네트워크 정리

**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 🡺 **표준화**
* 실제 컴퓨팅 장치나 네트워크 장치를 만들 때 **OSI모델을 참조하여 모든 통신장치를 만든다**.

**OSI 7계층**

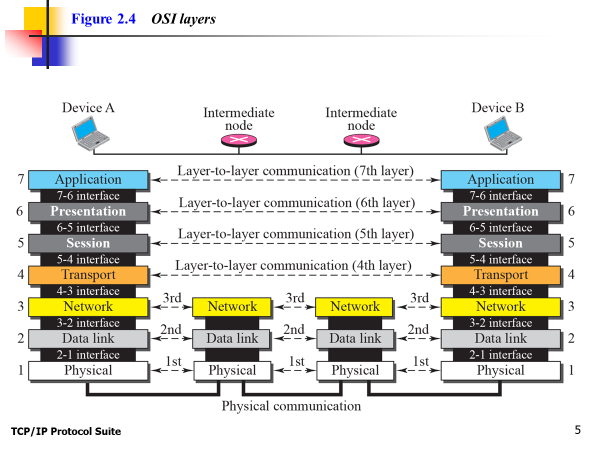
* 네트워크에서 **통신이 일어나는 과정**을 7단계로 나눈 것
* **계층마다 packet이름이 다르다** 🡺 오류발생 시 오류발생한 단계를 쉽게 알 수 있다.
* **계층을 나눈 이유**
  1. 통신이 일어나는 과정을 단계적으로 파악할 수 있기 때문
  2. 특정 단계에 이상이 생기면 다른 단계의 장비 및 소프트웨어를 건들이지 않고 이상이 생긴 단계만 고칠 수 있다.
* 다른 Layer의 data를 유지한 상태로 해당 Layer에서 할 이들과 여러 정보들을 담는 header를 만들어 data뒤에 붙여 다음 Layer로 보낸다.

1. **Encapsulation**: 데이터 전송 시 7🡪1단계로 data뒤에 헤더 20bytes씩 추가되어 이동
2. **Decapsulation**: 데이터 받을 때 1🡪7단계로 header를 제거하여 여러 검사를 하고 다음 layer로 이동

* OSI 7 layer model (1~3계층은 하드웨어로, 4~7계층은 소프트웨어로 구현)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Layer | 전송 단위 | 기능 |
| **1계층** | Physical Layer  (물리 계층) | Bits | **데이터를 전기적 신호로 변환**해서 통신 케이블로 주고받음 (0과 1)  알고리즘, 오류제어 기능이 없다. (이 계층만 없음)  장비: Hub(허브), Repeater(리피터), Cable |
| **2계층** | Data Link Layer  (데이터 링크 계층) | Frames | **물리적인 매체에 데이터를 실어 전송**  **Hop-to-Hop delivery:** 목적지까지 자동으로 데이터가 전달되는게 아니다. **MAC주소를 이용하여 정확한 장치로 정보를 전달하여 목적지까지 전달**  인접 노드간 신뢰성 있는 데이터 전송을 제어: 흐름제어, 오류제어, 회선제어  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: **routing (목적지로 가기 위한 경로 선택)**  장비: Router(라우터, 네트웍과 네트웍을 연결) |
| **4계층** | Transport Layer  (전송 계층) | Segment | **Process to Process:** 전체메시지를 발신지 대 목적지 간 제어와 에러를 관리  **종단간 신뢰성 있는 통신을 보장한다.**  **분할과 재조합, 연결제어, 흐름제어, 오류제어, 혼잡제어 수행**  **port number(app별 고유 번호)를 부여한다**. (어느 application으로 보낼 것인지 지정)  프로토콜: TCP, UDP |
| **5계층** | Session Layer  (세션 계층) | Data | 응용 프로그램 간의 연결을 지원  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  프로토콜: 텔넷, HTTP, SSH, SMTP, FTP |

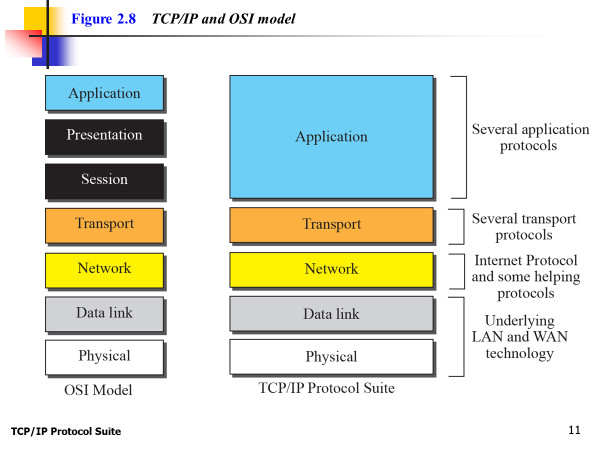
* 데이터 통신: 각 계층별 의사소통한다.



* Ethernet: 네트워크를 구성하는 방식 중 한 방법으로 각 기기들이 MAC주소를 가지고 호스트 간의 데이터를 주고받을 수 있는 방식이다.
* Ethernet implementation (네트워크에서 physical layer와 data link layer)
  1. 같은 네트웍에 있는 컴퓨터들을 연결하기 위해 **멀티탭** 역할을 하는 장비들을 사용
  2. **Filtering**: 단순 컴퓨터들을 연결하는 역할 경우 각각의 컴퓨터들의 위치를 알 수 없기 때문에 데이터를 모든 컴퓨터에게 보내게 된다. 하지만 원하는 컴퓨터의 위치를 알고 있다면 모두에게 보내지 않고 해당 link로만 보낼 수 있다.
  3. Physical layer 장비: repeater, hub 🡨 filtering 불가, 멀티탭 역할만
  4. Data link layer 장비: bridge, **switch** 🡨 filtering 가능(위치 모르면 모두에게, 알면 해당 link)

**TCP/IP 프로토콜**

* OSI 7계층보다 먼저 개발된 네트워크 모델, **5계층**으로 구성 (5,6,7단계가 하나의 단계) 🡪 1,2계층을 합치면 4계층
* 실질적인 통신은 TCP/IP프로토콜 모델을 기반으로 구현됨



* 통신을 위해 각 단계에서 필요한 주소

1. Application layer: Application-Specific address
2. Transport layer: Port address (application나타내는 주소)
3. **Network layer: Logical address (IP주소)**
4. **Data link layer: Physical address (MAC 주소)**

**네트워크 전달방식 비교**

* 네트워크 상에서 호스트와 호스트 간에 정보(data)를 주고받는 방식에는 2종류가 있다.
* 물리적 측면에서 연결기반인 circuit switching과 비연결기반인 packet switching

|  |  |
| --- | --- |
| 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 (Transport layer)**

* 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 |

**Transport Layer – TCP**

TCP/IP protocol suite에서 Transport Layer에 존재하는 프로토콜: TCP, UDP

* **서버와 클라이언트간 데이터를 신뢰성 있게 전달**하기 위해 만들어진 프로토콜
* **연결지향 프로토콜**: 데이터 전송 전, 데이터 전송을 위한 연결을 만든다.
* 데이터 전달되는 과정에서 손실되거나 순서가 뒤바뀌어 전달 가능하며, **TCP는 손실을 검색해서 이를 교정하고 순서를 재조합** 할 수 있도록 해준다. 🡺 application은 완성품 받으면 끝

**특징**

* **Stream delivery**: application이 만든 packet의 boundary를 유지하지 않고 나눠 보낸다.
  1. Application이 TCP에 전달한 메시지들은 크기가 일정하지 않으므로 TCP에서 정한 일정한 크기(segment)에 맞게 메시지를 나눠서 전송한다.
  2. 따라서 한 메시지의 시작위치와 끝위치를 알지 못하고 상관하지 않는다.

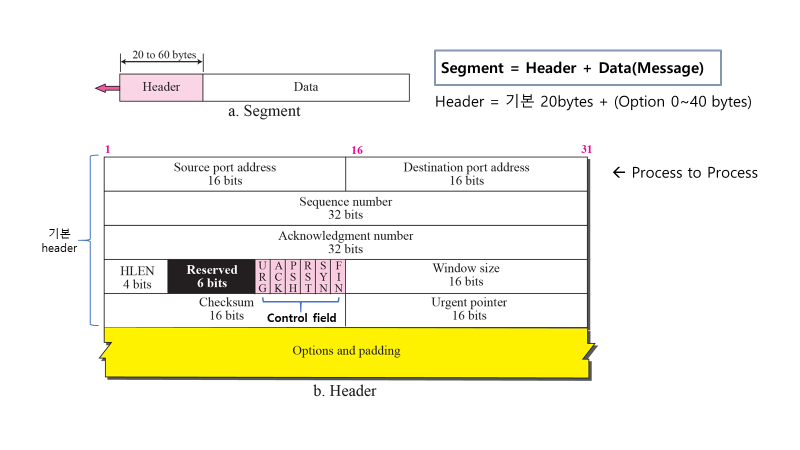
(↔ Boundary delivery: UDP에서 사용하는 방식으로 application이 만든 packet을 쪼개지 않고 보존하여 전송한다.)

* TCP는 **양방향**으로 서로 주고받기가 가능
  1. TCP에는 sending buffer와 receiving buffer가 존재한다.
  2. Buffer를 공용으로 사용하지 않기 때문에 서버는 client당 buffer가 하나씩 존재한다.
  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 segment 전송할 때 도착지에서 재조립하기 위해 sequence number를 붙여서 보낸다.
  1. 패킷들은 서로 다른 경로를 거치면서 전달되므로 순서가 뒤죽박죽을 도착한다.
  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. Cumulative: 한 segment를 전달받은 후 다음에 받기를 기대하는 segment의 시작 byte번호를 ack number로 상대에게 전달한다.

(↔ Selective: 받은 것에 대한 정보 Ack으로 보내기)

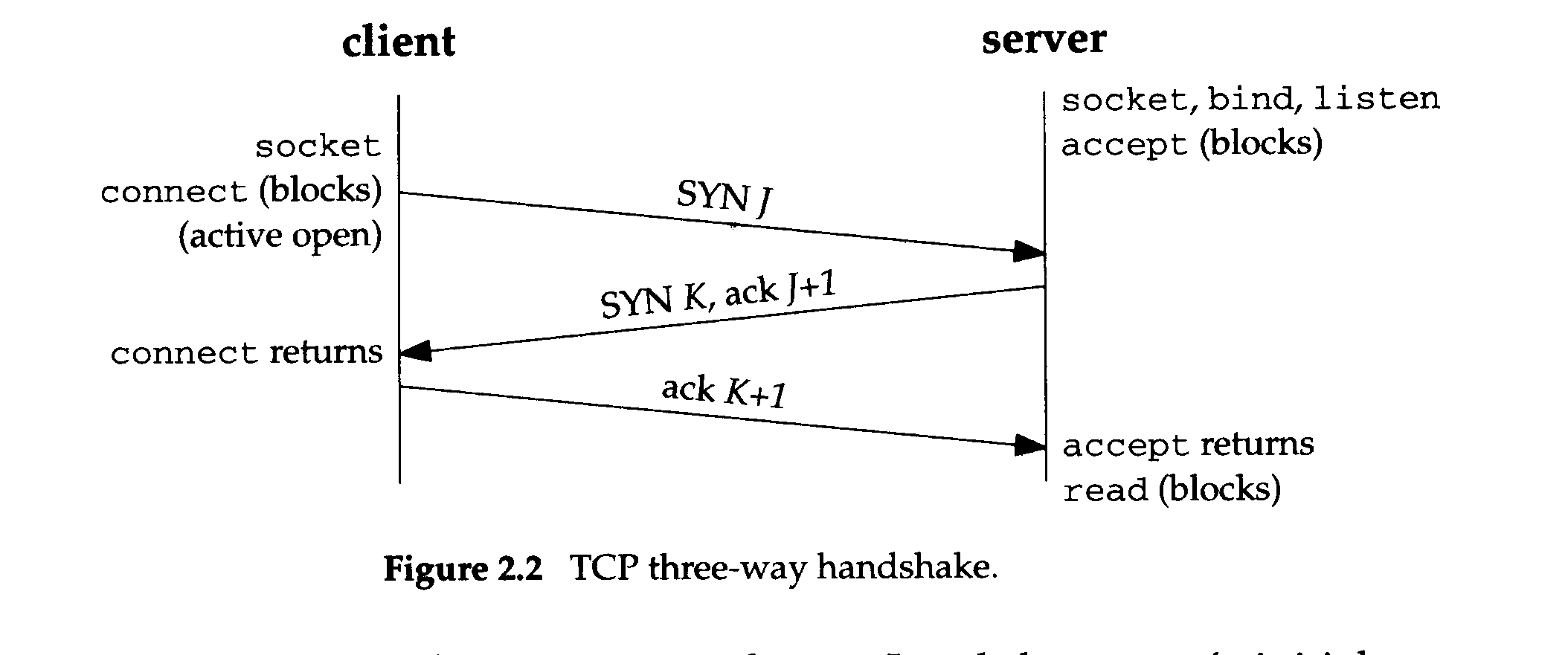
* TCP전송 제어 방법: **흐름제어**(전송 데이터양 조절), **오류제어**(통신 도중 데이터 유실 or 잘못된 데이터 수신 시 대처 방법), **혼잡제어**(네트워크 혼잡 대처) (UDP는 흐름, 혼잡제어 안함)

**TCP Segment Format**



1. HLEN: header의 길이저장. 기본20bytes에 option있을 경우 header길이, 경계 모르므로 데이터 시작부분을 알기 위해 저장. 전체길이/4의 몫을 저장한다.
2. **Control field**: 각 1bit씩 존재하며 flag처럼 사용한다. **packet받으면 여기 먼저 확인한다.**
   1. ACK: Acknowledgment is valid (데이터 전달받았다는 것)
   2. SYN: 연결 요청 packet
   3. FIN: 연결 종료 요청 packet
3. Window size: TCP 세그먼트를 보내는 호스트의 현재 TCP 버퍼 크기
4. TCP Checksum: TCP **데이터와 TCP 헤더의 error** 유무확인(다른 계층 checksum은 헤더에러만)
5. 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.

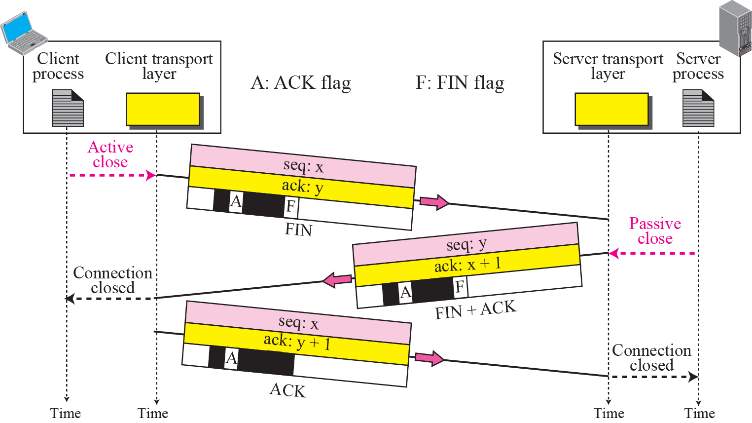
**TCP connection**

1. 연결 set-up 과정 (3way handshake): 초기 연결 과정으로 buffer를 생성하고 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

1. 연결 종료 과정 (3way handshake): 서버로부터 서비스 다 받고 client가 먼저 종료 요청한다. 서로 FIN전달하여 해당 buffer를 지운다.

① C🡪S: **FIN** | 연결 종료요청

② S🡪C: **FIN + ACK** |연결 종료요청 + 응답

③ C🡪S: **ACK** | 2번에 대한 응답

☆) ACK와 FIN을 따로 보내면 4way handshake도 가능

☆) 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. 연결 종료는 FIN 🡪 FIN+ACK 🡪 ACK순으로 이루어지며 마지막에 서버가 ACK을 받지 못해 FIN을 재전송할 경우를 생각해서 TIME\_WAIT상태를 일정시간 유지
2. Server와 client간 연결 종료된 후 다시 연결할 때 이전과 같은 port num을 쓰게 될 경우 이전 연결에서 늦게 전송된 데이터와 현재 데이터를 구분하기 어려운 문제를 해결하기 위해 TIME\_WAIT상태를 일정시간 유지하여 재연결시 같은 port num쓰는 것을 막는다.

(서버는 client buffer 찾을 때 {IP주소, port num}쌍으로 이루어진 client ID를 이용)

1. Half-Close (한쪽만 FIN을 보내어 반만 종료된 상태)

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

Client가 더 이상 보낼 정보가 없을 때 FIN을 보내게 되면 client는 보내기 불가능, 받기 가능상태가 되며 마찬가지로 server는 받기 불가능 보내기 가능상태가 된다.

**Flow control (흐름 제어)**

* **데이터를 수신하는 노드가** 전송하는 노드에게 **현재 자신의 상태에 대한 정보 보내주는 것**
* 이유: **송신자의 데이터 전송 속도가 수신자의 데이터 처리 속도보다 빠르면** 데이터 양이 수신 버퍼 크기를 넘게 되며 **데이터 손실이 발생** 🡺 흐름제어를 해야 한다!

1. Stop & wait 방식: 신뢰성을 보장하기 위한 데이터 전송의 흔한 방식으로 데이터를 보낸 후 상대방에게서 ACK이 올 때까지 기다렸다가 도착하면 그 때 데이터를 보낸다. 비효율적
2. **Sliding-Window 방식**: stop & wait개선한 것으로 ACK을 기다릴 필요 없이 각 패킷에 번호를 붙여서 연속적으로 보낸다. (단, **window size만큼만 보내기 가능, 보내는 양을 조절한다**.)
   * Window size(N): 패킷을 보내고 그 패킷에 대한 ACK을 받기전까지 보낼 수 있는 최대 양으로 ACK에는 수신자의 rwnd(Receive Window Size)가 같이 보내지며 N이 조정된다.
   * Window size = min(rwnd, cwnd) (cwnd는 네트워크 상황을 고려한 혼잡제어 크기)
   * 수신자 buffer의 빈 공간 생기는 속도는 application이 가져가는 속도로 속도가 빨라지면 N또한 커진다. 즉, 처리 속도에 따라 N은 가변적

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

* N: 5000B, 1packet당 1000B이므로 5개의 패킷으로 나눠 5000B를 보내기. 그리고 ACK받을 때까지 기다림. 이 이상은 전송하지 않는다.
* Silly Window Syndrome: 보내는 양이 작아서 header size overhead가 커지는 상황

1. 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
2. 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

**Error Control (에러제어 – packet 없어지는 경우)**

* 에러제어는 checksum, acknowledgement(ACK), 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 data온다면 같이 보내기

Rule3: 받은 packet 2개당 ACK한개 보내기, 500ms기다리다 2개 들어오면 바로 전송

* 1. 응급 상황 시 ACK 바로 보내기

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

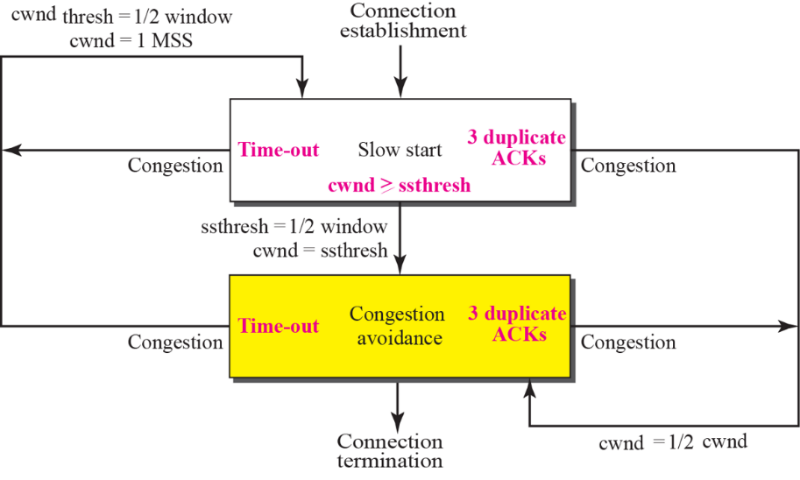
Rule4: packet이 없어진 것을 감지하면 500ms 기다리지 않고 바로 ACK보내기

Rule5: 잃어버린 packet이 오면 500ms 기다리지 않고 바로 ACK보내기

Rule6: 중복된 packet 받으면 바로 ACK 보내기

* 3 duplicated ACK: 중복된 ACK 세번 받으면 packet-loss를 감지하여 RTO time-out까지 기다리지 않고 다시 packet보낸다.
* 보통 ACK손실은 큰 문제를 발생시키지 않지만 ACK손실이 Deadlock 발생시키는 경우가 있다.
  + Packet 전송을 중단한 상태에서 rwnd=k (k!=0) 패킷이 손실된다면 클라이언트와 서버 둘 다 보낼(받을) 수 있는지 계속 기다리게 된다. 🡺 데드락 발생
  + 해결: 송신 측에서 수신 측으로 probe(1byte, 원래 500bytes)를 주기적으로 보내 ACK과 함께 rwnd 정보를 계속 받아 packet을 보낼 수 있는지를 확인

**Congestion control (혼잡 제어)**

* 네트워크 상황을 고려하여 혼잡이 발생하지 않도록 한다. (공용선을 사용하므로 혼잡 발생)
* 병목현상(Bottlenect): input보다 output이 더 작은 경우 버퍼 크기를 넘게 되면 packet을 버린다. 🡺 packet 손실의 대부분 이유 (혼잡에 의해 router가 버림), 전체 시스템에 영향을 끼친다.
* **혼잡발생 🡪 packet 손실 🡪 delay 증가 🡪 throughput(시간당 받는 양) 감소**
* cwnd: 혼잡 윈도우 크기, window size를 상대방만 고려(rwnd)하여 조정할 경우 size가 너무 커지면 네트워크가 죽을 수 있다. 이에 네트워크 상황을 고려한 cwnd도 함께 이용해 크기 설정
* 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 증가
  + AI(addictive increase): Threshold이후 1씩 값 증가
  + MD(multiplicative decrease): 혼잡 발생시 절반으로 값 줄이기
  + 값 증가는 RTT(왕복시간)마다 일어난다.

TCP Timers

Retransmission timeout(RTO): 재전송을 위한 타이머, RTT(round trip time)을 이용해 계산

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

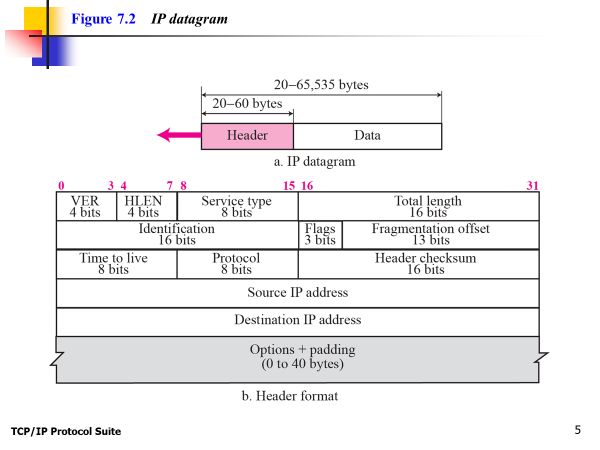
Keepalive Timer: 오랜 기간 동안 idle상태에 있는 것 방지, 서버가 2시간 동안 클라이언트로부터 세그먼트를 전송받지 못하면 probe 세그먼트 전송

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

**Network Layer – IP, URP, ICMP, ARP**

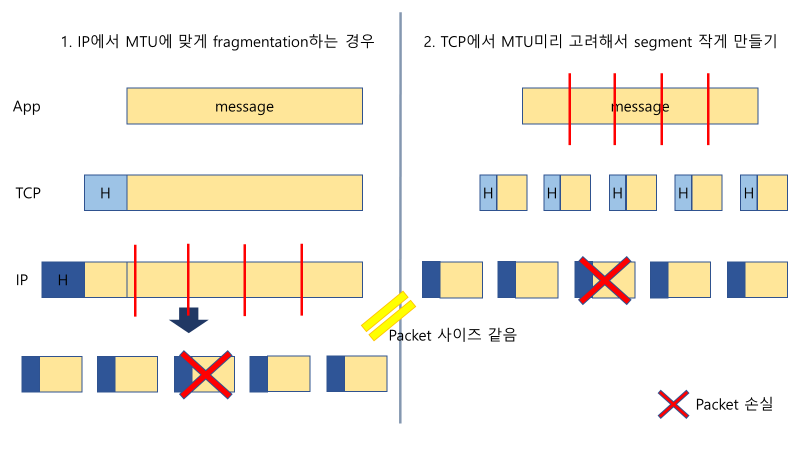
**IP**

* Header: routing과 delivery에 필요한 정보를 포함한다. (source to destination)
* 버전은 IPv4와 IPv6존재 (주소 길이가 다르다)
* Packet의 명칭은 datagram



* Datagram의 encapsulation: **데이터그램의 min길이=46bytes, max=MTU** (Maximum Transfer Unit, 데이터그램의 최대 크기)이다. min보다 작으면 46bytes를 만족하도록 padding추가 후 캡슐화, max보다 크면 단편화 후 캡슐화
* Fragmentation (단편화): 데이터그램의 크기가 크면 MTU 크기에 맞게 데이터그램의 데이터를 나누는 것 (같은 데이터그램임을 나타내는 의미로 ID가 같다)
  + 네트워크별로 MTU가 다르므로 해당 네트워크의 MTU에 맞게 사이즈 크면 단편화.
  + **단편 재조립: 최종 목적지에서** 단편들로부터 원래 데이터그램을 생성한다. 이때, header에 적힌 offset을 보고 순서를 맞춘다.
  + 재조립 time-out: IP계층은 데이터그램의 어떤 단편이 처음 도착하면 타이머를 시작하고 일반적인 타임아웃은 30~60초, 타임아웃 시간이 지나면 같은 ID를 가진 단편들 폐기
  + **단편화 단점: 단편 손실 시 데이터그램 전체를 재전송 해야한다.**

-> TCP에서 MTU고려해서 미리 MSS로 segment 나눌 경우 단점을 좀 더 예방할 수 있다.

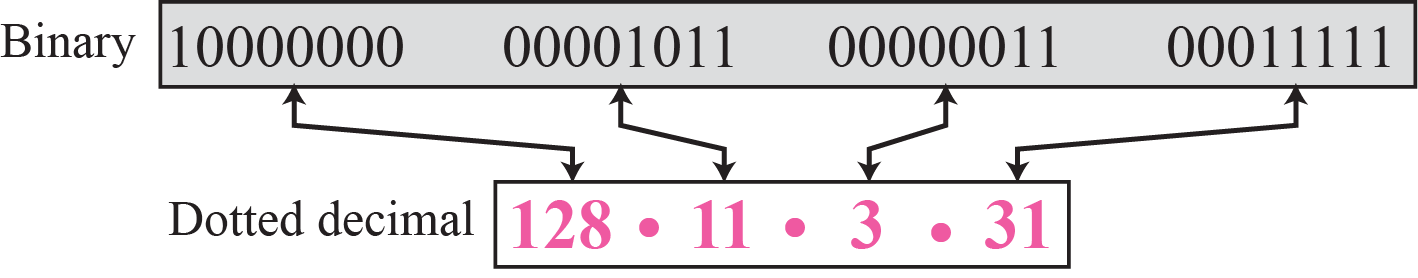


1번 경우 재조립을 위해 타임아웃 시간동안 기다렸다가 전체 폐기 후 3DA에 의해 packet loss감지 후 상대가 재전송하는 반면, 2번 경우는 재조립 기다리지 않고 3DA에 의해 재전송 받는다. 즉, TCP가 미리 segment작게 만들면 packet loss감지 좀 더 빠르다.

* Options
  1. Record route: 패킷의 전달 경로를 기록 (전달될 때마다 차례로 기록됨)
  2. Source route: 전달 경로를 packet에 미리 지정하고 그 경로를 따라 패킷을 전송하게 함
     1. Strict source route: 지정된 경로로만 이동 가능하고 경로 이상 시 ICMP메시지 전송/주로 network 관리자가 길을 만들고 그 경로로 잘 이동되는지 검사용으로 쓰인다.
     2. Loose source route: 지정된 경로를 거쳐서 가면 된다. 즉, 다른 경로 사용 가능
  3. Timestamp: 패킷 경로상의 라우터에 도착하는 시간을 기록
* Delivery and Forwarding: 네트워크 레이어는 하위 레이어에서 패킷이 움직이는 것을 다루는 역할을 하며 최종 목적지로 가기 위한 다음 hop을 routing table을 통해 찾아 패킷을 전달함

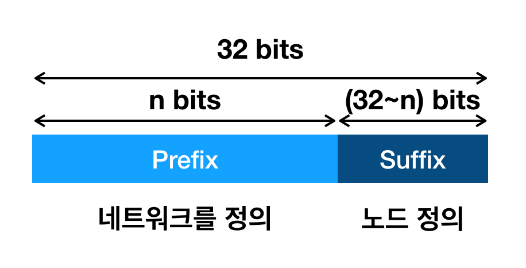
**IPv4 Address**

* IP계층에서 사용되는 Internet에 연결된 각 장치를 식별하기위한 식별자
* 32bit 주소이며 전세계 하나씩 밖에 없다. 🡨 현 네트워크 IP주소 체계는 이거 사용
* 표기: 실제는 32bit 숫자 1개 🡪 읽기 힘들어서 1byte씩 총 4byte로 끊어 표기

각 byte는 0~255범위 값

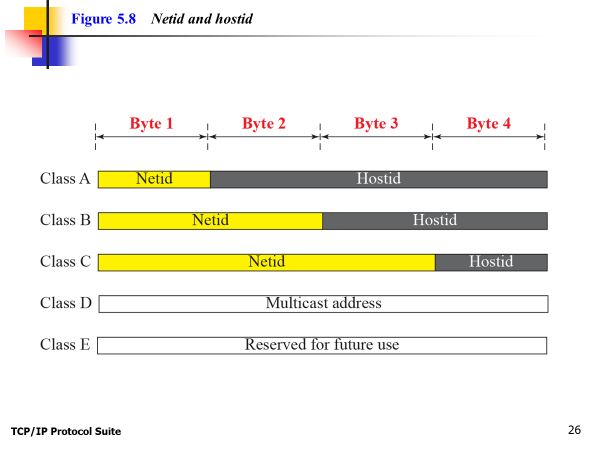
* 주소 표현 방식: 계층적 주소

주소 첫 부분은 접두사(Prefix)로 네트워크를 정의하고, 두번째 부분은 접미사(suffix)로 노드(장치의 인터넷으로 연결)를 정의한다. (prefix = netID, suffix = HosID)

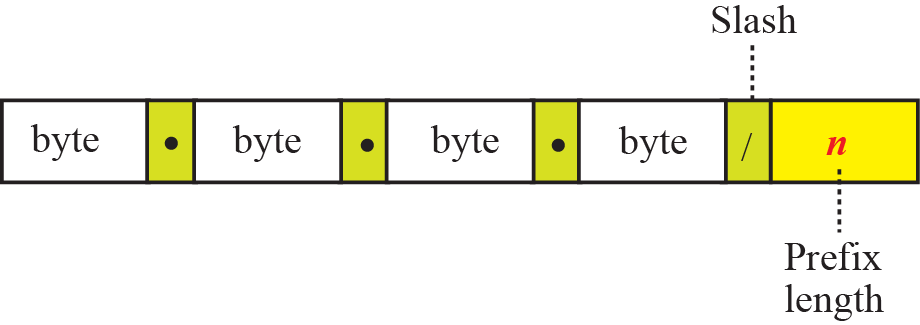


**접두사는 클래스 기반 주소 지정과 클래스 없는 주소 지정으로 나눠진다**.

1. **Classful addressing**: IP주소를 규격화된 크기별(클래스별)로 구분하는 방식
   * 총 5개의 class(A, B, C, D, E) 존재하고 구축할 네트웍의 크기에 따라 A, B, C로 나뉜다.
   * class별 구성비율이 다르다. Host ID의 비율이 클수록 구축가능한 네트웍의 크기가 크다.



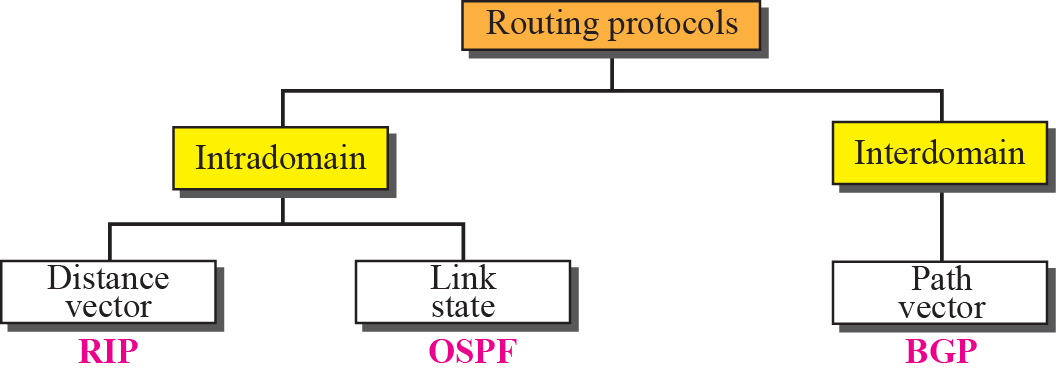
* 장점: 접두사가 고정이라 쉽게 주소 클래스를 찾을 수 있고 접두사와 접미사를 추출하기 위해 추가적인 정보가 필요 없다.
* 단점: 주소가 금방 고갈된다. (ex. 클래스 A경우 128개의 기관만 할당 가능) 또한, 네트웍의 크기가 크면 주소가 낭비된다. (이제 클래스 기반 사용하지 않는다..?)

1. **Classless addressing**: Class라는 규격화된 구분 없이 비트 단위별로 IP주소 범위를 가변적 구분하는 방식
   * Class는 byte단위, classless는 bit단위로 끊어 가져간다. 주소 옆에 slash를 이용해 prefix의 길이를 적는다.
   * Prefix길이가 작을수록 큰 네트웍을 의미

* 각 네트워크의 시작주소와 마지막 주소는 할당하지 않는다.
  + 시작주소: 네트워크 주소로 사용되며 각 네트워크를 구분
* Subnet: 하나의 network로 관리할 경우 네트웍이 너무 커서 관리가 비효율적, 보안에 취약하기 때문에 subnet으로 나눠 관리
* **Loopback address**: 한 컴퓨터에서 client와 server를 설정하고 네트워크 프로그래밍이 잘 동작하는지 보기 위한 주소. (127.#.#.# 형식)
* 사설 주소: 내부망에서 사용하는 주소로 정해져있음
* 전송 방식
  + Unicast – 1:1 communication (server : client)
  + Broadcast – 1:all (목적지 주소를 255.255.255.255로 설정하여 모두에게 전송)
    - Limited broadcast: 다른 네트웍으로 안넘어가도록 router가 block처리
    - Directed broadcast: 한 block에서만 쓸 수 있도록 설정한 것으로 block의 마지막 주소를 broadcast용으로 사용하며 router가 많이 보낸다.
  + Multicast – 1:N (특정 그룹한테 전송)
* **NAT(**네트워크 주소 변환)
* 사설 주소와 범용 주소의 연결을 제공하고 동시에 가상 사설 네트워크를 지원하는 기술
* 내부 통신을 위해 사설 주소를 사용하고 다른 네트워크와 통신에는 범용 인터넷 주소를 사용할 수 있도록 한다 **🡺 사설 주소를 범용주소로 바꿔주는 역할**
* 변환 테이블: NAT라우터가 주소를 적절하게 변환하게 해주는 테이블. [사설주소, 외부주소]
* 통신 순서
  1. 내부에서 먼저 데이터를 보내면 테이블에 정보가 추가된다.
  2. 사설 주소를 바꿔준다.
  3. 외부에서 데이터가 들어올 때 테이블을 참조
  4. 목적지 주소를 변환

**Unicast Routing Protocols**

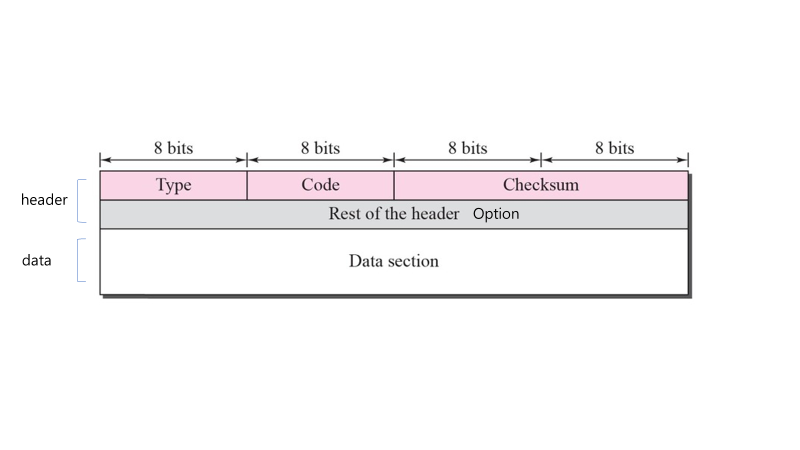
* **Routing table을 생성, 유지, 업데이트, 전달하는 프로토콜**
* Routing: 패킷을 전송하기 위한 수만은 경로 중에서 한가지 경로를 결정하는 것
* Routing table: routing을 위해 네트워크 상의 모든 router들이 목적지에 따라 패킷을 보낼 interface를 선 계산해 놓은 table
* 종류



* AS: 하나의 관리자에 의해 운용되는 router를 포함한 네트웍으로 router가 관리하는 범위의 네트웍 / 인터넷이 너무 커서 관리가 힘들기 때문에 인터넷을 AS들로 나눈다.
* Intradomain: AS내에서의 라우팅을 담당하는 라우팅 프로토콜
  + Distance vector: 라우팅 table에 목적지까지 가는데 필요한 거리(hop counter)와 방향만을 기록한다. 인접 라우터와 서로 table 공유하여 알지 못하는 경로를 채워 나간다.
    - * RIP 프로토콜은 벨만-포드 알고리즘 이용
  + Link state: 라우터가 목적지까지 가는 경로를 SPE(Shortest Path First)알고리즘을 통해 모든 라우팅 테이블에 기록해 두는 것. 모든 라우터가 모든 경로로 가는 cost가 기록된 topology를 갖고 있고 이를 기반으로 테이블 생성
    - * OSPE 프로토콜은 Dijkstra 알고리즘 이용
* Interdomain: 서로 다른 AS사이에서 사용되는 라우팅 프로토콜
  + Path vector: 수학적 계산이 아닌 rule과 policy기반으로 routing table 관리

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

* Error reporting, query message에 사용되는 프로토콜
* Router가 packet을 버리는 경우나 error에 의해 제대로 전송되지 않았을 때 sender에게 알려주기 위해 사용된다.
* ICMP message의 기본 format



1. Header: 에러 유형과 상황에 대한 정보가 포함됨
2. Data: message 종류에 따라 들어가는 정보가 다르다.

* Message 종류: error-reporting message, query message

1. Error-reporting: router나 destination에서 IP 패킷을 처리할 때 발생하는 문제를 보고 🡺 문제 생기면 자동으로 sender에게 전송됨
   * Data section: 받은 datagram의 IP header와 TCP header(맨 앞 8bytes만, source port, destination port, seq num이 포함됨)가 저장된다.

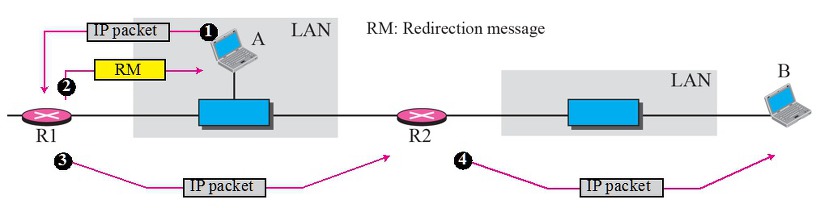
-> source address를 통해 어떤 IP에서 보내던 중 error발생했는지 알 수 있다.

-> seq num을 보고 몇 번째 packet이 전송 실패했는지 알 수 있다.

* + 다른 프로토콜과 마찬가지로 encapsulation되어 전송됨.

-> 전송 시 표기한 source address를 통해 오류 받는 쪽에서 error 발생 위치를 파악

* + Error 발생 경우
    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)



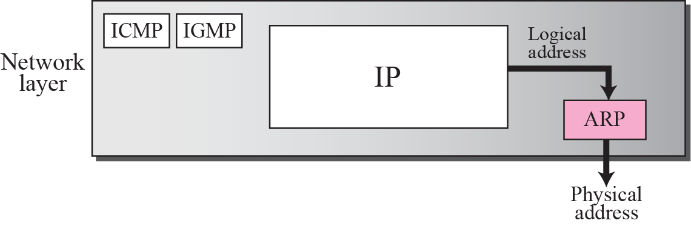
1. Query message: 네트웍 매니저나 host가 router나 다른 host에 대한 특정 정보를 얻는 것을 돕는다. 🡪 관리자가 물어봐야함
   * Echo-request message: 상대방 network layer 작동여부를 확인할 때 사용
   * Timestamp message: network layer에서 RTT를 측정할 때 사용

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

**ARP (Address Resolution Protocol)**

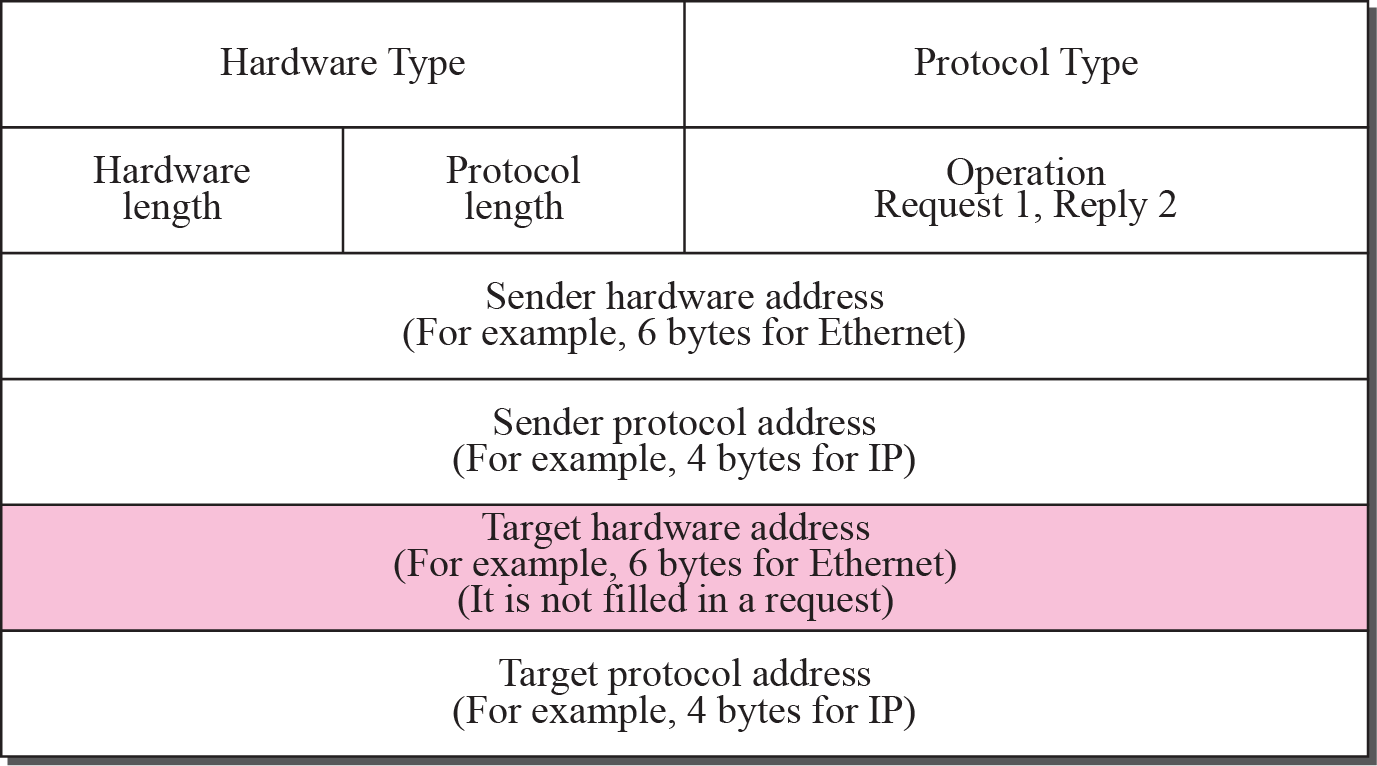
* IP프로토콜로부터 IP주소를 알려주면 상응하는 mac주소를 알려주는 프로토콜



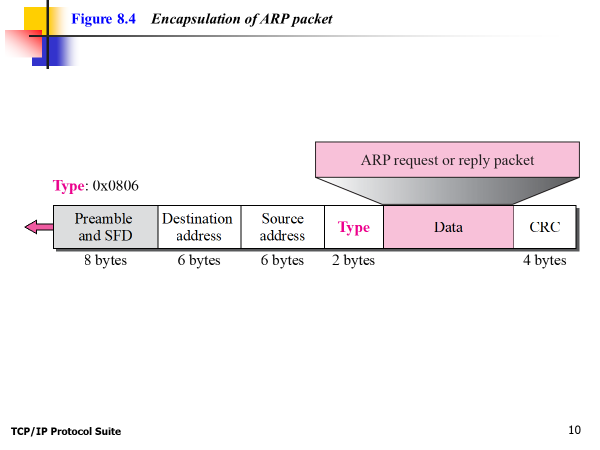
* Internetworking환경에서 MAC주소와 IP주소를 알아야 스위치, 라우터 조합의 네트워크 망에서 통신이 가능하다.
* 동작 원리:

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

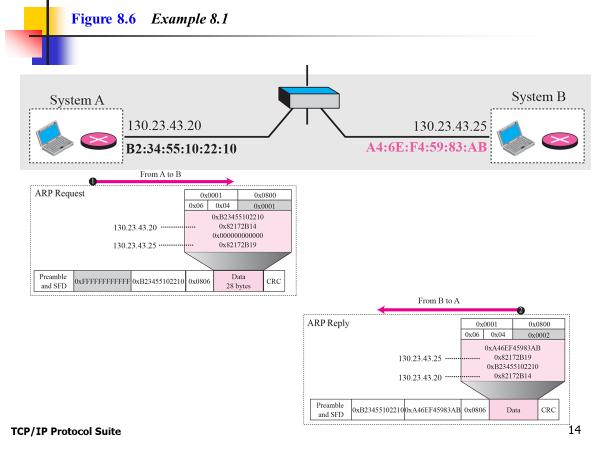


* Encapsulation of ARP packet



* 동작과정

1. Request - broadcast이므로 destination address의 모든 bit를 1로 설정하여 전송, 이때 ARP packet에서 Target hardware address부분은 궁금한 내용이므로 모든 bit 0으로 보낸다.
2. Reply – unicast이므로 destination과 source잘 적어서 전송, ARP packet의 주소내용 모두 채워서 전송한다. (sender hardware address에 자신의 mac주소 적어서 보내기)



* 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.