บทความ

“การเขียนโปรแกรมดักจับ packet ด้วย Libpcap ตอนที่ 1”

“Libpcap Programming Tutorial Part I”

โดย

นายชาวีร์ อิสริยภัทร์

ห้องปฏิบัติการวิจัยเทคโนโลยีเครือข่าย (Network Lab Technology)

ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC)

1.บทนำ

libpcap เป็น library สำหรับการดักจับ packet ที่ได้รับความนิยมอย่างสูง ถูกนำไปใช้งานอย่างแพร่หลาย ในปัจจุบันมีซอฟท์แวร์สำเร็จรูปมากมายที่พัฒนาด้วย libpcap ที่รู้จักกันดี ได้แก่ Ethereal, tcpdump, Snort, NMap, Ntop ฯลฯ จุดเด่นของ libpcap คือ มีรูปแบบ API ที่ใช้งานง่าย แต่มีประสิทธิภาพสูง สามารถจับ packet ได้ลึกถึงระดับ datalink ทั้งในโหมดปกติ และโหมด promiscuous อีกทั้งยังมีความสามารถในการเลือกจับเฉพาะบางชนิดของ packet ผู้ใช้ที่สนใจ และที่สำคัญ libpcap เป็น open-source library ที่สามารถให้นำมาใช้งานได้ฟรี

2. การติดตั้ง

ดาวน์โหลดซอสโค้ดของ libpcap จาก http://www.tcpdump.org/release/libpcap-0.9.7.tar.gz

จากนั้นแตกไฟล์ด้วยคำสั่ง

```
$ tar -xzf libpcap-0.9.7.tar.gz
```

เข้าไปใน directory libpcap-0.9.7 พิมพ์คำสั่ง ./configure ตามด้วยคำสั่ง make และ make install ตามลำดับ

```
$ ./configure
$ make
$ make install
```

3. การเขียนโปรแกรมใช้งาน Libpcap

ทุกโปรแกรมที่จะเรียกใช้ API ของ libpcap จะต้องทำการ include ไฟล์ pcap.h และต้องคอมไพล์โปรแกรมด้วยพารามิเตอร์ -lpcap เช่น :

```
$ gcc -o prog prog.c –lpcap
```

3.1 การค้นหา interface เพื่อการดักจับข้อมูล

ก่อนอื่นเราจำเป็นต้องทราบเสียก่อนว่าบนเครื่องของเรานั้นมี interface ใดอยู่บ้างที่เราสามารถเข้าไปดักจับข้อมูลออกมาได้ ภายใน API ของ libpcap นั้นจะอ้างอิงถึง interface ด้วยสตริงที่ระบุชื่อ เช่น "eth0", "eth1", "lo" ซึ่งหากทราบชื่อดังกล่าวก็เพียงแต่ระบุลงไป แต่หากไม่ทราบ libpcap ก็มีฟังก์ชันที่จะช่วยค้นหาชื่อของ interface ให้เรา prototype ของฟังก์ชันดังกล่าว คือ :

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

เมื่อเรียกใช้ฟังก์ชันนี้ ฟังก์ชันจะส่งค่ากลับมาเป็นชื่อของ interface ตัวแรกที่ไม่ใช่ loopback ซึ่งโดยทั่วไปแล้วมักจะเป็น interface ที่เราจะทำการดักจับข้อมูล หากมีความผิดพลาดเกิดขึ้นจากการเรียกฟังก์ชัน ข้อความแสดงสาเหตุของความผิดพลาดจะถูกส่งกลับมาทาง errbuf

ตัวอย่างการเรียกใช้ฟังก์ชัน pcap_lookupdev() :

```
#include <pcap.h>
#include <stdio.h>
int main() {
        char *device;
        char errbuf[PCAP_ERRBUF_SIZE];
        device = pcap_lookupdev(errbuf);
        if (device != NULL) printf("interface name: %s\n",device);
        else printf("error: %s.\n",errbuf);
        return 0;
}
```

ผลลัพธ์ :

```
interface name: eth0
```

ในกรณีที่มีหลาย interface อยู่บนเครื่องเดียวกัน แทนที่จะปล่อยให้ฟังก์ชัน pcap_lookupdev() เป็นผู้เลือก interface ให้ ผู้ใช้อาจต้องการทราบข้อมูลจำเพาะของ interface ทั้งหมดที่มีอยู่บนเครื่องก่อน เพื่อที่จะได้ตัดสินใจเลือกได้ถูกต้อง ในกรณีเช่นนี้ก็สามารถใช้ฟังก์ชัน pcap_findalldevs() เพื่อแสดงรายชื่อของ interface ทั้งหมดออกมา โดยฟังก์ชัน pcap_findalldevs() จะมี prototype ดังนี้ :

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

พารามิเตอร์แรกจะเป็นการส่งผ่านโดย reference กล่าวคือ alldevsp คือ address ของตัวแปรชนิด *pcap_if_t ที่เราผ่านเข้าไปเพื่อให้ฟังก์ชัน pcap_findalldevs() ทำการ allocate memory และสร้าง list ของ structure ของ interface อื่นๆต่อท้าย กล่าวอีกนัยหนึ่งก็คือ หลังจากการเรียกฟังก์ชันนี้ เราจะได้ linked-list ของข้อมูลชนิด pcap_if_t ที่มี alldevsp เป็น pointer ชี้ที่หัวของ list และเมื่อใช้งานเสร็จแล้ว เราก็สามารถคืนหน่วยความจำส่วนนี้ได้ด้วยการเรียกฟังก์ชัน

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

ตัวอย่างการเขียนโปรแกรมค้นหาชื่อ interface บนเครื่องด้วยฟังก์ชัน pcap_findalldevs() :

```
#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;
}
```

ผลลัพธ์ :

```
interface name: eth0
interface name: any
interface name: lo
```

อันที่จริงแล้ว นอกจากชื่อของ interface แล้ว ใน pcap_if_t ยังบรรจุรายละเอียดอื่นๆอีกหลายอย่าง 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() สามารถรายงานให้เราทราบ นอกจากชื่อของ interface แล้วยังมี description ซึ่งก็คือข้อความอธิบาย และ flags จะบอกคุณลักษณะเพิ่มเติมเกี่ยวกับ interface นั้นๆ ซึ่งในเวอร์ชันปัจจุบันใช้เพียงแค่รายงานว่าเป็น loopback หรือไม่ หาก interface นั้นเป็น loopback ค่า flags ก็จะถูกเซ็ตให้มีค่าเป็น PCAP_IF_LOOPBACK ในส่วนของ addresses นั้นจะบรรจุข้อมูลทั้งหลายที่เกี่ยวกับ network address ของ interface เช่น IPv4 address, IPv6 address, netmask ซึ่งจะขอกล่าวถึงรายละเอียดภายหลัง

3.2 การทดลองจับ packet อย่างง่าย

เมื่อเราได้ชื่อของ interface ที่ต้องการจะทำการดักจับข้อมูลเป็นที่เรียบร้อยแล้ว ถึงตรงนี้เราก็พร้อมที่จะเปิดการดักจับข้อมูล ซึ่งทำได้ด้วยการเรียกฟังก์ชัน

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

ค่าที่ส่งกลับมาจะเป็น handle สำหรับไว้ใช้อ้างอิง ในส่วนของพารามิเตอร์ของฟังก์ชัน ตัวแรกจะเป็นชื่อของ interface ที่เราจะทำการดักจับ (เช่น “eth0”) นอกจากระบุชื่อแล้วเรายังสามารถระบุเป็น "any" หรือ NULL ในกรณีที่ต้องการจับ packet ที่เข้ามาจากทุก interface สำหรับ snaplen คือขนาดของ buffer ที่จะให้จองเพื่อรอรับข้อมูล packet ที่จะเข้ามา, promisc เป็นการระบุว่าจะใช้ promiscuous mode หรือไม่ ซึ่งหากเปิดการดักจับใน mode นี้ทุก packet ที่ผ่านเข้ามาจะถูกดึงเข้ามาทั้งหมด แม้ว่า packet นั้นจะไม่ได้จ่าหน้าถึง interface นี้ก็ตาม, to_ms คือค่า read timeout มีหน่วยเป็นมิลลิวินาที หากกำหนดเป็นศูนย์จะหมายความว่าไม่มี timeout และท้ายสุดเช่นเคย *errbuf คือ pointer ชี้ตำแหน่งสำหรับเก็บ error message ไว้สำหรับรายงานความผิดพลาดจากการเรียกฟังก์ชัน

การเรียกฟังก์ชันนี้ เป็นเพียงแค่การเปิด session ของการดักจับ packet เท่านั้น การรอดักจับข้อมูลจริงๆนั้นจะเกิดขึ้นเมื่อมีการเรียกฟังก์ชัน :

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

โดยที่ p ก็คือ handle ที่ได้มาจากการเรียก pcap_open_live() ก่อนหน้านี้ และ h คือตำแหน่งของหน่วยความจำที่เราจะใช้รองรับ header ของข้อมูลที่จะเข้ามา ซึ่งส่วนนี้จะเป็น header ที่ libpcap เตรียมขึ้นมาเอง ไม่เกี่ยวข้องกับ header ของโปรโตคอลใดๆ มีไว้เพื่อบอกให้เราทราบถึงข้อมูลเพิ่มเติมเกี่ยวกับ packet ได้แก่ ts คือเวลาที่ได้รับ packet, caplen คือขนาดของก้อนข้อมูลที่จับมา และ len คือขนาดเต็มของ packet ซึ่งโดยปกติ caplen จะมีค่าเท่ากับ len เว้นเสียแต่ว่า packet ที่เข้ามานั้นใหญ่กว่าขนาดของ buffer ที่ได้เตรียมไว้ ข้อมูลที่จับได้ 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 จนกว่าจะมี packet ผ่านเข้ามาใน interface ที่กำลังดักฟังอยู่ ฟังก์ชันจึงจะ return ค่า pointer ที่ชี้ไบต์แรกของข้อมูล packet ที่จับมาได้ ข้อมูลก้อนนี้จะเริ่มจากไบต์แรกของ datalink protocol header ตามด้วย network protocol header , transport protocol header เรื่อยไปจนถึงไบต์สุดท้ายของ application payload (ถ้ามี) สำหรับ timestamp และขนาดของ packet นั้นไม่ได้เป็นส่วนหนึ่งของข้อมูล packet จึงเป็นเหตุผลว่าทำไม libpcap จำเป็นต้องมี header พิเศษเพิ่มเติมขึ้นมาต่างหาก และท้ายสุด ก่อนจะจบการทำงานก็ควรทำการปิด session ด้วย pcap_close()

ตัวอย่างโปรแกรมอย่างง่ายสำหรับดักจับ packet แรกที่ผ่านเข้ามาที่ eth0 :

```
#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 2007 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
```

3.3 การจับ packet แบบต่อเนื่อง

ในความเป็นจริงแล้ว คงจะมีไม่บ่อยนักที่เราจะพอใจกับ packet แรกเพียง packet เดียว เนื่องจากข้อมูลที่ส่งถึงกันโดยปกติมักจะมีขนาดใหญ่ และมักจะกระจายอยู่ในหลาย packet ดังนั้น application ที่จะเรียกใช้งาน libpcap จึงมักจะมีแนวโน้มที่จะต้องการกลไกในการจับข้อมูลแบบต่อเนื่อง วิธีหนึ่งที่สามารถทำได้ก็คือการสร้าง loop ขึ้นมาครอบ pcap_next() แต่ยังมีอีกวิธีหนึ่งที่สะดวกกว่า นั่นก็คือการใช้ฟังก์ชัน pcap_loop() ที่ libpcap เตรียมไว้ให้ และนำกลไกของ callback function เข้ามาช่วย

เมื่อพูดถึง callback function โดยทั่วไปจะหมายถึงฟังก์ชันที่ถูกกำหนดไว้ให้โปรแกรมมีการกระโดดไปเรียกเมื่อเกิดเหตุการณ์บางอย่างขึ้น ซึ่งในกรณีนี้ก็คือเหตุการณ์ที่มี packet ใหม่เข้ามา สิ่งที่เราต้องทำก็มีเพียงแค่การเตรียม callback function สำหรับการประมวลผลข้อมูล packet และบอกให้ libpcap รับทราบว่าจะต้องกระโดดมาเรียกฟังก์ชันนี้ด้วยการเรียกใช้ฟังก์ชัน pcap_loop() ซึ่งมี prototype คือ :

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

พารามิเตอร์ตัวแรกคือ session handle พารามิเตอร์ถัดมา cnt ก็คือจำนวน packet ที่เราจะรอรับ หากระบุเป็นค่าติดลบหมายความว่าให้ทำการจับ packet ไปเรื่อยๆจนกว่าจะเกิดความผิดพลาดอย่างหนึ่งอย่างใดขึ้นภายใน ส่วน callback ก็คือ function pointer ที่ชี้ไปยังฟังก์ชันที่เราต้องการให้กระโดดไปทำงานเมื่อมี packet ใหม่เข้ามา โดยที่ฟังก์ชันดังกล่าวจะต้องมี prototype ที่สอดคล้องกับ typedef นี้ :

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

กล่าวคือจะต้องเป็นฟังก์ชันที่รับพารามิเตอร์ 3 ตัว พารามิเตอร์ตัวแรกจะเป็นพารามิเตอร์ของการ callback, พารามิเตอร์ตัวถัดมาจะเป็น pointer ชี้ header ที่ libpcap สร้างขึ้น และพารามิเตอร์ตัวสุดท้ายคือ pointer ที่ชี้ก้อนข้อมูล packet

```
#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 timestr[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 2007 17:10:54 ICT
Packet of the size 104 bytes received at 26 Apr 2007 17:10:54 ICT
Packet of the size 42 bytes received at 26 Apr 2007 17:10:55 ICT
Packet of the size 76 bytes received at 26 Apr 2007 17:10:55 ICT
Packet of the size 104 bytes received at 26 Apr 2007 17:10:56 ICT
```

อันที่จริงแล้ว pcap_next() และ pcap_loop() ต่างก็มีข้อจำกัดด้วยกันทั้งคู่ นั่นคือโปรแกรมจะต้องมาหยุดรอที่จุดนี้เสมอจนกว่าจะมี packet ใหม่เข้า ซึ่งอาจก่อให้เกิดความยุ่งยากต่อการออกแบบโปรแกรม โดยเฉพาะโปรแกรมที่มีการควบคุมการทำงานผ่าน user interface ในกรณีนี้เราอาจพิจารณาเลือกใช้ฟังก์ชัน pcap_dispatch() แทน วิธีนี้จะเปิดโอกาสให้เราเลือกเองได้ว่าจะให้ฟังก์ชันมีการหยุดรอหรือไม่ ซึ่งจะต้องใช้คู่กันกับฟังก์ชัน pcap_setnonblock() prototype ของฟังก์ชันเป็นดังนี้ :

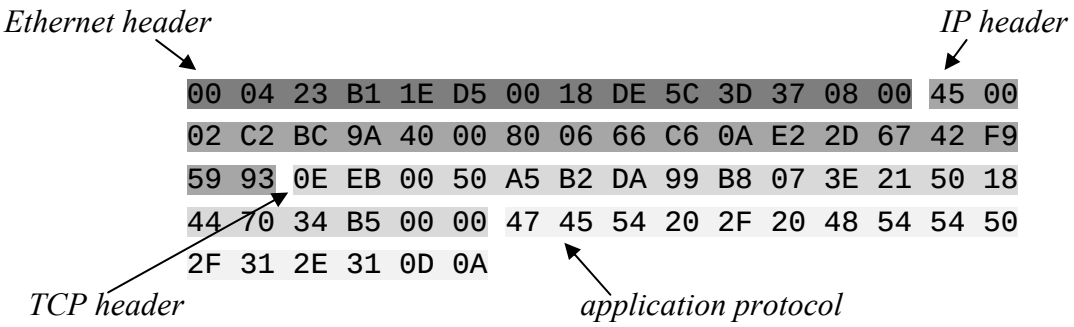
```
int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user);
int pcap_setnonblock(pcap_t *p, int nonblock, char *errbuf);
```

ฟังก์ชัน pcap_dispatch() จะมีการรับพารามิเตอร์เหมือนกับฟังก์ชัน pcap_loop() ทุกประการ ในส่วนของ pcap_setnonblock() นั้นจะใช้สำหรับการเลือกโหมดของ handle ว่าจะให้มีการหยุดรอจนกว่าจะมี packet ใหม่เข้ามา หรือว่าจะให้ return ทันทีแม้ไม่พบ packet ใหม่ ซึ่งเราสามารถกำหนดได้ด้วยการตั้งค่าของ nonblack เป็น 0 หรือเป็นตัวเลขอื่น

3.4 ตัวอย่างการประยุกต์ใช้งาน libpcap

เพื่อให้เห็นภาพการทำงานที่ชัดเจนขึ้น จึงขอยกตัวอย่างการเขียนโปรแกรมใช้ libpcap เพื่อตรวจวัดอัตราการอัพโหลด/ดาวน์โหลดข้อมูลแบบง่ายๆ หลักการพื้นฐานของโปรแกรมนี้จะใช้วิธีการจับทุก packet ที่ผ่านเข้ามาใน interface และคัดเลือกเอาเฉพาะ IPv4 packet มาตรวจสอบ header หากพบว่ามี source IP address ตรงกับ IP address ของ interface ก็แปลว่า packet นั้นเป็นข้อมูลอัพโหลดของ interface จึงจะทำการบวกขนาดของข้อมูลชุดนั้นทบเข้าไปในขนาดรวมเพื่อรอการคำนวณค่าเฉลี่ยของอัตราการอัพโหลดต่อไป ในขณะเดียวกันก็ทำการคำนวณอัตราการดาวน์โหลดในในลักษณะเดียวกันด้วย

ในการตรวจสอบรายละเอียดต่างๆของข้อมูล จำเป็นต้องอาศัยการตีความจากข้อมูลดิบของ packet ที่เราดักจับมาได้ โดยทั่วไปโครงสร้างของ packet ที่รับเข้ามาจะมีลักษณะของ header หุ้มเป็นชั้นๆ จากระดับชั้น datalink ไปจนถึงชั้น application ดังแสดงในภาพ



เราสามารถตรวจสอบโปรโตคอลของข้อมูลภายใต้เฟรมได้จากส่วนของ header ของ frame ซึ่งในกรณีของโปรโตคอล Ethernet ที่นิยมใช้กันใน Local Area Network นั้น โครงสร้างของ frame จะเป็นดังนี้

Ethernet frame header

Destination address	Source address	Type	Data	CRC
6 bytes	*6 bytes*	*2 bytes*	*46 - 1500 bytes*	*4 bytes*

header ของ Ethernet frame เริ่มต้นด้วย destination address และ source address ขนาด 6 ไบต์ บอกให้เราทราบว่า frame นี้ส่งออกมาจากที่ใด และจะส่งไปยังที่ใด แต่สิ่งที่เราสนใจจริงๆจะเป็นส่วนที่อยู่ถัดมา ซึ่งจะบ่งบอกชนิดโปรโตคอลของ payload ของ frame นั้น ตัวอย่างเช่นหากค่านี้เป็น 0x0800 ก็จะหมายถึง IPv4, 0x0806 จะหมายถึง ARP ฯลฯ ดังนั้นเราจึงสามารถตรวจสอบโปรโตคอล layer 3 โดยอ่านค่าจากไบต์ที่ 12 และ 13 นับจากส่วน

หัวของ libpcap packet (ซึ่งเริ่มจากไบต์ที่ 0) ซึ่งหากมีค่าเป็น 0x08 และ 0x00 ตามลำดับก็หมายความว่าข้อมูลภายใต้ frame นั้นเป็น IPv4 datagram

สำหรับ source และ destination IP address ที่เราต้องการนำมาตรวจสอบนั้น ต้องดึงออกมาจาก header ของ IP ซึ่งจะมีโครงสร้างดังนี้

Version *4 bits*	HLen *4 bits*	Services *8 bits*	Total length *16 bits*	
Identification *16 bits*			Flag *3 bits*	Fragmentation offset *13 bits*
Time to live *8 bits*		Protocol *8 bits*	Header checksum *16 bits*	
Source IP address *32 bits*				
Destination IP address *32 bits*				
Option *≤ 40 bytes*				

จากแผนผังจะเห็นว่าส่วนที่เป็น source IP address นั้นมีความยาว 4 ไบต์ เริ่มต้นที่ไบต์ที่ 12 จนถึงไบต์ที่ 15 และส่วนที่เป็น destination IP address จะเริ่มจากไบต์ที่ 16 จนถึงไบต์ที่ 19 ดังนั้นหากนับตั้งแต่ส่วนหัวของ frame ตำแหน่งเริ่มต้นของ source และ destination IP จะอยู่ที่ไบต์ที่ 26 และ 30 ตามลำดับ

Type = IP (0x0800)

```
00 04 23 B1 1E D5 00 18 DE 5C 3D 37 08 00 45 00
02 C2 BC 9A 40 00 80 06 66 C6 0A E2 2D 67 42 F9
59 93 0E EB 00 50 A5 B2 DA 99 B8 07 3E 21 50 18
44 70 34 B5 00 00 47 45 54 20 2F 20 48 54 54 50
2F 31 2E 31 0D 0A
```

Source IP = 10.226.45.103 Destination IP = 66.249.89.147

กล่าวโดยสรุปก็คือ ในการตรวจสอบว่า pcap_packet ที่จับมาได้นั้นเป็นข้อมูลที่เป็นการ download ของ interface นี้หรือไม่ ก็เพียงแต่ตรวจสอบว่า

1. pcap_packet[12] == 0x08 และ pcap_packet[13] == 0x00 ใช่หรือไม่ และ
2. เลข 32 บิตที่ตำแหน่งที่ 30 ของ pcap_packet มีค่าเท่ากับเลข 32 บิตของ IP address ของ interface หรือไม่

เราสามารถนำแนวคิดนี้ไปสร้างโปรแกรมบวกนับขนาด packet และคำนวณอัตราการอัพโหลดและดาวน์โหลดแบบง่ายๆ โดยอาศัยความรู้พื้นฐานเท่าที่ได้เรียนรู้มาตั้งแต่ต้น ได้ดังนี้ :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pcap.h>
#include <fcntl.h>
#include <signal.h>
#include <netinet/in.h>

#define INTERVAL 1

u_int32_t ifaddr;
int upload = 0;
int download = 0;
char ifname[16];
char errbuf[PCAP_ERRBUF_SIZE];
const u_char *packet;
struct sockaddr_in *addr;
pcap_if_t *devlist, *wk;
pcap_t *handle;

void finalize() {
	printf("Program terminated..\n\n");
	pcap_close(handle);
	exit(0);
}
```

```
void report(){
        system("clear");
        printf("Press Ctrl-C to terminate..\n\n");
        printf("uploading rate = %.1f KB/s \n",(float)(upload)/(float)(INTERVAL*1024));
                                /* แสดงอัตราการรับส่งข้อมูลเฉลี่ย ซึ่งเท่ากับ ขนาดของข้อมูลหารด้วยเวลา */
        printf("downloading rate = %.1f KB/s \n",
                (float)(download)/(float)(INTERVAL*1024));
        upload = 0;             /* เคลียร์ค่าสำหรับการนับรอบใหม่*/
        download = 0;
        alarm(INTERVAL);   /* ตั้งเวลาให้กลับมาเรียกฟังก์ชันอีกใน 1 วินาที */
}

void process_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {

        if ( *(packet+12) == 0x08 && *(packet+13) == 0x00) {
                                /* เลือกเฉพาะ frame ที่มี type เป็น 0x0800 ซึ่งหมายความว่า */
                                /* frame นี้บรรจุ IP datagram */
                if (*(u_int32_t *)(packet + 26) == ifaddr)
                        upload += header->len - 14;
                                /* ถ้าในส่วนของ IP header มี source ip เท่ากับ ifaddr ก็จะบวกค่า */
                                /* upload เพิ่มด้วยขนาดของ packet ที่ไม่รวม Ethernet frame header */

                if (*(u_int32_t *)(packet + 30) == ifaddr)
                        download += header->len - 14;
                                /* ถ้าในส่วนของ IP header มี destination ip เท่ากับ ifaddr */
                                /* ก็จะบวกค่า download เพิ่ม */
```

```
	}
}

int main(int argc, char *argv[]) {
	if (argc != 2) {
		printf("Usage: bw <interface name>\n\n");
		return -1;
	}
	strcpy(ifname,argv[1]);  /* ชื่อ interface ที่จะทำการจับ packet */
	pcap_findalldevs(&devlist,errbuf);
	for(wk=devlist;wk!=NULL;wk=wk->next){
		if (0 == strcmp(ifname,wk->name==NULL?"":wk->name)) {
			if (wk->addresses != NULL) {
				addr = (struct sockaddr_in *)(wk->addresses->next->addr);
				/* structure ที่บรรจุข้อมูลเกี่ยวกับ IPv4 ของ interface */
				ifaddr = ((addr->sin_addr).s_addr);
				/* IPv4 address เป็นเลข 32 บิต*/
			}
			break;
		}
	}

	if (wk == NULL) {
		printf("interface %s not found.\n",ifname);
		return -2;
	}
```

```
        else if (wk->addresses == NULL) {
                printf("interface %s does not have an ip address.\n",ifname);
                return -3;
        }

        handle = pcap_open_live(ifname, 1024, 1, 0, errbuf);
        pcap_setnonblock(handle, 0, errbuf);
                                        /* ให้ฟังก์ชัน pcap_dispatch() หยุดคอย packet */

        signal(SIGINT, finalize);       /* เมื่อมีการกด Ctrl-C ให้เรียกฟังก์ชัน finalize() */
        signal(SIGALRM, report);        /* เมื่อเกิด SIGALRM ให้เรียกฟังก์ชัน report() */
        report();
        while(1){                       /* วนลูปไปเรื่อยๆ */
                pcap_dispatch(handle, 1, process_packet, NULL);
                                        /* เมื่อมี packet เข้ามาจะเรียกฟังก์ชัน process_packet() */
        }
}
```

ผลลัพธ์ :

```
Press Ctrl-C to terminate..

uploading rate = 1.2 KB/s
downloading rate = 54.8 KB/s
```

4. ส่งท้าย

ที่ผ่านมา เราได้กล่าวถึงการติดตั้ง libpcap และสาธิตตัวอย่างการเขียนโปรแกรมเพื่อดักจับ packet รวมถึงการตีความข้อมูล packet แบบง่ายๆ ในตอนต่อไป จะลงลึกในรายละเอียดของการวิเคราะห์ข้อมูล packet ตั้งแต่ระดับชั้น datalink ขึ้นไปจนถึงระดับ application รวมไปถึงการคัดกรองชนิดของ packet ผ่าน API ของ Libpcap และการนำ pattern ของ L7 มาใช้ในการจำแนกประเภทของ application protocol

เอกสารอ้างอิง

[1] Stevens, Richard. TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley Professional, 1993.

[2] Carstens, Tim. “Programming with pcap”, http://www.tcpdump.org/pcap.htm

[3] “Manpage of PCAP”, http://www.tcpdump.org/pcap3_man.html