네트워크 정리

1. OSI계층과 TCP/IP
2. TCP
3. IP
4. URP, ICMP, ARP (모두 network layer)

**OSI계층과 TCP/IP**

**OSI (Open System Interconnection, 개방형 시스템간 상호 접속)**

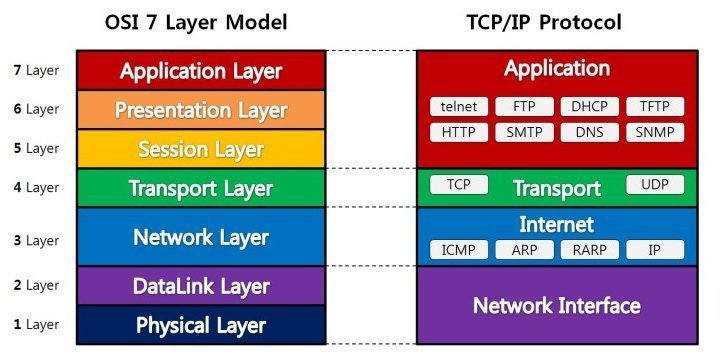
ISO(국제 표준화 기구)에서 컴퓨터의 통신, 네트워크간 상호접속을 용이하게 하기 위해 규정한 네트워크 프로토콜

실제 컴퓨팅 장치나 네트워크 장치를 만들 때 OSI참조 모델을 참조해서 모든 통신장치를 만든다.

표준화, 상호통신을 가능하게 한다.

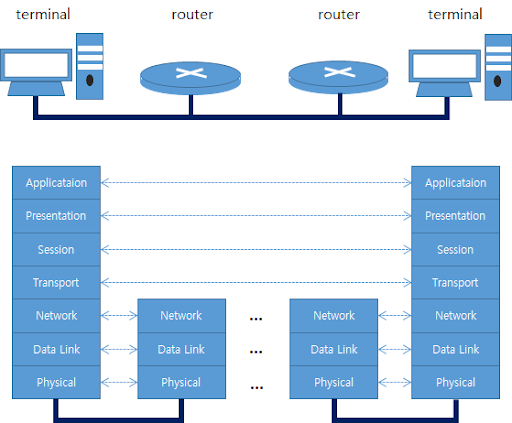
**OSI 7계층**

1. 네트워크에서 통신이 일어나는 과정을 7단계로 나눈 것
2. 계층마다 packet이름이 다르다 🡺 오류발생 시 오류발생한 단계를 쉽게 알 수 있다.
3. Encapsulation: 데이터 전송 시 7🡪1단계로 data뒤에 헤더가 20bytes씩 추가되어 전송
4. Decapsulation: 받을 때는 반대로
5. 계층을 나눈 이유
   * 통신이 일어나는 과정이 단계적으로 파악할 수 있기 때문
   * 특정 단계에 이상이 생기면 다른 단계의 장비 및 소프트웨어를 건들이지 않고 이상이 생긴 단계만 고칠 수 있다.
6. OSI 7 layer model (1~3계층(하위계층)은 하드웨어로, 4~7계층(상위계층)은 소프트웨어로 구현)



|  |  |  |  |
| --- | --- | --- | --- |
|  | Layer | 전송 단위 | 기능 |
| **1계층** | Physical Layer  (물리 계층) | Bits | 전기적, 기계적 특성을 이용하여 통신 케이블로 데이터를 전송  **데이터를 전기적 신호로 변환해서 주고받음**  알고리즘, 오류제어 기능이 없다.(이 계층만 없음)  장비: Hub(허브), Repeater(리피터), Cable |
| **2계층** | Data Link Layer  (데이터 링크 계층) | Frames | 오류없이 한 장치에서 다른 장치로 프레임(비트의 모음)을 전달하는 역할  **Hop-to-Hop delivery:** 목적지까지 자동으로 데이터가 전달되는게 아니다. **MAC주소를 이용하여 정확한 장치로 정보를 전달**  통신에서의 오류 찾기, 재전송이 가능  장비: Bridge(브리지), Switch(스위치) |
| **3계층** | Network Layer  (네트워크 계층) | Datagram | 다중 네트워크 링크에서 패킷을 발신지로부터 목적지로 전달할 책임을 갖는다  **From Source to Destination: 경로를 선택하고 IP 주소를 정하여 경로에 따라 패킷을 전달**  중요 기능: **데이터를 목적지까지 가장 안전하고 빠르게 전달하는 기능(라우팅)**  장비: Router(라우터, 네트웍과 네트웍을 연결) |
| **4계층** | Transport Layer  (전송 계층) | Segments | **Process to Process:** 전체메시지를 발신지 대 목적지 간 제어와 에러를 관리  **양 끝단의 사용자들이 신뢰성 있는 데이터를 주고받을 수 있도록 패킷들의 전송이 유효한지 확인하고 실패한 패킷은 다시 보내는 등 신뢰성 있는 통신을 보장한다.**  **port number(app별 고유 번호)를 부여한다**. (어느 application으로 보낼 것인지 지정)  프로토콜: TCP, UDP, ARP |
| **5계층** | Session Layer  (세션 계층) | Data | 통신 장치간 상호작용을 설정하고 유지하며 동기화한다.  사용자 간의 포트연결이 유효한지 확인하고 설정  연결 세션에서 데이터 교환과 에러 발생시의 복구를 관리 |
| **6계층** | Presentation Layer  (표현 계층) | Data | 데이터를 하나의 표현 형태로 변환  필요한 번역을 수행하여 두 장치가 일관되게 전송데이터를 이해할 수 있도록 함  Ex) 그래픽 등의 확장자 (jpg, gif, mpg) |
| **7계층** | Application Layer  (응용 계층) | Message | 사용자가 네트워크에 접근할 수 있도록 해주는 계층  응용 프로세스와 직접 관계하여 일반적인 응용 서비스를 수행  사용자 인터페이스, 전자우편, 데이터베이스 관리 등 서비스를 제공  Ex) 텔넷, HTTP, SSH, SMTP, FTP |

1. 데이터 통신



Internet: 여러 네트워크를 전세계적으로 연결한 컴퓨터 네트워크 (network of network)

Ethernet: 네트워크를 구성하는 방식 중 한 방법. 각 기기들이 MAC주소(다른 전자기기들과 유일하게 구분되는 식별자)를 가지고 이걸 통해 전가기기, 즉 호스트 간의 데이터를 주고받을 수 있는 방식. 호스트들을 연결시키는 데는 MAC주소를 식별할 수 있는 switch, router등이 활용. 식별불가능한 switch도 사용됨

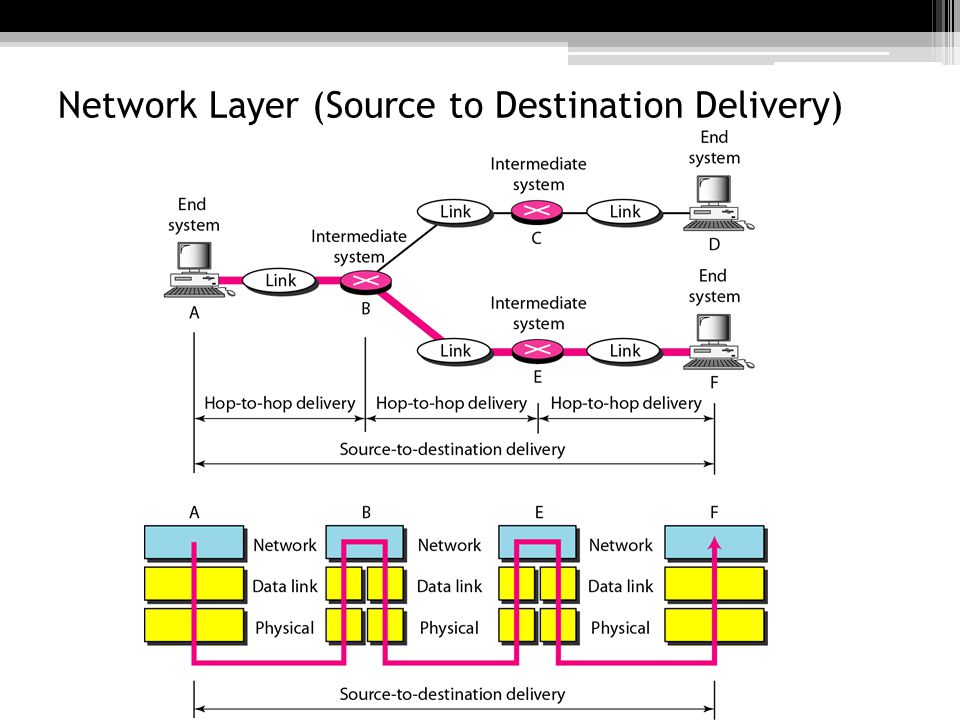
**Ethernet implementation (네트워크에서 physical layer와 data link layer)**

1. 같은 네트웍에 있는 컴퓨터들을 연결하는 장비들로 **멀티탭 역할**을 한다.
2. Filtering: 단순 컴퓨터들을 연결하는 역할 경우 각각의 컴퓨터들의 위치를 알 수 없기 때문에 데이터 전송 시 모든 컴퓨터에게 다 보내게 된다. 하지만 원하는 컴퓨터의 위치를 알고 있다면 모두에게 보내지 않고 해당 link로만 보낼 수 있다.
3. Physical layer 장비: repeater, hub 🡨 filtering 불가
4. Data link layer 장비: bridge, **switch** 🡨 **filtering 가능 (위치 모르면 모두에게, 알면 한 곳에만)**

**TCP/IP**

**TCP/IP 프로토콜은** OSI model을 기반으로 5,6,7단계가 하나의 단계 : **5계층**

1. 통신을 위해 각 단계에서 필요한 Address
   * Application layer: Application-Specific address
   * Transport layer: Port address (Port 주소) 🡨 application나타내는 번호
   * Network layer: Logical address (IP 주소)
     1. 여러 네트웍을 이용하기 때문에(?) 장비로 Router를 사용한다.
     2. Router: 2개 이상의 네트웍을 연결하는 장비로 각 네트웍마다 IP주소를 할당 받는다. 즉, 하나의 router가 LAN1에서는 F로 활동하고 LAN2에서는 T로 활동
     3. Router에는 목적지로 가기위한 다음 경로를 알려주는 routing table이 존재
     4. IP주소와 MAC주소가 1대1 mapping되는 ARP table이 존재한다
     5. 각 라우터에서 목적지로 가기위해 routing table에서 다음 이동할 IP주소를 찾고 (Source to destination) ARP table에서 해당 IP주소와 mapping된 MAC주소를 찾아 다음 장소로 이동한다. (Hop to Hop) 🡺 이를 목적지에 도착할 때까지 반복
     6. 라우터는 각 컴퓨터에도 존재
   * Data link layer: Physical address (mac 주소)
     1. 한 네트웍에서 이루어지는 통신
     2. mac주소: 컴퓨터마다 고유하게 정해진 번호 (48bit hex data로 이루어짐)
     3. 통신시 switch를 사용하며 switch는 컴퓨터들을 연결해 놓는 기능을 한다.
     4. Packet에 receiver의 mac주소를 작성함으로써 switch를 통해 연결된 모든 컴퓨터가 자신의 mac주소와 일치여부를 확인하여 일치하는 컴퓨터만 packet을 가져갈 수 있도록 한다. (위치 안다면 해당 mac주소 가진 컴퓨터로 연결)



🡸 physical address는 hop-to-hop으로 계속 변하지만 (실제 이동 용), logical address는 변경되지 않는다. (경로 찾기 용)

**네트워크 전달방식 비교**

네트워크 상에서 호스트와 호스트 간에 정보(data)를 주고받는 방식에는 2종류가 있다.

물리적 측면에서 연결기반인 circuit switching과 비연결기반인 packet switching이 존재

|  |  |
| --- | --- |
| Circuit Switching (회선 교환 방식)  = Connection Oriented (연결 기반) | Packet Switching (패킷 교환 방식)  = Connectionless |
| 중앙에서 경로 결정 | Router를 통해 분산으로 경로 결정 |
| Physical이 허용 가능한 크기까지 통째로 전송 | Packet단위로 전송 (500B ~ 1KB) |
| 연결이 끝날 때까지 경로 유지 (전용선 느낌) | packet들이 서로 다른 경로 선택가능 |
| 전화와 같은 실시간 통신에 사용 | Internet line |
| 사용되는 회선 전체를 독점하기 때문에 다른 사람이 끼어들 수 없다. | packet의 헤더에는 출발지와 목적지 정보가 존재 |
| 최대 성능 보장 (속도와 성능이 일정) | 성능 보장 안됨 |
| Bandwidth 낭비 (혼자 쓰므로) | Bandwidth 낭비 없음 (회선 공용) |
| 가끔 쓰면 비효율적(회선이 놀게 됨)이며 사고 발생 시 네트워크가 죽는다. | 혼잡 시 비효율(패킷 분실이 발생할 수 있다), 널널하면 효율(회선을 공용하므로) |

**Packet Switching**

* Packet switching은 물리적으로 회선을 공유하므로 비연결기반이다.
* Packet switching은 논리적 측면에서 데이터 전송 시 서버에 연결 요청을 하는 connection-oriented와 요청하지 않고 바로 보내는 connectionless로 나뉜다.

|  |  |  |
| --- | --- | --- |
|  | TCP | UDP |
| 논리적 측면 연결 | Connection-oriented | Connectionless |
| 속도 | network상황에 따라 다르다 | Application에 따라 다르다 |
| 1개 전송속도 | 연결 요청을 하므로 더 느림 | 연결 요청을 안하므로 더 빠름 |

**TCP (Transmission Control Protocol)**

TCP/IP protocol suite에서 Transport Layer에 존재하는 프로토콜

1. **서버와 클라이언트간 데이터를 신뢰성 있게 전달**하기 위해 만들어진 프로토콜
2. 데이터를 전송하기 전에 데이터 전송을 위한 연결을 만드는 **연결지향 프로토콜** 🡺 신뢰성 스트림 서비스
3. 데이터는 네트워크선로를 통해 전달되는 과정에서 손실되거나 순서가 뒤바뀌어서 전달될 수 있는데, TCP는 **손실을 검색해서 이를 교정하고 순서를 재조합** 할 수 있도록 해준다.

**특징**

* + Stream delivery: application이 만든 packet의 boundary를 유지하지 않고 나눠 보낸다.
    1. Application 🡪 TCP: 메시지(packet)을 보냄
    2. 각 패킷들은 크기가 일정하지 않으므로 TCP에서 정한 일정한 크기에 맞게 메시지를 잘라서 전송
    3. 즉, application이 만든 packet의 boundary를 유지하지 않고 이를 byte로 나눠 일렬로 나열한 후 정해진 상자 크기(Segment)에 맞게 잘라서 보낸다
    4. 따라서 한 메시지의 시작위치와 끝위치를 알지 못하고 상관하지 않는다.

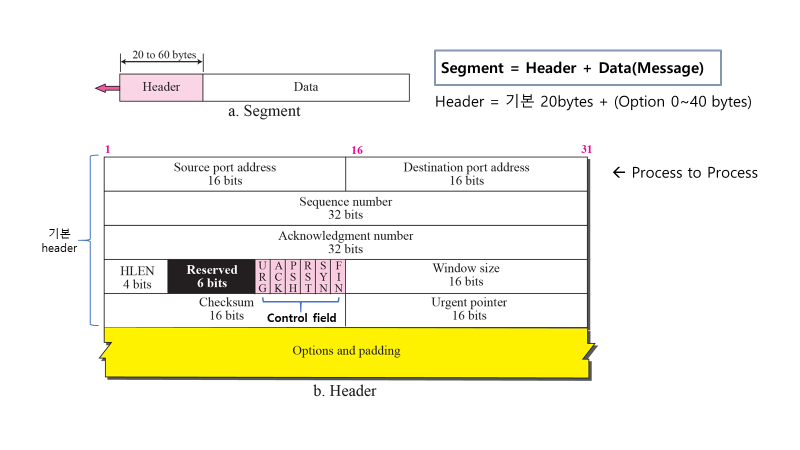
(↔ Boundary delivery: UDP에서 사용하는 방식으로 application이 만든 packet을 쪼개지 않고 보존하여 전송한다.)

* + TCP는 양방향으로 서로 주고받기가 가능하다
    1. TCP에는 sending buffer와 receiving buffer가 존재한다.
    2. 서버는 client당 buffer가 하나씩 존재 (buffer를 공용으로 사용하지 않는다.)
    3. Sending buffer: 3가지 구역으로 나뉜다.
       - Empty: 빈 공간
       - Sent: 전송했지만 전송 실패했을 경우 대비하여 관리
       - Not Sent: application에서 보냈지만 아직 전송하지 않은 상태
    4. Receiving buffer: 2가지 구역
       - Empty: 빈 공간
       - Not read: segment를 전달받고 제대로 된 메시지인지 확인하여 application으로 보냄
  + 각 Segment를 전송 시 번호(Sequence number)를 붙여서 보낸다. 🡨재조립시 사용됨
    1. 패킷들은 서로 다른 경로를 거치면서 전달되어 순서가 뒤죽박죽으로 전송된다.
    2. 전송되는 데이터의 Byte 순서 번호로 시작 byte부터 random하게 번호 부여
    3. Sequence number는 각 packet의 시작 byte번호이다.
  + Packet의 신뢰성을 보장하기 위해 Acknowledgement number 사용
    1. 한 segment를 전달받은 후 다음에 받기를 기대하는 segment의 시작 byte번호를 ack number로 상대에게 전달한다. (Cumulative)
    2. Cumulative: packet을 받은 후 다음에 받고 싶은 정보 보내기

(↔ Selective: 받은 것에 대한 정보 보내기)

1. TCP 전송 제어 방법: **흐름제어**(전송 데이터양 조절), **오류제어**(통신 도중 데이터 유실 or 잘못된 데이터 수신 시 대처 방법), **혼잡제어**(네트워크 혼잡 대처) (UDP는 흐름,혼잡제어 안함)

**TCP segment format**



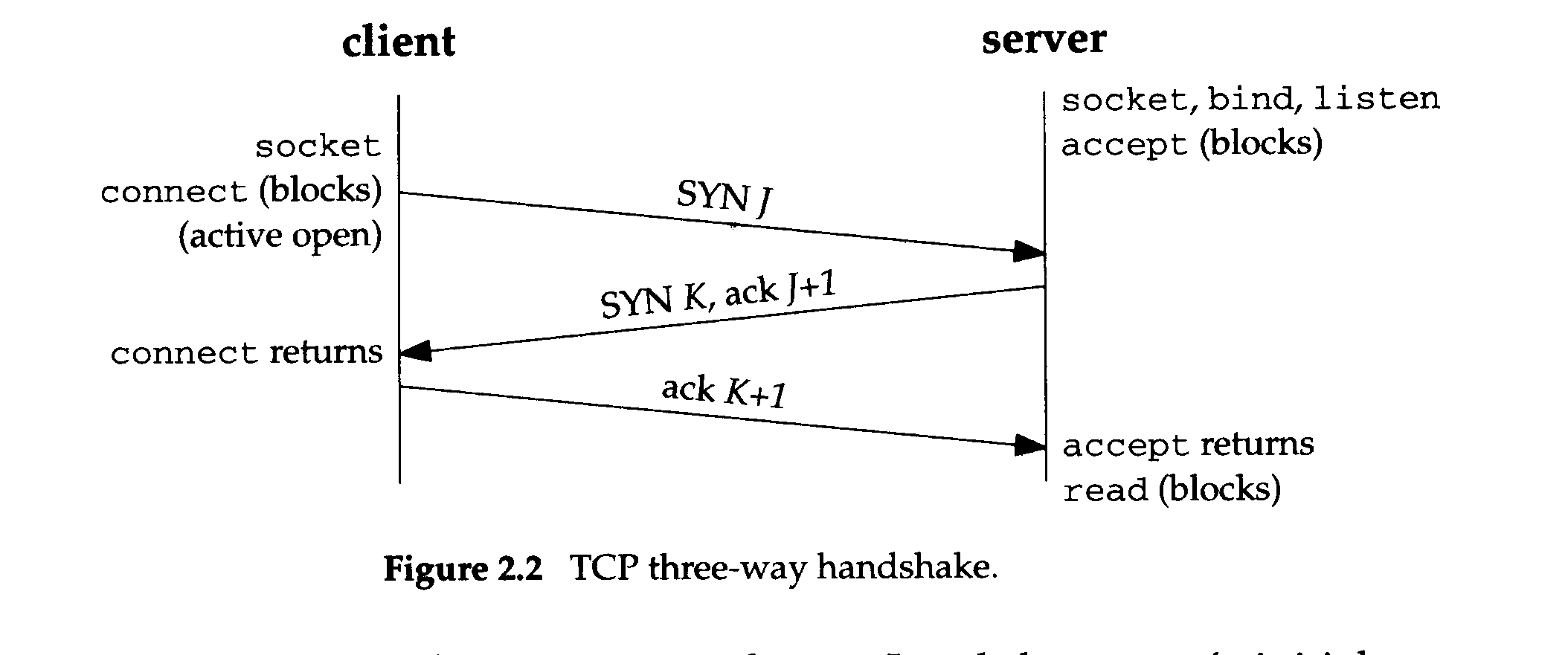
1. Source port: 데이터를 보내는 호스트의 TCP 포트
2. Destination port: 데이터를 받는 호스트의 TCP 포트
3. Sequence Number: TCP 세그먼트에 있는 첫번째 바이트에 대한 순번
4. Acknowledgement: 바이트에 대한 순번, 데이터를 전달받은 쪽이 받기를 원하는 number
5. HLEN: header의 길이저장. 기본20bytes에 option있을 경우 header길이, 경계 모르므로 데이터 시작부분을 알기 위해 저장. 전체길이/4의 몫을 저장한다.
6. Reserved: 예비 공간
7. Control filed: 각 1bit씩 존재하며 flag처럼 사용한다. 중요 filed는 ACK, SYN, FIN
   1. ACK: Acknowledgment is valid (데이터 전달받았다는 것)
   2. SYN: 연결 요청 packet
   3. FIN: 연결 종료 요청 packet
8. Window size: TCP 세그먼트를 보내는 호스트의 현재 TCP 버퍼 크기
9. TCP Checksum: TCP 데이터와 TCP 헤더의 error 유무확인
10. Option – MSS(Maximum segment size): segment에서 header를 뺀 data영역 (payload) size로 애플리케이션 계층에서 한번에 최대로 보낼 수 있는 데이터의 사이즈를 의미한다.
    1. 초기 set-up과정에서 SYN을 보낼 때 받고 싶은 size(MSS)를 상대방에게 전달하고 이는 set-up이후에 바꿀 수 없다.

**TCP connection**

TCP는 연결지향 (connection-oriented) 프로토콜이다. 물리적 연결지향(전용선 만들기)이 아닌 논리적 연결지향으로 서버에게 미리 연결요청 후 데이터를 보낸다.

(↔ UDP: connectionless 프로토콜로 연결요청을 하지 않고 일단 데이터를 보낸다)

1. **3way handshake (연결 set-up 과정)**

초기 연결 시 set-up과정으로 buffer 생성 + 데이터 송수신 준비를 완료한다.

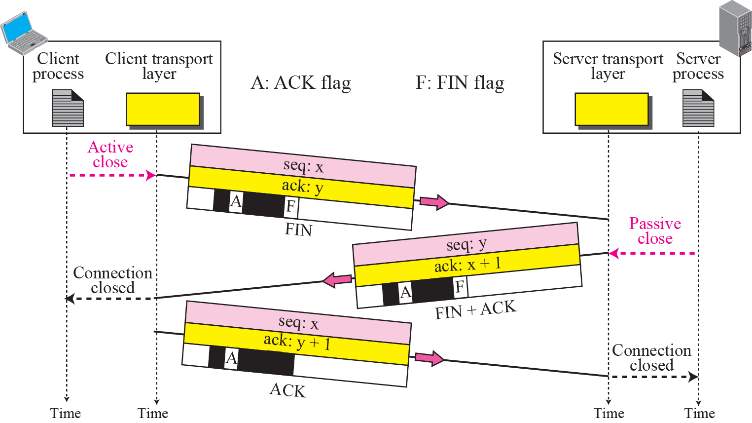
① C🡪S: **SYN** | 연결 요청(J: 랜덤번호)

② S🡪C: **SYN + ACK** |연결 요청 응답

③ C🡪S: **ACK** | 2번에 대한 응답(확인했다는 표시)

이후 C🡪S data 보낼 때 seq num: J+1로 전송, S🡪C data 전송 시 seq num: K+1로 전송

1. 연결 종료 과정 (3way handshake) 🡪 4way handshake도 가능(ACK과 FIN을 따로 보내는 것)

서버로부터 서비스 다 받고 client가 먼저 종료 요청한다. 양방향이므로 서로 FIN전달하며 해당 buffer를 지운다.

① C🡪S: **FIN** | 연결 종료요청

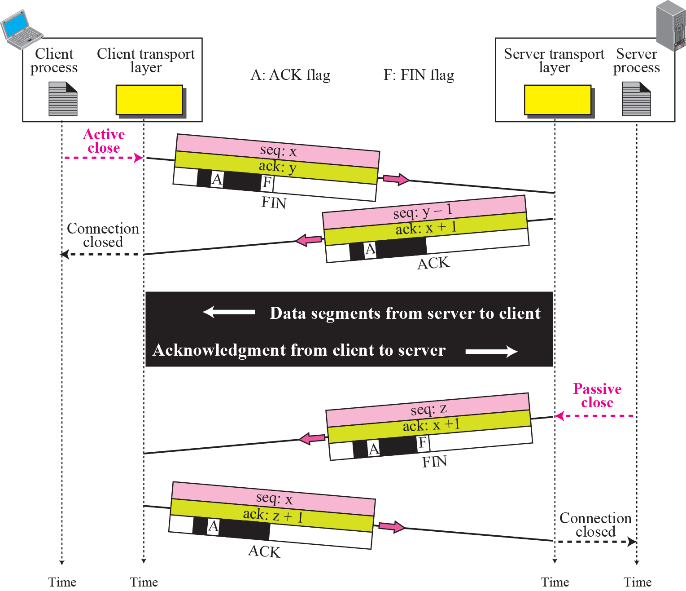
② S🡪C: **FIN + ACK** |연결 종료요청 + 응답

③ C🡪S: **ACK** | 2번에 대한 응답(확인했다는 표시)

☆) client는 서버로부터 FIN+ACK을 받은 후 바로 종료되지 않는다. TIME\_WAIT상태를 1~2분 정도 유지 후 closed로 이동하며 완전 종료된다. 🡺 바로 closed로 가지 않는 이유

1. S🡪C FIN+ACK에 대한 응답으로 C🡪S ACK을 전송한다. 이때 서버는 자신이 보낸 FIN+ACK이후 일정시간 동안(TIME-OUT) 상대로부터 ACK을 받지 못하면 FIN을 다시 보낸다. 이경우를 대비하여 client는 바로 종료하지 않고 TIME\_WAIT상태를 일정시간 유지
2. Server와 client간 연결이 종료된 후 다시 연결될 수 있으며 이때 문제 발생하지 않도록 TIME\_WAIT상태를 일정 유지한다. 서버는 client buffer를 찾을 때 (IP주소, port num)쌍으로 이루어진 client ID를 이용한다. 이때 만약 같은 client가 같은 port num을 가지고 2차연결한 뒤 1차연결 때 packet이 늦게 전송되면, 이 packet과 2차연결 때 packet이 구분되지 않는다. (1차와 2차가 IP주소와 port num이 같기 때문) 이 문제를 해결하기 위해 client 연결 종류 시 TIME\_WAIT상태를 일정시간 유지함으로써 이 기간동안 같은 client가 server로 연결요청 보낼 때 생성되는 port number를 이전 number와 다르게 설정한다.
3. Half-Close (한쪽만 FIN을 보내어 반만 종료된 상태)

**FIN**을 보낼 경우 **보내는 쪽은 sending buffer**가, **받는 쪽은 reading buffer**가 **사라진다**.

C🡪S로 FIN+(ACK)을 보낼 경우

1. client의 sending buffer 사라짐
2. server의 reading buffer 사라짐

S🡪C로 FIN+(ACK) 보낼 경우

1. client의 reading buffer 사라짐
2. server의 sending buffer 사라짐

🡺 client가 더 이상 보낼 정보가 없을 때 FIN을 보낸다. 이 경우 client는 더 이상 데이터를 보낼 수 없고 받기만 가능하다.

Server는 받기 불가능 보내기 가능하다.

**Flow control (흐름 제어)**

1. **데이터를 수신하는 노드가** 전송하는 노드에게 **현재 자신의 상태에 대한 정보를 보내주는 것**
2. 흐름제어 하지 않을 경우: **송신자의 데이터 전송 속도가 수신자의 데이터 처리 속도보다 빠를 경우** 데이터 양이 수신 버퍼 크기를 넘게 되며 **데이터 손실이 발생**
3. Stop & Wait 방식: 신뢰성을 보장하기 위한 데이터 전송의 흔한 방식으로 데이터를 보낸 후 상대방에게 ACK오면 그 때 데이터를 보낸다. 비효율적 (ACK이 올 때까지 기다려야 한다.)
4. Sliding-Window 방식: stop & wait를 개선한 방식
   * ACK을 기다릴 필요 없이 각 패킷에 번호를 붙여서 연속적으로 보내기 (단, window size만큼만 보내기 가능, 보내는 양을 조절한다)
   * Window size(N): packet1을 보내고 packet1에 대한 ACK을 받기전까지 보낼 수 있는 최대 양, ACK에는 수신자의 Receive Window Size(rwnd)가 같이 보내지며 N이 조정된다.
   * Window size = Min(rwnd, cwnd) (cwnd는 네트워크 상황을 고려한 혼잡제어 크기)
   * 수신자 buffer의 빈공간 생기는 속도는 application이 가져가는 속도로 속도가 빨라지면 N또한 커진다. 즉, 처리 속도에 따라 N은 가변적이다.

Ex) sending buffer: 10000B, receiver buffer: 5000B, 1packet당 1KB

* N: 5000B, 1pacet당 1000B이므로 5개의 패킷으로 나눠 5000B를 보내기. 그리고 ACK받을 때까지 기다림. 이 이상은 전송하지 않는다.

1. **Silly Window Syndrome**: 보내는 양이 작아서 header size overhead가 커지는 상황
2. 송신 측에서 발생하는 신드롬
   * 발생이유: application이 너무 천천히 TCP sending buffer에 데이터를 보내서 발생
   * 해결: Nagle 알고리즘
     1. 첫 데이터라면 size 상관없이 window size크기이내에 보냄
     2. 이후, 데이터가 buffer에 들어와도 MSS(Maximum segment size)를 만족할 때까지 전송하지 않는다. (packet하나 완성되면 보냄)
     3. MSS만족하지 않을 경우 다음 ACK 올때까지만(RTT(round trip time, 왕복시간)) 기다리고 보냄 🡺 계속 기다리면 application에 영향을 주기 때문
3. 수신 측에서 발생하는 신드롬
   * 발생이유: 수신 측에서 application이 처리하는 속도가 느려 rwnd크기 작게해서 보냄
   * Clark 해결방법
     1. Buffer의 가능 size가 MSS크기 or buffer/2크기가 될 때까지 rwnd=0으로 전송 (받을 공간 없으므로 전송하지 말라는 의미)
     2. Rwnd=0을 받은 송신측은 send를 중단한다.
     3. Buffer 가능 size가 MSS크기 or buffer/2크기가 되면 rwnd값을 제대로 전송
   * 해결2: 확인응답의 지연 (ACK전송 지연)
     1. ack안오면 client에서 window size크기만큼 전송하다가 stop된다.
     2. 일정 크기 이상의 빈 공간 생겼을 때 수신 측에서 ACK을 보냄으로써 송신측 window size 조정이 가능해진다
4. raw socket 경우 header를 내마음대로 만들 수 있다. 즉 보내는 쪽 IP주소를 바꿔 보내면 SYN 🡪 SYN + ACK 이후 server가 ACK을 받을 수 없다. 계속해서 IP를 변경하여 보내게 될 경우 server의 연결 요청 대기 큐가 꽉 차서 일반 사용자들의 연결 요청을 받을 수 없다.

* DoS: 한명이 공격하는 방식 🡪 네트워크, 서버 살아있는데 서비스 불가능하게 됨
* DDoS: 특정 client에서 무더기로 요청 오면 감지가 가능, 분산적으로 여러 사람이 공모해서 공격하는 방식

**Error Control (에러 제어 – packet이 없어지는 경우)**

미리 알 내용

1. Ack에 대한 Ack은 없다
2. Packing switching network이므로 순서 바꿔서 올 수 있다. 이를 TCP에서 해결해주기 때문에 Application은 신경쓰지 않아도 된다.

에러제어는 checksum, acknowledgement, time-out을 통해 달성한다.

ACK 규칙

1. ACK의 개수를 줄이자. (기본 연산)

**Rule1**: 보낼 data가 존재한다면 data와 ACK을 하나의 packet으로 보내기

**Rule2**: ACK 지연 – 500ms 기다려서 application에서 data 온다면 같이 보내기

**Rule3**: 받은 packet 2개당 ACK한개 보내기(그 이상이면 ACK 사라졌을 때 손실이 큼) – 500ms기다리다 2개 들어오면 바로 보내기 (ACK-delaying timer: 500ms, packet받은 후부터 타이머 시작)

1. 응급 상황 시 ACK 바로 보내기

RTO timer: 보내는 packet마다 timer 존재, packet 보낸 후 타이머 작동되고 time-out까지 ACK받지 못하면 재전송한다. (RTT(왕복시간)을 이용해 구하므로 혼잡상태에 따라 값 달라짐)

**Rule4**: packet이 없어진 것을 감지하면 500ms기다리지 않고 바로 ACK보냄(ex.원하는 패킷이 오지 않고 그 다음 패킷이 도착)

**Rule5**: 잃어버린 packet이 오면 500ms 기다리지 않고 바로 ACK보내기

**Rule6**: 중복된 packet 받으면 바로 Ack 보내기

+) 3 duplicated ACK: 중복된 ACK 세번 받는 것을 의미 🡪 packet-loss를 감지하여 time-out까지 기다리지 않고 다시 packet 보낸다.

+) Ack 손실: 다음 seq요청 ack 손실되고 이를 건너뛰고 client가 RTO timer 종료 전 다음 ack받으면 이전 것 문제없다고 판단하여 큰 문제 생기지 않는다.

+) Ack 손실이 deadlock을 일으키는 경우: rwnd=0 보낸 후 적정 빈 공간 생겨 rwnd=k보낼 때 이 packet이 손실된 경우

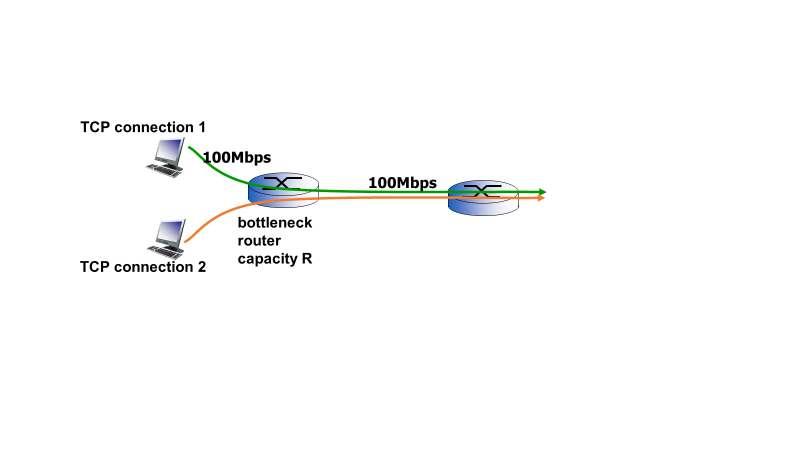
* Packet 전송을 중단한 상태에서 rwnd=k 패킷이 손실된다면 클라이언트와 서버 둘 다 보낼(받을) 수 있는지 계속 기다리게 된다. 🡪 데드락
* 해결: 송신 측에서 수신 측으로 probe(1B, 원래 data는 500B)를 주기적으로 보내어 ACK과 함께 rwnd 정보를 계속 받아 packet을 보낼 수 있는지를 확인한다.

**Congestion control (혼잡 제어)**

네트워크 상황을 고려하여 혼잡이 발생하지 않도록 한다. (공용선을 사용하므로 혼잡 발생)

혼잡 제어는 cwnd(congestion window)와 혼잡정책을 통해 달성

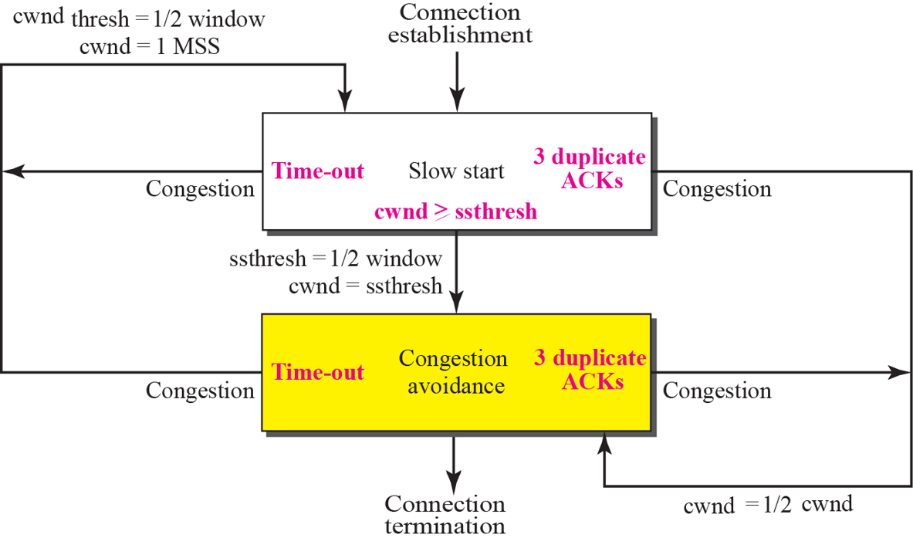
**병목현상(Bottlenect)**: 상대적으로 적은 공간에 많은 요청이 들어와 성능이 저하되면서 전체 시스템에 영향을 미친다. 즉, input보다 output이 더 작은 경우이며 버퍼 크기를 넘게 되면 packet을 버린다. 🡺 packet 손실의 대부분 이유 (혼잡에 의해 router가 버림)



혼잡 발생 🡪 packet 손실 🡪 delay 증가 🡪 throughput(시간당 받는 양) 감소

CWND 있는 이유: 흐름 제어는 네트워크는 고려하지 않고 상대방만 고려하여 window size를 조정한다. 이 경우 size가 너무 커지면 네트워크 죽을 수 있으므로 네트워크를 고려한 cwnd도 사용

혼잡 정책

* + 중앙통제가 아니므로 보내는 양 스스로 조절해야 한다.
  + Threshold: default값, 초기 외부 혼잡과 상관없음
  + SS(slow start): 처음시작 시 cwnd=1부터 threshold까지 2배씩 값을 증가시킨다.
  + AI(addictive increase): threshold 이후 1씩 값 증가
  + MD(multiplicative decrease): 혼잡 발생시 절반으로 값 줄여서 보내기

!) 값 증가는 RTT(왕복시간)마다 일어난다.

**TCP Timers**

Retransmission timeout(RTO): 재전송을 위한 타이머로 RTT(round trip time)을 이용해 계산된다.

Persistence timer: 교착상태를 해결하기 위해 사용, 영속타이머가 만료되면 probe 세그먼트 전송

Keepalive Timer: 오랜 기간 동안 idle 상태에 있는 것 방지, 서버가 2시간 동안 클라이언트로부터 세그먼트를 전송받지 못하면, probe 세그먼트 전송

* + 소프트웨어적 클라이언트가 죽으면 FIN 전송하지만 하드웨어적 클라이언트가 죽으면 FIN전송하지 않는다.

TIME-WAIT: FIN🡪FIN+ACK🡪ACK이후 들어가는 상태 (앞에서 설명)

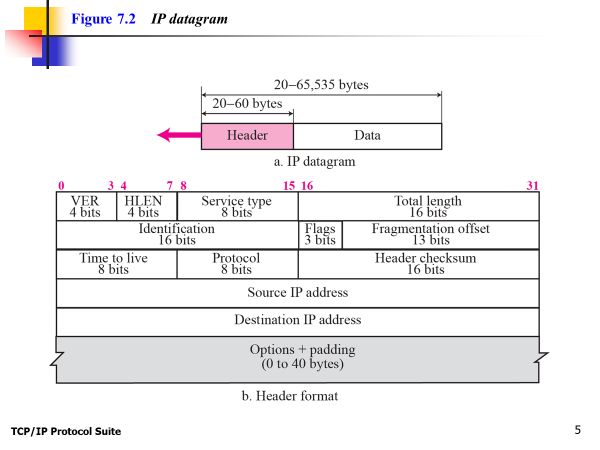
**IP**

TCP/IP protocol suite에서 network Layer에 존재하는 프로토콜

Hearder: routing과 delivery에 필요한 정보를 포함한다.

(routing: 송신 측에서 수신 측까지의 경로를 설정하는 행위)

**IP datagram format**

header: 기본 20B + 옵션 0~40B

VER: 버전을 알려준다. (IPv4와 IPv6)

HLEN: 헤더길이

Service type: 우선순위와 type적는용으로 거의 안 씀

Total length: IP datagram의 전체크기 (크기가 46B이상이면 패딩 없다)

Identification: ID, 같은 packet이라는 의미(단편화 후 재조립 때 사용됨)

Flag: 3bit중 뒤에 2bit만 사용한다.

D: 데이터가 온전히 보내지기 원함 유무 🡪 못쪼개면 버린다. M: 뒤에 쪼개진 fragment 존재여부표시이며 마지막일 경우 0, 존재하면 1이다 몇 개로 쪼개졌는지 정보 알 수 없으므로 fragmentation offset으로 순서 맞춘다.

Fragmentation offset: 네트워크별 MTU에 의한 단편화 후 재조립시 offset을 사용해 순서 맞춘다. 단편화할 때 data영역의 나눠지는 부분의 위치를 적음



Time to live(TTL): 이동 가능한 hop개수 🡺 router가 routing table잘못 만들면 loop이 생성되고 ack 계속 반복한다. 이를 막기 위해 hop이동 될 때마다 값이 감소하고 0이되면 이 packet 소멸된다.

Protocol: 같은 네트워크계층 혹은 상위 계층에 존재하는 해당 프로토콜로 전송 (TCP, UDP, ICMP, IGMP, OSPE)

Header checksum: 헤더의 에러를 체크한다. (TCP는 데이터까지 check)

이더넷 프레임에서 datagram의 캡슐화: datagram의 길이가 46B미만이면 padding을 추가해서 46B를 완성해서 전송한다. Ex) 실제 data가 1byte이면 TCP, IP 헤더 20B씩 이므로 총 41B 🡺5B의 패딩을 추가

**Fragmentation(단편화)**

데이터그램의 크기가 크면 MTU의 크기에 맞게 datagram의 데이터를 나누는 것

MTU: Maximum Transfer Unit, 데이터그램의 최대크기(캡슐화 시 datagram의 min=46B, max=MTU)

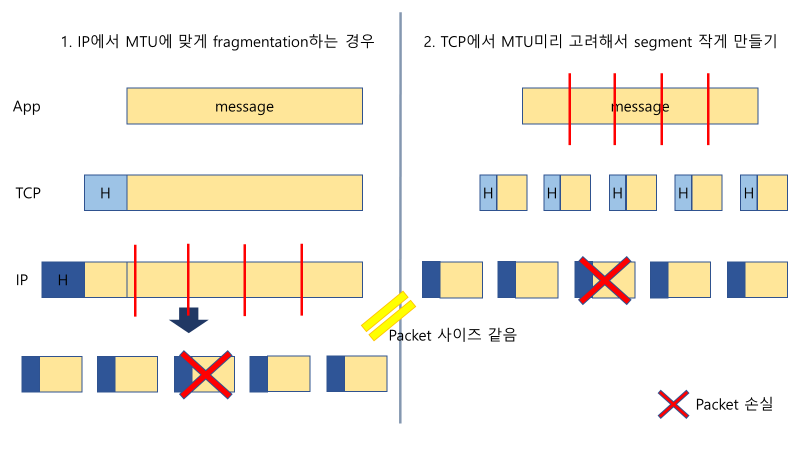
네트워크별로 MTU가 다르기 때문에 해당 네트워크에 맞게 MTU보다 크면 단편화를 실시한다. 이때, header는 계속 쫒아감

단편 재조립: 단편들로부터 원래 데이터그램을 생성 (최종 목적지에서 재조립된다, 다 도착 시 offset을 보고 순서를 맞춘다) 🡺 라우터에서 전달 소프트웨어의 부담을 완화, 동일한 데이터그램의 서로 다른 단편이 공통의 목적지로 갈 때, 다른 경로를 통해 가는 경우 발생(즉, 재조립이 불가능할 수 있음)

단편화 단점: 단편 손실 시 데이터 그램 전체를 재전송 해야함

재조립 타임아웃: IP계층은 데이터그램의 어떤 단편이 처음 도착하면 타이머를 시작하며 일반적인 타임아웃은 30~60초, 타임아웃 시간이 지나면 단편이 폐기된다.

\*) TCP에서 미리 mss로 segment 나눌 경우와 나누지 않고 IP에서 단편화 경우 패킷 손실 비교



1번 경우:

1. IP에서 단편화되므로 나눠진 5개의 packet은 같은 ID를 가진다. 따라서 목적지 IP에서 재조합시 Time-out까지 대기하다가 나머지 도착한 4개의 packet도 모두 버린다.
2. 받는쪽 TCP는 이 data를 받지 못하고 그 이후 3개의 data를 받은 후 3DA에의해 packet loss감지 후 재전송을 요청한다.

2번 경우: TCP에서 미리 나눠져서 보내지므로 각 packet은 다른 ID를 가진다. 따라서 packet이 1개 사라져도 나머지 4개는 제대로 전송되고 재조립을 시간 절약, 3DA에 의해 loss감지후 재전송

결론: IP는 TCP와 달리 특정 packet만 재전송 요청할 수 없다. 따라서 관련 packet하나가 사라지면 전체가 다 재전송되어야 하고 packet loss 감지가 오래걸린다. 그러므로 우리는 2와같이 행동하도록 하면 이 역할을 MSS가 해준다.

\*) TCP header에 MSS역할: data보낼 때 mss로 내 mtu고려해서 최대 이정도로 segment 크기 만들어서 보내줘! 🡺 1번을 좀 더 예방할 수 있으며 2번처럼 작동될 수 있다.

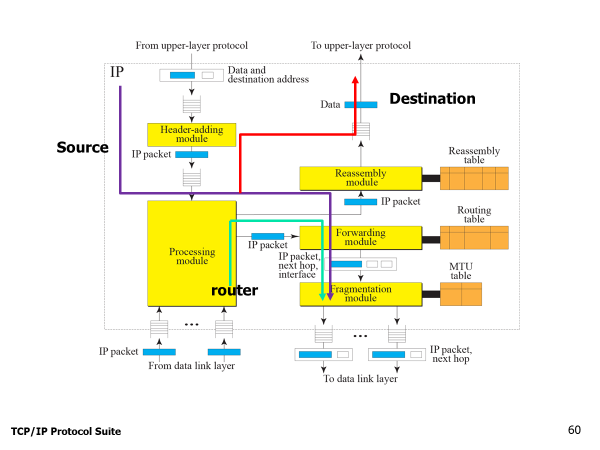
**Options (datagram format에서 option부분)**

1. Single-byte
   1. No operation: 옵션간 경계를 나타냄 (여러 옵션 있을 때)
   2. End of option: 옵션 필드의 끝을 나타냄
2. Multiple-byte
   1. Record route: 패킷의 전달 경로를 기록 (전달될 때마다 차례로 기록된다)
   2. Strict source route: 전달 경로를 packet에 미리 지정하고 그 경로를 따라서만 패킷을 전송하고자 할 때 사용하는 옵션
      1. Network 관리자가 사용하며 주로 길을 만들고 그 길로 잘 이동하는지 검사할 때 사용한다.
      2. 이경로로 못가게 되면 packet을 버린다. (경로 이상시 ICMP메시지 전송)
   3. Loose source route: strict source route와 마찬가지로 packet을 전송하며 다른 점은 주소 목록에 없는 경로를 사용할 수 있다. 즉, 정해놓은 주소를 거쳐가기만 하면 된다.
   4. Timestamp: 패킷 경로상의 라우터에 도착하는 시간까지 기록, 3가지 유형 존재
      1. 시간만 적기
      2. 시간 + IP주소
      3. Source routing + 시간적기 🡺 주소가 미리 주어지고 들른 시간 적기

\*) 단편화 시 각 fragment에 복사되어야 할 option은 source routing관련 option이다.

\*) ping: network set-up 잘 돼있는지, 상대방 살아있는지 확인할 때 사용/ 경로는 알 수 없다.

**IP header 만드는 과정 (패킷의 현 위치 별 경로가 다르다)**



**Delivery and Forwarding**

네트워크 레이어는 하위의 레이어에서 패킷이 움직이는 것을 다루는 역할을 하며 이를 패킷 delivery라고 한다.

Delivery에는 Indirect delivery와 direct delivery가 존재

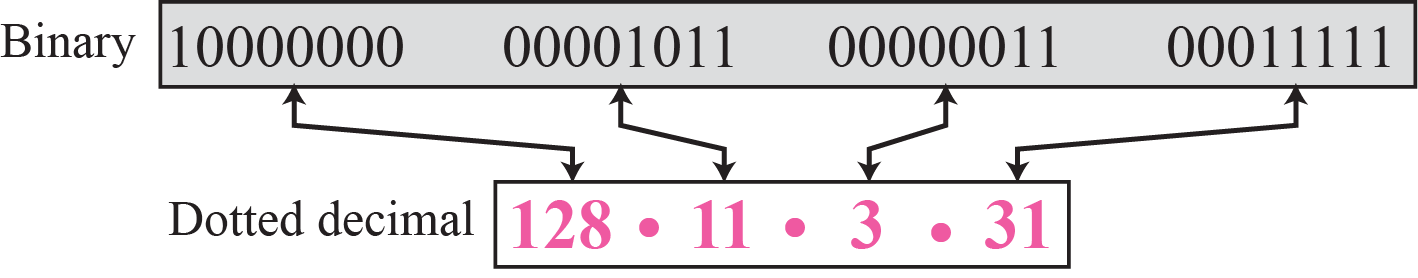
* + Direct delivery: 보내는 쪽과 최종 목적지가 같은 네트워크인 경우로 host🡪dest or router🡪dest가 같은 네트워크에서 이루어지는 것을 의미
  + Indirect delivery: 네트워크의 이동이 있는 경우로 router를 거쳐 전달되기 때문에 routing table이 필요하다.

Forwarding: 요즘 인터넷은 네트워크의 조합으로 구성되기 때문에 다음 hop까지 패킷을 전달하는 것을 의미

* + IP 프로토콜은 비연결형 프로토콜이지만 요즘 경향은 이를 연결 지향형 프로토콜로 변경하는 것이다.
  + 목적지 주소 기반 forwarding: IP가 비연결형 프로토콜로 사용될 때 사용
    1. 포워딩을 위해 routing table을 이용해서 다음 hop을 찾는다.
    2. Routing table은 네트워크 주소, 다음 라우터, 인터페이스 주소로 구성된다. / IP주소가 classful일 경우 class별로 테이블이 존재하고 classless경우 목적지 주소의 네트워크 주소를 알기 어려우므로 테이블에 마스크(/n)을 포함시킨다.
    3. 주소 집단화: 클래스 없는 주소지정을 사용할 때 모든 주소 공간을 관리 가능한 블록으로 나누기 때문에, 포워딩 테이블 항목 번호가 많아질 가능성 높다. 테이블의 크기가 증가하면 테이블을 검색하기 위한 시간이 증가한다. 이를 해결하기 위한 방법으로 테이블에서 next hop과 interface가 같은 주소 블록들을 묶어 더 큰 주소 블록으로 만들고 하나의 인터페이스로 합친다. (서로 다른 네트워크)
    4. 테이블은 항상 마스크가 긴 마스크부터 짧은 마스크 순으로 정렬. 마스크 짧은 것이 위에 있으면 잘못 전달될 가능성이 있다.
  + 레이블 기반 forwarding: IP가 연결 지향형으로 사용될 때 사용
    1. 1980년대에 IP를 연결 지향 프로토콜처럼 동작하고자 하는 시도가 시작됨.
    2. 비연결형 네트워크에서는 목적지 기반으로 패킷을 포워딩한다. 반면 연결 지향 네트워크에서는 패킷에 부착된 레이블을 기반으로 패킷을 포워딩한다. 라우팅은 기본적으로 테이블 내용에 대한 탐색을 기반으로 하는 반면 교환은 인덱스를 사용한 테이블 접근방식에 의해 수행된다.

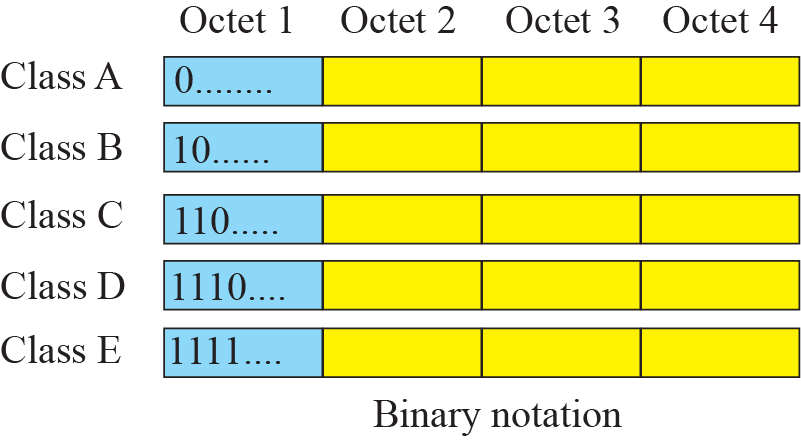
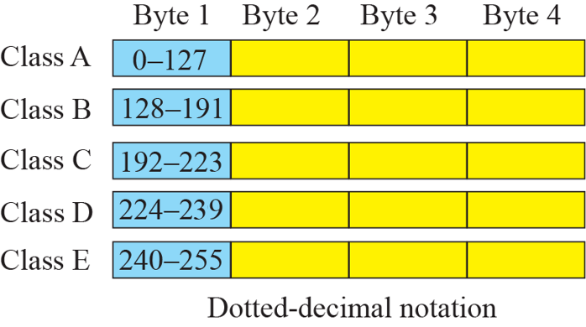
**IPv4 Address**

1. IP 계층에서 사용되는 Internet에 연결된 각 장치를 식별하기위한 식별자
2. IPv4 address: 32bit 주소이며 전세계 하나씩 밖에 없다.
3. 현 네트워크 IP주소 체계는 IPv4를 사용
4. 표기: 실제는 32bit 숫자 1개 🡪 읽기 힘들어서 1byte씩 총 4byte로 끊어 표현

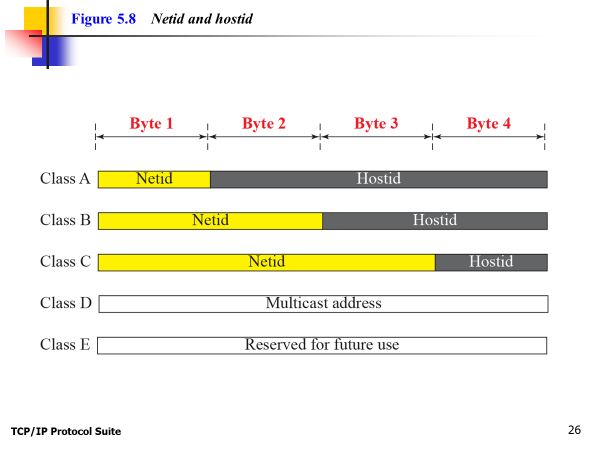
각 byte는 0~255범위 값

**주소 표현 방식**

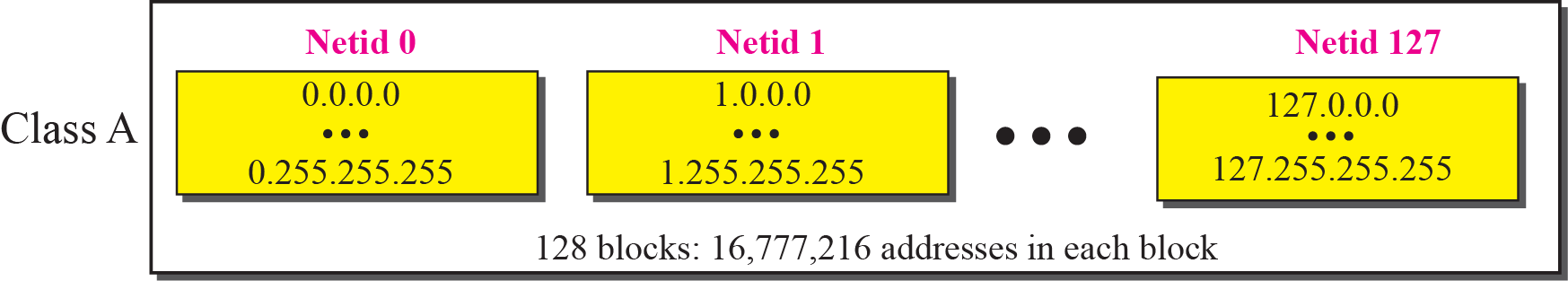
1. Classful addressing: 인터넷 상의 IP주소를 규격화된 크기별(클래스별)로 구분하는 방식
   * IP 주소를 Class (A, B, C 등)별로 규격화(유영화)시켜, 쉽게 식별하도록 함
   * 총 5개의 class(A, B, C, D, E)를 가지며 구축하고자 하는 네트웍의 크기에 따라 크게 A, B, C로 나뉜다. (D, E는 잘 안씀)
   * 32bit로 약 40억개의 주소가 존재하며 클래스별 비율을 나누면 A 50%, B 25%, C 12.5% 등으로 비율이 점점 절반씩 낮아진다.
   * 그림과 같이 binary 표현 경우 맨 앞만 보고 class구별 가능하며, 10진수 표현 시 byte1에 표기된 값의 범위에 따라 class를 구별할 수 있다.

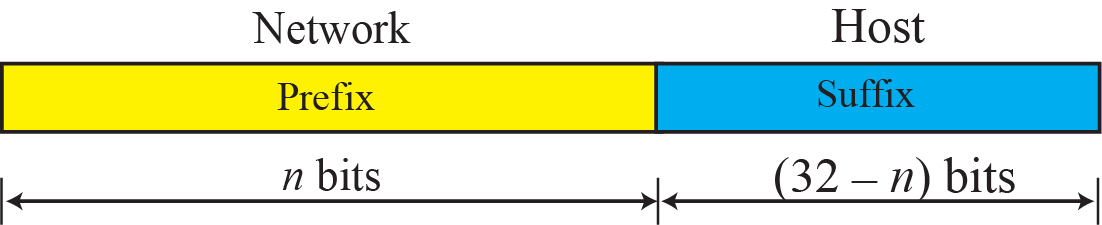
* + IP주소는 network ID와 Host ID로 구성되며 class별 구성 비율이 다르다. Host ID의 비율이 클수록 구축가능한 네트웍의 크기가 크다.



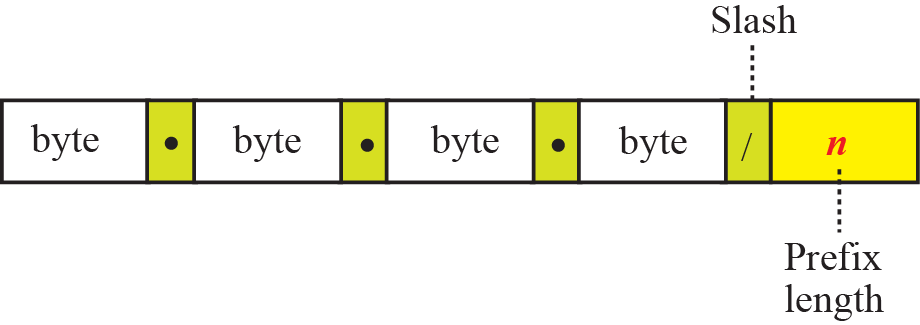
* + 각 네트웍은 네트웍 ID가 같다. (예를 들어, class A에 대해 각 block씩 network형성됨)



1. Class A: 크기가 크기 때문에 상당부분 낭비된다. (내가 쓰지 않는 주소 양도 불가능)
2. Class B: 한 블록당 16bit가 할당된다. (host ID로) 🡪 A보단 덜하지만 그래도 낭비
3. Class C: 한 블록당 8bit가 할당(256개의 주소 존재) 🡪 애매하게 크기 작음
4. Class D: 단일 블록으로 존재하며 multicast 주소로 쓰인다.
5. Class E: 단일 블록으로 존재하고 나중을 위해 예약 용도이다.
   * 주소가 주어지면 ①class 구별 (첫 byte이용) ②class별 netID찾기 순서로 일을 수행한다.
   * 어느 한 개의 주소를 통해 그 네트웍의 시작주소, 마지막주소, 크기를 알 수 있다.
   * 한 네트웍의 처음과 끝 주소는 할당되지 않는다. 첫 주소는 ID처럼 쓰인다. (네트웍 주소)
   * 네트웍 주소 존재 이유: router내에 routing table은 주소와 다음 이동경로를 저장한다. 이때 전체주소에 대해 테이블을 만들면 양이 너무 크다. 실제로 네트웍으로 들어가기 전 외부에서는 세부적인 목적지 주소를 알 필요없이 네트웍 주소만 알면 목적지가 있는 네트웍으로 이동 가능하다. 즉, router는 network ID만을 가지고 routing table을 관리하며 목적지 이동 경로 탐색하기 위해 목적지의 netID만 마스킹해서 routing table을 검색한다.
   * Subnet: 하나의 network으로 관리할 경우 네트웍이 너무 크기 때문에 관리가 비효율적이며 보안에 취약하기 때문에 subnet으로 나눠서 관리한다.
     1. 보안: broadcast경우 switch로 연결된 모든 장치에 들린다. 또한 내 위치 모르면 모두에게 전송되므로 보안에 취약하다.
     2. Subnet들은 모두 하나의 네트웍이므로 네트웍 ID같다.
6. Classless addressing: Class라는 규격화된 구분없이 비트 단위별로 IP주소 범위를 가변적으로 구분하는 방식
   * Class별 나눠져 있는 경우 주소의 낭비가 발생할 수 있다. Classless 주소방식은 무작위 나열로 필요한 만큼 가져오면 된다.
   * Class는 byte단위인 반면 classless는 bit단위로 끊어 가져간다.
   * Classless IP주소는 위와 마찬가지로 network과 host영역이 존재하고 network영역은 prefix, host영역은 suffix라고 부른다.

 🡨 prefix의 길이: 1~32

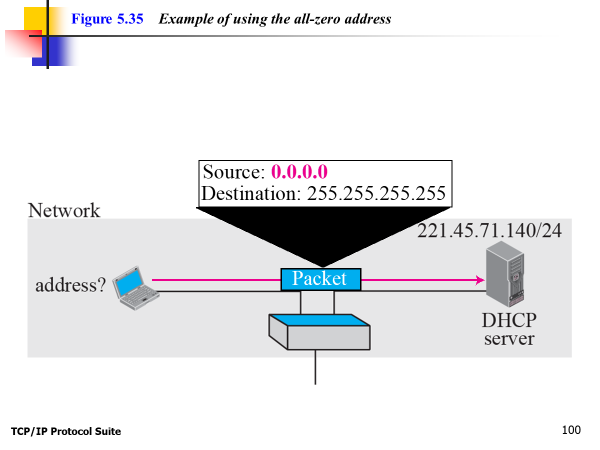
* + prefix길이: n일 때 가능 block의 개수 2^n, 한 block당 size 2^(32-n)
  + 표기: 주소 옆에 slash를 사용하여 prefix의 길이를 적는다.



1. Special address (classful addressing)

고정 IP: 잘 쓰지 않는 것까지 할당되므로 낭비가 심하다 / 외부에서 접속하기 편하다.

DHCP(유동 IP): 가정집에서 유동 IP를 사용한다.

1. 이 경우 컴퓨터를 킨다음 server에게 IP주소 할당을 요구한다.
2. 이 때 IP header는 [0.0.0.0 (all-zero address), 255.255.255.255]로 구성된다. 목적지의 이런 구성을 broadcast라고 한다.
3. 
   1. Unicast: 1:1 communication (server : client)
   2. Broadcast: 1:all (주소자체가 틀림, 모두에게 전송)
      1. Limited broadcast: 다른 네트웍으로 안넘어가도록 block 처리
      2. Directed broadcast: 한 block에서만 쓸 수 있도록 설정한 것으로 suffix가 다 1이다. (block의 마지막 주소를 broadcast용으로 사용하며 router가 많이 보낸다)
   3. Multicast: 1:N (특정 그룹한테 전송)
   4. Loopback address: 한 컴퓨터에서 client와 server를 설정하고 네트워크 프로그래밍이 잘 동작하는지 보는 것 (127로 시작하는 주소)
   5. 사설망에서 쓰이는 주소가 따로 존재하므로 주소를 보고 IP할당 받은 건지 사설망인지 구분 가능하다.
4. NAT (network address translation)

한 네트워크의 크기를 더 키우기를 원할 때 더 큰 크기에 주소들이 이미 다른 네트웍에 할당된 경우 문제를 해결하기 위해 사용됨.

1. 사설망에서 쓰이는 IP주소를 public 주소로 바꿔서 보내준다. (사설망 주소 여러곳마다 다 쓰기때문에 보내는건 되는데 NAT있어야 응답을 받을 수 있다.)
2. 응답Packet이 도착하면 기존 사설망 IP주소로 바꾼후 전송한다.

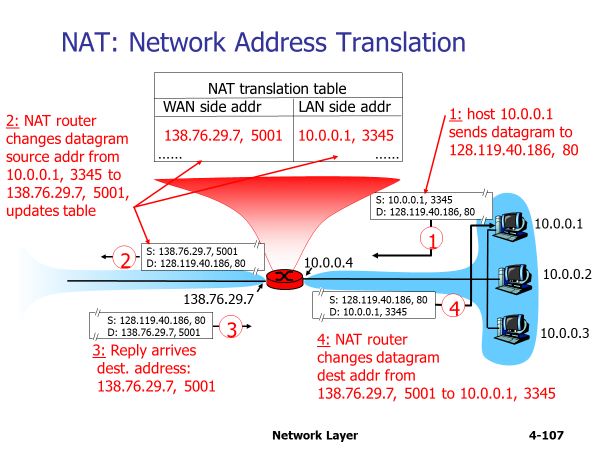
아래 그림 과정

1. 최초 도착시 사설망 IP에 해당하는 port num을 부여 (5001 port num)
2. Table에 저장 후 public 주소로 바꿔 전송
3. 응답 도착 시 NAT table에서 5001에 해당하는 사설망 IP확인 후 사설망 IP로 바꾼다.
4. 이후 응답 전송

\*) port forwarding: 미리 나에게 접속할 port 알려주고 공유기, NAT을 admin으로 바꿔준다.

\*) Hole punching: 사설망에서 NAT을 통해 구멍뚫고 이걸 랑데그 server에 등록한 뒤 일반 사용자가 접속시 랑데그 server에서 IP, Port 가져와 접속 후 NAT에서 뚫린 구멍으로 사설망에게 전송(연결)

\*) 랑데그 server: 내가 접속하고자 하는 server의 IP, port 가져온다.. (?)



**Unicast Routing Protocols**

Internet: 라우터에의해 연결된 네트웍들의 조합

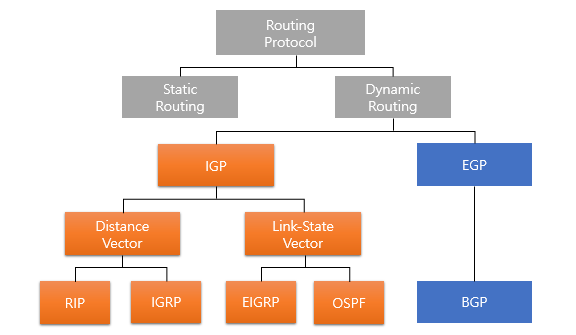
Routing: 패킷을 전송하기 위한 수많은 경로 중에서 한가지 경로를 결정하는 것

* + 라우팅에는 동적라우팅(변화하는 상황에 맞춰 라우터가 경로를 재설정하는 방식)과 정적라우팅(사람이 수동으로 미리 경로를 지정하는 방식)이 존재

Routing table: routing을 위해 네트워크 상의 모든 router들이 목적지에 따라 패킷을 보낼 interface를 선 계산해 놓은 table

Routing Protocol: 라우팅 테이블을 생성, 유지, 업데이트, 전달하는 프로토콜

1. Routing protocol 구성
   1. Routing table
      1. 패킷을 목적지로 라우팅할 때 참조하는 테이블
      2. 목적지 주소, output I/F, Metric 값
   2. Message
      1. 라우터 간 라우팅을 위해 교환하는 메시지
      2. 이웃 도달 메시지, 라우팅 정보
   3. Metric
      1. 라우팅 테이블 생성 및 업데이트 시 최적의 경로를 결정하는 기술
      2. 경로 길이, Hop 수, 대역폭, 비용, 신뢰성
2. 종류

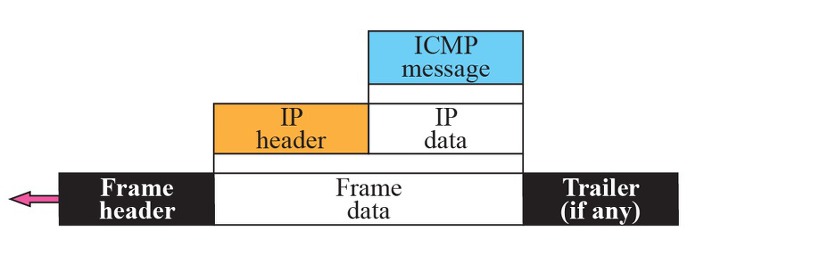


* 1. Static routing protocol: 수동식. 라우터 부하 경감. 고속 라우팅 가능/ 관리자의 관리부담 증가 및 정해진 경로 문제 발생 시 라우팅 불가능
  2. Dynamic routing protocol: 라우터 스스로 경로 동적 결정
  3. 내/외부 라우팅: 인터넷이 너무 커서 하나의 routing protocol이 처기하기 힘들기 때문에 인터넷을 AS(Autonomous system)들로 나눔 (AS는 여러 네트웍과 라우터들로 이루어짐)
  4. AS: 하나의 관리자에 의해 운용되는 router를 포함한 네트웍으로 router가 관리하는 범위의 네트웍
  5. IGP(interior gateway protocol, 책에서는 Intradomain): AS내에서의 라우팅을 담당하는 라우팅 프로토콜
     1. Distance vector: 라우팅 table에 목적지까지 가는데 필요한 거리(hop counter)와 방향만을 기록 (인접 라우터와 서로 table 공유하여 알지 못하는 경로를 채워나감) 🡺 RIP프로토콜은 벨만 포드 알고리즘이용
     2. Link-state vector: 라우터가 목적지까지 가는 경로를 SPE(Shortest Path First) 알고리즘을 통해 모든 라우팅 테이블에 기록해 두는 것 (모든 라우터) 🡺 OSPE 는 dijkstra알고리즘 이용
        1. 각 router가 모든 경로로 가는 cost가 기록된 topology를 갖고 있음
        2. 이 topology를 통해 cost와 next router가 표시된 routing table을 생성
        3. 각 router는 정보를 모두에게 보내고 각자의 전체의 topology를 만들고 topology를 가지고 최단경로를 구성
  6. EGP(Exterior gateway protocol, 책에서는 Interdomain): 서로 다른 AS사이에서 사용되는 라우팅 프로토콜
     1. BGP: 수학적 계산이 아닌 rule과 policy기반으로 routing table 관리

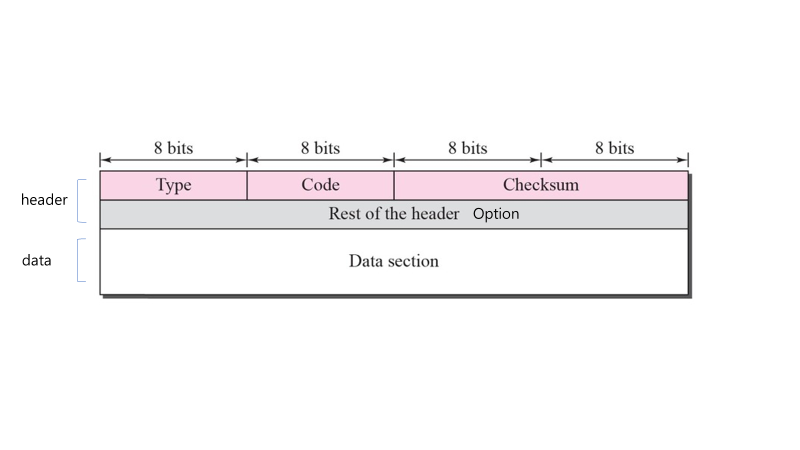
**ICMPv4 (Internet Control Message Protocol Version 4)**

Network Layer에 있는 Protocol 중 하나로 Error reporting, query message에 사용되는 프로토콜

* Router가 packet을 버리는 경우나 error에 의해 제대로 전송되지 않았을 때 sender에게 알려주기 위해 사용
* Encapsulation: 다른 프로토콜과 같은 형태로 IP data부분에 ICMP message가 들어가고 header가 붙는다.



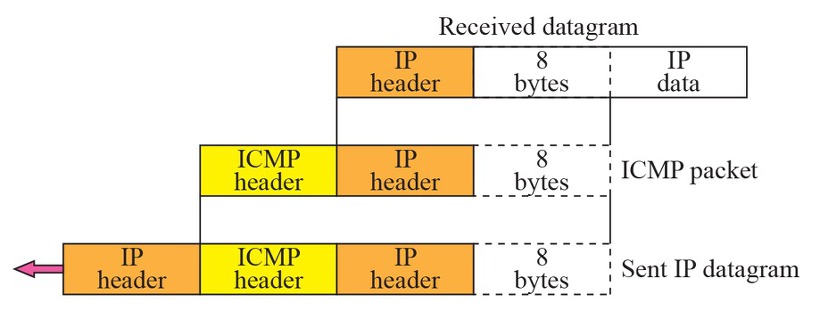
* ICMP message의 기본 format



Type: error type

Code: detail error정보, 무슨 상황인지에 대한 설명

* ICMP message 종류: error-reporting message, query message
  + Error-reporting: router나 destination에서 IP 패킷을 처리할 때 발생하는 문제 보고 🡺 router나 destination에 대해 자동으로 보고됨 (그림과 같이 packet 구성)

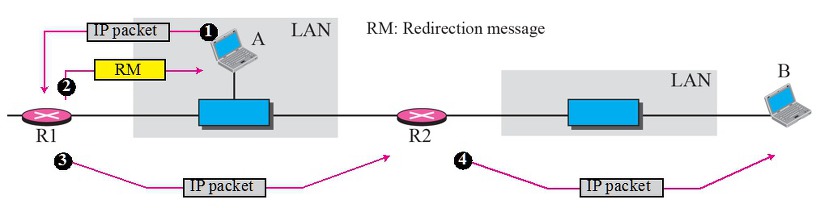


Error message의 data section: 받은 datagram의 일부 내용이 들어간다.

* Received datagram의 IP header: source와 destination을 토해 어떤 IP에 보내려던 것이 error가 났는지 알 수 있다.
* 8 bytes: TCP header의 맨 앞 8B으로 source port, destination port, seq num이 존재
  + 어떤 application(port)에 보내려던 것인지 알 수 있음
  + Seq num을 보고 몇 번째 packet이 전송 실패했는지 알 수 있음

Sent IP datagram에서 IP header: 메시지를 전달하기 위해 source, dest IP주소가 들어가며 이를 통해 오류정보를 받는 쪽은 error가 발생한 위치를 알 수 있다 (source address)

1. Destination unreachable: 여러 원인에 의해 에러 발생시 reporting해줌
   1. Link가 깨져서 (물리적) 더 이상 전송할 수 없을 때 ICMP 전송
   2. Destination의 port가 열리지 않거나 없을 때 ICMP전송
   3. Do not fragment이 1로 설정된 경우 MTU에 의해 더 이상 분해할 수 없으므로 버리고 ICMP 전송
   4. source routing옵션 사용할 경우 기재된 router를 갈 수 없을 때 packet을 버리고 ICMP 전송
   5. 이러한 종류를 icmp header에 code num을 통해 알 수 있다.
2. Source quench: 네트워크가 혼잡인 경우 packet을 버리고 ICMP 전송
   1. 혼잡발생을 경고하는 용도로 실제 혼잡제어는 TCP만 가능
3. Time exceeded
   1. Router가 TTL을 감소시킨 후 0이 됐을 때 packet을 버리고 ICMP전송
   2. destination에서 모든 fragment가 도착하지 않은 채로 time-out됐을 때 모든 fragment를 버리고 ICMP 전송
4. Parameter problems: router나 destination에서 IP헤더에 문제가 있을 때 packet을 버리고 ICMP 전송 (어느 부분이 이상한지 pointer를 같이 보냄)
5. Redirection: A🡪B로 전송할 때 A🡪R2가 아닌 A🡪R1🡪R2로 packet이 전송될 경우 ICMP를 이용해 redirection이 가능함
   1. R1으로 packet이 온 경우 R1 router가 source에게 R2로 가라는 경로를 알려준다. 이를 통해 A는 routing table을 수정
   2. A의 default router가 R1밖에 없어서 발생하는 문제로 A, R1, R2가 같은 네트웍이라는 것을 알려주는 것이다. (같은 local network에서의 잘못된 경로만 감지 가능)



* + Query message: 네트웍 매니저나 host가 router나 다른 host에대한 특정 정보를 얻는 것을 도움 🡺 관리자가 물어봐야 한다. Ex. Ping
    1. Echo-request message: host나 router가 보내는 것으로 상대방 network layer 작동여부를 확인할 때 사용한다.
       - Ping이라는 프로그램이 이 원리를 사용(echo request, echo reply)
    2. Timestamp message: network layer에서 RTT를 측정할 때 사용한다.
       - ICMP message의 data부분에는 original(packet 출발시점), receive(도착 시점), transmit(걸린 시각) timestamp가 저장됨
       - Receive-original을 통해 network에서 걸린 시간을 측정 가능
       - Timestamp message를 통해 두개의 다른 기기의 클락 동기화 가능

ICMP debugging tool

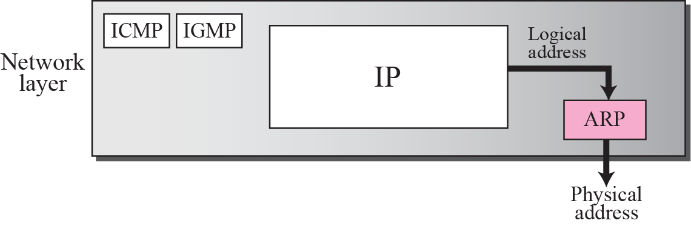
1. Ping: timestamp를 이용해 RTT측정할 수 있게 해주는 툴
2. Traceroute: TTL을 하나씩 늘려가며 해당 경로에 있는 router들의 주소를 알 수 있음

Ex) TTL=1🡪첫번쨰 hop에서 ICMP전송, TTL=2🡪두번째 hop에서 ICMP전송 …

**ARP(Address Resolution Protocol)**

Mac 주소 알려주는 protocol

IP프로토콜로부터 IP주소를 알려주면 상응하는 mac주소를 알려주고 data link layer로 이동



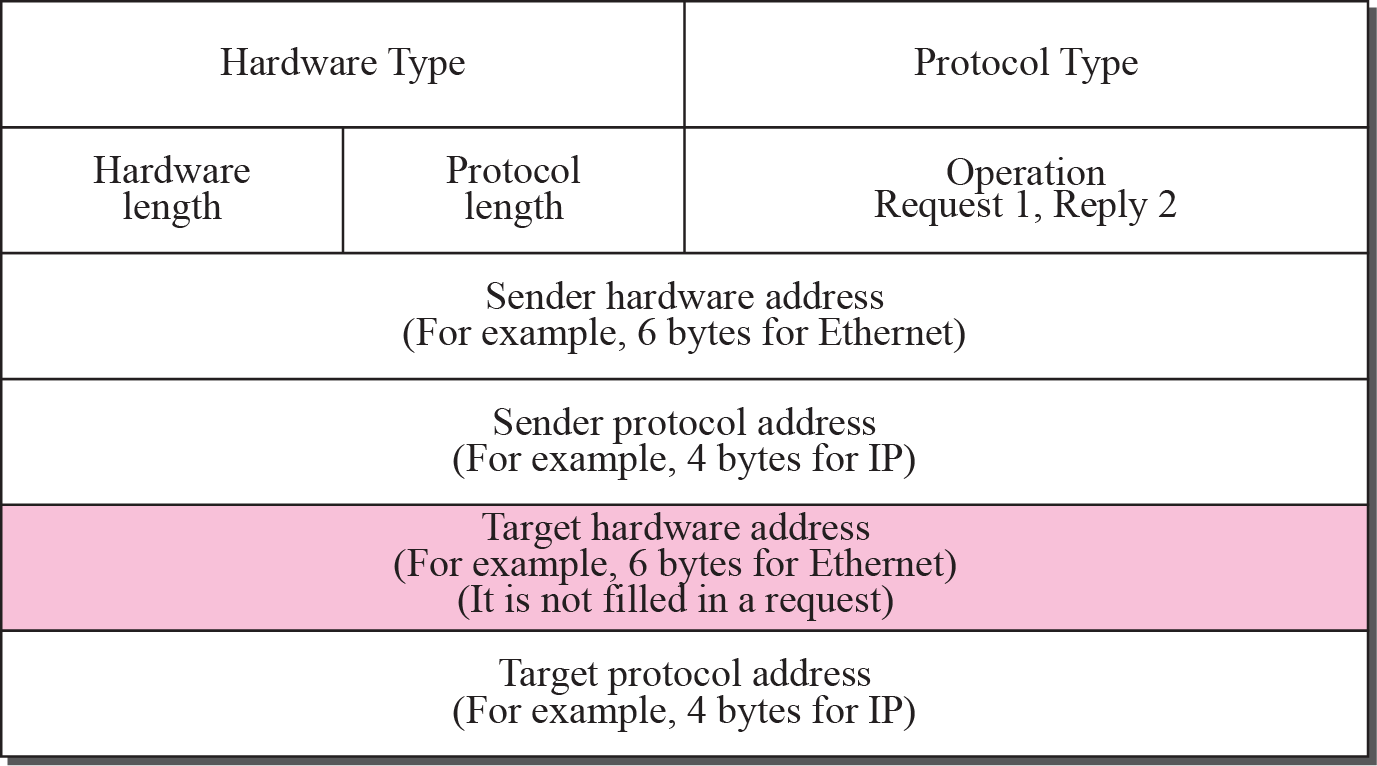
Arp 필요 이유: Host or router는 receiver의 logical(IP)주소를 알고 있다. 하지만 physical network을 통해 전송되기 위해 IP datagram은 캡슐화되어야 한다. 즉, sender는 receiver의 physical 주소가 필요하다.

* Inter networking환경에서 MAC주소와 IP주소를 알아야 스위치, 라우터 조합의 네트워크 망에서 통신이 가능하다.

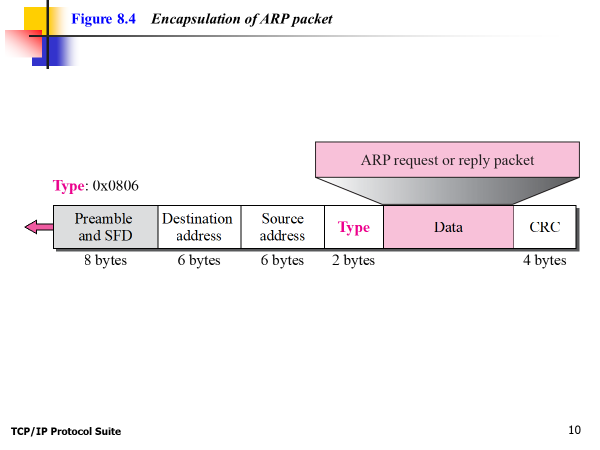
동작 원리:

1. Request: 송신자는 목적지의 물리적 주소가 필요하므로 물리주소 요청을 위한 ARP요청 패킷을 **브로드캐스트**로 전송 (이 IP주소를 가지고 있는 device는 mac주소를 알려주세요!)
2. Reply: 모든 호스트와 라우터가 송신자가 보낸 ARP 요청 패킷을 수신한 후 해당되는 수신자만 자신의 논리주소와 물리주소를 넣어 응답 패킷을 **유니캐스트**로 전송

ARP packet



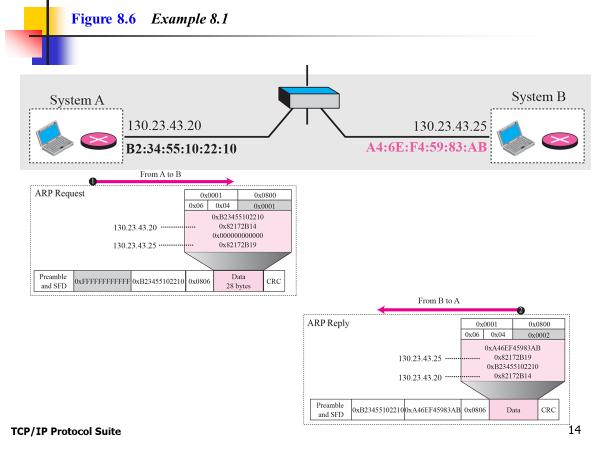
Encapsulation of ARP packet



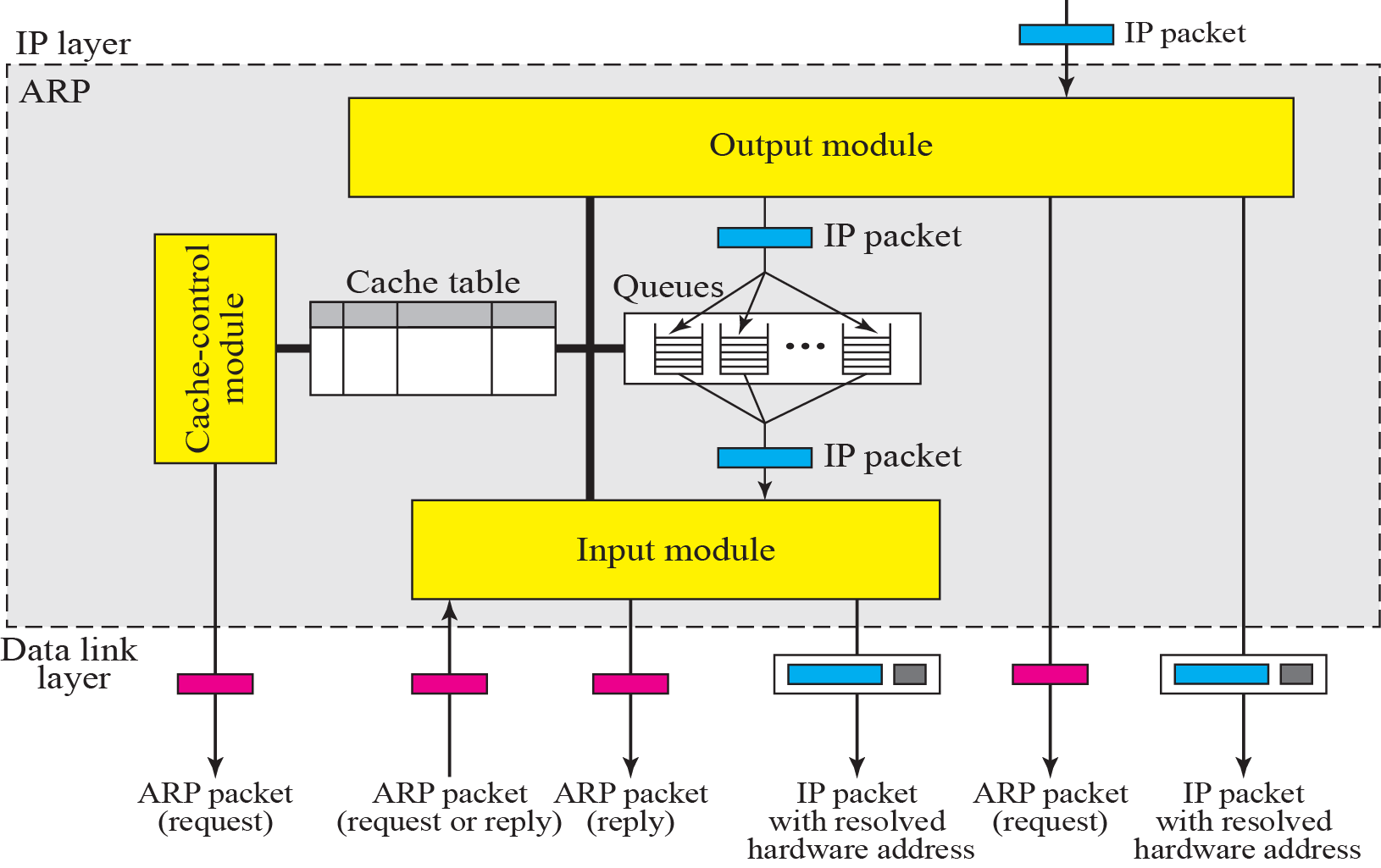
동작과정:

Request - broadcast이므로 destination address의 모든 bit를 1로 설정하여 전송, 이때 ARP packet에서 Target hardware address부분은 궁금한 내용이므로 모든 bit 0으로 보낸다.

Reply – unicast이므로 destination과 source잘 적어서 전송, ARP packet의 주소내용 모두 채워서 전송한다. (sender hardware address에 자신의 mac주소 적어서 보내기)



ARP components



Cache table: ARP의 효율적 수행을 위해 다른 host의 mac주소와 protocol주소를 저장한다. 일반적으로 request, reply둘다 상대방의 정보를 table에 저장한다. 항상 최신상태를 유지함으로 time-out넘어가면 정보 소멸되며 테이블에 정보가 없는 경우 request를 보낸다.