้ เอ็มที สถาปนาการเชื่อมต่อการสื่อสารระหว่าง ทีอี และ พีดีเอ็น ด้านนอก คำสั่ง V.25ter 'D' (Dial) จะทำให้เอ็มทีเข้าไปใน สถานะออนไลน์ข้อมูลV.25ter ด้วยทีอี จากนั้นจะกำหนดโปรโตคอลเลเยอร์ 2 การแนบบริการจีพีอาร์เอส และการกระตุ้น พีดีพี คอนเท็กซ์ อาจจะทำก่อน หรือระหว่างที่ พีดีพี เริ่มต้น ผลตอบสนอง ยืนยันการยอมรับคำสั่งที่กำลังเข้าสถานะออนไลน์ข้อมูล V.25ter : **CONNECT** ขณะที่โปรโตคอลเลเยอร์ 2 จบการทำงานเป็นผลให้ปิดกั้น พีดีพี หรือข้อผิดพลาดต่างๆที่จะเข้าไปใน สถานะคำสั่ง V.25ter และคืนค่า : **NO CARRIER** พารามิเตอร์ **<called_address>** ไอพีแอดเดรส **L2P>** โปรโตคอลเลเยอร์ 2 ที่ถูกใช้ระหว่าง ทีอี และ เอ็มที **<cid>** ตัวระบุ พีดีพี คอนเท็กซ์เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ ถ้าไม่มีคอนเท็กซ์ที่ถูกระบุ ภายในคอนเท็กซ์ 0 จะถูกกำหนดด้วย คุณภาพของบริการ และ เอพีเอ็น จาก EEPROM 1 2
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-9 แสดงคำสั่งโมเด็มร้องขอบริการจีพีอาร์เอส

ง.2.2 ATD *98# ร้องขอบริการจีพีอาร์เอส ไอพี	
คำสั่งประมวลผล ATD*98[*<cid>]#	คำสั่งนี้จะทำให้ เอ็มที สถาปนาการเชื่อมต่อการสื่อสารระหว่าง ทีอี และ พีดีเอ็น ด้านนอก คำสั่ง V.25ter 'D' (Dial) จะทำให้เอ็มทีเข้าไปใน สถานะออนไลน์ข้อมูลV.25ter ด้วยทีอี จากนั้นจะกำหนดโปรโตคอลเลเยอร์ 2 การ

	แนบบริการจีพีอาร์เอส และการกระตุ้น พีดีพี คอนเท็กซ์ อาจจะทำก่อน หรือ ระหว่างที่ พีดีพี เริ่มต้น ผลตอบสนอง ยืนยันการยอมรับคำสั่งที่กำลังเข้าสถานะออนไลน์ข้อมูล V.25ter : **CONNECT** ขณะที่โปรโตคอลเลเยอร์ 2 จบการทำงานเป็นผลให้ปิดกั้น พีดีพี หรือ ข้อผิดพลาดต่างๆที่จะเข้าไปใน สถานะคำสั่ง V.25ter และคืนค่า : **NO CARRIER** พารามิเตอร์ **<called_address>** ไอพีแอดเดรส **<L2P>** โปรโตคอลเลเยอร์ 2 ที่ถูกใช้ระหว่าง ทีอี และ เอ็มที **<cid>** ตัวระบุ พีดีพี คอนเท็กซ์เป็นตัวเลขที่ใช้ระบุคำอธิบาย พีดีพี คอนเท็กซ์ ถ้าไม่มีคอนเท็กซ์ที่ถูกระบุ ภายในคอนเท็กซ์ 0 จะถูกกำหนดด้วยคุณภาพของบริการ และเอพีเอ็น จาก EEPROM 1 2
อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-10 แสดงคำสั่งโมเด็มร้องขอบริการจีพีอาร์เอส ไอพี

ง.2.3 ATH การยกเลิกของเครือข่าย	
คำสั่งประมวลผล ATH	ผลตอบสนอง The V.25ter 'H' หรือ 'H0' (On-hook) คำสั่งใช้ยกเลิกเครือข่าย **RING** หรือ **+CRING: GPRS <PDP_type>,<PDP_addr>** ผลตอบสนอง เอ็มที ด้วย **OK**

อ้างอิง จีเอสเอ็ม 07.07	

ตารางที่ ง-11 คำสั่งโมเด็มของการยกเลิกของเครือข่าย

ง.3 การใช้คำสั่งหมุนโมดูลไร้สายจีพีอาร์เอส ด้วย ATD

จีพีอาร์เอส เอที คอมมานด์ จะใช้คำสั่ง "D"ในการเชื่อมต่อเครือข่ายจีพีอาร์เอส โดยมี 2 รหัส บริการจีพีอาร์เอสสำหรับ คำสั่ง ATD: ค่า 98 และ 99

ตัวอย่าง :

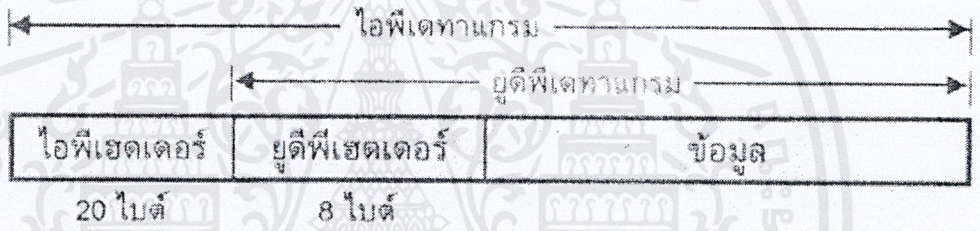
```
ATD*99#
CONNECT       // สถาปนาการเชื่อมต่อโดยใช้รหัสบริการ 99

ATD*99*123.124.125.126*PPP*1#
CONNECT       // สถาปนาการเชื่อมต่อโดยใช้รหัสบริการ 99 , ไอพีแอดเดรส 123.
              //และ L2P = PPP และใช้ CID 1.
              // CID ถูกกำหนดโดย AT+CGDCONT

ATD*99**PPP#
CONNECT       // สถาปนาการเชื่อมต่อโดยใช้รหัสบริการ 99 และ L2P = PPP

ATD*99***1#
CONNECT       // สถาปนาการเชื่อมต่อโดยใช้รหัสบริการ 99 และใช้ CID 1

ATD*99**PPP*1#
CONNECT       // สถาปนาการเชื่อมต่อโดยใช้รหัสบริการ 99 และ L2P = PPP และ
              // ใช้ CID 1 ซึ่งถูกกำหนดโดย AT+CGDCONT

ATD*98#
CONNECT       // สถาปนาการเชื่อมต่อไอพี ด้วยรหัสบริการ 98

ATD*98*1#
CONNECT       // สถาปนาการเชื่อมต่อไอพี ด้วยรหัสบริการ 98 ใช้ CID 1
              // CID ถูกกำหนดโดย AT+CGDCONT
```

ภาคผนวก จ.
โปรโตคอลใน สถาปัตยกรรม ทีซีพี/ไอพี

1.1 โปรโตคอล ยูดีพี

ยูดีพีเป็นโปรโตคอลระดับ ทรานสปอร์ตทำหน้าที่นำส่งข้อมูลจากโปรโตคอลประยุกต์ไปยังไอพี ข้อมูลรวมกับยูดีพีเฮดเดอร์เรียกว่า ยูดีพีเดทาแกรม หรือ ยูสเซอร์เดทาแกรม โดยมีรูปแบบการเอ็นแคปซูเลตดังรูปที่ จ-1

ยูดีพีให้บริการแบบ คอนเนกชั่นเลส กล่าวคือไม่สถาปนาการเชื่อมต่อระหว่างสถานีต้นทางและปลายทาง ยูดีพีส่งเดทาแกรมโดยไม่ตรวจสอบว่าสถานีปลายทางพร้อมที่จะติดต่อหรือไม่ การสื่อสารลักษณะนี้อาจเทียบได้กับการส่งจดหมาย ผู้ส่งเพียงแต่มอบหมายให้ไปรษณีย์จัดส่งโดยไม่ต้องทราบว่าผู้รับปลายทางพร้อมรับหรือไม่



รูปที่ จ-1 การเอ็นแคปซูเลตยูดีพี

หากมีปัญหาเกิดขึ้นกับยูดีพีเดทาแกรม เช่นเดทาแกรมสูญหาย หรือผิดลำดับหรือมีเดทาแกรมซ้ำกัน ยูดีพีจะไม่จัดการกับปัญหาเหล่านี้เนื่องจากไม่มีกลไกที่จะรับประกันความถูกต้องของเดทาแกรม โปรโตคอลประยุกต์ที่ใช้ยูดีพีต้องดำเนินการกับปัญหาเหล่านี้เอง

1.1.1 พอร์ต

หากพิจารณข้อมูลที่ไอพีลำเลียงมาถึงสถานีปลายทางและส่งผ่านตามลำดับชั้นขึ้นมาถึงระดับชั้นทรานสปอร์ต ในไอพีเฮดเดอร์จะระบุชนิดของโปรโตคอลว่าต้องการเรียกใช้บริการทีซีพีหรือยูดีพี ต่อจากนั้นทีซีพีและยูดีพีจะส่งข้อมูลไปยังโปรโตคอลระดับบนตามหมายเลขประจำโปรโตคอล หมายเลขที่ใช้ระบุโปรโตคอลประยุกต์เป็นตัวเลขขนาด 16 บิต และเรียกว่า พอร์ต กล่าวอีกนัยหนึ่งแล้วพอร์ตเป็นเสมือนแอดเดรสประจำโปรโตคอลในชั้นประยุกต์

ทีซีพี/ไอพีสงวนพอร์ตหมายเลข 1 ถึง 1023 ไว้ใช้ประจำโปรโตคอลประยุกต์โดยเรียกเลขพอร์ตนี้ว่า "well-know port" ซึ่งใช้เป็นมาตรฐานทุกสถานีหรืออุปกรณ์เครือข่าย

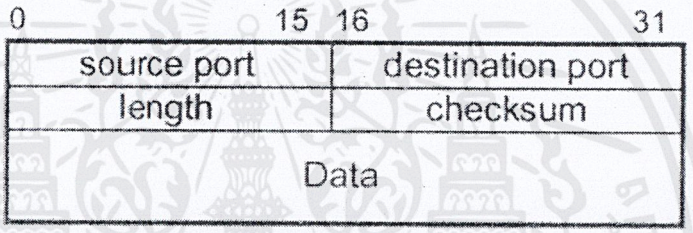
1.1.2 ซ็อกเก็ต

ซ็อกเก็ต หรือ ซ็อกเก็ตแอดเดรส หมายถึงคู่ของไอพีแอดเดรสและเลขพอร์ต ทีซีพีและยูดีพีจะอาศัยซ็อกเก็ตเป็นตัวแยกแยะโปรเซสต้นทางและปลายทางที่ติดต่อระหว่างกัน หากพิจารณาถึงโปรโตคอลประยุกต์ซึ่งอนุญาตให้บริการได้หลายเซสชันพร้อมๆกัน และจำเป็นต้องแยกแยะเซสชันที่เกิดขึ้นให้ได้ เพื่อให้ข้อมูลส่งถ่ายได้ถูกต้องตามคู่ต้นทางและปลายทาง

เซิร์ฟเวอร์ปลายทางจะใช้เลขพอร์ตตามพอร์ตมาตรฐานประจำแอพลิเคชันสำหรับสร้างซ็อกเก็ต แต่ไคลเอ็นต์ต้นทางจะเลือกเลขพอร์ตนอกเหนือจากพอร์ตมาตรฐานมาใช้เป็นซ็อกเก็ตต้นทาง

1.1.3 ยูดีพีเดทาแกรม

รูปที่ 3-12 แสดงถึงยูดีพีเดทาแกรมซึ่งประกอบด้วยฟิลด์ดังต่อไปนี้



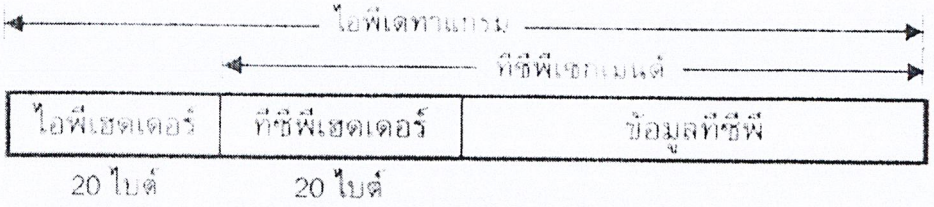
รูปที่ จ-2 ยูดีพีเดทาแกรม

- พอร์ตสถานีต้นทาง : ขนาด 16 บิต
- พอร์ตสถานีปลายทาง : ขนาด 16 บิต
- ความยาวของเดทาแกรม (ทั้งเฮดเดอร์และข้อมูล) เป็นจำนวนไบต์ : ขนาด 16 บิต
- ผลรวมตรวจสอบ คำนวณจากเฮดเดอร์และข้อมูล : ขนาด 16 บิต

ยูดีพีเดทาแกรมมีเฮดเดอร์ที่เรียบง่ายและมีฟิลด์จำนวนน้อย ฟิลด์ที่สำคัญมีเพียงพอร์ตต้นทางและปลายทาง และค่าผลรวมตรวจสอบโดยไม่มีฟิลด์อื่นใดใช้ในการรับประกันความเชื่อถือการส่งข้อมูล

1.2 โปรโตคอล ทีซีพี

ทีซีพีเป็นโปโตคอลที่ให้บริการชนิดที่ต้องมีการเชื่อมต่อ และรับประกันความเชื่อถือในการลำเลียงข้อมูล ทีซีพีรับประกันความเชื่อถือโดยทำหน้าที่ตรวจสอบเซกเมนต์ที่ผิดปกติและจัดส่งแพ็กเกตซ้ำใหม่ รวมทั้งจัดลำดับให้ถูกต้องก่อนส่งไปยังโปรแกรมประยุกต์ระดับบน เฮดเดอร์และข้อมูลของทีซีพีเรียกว่าเซกเมนต์ (segment) การเอ็นแคปซูเลตทีซีพีเซกเมนต์ในไอพีเดทาแกรมแสดงได้รูปที่ จ-3



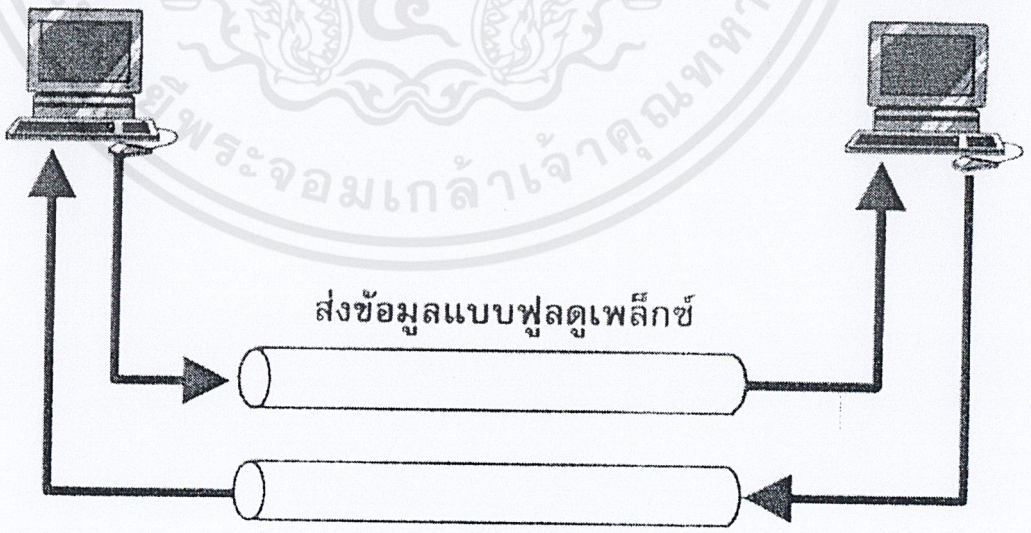
รูปที่ จ-3 การเอนแคปซูเลตทีซีพี

สถานีต้นทางจะต้องสถาปนาการเชื่อมโยงกับสถานีปลายทางก่อนส่งทีซีพีเซกเมนต์ การสถาปนาการเชื่อมโยงใช้ประโยชน์เพื่อให้มั่นใจว่าปลายทางพร้อมจะสื่อสารด้วย และเมื่อสิ้นสุดการส่งถ่ายข้อมูลแล้วก็จะปิดการเชื่อมโยง

การสถาปนาการเชื่อมโยงของทีซีพีอาจเปรียบได้กับการติดต่อทางโทรศัพท์ กล่าวคือเมื่อหมุนหมายเลขปลายทางแล้ว ผู้เรียกต้องรอให้ปลายทางรับสาย เมื่อทักทายและแจ้งให้ทราบว่าใครเป็นผู้เรียกสายแล้วจึงเริ่มการสนทนา กระบวนการสถาปนาของทีซีพีมีขั้นตอนเฉพาะ

ทีซีพีทำงานตามแบบไคลเอ็นต์-เซิร์ฟเวอร์ ไคลเอ็นต์จะเป็นผู้ร้องขอบริการและขอสถาปนาการเชื่อมโยง ส่วนเซิร์ฟเวอร์รับการร้องขอและให้บริการต่อไคลเอ็นต์ การขนถ่ายข้อมูลระหว่างเซิร์ฟเวอร์และไคลเอ็นต์เป็นแบบฟลูดูเพล็กซ์ เสมือนมีท่อลำเลียงสองท่อต่อเชื่อมระหว่างไคลเอ็นต์และเซิร์ฟเวอร์ดังรูปที่ จ-4

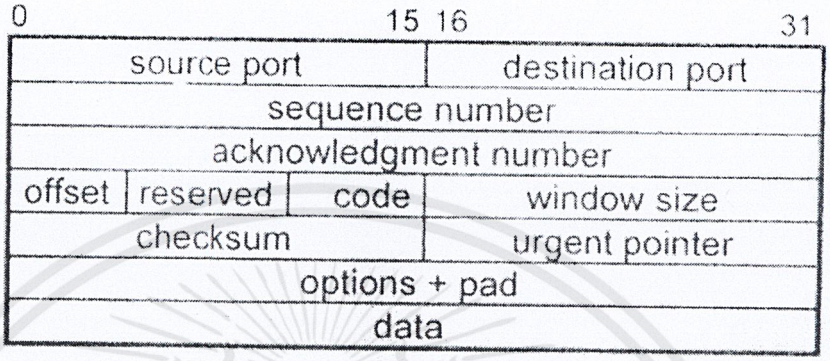
ทีซีพีลำเลียงข้อมูลในรูปของไบต์ขนาด 8 บิต หรือมักเรียกว่าออกเตท (octet) และเรียกบริการลำเลียงแบบนี้ว่า บริการสายข้อมูลแบบไบต์ (byte stream service) โปรโตคอลประยุกต์ปลายทางไม่จำเป็นต้องรับข้อมูลมาดำเนินการครั้งละไบต์เสมอ หากแต่จะดำเนินการไปตามการทำงานของแต่ละโปรโตคอล



รูปที่ จ-4 แบบจำลองการลำเลียงข้อมูลในทีซีพี

1.2.1 ทีซีพีเฮดเดอร์

ทีซีพีเฮดเดอร์ประกอบด้วยฟิลด์จำนวนมากทำหน้าที่ให้บริการตามฟังก์ชัน รูปที่ จ-5 แสดงเฮดเดอร์ของทีซีพี แต่ละฟิลด์มีความหมายดังต่อไปนี้



รูปที่ จ-5 ทีซีพีเฮดเดอร์

- Source port ขนาด 16 บิต : หมายเลขพอร์ตของสถานีต้นทาง
- Destination port ขนาด 16 บิต : หมายเลขพอร์ตของสถานีปลายทาง
- Sequence number ขนาด 32 บิต : ทีซีพีใช้ เลขลำดับ เป็นตัวนับจำนวนไบต์ที่ส่งทุกครั้งที่สถาปนาการเชื่อมโยงทีซีพีจะเลือกเลขลำดับเริ่มต้นสำหรับชี้ตำแหน่งข้อมูลไบต์แรกที่จะจัดส่ง หมายเลขเริ่มต้นไม่จำเป็นต้องเริ่มด้วย 1 แต่อาจเริ่มด้วยค่าใดๆก็ได้ ข้อมูลในเซกเมนต์ถัดไปจะมีเลขลำดับที่สัมพัทธ์เลขลำดับในเซกเมนต์ก่อนหน้า
- Acknowledgement number ขนาด 32 บิต : ค่ากำหนด เลขตอบรับ ซึ่งใช้ตอบกลับไปว่าได้รับข้อมูลแล้ว เลขตอบรับจะมีค่าเท่ากับเลขลำดับประจำเซกเมนต์บวกด้วยจำนวนไบต์ข้อมูลและบวกด้วยหนึ่ง เช่นเซกเมนต์หนึ่งมีเลขลำดับเท่ากลับ 21 และมีข้อมูล 20 ไบต์ เลขตอบรับที่ส่งกลับไปจะเท่ากับ 21+20+1 = 42 ซึ่งแจ้งว่าได้รับข้อมูลตั้งแต่ต้นถึงไบต์ลำดับที่ 41 แล้วและคาดว่าไบต์ถัดไปคือไบต์ที่ 42
- Offset (data offset) ขนาด 4 บิต : บอกถึงตำแหน่งเริ่มต้นของไบต์ข้อมูลหรืออีกนัยหนึ่งใช้บอกขนาดเฮดเดอร์ ตัวเลขนี้มีขนาดเป็นจำนวนเท่าของ 4 ไบต์ เช่นเดียวกับ ที่ใช้ในไอพีเดทาแกรมเฮดเดอร์ของทีซีพีมีความยาวขึ้นกับฟิลด์ option ตัวเลขในฟิลด์ offset จะเท่ากับ 5 ซึ่งเท่ากับ20 ไบต์ (5x4 =20) หากไม่ใช้ออฟชันใด
- Reserve (RSV) ขนาด 4 บิต : สำรองไว้ใช้ในอนาคต
- Code ประกอบด้วย 6 ฟิลด์ย่อย แต่ละฟิลด์ย่อยมีขนาด 1 บิต
- Window size ขนาด 16 บิต : สถานีปลายทางใช้ฟิลด์นี้แจ้งขนาดบัฟเฟอร์ที่มีอยู่ (หน่วยเป็นไบต์) สถานีที่ติดต่อด้วยต้องไม่ส่งข้อมูลเกินค่านี้

- Checksum ขนาด 16 บิต : ผลรวมตรวจสอบความถูกต้องของเซกเมนต์โดยคำนวณทั้งเฮดเดอร์และข้อมูล
- Urgent pointer ขนาด 16 บิต : พอยเตอร์ชี้ตำแหน่งไบต์ข้อมูลที่ต้องดำเนินการเร่งด่วนที่ต้องการให้โปรแกรมประยุกต์ดำเนินการทันที
- Option ขนาดแปรเปลี่ยนได้ : ใช้กำหนดงานเพิ่มเติมให้กับทีซีพีซึ่งจะมีหรือไม่ก็ได้
- Pad ขนาด 0 ถึง 24 บิต : ใช้เป็นส่วนที่ทำให้ขนาดของออฟชันเป็นจำนวนเท่าของ 32 บิต

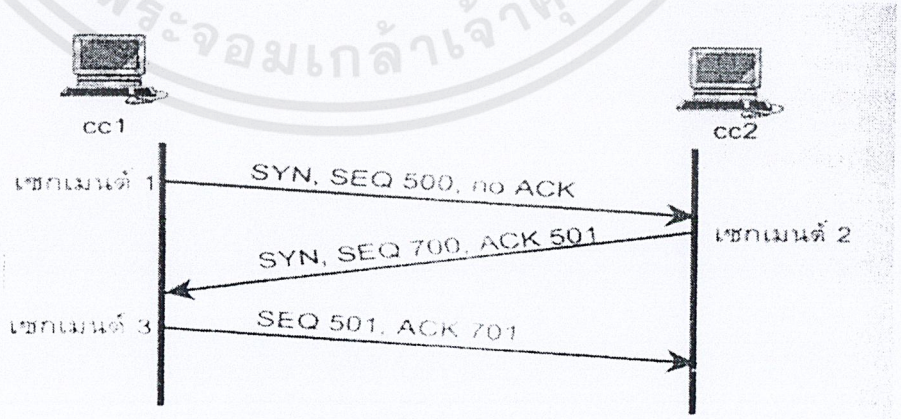
1.2.2 กลไกการทำงานของทีซีพี

1.2.2.1 การสถาปนาการเชื่อมโยง

เพื่อให้เห็นภาพขั้นตอนการทำงานของทีซีพี ขอยกตัวอย่างการสถาปนาการเชื่อมต่อระหว่างสถานีสองเครื่องคือ cc1 และ cc2 โดย cc1 เป็นฝ่ายขอบริการจาก cc2 ดังนั้นโปรโตคอลประยุกต์ด้าน cc2 เป็นเซิร์ฟเวอร์

cc2 จะอยู่ในสภาพที่เรียกว่า การเปิดแบบพาสซีฟ (passive open) คือเซิร์ฟเวอร์สั่งให้ทีซีพีรอรับการเชื่อมต่อ ส่วน cc1 จะเริ่มติดต่อเมื่อไคล์เอ็นท์สั่งงาน การเปิดการเชื่อมต่อจากไคล์เอ็นท์เรียกว่า การเปิดแบบแอคทีฟ (active open) กระบวนการนี้ประกอบด้วยการส่งเซกเมนต์ 3 เซกเมนต์ ดังรูปที่ 3-16 และมีรายละเอียดดังนี้

1. ทีซีพีของ cc1 เลือกเลขลำดับเริ่มต้น (ในที่นี้คือ 500) แล้วส่งเซกเมนต์ที่บรรจุเลขลำดับนี้ไปยัง cc2 พร้อมทั้งเซ็ตแฟล็ก SYN ให้เป็น "1" ขอให้สังเกตว่า เซกเมนต์นี้ (เซกเมนต์ 1) แฟล็ก ACK จะมีค่าเป็น "0"
2. ทีซีพีของ cc2 ได้รับเซกเมนต์จาก cc1 ก็จะเลือกลำดับเลขเริ่มต้นประจำตัวเช่นเดียวกัน (ในที่นี้คือ 700) แล้วตอบกลับด้วยเซกเมนต์ SYN (เซกเมนต์ 2) พร้อมทั้งเซ็ตแฟล็ก ACK เพื่อแจ้งว่าได้รับเซกเมนต์ 1 โดยเลขลำดับที่ได้รับจากทาง cc1 บวกด้วยหนึ่ง (ในที่นี้คือ 501)
3. ทีซีพีของ cc1 จะส่งเซกเมนต์ตอบรับกลับไป (เซกเมนต์ 3) โดยเซ็ตแฟล็ก ACK และใช้เลขลำดับที่ได้รับจาก cc2 บวกด้วยหนึ่ง



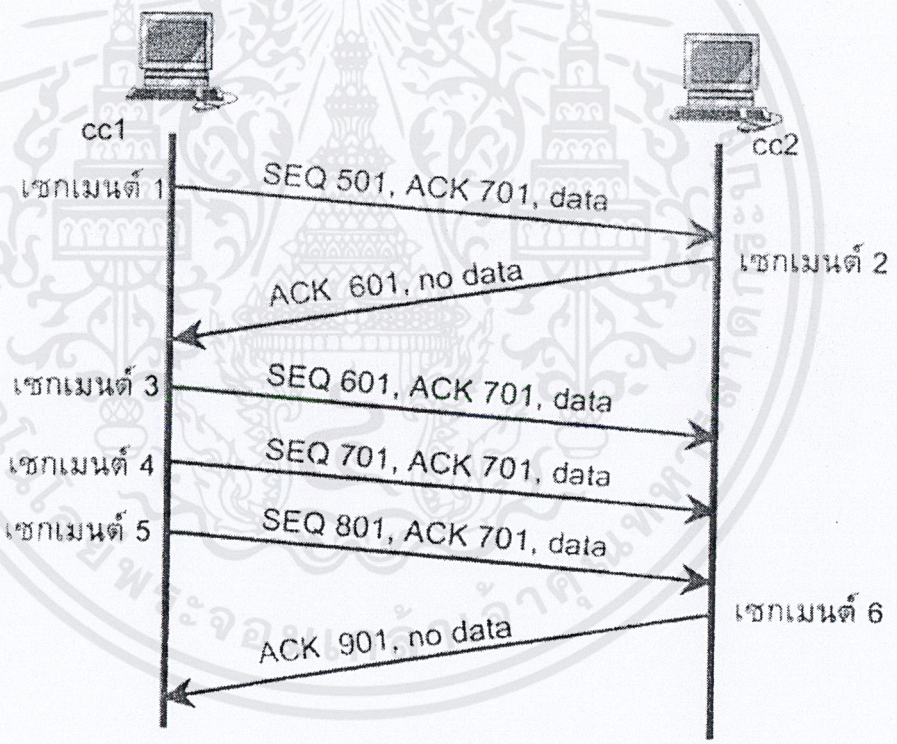
รูปที่ จ-6 การสถาปนาการเชื่อมโยงด้วยทีซีพี

ต่อจากนั้นทีซีพีของ cc1 จะแจ้งไปยังโปรโตคอลระดับบนว่าเชื่อมต่อแล้ว ส่วนทีซีพีของ cc2 เมื่อได้รับเซกเมนต์ตอบรับ (เซกเมนต์ 3) ก็จะแจ้งไปยังโปรโตคอลระดับบนว่าเชื่อมต่อแล้วเช่นกัน เมื่อสิ้นสุดขั้นตอนนี้การสถาปนาทั้งสองด้านก็จะสมบูรณ์และพร้อมจะส่ง ข้อมูลได้ กระบวนการสถาปนาแบบนี้เรียกว่า "ทรี เวย์ แฮนด์เช็ค" เพราะใช้ 3 เซกเมนต์โต้ตอบระหว่างกัน

1.2.2.2 การถ่ายโอนข้อมูล

การขนถ่ายข้อมูลเริ่มได้หลังจากสถาปนาเสร็จสิ้น แล้ว รูปที่ 3-17 แสดงขั้นตอนโดยสมมติให้ cc1 ส่งข้อมูลไปยัง cc2 ครั้งละ 100 ไบต์ จำนวน 4 ครั้ง

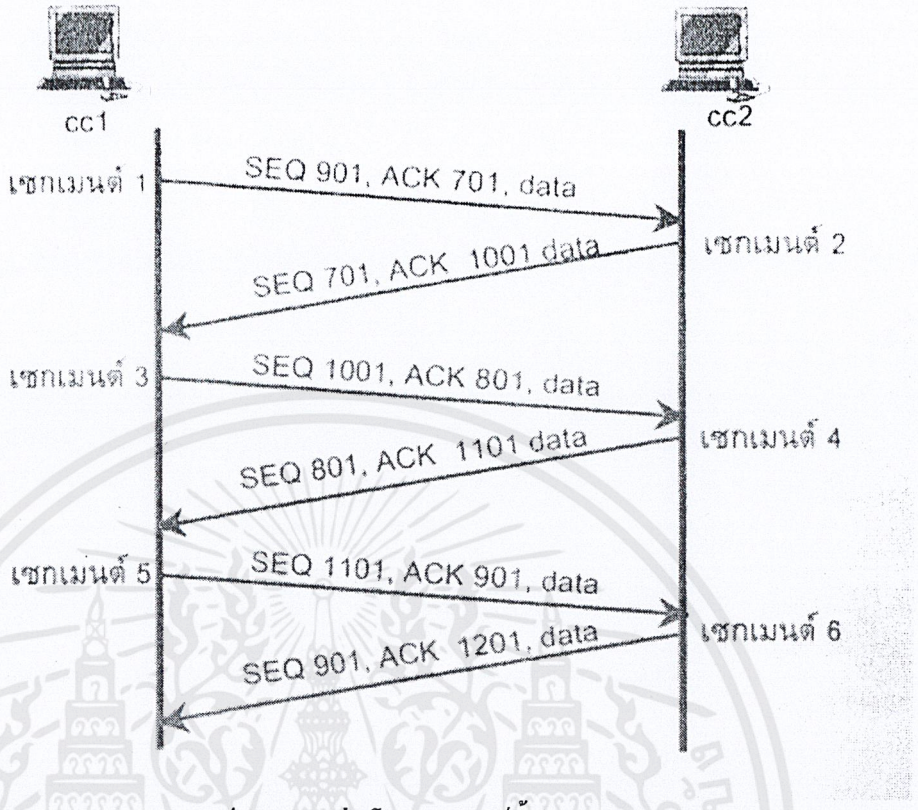
เซกเมนต์แรกที่ cc1 ส่งไปยังcc2 บรรจุข้อมูลไบต์ 501 ถึง 600 และใช้เลขตอบรับ 701 เมื่อ cc2 รับข้อมูลเซกเมนต์ 1 แล้ว ก็จะตอบกลับด้วยเซกเมนต์ 2 พร้อมเลขตอบรับ กำหนดข้อมูลที่คาดว่าจะได้รับต่อไปคือ 601



รูปที่ จ-7 ขั้นตอนการถ่ายโอนข้อมูล

(ก) สามารถส่งเซกเมนต์ไปอย่างต่อเนื่องได้ เช่นในแผนภาพมีการส่ง 2 เซกเมนต์ คือไบต์ 601, 701 และ 801 ทาง cc2 สามารถตอบกลับในคราวเดียวกันได้ (เซกเมนต์ 6)

ในกรณีที่มีการส่งข้อมูลทั้งสองทิศทางระหว่าง cc1 และ cc2 หลักการทั่วไปยังคงเป็นเช่นเดียวกัน เพียงจะมีการตอบรับพร้อมกับส่งข้อมูลระหว่างกันดังรูปที่ 3-18

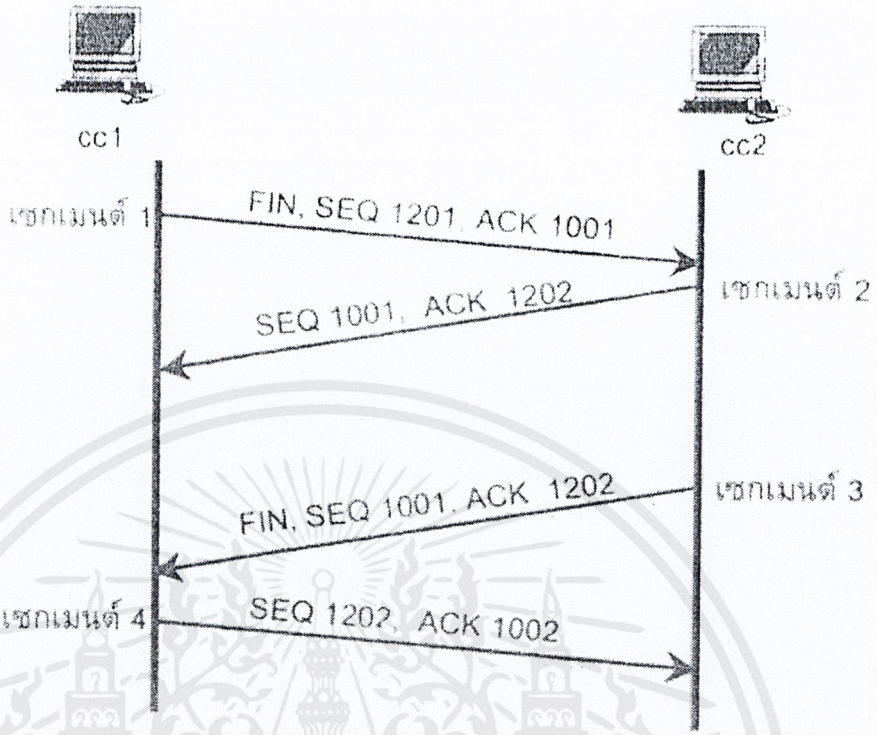


รูป 3-18 การถ่ายโอนเซกเมนต์ทั้งสองทิศทาง

1.2.2.3 การยกเลิกการเชื่อมต่อ

เนื่องจากการถ่ายโอนในทีซีพีเป็นแบบฟลูดูเพล็กซ์ การยกเลิกการเชื่อมต่อจึงต้องมีขั้นตอนเกิดขึ้นทั้งสองด้านดังรูปที่ 3-19 ในที่นี้สมมติว่า cc1 ไม่มีข้อมูลส่งอีกต่อไปจึงต้องเป็นฝ่ายขอปิดการเชื่อมต่อ โปรโตคอลประยุกต์ของ cc1 จะแจ้งไปยังทีซีพีว่าส่งข้อมูลหมดแล้ว ถัดจากนั้นจะมีการแลกเปลี่ยนเซกเมนต์จำนวน 4 เซกเมนต์ดังนี้

1. ทีซีพีของ cc1 ส่งเซกเมนต์พร้อมที่มีเลขลำดับและเลขตอบรับตามปกติแต่เซ็ตแฟลก FIN เป็น "1"
2. เมื่อทีซีพีของ cc2 ได้รับเซกเมนต์ FIN จะส่งเซกเมนต์ตอบรับ (เซกเมนต์ 2) และแจ้งขึ้นไปยังโปรแกรมประยุกต์ว่า cc1 ขอปิดการเชื่อมต่อ โปรแกรมประยุกต์ของ cc2 จะแจ้งกลับมายังทีซีพีว่าปิดการเชื่อมต่อได้
3. ทีซีพี cc2 ส่งเซกเมนต์ FIN ไปยัง cc1 (เซกเมนต์ 3)
4. เมื่อ cc1 ได้รับเซกเมนต์ FIN จะตอบรับกลับไป (เซกเมนต์ 4) และแจ้งไปยัง โปรแกรมประยุกต์ว่าการเชื่อมโยงปิดลงแล้ว

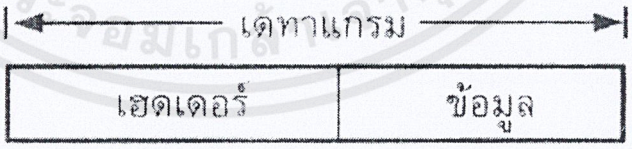


รูปที่ จ-9 การปิดการเชื่อมต่อ

1.3 โปรโตคอลไอพี

ไอพีเป็นโปรโตคอลแกนของทีซีพี/ไอพี โปรโตคอลอื่นไม่ว่าจะเป็นทีซีพีหรือยูดีพีต้องจัดข้อมูลเป็นในรูป เดทาแกรม (datagram) ซึ่งประกอบด้วยเฮดเดอร์ และข้อมูลตามรูป 3-20

หน้าที่หลักของไอพีคือ จัดขนาดของข้อมูลให้พอเหมาะและเลือกเส้นทางที่เหมาะสมเพื่อจัดส่ง เดทาแกรม ไอพีมีรูปแบบการจัดส่งเดทาแกรมเป็นแบบ “ไม่มีความน่าเชื่อถือ” และ “คอนเน็คชันเลส”



รูปที่ จ-10 ฟอร์แมตของเดทาแกรม

ความหมายของ “ ไม่มีความน่าเชื่อถือ ” คือไอพีไม่มีกลไกรับประกันว่าเดทาแกรมที่จะส่งไปถึงปลายทางได้สำเร็จ ไอพีให้บริการลำเลียงเดทาแกรมอย่างดีที่สุด หากมีความผิดปกติได้เกิดขึ้นระหว่างการส่งเดทาแกรม เช่นบัฟเฟอร์ของเราเตอร์ระหว่างทางจนเต็มไม่สามารถรับเดทาแกรมได้ สิ่งที่ไอพี

ดำเนินการกับเดทาแกรมคือ ทิ้งเดทาแกรมนั้นไป แล้วรายงานสาเหตุของปัญหากลับไปด้วยโปรโตคอล ไอซีเอ็มพี

ความหมายของ "คอนเน็คชันเลส" คือไอพีไม่สถาปนาการเชื่อมโยงเพื่อกำหนดเส้นทางการลำเลียงระหว่างต้นทางและปลายทาง ไอพีไม่เก็บสถานะใดๆ ของเดทาแกรมที่ส่งออกไป เดทาแกรมแต่ละชิ้นจึงเป็นอิสระต่อกัน และมีโอกาสไปถึงปลายทางโดยไม่เรียงลำดับ เช่น สถานีต้นทางส่งเดทาแกรม A และB ตามลำดับทั้ง A และ B อาจใช้เส้นทางต่างกันทำให้ B อาจไปถึงปลายทางก่อน Aได้

ผู้อ่านอาจตั้งคำถามว่าปัญหาดังกล่าวจะดำเนินการอย่างไร คำตอบเบื้องต้นในที่นี้คือ ไอพีไม่สนใจต่อปัญหาดังกล่าว แต่จะปล่อยให้เป็นภาระหน้าที่ของโปรโตคอลในระดับชั้นต่อไป ซึ่งได้แก่ ทีซีพีเป็นผู้ดำเนินการ ทั้งนี้เพื่อจำกัดหน้าที่ของไอพีเฉพาะเส้นทางและเชื่อโยงกับโปรโตคอลระดับเดทาลิงค์เท่านั้น

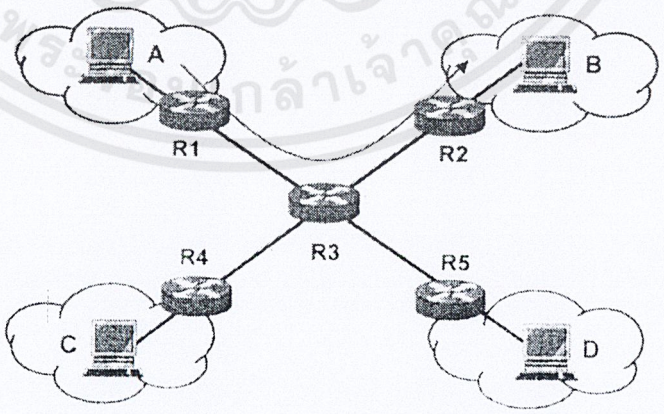
1.3.1 แบบแผนการทำงานของไอพี

อินเทอร์เน็ตประกอบด้วยเครือข่ายที่เชื่อมด้วยเราเตอร์ โฮสต์ในเครื่องข่ายเป็นทั้งสถานีต้นทางและปลายทางสำหรับรับส่งข้อมูล เราเตอร์จะทำหน้าที่เลือกเส้นทางลำเลียงเดทาแกรมไปยังจุดหมายที่กำหนด

พิจารณารูป 3-21 เมื่อโฮสต์ A ต้องการส่งข้อมูลไปยัง B โฮสต์ A จะเอ็นแคปซูเลตข้อมูลมาตามลำดับ เราเตอร์ R1 จะถอนแพ็กเกตเพื่อหาไอพีเดทาแกรม และ ตรวจสอบแอดเดรสปลายทางเพื่อเลือกอินเทอร์เฟสที่จะส่งต่อโดยเอ็นแคปซูเลตเดทาแกรมใหม่เพื่อส่งไปยัง R3

เราเตอร์ R3 จะดำเนินการเช่นเดียวกันและเลือกส่งแพ็กเกตไปยังเราเตอร์ R2 ซึ่งจะตรวจพบโฮสต์ B และจัดส่งเดทาแกรมให้ ถัดจากนั้น โฮสต์ B จะดีแคปซูเลตแพ็กเกตออกตามลำดับเพื่อนำข้อมูลส่งให้แอพลิเคชัน

เส้นทางจากโฮสต์ A ถึง B จะเดินทางผ่านเราเตอร์ 3 ตัว คือ R1, R2 และตามลำดับ เมื่อแพ็กเกตเดินทางผ่านเราเตอร์หนึ่งตัวจะเรียกว่า ผ่านหนึ่งขั้น เราเตอร์ที่อยู่ถัดไปจากเราเตอร์ตัวที่อ้างอิงจะเรียกว่า เราเตอร์ขั้นถัดไป (next-hoop router) เช่น R3 เป็นเราเตอร์ขั้นถัดไปของ R1

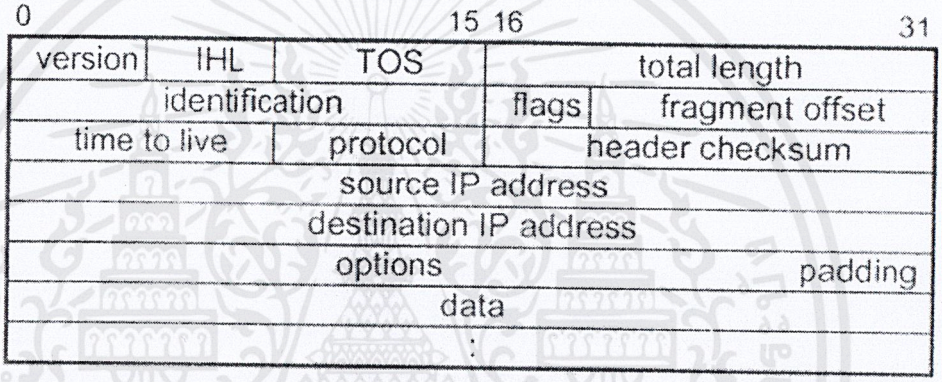


รูป จ-11 การส่งแพ็กเกตจากโฮสต์ต้นทางไปยังปลายทาง

ไอพีดูแลเรื่องที่เกี่ยวกับการจัดแอดเดรส ชนิดการให้บริการ และการจัดแบ่งขนาดข้อมูล (fragmentation) เมื่อพิจารณาถึงแบบอ้างอิงของทีซีพี/ไอพี จะมีโปรโตคอลอื่นที่ทำงานร่วมกับไอพี คือ เออาร์พี , อาร์เออาร์พี , และไอซีเอ็มพี รายละเอียดของโปรโตคอลทั้ง 3 จะกล่าวในหัวข้อต่อไป

1.3.2 ไอพีเดทาแกรม

ไอพีเดทาแกรม มีฟอร์แมตรูปที่ 3-22 และเก็บตัวเลขฐานสองตามแบบ บิ๊กเอ็นเดียน (big endian) เฟรมฟอร์แมตในทีซีพี/ไอพี นิยมเขียนแสดงเป็นแถวๆ ละ 32 บิต เรียงจากซ้ายไปขวาและจากบนลงล่าง ซีพียูบางรุ่นที่เก็บตัวเลขฐานสองตามแบบ ลิตเติ้ลเอ็นเดียน (little endian) จะต้องแปลงลำดับให้ถูกต้องก่อนที่จะส่งข้อมูล



รูปที่ จ-12 ไอพีเดทาแกรม

- version ขนาด 4 บิต : แสดงรุ่นของโปรโตคอล รุ่นที่ใช้งานขณะปัจจุบันมีค่า 4
- Internet Header Length (IHL) ขนาด 4 บิต : บอกความยาวเฉพาะเฮดเดอร์ของเดทาแกรมโดยนับจาก version จนถึงไบต์สุดท้ายก่อนที่จะถึงข้อมูล หน่วยนับความยาวจะบอกเป็นจำนวนเท่าของ 4 ไบต์ (หรือ 32 บิตเวิร์ด) หาก IHL มีค่าเท่ากับ 5 จะส่วนหัวจะมีขนาด 20 ไบต์ ซึ่งเป็นค่าที่บอกว่าไม่มี option และ padding อยู่ในเดทาแกรม
- Type of Service (TOS) ขนาด 8 บิต : ฟิลด์นี้ใช้กำหนดรูปแบบการให้บริการตามลักษณะโปรโตคอลแอพลิเคชัน จะกล่าวในหัวข้อต่อไป
- 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

- Time to Live (TTL) ขนาด 8 บิต : ฟิลด์นี้ใช้กำหนดเราเตอร์ที่เดทาแกรมจะเดินทางผ่านได้หรืออีกนัยหนึ่งคือกำหนดอายุของเดทาแกรมซึ่งมีค่าได้สูงสุดตามขนาดของฟิลด์คือ 2^8-1 หรือ 255 สถานีที่ส่งเดทาแกรมจะตั้งค่า TTL ไว้ที่ค่าใดค่าหนึ่ง เราเตอร์ที่รับเดทาแกรมจะปรับลดค่านี้ลงหนึ่งหน่วย หากลดลงเป็น 0 เราเตอร์จะทิ้งเดทาแกรมนั้นและรายงานกลับไปด้วยไปไอซีเอ็มพี วิธีนี้ช่วยป้องกันปัญหาเดทาแกรมวนรอบ (routing loop) สถานีต้นทางต้องเลือกใช้ค่านี้ให้เหมาะสม เนื่องจากหากมีค่าน้อยไปจะทำให้เดทาแกรมเดินทางไปไม่ถึงปลายทาง หรือหากตั้งไว้มากเกินไปก็จะสร้างภาระให้ระบบเมื่อมีความผิดปกติด้านการเลือกเส้นทาง ค่าโดยปกติที่ใช้ คือ 64
- Protocol ขนาด 8 บิต : ฟิลด์บอกชนิดของโปรโตคอลระดับบนที่เอ็นแคปซูเลตในเดทาแกรม เพื่อให้สถานีปลายทางและสามารถส่งข้อมูลไปยังโปรโตคอลระดับบนได้ถูกต้อง ค่าที่ใช้ปรำจำโปรโตคอลมีดังเช่น

1	ICMP
6	TCP
8	EGP
17	UDP
89	OSPF

- header checksum ขนาด 16 บิต : ใช้ตรวจสอบความผิดพลาดเฉพาะเฮดเดอร์โดยไม่ต้องรวมส่วนข้อมูล การคำนวณผลรวมตรวจสอบจะเริ่มต้นด้วยการให้ฟิลด์ checksum มีค่าเป็น 0 จากนั้นจึงบวกเฮดเดอร์ครั้งละ 16 บิต แบบเติมเต็มหนึ่ง (1's complement) เมื่อได้ผลลัพธ์แล้ว จะนำใส่ในฟิลด์ checksum ไอพีปลายทางเมื่อได้รับเดทาแกรมแล้วก็เพียงแต่บวกเฮดเดอร์ทั้งหมดครั้งละ 16 บิต แบบเติมเต็มหนึ่ง หากค่าที่ได้ไม่เท่ากับศูนย์แสดงว่ามีข้อผิดพลาดในเฮดเดอร์
- source IP address ขนาด 32 บิต : กำหนดไอพีแอดเดรสต้นทาง
- destination IP address ขนาด 32 บิต : กำหนดไอพีแอดเดรสปลายทาง
- option ขนาดไม่คงที่ : ใช้สำหรับข่าวสารเพิ่มเติมสำหรับเดทาแกรม ค่าที่ใช้ในปัจจุบันจะเกี่ยวข้องกับการรักษาความปลอดภัย และการบันทึกผลลัพธ์จากการทำงานของคำสั่ง traceroute หรือ ping ซึ่งจะกล่าวในหัวข้อต่อไป
- padding ขนาด 0 ถึง 3 ไบต์ : ใช้สำหรับผนวกเพิ่มเพื่อให้จำนวนไบต์ของ option รวมกับ padding เป็นจำนวนเท่าของ 32 บิต ค่าในฟิลด์ padding จึงไม่มีความสำคัญใด
- data ขนาดไม่คงที่ : ข้อมูลจากโปรโตคอลระดับบน

1.3.3 การแบ่งขนาดข้อมูล

จากหัวข้อที่ผ่านมากล่าวถึงเทคโนโลยีฮาร์ดแวร์สื่อสารอย่างเช่น อีเทอร์เน็ตหรือโทเค็นริงซึ่งจำกัดขนาดของเฟรมที่จะส่งข้อมูลออกไปได้ เมื่อไอพีรับข้อมูลจากโปรโตคอลระดับบนและเลือกอินเทอร์เฟสที่

จะส่งข้อมูลออกก็จะตรวจเอ็มทียูประจำอินเทอร์เฟส หากพบว่าเดทาแกรมเหมาะกับขนาดเอ็มทียู วิธีการนี้เรียกว่า การแฟรกเมนต์ (fragmentation)

เดทาแกรมที่ถูกแฟรกเมนต์จะมีเฮดเดอร์ประจำตัวเอง โดยมีข้อมูลเพิ่มเติมกำหนดลักษณะของแฟรกเมนต์ประกอบไปด้วย แต่ละชิ้นของแฟรกเมนต์จะเป็นเดทาแกรมที่สมบูรณ์ในตัวเอง เราเตอร์ระหว่างทางที่รับเดทาแกรมจะไม่รวมเดทาแกรม (reassembly) แต่จะปล่อยให้เป็นหน้าที่เราเตอร์ปลายทาง การบังคับรวมเดทาแกรมระหว่างทางจะสร้างปัญหาอย่างมาก 2 ประการ คือ ปัญหาแรกเนื่องจากเดทาแกรมแต่ละชิ้นไม่จำเป็นต้องผ่านเส้นทางเดียวกันหมด หากเราเตอร์ไม่ได้รับทุกเดทาแกรมก็จะไม่สามารถรวมเดทาแกรมให้สมบูรณ์ได้ ปัญหาที่สอง คือ เราเตอร์ต้องรอให้เดทาแกรมทุกชิ้นมาถึงครบ ซึ่งเท่ากับเป็นการหน่วงเวลาเดทาแกรมไว้โดยไม่จำเป็น การแฟรกเมนต์เดทาแกรมของไอพีใช้ฟิลด์ 3 ฟิลด์กำหนดข่าวสารเกี่ยวกับการแฟรกเมนต์ดังนี้คือ

- identification ขนาด 16 บิต : หมายเลขประจำชิ้นเดทาแกรม หากต้องมีการแฟรกเมนต์ก็จะใช้เป็นค่าเพื่อบอกว่าเป็นเดทาแกรมชุดเดียวกัน สถานีที่สร้างเดทาแกรมจะเพิ่มค่านี้ครั้งละหนึ่งให้แต่ละเดทาแกรมแต่ละชุดที่ส่งออกไป
- flags ขนาด 3 บิต : ใช้ควบคุมและแสดงรูปแบบของแฟรกเมนต์ โดยแบ่งออกเป็น 3 บิตคือ
  - บิตแรก ไม่ได้ใช้งานและมีค่าเป็น "0" เสมอ
  - บิต D ใช้กำหนดว่าให้มีการแฟรกเมนต์ได้หรือไม่ หากสถานีต้นทางกำหนดบิตนี้ให้เป็น "0" หมายถึงให้เราเตอร์แฟรกเมนต์เดทาแกรมได้ตามความจำเป็น หากกำหนดให้เป็น "1" หมายถึงบังคับไม่ให้แฟรกเมนต์ ถ้าเราเตอร์ระหว่างทางจำเป็นต้องแฟรกเมนต์ แต่ถูกบังคับไม่ให้แฟรกเมนต์ด้วยบิตนี้ เราเตอร์จะทิ้งเดทาแกรมนั้นไป และแจ้งความผิดพลาดกลับไปยังต้นทางด้วยไอซีเอ็มพี
  - บิต M หากเป็น "0" หมายถึงเป็นเดทาแกรมแฟรกเมนต์สุดท้าย หากเป็น "1" หมายถึงมียังแฟรกเมนต์อื่นตามมาอีก
- fragment offset ขนาด 13 บิต : เราเตอร์จะแฟรกเมนต์ข้อมูลออกเป็นบล็อกแต่ละบล็อกต้องเป็นจำนวนเท่าของ 8 ไบต์ ดังนั้นบล็อกที่เล็กที่สุดคือ 8 ไบต์ (เฉพาะข้อมูลโดยไม่รวมเฮดเดอร์ 20 ไบต์) ฟิลด์นี้ใช้บอกตำแหน่ง offset ของข้อมูลในแต่ละแฟรกเมนต์อ้างอิงจากข้อมูลเดิม หน่วยบอก ตำแหน่งนี้มีขนาดหน่วยละ 8 ไบต์ ด้วย เช่น หาก fragment offset มีค่า 64 หมายถึงเดทาแกรมนั้น บรรจุข้อมูลตำแหน่งที่ 64*8= 512 หรือไบต์ลำดับที่ 512 นับจากต้นข้อมูล

1.3.4 การรีแอสเซมบลี

สถานีปลายทางทำหน้าที่ รีแอสเซมบลี (reassembly) หรือรวมแต่ละเฟรกเมนต์เข้าด้วยกัน แฟรกเมนต์ที่อยู่ในเดทาแกรมชุดเดียวกันต้องมีค่าต่อไปนี้ตรงกันคือ identification, แอดเดรสต้นทาง, แอดเดรสปลายทาง และชนิดโปรโตคอล

ค่า "ความยาวรวม" ในแต่ละแฟรกเมนต์ของเดทาแกรมจะบอกเพียงขนาดเดทาแกรมนั้นหากแต่ละเดทาแกรมมาถึงปลายทางโดยไม่เรียงลำดับ สถานีปลายทางไม่มีทางทราบล่วงหน้าได้ว่าเดทาแกรมทั้งชุดมีขนาดเท่าไร ด้วยเหตุนี้สถานีปลายทางจำเป็นต้องเตรียมบัฟเฟอร์เพื่อการรีแอสเซมบลีให้เพียงพอ

1.4 โปรโตคอล พีพีพี

พอยต์-ทู-พอยต์ โปรโตคอล เป็น มาตรฐาน ในการส่งข้อมูลผ่านสายอนุกรม ที่ต่อกันแบบ พอยต์-ทู-พอยต์ ซึ่งแต่ละจุดปลายทางไม่จำเป็นต้องมีความเร็วในการรับส่งข้อมูลเท่ากันก็ได้ (ข้อมูลที่ส่งเป็นแบบแพ็กเกต)

มี 3 ส่วนประกอบหลัก :

- HDLC (High – Level Data Link Control) โปรโตคอล สำหรับทำการ เอ็นแคปซูเลชันเดทาแกรม เพื่อส่งผ่าน ลิงค์ ไป
- Extensible LCP (Link Control Protocol) ดูแลเรื่องการสถาปนา, กำหนดคุณสมบัติ, ทดสอบการเชื่อมต่อ
- NPCs ใช้ในการสร้าง กำหนดโปรโตคอลเน็ตเวอร์คเลเยอร์ ที่ต่างกัน พีพีพี ถูกออกแบบให้รองรับการใช้หลาย โปรโตคอล (ใน เน็ตเวอร์เลเยอร์ นั้น) ได้

การที่ คอมพิวเตอร์ จะติดต่อรับส่งข้อมูลใดๆผ่านพอร์ตอนุกรม จำเป็นต้องมีตัวควบคุมการส่ง ข้อมูล เพราะอัตราการส่งและรับของแต่ละ อุปกรณ์ อาจไม่เท่ากับ คอมพิวเตอร์ ซึ่งตัวควบคุมนั้นก็คือ พีพีพี

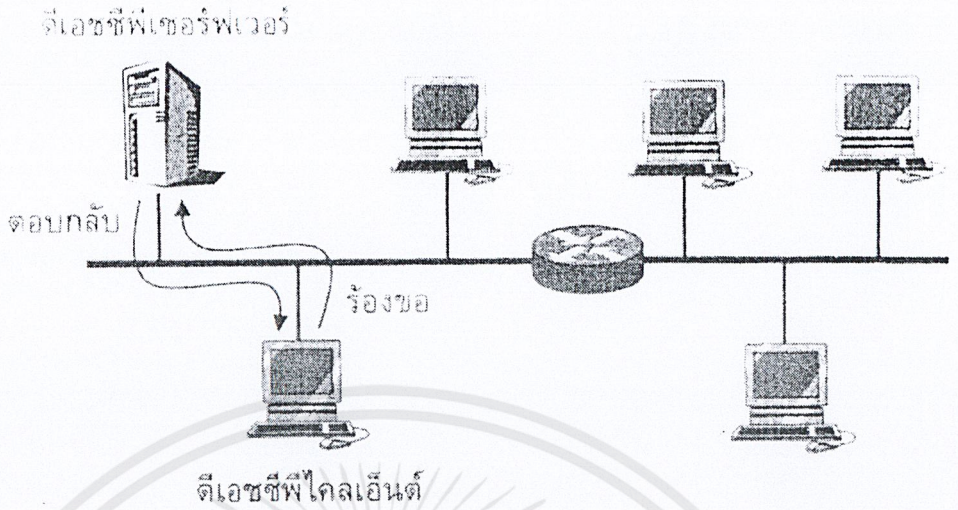
สรุปได้ว่า โทรศัพท์เคลื่อนที่สามารถติดต่อรับส่งข้อมูลกับ คอมพิวเตอร์โดยผ่าน โปรโตคอล พีพีพี และมี ส่วนประกอบ ยูอาร์ต เป็นตัว ควบคุมอัตราการรับส่งข้อมูล

1.5 โปรโตคอล ดีเอชซีพี

ดีเอชซีพี (DHCP : Dynamic Host Configuration Protocol) เป็นโปรโตคอลซึ่งทำหน้าที่กำหนดการติดตั้งค่าแบบไดนามิกให้โฮสต์ในเครือข่าย ดีเอชซีพีทำงานแบบไคลเอ็นท์ –เซิร์ฟเวอร์และมีบริการบางส่วนคล้ายกับที่มีในบูตพี แต่ดีเอชซีพีเซิร์ฟเวอร์ ให้ค่าแบบไม่ตายตัวกับไคลเอ็นท์ที่มาขอบริการ เช่น การให้ไอพีแอดเดรส หรือค่าอื่นประจำไคลเอ็นท์ ดีเอชซีพีสามารถทำงานข้ามเครือข่ายย่อยได้ จึงไม่จำเป็นต้องมีดีเอชซีพีเซิร์ฟเวอร์ให้บริการอยู่ในทุกซับเน็ต

ดีเอชซีพีอำนวยความสะดวกในการตั้งค่าประจำเครื่อง ซึ่งได้แก่การกำหนดไอพีแอดเดรสแบบอัตโนมัติในซับเน็ต หรือกำหนดไอพีเฉพาะสำหรับบางเครื่อง ตลอดจนให้บริการค่าอื่นๆ เช่น ไอพีเกตเวย์ ไอพีของดีเอ็นเอส เป็นต้น การใช้ดีเอชซีพีจึงลดภาระผู้ดูแลระบบในการให้บริการและลดปัญหาการตั้งไอพีแอดเดรสซ้ำซ้อน

เครือข่ายที่ต้องการให้บริการดีเอชซีพีจำเป็นต้องจัดเตรียมเครื่องหนึ่งเครื่องสำหรับใช้เป็นดีเอชซีพีเซิร์ฟเวอร์ และให้เครื่องที่เหลือเป็นดีเอชซีพีไคลเอ็นท์ ติดต่อเข้ามาขอบริการ ดังรูปที่ 3-23



รูปที่ จ-13 ดีเอชซีพี

1.6 โปรโตคอล เอชทีทีพี

เอชทีทีพี เป็นโปรโตคอลกำหนดการส่งข้อมูลระหว่างบราวเซอร์และเวบเซิร์ฟเวอร์ในระบบเวบ และยังใช้เป็นโปรโตคอลสำหรับแอพลิเคชันแบบไคลเอ็นท์-เซิร์ฟเวอร์ใดๆก็ได้ ชื่อเอชทีทีพีอาจสื่อความหมายถึงโปรโตคอลสำหรับการขนถ่ายข้อมูลไฮเปอร์เท็กซ์ ไบนารี เสียง ภาพ หรือข้อมูลอื่นๆในอินเทอร์เน็ต โดยมีขีดความสามารถเฉพาะในการทำงานกับลิงค์แบบไฮเปอร์เท็กซ์ภายใต้ข้อกำหนดในรูปของ ภาษาเอชทีเอ็มแอล (HTML : Hypertext Markup Language)

เอชทีทีพีเป็นโปรโตคอลไคลเอนท์ – เซิร์ฟเวอร์แบบทรานแซคชัน และอาศัยทีซีพีพอร์ต 80 เอชทีทีพีมีลักษณะการทำงานแบบ “ไร้สถานะ” (stateless) แต่ทรานแซคชันจะเป็นอิสระต่อกัน เมื่อผู้ใช้เรียกบราวเซอร์เพื่ออ่านเวบเพจ บราวเซอร์จะทำหน้าที่เป็น เอชทีทีพีไคลเอนท์ ติดต่อกับเอชทีทีพีเซิร์ฟเวอร์ (หรือเรียกว่า เวบเซิร์ฟเวอร์)

เอชทีทีพีไคลเอนท์จะเปิดการเชื่อมต่อด้วยทีซีพีกับเอชทีทีพีเซิร์ฟเวอร์พร้อมทั้งส่งคำสั่งกำหนดชื่อเวบเพจ เซิร์ฟเวอร์จะตอบกลับด้วยรหัสแสดงสถานะและเฮดเดอร์แบบไมม์รวมทั้งข้อมูลเวบเพจ เมื่อไคลเอนท์ได้รับคำตอบแล้วก็จะปิดการเชื่อมต่อ

บรรณานุกรม

[1] William Stallings,Ph.D. : Operating Systems Internals and Design Principles : Prentice-Hall International, Inc. , 2001

[2] Behrouz A. Forouzan : Data Communications and Networking : McGRAW-HILL International , 2001

[3] Andrew S. Tanenbaum : Computer Networks : Prentice-Hall International, Inc. , 1996

[4] Neil Matthew and Richard Stones : Beginning Linux Programming : Wrox Press Ltd, 1999

[5] Ian Sommerville : Software Engineering : Pearson Education Limited, 2001

[6] Jean J. Labrosse : MicroC/OS-II The Real-Time Kernel : Miller Freeman, 1999

[7] Timothy T. Gottleber , Timothy N. Trainor : More excellent HTML with an introduction to JavaScript : McGraw-Hill Higher Education, 2000

[8] สุรศักดิ์ สงวนพงษ์ : สถาปัตยกรรมและโปรโตคอลทีซีพี/ไอพี : ซีเอ็ดยูเคชั่น, 2543.

[9] นุกูล กระจาย : การเขียนโปรแกรมในดอสและวินโดวส์ด้วยบอร์แลนด์C++5.0 : ซีเอ็ดยูเคชั่น, 2540

[10] อภิเนตร อูนากูล : Object-Oriented Analysis and Design : โครงการผลิตตำราของคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2543

[11] ธันวา ศรีประโมง : การเขียนโปรแกรมภาษาซี สำหรับวิศวกรรม : โครงการตำราวิชาการ มหาวิทยาลัยมหานคร, 2538

เว็บไซต์อ้างอิง

http://www.howstuffwork.com/	Knowledge about network
http://www.networkthaionline.com/	Knowledge about network
http://micro.se-ed.com/	Knowledge about network
http://www.mycafecup.com/	Knowledge about network
http://networking.ittoolbox.com/	Knowledge about network
http://www.cisco.com/	Knowledge about network
http://www.sangoma.com/	Knowledge about network
http://www.itpapers.com/	Knowledge about network
http://www.stanford.edu/	Document for network programming
http://www.chipcenter.com/	porting MICROC/OS-II
http://www.protocols.com/	World of GPRS
http://www.radcom-inc.com	World of GPRS
http://www.gsmworld.com/	Technology GPRS
http://www.tdc.co.uk/	Technology GPRS
http://www.ericsson.com/	Technology GPRS
http://www.cellular-news.com/	Technology GPRS
http://www.siemens.com/wm	MC35 Terminal Manual and GPRS at command
http://www.dtac.co.th/	GPRS service
http://www.thaiesf.org/	Com 86 information
http://www.amd.com/	UART AMD186CC
http://docsrv.caldera.com:8457/	How to C++ programming
http://www.embedded.com/	Embedded systems programming
http://www-acc.scu.edu/	Internet Extender
http://ladsoft.tripod.com	PPP code
http://www.micro-webserver.com/	PPP code
http://www.windowsitlibrary.com/	DHCP
http://www.rampnet.com	Configuration by HTTP
http://www.netrack.com/	Configuration by HTTP
http://www.rspa.com/	Adaptable process model