2019년 데이터통신

- HW 02 -

제출일자	2019.03.08
이　　름	노효근
학　　번	201502049
분　　반	02

.

* 과제 목표

각 패킷의 종류 및 헤더구조를 알고, python socket programming과 struct 모듈을 통해 패킷캡처를 구현해서(이더넷,ip,tcp,udp)을 확인한다.

* Ethernet 과 IP_header

```
while True:
    packet = socket.recvfrom(20000)

    ethernet_header = struct.unpack('!6s6s2s',packet[0][0:14])
    dst_ethernet_addr = init(ethernet_header[0].hex())
    src_ethernet_addr = init(ethernet_header[1].hex())
    protocol_type = "0x" + ethernet_header[2].hex()

    print("<<<<<<Packet Capture Start>>>>>>")
    print("=======ethernet_header=======")
    print("src_mac_address : ", src_ethernet_addr)
    print("dest_mac_address : ", dst_ethernet_addr)
    print("ip_version : ", protocol_type)
```

이더넷 헤더를 보면 Destination Mac Address는 6바이트, Source Mac Address는 6바이트, Ether Type은 2바이트로 이루어져있음을 알 수 있다. 따라서 이더넷 헤더를 6바이트, 6 바이트, 2바이트씩 쪼개서 packet[0][0:14] 배열에 담은 후 바이트 수마다 잘린 값을 16진수로 변환하여 출력하면 된다. Mac주소를 xx:xx:xx:xx:xx:xx의 형식으로 출력하기 위해 반복문을 0에서 5까지 돌며 Mac address 정보에서 2의 배수만큼 쓰고 ':' 기호를 출력하도록 하여 반환하였다.

```
if(protocol_type=="0x0800"):
    print("=======ip_header=======")

    ip_header = struct.unpack('!1B1B1H1H1H1B1B1H', packet[0][14:26])

    version = ip_header[0]>>4
    print("ip_version : ", version)

    header_length = (ip_header[0]&15)<<2
    print("ip_Length : ", header_length)

    dif_ser_code = ip_header[1]//4
    print("differentiated_service_codepoint : ", dif_ser_code)

    ex_con_noti = ip_header[1]%4
    print("explicit_congestion_notification : ", ex_con_noti)

    ip_total_length = ip_header[2]
    print("total_length : ", ip_total_length)

    identification = ip_header[3]
    print("identification : ", identification)
```

IP 헤더 또한 위의 이더넷 헤더와 같은 방식으로 하면 된다. version과 header length는 합쳐서 1바이트, total length, identification, flag&flagment offset은 각각 2바이트, ttl, protocol type은 각 1바이트, check sum은 2바이트, source ip address, destination ip address는 4바이트라고 나와있 으므로 각 바이트 크기에 맞게 1바이트인 경우는 B, 2바이트인 경우는 H, 그리고 source ip 주소와 destination ip 주소는 4바이트지만 1바이트씩 써야하므로 파이썬 struct 모듈을 BBBB로 표현해서 파싱해준다. header length는 ip헤더의 길이를 설정하는 필드로서 4bit로 이루어져 있다. 1당 4바이트의 크기를 뜻하며 default 값은 5로서 헤더의 길이가 20byte로 설정된 것을 의미한다. flag는 ip_off의 상위 3비트가 flag의 비트이며 3비트 중 제일 처음에 오는 비트는 사용되지 않고 두 번째 오는 비트가 1이라면 해당 패킷에 대해서는 단편화할 수 없다는 표시이다. 마지막 비트는 more fragment 비트로 이 비트가 1로 설정되어있으면 패킷이 단편화 되었는 데 아직 끝이 아니라는 의미이다. 따라서 offset은 13비트로 데이터 파싱을 하면 되므로 오른쪽으로 shift연산을 13번 한 다.

* TCP_header 과 UDP_header

```
        n=0
        if(header_length>20):
            n = header_length-20
            opt_pad = struct.unpack('!3B1B', packet[0][34:(34+n)])

        if(protocol == 6):
#tcp 진행
            print("=======tcp_header=======")


            tcp_header = struct.unpack('!1H1H1L1L1B1B1H1H1H', packet[0][(34+n):
(54+n)])

            tcp_srcPort = tcp_header[0]
            print("src_port : ", tcp_srcPort)
```

처음 n = 0으로 변수를 설정해놓고, ipv4에서 헤더의 길이가 20이 넘는 경우 20바이트 넘 는 만큼을 n에 저장하고 파싱하도록 한다. tcp는 protocol값이 6인 경우이므로 조건문을 사용하여 if(protocol == 6)인 경우에만 tcp헤 더를 파싱하도록 한다.
TCP 헤더를 보면 source port, destination port는 각각 2바이트, sequence number, acknowledgement number는 각각 4바이트, data offset은 4bits, Reserved는 1bits, NS, CWR, ECE URG, ACK, PSH, RST, SYN, FIN 각각 1bits, window size, checksum, urgent pointer는 각각 2바이트씩이므로 각 바이트에 맞게 데이터를 unpack해준다.
Data offset은 4bits로 1바이트 정수로 데이터를 파싱한 값에 1111 0000(== 240)을 &연산 해주고, 오른쪽으로 4비트만큼 shift연산을 해주면 된다. Reserved는 3bits로 1바이트 정수로 데이터를 파싱한 값에 1비트만큼 오른쪽으로 shift연산 한 다음 0000 0111(== 7)을 &연산해서 구한다. NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN은 각각 1bits로 각각 1바이트 정수로 파싱한 값에 shift연산을 해줌으로써 구하면 된다.

```
        if(protocol == 17):
#udp 진행
            print("=======udp_header=======")
            udp_header = struct.unpack('!1H1H1H1H', packet[0][(34+n):(42+n)])

            udp_srcPort = udp_header[0]
            print("src_port : ", udp_srcPort)

            udp_dstPort = udp_header[1]
            print("dest_port : ", udp_dstPort)
```

UDP 헤더를 보면 Source Port, Destination Port, Length, Checksum 각각 2바이트씩이므 로 파싱한 데이터를 udp_header에 담아 출력해주기만 하면 된다.
데이터 파싱하는 방식은 앞에 했던 ipv4와 TCP와 같다.

```
u201502049@u201502049:~/바탕화면/DC$ sudo python3 DC02.py
<<<<<<Packet Capture Start>>>>>>
=======ethernet_header=======
src_mac_address :  00:50:56:f7:86:f9
dest_mac_address :  00:0c:29:f5:17:e3
ip_version :  0x0800
=======ip_header=======
ip_version :  4
ip_Length :  20
differentiated_service_codepoint :  0
explicit_congestion_notification :  0
total_length :  76
identification :  43643
flags :  0
>>>reserved_bit :  0
>>>non_fragments :  0
>>>fragments :  0
>>>fragment offset :  0
Time_to_live :  128
protocol :  17
header_checksum =  11384
source_ip_address :  91.189.89.198
dest_iP_address :  192.168.237.129
=======udp_header=======
src_port :  123
dest_port :  39042
leng :  56
header checkSum :  29674
```

```
u201502049@u201502049:~/바탕화면/DC$ sudo python3 DC02.py
<<<<<<Packet Capture Start>>>>>>
=======ethernet_header=======
src_mac_address :  00:50:56:f7:86:f9
dest_mac_address :  00:0c:29:f5:17:e3
ip_version :  0x0800
=======ip_header=======
ip_version :  4
ip_Length :  20
differentiated_service_codepoint :  0
explicit_congestion_notification :  0
total_length :  40
identification :  43644
flags :  0
>>>reserved_bit :  0
>>>non_fragments :  0
>>>fragments :  0
>>>fragment offset :  0
Time_to_live :  128
protocol :  6
header_checksum =  28535
source_ip_address :  210.89.160.88
dest_iP_address :  192.168.237.129
=======tcp_header=======
src_port :  443
dec_port :  56636
seq_num :  1207034550
ack_num :  3352148084
head_len :  20
flags :  80
>>>reserved :  0
>>>nonce :  0
>>>cwr :  0
>>>urgent :  0
>>>ack :  1
>>>push :  0
>>>reset :  0
>>>syn :  0
>>>fin :  0
window_size_value :  64240
checksum :  2597
urgent_pointer :  0
```