

## Ch1. 인터넷의 구성

1. Host: 인터넷 시스템의 end(클라이언트와 서버)
  2. Packet switches: packet(chunks of data)를 포워딩한다
  3. Communication links: 각 Host, Packet switch들을 연결하는 링크(bandwidth이라는 전송률을 가진다)
  4. Networks: host, router, link의 집합
- + Host to Host 통신은 네트워크를 segment로 나누고 segment마다 header를 덧붙여 보내는데, 이때 보내는 정보 단위를 패킷이라고 한다.

Internet: network of networks

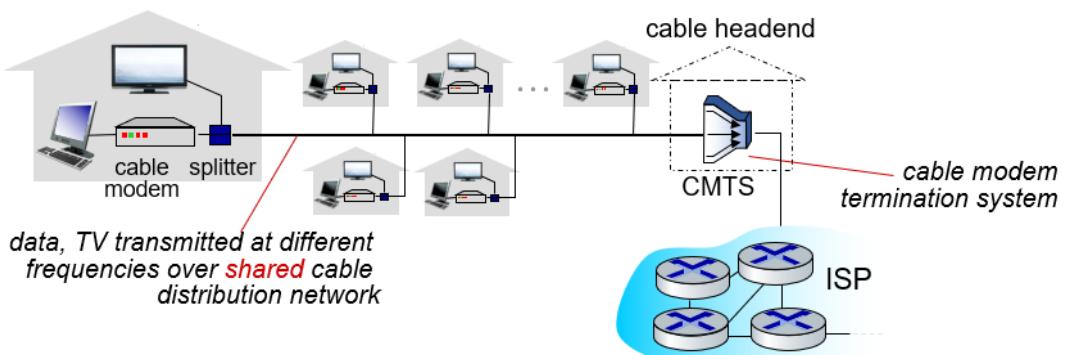
1. 상호 연결된 ISP(internet service protocol)
2. Protocol을 이용하여 정보를 주고받는다
3. 인터넷 표준
  - A. RFC-Request for Comments
  - B. IETF-Internet Engineering Task Force

Protocol: message 전달에 있어서 형식과 message 전달 순서를 정의하고 이에 따른 이벤트들을 정의하는 규약

## Network Edge

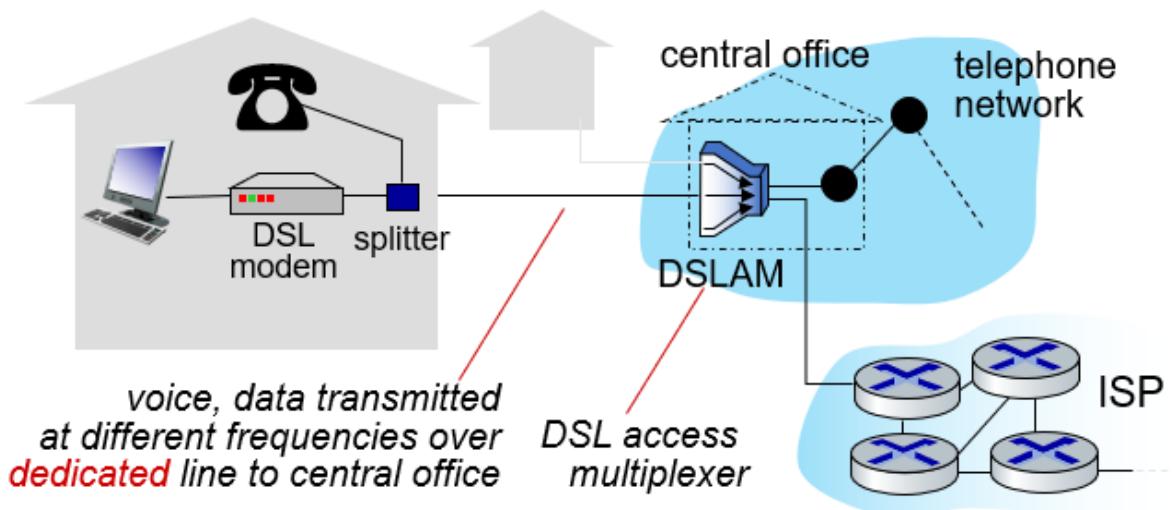
1. Access networks

### A. HFC (Hybrid Fiber Coax)



- i. 케이블 TV를 공급하는 선을 통하여 인터넷을 사용하는 방법
- ii. 많은 사람들이 하나의 선을 통하여 정보를 받기 때문에 혼선이 일어나거나 하이 트래픽으로 인한 버벅임이 생기기도 함
- iii. asymmetric하기 때문에 다운로드 속도는 40Mbps-1.2Gbps고 업로드 신호는 30-100Mbps의 전송률을 가진다

#### B. DSL (Digital Subscriber Line)



- i. 전화선을 통하여 인터넷을 사용하는 방법
- ii. HFC와 다르게 한 집에 한 선이 보급되었다
- iii. 다운로드 속도는 24-52Mbps이고 업로드 속도는 3.5-16Mbps이다.

#### 2. Host

- A. 데이터를 보낼 때, 작은 packet으로 나누어 링크로 transmission rate 속도로 링크로 전달한다
- B. 각각  $L$ 비트인 packet을  $R(\text{bit/sec})$ 의 transmission rate로 전달할 때 packet transmission delay는 다음과 같다

$$\text{packet transmission delay} = \frac{\text{time needed to transmit } L\text{-bit packet into link}}{R \text{ (bits/sec)}} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

#### 3. Links

- **bit:** propagates between transmitter/receiver pairs
- **physical link:** what lies between transmitter & receiver
- **guided media:**
  - signals propagate in solid media: copper, fiber, coax
- **unguided media:**
  - signals propagate freely, e.g., radio

### Coaxial cable:

- two concentric copper conductors
- bidirectional
- broadband:
  - multiple frequency channels on cable
  - 100's Mbps per channel

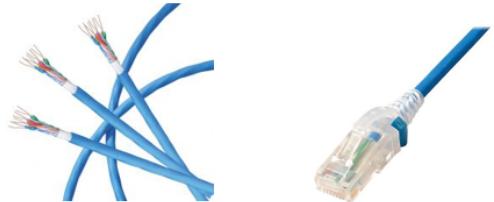


### Wireless radio

- signal carried in various “bands” in electromagnetic spectrum
- no physical “wire”
- broadcast, “half-duplex” (sender to receiver)
- propagation environment effects:
  - reflection
  - obstruction by objects
  - Interference/noise

### Twisted pair (TP)

- two insulated copper wires
  - Category 5: 100 Mbps, 1 Gbps Ethernet
  - Category 6: 10Gbps Ethernet



### Fiber optic cable:

- glass fiber carrying light pulses, each pulse a bit
- high-speed operation:
  - high-speed point-to-point transmission (10's-100's Gbps)
- low error rate:
  - repeaters spaced far apart
  - immune to electromagnetic noise



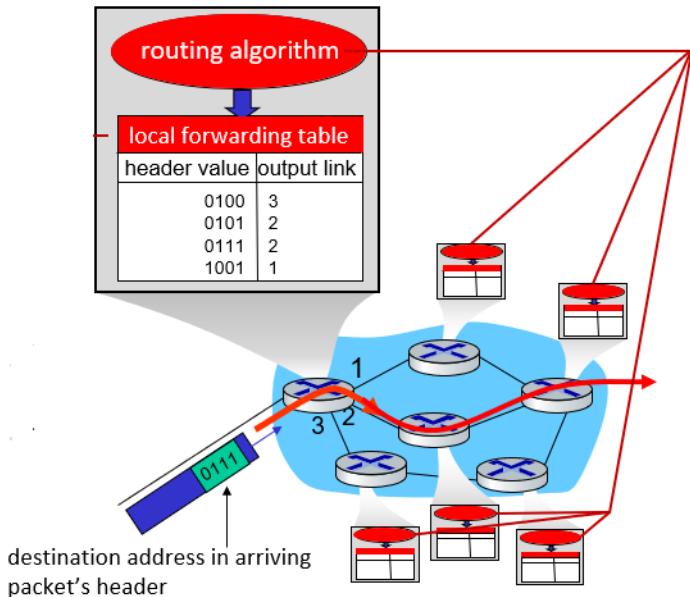
Introduction: 1-23

### Radio link types:

- **Wireless LAN (WiFi)**
  - 10-100's Mbps; 10's of meters
- **wide-area** (e.g., 4G cellular)
  - 10's Mbps over ~10 Km
- **Bluetooth:** cable replacement
  - short distances, limited rates
- **terrestrial microwave**
  - point-to-point; 45 Mbps channels
- **satellite**
  - up to 45 Mbps per channel
  - 270 msec end-end delay

## Network Core

### 1. Network Core의 주요 두 가지 기능



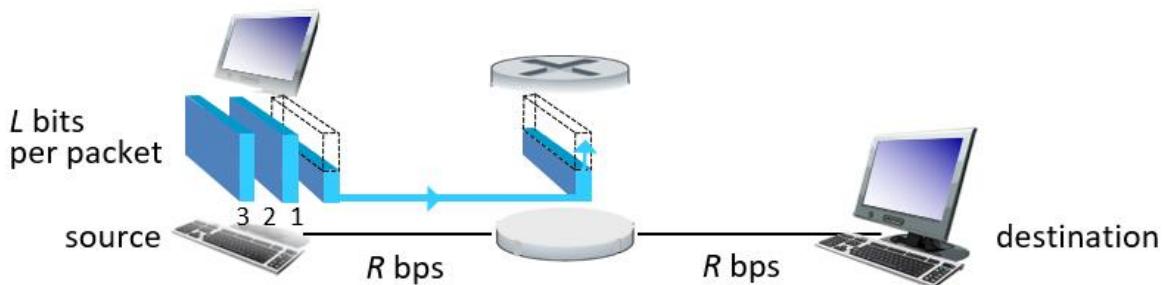
A. Routing: packet이 거쳐갈 source-destination path를 Routing algorithms를 통해 결정하는 것이다. (global action)

i. Routing algorithms: local forwarding table을 만드는 알고리즘

B. Forwarding: router의 input link에 도착한 packet들을 local forwarding table에 따라 적절한 output link로 보내주는 작업(local action) (=switching)

i. local forwarding table: Routing algorithms로 만들어진 테이블로 packet들의 destination을 통해 적절한 output link를 정해주는 table

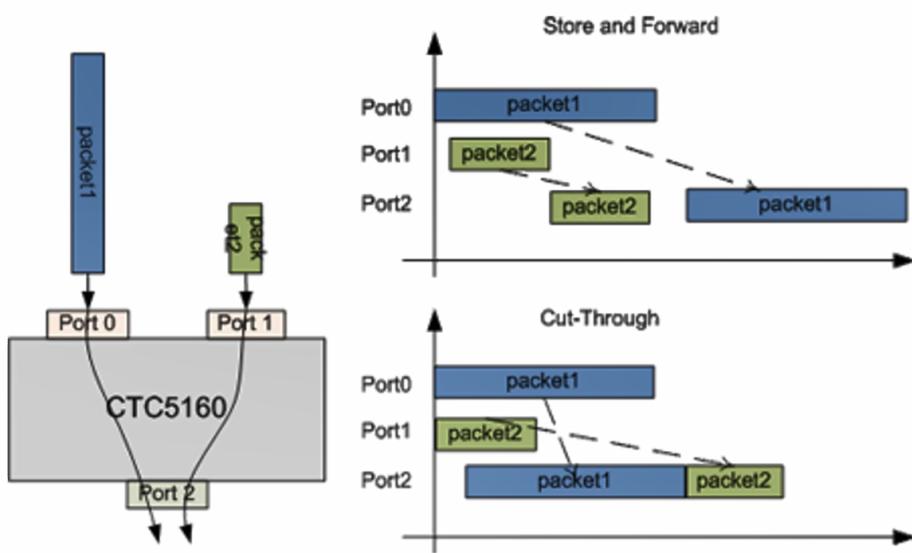
### 2. Packet switching



상황: 해당  $L$  bit인 packet을  $R \text{ bps}$ 의 전송속도를 가진 경로를 지나고 있다.

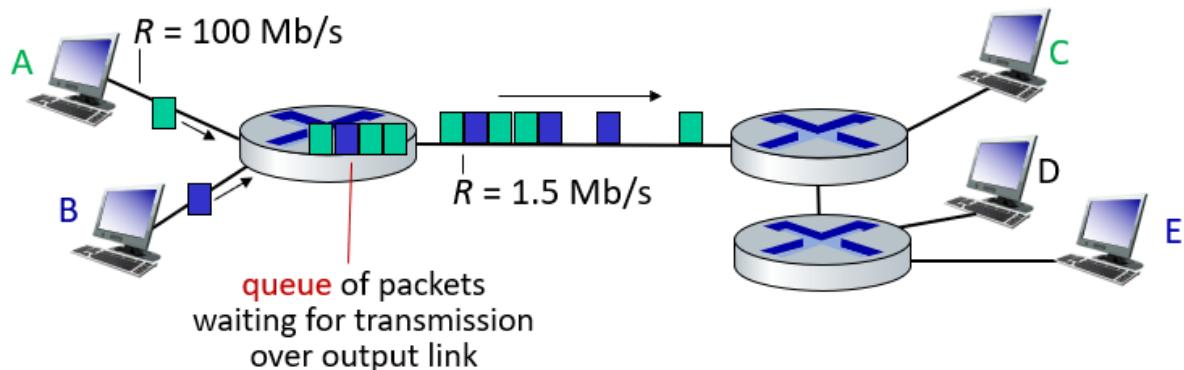
+ hop: 네트워크에서 source-destination 사이에 위치한 경로의 한 부분. Hop count는 경로에서 한 장치에서 다른 장치로 이동할 때마다 1씩 증가한다. (위의 경우, source-router, router-destination, 2 hop이다.)

- A. Packet transmission delay: L bit packet 파일을 R bps를 가지는 한 hop에서 packet 을 전달하기 위해서는  $L/R$  sec가 필요하다.
- B. Store-and-forward
  - i. 한 hop 사이에 packet을 전송할 때 그 packet이 다 전송될 때까지 기다린 후에 다음 hop으로 전송을 시작하는 방법
- C. Cut-Through
  - i. 전체 packet이 도착하기 전에 packet-header를 보고 목적지 주소로 가는 포트로 전송을 시작하는 방법
- D. 비교



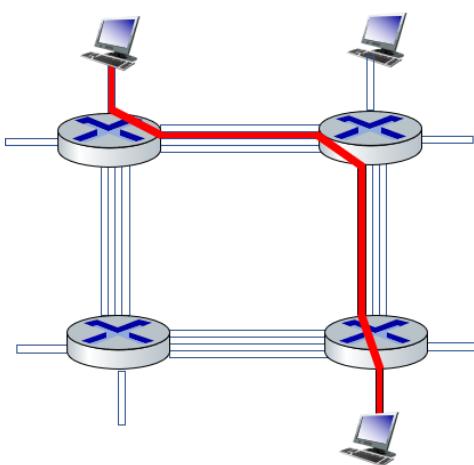
상황. Port0, 1로 packet1, 2를 들여오고 이 packet들은 port2로 전송된다

- i. Store and Forward에 비해 Cut-Through는 전송 시간이 짧다.
- E. Queueing



상황. Host-router 전송률 >> Router-Router 전송률이기 때문에 Router-Router 구간에서 병목현상이 일어난다. (Dumbbell structure, Bottleneck structure)

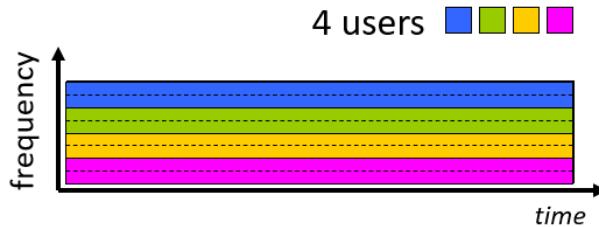
- i. Packet queuing and loss: Dumbbell structure처럼 한 쪽의 output link의 속도가 많이 느리다면 병목 지점의 queue(buffer)에 보내기 전의 packet들이 쌓이게 된다.
  - 1. Packet queuing: output link로 전송되는 것을 기다리며 packet은 queue 된다.
  - 2. Packet loss: queue(buffer)가 꽉 찬는데 계속 들어온다면 packet이 손실된다. (어떤 packet이 손실되는지는 network의 policy를 따른다)
- 3. Circuit switching



- A. Circuit switching: 종단 시스템 사이의 path 중 일부를 할당 받아 그 path로 통신하는 방식.
  - i. Dedicated resources: 자원 sharing이 없기 때문에 circuit-like (guaranteed) performance를 기대할 수 있다
  - ii. 할당 받은 circuit segment를 사용하지 않으면 no sharing이기 때문에 자원

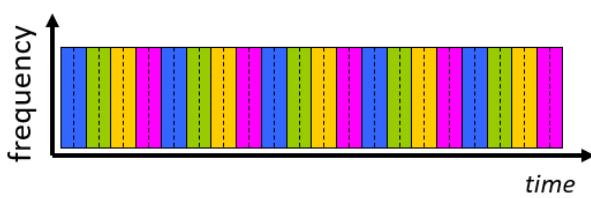
낭비가 발생할 수 있다

### B. Frequency Division Multiplexing (FDM)



- i. 광학적, 전자기적 frequency를 frequency band로 나누는 방법(frequency band)
- ii. 각각의 통신은 own band의 max rate로 통신할 수 있다

### C. Time Division Multiplexing (TDM)

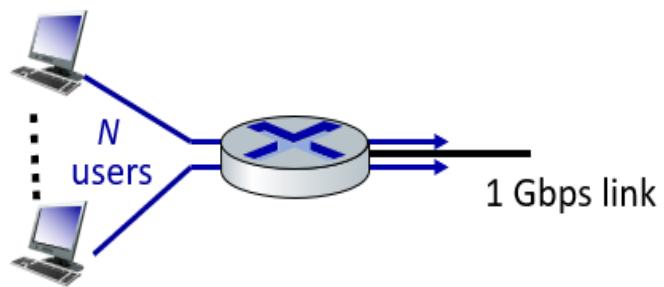


- i. 시간을 나누어 시간대별로 각각의 통신이 이용하는 방법(periodic slot)
- ii. 각각의 통신은 periodic slot의 시간동안 frequency band 전체를 사용할 수 있다

### 4. Packet switching과 Circuit switching의 정량적 비교

**example:**

- 1 Gb/s link
- each user:
  - 100 Mb/s when "active"
  - active p% of time

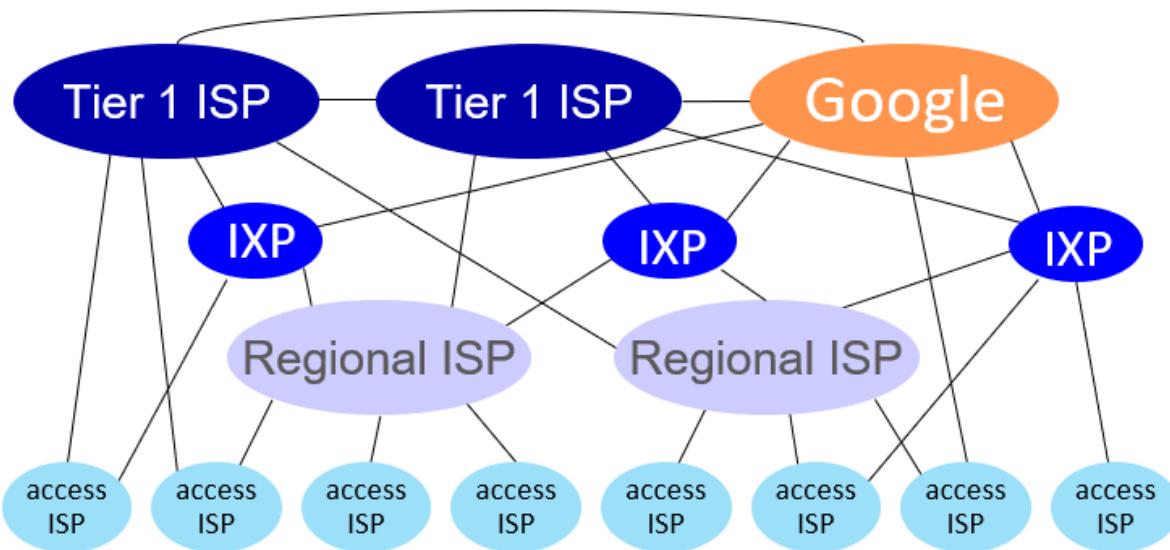


- A. 네트워크 최대 동시 접속자 수:  $1\text{Gb}/\text{s} = 1000\text{Mb}/\text{s}$ ,  $1000/100 = 10$ 이므로 최대 10명의 유저가 쓸 수 있다
- B. Circuit switching: TDM은 항상 1명만 쓰고 FDM도 회선을 분할 할당하는 거라 10명이 최대다

C. Packet switching: 사람  $n$ 명이 동시에 접속하고 있을 확률은  $P(n) = (N, n) * p^n * (1-p)^{N-n}$ 이다. 그래서  $1 - (P(n+1) + P(n+2) + \dots + P(N))$ 의 확률로 네트워크 사용에는 문제가 없고 이는 매우 높은 확률이라 많은 인원이 모여도 네트워크를 사용할 수 있다.

## 5. Internet structure

A. 문제상황: 호스트들은 access ISP들을 통하여 인터넷에 접속한다. >> 그렇다면 모든 ISP들이 interconnected해야 any two hosts가 통신할 수 있다. >> 그렇게 하기 위해서는 경제적인 비용도 많이 들고 지역마다 policy가 달라서 복잡해진다.

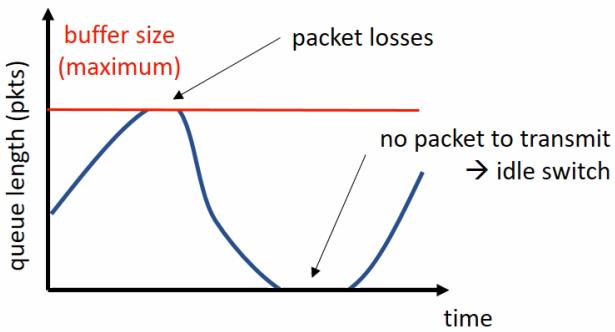


B. Regional ISP, Tier 1 ISP로 access ISP를 모으고 이 access ISP가 접속한 ISP들이 IXP(Internet Exchange Point)로 interconnect한다. 또한 Google, Facebook 같은 회사들은 content provider network를 이용하여 private network를 운영하기도 한다.

## Network performance

1. Packet loss: packet 도착 속도가 output link capacity보다 높으면 queue(buffer)가 쌓이고, queue가 다 차면 그 이후로 도착하는 packet들은 loss(drop)된다.

A. Buffer size: buffer size를 크게하거나 작게하는 것으로 다양한 효과를 얻을 수 있다.

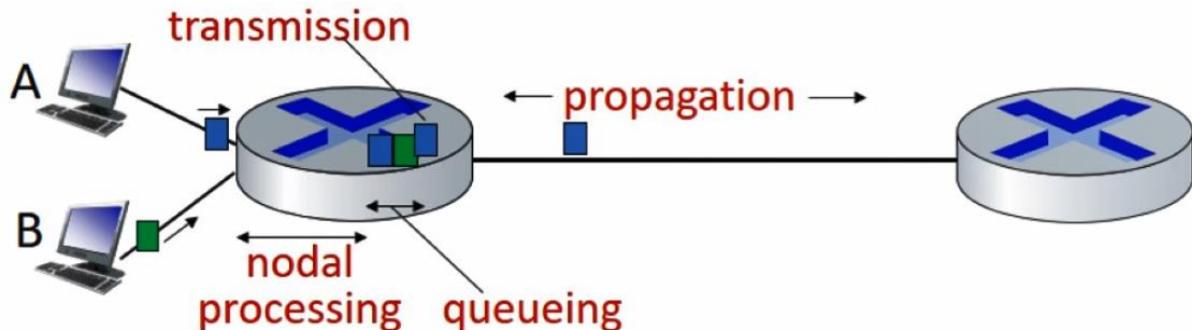


i. Buffer size와 특성의 관계

1. Buffer size가 크면 보관할 수 있는 packet의 양이 많아 지기 때문에 packet loss가 적어진다
2. Buffer size가 작으면 보관할 수 있는 packet의 양이 적다>> 평균 queue length가 작다>> 평균 queue delay가 작아진다
3. Buffer size가 크면 한동안 traffic이 적어도 bursty traffic 때 받은 패킷들이 많기 때문에 switch가 쉬는 경우가 적어져서 switch utilization이 높아진다

+ ) utilization: 장치가 실제로 얼마나 일을 하고 있는지 알려주는 지표로 장치의 최대 일/실제 하는 일

2. Packet delay: 한 노드마다 자료를 받아서 보내는데 걸리는 delay =  $d_{nodal}$



$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

$$\text{A. } d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

B.  $d_{proc}$ : nodal processing delay

- i. Bit 에러를 검출하고 routing을 통해 output link를 정하는 등, 노드가 packet을 받았을 때 처리해야 되는 알고리즘을 처리하는데 걸리는 시간(매우 작다)

C.  $d_{queue}$ : queueing delay

- i. Output link에서 transmission을 기다리는 시간(router의 congestion level에 좌우된다)

D.  $d_{trans}$ : transmission delay

- i. Packet을 output link로 옮기는 데 걸리는 시간

- ii.  $L$  bit의 packet을  $R$  bps로 옮긴다면  $d_{trans} = \frac{L}{R}$  sec 걸린다

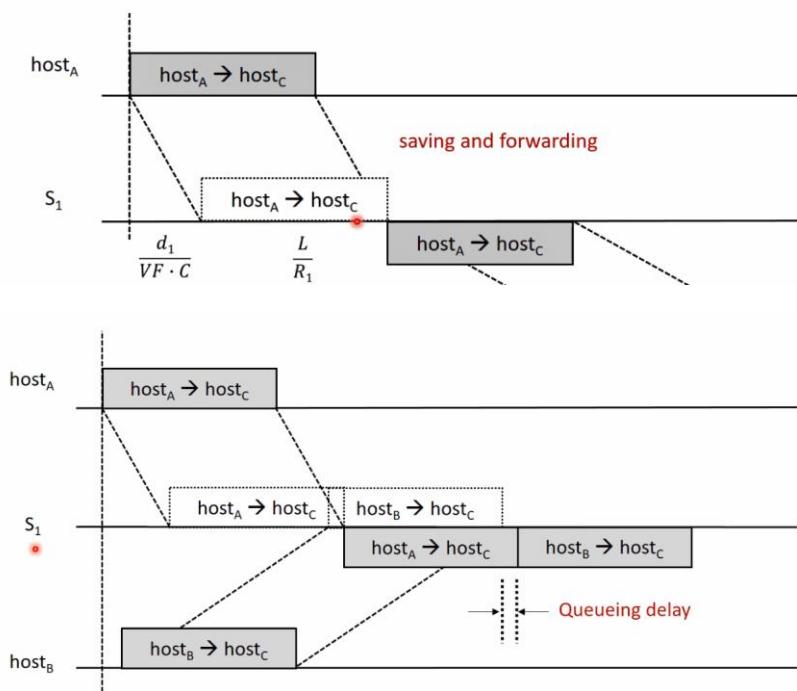
E.  $d_{prop}$ : propagation delay

- i. Output link로 다음 노드까지 옮기는데 걸리는 시간

- ii. Physical link 길이가  $d$  m이고 propagation speed가  $s$  m/sec면  $d_{prop} = \frac{d}{s}$  sec 걸린다

+ transmission delay는 packet을 output link로 옮기는 시간이고 propagation delay는 다음 노드로 옮기는데 걸리는 시간이라 전혀 다른 것이다

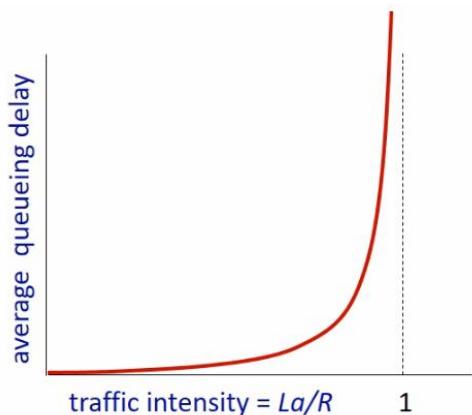
F. Packet 전송의 타임라인



### G. Packet queueing delay

i. Traffic intensity:  $\frac{L \cdot a}{R} = \frac{\text{arrival rate of bits}}{\text{service rate of bits}}$

1. a: 평균 packet arrival rate
2. L: packet 길이
3. R: link bandwidth



Traffic intensity와 average queueing delay가 지수함수 관계인 이유는 packet 도착이 일정하지 않고 randomness를 가지기 때문이다

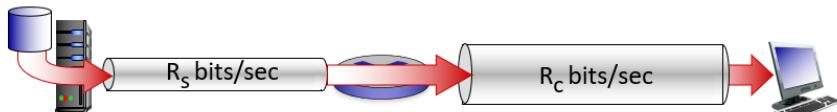
### 3. Throughput

A. Throughput: Client-server 사이에 보내지는 전송 속도이다. (한 hop의 속도가 아니라 end to end 속도다.)

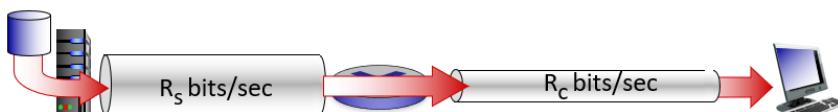
- i. 보통 계산은  $\min(R_{hop})$ 으로 구하면 된다

B. 왜 Throughput이 중요한가?

$R_s < R_c$  What is average end-end throughput?



$R_s > R_c$  What is average end-end throughput?



위의 상황에서는  $R_c$ 의 속도로 전송되지만 밑의 상황은 앞 hop에서 뒤의 처리량보다 빨리 보내서

**packet loss가 생기며, Rc만 있어도 되는데 Rs-Rc만큼 전송 리소스를 낭비하고 있다.**

- C. Bottleneck link: end-end path에 있는 end-end throughput을 제한하는 링크(전송 속도가 낮은 링크라고 봐도 무방할듯)
- 4. Bandwidth, throughput, goodput
  - A. Bandwidth: 링크의 데이터 전송 capability
  - B. Throughput: 링크의 측정된 unit time동안의 데이터 전송량
    - i. Bandwidth은 Throughput의 이상치이기 때문에 항상 Bandwidth $\geq$ Throughput
  - C. Goodput: Application-level의 throughput으로 전송한 데이터 중에서 useful한 데이터만 측정한다
    - i. Protocol overhead, 재전송된 데이터 패킷을 제외한 데이터의 양
    - ii. 좋은 application을 평가하려면 Throughput이 아니라 Goodput으로 평가해야한다
- 5. Bandwidth-delay product
  - A. 특정 시점에 해당 link에 있을 수 있는 데이터의 최대량
  - B. 보내졌지만 ACK는 받지 않은 데이터의 양
  - C. 결과를 알기 위해서 보내야 하는 데이터의 양(=시스템의 반응 속도)
  - D. Bandwidth\*(RTT or propagation delay)
  - E. Bandwidth delay product가 큰 네트워크를 long fat network(=LFN)이라고 한다.
  - F. 라우터의 버퍼나 RDT의 window 등 네트워크의 여러 파라미터를 정하는데 지표가 된다

## Network security

1. 왜 문제가 되는가: 인터넷은 원래 상호 신뢰가 가능한 사용자들 간의 통신이어서 초기 설계에서 이런 문제를 고려하지 않고 만들었기 때문이다.
2. Sniffing(packet interception): 원래 네트워크에서는 broadcast media에서 보낸 packet들을 수신자가 자신일 경우에만 받지만 promiscuous network interface로 모든 packet들을 읽어서 packet을 interception하는 것이다. 예시)Wireshark
3. IP spoofing(fake identity): packet의 header에 client 이름을 다른 것으로 적어서 자신의 정

체를 숨기는 것이다. 보통 DDoS공격에서 공격자의 정체를 숨기려 사용한다.

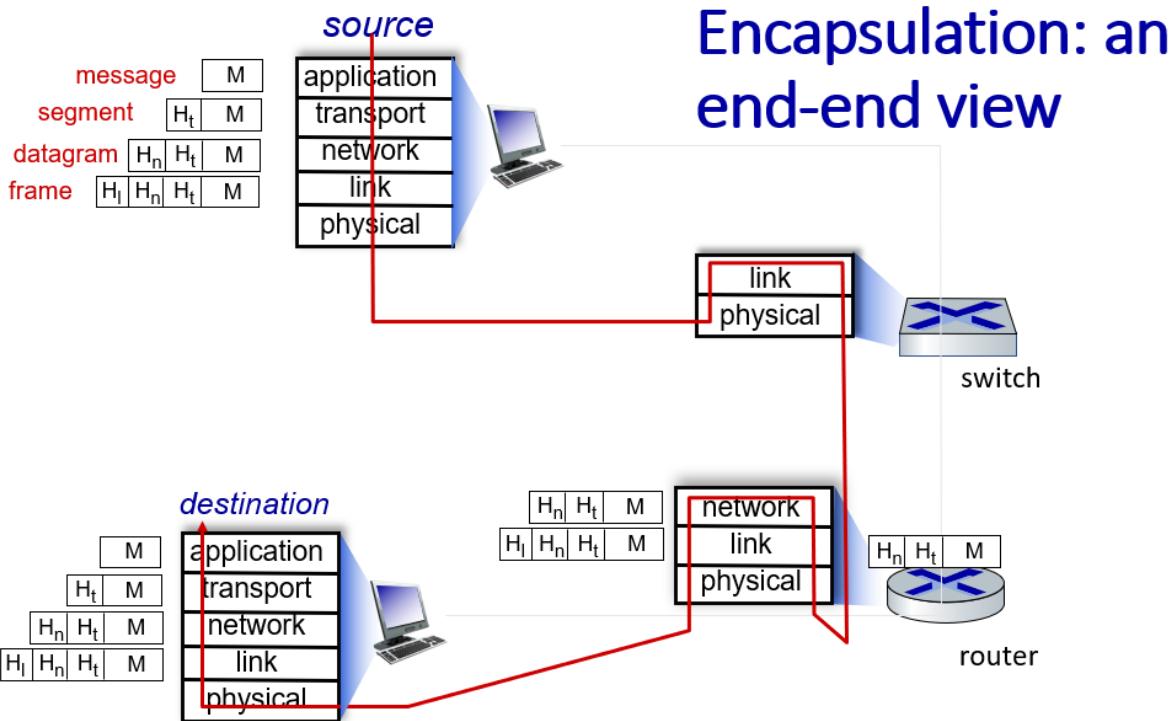
4. Denial of Service(DoS, denial of service): 서버에게 좀비pc를 이용해 traffic을 많이 보내 과부하를 일으켜 서버의 정상적인 작동을 못하게 한다
5. 공격을 막는 법
  - A. Authentication: client가 누구인지 증명시키는 것
  - B. Confidentiality: 암호화하는 것
  - C. Integrity checks: digital signature로 변조를 막는 것
  - D. Access restrictions: 아무나 네트워크를 사용 못하게 하고 VPN(virtual private network)를 통해 암호로 접속할 수 있도록 하는 것
  - E. Firewalls: specialized 'middleboxes'를 access and core network에 두어 패킷들을 통제하는 것

## Protocol layer

1. 어째서 layering할까?
  - A. 네트워크 시스템의 복잡한 구성을 단순하게 해준다
  - B. Modularization을 통해 보수와 업데이트를 쉽게 해준다
    - i. 한 layer의 내용을 고쳐도 다른 layer에는 영향을 주지 않는다
2. Internet protocol stack
  - A. Physical layer: binary 데이터를 signal로 바꾸어 point to point communication을 한다.
  - B. Link layer: Network layer로부터 받은 데이터를 경로 상의 한 노드에서 다른 노드로 이동시킨다
    - i. 송신자와 수신자에 대한 정보를 담아 누가 받아야 하는지 알려준다
    - ii. 같은 링크를 쓰는 노드들이 특정 시간에 다 송신만 한다면 아무도 받을 수 없다. 고로 누가 보내고 누가 받을지 조정해준다
    - iii. Shared Network/Link에 있는 노드 간에 쓰인다
    - iv. Ethernet, WiFi, PPP 프로토콜등이 있다.
    - v. Link layer 주소는 Mac address다

- C. Network layer: 데이터들을 source에서 destination까지 라우팅한다
- i. source에서 destination까지 가는 길을 정하는 계층이다
  - ii. 길을 정하기 위해서는 IP라는 주소가 필요한데 이는 unique한 주소다
    - 1. Link의 주소는 IP 주소처럼 unique하지 않아도 되는데 이는 Link는 shared network 안에서만 쓰이기 때문에 shared network에서만 unique하면 된다.
    - 2. IP주소는 보통 서브넷으로 구분하는데 destination이 있는 IP에 도달하면 이후로는 IP로는 구분할 수 없기 때문에 Link로 데이터를 전송한다.
  - iii. IP, routing 프로토콜이 있다
    - 1. IP 프로토콜은 네트워크의 unique한 주소를 정해준다
    - 2. Routing 프로토콜은 어떻게 source에서 destination으로 갈지 길을 찾아준다
- D. Transport layer: process-process 데이터 전송을 하는 계층
- i. Network 계층으로 dest가 어디 있는지 파악하고 Link 계층으로 dest를 찾아가서 transport layer에서 정해주는 port number을 통해 process를 찾아간다
    - 1. TCP, UDP 프로토콜이 있다
- E. Application layer: 네트워크 어플리케이션들을 서포팅하는 계층
- i. FTP, SMTP, HTTP 등의 프로토콜이 있다
3. OSI 7 layers
- A. Presentation layer: allow applications to interpret meaning of data
- i. Encryption, compression etc
- B. Session layer: synchronization, checkpointing, recovery of data exchange
4. Encapsulation: 각 계층마다 자신이 담당하는 기능과 관련된 정보를 Header에 넣고 상위 계층에서 전달된 정보를 Payload로 한다.
- A. Application layer: message(어플리케이션의 데이터)를 트랜스포트로 전달한다
  - B. Transport layer: message|| segment(port number)를 덧붙여 네트워크로 전달한다
  - C. Network layer: segment+message에 datagram(dest의 IP주소)를 덧붙여 링크로 전달한다
  - D. Link layer: datagram+segment+message에 frame(dest의 link layer주소)를 덧붙여

physical 계층으로 전달한다



## Ch2. Application layer

### 1. Principles of network applications

#### A. Client-Server paradigm

- i. Server와 client가 통신하는 형식
- ii. Server
  1. 항상 켜져 있으며 permanent IP address를 가진 host
- iii. Client
  1. server와 통신하는 host
  2. 항상 켜져있지는 않음
  3. 다양한 IP address를 가짐
  4. client들끼리는 직접적으로 통신하지 않는다

#### B. Peer-Peer architecture

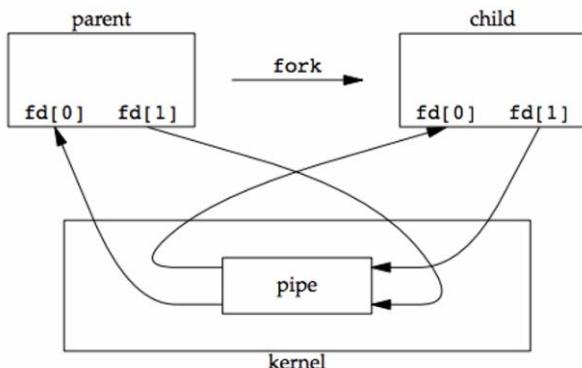
- i. 항상 켜져있는 server가 없다

- ii. 임의의 end system들이 직접 통신한다
- iii. Peer들은 Server가 되기도, Client가 되기도 한다
  - 1. Self-scalability: 새로운 peer들은 service 요구량을 증가시키지만 동시에 service capacity도 늘려준다
- iv. 관리하기 어렵다
  - 1. Selfish peer: 자기 할 일만 마치고 떠나기 때문에 이런 유저가 많아지면 p-p 구조가 돌아가기 어렵다
  - 2. peer들은 항상 켜져 있지는 않으며 IP주소가 바뀔 수도 있다

### C. Processes communicating

- i. Process: host에서 돌아가는 program
- ii. 같은 host에 있는 processes가 통신하면 inter-process communication(IPC)를 사용한다
  - 1. IPC는 OS에서 제공한다

#### IPC (inter-process communication)

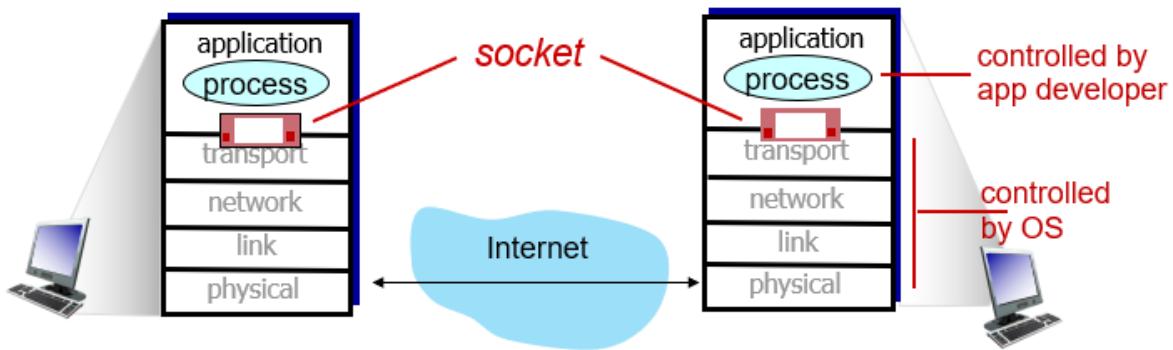


Parent가 fork로 child를 만들고 kernel에서 pipe를 만들고 pipe로 정보를 교환한다  
(Pipe는 대표적 IPC)

- iii. 다른 host의 processes간에 통신하면 'messages'를 교환하며 통신한다
  - 1. Client process: 통신을 시작하는 process
  - 2. Server process: 연결을 기다리는 process
  - 3. P2P architecture의 applications는 client process와 server process를 둘 다 가지고 있다

## D. Socket

- i. Process가 다른 host에 message를 전달할/전달받을 때 쓰는 통로
- ii. Addressing processes
  1. process에서 message를 보내기 위해서는 identifier가 필요하다
  2. 각 host들은 unique 32-bit IP address가 있으며 process들은 특정 port number와 연관되어 있다>>identifier는 IP address, port number를 포함한다



## E. Application-layer protocol

- i. Types of messages exchanged: 0| message가 request인지 response인지
- ii. Message syntax: message에 무슨 field가 있는지, field가 어떻게 생겼는지
- iii. Message semantics: field 속 정보가 가지는 의미
- iv. Rules: 언제 어떻게 message를 보내고 답해야 할지

+ ) protocol의 종류

- Open protocols: RFCs에 정의되어 있으며 모두가 쓸 수 있고, interoperability가 보장된다
- Proprietary protocols: 모두가 쓸 수 있는 protocol이 아니라 형식이 어떤지 알 수 없다

## F. Transport service 중 app에서 필요한 기능

- i. Data integrity
  1. 데이터가 통신과정에서 변조가 일어나지 않았는지 확인하는 기능
- ii. Timing

- 1. 통신과정에서 생기는 딜레이를 줄여주는 기능
- iii. Throughput
  - 1. 통신할 때 데이터의 크기가 최소 어느정도는 되도록 guarantee해주는 기능
- iv. Security
  - 1. 설명은 필요없겠고 encryption등의 기능

+ ) 하지만 이 모든 기능을 다 제공하게 되면 transport layer가 너무 무거워져서 필요한 기능만 제공하게 된다

<b>application</b>	<b>data loss</b>	<b>throughput</b>	<b>time sensitive?</b>
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video: 10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

서비스 종류에 따라 제공하는 기능의 종류가 달라진다

## G. Internet transport protocols services

- i. TCP service
  - 1. Reliable transport: 송/수신에 있어서 신뢰가능한 전송(packet loss가 없으며 보낸 순서대로 packet들이 도착하는 것)을 하게 해준다
  - 2. Flow control: 송신자는 수신자의 수신 속도 이상으로 송신할 수 없다
  - 3. Congestion control: network가 overloaded될 때 전송에 throttle을 건다
  - 4. Connection-oriented: client와 server 사이에 통신하기 위한 setup(handshaking 등)이 필요하다
  - 5. TCP에서 제공하지 않는 것: timing, minimum throughput guarantee, security
- ii. UDP service

1. Unreliable data transfer: TCP처럼 신뢰가능한 전송이 아니다
2. UDP에서 제공하지 않는 것: TCP에서 제공하지 않는 것+reliability, flow control, congestion control, connection setup
- iii. 서비스의 특징에 따라 TCP/UDP중 츠사 선택해야 한다

<b>application</b>	<b>application layer protocol</b>	<b>transport protocol</b>
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

파일 전송 같은 reliable이 필요하면 TCP를, transport에 부하가 적어야하는 프로그램들은 UDP를 사용하기도 한다

- iv. Securing TCP
  1. Transport Layer Security(TLS): application layer에서 라이브러리로 제공
    - A. Encrypted TCP 연결을 제공한다
    - B. Data integrity를 제공한다
    - C. End-point authentication을 제공한다
  2. Web and HTTP
    - A. Webpage's rendering
      - i. Webpage는 object들로 구성되며 이 object들은 Web server에 저장되어 있다.
      - ii. Webpage는 base HTML 파일과 base HTML 파일 안에 referenced된 object 파일로 이루어져 있다
      - iii. Base HTML 파일은 Uniform Resource Locator(URL)을 통해 다른 object 파일들을

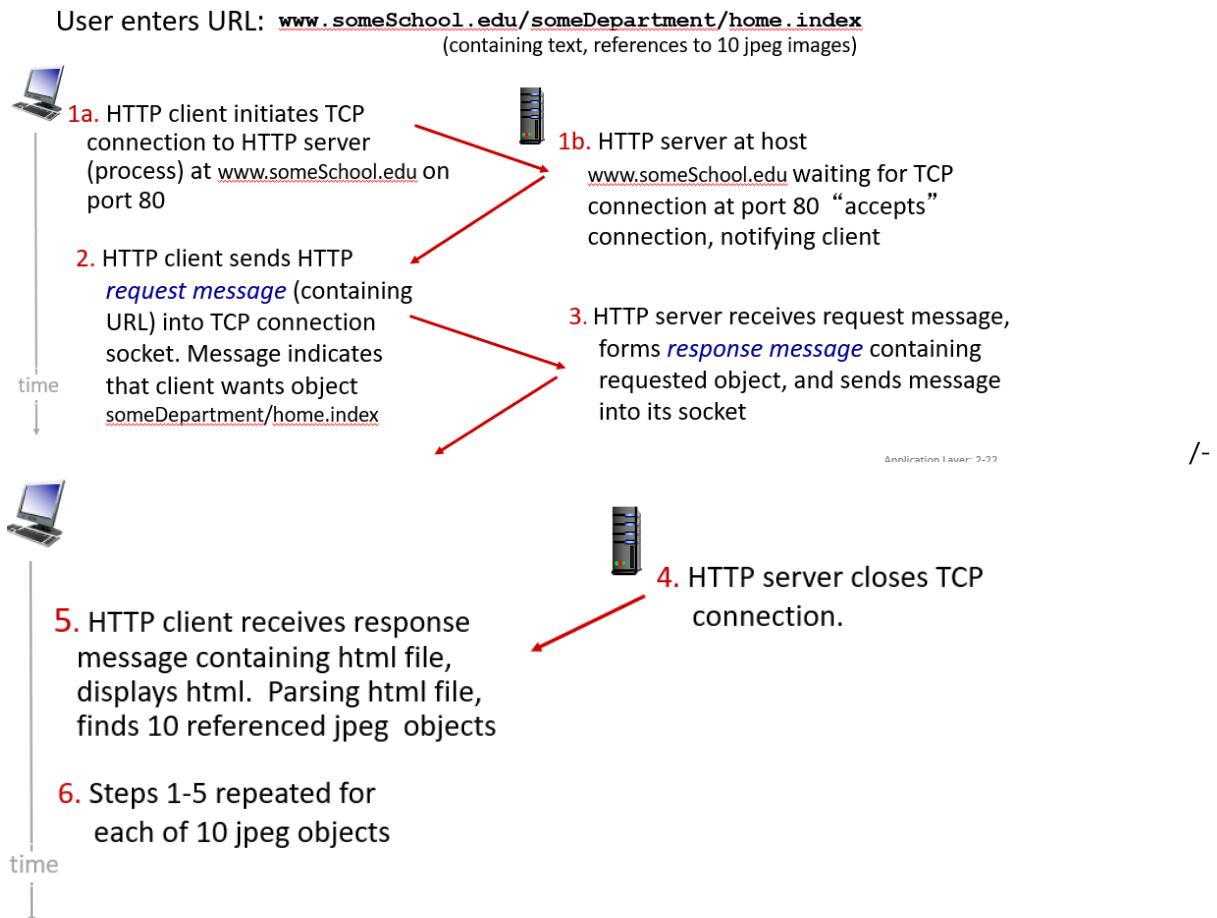
reference한다([www.someschool.edu/some/yes.jpg](http://www.someschool.edu/some/yes.jpg)에서 /some/yes.jpg는 path name이고 그 앞은 host name이다.)

B. Hypertext transfer protocol (HTTP)

- i. Web의 application layer protocol이다
- ii. Client
  1. HTTP protocol을 통해 request/receive하고 display하는 browser
- iii. Server
  1. HTTP protocol을 통해 request에 response하며 object를 보내는 Web server
- iv. HTTP와 TCP
  1. HTTP는 reliable transport가 필요해서 port 80을 통한 TCP connection이 필요하다
  2. HTTP는 stateless하다(과거의 request를 기억하지 않는다)
- v. 두 타입의 HTTP
  1. Non-persistent HTTP: 한 TCP 연결이 하나의 object 전송 후 닫히는 방식
    - A. 여러 object를 보내려면 여러 연결이 필요하다
    - B. 매번 TCP 연결을 해야되서 OS에 부담이 간다
    - C. browser들은 보통 multiple parallel TCP connection으로 object들을 받아온다
  2. Persistent HTTP: TCP 연결 하나를 열었을 때 한 object 전송이 끝나도 연결을 닫지 않고 다음 전송을 할 수 있는 방식
    - A. 하나의 연결로 여러 object를 보낼 수 있다
    - B. TCP 연결은 마지막 전송이 끝난 후 일정 시간동안 아무런 요청이 없으면 닫힌다

+)  
+)HTTP 1.0에서는 디폴트가 Non-persistent였지만 1.1부터는 디폴트가 Persistent다

### + ) Non-persistent HTTP 작동



#### vi. HTTP의 response time

1. Round trip time(RTT): packet의 front propagation과 back propagation을 합친 시간=클라이언트에서 서버에 request했을 때 서버의 반응을 알때까지 걸리는 시간
2. Non-persistent HTTP의 response time
  - A. TCP connection을 위한 1RTT
  - B. Client의 HTTP request가 갔다가 답장으로 오는 첫 packet 도착까지의 시간 1RTT
  - C. Object file 전송 시간
  - D. n개의 파일을 전송하는데 걸리는 시간:  $n \times 2\text{RTT} + \text{all file transmission time}$

3. Persistent HTTP의 response time

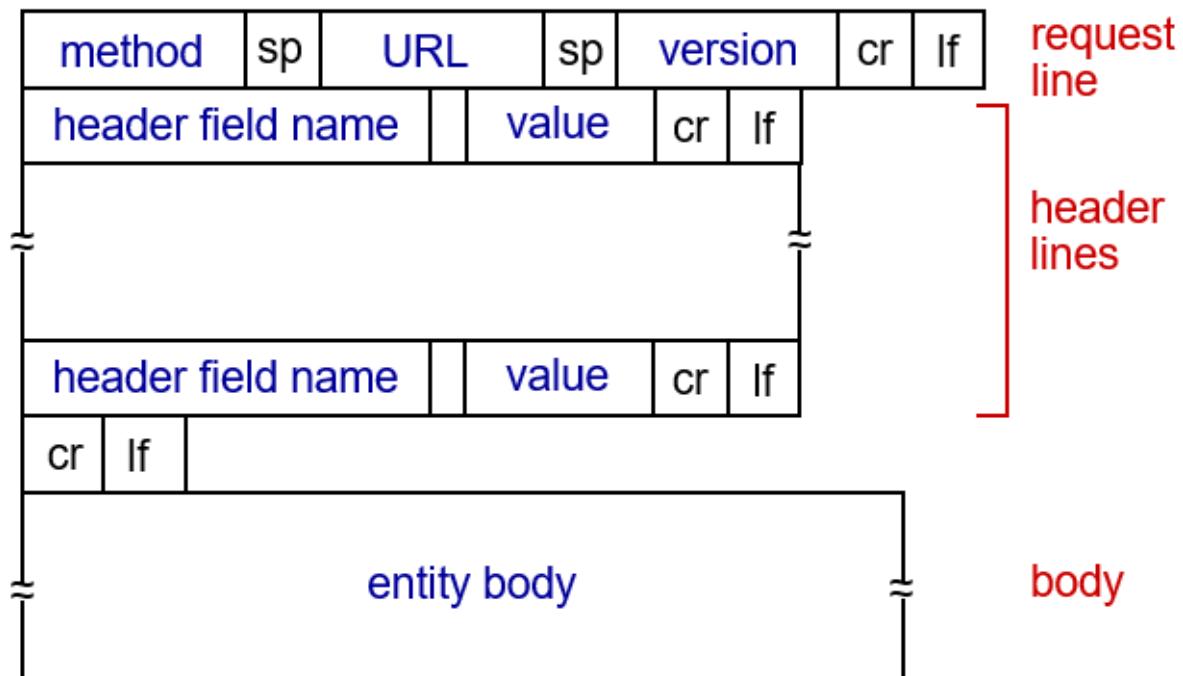
- A. Persistent HTTP는 매번 TCP connection을 만들지 않아도 되기에 Non-persistent의 response time에서 처음을 제외한 TCP connection time을 빼주면 된다
- B. n개의 파일을 전송하는데 걸리는 시간: RTT+n\*RTT+all file transmission time

vii. HTTP request/response message

1. HTTP request message: 아스키(사람이 읽을 수 있는 형태로 전달)

- A. Header
  - i. Request line: HTTP verb + URI (Uniform Resource Identifier) + HTTP version number
  - ii. Optional request headers: 'name:value'의 쌍으로 저장한다
- B. Blank line
- C. Body (Optional)
  - i. Additional information

+ ) HTTP request message 예시



- ASCII (human-readable format)

request line (GET, POST,  
HEAD commands)

header  
lines

carriage return, line feed  
at start of line indicates  
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
```

carriage return character  
line-feed character

\* Check out the online interactive exercises for more  
examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

## 2. HTTP response message

### A. Header

- i. Status line: HTTP version + Status code + Reason Phrase
- ii. Optional response headers: 'name:value'의 쌍으로 저장한다

### B. Blank line

### C. Body (Optional)

- i. Requested Resource

+ ) HTTP response message 예시

```
HTTP/1.1 200 OK
Date: Fri, 04 Sep 2015 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb 2014
ETag: "0-23-4024c3a5"
ContentType: text/html
ContentLength: 35
Connection: KeepAlive
KeepAlive: timeout=15, max = 100

<h1>Welcome to my home page!</h1>
```

## viii. Other request messages

1. POST method: 파일을 생성하는 요청
  - A. GET과 다르게 HTTP POST request message의 body에 요청할 내용을 집어넣는다
2. GET method: 파일을 찾아 보내달라는 요청
  - A. POST와 다르게 요청할 내용을 HTTP의 URL 필드에 쿼리('?')를 이용해서 요청한다
3. HEAD method: GET method인데 header만 요청하는 것이다
  - A. GET은 response에 무언가 body에 받아오지만 HEAD는 header만 받아온다
4. PUT method: 원래 없던 파일을 요청하면 만들고 있는 파일을 요청하면 payload 내용으로 파일을 바꾸는 요청
  - A. POST HTTP request message의 body에 있는 content로 특정 URL에 있는 파일을 완전히 replace한다.

## 5. GET vs POST

	GET	POST
History	Parameters remain in browser history because they are part of the URL. 파라미터들은 URL의 일부분이기 때문에 브라우저 히스토리에 남는다.	Parameters are not saved in browser history. 파라미터들이 브라우저 히스토리에 저장되지 않는다.
Bookmarked	Can be bookmarked. 요청 파라미터들이 URL로 인코딩되므로 즐겨찾기가 가능하다.	Can not be bookmarked. 요청 파라미터들이 request body에 포함되고 request URL에 나타나지 않으므로 즐겨찾기가 불가능하다.
button/re-submit behaviour	GET requests are re-executed but may not be re-submitted to server if the HTML is stored in the browser cache. GET 요청이 다시 실행되더라도 브라우저 캐시에 HTML이 저장되어있다면 서버에 다시 submit되지 않는다.	The browser usually alerts the user that data will need to be re-submitted. 브라우저가 보통 사용자에게 데이터가 다시 submit되어야 한다고 alert을 준다.

<b>Encoding type(enctype attribute)</b>	application/x-www-form-urlencoded	multipart/form-data or application/x-www-form-urlencoded Use multipart encoding for binary data.
<b>Parameters</b>	can send but the parameter data is limited to what we can stuff into the request line (URL). Safest to use less than 2K of parameters, some servers handle up to 64K 전송 가능하지만 URL에 넣을 수 있는 파라미터 데이터가 제한된다. 2K이하로 사용하는 것이 안전하며 몇몇 서버들은 64K까지 다룬다.	Can send parameters, including uploading files, to the server. 서버에 파일 업로드하는 것을 포함하여 파라미터를 전송할 수 있다.
<b>Hacked</b>	Easier to hack for <a href="#">script kiddies</a> script kiddies에 의해 해킹되기 쉽다.	More difficult to hack GET에 비해 좀 더 해킹하기 어렵다.
<b>Restrictions on form data type</b>	Yes, only ASCII characters allowed. 오직 ASCII characters만 허용된다.	No restrictions. Binary data is also allowed. 제한이 없다. binary data도 허용된다.
<b>Security</b>	GET is less secure compared to POST because data sent is part of the URL. So it's saved in browser history and server logs in plaintext. GET은 POST에 비해 보안에 약하다. 그 이유는 데이터가 URL의 일부로 전송되고 그 때문에 브라우저 히스토리에 저장되며 서버가 플레인 텍스트로 로그를 남기기 때문이다.	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs. POST는 GET에 비해 보안에 조금 더 안전하다. 그 이유는 파라미터들이 브라우저 히스토리나 서버 로그에 저장되지 않기 때문이다.
<b>Restrictions on form data length</b>	Yes, since form data is in the URL and URL length is restricted. A safe URL length limit is often 2048 characters but varies by browser and web server. form data가 URL에 포함되고 URL 길이가 제한되기 때문에 form data의 길이도 제한된다. 안전한 URL 길이는 2048 characters이나 브라우저나 웹 서버에 따라 달라진다.	No restrictions 제한이 없다.
<b>Usability</b>	GET method should not be used when sending passwords or other sensitive information. GET 메소드는 비밀번호와 같은 민감한 정보들을 전송하는데 사용해선 안된다.	POST method used when sending passwords or other sensitive information. POST 메소드는 비밀번호와 같은 민감한 정보를 전송하는데 사용된다.
<b>Visibility</b>	GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send. GET 메소드는 모두에게 보여진다. (브라우저의 주소창에 그대로 보여지고 그에 따라 전송가능한 정보의 양도 제한된다.)	POST method variables are not displayed in the URL. POST 메소드는 URL에 보여지지 않는다.
<b>Cached</b>	Can be cached GET은 idempotent하기 때문에 캐시가 된다. (같은 요청을 여러 번 보내도 항상 같은 응답이 온다.)	Not cached POST는 idempotent하지 않기 때문에 캐시가 되지 않는다. (같은 요청을 여러 번 보내도 다른 응답이 올 수 있다.)

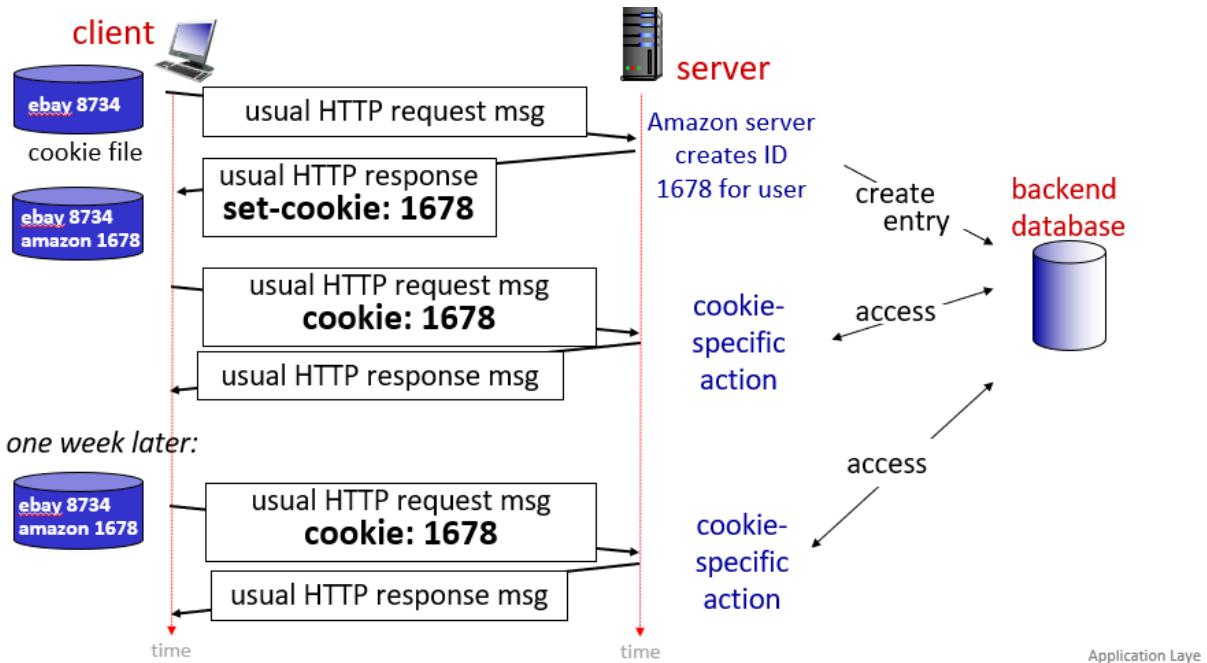
## 6. HTTP response status code

- A. (200, OK): request successes, requested object later in this message
- B. (301, Moved Permanently): requested object moved, new location specified later in this message (in Location: field)
- C. (400, Bad Request): request msg not understood by server
- D. (404, Not Found): requested document not found on this server
- E. (505, HTTP Version Not Supported)

## 7. User/server의 state를 유지하기 위한 방법: cookies

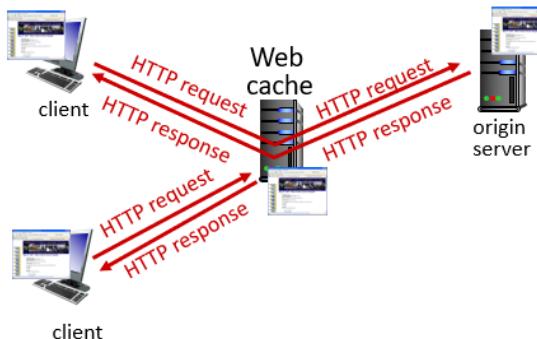
- A. server측 구현을 간단히 하기 위해 HTTP는 이전 state를 보관하지 않도록 만들어졌다(stateless protocol)>>하지만 점점 더 state가 필요한 일들이 늘었고 state를 저장하기 위한 방법으로 cookie를 선택하게 되었다
- B. 하지만 착각하지 않아야 하는게 HTTP는 절대로 stateful protocol이 아니다. Stateless protocol이지만 header에 cookie를 담아 application에서 처리하도록 하여 stateful처럼 보이게 하는 것뿐이다. (참고자료: <https://stackoverflow.com/questions/5836881/stateless-protocol-and-stateful-protocol>)
- C. 그래서 그 cookie가 뭔데: 웹 사이트/브라우저는 transaction들 사이에 state를 유지하기 위해 cookie를 사용한다
- D. Cookie의 4요소
  - i. HTTP response message의 cookie header line (아마 여기서 말하는 것은 첫 통신의 첫 message다)
  - ii. 다음 HTTP request message의 cookie header line
  - iii. User의 host에 있고 user의 browser에 관리되는 cookie file
  - iv. Web site의 백엔드 데이터 베이스
- E. Cookie로 HTTP 통신하는 법
  - i. 처음 client에서 server에 HTTP request msg를 보낼 때 server에서는 cookie를 만들고 이를 client와 backend DB에 보낸다
  - ii. 이후부터는 client가 접속할 때마다 header에 cookie를 담아 보내고 server는 backend DB의 값과 비교하여 client가 누구인지 state를

가져온다



## 8. Web caches (proxy server)

### A. Origin server에의 트래픽을 줄이기 위해 중간 물류 참고(?)를 두는 것



### B. 어떻게 돌아가는가?

#### i. Browser는 모든 HTTP request를 web cache로 보낸다

1. 'If' object in cache: cache는 client에게 object를 리턴한다
2. 'else': cache는 origin server에 object를 request하고 받은 후에 client에 object를 리턴한다

### C. 특징

#### i. Web cache는 client/server 두 가지 역할을 둘 다 할 수 있다

1. Client에 object를 리턴할 때는 server

2. Origin server에 object를 요청할 때는 client
  - ii. 만약 cache에 파일이 있다고 해도 out of date이면 안되기 때문에 origin server는 cache에 object의 allowable caching을 response header로 알려준다

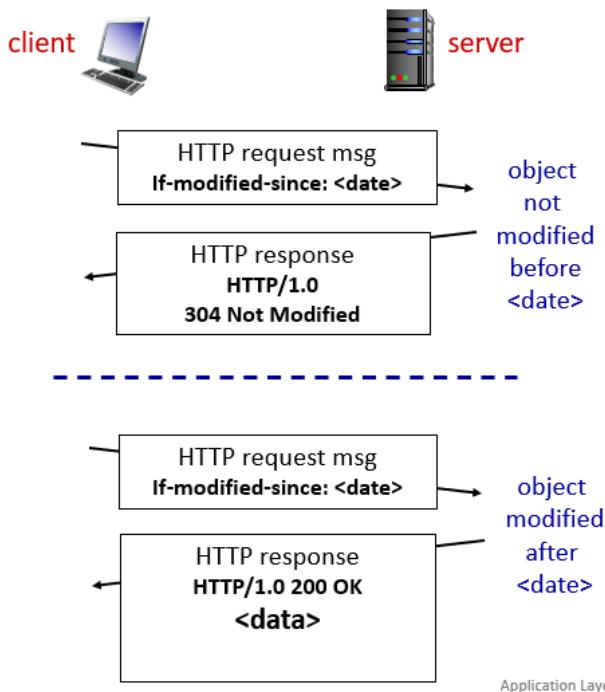
#### D. 장점

- i. Cache는 origin server보다 가깝기 때문에 response time을 줄인다
- ii. 어떤 기간으로의 access link에의 traffic을 줄인다
- iii. 비슷한 요청이 여러 곳에서 오는 경우 그 요청 파일을 저장해놓을 수 있다.

#### E. Conditional GET

- i. 쓰게 된 배경: Web cache는 proxy server를 통해서만 제공되는 것이 아니고 local cache로도 제공될 수 있다. (대충 한 번 방문했을 때 object를 local cache에 저장한다고 보면 될듯) 하지만 local cache에 있는 object가 up to date한 버전인지는 실제 server에게 물어봐야 하기 때문에 쓰게 되었다
- ii. 장점: 만약 up to date하다면 있는 object를 다시 가져오지 않아도 된다
- iii. 어떻게 작동하는가?

1. Client: cached copy의 날짜를 HTTP request에 보낸다
  - A. If-modified-since:<date>
2. Server: 만약 cached copy가 up to date하다면 아무런 object도 response에 담지 않는다
  - A. HTTP/1.0 304 Not Modified



## 9. HTTP2와 HTTP/3

### A. HTTP/2

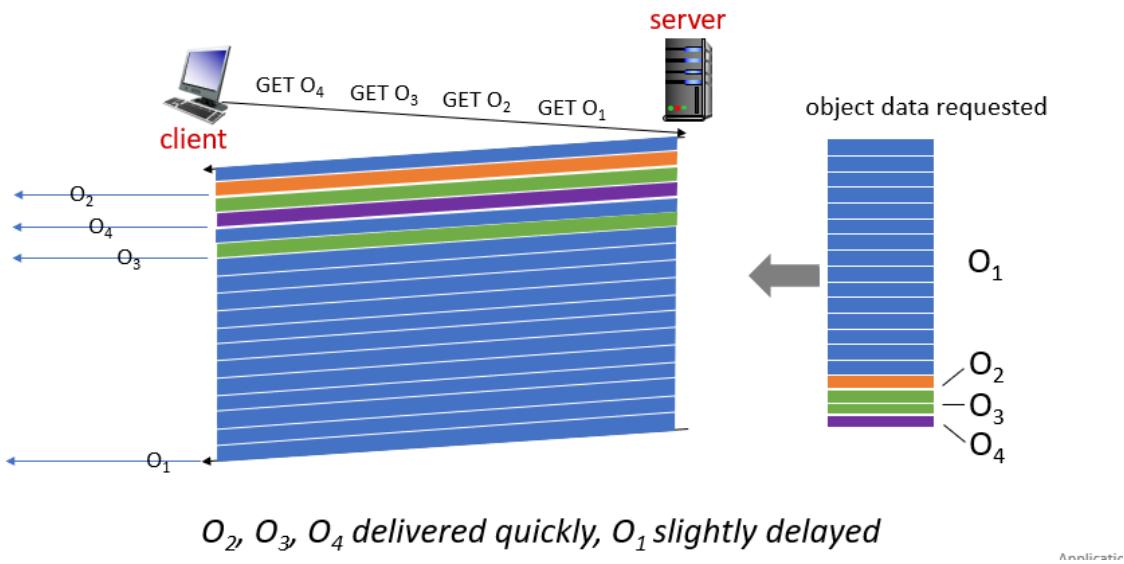
- 왜 나오게 되었는가: HTTP1.1에서 multiple, pipelined GETs를 single TCP connection으로 처리한다고 생각했을 때, first-come-first-served scheduling(FCFS)순서로 처리되기 때문에 만약 큰 object 1개+작은 object 3개를 처리한다면 작은 object들은 큰 object가 처리되는 동안 head-of-line(HOL) blocking을 겪어야 한다.
- 특징
  - Method, status code, most header field들은 1.1에서 안바뀜
  - Transmission order가 FCFS가 아니라 client-specified object priority에 따른다
  - client에서 어떤 object를 request했을 때 더 필요할 거라 생각되는 object가 있다면 unrequested하더라도 같이 push한다
  - HOL blocking을 완화하기 위해 object와 schedule을 frame으로 쪼갠다

HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



HTTP1.1에서 HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



HTTP2.0에서 HOL blocking 완화된 것

#### B. HTTP/3

- i. Quick UDP Internet Connections(QUIC)을 기반으로 만들어진 차세대 HTTP
- ii. UDP기반이라 처음 접속이 빠르며 security도 추가되었다
- iii. 구현은 UDP 위에 layer을 추가하여 거기에서 기능을 추가함

### 3. E-mail, SMTP, IMA++P

#### A. E-mail

i. E-mail의 3요소: user agents, mail servers, simple mail transfer protocol(SMTP)

##### 1. User agent (e.g.Outlook)

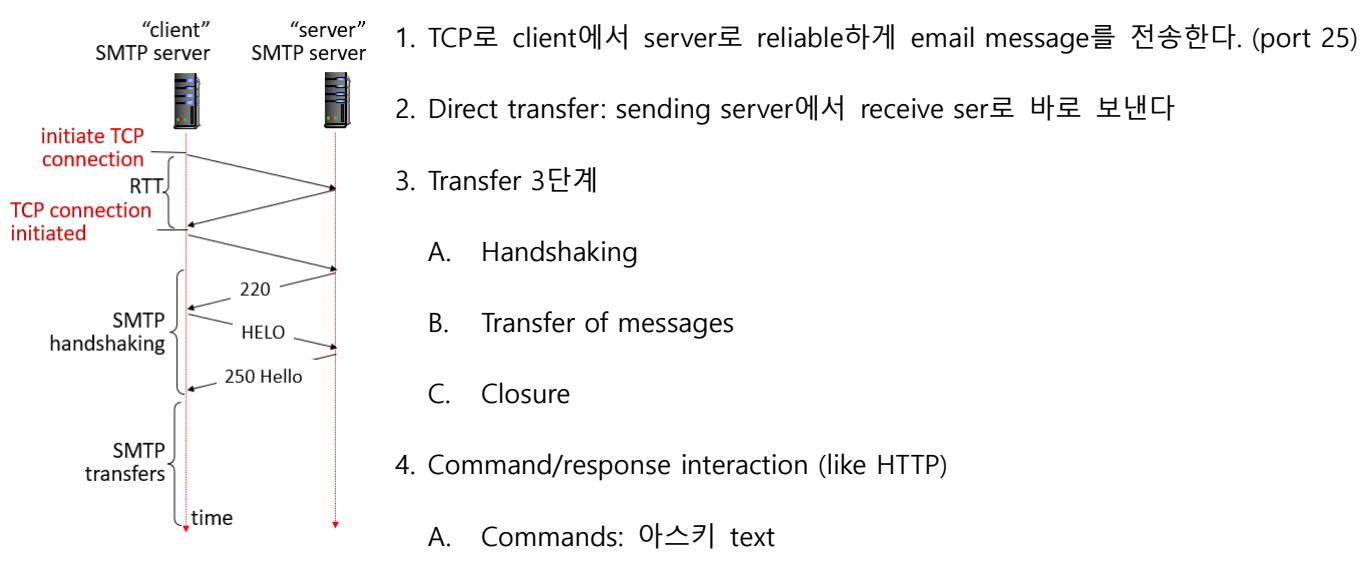
- A. E-mail 사용자가 사용하는 client이며 mail reader라고도 한다
- B. Mail message를 작성, 수정, 읽는데 쓴다
- C. 보내거나 받은 e-mail들은 server에 저장된다(대충 e-mail 작성은 UA에서, 저장은 mail server에 한다는 말인듯)

##### 2. Mail servers

- A. Mailbox: 사용자에게 온 mail을 가지고 있는 저장소
- B. Message queue: 사용자가 보내려는 mail을 가지고 있는 저장소

3. SMTP protocol: user agent>mail server, mail server>mail server 통신을 할 때 쓰이는 통신 규약

#### ii. SMTP RFC



#### iii. E-mail 보내는 시나리오

1. 송신자가 user agent로 receiver@xxx.xxx로 e-mail message를 보낸다

2. 송신자의 user agent는 message를 송신자의 mail server로 보내고 이 message는 message queue로 간다
3. Client side의 SMTP가 수신자의 mail server와 TCP connection을 만든다
4. SMTP client가 송신자의 message를 TCP connection을 통해 보낸다
5. 수신자의 mail server가 수신자의 mailbox에 보관한다
6. 수신자가 수신자의 user agent를 써서 message를 가져온다

```

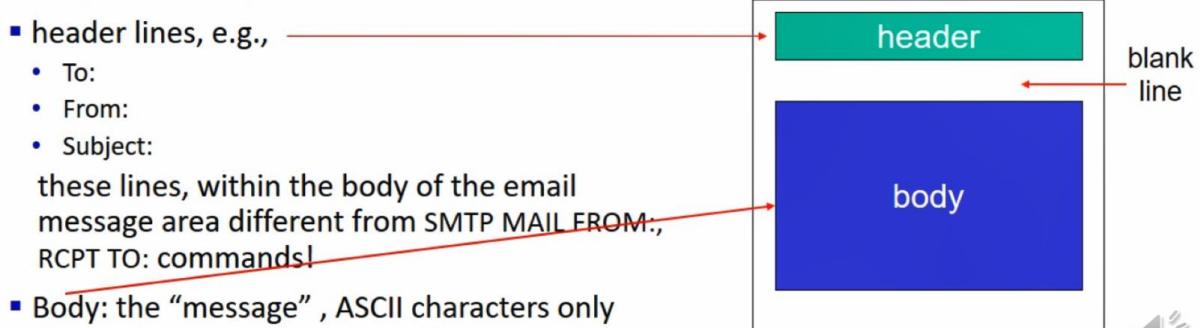
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection

```

#### SMTP interaction 예시

- iv. SMTP vs HTTP
  1. Protocol의 성격
    - A. HTTP: client가 server에게서 가져오는 pull
    - B. SMTP: client가 server에게 보내는 push
  2. HTTP, SMTP 둘 다 아스키를 이용한 command/response interaction과 status code를 가진다
  3. 여러 object를 보낼 때
    - A. HTTP: 각각의 object는 캡슐화되어 각자의 response message로 보낸다
    - B. SMTP: 모든 object는 하나의 response message로 보낸다

4. SMTP는 persistent connection을 쓴다
  5. Message 구성
    - A. SMTP: message(header & body)가 무조건 아스키여야 한다
    - B. HTTP: message(header & body)에 binary data가 들어갈 수도 있다
  6. SMTP server는 CRLF,CRLF로 message종료를 결정한다
- v. Mail message format
1. SMTP: e-mail message 교환 protocol in RFC 531(like HTTP)
  2. RFC 922는 e-mail message 자체의 syntax를 정의한다(like HTML)



vi. Mail access protocol

1. SMTP: SMTP는 수신자의 mail server까지만 보내기 때문에 받아오는건 다른 protocol로 해야 한다
2. IMAP(internet mail access protocol): mail server에 있는 데이터와 동기화
3. POP3(Post Office Protocol 3): mail server에 있는 데이터를 로컬로 다운로드 함(다운로드 후에 mail server에 있는 데이터를 삭제할지 말지 결정할 수 있다)
4. HTTP: web base e-mail일 경우 HTTP로 mail server의 mailbox에 접근할 수 있다

+ ) SMTP server들은 자기가 관리하는 client가 아니면 다른 곳으로 전송하지 않는다

#### 4. DNS (Domain Name System)

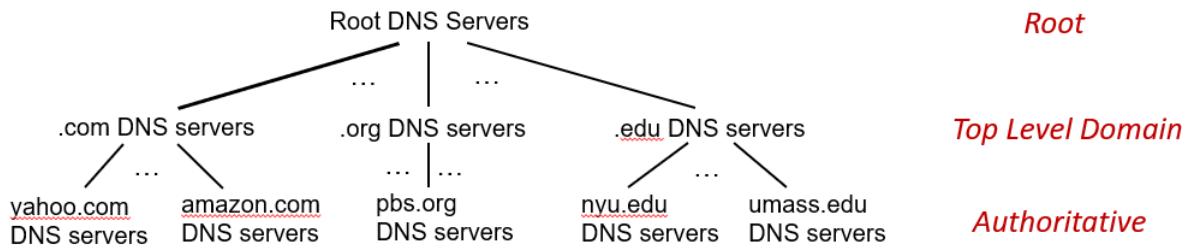
##### A. DNS는 왜 생겼고 왜 이렇게 생겼는가?

- i. DNS가 필요한 이유: 우리가 어떤 서비스를 이용하려면 서비스를 제공하는 server에 request해야한다.>>request를 하려면 server의 IP address를 알아야 하지만 32bit의 숫자인 IP address는 사람이 알기 어려워서 보통 도메인 네임이라는 형식으로 request한다.>>도메인 네임과 IP address를 mapping해주기 위해서 DNS서버가 필요하다.
- ii. DNS에 대한 개괄
  1. Distributed database: 한 서버에 맡기기에는 너무 큰 traffic이 생기기 때문에 많은 name server들의 히에라르키로 사용된다.
  2. Application-layer protocol: DNS 서버와 host들이 address/name translation을 위해 통신한다.
    - A. 'Complexity at network's edge'라는 설계 철학 때문에 core internet function임에도 application-layer protocol로 설계했다

##### B. DNS service

- i. Hostname/IP address translation: Hostname에 해당하는 IP address를 알려준다
- ii. Host aliasing: alias name을 canonical name으로 mapping해준다
  1. Canonical name(정식 명칭)은 alias name(별칭)보다 매우 복잡한 경우가 많아서 알기 쉬운 별칭을 쓸 수 있도록 해주는 것
  2. e.g. relay1.west-coast.enterprise.com ⇔ enterprise.com
- iii. mail server aliasing: host aliasing의 메일 서버 버전
- iv. load distribution: replicated web servers 같은 중복 서버의 traffic을 분산한다
  1. replicated web servers: 인기있는 사이트들은 여러 서버가 하나의 호스트 네임(canonical name)을 가지고 있다.
  2. DNS는 이 서버들의 IP 집합을 가지고 있으며 이 집합중에 traffic이 적은 서버 ip로 답을 하며 load를 한쪽에만 몰리지 않게 한다

### C. DNS structure, distributed, hierarchical database



- i. Root DNS server: DNS structure의 최상위 서버이며 local DNS에게 TLD server을 알려준다
  - 1. Root DNS server들끼리는 항상 동기화하고 있어야 한다
- ii. Top level domain(TLD) server: TLD에서는 .net, .com등의 상위 레벨 도메인과 .kr, .jp등의 모든 국가의 상위 레벨 도메인에 대한 서버가 있으며 이들은 authoritative DNS server을 알려준다.
- iii. Authoritative DNS server: 실제로 Host가 등록된 서버

### D. Local DNS: DNS hierarchy에는 strictly하게 속해 있지 않지만 DNS structure에서 중심이 된다

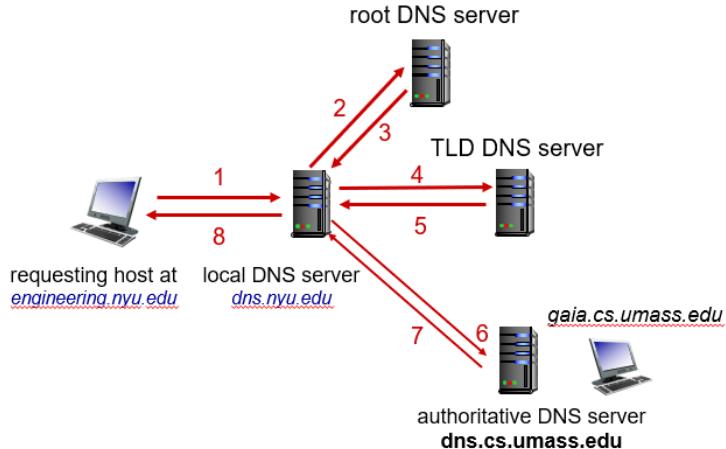
- i. Host가 DNS query를 할 때, local DNS가 받아서 DNS hierarchy에 resolution을 위해 query를 forwarding하거나 local cache에서 response를 찾아준다

### E. Client가 IP주소를 원할 때:

- i. Client가 root DNS server에 TLD DNS server를 찾는 query를 한다
- ii. Client가 TLD DNS server에 authoritative DNS server를 찾는 query를 한다
- iii. Client가 authoritative DNS server에 IP address를 찾는 query를 한다

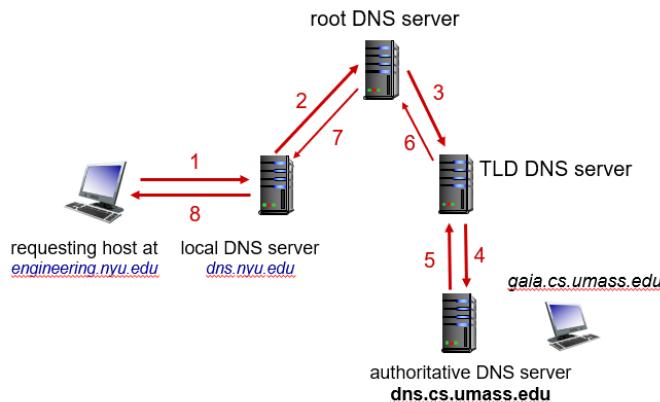
## F. DNS name resolution

### i. Iterated query



### ii. Recursive query

1. 단점: root DNS  
에 많은 load가  
있다



## G. Caching DNS information

- i. DNS response를 받았을 때 그 mapping을 caching하고 같은 DNS query를 할 경우 즉시 cache했던 mapping을 리턴하는 것
  1. Caching은 response time을 줄인다
  2. Cache는 일정 시간(Time to leave, TTL) 후에 삭제한다
  3. 보통 TLD 서버가 local DNS에 cache되어 있다
- ii. Cached information이 out of date일 수는 있지만 보통 아니므로 넘어간다

## H. DNS record: database는 resource record 형태로 정보를 저장하기에 DNS의 RR을 DNS record라고 한다

- i. RR format: (name, value, type, TTL)
  1. Type=A: (name, value) = (hostname, IP address)
  2. Type=NS: (name, value) = (domain, domain의 authoritative DNS 서버의

hostname)

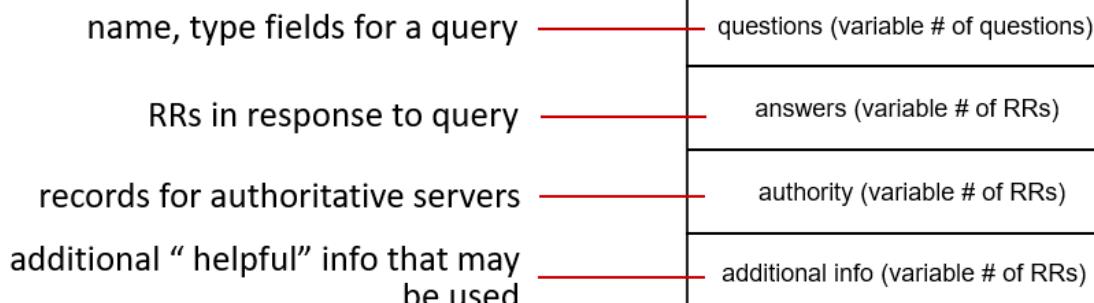
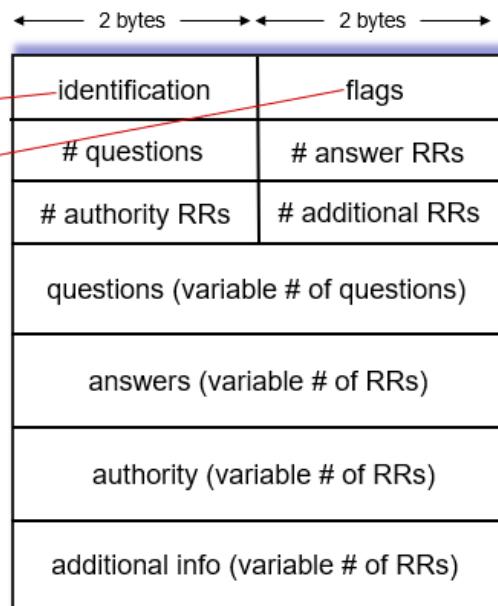
3. Type=CNAME: (name, value) = (alias name, canonical name)
4. Type=MX: (name, value) = (name, name과 관련된 SMTP mail server 이름)

#### I. DNS protocol message

- i. DNS message는 query와 reply message가 같은 format이다

message header:

- **identification:** 16 bit # for query, reply to query uses same #
- **flags:**
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

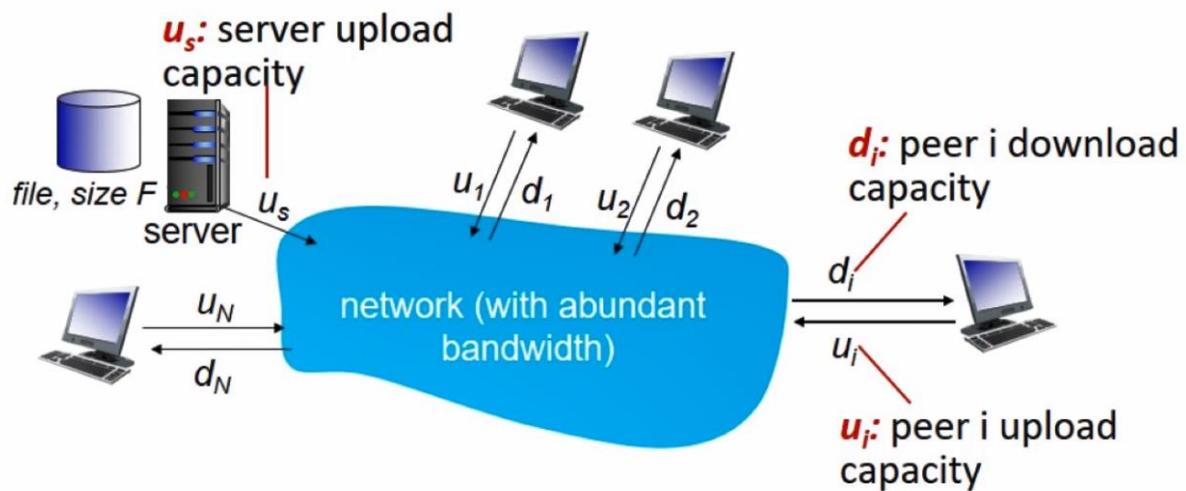


자세한 사항은 <https://programming119.tistory.com/159>로

## P2P (peer to peer) architecture

1. 항상 켜져있는 서버가 없이 임의의 end systems(peer)들이 '직접' 연결되어 있다
2. peer들은 서로 서비스를 request/provide 한다.
  - A. Self-scalability: 새로운 peer는 새로운 서비스 capacity이자 demand이다. >>> peer들에 따라 자기 확장을 할 수 있는 자기 확장성을 가졌다
3. peer들은 간헐적으로 IP주소를 바꾸고 서로 간헐적으로 연결한다 >> management가 복잡하다

## P2P vs C-S (client-server)



P2P와 C-S를 비교하기 위한 상황. ( $F$ 의 파일크기를 가진 파일을  $N$ 개의 클라이언트에게 나누어 주려고 한다)

1. C-S
  - A. 서버는  $NF$  bit를  $u_s$ 의 전송률로 업로드 해야 한다.
  - B. 클라이언트들은  $d_i$ 의 전송률로 다운로드하며 가장 작은 전송률은  $d_{min}$ 이다.

*time to distribute  $F$  to  $N$  clients using client-server approach*

$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in  $N$

2. P2P

- A. 서버는 C-S와 달리 한 개의 파일만 업로드 하면 돼서 F bit를 Us로 업로드 한다.
- B. 클라이언트의 다운로드는 최대 F/dmin이다.
- C. 서버와 클라이언트가 보내야하는 크기는 NF bit이며 업로드 속도는  $u_s + \sum u_i$ 이다.

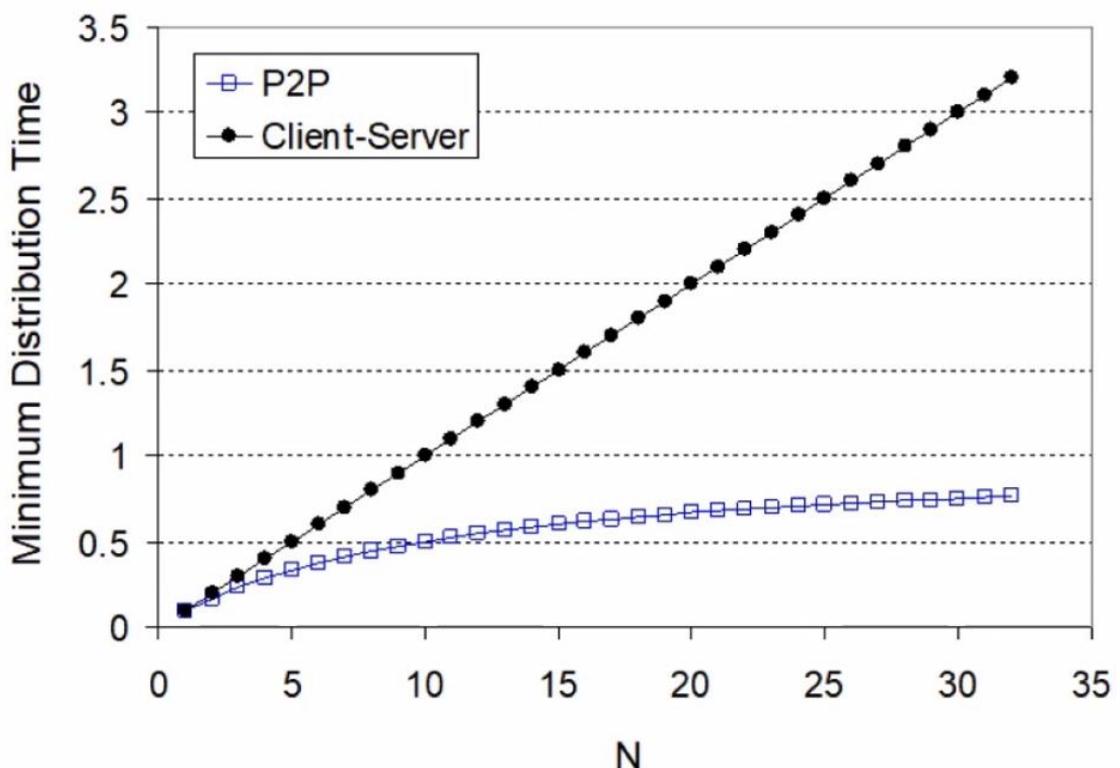
*time to distribute  $F$  to  $N$  clients using P2P approach*

$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

increases linearly in  $N$  ...  
... but so does this, as each peer brings service capacity

### 3. 결과

client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{min} \geq u_s$

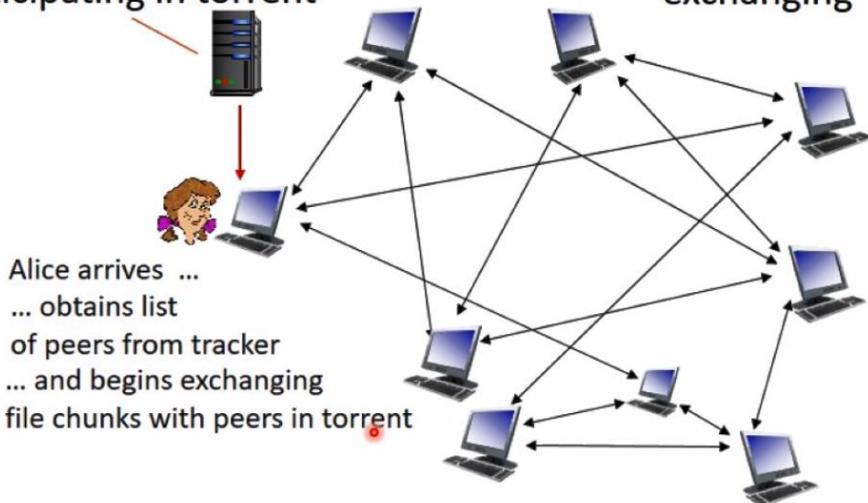


P2P가 C-S에 비해 월등히 빠르다

예시1. BitTorrent

**tracker:** tracks peers participating in torrent

**torrent:** group of peers exchanging chunks of a file

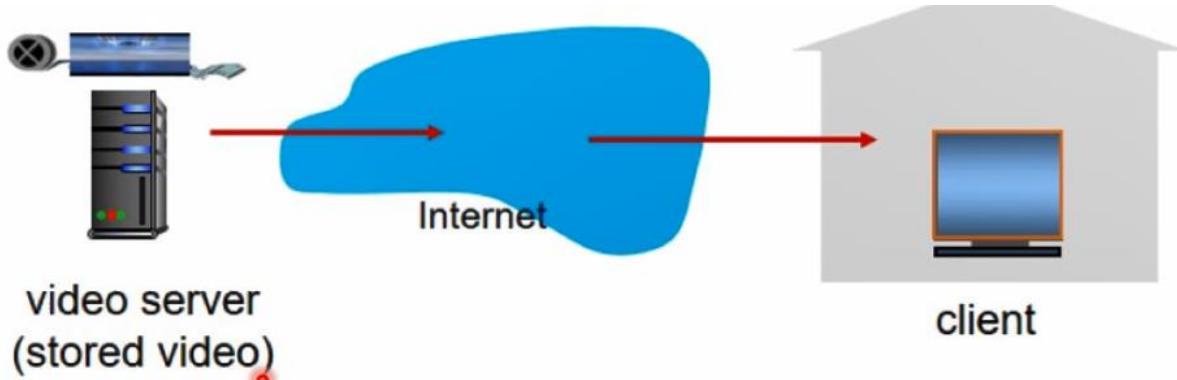


1. Tracker: 현재 torrent에 접속한 peer들을 파악하는 서버
  2. 파일은 256Kb의 chunk들로 나누어 전송한다
  3. Peer은 torrent에 접속하면 tracker에서 준 리스트에 따라 peer들의 'subset'인 neighbors에 접속한다
  4. Churn: 들어오거나 나가는 peer
  5. Requesting chunk
    - A. peer들은 항상 서로 다른 neighbor을 가지고 있다
    - B. peer들은 희소성이 강한 chunk부터 request한다
  6. sending chunk: tit-for tat
    - A. peer가 랜덤으로 four peers를 선택한다
    - B. 전송률이 높은 사람에게 더 많이 보낸다
    - C. 10secs마다 re-evaluate한다
    - D. 30sec마다 four peers를 랜덤으로 재선택한다 (몇 peers에 한정되지 않기 위해)
- 
1. 비디오 스트리밍과 CDN (Contents distribution network)
    - A. 멀티미디어-비디오
      - i. 일정한 속도로 재생되는 픽셀의 배열로 된 digital image의 sequence

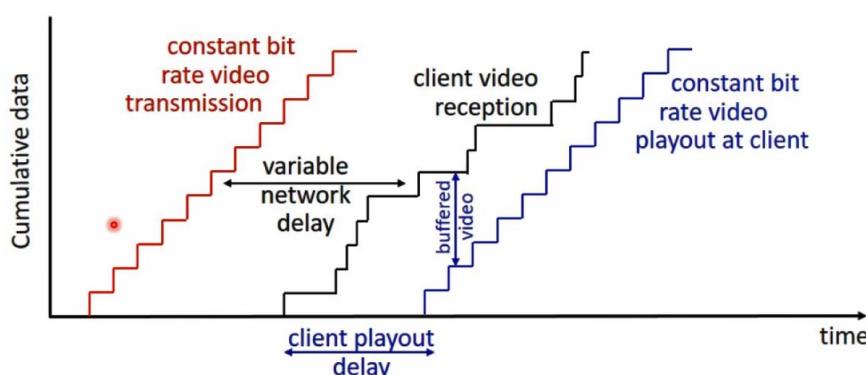
- ii. 이미지 내에서/이미지 간의 데이터 중복성을 사용하여 인코딩할 bit를 줄인다
  - 1. Spatial: 이미지 내에서 줄인다(e.g. 같은 색의 비트가 N개면 'N개'의 '특정한 색'이 있다고 2개의 값을 보낸다)
  - 2. Temporal: 이미지 간의 유사성으로 줄인다(e.g. 다음 이미지와 현재 이미지 사이의 차이나는 비트만 보낸다.)
- iii. 비디오 인코딩 방식
  - 1. CBR (constant bit rate): bit rate가 일정한 방식, 구현하기 편하지만 자원 낭비가 생길 수 있다.
  - 2. VBR (variable bit rate): bit rate가 spatial, temporal 차이량에 따라 달라진다. 구현하기 복잡하지만 자원 낭비가 없어서 CBR보다 결과물이 좋다.

## B. 비디오 스트리밍

### i. 비디오 스트리밍의 문제



- 1. 클라이언트마다, 시간마다 congestion(혼잡) level이 다르다
  - 2. congestion으로 인한 packet loss/delay가 delay playout, 즉 poor video quality를 만든다
- ii. 버퍼링은 무엇이며 어떻게 생기는가?



1. Continuous playout constraint: 영상 플레이가 시작되면, playback은 원래 타이밍과 같아야 한다.
2. Network delay(jitter)가 길어지면 playout requirement를 맞추기 위해 client-side buffer이 나온다>> poor video streaming

iii. DASH: Dynamic, Adaptive Streaming over HTTP

1. Server

- A. 비디오 파일을 여러 chunk들로 나눈다
- B. 각각의 chunk들은 다른 rate로 인코딩 된다
- C. Manifest file: 다른 chunk들의 URLs를 제공한다

2. Client

- A. 주기적으로 C-S bandwidth를 측정한다
- B. Manifest를 분석하여 하나의 chunk에 대해 request를 보낸다
  - i. 그 시점에 최적인 지점에 있는 서버에서 가능한 최대 coding rate를 고른다

3. Intelligence at client

- A. When to request chunk
- B. What encoding rate to request
- C. Where to request chunk

iv. Streaming video=encoding + DASH + playout buffering

C. CDNs (Content distribution networks)

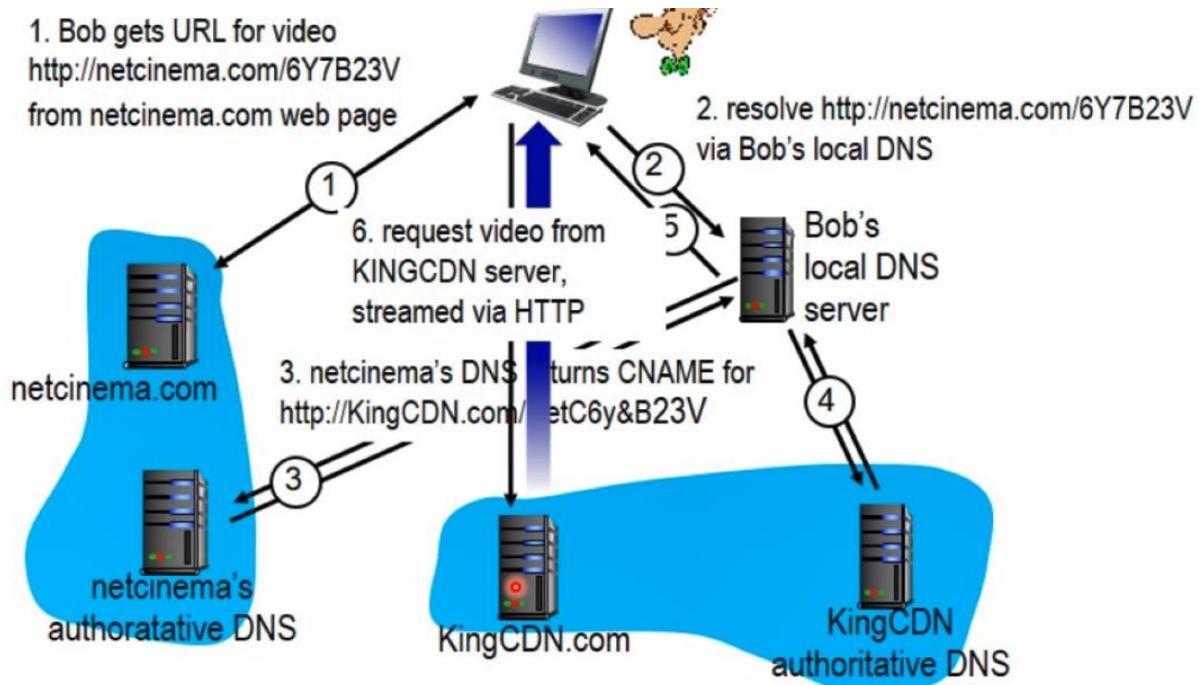
i. 어떻게 해야 여러 사람들이 한번에 스트리밍 서비스를 이용할 수 있을까?

1. Single, large 'mega-server'
  - A. 한 서버에 가는 트래픽이 너무 커지고 그 서버가 터지면 아무것도 못함. 또한 위치가 멀리 있으면 긴 거리가 문제가 된다
2. 여러 지역에 있는 서버에 video들의 여러 카피를 저장해놓기
  - A. Enter deep: 많은 수의 서버 클러스터(여러 대의 독립된 컴퓨터를 하나의 시스템으로 운영하는 것)를 세계 곳곳에 설치한다.

B. Bring home: 적은 수의 큰 서버 클러스터들을 좀 더 가까운 지점에 설치하여 운용하는 방식

### 3. CDNs

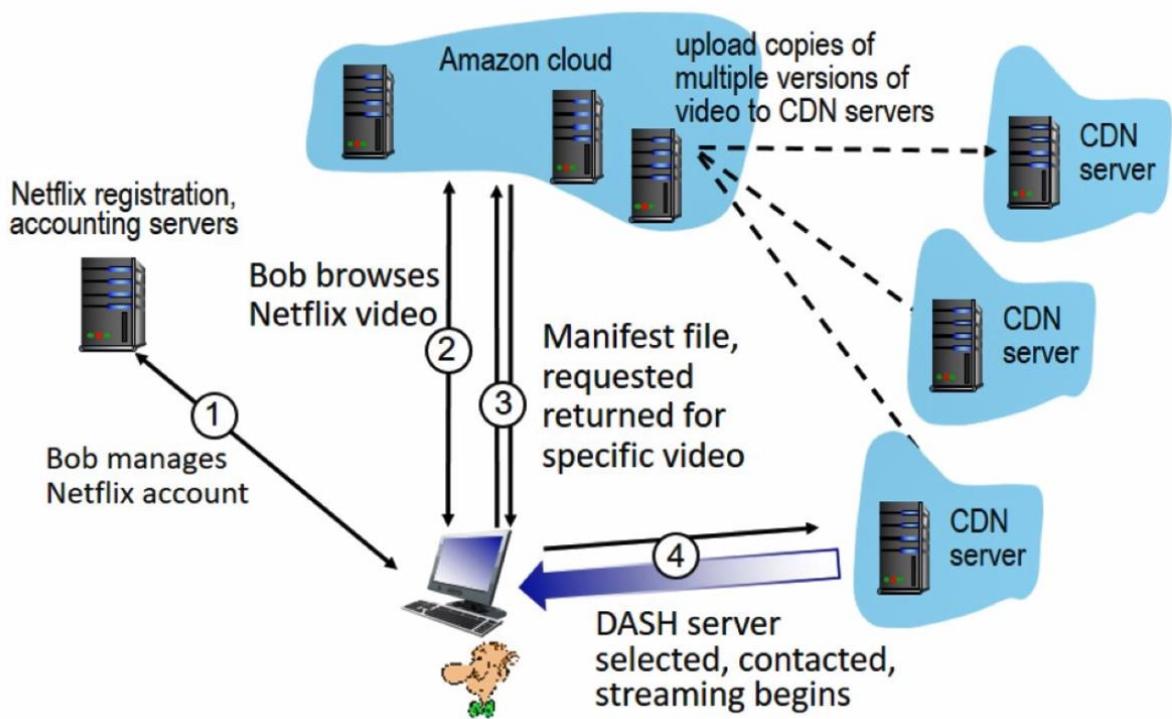
A. CDN: CDN 노드에 contents의 카피를 저장한다



### B. CDN 사용법

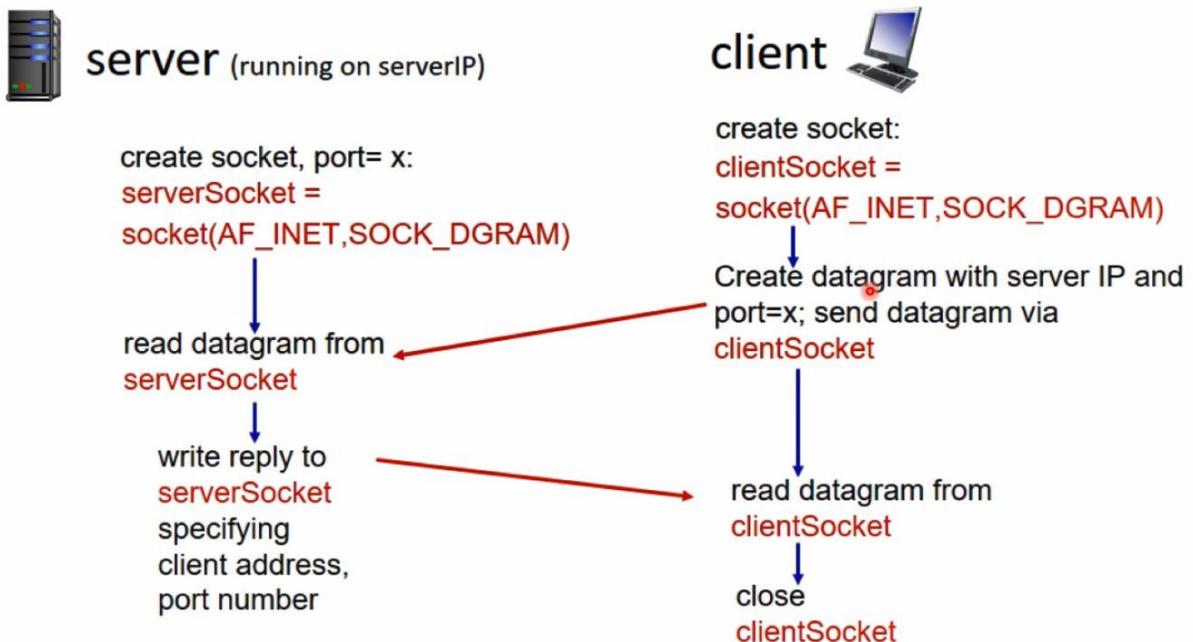
- i. 영상을 서비스하는 웹페이지 방문한다
- ii. 비디오의 링크를 클릭하면 LDNS를 통해 웹페이지의 authoritative DNS에서 CDN authoritative DNS를 거쳐 CDN 서버의 IP를 알려준다
- iii. 클라이언트는 CDN 서버에게 HTTP get 요청을 넣고
  1. DASH를 사용하지 않으면 >> get 요청에 응답한다
  2. DASH를 사용하면 >> CDN에서 manifest 파일을 받아오고 클라이언트는 그 파일에서 유동적으로 받아올 파일을 결정한다

### C. CASE study: Netflix



## 2. TCP/UDP socket programming

- A. Socket: door between application process and end-end-transport protocol
  - i. UDP: unreliable datagram
  - ii. TCP: reliable, byte stream-oriented
- B. UDP socket programming
  - i. Features
    - 1. No connection between client & server (=파일 전송전에 handshaking X)
      - A. 송신자는 각 packet마다 IP주소와 포트 넘버를 넣어야 한다 (connection을 만들고 하는 것이 아니기 때문)
      - B. 수신자는 각 packet에서 IP주소와 포트 넘버를 추출한다
      - C. Sender는 하나의 UDP 소켓을 만들면 여러 호스트와 통신할 수 있다 (어떤 호스트/포트와 통신하겠다고 정하지 않았기 때문이다.)



### Python UDPClient

```

include Python's socket library → from socket import *
serverName = 'hostname'
serverPort = 12000
create UDP socket for server → clientSocket = socket(AF_INET,
                                                       SOCK_DGRAM)
get user keyboard input → message = raw_input('Input lowercase sentence:')
attach server name, port to message; send into socket → clientSocket.sendto(message.encode(),
                           (serverName, serverPort))
read reply characters from socket into string → modifiedMessage, serverAddress =
                                               clientSocket.recvfrom(2048)
print out received string and close socket → print modifiedMessage.decode()
                                              clientSocket.close()

```

### Python UDPServer

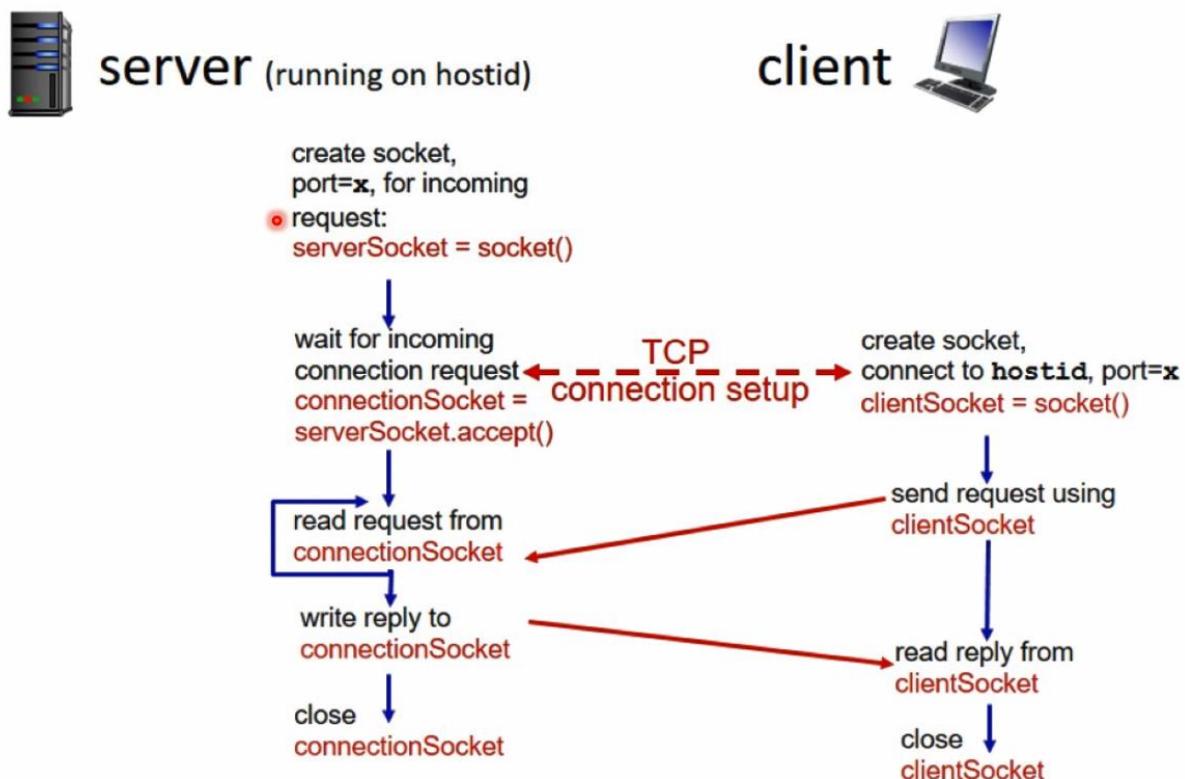
```

from socket import *
serverPort = 12000
create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port number 12000 → serverSocket.bind(("0.0.0.0", serverPort))
print ("The server is ready to receive")
loop forever → while True:
Read from UDP socket into message, getting → message, clientAddress = serverSocket.recvfrom(2048)
client's address (client IP and port) → modifiedMessage = message.decode().upper()
send upper case string back to this client → serverSocket.sendto(modifiedMessage.encode(),
                    clientAddress)

```

### i. Features

1. 통신 전 connection이 필수
  - A. Server process가 가장 먼저 실행되야 한다. (server는 client의 contact를 맞이할 socket을 가지고 있어야 하기 때문)
2. Socket을 만들 때, server process의 IP주소와 포트 넘버를 넣어서 Socket이 누구와 통신하기 위한 건지 명시해야 한다. (client가 socket을 만들 때, client TCP가 server TCP와 connection을 만든다)
3. Server와 client가 접속했을 때, server TCP는 새로운 socket을 만들어 그 socket으로 통신한다
  - A. 이는 server에 sub socket을 만들어 server가 여러 client와 통신할 수 있게 하기 위함이다



### Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
create TCP socket for server, -----> clientSocket = socket(AF_INET, SOCK_STREAM)
remote port 12000
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
No need to attach server name, port -----> clientSocket.close()
```

### Python TCPServer

```
from socket import *
serverPort = 12000
create TCP welcoming socket -----> serverSocket = socket(AF_INET,SOCK_STREAM)
server begins listening for -----> serverSocket.bind(('',serverPort))
incoming TCP requests -----> serverSocket.listen(1)
print 'The server is ready to receive'
loop forever -----> while True:
server waits on accept() for incoming -----> connectionSocket, addr = serverSocket.accept()
requests, new socket created on return
read bytes from socket (but -----> sentence = connectionSocket.recv(1024).decode()
not address as in UDP) -----> capitalizedSentence = sentence.upper()
connectionSocket.send(capitalizedSentence.
encode())
close connection to this client (but not -----> connectionSocket.close()
welcoming socket)
```

## Ch3. Transport-layer

### 1. Transport layer의 개괄

#### A. Transport service와 protocol

- i. 다른 host에서 돌아가는 application process들 사이의 logical communication을 제공한다
- ii. Transport protocol은 end system에서 작동한다
  1. Sender: application message를 segment로 나누어 network layer로 보낸다
  2. Receiver: segment를 모아 message로 만들고 application layer로 보낸다
- iii. 사용하는 protocol: TCP, UDP

## B. Transport layer과 Network layer의 비교

- i. Network layer: host 간의 logical communication을 담당한다
- ii. Transport layer: process 간의 logical communication을 담당한다
- iii. 비유로 살펴보는 차이점
  - 1. House 사이에 배달을 하는 집배원은 network layer이다
  - 2. 집배원에게 받은 편지를 수신인 아이에게 나누어 주는 첫째는 transport layer이다
  - 3. 첫째의 일처리와 집배원의 일처리는 관련 없다
  - 4. 집배원의 일처리가 되지 않으면 첫째의 일처리도 못한다

### *household analogy:*

*12 kids in Ann's house sending letters to 12 kids in Bill's house:*

- hosts = houses
- processes = kids
- app messages = letters in envelopes

각각의 집에 많은 아이들이 다른 집의 아이들에게 편지를 쓰고 집배원이 집으로, 첫째가 아이들에게 편지를 나누어 주는 상황

## C. Transport layer actions

- i. Sender
  - 1. Application layer에서 message를 전달받는다
  - 2. Segment 헤더 필드 value를 결정한다
  - 3. Segment를 만든다
  - 4. Segment를 network layer로 보낸다
- ii. Receiver
  - 1. Network layer에서 segment를 받는다
  - 2. 헤더 필드를 체크한다
  - 3. Application layer message를 추출한다
  - 4. Demultiplexing을 통해 message를 socket을 통해 application layer로 보낸다

## D. TCP와 UDP 개괄

- i. TCP (Transmission Control Protocol)
  - 1. Reliable, in-order delivery
  - 2. Congestion control: network에 부담을 주지 않는 전송을 한다(network에 부담주는 전송을 하면 packet loss가 일어날 확률도 높기 때문)
  - 3. Flow control: 송신자는 수신자의 처리 속도 이상으로 전송하지 않는다
  - 4. Connection setup: connection을 만든 후에 전송을 한다
- ii. UDP (User Datagram Protocol)
  - 1. Unreliable, unordered delivery
  - 2. 필요한 것만 있는 best-effort 프로토콜이다
- iii. TCP와 UDP에서 보장하지 않는 것
  - 1. Delay: 전송 딜레이가 어느 정도 이하라는 보장이 없다
  - 2. Bandwidth

## E. Multiplexing과 demultiplexing

- i. Multiplexing: 송신자의 application layer의 multi socket에서 message를 넘겨받아 transport layer의 header를 붙여 segment로 만드는 과정
  - ii. Demultiplexing: 수신자의 network layer에서 받아온 datagram에서 나온 segment header를 사용해 올바른 socket으로 넘겨주는 과정
    - 1. Demultiplexing의 과정
      - A. 호스트가 IP datagram을 받는다
        - i. 각각의 datagram은 source & destination IP를 가지고 있다
        - ii. 각각의 datagram은 하나의 transport layer segment를 가지고 있다
        - iii. 각각의 segment는 source & destination port number를
- 작각하지 말 것. IP address는 datagram에 있고 port number만 segment에 있다
- 
- The diagram illustrates the structure of a TCP or UDP segment. It consists of several horizontal layers. The top layer contains two fields: "source port #" and "dest port #", which are highlighted with a large red oval. Below these is a layer labeled "other header fields". The bottom layer is labeled "application data (payload)".
- TCP/UDP segment format

가지고 있다.

- B. Host는 IP address & port number을 사용하여 segment를 적절한 socket으로 보낸다

F. Connectionless demultiplexing (UDP)

i. Recall

1. Socket 생성할 때 host local port number을 명시해줘야 한다
2. UDP socket으로 보내 datagram을 만들 때 이하 정보는 필수다

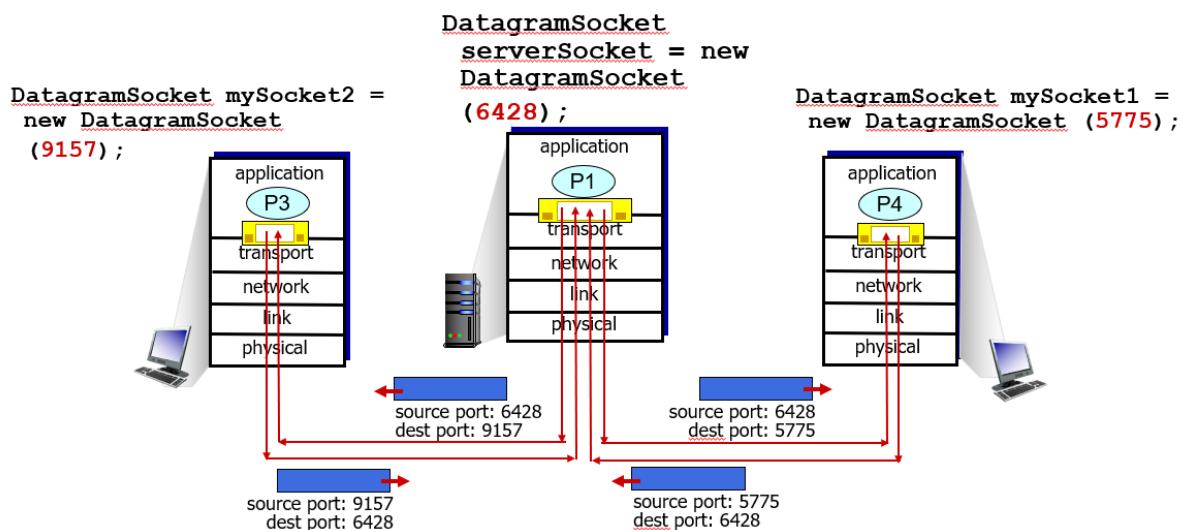
A. Destination IP address

B. Destination port number

ii. 수신자가 UDP segment를 받을 때

1. Segment에 있는 destination port number을 체크하여 socket으로 넘겨준다
- iii. IP/UDP datagram가 같은 destination port number가 같고 다른 source IP or port number 일 때
1. 수신자의 same socket으로 보내진다

iv. Connectionless demultiplexing 돌아가는 예시



1. Dest port가 같으면 같은 socket으로 가기 때문에 하나의 프로세스로 여러 프로세스와 통신할 수 있다
2. client에게 돌려줄 때는 받은 packet의 source port number/IP를 참조하여 보낸다

## G. Connection-oriented demultiplexing (TCP)

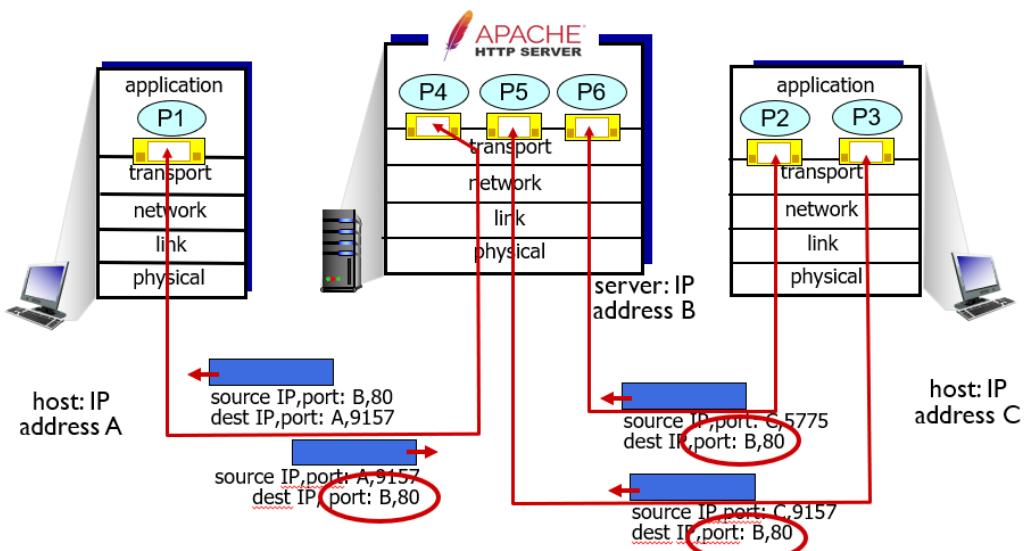
### i. 수신자가 TCP segment를 받을 때

1. 4가지 정보의 tuple로 어느 socket으로 보낼지 결정한다
  - A. Source IP address
  - B. Source port number
  - C. Dest IP address
  - D. Dest port number
2. Demultiplexing: 수신자는 4-tuple의 값 4가지를 모두 써서 socket을 결정한다

### ii. TCP socket의 특징

1. Server은 많은 TCP socket을 simultaneous하게 운용한다
  - A. 각각의 socket은 4-tuple로 identify된다
  - B. 모든 socket은 다른 client와 연결된다
- + ) socket programming TCP에서 이야기한 서브 프로세스를 다른 socket을 통해 연결시켜 주는 것

### iii. Connection-oriented demultiplexing 돌아가는 예시



1. Dest IP/port number가 같아도 4-tuple이 다르면 다른 socket/process와 연결된다

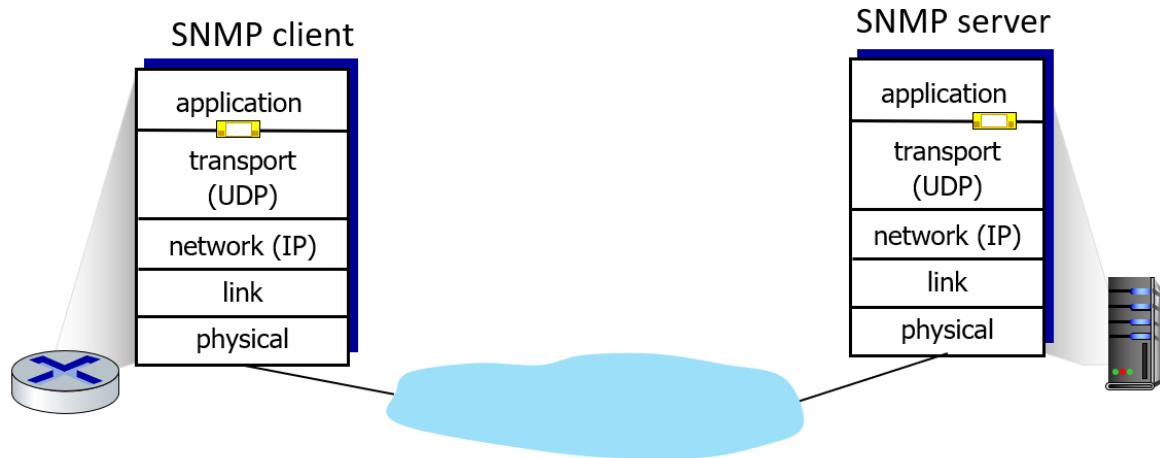
## H. Multiplexing, demultiplexing 정리

- i. Mult, demult 둘 다 segment/datagram의 header field에 기초하여 작동함
- ii. Mult, demult 둘 다 모든 layer에서 생긴다
- iii. TCP와 UDP demult 차이
  1. UDP: dest port number만 써서 specify한다
  2. TCP: 4-tuple의 값을 모두 써서 specify한다

## I. User Datagram Protocol (UDP)

- i. UDP는 어떤 protocol인가?
  1. 필요한 것만 있는 protocol
  2. Best effort protocol: data가 제대로 lost되지 않거나 보낸 순서대로 도착하는 것을 보장하지 않는다. (최선만 다할 뿐, 보장하는 것은 없다.)
  3. Connectionless
    - A. UDP 사이의 송신자, 수신자는 handshaking이 없다
    - B. 각각의 UDP segment는 독립적으로 다뤄진다
- ii. 왜 UDP를 사용하는가?
  1. No connection establishment: 초기 연결에 RTT delay가 없어서 초기 연결이 빠르다
  2. Simple: 각각의 segment가 독립적으로 다뤄지고 segment의 도착 확인 응답이 없기에 간단하다
  3. Header size가 TCP에 비해 작다
  4. No congestion control: TCP와 달리 UDP는 원하는 만큼 빠르게 파일을 전송 할 수 있다. (이는 network에 congestion을 일으키는 요인이 되기도 한다)
- iii. UDP의 쓰임새
  1. Streaming multimedia apps (loss tolerant, rate sensitive)
  2. DNS: 빠른 대답이 필요한 간단한 기능이기 때문
  3. HTTP/3: UDP 위에서 reliability와 congestion control을 추가하여 초기 접속을 빠르게 했다

#### iv. UDP가 작동 과정



##### 1. 송신자

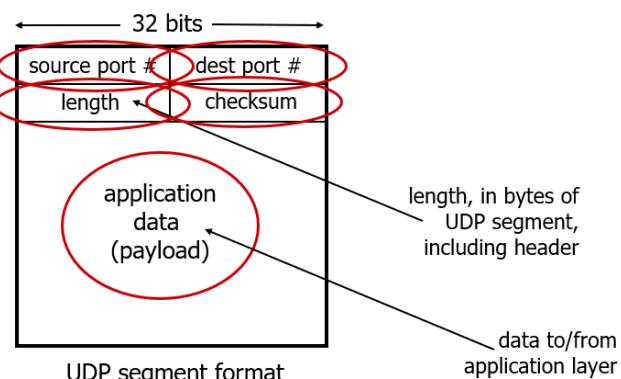
- Application layer로부터 message를 받는다
- UDP segment 헤더 필드의 값을 결정한다
- UDP segment를 생성한다
- Network layer로 넘겨준다

##### 2. 수신자

- Network layer로부터 segment를 받는다
- UDP checksum header value를 체크한다
- Application layer의 message를 추출한다
- Demultiplexes한 message를 socket을 통해 application layer로 전달한다

##### J. UDP segment header

- Length: header와 payload의 크기
- Checksum: segment의 모든 값을 더한 값의 1의 보수



K. UDP checksum: segment가 전송되는 동안 오류가 발생하지 않았나 판단하는 값

i. 송신자가 checksum을 만드는 방법

1. segment의 모든 값(payload + segment header + datagram의 IP address)을 16비트 워드의 정수 수열로 바꾼다
2. 바꾼 수열들을 다 더한다. (이 과정에서 생기는 overflow는 최하위 비트에 더해준다.)
3. 1의 보수를 취해준다

ii. 수신자가 checksum을 취급하는 방법: checksum을 이용하여 오류를 검출하긴 하지만 검출만 하지 딱히 대처를 하지 않는다

iii. 주의할 점: '0 1'이 '1 0'으로 바뀐 경우처럼 바뀌어도 오류 검출을 못하는 경우도 있다.

example: add two 16-bit integers

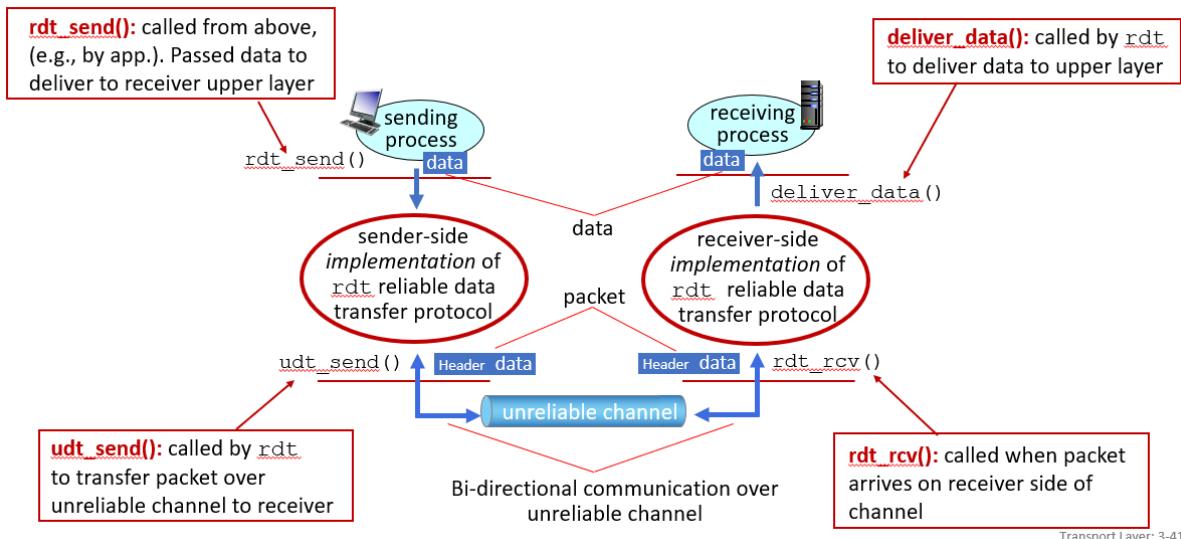
	1	1	1	0	0	1	1	0	0	1	1	0	
	1	1	0	1	0	1	0	1	0	1	0	1	
<hr/>													
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1
sum	1	0	1	1	1	0	1	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Checksum 구하는 예시

## L. Reliable data transfer (rdt)

### i. Rdt interface



1. 송신자의 process에서 `rdt_send()`를 통해 transport layer로 데이터를 보낸다
2. Transport layer에서 데이터를 packet으로 가공한 후에 `udt_send()`를 통해 unreliable channel로 보낸다
3. 수신자에게 도착했을 때 네트워크 인터페이스 카드가 `rdt_rcv()`를 호출한다
4. 별 문제가 없다면 `deliver_data()`를 통해 수신자의 process로 보낸다

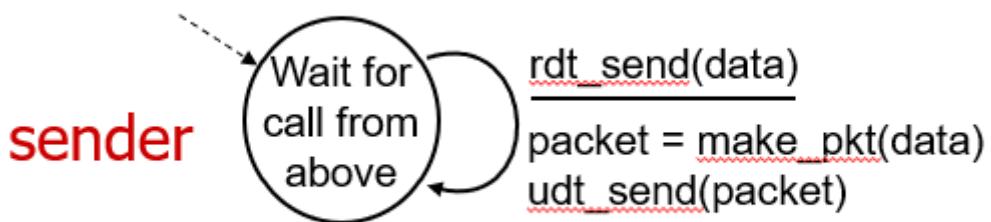
ii. Rdt 1.0: reliable transfer over a reliable channel

1. 특징

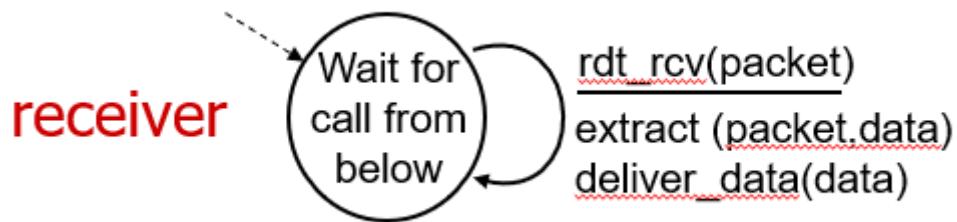
- A. Bit error가 없다
- B. Packet loss가 없다

2. FSM (Finite State Machine)

- A. 송신자: 오류가 없으니 위에서 송신 호출이 없으면 대기하고 호출이 있으면 packet을 만들어 보낸다



- B. 수신자: 오류가 없으니 수신할 packet이 없으면 대기하고 있으면 추출하여 위로 보낸다



iii. Rdt 2.0: channel with bit errors

1. 문제 상황

- A. Unreliable channel에서 packet의 error가 생겼을 경우 송신자에게 어떻게 알릴 것이며 어떻게 올바른 packet을 받아올 것인가?

2. 해결 방법

- A. Checksum을 통하여 bit error를 색출한다

- B. 제대로 된 packet을 받았을 때 ACK를 보내고 이외의 경우 NAK를 보내 송신자에게 상태를 알린다

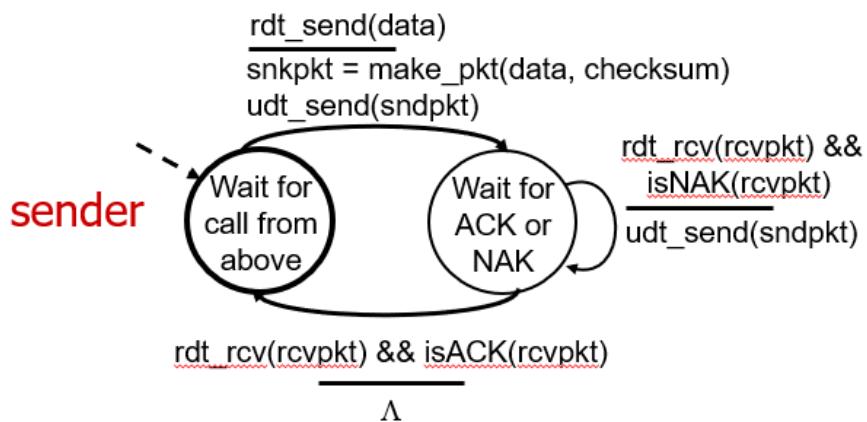
3. Stop and wait: 송신자는 packet을 하나 보내고 수신자의 대답(ACK, NAK)을 기다린다

4. FSM

A. 송신자

- i. 위에서 송신 호출을 기다리다가 호출이 있으면 packet을 만들어 보낸다

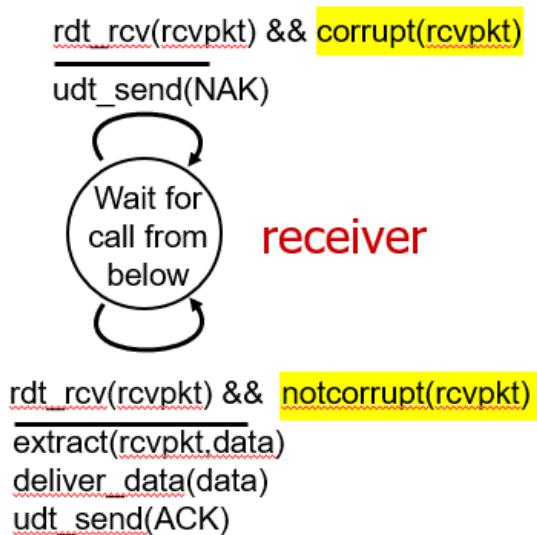
- ii. 수신자의 ACK 응답을 받으면 다시 대기 상태로 돌아가고 NAK 응답을 받으면 아까 만든 packet을 다시 보낸다



B. 수신자

- i. 송신자로부터 packet을 받기전에 대기하다가 packet을 받으면 checksum을 한다

- ii. checksum결과 오류가 없다면 데이터를 추출하여 위로 보내고 ACK 답장을 하고 아니면 송신자에게 NAK 답장을 한다.



#### iv. Rdt 2.1: handling garbled ACK/NAKs

##### 1. 문제 상황

- A. ACK/NAK에 error가 생기면 어떠하지?
- B. 재전송한 packet이 이미 있는 거면 어떠하지?

##### 2. 해결 방법

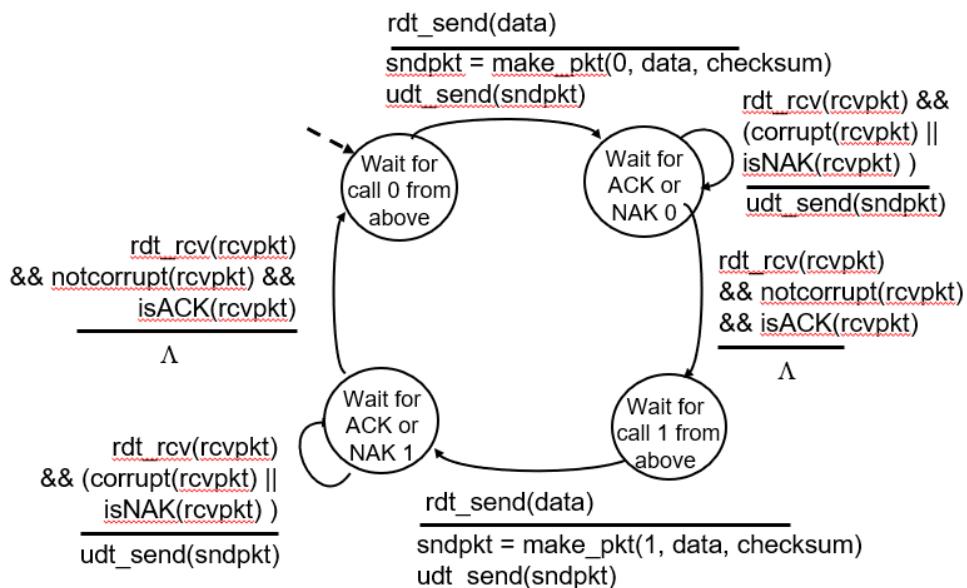
- A. ACK/NAK의 Checksum을 하여 ACK/NAK의 진위여부를 확인하고 에러가 있다면 재전송한다
  - B. Sequence number를 송신 packet에 넣어 받은 packet이 중복 수신한 packet인지 판단하고 중복이면 버린다
3. ACK/NAK checksum과 sequence number
- A. ACK/NAK checksum을 통해 NAK나 checksum 테스트 통과 못한 ACK를 색출하여 재전송한다
  - B. Sequence number를 송신 packet에 넣어 수신자가 이미 받은 packet이면 버리고 이런 중복 packet은 ACK로 보낸 response가 NAK로 변조되었기 때문이다. 그래서 ACK를 보내서 '이전 packet은 잘 받았으니 다음 packet을 보내 달라'는 요청을 한다.

##### 4. FSM

###### A. 송신자

- i. Packet을 보낼 때 packet의 sequence number를 보내 보내는 packet이

- ii. 보낸 packet에 대한 응답이 돌아오면 NAK/ACK인지, checksum 테스트를 통과하는지 확인한다.
  - iii. 만약 NAK거나 checksum 테스트를 통과하지 못한 ACK라면 보냈던 packet을 다시 보낸다. (이 경우 변조되어 NAK가 된 경우는 탐지하지 못한다)



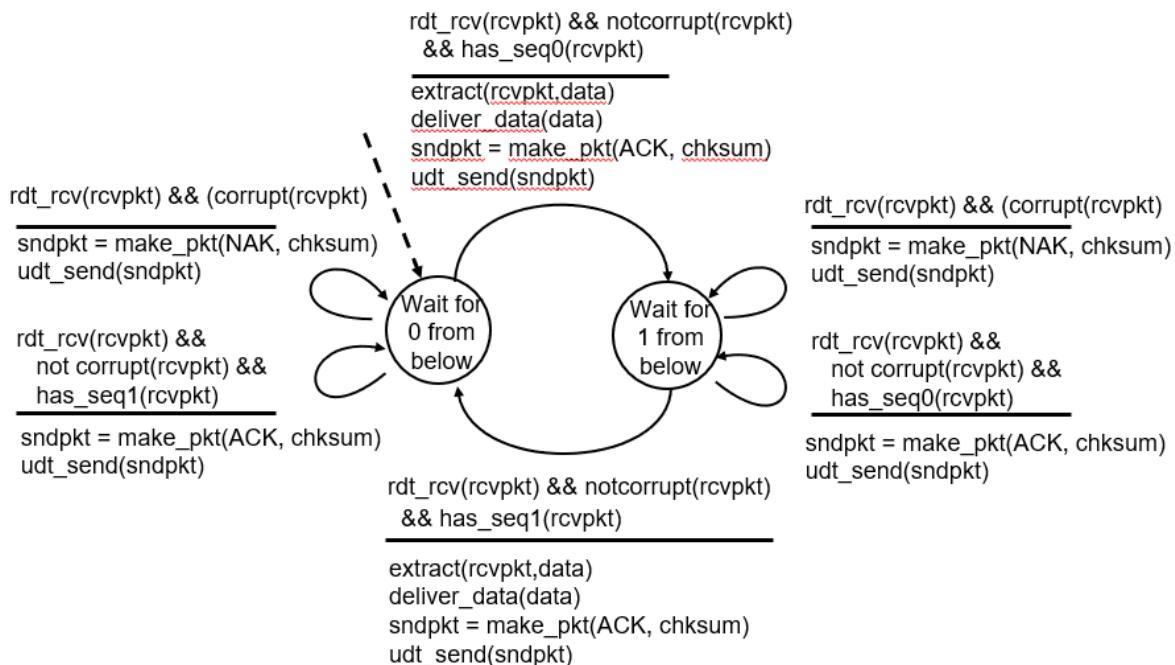
## B. 수신자

- i. 다음과 같은 경우 ACK를 전송한다

1. Checksum 테스트를 통과하고 sequence number가 일치할 때는 ACK를 전송하고 packet에서 데이터를 추출한 다음 '다음 packet'을 기다리게 된다
  2. Checksum 테스트를 통과했지만 sequence number는 일치하지 않을 때 ACK를 전송하고 받은 packet을 버린 후 '기다리던 packet'을 기다린다.

- ii. 다음과 같은 경우 NAK를 전송한다

- Checksum 테스트를 통과하지 못했을 때 NAK를 보내고 packet 을 버린 후 '기다리던 packet'을 기다린다



## 5. 특이점

- A. 송신자의 FSM의 state는 sequence number 수의 2배이다. (State로 어떤 sequence number에 대해 기억하는 상태와 기다리는 상태가 둘 다 필요하기 때문)
- B. 수신자는 자신의 ACK/NAK가 송신자에게 제대로 전달되었는지 모르기 때문에 sequence number로 duplicate 여부를 확인해야 한다

### v. Rdt 2.2: NAK-free protocol

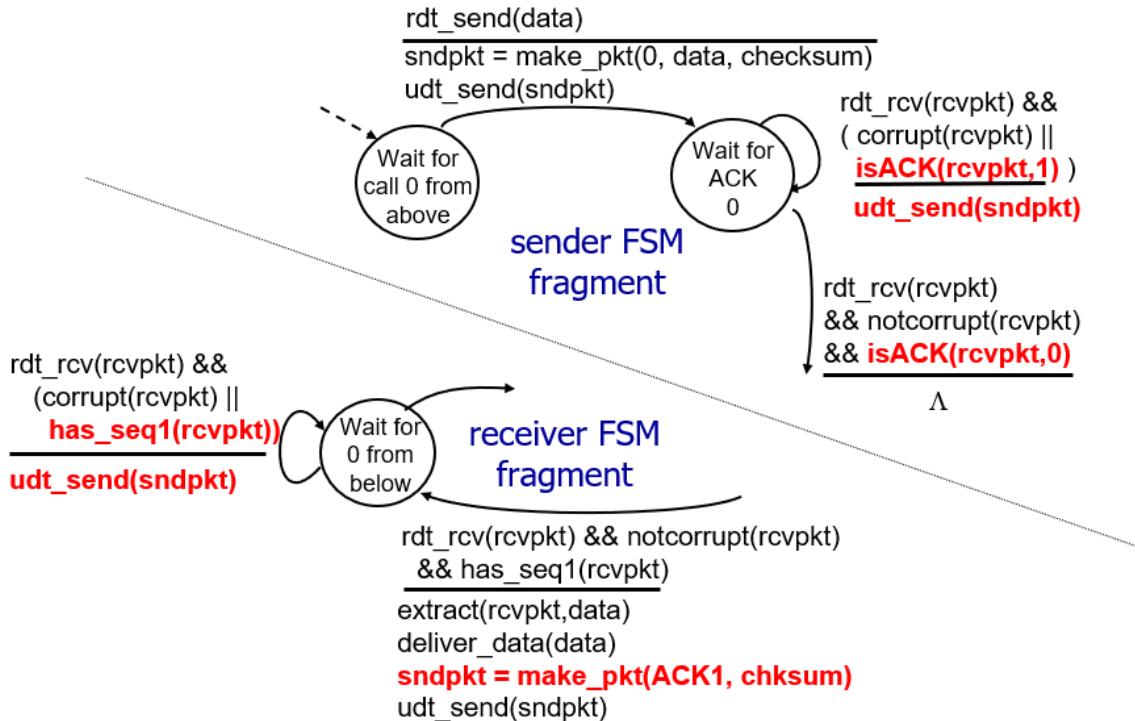
#### 1. Rdt 2.1과의 차이점

- A. NAK없이 ACK로만 통신 결과를 송신자에게 전달한다

#### 2. How to NAK-free

- A. NAK를 쓰는 대신에 마지막으로 받은 packet의 sequence number을 ACK에 넣는다
- B. 받은 packet에 문제가 있다면 정상적으로 수신한 마지막 packet의 sequence number를 ACK에 넣어 보낸다
- C. 만약 송신자가 받은 ACK의 sequence number가 이번에 보낸 sequence number가 아니면 최근에 보낸 packet에 문제가 있었다는 것이기 때문에 현재 packet을 다시 송신한다

### 3. FSM



#### vi. Rdt 3.0: channels with errors and loss

##### 1. 문제 상황

- A. 만약 packet이 loss된다면 seq #, checksum 등의 장치들은 소용이 없게 된다

##### 2. 해결 방법

- A. Reasonable한 시간을 기다린 후에도 대답이 없으면 loss한 걸로 취급해 packet을 재송신한다

##### 3. Timeout

- A. Reasonable한 시간을 설정하여 그 시간동안 packet에 대한 response가 없으면 packet이건 ACK/NAK건 loss되었다고 생각하여 재송신하는 방식

- B. 이미 다른 error에 대해서는 rdt 2.x에서 다루었기에 이를 재활용한다

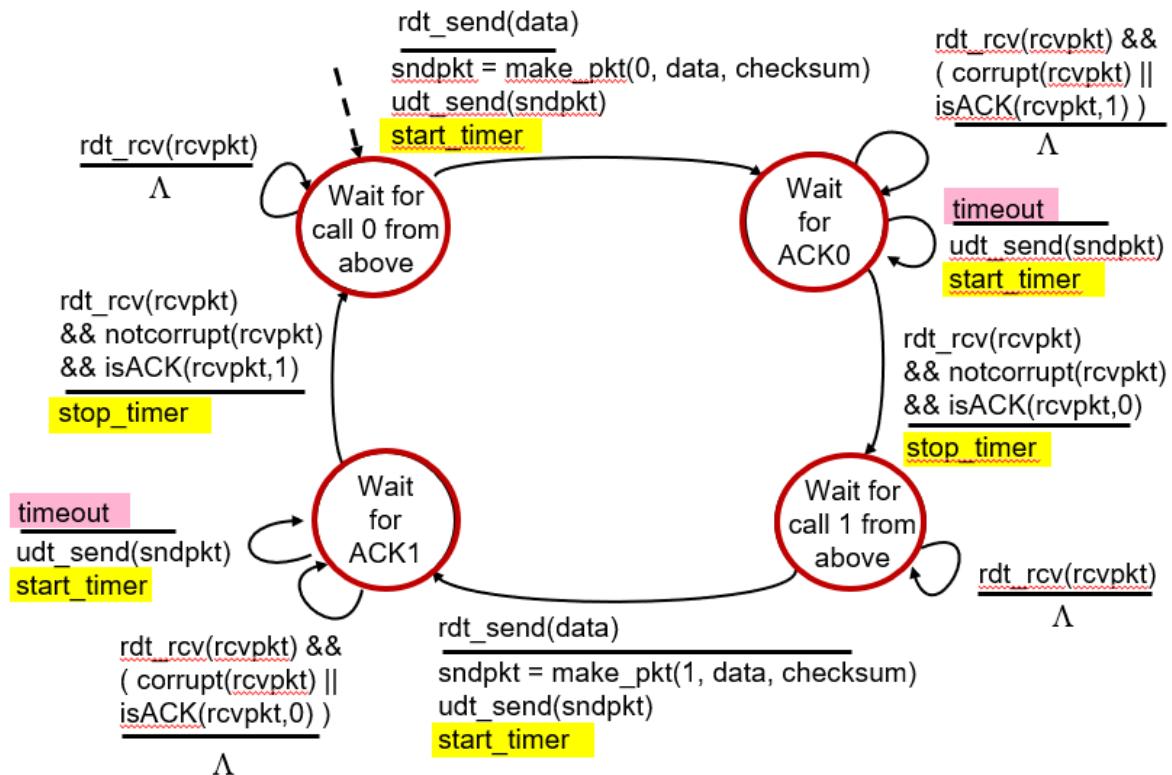
##### 4. FSM

###### A. 송신자

- i. Packet을 새로 보내거나 timeout되었을 때 start\_timer한다.
- ii. ACK가 checksum 테스트를 통과하지 못하거나 방금 보낸 sequence

number가 아니라면 stop\_timer를 하지 않고 그대로 timer를 진행 한다

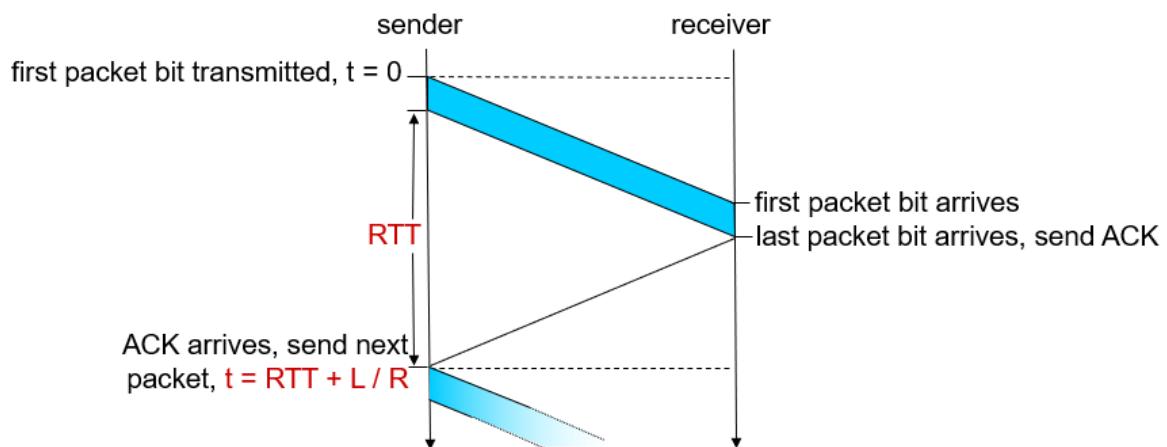
- iii. 계속해서 이상한 ACK를 받거나 packet이 loss된다면 timeout이 될 텐데 그 경우 packet을 재전송하고 start\_timer을 한다



## B. 수신자

- i. Rdt 2.2와 똑같다

## 5. Rdt 3.0 utilization: 송신자가 바쁘게 보내는 시간의 fraction



$$A.U_{sender} = \frac{\frac{L}{R}}{\frac{L}{R} + RTT}$$

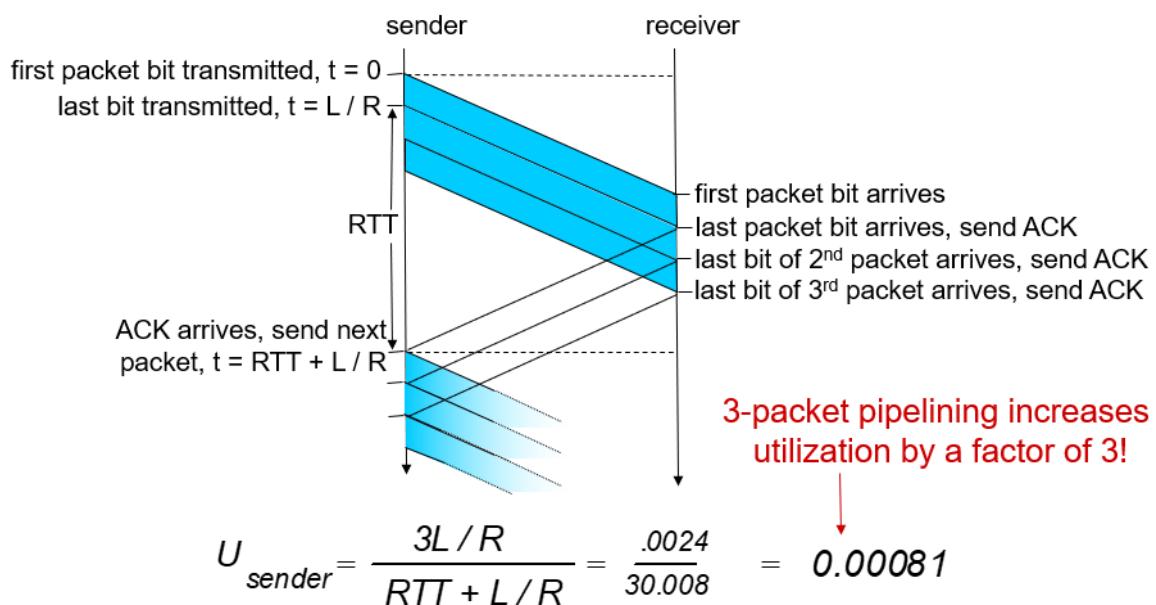
vii. Rdt 3.0: pipelined protocols operation

1. 문제 상황

A. 송신자의 utilization이 RTT 때문에 낮아질 수밖에 없다

2. 해결 방법

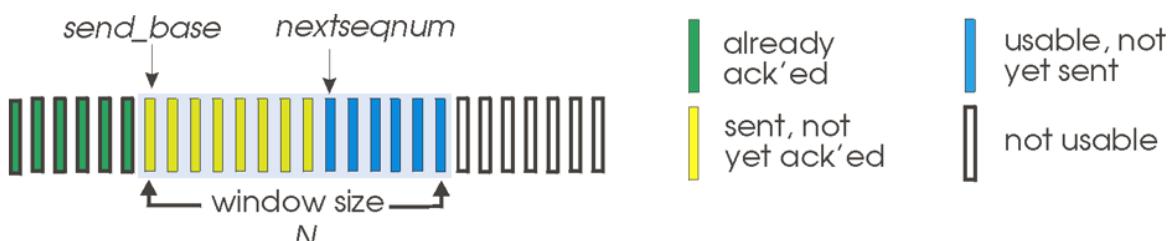
A. 한번에 여러 packet을 보내서 utilization을 높인다



3. Go-Back-N

A. Window: host들이 송신하거나 수신하는 packet 중에서 관심을 가지는 범위

B. 송신자: N이라는 크기의 window 안에 있는 packet들을 한번에 연속하여 보낸다



초록색은 이미 송신이 끝났고 window안의 packet들은 현재 송신이 가능한 packet들이다

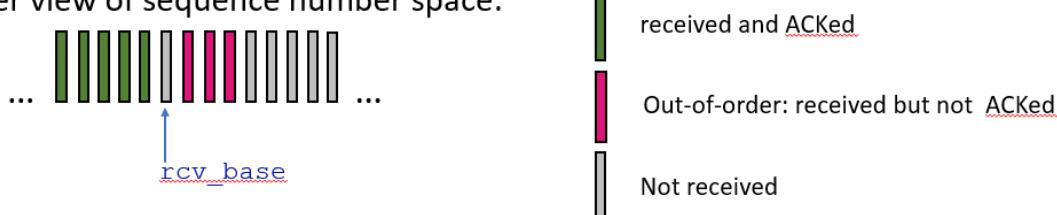
### C. 중요한 개념

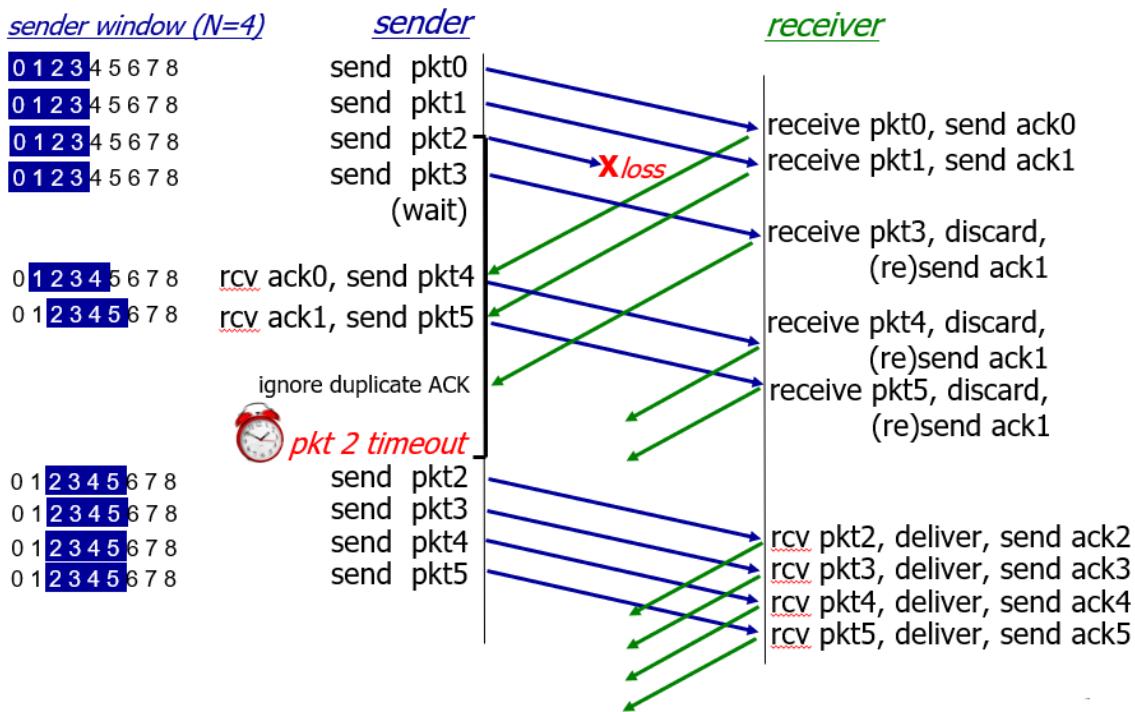
- i. Send\_base: 아직 ACK를 받지 못한 packet중 가장 앞의 packet의 sequence number
- ii. Cumulative ACK: ACK(n)의 답변이 오면 이는 n번까지는 제대로 전송되었으나 n+1번부터 전송해달라는 뜻이다.
  1. 이 경우 window를 n+1번으로 옮긴다

### D. 어떻게 go back N이 돌아가는가?

- i. Send\_base을 기준으로 window 안에 있는 packet들을 전송한다
  - ii. Timer를 oldest한 현재 송신중인 packet을 기준으로 한다
  - iii. Timeout(n)이 생긴다면 n번부터 window에 있는 packet들을 전부 재전송한다
- E. 수신자: 제대로 받은(checksum을 통과하고, 보낸 순서대로 온 것) packet들 중에 가장 높은 sequence number(rcv\_base)를 ACK에 넣어서 송신자에게 보낸다
- i. 이는 duplicate한 ACK를 만들 수도 있다
  - ii. Out-of-order한 packet을 받았을 경우
    1. Out-of-order packet은 discard할 수도, buffer할 수도 있다. 구현하는 개발자 나름
    2. 현재 rcv\_base를 담은 ACK를 재전송한다

Receiver view of sequence number space:

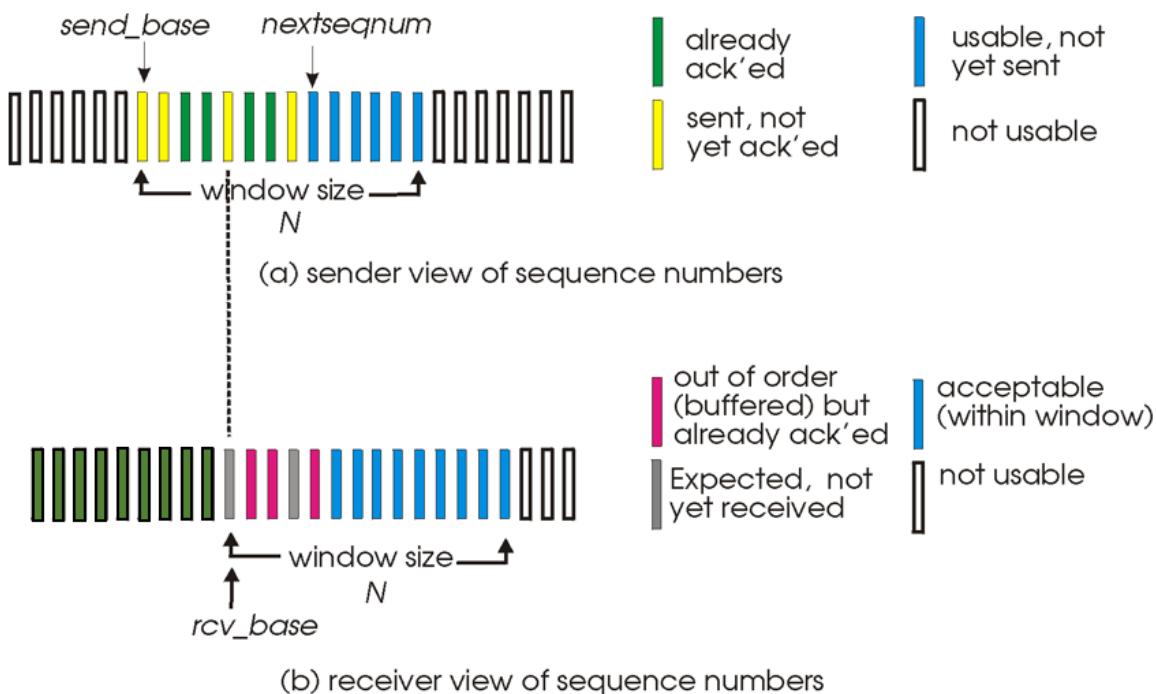




4. Selective repeat: Go-back-N에서는 timeout이 나면 모든 packet을 다시 보냈는데 selective repeat에서는 timeout난 packet만 보낸다

#### A. 특징

- 수신 host는 모든 packet을 개별적으로 처리한다
  - 개별적으로 packet을 buffer하고 in-order해지면 상위 계층으로 보낸다
- 송신자는 개별적으로 unACKed packet들을 time out/재전송 시킨다
  - 송신자는 각각의 unACKed packet에 대한 timer를 유지하고 있다



### iii. Selective repeat가 돌아가는 법

#### 1. 송신자

- A. 상위 계층에서 온 데이터가 있을 때
  - i. 만약 window에 송신할 sequence number가 남으면 packet으로 만들어 보낸다
- B. Timeout(n): n번째 packet이 타임아웃 되었다는 신호라서 n 번을 다시 보내고 timer도 재시작 한다
- C. ACK(n) in  $[sendbase, sendbase+N]$ 
  - i. N번 packet이 잘 전송되었다는 거니 packet n은 ACK 되었다고 마크한다
  - ii. 만약 unACKed packet이 ACK된다면 window base를 다음 unACKed packet으로 옮긴다

#### 2. 수신자

- A. Packet n in  $[rcvbase, rcvbase+N-1]$ 
  - i. ACK(n)을 보낸다
  - ii. Out-of-order면 buffer(저장)한다

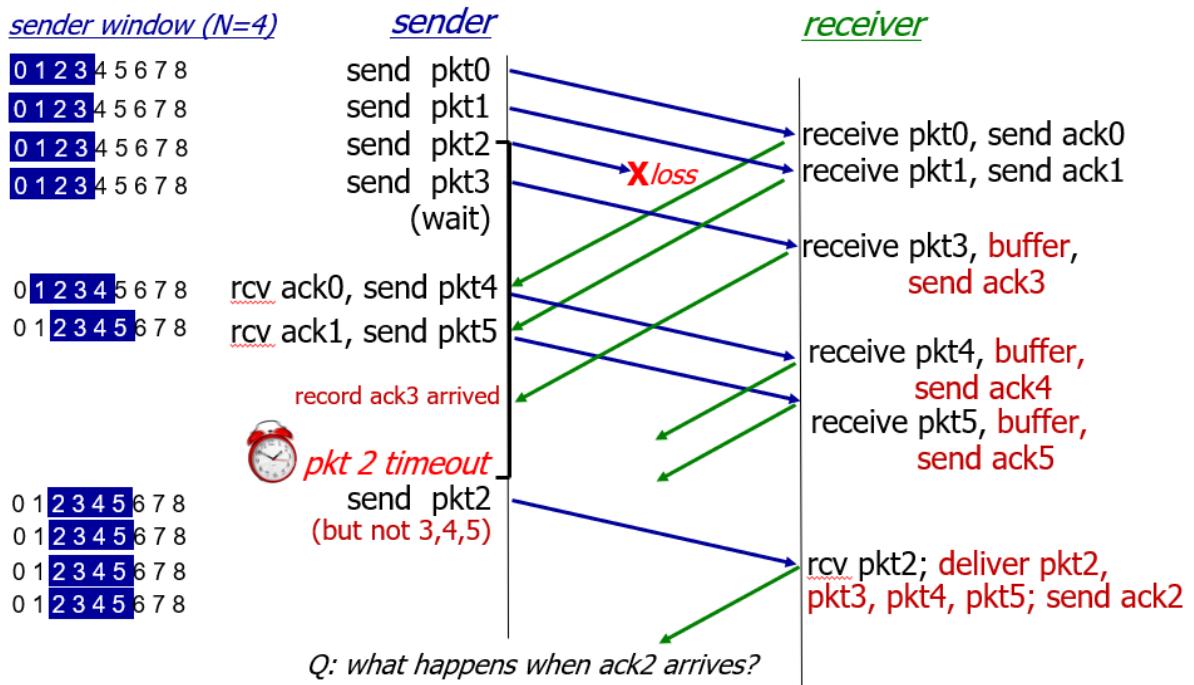
- iii. In-order하면 이미 저장되었는 다른 in-order한 packet 이랑 같이 상위 계층으로 보내고 window를 다음 아직 받지 않은 packet으로 옮긴다

#### B. Packet n in [rcvbase-N, rcvbase-1]

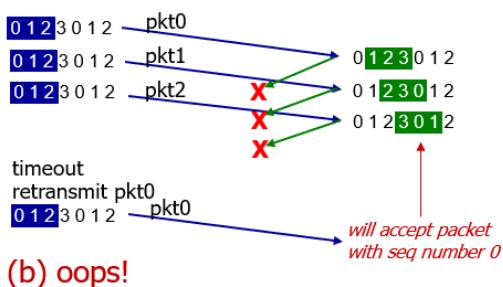
- i. 이미 받은 것을 재전송한 것이기 때문에 버리고 ACK(n)을 보내 이미 잘 받았다는 사실을 송신자에게 알려준다

#### C. 그 외

- i. 무시한다



#### 5. SR에 있어서 window-sequence number 딜레마



Timeout이후 전송하는 pkt0가 수신자 입장에서는 처음 pkt0의 재전송인지 다음 pkt0의 재전송인지 알 수 없다

A. Window 크기가 sequence number에 비해서 너무 크면 문제가 생길 수 있다.

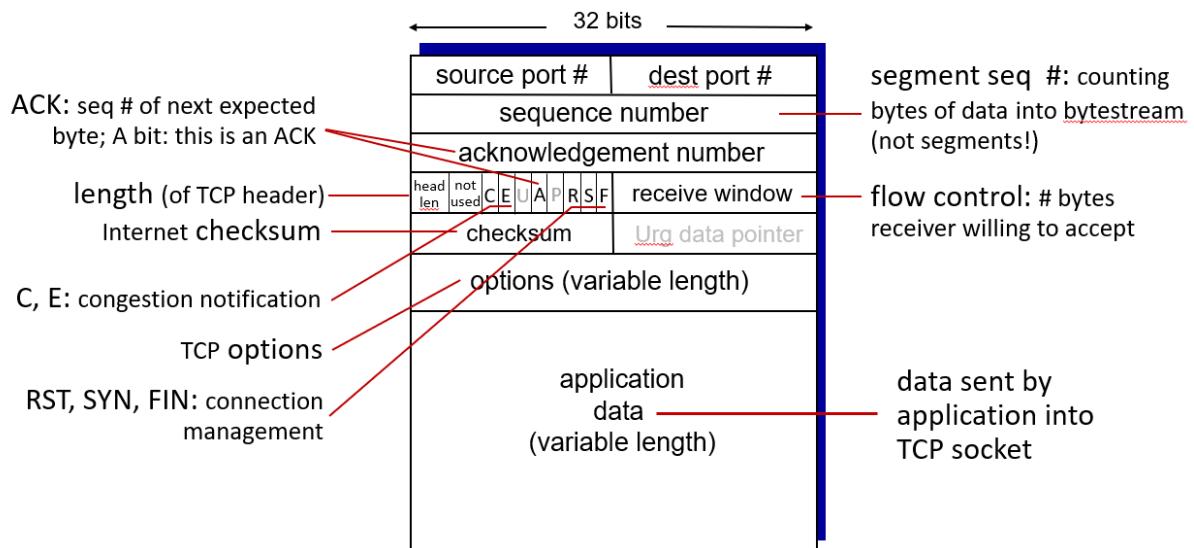
B. 그림에 보이는 문제를 해결하기 위해 window 크기는 적어도 sequence number의 반보다 작거나 같아야 한다.

## M. Transfer control protocol (TCP)

### i. TCP 개괄

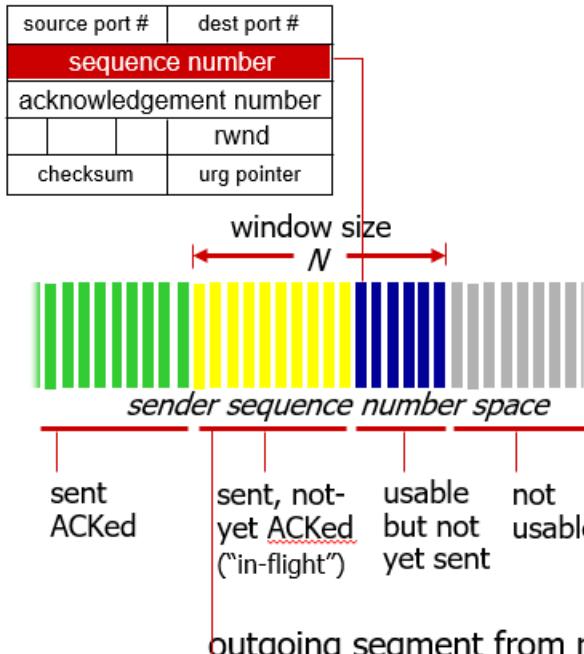
1. Reliable, in-order byte stream: 보낸 packet의 내용과 순서를 보장하며 보내는 데이터를 각기 나눠진 message가 아닌 하나의 stream으로 취급한다
2. Full duplex data: 하나의 connection으로 양방향으로 데이터를 보내고 받을 수 있다
  - A. MSS: maximum segment size
3. Cumulative ACKs
4. Pipelining: TCP의 congestion/flow control을 통해 window size를 결정한다
5. Connection-oriented: control message 교환인 'handshaking'으로 데이터 교환 이전에 송신자/수신자의 상태를 초기화한다
6. Flow controlled: 송신자는 수신자의 속도를 넘을 수 없다

### ii. TCP segment structure



1. Sequence number: 세그먼트에 들어가는 첫번째 byte의 byte stream에서의 순서(segment에서의 순서가 아니다)
2. Acknowledge number: tcp flag의 A가 설정되어 있을 때 사용하는 것으로 다음 전송에서 올 첫번째 byte의 byte stream에서의 순서
3. Flow control: 수신자가 받을 수 있는 byte의 수
4. Head length: TCP 헤더의 길이

## outgoing segment from sender



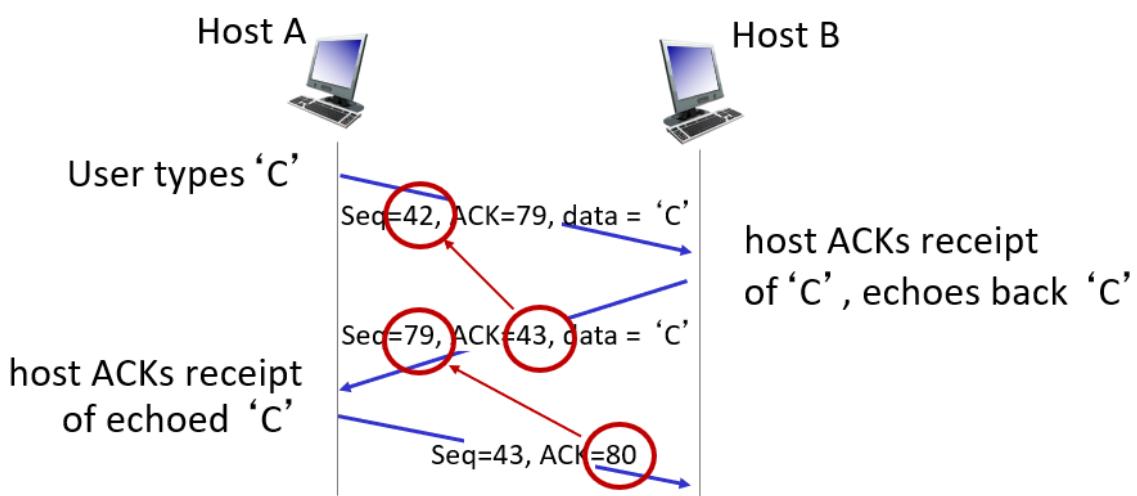
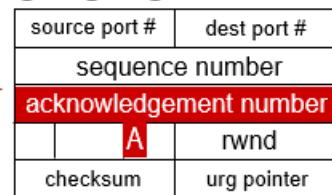
5. Checksum: packet이 error가 있는지 없는지 테스트하는 부분

6. Tcp flag: 제어 메시지 대신 제어 비트인 tcp flag로 제어한다

A, C, E: congestion notification

B. RST, SYN, FIN: connection management

## outgoing segment from receiver



## simple telnet scenario

+ ) 왜 초기 sequence number와 acknowledgement number이 0으로 초기화되지 않을까?

1. 이전 통신의 packet이 나중에 다른 통신에 전달되었을 때 현재 통신의 packet인지 아닌지 보기 위해

## 2. 보안을 위해서

iii. TCP의 RTT와 timeout: RTT를 너무 짧게 하면 필요 없는 timeout이 생겨 불필요한 재전송을 해야 되고 너무 길면 segment loss에 대해 느리게 반응하게 된다.

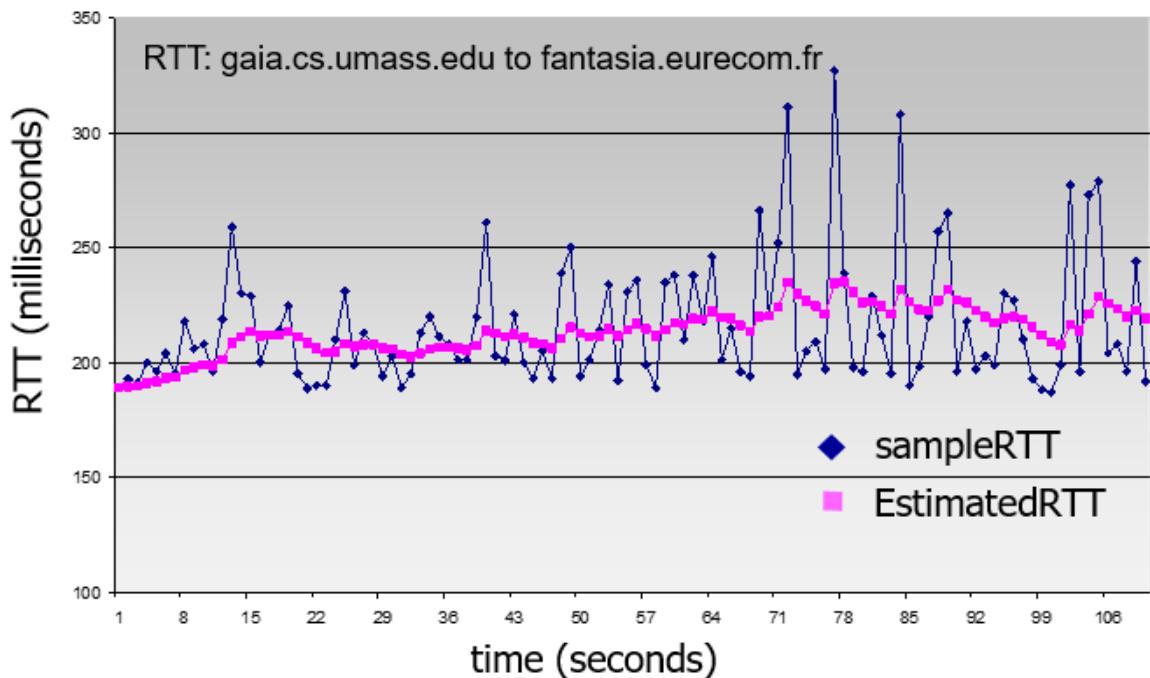
### 1. How to estimate RTT

A. SampleRTT: segment 전송으로부터 ACK 받을 때 까지의 측정된 시간

- i. 측정 시에 재전송은 무시한다
- ii. sampleRTT는 매우 다양한 값이 나올 수 있기 때문에 현재 RTT뿐만 아니라 '최근' RTT도 고려하여 평균을 내야 한다

B.  $EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * sampleRTT$

- i. 식의 뜻: 최근의 sampleRTT 일수록  $\alpha$ 를 곱한 횟수가 적기 때문에 ( $\alpha < 1$ ) 최종 EstimatedRTT에 더 큰 가중치를 가진다
- ii. Exponential weighted moving average (EWMA) 방식
- iii. 보통  $\alpha$ 값으로는 0.125를 쓴다



C. DevRTT: EWMA SampleRTT deviation from EstimatedRTT(대충 미분 때린 거 비슷한거)

- i. 식:  $DevRTT = (1 - \beta) * DevRTT + \beta * |sampleRTT - EstimatedRTT|$
- ii. 식의 뜻: EstimatedRTT처럼 최근의 (측정값-평균값)이 더 큰 가중치

를 가진다

- iii. 보통  $\beta$ 값으로는 0.25를 쓴다

D.  $TimeOutInterval = EstimatedRTT + 4 * DevRTT$

- i. 식의 뜻: 평균값에 safety margin을 더해주는 것이며 safety margin은 실제값과 평균값의 차이가 클수록 커지는 DevRTT로 했다
- iv. TCP reliable data transfer: 미리 이야기하자면 TCP의 rdt는 Go-back-N과 SR을 섞어 놓은 형태다. (Go-back-N처럼 loss가 발생했을 때 모든 packet을 재전송하지 않으며, SR처럼 packet마다 timer를 가지고 있지 않다.)
  - 1. 송신자(event별로 정리)

A. Event: application으로부터 데이터를 받았을 때

- i. Sequence number로 segment를 만든다
- ii. Sequence number는 segment안 byte stream의 첫 byte의 넘버다
- iii. Timer가 작동중이지 않으면 작동시킨다
  - 1. Timer의 기준은 항상 unACKed한 가장 오래된 segment다
  - 2. TimeOutInterval이 지나면 현재 timer는 만료된다

B. Event: timeout

- i. Timeout을 일으킨 segment를 재전송한다
- ii. Timer를 재시작한다

C. Event: ACK received

- i. ACK가 아직 unACKed한 segment에 대해서 도착한다면 이하의 일을 한다
  - 1. ACKed 상태를 업데이트한다
  - 2. 아직 unACKed segment가 존재한다면 timer를 재시작한다
- 2. 수신자(event에 따른 ACK 생성으로 정리)

A. Expected sequence number을 가진 In-order인 segment가 도착했고 expected sequence number까지의 데이터들은 모두 ACKed일 때

- i. Delayed ACK: 500ms 정도 다음 segment를 기다리고 오지 않으면

ACK를 보낸다. (곧이어 새로운 in-order segment가 온다면 ACK를 2개 보내야하기 때문에 통신 리소스 아끼는 명목인듯)

B. Expected sequence number을 가진 In-order인 segment가 도착했고 하나의 segment가 ACK를 기다리고 있을 때

i. 두 segment(방금 도착한거랑 기다리고 있던 것)에 대한 하나의 cumulative ACK를 즉시 보낸다

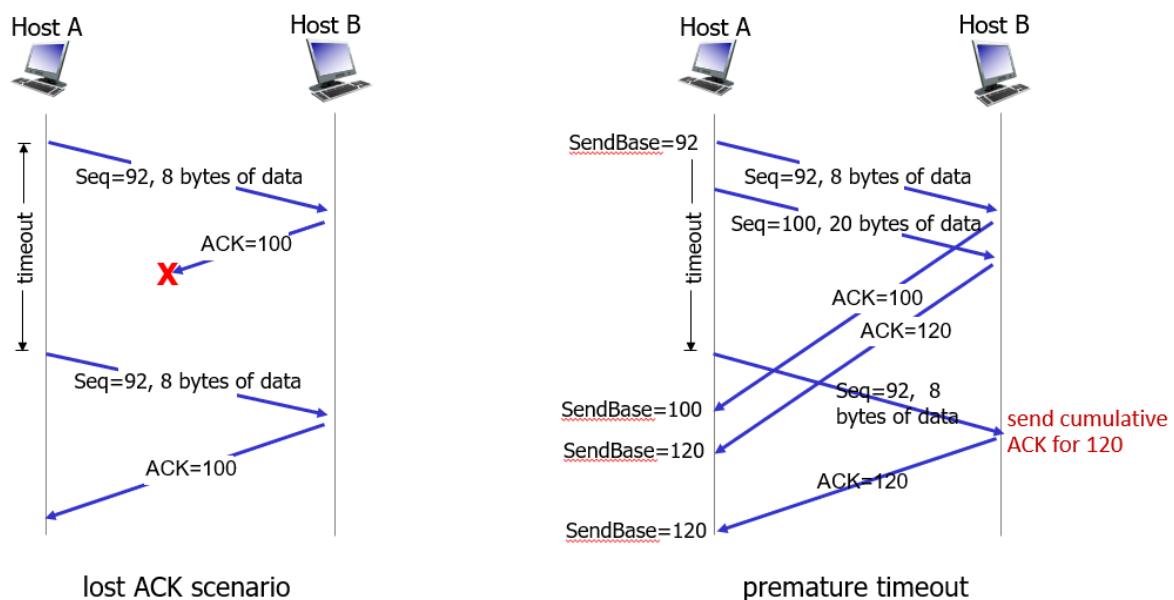
+ A와 B를 종합했을 때 별 일 없으면 ACK는 in-order segment 2개마다 보낸다는 것을 알 수 있다

C. Expected seq#보다 큰 Out-of-order segment가 도착하여 expected seq#와 gap이 생길 경우

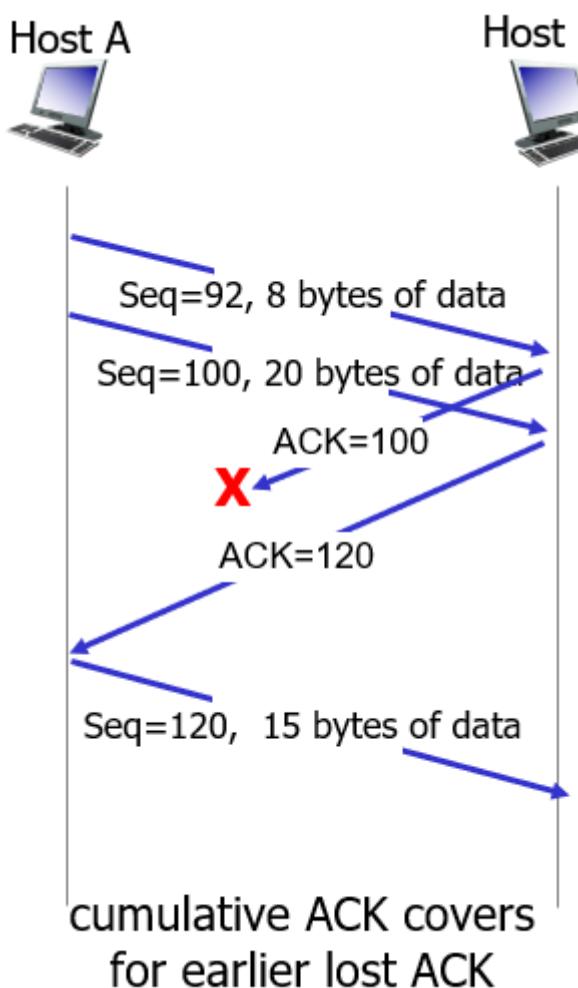
i. 즉시 duplicate ACK를 보내 expected byte(gap의 최솟값)의 seq#을 요구한다

D. 부분적으로, 전체적으로 gap을 채워주는 segment가 도착한 경우

i. 받은 segment가 gap의 최솟값에서 시작하는 경우 즉시 ACK를 보낸다



왼쪽은 ACK loss 때문에 segment를 재전송하는 것이고 오른쪽은 timeout이 나서 segment 92는 재전송했지만 segment 100이 재전송되기 전에 ACK 120이 도착했기에 segment 100은 재전송하지 않는다



ACK 100은 loss되었지만 ACK 120이 cumulative ACK역할을 해준다

3. TCP fast retransmit: 같은 데이터에 대해 ACK를 3번 더 받으면 그 다음 segment가 loss되었다는 의미기 때문에 timeout이 아니어도 즉시 가장 작은 unACKed된 seq#의 segment를 재전송한다

#### v. TCP flow control

##### 1. 문제상황

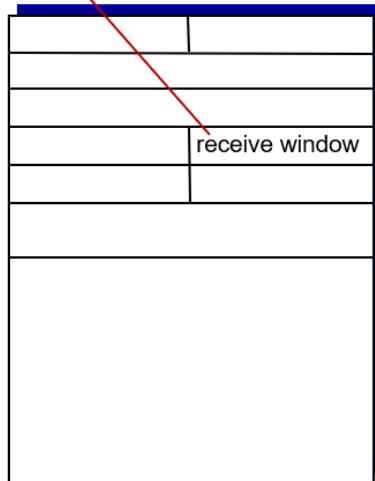
- A. Network layer가 수신자에게 수신자의 처리속도보다 빠르게 데이터를 보내면 TCP socket receiver buffer가 다 차서 오버플로우가 나지 않을까?

##### 2. 해결방법

- A. 송신자에게 현재 buffer에 남은 공간을 알려주고 그 이하로 보내게 하면 해결된다

##### 3. TCP flow control

flow control: # bytes receiver willing to accept



TCP segment format

A. RcvBuffer: socket 옵션으로 정해지는 전체 TCP 수신 버퍼의 크기이다(보통 4096byte이다)

B. Rwnd: 현재 수신 버퍼에 남아있는 공간의 크기이다. (rwnd 이하로 보내면 수신 버퍼에 오버플로우가 나지 않는 것을 보장한다)

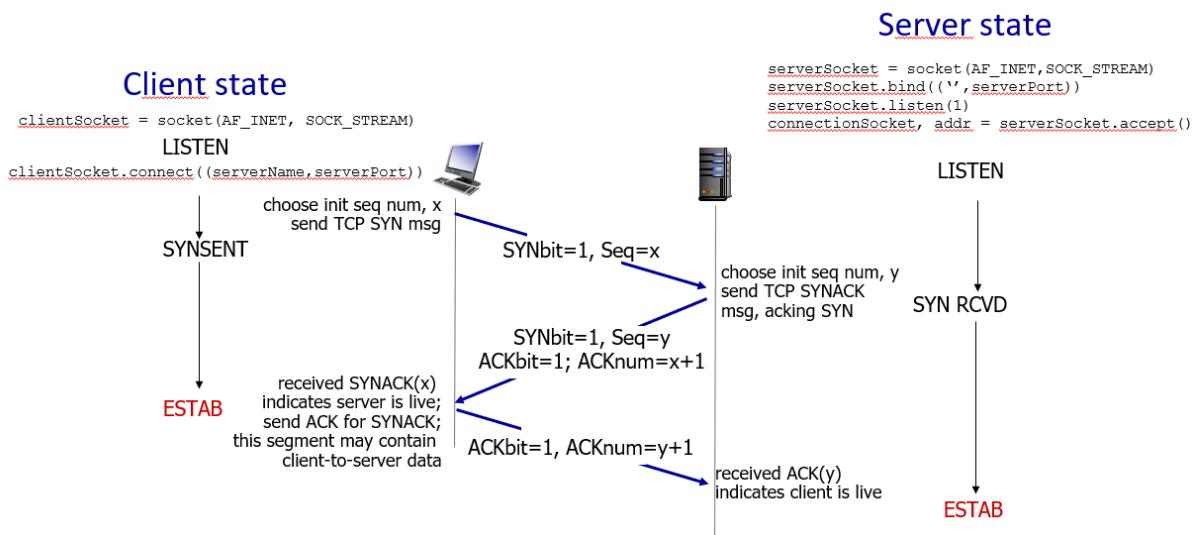
i. rwnd=RcvBuffer-buffered data

C. 수신자의 receive window 필드를 받은 송신자는 rwnd를 넘지 않도록 unACKed 데이터를 제한한다

#### vi. TCP 3-way handshake

1. 2-way handshake: 자신이 보낸 request/response가 잘 도착했는지 알 수가 없어서 retransmit을 하여 같은 서비스를 2번 제공하게 될 수도 있다

2. 3-way handshake



A. 1번째에서 SYNbit=1을 하여 서버에 연결의사가 있음을 밝히고 Seq=x를 통해 자신과 통신하기 위해서는 x번 seq#로 통신을 시작한다는걸 알아야 한다고 알려준다

B. 2번째에서 서버도 SYNbit=1을 하여 연결을 허가한다고 하고 seq=y를 통해 자신과 통신하기 위해서는 y번 seq#로 통신을 시작한다는걸 알아야 한다고 알려주며 아까보낸 x를 확인했기 때문에 x+1부터 보내달라

는 의미로 ACKnum=x+1을 했다

- C. 3번째부터는 통신할 데이터도 같이 보낼 수 있게 하여 2-way에 비해 통신 delay가 긴 3-way의 단점을 줄였다

### 3. TCP connection을 끝내는 법

- A. TCP connection 종료는 server, client 둘 다 할 수 있으며 종료 스텝을 시작하려면 TCP segment의 FIN bit=1을 하여 보내면 된다
- B. 상대방은 ACK에 FIN을 함께 보낸다
- C. 동시에 보내는 FIN도 handle될 수 있다

## vii. Principles of congestion control

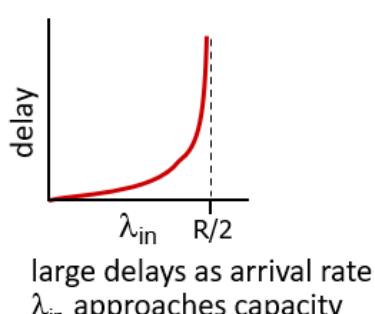
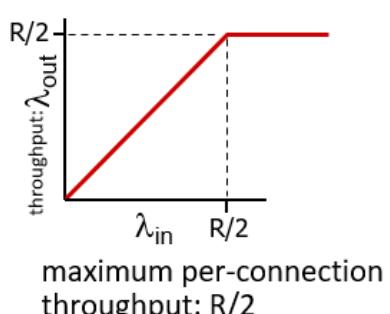
### 1. 왜 congestion control이 필요한가?

- A. Congestion: network가 다루기에 너무 많은 source들이 너무 많은 data를 너무 빠른 속도로 보내는 것
  - i. Router buffer에 queueing으로 인한 long delay가 생긴다
  - ii. Router buffer overflow로 인한 packet loss가 생긴다
- B. Flow control과 다른점: flow control은 송신자 1곳과 수신자 1곳간의 control이라면 congestion control은 특정 network 안의 모든 host와 관련된 일이다

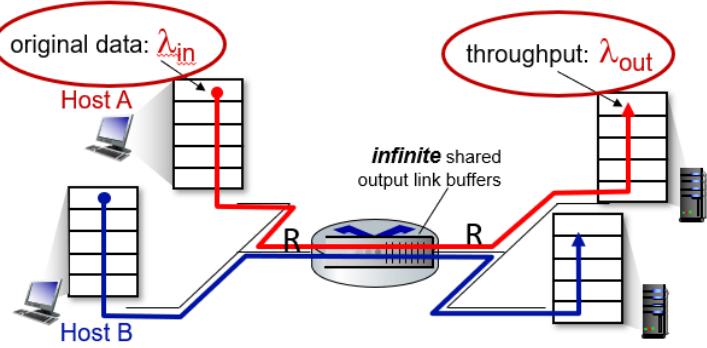
### 2. Congestion 원인/비용 시나리오

#### A. 시나리오 1

- i. 1개의 router와 infinite buffers(retransmission이 필요 없다)
- ii. Input, output link capacity: R
- iii. Two flows



2개의 flow가 R의 input/output link를 사용하기 때문에 각각  $R/2$ 가 maximum-throughput이고 이 경우 delay가 엄청 커진다



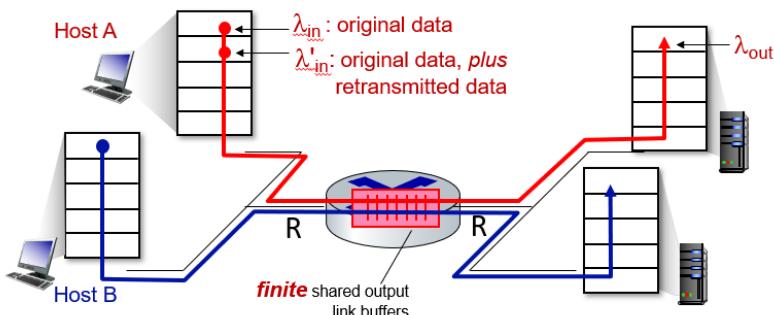
## B. 시나리오 2

- i. 시나리오 1에서 router buffer가 finite해진다

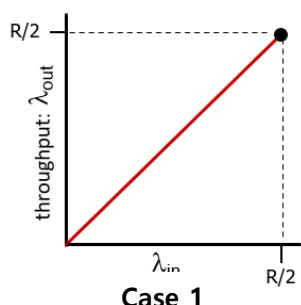
1. 송신자가 lost/timeout으로 인한 재전송을 해야 한다

A.  $\lambda_{in} = \lambda_{out}$

B.  $\lambda'_{in} \geq \lambda_{in}$



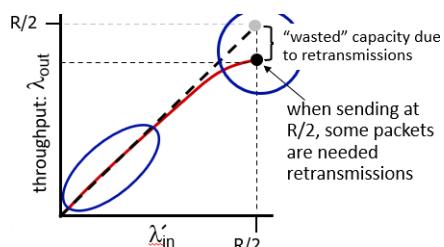
## ii. Case별 분석



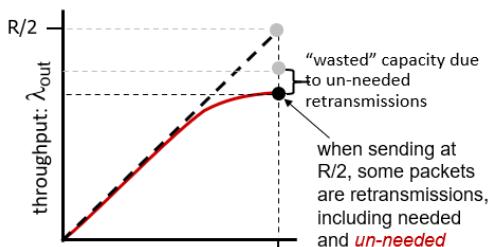
A. Case 1: 송신자가 buffer가 available할때만 전송을 한다(No retransmission,  $\lambda'_{in} = \lambda_{in}$ )

B. case 2: 송신자가 보내는 packet 중에 일부는 overflow로 drop되고, 송신자는 drop된 순간 알게 되어 재전송 한다

C. Case 3: case 2의 송신자가 prematurely timeout하기도 한다



Case 2. Retransmission으로 인해 case 1보다 throughput이 떨어졌다

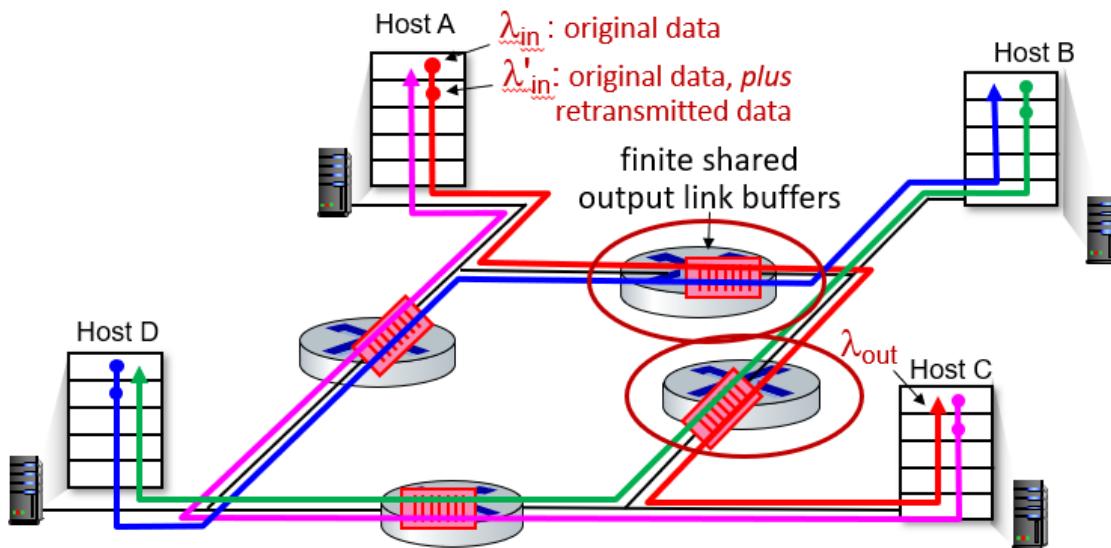


Case 3. Premature timeout으로 필요없는 재전송을 해서 throughput 악화

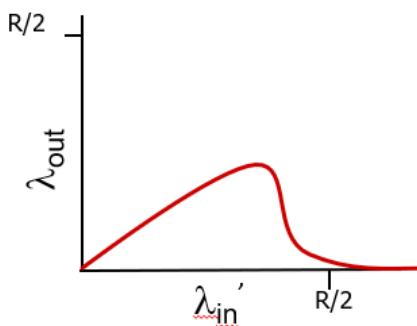
- iii. Congestion의 cost: retransmission으로 인한 traffic 증가 & throughput 하락

### C. 시나리오 3

- i. Four senders
- ii. Multi-hop paths
- iii. Timeout/retransmit



- iv. Multi-hop 관련: 어떤 hop에 보내지는 data양은 이전 hop에 보내지는 data양을 넘어서 수 없다
- v. 질문: 만약 모든 Host가 무제한으로 data를 보낸다면 어떻게 되나?



1. Host A>Host C 예시) 첫번째 hop에서 Host A의 전송 속도가 무제한이기 때문에 Host D에서 오는 packet들은 거의 drop되고 Host A가  $R$ 을 다 차지할 것이다
2. 두번째 hop으로 갈 수 있는 packet은  $R$ 만큼인데 Host B가 무제한으로 보내기 때문에 결국 Host A에서 온 packet들은 두번째 hop에서 drop되어 아무것도 Host C에 도달하지 못한다

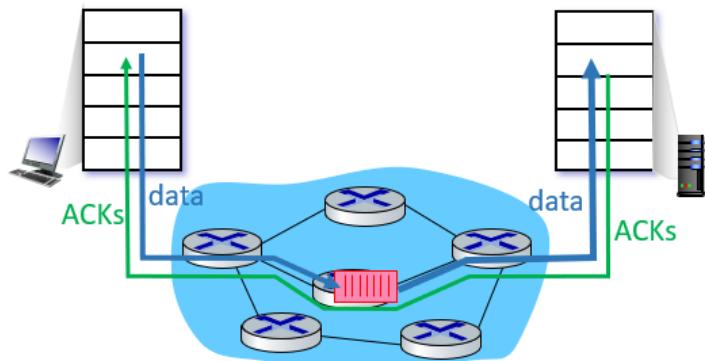
- vi. 결론: 이렇게 packet이 drop되면 이 packet을 전송하기 위해 쓰인 transmission capacity나 buffering 공간은 낭비된 것이며 성능만 저하된다>>another cost of congestion

Throughput은 전송속도를 빨리한다고 좋아지는 것이 아니며 ‘적당한’ 속도일 때 최고이다

### 3. Congestion control 하는 법

#### A. End-End congestion control

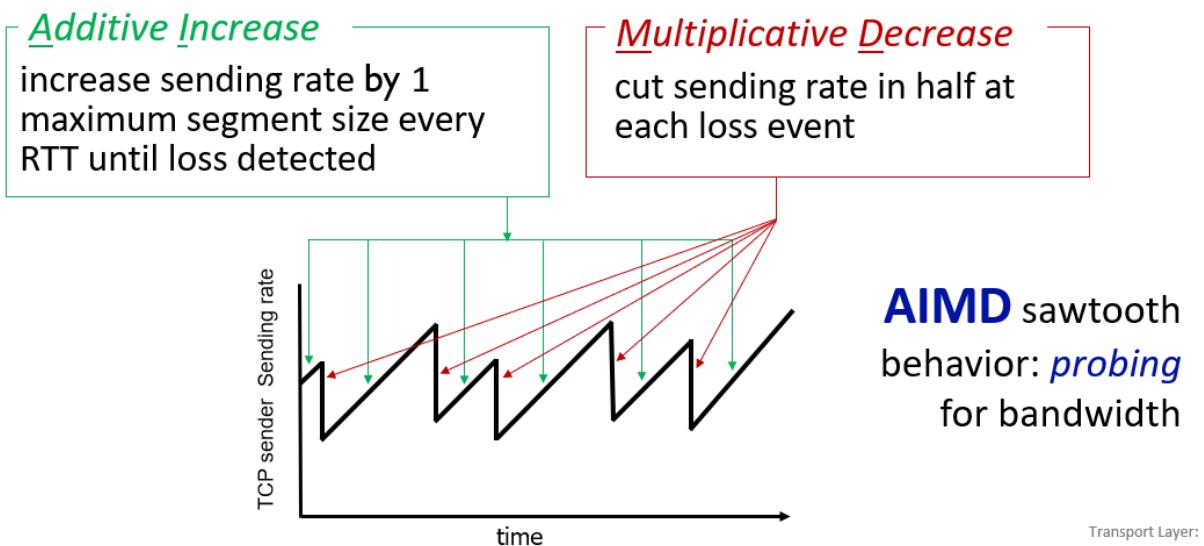
- i. network에서 explicit한 피드백이 없다
- ii. loss, delay로 congestion을 추측한다
- iii. TCP로 접근한다



#### B. Network-assisted congestion control

- i. Router가 직접적으로 피드백을 host에게 준다
  - ii. Congestion level을 알 수 있으며, explicitly하게 송신 속도를 set한다
- viii. TCP congestion control

1. TCP AIMD (Additive Increase Multiplicative Decrease): packet loss(congestion)이 생길때까지 송신 속도를 높이고 loss event 발생 시 속도를 낮춘다



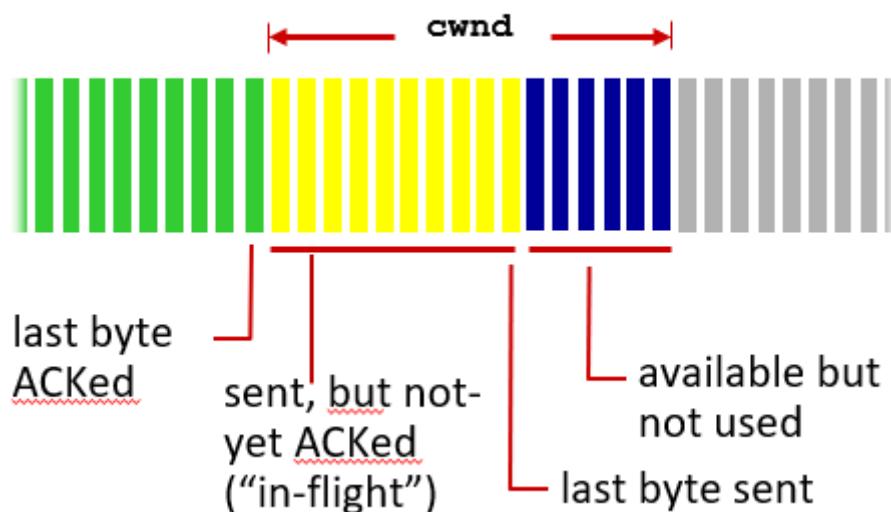
Transport Layer:

### A. Multiplicative decrease 방식

- i. 전송 속도를 반으로 한다(duplicated ACK 3개로 loss 발견 시)
- ii. 전송 속도를 1MSS로 한다(timeout/duplicated ACK 3개로 loss 발견 시)

### 2. TCP congestion control 개괄

#### sender sequence number space



초록색은 이미 처리 끝난 byte, 노란색+파란색은 이번에 보낼 byte들, 노란색은 현재 전송은 되었지만 ACK되지 않은 byte들, 파란색은 아직 상위 layer에서 오지 않은 byte들

- A. TCP 송신자: cwnd byte를 보낸 후에 RTT동안 ACK를 기다리고 그 후에 또 다시 보낸다

$$\text{i. } \text{TCP rate} \approx \text{cwnd}/\text{RTT} (\text{bytes/sec}) \quad (\text{TCP rate는 cwnd에 비례한다})$$

- B. TCP 송신자의 전송 제한:  $\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$

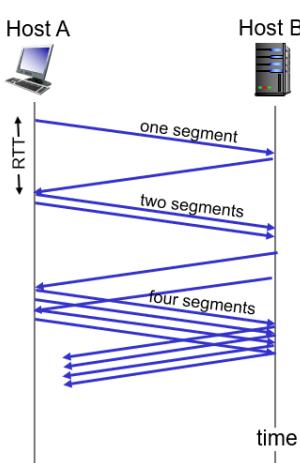
- C. Cwnd는 네트워크 congestion에 의해 동적으로 조절된다

### 3. TCP slow start

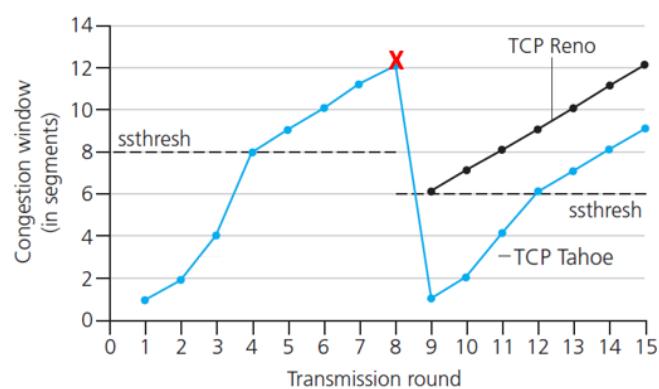
- A. Connection이 시작된 뒤로 첫번째 loss event가 생기기 전까지 exponential하게 전송 속도를 증가시키는 방법이다.

$$\text{i. 초기 cwnd}=1\text{MSS}$$

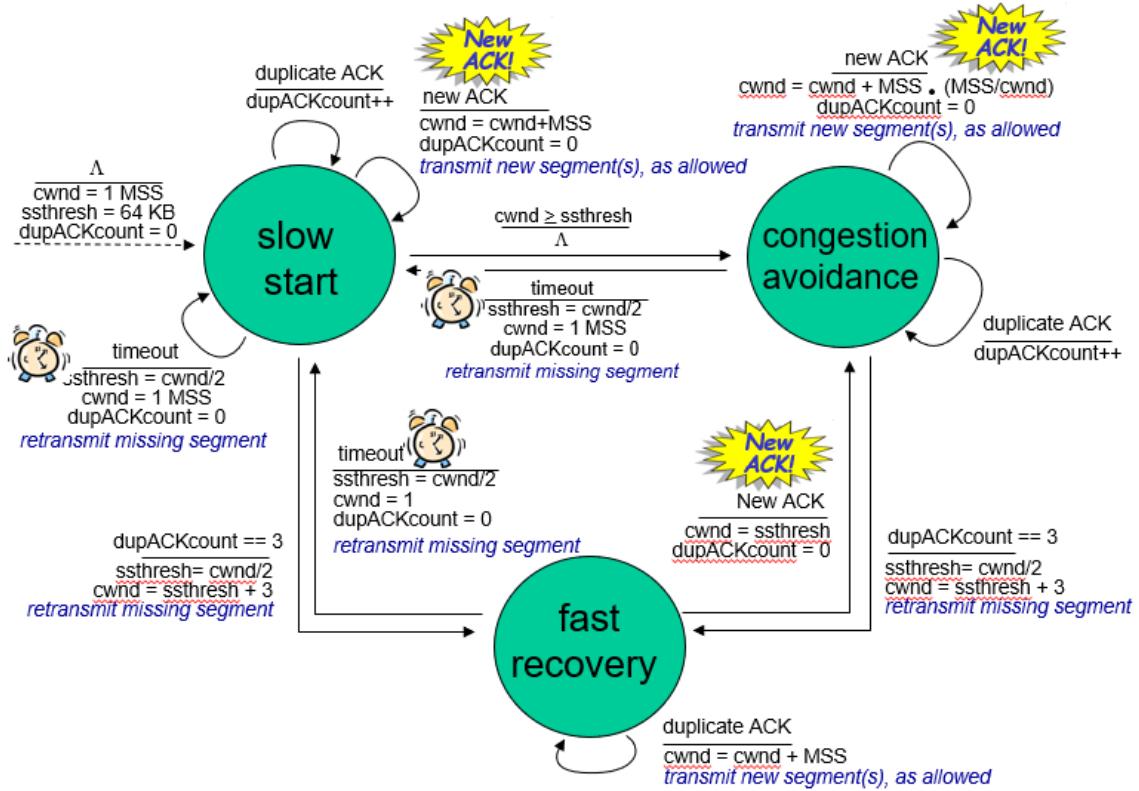
$$\text{ii. 매 RTT마다 cwnd를 2배로 늘린다}$$



4. TCP congestion avoidance: slow start로 어느정도 속도를 키웠을 때(ssthresh), congestion avoidance로 전환하여 cwnd를 선형적으로 증가시킨다
  - A. Ssthresh: 마지막 packet loss가 났을 때의 cwnd의 절반으로 ssthresh를 기준으로 exponential 증가와 linear 증가를 구분한다
5. TCP Tahoe와 TCP Reno
  - A. TCP Tahoe: timeout/duplicated ack 3개 둘 다 cwnd를 1로 줄인다
  - B. TCP Reno: timeout 때는 cwnd를 1로 줄이고 duplicated ack 3개일 때는 cwnd를 반으로 줄인다(ssthresh)

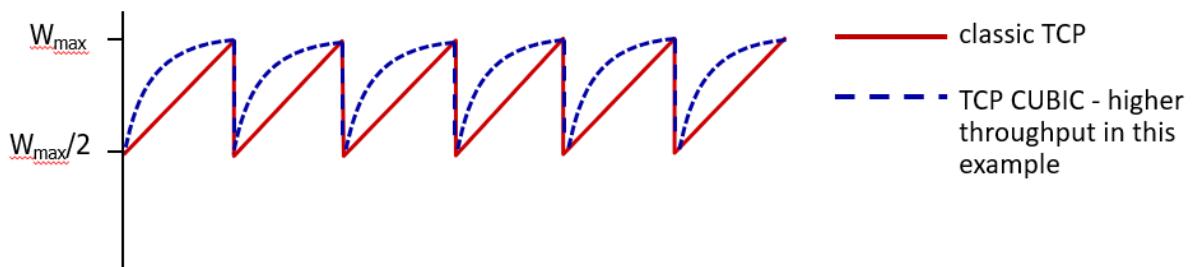


## 6. TCP congestion control summary



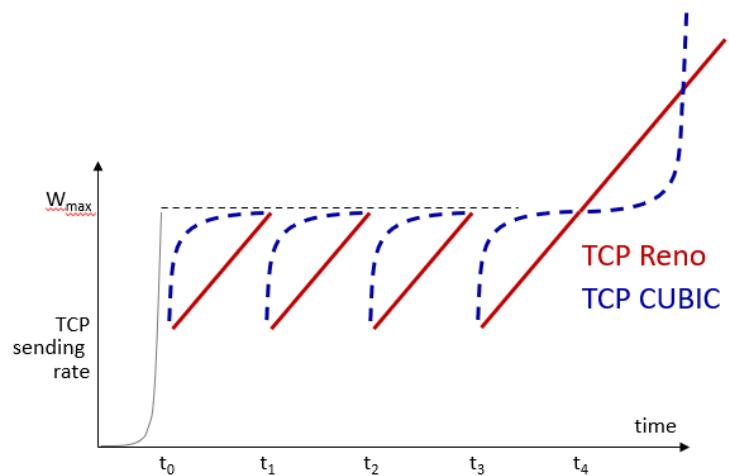
## 7. TCP CUBIC: bottleneck link의 congestion 상태가 많이 변하지 않는다는 사상에 기초해 만들어진 최신 congestion control

- W<sub>max</sub>: congestion loss가 발견된 시점의 전송 속도
- W<sub>max</sub>/2에서 W<sub>max</sub>로 가는 동안 처음에는 속도를 빠르게 하지만 W<sub>max</sub>에 가까워질수록 속도를 느리게 한다



### C. 좀 더 자세한 설명

- K. TCP window size가 W<sub>max</sub>에 reach할 거라 예상되는 시간
- K와 멀수록 빠르게 가까울 수록 느리게 전송 속도를 증가 시켜서 3 차함수(cube) 형태를 띠기에 CUBIC이라고 한다



## Ch4. Network layer: Data Plane

1. Network layer 개괄: segment를 송신자에게서 수신자로 보내는 역할을 한다

### A. Two key network layer functions

- i. Forwarding: packet들을 router의 input link에서 '적절한' output link로 보내는 것
- ii. Routing: packet의 source-dest 루트를 결정하는 작업
- iii. 결론: packet의 이동 경로를 계획(routing)하고 이동(forwarding)시키는 역할

### B. Data plane, Control plane으로 나누어 본 network layer

#### i. Data plane

1. Local한 Per-router 기능을 한다.
2. Router에 도착한 datagram을 어떻게 input port에서 output port로 forwarding하는지 결정한다

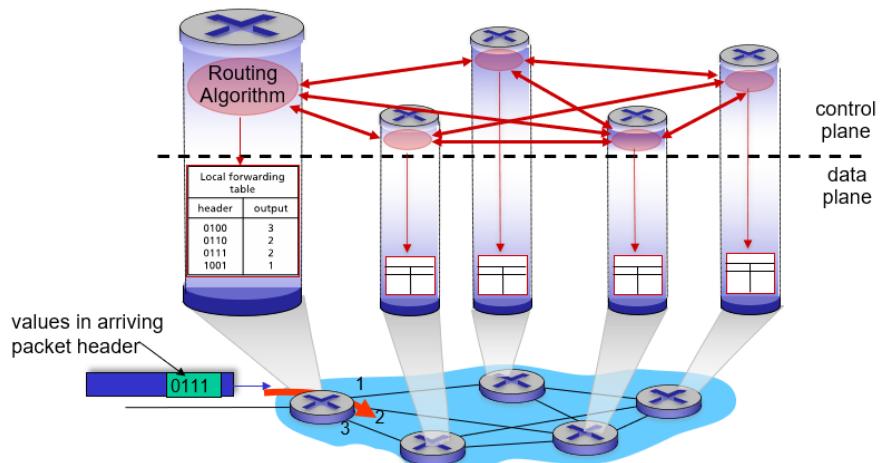
#### ii. Control plane

1. Network-wide한 기능을 한다.
2. Datagram이 source-dest간의 path에서 어떻게 routing될지 정한다
3. Control-plane의 접근법

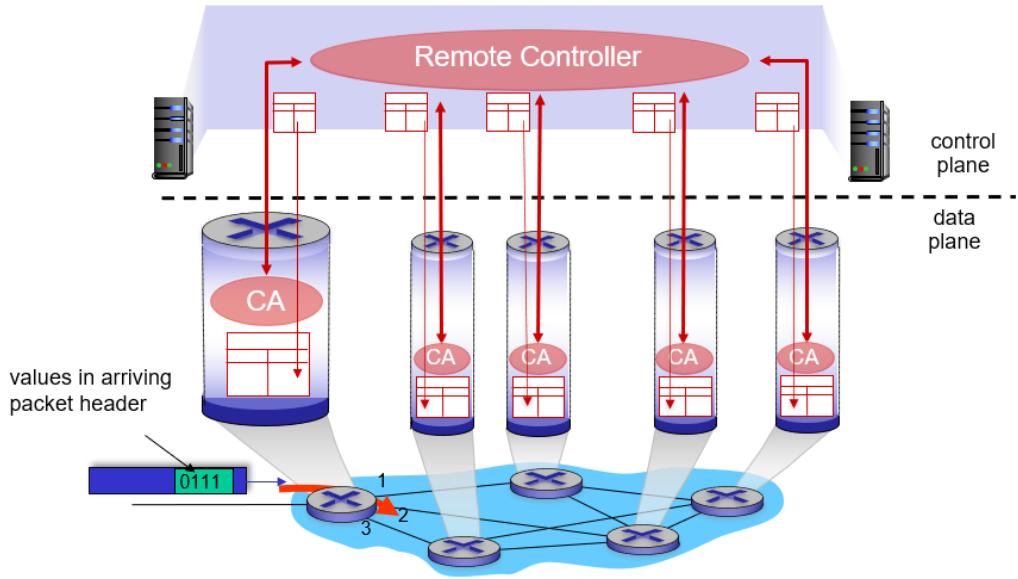
##### A. Traditional routing algorithms: router에서 행해진다

##### B. Software-defined networking(SDN): 원격 서버에서 행해진다

#### 4. Per-router vs SDN in control plane



각각의 모든 router에서 individual한 routing algorithm이 상호작용하며 route한다



원격 서버에서 계산한 forwarding table을 router들이 받아쓴다

- A. SDN의 장점: 중앙 제어의 양상을 띠기 때문에 제어가 쉽다
- C. Network service model: network layer은 guaranteed delivery나 timing등 여러 기능을 제공할 수 있지만 현재 사용중인 internet에서는 제공하지 않고 있다. 왜그럴까?
  - i. Best-effort model: 현재 가장 널리 사용되는 모델로 '열심히는 하지만 결국 보장은 못한다'는 모델이다

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no

Internet “best effort” service model

No guarantees on:

- i. successful datagram delivery to destination
- ii. timing or order of delivery
- iii. bandwidth available to end-end flow

## ii. 다른 network service model들

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

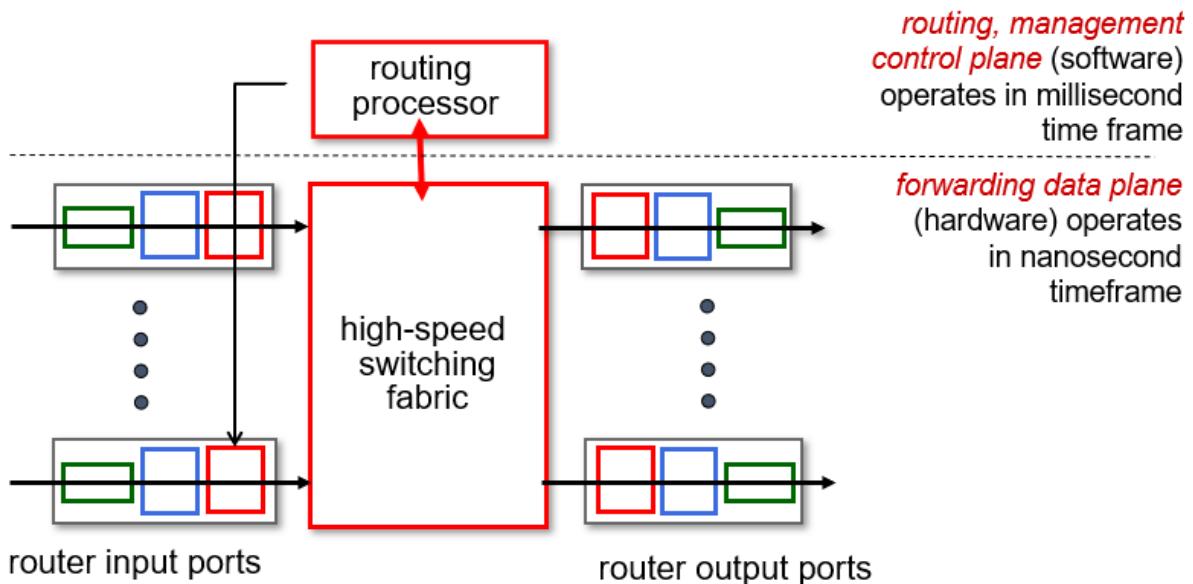
1. Best effort: Internet에서 쓰이는 모델로 아무것도 보장하지 못한다
2. CBR (Constant Bit Rate): ATM에서 쓰이는 모델로 일정한 bandwidth으로 보내며 loss, order, timing을 보장해준다
3. ABR (Available Bit Rate): ATM에서 쓰이는 모델로 최소 bandwidth 이상을 사용할 수 있지만 최소 bandwidth을 넘어가면 loss가 생길 수 있으며 order는 보장하지만 timing은 보장하지 않는다
4. Intserv Guaranteed: Internet에서 쓰이는 모델로 signaling(데이터가 전송될 수 있도록 통신 상태를 알리고 정확한 경로를 만드는 것)을 베이스로 한 기술로 service, resource를 reservation해서 QoS를 보장하는 기술이다. 좋은 기술이지만 오버헤드가 크다
5. Diffserv: Internet에서 쓰이는 모델로 앞의 Intserv Guaranteed의 오버헤드를 해결하기 위해 나온 서비스이다. 서비스 품질을 차별화해서 guarantee는 못 하지만 어느정도 좋은 서비스를 지원하는 모델이다

## iii. 왜 좋은 서비스 모델을 두고 best effort를 사용하는가?

1. Simplicity of mechanism: Internet은 단순한 구조로 인해 넓게 사용되었기 때문에 guarantee는 가능하지만 복잡하게 control해줘야 하는 서비스 모델을 사용하지 않는다
2. Sufficient provisioning of bandwidth: 기술이 발전하면서 bandwidth 확보가 용이해졌고 그 결과 '대부분의 시간동안' '좋은 퍼포먼스'를 가지는 real-time application을 사용할 수 있기 때문이다
3. Replicated, application-layer distributed services: client 네트워크에 가깝게 연결된 datacenter, CDN으로 인해 서비스 제공이 쉬워졌다
4. Elastic한 서비스의 congestion control이 작동을 도와주기 때문이다

## 2. Router은 어떻게 작동할까?

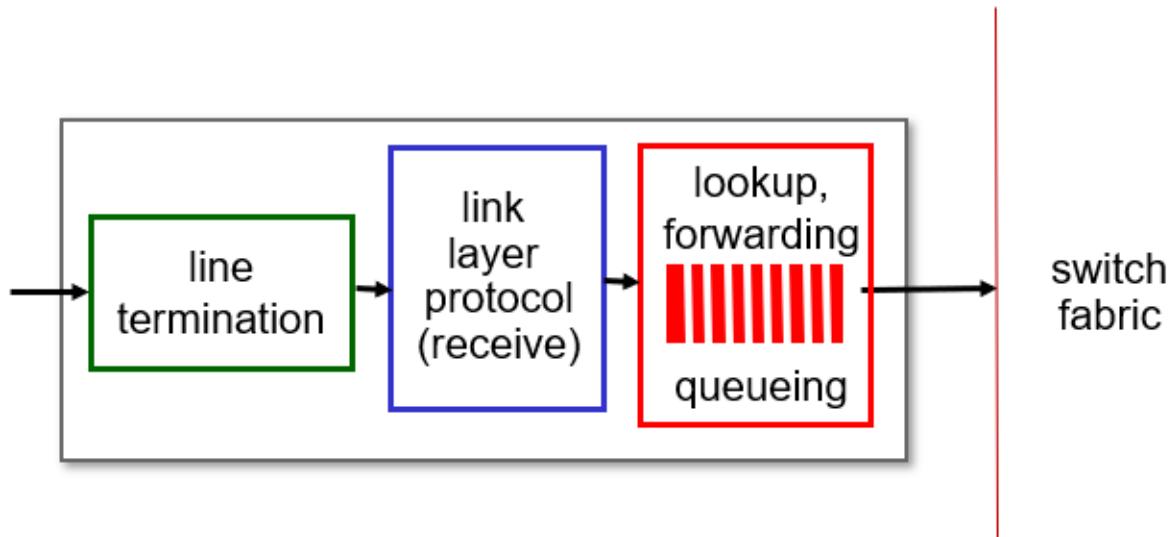
### A. Router architecture 개괄



Router는 여러 input port로 들어오는 데이터를 output port로 전달해줘야 하기 때문에 router 성능은 input port 처리량인 링크의 성능이 아니라 input port에서 output port로 얼마나 빨리 전달해주는지를 성능으로 간주한다

- i. Router는 여러 개의 network interface와 그 안의 memory, 그리고 cpu를 가진 임베디드 시스템이다

## B. Router input port



- i. Line termination: physical-layer의 bit-level 신호를 수신한다
- ii. Link layer protocol: link-layer 프로토콜을 써서 자신에게 수신된 것이 맞으면 상의 계층으로 보낸다
- iii. Decentralized switching
  - 1. Header field 값들을 이용하여 input port memory에 있는 forwarding table에 있는 output port를 찾는다. (match plus action)
  - 2. 목표: input port에 오는 data들을 line speed와 같거나 더 빨리 처리하는 것
  - 3. Input port queuing: 위의 line speed보다 switch fabric으로 forwarding하는 처리 속도가 느릴 경우 datagram들이 쌓여서 생기는 queue
  - 4. 다양한 forwarding
    - A. Destination-based forwarding: destination IP address에 따라서만 forwarding하는 방식(traditional)
    - B. Generalized forwarding: header field 값들의 모든 field를 base로 할 수 있는 forwarding 방식

- C. Destination-based forwarding: forwarding table에 IP 주소 하나마다 link interface(output port)를 할당하면 forwarding table이 너무 커지기 때문에 일정 범위를 한 link interface에 할당한다

forwarding table	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

- D. Longest prefix matching: forwarding table에서 주어진 dest address가 어디로 가야하는지 찾는다면 dest address와 가장 긴 prefix가 일치하는 것을 matching시켜준다

Destination Address Range	Link interface
11001000 00010111 00010*** ******	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011** ******	2
otherwise	3

- i. 위에서 1과 0으로 구성된 부분이 prefix이며 \*는 don't care를 뜻하는 것이다.
- ii. 예를 들어 11001000 00010111 00011000 11111111을 matching시킨다면 link interface 1, 2 둘 다 prefix는 일치하지만 1번이 longest prefix라서 1번으로 보낸다
- iii. 왜 longest prefix matching을 쓰는가: 하드웨어와 관련이 있다
  1. Ternary content addressable memories(TCAMS): forwarding table에는 0과 1 뿐만이 아니라 don't care 항인 \*도 표현해야 되기에 ternary한 특수 메모리를 만들어서 쓴다
  2. TCAM에서 address를 검색할 때 table size와 관계없이 한 사이클만에 결과를 알려줘야 되기에 이런 방법을 쓴다

E. Switching fabrics: input link에서 output link로 packet을 옮기는 부분

- i. Switching rate: input에서 output으로 packet을 옮기는 속도
- ii. R의 line rate를 가진 N개의 input port를 가진 router에서 이상적인 switching fabric은 NR의 switching rate를 가진 router이다(Non-blocking router). 하지만 비용 문제로 후술할 여러 방법으로 적은 switching rate로도 가용할 수 있게 한다
- iii. Switching fabric의 주요 3타입

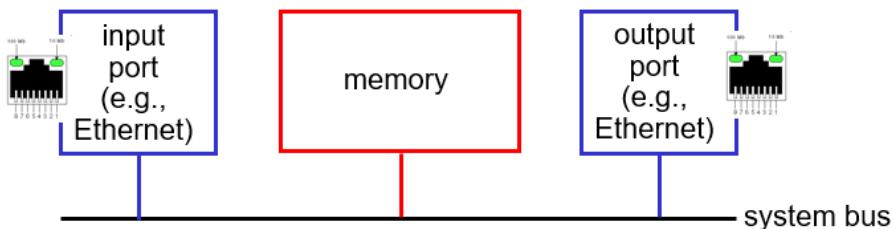
1. Switching via memory: 가장 traditional한 방법으로 컴퓨터에 여러 networking interface를 연결하여 CPU의 컨트롤에 따라 datagram을 전달

A. 작동 방식

- i. Input port에서 system bus를 통해 memory에 datagram을 복사
- ii. CPU에서 알려준 목적지를 토대로 system bus를 통해 memory에서 output port로 복사

B. 특징

- i. 한 datagram을 옮기는데 2번의 bus crossing이 필요하다
- ii. 최대속도가 memory bandwidth에 비례한다.
  1. memory에서 초당 B의 bandwidth를 가졌다면 input port에서 output port로 보내는 속도는  $B/2$ 이하여야 한다. (추측. Memory에 쓰는 것과 memory에서 보내는 것을 둘 다 포함하기 때문이다)
- iii. 요즘도 쓰이는 방식이기는 하지만 요즘 쓰는 방식에서는 input port에서 dest address를 찾아서 보낸다(추측. Input port에서 라우팅 프로세서를 쓰고 switching할 때는 라우팅 프로세서가 개입 안함)



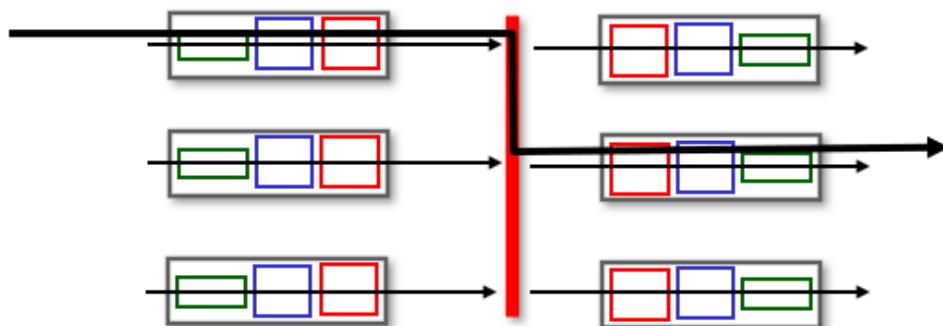
2. Switching via a bus: 공유 버스를 통해 라우팅 프로세서(CPU) 개입 없이 input port datagram을 output port로 이동(라우팅 프로세서의 개입이 없다는 말은 switching 할때 없다는 거지 input port에서 개입이 없다는게 아니다)

#### A. 작동 방식

- i. Input port에서 header에 이미 계산한 dest address를 담아서 공유 버스로 보낸다
- ii. 공유 버스는 모든 output link로 datagram을 보내고 output link에서는 datagram에 있는 dest address가 자신일 경우에는 받고 아니면 폐기한다
- iii. 받은 output link에서는 input link에서 받은 datagram header에 있는 dest address를 없애줘야 한다(이 switching에서만 쓸거기 때문이다)

#### B. 특징

- i. 한 input port가 공유 버스를 쓰는 동안 다른 input port는 공유 버스를 사용할 수 없다
- ii. Bus contention: 속도가 bus bandwidth로 제한된다



3. Switching via interconnection network: 각 포트마다 bus를 만들고 이들을 상호 연결시켜 병렬적으로 전송이 가능하도록 만든 방식(이 interconnection 방식 외에는 switching via a bus와 같다)

#### A. 작동 방식

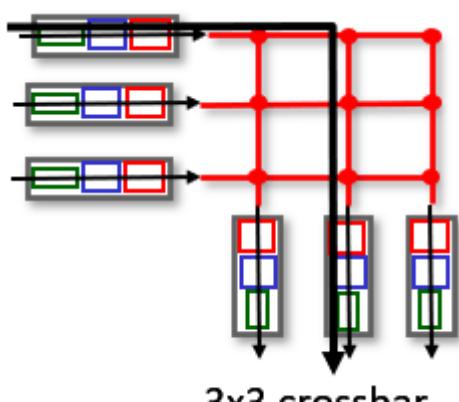
- i. 현재 free한 Input port(만약 그 input port에서 다른 datagram을 전송하고 있었다면 기다려야 한다)에서 bus로 전달될 때 스위치 구조 컨트롤러에서 input port에서 output port로 가도록 교차로 교차점

을 열거나 닫아서 길을 만들어 준다

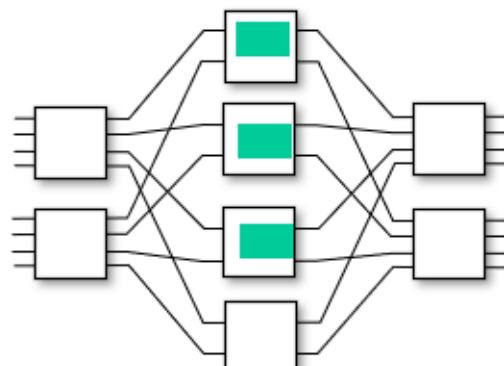
- ii. 전달된 후에는 switching via a bus처럼 datagram header에서 dest link 정보를 지워야 한다

## B. 특징

- i. 각각 n개의 input/output link가 있는 router라면 각각의 link에 연결된 2n개의 버스들이 interconnected된 구조이다
- ii. Multistage switch:  $n \times n$  스위치를 작은 여러 개의 스위치로 나누어 구성하여 여러가지 이점을 얻을 수 있다
  1. 다른 input link에서 같은 output link로 향하게 할 수 있다 (Switching via a bus는 무조건 한번에 하나만 보낼 수 있지만 interconnection network는 어느정도 switching 속도만 빠르면 언제든지 다른 input link에서 같은 output link로 보낼 수 있다)
  2. Router 안에 작은 switch로 나누어 한 switching fabric이 가지는 traffic이 작아져서 전체 Router의 switching 용량을 늘릴 수 있다(작은 switch는 적은 input port를 가질 것이기 때문에 그 안의 switching fabric은 non-blocking 스위칭처럼 사용 가능할 것이다)
- iii. Exploiting parallelism: datagram을 fixed된 length의 cell로 나누어 보낸 뒤 output link에서 다시 합칠 수 있도록 하는 기능



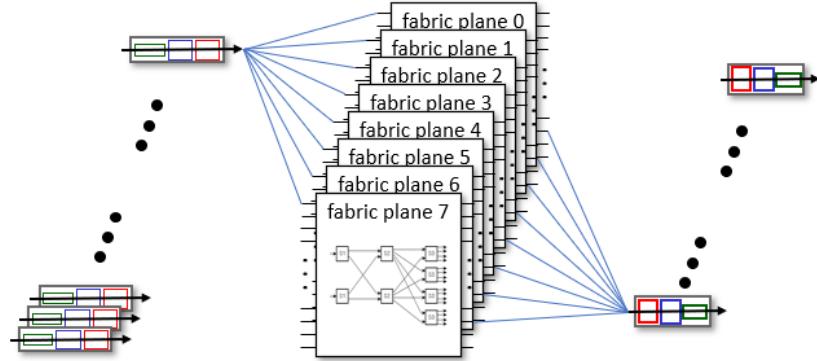
3x3 crossbar



8x8 multistage switch  
built from smaller-sized switches

Input/output link가 각각 3개라서 6개의 bus

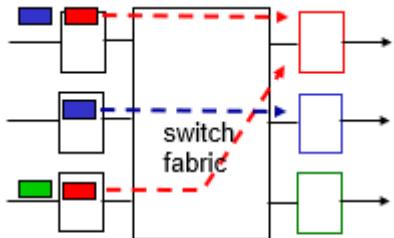
Datagram을 3개로 나누어 보낼 수 있게 하는  
multistage switch와 exploiting parallelism



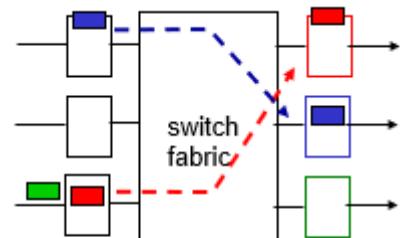
3단계 multistage switch를 이용하여 switching capacity를 높여주는 구조

iv. Queuing

1. Input port queuing: switch fabric이 input port들의 총 속도보다 느리면 input queue들에 queuing하게 된다
  - A. Queuing delay와 input buffer overflow로 인한 loss가 생길 수 있다
  - B. Head-of-the-Line(HOL) blocking: queue 안의 앞 datagram에서 queuing이 생기면 뒤의 datagram이 보낼 수 있는 상황이더라도 보낼 수 없게 된다

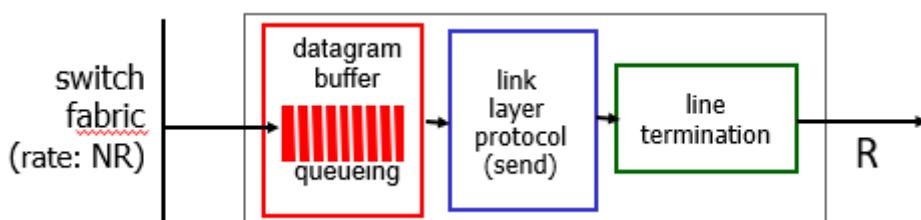


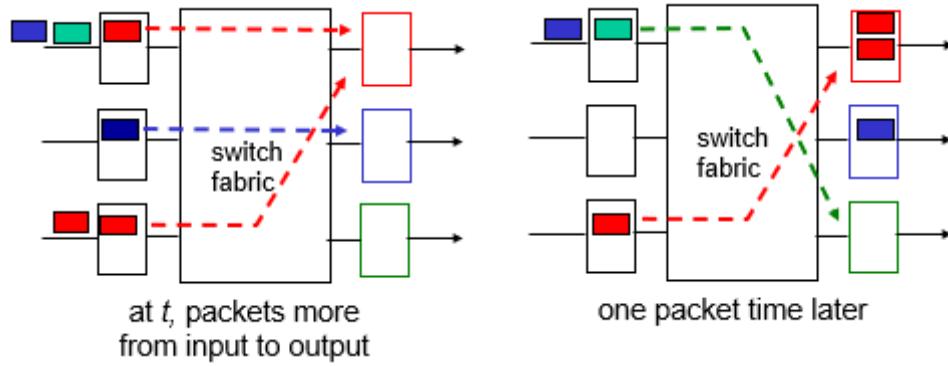
output port contention: only one red datagram can be transferred. lower red packet is **blocked**



one packet time later: green packet experiences HOL blocking

2. Output port queuing: switch fabric에서 output port에서 처리할 수 있는 것보다 빨리 보내주면 output queue에 쌓이게 되고 그 결과 queue가 다 찼을 때 무엇을 drop해야 하는지인 drop 문제와 어떤 datagram부터 처리를 해줘야 할지인 scheduling discipline 문제가 생긴다





- A. Buffering: datagram은 congestion이나 buffer 부족으로 생길 수 있다
- B. Priority scheduling: 누가 서비스 받아야 할지는 network neutrality(망 중립성)와 관련 있다

### 3. Buffering

- A. 보통 어떻게 buffer 크기를 결정할까?

- i. RFC3439 rule of thumb:  $typical\ RTT * C$ ( $typical\ RTT$ 는 250msec이고  $C$ 는 link capacity이다)
- ii. 좀 더 최근에는  $N$ 개의 flow가 있을 때  $typical\ RTT * C / \sqrt{N}$ 으로 정한다

- B. 너무 큰 buffering은 delay를 증가시킨다

- i. Buffering이 커서 TCP가 response 받을 때까지 시간이 더 늦어짐
- ii. 다시 상기해야 한다: 'keep bottleneck link just full enough but no fuller.'

### 4. Buffering management

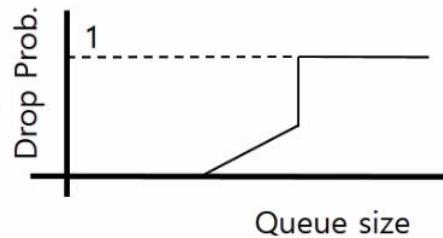
- A. Drop: queue가 꽉 찼을 때 또 들어오면 특정 packet을 drop한다

- i. Tail drop: 마지막으로 도착한 packet drop

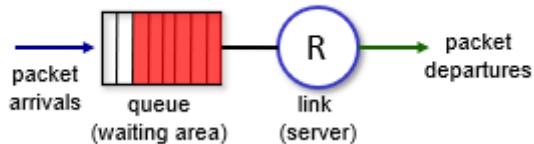
- ii. Priority: 특정 기준에 따라 drop할 packet을 정한다

- B. Marking: queue가 다 차기 전에 ECN이나 RED처럼 client한테 congestion함을 알리는 방법(이 경우 client는 보내는 속도를 조절하게 됨)

## Probabilistic dropping (Random Early Detection)

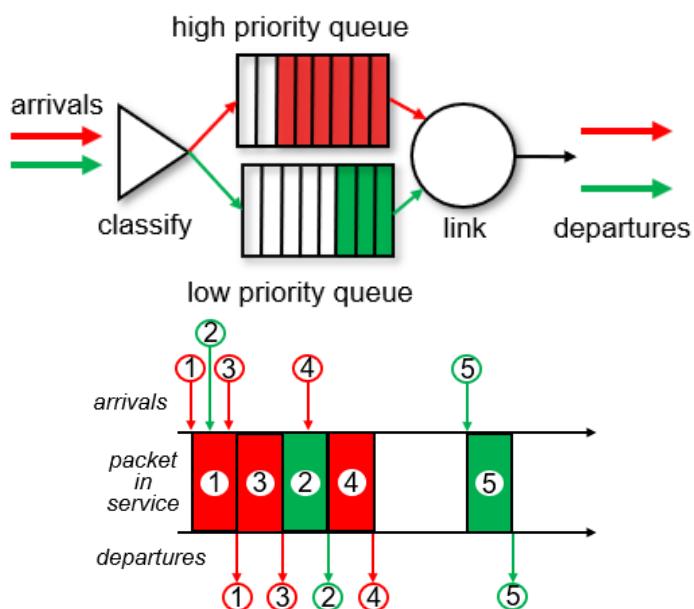


### Abstraction: queue



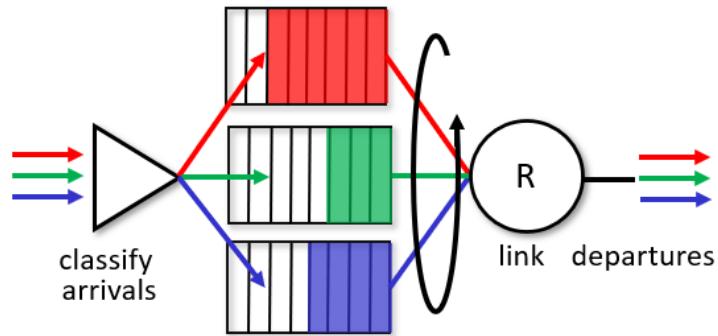
5. Scheduling policy: 도착한 packet들을 어떤 방식으로 service할지 정하는 규칙

- A. First come, First served(FCFS): 먼저 온 것을 먼저 서비스한다
- B. Priority: 들어오는 traffic들을 classify해서 priority가 높은 class의 traffic을 먼저 처리하는 방식
  - i. Header field의 어떤 값을 classification을 위해 사용할 수 있다
  - ii. 가장 높은 priority의 class queue를 가장 먼저 처리하며, 같은 class 안에서는 FCFS가 적용된다
  - iii. 낮은 queue의 traffic을 처리하는 중에 높은 queue의 traffic이 도착하면 그 경우에는 현재 처리 중이던 traffic 처리 후에 높은 queue를 처리한다



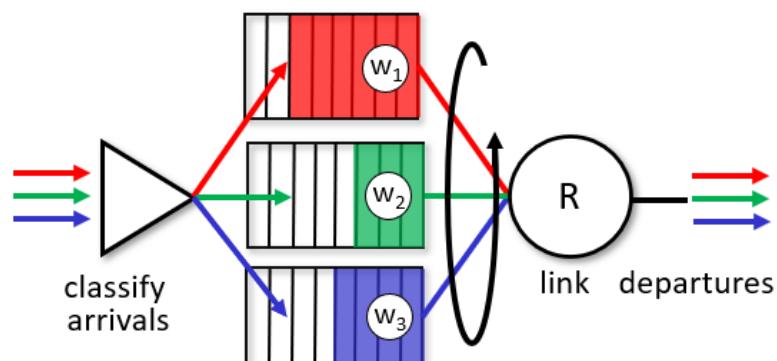
C. Round robin(RR): 도착하는 traffic을 classify하여 queue에 저장하되 우선순위는 없이 한번씩 돌아가면서 class의 traffic을 처리한다

- i. Link(server)는 cyclically, repeatedly class queue들을 스캔하여 다음 순서인 class에서 가능하면(class에 traffic이 있다면) traffic을 처리한다



D. Weighted fair queuing(WFQ): RR의 general한 버전으로 각 클래스 i weight  $w_i$ 가 있으며 거기에 따라 서비스한다

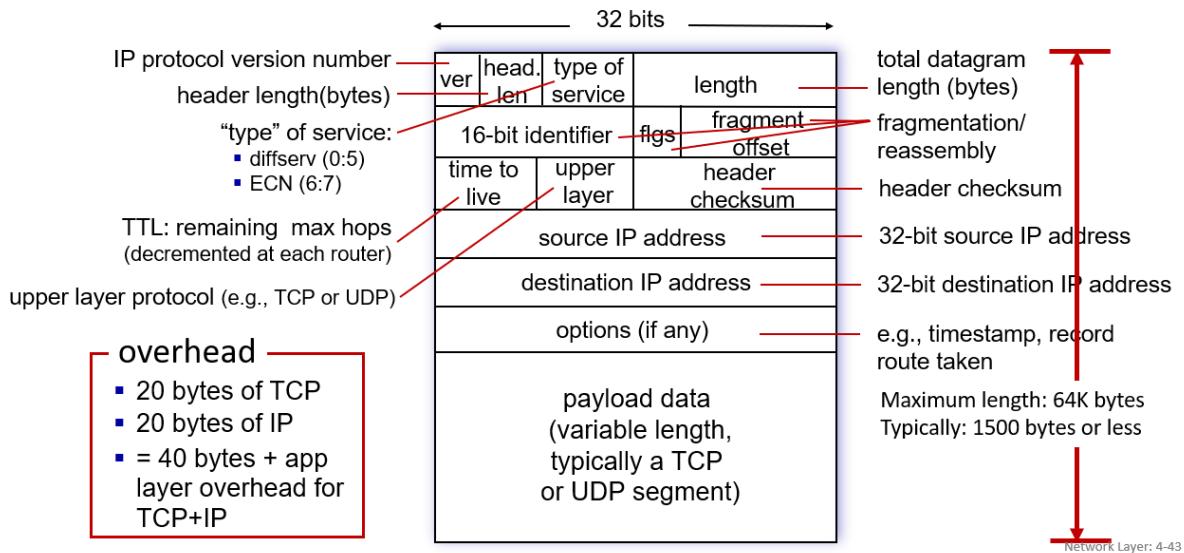
- i.  $W_i$ 의 weight를 가진 class는  $w_i / \sum w_j$  만큼 서비스 받는다(다만 분모의 저는 현재 class에 traffic이 있는 class의 weight만이다)
- ii. 이렇게 처리하면 적어도 minimum bandwidth guarantee가 된다



6. Network neutrality(망 중립성): 인터넷은 모든 사람들이 쓰는 것이기 때문에 서비스를 제공하는 ISP라도 마음대로 policy를 만들어선 안되며 중립적으로 운영해야된다는 말이다

## F. IP: Internet Protocol

- i. Datagram format: 보통 IP Datagram의 header 크기는 20 바이트로 가끔씩 options field가 있으면 header가 늘어난다(최대 60바이트까지)



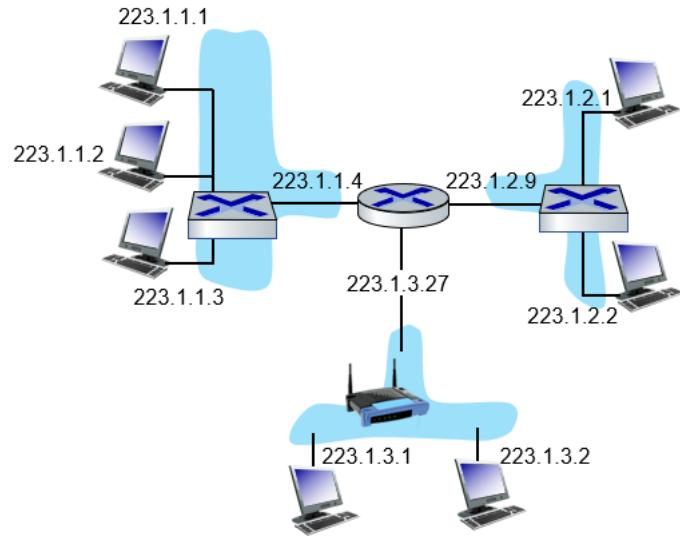
1. Ver: IP버전(IPv4인지 IPv6인지)를 알려주며 4비트
2. Head len: header의 길이를 알려주며 4비트
  - A. Head len의 기본 단위는 4바이트이기에 datagram header 최대 길이는  $4 \times 15 = 60$ , 60바이트다
3. Type of service(TOS): datagram과 관련된 DS(Differentiated Service, 차등 서비스, network service model에서 본 것) 6비트+ECN(Explicit Congestion Notification) 2비트로 총 8비트로 구성되어 있다
  - A. DS: 해당 datagram에 요구되는 서비스 질에 대한 유형을 나타냄
    - i. 상위 3비트
      1. 000: best effort(default)
      2. 000-111: 숫자가 높아질수록 우선도가 높아진다
    - ii. 하위 3비트: 앞의 2비트는 폐기 확률이고 마지막은 모르겠음
  - B. ECN: 혼잡 알림을 위해 사용됨
    - i. 00: ECN 기능을 사용하지 않음
    - ii. 01 또는 10: ECN 기능을 사용함

iii. 11: router에서 혼잡 발생을 알리는 표식

- C. Length: IP datagram의 총 길이,  $2^{16}$ 이므로 대략 64K바이트가 datagram의 최대 크기지만 대부분의 datagram은 1500바이트 이하이다.  
16비트
- D. 16-bit identifier: datagram fragmentation(패킷 분할)이 일어났을 경우 이 field를 통해 원본 datagram을 알 수 있다 16비트
- E. Flags: datagram이 분할되었는지 여부 3비트
- F. Fragment offset: 조각난 datagram의 부분을 가리키는 데 사용 13비트
- G. Time to live(TTL): routing에서 루프를 막기 위해 packet의 현재 남아 있는 최대 hop 수를 알려주는 field. 한 hop 이동할 때마다 1씩 줄이며 TTL이 0인 packet을 받으면 router는 packet을 폐기한다 8비트
- H. Upper layer: IP를 이용하는 상위계층 프로토콜(TCP나 UDP같은 transport layer protocol)이 무엇인지 알려주는 field. 8비트
- I. Header checksum: datagram이 전송되는 동안 내용 변조가 있었는지 알아보기 위한 field로 1의 보수로 계산한다. 매 router마다 TTL등이 바뀌기 때문에 매번 새로 계산해야 하며 이는 overhead의 원인이 되기도 한다. 16비트
- J. Source IP address: 송신자의 IP 주소 16비트
- K. Destination IP address: 수신자의 IP 주소 16비트
- L. Options: 꼭 들어가는 field가 아니며 이로 인해 IP의 header크기가 일정하지 않아(20바이트-60바이트) overhead의 원인이 된다

ii. IP addressing

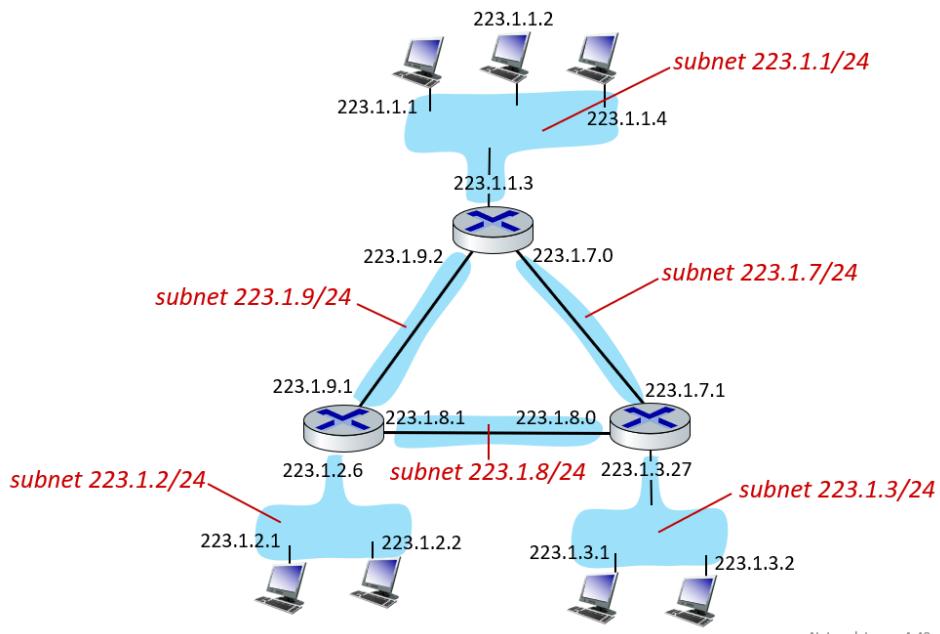
- 1. IP address: host/router의 interface와 연관된 32비트 identifier
  - i. Dotted-decimal IP address notation:  $223.1.1.1 = 11011111\ 00000001\ 00000001\ 00000001$
- 2. Interface: host/router와 physical link간의 connection
  - A. Router는 일반적으로 여러 개의 interface들을 가지고 있다
  - B. host들은 보통 1-2개의 interface를 가지고 있다



라우터는 3개의 interface, host들은 1개의 interface를 가지고 있다  
 파란색은 subnet이라는 집합이며 위의 subnet은 Ethernet switch로,  
 밑의 subnet은 WiFi base station으로 연결되어 있다

### 3. subnet

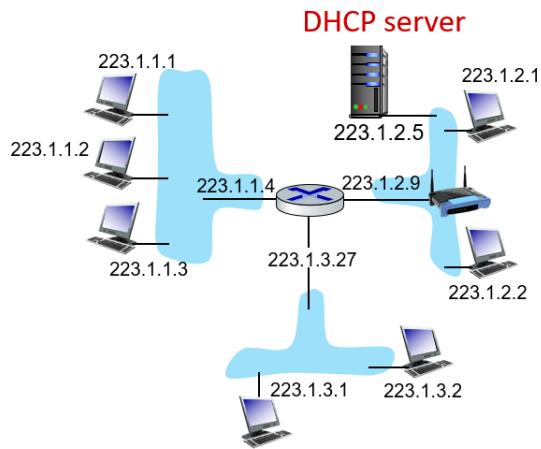
- A. subnet이란: 각각의 interface들을 host/router에서 떼어냈을 때 형성하는 isolated network를 말하는 것이다. (위의 그림에서 파란 부분처럼)



Network Layer: 4-49

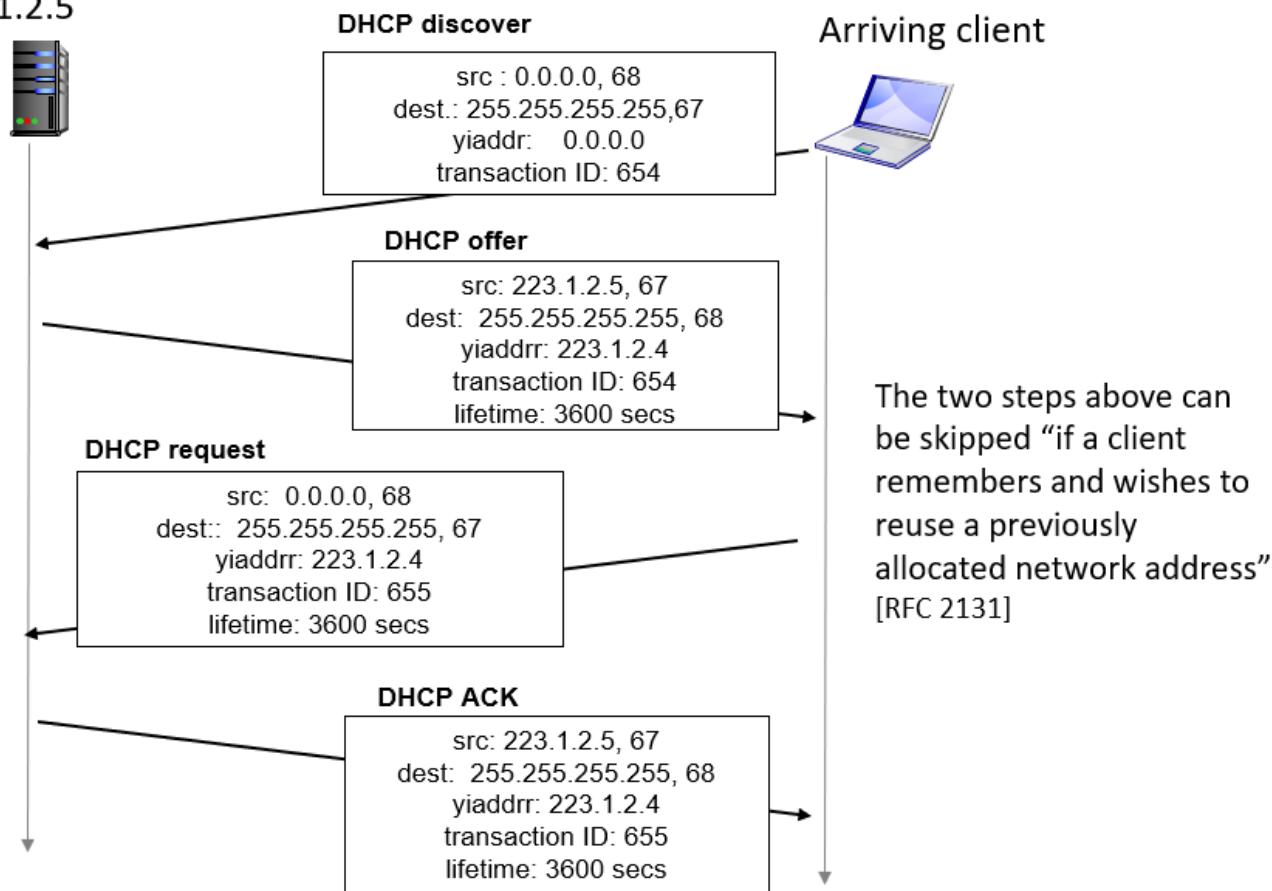
- B. subnet mask: 같은 subnet을 봤을 때 n번째 상위비트가 같은데 그 경우 n번째 비트까지 dotted-decimal IP address notation으로 a.b.c.d/n으로 표시하는데 이를 subnet mask라고 한다

- C. 브로드캐스트: subnet 안의 모든 host에게 전달되도록 하는 것으로 dest IP address를 255.255.255.255로 하면 된다
- D. Classless InterDomain Routing (CIDR)
  - i. classful addressing: 기존의 방식으로 a.b.c.d/n에서 n이 무조건 8, 16, 24로 각각 A, B, C클래스로 subnet이 분류되어야 한다. 하지만 이 경우  $2^{4+1}$ 개의 interface를 가진 subnet은 B타입이나  $2^8 - (2^{4+1})$ 개의 IP주소는 free하게 두기 때문에 낭비가 많았다
  - ii. CIDR: a.b.c.d/n에서 n이 임의의 수가 될 수 있으며 200.23.16.0/23의 경우에는 상의 23비트를 실제 address로 갖는다
- E. 어떻게 IP address를 얻어오는가: host 입장에서 IP address를 얻으며 network(subnet) 입장에서 IP address를 얻는가?
  - i. Host는 어떻게 IP address를 얻는가?
    1. 직접 설정한다
    2. Dynamic Host Configuration Protocol (DHCP): server에서 동적으로 address를 할당 받는다
  - A. 장점
    - i. 동적으로 할당하기 때문에 address를 재사용 가능하다 (host가 연결된 동안만 address를 가지고 있다)
    - ii. 할당 받고 사용되지 않는 주소가 없도록 할 수 있다
    - iii. mobile처럼 이동식 디바이스가 많아진 현대의 network에 join/leave하기 쉽다
  - 3. DHCP client server scenario: DHCP client가 어떤 network에 join 할 때 address가 필요하다



- A. 보통 DHCP server는 특정 router에 연결되어 있어서 router에 연결된 subnet들의 address를 관리한다. 또는 특정 subnet에 붙어 있기도 하다. 하지만 어느경우든 router는 DHCP server가 어디 있는지 알려줄 수 있어야 한다

DHCP server: 223.1.2.5



이 예시에서 DHCP server는 subnet에서 접속 가능하다고 가정한다(브로드캐스팅하면 찾을 수 있다)

Yiaddr field는 새롭게 도착한 client에게 줄 IP address를 뜻한다

## B. DHCP protocol 4단계

- i. DHCP discover[optional]: 새롭게 network에 접속한 client는 address를 얻어야 하지만 network의 IP 주소도 network의 DHCP server 주소도 모르기 때문에 dest address를 255.255.255.255(브로드캐스팅)로 설정하고 source address를 0.0.0.0으로 설정하여 UDP를 통해 브로드캐스트한다. 또한 DHCP discover은 transaction ID라는 field를 통해 구분한다
- ii. DHCP offer[optional]: host의 DHCP discover을 받은 DHCP server가 transaction ID, yiaddr, networkmask, lifetime(IP주소가 유효한 기간)을 담아서 브로드캐스팅 한다. (브로드캐스팅하는 이유는 subnet에 여러 DHCP server가 있을 수 있고 client는 여러 DHCP server중에서 최적의 DHCP server를 선택하라고 하는 것이다)
- iii. DHCP request: client가 받은 DHCP offer 중에서 최적의 DHCP server를 골라 그 DHCP offer의 정보를 담아 자신이 이 IP address를 쓰겠다고 이야기하는 브로드캐스팅이다
- iv. DHCP ack: DHCP server에서는 DHCP request에 대해 ack를 보낸다

## C. DHCP protocol 4단계의 특징

- i. Client가 이전에 할당 받은 network address를 기억하고 재사용하기 원한다면 1, 2단계는 skip할 수 있다
- ii. 모든 단계에서 브로드캐스팅한다

## D. DHCP protocol의 추가 정보 제공: IP address이외에도 client의 network 접속에 도움되는 정보를 제공한다

- i. client에서 first-hop router의 address(client가 어디서 router의 기능을 제공받을지 알려주기 위해)
- ii. local DNS server의 name과 IP address(client가 쓸 local DNS를 알려주기 위해)
- iii. network mask

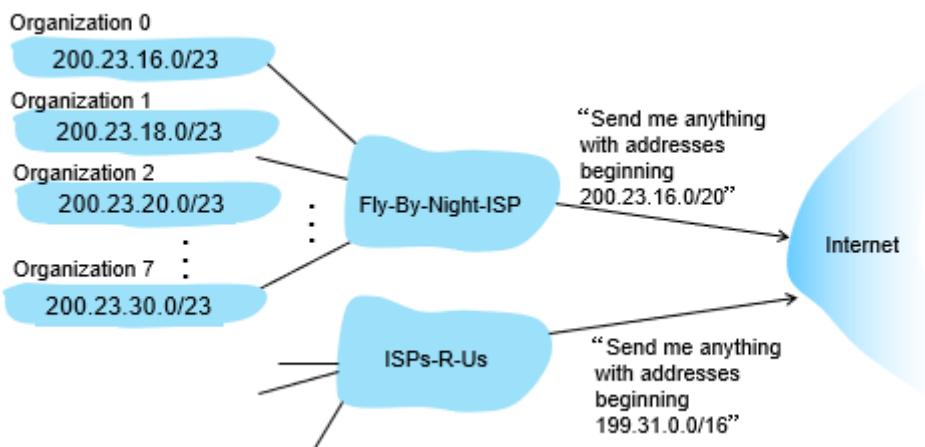
ii. Network는 어떻게 IP address를 얻는가?

- Network가 ISP에서 address space를 얻어오고 하위 network/host에게 address space의 특정 부분을 할당한다

A. 예시) ISP가 11001000 00010111 00010000 00000000  
 200.23.16.0/20를 할당 받았을 경우 하위 8 기관에 할당한다고 할 경우

Organization 0	<u>11001000 00010111 00010000 00000000</u>	200.23.16.0/23
Organization 1	<u>11001000 00010111 00010010 00000000</u>	200.23.18.0/23
Organization 2	<u>11001000 00010111 00010100 00000000</u>	200.23.20.0/23
...	...	...
Organization 7	<u>11001000 00010111 00011110 00000000</u>	200.23.30.0/23

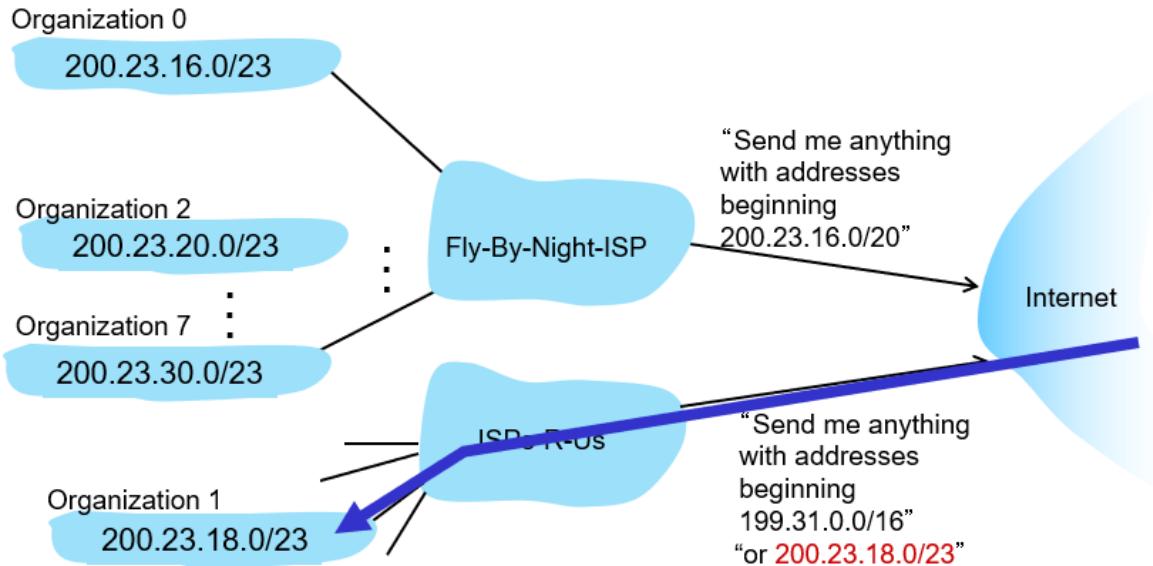
iii. Hierarchical addressing(route aggregation): 특정 ISP 하위에서 address space의 일부를 얻어가는 조직들은 subnet mask의 앞부분이 같으니(IPS subnet mask가 하위 조직 subnet mask들을 포함한다) Internet에 ISP의 subnet mask를 가지는 것이 있다면 ISP쪽으로 보내 달라고 할 수 있다



iv. Hierarchical addressing(more specific route): 만약 특정 ISP 산하의 조직 하나가 다른 ISP로 옮긴다면 어떻게 자기에게 오는 traffic들을 forwarding 받을까?

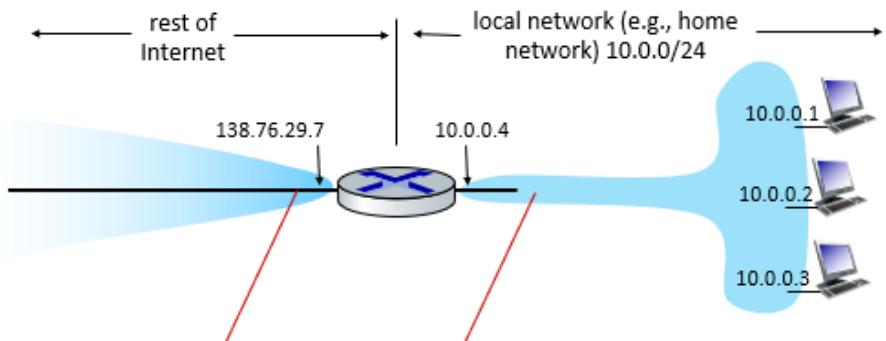
- 조직의 주소를 모두 새로운 ISP의 address space에 맡겨 고친다: 너무 노력이 많이 들어 효율적이지 않다
- ISP들이 자신의 subnet에 포함되지 않는 subnet mask도 함께 Internet에 알린다: 이 경우 앞에서 보았던 longest prefix matching기법 때문에 이전 ISP의 subnet보다 더 긴

prefix(subnet mask)를 가진 새로운 ISP로 옮긴 조직의 주소로 forwarding될 것이다



이전 ISP에서 200.23.16.0/20을 광고하고 현 ISP에서 200.23.18.0/23을 광고하고 이 경우 원래 organization 1으로 갈 traffic은 longest prefix를 가진 현 ISP쪽으로 forwarding된다

iii. Network address translation (NAT): Local network의 모든 장치가 하나의 IPv4 주소를 공유하여 주소를 효율적으로 사용하는 방식



*all datagrams leaving local network have same source NAT IP address: 138.76.29.7, but different source port numbers*

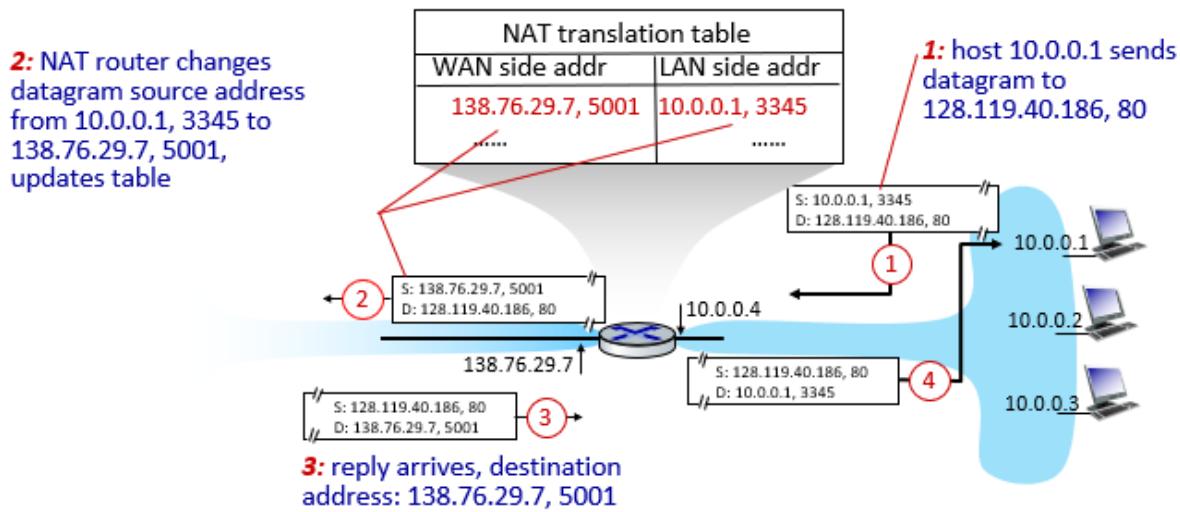
*datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)*

- 작동방식: local network의 모든 장치가 32-bit의 private IP address space(10/8, 172.16/16, 192.168/16 prefix)를 local network에서 사용하고 대외적으로는 하나의 IP로 통하고 port number로 구분하는 방식(port number은 16비트라  $2^{16} = 65536$ 만큼 동시 접속할 수 있다)

## 2. 장점

- 하나의 IP address로 모든 디바이스를 나타낼 수 있다

- B. Local network 안의 host 주소 변경을 외부에 알리지 않고 할 수 있다
- C. ISP를 바꿀 때 Local network 안 장비들의 IP address를 바꾸지 않아도 된다
- D. Local network 안의 장비들은 외부에서 직접적으로 접근할 수도, 볼 수도 없다



3. 작동: NAT router는 무조건 transparent해야 한다
  - A. Outgoing datagrams: 모든 outgoing datagram의 (Source IP address, port #)를 (NAT IP address, new port #)로 바꾼다
  - B. Remember(in NAT translation table): 모든 (Source IP address, port #)와 (NAT IP address, new port #) translation pair를 기억한다
  - C. Incoming datagram: 모든 incoming datagram의 (NAT IP address, new port #)를 NAT table에 있던 (Source IP address, port #)로 바꿔준다
4. NAT의 논쟁점
  - A. Router는 layer3의 정보만 써야하지만 NAT를 쓰기 위해서는 port number(transport layer의 정보)를 써야한다
  - B. IPv6로 주소 부족은 해결할 수 있다
  - C. Port number 등이 network layer에서 다뤄지기 때문에 end-to-end argument를 침해한다
  - D. NAT 뒤에서 server로 행동하는 client에게 접속할 방법이 없다
    - i. Server는 자주쓰는 port number를 가지고 대기하지만 밖에서는 저

server에 접속하기 위한 NAT에서의 port number를 모르기 때문

ii. NAT traversal이라는 해결책이 있다고 함

iv. IPv6: IPv4 주소는 모두 할당되었기에 새로운 address space를 만들었다

1. (optional)여러가지 casting

- A. 유니캐스트: 출발지와 목적지가 정확해야 하는 일대일 통신
- B. 브로드캐스트: 같은 네트워크에 있는 모든 장비에게 보내는 통신
- C. 멀티캐스트: 특정 그룹을 지정해서 해당 그룹원에게만 보내는 방식
- D. 애니캐스트: IPv6부터 가능하며 네트워크에 연결된 수신 가능한 노드 중에서 routing protocol 알고리즘에 따라 가장 가까이에 있다고 판단되는 node의 interface로 전달하는 방식
- E. 참조: <https://togl.tistory.com/42>

2. 추가 기능

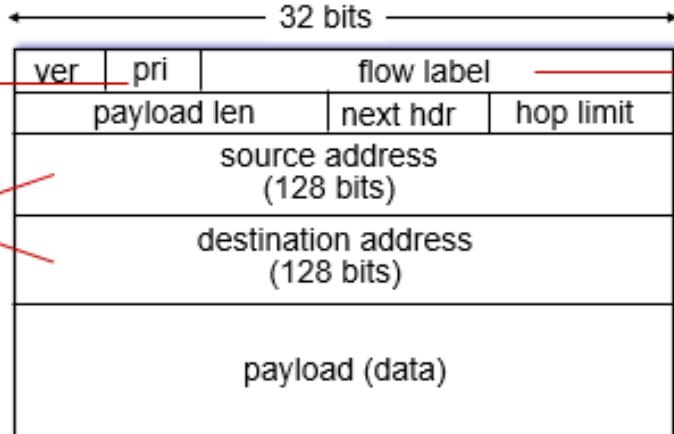
- A. Speed processing/forwarding: 40 byte로 고정된 header라서 처리 속도가 빨라짐(가변 길이가 아니기 때문)
- B. flow라는 개념을 추가하여 different network-layer treatment가 가능하다
  - i. flow: non-default 서비스 품질 서비스나 실시간 서비스 같은 특별한 처리를 요청하는 송신자에 대해 특정 흐름에 속하는 패킷 레이블링(실시간 처리가 필요한 서비스나 우선 순위를 가진 트래픽들을 보통 흐름으로 취급하는듯, 저자도 flow가 뭔지 제대로 정의되지 않았다고 하더라)

### 3. IPv6 Datagram Format

**priority:** identify priority among datagrams in flow

**128-bit**

IPv6 addresses



**flow label:** identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

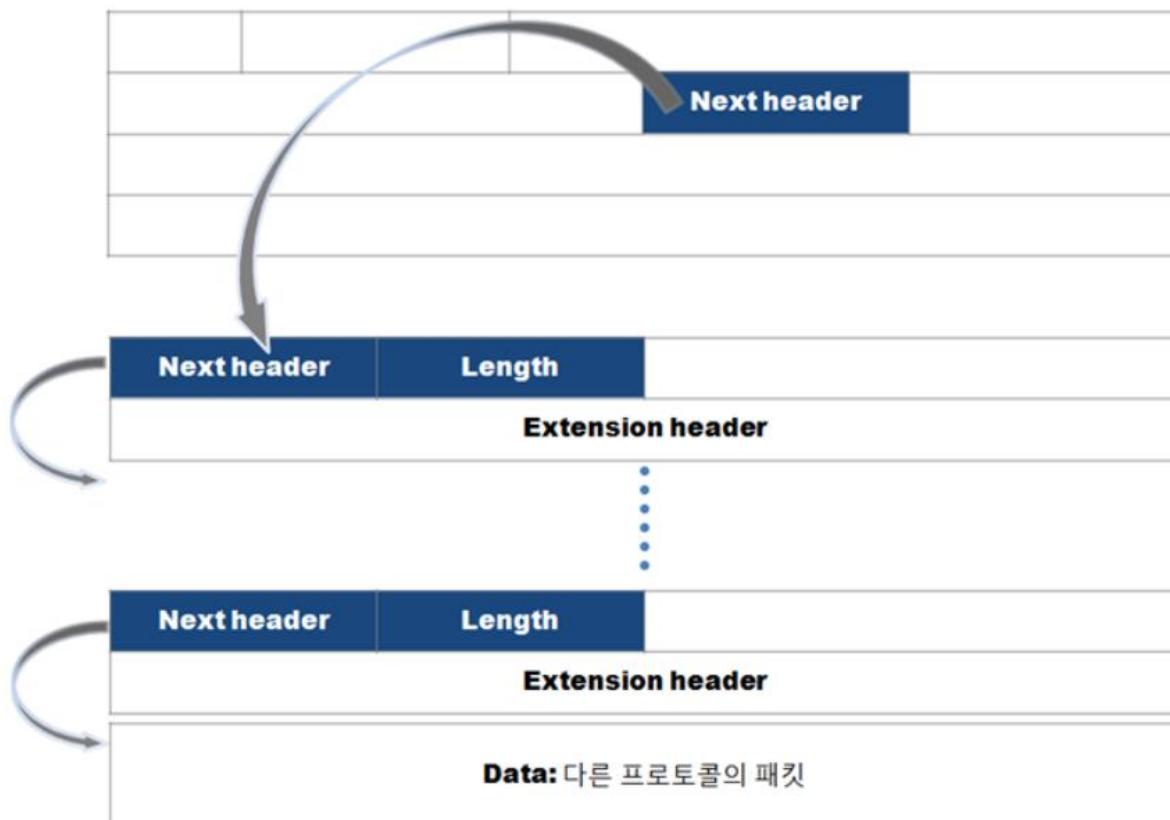
- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

Network Layer: 4-70

- A. Ver: IP 버전을 나타내며 값은 6이다(4로 바꾼다고 IPv4가 되는건 아니다)  
4비트
- B. Priority: 동시접속이 있을 시 패킷의 우선순위를 규정한다(IPv4의 TOS와 유사한 기능) 8비트
- C. Flow label: 같은 flow에 있는 datagram들을 식별한다 20비트
- D. Payload len: 헤더 영역을 제외한 나머지 부분의 길이를 바이트 단위로 나타낸다 unsigned integer이다 16비트
- E. Next hdr: (TCP, UDP) 또는 extension header pointer를 표시한다(IPv4의 protocol과 비슷하다) 8비트
- F. Hop limit: datagram의 현재 최대 이동할 수 있는 router 수, 한 번 전달 될 때마다 1씩 줄이고 hop limit이 0인 datagram을 받으면 router는 버린다(IPv4의 TTL과 비슷하다) 8비트
- G. Source address: 출발지의 IPv6 address 128비트
- H. Destination address: 도착지의 IPv6 address 128비트

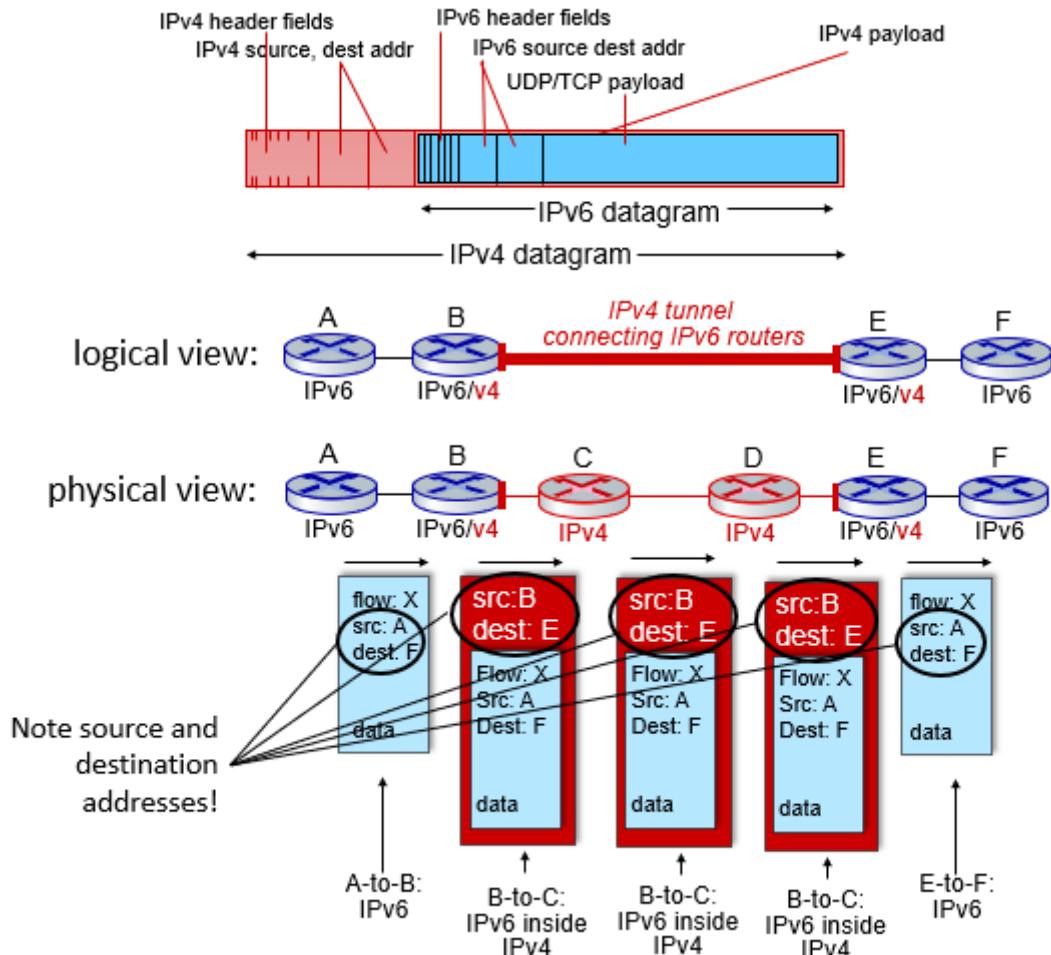
#### 4. IPv6와 IPv4의 차이점

- A. Fragmentation/reassembly: IPv6부터는 단편화/재결합을 출발지와 도착지에서만 할 수 있다. 만약 IPv6 datagram이 너무 커서 output link로 전달할 수 없으면 datagram을 폐기하고 'Packet Too Big'이라는 ICMP 오류 메시지를 송신자에게 보낸다. (단편화/재결합은 시간이 많이 걸리는 기능이었기에 end system에서만 하는 것이 IP 전달 속도를 증가시킨다)
- B. Header Checksum: Transport layer와 Link layer에서 checksum을 하기 때문에 IP 전달 속도를 증가시키기 위해 없앴다. (최근 network는 신뢰성이 높다는 믿음이 생겨서 오류가 생길 일이 적다고 생각하고 오류가 생기면 end system에서 drop하고 재전송하면 된다고 생각한다)
- C. Options: IPv4에서는 option field 때문에 가변 header 길이 때문에 처리 속도가 늦었다. 그래서 options에 들어갈 내용들을 extension header로 payload에 넣고 만약 extension header가 있으면 next hdr에서 다음 extension header이 어디있는지 알려주고 마지막 extension header에서 (TCP, UDP)를 정의한다



5. Transition from IPv4 to IPv6: 모든 IPv4가 갑자기 IPv6로 바뀔 수는 없기 때문에 IPv6가 어떻게 IPv4로 전달될 수 있는지 알아야 한다

- A. Tunneling: IPv6 datagram을 IPv4의 payload로 담아서 IPv4 router을 통과시킨다(실제로는 저렇게 IPv4 router을 지나지만 logical하게는 바로 IPv6 지원 router들끼리 연결된 걸로 한다)



## v. Generalized Forwarding, SDN

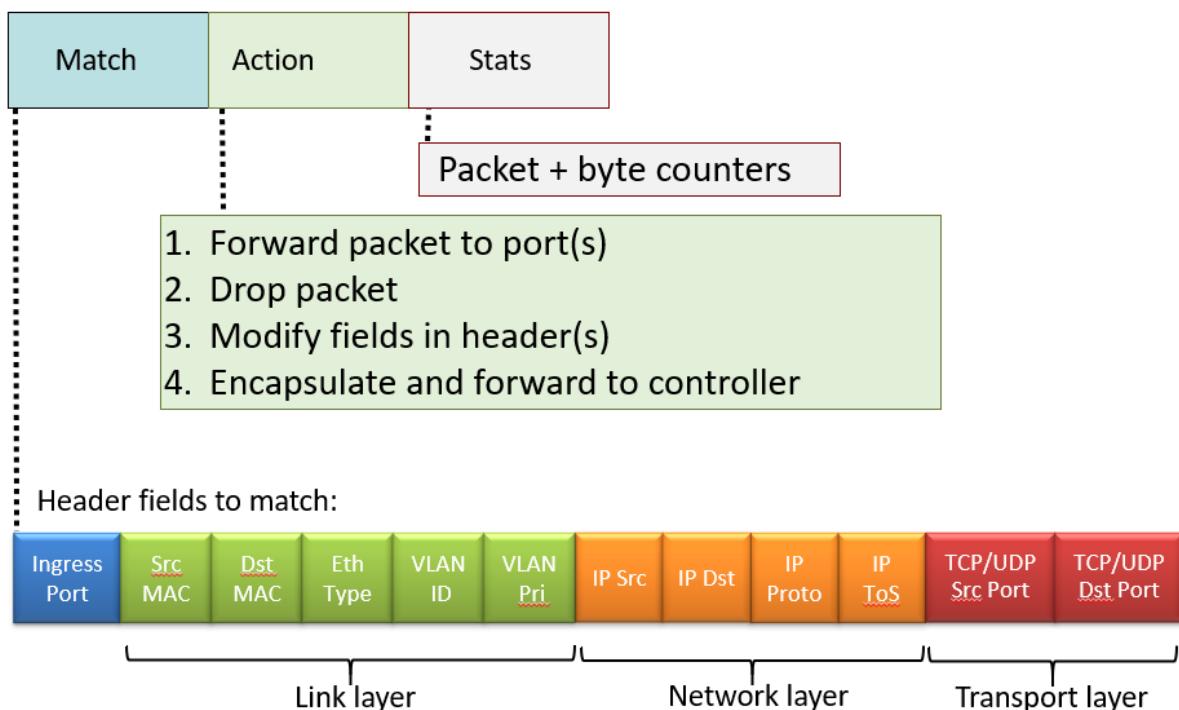
- Generalized forwarding: 기존의 destination-based forwarding에서는 dest IP address를 기준으로 forwarding을 했지만 NAT, DHCP처럼 추가적인 종류의 forwarding을 요구하는 기능들이 생기며 여러 가지 header field를 바탕으로 다양한 기능을 하는 generalized forwarding이라는 개념이 생겼다
- Software-defined networking (SDN): 패킷 스위치(switch와 router을 포함적으로 부르는 용어)들이 쓰는 forwarding table을 대신 계산해주는 원격 장치로 중앙 제어 장치 같은 느낌이다
- OpenFlow: SDN과 실제 패킷 스위치들이 통신하는 프로토콜, 이 프로토콜의

flow table로 generalized forwarding을 한다

#### 4. Flow table: generalized forwarding에서의 forwarding table

A. Flow: header field로 정의되는 것

B. Flow table entries



- i. Match: header field들로 만드는 패턴
- ii. Actions: matched packet에 대해 drop, forward, modify하거나 matched packet을 controller로 보낸다
- iii. Priority: action들이 중첩되는 경우 어떤 것을 우선할지 우선순위
- iv. Counters: 지금까지 얼마나 데이터(#bytes, #packets)를 처리했는지

C. 예시

- i. Destination based forwarding

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

## ii. Firewall

Switch Port	MAC <u>src</u>	MAC <u>dst</u>	Eth type	VLAN ID	VLAN <u>Pri</u>	IP <u>Src</u>	IP <u>Dst</u>	IP <u>Prot</u>	IP <u>ToS</u>	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	*	22 drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC <u>src</u>	MAC <u>dst</u>	Eth type	VLAN ID	VLAN <u>Pri</u>	IP <u>Src</u>	IP <u>Dst</u>	IP <u>Prot</u>	IP <u>ToS</u>	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

## iii. Link layer destination-based forwarding

Switch Port	MAC <u>src</u>	MAC <u>dst</u>	Eth type	VLAN ID	VLAN <u>Pri</u>	IP <u>Src</u>	IP <u>Dst</u>	IP <u>Prot</u>	IP <u>ToS</u>	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

## iv. Match+action 대표적인거

### Router

- **match:** longest destination IP prefix
- **action:** forward out a link

### Switch

- **match:** destination MAC address
- **action:** forward or flood

### Firewall

- **match:** IP addresses and TCP/UDP port numbers
- **action:** permit or deny

### NAT

- **match:** IP address and port
- **action:** rewrite address and port