컴퓨터 네트워크

팀 프로젝트 최종 보고서

제출일	2022.06.23	전 공	항공교통물류학부
과 목	컴퓨터네트워크	학 번	2020123103
담당교수	최자봉 교수님	이 름	하현정

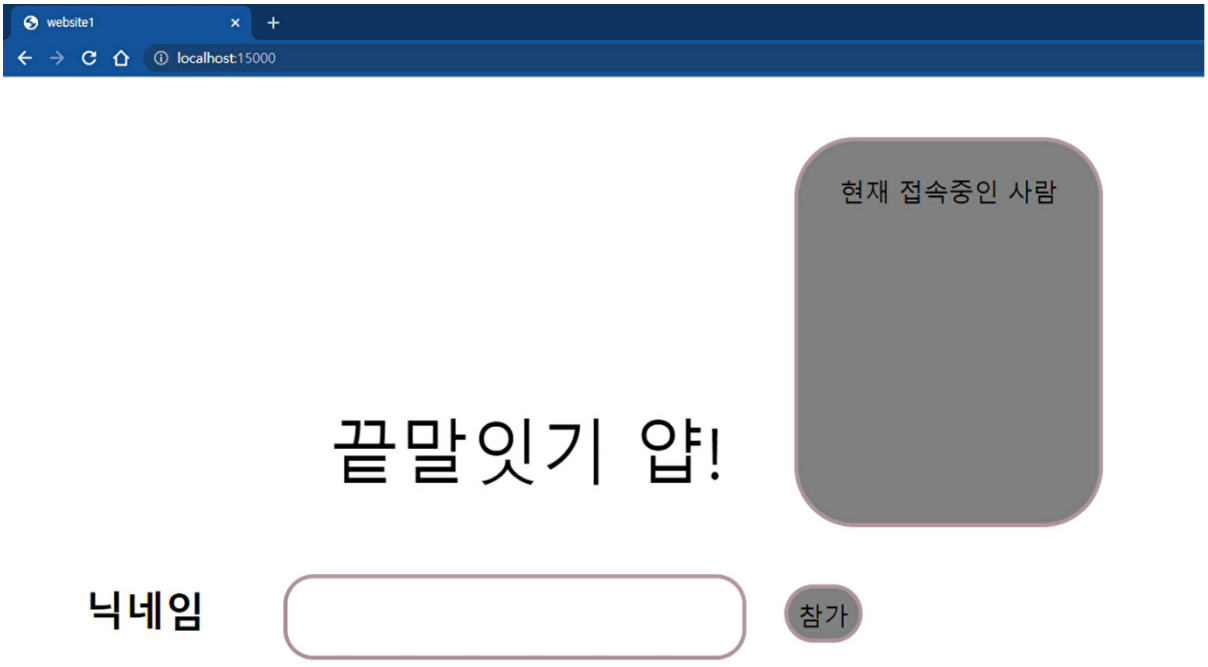
목차

1. 웹 서버에 대한 설명

1.1 프로젝트 개요

우리가 기획한 웹 서버 프로젝트는 끝말잇기 웹 게임 서버이다.
클라이언트들이 각자의 닉네임을 정하여 참가버튼을 누르면 게임화면으로 입장하게 된다. 모든 클라이언트들이 준비 버튼을 눌러 준비가 완료되면 게임이 시작된다. 한 게임의 시간은 총 300초이고, 클라이언트 별 한 턴의 시간은 10초이다. 이때 본인의 턴이 아닌 클라이언트들도 채팅에 참여할 수 있다. 게임이 종료되고 최종 점수와 우승자 그리고 꼴등이 결정되는데, 점수 산정 방식은 게임 중 채팅창에 입력한 글자수가 많을수록, 그리고 제한시간을 적게 이용할수록 해당 클라이언트에게 높은 점수를 부여하도록 설정했다.
프로젝트의 초기 계획으로는 게임에 '라운드 개념'을 이용하여 이전 게임의 꼴등은 다음 게임에 참여할 수 없도록 관전자로 분리하는 참가자 간의 경쟁요소를 도입하려 했다. 하지만 실제 구현에서는 게임 방식을 라운드 개념이 아닌 단판제로 변경했고, 대신 게임 한 판에 제한시간을 더 길게 배정하는 방식으로 진행 방식을 수정했다.
또한 초기에는 게임의 진행 순서(턴 순서)도 랜덤으로 배정할 계획이었지만, 실제 구현한 서버에서는 서버에 접속한 클라이언트들 순서대로 턴을 배정했다.

1.2 프로그램 동작 시나리오



1. 초기에 클라이언트가 서버에 TCP 연결을 하고 HTTP GET 요청을 하면 서버는 타이틀 화면을 보내준다.
이 타이틀 화면에서는 클라이언트가 웹 서버에서 사용할 닉네임을 설정할 수 있다.
또한 현재 웹 서버를 이용 중인 사람들의 닉네임도 확인할 수 있다.
2. 클라이언트가 서버에 POST로 닉네임을 보내고 채팅방 입장을 요청한다.
3. 서버는 클라이언트에 채팅방 화면 정보를 보내주는데, 여기에는 채팅창, 플레이어 목

록, 플레이어 상태, 점수판 등의 정보가 포함된다.

4. 클라이언트가 채팅을 치면 소켓으로 서버에 데이터를 전달하고, 연결되어 있는 모든 소켓으로 채팅 데이터를 다 보내줘서 모든 클라이언트들의 채팅창이 동기화된다. 동시에 준비 상태도 마찬가지로 작동한다.

5. 모든 클라이언트가 준비 버튼을 눌러 준비 상태가 되면 서버에서 끝말잇기 게임 로직을 실행한다.

프로젝트 구현과 패킷 분석 등은 게더 타운을 이용하여 다함께 진행하였으며, 패킷 캡쳐는 정제우님의 컴퓨터에서 진행하였다. 서버는 정제우 님의 컴퓨터로 열었고 서버의 IP주소는 119.149.104.73이다. 119.149.104.73:15000주소로 공유기를 통해 가서 포트포워딩을 통해 서브넷에 있는 192.168.200.157까지 이동한다. 실제 서버는 192.168.200.157:15000에서 실행이 되어 패킷의 서버 주소는 192.168.200.157로 뜬다.

클라이언트 - 최연우 (공유기): 59.6.233.237

클라이언트 - 김재혁 (공유기): 112.150.94.180

클라이언트 - 하현정 (공유기): 175.126.156.162

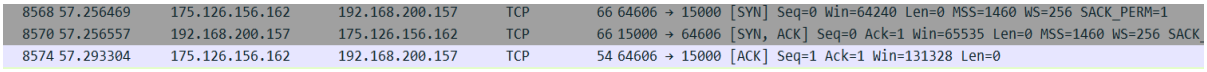
클라이언트 - 황규진 (이더넷): 183.109.36.160

클라이언트 - 정제우 (공유기): 192.168.200.157

2. 패킷의 캡쳐 및 분석

*내 컴퓨터의 ip로 필터링하여 나와 서버간의 통신의 패킷들을 중심으로 분석해보았다.

2.1 3way handshaking 과정 분석



3 way Hand-shaking 과정을 보여주는 패킷 캡쳐 내용이다. UDP에서는 connection management가 필요 없는 반면에(데이터 보낼 때 안전을 보장해주지 않음), TCP는 안정적인 데이터 공유를 위해 Connection을 맺는다. 그렇다면 왜 2 way handshaking이 아니라 3way handshaking 일까? 2way handshaking의 문제점을 생각해보면 된다. 만일 클라이언트가 연결을 시도하여 서버가 ok를 했는데 서버 입장에서는 자신의 응답을 클라이언트가 알아들었는지 몰랐는지 알 수가 없다. 네트워크 delay가 있으면 문제가 발생할 수 있기 때문에 3way handshaking을 사용한다. 3way handshaking을 step별로 살펴보자.

Step 1 : 클라이언트가 SYN 세그먼트를 보냄. (SYN flag=1, 헤더에 seq# 넣어줌)

Step 2 : 서버는 클라이언트가 보낸 SYN에 대한 ACK을 보냄. 그리고 SYN Flag=1, ACK Flag=1, ACK#=client_isn+1을 보냄. Full duplex이기 때문에 서버도 서버의 initial sequence number를 보내줌

Step 3 : 서버의 SYNACK을 받은 뒤 다시 그 SYN에 대한 ACK을 보냄. Syncronize되었으니 SYN flag는 0이 되고, ACK flag=1, Ack #=server_isn+1을 넣어서 보냄.

이러한 과정이 끝난 뒤에 클라이언트와 서버 사이에 데이터가 이동할 수 있게 된다.

Internet protocol에는 application layer, transport layer, network layer, link layer, physical layer가 있으며 이들은 복잡한 시스템을 독립적인 여러 layer로 구성하여 복잡도를 분산시킬 수 있어 각각 존재하게 된다. 또한, 다른 layer에 직접적인 영향을 주지 않으므로 시스템을 업그레이드하거나 유지보수하기가 쉬워진다는 장점이 있다. 이제부터 layer의 상

위계층부터 순서대로 분석해보려고 한다.

2.2 application layer 분석

```
8575 57.293304   175.126.156.162   192.168.200.157   HTTP    561 GET / HTTP/1.1
8576 57.296251   192.168.200.157   175.126.156.162   TCP    1514 15000 → 64606 [ACK] Seq=1 Ack=508 Win=130816 Len=1460 [TCP segment …
8577 57.296251   192.168.200.157   175.126.156.162   HTTP    947 HTTP/1.1 200 OK  (text/html)
8579 57.303512   175.126.156.162   192.168.200.157   TCP      66 64608 → 15000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM…
```

서버에게 파일을 요청(GET)하고, HTTP 프로토콜 버전 1.1로 통신하기를 원한다는 의미로, 게임 시작 화면으로 연결할 것을 요청한다. HTTP는 HTML파일을 주고받을 때 쓰는 프로토콜이다. HTTP는 TCP를 사용하며, stateless(이전 state는 기억하지 않음)이다. 어떤 history를 가지고 있는지 기억하게 되면 서버가 너무 복잡해지기 때문이다. 이러한 특성 때문에 HTTP는 쿠키를 사용한다. 따라서 서버가 유저번호를 부여한 뒤에 이를 backend batabase에 저장하고 응답을 줄 때 html정보 담겨질 때 쿠키번호를 같이 주게 된다. 클라이언트는 자신이 접속했던 URL 정보를 가지고 있기 때문에 이 정보와 쿠키번호를 함께 저장하고 그 다음엔 쿠키로 request를 하게 된다. 그럼 서버는 그것을 통해 응답을 하게 된다. HTTP는 non-persistent HTTP와 persistent HTTP가 존재하는데, non-persistent HTTP(HTTP1.0부터 지원)는 object마다 connection을 맺어야 하는 반면에, persistent HTTP(HTTP 1.1부터 지원)는 connection을 한 번 맺은 상태로 object를 여러 개 보낸다.

```
Hypertext Transfer Protocol
> GET / HTTP/1.1\r\n
  Host: 119.149.104.73:15000\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36 Edg/102.0.1245.39\
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: ko,en;q=0.9,en-US;q=0.8\r\n
  If-None-Match: W/"822-eAnF7xsVjK4478/wjZvJIF0hRX8"\r\n
```

메소드는 GET 방식이며 HTTP 버전은 1.1임을 알 수 있다. 따라서 이는 위에서 설명한 바와 같이 persistent HTTP를 지원한다. 이의 장점은 바로 빠른 응답시간이다. non-persistent의 응답시간은 (2RTT+file transmission time)*N인 반면에 persistent HTTP의 응답시간은 RTT*(RTT+file transmission time)*N이다. 메소드를 살펴보면, HTTP 1.0에서는 GET,POST,HEAD를 사용하고, HTTP 1.1에서는 GET,POST,HEAD,PUT,DELETE를 사용한다. 이 패킷의 메소드는 GET이라는 것을 볼 수 있다. GET method와 POST method의 차이를 보면 GET method는 데이터를 넣을 때 URL을 넣지만 POST method는 바디에 데이터를 넣어서 전송한다. 즉, GET method의 내용은 URL에서 데이터가 보이는 반면에 POST method는 바디에 데이터가 있기 때문에 URL에서 데이터가 보이지 않는다. DELETE method는 URL 필드에 특정 파일 삭제를 요청하는 메소드이고, PUT은 필드에 전체 리소스를 업로드 하는 메소드이다.

```
Hypertext Transfer Protocol
> HTTP/1.1 200 OK\r\n
  X-Powered-By: Express\r\n
  Content-Type: text/html; charset=utf-8\r\n
> Content-Length: 2122\r\n
  ETag: W/"84a-GGqF+B4XidNj0b8VOIwRIetr5RE"\r\n
  Date: Mon, 13 Jun 2022 12:43:40 GMT\r\n
  Connection: keep-alive\r\n
  Keep-Alive: timeout=5\r\n
  \r\n
  [HTTP response 1/5]
  [Time since request: 0.002947000 seconds]
```

HTTP response message를 살펴보면, 헤더라인에는 content-type으로 이 메시지의 형식을 볼 수 있으며 content-length를 통해 크기가 2122이라는 것을 알 수 있다. HTTP의

response status codes를 보면 이 패킷에서는 현재 200 OK 코드를 나타내고 있다. HTTP의 상태코드들을 살펴보면,

1xx:정보 , 2xx : 성공 , 3xx : 재지정 , 4xx : 클라이언트 오류 / 5xx : 서버오류를 나타낸다. 여기서는 상태코드가 200인 것으로 보아 요청이 성공적으로 처리되었다는 것을 볼 수 있다.

```
9743 61.235811    192.168.200.157    175.126.156.162    HTTP    167 HTTP/1.1 304 Not Modified
9796 61.371950    192.168.200.157    175.126.156.162    HTTP    183 HTTP/1.1 101 Switching Protocols
```

다른 패킷을 보면 304, 101 등의 상태코드를 확인할 수 있다.

```
Frame 9789: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\
> Interface id: 0 (\Device\NPF_{C5568CCA-208A-4AE7-94E7-6EBCE7FF5D25})
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun 13, 2022 21:43:44.676385000 대한민국 표준시
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1655124224.676385000 seconds
  [Time delta from previous captured frame: 0.008974000 seconds]
  [Time delta from previous displayed frame: 0.008974000 seconds]
  [Time since reference or first frame: 61.337550000 seconds]
  Frame Number: 9789
  Frame Length: 66 bytes (528 bits)
  Capture Length: 66 bytes (528 bits)
```

2.3 transport layer 분석

Transport layer는 application layer에서 데이터를 보내면 그것을 실제적으로 네트워크의 여러 entity를 통해 원하는 목적지까지 전달해주는 계층이다. Transport layer의 프로토콜로는 UDP, TCP가 있다. TCP는 유실이 되지 않아야 해서 조금 더 복잡하다. TCP는 reliable 한 전송을 위하여 congestion control, flow control, connection setup등 다양한 것을 수행한다. 반면 UDP는 신뢰성은 없지만 빠르다.

```
Transmission Control Protocol, Src Port: 64609, Dst Port: 15000, Seq: 1261, Ack: 158
  Source Port: 64609
  Destination Port: 15000
  [Stream index: 36]
  [Conversation completeness: Complete, WITH_DATA (31)]
  [TCP Segment Len: 0]
  Sequence Number: 1261    (relative sequence number)
  Sequence Number (raw): 1783101755
  [Next Sequence Number: 1261    (relative sequence number)]
  Acknowledgment Number: 15870    (relative ack number)
  Acknowledgment number (raw): 2181526101
  0101 .... = Header Length: 20 bytes (5)
```

TCP의 패킷을 살펴보면 처음으로 출발 포트와 도착지 포트를 볼 수 있다. 이 패킷의 sequence number와 다음 전달 될 next sequence number를 알 수 있다. 또한 next sequence number – sequence number의 값이 bytes of data인 것 또한 알 수 있다. Acknowledgement number은 다른 쪽으로부터 올 다음 바이트의 seq #이다. 이는 cumulative ACK을 사용하여 이 이전의 것들 것 모두 받았다는 의미로 사용이 된다.

```
> Flags: 0x010 (ACK)
  Window: 511
  [Calculated window size: 130816]
  [Window size scaling factor: 256]
  Checksum: 0x4994 [unverified]
```

Flags를 살펴보자. TCP는 URG,ACK,PSH,RST,SYN,FIN 총 6가지의 Flag를 가지고 있다.
먼저 URG flag는 긴급데이터라는 뜻으로 예를 들어 명령 수행 도중 ctrl+c를 눌러 인터럽트하는 행위이다. ACK flag는 패킷을 잘 받았다는 것을 알려주는 것이다. PSH는 Application 계층으로 전송하는 flag이다. RST는 비 정상적인 세션 연결 끊기이며 즉시 연결을 끊는다. SYN은 TCP 교환 과정의 첫번째 패킷으로 위의 3handsaking에서 ACK와 함께 보았다. 마지막으로 FIN은 세션을 종료 시킬 때 사용한다. FIN 패킷은 밑에서 다시 한 번 살펴볼 것이다.
Window는 수신 버퍼에서의 여유공간을 나타낸다. 즉, 수신 버퍼에서 받을 수 있는 버퍼의 사이즈를 의미하는 것이다. Application layer가 network layer보다 데이터를 늦게 가져가게 된다면 TCP socket receiver buffer에 이를 저장해야 한다. 그러나 버퍼가 꽉 차게 되면 네트워크 상의 문제가 발생한다. 따라서 이에 대한 처리를 위해 flow control을 사용하고, 이는 end 시스템 사이에서의 제어를 의미한다. Rwnd를 통해 수신쪽에서 receive window의 공간이 얼마있는지 알게 되어 sender 측에서 전송량을 조절하게 된다. lastbytesent-lastbyteacked가 rwnd의 크기보다 작거나 같아야 하기 때문에 이에 맞춰 조절을 하게 된다.

61456	494.374915	175.126.156.162	192.168.200.157	TCP	54 64636 → 15000 [ACK] Seq=527 Ack=181 Win=…
61457	494.374915	175.126.156.162	192.168.200.157	TCP	54 64636 → 15000 [FIN, ACK] Seq=527 Ack=181…
61459	494.375024	192.168.200.157	175.126.156.162	TCP	54 15000 → 64636 [ACK] Seq=181 Ack=528 Win=…

마지막으로 FIN flag이다. 위에서 flag를 살펴볼 때 설명했듯 FIN은 세션을 종료 시킬 때 사용한다. 즉, 3 way hand-shaking에 의해 연결된 것을 종료하는 용도이다. 끊을 때도 마찬가지로 클라이언트가 서버에게 요청하는 즉시 끊어버리면 안된다. 그러면 segment가 network 상에서 떠돌다가 오작동이 발생할 수 있기 때문이다. 따라서 FIN -> FINACK -> ACK 과정을 통해 연결을 끊는다.

16641	102.826168	192.168.200.157	175.126.156.162	TCP	1514 [TCP Retransmission] 15000 → 64614 [PSH, ACK] Seq=954 Ack=527 Win=130816 L
16666	102.938584	175.126.156.162	192.168.200.157	TCP	66 64614 → 15000 [ACK] Seq=527 Ack=2414 Win=131328 Len=0 SLE=954 SRE=2414
17412	106.233400	175.126.156.162	192.168.200.157	WebSoc…	114 WebSocket Text [FIN] [MASKED]
17416	106.234687	192.168.200.157	175.126.156.162	WebSoc…	108 WebSocket Text [FIN]
17461	106.524056	192.168.200.157	175.126.156.162	TCP	108 [TCP Retransmission] 15000 → 64609 [PSH, ACK] Seq=1065 Ack=678 Win=130560
58571	466.011236	175.126.156.162	192.168.200.157	TCP	66 64636 → 15000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
58576	466.011465	192.168.200.157	175.126.156.162	TCP	66 15000 → 64636 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK
58578	466.011717	175.126.156.162	192.168.200.157	TCP	54 [TCP Dup ACK 58560#1] 64609 → 15000 [ACK] Seq=1261 Ack=15867 Win=130816 Le
58579	466.011717	175.126.156.162	192.168.200.157	TCP	54 64635 → 15000 [ACK] Seq=1 Ack=1 Win=131328 Len=0

그렇다면, reliable data transfer을 위해서는 어떤 것들이 작동될까?
크게 두가지의 에러가 있다. 하나는 corruption (bit-error)이고, 하나는 패킷 유실이다(라우터에서 버퍼링을 하다가 버퍼가 다 차면 drop시켜버리는 것). Bit-error를 대비해 checksum혹은 ACK를 이용한다. 패킷 유실일 경우를 대비해서 time-out(일정시간이 지나도 ACK가 오지 않으면 다시 보냄), retransmission(재전송), sequence number(transport layer에서 segment로 나누는데 순서가 잘못된 경우의 에러를 방지하기 위하여)를 사용한다. 위의 패킷 캡쳐를 보면 duplicate ACK과 TCP retransmission을 볼 수 있다. Duplicate ACK은 정작 자신이 원하는 패킷이 오지 않아 계속해서 그 패킷을 달라고 요청하는 것이다. 송신측에서는 duplicate를 통해서 재전송을 해야 한다는 것을 알게 된다. 원래는 time out이 걸려야지만 retransmission을 했는데, 그러다보면 long delay가 발생한다. 따라서 이를 방지하기 위해 timeout이 오기 전 3개의 duplicate ACK이 오면 fast retransmit을 한다.

2.4 network layer 분석

네트워크상의 라우터들에게는 network layer까지만 존재한다. Application, transport layer가 존재하지 않는다. 즉 이들은 end-end system이고 라우터는 core system이라 생각하면 된다. 라우터는 데이터를 교차로 역할을 통해 목적지로 잘 전달해주는 것을 수행한다. 패킷을 가지고 최종

적으로 입력 링크로 받아 어떤 출력 링크로 보낼 것인지 결정한다. Network layer의 주요 function은 forwarding과 routing이다. 따라서 이 기능들을 수행하기 위한 것들로 헤더가 구성되어 있다. 그럼 이제 패킷을 살펴보자.

```
Internet Protocol Version 4, Src: 192.168.200.157, Dst: 175.126.156.162
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0xb013 (45075)
> Flags: 0x40, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: TCP (6)
  Header Checksum: 0x6fa1 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.200.157
  Destination Address: 175.126.156.162
```

이 패킷은 IPv4를 사용하고 있다는 것을 알 수 있다. IP버전은 4와 6이 있다. 6은 IPv4에서의 ip 주소 고갈로 인해 생겨났다. ip주소 고갈을 해결하기 위한 다양한 방법은 뒤에서 더 살펴볼 것이다. Ipv6와 IPv4의 가장 큰 차이는 IPv6는 4와 다르게 fragment를 종단에서만 수행하며, 헤더의 크기를 고정시켜두었다는 것이다. 그럼으로 인해 조금더 speed를 높이고 처리를 빠르게 할 수 있다. Internet protocol에서는 ip주소를 통해 라우팅을 하기 때문에 도착지와 출발지의 ip주소를 볼 수 있다. 다음으로 header length는 TCP 헤더의 크기와 같은 20bytes이다. Identification과 flags, fragment offset은 IP fragmentation/reassembly와 관련이 있다. Datagram이 network에서 보내질 때 link에서 보낼 수 있는 크기가 다를 수 있다. Ip는 군용으로 만든 것이고 전세계 미군이 주둔 중이기에 거기에 쓰고 있는 네트워크로 구성하여 각 나라별로 link layer가 다르더라도 network layer부터는 똑같이 쓸 수 있도록 해두었다. Link layer가 보낼 수 있는 frame의 사이즈가 종류에 따라 다르고 각각의 MTU를 가지고 있다. 만약 datagram이 MTU를 넘는다면 fragmentation을 하게 된다. MTU만큼 데이터를 쪼개는 것이다. 만약 4000byte의 데이터그램이 있고 MTU가 1500byte라면 fragmentation은 1480byte+20byte(헤더) , 1480byte+20byte, 1020byte+20byte로 쪼개진다. 이렇게 분리된 datagram을 다시 합치기 위해서는 id가 필요한 것이다. 같은 id끼리 뭉치게 된다. Flag는 3개의 bit로 되어 있으며 첫번째 비트는 0이며, 두번째 비트는 1일땐 fragmentation 지원한다는 것을, 0일땐 fragmentation을 지원하지 않는 다는 것을 의미한다. 세번째 비트는 1이면 뒤에 더 있다는 뜻이고 0이면 자신이 가장 마지막이라는 것이다, 즉 두번째 비트가 0인 것을 알 수 있다. Offset을 통해서는 자신이 몇번째 순서인지 알 수 있다. Offset이 0이라면 자신이 가장 앞 순서이며, start MTU가 끝나는 지점에서 8로 나눈 값이 자신의 offset이 된다. TTL은 몇 개의 홉을 거칠 수 있는지 알려준다. 이는 라우터를 건너뛸때마다 하나씩 감소된다. 만약 TTL이 없으면 주소가 불명확한 것은 인터넷에 계속 떠돌아다니게 된다. Protocol은 6이면 TCP, 17이면 UDP를 의미한다.

2.5 link layer 분석

2.5.1 ethernet 헤더 분석

데이터가 이동시에 여러 link와 router를 거치게 되는데 link의 종류는 다를 수 있다. 따라서

link는 reliable data transfer일 수도 있고 아닐 수도 있다. Link layer에서도 flow control, error detection, error correction 등의 서비스를 수행한다. Error detection을 위해서는 원래 datagram에 redundant 한 정보를 더 넣는다. Parity checking, checksum, cyclic redundancy check등이 있다. Link layer를 분석하기 위해서는 ARP에 대해 알아야 한다. ARP는 네트워크 계층의 프로토콜이지만 Link layer에게 필요하다. ARP는 ip 주소를 이용하여 MAC 주소를 알아내는 Protocol이다. 그렇다면 이것이 왜 필요할까? ip주소로 subnet이 있는 곳까지 도달하면 link layer가 ip주소를 가진 host까지 deliver해야 한다. 그러나 link layer는 MAC주소를 사용한다. 따라서 ip주소에서 MAC주소로 바꾸는 것이 필요하다. 이것이 ARP의 역할이다. 따라서 ip주소와 MAC주소의 매칭 테이블을 통해 이 역할을 수행한다.

ARP의 과정은 어떻게 될지 살펴보자.

1556	14.232854	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
1621	15.258254	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
1698	16.281550	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
5336	41.882493	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
5578	42.905149	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
11295	69.530065	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
11426	70.554133	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
15961	97.183817	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
16209	99.226330	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.179? Tell 192.168.200.254
43731	324.997510	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.142? Tell 192.168.200.254
48221	368.543934	Shenzhen_54:77:00	Broadcast	ARP	42	Who has 192.168.200.142? Tell 192.168.200.254

Step 1. 어떤 호스트가 다른 호스트의 ip주소를 가지고 ARP query를 브로드캐스팅 한다. 이때의 destination MAC주소는 알 수 없기 때문에 FF-FF-FF-FF-FF-FF형태로 LAN 상의 모든 노드들에게 발송을 한다.

```
Ethernet II, Src: Shenzhen_54:77:00 (94:fb:b2:54:77:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Destination: Broadcast (ff:ff:ff:ff:ff:ff)
> Source: Shenzhen_54:77:00 (94:fb:b2:54:77:00)
  Type: ARP (0x0806)
Address Resolution Protocol (request)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: Shenzhen_54:77:00 (94:fb:b2:54:77:00)
  Sender IP address: 192.168.200.254
  Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.200.179
```

패킷을 보면, destination은 broadcast이며, target MAC주소 또한 아직 없는 것이 보인다.

Step 2. B에서 ARP query를 받고 A에게 MAC주소를 담아 응답을 한다

Step 3. A에서는 응답을 받은 뒤에 테이블에 MAC주소와 ip주소를 매핑하여 저장한다. 이제 A는 MAC주소를 알았으니 link layer을 통해 전송할 수 있게 된다.

```
Ethernet II, Src: Shenzhen_54:77:00 (94:fb:b2:54:77:00), Dst: RivetNet_e5:ae:3b (9c:b6:d0:e5:ae:3b)
> Destination: RivetNet_e5:ae:3b (9c:b6:d0:e5:ae:3b)
> Source: Shenzhen_54:77:00 (94:fb:b2:54:77:00)
  Type: IPv4 (0x0800)
```

Ethernet II 헤더를 보면 위에서 말했듯 link layer는 MAC주소를 활용하기 때문에 도착지와 출발지의 MAC주소를 확인할 수 있다. 이더넷 타입을 보았을 때 0x0800이면 Ipv4, 0x0806이면 ipv6라는 것을 알 수 있다.

```
  Address: RivetNet_e5:ae:3b (9c:b6:d0:e5:ae:3b)
  .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
ource: Shenzhen_54:77:00 (94:fb:b2:54:77:00)
  Address: Shenzhen_54:77:00 (94:fb:b2:54:77:00)
  .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
  .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
```

IG bit는 멀티캐스트 통신인지 유니캐스트 통신인지 알 수 있다. 0으로 set되어 있고, individual address로 표시되어 있는 것으로 보아 유니캐스트 통신이라는 것을 알 수 있다. LG bit는 원래 부여된 주소와 다른 주소 여부를 확인할 수 있다. LG bit가 0임으로 NIC 의 MAC address가 원래 부여된 주소 그대로 사용하고 있음을 알 수 있다. 이것이 1이라면 도착지의 MAC address는 원래 부여된 주소가 아닌 다른 주소를 사용하고 있음을 알 수 있다.

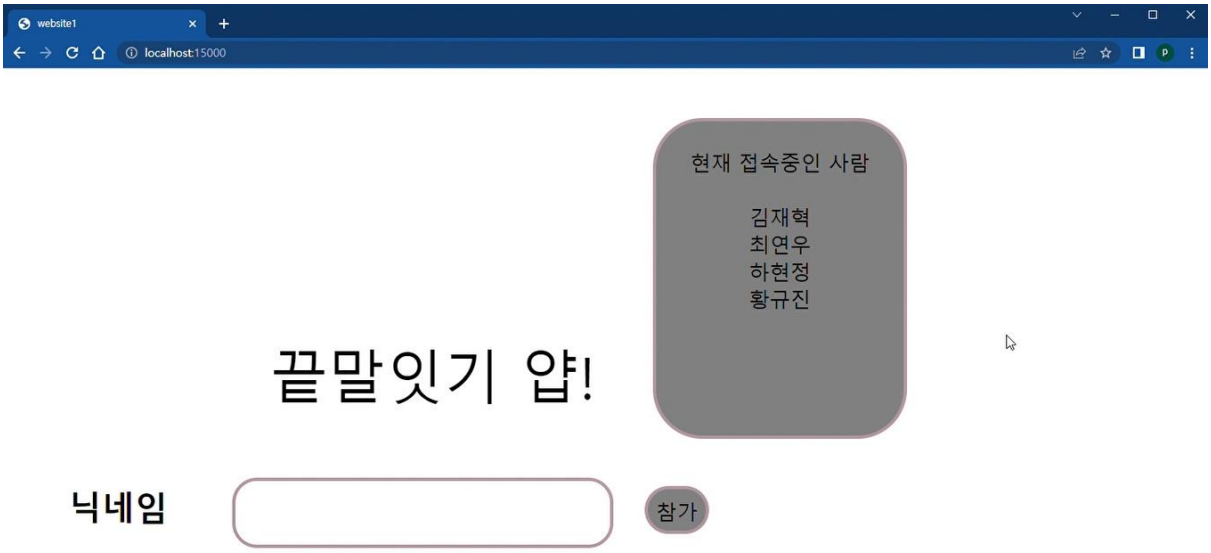
2.5.2 frame 헤더 분석

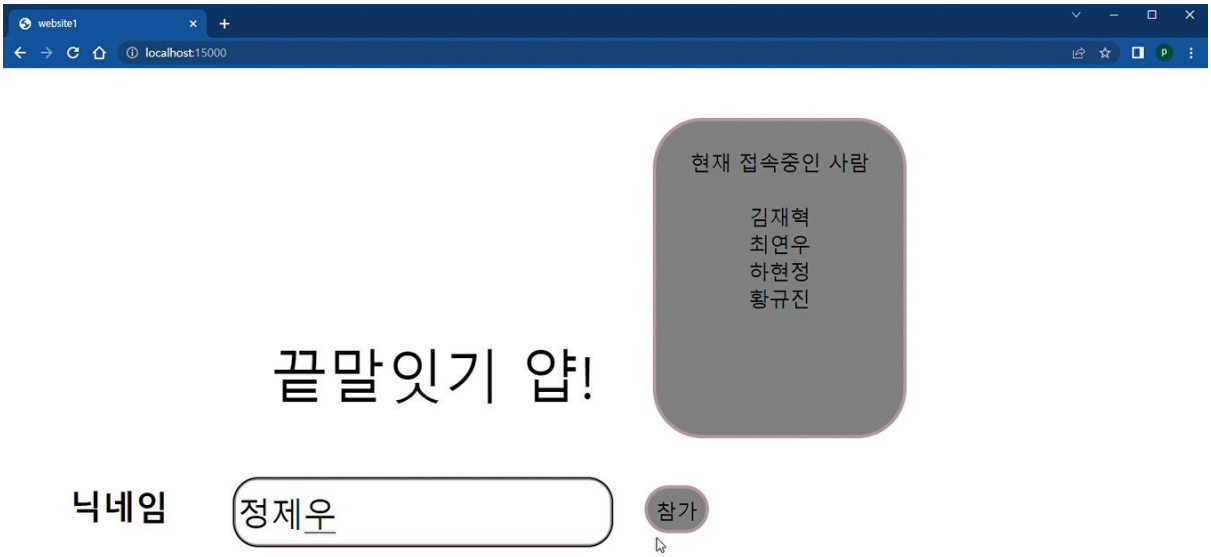
```
Frame 8584: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{
> Interface id: 0 (\Device\NPF_{C5568CCA-208A-4AE7-94E7-6EBCE7FF5D25})
  Encapsulation type: Ethernet (1)
  Arrival Time: Jun 13, 2022 21:43:40.658367000 대한민국 표준시
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1655124220.658367000 seconds
  [Time delta from previous captured frame: 0.000000000 seconds]
  [Time delta from previous displayed frame: 0.000000000 seconds]
  [Time since reference or first frame: 57.319532000 seconds]
  Frame Number: 8584
  Frame Length: 54 bytes (432 bits)
  Capture Length: 54 bytes (432 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ethertype:ip:tcp]
  [Coloring Rule Name: TCP]
  [Coloring Rule String: tcp]
```

Encapsulate type은 캡슐화의 종류를 나타낸다. 또한, frame number, frame length에서는 각각 캡쳐한 패킷의 번호 순서와 프레임의 크기를 알 수 있다. Frame is marked는 프레임이 손상되었는지의 여부를 확인할 수 있으며, frame is ignored는 프레임이 무시되었는지의 여부를 확인할 수 있다. Ehternet은 unreliable하고 connectionless이다. Ehternet의 MAC protocol은 CSMA/CD with binary backoff가 있다. (충돌이 발생할 때마다 더 큰 set에서 random하게 숫자를 정하여 기다리는 것)

여기까지 각 계층별 패킷 분석을 통해 분석해본 내용이다. 다음으로 실행 화면에 대한 내용이다.

3. 실제 실행 화면





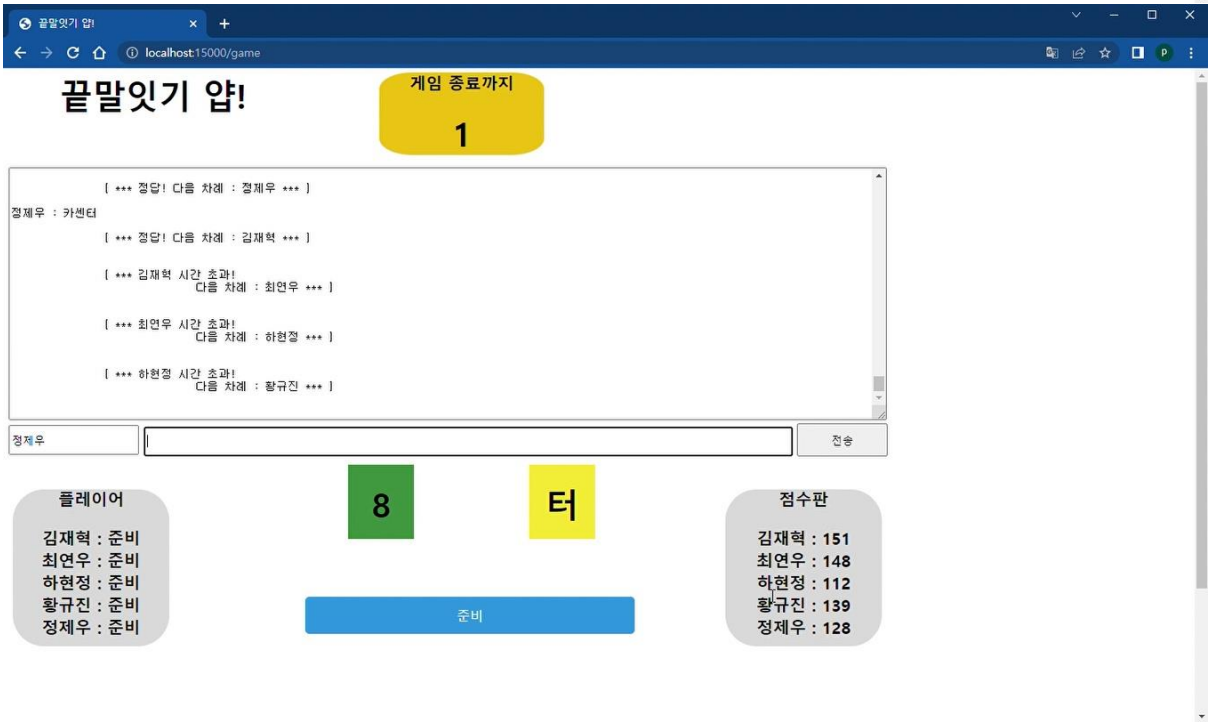
접속한 뒤 닉네임을 입력하고 참가를 누르면 현재 접속중인 사람에 이름이 뜸

9728	61.199246	175.126.156.162	192.168.200.157	HTTP	717	POST /game HTTP/1.1 (application/x-www-form-urlencoded)
9731	61.201633	192.168.200.157	175.126.156.162	TCP	1514	15000 → 64606 [ACK] Seq=2354 Ack=1171 Win=130048 Len=1460 [T
9732	61.201633	192.168.200.157	175.126.156.162	TCP	1514	15000 → 64606 [ACK] Seq=3814 Ack=1171 Win=130048 Len=1460 [T
9733	61.201633	192.168.200.157	175.126.156.162	TCP	1514	15000 → 64606 [ACK] Seq=5274 Ack=1171 Win=130048 Len=1460 [T
9734	61.201633	192.168.200.157	175.126.156.162	TCP	1514	15000 → 64606 [ACK] Seq=6734 Ack=1171 Win=130048 Len=1460 [T
9735	61.201633	192.168.200.157	175.126.156.162	HTTP	1412	HTTP/1.1 200 OK (text/html)

서버에 접속하기 위한 GET method와 OK 응답 패킷

처음엔 대기 상태이다가 준비를 누르면 준비로 상태로 바뀜

게임이 시작되면 밑의 노란색 박스의 글자로 시작하는 말을 채팅창에 입력
초록색 박스안의 숫자는 개인의 타이머를 의미함.

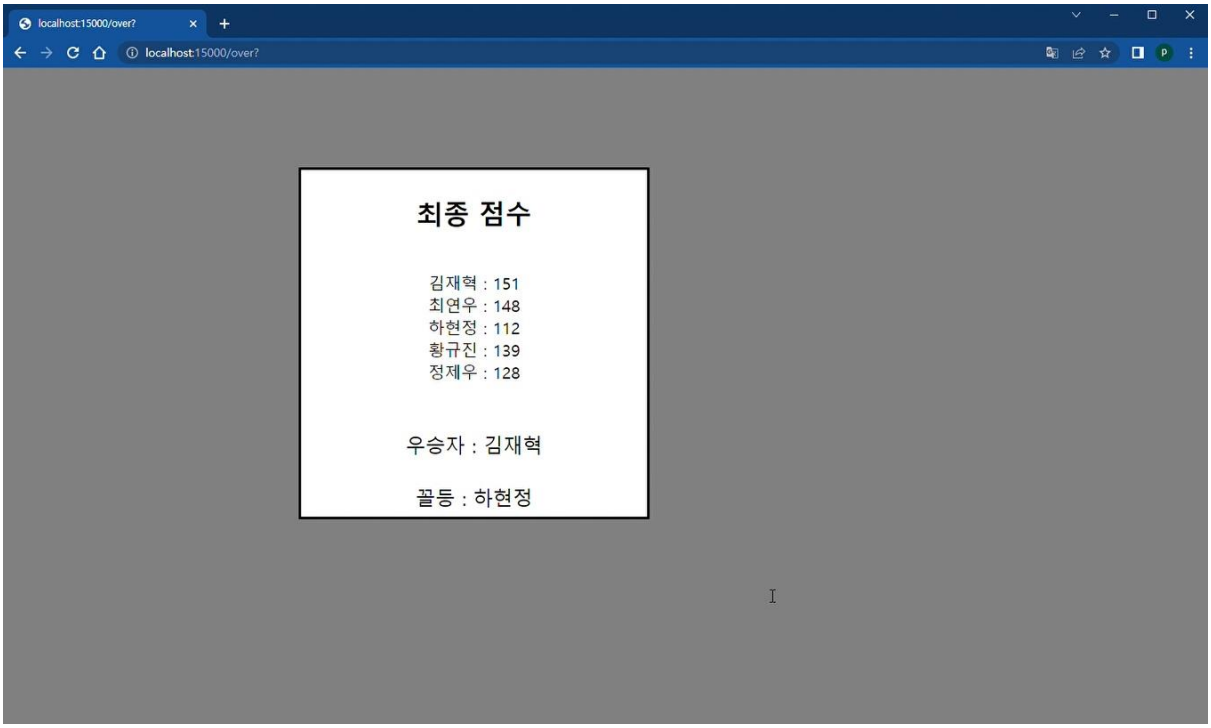


정답을 맞추거나, 시간이 초과되면 다음 차례로 넘어가게 됨.

단 정답을 맞춘 경우 점수를 받음.

```
Line-based text data (1 lines)
   42["response_message","황규진 : 국기"]
```

➔ 채팅 입력 시의 패킷에 황규진 님이 채팅에 친 국기가 뜨는 모습



마지막으로는 최종 점수가 뜨며 우승자와 꼴등이 나타남.
점수 기준 : 글자 수가 높울수록, 자신의 타이머 제한 시간 10초를 많이 남길수록 더 높은 점수를 얻음.

4. 결론

4-1 개인 기여율

패킷을 분석하고 이 팀프로젝트와 같이 서버를 구축하고 만드는 것을 한번도 해본 적이 없어 어려움이 많았다. 팀원들에게 최대한 피해를 끼치지 않기 위해 같은 조원들에게 묻기도 하고 스스로 공부를 많이 하려 노력하였다. HTML, CSS등의 언어를 스스로 공부해보며 프론트 제작을 맡았다. 또한, 분석 보고서 전체 내용을 회의를 통해 함께 수정하고 검토하였으며 패킷 캡쳐에 클라이언트로 참여하였다.

4-2 팀원 평가

김재혁 : 9. 프론트 제작
정제우 : 11. Node.js를 이용한 서버 제작, 제작된 웹페이지에 자바 스트립트 코드 삽입, 서버와 클라이언트 통신을 위한 웹 소켓 구현, 게임 알고리즘 구현
최연우 : 10. 한글처리 및 게임 알고리즘, node.js를 이용한 서버 제작
황규진 : 10. 서버, 클라이언트 상호작용 및 node.js를 이용한 서버 제작

4-3 프로젝트를 마치며

수업시간에 각각의 layer를 여러 주차에 거쳐 따로 배우다 보니 서로 연결점을 찾는 것이 어려워 각 계층끼리의 연결 동작들이 머리속에 정리가 잘 되지 않았다. 이번 프

로젝트를 한 뒤 패킷 분석을 하나씩 하면서 각 계층에 대한 역할, 계층 간의 역할 등에 대해 더욱 심도 있게 이해할 수 있게 되었다. 또한, 네트워크 수업시간과는 별개로 프론트 작업을 수행하기 위해 그에 필요한 언어를 독학하여 서버를 구축해본 것도 값진 경험이었다.