모바일 컴퓨팅 HW4

2018037356 – 안동현

1. 아래는 google.co.kr 에 tracert 를 실행한 결과이다. 또한 동시에 와이어샥을 통해 ICMP 패킷을 캡쳐하였다.

(a) (각 패킷의 routing path 가 동일하다는 가정하에) 4 번째 hop 까지의 ping latency 가 더 먼 거리를 왕복하는 5 번째 hop 의 ping latency 보다 긴 이유를 설명하시오. (5 pts)

먼저 가정으로 각 패킷의 routing path가 동일하다고 했습니다. 해당 가정을 매번 보내는 모든 ICMP 요청 패킷이 항상 동일한 루트로 움직인다고 해석한다면,

4번째 홉이 5번째 홉보다 레이턴시가 긴 이유는 둘의 경로가 다르다고 해석할 수는 없습니다.

6번 홉을 보면 3번째 핑에 대한 레이턴시가 현저히 느린 것을 볼 수 있습니다. 그 이유는 저희가 사용하는 네트워크는 불안정하여서 트래픽에 따라 패킷이 손실될 수도 있고, 전송 속도가 느려질 수도 있습니다.

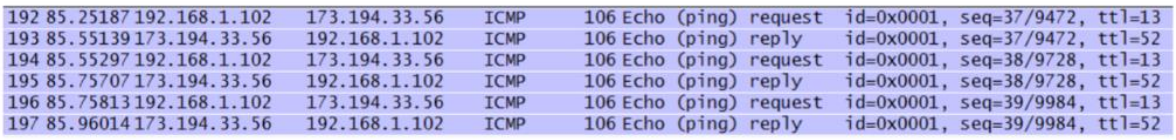
특히나 라우터의 경우 ICMP에 대한 답장은 사실 해도 그만 안해도 그만인 우선순위가 굉장히 낮다고 볼 수 있는 프로토콜입니다.

패킷은 홉을 지날 때 마다 TTL을 하나씩 깎고, TTL이 0이 되면 패킷 손실에 대한 ICMP를 준비하고 다시 발송합니다. 이때 다른 트래픽에 대한 처리가 더 우선이라면 이 과정은 더 미뤄질 수 밖

에 없고 거기에 대한 추가적인 레이턴시가 4번째 홉에서 라우팅 테이블을 보고 바로 5번째 홉으로 패킷을 던져주는 것보다 더 길 가능성이 있다는 소리입니다.

따라서 이러한 연유로 4번째 홉의 핑 레이턴시보다 5번째 홉의 핑 레이턴시가 더 짧은 것이라고 생각할 수 있습니다.

(b) 아래의 ping request, reply를 보면 TTL값이 다른것을 알 수 있다. 왜 그런지 설명하시오. (request와 reply가 거쳐가는 hop count는 갖다고 가정) (5 pts)



192	85.25187	192.168.1.102	173.194.33.56	ICMP	106	Echo (ping) request id=0x0001, seq=37/9472, ttl=13
193	85.55139	173.194.33.56	192.168.1.102	ICMP	106	Echo (ping) reply id=0x0001, seq=37/9472, ttl=52
194	85.55297	192.168.1.102	173.194.33.56	ICMP	106	Echo (ping) request id=0x0001, seq=38/9728, ttl=13
195	85.75707	173.194.33.56	192.168.1.102	ICMP	106	Echo (ping) reply id=0x0001, seq=38/9728, ttl=52
196	85.75813	192.168.1.102	173.194.33.56	ICMP	106	Echo (ping) request id=0x0001, seq=39/9984, ttl=13
197	85.96014	173.194.33.56	192.168.1.102	ICMP	106	Echo (ping) reply id=0x0001, seq=39/9984, ttl=52

먼저 request에 대한 ICMP는 저희의 네트워크에서 나간 패킷입니다. 저희가 tracerout을 통해서 경로를 알기 위해 일부러 TTL을 1부터 2,3,.... 이런식으로 설정해서 보낸 패킷이고 만약 목적지에 도착했다면 목적지에서는 핑에 대한 리퀘스트로 핑을 날려주는 겁니다. (여기에서의 ICMP는 Type: 11 (Time-to-live exceeded) 가 아니고 그냥 요청 핑에 대한 답장 핑입니다.) 다시 말해서 원래대로라면 TTL을 경로에 딱 맞춘 값으로 보내지 않는다는 겁니다.

위 사진에서 출발지 173.194.33.56은 저희가 보낸 패킷이 아니라 해당 IP주소의 머신이 저희에게 답장을 준 것입니다. 따라서 해당 위치에서 설정돼 있는 TTL로 패킷이 오는 것입니다.

저희의 머신에서 상대방한테까지 보내는데 13이라는 TTL이 필요했으므로, 아마도 상대방에서 저희에게 보낸 해당 패킷은 초기에는 52 + 13 = 65로 설정돼 있었고, 라우터를 지나면서 TTL이 깎이고 52가 되었을 때 저희의 머신에서 와이어 샤크가 캡쳐를 했다고 판단할 수 있습니다.

2. 수업시간에 보여준 데모와 같이 www.unza.zm에 (외부 사이트로 연결합니다.) ping을 보내고, 동시에 wireshark을 활용하여 packet을 capture하세요. 캡쳐한 icmp packet을 각 필드별로 살펴본뒤 관련내용을 문서로 작성하세요. (5 pts)

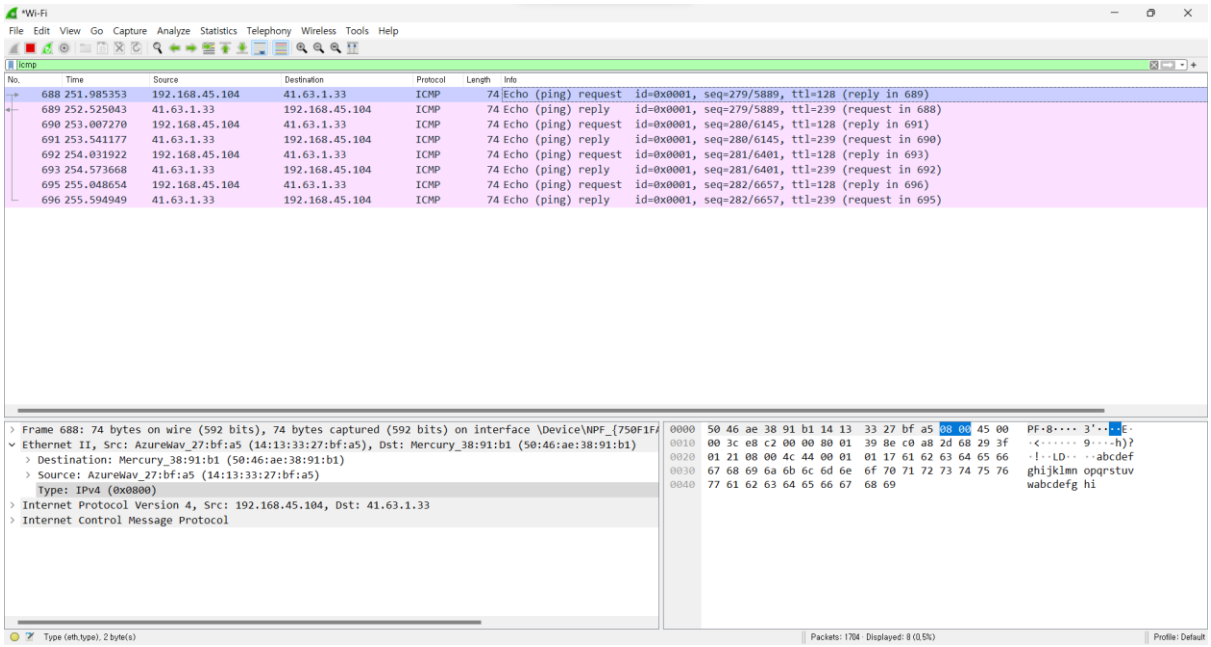
```
C:₩Windows₩System32>ping www.unza.zm

Ping www.unza.zm [41.63.1.33] 32바이트 데이터 사용:
41.63.1.33의 응답: 바이트=32 시간=539ms TTL=239
41.63.1.33의 응답: 바이트=32 시간=534ms TTL=239
41.63.1.33의 응답: 바이트=32 시간=541ms TTL=239
41.63.1.33의 응답: 바이트=32 시간=546ms TTL=239

41.63.1.33에 대한 Ping 통계:
    패킷: 보냄 = 4, 받음 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
    최소 = 534ms, 최대 = 546ms, 평균 = 540ms

C:₩Windows₩System32>
```

먼저 해당 사이트에 핑을 던져봤습니다. 총 네번의 시도가 있었고, 네번의 시도 모두 패킷 손실 없이 도착하는 모습을 볼 수 있었습니다.



다음은 해당 핑에 대해서 wireshark가 캡쳐한 패킷들의 목록입니다.

4번의 요청에 대해서 4번의 답장이 온 것을 확인 할 수 있었습니다.

TTL을 보면 저의 네트워크에서 나간 패킷은 128의 크기를 가지고 있었습니다만, 이것이 목적지까지의 홉의 개수를 말해주지는 않습니다. 위에서 언급했듯, 일부러 trace route를 위해서 TTL을 조정해주지 않는 이상 넉넉잡아서 보내주기 때문입니다.

사이트가 보낸 답장의 TTL은 239임을 확인 할 수 있었습니다. 해당 TTL은 여러 홉을 거쳐서 저의

네트워크에 들어온 이후의 TTL이므로, 실제로는 이것보다 높게 설정해서 보냈을 것입니다.

이제 패킷을 더 세심히 살펴보겠습니다.

길이는 모두 74바이트 구조를 가지고 있습니다. wire shark는 OS 커널 어딘가에서 패킷을 캡쳐하고, 해당 위치에서는 이더넷 프레임의 헤더, 역시 같이 캡쳐됩니다. 과제 1에서 언급했듯, crc는 기본적으로 하드웨어 수준에서 생성되거나, 이미 FCS 검사를 통과한 상태로 들어오기에 제외하므로

14바이트의 이더넷 헤더가 있고, IP 헤더가 20바이트가 붙습니다. 이후, ICMP 메시지에서

ICMP헤더가 8바이트 이고, 나머지 데이터 부분으로 채워집니다. 이때 여기서 캡쳐한 패킷은 에러 메시지가 아니라 쿼리 메시지 입니다! 따라서 IP헤더와 TCP 헤더를 데이터로 들고있지는 않습니다.

이때 14 + 20 + 8 = 42인 만큼 74 – 42 = 32바이트가 남고 해당 부분이 데이터의 길이입니다.

이렇게 한다면 핑에 대한 패킷 길이가 74바이트로 표시되는 점이 어느정도 납득이 갑니다.

```
Ethernet II, Src: AzureWav_27:bf:a5 (14:13:33:27:bf:a5), Dst: Mercury_38:91:b1 (50:46:ae:38:91:b1)
  > Destination: Mercury_38:91:b1 (50:46:ae:38:91:b1)
  > Source: AzureWav_27:bf:a5 (14:13:33:27:bf:a5)
    Type: IPv4 (0x0800)
```

```
Internet Protocol Version 4, Src: 192.168.45.104, Dst: 41.63.1.33
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xe8c2 (59586)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x398e [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.45.104
    Destination Address: 41.63.1.33
```

```
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4c44 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 279 (0x0117)
    Sequence Number (LE): 5889 (0x1701)
    [Response frame: 689]
  v Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677616263646566676869
      [Length: 32]
```

예측한 대로, 이더넷 헤더 14바이트, IP헤더 20바이트, ICMP 헤더 8바이트. Data section 32바이트

를 확인할 수 있습니다.

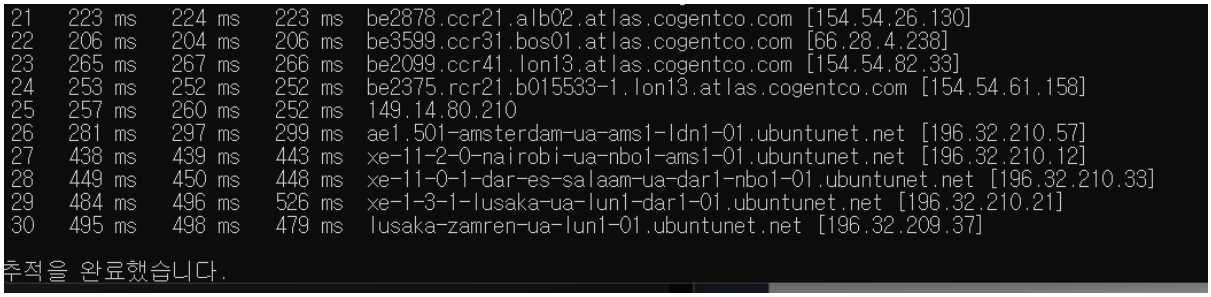
```
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x5444 [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 279 (0x0117)
    Sequence Number (LE): 5889 (0x1701)
    [Request frame: 688]
    [Response time: 539.690 ms]
  Data (32 bytes)
```

답장 역시 비슷한 구조를 가지고 있고, 핑의 Type이 8 대신 0임을 확인할 수 있습니다.

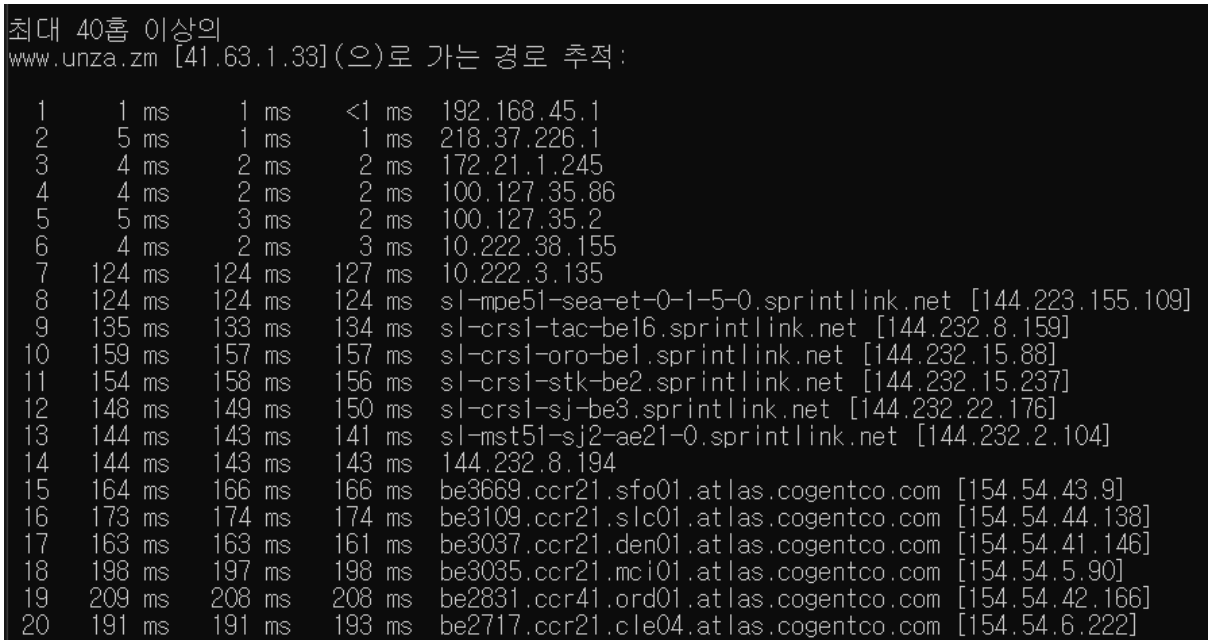
또한 이러한 데이터들로 미루어 볼 때

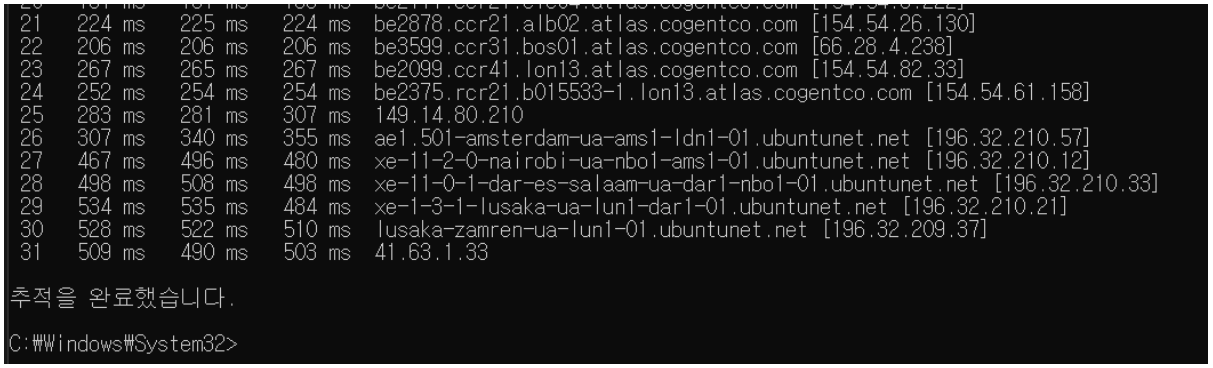
위 32바이트 사용이 바로 Data section의 길이라고 생각했습니다.

3. 수업시간에 보여준 데모와 같이 www.unza.zm에 (외부 사이트로 연결합니다.) traceroute을 해보고, 동시에 wireshark을 활용하여 packet을 capture하세요. 캡쳐한 icmp packet을 각 필드별로 살펴본 뒤 관련내용을 문서로 작성하세요. (5 pts)



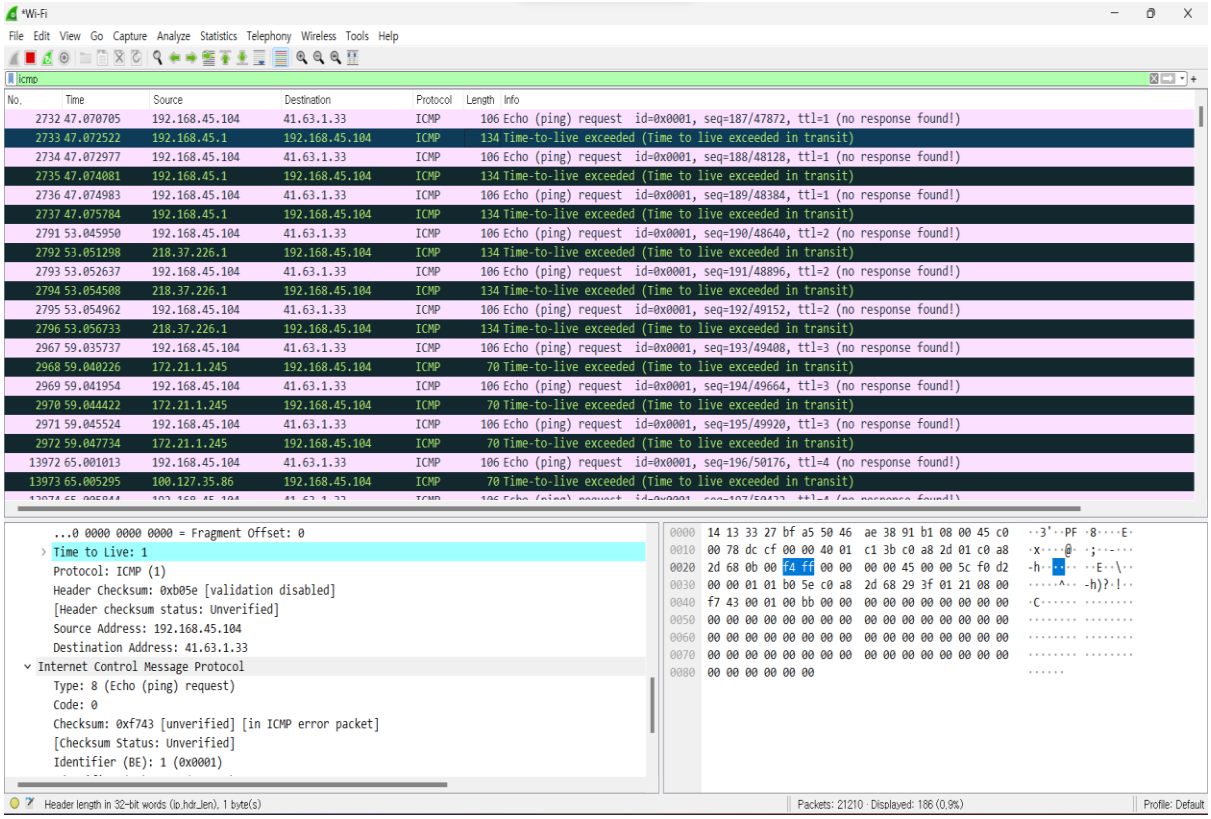
먼저 기본 설정인 최대 30홉으로는 trace route를 실패하였습니다. 따라서 최대 홉 40으로 다시 실행했습니다.





31홉 만에 trace route이 완료되는 것을 볼 수 있었습니다. 손실된 ICMP는 보이지 않았고, PTR 레코드가 존재하는 주소와 그렇지 않는 주소들이 있다는 것을 확인할 수 있습니다.

이제 wire shark로 더욱 자세히 보겠습니다.



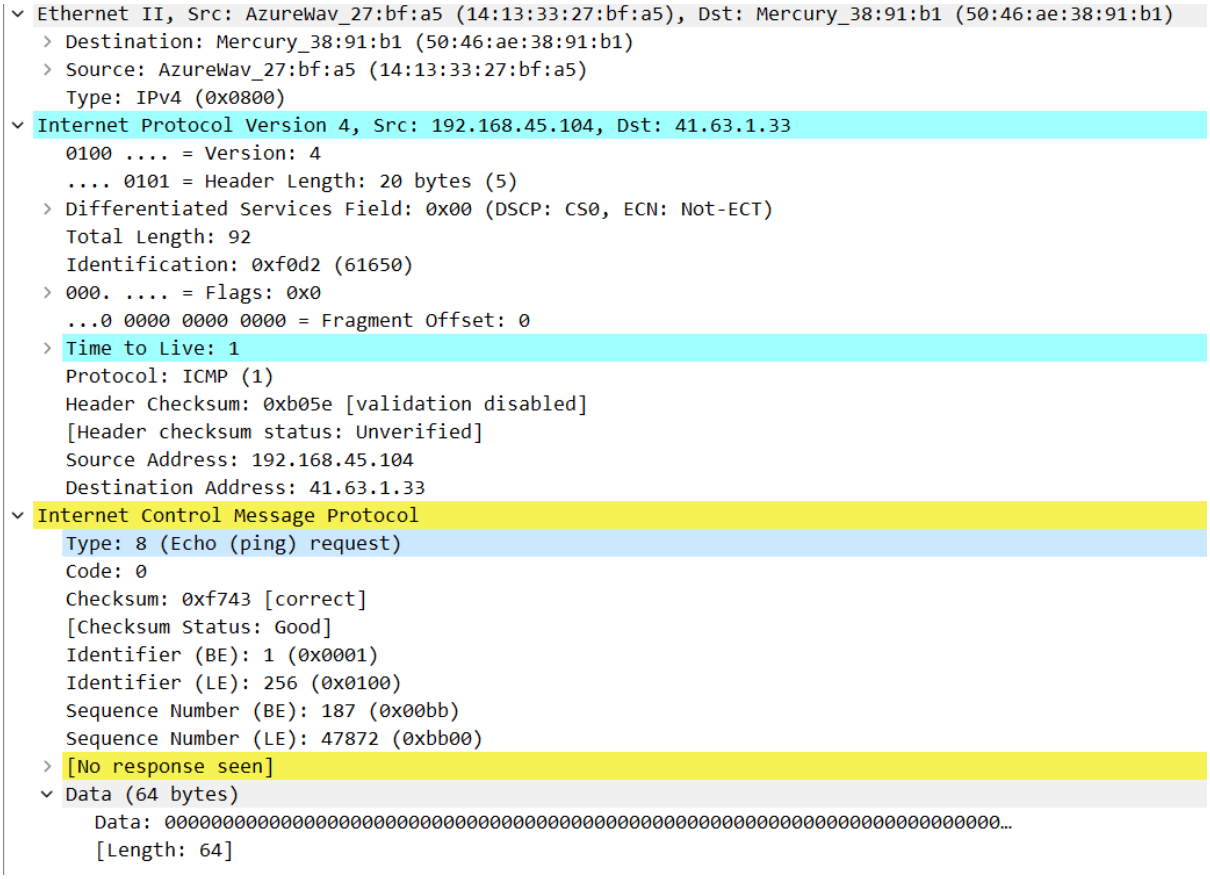
TTL을 설정한 ping을 날리고, Type: 11 (Time-to-live exceeded) 를 통해서 ICMP가 들어오는 모습을 볼 수 있습니다.

각 홉마다 3번의 핑을 날려주는 것을 확인할 수 있고, TTL을 1 -> 2 -> 3 -> 4.....로 늘려가며 패킷을 보내는 것 역시 확인 할 수 있었습니다.

재미있는 점은 패킷의 길이 입니다. 이전 문제에서는 74이던 길이가 던지는 핑에 대해서는 106, 받아본 핑에 대해서는 134, 70, 110, 182까지 다양했습니다.

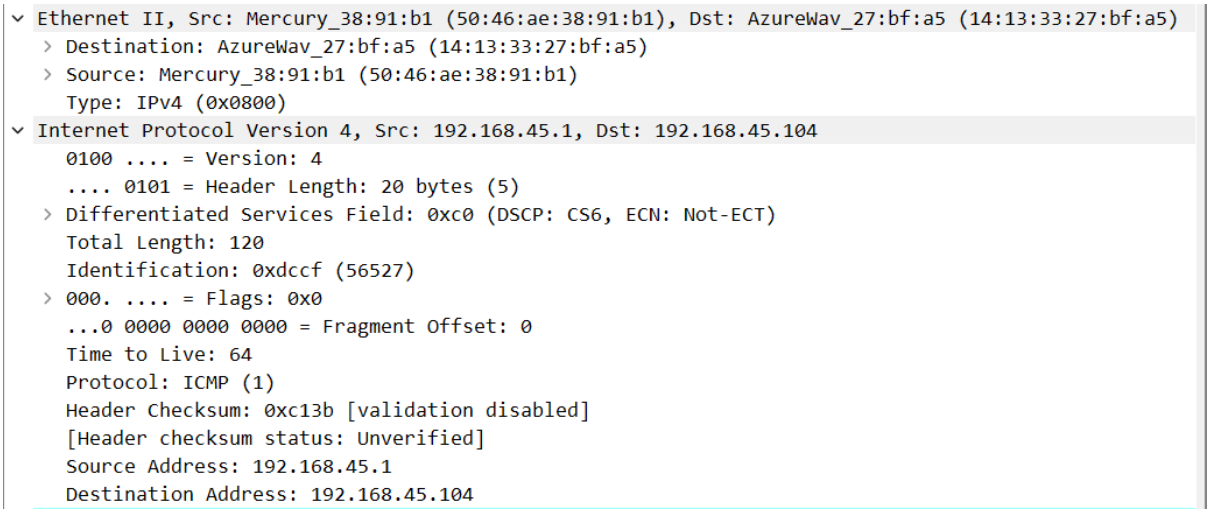
그렇다면 왜 이렇게 길이의 차이가 나는 걸까요?

먼저 던지는 핑에 대해서는 어째서 106의 길이를 가지는지 분석해보겠습니다.

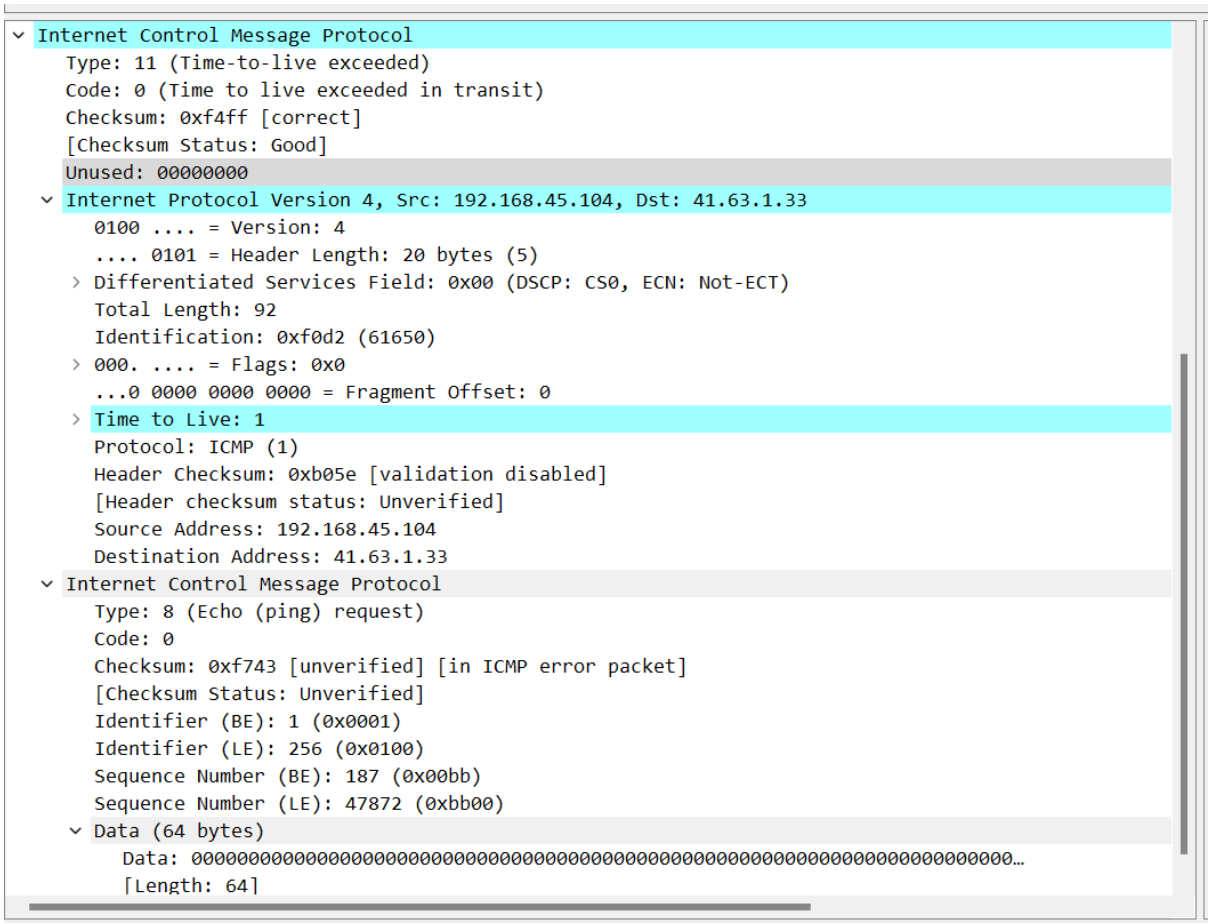


기본적으로 이더넷 헤더 14 + IP 헤더 20 + ICMP 헤더 8 + 데이터 64 = 106을 가짐을 확인할 수 있습니다. 해당 데이터 부분에는 trace route을 위한 핑의 쿼리에 대한 정보가 담겨져 있을거라고 생각할 수 있습니다.

그럼 두번째 134의 길이를 살펴보겠습니다.



윗 부분(이더넷 헤더 + IP헤더) 는 별로 다를게 없습니다. 저의 라우터에서 마지막으로 저에게 패킷을 보냈기에 source가 저의 라우터이고, Dest가 저의 개인 머신인 것을 확인 할 수 있습니다.



아래 부분은 다른 점이 많습니다. ICMP로 Type: 11 (Time-to-live exceeded)가 들어오는 것을 볼 수 있고 에러 메시지인 만큼

- Data section:
  1. **Error Messages** carries information to find the original packet that had the error (including IP header and first 8 bytes of IP paylord
  2. **Query Messages** carries extra information based on type of the query

해당 강의 슬라이드처럼 추가적으로 드랍된 패킷의 IP 헤더와 payload의 첫 8바이트를 포함합니다. 즉 저의 개인 주소 -> www.unza.zm 로 보내는 IP 헤더20바이트와, 그때 사용한 ICMP(ping request)의 헤더 8바이트가 추가적으로 붙습니다! 따라서 106 + 28 = 134 바이트가 되는 것을 확인할 수 있었습니다!

다른 패킷들도 전부 살펴본 결과 결론적으로 에러 메시지인 ICMP들의 길이들이 다른 이유는

Data 부분의 길이들이 다르던가 + ICMP Multi-Part Extensions 이라는 옵션이 추가적으로 붙었던 가의 이유였습니다.

또한 신기한 점은 위 문제에서 ping에 대한 Data부분은 어떠한 값(아마도 핑에 대한 쿼리 메시지) 가 들어있긴 했는데, trace route 과정에서는

1. 아예 없음

2. 64바이트 들어가 있음(근데 전부 0)

3. 40바이트 들어가 있음(근데 전부 0)

4. 64바이트 들어가 있음(근데 전부 0) + ICMP Multi-Part Extensions 옵션 추가

라는 것입니다. 즉 아무 정보도 들어가 있지 않은 패딩 같이 느껴졌습니다.

다시한번 tracert를 했을 때도 동일한 부분에서 길이들이 달랐어서 특정한 홉에서만 저렇게 제각각으로 작동한다는 생각이 들었습니다. 검색을 해봐도 관련 정보가 나오지 않아서 정확한 이유는 알아내는 데에 실패했지만,

네트워크는 그 홉마다 다른 정보를 가지고 있고, 방화벽, MTU 등등 신경 써야할 부분이 매우 많기에 특정 설정이 되는 것에 따라서 data의 구조가 바뀌는 것이 아닌가 하는 생각이 들었습니다.

4. 느낀점

과제를 하면서 어려웠던 점은 trace route을 하면서 패킷의 구조에 대한 분석이였습니다. 결국 정확한 이유를 알아내지 못하여서 추가적인 세밀한 분석이 필요해 보였습니다. 하지만 그만큼 재밌다는 생각이 들었고 특히 강의에서 들었던 부분들을 분석에 활용해보는 것이 매우 즐거운 경험이였습니다.