

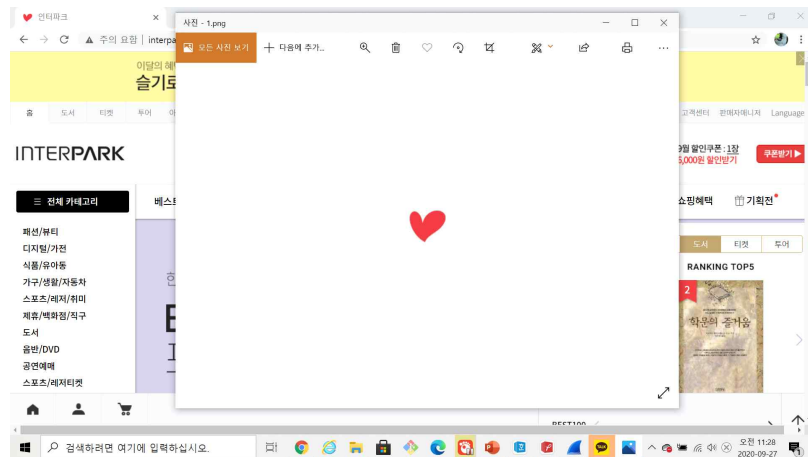
Computer Networks Project 1

컴퓨터과학과 2015147533 유현석

1-1 wireshark

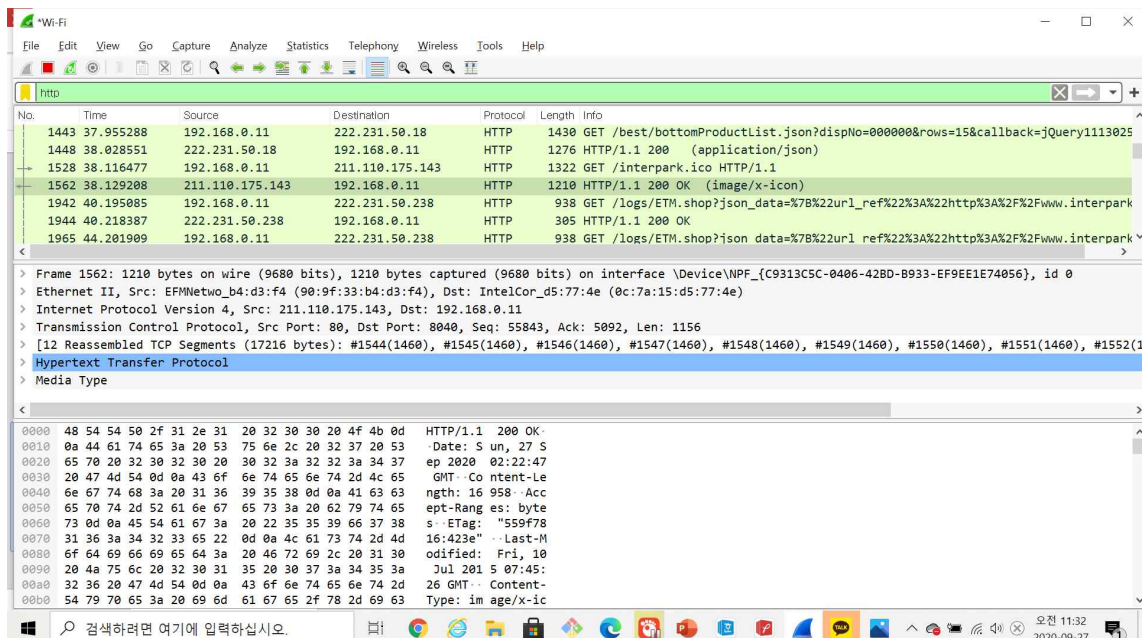
1. Exported image and Webpage and Packet Information

Exported image:



Webpage : <http://www.interpark.com/>

Packet Information :



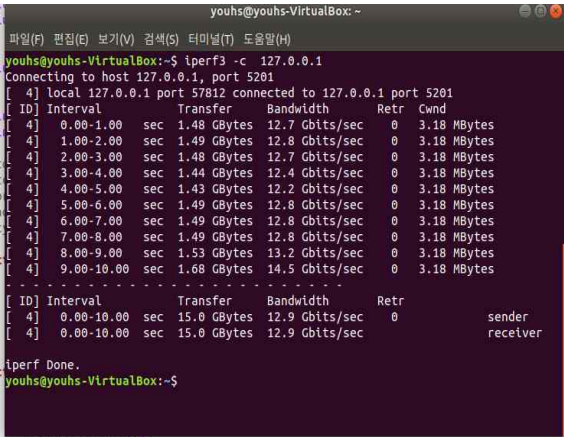
2. Comments for each step on capturing packets and exporting images

1. Install the wireshark and execute the wireshark
2. Find out the website that use HTTP
3. After find out the site then check the packet that contain the image information.
4. See hypertext transfer protocol, then it have the file data which is image.
5. Export packet byte and save as PNG file name.
6. Check the image file.

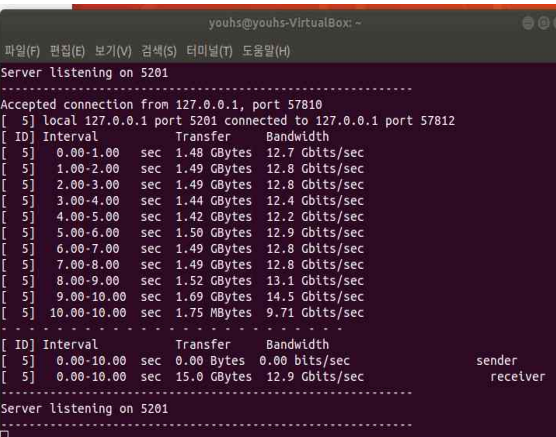
1-2 iperf3

1. Screenshot of IPerf3

<client>



<server>



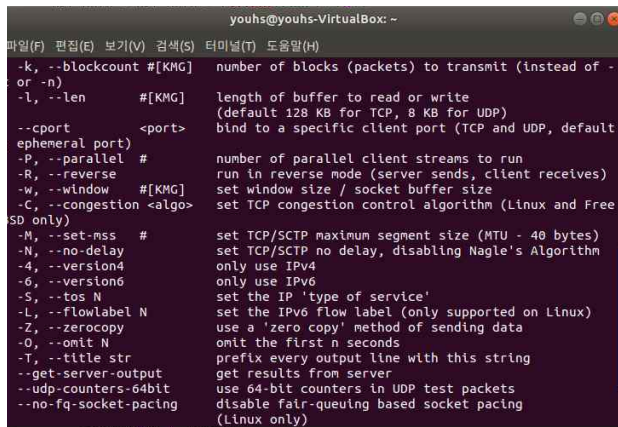
2. Explanation of Transfer Bitrate and Cwnd

Transfer : 1초동안 전송한 용량을 나타냅니다.

Bitrate : 특정한 시간 단위(초 단위)마다 처리하는 비트의 수

Cwnd : 송신자가 보낼 수 있는 segment의 크기

3. Description of several commands -b -n -w -l



```
youhs@youhs-VirtualBox: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
-k, --blockcount #[KMG] number of blocks (packets) to transmit (instead of -  
or -n)  
-l, --len #[KMG] length of buffer to read or write  
                        (default 128 KB for TCP, 8 KB for UDP)  
--cport <port> bind to a specific client port (TCP and UDP, default  
ephemeral port)  
-P, --parallel # number of parallel client streams to run  
-R, --reverse run in reverse mode (server sends, client receives)  
-w, --window #[KMG] set window size / socket buffer size  
-C, --congestion <algo> set TCP congestion control algorithm (Linux and Free  
SD only)  
-M, --set-mss # set TCP/SCTP maximum segment size (MTU - 40 bytes)  
-N, --no-delay set TCP/SCTP no delay, disabling Nagle's Algorithm  
-4, --version4 only use IPv4  
-6, --version6 only use IPv6  
-S, --tos N set the IP 'type of service'  
-L, --flowlabel N set the IPv6 flow label (only supported on Linux)  
-Z, --zerocopy use a 'zero copy' method of sending data  
-O, --omit N omit the first n seconds  
-T, --title str prefix every output line with this string  
--get-server-output get results from server  
--udp-counters-64bit use 64-bit counters in UDP test packets  
--no-fq-socket-pacing disable fair-queuing based socket pacing  
                        (Linux only)
```

-b : bandwidth n 대역폭을 n bits/sec로 설정합니다.

-n : bytes n 전송할 바이트를 설정합니다.

-w : 소켓 버퍼 사이즈를 설정합니다.

-l : 버퍼 읽고 쓰는 크기를 설정합니다.

1-3 packet capture coding

1. Introduction

Code language : C

Version :

```
youhs@youhs-VirtualBox:~$ gcc --version  
gcc (Ubuntu 7.5.0-3ubuntu1-18.04) 7.5.0  
Copyright (c) 2017 Free Software Foundation, Inc.
```

Os :

```
youhs@youhs-VirtualBox:~$ cat /etc/issue  
Ubuntu 18.04.4 LTS \n \l
```

Library : <stdio.h> <pcap.h>

2. Block Diagram

1. 전역 변수 와 패킷에 포함된 정보를 저장한 구조체들

```
int total_header_size = 0; //from ip header to end
int tcp_header_size = 0; //TCP header size
int proto = 0; //TCP(1) or UDP(2)
int no2 = 0; //HTTP(1) or DNS(2)
int type = 0; //Each num of loop

struct eth_header
{
    u_char ether_dhost[6];
    u_char ether_shost[6];
    u_short ether_type;
};

//ether header struct
struct ip_header
{
    u_char ip_info[2];
    u_char ip_size[2];
    u_short ip_info2[2];
    u_char ip_info3[12];
};

struct tcp_header
{
    u_char tcp_info[4];
    u_char tcp_info2[8];
};

//tcp struct
struct dns_header
{
    u_char dns_info[12];
};

//dns struct
struct http_header
{
    u_char http_info[10000];
};

//http header struct
```

-> 전역 변수 설명 :

total_header_size : ip 헤더부터 시작하는 전체 패킷의 길이 저장

tcp_header_size : TCP header가 추가적인 정보를 갖고 있다면 그 길이 저장

proto : TCP인지 UDP인지 여부 파악

no2 : HTTP 인지 DNS인지 여부 파악

type : 패킷이 몇 번째 순서인지 알 수 있고 그에 따라 request인지 response인지 구분

-> 구조체 설명

eth_header : 패킷에서 이더넷에 해당하는 정보를 저장할 구조체

ip_header : 패킷에서 아이피에 해당하는 정보를 저장할 구조체

tcp_header : 패킷에서 TCP에 해당하는 정보를 저장할 구조체

dns_header : 패킷에서 DNS에 해당하는 정보를 저장할 구조체

http_header : 패킷에서 HTTP에 해당하는 정보를 저장할 구조체

2. 패킷에 포함된 정보인 이더넷 헤더 함수

```
//=====ether-net header =====
/*
    struct eth_header *eth = (struct eth_header *)pkt_data;
    u_char *b=eth->ether_dhost;
    printf("xx : xx : xx : xx : xx : xx fenkfenk\n",b[0],b[1],b[2],b[3],b[4],b[5]);
*/
    pkt_data = pkt_data + (sizeof(char) *14);
//===== when the HTTP not in =====
```

-> 패킷에서 이더넷에 부분에 해당하는 정보를 저장할 수 있고 그 후 이더넷에 해당하는 크기만큼 이동하여 IP 헤더의 시작점으로 포인터 이동 할 수 있습니다.

3. HTTP에 해당하는 정보를 얻기 위한 함수 안에 있는 IP 헤더 함수

```
//=====when the HTTP get in =====
if(no2==1){
    //=====ip header =====
    struct ip_header *ip = (struct ip_header *)pkt_data;
    pkt_data = pkt_data + (sizeof(char) *2);
    u_char *sll_size = ip->ip_size;
    total_header_size = (sll_size[0]*256+sll_size[1]);
    total_size=n; //ip total size
    pkt_data = pkt_data + (sizeof(char) *6);
    u_char *sll_ip2 = ip->ip_info;
    //checking what protocol type (6=tcp 17=udp)
    if(sll_ip2[1] == 6){
        proto =1;
    }
    else{
        proto =2;
    }
    //printf("sd protocol\n",sll_ip2[1]); // 1= icmp 6=tcp 17=udp
    pkt_data = pkt_data + (sizeof(char) *(12));

    if(proto ==1){
        //=====tcp header =====
    }
}
```

-> no2에 저장된 정보에 따라서 http에 해당 될 때 해당 패킷에 있는 IP 헤더의 정보를 저장할 수 있게 해주는 부분입니다. 여기에서 total_header_size의 정보를 저장하고 protocol 타

입의 정보를 저장합니다.(TCP일 경우 1, UDP일 경우 2를 저장합니다) 또한, source의 ip 정보와 destination ip 정보 역시 저장해줍니다. 그 후 역시 다음 헤더부분으로 이동해줍니다.

4. HTTP에 해당하는 정보를 얻기 위한 함수 안에 있는 TCP 헤더 함수

```
//=====tcp header =====
struct tcp_header *tcp = (struct tcp_header *)pkt_data;
u_char *sli_tcp = tcp->tcp_info;
//source port
s_port = (sli_tcp[0]*256)+sli_tcp[1];
//print the source port

//destination port
d_port = (sli_tcp[2]*256)+sli_tcp[3];

pkt_data = pkt_data + (sizeof(char) * (4));
u_char *sli_tcp2 = tcp->tcp_info2;

//tcp data offset size
tcp_header_size = (sli_tcp2[0]/10);
printf("tcp data offset size: %d\n", tcp_header_size);
//
if(tcp_header_size>5){
    pkt_data = pkt_data + (sizeof(char) * (16+((tcp_header_size-5)*4)));
}
else{
    pkt_data = pkt_data + (sizeof(char) * (10));
}

}

if(proto==2){
//=====udp header =====
```

-> 위에 IP 파트에서 그 정보가 TCP로 전송 되었다고 알려주면 실행됩니다. 이때 source port와 destination port의 정보를 저장해줍니다. 그리고 TCP의 추가적인 정보가 있는지 여부를 파악하고 만약 존재하면 그 크기만큼 추가로 이동하여 HTTP 헤더가 시작하는 지점으로 이동해줍니다.

5. HTTP에 해당하는 정보를 얻기 위한 함수 안에 있는 UDP 헤더 함수

```
//=====udp header =====
struct tcp_header *tcp = (struct tcp_header *)pkt_data;
u_char *sli_tcp = tcp->tcp_info;
//source port
s_port = (sli_tcp[0]*256)+sli_tcp[1];
//destination port
d_port = (sli_tcp[2]*256)+sli_tcp[3];

pkt_data = pkt_data + (sizeof(char) * (8));
tcp_header_size = 5;

}

//=====http header =====
```

-> 위에 IP 파트에서 그 정보가 UDP로 전송 되었다고 알려주면 실행됩니다. 이때 source port와 destination port의 정보를 저장해줍니다. 그 후 HTTP 헤더가 시작하는 지점으로 이동해줍니다.

6. HTTP에 해당하는 정보를 얻기 위한 함수 안에 있는 HTTP 헤더 함수

```
//=====http header =====
total_header_size = (total_header_size-40)-((tcp_header_size-5)*4);
if(total_header_size !=0){
    struct http_header *http = (struct http_header *)pkt_data;
    u_char *sli_http = http->http_info;

    int boolhttp=0;
    for(int a=0;a<total_header_size;a++){
        if(sli_http[a] == '\r'){
            if(sli_http[a+1] == '\n' && sli_http[a+2] == '\n' && sli_http[a+3] == '\n'){
                boolhttp=1;
            }
        }
    }
    if(boolhttp==1){
        type++;
        printf("type: %d\n", type);
        printf("source ip: %d.%d.%d.%d\n", sli_ip2[4], sli_ip2[5], sli_ip2[6], sli_ip2[7]); //print the source ip
        printf("source port: %d\n", s_port);
        printf("destination ip: %d.%d.%d.%d\n", sli_ip2[8], sli_ip2[9], sli_ip2[10], sli_ip2[11]); //print the destination ip
        printf("destination port: %d\n", d_port); //print the destination port
        printf("HTTP\n");
        if(type%2 == 1){
            printf("Request\n");
        }
        else{
            printf("Response\n");
        }
        for(int a=0;a<total_header_size;a++){
            printf("%c", sli_http[a]);
            if(sli_http[a] == '\r'){
                if(sli_http[a+1] == '\n' && sli_http[a+2] == '\n' && sli_http[a+3] == '\n'){
                    pkt_data = pkt_data + (sizeof(char) * (total_header_size));
                    break;
                }
            }
        }
    }
}

//=====when the DNS get in =====
```

-> HTTP로 해당하는 데이터가 들어왔지만 그 정보가 정확히 HTTP 헤더인지 파악을 하기

위해서 먼저 첫 번째 줄에 HTTP 정보가 들어있는지 파악을 해줍니다. 그 후 지금까지 저장한 정보들(response, request 여부 s_ip, s_port, d_ip, d_port)을 출력하고 헤더와 바디를 구분하여 출력해줍니다. 헤더와 바디를 구분 짓는 과정은 \r\n\r\n을 기준으로 잘라 주었습니다.

7. DNS에 해당하는 정보를 얻기 위한 함수 안에 있는 IP 헤더 함수

```

//*****when the DNS get in *****
else{
//=====ip header =====
struct ip_header *ip = (struct ip_header *)pkt_data;

pkt_data = pkt_data + (sizeof(char) * 2);
u_char *sli_tcp = ip->ip_info;

total_header_size = (sli_size[0]*256+sli_size[1]);
//printf("Md ip total size\n", total_header_size); //ip total size

pkt_data = pkt_data + (sizeof(char) * 6);
u_char *sli_ip2 = ip->ip_info2;

//checking what protocol type (6=tcp 17=udp)
if(sli_ip2[1] == 0){
    proto =1;
}
else{
    proto =2;
}
//printf("Md protocol\n",sli_ip2[1]); // 1= icnp 6=tcp 17=udp
pkt_data = pkt_data + (sizeof(char) * (12));
if(proto ==1){

```

-> no2에 저장된 정보에 따라서 DNS에 해당 될 때 해당 패킷에 있는 IP 헤더의 정보를 저장할 수 있게 해주는 부분입니다. 여기에서 total_header_size의 정보를 저장하고 protocol 타입의 정보를 저장합니다.(TCP일 경우 1, UDP일 경우 2를 저장합니다) 또한, source의 ip 정보와 destination ip 정보 역시 저장해줍니다. 그 후 역시 다음 헤더부분으로 이동해줍니다.

8. DNS에 해당하는 정보를 얻기 위한 함수 안에 있는 TCP 헤더 함수

```

//=====tcp header =====
struct tcp_header *tcp = (struct tcp_header *)pkt_data;
u_char *sli_tcp = tcp->tcp_info;
//source port
s_port = (sli_tcp[0]*256)+sli_tcp[1];
//destination port
d_port = (sli_tcp[2]*256)+sli_tcp[3];
pkt_data = pkt_data + (sizeof(char) * (4));
u_char *sli_tcp2 = tcp->tcp_info2;

//tcp data offset size
tcp_header_size = (sli_tcp2[0]/16);
printf("Md Dataoffset\n", tcp_header_size);
//
if(tcp_header_size>5){
    pkt_data = pkt_data + (sizeof(char) * (16*((tcp_header_size-5)*4)));
}
else{
    pkt_data = pkt_data + (sizeof(char) * (16));
}
}
//jump tcp to dns with data offset size
if(proto ==2){
//=====udp header =====

```

-> 위에 IP 파트에서 그 정보가 TCP로 전송 되었다고 알려주면 실행됩니다. 이때 source port와 destination port의 정보를 저장해줍니다. 그리고 TCP의 추가적인 정보가 있는지 여부를 파악하고 만약 존재하면 그 크기만큼 추가로 이동하여 HTTP 헤더가 시작하는 지점으로 이동해줍니다.

9. DNS에 해당하는 정보를 얻기 위한 함수 안에 있는 UDP 헤더 함수

```

if(proto ==2){
//=====udp header =====
struct tcp_header *tcp = (struct tcp_header *)pkt_data;
u_char *sli_tcp = tcp->tcp_info;
//source port
int s_port = (sli_tcp[0]*256)+sli_tcp[1];
//destination port
int d_port = (sli_tcp[2]*256)+sli_tcp[3];
pkt_data = pkt_data + (sizeof(char) * (8));
//jump udp to dns
}
//=====dns header =====

```

-> 위에 IP 파트에서 그 정보가 UDP로 전송 되었다고 알려주면 실행됩니다. 이때 source port와 destination port의 정보를 저장해줍니다. 그 후 HTTP 헤더가 시작하는 지점으로 이동해줍니다.

10. DNS에 해당하는 정보를 얻기 위한 함수 안에 있는 DNS 헤더 함수

```
//=====dns header =====
//print the information
printf("id ", type);
printf("id.%d.%d.%d", sll_ip2[4], sll_ip2[5], sll_ip2[6], sll_ip2[7]); //print the source ip
printf("id ", s_port);
printf("id.%d.%d.%d", sll_ip2[8], sll_ip2[9], sll_ip2[10], sll_ip2[11]); //print the destination
printf("id ", d_port);
printf("DNS ID : ");

struct dns_header *dns = (struct dns_header *)pkt_data;
u_char *sll_dns = dns->dns_info;

//DNS ID print
printf("%02x%02x\n", sll_dns[0], sll_dns[1]);

int pri = 0;
int num1 = sll_dns[2];
int num2 = sll_dns[3];

//DNS QR
printf("id | ", ((num1/16)/8));
//DNS OP code
printf("id", ((num1/16)%8)/4);
printf("id", (((num1/16)%8)%4)/2);
printf("id", (((num1/16)%8)%4)%2);
printf("id | ", ((num1%16)/8));

//DNS AA
printf("id | ", ((num1%16)/4));
//DNS TC
printf("id | ", (((num1%16)%8)%4)/2);
//DNS RD
printf("id | ", (((num1%16)%8)%4)%2);
//DNS RA
printf("id | ", ((num2/16)/8));
//DNS Z
printf("id", ((num2/16)%8)/4);
printf("id", (((num2/16)%8)%4)/2);
printf("id | ", (((num2/16)%8)%4)%2);
//DNS RCODE
printf("id", ((num2%16)/8));
printf("id", ((num2%16)%8)/4);
printf("id", (((num2%16)%8)%4)/2);
printf("id", (((num2%16)%8)%4)%2);
//DNS INFO
printf("QDCOUNT : %x \n", (sll_dns[4]*256)+sll_dns[5]);
printf("ANCOUNT : %x \n", (sll_dns[6]*256)+sll_dns[7]);
printf("NSCOUNT : %x \n", (sll_dns[8]*256)+sll_dns[9]);
printf("ARCOUNT : %x \n", (sll_dns[10]*256)+sll_dns[11]);
printf("\n");
}
```

-> DNS로 들어온 것을 확인한 후, 위에서 얻은 정보들(response, request 여부 s_ip, s_port, d_ip, d_port))을 출력해줍니다. 그 후 DNS 헤더에 해당하는 정보들을 출력해줍니다. (DNS_ID -> 16진수로 변환, QR, OPcode, AA, TC, RD, RA, Z, Rcode -> 2진수로 변환, QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT -> 16진수로 변환)

11. main 함수 (network device 선택, HTTP or DNS 선택, 위에서 정의한 함수 불러오기)

```
int main(int argc, char **argv) {
    pcap_t *adhandle; //packet capture descriptor
    char errbuf[PCAP_ERRBUF_SIZE]; //error buf
    pcap_if_t *alldevs; //network interface
    pcap_if_t *d; //network interface

    bpf_u_int32 netp;
    bpf_u_int32 maskp;
    char *dev;
    char *net;
    char *mask;
    int ret;

    struct pcap_addr *a;
    int i = 0;
    int no;

    dev = pcap_lookupdev(errbuf);
    if(dev == NULL){
        printf("%s\n", errbuf);
        return 0;
    }
    printf("DEV : %s\n", dev);
    ret = pcap_lookupnet(dev, &netp, &maskp, errbuf);
    if(ret == -1){
        printf("%s\n", errbuf);
        return -1;
    }
    if (pcap_findalldevs(&alldevs, errbuf) < 0) {
        printf("pcap_findalldevs error\n");
        return -1;
    }
    for (d=alldevs; d; d=d->next) {
        printf("id : %s\n", ++i, (d->description)?(d->description):(d->name));
    }
    printf("number : ");
    scanf("%d", &no);

    printf("Http =1 or Dns=2 number : ");
    scanf("%d", &no2);

    if ((no1 > 0 && no1 <= 1)) {
        printf("number error\n");
        return -1;
    }
    for (d=alldevs, i=0; d; d=d->next) {
        if (no1 == ++i) break;
    }
    if(no2 == 1){
        if ((adhandle= pcap_open_live(d->name, 65536, 1, 1000, errbuf))){
            printf("pcap_open_live error %s\n", d->name);
            pcap_freealldevs(alldevs);
            return -1;
        }
        struct bpf_program fcode;
        if (pcap_compile(adhandle, // pcap handle
            &fcode, // compiled rule
            FILTER_RULE1, // filter rule
            0, // optimize
            netp < 0){
            printf("pcap compile failed\n");
            pcap_freealldevs(alldevs);
            return -1;
        }
        if (pcap_setfilter(adhandle, &fcode) < 0){
            printf("pcap compile failed\n");
            pcap_freealldevs(alldevs);
            return -1;
        }
        pcap_freealldevs(alldevs);
        pcap_loop(adhandle, 0, packet_handler, NULL);
        pcap_close(adhandle);
    }
    return 0;
}

else if(no2 == 2){
    if ((adhandle= pcap_open_live(d->name, 65536, 1, 1000, errbuf))){
        printf("pcap_open_live error %s\n", d->name);
        pcap_freealldevs(alldevs);
        return -1;
    }
    struct bpf_program fcode;
    if (pcap_compile(adhandle, // pcap handle
        &fcode, // compiled rule
        FILTER_RULE2, // filter rule
        0, // optimize
        netp < 0){
        printf("pcap compile failed\n");
        pcap_freealldevs(alldevs);
        return -1;
    }
    if (pcap_setfilter(adhandle, &fcode) < 0){
        printf("pcap compile failed\n");
        pcap_freealldevs(alldevs);
        return -1;
    }
    pcap_freealldevs(alldevs);
    pcap_loop(adhandle, 0, packet_handler, NULL);
    pcap_close(adhandle);
}
return 0;
}
```

-> main 함수 설명

pcap_lookupdev : 해당하는 device를 찾아줍니다.

pcap_lookupnet : 해당하는 net 정보를 저장합니다.

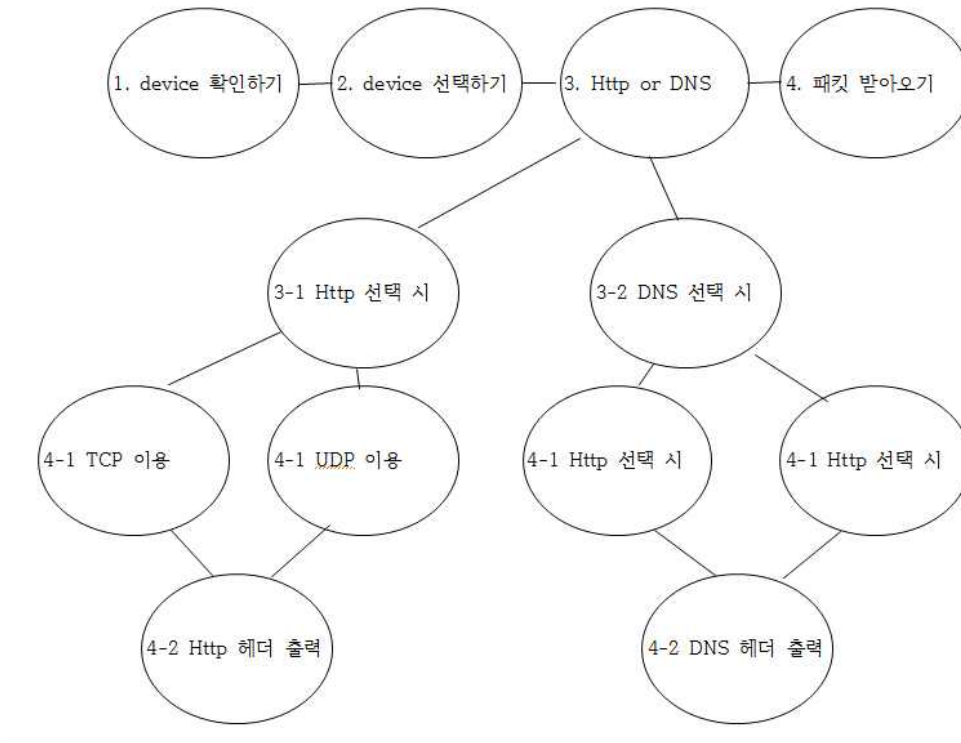
pcap_findalldevs : 모든 device의 정보를 찾고 그 후 출력해줍니다.

이후에 device를 선택하고 http와 dns를 선택해줍니다.

pcap_compile + pcap_setfilter : http와 dns 중 선택한 정보만 패킷으로 수신합니다.

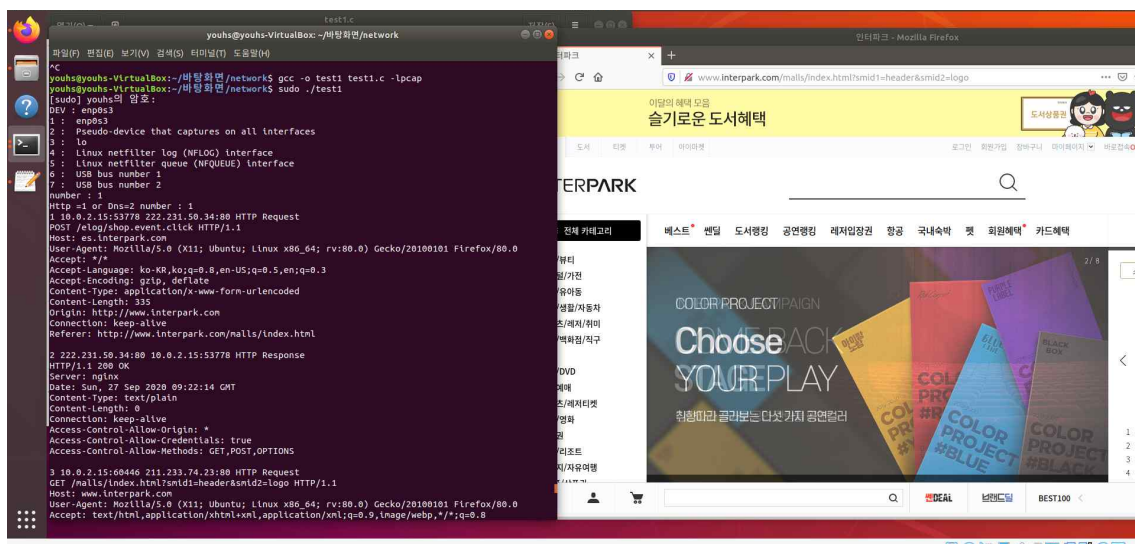
pcap_loop : filter를 통과해 들어오는 패킷 정보들을 packet_handler 함수로 통해 처리합니다.

3. Flow Chart

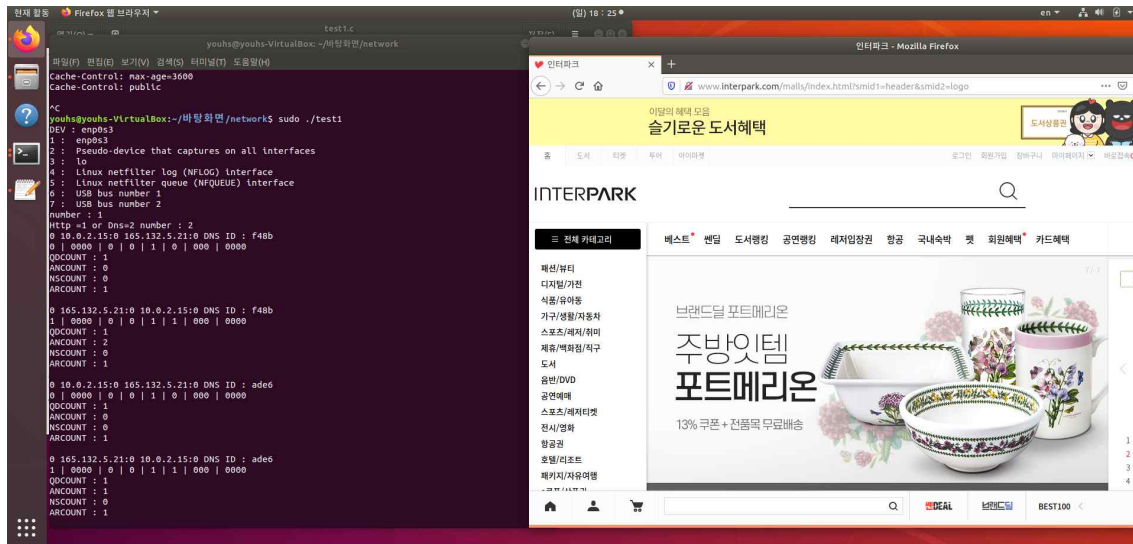


4. Result screenshot

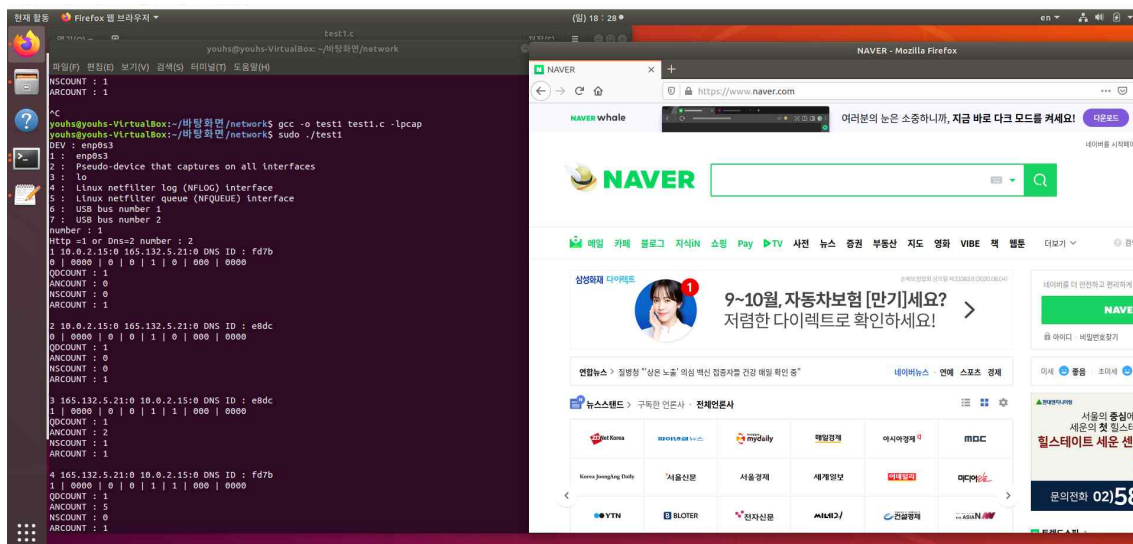
1. Interpark (HTTP)



2. Interpark (DNS)



3. Naver (DNS)



5. Reference

1. <https://kskang.tistory.com/476>
2. <https://hand-over.tistory.com/30>
3. <https://www.tcpdump.org/>