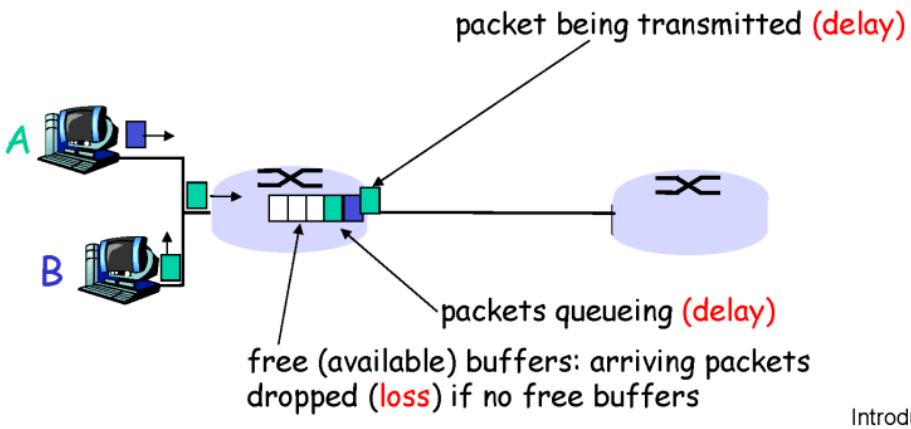
네트워크 계층 1, 2

How do loss and delay occur?

packets *queue* in router buffers

- packet arrival rate to link exceeds output link capacity
- packets queue, wait for turn



Introduction 1-31

packet이 큐의 맨 앞으로 와서 링크로 올라가는 시간이 transmitted dealy

그리고 link 에서 다음 라우터로 전달되는 시간이 빛의 속도

라우터가 처리할 수 있는 양보다 많은 packet 이 들어왔을 때 임시 공간에 packet 을 두는데 이것이 queueing delay이다.

만약 queue가 가득 찬 상황이면(no free buffers) 어쩔 수 없이 packet이 dropped(유실)되는 상황이 발생하고 이는 , 현재 인터넷에서 발생하는 loss 의 대부분을 차지하고 있음

HTTP connections

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

persistent HTTP

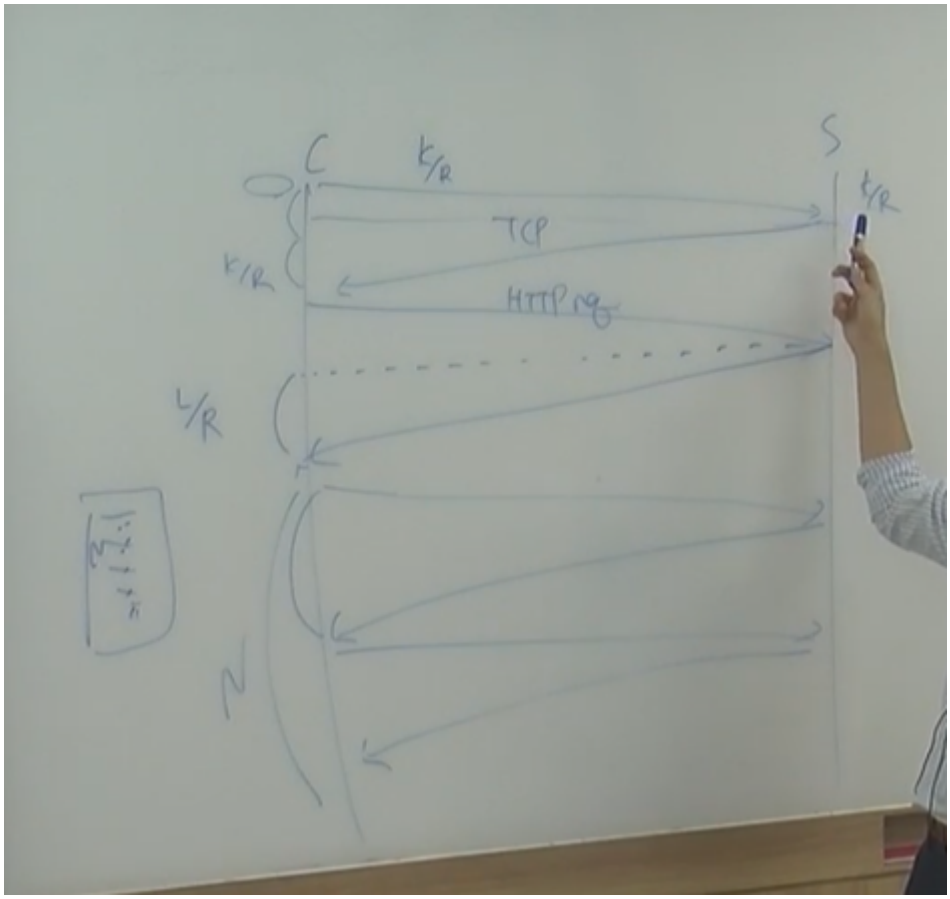
- multiple objects can be sent over single TCP connection between client, server

한번 연결한 TCP connetion 을 계속 사용할 것인지 아닌지에 따라 분류된다.

http protocol 에서는 기본적으로 persistent를 사용하고 있음

Sample problem

- What's end-to-end delay using persistent HTTP?
  - Control messages (e.g. TCP handshake, HTTP request) = **K** bit long
  - Base HTML object = **L** bits
  - **N** reference objects, each **L** bit long
  - Link bandwidth = **R** bps
  - Propagation delay = **d** seconds



처음에 three handshake 가 이루어 지는데, 3번째에는 데이터를 포함한 ack, (http request)도 같이 전달된다.

제일 처음에는 tcp 연결을 위한 tcp 씬 패킷이 전달되는데, 이때의 속도는 k/R + d

http request까지 걸리는 시간은 (k/R + d) * 3 ⇒ packet size가 똑같기 때문에

…. 필기 안 했음

UDP는 필드가 네개 밖에 없음 (기본적인 기능들만 해준다)

Recap: Principles of Reliable Data Transfer

- What can happen over unreliable channel?
  - Packet error, packet loss
- What mechanisms for **packet error**?
  - Error detection, feedback, retransmission, sequence#
- What mechanisms for **packet loss**?
  - Timeout!
- We built simple reliable data transfer protocol
  - Real-world protocol (e.g., TCP) is more complex, but *with same principles!*

신뢰성 없는 채널에서 발생하는 문제점: 패킷 오류와 패킷 유실

packet error 가 발생했을 때 처리하기 위한 메카니즘:

error 발생을 확인하고, 보낸 사람한테 feedback 하고, sender는 다시 재전송하면서 새로운 패킷을재전송. 이 때 패킷을 구별하기 위한 시퀀스 넘버가 필요하다.

패킷 유실이 발생했을 때: 타임아웃으로 확인하고 재전송한다.

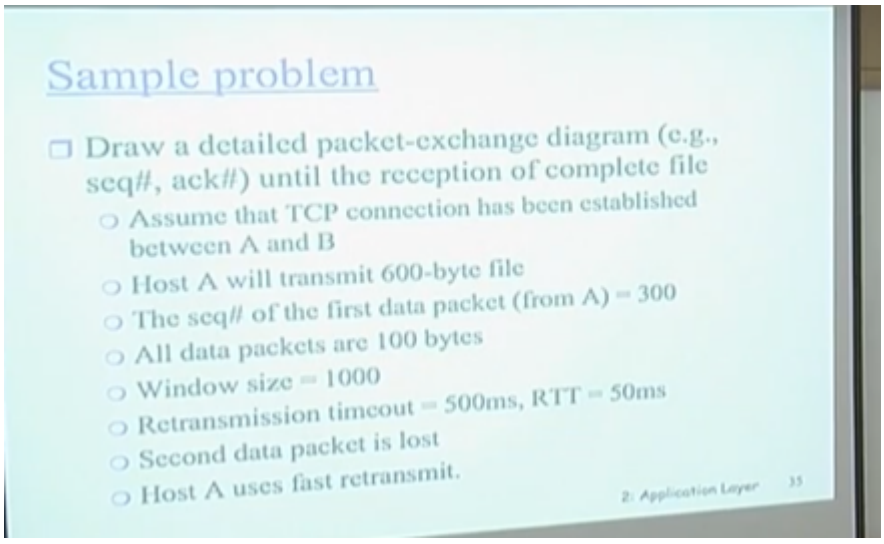
TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

Transport Layer 3-52

point-to-point: 두 개의 소켓만 통신하는 상황



TCP connection은 이미 맺어진 상황

A는 600바이트 파일을 보낼 것

seq은 300부터 시작

타임아웃은 500

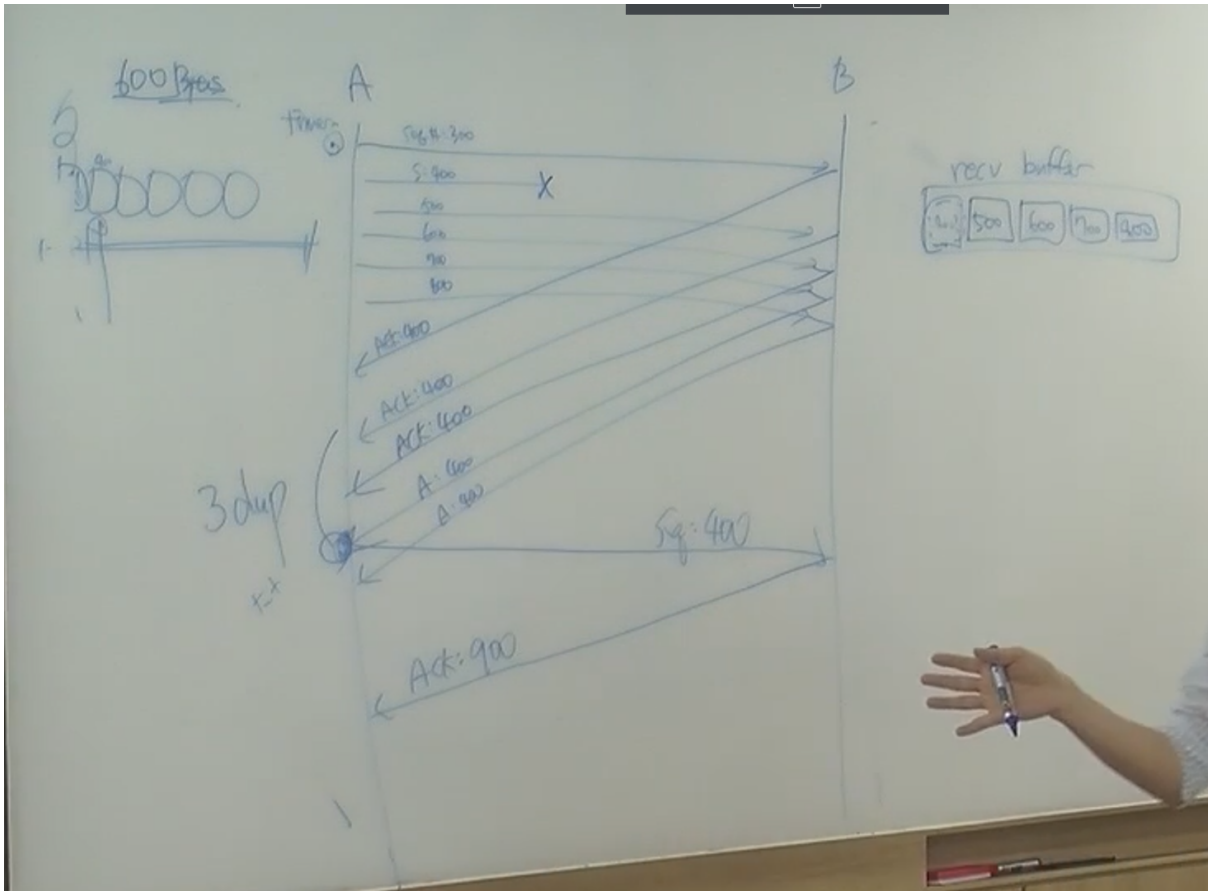
round trip time 은 50

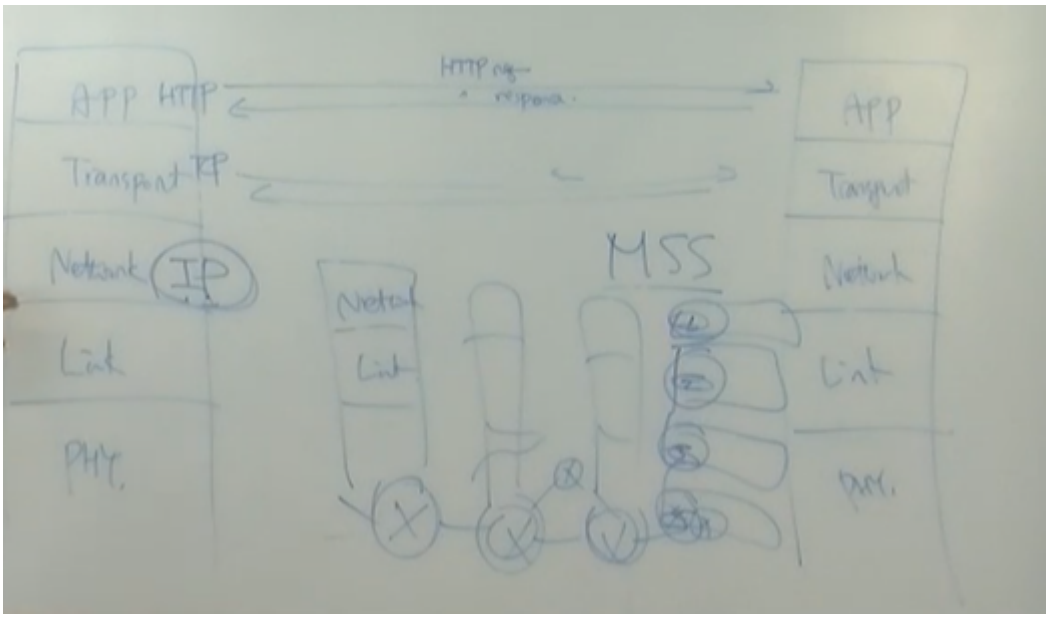
A는 fast retransmit 을 사용(3 duplicated ack일 때 유실로 판단)

윈도우 사이즈 1000이고, A가 보낼 데이터는 600이니까 한꺼번에 다 보낼 수 있는 상황

packet의 단위는 100이기 때문에 6개의 packet을 보낸다

1. A 가 300번을 보내고, B는 400ack를 보낸다.
2. A 가 400번을 보내지만 유실된다.
3. A 가 500번을 보내고, B는 400 ack를 보낸 후 500을 receiver buffer에 저장한다
4. A가 ack 400을 받았을 때는 B가 300은 잘 받았다는 의미니까 윈도우를 한 칸 전진하고, 300에 물려있던 time out을 400에 물린다.
5. A가 ack 400을 두번째로 받았을 때는 할 일이 없으니 아무것도 하지 않는다.
6. ack 400 3번 이상 받았을 때 400번을 다시 보낸다.
7. B는 seq 400을 받고 ack 900을 보내면서, A는 윈도우를 완전히 전진시키고 다음 데이터를 보낸다.

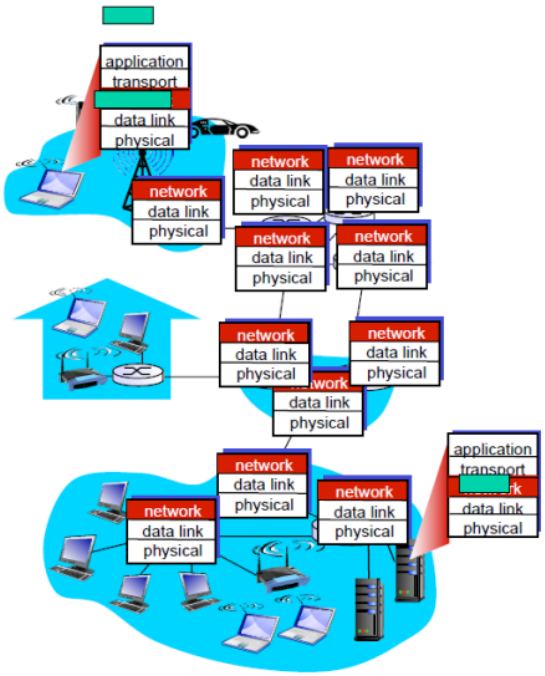




http 메시지가 tcp segment에 담기고, TCP segment 가 ip packet에 담겨서 보내지면 라우더는 ip packet을 받아서 헤더의 내용을 분석하고 알맞은 방향으로 보내준다.

Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



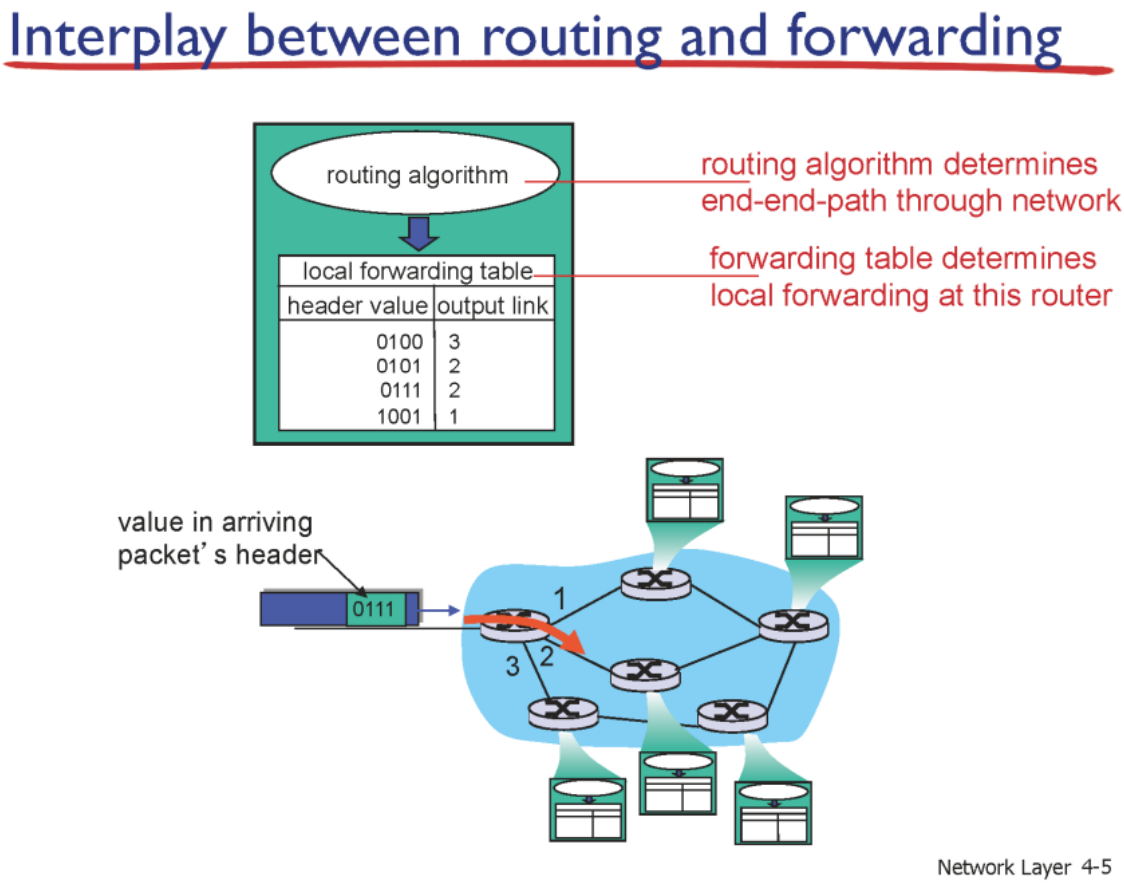
Network Layer 4-3

Two key network-layer functions

- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to dest.
  - *routing algorithms*

analogy:

- *routing:* process of planning trip from source to dest
- *forwarding:* process of getting through single interchange



라우터에서 하는 작업은 두 가지:

forwarding: forwarding table을 통해 packet의 목적지를 확인하고, 해당 entry로 보낸다. (0101이면 2번 링크로)

(forwarding table이 있어야 forwarding이 가능한데, table은 routing 알고리즘이 만든다.)

routing: 전 세계 호스트가 너무 많으니까 예제처럼 자세하게 적혀 있지 않고, 주소 범위로 forwarding table을 만든다. (대전, 서울, 부산, 광주 등등)

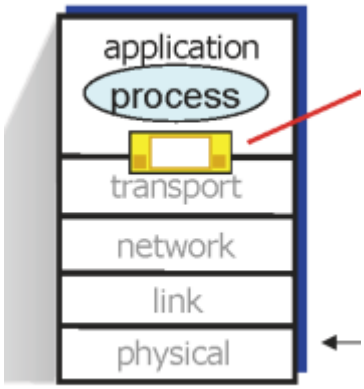
네트워크 계층 2

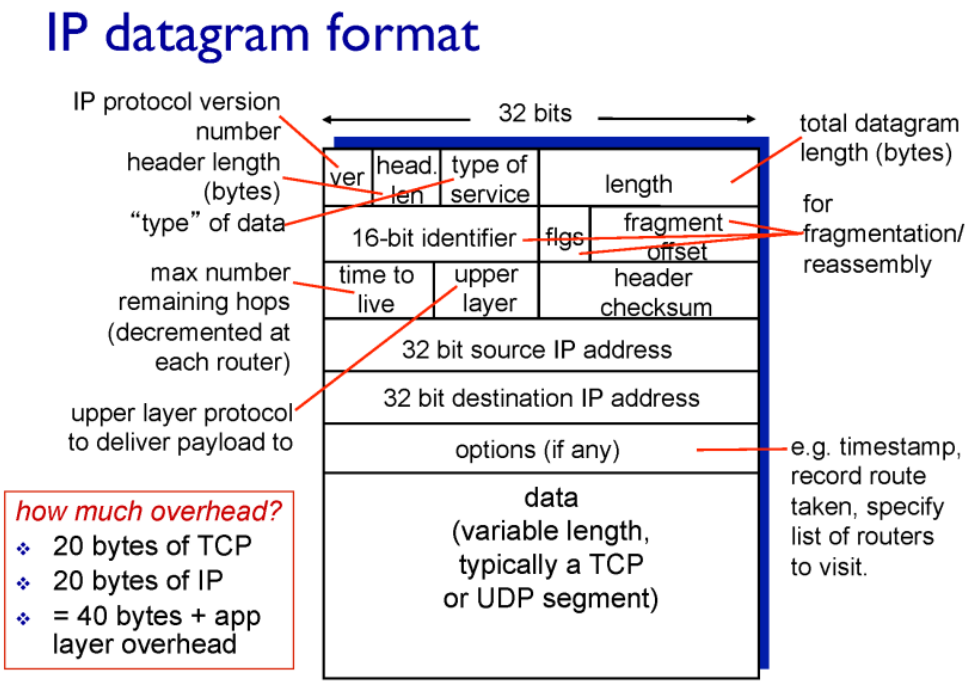
애플리케이션 계층에서는 데이터 전송 단위를 메시지라고 한다.

transport에서 전송 단위는 segment, (segment 는 header와 데이터 부분으로 이루어져 있다. 이 때 데이터 부분에 애플리케이션 메시지가 들어가게 된다.)

네트워크 계층의 전송 단위는 전송단위는 패킷 (header 와 데이터 부분으로 이루어져 있다. 데이터 부분에 tcp 세그먼트가 들어가게 된다.)

링크 계층의 전송단위는 프레임(header와 데이터 부분으로 이루어져 있고, 데이터 부분에 packet이 들어가게 된다.)





source IP address는 보내는 사람 주소

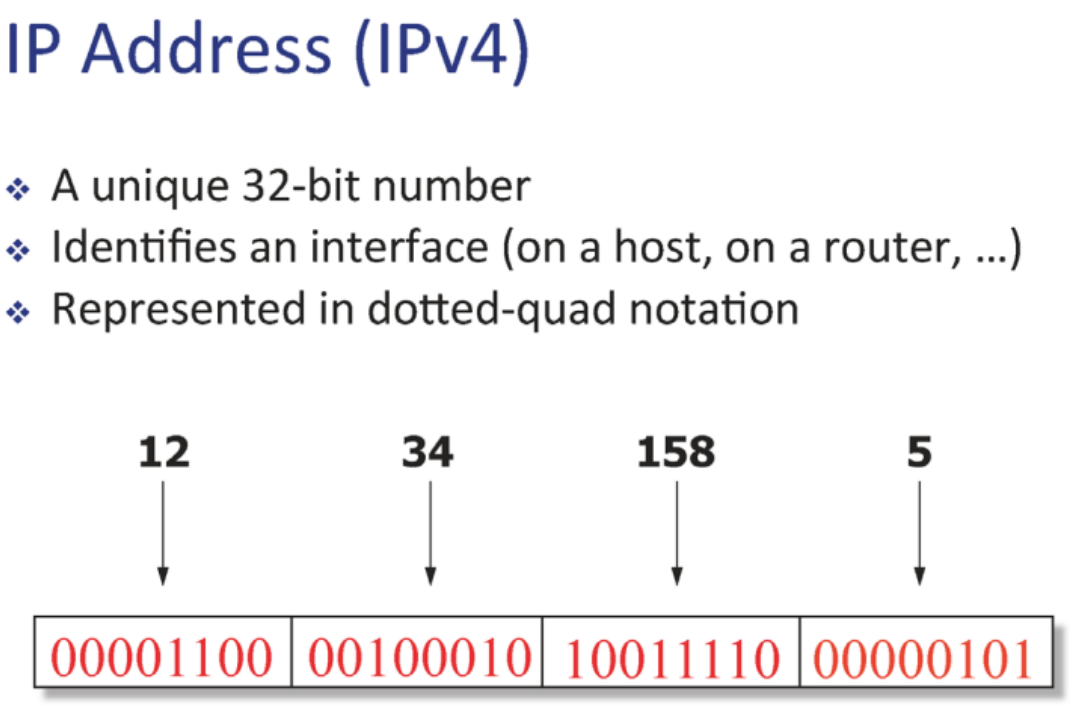
1. time to live: router를 거칠수록 -1 이 되도록 되어 있고, 0이 되는 순간 이 패킷은 버려지게 된다. packet이 destination IP 도착하지 못 하고, 계속해서 네트워크 자원을 소모하는 것을 방지하기 위한 필드다.

2. upper layer: TCP인지 UDP인지 나타내는 필드로, 이 필드를 통해 receiver는 어떤 방법을 사용해야 할지 알 수 있다.

TCP헤더와 IP헤더 = 40바이트

40바이트 + 데이터

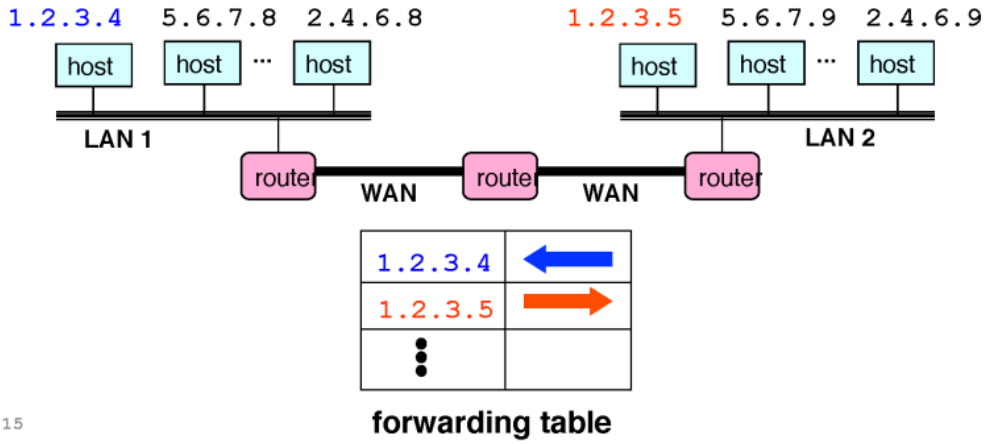
인터넷에는 40바이트짜리 packet이 많이 돌아 다니는데 이는 ack 정보만 담긴 packet이다.



IP주소는 32비트 체계로 이루어져 있다. 사람이 읽을 때는 8비트씩 끊어서 10진수로 변환해서 읽는다. 한 단위는 최대 255까지 갈 수 있다. (한 단위당 8비트이니까)

IP주소는 host에 들어있는 network 인터페이스를 지칭한다. (network 인터페이스 카드)

- Suppose hosts had arbitrary addresses
  - Then every router would need a lot of information
  - ...to know how to direct packets toward every host

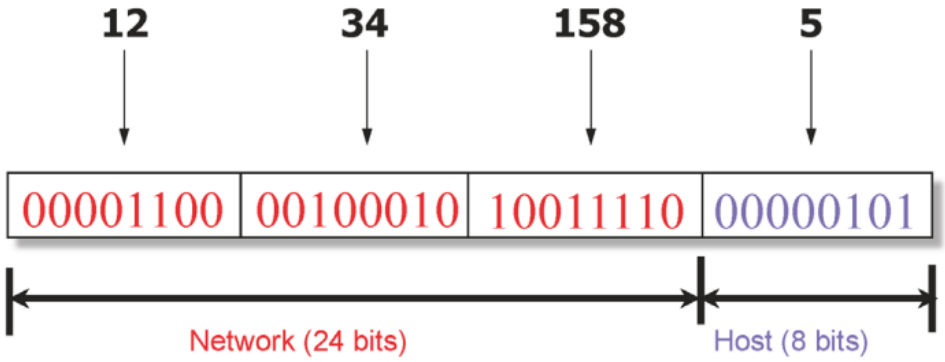


15

IP주소를 무작위로 배정하게 되면 보내야 할 곳의 방향이 너무 많이 생기게 된다.
(forwarding table이 커지기 때문에 주소 검색이 힘들어진다)

Hierarchical Addressing: IP Prefixes

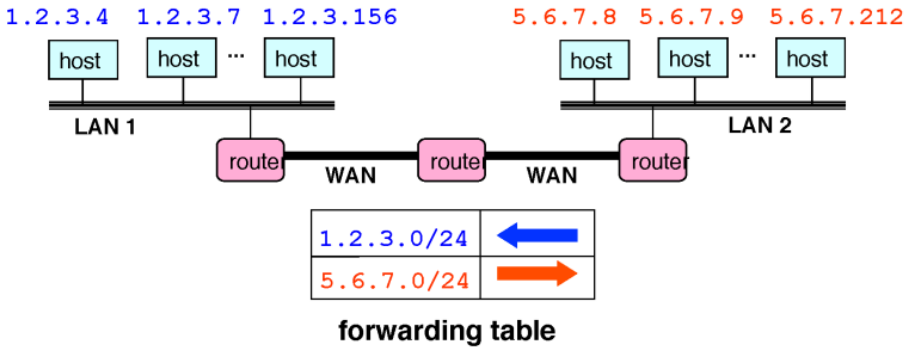
- Network and host portions (left and right)
- 12.34.158.0/24 is a 24-bit **prefix** with 2^8 addresses



따라서 IP주소는 체계를 가지게 된다.

앞부분은 네트워크 아이디(prefix), 뒷부분은 호스트이다.

- Number related hosts from a common subnet
  - 1.2.3.0/24 on the left LAN
  - 5.6.7.0/24 on the right LAN



18

같은 prefix 를 가진(네트워크) 주소끼리 묶이게 된다.

한양대, 시립대 모두 네트워크를 갖는데, prefix의 크기는 다 다르다.

Classful Addressing

- In the old days, only fixed allocation sizes
  - Class A: 0*
    - Very large /8 blocks (e.g., MIT has 18.0.0.0/8)
  - Class B: 10*
    - Large /16 blocks (e.g,. Princeton has 128.112.0.0/16)
  - Class C: 110*
    - Small /24 blocks (e.g., AT&T Labs has 192.20.225.0/24)
  - Class D: 1110* for multicast groups
  - Class E: 11110* reserved for future use
- This is why folks use dotted-quad notation!

옛날 방식:

class A

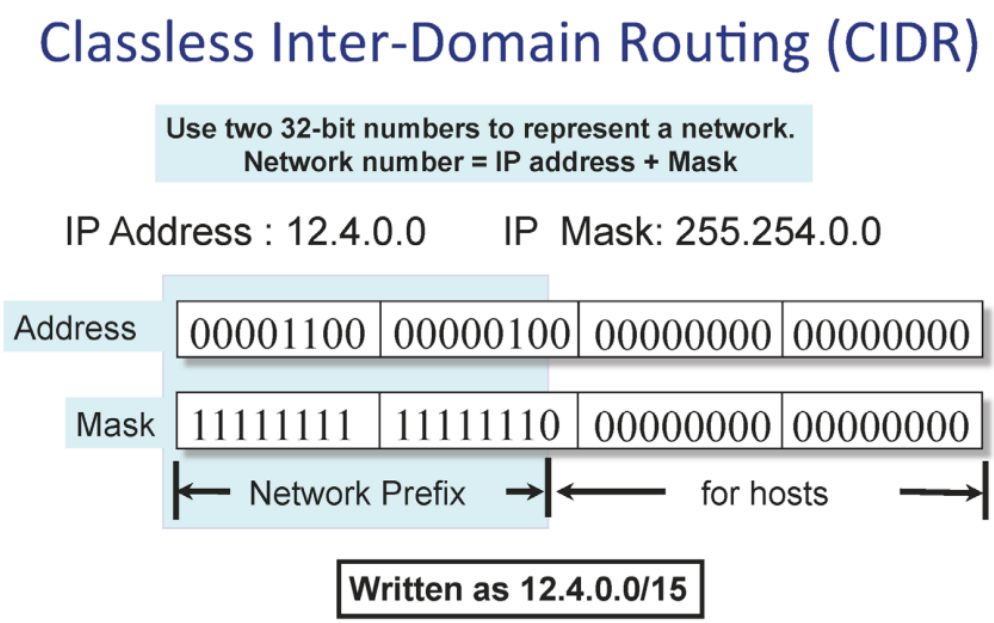
/8 의 네트워크 주소를 배정 받으면, 나머지 2^24개의 호스트를 보유할 수 있다.

클래스 A의 앞은 0으로 시작. 전세계에서 128개의 기관만이 이 주소를 받을 수 있었다.

class B

2^16기관이 배정 받을 수 있으며, 각 기관들은 2^16 호스트를 보유할 수 있다.

비효율적인 방법이었기 때문에 90년대 중반에 방법이 바뀌게 됐다.



class를 없애고 도입한 새로운 방법

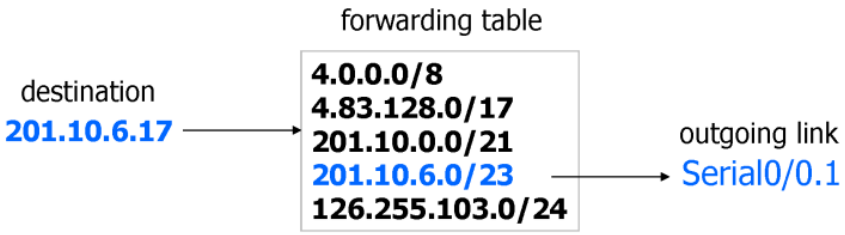
prefix가 8비트 단위가 아닌 자유롭게 끊어지게 된다. 예제 사진에서는 prefix는 15비트를 차지한다.

만약 한양대학교가 1000개의 호스트를 보유하고 싶으면, prefix는 22/ host는 10을 가져가면 된다. (2^10 = 1024)

옛날 클래스 방식이었으면 prefix를 4개(한 개당 256 호스트) 가지고 있었어야 함.

Longest Prefix Match Forwarding

- Destination-based forwarding
  - Packet has a destination address
  - Router identifies longest-matching prefix
  - Cute algorithmic problem: very fast lookups



29

라우터 안에 있는 forwarding table에서 destination adress에 매칭되는 entry 를 찾는다.

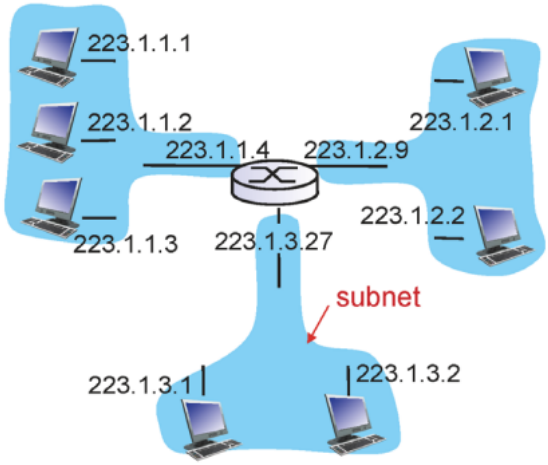
세번째와 네번째가 매칭이 된다.

여러개 매칭되는 것 중에서 가장 구체적으로 매칭이 되는, prefix가 가장 긴 entry에 매칭을 시킨다.

>>이런 일을 하는 것이 라우터

Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits
- *what's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without intervening router*



network consisting of 3 subnets

같은 prefix를 가진 device 집합 > subnet

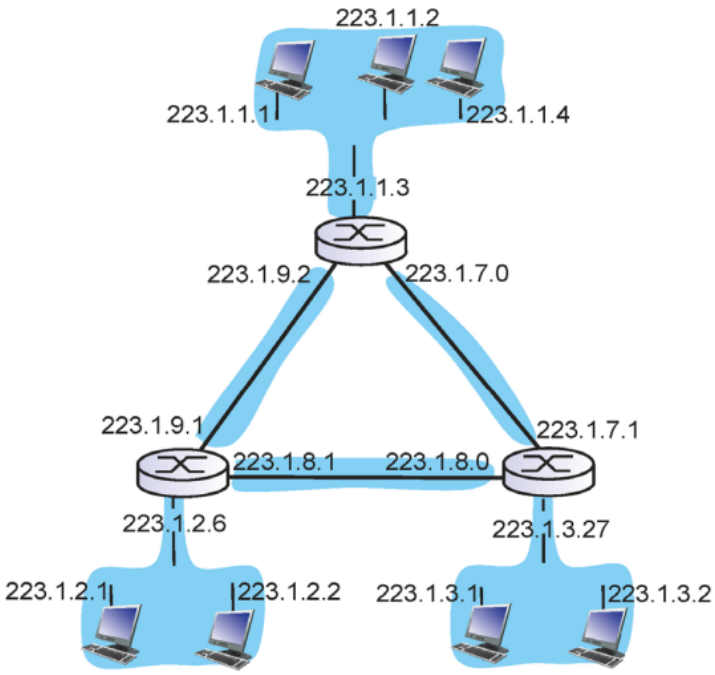
subnet > 라우터를 거치지 않고 접근이 가능

라우터도 인터페이스가 있기 때문에 IP주소를 갖는다. (라우터 갯수만큼, 사진에서는 3개를 갖는다)

라우터의 IP주소들의 prefix가 다 다르다. 따라서 라우터는 여러개의 subnet에 속하게 된다.

Subnets

how many?



위의 경우 subnet은 6개

IP4 일 때는

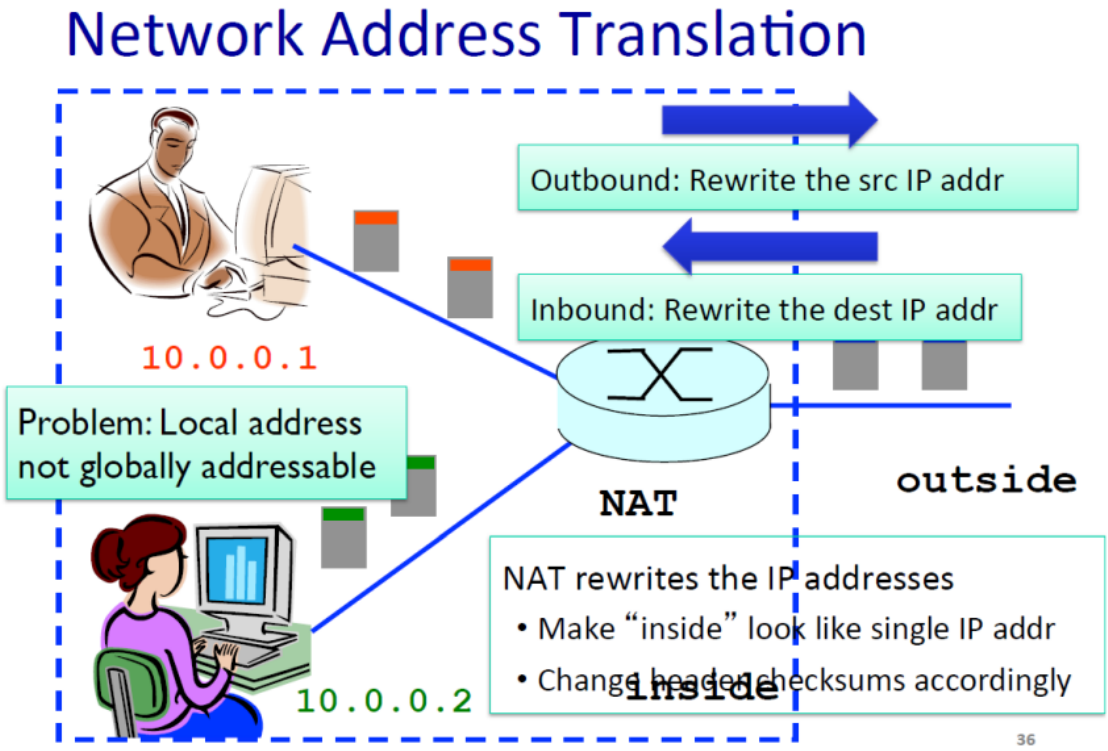
주소공간이 32비트 (최대 2^32 host = 40억이 가능)

IP6 을 설계 (90년대 디자인)

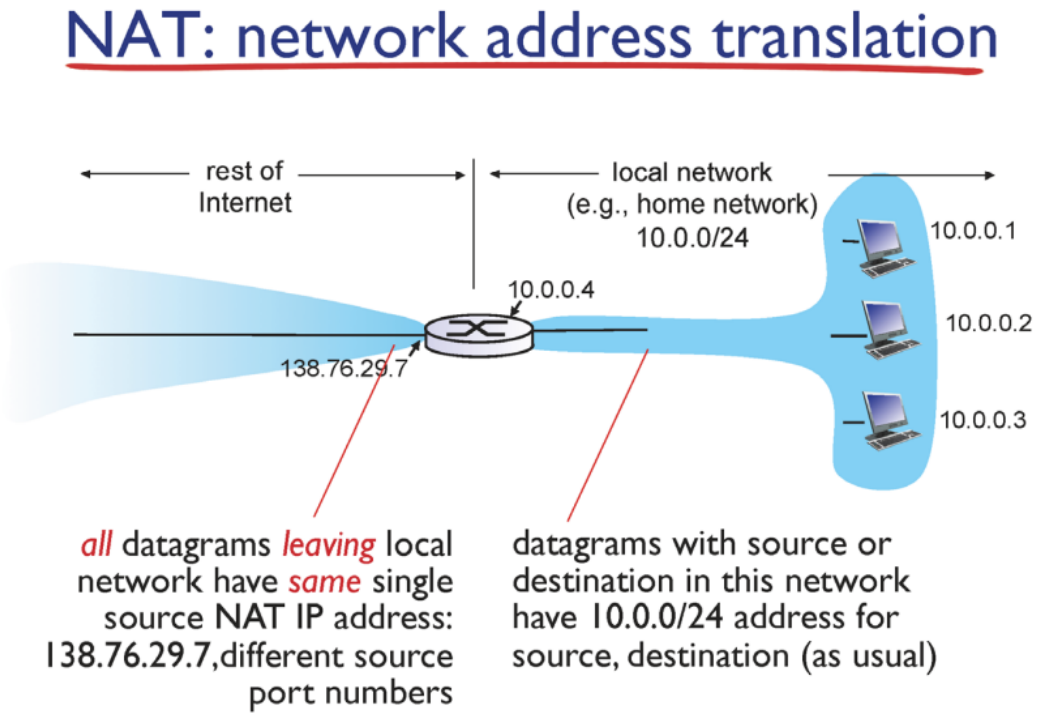
주소공간 128bit (최대 2^128 가능. 지구상 모레알 갯수만큼 가능)

현재 아직까지 IP4를 쓸 수 있는 이유는?

>> 현재 40억개의 주소 공간을 서로 공유해 가면서 쓰고 있음 (Network Address Translation)



네트워크 내부에서는 유일한 주소를 사용. 이런 IP주소가 외부로 나갈 경우, NAT 라우터에서 IP 주소를 변경한다.



아이피 주소가 외부로 나갈 때 유일한 IP주소가 아니기 때문에 라우터에서 주소를 변경한다. 들어올 때도 주소를 바꿔준다. >> source IP와 포트넘버를 바꾼다. (이 때 라우터가 하는 짓은 헤더부분 뿐만 아니라 즉 데이터, 우편부가 편지봉투를 뜯어서 내용을 변경하는 것과 같다.)

IP주소는 호스트 인터페이스 찾아갈 때 쓰이고, 포트넘버는 호스트를 찾아갔을 때 호스트 내부에서 프로세스를 찾아갈 때 쓰는 것이다.