알 수 없는 패킷 프로토콜 분석 기법 제안

https://youtu.be/ZECLmvY1tLs

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

패킷 분석

- 하나 이상의 패킷 내용이나 메타 데이터의 조사
- 주요 패킷의 식별, 흐름 분석, 재현

- 분석도구
  - 무료 오픈소스 : 와이어 샤크

패킷 분석 기법

- 패턴 매칭
  캡처한 패킷에서 특정 값이 일치하는 것을 검색하여 주요 패킷 식별

- 프로토콜 필드 파싱
  주요 패킷의 프로토콜 필드의 내용을 추출

- 패킷 필터링
  프로토콜 메타 데이터의 필드 값을 기준으로 패킷을 구분

흐름 분석

- 관련 패킷들의 순서 조사

- 흐름 분석 기법

  상호 통신과 흐름 목록

  - 캡처된 패킷에서 모든 통신 흐름 또는 특정 흐름 목록화

  흐름 추출

  - 다량의 흐름을 분리하거나 의심스러운 흐름의 추가 분석을 위해 디스크에 저장

  파일과 데이터 카빙

  - 흐름을 재구성하여 파일이나 기타 흥미로운 데이터 추출(카빙)

목표

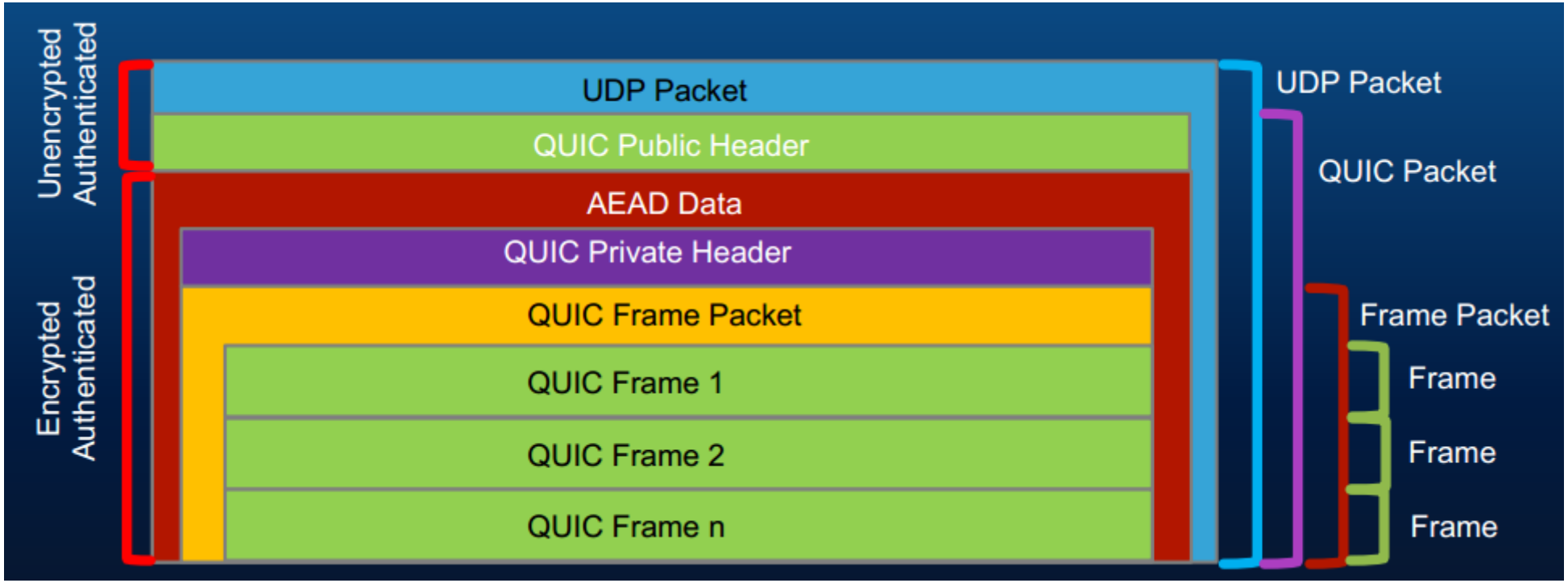
- 계층적 군집 분석과 순차 패턴으로 알 수 없는 패킷 분석
- 추후에 흐름분석까지 하여 프로토콜 분석
- 제안 기법으로 QUIC 패킷 분석

QUIC

- Google의 Jim Roskind가 처음에 설계 한 범용 전송 네트워크 프로토콜. 2012 년에 구현 및 배포
- QUIC는 TCP 연결과 거의 동일하지만 대기 시간이 크게 단축되는 것을 목표
- QUIC는 손실 복구를 포함하지 않는 UDP를 기본으로 사용

QUIC

페이로드가 암호화되어 보호됨



QUIC header

Long 헤더

- 1-RTT 키를 설정하기 전에 전송되는 패킷에 사용
- Initial, handshake, retry

Short 헤더

- 1-RTT 키가 협상 된 후에 사용

QUIC header

Long 헤더

1-RTT 키를 설정하기 전에 전송되는 패킷에 사용

Initial, handshake, retry

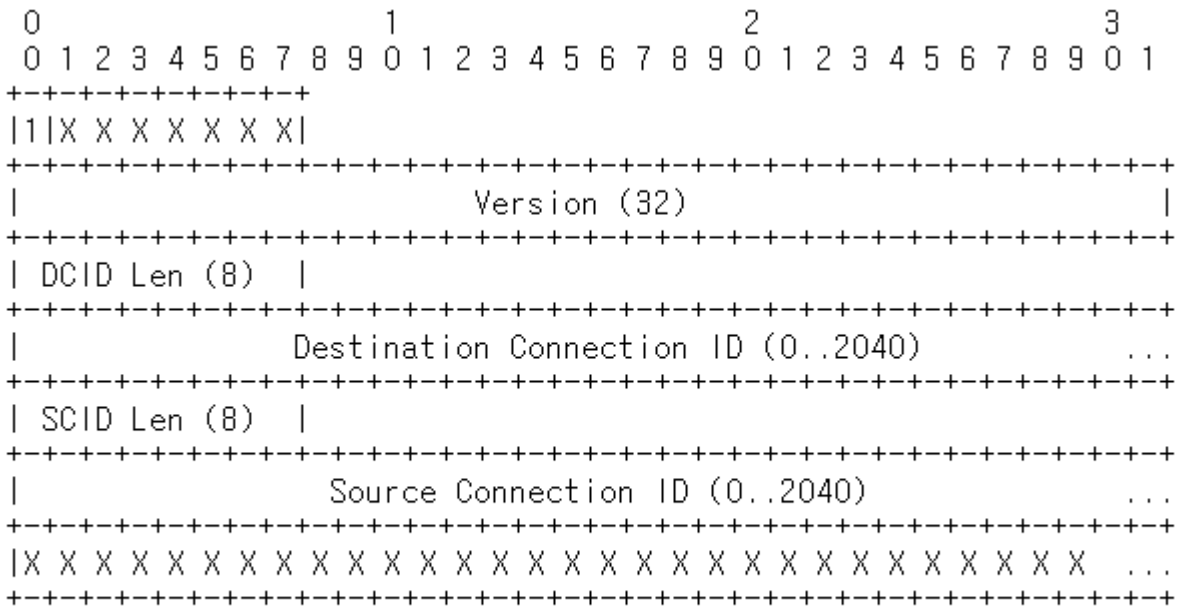


Figure 1: QUIC Long Header

Quic/46

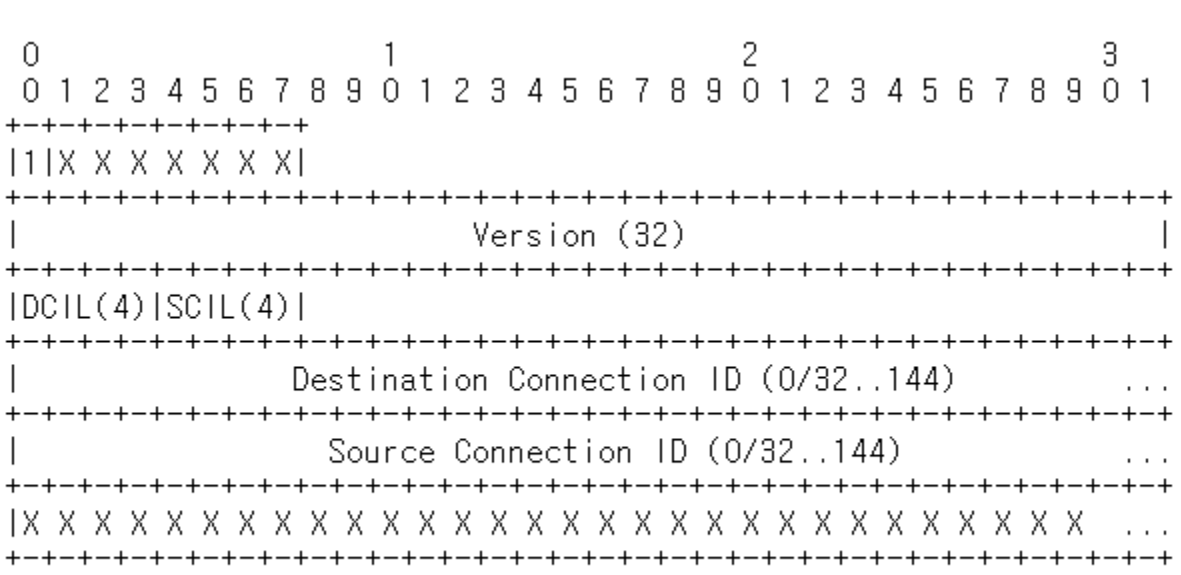


Figure 1: QUIC Long Header

Version-Independent Properties of QUIC draft-ietf-quic-invariants-04

QUIC header(ClientHello)

왕복 handshake – 클라이언트가 client hello 보냄

```
31 3.438349  192.168.219.103        216.58.200.4                    UDP          1392 54529 → 443 Len=1350
```

Connection ID : ef2b0d6428ac112a

IP

UDP,

version

ClientHello

```
0000  18 c5 01 b7 66 a9 d0 50  99 a6 10 bc 08 00 45 00   ····f··P ····· E·
0010  05 62 f7 9e 40 00 80 11  00 00 c0 a8 db 67 d8 3a   ·b··@··· ····· g·:
0020  c8 04 d5 01 01 bb 05 4e  41 af c3 51 30 34 36 50   ·· ····· N A··Q046P
0030  ef 2b 0d 64 28 ac 11 2a  00 00 00 01 fb be fa 7c   ·+·d(··* ····· ···|
0040  32 fa d9 f2 98 fe 9e 5e  a0 01 04 00 43 48 4c 4f   2····· ^ ···· CHLO
0050  11 00 00 00 50 41 44 00  d5 02 00 00 53 4e 49 00   ···· PAD· ···· SNI·
0060  e3 02 00 00 56 45 52 00  e7 02 00 00 43 43 53 00   ···· VER· ···· CCS·
0070  f7 02 00 00 55 41 49 44  2c 03 00 00 54 43 49 44   ···· UAID ,··· TCID
0080  30 03 00 00 50 44 4d 44  34 03 00 00 53 4d 48 4c   0··· PDMD 4··· SMHL
0090  38 03 00 00 49 43 53 4c  3c 03 00 00 4e 4f 4e 50   8··· ICSL <··· NONP
00a0  5c 03 00 00 4d 49 44 53  60 03 00 00 53 43 4c 53   \··· MIDS `··· SCLS
00b0  64 03 00 00 43 53 43 54  64 03 00 00 43 4f 50 54   d··· CSCT d··· COPT
00c0  64 03 00 00 49 52 54 54  68 03 00 00 43 46 43 57   d··· IRTT h··· CFCW
00d0  6c 03 00 00 53 46 43 57  70 03 00 00 2d 2d 2d 2d   l··· SFCW p··· ----

03a0  2d 2d 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 2d 2d 2d 2d   -------- --------
03b0  2d 77 77 77 2e 67 6f 6f  67 6c 65 2e 63 6f 6d 51   -www.goo gle.comQ
03c0  30 34 36 01 e8 81 60 92  92 1a e8 7e ed 80 86 a2   046··· `· ···~ ····
```

QUIC header(rejection)

왕복 handshake – 서버에서 소스 주소 토큰과 서버 인증서를 포함하여 클라이언트가 진행하는데 필요한 정보 보냄

45 3.510313 216.58.200.4 192.168.219.103 UDP 1392 443 → 54529 Len=1350

Connection ID : ef2b0d6428ac112a

```
0020  db 67 01 bb d5 01 05 4e  e5 2e c3 51 30 34 36 05   ·g·····N ·.·Q046·
0030  ef 2b 0d 64 28 ac 11 2a  00 00 00 01 d6 6d d9 af   ·+·d(··* ·····m··
0040  79 bf 65 61 73 05 d5 fa  40 01 1c 02 01 00 80 01   y·eas··· @·······
0050  52 45 4a 00 07 00 00 00  53 54 4b 00 38 00 00 00   REJ····· STK·8···
0060  53 4e 4f 00 6c 00 00 00  50 52 4f 46 b4 00 00 00   SNO·l··· PROF····
0070  53 43 46 47 47 01 00 00  52 52 45 4a 4b 01 00 00   SCFGG··· RREJK···
0080  53 54 54 4c 53 01 00 00  43 52 54 ff e8 09 00 00   STTLS··· CRT·····
```

QUIC header

Short 헤더

1-RTT 키가 협상 된 후에 사용

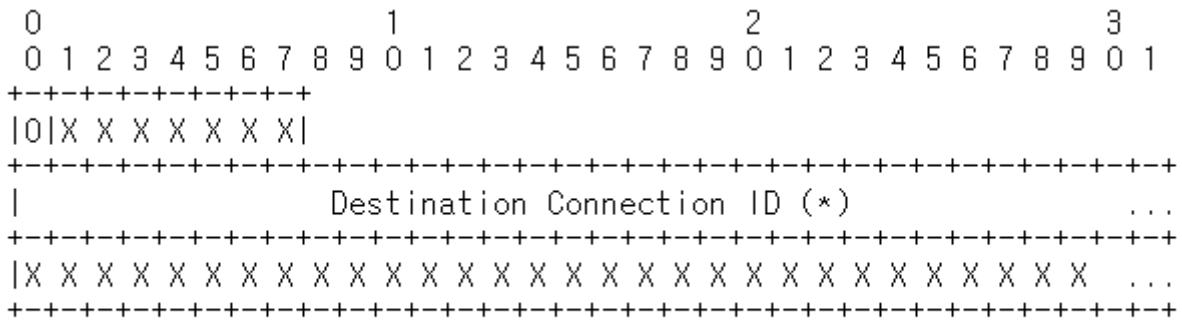


Figure 2: QUIC Short Header

```
0020  c8 04 d5 01 01 bb 00 24  3c 85 40 ef 2b 0d 64 28   ······$ <·@·+·d(
0030  ac 11 2a 0b 6f 6c eb fa  b6 c7 65 87 60 41 29 52   ··*·ol·· ··e·`A)R
0040  64 a3 53 8e 5d 9e                                 d·S·]·

0020  c8 04 d5 01 01 bb 00 24  3c 85 40 ef 2b 0d 64 28   ·····$ <·@·+·d(
0030  ac 11 2a 0c aa 61 cf 31  86 ba 43 04 c3 7e 73 db   ··*··a·1 ··C··~s·
0040  9d 97 6e c5 75 b3                                 ··n·u·

0020  c8 04 d5 01 01 bb 00 24  3c 85 40 ef 2b 0d 64 28   ·····$ <·@·+·d(
0030  ac 11 2a 0d 2d 26 f6 c9  0b 5d be d9 c1 ae 26 cc   ··*·-&·· ·]····&·
0040  a7 42 4a 56 a6 23                                 ·BJV·#
```

Google QUIC 46 version Header Format

Header Form(HF) : long header 일 경우 1, short header 일 경우 0

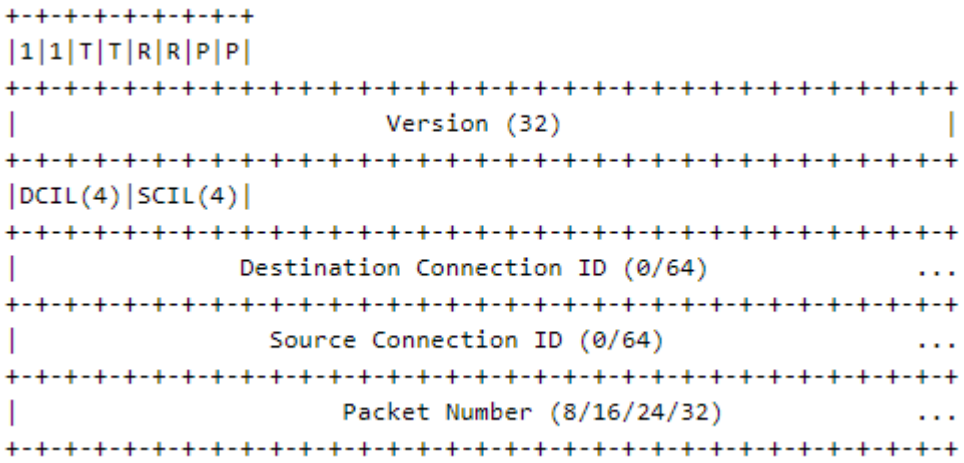
Fixed Bit (FB) : 1로 고정된 비트 0 값을 포함하는 패킷은 이 버전에서 유효한 패킷이 아님

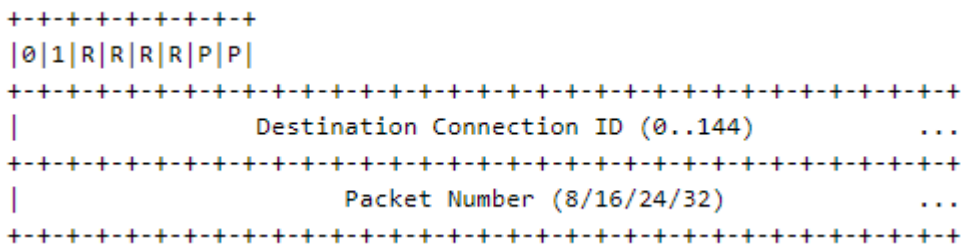
Long Packet Type(T) :

Initial Packet(0), 0RTT Packet[1], Handshake Packet[2], Retry Packet[3](이 번에는 아직 사용 안함)

Reserved Bits (R) : 미래에 사용하기 위해 남겨둔 예비 필드 0으로 채워둠

Packet Number Length(P)





https://docs.google.com/document/d/1FcpCJGTDEMblAs-Bm5TYuqhHyUqeWpqrItw2vkMFsdY/edit#

https://tools.ietf.org/html/draft-mcquistin-quic-augmented-diagrams-00#section-6

Google QUIC 46 version Header Format

Header Form(HF) : long header 일 경우 1, short header 일 경우 0

Fixed Bit (FB) : 1로 고정된 비트 0 값을 포함하는 패킷은 이 버전에서 유효한 패킷이 아님

Long Packet Type(T) :

Initial Packet(0), 0RTT Packet[1], Handshake Packet[2], Retry Packet[3](이 번에는 아직 사용 안함)

Reserved Bits (R) : 미래에 사용하기 위해 남겨둔 예비 필드 0으로 채워둠

Packet Number Length(P)

Pacet Number

```
0000  d0 50 99 a6 10 bc 18 c5  01 b7 66 a9 08 00 45 00   ·P······ ··f···E·
0010  00 30 00 00 40 00 36 11  5a 4e ac d9 a1 84 c0 a8   ·0··@·6· ZN······
0020  db 68 01 bb c9 3f 00 1c  66 75 40 37 ed 34 06 92   ·h···?·· fu@7·4··
0030  a3 53 e3 48 3b 46 48 8d  7a 1d f4 52 36 f8         ·S·H;FH· z··R6·

0000  d0 50 99 a6 10 bc 18 c5  01 b7 66 a9 08 00 45 00   ·P······ ··f···E·
0010  00 48 00 00 40 00 36 11  5a 36 ac d9 a1 84 c0 a8   ·H··@·6· Z6······
0020  db 68 01 bb c9 3f 00 34  49 6d 41 00 38 78 a6 ff   ·h···?·4 ImA·8x··
0030  62 82 b6 a9 e7 92 52 36  ee 2e fb 41 20 96 d7 44   b·····R6 ···A ··D
0040  6b 51 ac d9 93 4a df ae  fd af 18 ac 98 e2 e8 17   kQ···J·· ········
0050  7a 92 7e e2 90 05                                 z·~···
```

Google QUIC 46 version Header Format

Packet Header Type	From	To	Connection ID
Long header	Client	Server	8-byte Destination Connection ID 0-byte Source Connection ID
Long header	Server	Client	0-byte Destination Connection ID 8-byte Source Connection ID
Short header	Client	Server	8-byte Destination Connection ID
Short header	Server	Client	0-byte Destination Connection ID

알 수 없는 프로토콜 탐지 흐름

알고있는 프로토콜 제거

패킷

알고있는 프로토콜 제거

유사한 패킷으로 분류

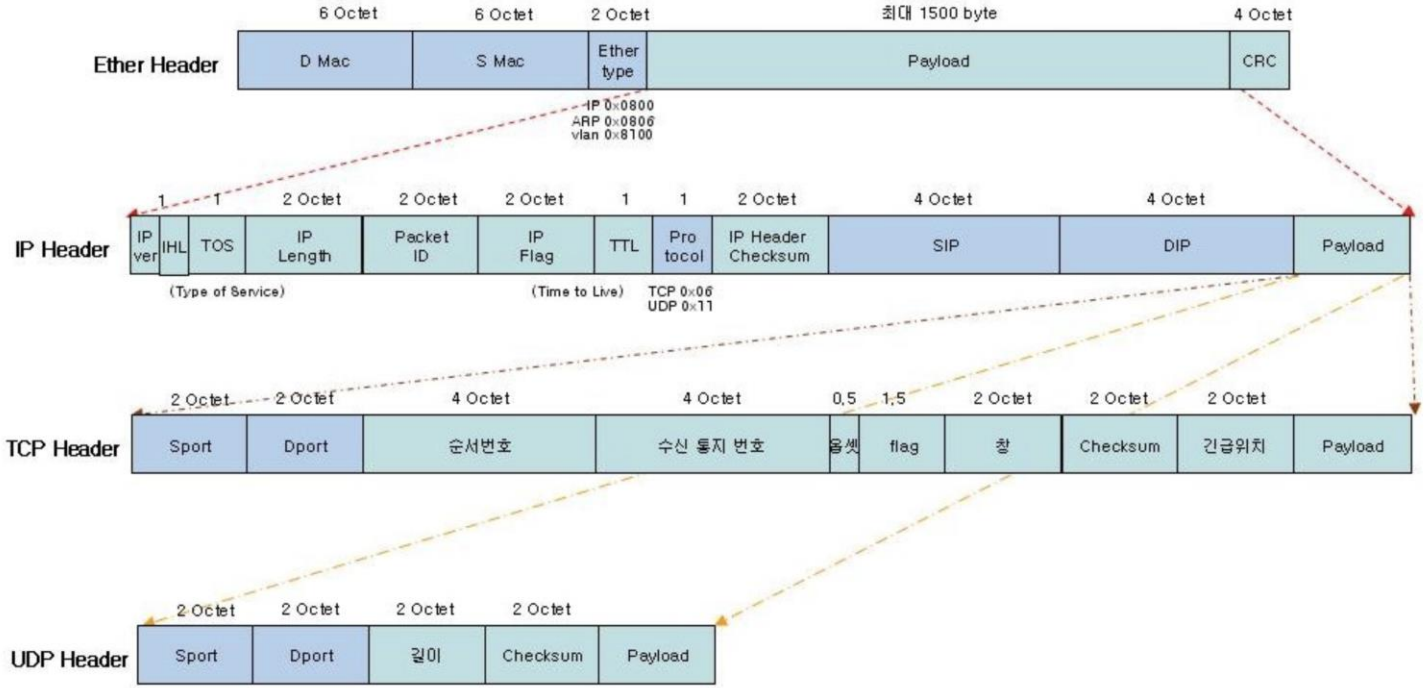
패킷간 공통부분 탐색

나머지 부분 탐색

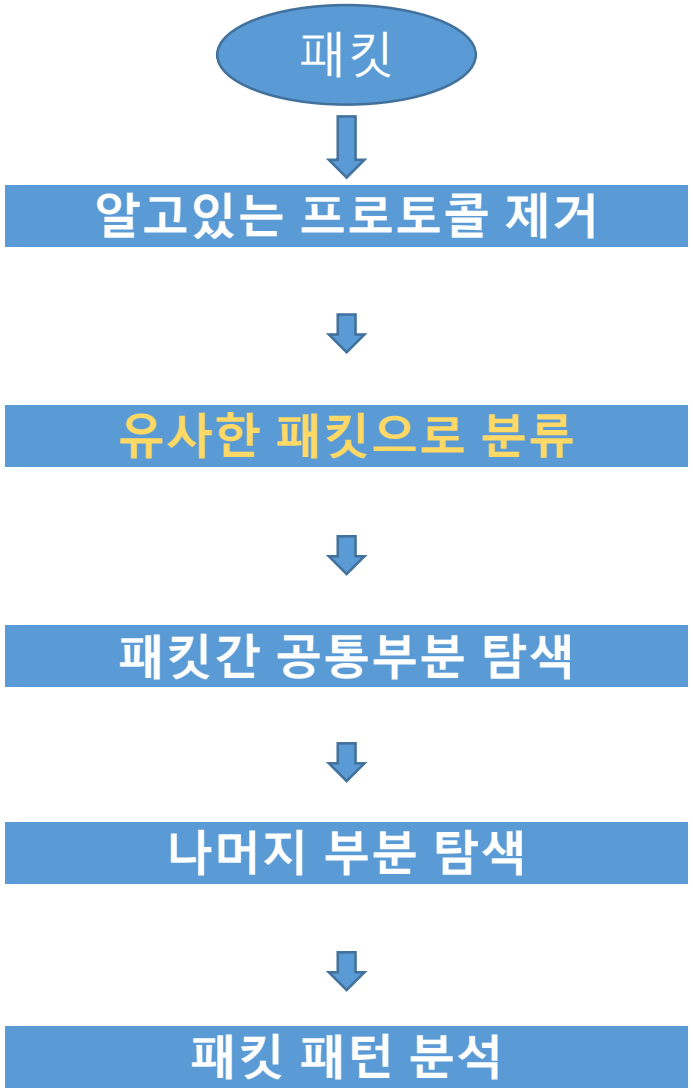
패킷 패턴 분석

➢ 알 수 없는 프로토콜 찾기
-> 상위 계층 제거 payload 부분 추출

Packet Analysis
패킷의 계층적 구조



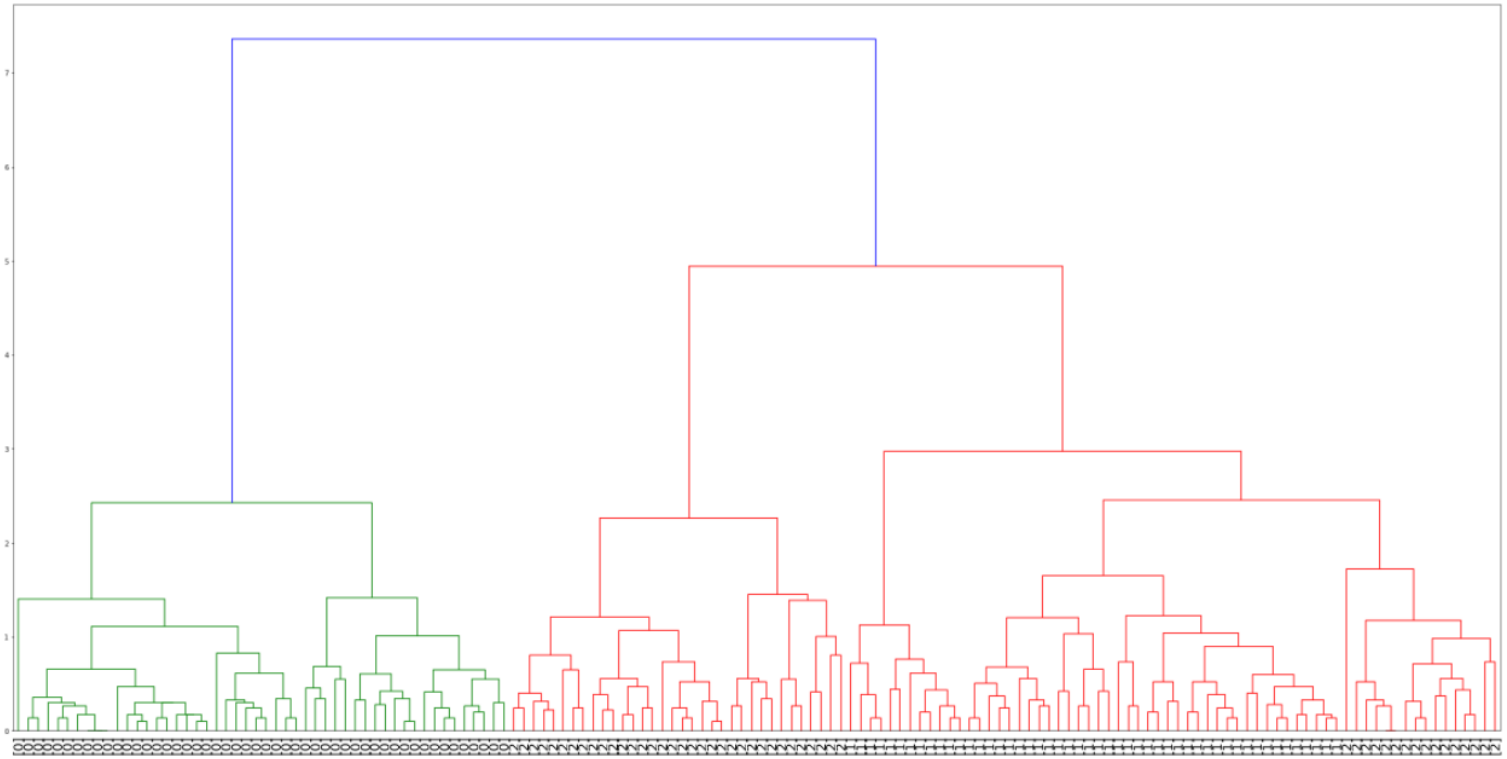
유사한 패킷으로 분류



- 프로토콜이 여러 개일 경우 분류
- 계층적 클러스터링 기법 사용

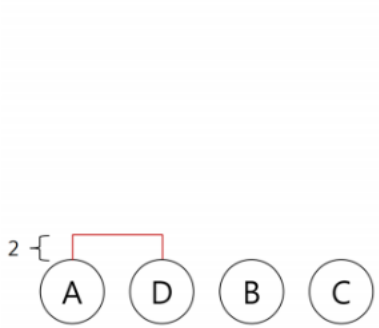
계층적 군집 분석(Hierarchical clustering)

- 비슷한 군집끼리 묶어 가면서 최종적으로는 하나의 케이스가 될 때까지 군집을 묶는 클러스터링 알고리즘
- 군집간의 거리를 기반으로 클러스터링을 하는 알고리즘이며, K Means와는 다르게 군집의 수를 미리 정해주지 않아도 됨

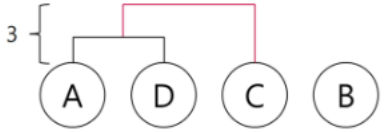


계층적 군집 분석(Hierarchical clustering)

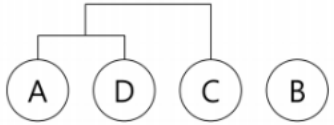
	A	B	C	D
A		20	7	2
B			10	25
C				3
D				



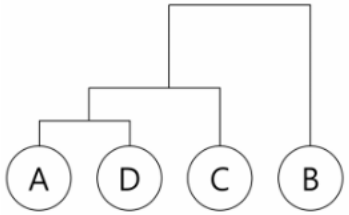
	A	B	C	D
A		20	7	2
B			10	25
C				3
D				



	AD	B	C	
AD		20	3	
B			10	
C				



	ADC	B		
ADC		10		
B				



	AD CB			
AD CB				

계층적 군집 분석(Hierarchical clustering)

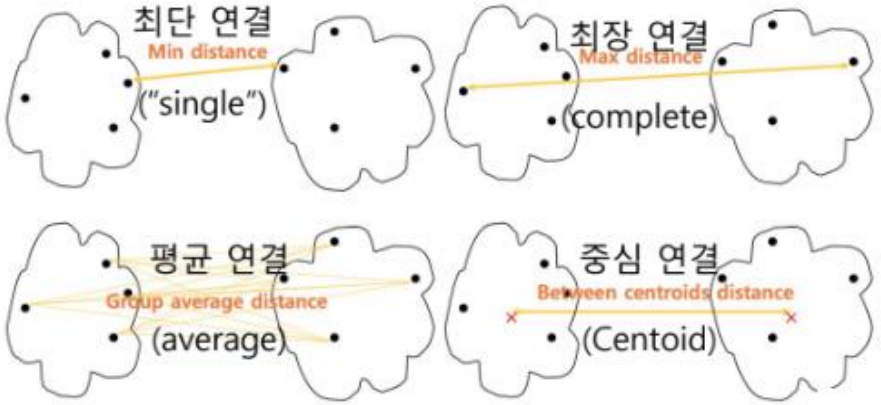
•single (최단연결) : 가장 가까운 개체끼리 연결

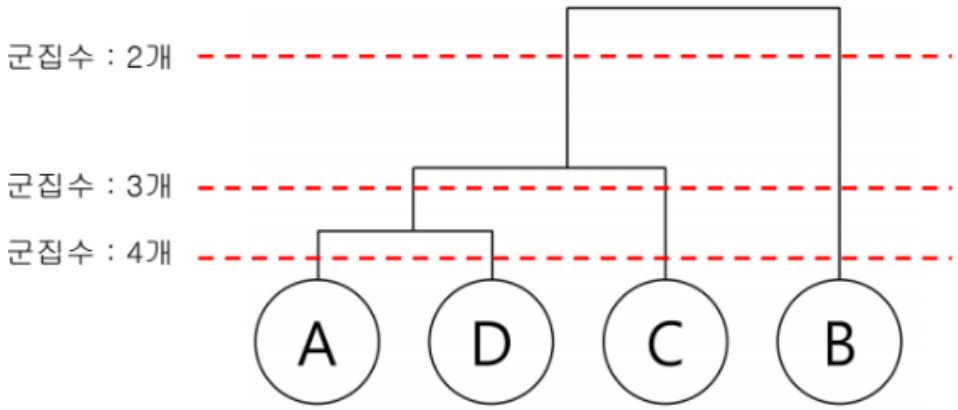
•complete (최장연결): 가장 먼 개체끼리 연결 (보수적으로)

•average (평균연결) : 모든 점들을 모두 연결하여 평균 계산함.
속도가 느려질 수 있지만 이상치에 덜 민감할 수 있다.

•centroid (중심연결): 군집의 중심을 잡아 거리를 계산.
평균보다 계산 양이 적어져 더 빠를 수 있음.

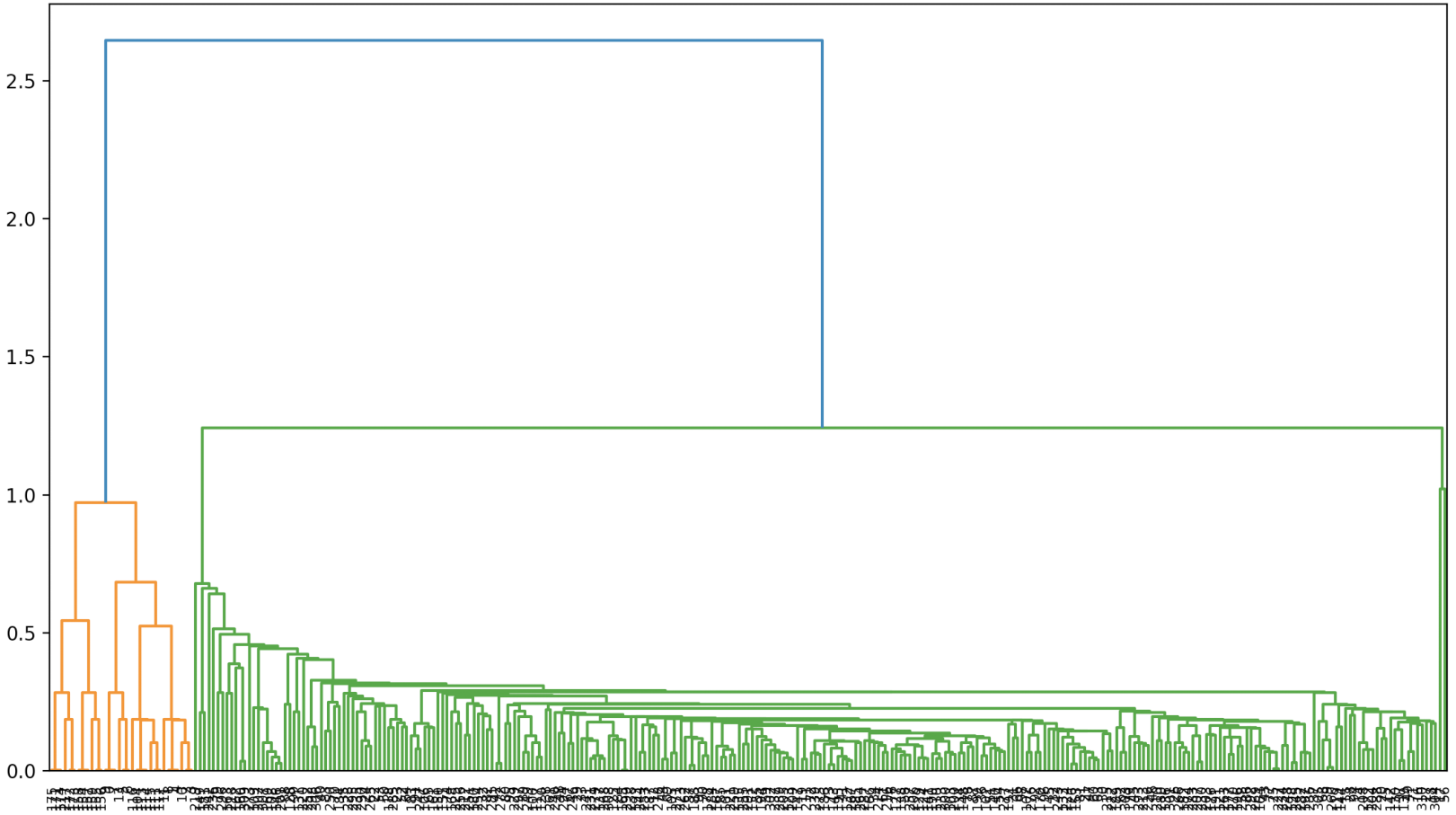
하나의 군집이 형성되면 그 군집에 속한 개체들은 항상 같은 군집에 속함 -> 수정하지 못함



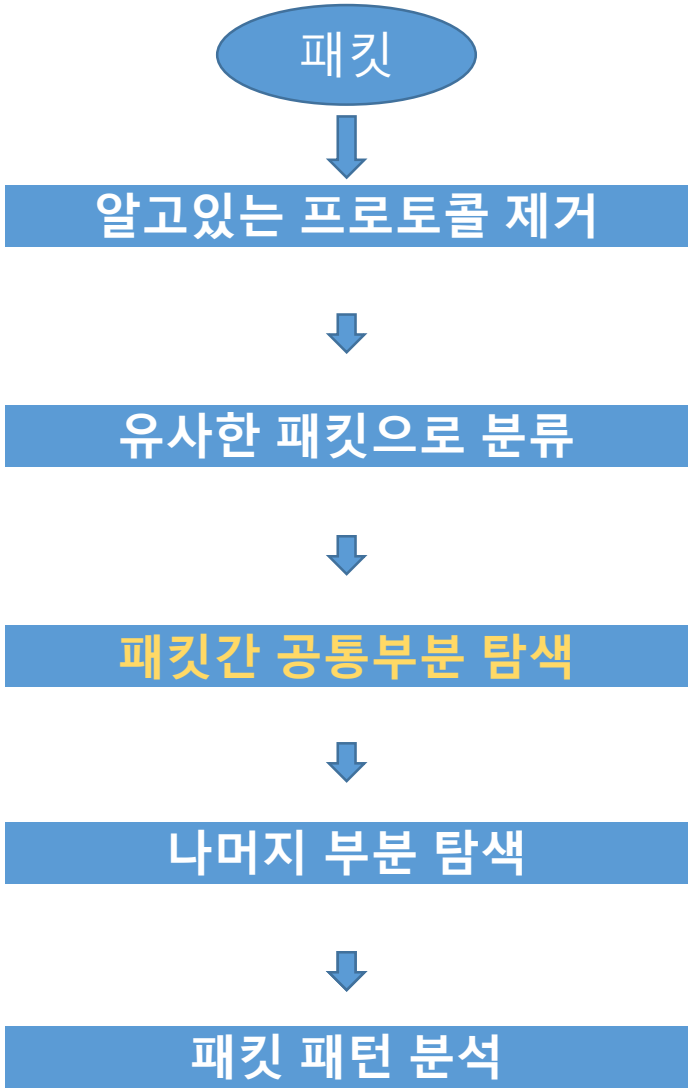


계층적 클러스터링 적용

311개의 QUIC 패킷 계층적 클러스터링 적용
32개의 Long 헤더와 279개의 Short 헤더 구분



패킷간 공통부분 탐색



- 패킷 구조 분석 위해서 유사한 패킷의 공통부분을 탐색
- 순차패턴분석 (Sequence Pattern Analysis) 사용

순차적 패턴 분석

- 순차 패턴 마이닝(sequential pattern mining)은 대량의 데이터에 숨겨진 "순차적 패턴"을 찾는 분석방법
- 연속하여 일어나는 패턴을 찾는데 유용한 방법으로, 커머스 분야에서 고객이 어떤 순서로 제품을 구매하는지 분석 활용

고객 ID	시퀀스
1	({맥주, 주스}, {기저귀, 맥주}, {우유, 과자})
2	({맥주, 땅콩}, {기저귀}, {우유, 과자}, {사과})
3	({기저귀, 맥주}, {우유}, {맥주, 주스}, {과자})
4	({우유, 주스}, {맥주, 땅콩}, {맥주, 기저귀})

순차적 패턴 분석

- 순차 패턴 마이닝에서는 패턴의 빈번함을 정의하기 위해 지지도(support)라는 척도를 사용하는데, 지지도란 전체 시퀀스 중 특정 패턴을 포함하는 비율입니다.
-
- 사용자가 지지도 값을 정하면, 이 값도 큰 지지도를 가진 패턴을 순차 패턴 알고리즘이 찾습니다. 이렇게 큰 지지도 값을 가진 패턴을 빈번한 패턴(frequent pattern)이라고 합니다.

고객 ID	시퀀스
1	({맥주, 주스}, {기저귀, 맥주}, {우유, 과자})
2	({맥주, 땅콩}, {기저귀}, {우유, 과자}, {사과})
3	({기저귀, 맥주}, {우유}, {맥주, 주스}, {과자})
4	({우유, 주스}, {맥주, 땅콩}, {맥주, 기저귀})

지지도	마이닝 결과
50 %	(맥주, 기저귀, 우유)
75 %	(기저귀, 우유)

순차 패턴 적용

```
#순차패턴분석
ps = PrefixSpan(data)
ps.minlen=2        #배열 크기 개 이상
b= ps.topk(50)      #상위 개
print(b)

#순차패턴의 모든결과 1차원 배열에 넣기
s=[]
for i in range(len(b)):
    for j in range(len(b[i][1])):
        s.append(b[i][1][j])

#같은값 제거
list = []
for v in s:
    if v not in list:
        list.append(v)
print(list)

#전 단계의 리스트의 값을 패킷에서 탐색 후 해당 값이 가장 많이나온 인덱스를 저장
point=[]
index=[[] for i in range(len(list))]
for j in range(len(list)):
    for i in range(len(data)):
        for p in enumerate(data[i]):
            if(p[1]==list[j]):
                index[j].append(p[0])
    point.append(Counter(index[j]).most_common(1)[0][0])

#크기 순으로 정렬
point=sorted(point)
print(point)
```

```
[(32, [81, 48]), (32, [81, 48, 52]), (32, [81, 48, 52, 0]), (32,
[81, 48, 52, 0, 2, 54, 80, 195]
[0, 1, 2, 3, 4, 5, 14, 17]
```

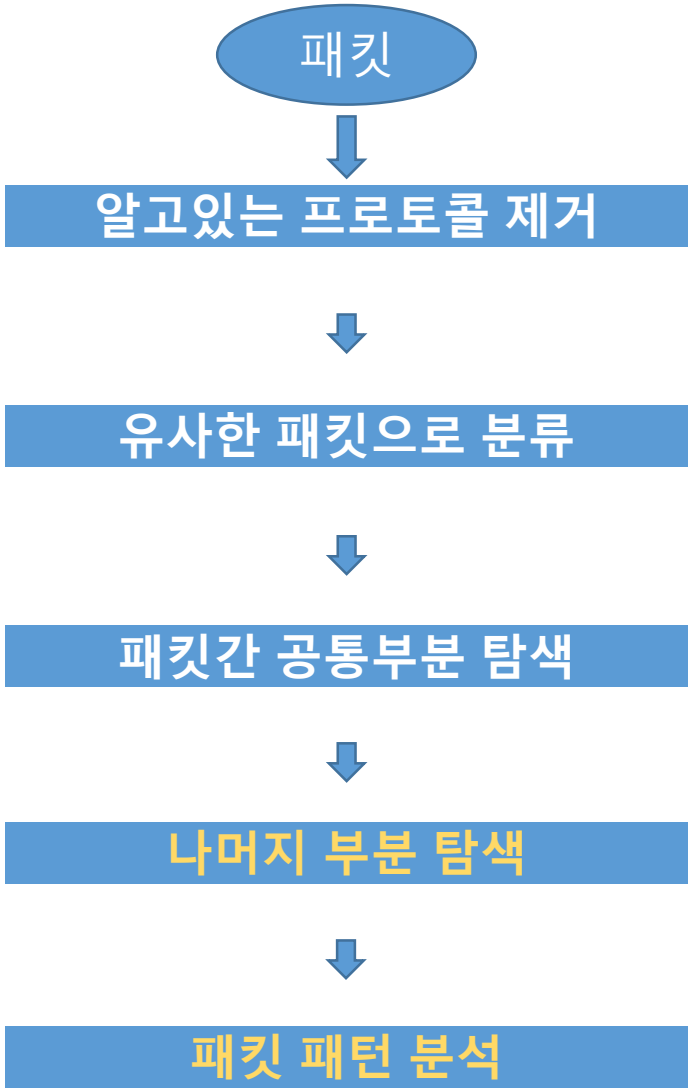
```
[(216, [64, 8]), (216, [64, 80]), (211, [64, 97]), (208, [64, 86
[64, 8, 80, 97, 86, 125, 126, 128, 144]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

순차 패턴 적용

```
[[195.  81.  48.  52.  54.  80.   0.   1.]
 [195.  81.  48.  52.  54.  80.   0.   1.]
 [195.  81.  48.  52.  54.   5.   0.   1.]
 [195.  81.  48.  52.  54.   5.   0.   2.]
 [195.  81.  48.  52.  54.  80.   0.   2.]
 [195.  81.  48.  52.  54.   5.   0.   1.]
 [195.  81.  48.  52.  54.   5.   0.   2.]
 [195.  81.  48.  52.  54.  80.   0.   2.]
 [195.  81.  48.  52.  54.  80.   0.   3.]
 [195.  81.  48.  52.  54.  80.   0.   3.]
 [211.  81.  48.  52.  54.  80.   0.   4.]
 [211.  81.  48.  52.  54.   5.   0.   3.]
```

```
[[ 64.   8.  97. ... 125.  86.  80.]
 [ 64. 230. 139. ...   8. 202. 107.]
 [ 64. 230. 139. ...   8. 202. 107.]
 ...
 [ 64.   8.  97. ... 125.  86.  80.]
 [ 64. 223.  69. ...  68.  68. 118.]
 [ 65.   0. 175. ...  99. 161. 235.]]
```

추후 작업



- 패킷 순서, 암호화 부분 등
- 프로토콜의 흐름 분석

Q & A