โปรแกรมวิเคราะห์แพ็คเก็ตและตรวจสอบความปลอดภัย บนเครือข่ายแลนแบบไร้สาย

Packet Analyzer and Security Detection System in Wireless Network

โดย

จุติพงศ์ ชูตระกูร

อาจารย์ที่ปรึกษา

อาจารย์ลภัส ประดิษฐ์ทัศนีย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคเรียนที่ 2 ปีการศึกษา 2550

PACKET ANALYZER AND SECURITY DETECTION SYSTEM IN WIRELESS NETWORK

JUTIPONG CHUTRAKUL

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF SCIENCE PROGRAM IN INFORMATION TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2/2007

ใบรับรองปริญญานิพนธ์ ประจำปีการศึกษา 2550

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมวิเคราะห์แพ็คเก็ตและตรวจสอบความปลอดภัยบนเครือข่ายแลนแบบไร้สาย

PACKET ANALYZER AND SECURITY DETECTION SYSTEM IN WIRELESS NETWORK

ผู้จัดทำ

นายจุติพงศ์ ชูตระกูร รหัสประจำตัว 47070128

.....อาจารย์ที่ปรึกษา

(อาจารย์ลภัส ประดิษฐ์ทัศนีย์)

หัวข้อ	โปรแกรมวิเคราะห์แพ็คเก็ตและตรวจสอบความปลอดภัยบนเครือข่ายแลนแบบไร้สาย	
นักศึกษา	นายจุติพงศ์ ชูตระกูร	47070128
ปริญญา	วิทยาศาสตรบัณฑิต	
สาขาวิชา	เทคโนโลยีสารสนเทศ	
แขนงวิชา	เทคโนโลยีสารสนเทศ	
ปีการศึกษา	2550	
อาจารย์ที่ปรึกษา	อาจารย์ลภัส ประดิษฐ์ทัศนีย์	

บทคัดย่อ

โครงงานนี้นำเสนอโปรแกรมวิเคราะห์แพ็คเก็ตและตรวจจับความปลอดภัยบนเครือข่ายแลนแบบไร้สาย โดยการตรวจจับความปลอดภัยนั้นอยู่ภายใต้สมมติฐานของการวิเคราะห์ค่าอัพโหลดดาวน์โหลดและขนาดเฉลี่ยของแพ็คเก็ตแต่ละแพ็คเก็ตของคู่สัญญาณแอ็คเซสพอยท์และเครื่องไคลเอนต์แต่ละคู่สัญญาณ โดยมีการนำไปเปรียบเทียบกับค่ามาตราฐานที่ระบบได้กำหนดไว้ ถ้าคู่สัญญานไหนมีค่าอัพโหลดหรือดาวน์โหลดหรือขนาดเฉลี่ยของแพ็คเก็ตที่ผิดปกติ ระบบก็สามารถตรวจสอบและรับรู้ค่าที่ผิดปกติของคู่สัญญานนั้นได้ โดยโครงงานที่เราพัฒนาขึ้นนั้นสามารถที่จะนำไปพัฒนาต่อเพื่อที่จะนำไปสู่ระบบที่สามารถตรวจจับความปลอดภัยหรือวิเคราะห์ค่าต่างๆ ของแพ็คเก็ตได้หลากหลายยิ่งขึ้น

Title	Packet Analyzer and Security Detection System in Wireless Network
Student	Mr. Jutipong Chutrakul 47070128
Degree	Bachelor of Science
Programme	Information Technology
Academic Year	2007
Advisor	Lapas Pradittasnee

ABSTRACT

The project present Program Packet Analyzer and Security Detection System in Wireless Network. This program will make a security detection followed by the hypothesis of uploading, downloading data. Moreover, the program will calculate the average size of packets in each pair of signal between client and access point to make a security detection too. The program will compare an obtained value with the standard. If the calcuated value of one signal is weird, we can track what signal is abnormal. This project is designed to be developed to improve the system for increasing the capability of security detection and analyzing packet in vary way.

กิตติกรรมประกาศ

โครงงานพัฒนาโปรแกรมวิเคราะห์แพ็คเก็ตและตรวจสอบความปลอดภัยบนเครือข่ายแลนแบบไร้สายสำเร็จลุล่วงไปได้ด้วยดีด้วยคำแนะนำ และคำปรึกษาที่เป็นประโยชน์จาก อาจารย์ ลภัส ประดิษฐ์ทัศนีย์ ซึ่งเป็นอาจารย์ที่ปรึกษาโครงงาน ข้าพเจ้าขอขอบคุณอาจารย์เป็นอย่างสูงเปิดโอกาสและสละเวลาให้คำแนะนำ คำปรึกษา และรับฟังความคิดเห็นเป็นอย่างดี

ขอขอบคุณคณาจารย์ คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่านที่ได้ประสิทธิ์ประสาทวิชาให้กับข้าพเจ้าในการนำความรู้มาประยุกต์ใช้ในโครงงานและการประกอบอาชีพต่อไป ขอขอบคุณเพื่อนๆ พี่ๆ ทุกคนที่คอยให้ความช่วยเหลือ ให้คำแนะนำต่างๆ เสมอมา

สุดท้ายนี้ข้าพเจ้าขอขอบพระคุณบิดา มารดาและครอบครัว ที่เป็นกำลังใจและให้การสนับสนุนในทุกๆ เรื่องตลอดมา

จุติพงศ์ ชูตระกูร

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

รูปที่ หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันเทคโนโลยีเครือข่ายแลนแบบไร้สายหรือ WLAN (Wireless LAN) กำลังได้รับความนิยมเป็นอย่างมาก เนื่องจากประโยชน์ของเครือข่ายแลนแบบไร้สายมีอยู่มากมายโดยเฉพาะอย่างยิ่งสามารถสร้างความสะดวกสบายในการใช้งานและการติดตั้งอุปกรณ์ก็สามารถทำได้โดยง่าย แต่อย่างไรก็ตามความง่ายและสะดวกต่อการติดตั้งเพื่อใช้งานนั้นก็นำมาซึ่งความไม่ปลอดภัยของเครือข่ายด้วยเช่นกัน เพราะสัญญาณไวร์เลสสามารถสร้างออกมาได้มากมายหลายรูปแบบและตัวกลางที่เครือข่ายแลนแบบไร้สายส่งผ่านนั้นก็คืออากาศทำให้ง่ายต่อการบุกรุกและแทรกแซงเข้ามาเพื่อทำให้เกิดปัญหา ดังนั้นจึงทำให้เกิดโครงงานนี้ขึ้นมาโดยจะเขียนโปรแกรมบนระบบปฏิบัติการลินุกซ์เพื่อตรวจจับสัญญานไวร์เลสและจะนำสัญญานที่ตรวจจับได้มาวิเคราะห์ว่าสามารถบ่งบอกอะไรได้บ้างจากรูปแบบของสัญญาณที่มี โดยการวิเคราะห์นี้จะสามารถช่วยในแง่ของการรักษาความปลอดภัยหรือตรวจจับและป้องกันไม่ให้ผู้ที่ไม่ได้รับอนุญาต มีสิทธิ์ที่จะเข้ามาใช้งานภายในเครือข่ายผ่านทางอุปกรณ์ไวร์เลสได้ โดยเราจะพิจารณาจากรูปแบบของแพ็คเก็ตที่ผิดปกติ โดยเราสามารถนำสัญญาณที่ตรวจจับได้นั้นมาวิเคราะห์เพื่อตรวจสอบและหาวิธีแก้ไขปัญหาที่เกิดขึ้นต่อไป

1.1 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. เพื่อศึกษารูปแบบของเฟรมข้อมูลที่รับส่งในครือข่ายแลนแบบไร้สายภายใต้มาตรฐาน IEEE 802.11
2. เพื่อพัฒนาโปรแกรมที่ทำงานบนระบบปฏิบัติการลินุกซ์ให้สามารถดักจับแพ็คเก็ตต่างๆ ที่มีในระบบเครือข่ายไร้สายได้อย่างถูกต้อง
3. เพื่อพัฒนาโปรแกรม Packet Analyzer ขึ้นมาเพื่อแสดงผลของแพ็คแก็ตรูปแบบต่าง ๆ ที่ดักจับมาเพื่อให้สามารถนำมาวิเคาระห์ต่อไปได้
4. เพื่อพัฒนาระบบตรวจสอบความปลอดภัยบนระบบเครือข่ายแลนแบบไร้สาย โดยอาศัยข้อมูลจากแพ็คเก็ตที่ดักจับได้

1.2 ขอบเขตของงาน

1. พัฒนาโปรแกรมบนระบบปฏิบัติการลินุกซ์ที่สามารถดักจับและจัดเก็บข้อมูลของแพ็คเก็ตต่างๆ ที่เกิดขึ้นในระบบเครือข่ายแลนแบบไร้สายได้ครบทุกรูปแบบ
2. พัฒนาโปรแกรม Packet Analyzer เพื่อทำวิเคราะห์ข้อมูลสัญญาไวร์เลสที่ดักจับได้
3. โปรแกรมสามารถบันทึกไฟล์ข้อมูลเก็บไว้เพื่อนำมาดูย้อนหลังได้
4. พัฒนาระบบความปลอดภัยบนระบบเครือข่ายแลนแบบไร้สาย

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1. ทำให้ผู้พัฒนาเข้าใจในวิธีการเขียนโปรแกรมดักจับแพ็คเก็ตในมาตรฐาน IEEE 802.11 บนระบบปฏิบัติการลินุกซ์
2. ทำให้ผู้พัฒนาเข้าใจและพัฒนาโปรแกรมที่สามารถตรวจสอบความปลอดภัยบนเครือข่ายแลนแบบไร้สายได้
3. โปรแกรมที่พัฒนาขึ้นสามารถดักจับและจัดเก็บแพ็คเก็ตได้อย่างถูกต้อง
4. สามารถนำผลที่ได้จากการวิเคราะห์มาหาวิธีแก้ไขปัญหาที่อาจเกิดขึ้นที่เกิดจากการใช้งานเครือข่ายแลนแบบไร้สาย

บทที่ 2
เครือข่ายแลนแบบไร้สายและลินุกซ์

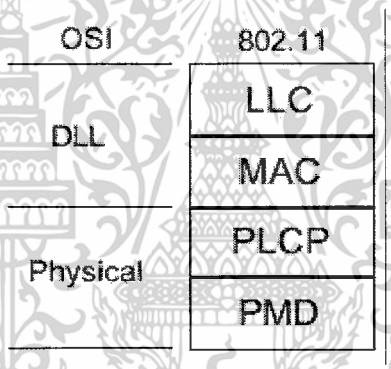
2.1 มาตรฐานของ IEEE 802.11

เครือข่ายไร้สาย (Wireless Lan) จะใช้มาตรฐานแบบ IEEE 802.11 ซึ่งเป็นหนึ่งในสมาชิกของมาตรฐาน IEEE 802 ซึ่งเป็นมาตรฐานที่กำหนดรายละเอียดที่เกี่ยวกับเครือข่ายแลนและได้พัฒนาเทคโนโลยีการเชื่อมต่อแบบไร้สายกันมาอย่างต่อเนื่อง ซึ่งเราจะสังเกตได้จากตัวอักษรที่ต่อท้ายจาก IEEE 802.11 ที่จะแบ่งเทคโนโลยีที่ถูกพัฒนาขึ้น สิ่งที่แตกต่างอย่างเห็นได้ชัดในแต่ละเทคโนโลยีนั่นก็คือความเร็วในการรับส่งข้อมูล

- **มาตรฐาน IEEE 802.11b** มาตรฐานนี้เริ่มต้นใช้งานเมื่อประมาณปี 2542 เป็นมาตรฐานที่คนทั่วโลกนิยมใช้กันมากที่สุด โดยมาตรฐาน IEEE 802.11b สามารถรองรับการรับส่งข้อมูลได้ 11 Mbps และได้ใช้เทคโนโลยีแบบ CCK (Complimentary Code Keying) บวกกับ DSSS (Direct Sequence Spread Spectrum) ผ่านคลื่นความถี่วิทยุ 2.4 GHz (เป็นย่านความถี่ที่เรียกว่า ISM (Industrial Scientific and Medical) ซึ่งถูกจัดสรรไว้อย่างสากลสำหรับการใช้งานอย่างสาธารณะด้านวิทยาศาสตร์ อุตสาหกรรม และการแพทย์ โดยอุปกรณ์ที่ใช้ความถี่ย่านนี้ก็เช่น IEEE 802.11 Bluetooth โทรศัพท์ไร้สาย และเตาไมโครเวฟ ส่วนใหญ่แล้วอุปกรณ์ IEEE 802.11 WLAN ที่ใช้กันอยู่ในปัจจุบันจะเป็นอุปกรณ์ตามมาตรฐาน IEEE 802.11b นี้และใช้เครื่องหมายการค้าที่รู้จักกันดีในนาม Wi-Fi ซึ่งเครื่องหมายการค้าดังกล่าวถูกกำหนดขึ้นโดยสมาคม WECA (Wireless Ethernet Compatability Alliance) โดยอุปกรณ์ที่ได้รับเครื่องหมายการค้าดังกล่าวได้ผ่านการตรวจสอบแล้วว่าเป็นไปตามมาตรฐาน IEEE 802.11b และสามารถนำไปใช้งานร่วมกับอุปกรณ์ยี่ห้ออื่นๆ ที่ได้รับเครื่องหมาย Wi-Fi ได้
- **มาตรฐาน IEEE 802.11a** มาตรฐาน IEEE 802.11a ได้ถูกปรับปรุงให้ใช้คลื่นความถี่วิทยุ 5 GHz และปรับปรุงให้สามารถรองรับการรับส่งข้อมูลรวดเร็วมากขึ้นถึง 54 Mbps โดยใช้เทคโนโลยี OFDM (Orthogonal Frequency Division Multiplexing) สำหรับคลื่นความถี่ที่นำมาใช้กับ IEEE 802.11a นั้นจะเป็นคลื่นความถี่สาธารณะของประเทศสหรัฐอเมริกาเนื่องจากจะมีสัญญาณอื่นรบกวนน้อยกว่า 2.4 GHz ที่ใช้ในแบบ IEE 802.11b แต่การกระจายสัญญาณของมาตรฐาน IEEE 802.11a จะกระจายสัญญาณได้สั้นกว่า มาตรฐานแบบ IEE 802.11a นี้จะมีความนิยมใช้กันน้อยในบางประเทศ อันเนื่องมาจากคลื่นความถี่ 5 GHz ได้ถูกจัดสรรเอาไว้ใช้งานประเภทอื่นแล้ว

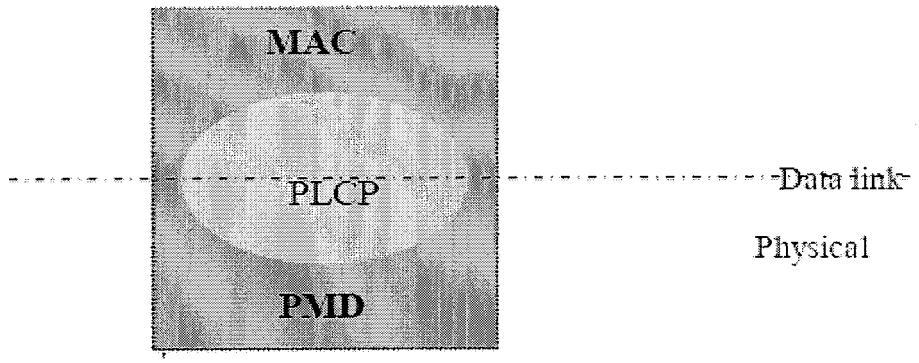
- **มาตรฐาน IEEE 802.11g** มาตรฐาน IEEE 802.11g ถูกสร้างขึ้นมาเพื่อให้สามารถใช้งานร่วมกับอุปกรณ์ที่เป็นมาตรฐานแบบ IEEE 802.11b ร่วมกันได้ เราอาจจะเห็นอุปกรณ์ชนิด เช่น โน้ตบุ๊ค PDA จะรองรับการใช้งานเครือข่ายไร้สายแบบ 802.11b/802.11g ในด้านการใช้ช่องสัญญาณจะใช้คลื่นความถี่ 2.4 GHz เหมือนกับ IEEE 802.11b แต่จะสามารถรับส่งข้อมูลได้รวดเร็วขึ้นกว่า คือ 54 Mbps สำหรับเทคโนโลยีที่นำมาใช้นั้นจะเป็นแบบ OFDM (Orthogonal Frequency Division Multiplexing)

- **มาตรฐาน IEEE 802.11n** มาตรฐาน IEEE 802.11n เป็นมาตรฐานใหม่ของเครือข่ายไร้สาย (Wireless Lan) ซึ่งคาดว่าจะเข้ามาใช้งานแทนที่ มาตรฐาน IEEE 802.11a , IEEE 802.11b และ IEEE 802.11g ที่ใช้งานอยู่ในปัจจุบัน สำหรับมาตรฐานใหม่ของเครือข่ายไร้สายนี้จะมีความเร็วในการรับส่งข้อมูลสูงถึง 320 Mbps และเพิ่มความแรงของสัญญาณให้สามารถส่งได้ไกลขึ้นมากกว่าเดิม



รูปที่ 2.1 แสดงคุณลักษณะของ IEEE 802.11

โดย IEEE 802.11 การส่งและการรับข้อมูลนั้นจะเป็นหน้าที่ของระดับชั้นฟิสิคัล (Physical layer) การที่ในระดับชั้นฟิสิคัลของเครือข่ายไร้สายใช้คลื่นวิทยุเป็นตัวกลางในการรับส่งข้อมูล 802.11 จึงได้ทำการแบ่งฟิสิคัลคอมโพเนนท์ (Physical component) ออกเป็นสองชนิดคือ Physical Layer Convergence Procedure (PLCP) ซึ่งจะใช้สำหรับแปลงฟอร์แมตของข้อมูลผู้ใช้ให้สามารถส่งไปบนระบบรับส่งสัญญาณได้และคอมโพเนนท์อีกตัวคือ Physical Medium Dependent (PMD) ซึ่งใช้สำหรับรับส่งข้อมูล ซึ่งจะขึ้นอยู่กับเทคโนโลยีที่ใช้ และในส่วนของระดับชั้นดาต้าลิงค์ (Data Link layer) ซึ่งในระดับชั้นดาต้าลิงค์จะมีแมค (MAC: Media Access Control) เป็นตัวควบคุมการเข้าถึงตัวกลางและการส่งข้อมูลและแอลแอลซี (LLC: Logical Link Control) จะสร้างและดูแลการเชื่อมต่อระหว่างอุปกรณ์



รูปที่ 2.2 แสดงองค์ประกอบฟิสิคัล

2.2 องค์ประกอบและรูปแบบการเชื่อมต่อต่าง ๆ ของมาตรฐาน IEEE 802.11

2.2.1 องค์ประกอบของเครือข่าย 802.11

ประกอบด้วย 4 องค์ประกอบหลัก คือ

1. ระบบกระจาย (Distribution system) จะให้บริการโดยการเชื่อมต่อแอคเซสพอยต์ (access point) ต่างๆ เข้าด้วยกัน เมื่อมีเฟรมข้อมูลมาถึงระบบกระจายจะทำการส่งเฟรมนั้นไปยังแอคเซสพอยต์ที่ดูแลสถานี (station) ปลายทางอยู่และแอคเซสพอยต์ปลายทางจะทำการส่งเฟรมข้อมูลที่ได้รับมาไปให้สถานีปลายทาง หน้าที่ของระบบกระจายคือการตรวจสอบว่าสถานีอยู่ที่ใด เพื่อที่จะทำการส่งเฟรมไปให้ได้อย่างถูกต้อง
2. แอคเซสพอยต์ (access point) การที่เครือข่ายไร้สายจะสามารถติดต่อกับเครือข่ายอื่นๆ ได้นั้น ต้องทำการแปลงเฟรมข้อมูลในรูปแบบ 802.11 ให้กลายเป็นเฟรมข้อมูลรูปแบบอื่นที่เหมาะสมในการสื่อสารกับเครือข่ายนั้นๆ โดยแอคเซสพอยต์จะมีหน้าที่หลักในการแปลงเฟรมข้อมูลนี้
3. ตัวกลางไร้สาย (Wireless medium) IEEE 802.11 สามารถรองรับระดับชั้นฟิสิคัล (Physical layer) ได้หลายรูปแบบแต่ที่รู้จักและนิยมโดยทั่วไปคือ คลื่นความถี่วิทยุ (RF)
4. สถานี (station) อุปกรณ์ลูกข่ายโดยจะต้องติดตั้งส่วนเชื่อมต่อกับเครือข่ายไร้สาย

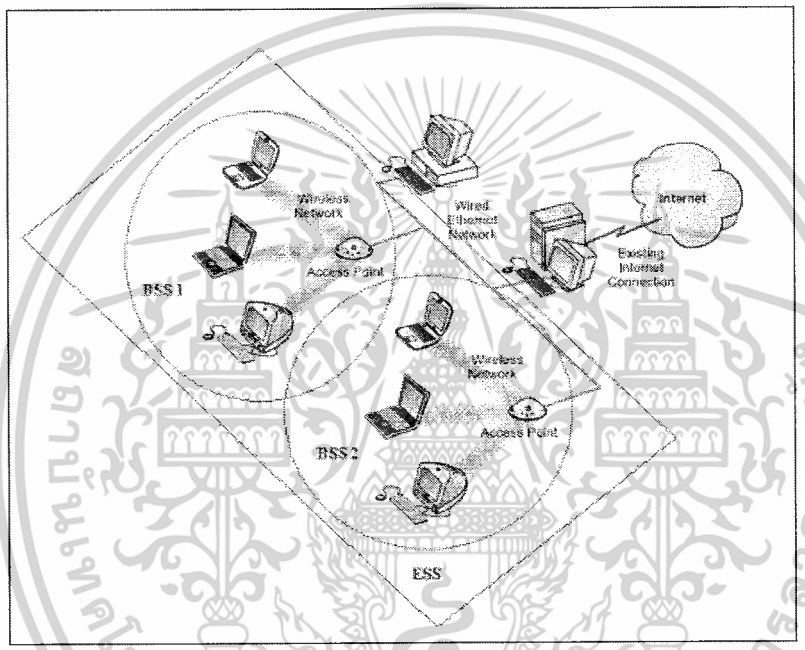
2.2.2 รูปแบบการเชื่อมต่อของมาตรฐาน IEEE 802.11

มาตรฐาน IEEE 802.11 ได้กำหนดลักษณะการเชื่อมต่อของอุปกรณ์ภายในเครือข่าย WLAN ไว้ 2 ลักษณะคือโหมด Infrastructure และโหมด Ad-Hoc หรือ Peer-to-Peer

2.2.2.1 โหมด Infrastructure

โดยทั่วไปแล้วอุปกรณ์ในเครือข่าย IEEE 802.11 WLAN จะเชื่อมต่อกันในลักษณะของโหมด Infrastructure ซึ่งเป็นโหมดที่อนุญาตให้อุปกรณ์ภายใน WLAN สามารถเชื่อมต่อกับเครือข่ายอื่นได้ ในโหมด Infrastructure นี้เครือข่าย IEEE 802.11 WLAN จะประกอบไปด้วย

อุปกรณ์ 2 ประเภทได้แก่ สถานีผู้ใช้ (Client Station) ซึ่งก็คืออุปกรณ์คอมพิวเตอร์ (Desktop Laptop หรือ PDA ต่างๆ) ที่มีอุปกรณ์ Client Adapter เพื่อใช้รับส่งข้อมูลผ่าน IEEE 802.11 WLAN และสถานีแม่ข่าย (Access Point) ซึ่งทำหน้าที่ต่อเชื่อมสถานีผู้ใช้เข้ากับเครือข่ายอื่น การทำงานในโหมด Infrastructure มีพื้นฐานมาจากระบบเครือข่ายโทรศัพท์มือถือ กล่าวคือสถานีผู้ใช้จะสามารถรับส่งข้อมูลโดยตรงกับสถานีแม่ข่ายที่ให้บริการแก่สถานีผู้ใช้นั้นอยู่เท่านั้น ส่วนสถานีแม่ข่ายจะทำหน้าที่ส่งต่อ (forward) ข้อมูลที่ได้รับจากสถานีผู้ใช้ไปยังจุดหมายปลายทางหรือส่งต่อข้อมูลที่ได้รับจากเครือข่ายอื่นมายังสถานีผู้ใช้

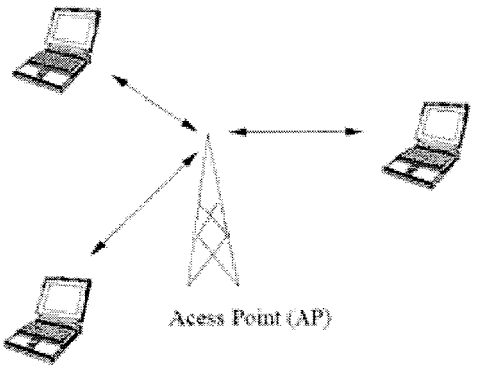


รูปที่ 2.3 แสดง BSS และESS

Basic Service Set (BSS)

Basic Service Set (BSS) หมายถึงบริเวณของเครือข่าย IEEE 802.11 WLAN ที่มีสถานีแม่ข่าย 1 สถานี ซึ่งสถานีผู้ใช้ภายในขอบเขตของ BSS นี้ทุกสถานีจะต้องสื่อสารข้อมูลผ่านสถานีแม่ข่ายดังกล่าวเท่านั้น เช่น สถานี A ต้องการส่งข้อมูลไปให้สถานี B (โดยทั้งสถานี A และสถานี B อยู่ใน BSS เดียวกัน) สามารถทำได้โดย

- สถานี A จะส่งข้อมูลไปให้แอคเซสพอยต์ (สถานีแม่ข่าย)
- แอคเซสพอยต์จะทำการส่งข้อมูลที่ได้รับมาจากสถานี A ไปยังสถานีปลายทางนั่นก็คือ สถานี B



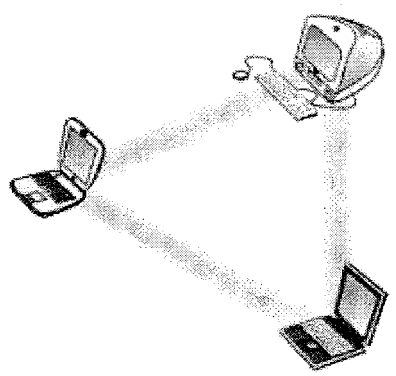
รูปที่ 2.4 แสดงการเชื่อมต่อแบบ InfraStructure

Extended Service Set (ESS)

Extended Service Set (ESS) หมายถึงบริเวณของเครือข่าย IEEE 802.11 WLAN ที่ประกอบด้วย BSS มากกว่า1 BSS ซึ่งได้รับการเชื่อมต่อเข้าด้วยกัน กลายเป็นเครือข่ายขนาดใหญ่คลอบคลุมพื้นที่การใช้งานได้กว้างขึ้น สถานีต่างๆที่อยู่ใน ESS เดียวกันจะสามารถติดต่อสื่อสารกันได้ แม้ว่าแต่ละสถานีจะอยู่คนละ BSS หรืออยู่ในขณะที่กำลังเปลี่ยนจาก BSS หนึ่งไปเป็นอีก BSS หนึ่งก็ตาม การที่สถานีต่างๆ ใน ESS จะสามารถติดต่อสื่อสารกันได้นั้น ตัวกลางไร้สายจะต้องทำงานเสมือนการติดต่อในระดับดาต้าลิงค์โดยมี แอคเซสพอยต์เปรียบเสมือนบริดจ์ (bridge) ที่คอยเชื่อมต่อระหว่างสถานีต่างๆใน ESS โดย แอคเซสพอยต์ในแต่ละ BSS จะเชื่อมต่อกับฮับ (hub) หรือสวิตช์ (switch) ของเครือข่าย

2.2.2.2 โหมด Ad-Hoc หรือ Peer to Peer

เครือข่าย IEEE 802.11 WLAN ในโหมด Ad-Hoc หรือ Peer-to-Peer เป็นเครือข่ายที่ปิดคือไม่มีสถานีแม่ข่ายและไม่มีการเชื่อมต่อกับเครือข่ายอื่น บริเวณของเครือข่าย IEEE 802.11 WLAN ในโหมด Ad-Hoc จะถูกเรียกว่า Independent Basic Service Set (IBSS) ซึ่งสถานีผู้ใช้หนึ่งสามารถติดต่อสื่อสารข้อมูลกับสถานีผู้ใช้อื่นๆในเขต IBSS เดียวกันได้โดยตรงตราบเท่าที่สัญญาณจะไปถึงโดยไม่ต้องผ่านสถานีแม่ข่ายแต่สถานีผู้ใช้จะไม่สามารถรับส่งข้อมูลกับเครือข่ายอื่นๆได้ โดยในการเชื่อมต่อแบบ Ad-Hoc จะนิยมใช้เป็นการเชื่อมต่อแบบชั่วคราว เช่น ในการประชุมจะทำการเชื่อมต่อสถานีต่างๆ เข้าด้วยกันเพื่อใช้ในการแลกเปลี่ยนข้อมูล เมื่อการประชุมเสร็จสิ้นเครือข่ายนั้นก็จะถูกยกเลิก



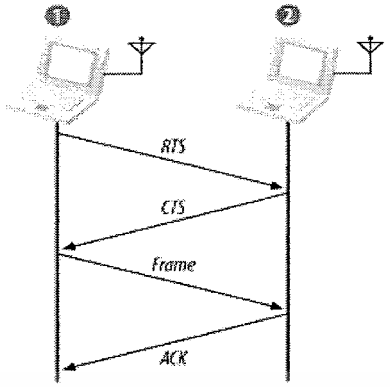
รูปที่ 2.5 แสดงโหมด Ad-Hoc หรือ Peer to Peer

2.3 IEEE 802.11 Mac Fundamental

องค์ประกอบของเครือข่ายไร้สายตามมาตรฐาน 802.11 ได้แก่ เครื่องคอมพิวเตอร์แบบเคลื่อนที่ และสถานี (Acess Point) การเข้าใช้ตัวกลางของแต่ละสถานีเคลื่อนที่แบ่งเป็นสามแบบ คือ Distributed Coordination Function (DCF) Point Coordination Function (PCF) และ Hybrid coordination function (HSF)

2.3.1 การใช้งานของสื่อแบบกระจาย DCF (Distributed Coordination Function)

ตามมาตรฐาน 802.11 การเข้าถึงตัวกลางบนเครือข่ายไร้สายนั้นไม่สามารถที่จะใช้กลไกวิธีแบบ CSMA/CD หรือเรียกว่า การตรวจการใช้สื่อแบบตรวจการชน ที่ใช้งานบนเครือข่ายอีเทอร์เนตแบบมีสายได้ แต่จะมีการนำกลไกของ CSMA/CD มาปรับปรุงเพื่อใช้งานบนเครือข่ายไร้สาย ซึ่งเรียกว่า CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) โดยการทำงานแบบร่วมกัน (DCF) ก็ใช้วิธีการเข้าใช้สื่อแบบ CSMA/CA โดยกลไกดังกล่าวเป็นกลไกเพื่อหลีกเลี่ยงการชนซึ่งคล้ายกับอีเทอร์เนต ซึ่งการทำงานของ CSMA/CAนี้ โหนดใดที่ต้องการส่งข้อมูลก่อนที่จะทำการส่งจะมีการส่งรหัสควบคุม (RTS : Request to Send) ไปยังโหนดปลายทางก่อนและเมื่อโหนดปลายทางพร้อมที่จะรับก็จะทำการส่งรหัสควบคุม (CTS : Clear To Send) กลับไปยังโหนดต้นทาง เมื่อโหนดต้นทางได้รับการตอบรับจากโหนดปลายทางว่าพร้อมที่จะรับข้อมูลก็จะดำเนินการส่งเฟรมข้อมูลไปยังปลายทางทันที

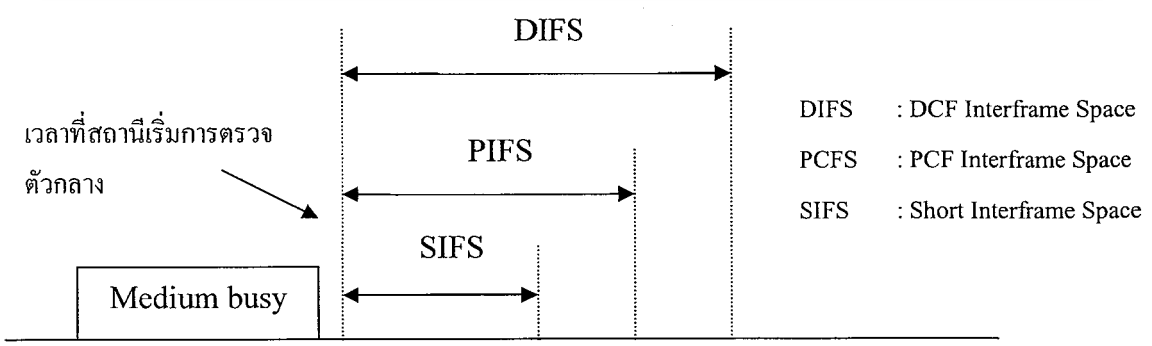


รูปที่ 2.6 แสดงการใช้ RTS และ CTS

การควบคุมการใช้สื่อแบบกระจายเป็นการควบคุมพื้นฐานของมาตรฐาน 802.11 ทุกสถานีสามารถติดต่อสื่อสารกันได้โดยตรงไม่ต้องอาศัยสถานีฐานหรือโครงสร้างทางเครือข่าย กลุ่มของสถานีที่มีการติดต่อสื่อสารดังกล่าวเรียกว่า เครือข่าย ad-hoc หรือ IBSS

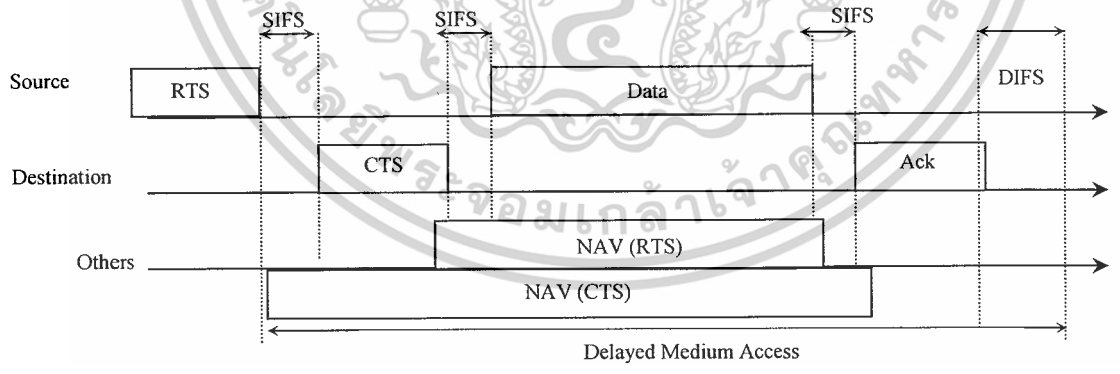
การทำงานของโปรโตคอลหลีกเลี่ยงการชนแต่ละสถานีจะได้ส่งข้อมูลก็ต่อเมื่อตรวจพบว่าช่องสัญญาณว่างเป็นระยะเวลาเท่ากับช่องว่างระหว่างเฟรม มาตรฐาน 802.11 แบ่งเฟรมออกเป็น 3 ประเภทคือ เฟรมการจัดการ (Management frame) เฟรมควบคุม (Control frame) และเฟรมข้อมูล (Data frame) ซึ่งเฟรมควบคุมทำหน้าที่ ควบคุมการทำงานของการรับส่งข้อมูลในเครือข่ายเช่นเฟรมตอบรับหรือเฟรม RTS/CTS เป็นต้น ดังนั้นเฟรมควบคุมจึงควรมีไพรออริตีสูงสุด มาตรฐาน 802.11 จึงกำหนดขนาดช่องว่างระหว่างเฟรมดังรูปที่ 2.7

DIFS เป็นช่องว่างระหว่างเฟรมที่ยาวที่สุดจึงมีไพรออริตีต่ำสุด ใช้สำหรับการส่งข้อมูลของสถานีทั่วไป PIFS เป็นช่องว่างระหว่างเฟรมที่สั้นกว่า DIFS ใช้สำหรับเฟรมที่ใช้ในการควบคุมใช้สื่อแบบรวมศูนย์ และเฟรมการจัดการ เช่น บีคอนจากแอคเซสพอยต์ จึงถูกออกแบบให้มีไพรออริตีสูงกว่า DIFS ช่องว่างระหว่างเฟรมที่มีไพรออริตีสูงสุดคือ SIFS ถูกออกแบบมาสำหรับเฟรมที่ใช้ควบคุมการทำงานของเครือข่าย ซึ่งต้องการการตอบรับทันทีเช่น เฟรมตอบรับ เฟรมร้องขอการส่งข้อมูล (RTS/CTS) เป็นต้น



รูปที่ 2.7 แสดงขนาดของช่องว่างระหว่างเฟรมในมาตรฐาน 802.11

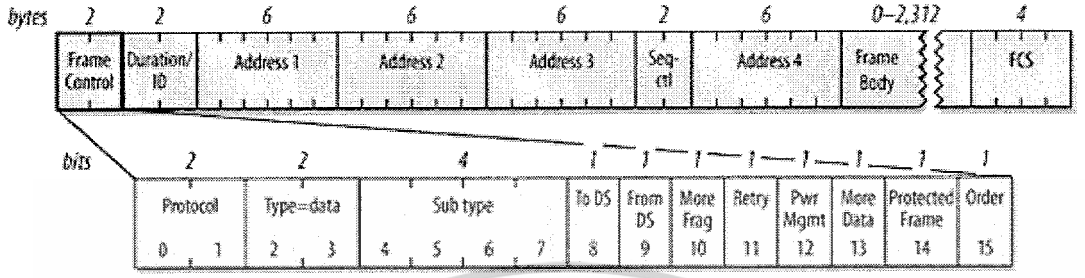
การเข้าใช้สื่อแบบกระจายในเครือข่ายแลนไร้สายจะเริ่มสุ่มเวลาเข้าใช้สื่อเมื่อตรวจพบว่าช่องสัญญาณว่างเป็นระยะเวลา DIFS แต่ละสถานีจะสุ่มเวลาเข้าใช้สื่ออยู่ในช่วง (0, w-1) ค่า w คือช่วงเวลาสำหรับแข่งขันของแต่ละสถานี ซึ่งค่า w จะขึ้นอยู่กับจำนวนครั้งการชนกันของข้อมูล ในภาวะเริ่มต้นค่า w จะถูกตั้งไว้เท่ากับ CW_{min} ซึ่งเป็นช่วงการแข่งขันที่สั้นที่สุด ค่า w จะเพิ่มขึ้นเป็น 2 เท่าทุกครั้งที่เกิดการชนกันของข้อมูล ดังนั้นค่า w จึงอยู่ในช่วง 0 ถึง $2^i CW_{min}$ เมื่อ i คือจำนวนครั้งของการชนกัน ซึ่งตัวนับเวลาจะทำงานก็ต่อเมื่อตรวจพบว่าช่องสัญญาณว่างและจะหยุดทำงานเมื่อตรวจดูพบว่าช่องสัญญาณถูกใช้อยู่ เมื่อตัวนับลดค่า w จนเท่ากับศูนย์จึงเริ่มส่งเฟรม RTS ไปที่สถานีปลายทางดังรูปที่ 2.8 รอจนช่องสัญญาณว่างเป็นเวลาเท่ากับ SIFS สถานีปลายทางจะส่งเฟรม CTS กลับมาที่สถานีต้นทาง ในข้อมูล RTS หรือ CTS จะมีตำแหน่งของต้นทาง ปลายทาง และเวลาที่ใช้ในการส่งข้อมูลชุดนั้น สถานีที่ไม่ใช่สถานีต้นทางและปลายทางหากได้รับเฟรม RTS หรือเฟรม CTS จะเซตค่า NAV (Network Allocation Vector)ให้มีค่าเท่ากับช่วงเวลาเท่ากับที่สถานีต้นทางส่งข้อมูลไปที่สถานีปลายทางจนเสร็จ แต่ละสถานีสามารถส่งข้อมูลได้ก็ต่อเมื่อค่า NAV เท่ากับศูนย์ จึงเป็นการตรวจช่องสัญญาณแบบเสมือนที่ชั้นควบคุมการใช้สื่อ เมื่อสถานีต้นทางได้รับเฟรม CTS จึงเริ่มส่งข้อมูลได้ หลังจากสถานีปลายทางได้รับข้อมูลเป็นระยะเวลา SIFS สถานีปลายทางจะส่งเฟรมตอบรับมาที่สถานีต้นทาง หากเกิดการชนกันของข้อมูลสถานีปลายทางจะไม่ส่งเฟรมตอบรับมาที่สถานีต้นทาง เมื่อสถานีต้นทางไม่ได้รับเฟรมตอบรับในช่วงเวลาที่กำหนดจะถือว่าเกิดการชนกันของข้อมูล ดังรูปที่ 2.8 สถานีต้นทางจะเริ่มสุ่มเวลาเข้าใช้สื่อและตรวจช่องสัญญาณเพื่อส่งข้อมูลใหม่



รูปที่ 2.8 แสดงการเข้าใช้สื่อของโปรโตคอลหลีกเลี่ยงการชน (CSMA/CA) ที่ใช้ RTS/CTS

2.3.2 Frame Format

สำหรับเฟรมข้อมูลของ MAC layer จะประกอบด้วย 9 ฟิลด์ ดังรูปที่ 2.9



รูปที่ 2.9 แสดงรูปเฟรมของ mac layer และ ฟิลด์เฟรมควบคุม

- ฟิลด์ควบคุม (Frame Control) มีขนาด 2 ไบต์ ฟิลด์นี้ใช้สำหรับเก็บชนิดของเฟรมข้อมูลและการควบคุมต่างๆ
- ฟิลด์ระยะเวลา (Duration/ID Field) ใช้สำหรับเก็บค่า NAV หรือ ID ของเฟรมข้อมูล
- ฟิลด์แอดเดรส (Address Fields) ใช้สำหรับเก็บ address นั้นจะมีอยู่ด้วยกัน 4 ฟิลด์ ฟิลด์ละ 6 ไบต์
- ฟิลด์หมายเลขลำดับ (Sequence Control Field) ใช้สำหรับเก็บหมายเลขลำดับของเฟรม เพื่อช่วยในการควบคุมการไหลของเฟรมข้อมูล (flow control)
- ฟิลด์ข้อมูล (Frame Body) ใช้สำหรับเก็บข้อมูลที่ส่งมาจากเลเยอร์ข้างบนซึ่งฟิลด์นี้จะมีขนาดระหว่าง 0-2312 ไบต์
- ฟิลด์ตรวจสอบความผิดพลาดของข้อมูล (Frame Check Sequence) ใช้สำหรับตรวจสอบว่าข้อมูลมีความผิดพลาดในระหว่างการส่งหรือไม่ โดยจะใช้วิธีตรวจสอบแบบ CRC-32]

2.4 IEEE 802.11 Framing in Detail

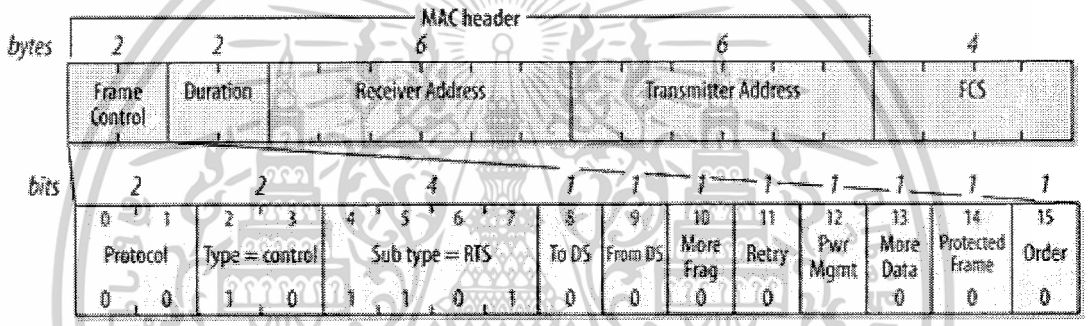
การติดต่อสื่อสารแบบไร้สายมีความแตกต่างกับการสื่อสารแบบมีสายมาก เช่น เรื่องของความปลอดภัย แบนด์วิดท์ที่มีอยู่อย่างจำกัด และปัญหาโหนดที่ซ่อน (Hidden terminal) ในมาตรฐาน 802.11 มีวิธีการติดต่อสื่อสารแบบรวมศูนย์เป็นทางเลือกหนึ่ง ซึ่งมีแอคเซสพอยต์เป็นศูนย์กลาง การที่แต่ละสถานีจะเข้ามาใช้ในเครือข่ายจึงจำเป็นต้องผ่านการตรวจสอบการมีสิทธิเข้าใช้เครือข่าย ดังนั้น เฟรมข้อมูล เฟรมที่ใช้พิสูจน์ตัวตน เฟรมร้องขอการส่งข้อมูล จึงมีหน้าที่การทำงานที่ต่างกัน มาตรฐาน 802.11 จึงแบ่งเฟรมออกเป็น 3 ประเภท คือ เฟรมการจัดการ เฟรมควบคุม และเฟรมข้อมูลแต่ละประเภทมีความสำคัญต่างกัน จึงมีการกำหนดช่องว่างระหว่างเฟรมของเฟรมแต่ละประเภท และหน้าที่จองเฟรมแต่ละเฟรมจะมีดังต่อไปนี้

2.4.1 เฟรมการจัดการ (Management Frame)

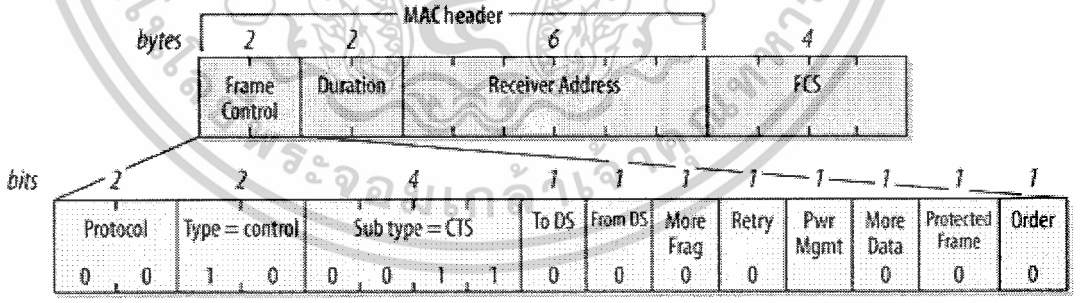
เฟรมการจัดการถูกออกแบบมาเพื่อใช้เมื่อมีการติดต่อสื่อสารครั้งแรกระหว่างสถานีใดๆกับแอคเซสพอยต์ หน้าที่ของเฟรมประเภทนี้คือใช้ติดต่อเชื่อมโยงและพิสูจน์การมีสิทธิเข้าใช้เครือข่าย

2.4.2 เฟรมควบคุม (ControlFrame)

หลังจากที่ใช้เฟรมการจัดการตรวจสอบการมีสิทธิเข้าใช้งานเสร็จสิ้นแล้ว ก่อนที่จะส่งข้อมูลได้ มาตรฐาน 802.11 ได้กำหนดให้สถานีที่ต้องการส่งข้อมูลได้ส่งเฟรมควบคุมก่อนดังนี้ ส่งเฟรมร้องขอ (RTS) ก่อนและรอรับเฟรมตอบรับ (CTS) เพื่อลดปัญหาของโหนดที่ซ่อนหลังจากส่งข้อมูลเสร็จปลายทางจะส่งเฟรม Ack กลับมาที่ต้นทางที่ส่งเฟรม เฟรม RTS CTS และ Ack เป็นเฟรมควบคุมที่สำคัญในการติดต่อสื่อสารในมาตรฐาน 802.11



รูปที่ 2.10 แสดงรูปแบบของเฟรมควบคุมร้องขอ (RTS)

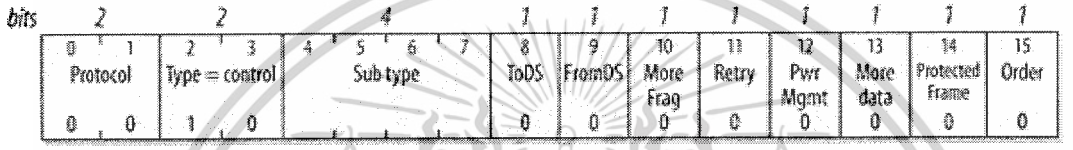


รูปที่ 2.11 แสดงรูปแบบของเฟรมตอบรับ (CTS) และ Ack

จากรูปที่ 2.10 และ รูปที่ 2.11 จะอธิบายฟิลด์ต่างๆ ได้ดังนี้

- Frame control ทำหน้าที่หลักในการบอกประเภทของเฟรมหนึ่งว่าเป็นเฟรมจัดการ เฟรมควบคุม หรือเป็นเฟรมข้อมูล และบอกประเภทฟังก์ชันของเฟรมในแต่ละประเภทด้วยและมีหน้าที่อื่นๆ เช่น การประหยัดพลังงาน และการเข้ารหัสด้วย Wired Equivalent Privacy Algorithm (Wep)

- Duration มีขนาด 2 ไบต์ ทำหน้าที่บอกให้สถานีได้รับเฟรมนี้เซ็ตค่า NAV (Network Allocation Vector) เท่ากับค่าในฟิลด์นี้ ซึ่งมีค่าเป็นมิลลิวินามี ค่าในฟิลด์นี้จะถูกเซ็ตไว้เท่ากับช่วงเวลาที่จะมีการติดต่อสื่อสารกันดังรูปที่ 2.11 เพื่อให้แต่ละสถานีไม่ต้องตรวจช่องสัญญาณในชั้นฟิสิคัลตลอดเวลาและลดปัญหาโหนดที่ซ่อน
- RA (Receiver Address) แสดงแม็คแอดเดรสของผู้รับ ขนาด 6 ไบต์
- TA (Transmitter address) แสดงแม็คแอดเดรสของผู้ส่ง ขนาด 6 ไบต์
- FSC (Frame check sequence) ตรวจสอบความถูกต้องของเฟรมด้วย CRC (Cyclic Redundancy Check Sequence) ขนาด 4 ไบต์

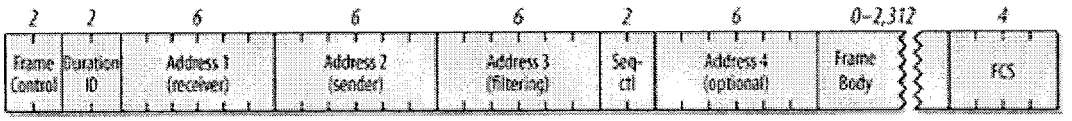


รูปที่ 2.12 แสดงรายละเอียดภายในฟิลด์ควบคุม (Frame control)

- Protocol version ใช้แสดงเวอร์ชั่นของโปรโตคอลในเวอร์ชั่นปัจจุบันจะแสดงเลข0
- Type ในเฟรมควบคุมค่าของฟิลด์ type จะเป็น 01 เสมอ โดยจะนับเล็กจากบิตตัวหลังก่อน เช่นในรูปที่ 2.12 ฟิลด์ Type จะแสดงเลข 10 แต่จริงๆ จะอ่านจากหลังมาหน้าเป็น 01
- Subtype ใช้บอกว่าเป็นเฟรมควบคุม ชนิด RTS หรือ CTS หรือ ACK เช่น ถ้าเป็น RTS ค่าจะเป็น 1101 CTS จะเป็น 1100 และACK จะเป็น 1101
- ToDS and FromDS bits มีขนาด 1 บิตต้องเป็นเลขศูนย์เสมอ มิฉะนั้นระบบจะไม่สามารถรับหรือส่งเฟรมควบคุมเฟรมนี้ได้
- ส่วนฟิลด์ที่เหลือจะเซ็ตค่าเป็นเลข 0 หมดในเฟรมควบคุม

2.4.3 เฟรมข้อมูล (Data Frame)

หน้าที่หลักของเฟรมข้อมูล คือ บรรจุข้อมูลไปยังปลายทาง สามารถส่งข้อมูลขนาดใหญ่ที่สุดได้ 2312 ไบต์



รูปที่2.13 แสดงรูปของเฟรมข้อมูล

- Address 1 คือ แมคแอดเดรสของอุปกรณ์ที่จะรับเฟรมนี้
- Address 2 คือ แมคแอดเดรสของอุปกรณ์ที่ทำการส่งเฟรมนี้

มาตรฐาน IEEE 802.11 นั้นมีกลไกการกำหนดแอเดรสในเฟรมข้อมูลค่อนข้างซับซ้อน เนื่องจากจะต้องมีแอดเซสน์พ้อยเข้ามาเกี่ยวข้องกับการส่งด้วย ดังนั้นจึงสามารถแบ่งการกำหนดแอดเดรสในเฟรมข้อมูลออกได้เป็น 4 กรณีโดยการกำหนดค่าในฟิลด์ควบคุม (Frame Control) ซึ่งจะกำหนดในฟิลด์ย่อยคือ To DS และ From DS โดยตัวอย่างจะเป็นดังต่อไปนี้

Function	ToDS	FromDS	Address 1 (receiver)	Address 2 (transmitter)	Address 3	Address 4
IBSS	0	0	DA	SA	BSSID	Not used
To AP (infra.)	1	0	BSSID	SA	DA	Not used
From AP (infra.)	0	1	DA	BSSID	SA	Not used
WDS (bridge)	1	1	RA	TA	DA	SA

ตารางที่ 2.1 แสดงตัวอย่างรูปแบบการกำหนดค่าการส่งทั้ง 4 กรณีในเฟรมข้อมูล

- กรณีที่ 1 ค่า ToDS = 0 และค่า FromDs = 0

หมายความว่าเฟรมข้อมูลถูกส่งเฉพาะกับสถานีภายใน BSS เดียวกันเท่านั้นไม่มีการส่งข้ามไปยัง BSS อื่นดังนั้นเฟรมตอบรับ (ACK) จึงสามารถส่งกลับไปให้สถานีที่ส่งได้โดยตรงโดยไม่ต้องส่งผ่านสถานีใดๆ

- กรณีที่ 2 ค่า ToDS = 1 และค่า FromDs = 0

หมายความว่าข้อมูลนี้จะถูกส่งจากสถานีไปยัง แอ็กเซสพอยต์เพื่อที่จะส่งต่อไปยังแอ็กเซสพอยต์อื่น แสดงว่าสถานีปลายทางอยู่คนละBSSกัน ซึ่งจะสังเกตได้ว่ากรณีนี้ ในฟิลด์Address 3 จะเก็บค่าของสถานีปลายทางที่รับข้อมูลที่อยู่ BSS อื่นคนละ BSS กับสถานีต้นทาง

- กรณีที่ 3 ค่า ToDS = 0 และค่า FromDs = 1

หมายความว่าเฟรมนี้จะถูกส่งจากแอ็กเซสพอยต์ไปยังสถานีปลายทางที่รับ ดังนั้นเฟรมตอบรับจะต้องส่งกลับจากสถานีปลายทางไปยังแอ็กเซสพอยต์ จะสังเกตได้ว่ากรณีนี้ ในฟิลด์Address 3 จะเก็บค่าาของสถานีต้นทางที่ส่งข้อมูลที่อยู่ BSS อื่นคนละ BSS กับสถานีปลายทาง

- กรณีที่ 4 ค่า ToDS = 1 และค่า FromDs = 1

หมายความว่าเฟรมข้อมูลนี้จะถูกส่งกันระหว่างแอ็กเซสพอยต์ จะสังเกตได้ว่ากรณีนี้ ในฟิลด์ Address 3 จะเก็บค่าของสถานีปลายทางที่รับข้อมูลและจะมีการใช้ Address 4 ซึ่งเก็บค่าของสถานีต้นทางที่ส่งข้อมูล

2.5 ทฤษฎีเบื้องต้นของการโปรแกรมมิ่งบนลีนุกซ์

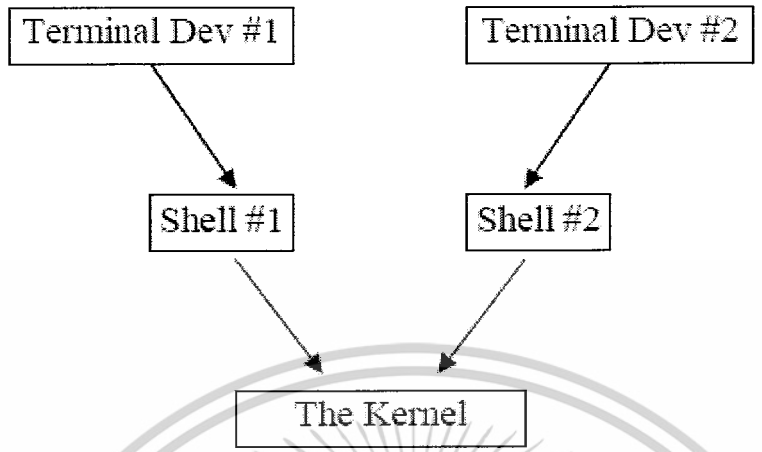
ลีนุกซ์เป็นระบบปฏิบัติการแบบ 32 บิต ที่เป็นยูนิกซ์โคลน สำหรับเครื่องพีซีและแจกจ่ายให้ใช้ฟรีซึ่งมีให้เลือกใช้งานได้อย่างหลากหลาย โดยแต่ละระบบปฏิบัติการก็มีความสามารถหรือจุดเด่นที่แตกต่างกันไป แม้ว่าจะอยู่ในตระกูลลีนุกซ์เหมือนกันก็ตาม แต่ก็มีความแตกต่างให้ความสามารถ

ที่แตกต่างกันอยู่บ้าง เช่น การให้บริการเป็นเซิร์ฟเวอร์ในงานต่างๆหรือแอพพลิเคชั่นที่หลากหลาย เป็นต้น ลีนุกซ์นั้นสนับสนุนการใช้งานแบบหลากงานหลายผู้ใช้ (MultiUser-MultiTasking) มีระบบ X วินโดวส์ ซึ่งเป็นระบบการติดต่อผู้ใช้แบบกราฟฟิกที่ไม่ขึ้นกับระบบปฏิบัติการหรือฮาร์ดแวร์ใดๆ (มักใช้กันมากในระบบยูนิกซ์) และมาตรฐานการสื่อสาร TCP/IP ที่ใช้เป็นมาตรฐานการสื่อสารในอินเทอร์เน็ตมาให้ในตัว ลีนุกซ์มีความเข้ากันได้ (Compatible) กับ มาตรฐาน POSIX ซึ่งเป็นมาตรฐานอินเทอร์เฟสที่ระบบยูนิกซ์ส่วนใหญ่จะต้องมีและมีรูปแบบบางส่วนที่คล้ายกับระบบปฏิบัติการยูนิกซ์จากค่าย Berkeley และ System V โดยความหมายทางเทคนิคแล้วลีนุกซ์ เป็นเพียงเคอร์เนล (Kernel) ของระบบปฏิบัติการ ซึ่งจะทำหน้าที่ในด้านของการจัดสรรและบริหารโพรเซสงาน การจัดการไฟล์และอุปกรณ์ I/O ต่างๆ แต่ผู้ใช้ทั่วๆไปจะรู้จักลีนุกซ์ผ่านทางแอพพลิเคชั่นและระบบอินเทอร์เฟสที่เขาเหล่านั้นเห็น (เช่น Shell หรือ X วินโดวส์) โดยภายใน X windows จะประกอบไปด้วยส่วนของตัวจัดการวินโดวส์และสิ่งแวดล้อมในการทำงานของระบบปฏิบัติการ ซึ่งชนิดหรือประเภทส่วนประกอบทั้งสองตัวนี้อาจจะแตกต่างกันได้ในลีนุกซ์ต่างชนิดกัน ซึ่งในส่วนของสิ่งแวดล้อมในการทำงาน ที่เป็นที่นิยมมีด้วยกันอยู่สองตัวคือ GNOME และ KDE โดยเมื่อลงลีนุกซ์ จะสามารถเลือกว่าจะเอาสิ่งแวดล้อมในการทำงานดังกล่าวว่าจะใช้ชนิดใด

ความแตกต่างของลีนุกซ์นี้ เพราะฉะนั้นในการใช้งานควรเลือกใช้ลีนุกซ์ให้เหมาะสมกับงานที่ต้องใช้ แต่ก็สามารถเลือกแพ็กเก็ตเพิ่มเติมเพื่อให้ความสามารถที่ไม่มีของลีนุกซ์บางตัวมีความสามารถนั้นได้เหมือนกันโดยผ่าน “ตัวจัดการแพ็กเกจ” หรือ “package manager” โดยตัวจัดการแพ็กเกจที่ได้รับความนิยมคือ “Red Hat Package- Manager: RPM” ซึ่งระบบปฏิบัติการลีนุกซ์หลายตัวก็ใช้ตัวนี้เช่นกัน นอกจากนี้ก็สามารถไปดาวน์โหลดโปรแกรมต่างๆ มาใช้ได้ โดยโปรแกรมดังกล่าวจะอยู่ในรูปแบบ “Tarball format” ซึ่งเป็นรูปแบบในการบีบอัดไฟล์แบบหนึ่งซึ่งจะมีสคริปต์ที่ใช้ในการติดตั้งโปรแกรมปนอยู่ด้วย ซึ่งวิธีนี้ไม่ต้องผ่านตัวจัดการแพ็กเก็ต แต่ในการจัดการลบโปรแกรมเหล่านี้ออกก็จะสามารถทำได้ยากตามไปด้วยเช่นกัน

ปัจจุบันได้มีการนำระบบปฏิบัติการลีนุกซ์ไปประยุกต์เป็นระบบปฏิบัติการสำหรับงานด้านต่างๆ เช่นงานด้านการคำนวณทางวิทยาศาสตร์ใช้เป็นสถานีงาน สถานีบริการ อินเทอร์เน็ต อินทราเน็ต หรือใช้ในการเรียนการสอนและการทำวิจัยทางคอมพิวเตอร์ใช้พัฒนาโปรแกรม เนื่องจากมีเครื่องมือมากมาย เช่น โปรแกรมภาษาซี (C) ซีพลัสพลัส (C++) ปาสคาล (Pascal) ฟอร์แทรน (Fortran) ลิสป์ (Lisp) โปรล็อก (Prolog) เอดา (ADA) มีภาษาสคริปต์ เช่น เชลล์ (Shell) บาสช์เชลล์ (Bash Shell) ซีเชลล์ (C Shell) คอร์นเชลล์ (Korn Shell) เพิร์ล (Perl) พายตัน (python) TCL/TK นอกจากนี้ยังมีโปรแกรมประยุกต์ในสาขาต่างๆ อีกมากมาย

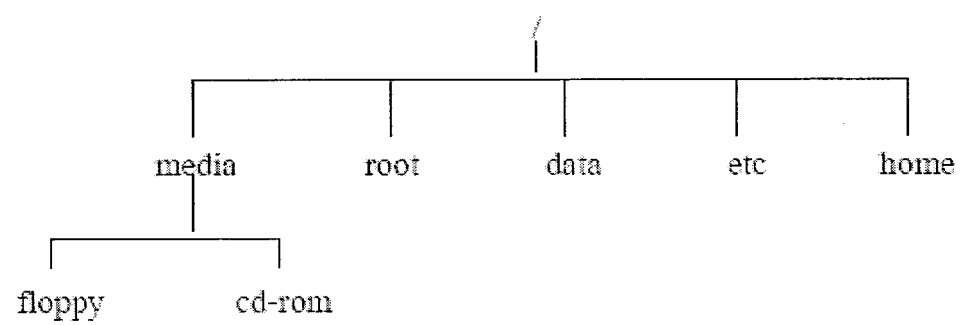
2.5.1 การใช้งานบนลีนุกซ์



รูปที่ 2.14 แสดงการใช้งานบนลีนุกซ์

ในการทำงานต่างๆ จะดำเนินการผ่านซอฟท์แวร์ ซึ่งหากงานนั้นต้องอาศัยการบริการจากฮาร์ดแวร์ ตัวซอฟท์แวร์ก็ต้องทำการร้องขอให้ฮาร์ดแวร์ทำงานตามที่ต้องการได้ โดยในทุกระบบปฏิบัติการประกอบขึ้นด้วยจากการทำงานหลายๆ ส่วน โดยจะมีส่วนประกอบหลักส่วนหนึ่งที่ทำหน้าที่ในการเรียกส่วนประกอบอื่นๆ ทั้งหมด ซึ่งเรียกว่า เคอร์แนล (kernel) เมื่อผู้ใช้ตอบโต้กับเครื่องคอมพิวเตอร์นั้นก็เปรียบเสมือนเป็นการตอบโต้กับเคอร์แนลนั่นเอง เพียงแต่การโต้ตอบกับเคอร์แนลนั้นไม่สามารถโต้ตอบได้โดยตรง แต่ต้องผ่านช่องทางของผู้ใช้เท่านั้น ซึ่งช่องทางดังกล่าวเรียกว่าเทอร์มินอล (terminal) ซึ่งลีนุกซ์รองรับการทำงานได้หลายๆ เทอร์มินอลพร้อมๆ กัน ซึ่งในการเข้าใช้แต่ละครั้งจะต้องมีอินเทอร์เฟสในการติดต่อกับเคอร์แนลซึ่งจะเรียกจุดที่ที่เชื่อมต่อกันนี้ว่า เปลือกระบบ (shell) โดยเปลือกระบบจะคอยทำหน้าที่ส่งข้อมูล (อินพุท) จากผู้ใช้ไปเป็นอินพุทของเคอร์แนลสำหรับการประมวลผลโดยปริยายแล้ว เปลือกระบบที่ใช้ในลีนุกซ์คือ BASH Shell (command line)

2.5.2 โครงสร้างของแฟ้มข้อมูล (File System)



รูปที่ 2.15 แสดงตัวอย่างโครงสร้างไฟล์บนลีนุกซ์

ในระบบลีนุกซ์ทำการเก็บข้อมูลโดยใช้ไฟล์และไดเร็คทอรีเข้ามาช่วย โดยจะมีลักษณะเป็นรูปแบบของ hierarchy หรือโครงสร้างแบบต้นไม้ ไดเร็คทอรีจะเปรียบเสมือนแฟ้มหลักที่สามารถเก็บไฟล์ต่างๆ โดยนอกจากจะเก็บไฟล์ต่างได้แล้วก็สามารถเก็บไดเร็คทอรีด้วยกันเองได้ด้วย

ไดเร็คทอรีลำดับบนสุดจะถูกเรียกว่า ไดเร็คทอรีราก (root directory) ซึ่งประกอบไปด้วยไฟล์และไดเร็คทอรีต่างๆ ในไดเร็คทอรีที่ย่อยลงมาก็อาจจะประกอบลงไปด้วยไฟล์และไดเร็คทอรีย่อยไปเรื่อยๆ ในแต่ละไฟล์จะต้องมีชื่อกำกับอยู่และจะมีการอ้างชื่ออยู่สองแบบที่ใช้อ้างถึงไฟล์ได้ คือ การอ้างชื่อแบบยาวและแบบสั้น ตัวอย่างการอื่อแบบสั้นก็เช่น “note” ส่วนการอ้างชื่อแบบยาวก็จะอ้างเป็น “/home/mary/note” การอ้างชื่อแบบยาวจะเห็นได้ว่าตัวอักษรซ้ายสุดคือ “/” ซึ่งจะระบุถึงไดเร็คทอรีรากอันเป็นไดเร็คทอรีลำดับบนสุด ในไดเร็คทอรีรากจะมีไดเร็คทอรี “home” บรรจุอยู่ และภายใต้ไดเร็คทอรี “home” ก็จะมีไดเร็คทอรี “mary” และไฟล์ “note” ของเราก็จะอยู่ภายใต้ไดเร็คทอรี “mary” สรุปได้ว่า การอ้างชื่อแบบยาวชื่อทั้งหมดก่อนที่จะถึงชื่อไฟล์ (คือ”/home/mary/”) จะเป็นชื่อของไดเรคทอรี่ชั้นต่างๆ ส่วนชื่อ ”note” จะเป็นชื่อไฟล์จริงๆ โดยในลีนุกซ์จะมีไดเร็คทอรีที่สำคัญ ได้แก่

/	ไดเร็คทอรีราก
/bin	โปรแกรมคำสั่งทั่วไปของระบบ
/boot	ไฟล์ที่ใช้ในการบู๊ตระบบ
/dev	ดีไวซ์ไฟล์สำหรับติดต่อกับระบบ
/etc	คอนฟิกไฟล์ของระบบ
/home	โฮมไดเร็คทอรีของผู้ใช้งานทั่วไปในระบบ
/lib	แชร์ไลบรารีและเคอร์เนลของโมดูล
/mnt	ไดเร็คทอรีสำหรับทำการเมาท์พาร์ติชันขึ้นมาชั่วคราว
/opt	ซ็อฟท์แวร์แพ็กเกจเพิ่มเติมที่ติดตั้งเข้ามาในระบบ
/root	โอมไดเร็คทอรีสำหรับผู้ดูแลระบบ
/sbin	โปรแกรมคำสั่งสำหรับจัดการระบบ
/tmp	ไฟล์ชั่วคราว
/usr	เก็บโปรแกรมและข้อมูลที่สามารถใช้งานร่วมกันกับเครื่องอื่นๆได้
/var	ข้อมูลทั่วไปของระบบที่ถูกใช้โดยโปรแกรมต่างๆ

- โครงสร้างของไดเร็คทอรี /etc

เก็บข้อมูลเกี่ยวกับคอนฟิกไฟล์ทั้งหลายของระบบ ส่วนข้อมูลที่เป็นแบบไบนารีซึ่งสมัยก่อนเก็บไว้ในไดเร็คทอรีนี้จะย้ายไปอยู่ใน /usr/sbin

- โครงสร้างของไดเร็คทอรี /sbin

เก็บโปรแกรมที่ผู้ดูแลระบบหรือรูทเอาไว้ใช้งานในการบริหารและควบคุมดูแลระบบ เช่น เมาท์ระบบไฟล์ บู๊ตหรือชัตดาวน์ระบบ เป็นต้น

- โครงสร้างของไดเร็คทอรี /usr

เก็บไฟล์ที่สามารถใช้กับเครื่องอื่นๆ ได้ ไดเร็คทอรีนี้ควรจะมีพาร์ติชั่นของตัวเองและจะต้องมีสถานะเป็น read-only หากเมาท์เข้ามาในระบบ

- คำสั่งที่ใช้ปฏิบัติการกับไดเร็คทอรี

pwd	ใช้เมื่อต้องการทราบว่าในขณะนี้อยู่ในไดเร็คทอรีไหนซึ่งจะแสดงชื่อของไดเร็คทอรีที่กำลังอยู่ในขณะนั้นออกมาให้
cd [ditrectory]	ใช้เมื่อต้องการย้ายหรือเปลี่ยนการอ้างอิงถึงไฟล์และโฟลเดอร์ไปยังไดเร็คทอรีอื่น
cd	ใช้เมื่อต้องการกลับไปยังไดเร็คทอรีแรกสุดที่เข้ามาหลังจากล็อกอินแล้วซึ่งก็คือ home directory
cd..	ใช้อ้างอิงถึงไดเร็คทอรีที่อยู่เหลือลำดับขึ้นไป รวมถึงไดเร็คทอรีของตัวเองด้วย
mkdir [directory]	ใช้คำสั่งนี้เมื่อต้องการสร้างไดเร็คทอรีใหม่
rm [directory]	ใช้คำสั่งนี้เมื่อต้องการลบไดเร็คทอรี

- คำสั่งที่ใช้ปฏิบัติการกับไฟล์

ลีนุกซ์เป็นระบบที่เกี่ยวข้องกับผู้ใช้หลายคน ดังนั้นลีนุกซ์จึงมีการกำหนดเกี่ยวกับเรื่องสิทธ์ที่จะเข้าถึงแฟ้มข้อมูล (file permission) ในระบบได้อย่างซับซ้อนโดยจะมีการแบ่ง user permission ระบุระดับการอณุญาติให้เข้าถึงไฟล์เช่น อ่านไฟล์ได้ เขียนไฟลีได้ และสามารถเรียกใช้งานหรือรันไฟล์ได้เป็นต้น

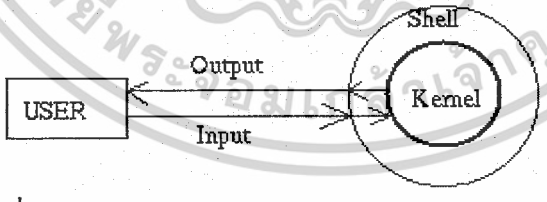
cp file1 file2	ใช้เมื่อต้องการก๊อปปี้ไฟล์ข้อมูล (หรืออาจจะเป็นไดเร็คทอรีก็ได้) โดยจะทำการก๊อปปี้จากไฟล์1 ไปไฟล์2
Rm filename	ใช้เมื่อต้องการลบไฟล์ข้อมูล (หรืออาจจะเป็นไดเรคทอรี)
Mv filename directory	ใช้เมื่อต้องการย้ายไฟล์ข้อมูลจากไดเร็คทอรีหนึ่งไปยังอีกไดเร็คทอรีหนึ่ง โดยจะย้ายไฟล์ filename จาก ไดเร็คทอรีปัจจุปันไปยังไดเร็คทอรี นอกจากนี้ยังสามารถใช้เปลี่ยนชื่อไฟล์ได้อีกด้วย เช่น mv file1 file2 จะเปลี่ยนชื่อจาก file1 เป็น file2

2.5.3 เคอร์เนลบนลีนุกซ์

เคอร์เนลบนลีนุกซ์นั้นเปรียบเสมือนแกนกลางของระบบ ซึ่งจะทำหน้าที่พื้นฐานในระบบ เช่น สร้างและจัดการเกี่ยวกับโปรเซส ให้บริการเกี่ยวกับการใช้งานระบบไฟล์ ทำงานเกี่ยวกับการสื่อสารข้อมูล และจัดการกับหน่วยความจำ รวมทั้งติดต่อกับฮาร์ดแวร์ด้วย

เคอร์เนลมีขนาดประมาณ 500-800 กิโลไบต์ ทั้งนี้ขึ้นอยู่กับจำนวนส่วนประกอบที่นำไปใส่ในเคอร์เนล ซึ่งขนาดดังกล่าวยังไม่รวมเนื้อที่เก็บข้อมูล (อีกประมาณ 1 MB) ซึ่งเมื่อรวมๆ แล้วจะกินเนื้อที่หน่วยความจำประมาณ 2 MB ลักษณะของเคอร์เนลในระยะแรกๆ จะรวมการทำงานต่างๆ เข้าด้วยกันเป็นชุดเดียวโดยไม่ได้แบ่งออกเป็นโปรแกรมย่อย ทำให้แอดเดรสเสปซ (address space) ของส่วนการทำงานต่างๆ ของลีนุกซ์เป็นแอดเดรสสเปซเดียวกัน จึงมีโอเวอร์เฮดน้อยกว่าการทำงานที่ต้องสลับไปสลับมาระหว่างฟังก์ชันต์ต่างๆ ของเคอร์เนล แต่มีข้อเสียตรงที่เราต้องเตรียมเคอร์เนลให้สนับสนุนลักษณะที่ต้องการเท่าที่จะทำได้ ทำให้เคอร์เนลอาจจะมีขนาดใหญ่เกินไปสิ้นเปลืองเนื้อที่ แต่ในปัจจุบันเราสามารถแยกบางส่วนของเคอร์เนลออกมาได้ ส่วนที่แยกออกมาเรียกว่าโมดูล (module) ซึ่งโดยทั่วไปแล้วโมดูลมักจะเป็นไดรเวอร์ของอุปกรณ์ต่างๆ เราสามารถสั่งให้เคอร์เนลโหลดโมดูลลงไปในหน่วยความจำ เพื่อให้เคอร์เนลมีความสามารถบางอย่างหรือสามารถใช้งานอุปกรณ์ฮาร์ดแวร์ที่ต้องการได้ และเมื่อไม่ต้องการความสามารถนั้นแล้วก็สามารถเอาโมดูลนั้นออกเพื่อเป็นการประหยัดเนื้อที่ในหน่วยความจำได้ตลอดเวลา วิธีนี้มีข้อดีหลายประการ เช่น เคอร์เนลของระบบที่ได้จะมีขนาดค่อนข้างเล็ก ทำให้ประหยัดเนื้อที่ในหน่วยความจำ และไม่จำเป็นต้องสร้างเคอร์เนลใหม่เมื่อต้องการเพิ่มเติมความสามารถอื่นๆ ลงไปในเคอร์เนล

2.5.4 การเขียนโปรแกรมเชลล์ (Shell Script)



รูปที่ 2.16 แสดงการติดต่อของระบบลีนุกซ์ผ่านทางเชลล์

ในการติดต่อกับระบบลีนุกซ์นั้น เราจะติดต่อโดยผ่านทางโปรแกรมเล็กๆโปรแกรมหนึ่งที่เรียกว่า Shell (เชลล์) เชลล์นั้นจะเป็นชั้นของโปรแกรมประยุกต์ที่จะคอยตีความคำสั่งจากผู้ใช้ไปให้เคอร์เนลของระบบ จากนั้นก็จะแสดงผลลัพท์ที่ได้จากเคอร์เนลกลับมาให้ผู้ใช้อีกทีหนึ่งนั่นคือเชลล์จะทำหน้าที่เป็นตัวเป็นตัวกลางคอยประสานงานการใช้งานระหว่างผู้ใช้และคอมพิวเตอร์ ตัวอย่างคำสั่งเชลล์ซึ่งจะแสดงให้เห็นว่าเชลล์ทำอะไรบ้าง

สมมติเราป้อนคำสั่ง $ sort –n phonelist > phonelist.sorted ให้กับเชลล์ ซึ่งจะมีความหมายคือ ให้เรียงลำดับในแต่ละบรรทัดของไฟล์ phonelist (เรียงลำดับตามตัวเลข) แล้วเก็บผลลัพท์ลงไปในไฟล์ใหม่ชื่อ phonelist.sorted ขั้นตอนการทำงานในเชลล์ มีดังนี้

1. แตกคำแต่ละคำในบรรทัดคำสั่งออกเป็น sort, -n, phonelist, > และ phonselist.sorted
2. แยกและวิเคราะห์หน้าที่ของคำทั้งหมดเช่น sort เป็นคำสั่งให้จัดบรรทัด –n และphonelist เป็น argument ที่ส่งให้กับคำสั่ง
3. ส่วน > และ phonelist.sorted เป็นคำสั่งที่เกี่ยวกับการนำข้อมูลผลลัพท์เข้า-ออกและการกำหนดทิศทางของการนำผลลัพท์ออก ดังนั้น ">phonelist.sorted" ก็คือการนำผลลัพท์จากอุปกรณ์แสดงมาตรฐาน(standard output) โดยนำไปเก็บที่ phonselist.sortedโดยหาคำสั่ง sortและเริ่มทำงานโดยใช้ argument สองตัวคือ –n และ phonelist

- การควบคุมลักษณะของการแสดงผล

การแสดงผลของเชลล์ สามารถส่งรหัสพิเศษ (escape sequence) ไปทำการควบคุมลักษณะการแสดงผลได้โดยรหัสเหล่านั้นจะขึ้นต้นด้วย หัสของ escape (\033) ดังนั้นจึงเรียกรหัสชุดเหล่านี้ว่า escape sequence และสามารถทำการบังคับให้ระบบทำการแสดงผลตัวอักษรแบบกระพริบ กลับขาว กลับดำ หรือขีดเส้นใต้ ได้ตัวอย่างของ หัสต่างๆ มีดังต่อไป นี้

\033[line;colH	ตำแหน่งของ prompt จะไปอยู่บน บรรทัด line และคอลัมน์ col
\033[2J	ลบหน้าจอ (คล้ายกับ tput clear บนยู นิกซ์ หรือ CLS บนดอส)
\033[4m	เข้าสู่โหมดขีดเส้นใต้ ตัว อักษร ที่พิมพ์ต่อจากรหัสชุดนี้จะ เป็น ตัว อักษร ที่มีการขีดเส้นใต้ ทั้ง หมด
\033[5m	เข้าสู่โหมดกระพริบ (blink)
\033[7m	โหมดกลับขาวดำ (reverse)
\033[m	กลับสู่โหมดปกติ

- การใช้งานตัวแปรและตัวควบ คุมอักขระพิเศษ (meta character)

ในการใช้งาน shell จะสังเกตุเห็นว่ามี ตัวอักขระบางตัวที่มีความหมายพิเศษ ซึ่งเชลล์จะต้องทำการตีความเสียก่อน ตัวอย่างของตัวอักขระพิเศษ เช่น

*	แทนตัวอักษรอะไรก็ได้ กี่ตัว ก็ได้
?	แทนตัวอักษรอะไรก็ได้จำนวน หนึ่ง ตัว
[...]	ให้เลือกเอาตัวอักษรในเครื่อง หมายก้ามปูหนึ่งตัว
[!..]	แทนตัวอักษรทุกตัวที่ไม่ ได้อยู่ในก้ามปู

ตัวอย่างเช่น สมมติมีไฟล์ในไดเร็คทอรีปัจจุบันดังต่อไปนี้

a aa ab aaa aba

abb baa bba bcb

คำสั่ง ls * จะเป็นการแสดงรายชื่อของไฟล์ทุกไฟล์เนื่องจาก * จะแทนตัวอักษร อะไร ก็ได้ และไม่จำกัดจำนวน แต่หากเป็น

$ ls a* ก็จะแสดงรายชื่อของไฟล์ a aa ab aaa aba และ abb เนื่องจากเป็นการแสดงชื่อไฟล์ที่ขึ้นต้นด้วย a และตามท้ายด้วยตัวอักษร อะไร ก็ได้ ไม่จำกัดจำนวน

$ ls a? จะแสดงชื่อไฟล์ aa และ ab เนื่องจากเป็นการแสดงชื่อไฟล์ที่ขึ้นต้นด้วย a และตามด้วยตัวอักษรอะไรก็ได้ หนึ่งตัว

$ls a[ab]a จะแสดงชื่อไฟล์ aaa และ aba เนื่องจากเป็นการแสดงชื่อไฟล์ที่ขึ้นต้นด้วย a และลงท้ายด้วย a มีความยาวสามตัวอักษร ส่วนตัวอักษรกลางเป็น ตัว a หรือ b ก็ได้

$ls ?[!ab]? จะแสดงชื่อไฟล์ bcb เนื่องจากเป็นการแสดงชื่อไฟล์ที่มีความยาวสามตัวอักษรและจะขึ้นต้นด้วยตัวอักษรอะไรก็ได้ รวมทั้งลงท้ายด้วยตัวอักษรอะไรก็ได้ แต่ตัวกลางจะต้องไม่เป็นตัว a และ b

- การลบความหมายของตัวอักษรพิเศษ

เนื่องจากเชลล์จะทำการตีความตัวอักษรพิเศษดังกล่าวข้างต้น แต่ถ้าหากเราต้องการนำตัวอักษรพิเศษเหล่านี้มาใช้งานจริงๆ ตัวอย่างเช่น ต้องการจะให้ระบบพิมพ์คำว่า "ls *" ซึ่งเราจะไม่สามารถใช้คำสั่ง echo ls * ได้ โดยที่ระบบจะพิมพ์ "ls" และรายชื่อของไฟล์ในไดเร็คทอรีปัจจุบันทั้งหมดออกมาแทน ("echo *" คือการพิมพ์รายชื่อของไฟล์ในไดเร็คทอรีปัจจุบันทั้งหมดออกมา) ดังนั้นจึงต้องมีวิธีการลบความหมายของตัวอักษรพิเศษ 3 วิธี ดังต่อไปนี้

'...' การครอบด้วยเครื่องหมายเขาเดี่ยว วิธีนี้เชลล์จะไม่ตีความตัวตัวอักษรพิเศษ

"..." การครอบด้วยเครื่องหมายเขาคู่ วิธีนี้เชลล์จะไม่ตีความตัวอักษรพิเศษ แต่จะตีความของ $ ซึ่งจะใช้เป็นสัญลักษณ์ของตัวแปรเชลล์ (shell variable)

\ ใช้ลบความหมายของ ตัว อักขระพิเศษที่ตามหลังมาหนึ่งตัว

ตัวอย่างของการใช้คำสั่งการลบความหมายตัวอักษรพิเศษ

```
echo "ls *"                --> ls *
echo 'ls *'                --> ls *
echo ls \*                 --> ls *
echo HOME is $HOME         --> HOME is /home/guest
echo "HOME is $HOME"       --> HOME is /home/guest
echo 'HOME is $HOME'       --> HOME is $HOME
echo 'HOME is \$HOME'      --> HOME is \$HOME
echo "pwd is `pwd`"        --> pwd is /home/guest
```

- การเปลี่ยนทิศทางของช่องทางการสื่อสาร (I/O redirection)

ในการสื่อสารระหว่างเทอร์มินอลกับตัวโฮสต์ของคอมพิวเตอร์จะมีช่องทางที่ใช้กันอยู่ 3 ช่องทาง ซึ่งก็คือ STDIN (ช่องทางรับ ข้อ มูลมาตรฐาน) STDOUT (ช่องทางแสดงผลลัพธ์มาตรฐาน) และ STDERR (ช่องทางแสดงผลลัพธ์ที่เป็นข้อผิดพลาดมาตรฐาน) โดยเราจะศึกษาจากคำสั่ง “cat” ซึ่งคำสั่งนี้จะรับข้อมูลจากคีย์บอร์ดและแสดงผลทางจอภาพ ดังนั้นในที่นี้ STDIN คือ คีย์บอร์ด และ STDOUT คือ จอภาพหรือมอนิเตอร์

```
$ cat
This is first line          : พิมพ์ข้อความ "This is first line"
This is first line          : cat แสดงผลลัพธ์ คือข้อความที่พึ่งพิมพ์ลงไป
This is second line         : พิมพ์ข้อความ "This is second line"
This is second line         : cat แสดงผลลัพธ์ คือข้อความที่พึ่งพิมพ์ลงไป
```

ต่อไปนี้เราจะเปลี่ยนช่องทางแสดง ผลลัพธ์ของโปรแกรม ซึ่งปกติแสดงออกมา ที่จอภาพให้ไปเก็บลงไฟล์แทน

```
$ cat > myfile
This is first line          : พิมพ์ข้อความ "This is first line"
This is second line         : พิมพ์ข้อความ "This is second line"
```

จะเห็นว่าครั้งนี้โปรแกรม cat ไม่แสดงผลลัพธ์ออกมาให้เหมือนตัวอย่างแรก ซึ่งเป็นเพราะว่าโปรแกรม cat ได้ทำการเก็บผลลัพธ์ลงไปในไฟล์ที่ชื่อ "myfile" แทน นั่นคือ ตอนนี้ STDOUT ได้ถูกเปลี่ยนเป็นไฟล์ "myfile" แล้ว นั่นคือเราสามารถใช้เครื่องหมายมากกว่า ">" ทำการเปลี่ยนช่องทางการแสดงผลลัพธ์ได้ การใช้เครื่องหมายมากกว่า ">" จะเป็นการเขียนทับไฟล์ที่ระบุลงไป แต่ถ้าเราต้องการให้เขียนผลลัพธ์ ไปต่อท้ายข้อมูลเดิมที่มีอยู่แล้วในไฟล์ก็สามารถทำได้ โดยการใช้เครื่องหมายมากกว่าซ้อนกันสองตัว ">>" ดังตัวอย่าง $ cat >> myfile

การนำข้อเข้ามูลหรือ STDIN นั้นในที่นี้เราสามารถที่จะเปลี่ยนแปลงช่องทางนำเข้าข้อมูลได้ โดยใช้คำสั่ง “sort” โดยปกติ จะทำการรับข้อมูลเข้าจากคีย์บอร์ดแล้วแสดงผลลัพธ์กลับออกมาทางจอภาพแต่เราสามารถเปลี่ยนทิศทางของช่องทางการนำข้อมูลเข้าจากคีย์บอร์ดไปเป็นนำข้อมูลเข้ามาจากไฟล์ได้ตามตัวอย่างต่อไปนี้

ทำการสร้างไฟล์ชื่อ "namelist" เพื่อเตรียมไว้ให้กับโปรแกรม sort

```
$ cat > namelist
Peter                       : พิมพ์ Peter
Mary                        : พิมพ์ Mary
John                        : พิมพ์ John เสร็จแล้วกด Ctrl-D
```

```
$ sort < namelist
John
Mary
Peter
```

จะเห็นว่าเราสามารถใช้เครื่องหมายน้อยกว่า "<" เพื่อทำการเปลี่ยนช่องทางการนำข้อมูลเข้าได้ กรณีนี้ STDIN จะเปลี่ยนไปเป็นไฟล์ที่ชื่อ "namelist"

- การเรียกคำสั่งเชลล์หลายคำสั่งในหนึ่งบรรทัด

เราสามารถสั่งให้ลีนุกซ์ทำงานหลายคำสั่งได้ภายในหนึ่งบรรทัด โดยจะต้องทำการแยกแต่ละคำสั่งด้วยเครื่องหมาย semicolon ";" ดังตัวอย่างต่อไปนี้ คำสั่งสุดท้ายไม่ต้องใส่ ";"

```
$date;
pwd;
echo  "Hello world"
Wed Sep 3 00:36:35 GMT+7 1997          ----- คำสั่ง date
/home/twg                              ----- คำสั่ง pwd
Hello world                            ----- คำสั่ง echo  "Hello world"
```

จะเห็นว่าเชลล์ทำงานและแสดงผลลัพธ์ตามลำดับคำสั่งที่เราพิมพ์ไปข้างต้น หากต้องการเปลี่ยนทิศทางของผลลัพธ์ของคำสั่งทั้งหมดจะต้องครอบคำสั่งทั้งหมดด้วยเครื่องหมายปีกกา "{}" หรือไม่ก็ครอบด้วยเครื่องหมายวงเล็บ "()" เพราะหากไม่ครอบไว้แล้วจะกลายเป็นการเปลี่ยนทิศทางเฉพาะคำสั่งสุดท้ายเท่านั้น

```
$ { date; pwd; echo "Hello world" } > outfile
```

หรือ `$ ( date; pwd; echo "Hello world" ) > outfile`

- การสร้างฟังก์ชั่นของเชลล์

เราสามารถทำการนิยามฟังก์ชั่นของเชลล์ขึ้นมาใหม่ได้ โดยที่ฟังก์ชันนั้นก็จะเป็นการนำเอาคำสั่งของเชลล์หลายๆ คำสั่งเข้ามารวมไว้ด้วยกันและเราก็สามารถเรียกใช้งานฟังก์ชั่นนั้นแทนได้ ตัวอย่างข้างล่างจะแสดงถึงการนิยามฟังก์ชัน

```
$ newfunction() { date; pwd; echo "Hello world" }
```

ตัวอย่างนี้จะแสดงถึงการเรียกใช้งานฟังก์ชัน จะเห็นว่าเป็นเพียงการเรียกชื่อฟังก์ชันก็สามารถพิมพ์ข้อความตามคำสั่งที่เราได้สร้างไว้ในฟังก์ชันได้

```
$newfunction
Web Sep 3 00:54:52 GMT+7 1997
/home/twg
Hello world
```

- การใช้งานไปป์ (pipe)

การใช้งานไปป์เป็นการนำเอาผลลัพธ์จากคำสั่งหนึ่งไปเป็นข้อมูลเข้าของอีกคำสั่งหนึ่งซึ่งโปรแกรมที่ใช้งานในลักษณะนี้จะถูกเรียกว่า "ตัวกรองข้อมูล" หรือ "ฟิลเตอร์ (filter)" ตัวอย่างของการใช้ไปป์

$ ls | grep file | sort

เป็นการเรียงลำดับข้อมูลที่ได้ จากการคัดเลือกรายชื่อไฟล์ที่มี ชื่อ "file" เป็นส่วนประกอบ

- ตัวแปรเชลล์

ตัวแปรเชลล์จะมีการเก็บค่าไว้ในลักษณะของสตริง ในกรณีที่จะใช้งานเป็นลักษณะของตัวเลขจะต้องใช้โปรแกรม "expr" เข้ามาช่วย การกำหนดค่าให้กับตัวแปรจะถือว่าเป็นการประกาศตัวแปรด้วยซึ่งจะมีรูปแบบดังต่อไปนี้

ชื่อตัวแปร=ค่าตัวแปร

โดยห้ามมีช่องว่าง (space) คั่นระหว่างเครื่อง หมายเท่ากับ ตัวอย่างเช่น

NAME=KLAUSE

หากมีการกำหนดค่าที่เป็นประโยคข้อความยาวๆที่มีช่องว่างอยู่ในประโยคนั้นจะต้องใช้เครื่องหมายเขาคู่ครอบประโยคนั้น เช่น

NAME="Peter Klause"

การตั้งชื่อของตัวแปรสามารถใช้ตัวอักษร ตัวเลข ตัวขีดเส้นใต้ ห้ามใช้ตัวอักษรพิเศษต่างๆ เช่น $, *, #, () และห้ามขึ้นต้นด้วยตัวเลข ตัวอย่างชื่อตัวแปร เช่น VAR1 VAR_NAME Variable แต่โดยทั่วไปแล้วการตั้งชื่อของตัวแปรมักจะใช้ตัวอักษรตัวใหญ่ในการตั้งชื่อการใช้งาน ตัวแปรเชลล์หรือการอ้างถึงค่าของตัวแปรนั้นจะต้องใช้เครื่องหมาย "$" นำหน้าตัวแปรนั้นเช่นถ้าใช้คำสั่ง $echo $NAME ในกรณีที่กำหนดค่าของตัวแปรเป็น "Peter Klause" จะได้ผลลัพธ์เป็น Peter Klause แต่ถ้าใช้คำสั่ง $echo NAME เนื่องจากไม่ได้อ้างถึงค่าของตัวแปร ดังนั้นจะได้ผลลัพธ์เป็น NAME

ในกรณีที่จะทำให้เชลล์สามารถแยกชื่อของตัวแปรออกจากค่าอื่นๆ เราสามารถทำได้โดยทำการครอบชื่อของตัวแปรนั้นด้วยเครื่องหมายปีกกา "{}" ดังตัวอย่าง

$ MYDIR=/home/mary/

$ cat ${MYDIR}myfile

คำสั่งข้างต้นเป็นการสั่งให้ทำการพิมพ์รายละเอียดของแฟ้มที่ชื่อ /home/mary/myfile (ซึ่งเป็นการเอาค่าของตัวแปร (/home/mary/) มาต่อกับชื่อไฟล์ (myfile)) ออกมาให้ หากเราไม่ครอบชื่อของตัวแปรด้วยเครื่องหมายปีกกาแล้วเชลล์จะมองชื่อตัวแปรเป็น $MYDIRmyfile (ซึ่งไม่มีค่าอะไรอยู่) ไม่ใช่ $MYDIR ตามที่เราต้องการ การยกเลิกตัวแปรสามารถทำได้โดยใช้คำสั่ง

$ unset ชื่อตัวแปร

- ขอบเขตของตัวแปรเชลล์

ตัวแปรเชลล์ที่ถูกกำหนดขึ้นมาจะมีลักษณะเป็นตัวแปรท้องถิ่นซึ่งจะรู้จัก แต่เฉพาะในเชลล์ของชั้นตนเองเท่านั้นเชลล์ในชั้นอื่นๆ จะไม่รู้จักตัวแปรดังกล่าวจนกว่าเราได้ทำการประกาศตัวแปรให้เชลล์ชั้นอื่นๆได้รู้จักซึ่งสามารถทำได้โดยใช้คำ สั่ง "export"

การเรียกเชลล์ซ้อนกันหลายชั้นหรือซับเชลล์ (sub shell) นั้นหมายถึงเราพิมพ์คำสั่ง "sh" หรือ "ksh" ซึ่งจะเป็นการเรียกโบนเชลล์และคอร์นเชลล์ซ้อนขึ้นมาจากเชลล์ปัจจุบัน เมื่อเรียกขึ้นมาแล้วจะเห็นเป็นเครื่องหมาย prompt ของระบบตามปกติ ซึ่งจะหมายความว่าซับเชลล์ได้ถูกเรียกขึ้นมาใหม่แล้ว เราอาจใช้คำสั่ง "ps" เพื่อ ตรวจดูว่าซับเชลล์ถูกเรียกขึ้นมาหรือไม่

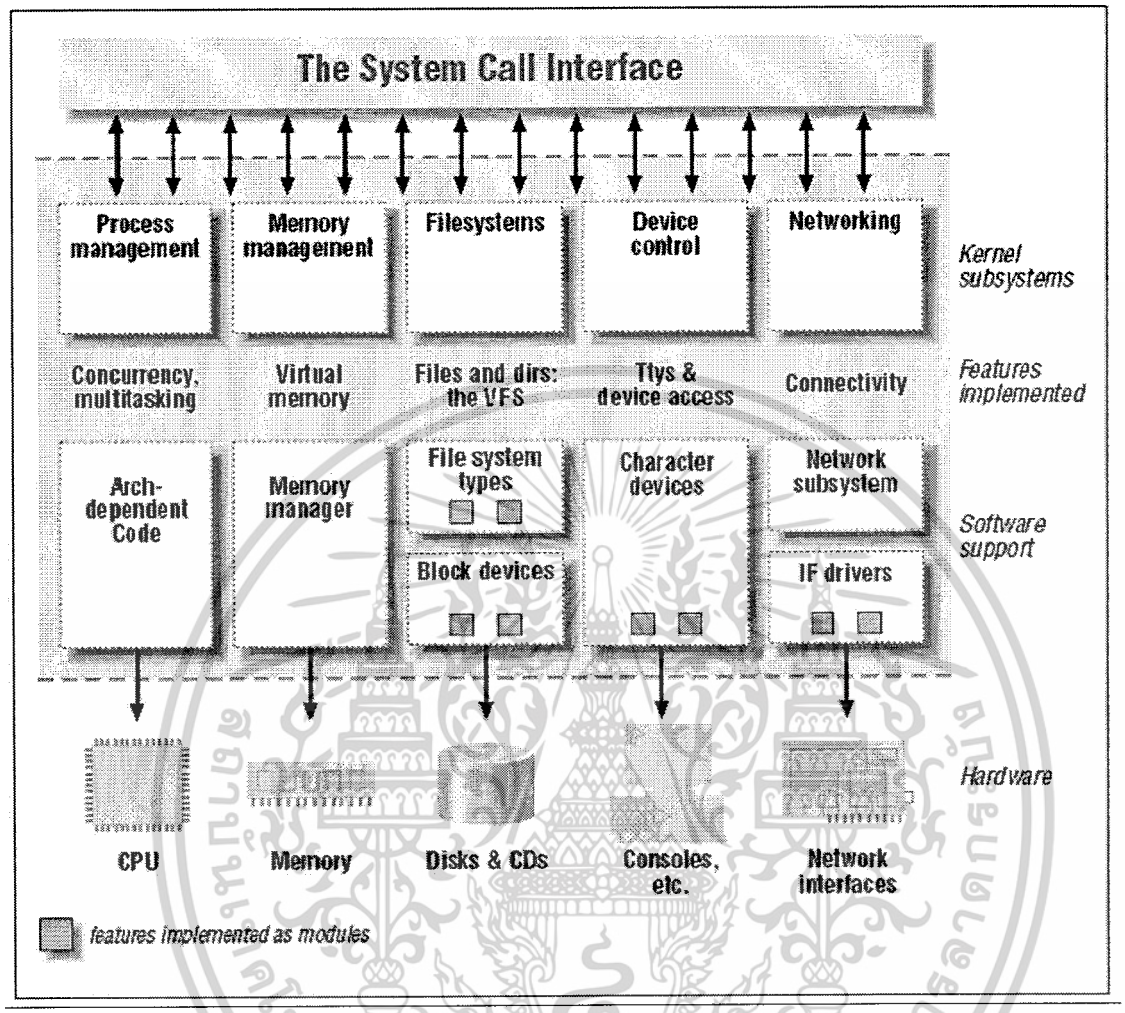
ตัวอย่างการเรียกซับเชลล์และการ ประกาศตัวแปรให้เชลล์ชั้นอื่นๆ ได้รู้จัก

$ NAME=KLAUSE	: ประกาศตัวแปร NAME ในเชลล์ชั้นปัจจุบัน
$ sh	: เรียกซับเชลล์ซ้อนขึ้นมา
$ echo $NAME	: ไม่พิมพ์ผลลัพธ์ใดๆ ออกมาเนื่องจากซับเชลล์ไม่รู้จัก
$ exit	: ออกจากซับเชลล์กลับมาสู่เชลล์ปัจจุบัน
$ export NAME	: ประกาศตัวแปร NAME ให้กับเชลล์ชั้นอื่นๆ
$ sh	: เรียกซับเชลล์อีกครั้งหนึ่ง
$ echo $NAME	
KLAUSE	: คราวนี้ซับเชลล์รับรู้ตัวแปร NAME แล้วจึงพิมพ์ผลลัพธ์ออกมาให้
$ readonly NAME	: คำสั่งนี้จะทำให้ไม่สามารถเปลี่ยนแปลงค่าตัวแปร

- ตัวแปรสงวน หรือตัวแปรของระบบ

ในระบบลีนุกซ์และยูนิกซ์จะมีตัวแปรสงวนที่ตัวโอเอสจะนำไปใช้ประโยชน์โดยเฉพาะเท่านั้น แต่หากเรารู้ความหมายของตัวแปรเหล่านั้นเราก็สามารถแก้ไขเปลี่ยนแปลงค่าของตัวแปรดังกล่าวได้ เราสามารถตรวจสอบค่าของตัวแปรระบบได้

2.5.5 การติดต่อระหว่างอุปกรณ์กับระบบปฏิบัติการลีนุกซ์



รูปที่ 2.17 แสดงโครงสร้างการทำงานในระดับเคอร์เนล

โพรเซสทุกโพรเซสนอกจากจะต้องเกี่ยวข้องกับหน่วยประมวลผลกลางกับหน่วยความจำแล้ว บางครั้งยังต้องมีการติดต่อกับอุปกรณ์อื่นๆ ด้วย เพื่อให้การทำงานสำเร็จโดยโพรเซส หากต้องการติดต่อกับอุปกรณ์หรือฮาร์ดแวร์ใด โพรเซสจะต้องติดต่อตามช่องทางที่ระบบปฏิบัติการได้มีไว้ให้ นั่นก็คือ system call จากนั้นระบบปฏิบัติการจะไปทำการเรียกเคอร์เนลเพื่อให้มารับคำร้องขอการใช้งานของโพรเซสต่อไป ซึ่งในการร้องขอใช้ทรัพยากรแต่ละตัวก็จะมีเคอร์เนลมารองรับแตกต่างกันไปโดยแยกได้ดังนี้

- Process Management เป็นส่วนที่ใช้สำหรับจัดการโพรเซสต่างๆ มีหน้าที่ในการสร้างหรือทำลาย โพรเซส รับผิดชอบเกี่ยวกับการติดต่อระหว่างโพรเซสหรือกับอุปกรณ์รับข้อมูล/แสดงข้อมูล(I/O device)และรับผิดชอบการจัดการตารางการใช้งานหน่วยประมวลผลกลางของแต่ละโพรเซสอีกด้วย

- Memory Management เป็นส่วนที่จัดการเกี่ยวกับ การใช้งานหน่วยความจำรับผิดชอบการสร้างหน่วยความจำเสมือน เพื่อรองรับโพรเซสบางส่วนที่ไม่สามารถถูกจัดเก็บไว้ในหน่วยความจำหลักได้เพียงพอ
- File systems เนื่องจากระบบปฏิบัติการลีนุกซ์ (ซึ่งมีพื้นฐานจากระบบปฏิบัติการยูนิกซ์) มักมองทุกสิ่งอยู่ในรูปของไฟล์ทำให้ขนาดงานของส่วนนี้ จะมีขนาดใหญ่โดยหน้าที่หลักๆ ของเคอร์แนลในส่วนนี้คือ ต้องทำการสร้างระบบไฟล์บนอุปกรณ์ที่ไม่มีโครงสร้างให้เกิดมีโครงสร้างที่ลีนุกซ์เข้าใจได้
- Device Control การทำงานทุกอย่างของระบบจะต้องเกิดขึ้นที่อุปกรณ์ต่างๆ ซึ่งอุปกรณ์เหล่านี้จะถูกควบคุมด้วย "โค้ดโปรแกรม" ซึ่งนั่นคือ ไดร์เวอร์ (devicedriver) ซึ่งโค้ดคำสั่งต่างๆ ก็แตกต่างกันไปขึ้นอยู่กับอุปกรณ์ แม้กระทั่งอุปกรณ์ประเภทเดียวกันและยี่ห้อเดียวกัน ก็อาจมีไดร์เวอร์ที่แตกต่างกันซึ่งอาจจะให้ ฟังก์ชันการทำงานที่แตกต่างกันตามไปด้วย
- Networking รับผิดชอบการทำงานเกี่ยวกับเครือข่าย โดยมักจะถูกจัดการโดยระบบปฏิบัติการ อินเทอร์เฟสหรืออุปกรณ์เครือข่ายจะทำการส่งข้อมูลและรับข้อมูลเท่านั้น โดยตัวอุปกรณ์เองจะไม่สนใจว่า ข้อมูลแพ็กเก็ตดังกล่าวนั้น เป็นของโพรเซสใด เนื่องจากแพ็กเก็ตมักจะถูกส่งมาอย่างไม่เกี่ยวข้องกัน หรือไม่ต่อเนื่องกัน โดยอินเทอร์เฟสจะทำการเก็บแพ็กเก็ตและถอดข้อความออกก่อนที่จะส่งไปให้โพรเซสต่อไป โดยระบบจะคอยทำหน้าที่ในการจัดการส่งข้อมูลระหว่างโปรแกรมกับอุปกรณ์เครือข่าย นอกจากนั้นเคอร์แนลในส่วนนี้จะต้องจัดการในส่วนงานจำพวก การสร้างเส้นทางเดินข้อมูล(routing) การคำนวนหาตำแหน่ง(address resolution) ด้วย

ข้อดีของระบบปฏิบัติการลีนุกซ์ประการหนึ่งคือ ขณะระบบปฏิบัติการทำงานอยู่สามารถทำการเพิ่มฟีเจอร์ต่างๆ ให้กับลีนุกซ์ได้ โดยส่วนของโค้ดคำสั่งที่เพิ่มเข้าไปโดยจะเรียกว่า "module" โดยแต่ละโมดูลจะต้องเป็นอ๊อบเจ็คโค้ดเท่านั้น โดยในการทำงานของโมดูลจะแยกตามชนิดของอุปกรณ์โดยในลีนุกซ์แบ่งอุปกรณ์ออกเป็น 3 ประเภทคือ

- Character Device อุปกรณ์ชนิดนี้จะทำการส่งข้อมูลอยู่ได้ทีละ 1 ไบท์เท่านั้นโดยคำสั่งพื้นฐานที่มักใช้คือ read write open และ close ในการติดต่อกับอุปกรณ์ประเภทนี้ต้องทำการติดต่อผ่านไฟล์โหนด โดยอุปกรณ์ประเภทนี้มักเป็นอุปกรณ์ที่ใช้ในการส่งรับข้อมูลเป็นลำดับ (sequential)
- Block Device อุปกรณ์ชนิดนี้ทำการส่งข้อมูลทีละหลายๆ ไบท์ โดยลักษณะของอุปกรณ์ประเภทนี้คือ เป็นอุปกรณ์ที่มีระบบไฟล์เป็นของตนเองโดยการใช้งานจะต้องมี mount point จำพวก เช่น ฮาร์ดดิสก์ ซีดีรอม เป็นต้น

- Network Interface การทำงานเครือข่ายจะต้องผ่านอินเทอร์เฟส ซึ่งจะมีหน้าที่ในการส่งข้อมูลตามที่เคอร์แนลสั่งการมา แต่ไม่รู้ว่างานที่ทำนั้นเป็นของทรานแซคชั่นใด การทำงานจะแตกต่างกับสองชนิดแรกคือ จะไม่ยุ่งเกี่ยวกับไฟล์เคอร์แนลจะทำการเรียกไดร์เวอร์โดยตรง

โมดูลจะทำหน้าที่คอยรับคำร้องขอ (request) จากโพรเซสต่างๆ ซึ่งอาจเกิดขึ้นเมื่อไรก็ได้ เพราะฉะนั้นโมดูลจะมีความแตกต่างกับโปรแกรมในระดับของยูสเซอร์สเปซ (user-space) ทั่วๆ ไป คือจะไม่มีฟังก์ชัน main() ที่เป็นฟังก์ชันเมื่อหลักในการทำงาน และเนื่องจากโมดูลนั้นฝังตัวอยู่ในเคอร์แนลสเปซ (kernel-space) ไลบารีต่างๆ จะต้องนำมาจากสิ่งที่มีอยู่ในระดับเคอร์แนลเท่านั้นไม่สามารถเชื่อมโยง(include) ได้เหมือนปกติที่เขียนในระดับของ user-space

ตัวอย่างในการเขียนโมดูล (hello.c)

```
#include <linux/kernel.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL"); //Display license
static int init_moudule(void){
printk(This is hello Module\n");
return 0; }
static void cleanup_module(void){
printk("Goodluck\n"); }
```

โดยในการเพิ่มโมดูลลงในเคอร์ในสามารถทำได้โดยใช้คำสั่ง insmod

```
root# insmod ./hello.o
This is Hello Module
```

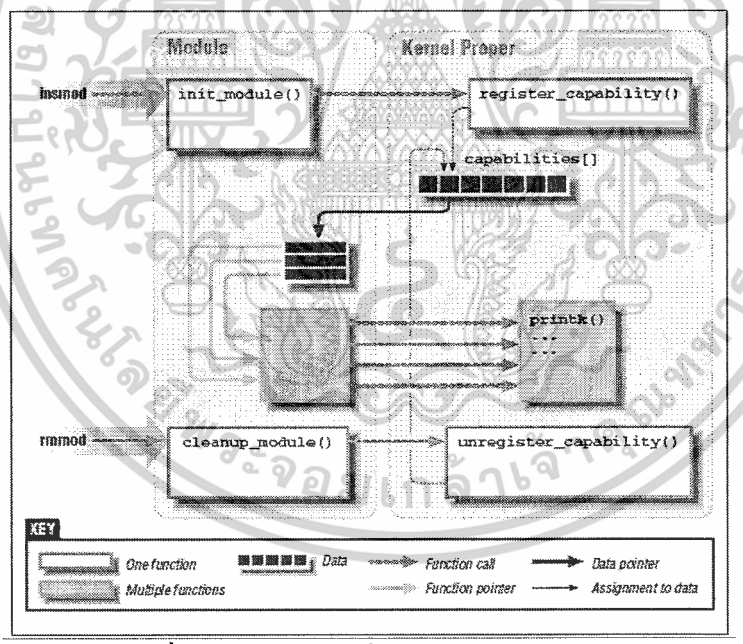
ผลลัพธ์ที่ได้จากการเพิ่มโมดูล hello (โค้ดตามด้านบน) คือมีข้อความว่า “This is Hello Module” ตามคำสั่งที่ได้เขียนไว้ในข้างต้น(หากไม่พบข้อความใดๆ ปรากฏให้ใช้คำสั่ง dmesg หรือดูข้อมูลในไฟล์ /var/log/kern.log) โดยเมื่อโปรแกรมถูกเพิ่มเข้าไปในเคอร์แนล จะทำการเรียกฟังก์ชัน “intinit_module(void)” ซึ่งภายในฟังก์ชันดังกล่าว มีการสั่งให้พิมพ์ “This is Hello Module” ดังกล่าวออกมาเช่นเดียวกันกับฟังก์ชัน “void cleanup_module(void)” ซึ่งจะถูกเรียกเมื่อทำการเอาโมดูลนี้ออกโดยใช้คำสั่ง rmmod โดยทั้ง init_module() และ cleanup_module() สามารถเปลี่ยนแปลงชื่อฟังก์ชันได้โดยต้องทำการเรียกฟังก์ชัน moudule_init(ชื่อฟังก์ชัน) และ module_exit(ชื่อฟังก์ชัน)

root# rmmod hello

Goodluck

สังเกตว่าในโค้ดคำสั่งด้านบนจะไม่มีการใช้ไลบารีหรือฟังก์ชันที่พบในภาษาซี ที่ถูกเขียนในระดับของแอพพลิเคชั่นเลย เนื่องจากโค้ดดังกล่าวนั้นถูกเขียนเพื่อให้ทำงานในระดับของเคอร์แนล เพราะฉะนั้นไลบารีและคำสั่งต่างๆ ต้องเป็นของเคอร์แนลเท่านั้น แต่โดยทั่วไปลักษณะการใช้งานก็ยังคงเป็นภาษาซีอาจมีความแตกต่างกันบ้างเล็กน้อย โดยสามารถศึกษาไลบารีของเคอร์แนลในลีนุกซ์ได้ใน /usr/include/linux หรือ /usr/include/asm แต่โดยหลักๆ แล้วในการเขียนโมดูลต้องมีการใช้ไลบารี module.h

นอกจากนี้ในการสร้างโมดูลจะต้องมีการประกาศความสามารถของโมดูลด้วย เพื่อบ่งบอกว่าแอพพลิเคชั่นต่างๆ สามารถเรียกใช้การทำงานจากโมดูลนี้มีอะไรบ้าง โดยในการประกาศจะต้องประกาศด้วยเป็นตัวแปรชนิดโครงสร้างและในการเรียกใช้งานโมดูลนั้นจะต้องถูกเรียกใช้งานผ่านไฟล์ซึ่งจะต้องสร้างโหนดขึ้นมาเพื่อรองรับการเรียกใช้งานจากระดับแอพพลิเคชั่น



รูปที่ 2.18 แสดงการเชื่อมโมดูลกับเคอร์เนล

2.6 GCC

GCC ย่อมาจาก GNU C Compiler เป็นโปรแกรมที่สำคัญและจะขาดไม่ได้เลย หากต้องการพัฒนาโปรแกรมด้วยภาษา C เพราะโปรแกรม GCC จะช่วยแปล (Compile) ภาษา C ให้เป็นภาษาเครื่อง (Machine Language) และทำการ Link เพื่อให้เกิด File ที่สามารถนำมาทำงานได้ (Executable File) ที่เรียกกันว่า ตัวแปลภาษา หรือ Compiler

ขั้นตอนการทำงานของ GCC

Preprocessing

เป็นกระบวนการที่รับภาษา C (Source File) เข้ามา แล้วทำหน้างานดังนี้

- นำเอาคำสั่งหมายเหตุ (Comment) ออกจากโปรแกรม
- ทำงานตาม Preprocessor Directives (บรรทัดที่ขึ้นต้นด้วย #) ซึ่งมีหลายคำสั่ง เช่น #include, #define

Compilation

เป็นกระบวนการแปลภาษา C (Source File) ที่ได้จากกระบวนการ Preprocessing เป็นภาษา Assembly (Assembly Code)

Assembly

เป็นกระบวนการแปลภาษา Assembly เป็น Object Code ใน ระบบ UNIX ซึ่งจะเป็นภาษาเครื่อง แต่ยังไม่สามารถนำมาทำงานได้ เพราะยังไม่ได้ทำการ Link เข้ากับ Library

Linking

เป็นกระบวนการที่นำเอา Object Code มา Link เข้ากับ Library จะทำให้เกิด File ที่สามารถทำงานได้ คือ Executable File

GCC มีรูปแบบการใช้งานดังนี้

gcc [option | filename] : ซึ่ง Option mสำคัญๆ ซึ่งใช้ในการพัฒนาโปรแกรมมีดังนี้

-c file

เป็นการ Compile Source File แต่ไม่ทำการ Link จะทำให้เกิด Object File ที่มีชื่อเหมือนกับ Source File ที่ถูก Compile เช่น หาก Source File ชื่อ test.c ใช้คำสั่ง

gcc -c test.c

จะได้ File ชื่อ test.o หากผลการ Compile นั้นถูกต้อง

-o file

เป็นการกำหนดชื่อ Output File ให้เป็นไปตามชื่อ file ที่ระบุหลัง -o ในกรณีการ Compile ภาษา C โดยปกติ Output File ที่ได้จะเป็น Executable File ยกตัวอย่างเช่น การใช้คำสั่ง

gcc -o test test.c

คือการสั่งแปล Source File ภาษา C ชื่อ test.c หากผลการ Compile ถูกต้องจะได้ Executable File ชื่อ test

-g

เป็นการ Compile โดยเปิด Debugging Information สำหรับโปรแกรม GDB

File ที่ควรรู้จัก มีดังนี้

- File.c เป็น C Source File
- File.h เป็น C Header (Preprocessor) File
- File.o เป็น Object File
- a.out เป็น Link Edited Output ในกรณีที่ไม่ใช้ Option -o กำหนดชื่อ Output File

2.7 Libpcap

Libpcap (libraly for capture Packet) เป็นไลบราลีที่ใช้สำหรับดักจับแพ็กเก็ตที่ได้รับความนิยมสูงชนิดหนึ่ง ซึ่งถูกพัฒนาเพื่อนำไปใช้ต่ออย่างแพร่หลาย โดยในปัจจุบันได้มีซอฟต์แวร์สำเร็จรูปที่ถูกพัฒนาขึ้นมาด้วย libcap ที่รู้จักกันดี ได้แก่ Ethereal tcpdumb Snort Nmap และ Ntop จุดเด่นของไลบราลี libpcap คือมีรูปแบบ API ที่ใช้งานง่ายแต่มีประสิทธิภาพสูง โดยสามารถจับแพ็กเก็กได้ลึกถึงระดับดาต้าลิงค์ทั้งในโหมดปกติ (normal) และโหมดโพมิสคิวอัส (promiscuous) อีกทั้งยังมีความสามารถในการเลือกจับเฉพาะบางชนิดของแพ็กเก็ตผู้ใช้ที่สนใจและที่สำคัญ libpcap เป็น open-source library ซึ่งสามารถให้นำมาใช้งานได้ฟรี

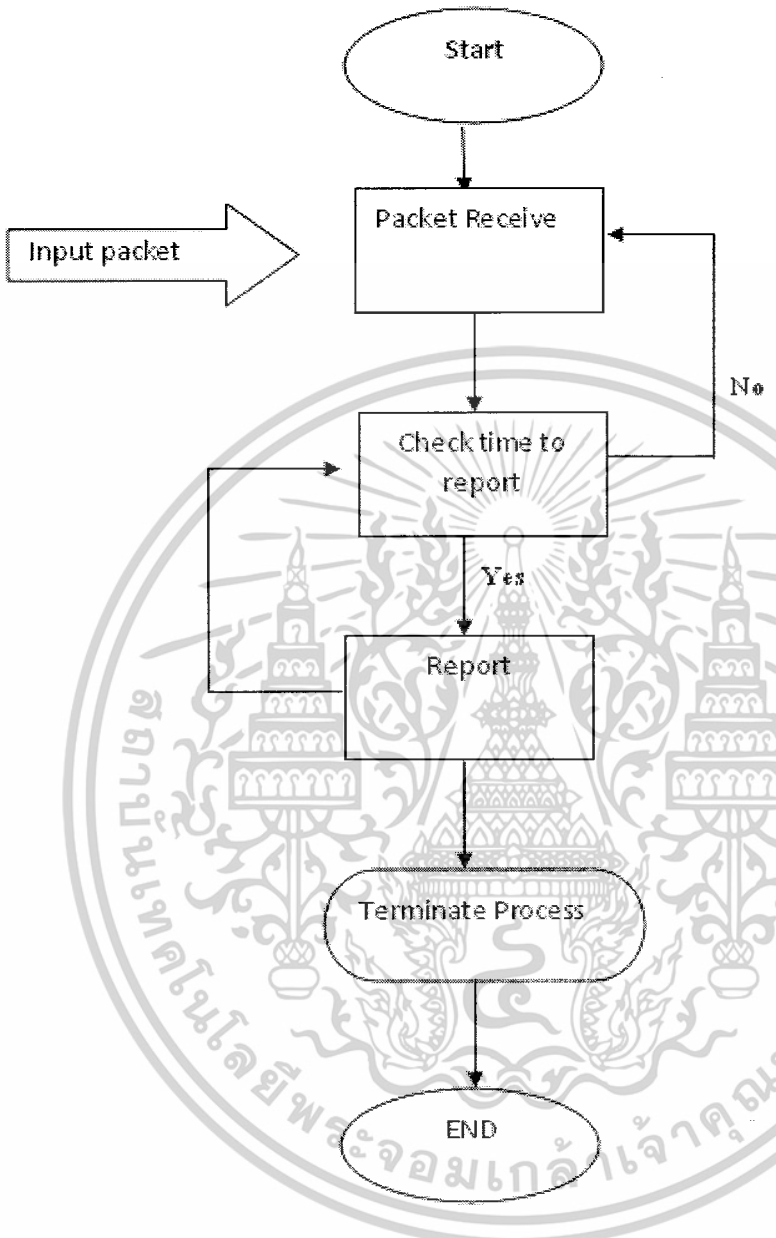
ตัวอย่างฟังก์ชั่นที่สำคัญ

1. pcap_t *pcap_open_live(const char *device int snaplen, int promisc, int to_ms ,char *errbuf)
   เป็นฟังก์ชันที่ทำหน้าที่ในการเปิด Sessoin ในการดักจับแพ็คเก็ตต้องเรียกทุกครั้ง
2. int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
   เป็นฟังก์ชั่นที่ใช้ในการดักจับแพ็คเก็ตแบบง่ายๆ
3. pcap_close(handle);
   เป็นฟังก์ชั่นที่ใช้มนการทำการปิด Session

บทที่ 3
การออกแบบโครงงาน

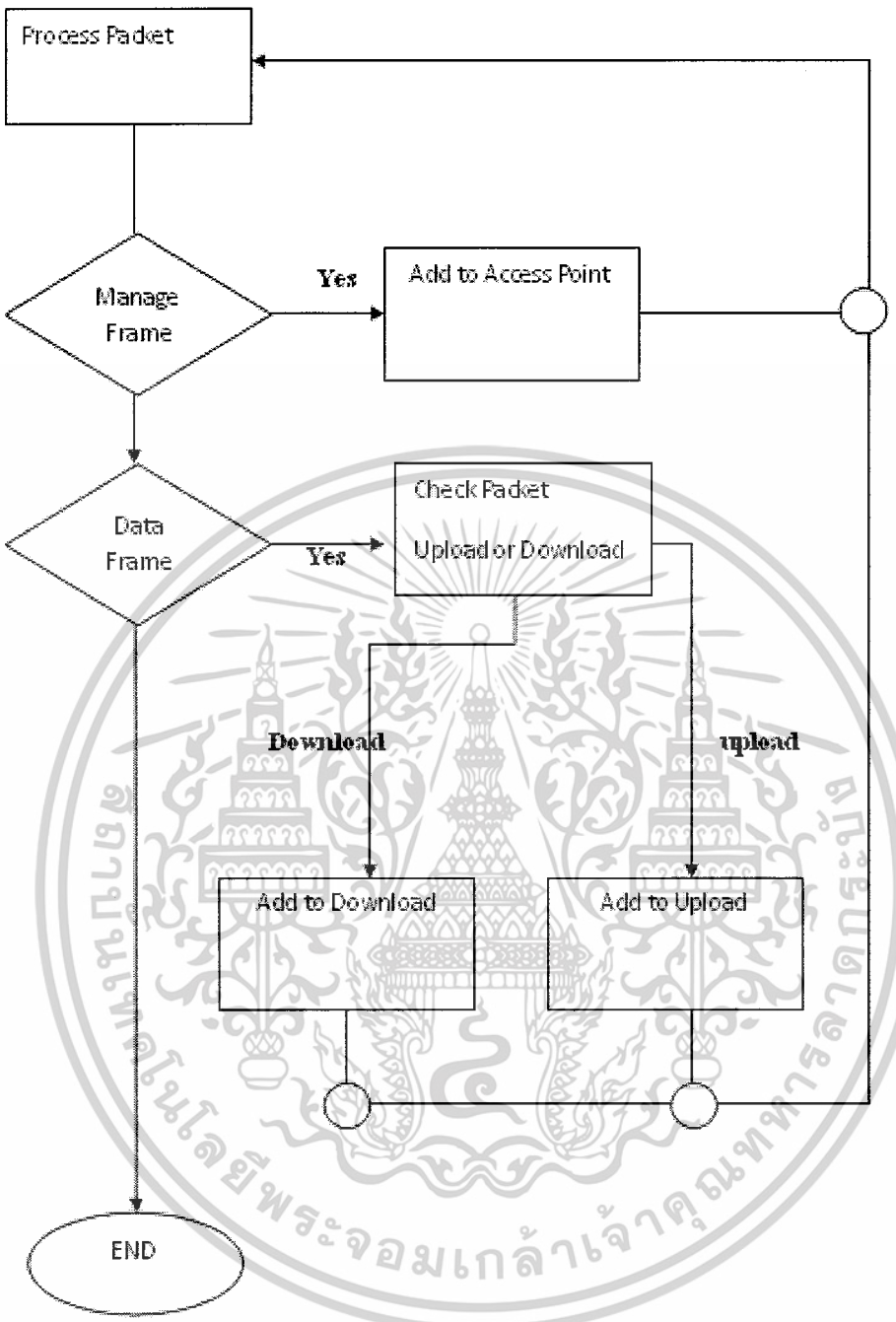
3.1 ภาพรวมในการออกแบบโครงงาน

1. การดักจับข้อมูล โปรแกรมของเราต้องทำการดักจับเฟรมข้อมูลทุกเฟรมข้อมูลที่อยู่ภายในระบบ เครือข่ายแลนแบบไร้สายได้และแยกชนิดของเฟรมต่างๆ ที่โปรแกรมดักจับได้
2. การแสดงผลของตัวโปรแกรม โปรแกรมของเราเลือกที่จะแสดงผลในส่วนของจำนวนแอคเซสพอยต์ภายในระบบทั้งหมดที่มี โดยตัวโปรแกรมเลือกที่จะแสดงค่าแมคแอดเดรสของแต่ละแอคเซสพอยต์ แสดงชื่อ SSID ของแต่ละแอคเซสพอยต์ และโปรแกรมจะแสดงคู่สัญญาณที่มีการเชื่อมต่อกันภายในระบบ เช่น ไคลเอนท์ตัวนี้กำลังติดต่ออยู่กับแอคเซสพอยต์ตัวไหนภายในระบบ โดยจะแสดงเป็นค่าแมคแอดเดรสของแอคเซสพอยต์และแมคแอดเดรสของไคลเอนท์
3. การแจ้งเตือนในเรื่องความปลอดภัย เราจะมีการแสดงค่าที่บอกถึงความผิดปกติของคู่สัญญาณแอคเซสพอยต์กับไคลเอนท์ตัวใดที่โปรแกรมตรวจสอบว่ามีความผิดปกติ



รูปที่ 3.1 แสดงการทำงานโดยรวมของโปรแกรม

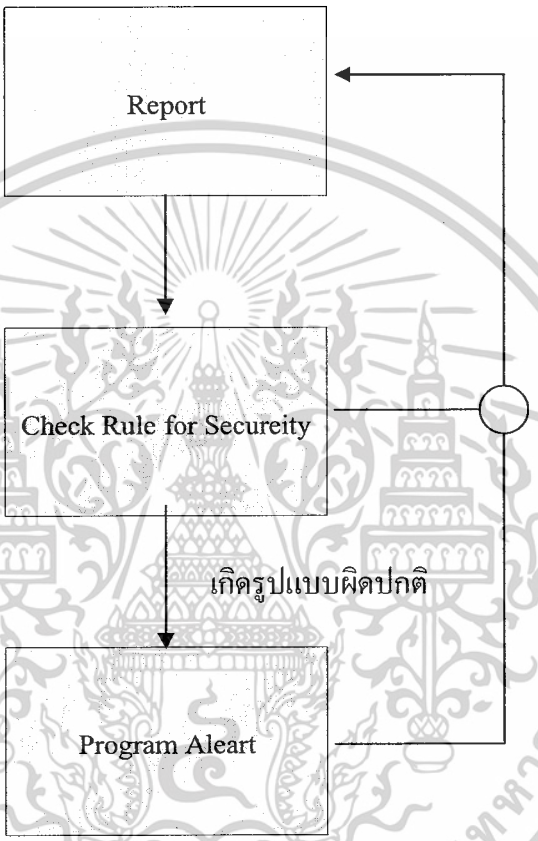
จากรูปที่ 3.1 การทำงานของตัวโปรแกรมเรานั้นเมื่อโปรแกรมเริ่มทำงานโปรแกรมจะทำการดักจับแพ็คเก็ตที่เข้ามาเรื่อยๆ โดยจัดเก็บข้อมูลของแพ็คเก็ตแต่ละแพ็ตเก็ตลงในตัวแปรที่โปรแกรมได้สร้างขึ้น โดยโปรแกรมจะทำการตรวจสอบเวลา เพื่อที่จะทำการแสดงผลสถานะของระบบในเวลานั้น เมื่อถึงเวลาโปรแกรมก็จะทำการแสดงผลแล้วทำการเคลียค่าแพ็คเก็ตต่างๆ ที่เก็บไว้แล้วเริ่มทำการจับแพ็คเก็ตใหม่ แล้วทำการเริ่มตรวจสอบเวลาอีกครั้งไปเรื่อยๆ จนกว่าจะมีคำสั่งให้หยุดการทำงานของโปรแกรม



รูปที่ 3.2 แสดงขั้นตอนการแยกเฟรมและบันทึกค่าเฟรมแต่ละชนิด

จากรูปที่ 3.2 แสดงขั้นตอนการออกแบบโปรแกรม ในการแยกเฟรมแต่ละชนิดแล้วทำการเก็บข้อมูลของเฟรมแต่ละชนิดลงในตัวแปร ในฟังก์ชั่น Process Packet จะเป็นฟังก์ชั่นที่ทำงานเมื่อมีข้อมูลเข้ามาโดย Process Packet จะทำการตรวจสอบว่าแพ็คเก็ตที่เข้ามาเป็นชนิดอะไร โดยถ้าเป็น Management Frame ก็ส่งไปเก็บข้อมูลที่ฟังก์ชั่น AddtoAccessPoint ถ้าไม่ใช่ก็ตรวจสอบต่อว่าเป็น Data Frame หรือไม่ ถ้าใช่ Data Frame ก็ต้องทำการตรวจสอบอีกว่าเป็นเฟรมชนิดอัพโหลดหรือ

ดาวน์โหลด ถ้าเป็นเฟรมชนิดอัพโหลดก็ส่งไปที่ฟังก์ชัน AddToUpload เพื่อทำการเก็บข้อมูล หรือถ้าเป็นดาวน์โหลดก็จะส่งไปที่ฟังก์ชัน AddToDowload เพื่อทำการเก็บข้อมูลต่อไป ส่วนถ้าไม่ใช่แพ็คเก็ตชนิด Data Frame ก็จะเป็นชนิด Control Frame ซึ่งโปรแกรมของเราจะไม่ทำอะไรกับแพ็ตเก็ตชนิดนี้ก็จะปล่อยผ่านไป



รูปที่ 3.3 แสดงขั้นตอนการออกแบบในด้านความปลอดภัย

จากรูปที่ 3.3 แสดงขั้นตอนการออกแบบในส่วนของด้านความปลอดภัย การตรวจสอบความปลอดภัยของตัวโปรแกรมนั้นจะมีการตรวจสอบความปลอดภัยในขั้นตอนการแสดงผลของโปรแกรมเมื่อโปรแกรมทำการแสดงผล (Report) โปรแกรมจะทำการตรวจสอบว่า คู่สัญญาณภายในระบบคู่ไหนมีค่าที่ผิดปกติหรือไม่ ถ้าคู่สัญญานคู่ไหนผิดปกติระบบก็จะทำการแจ้งเตือนออกมา โดยเมื่อทำการแจ้งเตือนแล้วระบบก็จะทำการทำงานต่อไป โดยระบบจะทำเพียงการแสดงค่าที่ผิดปกติให้ผู้ใช้ได้ทราบเท่านั้น

บทที่ 4
การพัฒนาโครงงาน

4.1 ภาพรวมของการพัฒนาโครงงาน

โครงงานนี้ได้พัฒนาขึ้นมาโดยมีวัตถุประสงค์ในการดักจับแพ็คเก็ตของเฟรมข้อมูลที่ทำการรับส่งกันภายในระบบเครือข่ายแลนแบบไร้สาย โดยแพ็คเก็ตที่เราดักจับมาได้นั้น จะมีการนำมาวิเคราะห์รูปแบบของแพ็คเก็ตแล้วนำไปแสดงผลในส่วนต่างๆ ของเฟรมข้อมูลสัญญาณไวร์เลสได้ และข้อมูลที่เราดักจับมาได้นั้นเราสามารถที่จะเก็บบันทึกเป็นไฟล์ข้อมูลเพื่อที่จะนำไปวิเคราะห์รูปแบบของสัญญาณที่เกิดขึ้นในระบบเครือข่ายแลนแบบไร้สายต่อไป

ในโครงงานนี้ผู้พัฒนาได้พัฒนาโปรแกรมโดยใช้ภาษาซีบนระบบปฏิบัติการลินุกซ์ ซึ่งระบบปฏิบัติการลินุกซ์เป็นระบบปฏิบัติการคอมพิวเตอร์ที่เหมาะกับการพัฒนาซอฟต์แวร์ที่เกี่ยวข้องกับระบบเครือข่าย นอกจากนั้นระบบปฏิบัติการลินุกซ์ยังเป็นที่นิยม ดังนั้นจึงมีซอฟต์แวร์จำนวนมากพัฒนาขึ้นภายใต้ระบบนี้ ทำ ให้ง่ายและสะดวกต่อการนำไลบรารีที่มีการพัฒนาแล้วมาประยุกต์ใช้งาน โดยมีการใช้ ไลบรารีชื่อว่า libpcap (libraly for capture packet) เป็นไลบรารีหลักที่ใช้ในส่วนของการดักจับแพ็คเก็ตและบันทึกข้อมูลที่ดักจับได้

โดยในส่วนของการจับข้อมูล libpcap มีฟังก์ชันที่เอาไว้เรียกใช้ในการจับข้อมูล โดยข้อมูลที่จับมาได้นั้น โปรแกรมของเราจะสร้างหน่วยความจำเฉพาะขึ้นมาเพื่อรองรับข้อมูลที่มีการดักจับได้ โดยเราสามารถที่จะนำข้อมูลจากแพ็คเก็ต โดยเลือกเพียงบางส่วนที่เราต้องการขึ้นมาแสดงบนหน้าจอโปรแกรมที่เราได้ทำการเตรียมไว้เป็นการแสดงข้อมูลของแพ็คเก็ตต่างๆ ที่เราสามารถจับได้ในเวลานั้น

หลักการของการวิเคราะห์ข้อมูลและตรวจสอบความปลอดภัยของตัวโปรแกรมนั้น โปรแกรมของเราจะทำการแยกชนิดของแพ็คเก็ตแต่ละแพ็คเก็ตและพิจารณาไปตามเงื่อนไขของแต่ละชนิดแพ็คเก็ต โดยมีการวิเคราะห์ข้อมูลของแพ็คเก็ตเพื่อบ่งบอกว่าจากแพ็คเก็ตที่เราจับได้มาสามารถที่จะบ่งบอกอะไรได้บ้างและในเรื่องของการรักษาความปลอดภัย โปรแกรมของเราสามารถที่จะตรวจสอบรูปแบบของคู่สัญญาณที่ผิดปกติได้และสามารถทำการบันทึกค่าและรายละเอียดของคู่สัญญาณที่ผิดปกติภายในระบบ

4.2 การพัฒนาโครงงงานในส่วนการดักจับแพ็คเก็ต

การพัฒนาโครงงานในส่วนนี้นั้นเราได้นำไลบราลีที่ใช้ในการพัฒนาในส่วนของการดักจับแพ็คเก็ต ไลบรารีที่นำเข้ามาใช้มีชื่อว่า libpcap (library for capture packet) และต้องมีการทำการเซ็ตค่าอินเตอร์เฟสไร้สายที่ใช้งานในโครงงานนี้ โดยมีรายละเอียดและวิธีการใช้งานเบื้องต้นดังนี้

4.2.1 การใช้Libpcapเบื้องต้น

Libpcap เป็นไลบรารีที่ใช้สำหรับดักจับแพ็คเก็ตที่ได้รับความนิยมสูงชนิดหนึ่ง ซึ่งถูกพัฒนาเพื่อนำไปใช้ต่ออย่างแพร่หลาย โดยในปัจจุบันได้มีซอฟต์แวร์สำเร็จรูปที่ถูกพัฒนาขึ้นมาด้วย libยcap ที่รู้จักกันดี ได้แก่ Ethereal tcpdumb Snort Nmap และ Ntop จุดเด่นของไลบรารี libpcap คือมีรูปแบบ API ที่ใช้งานง่ายแต่มีประสิทธิภาพสูง โดยสามารถจับแพ็กเก็กได้ลึกถึงระดับดาต้าลิงค์ทั้งในโหมดปกติ (normal) และโหมดโพมิสคิวอัส (promiscuous) อีกทั้งยังมีความสามารถในการเลือกจับเฉพาะบางชนิดของแพ็กเก็ตผู้ใช้ที่สนใจและที่สำคัญ libpcap เป็น open-source library ซึ่งสามารถให้นำมาใช้งานได้ฟรี ทุกโปรแกรมที่จะเรียกใช้ API ของ libpcap จะต้องทำการ include ไฟล์ pcap.h และต้องคอมไพล์โปรแกรมด้วยพารามิเตอร์ –lpcap เช่น

```
$ gcc -o prog prog.c –lpcap
```

- การค้นหาอินเตอร์เฟส (Interface) เพื่อดักจับข้อมูล

ก่อนเราจะทำการดักจับแพ็คเก็ตเราต้องทราบก่อนว่าเครื่องเรามีอินเตอร์เฟซใดบ้างที่สามารถเข้าไปดักจับข้อมูลออกมาได้ ภายใน API ของ libpcap นั้นจะอ้างอิงถึงส่วนต่อประสานผู้ใช้ด้วยสตริงที่ระบุชื่อ เช่น eth0 eth1 และ lo ซึ่งหากทราบชื่อดังกล่าวก็สามารถระบุลงไป แต่หากไม่ทราบ libpcap ก็มีฟังก์ชันที่จะช่วยค้นหาชื่อของส่วนต่อประสานผู้ใช้ให้เราโดย โปรโตไทป์ของฟังก์ชัน

```
char *pcap_lookupdev(char * errbuf);
```

เมื่อเรียกใช้ฟังก์ชันนี้ ฟังก์ชันจะส่งค่ากลับมาเป็นชื่อของอินเตอร์เฟซ ตัวแรกที่ไม่ใช่ลูปแบ็ค (loopback) ซึ่งโดยทั่วไปแล้วมักจะเป็นอินเตอร์เฟซที่เราจะทำการดักจับข้อมูล หากมีความผิดพลาดเกิดขึ้นจากการเรียกฟังก์ชันข้อความแสดงสาเหตุของความผิดพลาดจะถูกส่งกลับมาทาง errbuf

ตัวอย่างการเรียกใช้ฟังก์ชั่น pcap_lookupdev

```
#include <stdio.h>
#include <pcap.h>
int main() {
char errbuf[PCAP_ERRBUF_SIZE];
pcap_if_t *devlist, *wk;
pcap_findalldevs(&devlist,errbuf);
for (wk=devlist;wk!=NULL;wk=wk->next)
printf("interface name: %s\n",wk->name==NULL?"":wk->name);
pcap_freealldevs(devlist);
return 0;
}
```

ผลลัพท์

```
interface name: ath0
```

ในกรณีที่มีหลายอินเตอร์เฟซอยู่บนเครื่องเดียวกันแทนที่จะปล่อยให้ฟังก์ชัน pcap_lookupdev() เป็นผู้เลือกอินเตอร์เฟซให้ ผู้ใช้อาจต้องการทราบข้อมูลจำเพาะของอินเตอร์เฟซทั้งหมดที่มีอยู่บนเครื่องก่อน เพื่อที่จะได้ตัดสินใจเลือกได้ถูกต้องก็สามารถใช้ฟังก์ชัน pcap_findalldevs() เพื่อแสดงรายชื่อของอินเตอร์เฟซทั้งหมดออกมา โดยฟังก์ชัน pcap_findalldevs() จะมีโปรโตไทป์ ดังนี้

```
int pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf);
```

พารามิเตอร์แรกจะเป็นการส่งผ่านโดยการอ้างอิง (pass by reference) กล่าวคือ alldevsp คือแอดเดรสของตัวแปรชนิด *pcap_if_t ที่เราผ่านเข้าไปเพื่อให้ฟังก์ชัน pcap_findalldevs() ทำการจองพื้นที่ในหน่วยความจำและสร้างลิสต์ (list) ของโครงสร้างของส่วนต่อประสานผู้ใช้อื่นๆ ต่อท้าย กล่าวคือ หลังจากการเรียกฟังก์ชันนี้ เราจะได้ linked-list ของข้อมูลชนิด pcap_if_t ที่มี alldevsp เป็นพอยเตอร์ที่หัวของลิสต์และเมื่อใช้งานเสร็จแล้ว ก็สามารถคืนหน่วยความจำส่วนนี้ได้ด้วยการเรียกฟังก์ชัน

```
void pcap_freealldevs(pcap_if_t *alldevsp);
```

ตัวอย่างการเขียนโปรแกรมค้นหาชื่ออินเตอร์เฟซบนเครื่องด้วยฟังก์ชัน pcap_findalldevs()

```
int main() {
char errbuf[PCAP_ERRBUF_SIZE];
pcap_if_t *devlist, *wk;
pcap_findalldevs(&devlist,errbuf);
for (wk=devlist;wk!=NULL;wk=wk->next)
printf("interface name: %s\n",wk->name==NULL?"":wk->name);
pcap_freealldevs(devlist);
return 0;
}
```

ผลลัพท์

```
interface name: ath0
interface name: any
interface name: lo
```

นอกจากชื่อของอินเตอร์เฟซแล้วใน pcap_if_t ยังบรรจุรายละเอียดอื่นๆ ดังจะเห็นได้จากโครงสร้างข้อมูลที่นิยามไว้ใน pcap.h

```
typedef struct pcap_if pcap_if_t;
        struct pcap_if {
        struct pcap_if *next;
char *name;
char *description;
struct pcap_addr *addresses;
bpf_u_int32 flags;
}
```

สิ่งที่ฟังก์ชัน pcap_findalldevs() สามารถรายงานให้ทราบ นอกจากชื่อของอินเตอร์เฟซแล้วยังมีข้อความอธิบาย (description) และคุณลักษณะเพิ่มเติมเกี่ยวกับอินเตอร์เฟซนั้นๆ (flags) ซึ่งในเวอร์ชันปัจจุบันใช้เพียงแค่รายงานว่าเป็นลูปแบ็คหรือไม่ หากอินเตอร์เฟซนั้นเป็นลูปแบ็คค่าคุณลักษณะเพิ่มเติมเกี่ยวกับส่วนต่อประสานผู้ใช้ก็จะถูกเซ็ตให้มีค่าเป็น PCAP_IF_LOOPBACK ในส่วนของแอดเดรสนั้นจะบรรจุข้อมูลทั้งหลายที่เกี่ยวกับเน็ตเวิร์คแอดเดรสของอินเตอร์เฟซ เช่น IPv4 address IPv6 address และ netmask ซึ่งจะขอกล่าวถึงรายละเอียดภายหลัง

- การทดลองดักจับแพ็คเก็ตอย่างง่าย

```
pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms ,char *errbuf )
```

ค่าที่ส่งกลับมาจะเป็น handle สำหรับไว้ใช้อ้างอิงในส่วนของพารามิเตอร์ของฟังก์ชัน ตัวแรกจะเป็นชื่อของอินเตอร์เฟซที่เราจะทำการดักจับ เช่น eth0 นอกจากระบุชื่อแล้วเรายังสามารถระบุเป็น "any" หรือ "NULL" ใน กรณีที่ต้องการจับแพ็กเก็ตที่เข้ามาจากทุกอินเตอร์เฟซสำหรับ snaplen คือขนาดของบัฟเฟอร์ที่จะให้จองเพื่อรอรับข้อมูลแพ็กเก็ตที่จะเข้ามา promisc เป็นการระบุว่าจะใช้โหมดโพมิสคิวอัสหรือไม่ ซึ่งหากเปิดการดักจับในโหมดนี้ทุกแพ็กเก็ตที่ผ่านเข้ามาจะถูกดึงเข้ามาทั้งหมด แม้ว่าแพ็กเก็ตนั้นจะไม่ได้จ่าหน้าถึงอินเตอร์เฟซนี้ก็ตาม to_ms คือค่า read timeout มีหน่วยเป็นมิลลิวินาที หากกำหนดเป็นศูนย์จะหมายความว่าไม่มี timeout และ *errbuf คือพอยเตอร์ชี้ตำแหน่งสำหรับเก็บข้อความแสดงความผิดพลาด (error message) ไว้สำหรับรายงานความผิดพลาดจากการเรียกฟังก์ชัน ซึ่งการเรียกฟังก์ชันที่ผ่านมาเป็นการเปิดเซสชันของการดักจับแพ็กเก็ตเท่านั้น การรอดักจับข้อมูลจริงๆ จะเกิดเมื่อเรียกฟังก์ชัน

```
const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h);
```

โดยที่ p ก็คือ handle ที่ได้มาจากการเรียก pcap_open_live() ก่อนหน้านี้ และ h คือตำแหน่งของหน่วยความจำที่จะใช้รองรับเฮดเดอร์ (header) ของข้อมูลที่จะเข้ามา ซึ่งส่วนนี้จะเป็นเฮดเดอร์ที่ libpcap เตรียมขึ้นมาเอง ไม่เกี่ยวข้องกับเฮดเดอร์ของโปรโตคอลใดๆ มีไว้เพื่อบอกให้เราทราบถึงข้อมูลเพิ่มเติมเกี่ยวกับแพ็กเก็ต ได้แก่ ts คือเวลาที่ได้รับแพ็กเก็ต caplen คือขนาดของข้อมูลที่จับมาและ len คือขนาดเต็มของแพ็กเก็ต ซึ่งโดยปกติ caplen จะมีค่าเท่ากับ len เว้นแต่ว่าแพ็กเก็ตที่เข้ามานั้นใหญ่กว่าขนาดของบัฟเฟอร์ที่ได้เตรียมไว้ข้อมูลที่จับได้ caplen จึงจะมีขนาดเล็กกว่า len โครงสร้างของ struct pcap_pkthdr เป็นดังนี้

```
struct pcap_pkthdr {
        struct timeval ts;
        bpf_u_int32 caplen;
        bpf_u_int32 len;
}
```

เมื่อมีการเรียกฟังก์ชัน pcap_next() โปรแกรมจะหยุดรอที่จุดนี้ จนกว่าจะมีแพ็กเก็ตผ่านเข้ามาในอินเตอร์เฟซที่กำลังดักรออยู่ ฟังก์ชันจึงจะรีเทิร์นค่าพอยเตอร์ที่ชี้ไบต์แรกของข้อมูลแพ็กเก็ตที่จับมาได้ ข้อมูลนี้จะเริ่มจากไบต์แรกของ datalink protocol header ตามด้วย network protocol header และ transport protocol header เรื่อยไปจนถึงไบต์สุดท้ายของ application payload (ถ้ามี) สำหรับ timestamp และขนาดของแพ็กเก็ตนั้นไม่ได้เป็นส่วนหนึ่งของข้อมูลแพ็กเก็ต จึงเป็นเหตุผลว่าทำไม libpcap จำเป็นต้องมีเฮดเดอร์พิเศษเพิ่มเติมขึ้นมาต่างหากและก่อนจะจบการทำงานก็ควรทำการปิดเซสชัน (session) ด้วย pcap_close()

```
#include <stdio.h>
#include <pcap.h>
#include <time.h>
int main() {
    char timestr[64];
    char errbuf[PCAP_ERRBUF_SIZE];
    const u_char *packet;
    pcap_t *handle;
    struct pcap_pkthdr header;
    handle = pcap_open_live("eth0", 1024, 1, 0, errbuf);
    packet = pcap_next(handle, &header);
    strftime(timestr, sizeof(timestr), "%d %b %Y %H:%M:%S %Z",
    localtime(&header.ts.tv_sec));

    printf("Packet of the size %d bytes received at %s\n",header.len, timestr);
    printf("Packet data :\n");
    for (pt=packet; pt!=packet+header.caplen; pt++) printf("%02X ",*pt);
    pcap_close(handle);
}
```

ผลลัพธ์

```
Packet of the size 76 bytes received at 26 Apr 2008 12:46:36 ICT

Packet data :
00 50 56 F2 9E 7A 00 0C 29 E0 C2 89 08 00 45 00 00 3E 31 EF 40 00 40 11 4A EB C0 A8
9E 81 C0 A8 9E 02 80 02 00 35 00 2A A8 49 8D 0E 01 00 00 01 00 00 00 00 00 00 03 77 77
77 06 67 6F 6F 67 6C 65 02 63 6F 02 74 68 00 00 1C 00 01
```

- การดักจับแพ็ตเก็ตแบบต่อเนื่อง

ในความเป็นจริงคงมีไม่บ่อยนักที่เราจะพอใจกับแพ็กเก็ตแรกเพียงแพ็กเก็ตเดียว เนื่องจากข้อมูลที่ส่งถึงกันโดยปกติมักจะมีขนาดใหญ่และมักจะกระจายอยู่ในหลายแพ็กเก็ต ดังนั้นแอ็พพลิเคชันที่จะเรียกใช้งาน libpcap จึงมักจะมีแนวโน้มที่จะต้องการกลไกในการจับข้อมูลแบบต่อเนื่อง วิธีหนึ่งที่สามารถทำได้ก็คือการสร้างลูป (loop) ขึ้นมาครอบ pcap_next() แต่ยังมีอีกวิธีหนึ่งที่สะดวกกว่านั่นก็คือการใช้ฟังก์ชัน pcap_loop() ที่ libpcap เตรียมไว้ให้ และนำกลไกของ callback function เข้ามาช่วย

callback function โดยทั่วไปจะหมายถึงฟังก์ชันที่ถูกกำหนดไว้ให้โปรแกรมมีการกระโดดไปเรียกเมื่อเกิดเหตุการณ์บางอย่างขึ้น ซึ่งในกรณีนี้ก็คือเหตุการณ์ที่มีแพ็กเก็ตใหม่เข้ามา สิ่งที่เราต้องทำก็มีเพียงแค่การเตรียม callback function สำหรับการประมวลผลข้อมูลแพ็กเก็ตและบอกให้ libpcap รับทราบว่าจะต้องกระโดดมาเรียกฟังก์ชันนี้ด้วยการเรียกใช้ฟังก์ชัน pcap_loop() ซึ่งมีโปรโตไทป์ คือ

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
```

พารามิเตอร์ตัวแรก คือ session handle พารามิเตอร์ถัดมา cnt คือจำนวนแพ็กเก็ตที่เราจะรอรับ หากระบุเป็นค่าติดลบ หมายความว่าให้ทำการจับแพ็กเก็ตไปเรื่อยๆ จนกว่าจะเกิดความผิดพลาดอย่างหนึ่งอย่างใดขึ้น ภายในส่วน callback ก็คือฟังก์ชันพอยเตอร์ที่ชี้ไปยังฟังก์ชันที่เราต้องการให้กระโดดไปทำงานเมื่อมีแพ็กเก็ตใหม่เข้ามา โดยที่ฟังก์ชันดังกล่าวจะต้องมีโปรโตไทป์ที่สอดคล้องกับ typedef นี้

```
typedef void (*pcap_handler)(u_char *, const struct pcap_pkthdr *, const u_char *);
```

กล่าวคือจะต้องเป็นฟังก์ชันที่รับพารามิเตอร์ 3 ตัว พารามิเตอร์ตัวแรกเป็นพารามิเตอร์ของการ callback พารามิเตอร์ตัวถัดมาจะเป็นพอยเตอร์ชี้เฮดเดอร์ที่ libpcap สร้างขึ้น และพารามิเตอร์ตัวสุดท้ายเป็นพอยเตอร์ที่ชี้ข้อมูลแพ็กเก็ต

```
#include <stdio.h>
#include <pcap.h>
#include <time.h>
void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *pcaket) {
    char t imestr[256];
    strftime(timestr, sizeof(timestr), "%d %b %Y %H:%M:%S %Z",
    localtime(&header->ts.tv_sec));
    printf("Packet of the size %d bytes received at %s\n ",header->len,timestr);
}
int main() {
    char errbuf[PCAP_ERRBUF_SIZE];
    const u_char *packet, *pt;
    pcap_t *handle;
    struct pcap_pkthdr header;
    handle = pcap_open_live("eth0", 1024, 1, 0, errbuf);
    pcap_loop(handle, 5, process_packet, NULL);
    pcap_close(handle);
}
```

ผลลัพธ์

```
Packet of the size 76 bytes received at 26 Apr 2008 17:10:54
Packet of the size 104 bytes received at 26 Apr 2008 17:10:54
Packet of the size 42 bytes received at 26 Apr 2008 17:10:55
Packet of the size 76 bytes received at 26 Apr 2008 17:10:55
Packet of the size 104 bytes received at 26 Apr 2008 17:10:56
```

อันที่จริงแล้ว pcap_next() และ pcap_loop() ต่างก็มีข้อจำกัดด้วยกันทั้งคู่ นั่นคือโปรแกรมจะต้องมาหยุดรอที่จุดนี้เสมอจนกว่าจะมีแพ็กเก็ตใหม่เข้า ซึ่งอาจก่อให้เกิดความยุ่งยากต่อการออกแบบโปรแกรม โดยเฉพาะโปรแกรมที่มีการควบคุมการทำงานผ่านอินเตอร์เฟซ ในกรณีนี้เราอาจพิจารณาเลือกใช้ฟังก์ชัน pcap_dispatch() แทน วิธีนี้จะเปิดโอกาสให้เราเลือกเองได้ว่าจะให้ฟังก์ชันมีการหยุดรอหรือไม่ ซึ่งจะต้องใช้คู่กันกับฟังก์ชัน pcap_setnonblock() โปรโตไทป์ของฟังก์ชันเป็นดังนี้

```
int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
int pcap_setnonblock(pcap_t *p, int nonblock, char *errbuf);
```

ฟังก์ชัน pcap_dispatch() จะมีการรับพารามิเตอร์เหมือนกับฟังก์ชัน pcap_loop() ทุกประการ ในส่วนของ pcap_setnonblock() นั้นจะใช้สำหรับการเลือกโหมดของ handle ว่าจะให้มีการหยุดรอจนกว่าจะมีแพ็กเก็ตใหม่เข้ามาหรือว่าจะให้ return ทันทีแม้ไม่พบแพ็กเก็ตใหม่ ซึ่งเราสามารถกำหนดได้ด้วยการตั้งค่าของ nonblack เป็น 0 หรือเป็นตัวเลขอื่น

4.2.2 อินเตอร์เฟสที่ใช้ในโครงงาน

จากบทที่ 3 ในขั้นตอนของการติดตั้งนั้นในระบบปฏิบัติการลีนุกซ์ ไดร์เวอร์ของการ์ดไวร์เลสปกติทางผู้ผลิตจะไม่มีการสร้างไดร์เวอร์ขึ้นมาเพื่อรองรับกับระบบปฏิบัติการแบบลีนุกซ์ชนิดต่างๆ เราจึงต้องหาไดร์เวอร์ที่มีการพัฒนาขึ้นมาเพื่อนำมาใช้กับระบบปฏิบัติการลีนุกซ์และกับการ์ดไวเลสชนิดนั้นโดยเฉพาะ โดยการพัฒนาโครงงานครั้งนี้เราได้นำไดร์เวอร์ชื่อ madwifi ซึ่งรองรับการ์ดไวร์เลสชนิด อเทรอส (Atheros) โดยในบทที่ 3 ได้สอนขั้นตอนการติดตั้งไปแล้ว

โดยอินเทอร์เฟสที่เราจะนำมาใช้งานนั้น อินเทอร์เฟสของการ์ดไวเลสเมื่อเราเสียบเข้าไปกับตัวเครื่องเมื่อมีการติดตั้งไดร์เวอร์เรียบร้อยแล้ว การ์ดไวเลสจะอยู่ใน โหมด Managed โดยอัตโนมัติ ซึ่งในโหมดนี้ จะทำงานเสมือนเป็น Station โดยการดักจับแพ็คเก็ตของเราจะทำการในโหมดนี้ไม่ได้เราต้องทำการทำลาย อินเทอร์เฟสของเราที่อยู่ในโหมด Manage ทิ้งก่อนด้วยคำสั่ง

```
wlanconfig ath0 destroy
```

หลังจากเสร็จคำสั่งนี้อินเทอร์เฟสการ์ดไวเลสของเราที่สร้างขึ้นด้วยไดร์เวอร์ Madwifi จะหายไปเราจึงต้องสร้างอินเทอร์เฟสขึ้นมาใหม่ให้เหมาะกับการใช้งานกับตัวโครงงานเราคือโหมด Monitor โดยคำสั่ง

```
ath0      IEEE 802.11g  ESSID:""
          Mode:Monitor  Channel:0  Access Point: Not-Associated
          Bit Rate:0 kb/s   Tx-Power:18 dBm   Sensitivity=0/3
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/94  Signal level=-95 dBm  Noise level=-95 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

รูปแสดงผล iwconfig จะเป็นการสร้างอินเตอร์เฟสของการ์ดไวร์เลสขึ้นมาในโหมดมอนิเตอร์ ซึ่งการทำงานในโหมดมอนิเตอร์นั้น ในโหมดนี้การ์ดไวเลส จะเป็นโหมดที่ไม่มีการส่งข้อมูลใดๆ ออกจากตัวอุปกรณ์เลย นั่นคือการทำงานในโหมดนี้จะเป็นโหมดที่รับและฟังข้อมูลการสื่อสารระหว่างสถานีลูกข่ายกับแม่ข่ายทั้งหมดโดยไม่มีการกรองข้อมูลใดๆ ทั้งสิ้น ซึ่งการทำงานในโครงงานของเราเราจำเป็นที่จะต้องเซ็ตค่าของอินเทอร์เฟสเราให้อยู่ในโหมดมอนิเตอร์

4.3 การแปลงความหมายค่าต่างๆของแพ็คเก็ต

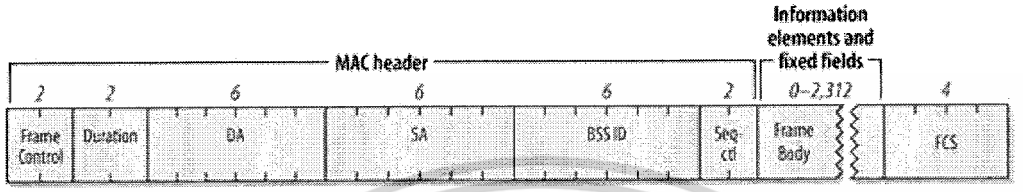
หลังจากที่เราทำการดักจับแพ็คเก็ตจากอินเทอร์เฟสของการ์ดไวร์เลสได้แล้วนั้นการตีความส่วนต่างๆของแพ็คเก็ตที่ดักจับได้เราจะตีความจากข้อมูลดิบของแพ็คเก็ต โดยทั่วไปโครงสร้างของแพ็คเก็ตนั้นจะถูกหุ้มโดย header เป็นชั้นๆ โดยส่วนที่เราสนใจจะเป็นส่วนที่เป็นรูปแบบของเฟรมในมาตราฐาน IEEE 802.11 ซึ่งจะเริ่มนับจาก ไบท์ที่ 144 เป็นต้นไป แล้วแต่ชนิดของเฟรมไวร์เลสนั้น ดังรูปที่ แสดงรูปแบบของ Management Frame แบบแพ็คเก็ตดิบๆ ที่เก็บข้อมูลทีละไบท์

```
44 00 00 00 90 00 00 00 61 74 68 30 00 00 00 00
00 00 00 00 00 00 00 00 44 00 01 00 00 00 04 00
31 AA 01 00 44 00 02 00 00 00 04 00 DD 33 9F 2B
44 00 03 00 00 00 04 00 0B 00 00 00 44 00 04 00
00 00 04 00 37 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 44 00 06 00 00 00 04 00 DB FF FF FF
44 00 07 00 00 00 04 00 A4 FF FF FF 44 00 08 00
00 00 04 00 02 00 00 00 44 00 09 00 00 00 04 00
00 00 00 00 44 00 0A 00 00 00 04 00 59 00 00 00
80 00 00 00 FF FF FF FF FF FF 00 13 F7 80 EF 81
00 13 F7 80 EF 81 B0 C2 B7 31 9F 2B 00 00 00 00
64 00 61 04 00 09 42 65 41 4E 2D 57 49 46 49 01
05 82 84 8B 96 2C 03 01 0B 05 04 00 01 00 00 2A
01 00 32 08 0C 12 18 24 30 48 60 6C DD 07 00 50
```

รูปที่ 4.1 แสดงตัวอย่างแพ็คเก็ต 1 แพ็คเก็ต ชนิด Management Frame

จากรูป จะเป็นแพ็คเก็ตขนาด 229 ไบท์ ส่วนที่แรเงาจะเป็นส่วนของเฟรมในมาตรฐาน IEEE 802.11 ในส่วน Mac Header ส่วนก่อนหน้านั้นจะเป็น header ในส่วนของ Prism Monitoring Header ซึ่งเราจะไม่สนใจ

ถ้านำ packet ที่ดักจับได้นั้น มาเริ่มนับตั้งแต่บิทที่ 144 แล้วนำเอามาเทียบกับรูปแบบเฟรมตามมาตราฐาน IEE 802.11 จะเป็นดังนี้

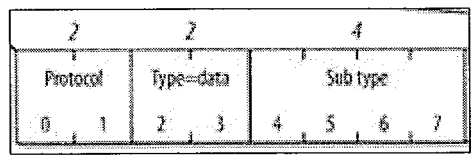


รูปที่ 4.2 แสดงตัวอย่างของ Management Frame

รูปที่ 4.3 แสดงการแปลงความหมายแพ็คเก็ต

ซึ่งการตีค่าความหมายของแพ็คเก็ต โปรแกรมเรานั้นอย่างแรกที่เราจะทำการหาคือ เราจะทำการหาก่อนว่า Frame ที่จับมาได้นั้นเป็น Frame ชนิดอะไร Data Frame Control Frame หรือ Management Frame โดยทำการหาจาก ไบต์แรกของ Mac Header ซึ่งหลักการที่เราจะนำมาพิจารณามีดังนี้

โดยเราจะให้พอยต์เตอร์ที่สร้างขึ้นมาชี้ค่าไปที่ไบต์แรกของ Mac Header ก่อนเราจะได้ค่าของ Field Flame Control byte แรกมาจากรูปที่ 4.3 จะได้ค่าของ เลขฐานสิบหกค่า 80 มาซึ่งถ้าเรานำค่าเลขฐานสิบหก ที่ได้มานั้นแปลงเป็นเลขฐานสองก็จะได้ค่าว่า 80 = 1000 0000 โดยเราจะทำการเช้ค บิทที่ 3 และ 4 โดยนับจากทางขวา เราจะได้ค่า บิท 00 ซึ่งตรงกับไทป์ ของ Management Frame ซึ่งถ้าเป็น เฟรมชนิดอื่นค่าที่ได้ก็จะต่างออกไปเช่นค่า 01 เป็น Type ของ Control Frame และถ้า มีค่าเท่ากับ 10 จะเป็นชนิดของดาต้าเฟรม



รูปที่ 4.4 แสดงค่า 1 ไบต์แรกของ Frame Control

ตัวอย่างโค้ดในส่วนที่เราทำการแปลความหมายข้อมูลนั้นจะเป็นดังนี้

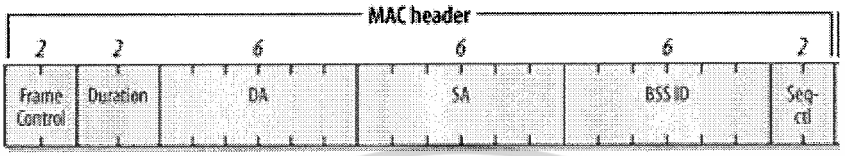
```
const u_char *pt;    // สร้างตัวแปรพ้อยเตอร์ขึ้นมา
pt = packet+144;    // ชี้ไปที่ตำแหน่งที่ 144 ของแพ็คเก็ต
const u_char packetType = (*pt >> 2) & 0x03;  // ทำการเก็บค่าบิทที่ 2

if(packetType == 0x00)   {} // Management    เปรียบเทียบค่าว่าเป็นชนิดอะไร

else if(packetType == 0x01)  {} // Control
else if(packetType == 0x02)  {} // Data
```

โดยจากบรรทัดแรก เราต้องทำการชี้พอยน์เตอร์ pt ให้มีค่าตรงกับพอยน์เตอร์แพ็ตเก็ตที่ทำการรับค่าไบต์แรกของข้อมูลแพ็คเก็ตก่อน จากนั้นเราทำการเลื่อนตำแหน่งไบท์ไป 144 ตำแหน่งจะไปหยุดที่ไบท์แรกของ Mac Header แล้วเราจึงทำการเปรียบเทียบว่าเป็นเฟรมชนิดอะไร โดยเราทำการชิพค่าไป 2 ตำแหน่ง เพื่อที่จะให้ค่าไปหยุดที่บิตที่ 3 ซึ่งเป็นค่าที่เริ่มเก็บไทป์ของเฟรมนั้น แล้วจึงทำการเปรียบเทียบโดยทำการแอนด์กับค่า 0x03 ซึ่งค่าที่แอนด์ออกมาได้นั้นจะมีค่าเท่ากับตัวเดิมแล้วจึงนำมาเปรียบเทียบว่าเป็นเฟรมชนิดอะไร

โดยหลักการสำคัญของโปรแกรมเรานั้นอยู่ที่การวิเคราะห์ให้ได้ว่าแพ็คเก็ตที่รับมานั้นเป็น แพ็คเก็ตของเฟรมชนิดอะไรเมื่อวิเคราะห์ได้แล้ว รูปแบบของเฟรมในชนิดนั้นๆ ก็จะแตกต่างกันแต่เรารู้แล้วว่าเป็นชนิดอะไรเราจึงสามารถที่จะดึงค่าอื่นๆที่ต้องการออกมาได้ตามมาตรฐานของ IEE 802.11 โดยในตัวอย่างจะเป็นการทำการแสดงค่าของ ManageMent Frame

```
if(((*pt >> 2) & 0x03) == 0x00)  Manage ment Frame
{       frame->Append(_T("Frame Type : Management "));
        frame->Append(wxString::Format(_T("(%02X)\n"), *pt));
        pt = pt+4;
        Destination Address : %02X:%02X:%02X:%02X:%02X:%02X
        pt = pt+6;
        Source Address : %02X:%02X:%02X:%02X:%02X:%02X
        pt = pt+6;
        BSS Id : %02X:%02X:%02X:%02X:%02X:%02X
}
```

จากโค้ดจะเห้นว่าเมื่อค่าที่จับมาได้บอกว่าเป็นชนิดของ Management Frame นั้น เราก็จะมีการทำการแสดงค่าอื่นๆ ออกไป โดยรูปแบบหลักของ Management Frame คือ 4 ไบต์ถัดจากไบต์แรกนั้นจะแสดงค่าของ Address 1 ซึ่งคือ Destination Address ซึ่งมีขนาด 6 ไบท์แล้ว Address 2 คือ Source Address ซึ่งมีขนาด 6 ไบต์เท่ากัน และ Address 3 คือ BSSID มีขนาด 6 ไบต์ ดังรูปที่ 4.5 โดยค่าของแอดเดรสที่แสดงจะแสดงในรูปแบบของเลข ฐานสิบหก



รูปที่ 4.5 แสดง Mac header ชนิด Management Frame

4.4 การบันทึกแพ็คเก็ตที่จับมาได้

หลังจากที่เราทำการจับแพ็คเก็ตได้แล้วนั้นเราสามารถที่จะนำแพ็คเก็ตนั้นมาบันทึกลงเป็นล็อกไฟล์ซึ่งเราสามารถที่จะนำมาดูค่าภายหลังได้ หรือ เราสามารถที่จะนำค่าที่บันทึกไปใช้ในงานในภายหลังได้ โดยค่าในล็อกไฟล์ที่เก็บนั้นจะทำการบันทึกค่าลงไปในแต่ละไบต์ของแพ็คเก็ตที่ดักจับได้ โดยจะบันทึกค่าลงไปทุกไบต์ที่ดักจับได้และทุกเฟรมตั้งแต่ที่มีการดักจับเกิดขึ้นจนจบ โดยตัวอย่างของโค้ดจะเป็นดังนี้

```
FILE* fp; // การเปิดไฟล์
fp = fopen("packet.log", "a+");// สร้างไฟล์ชื่อ packet.log

strftime(timestr, sizeof(timestr), "%d %b %Y %H:%M:%S %Z",
        localtime(&header->ts.tv_sec));


fprintf(fp, "Packet of the size %d bytes received at %s\n ",header->len,timestr);
fprintf(fp, "Packet data :\n");

for (pt=packet; pt!=packet+header->caplen; pt++) {
        fprintf(fp, "%02X ",*pt);
}
fprintf(fp, "\n \n");
fclose(fp); // การปิดไฟล์
```

จากโค้ดจะเห็นว่าก่อนทำการบันทึกเราจะมีการสร้างตัวแปลเป็นชนิดของไฟล์และมีการทำการเปิดไฟล์ โดยคำสั่ง fopen ซึ่งมีพารามิเตอร์เป็น a+ แสดงว่าให้เปิดไฟล์เดิมและบันทึกค่าต่อลงไปจากค่าเดิม ถ้าไม่มีไฟล์เดิมอยู่ก่อนก็จะสร้างไฟล์ใหม่ขึ้นมา มีการบันทึกค่าของขนาดแพ็คเก็ตจากตัวแปล header->len และมีการแสดงเวลาที่ได้รับไฟล์ ขั้นตอนในการเริ่มบันทึกข้อมูลจะเริ่มที่บรรทัด for (pt=packet; pt!=packet+header->caplen; pt++) ซึ่งจะชี้ตัวแปร pt ไปที่ตำแหน่งเริ่มต้นของ แพ็คเก็ตแล้วจะบวกค่า pt ไปเรื่อยๆจนกว่าค่าของ pt จะเท่ากับค่าของ packet + header->caplen ซึ่งเมื่อถึงไฟล์ที่ไบท์นี้จะแสดงว่ามาถึงไฟล์สุดท้ายของแพ็คเก็ตนั้นแล้วจึงทำการออกจากลูป for แล้วจึงทำการปิดไฟล์ด้วยคำสั่ง fclose()

4.5 การวิเคราะห์แพ็คเก็ต

จะเห็นได้ว่าจากการที่เราตีความค่าความหมายต่างๆ ของแพ็คเก็ตแต่ละชนิดได้แล้ว เราก็สามารถที่จะบอกได้ว่าในระบบของเรานั้นมีแอคเซสพอยท์อยู่ทั้งหมดกี่เครื่อง หรือ มีเครื่องไคลเอนท์ภายในระบบทั้งหมดจำนวนกี่เครื่อง หรือแม้กระทั่งเราสามารถที่จะรับรู้ได้ถึงทิศทางของแพ็คเก็ตแต่ละแพ็คเก็ตที่ทำการส่งว่าแพ็คเก็ตไหนเป็นแพ็คเก็ตที่กำลังจะส่งออกไปยังเครือข่ายภายนอก หรือแพ็คเก็ตไหนเป็นแพ็คเก็ตที่พึ่งรับเข้ามา ทำให้เราสามารถรู้ค่าอัพโหลด และดาวน์โหลดของคู่สัญญาณแอคเซสพอยท์และเครื่องไคลเอนท์แต่ละคู่สัญญาณ และสามารถรู้จำนวนของแพ็คเก็ตที่ภายในช่วงระยะเวลาหนึ่งๆ นั้นเครื่องไคลเอนท์และแอสเซสพอยน์ในแต่ละคู่สัญญาณนั้นมีการทำการรับส่งข้อมูลแบบดาวน์โหลดเป็นจำนวนเท่าไหร่ หรือ รับส่งข้อมูลแบบอัพโหลดเป็นจำนวนเท่าไหร่ ทำให้เราสามารถที่จะหาค่าเฉลี่ยของขนาดแพ็คเก็ตในแต่ละคู่สัญญาณนั้นได้ โดยระบบเรานั้นมีฟังก์ชันในการรายงานสถานะของระบบเป็นวินาทีโดยหลักการทำงานของตัวระบบนั้น ระบบจะทำการรับแพ็คเก็ตแต่ละชนิดมาเรื่อยๆ แล้วทำการเก็บค่าตัวแปรที่ได้รับมาตามแต่ละชนิดของแพ็คเก็ต และเมื่อถึงเวลาที่โปรแกรมจะต้องรายงานสถานะของตัวระบบ โปรแกรมจะทำการแสดงสถานะของระบบออกมา

4.5.1 การวิเคราะห์จำนวนแอคเซสพอยท์ภายในระบบ

การวิเคราะห์จำนวนแอคเซสพอยท์ภายในระบบนั้น การที่เราจะรู้ได้ว่าในระบบของเรามีแอคเซสพอยท์เป็นจำนวนทั้งหมดกี่เครื่องเราสามารถรับรู้ได้จากเฟรมชนิด Management Frame ซึ่งแอคเซสพอยท์จะทำการส่งManagement Frame ชนิดหนึ่งเรียกเฟรมชนิดนั้นว่า บีคอนเฟรม (Beacon Frame) บีคอนเฟรมนั้นมีหน้าที่ทำการบอกตัวตนของแอคเซสพอยท์ในเครือข่ายนั้นให้เครื่องไคลเอนท์ในระบบรับรู้ถึงการมีตัวตนว่ามีแอคเซสพอยท์ตัวนั้นอยู่ในระบบ โดยบีคอนเฟรมจะถูกเซ็ตค่าให้ส่งออกมาตลอดเวลา แล้วแต่ว่าแอคเซสพอยท์แต่ละตัวว่าจะมีการส่งถี่ขนาดไหน โดยการที่เราจะรับรู้ได้ว่าในระบบของเรามีแอคเซสพอยท์กี่เครื่อง

เราก็จะทำการวิเคราะห์แต่เฟรมซึ่งเป็นชนิด Management และเป็นบีคอนเฟรม โดยบีคอนเฟรมนั้นแอดเดรสที่ 1 นั้นจะคือ Destination Adrress ซึ่งถ้าเป็นเฟรมชนิดบีคอนเฟรมนั้นแอดเดรสในตำแหน่งนี้จะถูกเซ็ตค่าให้เป็นค่าแบบบรอดแคสท์คือส่งไปทุกตัวในระบบ ซึ่งจะมีค่า FF-FF-FF-FF-FF-FF ส่วนแอดเดรสที่ 2 นั้นจะเป็นค่าของ Source Address ซึ่งนั่นก็คือ ค่าแมคแอดเดรสของแอคเซสพอยท์นั่นเอง และแอดเดรสที่ 3 ซึ่งเป็นตำแหน่งของ BSS ID ซึ่งถ้าเป็นชนิดของบีคอนเฟรมที่ปล่อยจากแอคเซสพอยท์จริงๆ ค่า BSSID ต้องมีค่าเท่ากับค่าของ Source Address ซึ่งก็คือค่าในตำแหน่งแมคแอดเดรสที่ 2 ต้องมีค่าเท่ากับตำแหน่งแมคแอดเดรสที่ 3

```
struct MacAccessPoint    // ตัวแปรแบบโครงสร้างที่เก็บค่าของแอคเซสพอยท์
{
        unsigned char name[255];  // เก็บชื่อ SSID
        unsigned char mac1[6];   // เก็บ SOUCE ADDRESS
        unsigned char mac2[6];  // เก็บ BSSID
};
```

ตัวอย่างโค้ดการเปรียบเทียบค่าว่าเป็นเฟรมชนิดบีคอนเฟรมหรือไม่

```
bool isBeacon = true; // ตัวแปรชนิดบูลีนให้ค่าเป็นทรู

for(int i=0;i<6;i++) { // ทำการตรวจสอบค่าแมคแอดเดรส BSSID และSource
        if(ap.mac1[i]!=ap.mac2[i])
        {
                isBeacon = false;
        }
}
        if(isBeacon) //ถ้าค่าแมคแอดเดรสเท่ากันแสดงว่าเป็นบีคอน
        {
                AddAccessPoint(ap);
        }
```

โดยภายในโปรแกรมจะมีการสร้างตัวแปรขึ้นมาเก็บค่าของแมคแอดเดรสในตำแหน่งที่ 2 ไว้เป็นแมคแอดเดรสของแอคเซสพอยท์ ซึ่งถ้าตัวโปรแกรมจับบีคอนเฟรมอีกแพ็คเก็ตได้ขึ้นมาก็จะมีการนำค่าแอดเดรสมาเปรียบเทียบ ถ้าเป็นค่าเดิมตัวโปรแกรมก็จะไม่ทำการบันทึกค่าเพิ่ม แต่ถ้าเป็นคนละตัวกันโปรแกรมก็จะทำการบันทึกค่าแมคแอดเดรสเพิ่มลงไป เป็นการเก็บค่าแมคแอดเดรสของแอคเซสพอยท์ ในระบบทั้งหมด เท่านี้โปรแกรมก็จะรับรู้ได้ว่าภายในระบบมี แอ็คเซสพอยท์ทั้งหมดกี่ตัว และค่าที่สำคัญอีกค่านึงคือค่า SSID ซึ่งจะบอกชื่อของแอคเซสพอยท์ แต่ละตัว โดยเราจะมีการทำการเก็บค่า SSID ของแอคเซสพอยท์ นั้น โดยค่า SSID จะอยู่ในไบต์ที่ 182 นับจากไบต์แรกของตัวข้อมูล โดยแต่ละ SSID จะมีไบต์ที่แสดดงค่าบอกว่า SSID แต่ละตัวของแอคเซสพอยท์นั้นจะเก็บชื่อโดยคิดเป็นกี่ตำแหน่งซึ่งไบต์ที่บอกนั้นจะอยู่ก่อนหน้าไบต์ที่บอกชื่อไบต์ก็คือตำแหน่งที่ 181 นับจากไบต์แรก ซึ่งเราต้องทำการเก็บค่าของชื่อของ SSID โดยตรวจสอบจากค่าที่บอกก่อนว่าแพ็คเก็ตนี้มี SSID เป็นจำนวนกี่ตัวอักษรเราก็บันทึกค่า SSID ตามจำนวนนั้น

ตัวอย่าง โค้ดส่วนที่ทำการบันทึกค่า SSID

```
pt = packet + 181;          // ชี้ไปยังตำแหน่งที่บอกจำนวน SSID
int length = *pt;   // เก็บค่าขนาด SSID
pt = packet + 182;   // ชี้ไปยังไบต์แรกของชื่อ SSID

        for(int i=0;i<length;i++)   // ทำการบันทึกค่า SSID ตามจำนวนขนาดของมัน
        {
                ap.name[i] = *(pt + i);
        }

ap.name[length] = 0;  // เซ็ตให้ค่าสุดท้ายของชื่อ SSID มีค่าเท่ากับศูนย์
```

ตัวอย่างโค้ดในส่วนการเก็บค่าแอคเซสพอยท์

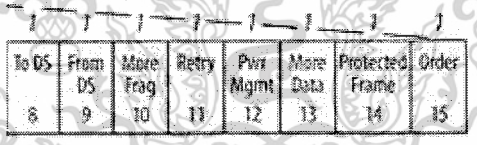
```
void AddAccessPoint(MacAccessPoint mac){
        bool isFound = false;
        vector<MacAccessPoint>::iterator it;
        for(it = accesspoint.begin(); it != accesspoint.end(); it++)
        {
                bool isEqual = true;
        for(int i=0;i<6;i++)//ทำการตรวจสอบค่าแมคแอดเดรสว่าซ้ำหรือไม่
        {
                if(mac.mac1[i] != it->mac1[i])
                isEqual = false;
        }
                if(isEqual){ // ถ้าแมคแอดเดรสมันซ้ำก็ไม่ทำอะไร
                        isFound = true;
                }
                if(!isFound)
                {
                accesspoint.push_back(mac); // ถ้าไม่ซ้ำก็ทำการเพิ่มค่า
                }
        }
```

ในส่วนของฟังก์ชั่นแอคเซสพอยต์ จะเป็นฟังก์ชั่นที่ทำการตรวจสอบค่าของแมคแอดเดรสที่ส่งเข้ามาโดยจะทำการตรวจสอบว่าซ้ำกับแอคเซสพอยต์เดิมที่เราทำการเก็บค่าไปแล้วหรือไม่ ถ้าซ้ำก็ไม่ทำการเก็บแต่ถ้าไม่ซ้ำก็ทำการบันทึกค่าลงเวคเตอร์

4.5.2 การวิเคราะห์ค่าอัตราการอัพโหลดและดาวน์โหลดของแต่ละคู่สัญญาณ

```
struct Pair  // ตัวแปรแบบโครงสร้างของ Data Frame
        {
                u_char accessPoint[6]; // เก็บแมค  Access Point
                u_char client[6]; // เก็บ แมค ไคลเอ้น
                unsigned long packetLength; // เก็บจำนวนขนดาแพ็คเก็ต
                unsigned long packetCount; //เก็บจำนวนแพ็คเก็ต


        };
```

การที่เราจะวิเคราะห์ค่าอัพโหลดและดาวน์โหลดของแต่ละคู่สัญญาณนั้น เราต้องเจาะจงชนิดแพ็คเก็ตที่เราต้องทำการตรวจสอบโดย ในสมมติฐานที่เราจะวิเคราะห์นี้นั้นเราจะเจาะจงไปที่แพ็คเก็ตที่มีชนิดเป็นดาต้าเฟรม โดยเราสามารถรู้ได้ว่าแพ็คเก็ตที่เป็นดาต้าเฟรมนั้นเป็นแพ็คเก็ตที่กำลังรับหรือถูกส่งมาจากแอคเซสพอยต์ โดยฟิลด์ที่เราจะต้องทำการดูค่านั้นจะอยู่ในส่วนของแมคเฮดเดอร์ โดยอยู่ในส่วนของ Flame Control ไบต์ที่ 2 ดังรูปที่ 4.7



รูปที่ 4.6 แสดงค่าไบต์ที่ 2 ของฟิลด์ Flame Control

จากรูปที่ 4.7 จะเห็นว่า 2 บิตแรกคือค่า To DS (to distributed) กับ From DS (from distributed) เป็น 2 บิตที่ใช้แสดงค่าว่าแพ็คเก็ตนั้นถูกส่งมาจากแอคเซสพอยต์ หรือ แพ็คเก็ตนั้นกำลังส่งไปยังแอคเซสพอยต์ โดยเราสามารถนำมาวิเคราะห์ได้ว่าถ้าเป็นแพ็คเก็ตที่ถูกส่งมาจากแอคเซสพอยต์ ค่า ToDs จะเป็นเลข 0 และ FromDs จะเป็น 1 ขนาดของแพ็คเก็ตนั้นก็จะเป็นค่าดาวน์โหลดที่เครื่องไคลเอนท์ที่เป็นคู่สัญญาณกันกำลังดาวน์โหลดมาอยู่ แต่ถ้าค่าของ ToDSเป็น 1 และ FromDs เป็น 0 แสดงว่าแพ็คเก็ตนั้นกำลังจะถูกส่งออกไปยังดิสทริบิวท์แสดงว่าแพ็คเก็ตนั้นเป็นแพ็คเก็ตที่กำลังจะส่งไปยังแอคเซสพอยต์ ค่าขนาดของแพ็คเก็ตก็จะเป็นค่าอัพโหลดที่เครื่องไคลเอนท์ที่เป็นคู่สัญญาณกับแอคเซสพอยต์กำลังอัพโหลดอยู่ โดยจากการที่เรารู้ว่าแพ็คเก็ตไหนเป็นแพ็คเก็ตที่ดาวน์

โหลดหรืออัพโหลด เราก็จะหาค่าอัพโหลดและค่าดาวน์โหลดเฉลี่ยเป็นขนาดต่อวินาทีได้ และเราสามารถที่จะหาขนาดแพ็คเก็ตเฉลี่ยที่ทำการดาวน์โหลดหรืออัพโหลดได้

ตัวอย่างโค้ดการแยกชนิดเฟรมว่าเป็นอัพโหลดหรือดาวน์โหลด

```
pt = packet+145; // ไปยังตำแหน่งที่ 145 ของแพ็คเก็ต
const u_char direction = *pt & 0x03;  ทำการเก็บค่า2 บิตแรก
if(direction == 0x01)      // data frame upload
else if(direction == 0x02)     // data frame download
```

จากตัวอย่างโค้ดการแยกชนิดเฟรมอัพโหลดหรือดาวน์โหลดเราจะเริ่มต้นโดยการชี้พอยเตอร์ไปยังตำแหน่งไบต์ที่เราต้องการคือตำแหน่งไบต์ที่ 145 นับจากไบต์แรกแล้วจึงทำการตรวจสอบค่าในตำแหน่ง 2 บิตแรก ถ้าเป็น 01 คือ เฟรมชนิดอัพโหลด ถ้าเป็น 10 คือเฟรมชนิดดาวน์โหลด

ตัวอย่างโค้ดการบันทึกค่าต่างๆ ของดาต้าเฟรม

```
Pair uploadPacket; // สร้างตัวแปรแบบ โครงสร้าง
                pt = packet+148; // ชี้ไปยังไบท์ที่ 148
                for(int i=0;i<6;i++) // เก็บค่าแมคแอดเดรส
                {
                        uploadPacket.accessPoint[i] = *(pt+i);
                }
                pt = packet+154; // ชี้ไปยังไบท์ที่ 154
                for(int i=0;i<6;i++)// เก็บค่าแมคแอดเดรส
                {
                        uploadPacket.client[i] = *(pt+i);
                }
                uploadPacket.packetLength = header->len; // เก็บค่าขนาดแพ็คเก็ต
                uploadPacket.packetCount = 1;

                AddToUpload(uploadPacket);
```

เมื่อเราทำการแยกชนิดของดาต้าเฟรมได้แล้วว่าเป็นชนิดของเฟรมแบบดาวน์โหลดหรืออัพโหลด เราจะทำการเก็บค่าแมคแอดเดรสของแอสเซสพอยต์และแมคแอดเดรสของเครื่องไคลเอนท์ที่ทำการรับส่งแพ็คเก็ตนั้น โดยจะมีการทำการเก็บลงตัวแปรแบบโรงสร้างที่ได้ประกาศไว้ในตอนแรกและเก็บค่าขนาดของแพ็คเก็ตนั้นจาก header->len จะเป็นขนาดของแพ็คเก็ตและจะทำการส่งตัวแปรแบบโครงสร้างไปยังฟังก์ชั่น AddToUpload เพื่อทำการเก็บค่าคู่สัญญาณต่อไป

ตัวอย่างโค้ดการเก็บค่าเฟรมชนิดอัพโหลด

```
void AddToUpload(Pair pair)   // ฟังก์ชั่นที่ทำการเช้คเฟรมชนิดอัพโหลด
{
        bool isFound = false;
        vector<Pair>::iterator it;
        for(it = upload.begin(); it != upload.end(); it++){
                bool isEqual = true;
                for(int i=0;i<6;i++)  // ทำการเช้คว่าคู่สัญญาณซ้ำหรือไม่
                {
                        if(pair.accessPoint[i] != it->accessPoint[i])
                                isEqual = false;
                        if(pair.client[i] != it->client[i])
                                isEqual = false;
                }
                if(isEqual) // ถ้าคู่สัญญาณที่ได้มาซ้ำ{
                        isFound = true;
                        it->packetCount++;    // เพิ่มจำนวนแพ็คเก็ตของคู่สัญญาณ
                        it->packetLength = it->packetLength + pair.packetLength;
                         // เพิ่มขนาดแพ็คเก็ตของคู่สัญญาณ
                }
        }
        if(!isFound) // ถ้าค่าของคู่สัญญาณไม่ซ้ำกับที่เก็บไว้แล้ว{
                upload.push_back(pair);   // เพิ่มคู่สัญญาณนั้นลงเวคเตอร์
        }
}
```

4.6 การตรวจสอบความปลอดภัยของระบบ

การตรวจสอบความปลอดภัยภายใจโปรแกรมของเรานั้นพัฒนาโดยขึ้นอยู่กับสมมติฐานของการตรวจจับค่าอัพโหลดและดาวน์โหลดและ ขนาดเฉลี่ยของแต่ละแพ็คเก็ตในช่วงเวลาที่เรากำหนดไว้ให้โปรแกรมรายงาน โดยโปรแกรมของเราผู้ที่ทำการควบคุมการทำงานเปรียบเสมือนแอดมินภายในระบบ ที่สามารถควบคุมค่าเทรสโชของระบบได้ว่าถ้าแอสเซสพอยต์ หรือเครื่องไคลเอนท์ของคู่สัญญาณไหนมีค่าอัพโหลดหรือดาวน์โหลดที่เกินจากค่าเทรสโชที่ระบบได้กำหนดไว้ โปรแกรมก็สามารถที่จะทำการแสดงคู่สัญญาณที่ผิดปกติหรือมีค่าเทรสโชเกินได้ โดยจากสมมติฐานนี้เราสามารถที่จะนำไปใช้กับการตรวจสอบได้จริงๆ เช่น เครื่องไคลเอนท์ของเราในระบบอาจโดนไวรัสที่ทำการบังคับให้ส่งเมลล์ออกจากระบบตลอดเวลาทำให้ค่าอัพโหลดของเครื่องไคลเอนท์ที่มีการโดนไวรัสจะมีการส่งค่าอยู่ตลอดเวลา โดยเราสามารถที่จะตรวจสอบจากตัวโปรแกรมได้ว่าเครื่องไคลเอนท์ไหนที่มีค่าในการอัพโหลดที่ผิดปกติ หรือ การตรวจสอบเครื่องไคลเอนท์ภายในระบบที่มีการใช้โปรแกรมดาวน์โหลดทำให้กินแบนด์วิทภายในระบบ โปรแกรมของเราก็สามรถที่จะ รายงานออกมาได้ว่าเครื่องไคลเอนท์คู่ไหนที่มีค่าดาวน์โหลดที่ผิดปกติ แล้วเราสามารถที่จะเก็บค่าการแสดงผลของการที่มีแพ็คเก็ตผิดปกติในครั้งนั้นๆ ได้

4.6.1 เงื่อนไขในการตรวจสอบความปลอดภัย

1. ค่าอัพโหลดของแอคเซสพอยต์และไคลเอนท์ในแต่ละคู่สัญญาณมีค่าเกินกว่าค่าเทรสโชที่ตั้งไว้ ระบบจะแสดงผลคู่สัญญาณที่มีค่าอัพโหลดที่ผิดปกติ
2. ค่าดาวน์โหลดของแอคเซสพอยต์และไคลเอนท์ในแต่ละคู่สัญญาณที่มีค่าเกินกว่าค่าเทรสโชที่ตั้งไว้ ระบบจะแสดงผลคู่สัญญาณที่มีค่าอัพโหลดที่ผิดปกติ
3. ขนาดของแพ็ตเก็ตเฉลี่ยในการอัพโหลดหรือดาวน์โหลดถ้ามีขนาดแพ็คเก็ตเฉลี่ยที่ผิดปกติจากที่ตั้งไว้ ระบบจะแจ้งเตือนออกมา

โดยในเงื่อนไขการตรวจสอบความปลอดภัยของตัวโปรแกรมนั้นถ้าคู่สัญญาณใดมีการผิดปกติตามเงื่อนไขข้างบนเงื่อนไขใดเงื่อนไขหนึ่ง ระบบก็จะทำการแจ้งเตือนว่าคู่สัญญาณนั้นเป็นคู่สัญญาณที่ผิดปกติ

ตัวอย่างโค้ดในการตรวจสอบความปลอดภัย

```
if( upload[i].packetLength / (float)(INTERVAL * 1024) >เทรสโช Up
||
upload[i].packetLength / (upload[i].packetCount * 1024)  > เทรสโชUpSize)
{        }
```

จากตัวอย่างโค้ดจะเห็นว่าเราจะมีการเอาค่าของอัตราการอัพโหลดเฉลี่ยต่อวินาทีซึ่งหาได้โดยการนำขนาดของแพ็คเก็ตทั้งหมดหารด้วยเวลาแล้วเอามาเปรียบเทียบกับค่าที่ระบบได้ทำการกำหนดไว้ถ้ามีค่ามากกว่าที่ระบบกำหนดไว้ก็จะมีการทำการแจ้งเตือน หรือ ถ้าขนาดเฉลี่ยของแพ็คเก็ตในแต่ละคู่สัญญาณซึ่งหาได้โดยการเอาขนาดแพ็คเก็ตทั้งหมดหารด้วยจำนวนแพ็คเก็ตในแต่ละคู่สัญญาณก็จะได้ขนาดแพ็คเก็ตเฉลี่ย ถ้ามีขนาดที่ผิดปกติไประบบก็จะทำการแจ้งเตือน ในตัวอย่างโค๊ดในที่นี้เป็นโค๊ดในส่วนของแพ็คเก็ตชนิดอัพโหลด ถ้าเป็นชนิดดาวน์โหลดก็มีเงื่อนไขในการคิดเหมือนกัน

บทที่ 5

การทดสอบและผลการทดลอง

5.1 อุปกรณ์ที่ใช้ในการทดลอง

1. เครื่องคอมพิวเตอร์ที่ลงระบบปฏิบัติการลินุกซ์ 1 เครื่อง
2. การ์ดไวร์เลสที่มีชิปเซ็ตเป็นชนิด Atheros
3. มีการติดตั้ง Mad Wifi ซึ่งเป็นไดร์เวอร์การ์ดไวร์เลสและสามารถให้อินเตอร์เฟสการ์ดไวร์เลสทำงานในโหมดมอนิเตอร์ได้
4. เครื่องคอมพิวเตอร์ที่ใช้ต้องทำการติดตั้งไลบรารี libpcap
5. เครื่องคอมพิวเตอร์ที่ติดตั้งการ์ดไวร์เลส
6. แอคเซสพอยด์

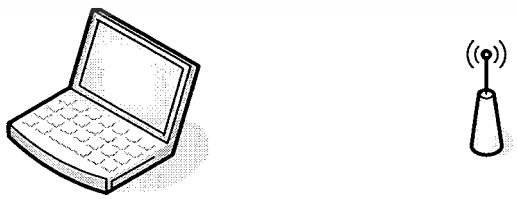
5.2 การจัดเตรียมด้านฮาร์ดแวร์และซอฟต์แวร์

1. ติดตั้งการ์ดไวร์เลสเข้ากับเครื่องคอมพิวเตอร์ที่ลงระบบปฏิบัติการลินุกซ์
2. ทำการสร้างอินเตอร์เฟสของการ์ดไวร์เลสจากไดร์เวอร์ Mad Wifi แล้วเซ็ตค่าโหมดของอินเตอร์เฟสเป็นโหมดมอนิเตอร์
3. ทำการเชื่อมต่อเครือข่ายดังรูปแต่ละการทดลอง

5.3 วิธีการทดลอง

5.3.1 การทดลองที่ 1

อุปกรณ์ภายในระบบ ประกอบด้วยเครื่องคอมพิวเตอร์ที่ติดตั้งโปรแกรมจำนวน1 เครื่อง แอคเซสพอยด์ในระบบจำนวน 1 ตัว



รูปที่ 5.1 แสดงระบบของการทดลองที่ 1

ขั้นตอนการทดลอง

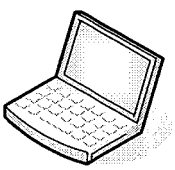
1. ทำการเปิดโปรแกรมแล้วทำการรันโปรแกรมเพื่อดูผลลัพธ์ที่ได้
2. บันทึกผลการทดลอง

```
Date is 2008/05/21
Time is 10:23:27
Accesspoint in Network..

00 0B 86 A8 D0 F1 : WiFi_KMITL

                                        --- UPLOAD ---

  Access point        Client          Upload/S        Average Packet Size        Warning Paacket


                                        --- DOWNLOAD ---

  Access point        Client          Download/S      Average Packet Size        Warning Paacket
```

รูปที่ 5.2 แสดงไฟล์ที่บันทึกของผลการทดลองที่ 1

5.3.2 การทดลองที่ 2

อุปกรณ์ภายในระบบประกอบด้วยเครื่องคอมพิวเตอร์ที่ทำการติดตั้งโปรแกรมจำนวน 1 เครื่องและแอคเซสพอยด์ 3 ตัวที่มี ค่า SSID เป็น WIFI_KMITL ,WIFI_KMITL และ CRSC_WIFI

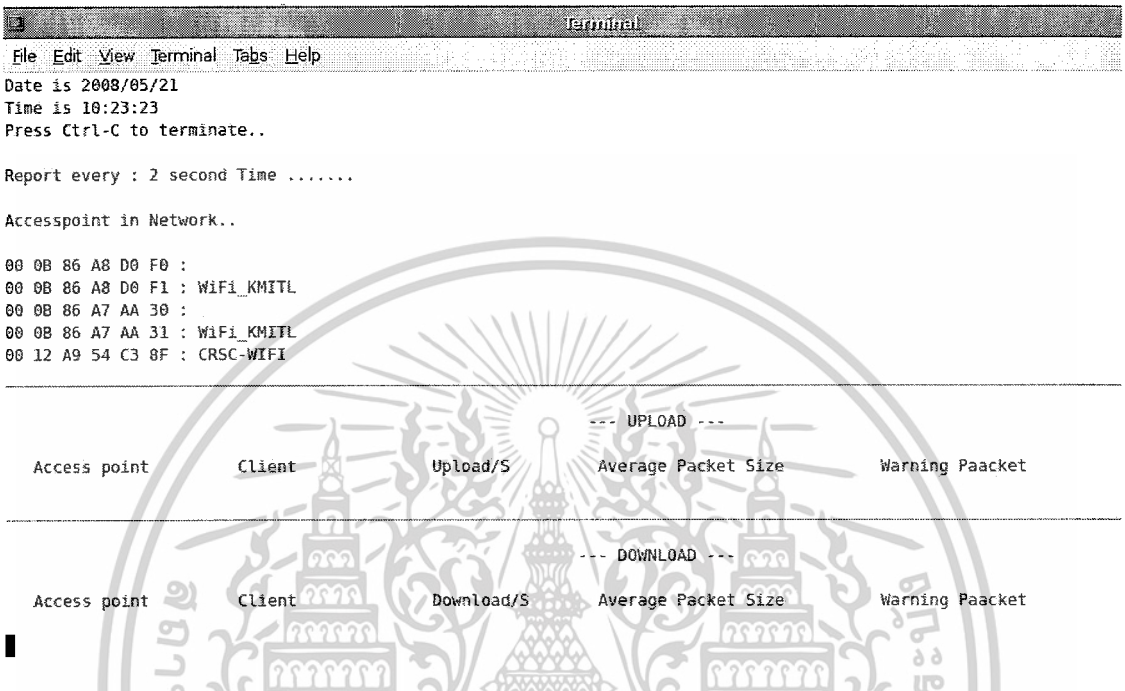


คอมพิวเตอร์ที่ติดตั้งโปรแกรม

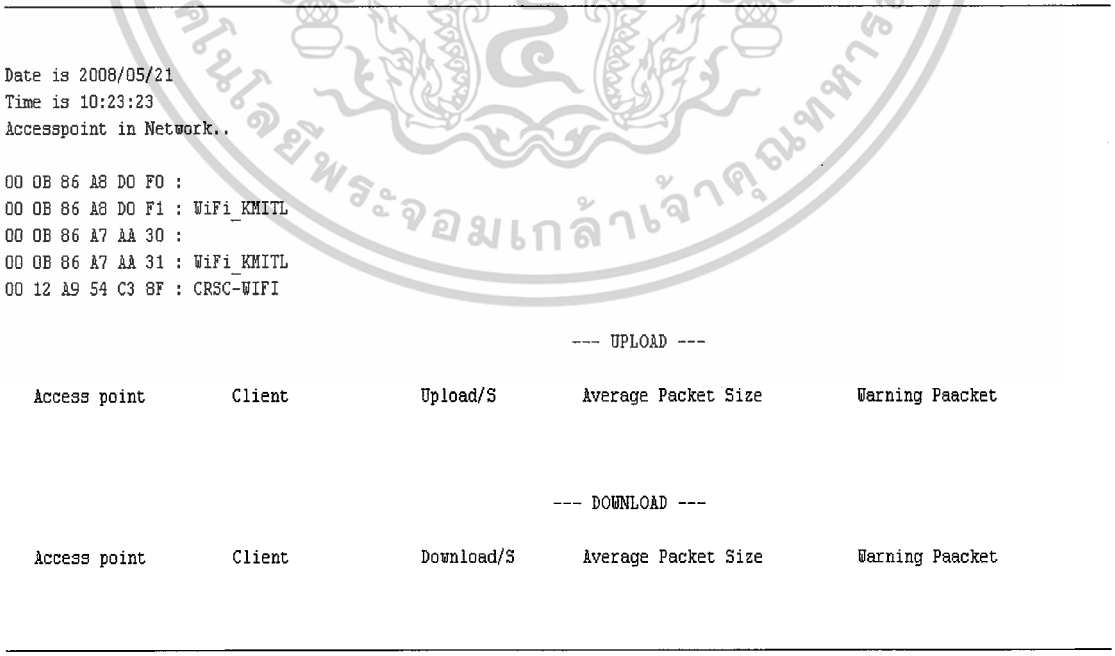
รูปที่ 5.3 แสดงระบบของการทดลองที่ 2

ขั้นตอนการทดลอง

1. ทำการเปิดโปรแกรมแล้วทำการรันโปรแกรมเพื่อดูผลลัพธ์ที่ได้
2. บันทึกผลการทดลอง



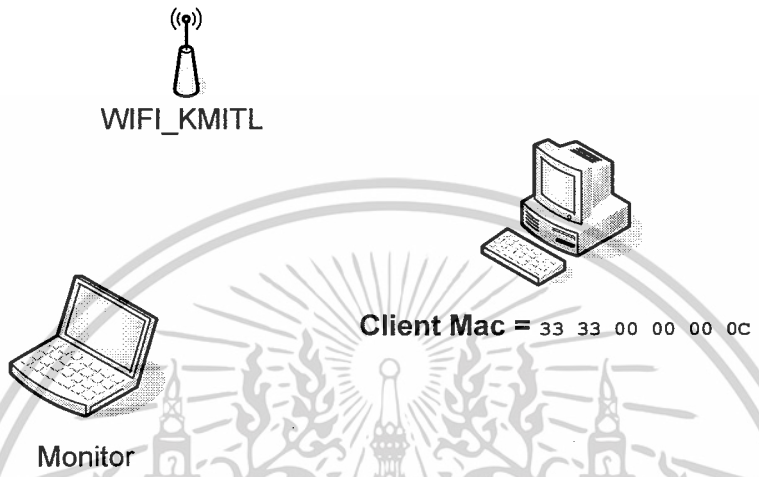
รูปที่ 5.4 แสดงหน้าจอโปรแกรมของผลการทดลองที่ 2



รูปที่ 5.5 แสดงไฟล์ที่บันทึกของผลการทดลองที่ 2

5.3.3 การทดลองที่ 3

อุปกรณ์ภายในระบบประกอบด้วยเครื่องคอมพิวเตอร์ที่ทำการติดตั้งโปรแกรมจำนวน 1 เครื่อง แอคเซสพอยด์ 1 เครื่องที่มีค่า SSID เป็น WIFI_KMITL และเครื่องไคลเอนท์ที่มีหมายเลขแมคแอดเดรสเท่ากับ 33 33 00 00 0C



รูปที่ 5.6 แสดงระบบของการทดลองที่ 3

ขั้นตอนการทดลอง

1. ทำการเปิดโปรแกรมแล้วทำการรันโปรแกรมเพื่อดูผลลัพธ์ที่ได้
2. ให้เครื่องไคลเอนท์ทำการใช้งานแอคเซสพอยด์ที่มี SSID WIFI_KMITL
3. บันทึกผลการทดลอง

```
Date is 2008/05/21
Time is 10:23:39
Accesspoint in Network..

00 0B 86 A8 D0 F1: WiFi_KMITL

                                        --- UPLOAD ---

   Access point         Client           Upload/S        Average Packet Size        Warning Paacket


                                        --- DOWNLOAD ---

   Access point         Client           Download/S      Average Packet Size        Warning Paacket

00 0B 86 A8 D0 F1  : 33 33 00 00 00 0C    3.455 KB/s      0.691 KB
```

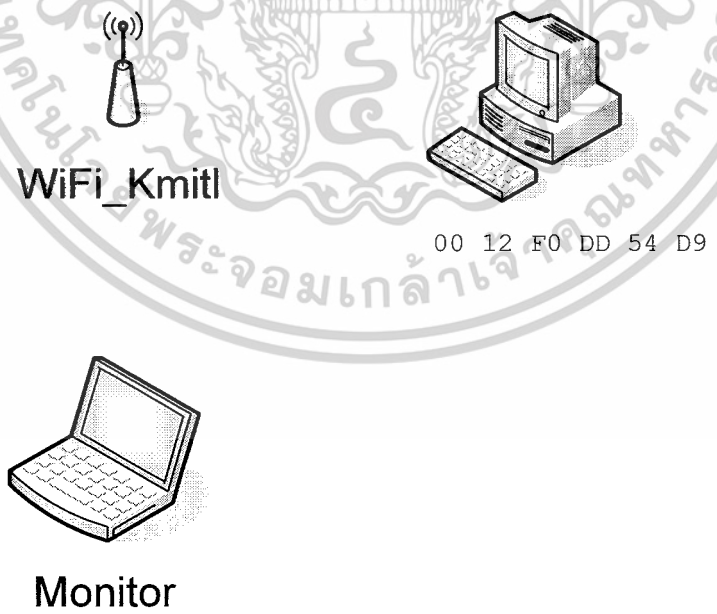
รูปที่ 5.7 แสดงไฟล์ที่บันทึกของผลการทดลองที่ 3

สรุปผลการทดลองที่ 1 2 และ 3

จากการทดลองใน 3 การทดลองแรกนั้นจะเห็นได้ว่าโปรแกรมของเราจะดักจับ Beacon Frame ซึ่งเป็นเฟรมชนิด Mananagement Frame ขึ้นมาแล้วทำการเก็บค่าแมคแอดเดรสและค่า SSID ไว้โดยโปรแกรมจะมีการทำการตรวจสอบค่าแมคแอดเดรส ถ้าเป็นแมคแอดเดรสที่มาจาก Beacon Frame ที่เป็นของแอคเซสพอยด์ตัวเดิมโปรแกรมก็จะไม่ทำการบันทึกค่าเพิ่มลงไป แต่ถ้าเป็นแมคแอดเดรสของแอคเซสพอยด์คนละตัวโปรแกรมก็จะทำการบันทึกค่าเพิ่มลงไป นอกจากนี้ ในการทดลองที่ 3 มีการเพิ่มเครื่องไคลเอนท์ที่มีอินเตอร์เฟสการ์ดเข้ามาติดต่อกับ แอคเซสพอยด์ที่มี SSID เป็น WIFI_KMITL จะเห็นว่าจากการที่เราทำการเปิดโปรแกรมเพื่อดักจับแพ็คเก็ตแล้วโปรแกรมจะแสดงแพ็คเก็ตโดยดาวน์โหลดข้อมูลจากแอคเซสพอยด์ขึ้นมาแสดงได้

5.3.4 การทดลองที่ 4

อุปกรณ์ภายในระบบประกอบด้วยเครื่องคอมพิวเตอร์ที่ทำการติดตั้งโปรแกรมจำนวน 1 เครื่อง แอคเซสพอยน์ 1 เครื่องที่มีค่า SSID เป็น WIFI_KMITL และเครื่องไคลเอนท์ที่มีหมายเลขแมคแอดเดรสเท่ากับ 00 0E 9B A3 D5 BD โดยการทดลองในครั้งนี้จะเป็นการทดลองโดยให้เครื่องไคลเอนท์ลองอัพโหลดและดาวน์โหลดไฟล์จากแอคเซสพอยต์โดยถ้าเครื่องไคลเอนท์มีค่าอัพโหลดหรือดาวน์โหลดผิดปกติจากค่าที่ตั้งไว้ ระบบจะมีการแจ้งเตือน



รูปที่ 5.8 แสดงระบบของการทดลองที่ 4

ขั้นตอนการทดลอง

1. ทำการเปิดโปรแกรมแล้วทำการรันโปรแกรมเพื่อดูผลลัพธ์ที่ได้
2. ให้เครื่องไคลเอนท์ทำการใช้งานแอคเซสพอยด์ที่มี SSID WIFI_KMITL
3. เซ็ตค่าเทรสโชของระบบ โดยการอัพโหลดให้ไม่เกิน 2 KB/S และค่าดาวน์โหลดให้มีค่าไม่เกิน 10 KB/S
4. ให้เครื่องไคลเอนท์อัพโหลดไฟล์ไปยังนอกระบบ
5. บันทึกผลการทดลอง

```
Date is 2008/05/21
Time is 15:37:06
Accesspoint in Network..

00 0B 86 A9 C7 60 :
00 0B 86 A9 C7 61 : WiFi_KMITL
00 04 E2 FF 6A D3 :

                                        --- UPLOAD ---

   Access point         Client          Upload/S        Average Packet Size        Warning Paacket

00 0B 86 A9 C7 61  : 00 12 F0 DD 54 D9    2.822 KB/s      0.202 KB                    ******

                                        --- DOWNLOAD ---

   Access point         Client          Download/S      Average Packet Size        Warning Paacket


00 0B 86 A9 C7 61  : 00 12 F0 DD 54 D9    1.018 KB/s      1.006 KB
```

รูปที่ 5.9 แสดงไฟล์ที่บันทึกของผลการทดลองที่ 4

```
Date is 2008/05/21
Time is 15:39:29
Accesspoint in Network..

00 0B 86 A9 C7 60 :
00 0B 86 A9 C7 61 : WiFi_KMITL
00 04 E2 FF 6A D3 :

                                                --- UPLOAD ---

   Access point          Client              Upload/S          Average Packet Size          Warning Paacket

00 0B 86 A9 C7 61  : 00 12 F0 DD 54 D9       0.133  KB/s        0.164 KB


                                              --- DOWNLOAD ---

   Access point          Client              Download/S        Average Packet Size          Warning Paacket

00 0B 86 A9 C7 61  : 00 12 F0 DD 54 D9       11.013 KB/s       1.352 KB                      ******
```

รูปที่ 5.10 แสดงไฟล์ที่บันทึกของผลการทดลองที่ 4

สรุปผลการทดลองที่ 4

จากการทดลองระบบของเราประกอบด้วย แอคเซสพอยต์ 1 เครื่องและเครื่องไคลเอนท์จำนวน 1 เครื่อง โดยเราให้เครื่องไคลเอนท์ภายในระบบทำการดาวน์โหลดและอัพโหลดน์ไฟล์จากแอคเซสพอยต์ ซึ่งระบบได้ทำการเซ็ตค่าเทรสโชของระบบไว้โดยไม่ให้มีการอัพโหลดเกินค่า 2 KB/S และดาวน์โหลดไม่ให้เกิด 10 KB/S ซึ่งถ้าคู่สัญญาณใดมีค่าเกินกว่าเทรสโชที่ระบบได้ตั้งไว้ระบบก็จะมีการแจ้งเตือนซึ่งจากรูปที่ 5.9 เครื่องไคลเอนท์มีการอัพโหลดน์ไฟล์ด้วยความเร็ว 2.822 KB/S ซึ่งเกินจากค่าเทรสโชที่ได้ตั้งไว้คือ 2 KB/S ระบบจะแจ้งเตือน ส่วนค่าดาวน์โหลดนั้นปกติ โดยดาวน์โหลดด้วยความเร็วเฉลี่ย 1 KB/S ระบบจะไม่ทำการแจ้งเตือนว่าคู่สัญญาณนั้นผิดปกติ ต่อมาดูในรูปที่ 5.10 เครื่องไคลเอนท์มีการดาวน์โหลดด้วยความเร็วเฉลี่ย 11.013 KB/S ซึ่งมีค่าเกินจากค่าเทรสโชที่ตั้งไว้ คือ 10 KB/S ระบบก็จะเตือนว่าในฝั่งของการดาวน์โหลดของเครื่องไคลเอนท์มีความผิดปกติ

บทที่ 6
บทสรุปและข้อเสนอแนะ

การสื่อสารในระบบเครือข่ายแลนแบบไร้สาย IEEE 802.11 นั้นตัวกลางที่ใช้เป็นสื่อในการสื่อสารก็คืออากาศ เราไม่สามารถที่จะรู้ได้เลยว่าภายในอากาศเวลานั้นมีการรับส่งกันภายในเครือข่ายเป็นจำนวนเครื่องไคลเอ็นต์กี่เครื่องหรือแอ็คเซสพอยท์กี่ตัวที่ทำงานอยู่ในเครือข่ายนั้น เพราะเราไม่สามารถมองเห็นแพ็คเก็ตที่วิ่งอยู่ด้วยตาเปล่าได้ การที่เราได้พัฒนาโปรแกรมที่ทำการดักจับแพ็คเก็ตภายในระบบเครือข่ายแลนแบบไร้สายทำให้เราสามารถรู้ได้ว่าในเวลานั้นมีแอ็คเซสพอยท์กี่เครื่องในเครือข่ายหรือมีเครื่องไคลเอ็นต์กี่เครื่อง มีการทำการรับส่งเฟรมอะไรบ้าง โดยรูปแบบของแพ็คเก็ตข้อมูลที่ใช้ในการรับส่งภายในระบบเครือข่ายแลนแบบไร้สายนั้นก็มีหลายรูปแบบ การที่เราจะวิเคราะห์ว่าแต่ละแพ็คเก็ตนั้นมีข้อมูลอะไรบ้าง เราต้องทำการแยกชนิดให้ได้ก่อนว่าแพ็คเก็ตนั้นเป็นข้อมูลในชนิดอะไรเราถึงจะบ่งบอกข้อมูลในฟิลด์ต่อๆ ไปของแพ็คเก็ตนั้นได้ เพราะข้อมูลของแพ็คเก็ตในแต่ละชนิดส่วนมากจะตายตัวตามมาตรฐาน IEEE 802.11

การที่เราพัฒนาโปรแกรมดักจับแพ็คเก็ตโดยใช้ ไลบรารีของ libpcap นั้นเราสามารถจับแพ็คเก็ตได้ลึกถึงในชั้นของดาต้าลิ้งค์ โดยโปรแกรมของเรานั้นสามารถเลือกจำนวนแพ็คเก็ตที่ต้องการดักจับได้ว่าเราต้องการเอาแพ็คเก็ตจำนวนเท่าไหร่ขึ้นมาแสดงผลในการแสดงครั้งนั้นหรือเราสามารถกำหนดค่าเป็นมิลลิวินาทีได้ ในการแสดงผลหากเราต้องการค่าเวลาที่ใช้ในการแสดงผลในช่วงนั้น และโปรแกรมของเราสามารถกรองแพ็คเก็ตที่ต้องการจะจับขึ้นมาได้ เช่น ต้องการให้แสดงผลแค่แพ็ตเก็ตชนิด Management โปรแกรมของเราก็สามารถทำได้

โปรแกรมของเรานั้นเป็นโปรแกรมที่ทำการวิเคราะห์แพ็คเก็ตและทำการตรวจสอบความปลอดภัยในระดับหนึ่ง โดยการวิเคราะห์แพ็คเก็ตนั้นจากความสามารถของโปรแกรมทำให้เรารู้ถึงการมีตัวตนของแอ็คเซสพอยท์ภายในระบบว่าระบบของเรามีแอ็คเซสพอยท์อยู่เป็นจำนวนทั้งหมดกี่เครื่องหรือรู้คู่สัญญาณว่าในเวลานั้นๆ มีเครื่องไคลเอ็นต์เครื่องไหนทำการติดต่อกับแอ็คเซสพอยท์ตัวใดบ้าง และการตรวจสอบความปลอดภัยของโปรแกรมเราก็พัฒนาอยู่ภายใต้สมมติฐานของการตรวจวัดค่าอัพโหลดและดาวน์โหลดของเครื่องไคลเอ็นต์กับแอ็คเซสพอยท์ในแต่ละคู่สัญญานโดยเราทำการตรวจวัดว่าถ้าคู่สัญญานไหนมีค่าอัพโหลดหรือดาวน์โหลดที่ผิดปกติโปรแกรมของเราก็สามารถที่จะทำการรับรู้ได้

จากการที่เราพัฒนาโปรแกรมดักจับแพ็คเก็ตและวิเคราะห์ข้อมูลและตรวจสอบความปลอดภัยได้นั้น เราสามารถนำโปรแกรมที่เราพัฒนาขึ้นมาไปพัฒนาต่อได้หลายรูปแบบ เพราะเราทราบว่าในแต่ละฟิลด์ของแพ็คเก็ตที่เราทำการดักจับได้บ่งบอกอะไรบ้าง หรือเป็นแพ็คเก็ตชนิดอะไร มาจากแอดเดรสไหน หรือต้องการส่งหาใคร หรือเป็นรูปแบบของเฟรมที่ใช้ในการทำคำสั่งอะไร และ

โปรแกรมของเรายังมีความยืดหยุ่นในส่วนของโค้ดในตัวโปรแกรม โดยยกตัวอย่างจากหลักการเบื้องต้นที่เราได้จากการทำโครงงานนี้ ซึ่งเราสามารถที่จะนำไปพัฒนาต่อในส่วนของความปลอดภัย เช่น พัฒนาเป็นโปรแกรมทางด้านความปลอดภัยในรูปแบบอื่นต่อไปได้ โดยอาจจะนำค่าของล็อก (log) ที่มีการเก็บไว้ หลังจากทำการจับแพ็คเก็ตมาตรวจสอบกับเงื่อนไข (rule) ที่มีการสร้างขึ้นว่าพบรูปแบบของการโจมตีระบบในแพ็คเก็ตรูปแบบไหนหรือเปล่า หรือในส่วนของการวิเคราะห์เราอาจวิเคราะห์ในส่วนอื่นๆ ของระบบหรือรูปแบบแพ็คเก็ตที่ได้รับได้หลากหลายยิ่งขึ้น

บรรณานุกรม

ชาวีร์ อิสริยภัทร์ , **การเขียนโปรแกรมดักจับแพ็ตเก็ต Libpcap** , ห้องปฏิบัติการวิจัยเทคโนโลยีเครือข่าย ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ

สันติ ศรีลาศักดิ์, วรวุฒิ เที่ยงธรรม, **เจาะประเด็นงานเขียนโปรแกรมบนลีนุกซ์**, นนทบุรี : ออฟเซ็ทเพรส , 2543

อรพิณ ประวัติบริสุทธิ์ , **คู่มือเรียนภาษาซี** , กรุงเทพมหานคร : โปรวิชั่น, 2543

โอภาส เอี่ยม สิริวงศ์ , **เครือข่ายคอมพิวเตอร์และการสื่อสาร**, กรุงเทพมหานคร : ซีเอ็ดยูเคชั่น, 2548

Neil Gray , **A Beginner C++** , United States of America. 2003

Matthew S. Gast, **802.11® Wireless Networks: The Definitive Guide, Second Edition**, O'Reilly Media , United States of America. 2005

Tim Carstens " **Programming with pcap**" [Online]. Available : http://www.tcpdump.org/pcap.htm

ประวัติผู้เขียน

ชื่อ-สกุล	นายจุติพงศ์ ชูตระกรู
เกิดเมื่อวันที่	8 เมษายน พ.ศ. 2529
สถานที่อยู่ปัจจุบัน	18/124-6 หมู่บ้านพระปิ่น 2 ถนนศาลาธรรมศพณ์ แขวง ศาลาธรรมศพณ์ เขตทวีวัฒนา กรุงเทพ 10170

ประวัติการศึกษา

มัธยมศึกษาตอนต้น	โรงเรียนวัดราชบพิธ
มัธยมศึกษาตอนปลาย	โรงเรียนวัดราชบพิธ
อุดมศึกษา	คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง