2장 TCP/IP의 데이터를 전기 신호로 만들어 보낸다.

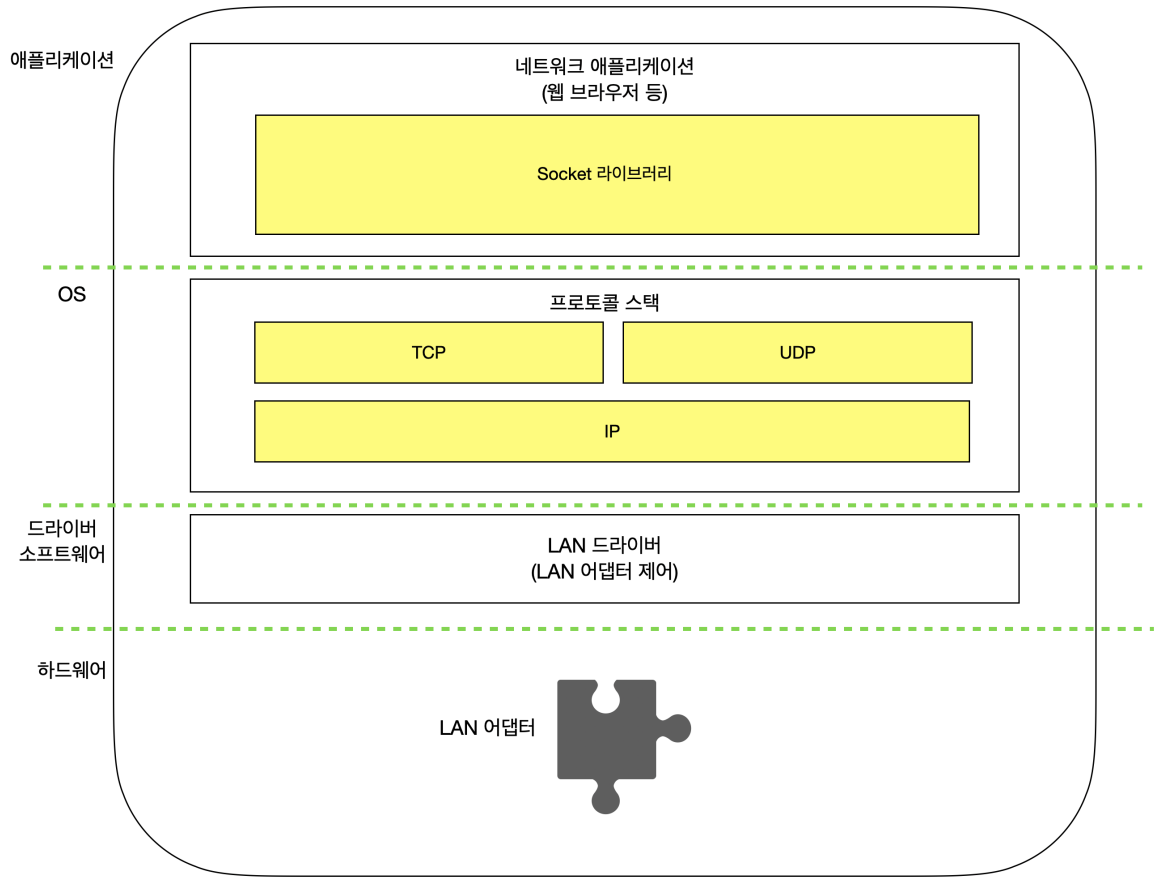
분류	성공과 실패를 결정하는 1% 네트워크 원리
생성일시	@2022년 6월 6일 오후 2:39
최종편집일시	@2022년 6월 10일 오전 10:51
다이어리 기록	
참고	1) https://github.com/Stacked-Book/BookRecord 2) https://developer.mozilla.org/ko/docs/Web/HTTP/Methods 3) http://www.ktword.co.kr/test/view/view.php?nav=&m_temp1=1889&id=1103 4) https://www.joinc.co.kr/w/Site/Network_Programing/Documents/IntroTCPIP
태그	스터디그룹

▼ **목차**

소켓을 작성한다.

1. 프로토콜 스택의 내부 구성

- 소프트웨어 - (OS내장) 프로토콜 스택
- 하드웨어 - LAN 어댑터



계층 구조

> 애플리케이션 아랫 부분에는 Socket 라이브러리가 있으며 그 안에는 리졸버가 내장돼 있다.
> 프로토콜 스택의 윗부분에는 TCP라는 프로토콜을 사용하여 데이터 송수신을 담당하는 부분과 UDP라는 프로토콜을 사용하여 데이터를 송수신을 담당하는 부분이 있다.
> **이 둘이 애플리케이션에서 보낸 의뢰를 받아 송수신 동작을 실행한다.**

- 애플리케이션에서 데이터를 송수신 시작한다.
  이때 소켓 라이브러리를 사용하여 리졸버로 DNS 서버를 조회하는 등의 동작을 수행한다.
- OS 내부에 있는 프로토콜 스택이 그 다음 작업을 의뢰 받는다.
  - TCP 혹은 UDP로 데이터를 송수신 한다.
  - IP 프로토콜로 패킷 송수신 동작을 제어한다.
    ICMP(패킷 운반시, 오류 통시, 제어용 메시지) 혹은 ARP(IP에 대응하는 MAC주소 조사)로 동작한다.
- LAN 드라이버는 LAN 어댑터라는 하드웨어를 제어한다.
  LAN 어댑터라는 하드웨어가 실제 송수신 동작, 케이블 신호 송수신 동작을 제어한다.

❓ **TCP, UDP는 같은 역할을 하는거 같은데 TCP와 UDP는 무슨 상황에 사용하는 걸까?**

- 브라우저나 메일 등의 **일반적인 애플리케이션이 데이터**를 송수신할 경우는 **TCP**
- **DNS** 서버에 대한 **조회 등에서 짧은 제어용 데이터**를 송수신할 경우에는 **UDP**

2. 소켓의 실체는 통신 제어용 제어 정보이다.

> 소켓 내부에 제어 정보를 기록하는 메모리 영역이 존재한다.
> 이 곳에 통신 동작 제어용 정보를 기록하는 역할을 한다.

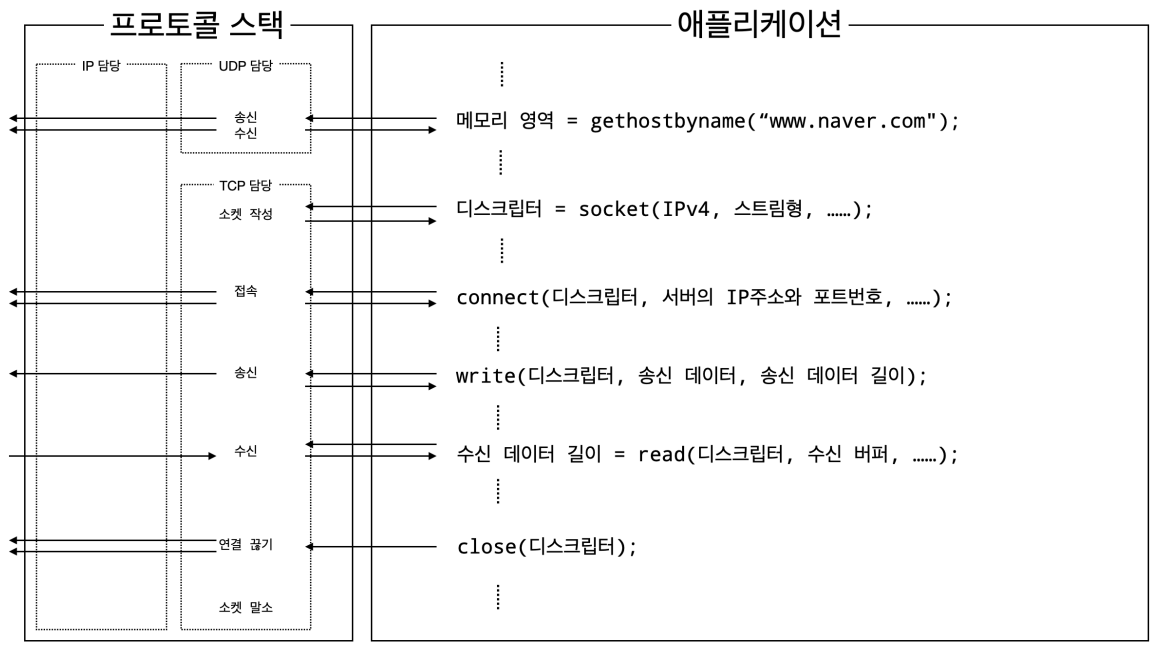
- 통신 상대의 IP주소, 포트번호
- 통신 동작이 어떤 진행 상태에 있는가

> 🌟 **소켓은 개념적인 것이어서 실체가 없다.**
> 제어 정보가 소켓의 실체라고 할 수 있다. 즉 제어 정보를 기록한 메모리 영역이 소켓의 실체라고 생각하면 된다.

- 프로토콜 스택은 이 제어 정보를 참조하면서 동작한다.

소켓을 만든다는 동작은 제어 정보를 추가하고 상태를 기록하거나 송수신 데이터를 일시적으로 저장하는 버퍼 메모리를 준비하는 등 통신을 준비하는 작업이다.

3. Socket을 호출했을 때의 동작



메시지 송신 동작

> socket이나 connect은 Socket 라이브러리의 프로그램 부품이다.
> 호출했을 때 프로토콜 스택의 내부의 움직임을 보자.

- 먼저 `socket` 메서드를 호출하여 프로토콜 스택에 의뢰해 소켓을 하나 생성한다.
  - 프로토콜 스택은 소켓 한 개가 사용하는 메모리 영역을 확보한다.
  - 아직 통신 이전이며, 초기 상태의 메모리를 기록한다.
  - 프로토콜 스택은 소켓에 대한 디스크립터를 어플리케이션에 알려준다.
  - 이 디스크립터를 통해 어플리케이션은 이후 데이터 송수신을 프로토콜 스택에 의뢰한다.
  - 디스크립터만 있으면 프로토콜 스택이 소켓의 통신 상태, 상대 소켓 등등의 기타 정보를 모두 알 수 있다.

    → **애플리케이션은 해당 정보에 대해서 알 필요가 없다.**

TCP 추가 공부
TCP 추가 공부 2

서버에 접속한다.

> 🌟 **접속이란**
> 애플리케이션은 소켓 생성 후 `connect` 메서드를 호출하여 접속 동작을 시작한다.
> 여기서 말하는 접속 동작은 케이블을 연결해주는 접속 동작이 아니다.
> 주로 필요한 회선 및 케이블은 이미 연결 돼 있다.

❓ 그럼 접속이 무엇을 말하는건데?

> 데이터를 주고받고자 하는 대상과 필요한 정보를 주고 받아서 기록하고 데이터 송수신이 가능한 상태로 만드는 것

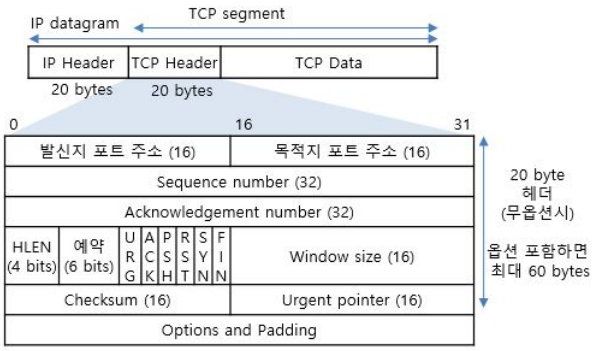
- **초기상태에는 아무것도 기록돼 있지 않으므로 상대 소켓에 대한 IP 주소와 포트번호에 대한 정보가 필요하다.**

  : 1장에서 브라우저는 URL을 바탕으로 서버의 IP를 조사하고 포트번호도 정해진 규칙으로 사용하도록 정해져 있으므로 필요한 정보를 알고 있다. 따라서 브라우저는 이러한 정보를 프로토콜 스택에 알리는 동작이 필요한데 ,이것이 접속동작의 한가지 역할이다.

- connect 메서드는 어플리케이션이 알고 있는 상태 소켓의 IP주소(DNS 리졸버를 통해서 획득한 정보) 및 포트 번호(사용자가 이미 알고 있는 정보)를 프로토콜 스택에 알려 소켓에 기록하는 로직을 수행한다.
- 대상 소켓 B도 소켓을 생성한 후에 자신과 접속하고 하는 소켓 A를 알지 못하므로 소켓을 생성하고 소켓 A가 접속을 원한다는 요청을 받기 이전까지 대기한다. 요청을 받으면 소켓 B도 A의 정보를 기록하여 통신할 수 있게 된다.

소켓에 접속하는 connect 메서드 수행 시 데이터 송수신 메머리 버퍼도 확보한다.

1. 소켓 앞에 제어 정보를 기록한 헤더를 배치한다.



참고 링크(3)

제어정보

- **클라이언트와 서버가 서로 연락을 절충하기 위해 주고 받는 제어 정보**

  접속 동작뿐만 아니라 데이터를 송수신하는 동작이나 연결을 끊는 동작도 포함한다

  - TCP 프로토콜의 사양으로 규정하고 있다
  - 클라이언트와 서버 사이에 주고 받는 패킷의 맨 앞부분에 부가한다
  - 접속 동작의 단계에서는 데이터가 없고 패킷의 내용은 제어 정보만으로 이루어져 있다
  - 제어 정보를 패킷의 맨 앞부분에 배치하는 곳부터 **헤더**라고 부른다

- **소켓에 기록하여 프로토콜 스택의 동작을 제어하기 위한 정보**

  애플리케이션에서 통지된 정보, 통신 상대로부터 받은 정보 등이 수시로 기록된다

  - 송수신 동작의 진행 상황 등도 수시로 기록되고, 프로토콜 스택은 하나하나 차례로 정보를 참조하면서 움직인다
  - 그러므로 소켓의 제어 정보는 프로토콜 스택의 프로그램과 일체화되어 있다
  - 소켓에 기록하는 제어 정보는 프로토콜 스택을 만드는 사람에 따라 달라지므로 간단히 설명할 수 없다

```
송신측 : 데이터 송신 동작을 개시합니다
수신측 : 예, 알겠습니다
송신측 : x번째 데이터를 보냅니다
수신측 : x번째 데이터를 받았습니다
// 이런 식의 대화가 헤더에 기록된 제어 정보에 의해 이루어진다.
```

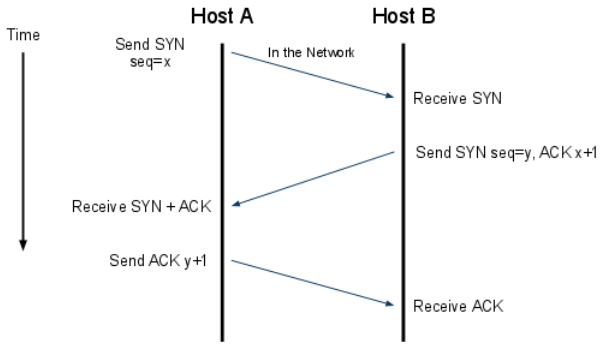
2. 접속 동작의 실체

> 접속 동작은 애플리케이션이 Socket 라이브러리의 connect를 호출하는 곳부터 시작된다.

```
connect(<디스크립터>, <서버측의 IP주소와 포트번호>, ......)
```

접속 동작의 처음은 TCP 담당 부분에서 접속을 타나내는 제어 정보를 기록한 TCP 헤더를 만드는 것이다.
그리고 TCP 헤더의 송신처와 수신처의 포트 번호로 접속하는 소켓을 지정한다.

https://gmlwjd9405.github.io/2018/09/19/tcp-connection.html



참고 링크(4)

- **클라이언트 → 서버**
  - 접속 해야하는 소켓을 지정한후 컨트롤 비트(SYN)를 1로 만든다.
- **서버 → 클라이언트**
  - 서버측의 TCP 담당부분이 TCP 헤더를조사하여 수신처 포트번호에 해당하는 소켓을 찾아낸다.
  - 해당 소켓에 필요한 정보를 기록하고 접속 동작이라는 상태로 변경
  - SYN 플래그 비트를 1로 만든다.
  - ACK 플래그 비트를 1로 만든다.

    → 패킷을 받은 것을 알리기 위한 동작
- **클라이언트 → 서버**
  - 서버로부터 받은 응답의 TCP 헤더를 조사
  - 소켓에 서버의 IP주소나 포트 번호 등과 함께 접속 완료를 나태내는 제어 정보를 기록
  - ACK 플래그 비트를 1로 만든 TCP 헤더를 서버에 송신

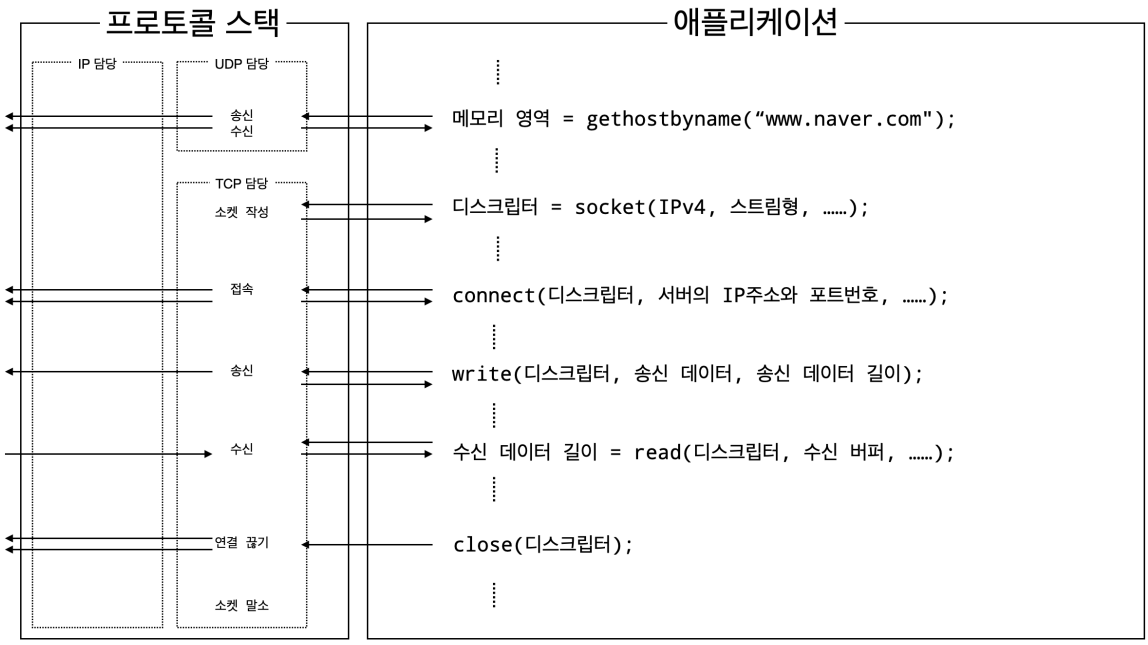
위 과정을 거치면 데이터를 송수신할 수 있는 상태가 된다.

이 때 파이프와 같은 것으로 소켓이 연결됐다고 생각하면 편하다. 이러한 파이프를 커넥션이라고 한다.

커넥션은 close 를 호출하여 연결을 끊을 때까지 존재한다.

데이터를 송수신한다.

> 프로토콜 스택에 HTTP 리퀘스트 메시지를 넘기며,
> connect에서 애플리케이션에 제어가 되돌아오면 데이터 송수신 동작이 들어오게 된다.
> 이 동작은 애플리케이션 write를 호출하여 프로토콜 스택에 송신 데이터를 건네주는 것부터 시작한다.



메시지 송신 동작

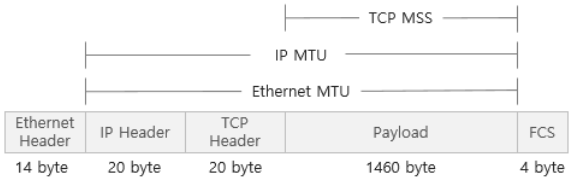
> 🌟 **프로토콜 스택은 받은 데이터의 내용에 무엇이 쓰여져있는지 알지 못합니다.**

- **데이터를 곧바로 송신하지 않고 자체의 버퍼 메모리 영역에 저장**하고, 다음 데이터를 기다린다.

  ❓ **왜 바로 안보내고 저장을 했다가 보낼까?**

  > 한 번의 송신 의뢰에서 건네주는 데이터의 길이는 애플리케이션의 사정에 따라 결정되고, 프로토콜 스택에서 제어할 수 없기 때문입니다.
  > 이런 상황에서 받은 데이터를 곧바로 보낸다면 작은 패킷을 많이 보내게 되며 성능이 저하됩니다.

  - 애플리케이션에서 프로토콜 스택에 건네주는 데이터의 길이는 다양하다.
  - 받은 데이터를 곧바로 보내는 단순한 방법이면 작은 패킷을 많이 보낼 수도 있다.



1. 어느 정도까지 저장하고 송신할지 판단하는 두 가지 요소

1. **한 패킷에 저장할 수 있는 데이터의 크기**

   MTU → 한 패킷에 저장할 수 있는 데이터의 크기 (이더넷에서는 보통 1500바이트)
   MSS → 헤더를 제외하고 한 개의 패킷으로 운반할 수 있는 TCP 데이터의 최대 길이

   애플리케이션에서 받은 데이터가 MSS를 초과하거나 MSS에 가까운 길이에 이르기까지 데이터를 저장하고 송신한다.

2. **타이밍**
   - 프로토콜 스택 내부에 타이머가 있어서 일정 시간이 경과하면 MTU 여부에 상관없이 패킷을 전송한다.

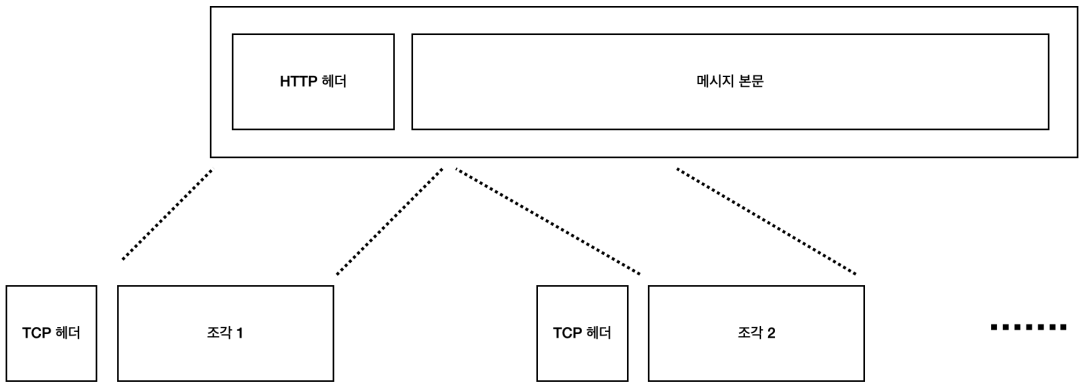
❓ **그럼 둘 중에 무엇을 좀 더 중시할까? 어떤 차이가 있을까?**

> 1번을 중시하면 네트워크 효율은 좋아지지만, 송신 동작이 지연된다.
> 2번을 중시하면 지연은 적어지지만 이용 효율이 떨어진다.
> 이 둘에 대한 절충 규정은 존재하지 않는다. 즉 개발자의 역량에 따라 성능이 달라질 수 있다는 말이다.

애플리케이션측에서 송신의 타이밍을 제어할 수 있다.
데이터 송신을 의뢰할 때 옵션을 지정할 수 있고, 브라우저 같은 경우는 버퍼에 머무는 만큼 응답 시간이 지연되기 때문에 옵션을 사용해서 바로 보내도록 할 수 있다.

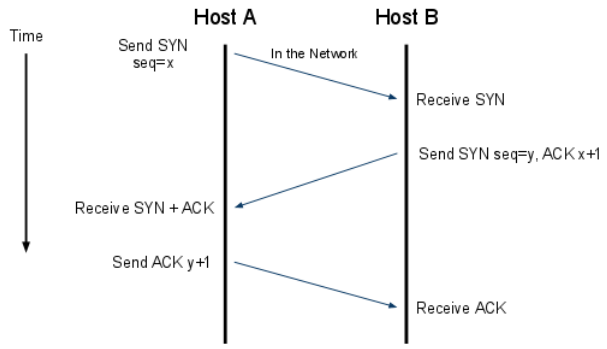
2. 데이터가 클 때는 분할하여 보낸다.

> HTTP 리퀘스트 메시지가 MSS 길이를 초과한 길이라면, 송신 버퍼에 들어있는 데이터를 맨 앞부터 차례대로 MSS의 크기에 맞게 분할하고, 분할한 조각을 한개씩 패킷에 넣어 송신한다.



3. ACK 번호를 사용하여 패킷이 도착했는지 확인한다.

> TCP에는 송시한 패킷이 상대에게 올바르게 도착했는지 확인하고, 도착하지 않았다면 다시 송신하는 기능이 있으므로 패킷을 송신한 후에는 확인 동작으로 넘어간다.



참고 링크(4)

- TCP 담당부분은 데이터를 조각으로 분할할 때 조각이 통신 개시부터 따져서 몇 번째 바이트에 해당하는지를 세어둔다.
- 데이터의 조각을 송신할 때 세어둔 값을 TCP 헤더에 기록하는데 **시퀀스 번호**라는것이 이에 해당한다.

  ❓ **그럼 시퀀스 번호는 1부터 시작하는건가?**

  > 시퀀스 번호는 난수이다.
  > 항상 1부터 시작하면 악의적인 공격을 당할 수도 있기 때문이다.

  이러한 시퀀스 번호를 통해서 패킷이 누락됐는지 알 수 있다.

  → 140번까지 받은 다음 141번을 받으면 누락이 안됐다는 것이고 만약 250번을 받으면 중간이 누락된 것이다.

실제 동작 과정

- 접속 동작을 실행할 때 클라이언트에서 서버로 보내는 데이터에 관한 시퀀스 번호의 초기값을 서버에 통지
- 서버에서 산출한 ACK 번호, 서버에서 클라이언트로 보내는 데이터에 관한 시퀀스 번호의 초기값을 포함하여 응답

- 클라이언트는 서버에서 받은 시퀀스 번호로부터 ACK 번호를 산출하여 서버에 전송한다.
- 클라이언트에서 버버로 데이터와 시퀀스 번호를 송신
- 서버는 ACK를 응답

> 🌟 **TCP는 이 방법으로 상대가 데이터를 받았는지 확인한다.**
> 확인될 때까지 송신한 패킷을 버퍼 메모로에 보관해둔다.
> 송신한 데이터에 대응하는 ACK 번호가 상대로부터 돌아오지 않으면 패킷을 재전송할 수도 있다.

4. 패킷 평균 왕복 시간으로 ACK 번호의 대기 시간을 조정한다.

> ACK 번호가 돌아오는 것을 기다리는 시간을 **타임아웃 값**이라고 한다.

- 대기시간은 너무 짧지도, 길지도 않은 적절한 값으로 설정해야 한다.
  → **하지만 이것은 너무 어렵다. 쓰레드 풀에서 몇 개의 쓰레드를 써야 하는지 결정해야 할 때와 똑같다.**
- 그래서 TCP는 대기시간을 동적으로 변경하는 방법을 취하고 있다.
  즉 ACK 번호가 돌아오는 시간을 기준으로 대기 시간을 판단하는 것이다.

> 🌟 **데이터 송신 동작을 실행하고 있을 때 항상 ACK 번호가 돌아오는 시간을 계측해둔다.**
> ACK 번호가 돌아오는 시간이 지연되면 이것에 대응하여 대기 시간도 늘리면 된다.
> 반대로 ACK 번호가 곧바로 돌아오면 대기 시간을 짧게 설정한다.

5. 윈도우 제어 방식으로 효율적으로 ACK 번호를 관리한다.

> 윈도우제어라는 방식에 따라 송신과 ACK 번호 통지 동작을 실행한다.

- 그러나 ACK 번호를 기다리지 않고 차례로 패킷을 보내면 수신측의 능력을 초과하여 패킷을 보내는 사태가 일어날 수도 있다.
- 수신측의 TCP는 패킷을 수신하면 일단 수신용 버퍼 메모리에 데이터를 일시 보관한다.
  수신측에서는 ACK 번호를 계산하거나 조각을 연결하여 원래 데이터를 복원한 후 애플리케이션에 건네줘야 한다.
  그런데 애플리케이션에 건네주는 속도보다 빠른 속도로 데이터가 도착하면 수신 버퍼에 데이터가 차곡차곡 쌓여서 곧 넘쳐버린다.

> 🌟 **윈도우 제어 방식의 개념**
> 위 문제는 수신측에서 송신측에 수신 가능한 데이터 양을 통지하고, 송신측은 이양을 초과하지 않도록 송신 동작을 실행하는데 이것이 이다.

- TCP 헤더의 윈도우 필드에서 이것을 알려준다.
  수신 가능한 데이터 양의 최대값을 윈도우 사이즈라고 부르고, 보통 수신측의 버퍼 메모리의 크기와 같은 크기가 된다.

6. ACK 번호와 윈도우를 합승한다.

> 윈도우 통지가 필요한 것은 수신측이 수신 버퍼에서 데이터를 추출하여 애플리케이션에 건네주었을 때이다.
> 수신측에서 애플리케이션에 데이터를 건네주고 수신 버퍼의 빈 영역이 늘어났을 때
> 이것을 송신측에 통지해야 하는데 이것이 **윈도우 통지의 타이밍**이다.

- ACK 번호는 데이터를 수신한 후 즉시 보낸다고 볼 수 있다.
- ACK 번호 통지와 윈도우 통지의 패킷이 하나씩 따로따로 송신측에 보내진다.
  → **효율성 저하**
- 수신측은 ACK 번호나 윈도우를 통지할 때 소켓을 바로 보내지 않고 **잠시 기다렸다가** 다음 통지 동작이 일어나면 한 개의 패킷으로 묶어서 보낸다
- 복수의 ACK 번호 통지가 연속해서 일어나는 경우도 **최후의 것만 통지**하고 도중의 것은 생략해도 상관 없다.
- 복수의 윈도우 통지가 일어났을 때는 **최후의 것만 통지**한다.

7. HTTP 응답 메시지를 수신한다.

> 송신측(클라이언트)는 HTTP 리퀘스트 메시지를 보내면 웹 서버에서 응답 메시지가 돌아오기를 기다리고 수신한다.

이를 위해서 브라우저는 read 프로그램을 호출한다.

수신 작업은 위에서 설명한것과 동일하게 동작한다.

- TCP 헤더의 내용을 조사하여 누락된 데이터를 확인

→ 문제 없으면 ACK 번호 응답

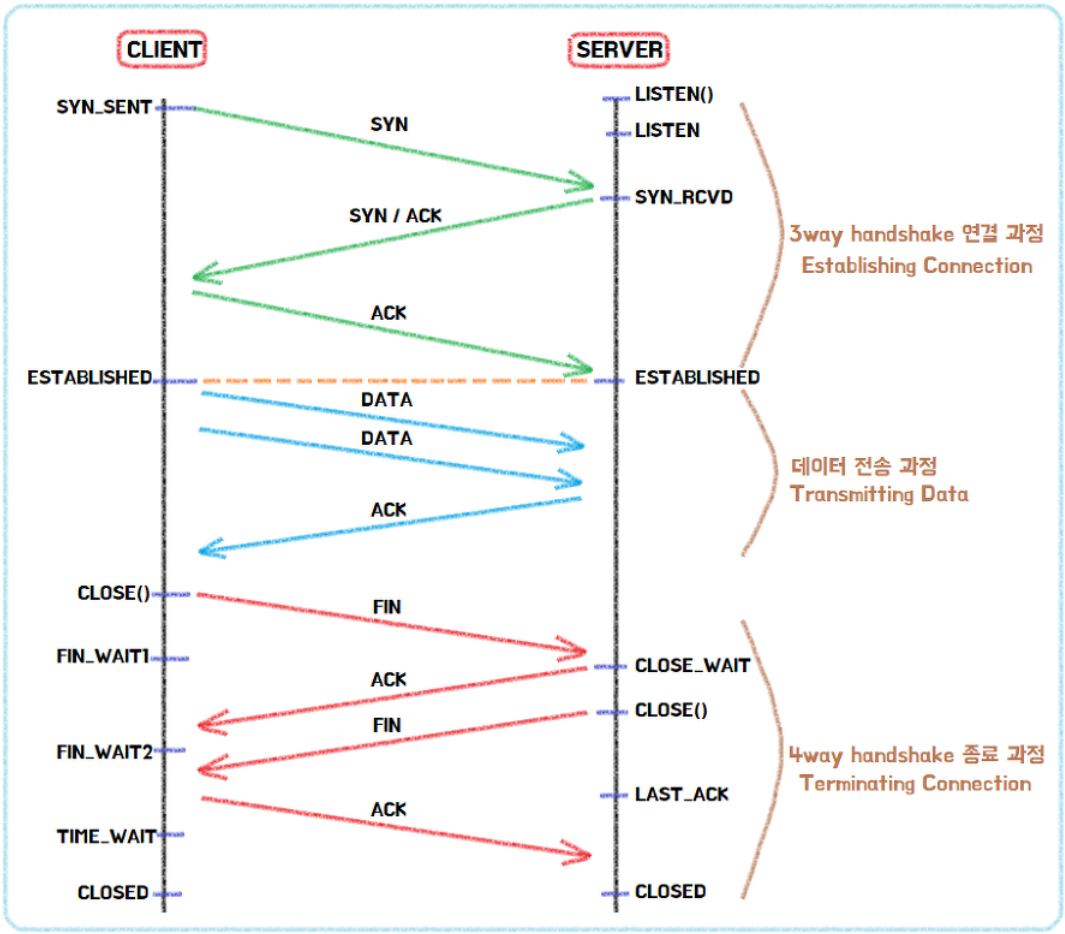
- 버퍼에 일시 보관하고 조각을 연결하여 애플리케이션 메모리 영역으로 전달

서버에서 연결을 끊어 소켓을 말소한다

1. 데이터 보내기를 완료했을 때 연결을 끊는다.

> 데이터 송수신 동작이 끝난 후 프로토콜 스택의 움직임을 설명한다.

- 애플리케이션이 송신해야 하는 데이터를 정부 송신완료 했다고 판단할 때
- 웹의 경우 리퀘스트 메시지를 보내고 서버에서 응답 메시지를 반송하면 데이터 송수신이 끝나므로 서버측에서 연결끊기 단계에들어간다.(HTTP 1.0)
  - HTTP 1.1의 경우에는 서버가 응답메시지를 반송한 후 계속 클라이언트가 다음 리퀘스트 메시지를 보내는 것이 가능하므로, 이때는 클라이언트 측에서 먼저 연결 끊기 단계에 들어갈 수 있다.



https://www.crocus.co.kr/1362

1. 서버와 클라이언트가 TCP 연결이 되어있는 상태에서 클라이언트가 접속을 끊기 위해 CLOSE() 함수를 호출하게 된다.
   이후 CLOSE() 함수를 호출하면서 FIN segment를 보내게 되고 클라이언트 FIN_WAIT1 상태로 변환된다.
2. 서버는 클라이언트가 CLOSE() 한다는 것을 알게되고 CLOSE_WAIT 상태로 바꾼 후 ACK segment를 전송한다.
   즉 클라이언트가 끊을 것이라는 신호를 받았다는 의미이고, CLOSE_WAIT를 통해 자신의 통신이 끝날때까지 기다리는 상태가 된다.
3. ACK segment를 받은 클라이언트는 FIN_WAIT2로 변환되고 이 때 서버는 CLOSE() 함수를 호출하고 FIN segment를 클라이언트에게 보낸다.
4. 서버도 연결을 닫았다는 신호를 클라이언트가 수신하면 ACK segment를 보낸 후 TIME_WAIT 상태로 전환된다.

2. 소켓을 말소한다.

> 소켓을 바로 말소하지 않고 잠시 기다린 후 소켓을 말소 (오동작을 막기 위해)

오작동의 예시

- 클라이언트가 FIN 송신
- 서버가 ACK 번호 송신
- 서버가 FIN 송신
- 클라이언트가 ACK 번호 송신 - **이 부분에서 오류가 발생했을 경우**

서버는 ACK가 돌아오지 않으므로 다시 한 번 FIN 송신을 하게 된다.

만약 클라이언트의 소켓이 말소되면 기록돼 있던 제어정보가 없어지므로 소켓에 할당되어 있던 포트번호도 알 수 없게 된다.

또 이 시점에 다른 애플리케이션이 생성한 소켓에 같은 포트번호가 할당될 수 있다.

이때 FIN을 받게 되면 그 소켓이 연결 끊기 동작에 들어갈 수 있다.

> 🌟 **즉 일반적으로 보통 몇 분 정도 기다리고 소켓을 말소한다.**

IP와 이더넷의 패킷 송수신 동작

1. 패킷의 기본

> IP 담당 부분이 어떻게 패킷을 송신하는가를 알아볼 단계이다.

패킷은 헤더와 데이터 두 부분으로 구성된다.

- **헤더**에는 수신처 주소 등의 제어 정보가 들어가있고
- **데이터 부분**에는 그 자체로 데이터가 담긴다.

패킷 통신 기본 구조

> 패킷의 송신처가 되는 기기가 패킷을 만든다.

- 가장 가까운 중계장치에 송신한다.
- 중계 장치는 패킷의 헤더를 조사하여 다음 목적지를 판단한다.
  → 수신처가 어느 방향에 있는지에 대한 정보를 기록한 표 같은 값을 사용한다.
- 최종적으로 수신처의 기기에 도착한다.

> 🌟 **TCP/IP 패킷 구조는 이 기본에서 발전한 것이다.(단 조금 더 복잡하다.)**
> **라우터와 허브라는 두 종류의 패킷 중계장치가 역할을 분담하면서 패킷을 운반한다.**

2. 패킷 송수신 동작의 개요

> 패킷을 운반하는 것은 라우터와 허브가 하므로 IP 담당 부분은 패킷을 송출하기만 하는 입구 부분이다.

❓ 단지 입구 역할만 하는가?

> 입구 부분이지만 해야 할 일이 존재한다.
> TCP 담당 부분으로부터 데이터조각에 TCP 헤더가 부가된 것을 받는다.
> 이것이 들어가는 내용물이 되고 통신 상대의 IP 주소를 나타낸다.
> TCP 담당 부분은 IP 담당 부분에게 이 내용물을 상대에게 전달해달라고 의뢰하는 것이다.

IP 헤더 / MAC 헤더

- IP 헤더
  - IP 프로토콜에 규정된 규칙에 따라 패킷의 목적지가 되는 엑세스 대상 서버의 IP 주소를 기록
- MAC 헤더
  - 이더넷 등의 LAN을 사용하여 가장 가까운 라우터까지 패킷을 운반할 때 사용하는 제어정보를 기록 → MAC 주소

이렇게 만들어진 패킷을 네트워크용 하드웨어(이더넷, 무선LAN - LAN어댑터)에 건네준다.

송신 동작

LAN어댑터에 0이나 1의 비트가 이어진 디지털 데이터를 보내고, LAN 어댑터가 전기나 빛의 신호 상태로 바꾸어 케이블에 송출

수신 동작

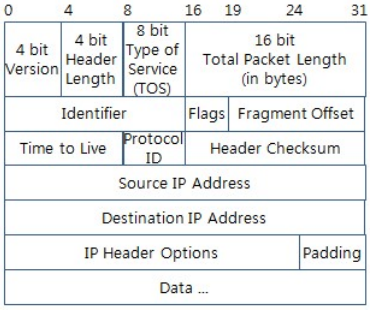
송신과 반대로 신호 상태의 패킷을 디지털 데이터로 변환한 후 IP 담당 부분에 전달

📋 **간단 정리**

- TCP 담당 부분의 데이터 송수신 동작의 단계와 상관없이 IP 패킷은 모두 같다.

  → 제어 패킷이든 데이터 패킷이든 역할에 상관없이 같다.
- 의뢰받은 내용물을 패킷의 모습으로 만들어 상대에게 송신하거나 전달한 패킷을 수신만 한다.

3. 수신처 IP 주소를 기록한 IP 헤더를 만든다.



IPv4 헤더 구성 http://www.ktword.co.kr/test/view/view.php?m_temp1=1859

> IP 헤더에 들어가는 내용 중 가장 중요한 것은 수신처의 IP 주소이다.
> TCP 담당 부분에서 통지된 통신 상대의 IP 주소를 그대로 설정한다.
> 송신처의 IP 주소의 경우는 LAN 어댑터에 의해 결정된다.

IP 주소는 사실 컴퓨터에 할당되는 것이 아니라 **LAN 어댑터에 할당된다.**

여러 개의 LAN 어댑터를 장착하면 각 LAN 어댑터에 서로 다른 IP 주소가 할당되기 때문이다.

즉 LAN 어댑터가 결정되면 IP 주소가 결정된다.

🌟 **패킷을 건네줄 상대를 판단하는 방법은 라우터가 IP용 표(경로표, 라우터 테이블)를 사용하여 다음 라우터를 결정하는 동작과 같다.**

프로토콜 번호라는 필드에도 값을 설정한다.

여기에는 패킷에 들어간 내용물이 어디에서 의뢰받은 것인지를 나타내는 값을 설정하며 TCP는 06 UDP는 17과 같은 규칙이 정해져 있다.

4. 이더넷용 MAC 헤더를 만든다.

> 이더덧에는 TCP/IP 개념이 통용되지 않는다.
> 이더넷은 다른 구조로 패킷의 수신처를 판단하며 이 구조를 따르지 않으면 패킷을 운반할 수 없다.
> 이더넷 수신처 판단 구조로 사용하는 것이 MAC 헤더이다.

MAC 헤더 구성 요소

- 수신처 MAC 주소
- 송신처 MAC 주소
- 이더 타입(Ether type)
  - 사용하는 프로토콜의 종류를 나타낸다.
  - 이더넷의 내용물은 IP나 ARP라는 프로토콜의 소켓이며, 규칙에 따라 값을 기록하게 된다.

5. ARP로 수신처 라우터의 MAC 주소를 조사한다.

ARP(Address Resolution Protocol)

- 이더넷에는 연결되어 있는 전원에게 패킷을 전달하는 브로드캐스트라는 구조로 돼 있다.
- 이 브로드캐스트를 이용하여 연결된 전원에게 MAC 주소를 조사하게 된다.
- 패킷을 보낼 때마다 이 동작을 하면 ARP 패킷이 불어나기 때문에 ARP 캐시라고 불리는 메모리에 저장한다.

즉 패킷을 송신할 때 우선 캐시를 조사하여 상대의 MAC 주소가 저장돼 있으면 ARP 조회 과정을 거치지 않고 캐시에 보존되어 있는 값을 사용하는 것이다. 하지만 캐시에 저장된 값은 **일정 시간이 지나면 삭제**된다.

→ 캐시의 내용과 현실 사이에 일치하지 않을 수도 있기 때문이다.

MAC 헤더를 IP 헤더 앞에 붙이면 패킷이 완성된다.
이렇게 패킷을 만들기까지가 IP 담당 부분의 역할이다.

6. 이더넷의 기본

> 이더넷은 다수의 컴퓨터가 여러 상대와 자유롭게 적은 비용으로 통신하기 위해 고안된 통신 기술이다.
> 패킷의 수신처 MAC 주소에 따라 누구에게 갈 것인지를 알고, 송신처 MAC 주소에 따라 누가 송신한 것인지 알며 이더 타입에 의해 패킷의 내용물로 무엇이 들어있는지를 아는 원리로 패킷이 전달된다.

7. IP 패킷을 전기나 빛의 신호로 변환하여 송신한다.

> 아직 IP가 만든 패킷은 메모리에 기억된 디지털 데이터이다.
> 그렇기에 전기나 빛의 신호로 변환하여 네트워크 케이블에 송출해야 한다.

이러한 동작을 실행하는 것이 LAN 어댑터이다.

- LAN 어댑터를 제어하려면 LAN 드라이버 소프트웨어가 필요하다.
  → 모든 하드웨어는 제어하기 위한 드라이버 소프트웨어가 필요하다.
- LAN 어댑터는 사용하기 위해 초기화 작업으로 이더넷의 송수신 동작을 제어하는 MAC 이라는 회로에 MAC 주소를 설정한다.
- 이 MAC 주소는 전 세계에서 중복되지 않도록 일원화되어 관리되고 있따.
- LAN 어댑터의 ROM에 제조할 때 기록되며 LAN 드라이버가 MAC 회로에 설정한다.

8. 패킷에 3개의 제어용 데이터를 추가하고 허브를 향해 패킷을 송신한다.

> LAN 드라이버는 IP 담당부분에서 패킷을 받으면 그것을 LAN 어댑터의 버퍼 메모리에 복사한 뒤 MAC 회로에 명령을 보내면 MAC 회로의 작업이 시작된다.

프리 엠블

- 송신하는 패킷을 읽을 때의 타이밍을 잡기 위한 거
- 비트열이기에 0과 1로 이루어져있다.

❓ **이런 프리엠블이 왜 필요한가?**

> 디지털 데이터를 전기 신호로 나타날 때 비트 값을 전압이나 전류의 값에 대응시킨다.
> 각 비트를 구별할 수 있는 클록이라는 신호가 있고, 이 클록 신호의 타이밍을 판단하는 것이 중요하다.

스타트 프레임 딜리미터

- 프리엠블에 이어지는 비트
- 마지막 비트 패턴이 조금 다르다 → 1011
- 패킷의 시작을 알리는 역할을 한다.

프레임 체크 시퀀스(FCS)

- 패킷을 운반하는 도중 잡음 등의 영향으로 파형이 흐트러질 수 있다.
- 위 오류를 검출하기 위한 비트열이다.
- 패킷의 앞 부분부터 끝까지 어떠한 계산식에 의해 계산된 값을 넣어둔다.

9. 허브를 향해 패킷을 송신한다.

> 제어용 데이터를 추가하면 케이블에 송출하는 패킷이 완성된다.
> 반이중 모드의 동작은 신호의 충돌을 피하기 위해 아래와 같이 행동한다.

- 케이블에 다른 기기가 송신한 신호를 흐르는지 조사
  → **흐르지 않으면 송신 동작 시작!**
- MAC 회로가 디지털 데이터를 전기 신호로 변환
- PHY(MAU) 회로가 케이블에 송출하는 형식으로 변환하여 케이블에 송신
  → 송신뿐만아니라 수신 신호선에서 신호가 흘러들어오는지 감시한다.

❓ **신호를 송신하고 있는 사이 수신 신호가 흘러오면?**

> 신호의 충돌이 일어나게 된다.

- 송신 동작을 중지하고 충돌 사실을 다른 기기에 알린다 → **재밍 신호**
- 대기시간을 계산하여, 그 대기시간만큼 기다린 후 재전송한다.

  → MAC 주소를 바탕으로 난수를 생성하여 대기시간을 계산
- 다시 충돌시 대기 시간을 2배로 늘려 반복

열 번째 시도후 해결되지 않으면 오류로 판단

10. 돌아온 패킷을 받는다.

> 수신 동작은 수신 신호선에서 신호를 받아들이는 것부터 시작한다.

- 프리앰블을 이용하여 타이밍 계산
- 스타트 프레임 딜리미터 후 부터 디지털 데이터로 변환하여 동작을 개시
- PHY회로에서 MAC 회로로 보낸다
- MAC회로에서 FCS 값을 검사한다. - 오류시 폐기
- MAC 헤더의 수신처 MAC 주소를 조사 - 자신의 것이 아닐시 폐기
- 본체에 통지

  → **인터럽트**라는 구조를 사용

  ❓ **인터럽트란?**

  > 컴퓨터 본체가 실행하고 있는 작업에 끼어들어 LAN 어댑터쪽에 주의시키는 것

- LAN 드라이버가 LAN 어댑터의 버퍼 메모리에서 수신한 패킷 추출
- MAC 헤더의 프로토콜 판별
- 프로토콜에 맞는 프로토콜 스택에 건네준다

11. 서버의 응답 패킷을 IP에서 TCP로 넘긴다!

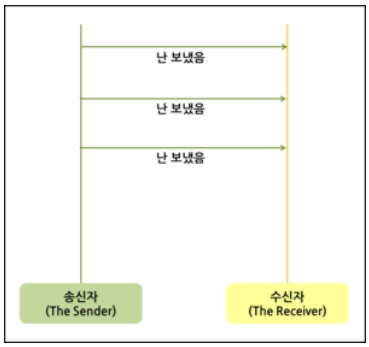
> 웹 서버에서 패킷이 돌아온 것으로 간주하고 프로토콜 스택의 동작을 추적해보자.
> LAN 드라이버는 tcp/ip 프로토콜 스택에 패킷을 건넨다.

- IP 담당 부분은 IP 헤더 부분을 조사하여 포멧에 문제가 없는 확인하고, 수신처의 IP 주소를 조사한다.
- 자신의 IP 주소와 동일한 TCP 담당부분에 건네준다.

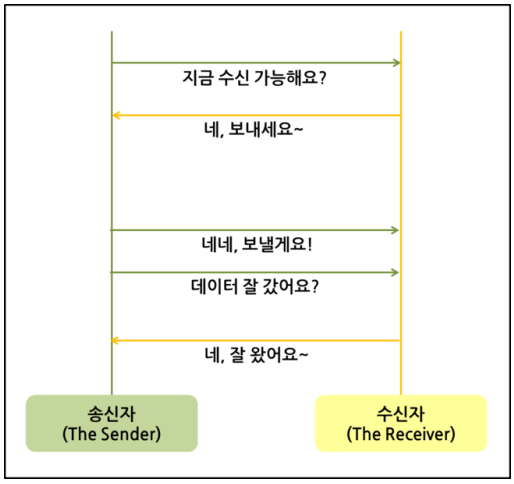
UDP 프로토콜을 이용한 송수신 동작

> 대부분의 애플리케이션은 TCP 프로토콜을 사용해 데이터를 송수신해 실행한다고 말했다.
> 하지만 모든 것이 그렇지만은 않다.

이미지 및 정보 참고



UDP 통신 과정



TCP 데이터 송신 과정

보자마자 느껴지는게 UDP는 일방적이라는게 느껴진다.

❓ 그럼 어떠한 곳에서 사용할까?

- **제어용 짧은 데이터**
  - DNS 서버에 대한 조회 등 제어용으로 실행하는 정보 교환은 한 개의 패킷으로 끝나는 경우가 많다.

    → 만약 오류 발생시 응답이 돌아오지 않으므로 데이터를 요청하면 된다.
- **음성 및 동영상 데이터**
  - 음성이나 영상 데이터는 결정된 시간안에 데이터를 건네줘야 한다.

    → TCP의 경우 시간이 더 걸리므로 재생 타이밍이 맞지 않을 수 있다.

> 🌟 **즉 신뢰성이 필요한 애플리케이션에서는 TCP를**
> **간단한 데이터를 빠른 속도로 전송하고자 할 경우 UDP를 사용한다.**

TCP	UDP
Connection-oriented protocol(연결지향형 프로토콜)	Connection-less protocol(비 연결지향형 프로토콜)
Connection by **byte** stream(바이트 스트림을 통한 연결)	Connection by **message** stream(메세지 스트림을 통한 연결)
Congestion / Flow control(혼잡제어, 흐름제어)	NO Congestion / Flow control(혼잡제어와 흐름제어 지원 X)
Ordered, Lower speed(순서 보장, 상대적으로 느림)	Not ordered, Higer speed(순서 보장되지 않음, 상대적으로 빠름)
Reliable data transmission(신뢰성 있는 데이터 전송 - 안정적)	Unreliable data transmission(데이터 전송 보장 X)
TCP packet : Segment(세그먼트 TCP 패킷)	UDP packet : Datagram(데이터그램 UDP 패킷)
HTTP, Email, File transfer에서 사용	DNS, Broadcasting(도메인, 실시간 동영상 서비스에서 사용)