4주차

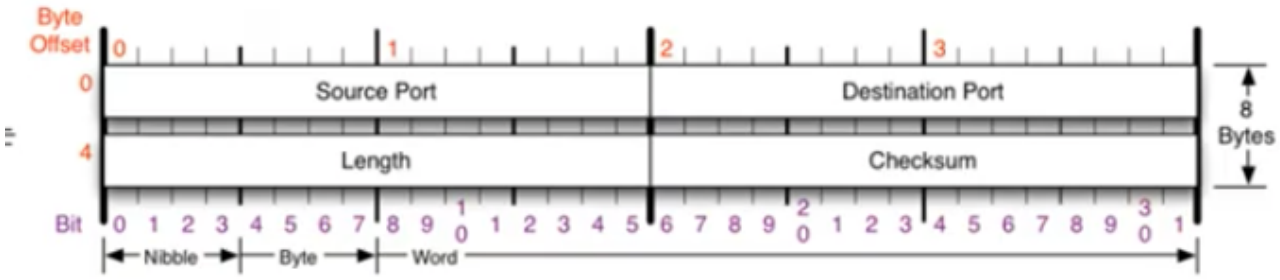
08. 비연결지향형 UDP 프로토콜 - 이론

https://www.youtube.com/watch?v=3MkI3FBFzX8

UDP의 역할

- 사용자 데이터그램 프로토콜(User Datagram Protocol, UDP)
- 유니버설 데이터그램 프로토콜(Universal Datagram Protocol)이라도 함.
- UDP의 **전송 방식은 너무 단순**해서 서비스의 신뢰성이 낮고,
- 데이터그램 도착 순서가 바뀌거나, 중복되거나, 심지어는 통보 없이 누락시키기도 함.
- UDP는 일반적으로 **오류의 검사와 수정이 필요 없는** 프로그램에서 수행할 것으로 가정함.
- 상대방과 연결된 상태를 지향하지 않음 → 연결된 상태에서 데이터를 주고받는 것이 아니므로 신뢰성이 낮음.
- 안전한 연결을 지향하지 않음.

UDP 프로토콜의 구조



- Source Port(2byte)
- Destination Port(2btye)
- 길이(header + payload, 2byte)
- Checksum(2byte)

UDP 프로토콜을 사용하는 대표적인 프로그램

- 도메인을 물으면 IP를 알려주는 **DNS 서버**
- UDP로 파일을 공유하는 **tftp 서버**
- 라우팅 정보를 공유하는 **RIP 프로토콜**

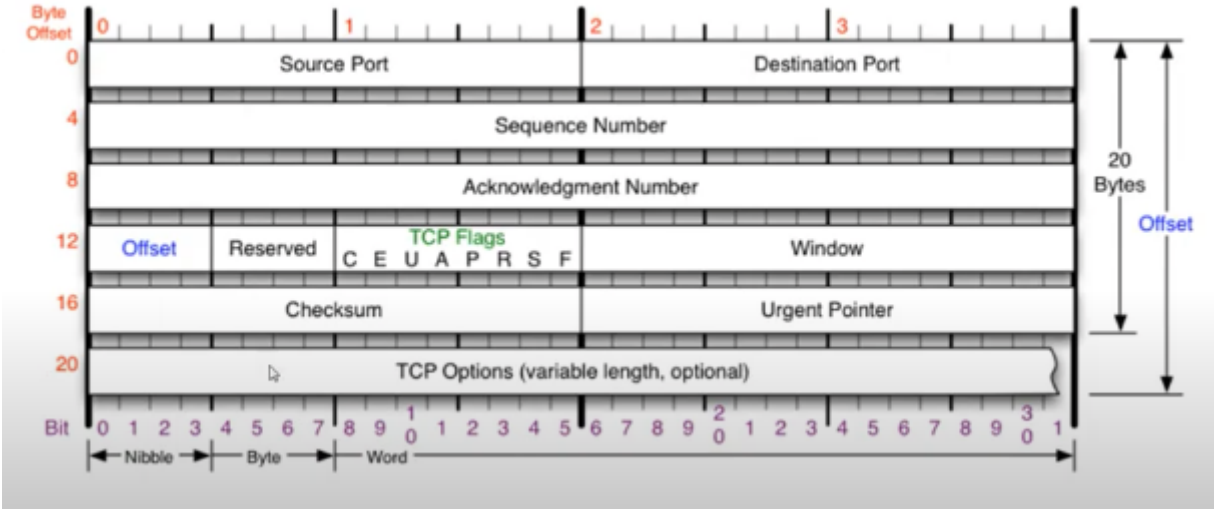
09. 연결지향형 TCP 프로토콜 - TCP 프로토콜 구조와 TCP의 플래그

[연결지향형 TCP 프로토콜 - TCP 프로토콜 구조와 TCP의 플래그]

TCP의 역할

- 전송 제어 프로토콜(Transmission Control Protocol, TCP)
- 인터넷에 연결된 컴퓨터에서 실행되는 프로그램 간에 통신을 **안정적으로, 순서대로, 에러 없이** 교환할 수 있게 함.
- TCP의 안정성을 필요로 하지 않는 애플리케이션의 경우 일반적으로 TCP 대신 비접속형 사용자 데이터그램 프로토콜(User Datagram Protocol)을 사용함.
- TCP는 UDP보다 여러 기능을 넣다보니 안정적이지만 느림.(그러나 우리가 체감할 수 없을 정도임.)

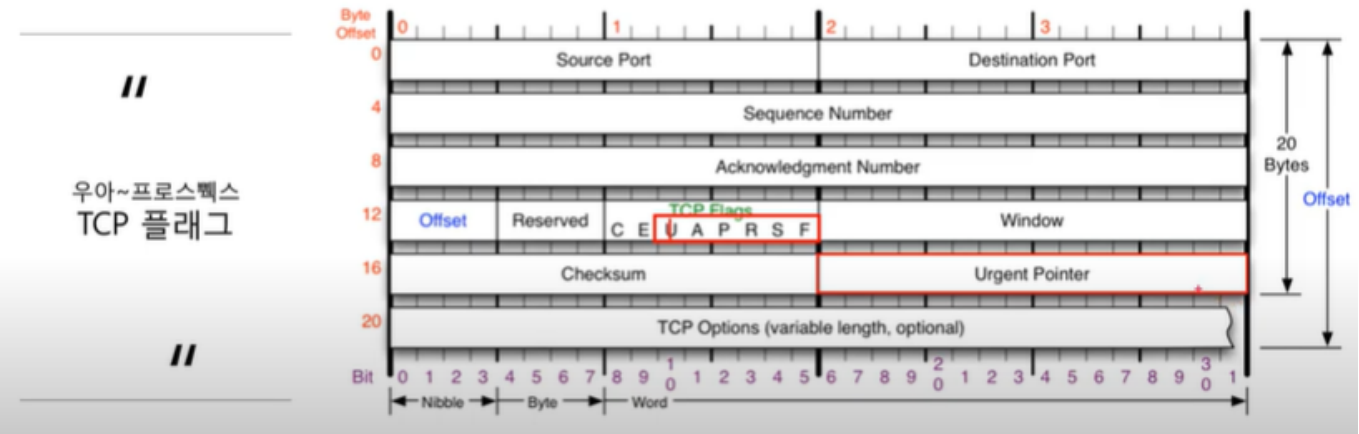
TCP 프로토콜의 구조



- Source Port: 2 byte - 출발지 포트
- Destination Port: 2 byte - 목적지 포트
- Sequence Number: 4 byte -
- Acknowledgement Number: 4 byte
- Offset: 헤더의 길이
- Reserved: 예약된 필드로 사용하지 않는 필드
- Checksum
- Window: 내 사용 공간이 얼마나 남아있는지 상대방에게 알려주는 필드로, 나의 남아있는 TCP 버퍼 공간을 알려줌(상대방이 나한테 얼만큼 더 보낼 수 있는지)
- Usgent Pointer
- TCP Option: 일반적으로 잘 안 붙고 붙더라도 4byte씩 붙으며 총 10개까지 붙을 수 있음
- 가장 일반적인 길이는 20byte이나, 최대 60byte까지 늘어날 수 있음.

TCP 플래그

TCP 플래그의 종류



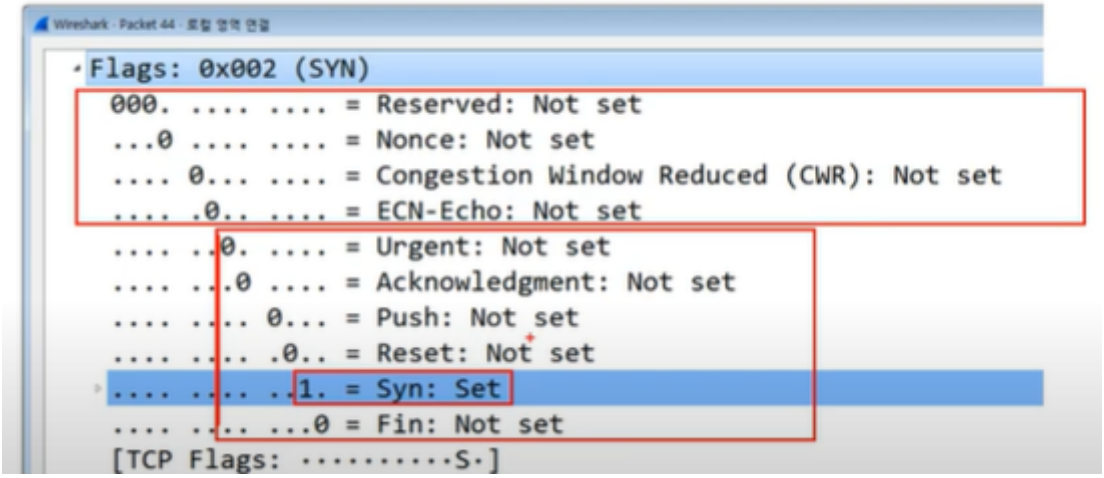
C E U A P R S F

TCP는 통신 과정에서 지속해서 상대방과의 연결 상태를 물어보는데, 어떤 플래그를 보내는지에 따라 TCP의 기능이 달라짐.

각 플래그의 기능

TCP 제어 플래그

- U flag
  - urgent Flag(긴급비트)로, 우선순위가 높은 데이터가 포함되어있을 때 지정
  - U가 1로 세팅되어있으면 빨리 처리해줌.
  - Urgent Pointer는 어디부터 긴급 데이터인지 알려주는 위치 값(잘 몰라도 상관x)
- A flag
  - ACK Flag(승인 비트)로, TCP에서 중요하고 많이 사용됨.
  - 연결해도 되는지 승인해주는 비트로 데이터를 보내도 되는지 확인 응답을 받는 필드
- P flag
  - Push Flag로, TCP 버퍼가 일정한 크기만큼 쌓여야 패킷을 추가적으로 전송하는데, 버퍼 상관없이 계속해서 데이터를 밀어넣겠다는 표현
  - 많이 사용하지 않음
- R flag
  - Reset Flag로, 상대방과 연결이 되어있는 상태에서 추가적으로 데이터를 주고받을 때 문제가 발생해서 둘 사이의 연결관계를 reset 하는 Flag
- S flag (제일 중요)
  - SYN Flag(싱크 비트)로, 동기화 플래그
  - 상대방과 연결을 시작할 때 무조건 사용하는 플래그로써, 해당 비트가 처음 보내지고 난 이후부터 둘 사이의 연결이 서로 동기화되기 시작함
  - 상대방과 상태를 계속 주고받으면서 상태를 동기화함.(보내도 돼?→보내도 돼 / 잘 받았어?→잘 받았어)

- F flag
  - Fin Flag로 종료 플래그
  - 연결을 끊을 때 사용하는 플래그
- 

- 위에 네 개는 잘 안씀. 아래 쪽 네모는 자주 씀.

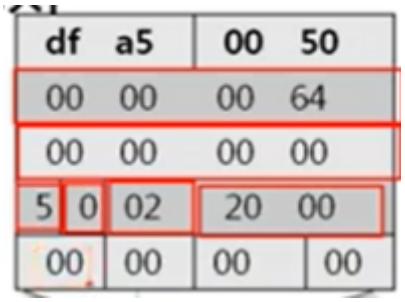
TCP를 이용한 통신 과정

연결 수립 과정

TCP를 이용한 데이터 통신을 할 때 프로세스와 프로세스를 연결하기 위해 **가장 먼저! 무조건! 수행되는 과정**
통신을 하려고 할 때 가장 먼저 수행되는 과정이며, 이 세 과정이 모두 수행되고 난 이후에 데이터가 전달이 되기 시작함.

3Way Handshake

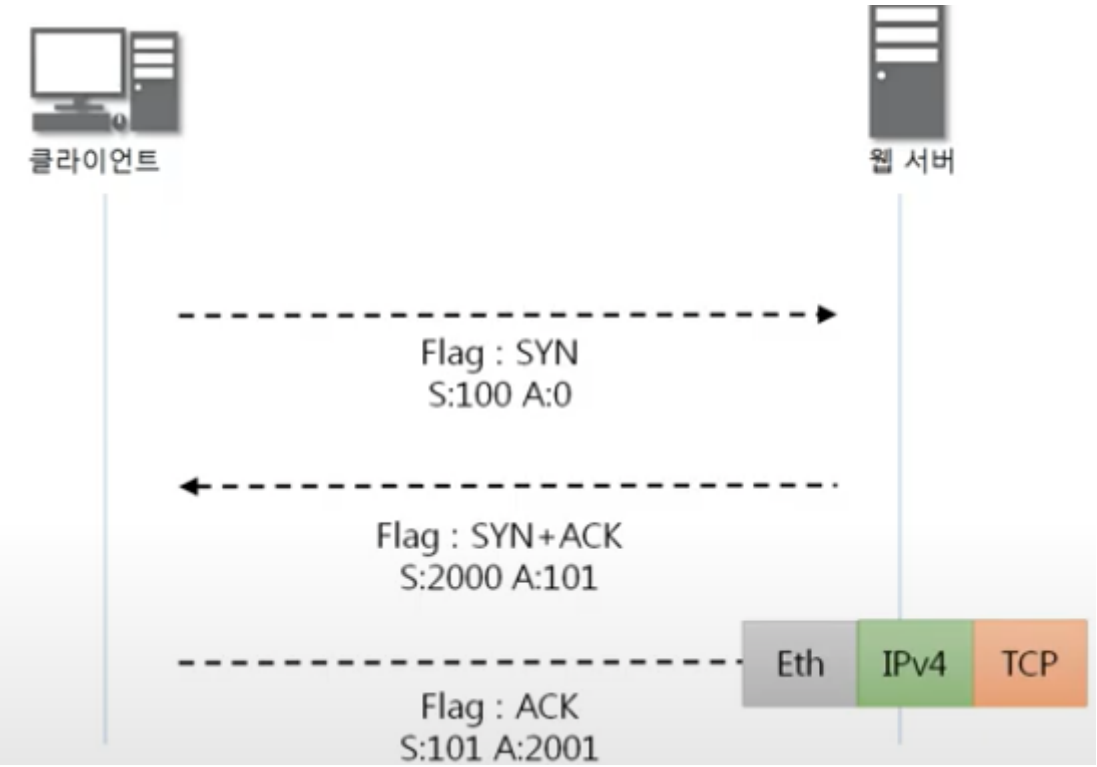
```
1. 클라이언트가 서버에게 연결해도 되는지에 대한 요청 패킷을 보내고,
2. 서버가 클라이언트의 요청을 받아들이는 패킷을 보내고,
3. 클라이언트는 이를 최종적으로 수락하는 패킷을 보낸다.
위의 3개의 과정을 3Way Handshake라고 부른다.
```



5: 헤더길이(tcp 20바이트 / 4 =>5)
0: reserved (사용안함.)

02: flag (sync 플래그)
20 00: window

3way Handshake



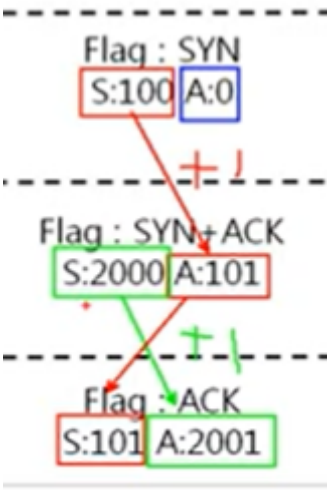
1. 클라이언트는 Ethernet, IPv4, TCP 인캡슐레이션을 해서 서버로 보냄.
   (연결을 수립하는 과정이므로 TCP 뒷쪽에 payload는 없고, TCP의 출발지 포트번호는 모르므로 사용자 포트번호 중 아무거나 사용함. 목적지 포트번호는 웹으로 요청하므로 80번 사용, SYN flag와 ACK flag 포함하여 보냄)
2. Flag에는 SYN 세팅, S에는 100 A에는 0을 세팅하여 보냄
3. 서버가 요청을 받았으면 디캡슐레이션을 통해 내용을 확인
4. 서버에서 출발지 포트 번호는 서버의 80번 포트이고, 목적지 포트는 클라이언트가 요청했던 포트로 설정하여 보냄, Flag는 SYN와 ACK가 함께 세팅되어 클라이언트로 전달.
5. Sequence 번호는 2000, Acknowloge 번호는 101로 세팅
6. 클라이언트에서는 ACK Flag만 세팅해서 서버로 다시 보냄.
7. Flag에는 ACK 세팅, Sequence에는 101, Ack에는 2001 세팅하여 보내줌
   → 이 과정이 3way handshake임
   → 이 과정 이후에 클라이언트가 서버에 요청을 보내기 시작하는 것.

처음에 클라이언트가 Sequence 번호를 100번, ACK 번호를 0번으로 세팅하여 보냈다고 가정한다.(Sequence 번호는 보통 랜덤한 값이 부여됨) 받는 쪽에서 해당 값과 동기화 시킨다. 받는 쪽(서버)에서 ACK 번호는 받은 Sequence 번호 + 1을 하고, Sequence 번호에는 랜덤한 값을 하나 생성한다. 예제로 Sequence 번호에는 2000을 세팅하고 현재 ACK 번호로는 101이 세팅되어있

다. 다시 받는 쪽(클라이언트)에서 ACK 번호는 받은 Sequence 번호 + 1이므로 2001세팅되고, Sequence 번호로는 처음 보내는 값이 아니기 때문에 동기화가 되어 받은 ACK 번호가 된다.

보안에 관심이 있다면...
➡ 이 번호 계산을 통해 세션 하이재킹 공격이나 DOS 공격 등을 할 수 있다.

***세션 하이재킹:**
연결된 후에, 클라이언트가 요청을 보냈을 때, 클라이언트가 아닌 누군가가 동기화된 값을 계산하여 그대로 서버로 보내면, 클라이언트 대신 그 다른 누군가와 통신하기 시작함.

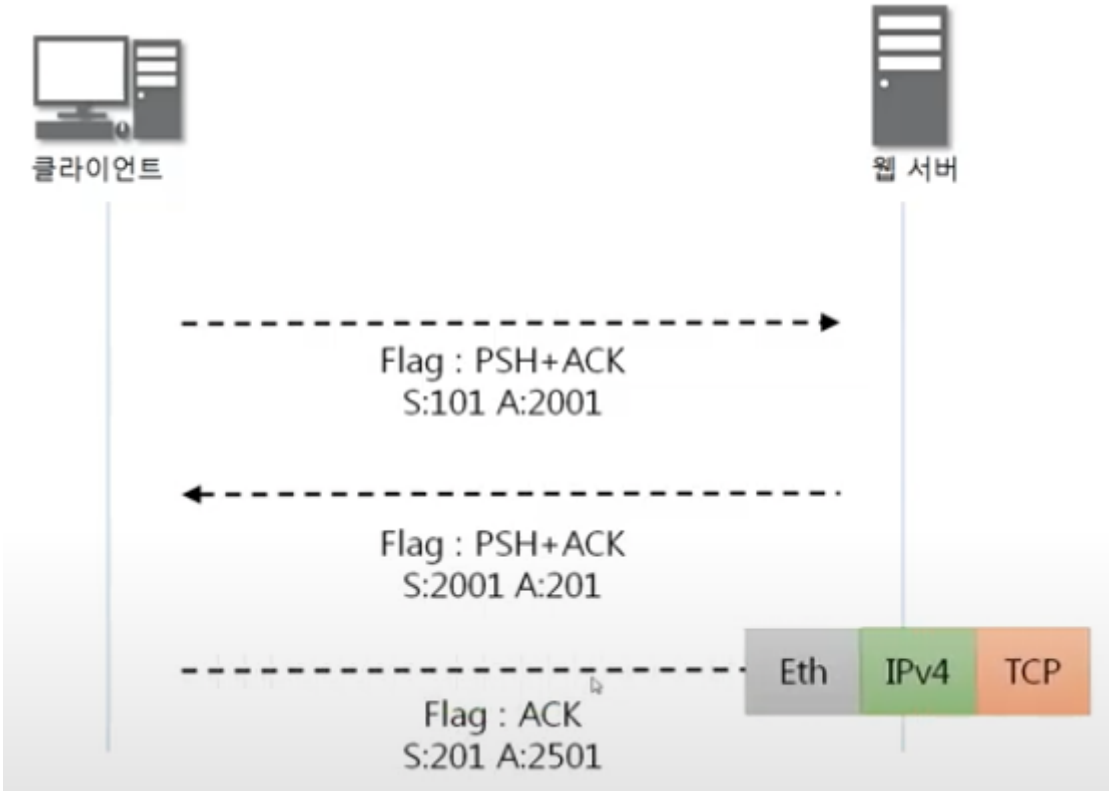


A를 시퀀스(S) 번호 +1로 설정(100→101, 2000→2001)
S는 받은 A번호로 설정((처음에는 랜덤설정→S:2000, A:101→S:101)

데이터 송수신 과정

- TCP를 이용한 데이터 통신을 할 때
  단순히 TCP 패킷만을 캡슐화해서 통신하는 것이 아닌
  페이로드를 포함한 패킷을 주고받을 때의 일정한 규칙
- (아까는 데이터를 주고받는게 아니므로 페이로드 존재x
  데이터를 보낼때는 페이로드를 포함한 패킷을 주고받아야 함.)
- 연결이 수립된 상태에서 데이터를 송수신할 때에는 연결이 수립되었을 때의 SEQ 번호와 ACK 번호를 그대로 이어간다.
- 마지막에 클라이언트가 서버로 패킷을 보내고 끝나는데,(연결된 상태) 요 상태에서 클라이언트가 다시 데이터를 보냄.(데이터 송수신 상태)

1. 보낸 쪽에서 또 보낼 때는 SEQ 번호와 ACK 번호가 그대로
2. 받는 쪽에서 SEQ번호는 받는 ACK 번호가 됨.
3. 받는 쪽에서 ACK번호는 받는 SEQ번호 + 데이터의 크기

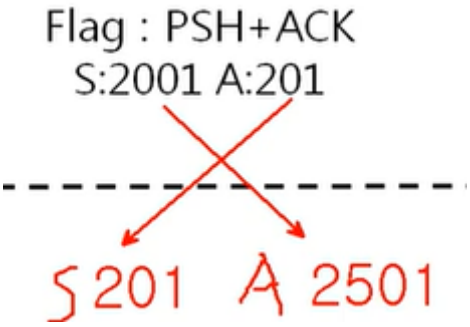


1. 데이터를 보내기 때문에 Push나 ACK 등을 세팅해서 인캡슐레이션 하여 보냄.
2. SEQ 번호와 ACK 번호는 연결할 때 사용했던 것 그대로 사용.
3. 디캡슐레이션한 다음 데이터 내용을 보고, 다시 클라이언트로 데이터를 보내면서 ACK를 포함시켜 보내줌.
4. ACK번호는 받은 SEQ번호 + 데이터의 크기 로 지정되며, SEQ 번호는 받은 ACK 번호 그대로 사용함.
5. 최종적으로 원하는 데이터를 받았을 시에 SEQ번호는 ACK번호 그대로 사용하고, ACK 번호는 SEQ번호 + 데이터 크기로 설정됨.

연결을 끊는 방식은 TCP를 사용하는 프로그램을 만드는 곳에 따라 다르며, ACK와 FIN을 함께 보내기도 하고 따로 보내기도 함.

데이터 크기: 100

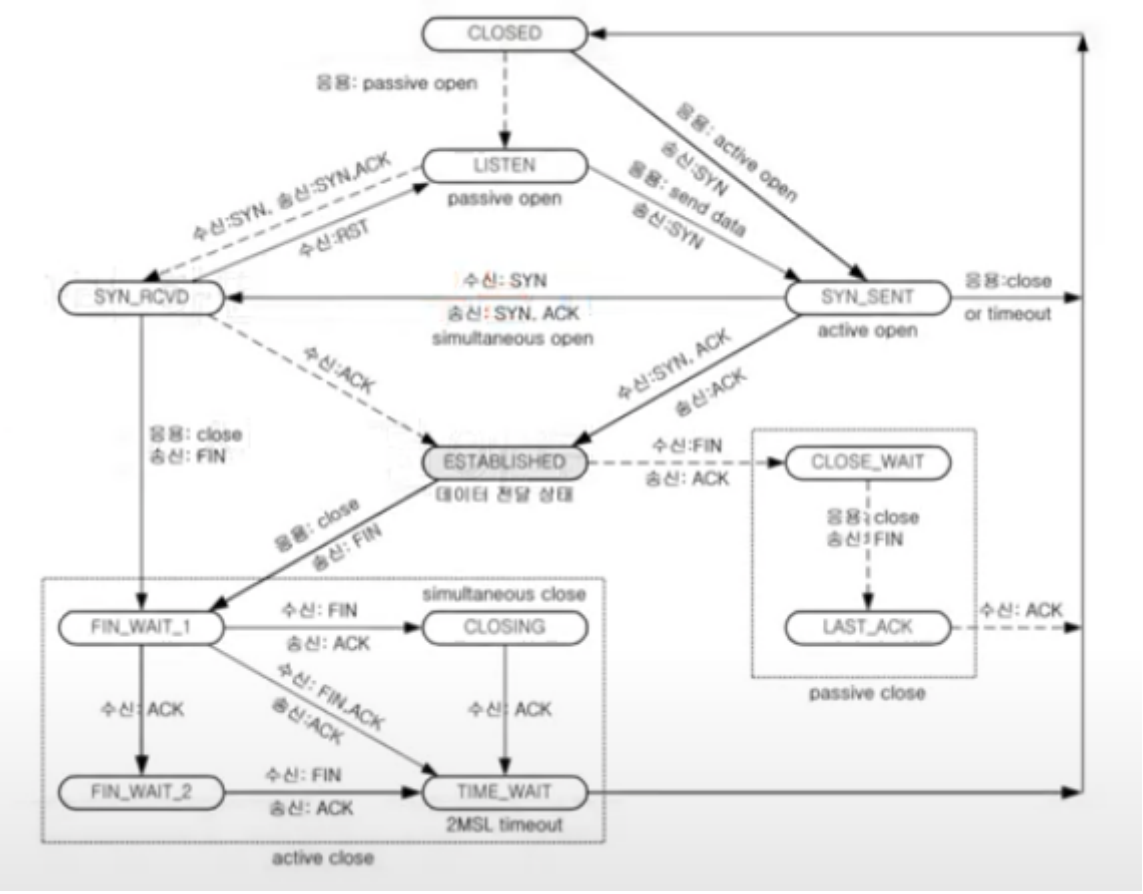
S→A(+100), A→S(그대로)



S→A(+500), A→S(그대로)

TCP 상태전이도

TCP 연결 상태의 변화



각자 보내는 패킷에 따라 상태가 변화하는 과정을 그려놓은 그래프

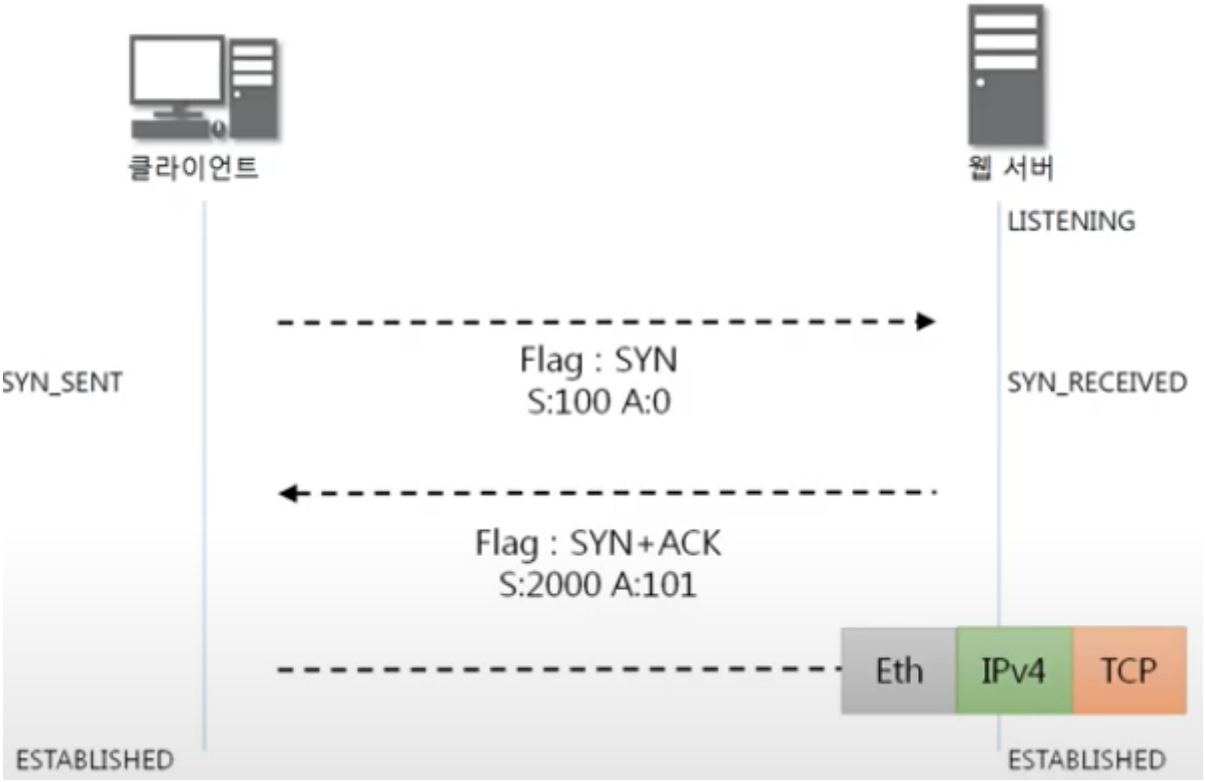
- 실선: 클라이언트의 상태 변화
- 점선: 서버의 상태 변화
- 여기서 `LISTEN`과 `ESTABLISHED` 두 개가 중요하다.
- `LISTEN`상태에서 `ESTABLISHED` 상태가 돼야 데이터를 주고받을 수 있는 상태가 된 것.

- LISTEN
  - 4계층은 포트 번호를 사용하는데, 포트 번호를 열어 놓고 있는 상태를 말한다.
  - 서버가 클라이언트의 요청을 계속 듣고 있는 상태.
  - 서버가 포트를 열어서 LISTEN상태로 만들 때 passive open
- ESTABLISHED
  - 연결이 서로 수립이 된 상태를 말한다.
  - 3 way handshake 상태가 끝나면 ESTABLISHED 상태가 되어 통신이 가능해진 것.
- CLOSED
  - 클라이언트도 본인의 포트를 사용하고 원래는 포트가 닫혀있는 상태
  - 클라이언트가 포트를 사용할 때 active open
- SYN_SENT
  - 클라이언트가 포트를 열면서 SYN 패킷을 flag 세팅하여 서버로 전달
- SYN_RCVD

- SYN를 받은 서버는 SYN_RCVD 상태가 되고, SYN, ACK 패킷을 보내게 됨

*몰라도 됨
active open: 능동적으로 포트를 여는 애 = 클라이언트
passive open: 수동적으로 가만히 있는 애 = 서버

복잡하니까 3WayHandshaking과 함께 보면..



1. 클라이언트가 패킷을 만들어서 서버로 보냄. 이 때 active open하게 된 것이고, 동시에 SYN_SENT 상태가 됨
2. 서버는 LISTEN 상태여야 해당 요청을 받을 수 있고, 패킷을 받았기 때문에 SYN_RECEIVED 상태가 된다.
3. 서버는 패킷을 만들어 클라이언트로 보내면 서버는 SYN_RECEIVED 상태 그대로임.
4. 클라이언트가 ACK 패킷을 서버로 보내게 되면 클라이언트는 ESTABLISHED 상태가 되고, 해당 패킷을 받은 서버에서도 ESTABLISHED 상태가 됨.

→ 이제 데이터를 주고받을 수 있는 상태가 된 것.

널널한 개발자

[이해하면 인생이 바뀌는 TCP 송/수신 원리 - YouTube]

웹 서버는 소켓을 통해 클라이언트와 통신한다.
프로세스가 소켓 파일에 대해 기본적으로 할 수 있는 기능은 rwx가 있다.

r(read): 읽기
w(write): 쓰기 = send
x(execute): 실행

- 먼저 클라이언트가 웹 서버에게 A 파일을 요청해서 웹 서버가 해당 데이터를 전달해주는 과정을 생각해보자.
- 제일 먼저 웹 서버는 하드디스크로부터 A 파일을 읽어 들인다.
- 웹 서버의 프로세스-소켓 수준에는 할당되어 있는 버퍼 메모리가 있다. (메모리의 크기는 개발자가 결정)
- 버퍼 메모리의 크기는 A 파일의 전체 크기보다 작고, 버퍼 메모리의 크기에 맞게 A 파일을 분할해서 하나씩 읽어 들인다.
- 웹 서버의 프로세스가 하드디스크로부터 A 파일을 모두 읽어 들인 뒤, 다시 하위 계층으로 보내면서 클라이언트의 프로세스로 A 파일 데이터가 전달된다.
- 하위 계층인 TCP 수준의 버퍼 메모리에 옮겨질 때에 데이터는 segment 단위로 쪼개진다. segment 단위로 쪼개질 때에는 하나의 segment마다 일련번호가 붙게 된다.
- 다음 하위 계층인 IP 수준으로 내려오면서 데이터는 packet 단위가 된다.
- 또 다음 하위 계층인 NIC 수준으로 내려오면서 데이터는 프레임 단위가 된다.
- `세그먼트 = 내용물 | 패킷 = 택배상자 | 프레임 = 택배 트럭`으로 생각하자.
- 하나의 프레임 씩 클라이언트 측으로 전달되면서 이번에는 반대로 하위 계층에서부터 상위 계층으로 데이터가 전달된다.
- 웹 서버에서 진행됐던 순서가 반대가 되면서 `프레임 → 패킷 → 세그먼트` 순서로 데이터가 다시 분리되고, segment는 하나 씩 클라이언트의 TCP 메모리에 쌓이게 된다.
- 보통 segment 2개 정도가 쌓이면 `수신측`에서 `송신측`으로 ACK 3번 신호를 보낸다.
- `송신측`은 해당 신호를 받으면 `수신측`이 1~2번 세그먼트까지 잘 받았구나, 이어서 3번 세그먼트를 보내면 되겠구나' 라고 판단한다.
- `송신측`은 `수신측`으로부터 ACK 신호를 받을 때까지 기다리고 있는데, 이 기다리는 시간 때문에 속도 지연이 발생한다.
- 특히 중요한 것이 하나 더 있다.
- `수신측`의 TCP 버퍼에 남아있는 메모리 크기를 `Window size` 라고 하는데, ACK 신호를 보낼 때 이 `Window size` 정보도 같이 보낸다.
- 송신측은 ACK 신호를 받은 뒤, 윈도우 사이즈를 보고 3번 segment를 보낼지 말지 판단한다.
- 수신측의 `Window size`가 보내려는 segment의 `MSS(Maximum Segment Size)`보다 크면 보내고, 작다면 보내지 않는다.
- `Window size`가 MSS보다 작다는 것은 수신측에서 데이터를 받을 여유 공간이 없다는 것을 뜻하기 때문이다.
- 이 `Window size`가 여유로워질 때까지 `송신측`은 데이터를 보내지 않고 기다리게 되고, 여기서도 속도 지연이 발생하게 된다.

- `Window size`가 여유가 생기려면 TCP 버퍼에 쌓여있던 데이터들을 소켓 파일 I/O 버퍼 메모리에 올려야 하는데, 이 속도가 네트워크에서 데이터를 수신하는 속도보다 빨라야 `송신측`이 기다리는 시간이 짧아진다.
- 네트워크의 지연을 파악하려면 가장 먼저 프로그램의 TCP 버퍼에서 File I/O 버퍼로 올라가는 속도를 파악해야 한다.