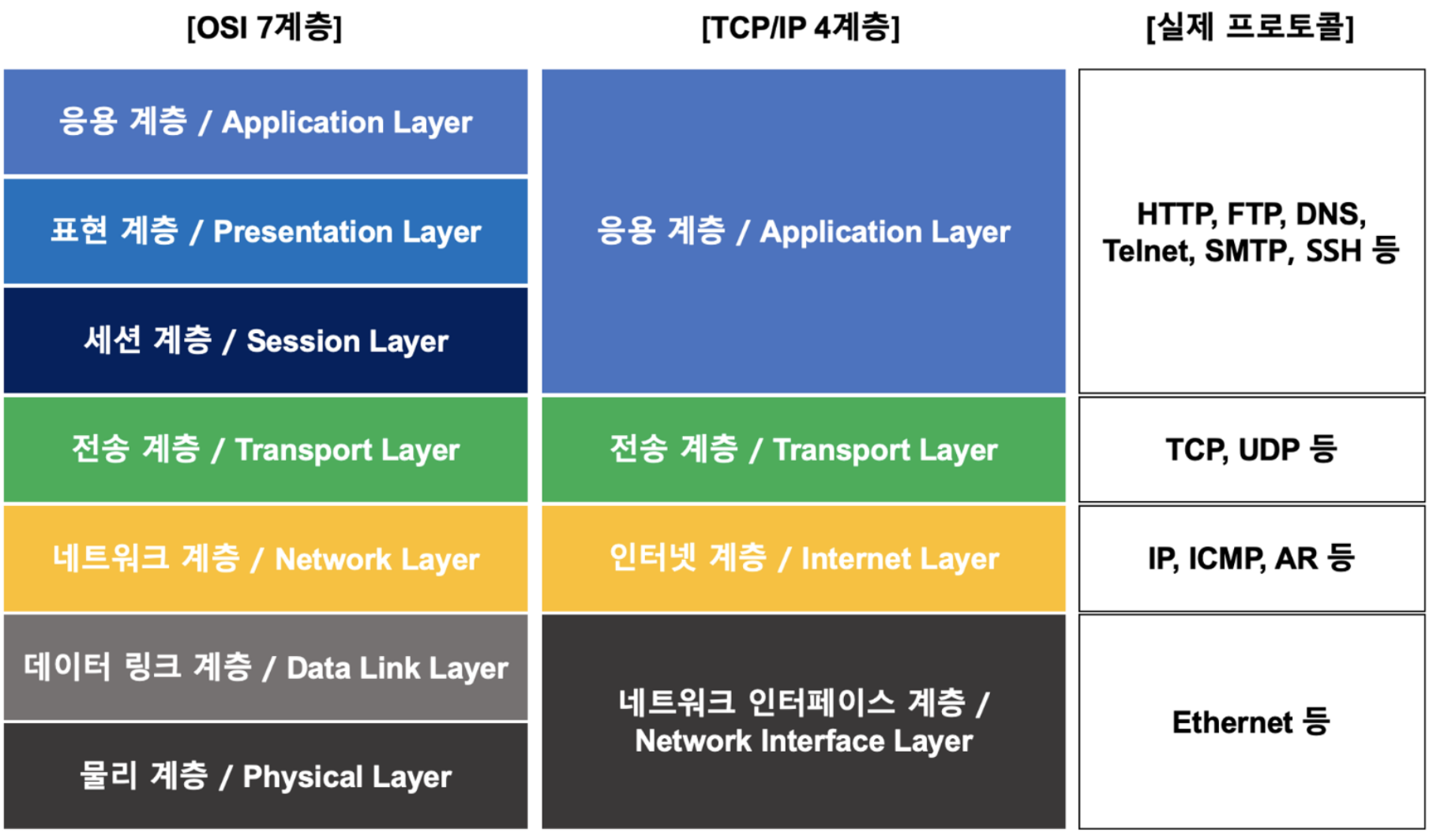
# TCP/IP

## ▼ OSI 7계층 & TCP/IP 4계층




- `OSI 7계층`
  - **네트워크 통신을 표준화한 모델**
  - 서로 다른 컴퓨터들이 데이터를 주고받을 수 있도록 표준화된 규칙
- `TCP/IP 4계층`
  - OSI 모델을 기반으로 **실제 인터넷에서 사용되는 단순화된 모델**
  - `TCP/IP` **:** 현재의 인터넷에서 컴퓨터들이 서로 정보를 주고받는데 쓰이는 프로토콜의 모음

### 특징

- 각 계층은 **하위 계층의 기능을 이용**하고, **상위 계층에게 기능을 제공**
  - ex) HTTP는 TCP과 IP을 이용해서 작동
- 일반적으로 상위 계층의 프로토콜은 소프트웨어로, 하위 계층의 프로토콜은 하드웨어로 구현됨
- `캡슐화`와 `역캡슐화`
  - `캡슐화` **:** 통신 프로토콜의 특성을 포함한 **정보를 Header에 포함시켜서 하위 계층에 전송하는 것**
  - `역캡슐화` **:** 통신 상대측에서 이러한 Header를 **역순으로 제거하면서 원래의 Data를 얻는 과정**

### 계층은 왜 나눌까?

통신이 일어나는 과정을 단계별로 알 수 있고, 특정한 곳에 이상이 생기면 다른 단계를 건드리지 않고도 **이상이 생긴 단계만 고칠 수 있기 때문**

## ▼ TCP와 UDP 비교

### Transport layer(전송계층)

- end point간 **신뢰성**있는 데이터 **전송**을 담당하는 계층
- **신뢰성** : 데이터를 순차적, 안정적인 전달
- **전송** : 포트 번호에 해당하는 프로세스에 데이터를 전달

## TCP와 UDP는 왜 나오게 됐는가?

IP 프로토콜의 한계를 해결하기 위해서

- 비연결성
  - 패킷을 받을 대상이 없거나 서비스 불능 상태여도 패킷전송
- 비신뢰성
  - 중간에 패킷 사라지거나, 순서대로 안와도 해결 못함
- 프로그램 구분
  - 같은 ip를 사용하는 서버에서 통신하는 애플리케이션이 둘 이상이면 어디로 보내야하는지 모름

## TCP(Transmission Control Protocol)

연결형, 신뢰성 전송 프로토콜

- TCP로 전송하는 패킷을 `segment`라고 부름
  - `segment` : 어플리케이션 단에서 받은 데이터를 TCP 프로토콜 안에서 자르고 TCP Header를 데이터에 추가한 것
- **TCP Segment Format**

**TCP segment header**

| *Offsets* | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Octet | Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | Source port | | | | | | | | | | | | | | | | Destination port | | | | | | | | | | | | | | | |
| 4 | 32 | Sequence number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 64 | Acknowledgment number (if ACK set) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 96 | Data offset | | | | Reserved<br>0 0 0 | | | | NS | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size | | | | | | | | | | | | | | |
| 16 | 128 | Checksum | | | | | | | | | | | | | | | | Urgent pointer (if URG set) | | | | | | | | | | | | | | | |
| 20 | 160 | Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ⋮ | ⋮ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 60 | 480 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Source/Destination Port: 전송 포트번호로 하기 때문

Sequence Number/ Acknowledgement Number : 순차전송을 위해

ACK SYN FIN Flags : TCP 연결 제어 및 데이터 관리를 위해

- SYN : 커넥션 연결시 쓰는 flag
- FIN : 커넥션 끊을 때 쓰는 flag
- ACK : data를 전송하면 수신자가 받았음을 알려주기 위해 쓰는 flag

## 특징 : 신뢰성 보장

- **연결형 서비스**
  - 데이터 전송 전, 논리적 연결을 설정한 후 데이터를 전송
  - 3-way handshaking - 양방향 통신
- **흐름 제어(Flow control)**
  - 수신 측에서 처리하지 못해 손실되는 데이터가 없도록 하기 위한 기능

    - 데이터 처리 속도를 조절하여 수신자의 버퍼 오퍼플로우를 방지 (송신 측이 수신 측의 처리속도 보다 더 빨리 데이터를 보내지 못하도록 제어)
- **혼잡 제어(Congestion control)**
    - 망의 혼잡으로 인해 발생할 수 있는 데이터 손실을 막기 위한 기능
    - 네트워크 내의 패킷 수가 과도하게 증가하지 않도록 방지 (송신 측의 데이터 전달과 네트워크의 처리 속도 차이를 해결하기 위해 송신 측의 데이터 전송 속도 제어)
- 데이터의 **순차 전송**을 **보장**
- **오류감지(Error detection)**
    - 체크썸을 통해 확인
- ex) **파일전송**과 같은 신뢰성이 중요한 서비스에 사용

### 단점

- 매번 커넥션 연결해서 시간 손실 (3-way handshaking)
- 패킷을 조금만 손실해도 재전송

## UDP(User Datagram Protocol)

비연결형, 비신뢰성 전송 프로토콜

- UDP로 전송하는 패킷을 `datagram`라 부름
    - 어플리케이션 단에서 내려온 데이터를 UDP는 쪼개지 않음

### 특징

- **비연결형 서비스**
    - 패킷 전송 전, 호스트 간 연결을 설정하지 않고 데이터를 단방향으로 전송
    - Conectionless - 3-way handshaking X
- **최소한의 사항만 보장 ⇒ 낮은 신뢰성, 빠른 속도**
    - 흐름 제어 X, 혼잡 제어 X
    - 데이터 전송 보장 X : 데이터의 중복 및 손실 발생 가능
    - 데이터 순서 보장 X
    - 데이터 수신 확인 X
- IP와 거의 같고 **PORT + 체크썸(Error Detection)** 정도만 추가
- ex) **스트리밍**과 같이 연속성이 더 중요한 서비스에 사용

## TCP vs UDP

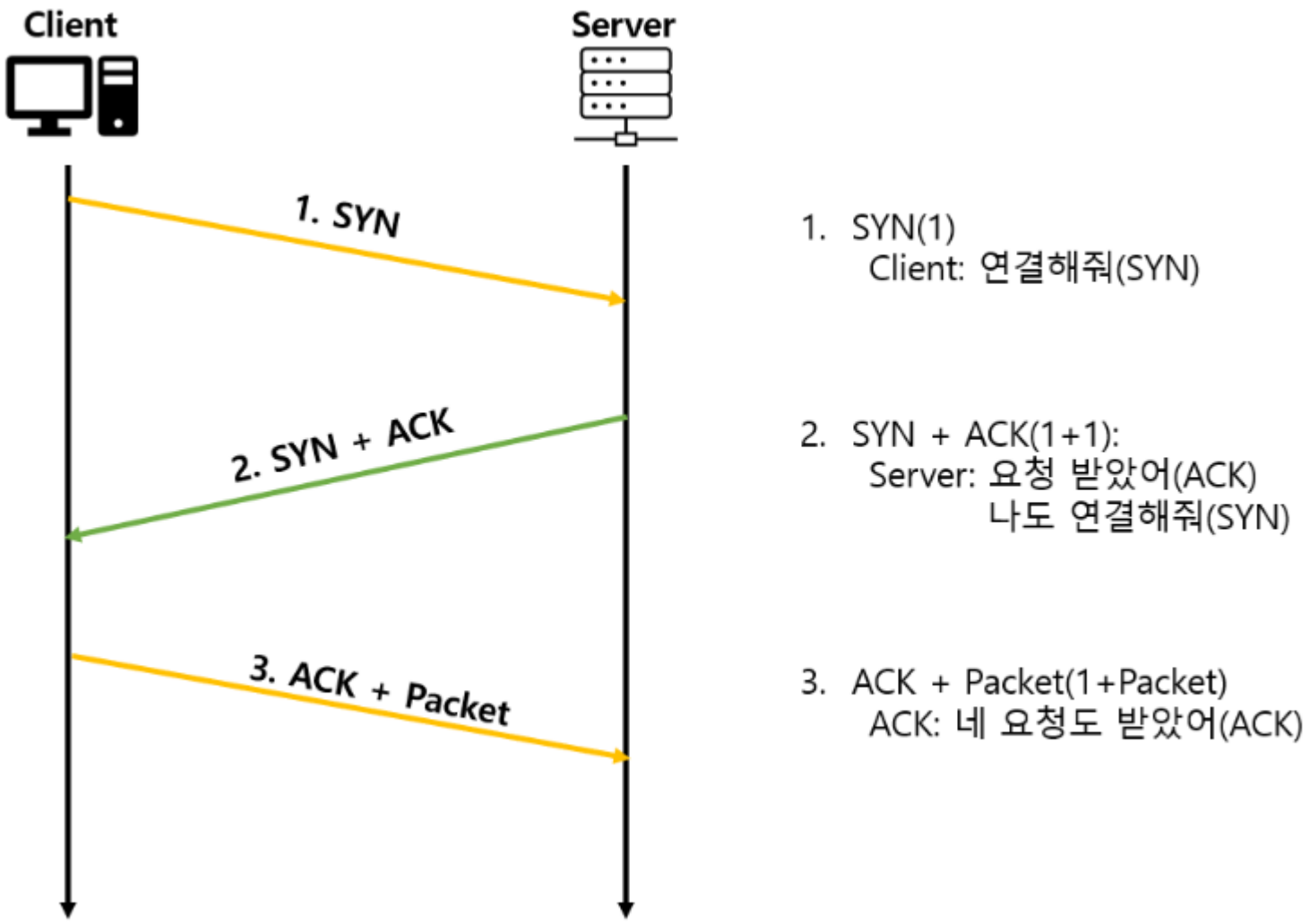
| 프로토콜 종류 | TCP | UDP |
| --- | --- | --- |
| 연결 방식 | 연결형 서비스 | 비연결형 서비스 |
| 패킷 교환 방식 | 가상 회선 방식 | 데이터그램 방식 |
| 전송 순서 | 전송 순서 보장 | 전송 순서가 바뀔 수 있음 |
| 수신 여부 확인 | 수신 여부를 확인함 | 수신 여부를 확인하지 않음 |
| 통신 방식 | 1:1 통신 | 1:1 OR 1:N OR N:N 통신 |
| 신뢰성 | 높다. | 낮다. |
| 속도 | 느리다. | 빠르다. |

## ▼ 3-way handshake

TCP 통신은 아래와 같은 3단계의 과정을 거칩니다.

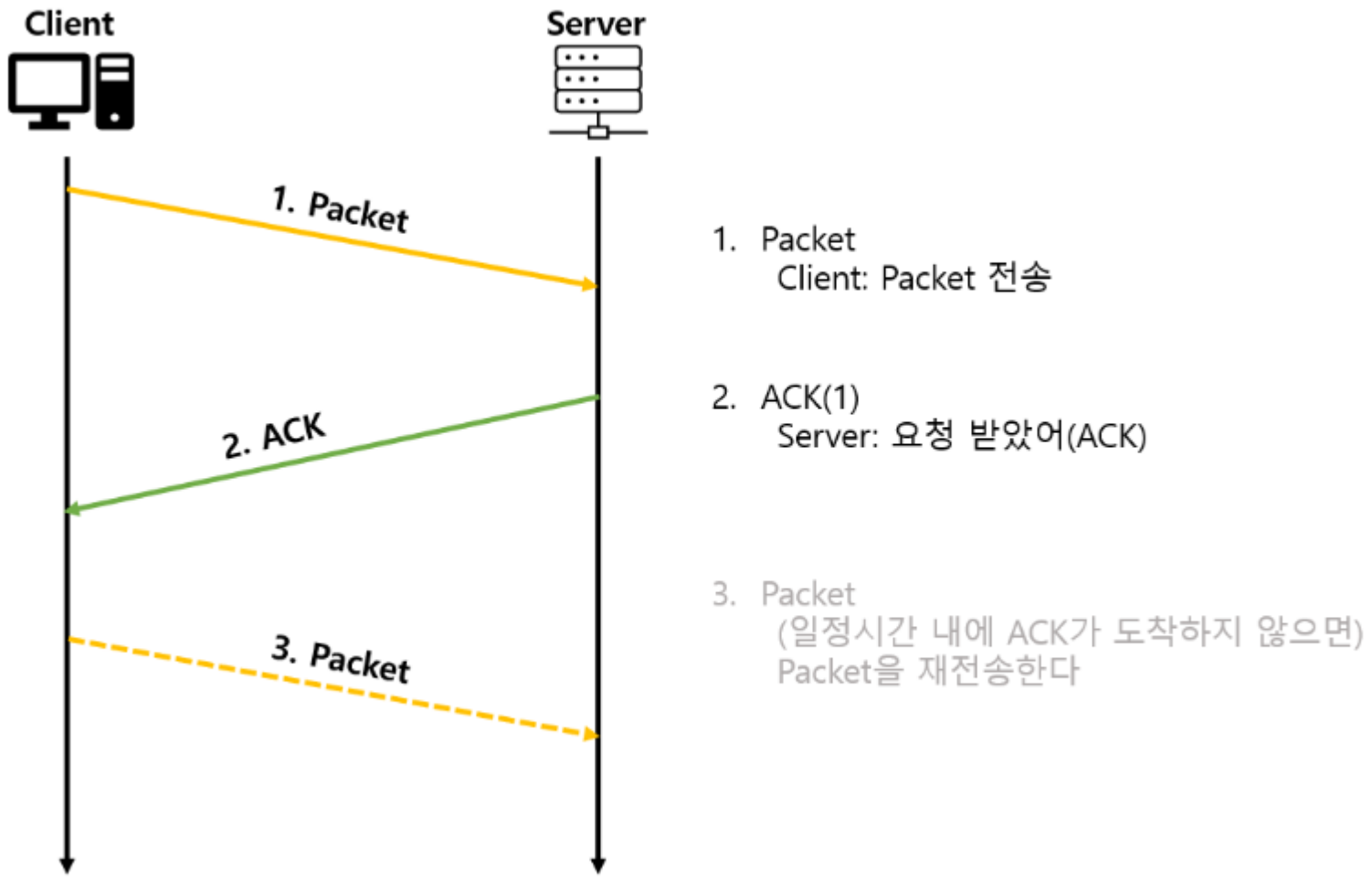
1. Connection setup (tcp 연결 초기화) - 3way handshaking
2. Data transfer (데이터 전송)
3. Connection termination (tcp 연결 종료) - 4way handshaking

### 3-way handshaking : 커넥션 생성



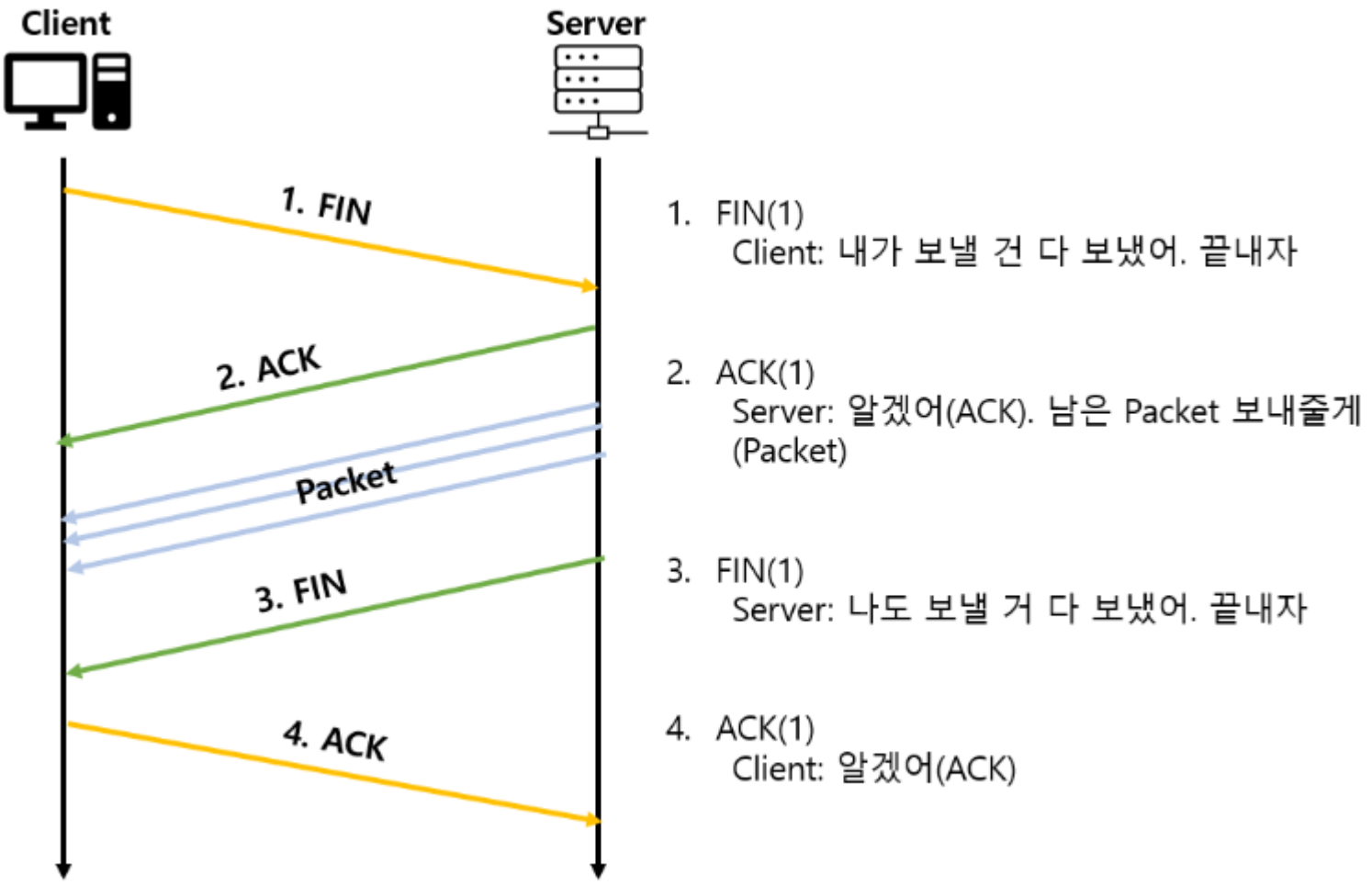

1. 클라이언트가 서버에게 접속을 요청하는 **SYN** 패킷을 보냄
2. 서버는 요청을 수락하는 ACK를 포함하여 **SYN+ACK** 패킷을 클라이언트에게 발송
3. 클라이언트가 이것을 수신한 후, 다시 **ACK**를 서버에게 발송하면 연결이 이루어짐

### Data 전송




1. 클라이언트가 **패킷** 송신
2. 서버에서 요청 받았다는 **ACK** 송신
3. 만약 클라이언트가 ACK 수신하지 못하면 패킷 재전송

### 4 way handshaking : 커넥션 해제

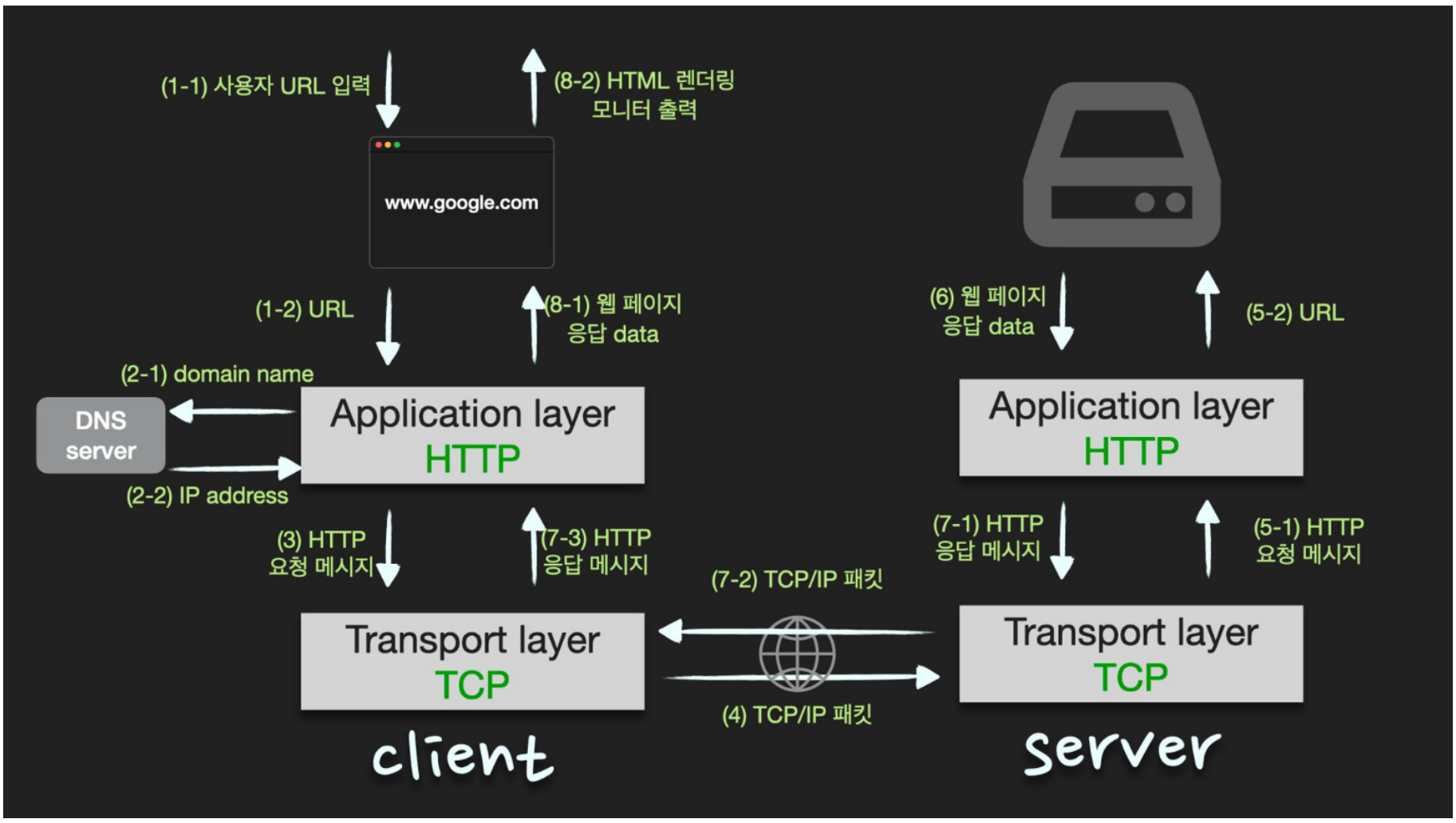



1. 데이터를 전부 송신한 클라이언트가 **FIN** 송신
2. 서버가 FIN 세그먼트를 받았다는 응답에 대한 **ACK** 송신
3. 서버가 남은 **패킷** 송신 (일정시간 대기)

4. 패킷 모두 전달한 다음에 서버가 **FIN** 패킷 송신
5. 클라이언트가 **ACK** 송신받게 되면 연결 종료

- **연결 종료시 4 way handshaking 하는 이유**
  - 회선내에 존재하는 패킷의 유실을 방지하기 위해

## ▼ www.google.com을 주소창에 쳤을 때 화면이 나오기까지의 과정




1. 유저가 브라우저에서 www.google.com(**URL**)을 **입력**을 하면 HTTP request message를 생성합니다.
2. IP주소를 알아야 전송을 할 수 있으므로, **DNS** lookup을 통해 해당 **domain의 server IP주소를 알아냅니다.**
3. **반환된 IP주소**(구글의 server IP)**로 HTTP 요청 메시지**(request message) **전송 요청**을 합니다.
   a. 생성된 HTTP 요청 메시지를 TCP/IP층에 전달합니다.
   b. HTTP 요청 메시지에 헤더를 추가해서 **TCP/IP 패킷을 생성**합니다.
4. 해당 **패킷**은 전기신호로 랜선을 통해 네트워크로 전송되고, **목적지 IP에** 도달합니다.
5. 구글 server에 도착한 패킷은 unpacking을 통해 **message를 복원**하고 server의 process로 보냅니다.
6. server의 process는 HTTP 요청 메시지에 대한 response data를 가지고 **HTTP 응답 메시지**(response message)를 **생성** 합니다.
7. HTTP 응답 메시지를 전달 받은 방식 그대로 **client IP로 전송**을 합니다.
8. **HTTP response 메시지**에 담긴 데이터를 토대로 **웹브라우저에서 HTML 렌더링**을 하여 모니터에 검색창이 보여집니다.

▼ 간단히

1. 사용자가 브라우저에 URL 입력
2. 브라우저는 DNS를 통해 서버의 IP 주소를 찾는다
3. client에서  HTTP request 메시지 ⇒ TCP/IP 패킷 생성 ⇒ server로 전송
4. server에서 HTTP requset에 대한 HTTP response 메시지 ⇒ TCP/IP 패킷 생성  ⇒ client로 전송
5. 도착한 HTTP response message는 웹 브라우저에 의해 출력(렌더링)