

**목차**

1. **개요  
   1.1 게임 방법  
   1.2 게임 구동**
2. **패킷 전송 및 수신 과정 분석  
   2.1 로그인 화면  
   2.2 대기 화면  
   2.3 게임 화면  
   2.4 채팅 화면  
   2.5 투표 화면  
   2.6 승/패 결과 화면과 종료**

1. **개요**

**1.1 게임 방법**

* 클라이언트 5명이 마피아 게임을 진행한다.
* 클라이언트에게 랜덤으로 직업(마피아 1명, 시민 4명)이 배정되고, 마피아가 죽거나 시민과 마피아가 한 명씩 남으면 게임이 끝난다.
* 서버 역할을 하는 컴퓨터 1대와 클라이언트 역할을 하는 5대의 컴퓨터로 게임을 수행하였다.
* 온라인 마피아 게임 프로그램을 통해 서버와 클라이언트 사이의 패킷을 캡쳐하기 위해 Wireshark 프로그램을 사용하였다.
* 팀원 모두 Django를 학습하여 MVT모델을 사용하여 구현하였다.
* 서버 컴퓨터는 8000번 포트로 서버를 열었고, 모든 클라이언트에 대해 외부 접속을 허용하기위해 setting.py 파일의 ALLOWED\_HOSTS 부분을 수정하였다.
* 보고서에 사용한 서버와 클라이언트 IP/MAC 주소
  + **서버 ip= 192.168.43.44, mac= 80:32:53:b3:1d:7a**
  + **클라이언트 ip = 192.168.43.95, mac= ae:cc:fe:6e:40:49**

**1.2 게임 구동**

검은색, 모니터, 앉아있는, 화면이(가) 표시된 사진

자동 생성된 설명

*[그림 1-1] 클라이언트의 게임 시작 화면*

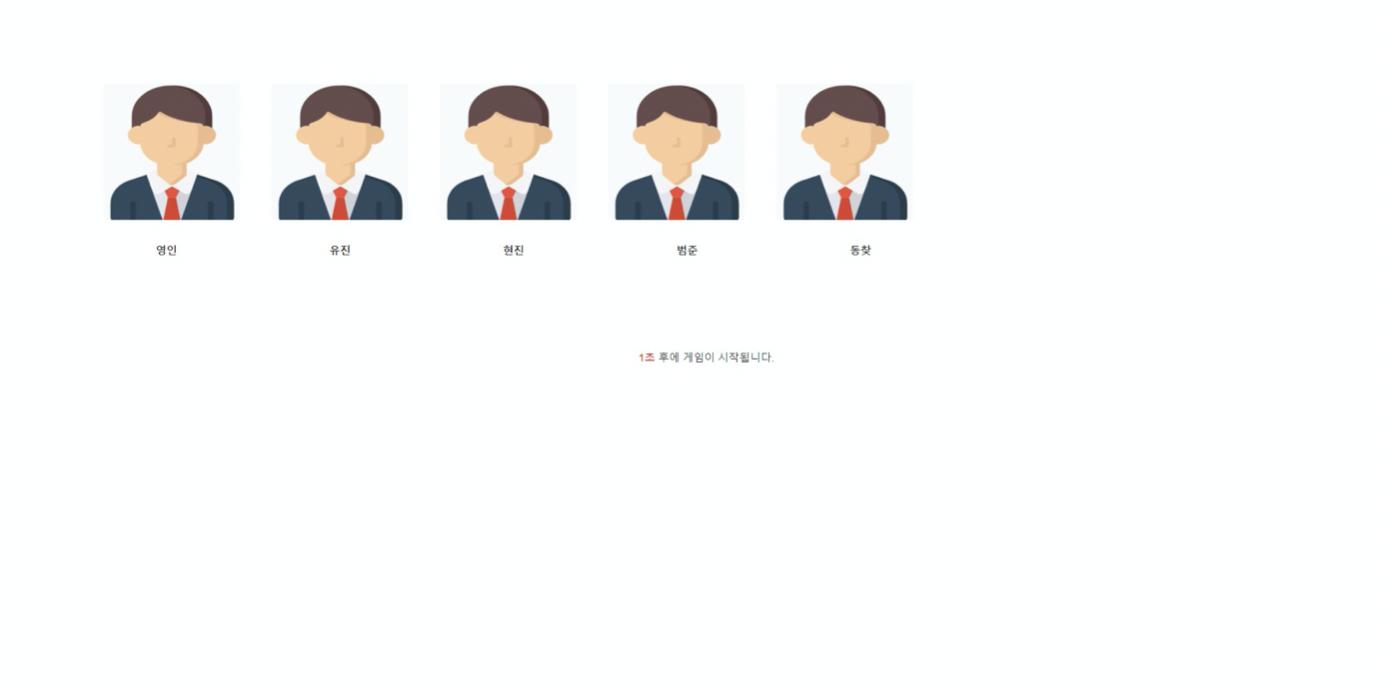
* 기본적으로 게임 프로그램을 실행하면 [그림1-1]과 같은 로그인 화면이 뜨며, 이름을 작성하고 [게임참가] 버튼을 누르면 게임을 시작할 수 있다.

*텍스트, 창문이(가) 표시된 사진

자동 생성된 설명*

*[그림 1-2] 서버 컴퓨터의 cmd화면*

* 서버 컴퓨터에 클라이언트 한 명이 게임 url에 접속할 때 마다 cmd화면에 클라이언트 번호가 순차적으로 생기며, 해당 방에 접속했다는 표시를 한다.



*[그림 1-3] 게임 준비 waiting room 화면*

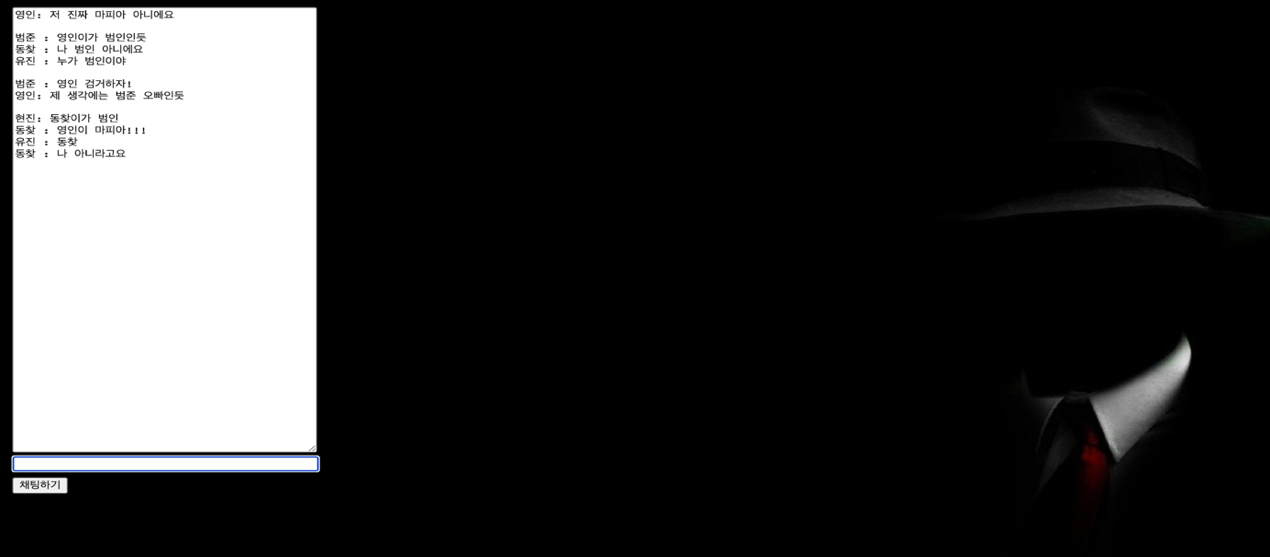
* 게임 waiting room에 참가자가 5명이 들어오면 3초 후에 게임이 시작된다*.*

모니터, 앉아있는, 노트북, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

*[그림 1-4] 게임 화면*

* 채팅 시작 버튼을 눌러 채팅 화면으로 넘어 갈 수 있게 한다.



*[그림 1-5] 채팅 화면*

* 하고 싶은 말을 하고 채팅하기 버튼을 넣으면 메시지가 전송되고 다른 참가자의 메시지도 확인할 수 있다.
* 3분 동안의 시간 제한이 있고 3분 지나면 투표 페이지로 넘어간다.

모니터, 앉아있는, 노트북, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

*[그림 1-4] 게임 화면*

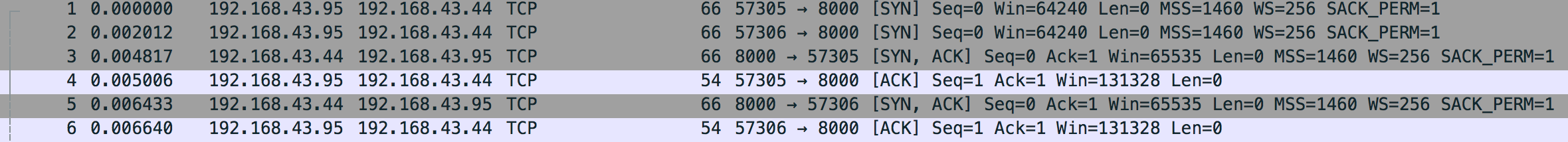
* 이전 [그림 1-4] 게임 화면으로 돌아와서 예상되는 마피아를 투표한다.

**2. 패킷 전송 및 수신 과정**

**2.1 로그인 화면**

A. 서버와 클라이언트 연결

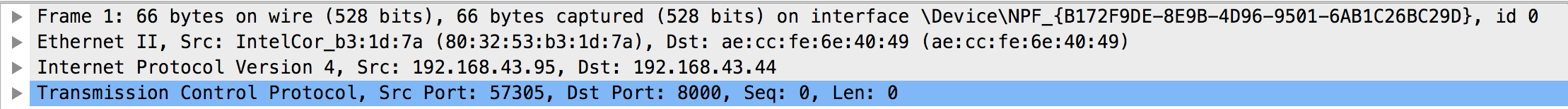
* 3-way handshaking을 통해 TCP연결



*[그림 2-1-1] Client (192.168.43.95)와 Server(192.168.43.44)가 3-way handshaking을 통해 연결*

**Frame No. 1 :**





*[그림 2-1-2] 패킷 번호 1의 패킷 내용 (first handshaking)*

* 클라이언트는 임의의 포트번호(57305)를 클라이언트에 할당하고, 포트번호(57305)를 포함하여 [SYN]을 서버에 전송한다.

**# frame 1** 프레임 번호는 1번이고, 사용한 인터페이스 id는 0번인 것을 알 수 있다.

**# ethernet2 frame header**

* + - source의 MAC 주소는 80:32:53:b3:1d:7a 이고, destination의 MAC 주소는 ae:cc:fe:6e:40:49 인 것을 알 수 있다.
* 이 부분에서 type는 IPv4를 사용하고, unicast로 통신이 이루어짐을 알 수 있다.)

**# Internet Protocol Version 4**

* + IPv4을 사용하고, src의 ip는 182.168.43.95이고, dst의ip는 192.168.43.44이다.
* ipv4이므로, checksum을 가지고 있고, header checksum을 확인할 수 있다.
* TTL은 128인 것을 확인할 수 있다.

**# Transmission Control Protocol (TCP, transport layer)**

* src port는 57306이고, dst port는 8000번인 것을 알 수 있다.
* 이 부분을 살펴보면 tcp의 checksum은 0xd802임을 알 수 있다.
* Sep 번호는 0임을 확인할 수 있다.

**Frame No. 3 :**





*[그림 2-1-3] 패킷 번호 3의 패킷 내용(second handshaking)*

* 서버는 클라이언트의 SYN을 받고, SYN received 상태가 된다. 클라이언트에게 연결을 허용한다는 의미의 [SYN, ACK] 패킷을 응답

**# Internet Protocol Version 4**

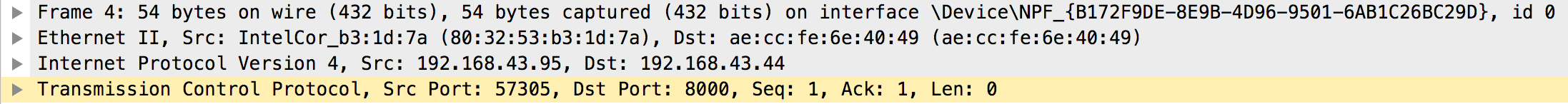
* IPv4을 사용하고, src의 ip는 서버(192.168.43.44)이고, 이고, dst의ip는 클라이언트(182.168.43.95)인 것을 확인할 수 있다.
* flags : 0x4000 으로 Don’t fragment 으로, 단편화(fragment)가 진행되어 있지 않음을 알 수 있다.

**# Transmission Control Protocol (TCP, transport layer)**

* src port는 서버(8000)이고, dst port는 클라이언트(57306)인 것을 알 수 있다.
* 이 부분을 살펴보면 tcp의 checksum은 0xdfc4임을 확인이 가능하다.
* Seq 번호가 0인 것으로 보아 No.2 패킷에 대한 응답인 것을 확인할 수 있다.
* Ack : 1 인 것을 보아 다음으로 받기를 기대하는 seq번호가 1인 것을 알 수 있다.

**Frame No. 4 :**





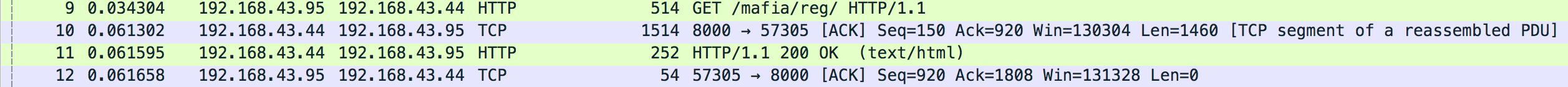
*[그림 2-1-4] 패킷 번호 4의 패킷 내용 (3th handshaking)*

* 클라이언트는 요청에 의한 서버의 응답을 확인했다는 의미로 [ACK]패킷을 서버에 전송
* 3-way handshaking이 완성되었다.

**# Transmission Control Protocol**

* TCP, transport layer
* src port는 클라이언트(57306)이고, dst port는 서버(8000)인 것을 알 수 있다.
* No.3에 대한 ack임을 알 수 있고, ack : 1을 전송하면서 마지막 TCP연결을 마친다.

B. 로그인 화면 로딩



*[그림 2-1-5] HTTP와 TCP를 사용해서 로그인 화면을 불러오는 패킷*

**Frame No. 9 :**

****

스크린샷이(가) 표시된 사진

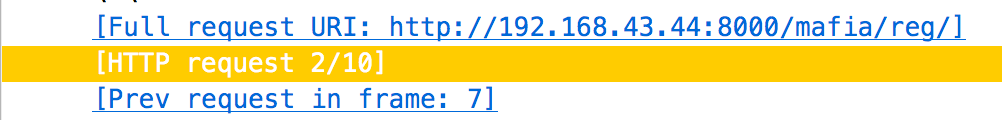
자동 생성된 설명

*[그림 2-1-6] 패킷 번호 9번의 패킷 상세보기 화면*

* 클라이언트와 서버의 연결이 끝난 후, 클라이언트는 서버에게 HTTP요청을 한다.
* 3-way handshaking 패킷과 같이 **frame , ethernet2 frame header (link layer), Internet Protocol Version 4.. , Transmission Control Protocol..**을 통해서 패킷의 상세 정보를 확인할 수 있다.

**# Hypertext Transfer Protocol**

* HTTP type으로 패킷을 전달한다.
* Request Method는 GET 형식이라는 것을 알 수 있고, 요청하는 URI는 /mafia/reg/ 인 것을 알 수 있다.
* Packet bytes 부분의 첫 번째 줄에 ‘GET /mafia/reg’ 로 되어 있는 것을 확인할 수 있다.

그림 2-3

* 그림 2-3의 [HTTP request 2/10]이라는 메시지를 통해 HTTP request 요청을 보내고 있다는 것을 쉽게 알 수 있다.
* HTTP version은 HTTP/1.1 이라는 것을 알 수 있다.

**Frame No. 10 :**



스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-1-7] 패킷 번호 10번 패킷과 상세보기 화면*

* 서버가 클라이언트에게서 No.9 패킷을 받았을 때, 잘 받았다는 [ACK] 메시지를 TCP형식으로 보낸다.

**# Transmission Control Protocol**

* src, dst의 port 번호를 확인할 수 있다.
* seq 번호는 150이고, 다음에 받기를 기대하는 ack는 920임을 알 수 있다.
* Checksum, window size등의 정보를 확인할 수 있다.
* 을 통해 frame 9 (No.9)에 대한 응답 패킷이라는 것을 알 수 있다.

**Frame No. 11 :**



스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-1-8] 패킷 번호 11번 패킷과 상세 내용 화면*

**# Hypertext Transfer Protocol**

* HTTP version은 HTTP/1.1 이라는 것을 알 수 있다.
* Response Phrase를 보면 이전 요청(frame 9)에 대해 OK응답을 받은 것을 알 수 있다.

**# Line-based text data: text/html**

스크린샷이(가) 표시된 사진

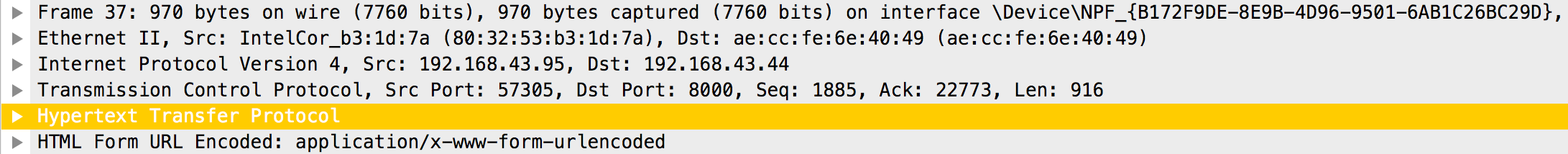
자동 생성된 설명

*[그림 2-1-9] 전달된 register.html 파일의 일부*

* 이 부분에서 text/html 형식으로 전달되는 데이터를 확인할 수 있다.
* Register.html 파일의 내용을 확인이 가능하다.

**Frame No. 37 :**





스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-1-10] 패킷 번호 37번 패킷과 상세 내용*

* [게임참가] 버튼을 클릭하는 동시에 frame 37의 패킷이 POST 형식으로 전송된다.

**# Hypertext Transfer Protocol (HTTP)**

* POST /mafia/regCon/ HTTP/1.1\r\n 부분을 보면 앞의 패킷과 같이 요청 형식(POST), 요청 URI(mafia/regCon), 버전(HTTP/1.1)등 정보를 알 수 있다.

**# HTML Form URL Encoded**

스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-1-11] 패킷 번호 37의 HTML Form URL Encoded 상세 부분*

* 유저가 이름을 입력한 후, 클릭하면 HTML에 포함된 item들이 Django의 DB로 전달된다.
* 이름(name), 직업(job), 투표수(votes), 생명(life)등의 유저 정보가 전달된다.

**2.2 대기 화면**

A. 대기 화면 입장

사진, 쥐고있는, 목재의, 방이(가) 표시된 사진

자동 생성된 설명

*[그림 2-2-1] 클라이언트가 서버에게 웹 페이지 요청*

* 마피아 게임에서 클라이언트가 첫 페이지를 서버에게 요청했을 때 3-way handshaking하여 연결하였고 그 연결을 그대로 사용한다 -> 추가적인 handshaking이 필요 없다(HTTP 1.1은 서버와 클라이언트가 단일 TCP로 연결하여 여러 객체들(test/html)을 전송할 수 있다
* HTTP 1.1은 Persistent 방식으로 서버가 응답을 보낸 후에도 연결을 그대로 열어 두어서 동일한 서버와 클라이언트 사이에 HTTP 메시지를 전송할 수 있다
* 클라이언트는 서버에게 IP/MAC 주소를 이용하여 web objects를 요청하고 서버로부터 요청에 대한 응답으로 세그먼트들을 받으면 응답으로 ACK를 전송한다
* No. 43에서 서버가 클라이언트에게 HTTP 방식으로 text/html 데이터를 전송하는 것을 볼 수 있다

스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-2-2] No. 39(클라이언트가 서버에게 HTTP 요청)*

* + Method = GET : URL을 가지고 있는 정보 검색을 위하여 서버에게 요청 -> 연결이 유지되기 때문에 추가적으로 GET Method를 계속 보낼 수 있다
  + Host : 클라이언트가 서버에게 요청한 웹 페이지의 URL, 필수 헤더
  + Connection : Keep-alive가 있기 때문에 서버에게 정한 시간만큼 연결을 유지(TCP연결 유지) -> HTTP 1.0에서는 Connection less방식이기 때문에 매번 Port를 열어야 합니다, 하지만 이는 비효율적이므로 HTTP의 1.1의 기능인 Keep-alive의 time-out내에서 클라이언트가 서버에게 request를 재 요청하면 서버는 Port를 새로 여는 것이 아니라 기존에 열려 있는 Port를 이용하여 전송하는 것 입니다
  + Accept : 클라이언트가 서버의 응답으로 받기 원하는 데이터의 종류(text/html)
  + Referrer : web objects 요청을 시도한 페이지의 URL을 표시하고 어떤 페이지를 통해서 들어 왔는지에 대한 정보 제공(/mafia/reg/ 웹 페이지를 통해서 들어옴)
  + 서버가 /mafia/room/의 objects을 클라이언트에게 전송한다. 이 때 서버는 한번에 전송하지 않고 세그먼트로 나누어서 전송한다
  + Previous request이 No. 37이고 Next request가 No. 45을 보아 처음 3-way handshaking을 한 것이 지속적으로 유지되는 것을 알 수 있다

스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-2-3] No. 40(서버가 세그먼트들을 나누어서 클라이언트에게 전송)*

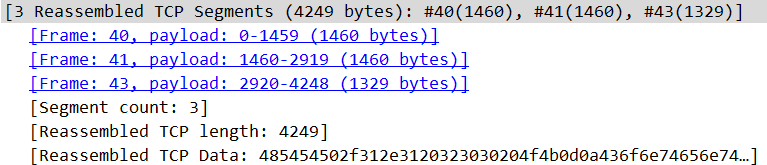
* 서버로부터 받았기 때문에 서버의 Port(8000)와 클라이언트의 Port(57305)를 알 수 있다(Source Port: 8000, Destination Port: 57305)
* Segment의 길이, Window size value, Checksum 등을 알 수 있다.
* 현재 Sequence number는 23044이고 Next sequence number가 24504인데 차이가 많이 나는 이유는 Mafia APP을 구동 시킬 때 서버 1개와 클라이언트 5개로 구성하였기 때문에 Sequence number의 차이가 큰 것이다
* 해당 세그먼트는 43번 Frame으로 모이는 것을 알 수 있다(Reassembled PDU in frame: 43)
* 서버가 클라이언트에게 보낸 세그먼트 데이터는 1460 Bytes이다(TCP segment data)

스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-2-4] No. 42(클라이언트는 잘 받았다고 응답(ACK))*

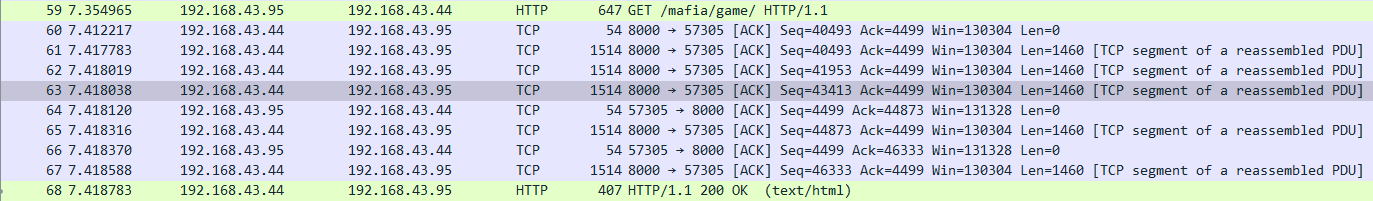
* + 서버로부터 세그먼트들을 받은 클라이언트는 잘 받았다는 응답으로 ACK를 전송
  + 클라이언트와 서버의 Port, Header의 길이, Window size value, Checksum 등이 담겨있다
  + No. 42 패킷에서는 서버로부터 받은 NO. 40, 41의 세그먼트에 대해서만 ACK 전송
  + 마지막 세그먼트는 서버의 응답신호와 함께 NO. 43에 포함해서 클라이언트에게 전송하고 No. 44에서 마지막 세그먼트를 잘 받았다는 ACK를 전송
  + TCP 연결을 유지하고 있기 때문에 이후에도 클라이언트는 GET Method로 서버에게 요청 가능
  + 기본적으로 와이어샤크는 TCP를 캡처할 때 sequence number와 ACK number를 상대 숫자로 변환(relative sequence/ack number)



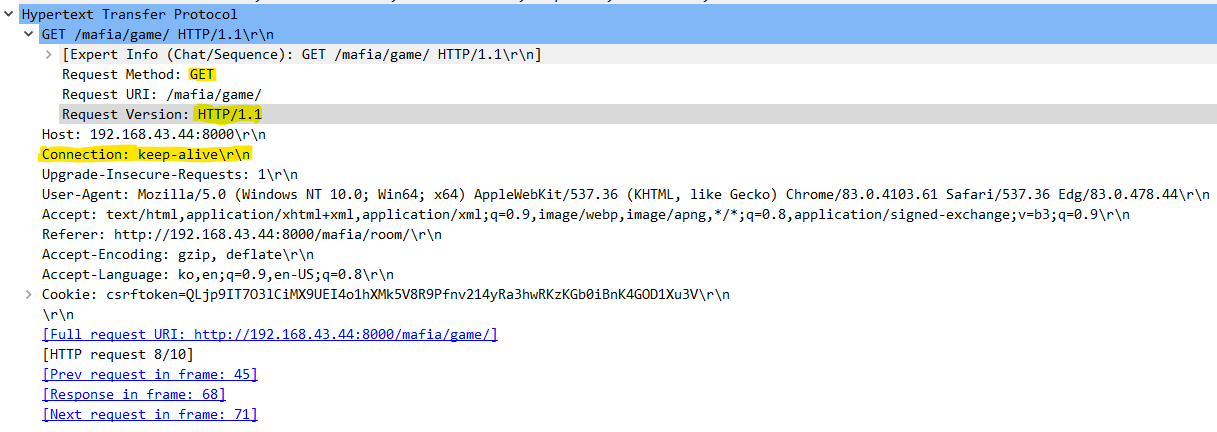
*[그림 2-2-5] No. 43(서버에서 받은 세그먼트를 클라이언트는 재조합)*

* 서버는 3개의 세그먼트로 나누어서 클라이언트에게 전송하였고 각각 1460 bytes, 1460 bytes, 1329 bytes로 나눴다 이 때 Frame number로 40, 41, 43을 각각 받고 클라이언트는 서버로부터 받은 나누어진 세그먼트들을 합치고 합친 내용을 출력한다
* 서버가 보낸 마지막 응답(세그먼트)에서는 총 세그먼트의 개수를 알 수 있다(Segment count : 3)
* 마지막 세그먼트의 패킷에서는 해당 html의 데이터도 볼 수 있다(No. 43에서 Line-based text data부분에 나와있다)
* 서버가 보낸 총 세그먼트가 4248 bytes인데 Reassembled TCP의 길이가 4249 bytes인 것은 3-way-handshaking할 때 1byte가 이미 추가되었으므로 1byte부터 시작하는 것이다
* TCP연결에서 APP이 사용할 수 있는 MSS(Maximum Segment Size)는 1460 Bytes인 것을 볼 수 있다(MTU(1500 Bytes) – (IP Header(20 Bytes) + TCP Header(20 Bytes)) = MSS(1460 Bytes))

**2.3 게임 화면**

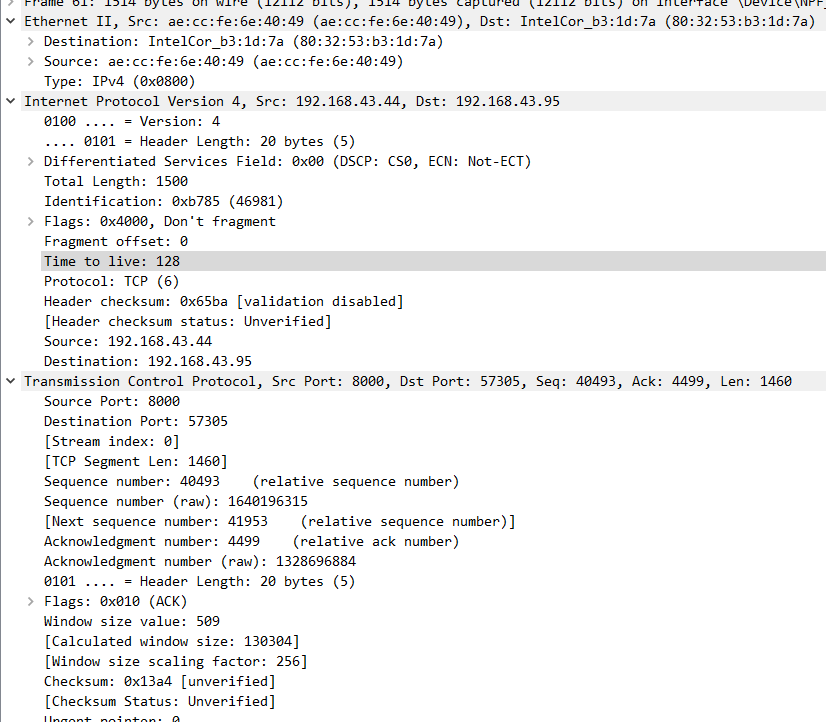
A. 투표화면 띄우기

*[그림 2-3-1]*

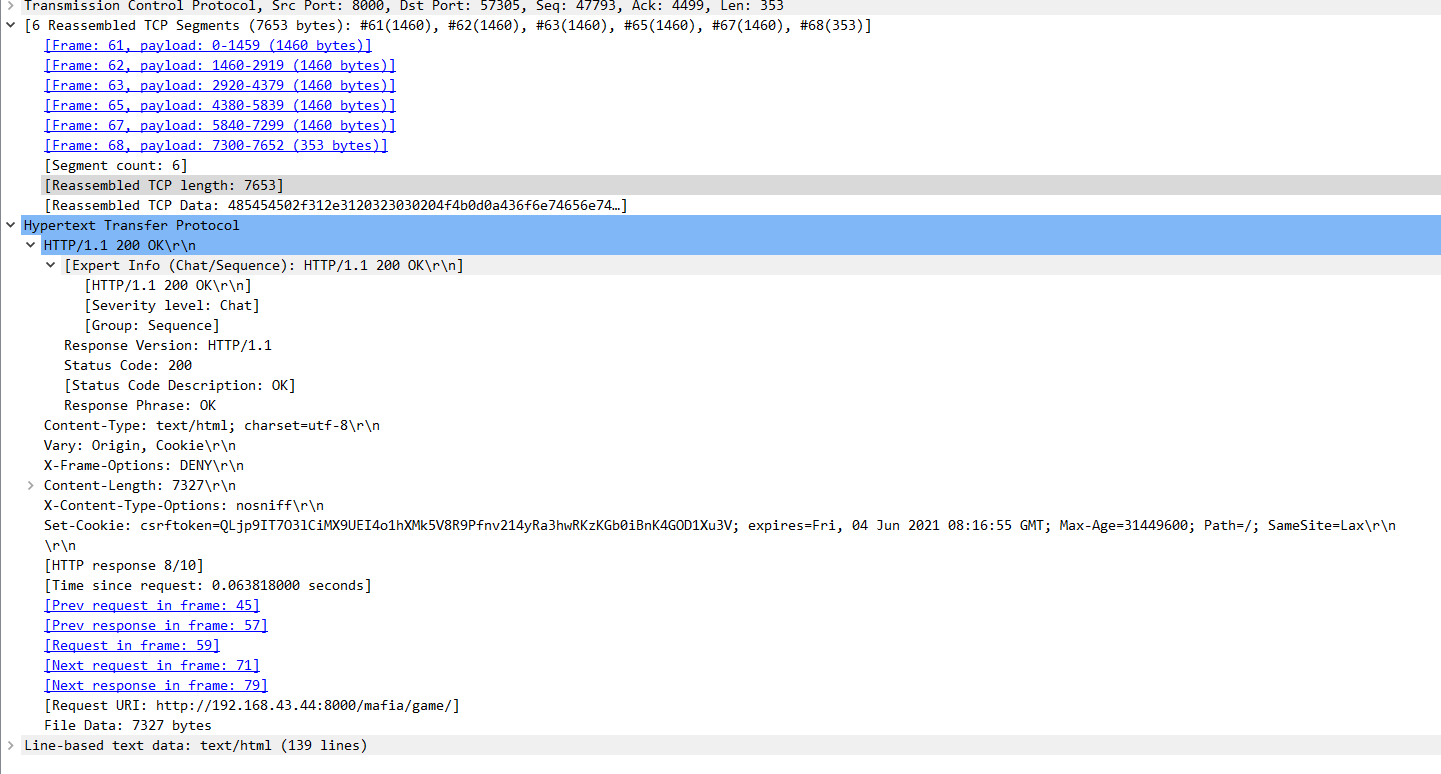
*[그림 2-3-2] Client (192.168.43.95)와 Server(192.168.43.44)가 URL을 요청*

**Frame No. 59 ~ 67 : HTTP/ 1.1 protocol로 packet 요청**

* Hypertext Transfer protocol을 사용(HTTP/ 1.1버전 사용 – connection을 object를 다 가져올 때 까지 유지합니다.)
* GET : 문서를 받고 싶을 때의 명령(직접적으로 object를 가져올 때는 GET의 명령어를 사용해 URL을 줘서 가져옵니다.)
* connection : Keep-alive : HTTP/1.1부터는 object를 다 가져올 때까지 지속적인 연결의 개념을 도입했습니다. (HTTP/1.0보다 효율이 더 좋아짐.) 🡪 TCP를 사용

*[그림 2-3-3]*

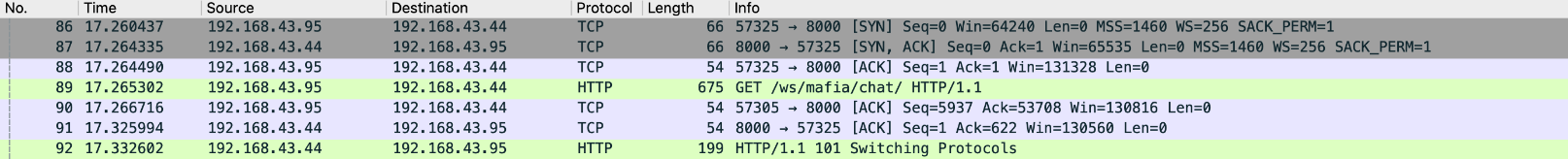
* network Layer(Internet protocol version 4)에서 TTL은 datagram의 생존 기간을 의미
* 시간 단위는 hop을 사용합니다. (router를 1개 지날 때마다 one-hop) (os마다 TTL의 범위가 다르다.)
* Link layer에서 Ethernet protocol을 사용해서 데이터의 길이는 1460이 되는 것을 볼 수 있습니다. (40은 Ethernet protocol Header의 크기)

*[그림 2-3-4]*

* NO. 68 : 서버로부터 붙은 Frame을 assemble합니다.
* NO. 68 : 서버로부터 클라이언트에게 200 OK라는 메시지를 받습니다. HTTP 200 OK는 요청이 성공했음을 나타내는 성공 응답 상태 코드입니다.

**2.4 채팅 화면**

A. 웹소켓 연결



*[그림 2-4-1] 웹소켓 연결*

Websocket도 HTTP와 비슷한 방식으로 연결한다.

1. 3-way handshake : No. 86,87,88

No. 86: 클라이언트는 서버에게 SYN을 보내 연결 요청을 한다.

No. 87: 서버가 클라이언트에게 패킷을 잘 받았고 연결 요청을 수락한다. 이때 ACK는 Seq+1 하여 보낸다.

No. 88: 클라이언트가 서버에게 패킷 잘 받았고 데이터를 송수신 하자는 패킷을 보낸다. 받은 ACK가 1이므로 Seq가 1이고 ACK는 이전 Seq+1 을 하여 보낸다.

1. 연결 성립 후, 클라이언트는 서버에게 GET 방식으로 HTTP 요청: No. 89

No. 89: GET ws/mafia/chat HTTP/1.1 로 서버에게 chat.html을 요청하고 있다.

조류이(가) 표시된 사진

자동 생성된 설명

*[그림 2-4-2] GET ws/mafia/chat HTTP/1.1 로 서버에게 chat.html을 요청*

* URL에 ws가 붙어있는 이유는 HTTP요청과 websocket 요청을 구분 짓기 위한 것이다.

Websocket은 HTTP 1.1 프로토콜과 호환되는 요청 응답 메시지로 서버와 Hand shaking을 수행한다.

* 클라이언트가 Websocket 연결을 설정하고자 한다는 것을 서버에게 알려주는   
  “upgrade” 헤더가 이 요청에 포함된다.

1. 서버가 클라이언트에게 HTML 전송: No. 92  
   No. 92: HTTP/1.1 101 Switching Protocols 로 response하여 연결되었다.

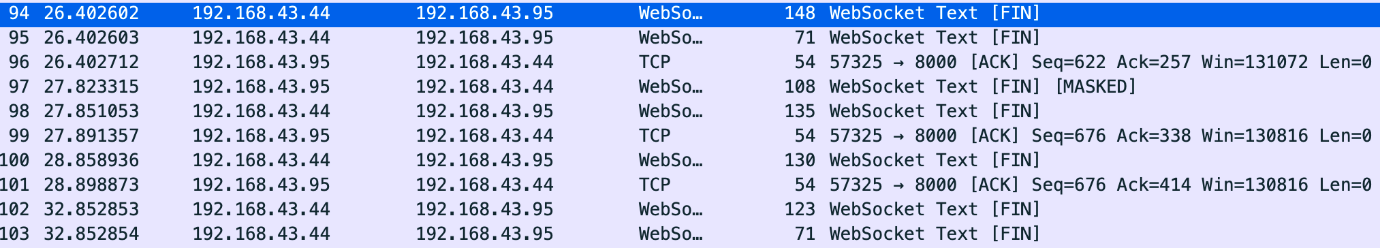
스크린샷, 조류이(가) 표시된 사진

자동 생성된 설명

*[그림 2-4-3] HTTP/1.1 101 Switching Protocols 로 response*

* 연결이 설정된 후 서버는 “upgrade”를 통해 응답한다.
* 클라이언트가 보낸 WebSocket-Key에 고정 GUID를 붙여 SHA-1해싱과 base64로 인코딩한 값이 WebScoekt-Accept에 채워 보낸다.

B. 웹소켓 텍스트



*[그림 2-4-4] 웹소켓 텍스트 패킷*

**-Websocket Text [FIN]**

스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-4-5] 웹소켓 텍스트 [FIN]*

**\*Fin:** 1bit로 이 프레임이 메시지를 구성하는 최종 프레임인 것을 알 수 있다.

**\*Reserved:** 0은 확장이 되지 않음을 뜻한다.

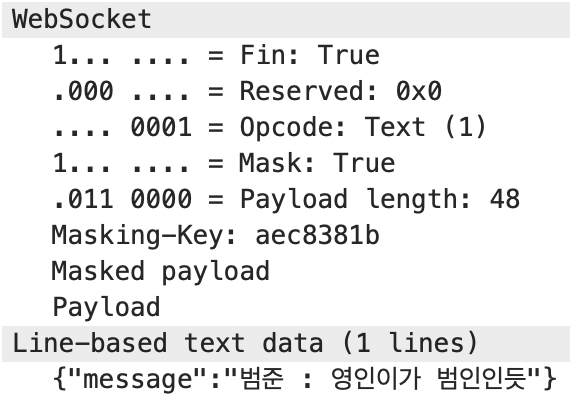
**\*Opcode:** 0x01은 텍스트를 뜻한다.

**\*Mask:** 서버가 보낸 패킷으로 0으로 마스크되어 있지 않다. (인코딩 되어 있지 않다.)

**\*Payload\_len:** 운반되는 데이터의 길이가 92이다

**\*message:** 채팅창에 다른 사람이 친 json 형태의 메시지이다.

**- Websocket Text [FIN] [MASKED]**



*[그림 2-4-6] 웹소켓 텍스트 [FIN] [MASKED]*

**\*Fin:** 1bit로 이 프레임이 메시지를 구성하는 최종 프레임인 것을 알 수 있다.

**\*Reserved:** 0은 확장이 되지 않음을 뜻한다.

**\*Opcode:** 0x01은 텍스트를 뜻한다.

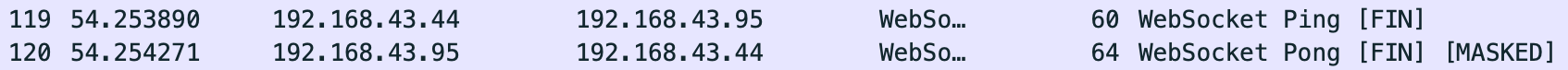
**\*Mask:** 1로 연결이 마스크 되어있다.

**\*Payload\_len:** 운반되는 데이터의 길이가 48이다

**\*message:** 채팅창에 본인이 친 json 형태의 메시지이다.

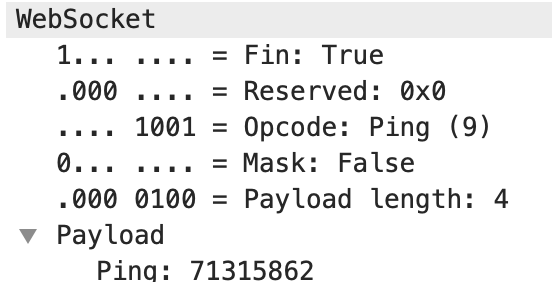
C. 웹소켓 핑퐁

Heartbeat로 클라이언트가 계속 연결되어 있는지 알 수 있다.



*[그림 2-4-7] 웹소켓 핑퐁 패킷*

**Ping :No. 119**



*[그림 2-4-8] 웹소켓 핑*

**\*Fin:** 1bit로 이 프레임이 메시지를 구성하는 최종 프레임인 것을 알 수 있다.

**\*Reserved:** 0은 확장이 되지 않음을 뜻한다.

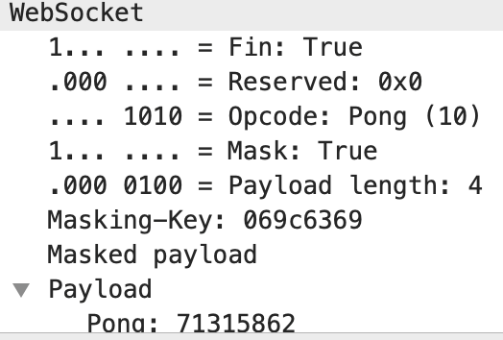
**\*Opcode:** 0x09는 ping을 뜻한다.

**\*Mask:** 서버가 보냈으므로 Mask 비트가 0으로 연결이 마스크 되어있지 않다.

(메세지가 인코딩 되어 있지 않다. )

**\*Payload\_len:** 운반되는 데이터의 길이가 4이다.

**Pong: No. 120**

****

*[그림 2-4-9] 웹소켓 퐁*

**\*Fin:** 1bit로 이 프레임이 메시지를 구성하는 최종 프레임인 것을 알 수 있다.

**\*Reserved:** 0은 확장이 되지 않음을 뜻한다.

**\*Opcode:** 0x0a는 pong을 뜻한다.

**\*Mask:** bit가 1로 마스크 되어있어 메시지가 인코딩 되어 있다.

**\*Payload\_len:** 운반되는 데이터의 길이가 4이다.

D. TCP WINDOW UPDATE



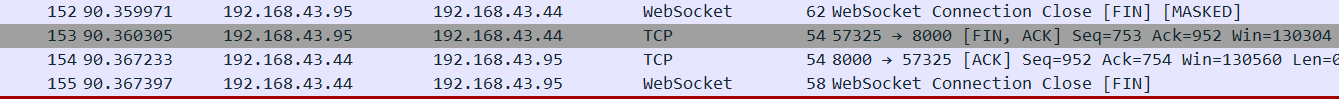
테이블이(가) 표시된 사진

자동 생성된 설명

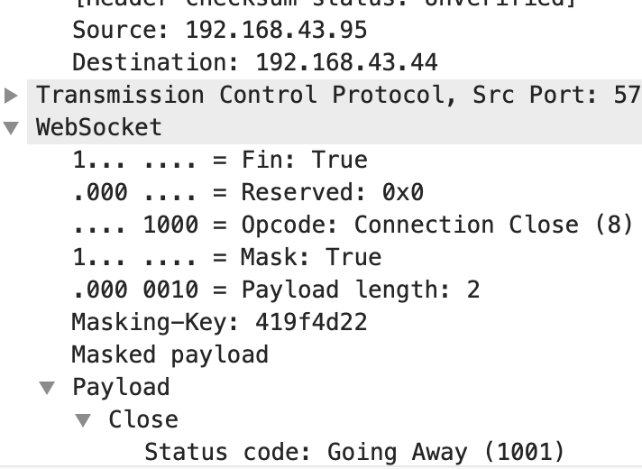
*[그림 2-4-10] TCP WINDOW UPDATE 패킷*

서버가 클라이언트에게 버퍼가 얼마나 남았는지 계산해서 calculated window size에 넣어서 알려줌으로써 남은 크기에 따라 서버가 보내는 속도를 조절할 수 있게 한다.

E. 웹소켓 포트 연결 종료

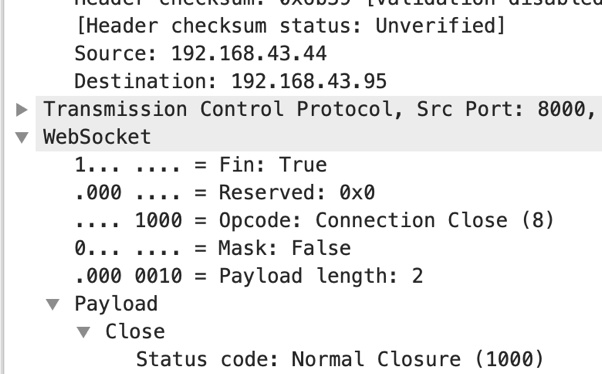


*[그림 2-4-11] HTTP와 웹소켓 포트 연결 종료*

**

*[그림 2-4-12] No.152*

**No.152**. (MASKED) WebSocket Connection close FIN을 Client(192.168.43.95) Server(192.168.43.44)에게 보내 연결 종료를 요청한다. Payload에는 Close가 들어있다.

**

*[그림 2-4-13] No.155*

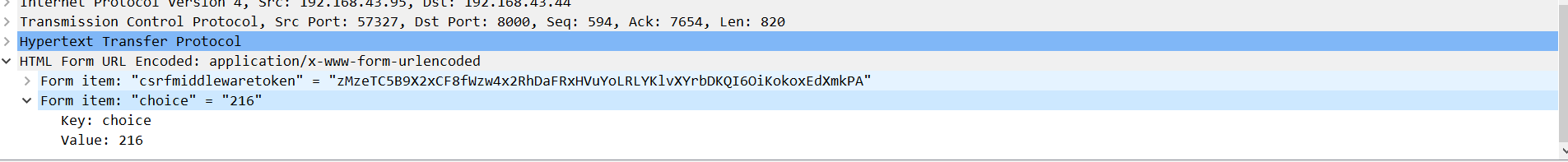
**No.155**. WebSocket Connection close FIN을 Server(192.168.43.44)가 Client(192.168.43.95)에게 보내며 종료된다. Status code를 보면 정상적으로 종료 된 것을 알 수 있다.

**2.5 마피아 투표**

A. 투표

\*투표

각각의 참가자가 투표 html에서 누른 choice 버튼이 server로 전달이 잘 되었음을 아래 이미지에서 확인할 수 있다.



*[그림 2-5-1 choice]*









*[그림 2-5-2 each other IP]*

위의 이미지를 통하여 서버를 열었던 관리자 1명(192.168.43.44) 을 제외, 참가자 5명이 각자 다른 frame 에서 다른 IP로 게임의 마지막에서 자신이 원하는 마피아를 선택하였고 그 정보를 서버로 전송하였다.

\* Frame143

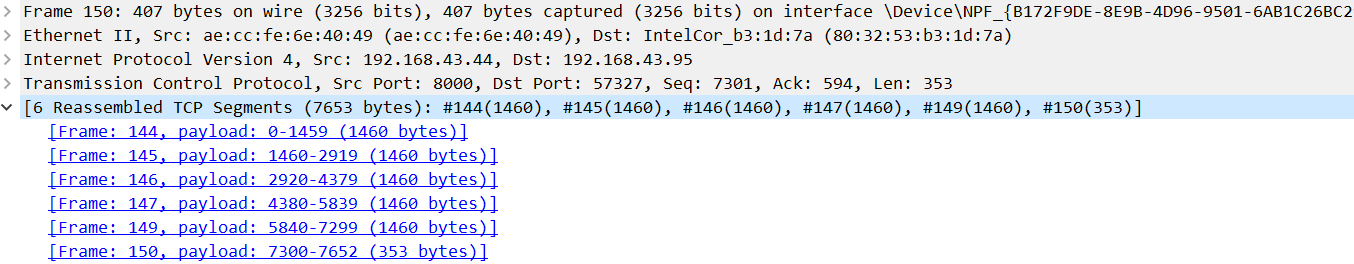


*[그림 2-5-3 GET request]*

GET 방식으로 투표 html 인 /mafia/game/ HTTP을 request 했다.

Responsing은 frame 150에서 진행했다.

다음 request 는 frame 159에서 일어나는 것을 확인할 수 있다.



*[그림 2-5-4 TCP segments]*

Frame 144, 145, 146, 147, 149, 150 에서 보이듯이 server가 TCP segment를 나누어서 transfer했고 reassemble 되었다.

**2.6 승/패 결과 화면과 종료**



텍스트, 앉아있는, 테이블, 쥐고있는이(가) 표시된 사진

자동 생성된 설명

*[그림 2-6-1] 결과창 띄어주고 종료하는 부분 캡쳐*

A. 클라이언트가 서버에게 HTTP요청

**NO.159**

스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-6-2] 서버에게 HTTP 요청*

* 요청 라인이 POST /mafia/polls/ HTTP/1.1인 것을 보아 클라이언트가 서버에게 POST 방식으로 HTTP Request 요청을 보내는 것을 확인할 수 있다.
* 헤더 라인을 보면 Connection이 Keep-alive이기 때문에 서버에게 정한 시간만큼 연결을 유지한다.
* user-agent를 보면 클라이언트가 크롬을 사용하고 있다는 점을 알 수 있다.
* Referer를 통해 클라이언트가 넘어온 곳을 알 수 있다.
* Accept language가 ko인 것을 보면 한국어 버전을 원한다는 것을 알 수 있다.
* [Response in frame: 167]을 보면 이에 대한 응답이 167번에서 일어난다는 것을 알 수 있다.

GET 방식과 POST 방식의 차이점을 살펴보자면 우선 GET 방식은 HTTP 프로토콜에서 헤더 부분만 존재하고 바디 부분이 존재하지 않는다. 또한 GET 방식은 헤더 안에 URL에 데이터를 담아 전달하며 URL에 길이 제한 때문에 데이터 입력 제한이 생긴다. 반면에 POST 방식은 바디 안에 데이터를 전달한다. 그래서 GET 방식은 데이터가 노출되고 POST 방식은 데이터가 노출되지 않는다.

B. 서버가 클라이언트에게 HTTP로 text/html 데이터 전송

**NO.167**

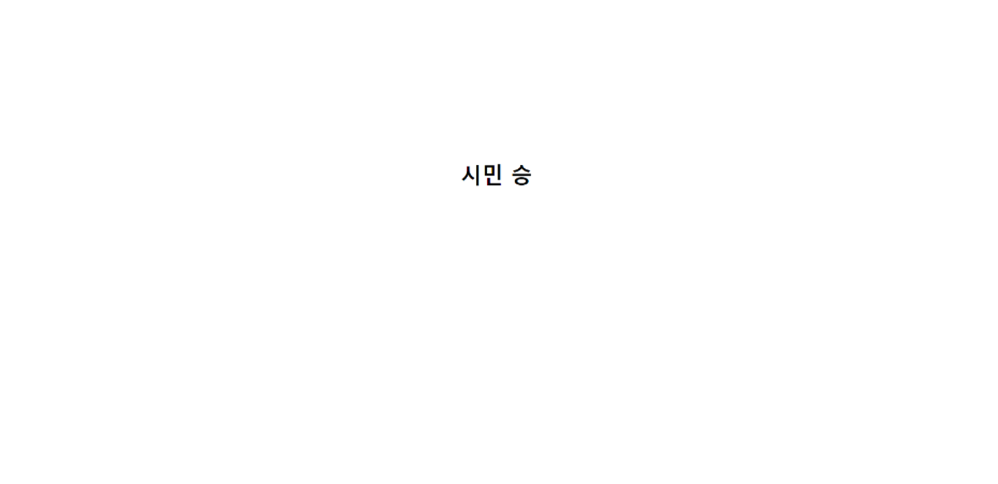
스크린샷이(가) 표시된 사진

자동 생성된 설명

*[그림 2-6-3] No.167 패킷*

* 상태 라인이 HTTP/1.1 200 OK인 것을 보면 정상적으로 요청한 내용을 전송했다는 것을 알 수 있다.
* Content-Type 헤더를 통해 text/html이라는 것을 알 수 있다.
* [Request in frame: 159]를 통해 159번의 요청에 대한 응답이라는 것을 알 수 있다.
* 패킷의 상세정보에서 [Line-based text data]항목을 보면 게임 결과 html 내용을 확인할 수 있다.

위의 결과로 아래와 같은 시민 승리창이 뜨는 것을 볼 수 있다.



*[그림 2-6-4] 결과 화면*