컴퓨터 네트워크

▼ Internet : “Network of networks”

- Elements of Internet

  Hosts(End system = 단말기) : Billions of connected computing devices

  Packet switches(routers, switches) : forward packets, 생산되는 정보들을 중계

  Communication links : fiber, copper, radio, satellite(유&무선)

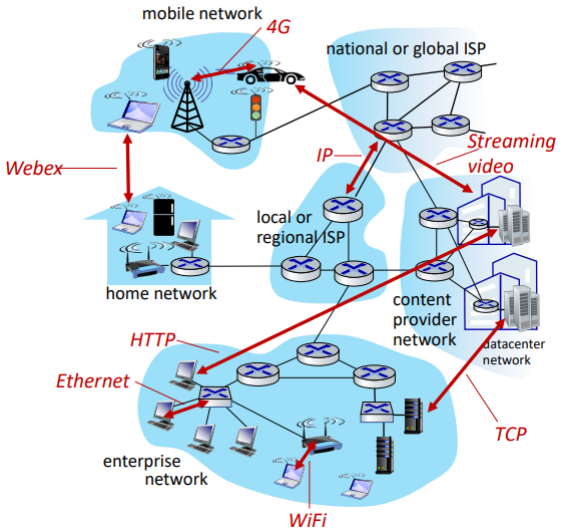
  Networks : collection of devices, routers, links managed by an organization

- protocols : 주고받는 메세지의 형식, 순서, 송수신시 수행되는 작업

  control sending & receiving of messages

  ex) Http, streaming video, Skype, TCP/IP, WiFi, 4G, Ethernet

  Internet Standards : 인터넷에 접속하기 위한 Protocol ex) TCP/IP



- 네트워크 구성

사용자가 스마트폰으로 인터넷에 접속 : Network edge인 스마트폰이 Access networks인 와이파이를 통해 접속하여 Network Core의 기능을 통해 인터넷 서버로 패킷 전송 & 수신

1. Network edge

   네트워크의 가장자리, 수많은 hosts 존재

   hosts(단말기) : clients and servers

2. Access networks

   네트워크에 접근하기 위한 네트워크, end system이 인터넷을 사용할 수 있도록 길을 열어줌

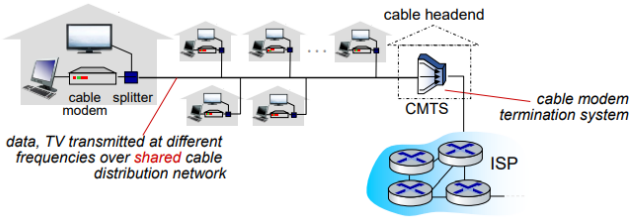
   유&무선 통신, 기지국, router 존재

   집 : residential access nets 설치

   학교, 회사, 기관 등 : Institutional access networks 설치

   기지국 , WiFi, 4G/5G : mobile access networks 설치

   - cable-based access : 기존의 케이블 TV로 인터넷 서비스 연결

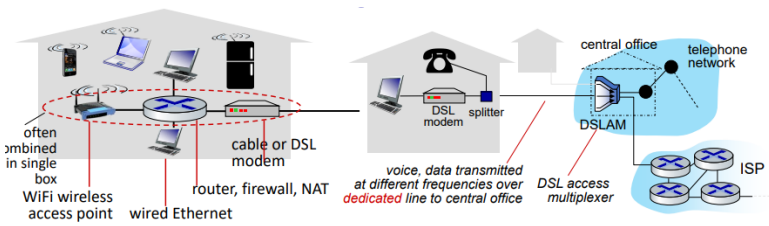


   다수의 가구가 나눠써야하므로 전송속도 낮음

   전기신호 사용 : FDM(Frequency division multiplexing)을 사용하여 주파수에 따라 케이블TV(낮은 주파수 사용), 인터넷 데이터(높은 주파수 사용) 전송

   케이블 네트워크는 가정집을 ISP(Internet Service Provider) router에 연결

   - digital subscriber line(DSL) : 기존 전화선을 이용한 인터넷 서비스



   각 집에서 Central office의 DSLAM으로 연결된 dedicated line로 인터넷 data, 전화를 받음

   DSL 전화선에서 온 data는 internet으로, voice는 전화망으로 이동

   다른집과 데이터 공유 X → 충돌 X

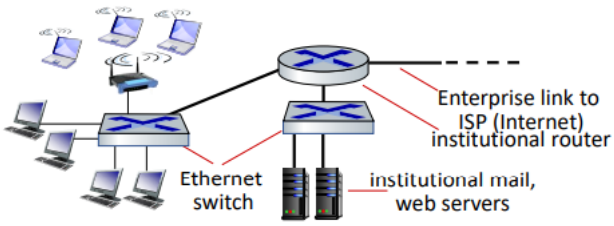
   24 ~ 52Mbps downstream / 3.5 ~ 16Mbps upstream tx rate

   upload보다는 download를 많이 하므로 downstream tx rate가 높도록 설계 ⇒ asymmetric

   downstream : 사용자가 받는 data

upstream : 사용자가 보내는 data

- Home networks & Enterprise networks(Ethernet) : LAN선(에더넷 케이블) 사용

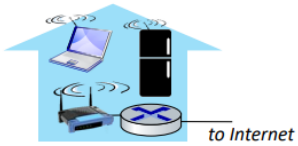


institutional router : ethernet보다 기능 좋음(ethernet끼리 연결 → 하나의 네트워크로 보게 함 : network of network)

- Wireless access networks : End system을 라우터에 연결

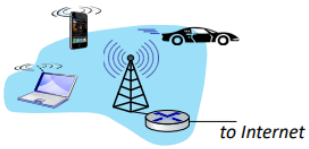
  AP(Access Point) : 무선랜을 구성하는 장치(유선랜과 무선랜을 연결시켜주는 장치)

  1. Wireless LANs (WLANs)



  802.11n/ac(WiFi 5/6/7) : 450 ~ 11000Mbps tx rate

  2. WIde-Area cellular access networks : 기지국을 통해 서비스



  100~2000Mbps → 4G/5G

3. Network Core

   전체 네트워크 시스템의 중앙에 위치, 데이터를 전송하는 핵심 역할, pakcet을 받아 넘겨줌

   routers가 그물처럼 얽혀있음 ⇒ network of networks

💡 Switch vs Router

Switch : 목적지로 출발한 데이터를 중간에 적합한 경로로 스위칭해줌
- 여러 장치를 연결하여 네트워크 형성
- Data Link Layer → MAC 주소 기반으로 동작

Router : 목적지로 가는 적합한 경로를 찾아주는 라우팅기능
- 서로 다른 네트워크를 함께 연결한 네트워크 형성
- Network Layer → IP 주소 기반으로 동작

- Host(End System : 단말기)

  access network로 패킷을 전송(transmission rate R : capacity, link bandwidth)

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

- The Network Core : 여러 네트워크를 연결

  수많은 라우터들이 그물처럼 얽혀있는 구조, Packet을 받아 전달

▼ The Nework Core > Packet-Switching : 네트워크 코어에서 패킷을 교환하는 방식 ⇒ 실질적으로 패킷이 어디로 갈지 판단하고 전송하는 기능

Host가 application Layer의 data를 Packet으로 나누어 전송

패킷은 Store-and-Forward 방식으로 전송

장점 :

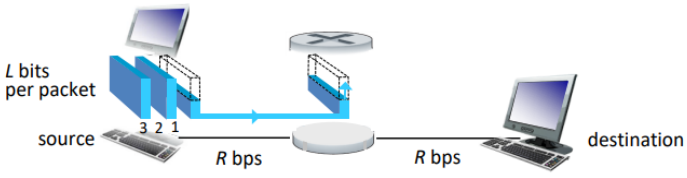
1. 회선 이용률 높음
2. 고 신뢰성, 고품질, 고효율

⇒ 대용량 데이터 전송이 필요한 인터넷 통신에 적합

단점 :

1. 경로에서의 각 교환기에서 지연 발생(Queueing delay)

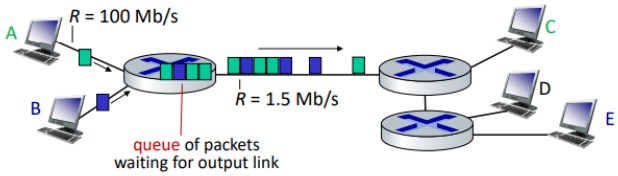
- Store-and-Forward : 라우터가 패킷을 온전히 다 수신한 후 저장을 하고 다시 보내는 방식



Transmission delay : L-bit의 패킷을 Rbps의 회선으로 보낼때, $\frac{L}{R}$초의 딜레이 소요

🧽 One-hop numerical example:
Ex) L = 10kbits, R = 100Mbps
⇒ one-hop tx delay = 10k / 100M = 0.1msec

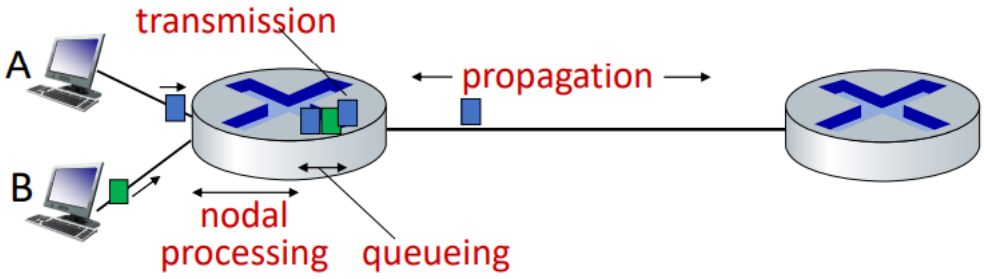
- Queueing delay & loss : 패킷이 Router에 Queue형식으로 들어와 전송 대기(Queueing delay), 만약 저장할 공간이 사라지면 패킷을 버림(Loss)



Queue에서 출발하는 속도보다 Queue로 도착하는 속도가 빠르므로 Queue에 패킷이 쌓임 ⇒ 큐에 공간이 없다면 dropped(loss)

- Delay

> Router에서 발생하는 Delay
>
> 1. Processing Delay (처리 지연) : 패킷헤더를 조사하고 그 패킷을 어디로 보낼지 결정하는 시간
> 2. Queueing Delay (큐 지연) : 패킷은 큐에서 전송되기를 기다리면서 지연 발생
> 3. Transmission Delay (전송 지연) : 모든 비트들을 링크로 밀어내는데 걸리는 시간
> 4. Propagation Delay (전파 지연) : 패킷이 출발해서 도착하기까지 걸리는 시간



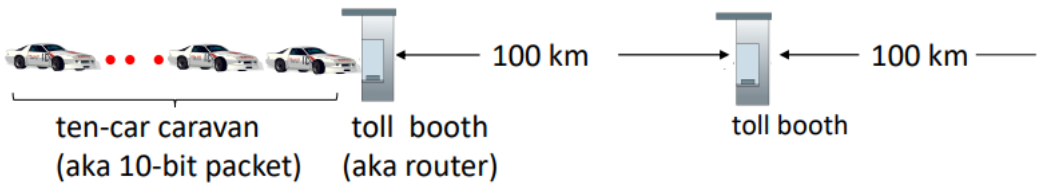
$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

d_{proc} : check bit errors, determine output link, typically < msec

d_{queue} : waiting at ouput link for transmission, 가장 큰 delay

d_{trans} : $\frac{L}{R}$ (L : packet length, R : transmission rate)

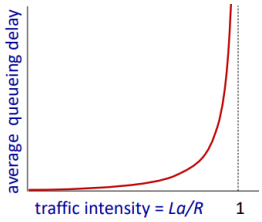
d_{prop} : $\frac{d}{s}$ (d = length of physical link, s = propatation speed)



Cars “propagate” at 100km/h
router takes 12 sec to service car

⇒ 모든 차들을 physical link로 push하는데 걸리는 시간(transmission delay) : 12sec * 10 = 120sec = 2min
⇒ physical link를 지나는 시간(propagation delay) :
$\frac{100km}{100km/h}$ = 1h
답 : 2min + 1h = 62min

Queueing delay :



R : link bandwidth(bps)

L : packet length(bit)

a : average packet arrival

⇒ traffic intensity = $I = \frac{La}{R}$

- 1 > $\frac{La}{R}$ > 0 : small queueing delay
- $\frac{La}{R}$ = 1 : large queueing delay
- $\frac{La}{R}$ > 1 : infinite delay

⇒ Queueing delay : $I(1-I)\frac{L}{R}$ for $(I<1)$

- Network Core (router가 그물처럼 얽혀있는 구조)의 기능
  1. Forwarding : 패킷을 수신한 포트에서 출발지 포트로 전달 (Local action)

     라우터, 스위치에서 수행

     forwarding table의 목적지 주소 분석 → 패킷을 전달해야할 포트 선택

     forwarding table을 만드려면 중계기끼리 정보교환 필요

  2. Routing : 패킷이 어디로 갈지 판단 (Global action)

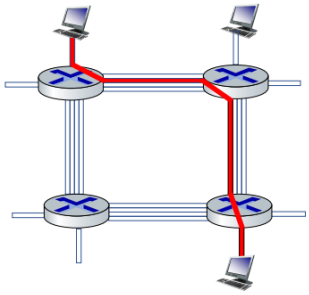
     Router는 라우팅 테이블(Routing table)을 사용하여 패킷을 수신하는 경로 결정 & 해당 경로로 패킷을 전송

     라우팅 프로토콜을 사용하여 네트워크 간 경로 정보 교환 & 라우팅 테이블(Routing table) 업데이트 (routing algorithm)

     ⇒ 가장 빠르고 안정적인 경로를 찾아서 패킷을 전달하는 것이 목적

▼ The Nework Core > Circuit-Switching : 하나의 회선을 할당하여 데이터를 주고받는 방식

연결되면 출발지부터 목적지까지 전송



장점 :

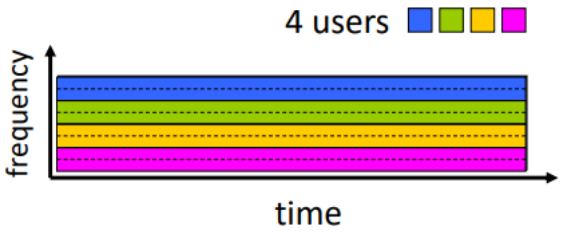
1. 데이터 전송 중에도 안정적인 연결 유지
2. 전송속도 일정

⇒ 전화통화와 같은 실시간 음성 통신에 적합

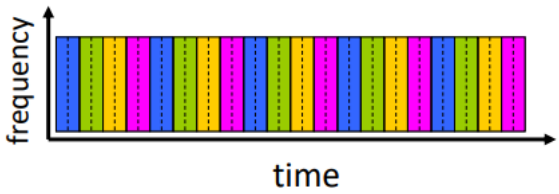
단점:

1. 사용되는 회선 전체를 독점(no sharing)하기 때문에 다른 사람 사용X
2. 전송량이 적을 경우 비효율적
3. 회선이 예약되어있지 않으면 전송 불가
4. 대규모 네트워크 형성 어려움

- FDM (Frequency Division Multiplexing) : 좁은 주파수 대역으로 나누어 전



- TDM (Time Division Multiplexing) : 시간을 여러개의 슬롯으로 나누어 전송



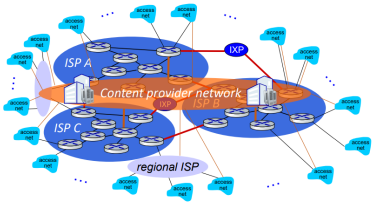
Packet Switching vs Circuit Switching

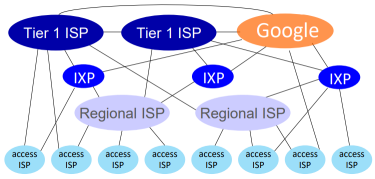
1 Gbit/s link
each user : 100Mb/s, active 10% of time(1/10확률로 전송)

Circuit-Switching : 10 users (10 X 100Mb/s = 1Gbp/s)
Packet-Switching : with 35 users, P is less than 0.004 when 10 users active at same time.

- Internet structure

  Host(End system)은 ISPs(Internet Servie Providers)에 접근하여 인터넷에 접속



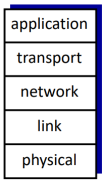


  access ISP를 효과적으로 연결하기 위해 ⇒ 여러개의 Regional ISPs & global ISPs 설치 ⇒ global ISP를 연결시켜주는 internet exchange point & perring link (interconnected)

- Network Security :

- Layering(계층화) : 복잡한 컴퓨터 네트워크를 쉽게 설계/구축/이해하는 방법

  분할정복(divide & conquer) : 복잡한 문제를 작은 단위로 분할(상하관계)

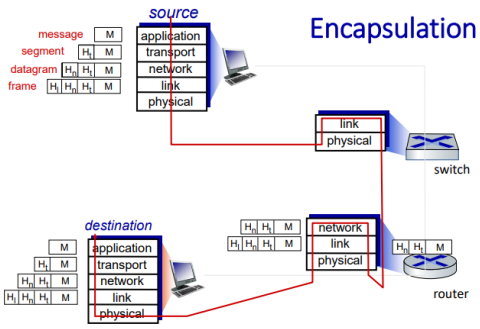


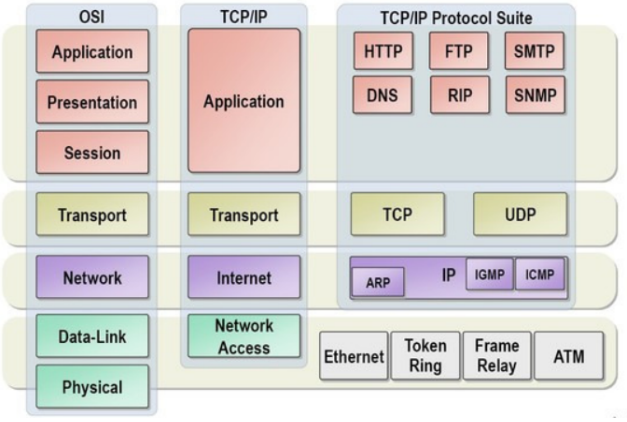
  1. Physical layer : 한 노드에서 다음 노드로 비트 스트림을 전송
  2. Data link layer : 한 노드에서 다음 노드로 프레임을 이동
  3. Network layer(Internetwork layer) : 서로 다른 기능의 네트워크를 묶어주는 기능

- Encapsulation

  switch : store-and- forward X & data link layer까지 encapsulate

  router : store-and-forward & network layer까지 encapsulate





▼ Transport Layer :

송신자와 수신자를 연결하는 통신서비스 제공 ⇒ 데이터 전달

- Transport Service에서 필요한것

1. Data integrity

   Some Apps require 100% reliable data transfer 가능(모든 데이터가 그대로 전달)

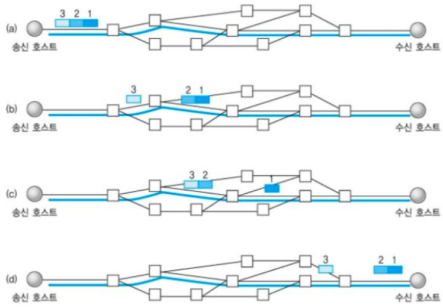
   Other Apps tolerate some loss

2. timing : delay에 민감한 Apps에 필요

3. Throughput :

application	data loss	throughput	time sensitive
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	10 msec
streaming audio/video	loss-tolerant	same as above	few secs
interactive games	loss-tolerant	Kbps+(크게 필요 없음)	10 msec
text messaging	no loss	elastic	

- Protocol in Transmission Layer : TCP & UDP
- TCP(Transmission Control Protocol) service :

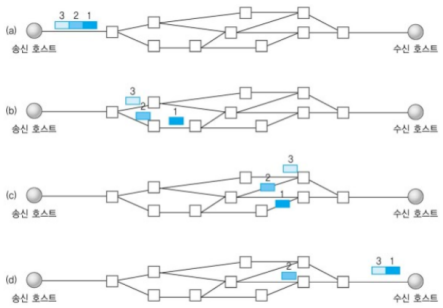


- reliable transport : bit과 순서가 유지, 손실x ⇒ 신뢰성 높음 ⇒ 속도 느림
- flow control : receiver 관련
- 연결 추구 : 서버와 클라 사이의 연결 요구
- congestion control : 네트워크 코어에 너무 많은 패킷이 몰리면 지연시간 증가되는것을 방지
- timing & throughput 구현 X
- 보안기능 X ⇒ TLS라는 규약을 TCP와 같이 사용
- 오류 발생 → 재전송

ex) file 송수신 & e-mail & Web documents & streaming 등등

> 🔑 Flow Control & Congestion Control :
> Flow Control(흐름제어) : 데이터를 송신, 수신하는 곳의 데이터 처리속도를 조절 ⇒ 수신자의 버퍼 오버플로우를 방지하는 것
> (송신하는 곳에서 감당안되는 데이터를 많이 보내면 수신자에서 문제 발생)
> Congestion Conrtol(혼잡제어) : 네트워크 코어 내의 패킷 수가 넘치게 증가하지 않도록 방지
> (정보량이 많아지면 패킷을 적게 전송하여 혼잡 붕괴 현상 방지)

- UDP(User Datagram Protocol) service



- 순서가 변하고 bit 손실 가능성 존재 ⇒ 신뢰성 낮음 ⇒ 속도 빠름
- 데이터그램 방식(비연결형)
- IP(best effort)의 특성 그대로 사용가능
- timing & throughput & congestion control & flow control 제공 X
- 오류 검사만 수행 (재전송 x)

ex) Internet telephony & interactive game

⇒ 데이터를 신뢰성 있게 전송해야 하는 경우 : TCP

전송속도가 빠르며, 손실된 데이터를 복구할 필요가 없는 경우 : UDP

▼ Application Layer :

Server : 변하지 않는 IP 주소 가짐, always-on host(상시대기)

Clients : dynamic IP 주소 가짐, server와 컨택, 소통함 clients끼리 직접적으로 소통 불가능 ex) HTTP, IMAP, FTP

- Socket : 프로그램이 네트워크에서 데이터를 주고받을 수 있도록 Server와 Client간의 통신을 지원
  파일과 비슷하지만 서로 소켓을 연결해야한다는 점에서 다름
  transport layer와 application layer 사이에 존재
  1대1, 다대다 통신 가능
  Socket 구분자로 IP주소와 포트번호, 그리고 프로토콜 이름을 사용(IP주소, 포트번호 같은 소켓 존재 가능)

- Web : consists of objects ⇒ web server에 저장되어있음(HTML file, JPEG image, 등등), 각 개체는 URL로 주소 지정 가능

- HTTP(Hypertext Transfer Protocol) : 웹 서버간의 통신을 위한 프로토콜(Application layer)

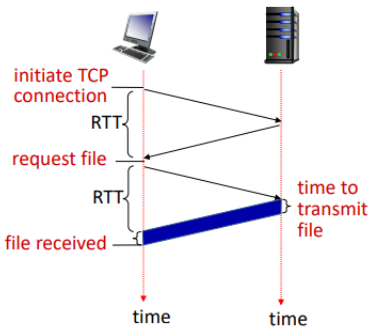
> 🔑 Client :
>
> - 브라우저는 http 프로토콜을 사용하여 서버에 요청하고 수신 → Web에 display
>
> Server :
>
> - Web은 http 프로토콜을 사용하여 요청과 답에 응해 object를 전송

클라이언트가 서버와 TCP 커넥션 생성(port 80)
→ Server가 클라이언트에서의 TCP 커넥션 수락
→ 양방향(full-duplex) 통신 가능
→ 파일 교환 후 connection close

  - Stateless : client와 server가 어떤 기능을 교환했는지 저장x

  - Non-persistent HTTP : TCP 연결 한번에 하나의 객체 전송
    두 개 이상의 객체를 전송하려면 두번 이상의 연결 필요
    1. Clinet가 Server에 TCP커넥션 연결 요청
    2. 연결 요청에 대한 응답(응답 패킷이 client로 와야함)
    3. server로부터 응답을 받는 순간부터 TCP 연결
    4. request를 보내서 받고자 하는 패킷을 서버에 요청
    5. server는 해당 파일을 전송하고 client가 받으면 server는 TCP connetction 닫음

    RTT : 패킷 하나가 cliet에서 server로 이동했다가 다시 돌아오는 시간
    Non-persistent HTTP response time = 2RTT + file transmission time



- Persistent HTTP : TCP 커넥션을 열고 모든 오브젝트 전송

⇒ 많은 오브젝트를 한 커넥션으로 전송 가능 ⇒ response time 감소

- HTTP message

  \r : CR(carriage return character) → 커서를 맨 앞으로 옮기기

  \n : LF(line-feed character) → 한줄 내리기

  ⇒ \r\n : 줄바꿈

  - Request : ASCII 사용 ⇒ 사람이 읽을 수 있는 포맷
  - Request Message

  > 🔑 Request line : 데이터 처리 방식(POST, GET, HEAD, PUT ...), 기본 페이지, 프로토콜 버전 포함
  >
  > Header lines : Host, User-Host, Accept 등의 정보

```
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
\r\n
```

1. GET method(조회) : 데이터를 읽거나 검색할때 사용(수정x)

   - URL에 데이터를 포함시켜 요청(query string parameters로 전달)

   ⇒ 보안에 취약

   - 전송 길이에 제한 & 캐싱 가능

   - 북마크 가능

   - 같은 요청을 여러번 수행해도 같은 응답 수신

2. POST method(등록) : 데이터를 서버로 보내 리소스 생성 or 수정할때 사용

   - 데이터를 message body에 포함시켜 요청(데이터 노출x)

   ⇒ 보안에 좋음

   - 전송길이에 제한x & 캐싱 불가능

   - 북마크 불가능

   - 같은 요청을 여러번 수행해도 결과가 다름

3. HEAD method : GET 요청과 동일한 결괏값 반환 but 응답본문 포함x

   - HTTP 요청 시 헤더 정보만 전송 (GET method는 헤더&데이터 전송)

   - GET method보다 속도 빠름 : GET과 동일한 응답을 요구하지만 서버의 응답의 본문이 리턴되냐의 차이

   ⇒ 빠르게 서버의 상태 확인 가능

4. PUT method(수정) : 서버에 새로운 객체를 업로드

   - 동일한 요청을 여러번 수행하면 항상 동일한 결과 생성

   - 요청한 URL에 객체가 존재하는 경우 기존의 존재하던 객체를 대체

◦ Response Message

> 🔑 status line : 프로토콜 버전(HTTP version), 성공 및 실패 여부(Status Code), 코드에 대한 설명
> Header lines : Date, Server, Length 등의 정보

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

◦ HTTP response status codes

   - 200 OK : 요청 성공, 요청된 객체가 다음 메세지로 옴

   - 301 Moved Permanetly : 해당 URL이 영구적으로 새로운 URL로 변경

   - 400 Bad Request : 서버가 요청을 인식 못함

   - 404 Not Found : 서버에서 요청된 객체를 찾지 못함

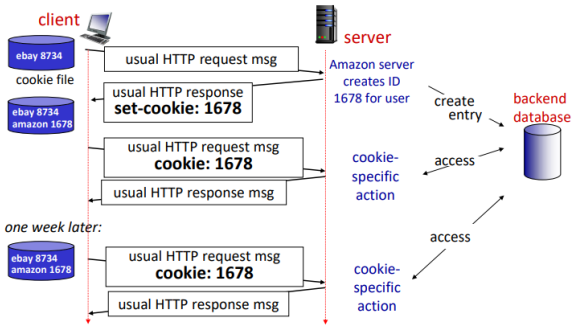
- Cookies : 서버가 사용자의 브라우저에 전송하는 데이터 → client가 저장 & 관리

  HTTP의 stateless : client와 server간의 정보 저장x ⇒ Cookies로 보완

  client, server가 추적할 필요 없음

  모든 HTTP 요청이 서로 독립적임, 복구 필요x

  ⇒ 쿠키를 사용함으로써 브라우저의 일부 상태 유지



1. request message후 response message에서 set-cookie헤더에 cookie를 담아보냄

2. 이후 client가 해당 서버로 요청을 할때 이전에 저장돼있던 모든 쿠키를 담아 회신

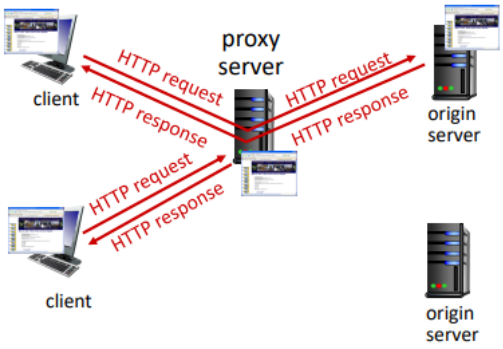
- Cookies and privacy

사용자의 브라우저가 다양한 웹사이트에서 사용가능한 ID 생성 → 제3자가 해당 사용자가 다양한 웹사이트에 방문한 정보 수집 & 추적 가능 → 수집된 정보는 또다른 제3자에게 판매되어 광고 타겟팅 가능

- Web Caches : 여러 사용자가 동일한 내용의 request를 반복적으로 요청할때 proxy server(임시 저장소)에서 가져오면 효율적임 (동적 페이지에서는 기능x) ⇒ client와 server사이의 중계역할

  - caches안에 있는 객체일때, cache가 client에게 객체 반환

  - origin server에 있는 객체일때, cache가 객체를 받고, 다시 client에게 객체 반환 & cache에 객체 저장



- 장점 :

1. client에 대한 응답시간 줄임
2. 원본 서버로 요청을 전달하지 않아도 되어 access link의 트래픽 감소 ⇒ 대역폭이 제한된 link에서 중요

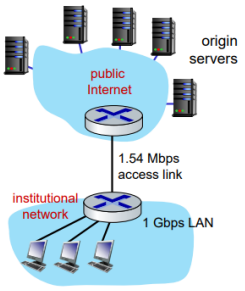
- Caching example

RTT(from router to server) : 2sec

Web object size : 100k bits

average request rate : 15/sec

average data rate : 1.5Mbps



LAN 사용률 : $\frac{1.5Mbps}{1Gbps} = 0.0015$

access link 사용률 : $\frac{1.5Mbps}{1.54Mbps} = 0.97$

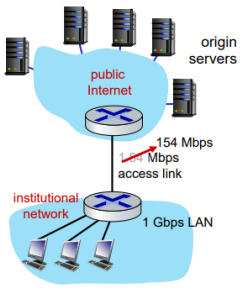
⇒ queueing delay 매우 큼

⇒ access link의 속도를 올리던가 data rate를 낮추던가

delay : internet delay(2 sec) + access link delay(minutes) + LAN delay(usecs)

---

access link rate : 154Mbps



RTT : 2sec

LAN 사용률 : $\frac{1.5Mbps}{1Gbps} = 0.0015$

access link 사용률 : $\frac{1.5Mbps}{154Mbps} = 0.0097$

⇒ queueing delay 줄어듦

delay : internet delay(2 sec) + access link delay(msecs) + LAN delay(usecs)

install a web cache

suppose cache hit rate = 0.4

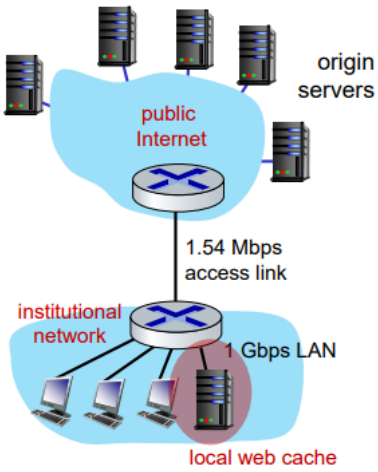
⇒ 40% requests satisfied at caches

⇒ 60% use access link

average data rate = $1.5Mbps \text{ x } 0.6 = 0.9Mbps$

access link 사용률 : $\frac{0.9Mbps}{1.54Mbps} = 0.58$

⇒ 위의 예시보다 줄어들음 ⇒ delay 감소



- Conditional GET method : 캐시에 최선 버전이 있을경우 객체 전송x

  ⇒ 객체 전송 지연x, link 이용률 감소

  - Cache : 시간을 지정하고 지정한 시간까지의 변경사항 전송

    변하지 않았다면 not modified, 변했다면 변경된 데이터 전송

- HTTP/1

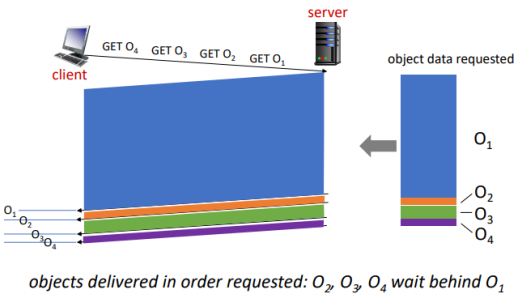
  - client는 여러 GET요청을 TCP연결에 한꺼번에 보냄

  - server는 요청을 순서대로 처리하여 응답 전송

  ⇒ client와 server간 오버헤드 줄이고 지연 최소화

  HOL(Head-Of-Line) blocking : 여러 GET요청을 하나의 TCP연결로 보내므로, 작은 객체가 큰 객체의 전송이 완료될때까지 기다려야 하는 경우

  ⇒ 전송시간 증가, 네트워크 성능 저하



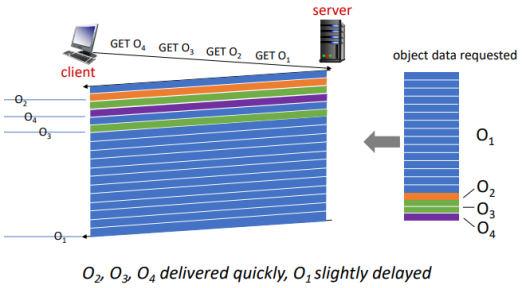
- HTTP/2 : HTTP/1의 기능 보완, 개선

  - HTTP/1과 같이, 하나의 TCP연결에서 여러개의 요청과 응답을 파이프라인으로 처리

  - 다중화 기술 사용 ⇒ HOL blocking 문제 해결

  - 객체간 우선순위 설정 ⇒ 요청 순서 조절 ⇒ 중요한 객체를 먼저 받음으로써 로딩시간 감소

  - 객체를 여러 개의 프레임으로 나누어 전송 ⇒ HOL blocking 문제 해결



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

- HTTP/3

  - 자체적인 transport layer 프로토콜

  - UDP를 사용하여 통신, 개별 객체에 대한 에러 및 혼잡제어 가능

  - 기존의 TCP연결만을 사용하는 HTTP/2와 달리, UDP사용

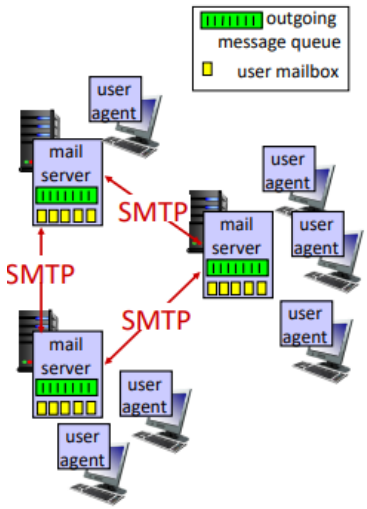
- E-mail

  Major components :

  - user agents

  - mail servers

  - simple mail transfer protocol : SMTP



  - User Agent : mail reader

  - 작성, 편집, 읽기 담당

  - message는 Mail Server에 저장

  - 수신사는 Mail Server에 접근하여 읽음

  - Mail Server : 사용자의 이메일을 관리하고 처리

  - 사용자의 이메일을 받고 저장

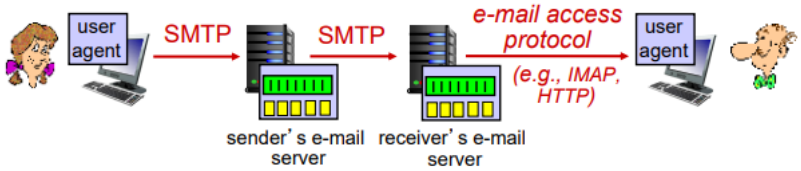
  - 서버간 이메일을 전송하기 위한 프로토콜 사용 : SMTP

- SMTP : use TCP(reliably transfer) ⇒ 포트 25를 통해 서버로 안정적으로 전송하는데 사용되는 프로토콜

  - 명령과 응답으로 상호작용 (7-bit ASCII text)

  - 전송 과정 : Greeting → Transfer of messages → Closure

  1. 송신자가 송신자의 mail server로 전송(메세지는 메세지 큐에 대기)
  2. client(STMP)는 수신자의 mail server로 TCP connection 염
  3. SMTP client는 메세지를 TCP connection을 통해 수신자의 mail server로 전송
  4. 수신자의 mail server에 메세지 도착
  5. 수신자 메세지 읽음



- SMTP : pull 방식(message를 보내는 방식)

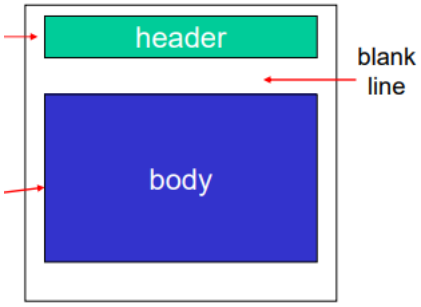
  여러 객체를 multipart message로 전송

  SMTP는 지속적인 연결 사용 → 한 번의 연결로 여러 메시지 전송

- Mail message format

  Header : 수신자, 송신자, 객체

  Body : the message(ASCII characters only)



- DNS(Domain Name System) : 도메인 이름을 IP주소로 변환하는 시스템

  많은 name server로 계층 구조를 가지는 분산 데이터베이스(IP 프로토콜과는 독립적)

  Network Core에서 뺌 ⇒ Application layer로 기능 구현(Network Edge로 기능 모음) ⇒ Core의 simplicity → Scalable한 network 추구)

  network edge에서 복잡도 증가

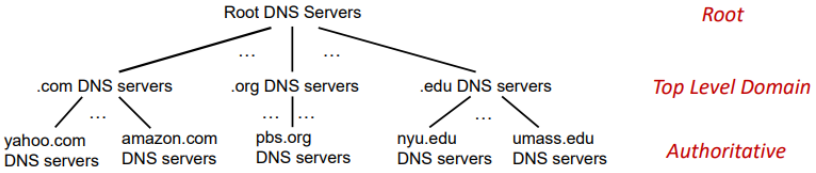
  - Host Aliasing(호스트 에일리어싱) : 같은 IP주소를 갖는 동일한 host가 여러 도메인 이름으로 불리는것
  - Load Distribution : 여러개의 IP주소를 하나의 도메인 이름과 대응시키는 것

    ⇒ 서버로 사용자의 request가 몰리지 않도록 기능

  DNS를 중앙화(centralize)하지 않는 이유

  1. single point of failure(단일 장애 지점)발생 → 전체 시스템에 접근 불가능
  2. traffic이 한곳으로 몰려 과부화 발생
  3. 멀리 떨어진 database에 의해 delay 증가
  4. 유지 보수 어려움
  5. 많은 query요청을 처리해야하므로 centralize 어려움 → DNS 서버들이 분산되어 운영

  - Namespace : 도메인별로 트리화하여 계층적으로 구성한 형태



최상위 계층 : Root DNS

그 아래층 : Top Level Domain(TLD) DNS

하위 계층 : SLD DNS

> 🔑 Example
> ”google.com” ← 도메인 서버 이름
> 1. 루트 서버 : “.”
> 2. TLD(최상위 도메인(.com)을 관리하는 네임 서버) : “com."
> 3. “google.com” 도메인을 관리하는 네임서버 : “google.com.”
> ⇒ 아래 계층으로 내려갈수록 서버이름 앞으로 단어가 붙음

⇒ DNS 도메인 이름을 효율적으로 관리

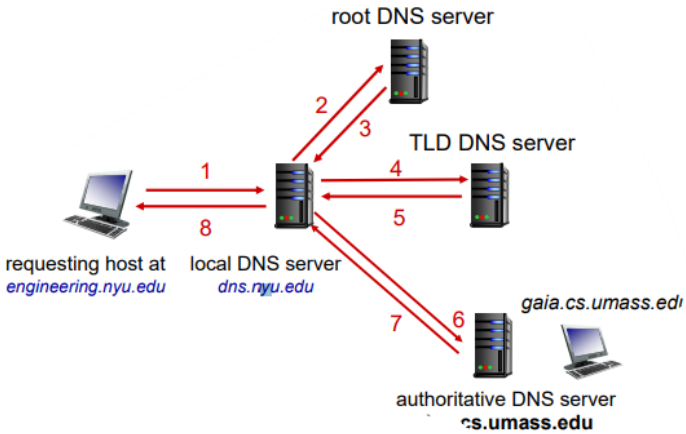
> 🔑 Example
> ”www.amazon.com”을 IP주소로 변환
> 1. client는 root server에 query하여 .com DNS server 찾음
> 2. client는 .com DNS server에 query하여 amazon.com DNS server 찾음
> 3. client는 amazon.com DNS server에 query하여 www.amazon.com의 IP주소 가져옴

- Root DNS :
- TLD DNS :
- Authoritative DNS server :

- DNS name resolution

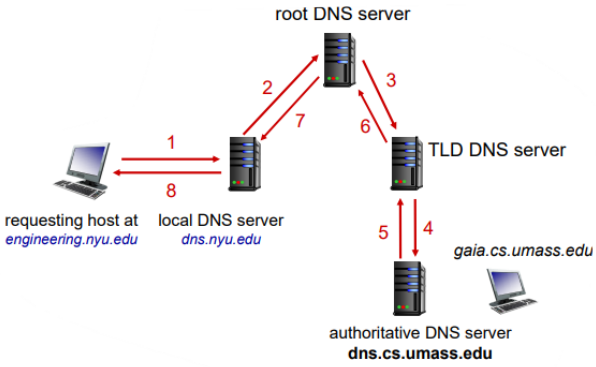
1. iterated query

- Local DNS server에서 각각의 DNS server에 query 전송(Local DNS server가 반복)
- 도메인이 복잡해질수록 반복횟수 증가



2. recursive query

- 매 단계마다 이전 단계의 DNS server를 대신하여 응답을 함



- Local DNS server : 임시 저장된 캐시를 이용하여 즉각 응답 가능

  - 최신정보가 아닐 수 있음

  - 캐시를 사용하지 않으면 지연시간 증가한다.

- DNS records

  RR format : (name, value, type, ttl)

  1. type = A

     name : hostname

     value : IP address

     (www.enterprise.com, 146.54.240.83, A)

  2. type = NS

     name : domain

     value : hostname of authoritative DNS

     (www.enterprise.com, dns.enterprise.com, NS)

  3. type = CNAME

     name : alias name for canonical name

     value : canonical name

     (www.enterprise.com, east4.enterprise.com, CNAME)

  4. type = MX

     value : name of mailserver

     (enterprise.com, mail.enterprise.com, MX)

- DNS protocol messages : query와 reply는 같은 포맷을 가짐 → application layer

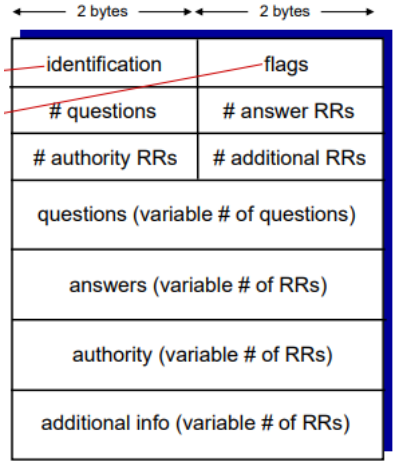
<Message Header> 4bytes x 3 = 12bytes

- identification : query & reply use 16bit(2bytes)

- flags : 어떤 유형의 메세지인지 확인

(query인지 reply인지는 17번째 bit을 확인해야함)

0x1333		0x0100	
1		0	
0		0	
4	‘c’	‘h’	‘a’
‘l’	4	‘f’	‘h’
‘d’	‘a’	3	‘e’
‘d’	‘u’	0	Continued on next line
1	1		



identification : 0x1333 flags : 0x0100

questions : 1 # answer RRs : 0

authority RRs : 0 # additional RRs : 0

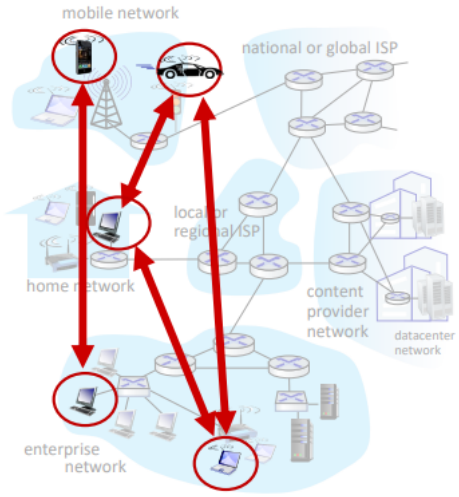
type A : 1 (마지막줄 노란색 첫번째 bit)

chal.fhda.edu에 대한 IP주소를 요청, domain name이 가변적이므로 {길이, 각 char}형식 다음에 empty string 사용

“chal.fhda.edu” ⇒ 4 ‘c’ ‘h’ ‘a’ ‘l’ 4 ‘f’ ‘h’ ‘d’ ‘a’ 3 ‘e’ ‘d’ ‘e’ 0 ← 각 character가 하나의 byte 사용

- Peer-to-Peer(P2P) architecture :

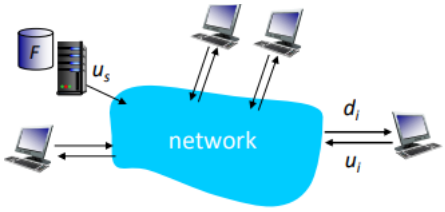
- 서버가 항상 온라인은 아님
- 임의의 end systems이 직접 통신
- peer가 서로의 peer에 서비스를 요청하고 제공받음
- 간헐적으로 연결되고 IP주소를 변경
- client가 많아질수록 성능 증가 (분산구조 self scalability)



- File distribution time
  - client-server

  - server transmission : upload N file copies → NF/u_s
  - client : each client download file copy → d_{min} = min client download rate ⇒ min download time : F/d_{min}

  ⇒ F의 파일을 N개의 client에게 분배하는 시간 $D_{c-s} \geq max\{NF/u_s, F/d_{min}\}$



  - P2P

- server transmission : upload at least one copy → F/u_s(파일 업로드 되는 시간)

- client : each client download file copy → F/d_{min} (가장 느린 client가 전송받는 시간)

- clients :

⇒ F개의file을 N개의 client에게 분배하는 시간 $D \geq max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$($u_s$: 서버의 upload rate, u_i : clients의 upload rate)

→ client수가 많아지면 $\sum u_i$증가 ⇒ D 감소

- P2P file distribution : BitTorrent

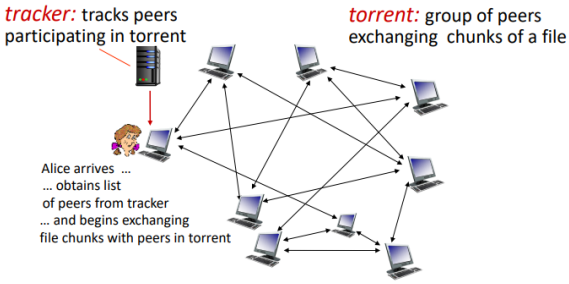
  - file divided into 256kbit chunks

  - peers send/receive file chunks → peer의 주소를 알아야함

  tracker(a.k.a server : 접속하려는 사용자는 다른 사용자의 IP주소를 받아옴)

  torrent : 여러 사용자로부터 한 파일에 대한 여러 청크를 받아옴

  churn : 사용자가 참여했다 나갔다 반복하는 상황, churn 증가 ⇒ 성능 감소



  - Requesting chunks :

    각각의 사용자들은 다른 chunk를 가지고 있음

    주기적으로 한 사용자는 이웃 사용자에게 각자 가지고있는 청크의 리스트 요청

    가장 드문 것(rarest first)을 먼저 요청

  - Sending chunks :

    가장 속도가 빠른 4명의 사용자에게 청크 보냄

    매 30초마다 임의의 사용자에게 chunk 전송

    ⇒ 낙관적으로 중단없는 전송 실현

- Video Streaming and CDNs :

  challenge of video streaming :

  1. scale : 한 컴퓨터가 traffic 감당x → single server approach x
  2. heterogeneity(이질성) : 각각의 사용자가 각자 다른 상황

  ⇒ solution : distributed, application-level infrastructure

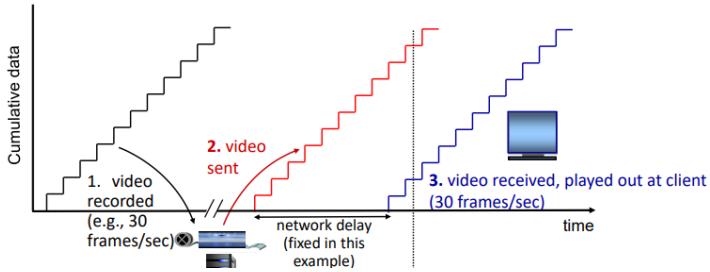
- Video : sequence of images ex) 24 images / sec

  이미지 내(spatial)혹은 이미지 간(temporal)의 중복을 사용하여 인코딩에 사용되는 비트 수를 줄임

  - CBR(constant bit rate) : 일정한 데이터량 전송
  - VBR(variable bit rate) : 가변적인 데이터량 전송

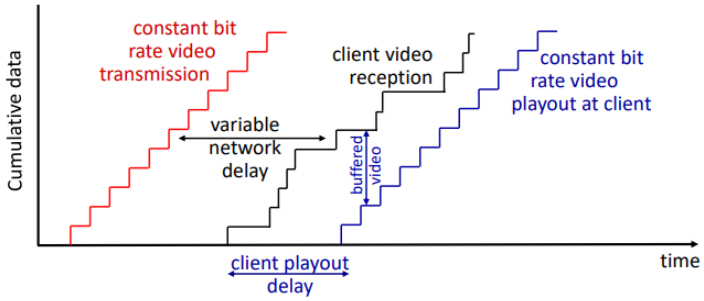
- Streaming stored video : 서버는 인터넷을 통해 사용자에게 비디오 프레임 전송 → 인터넷의 딜레이에 의해 사용자에 도달하는 속도 일정x → 데이터를 버퍼링 후 playout

  <딜레이가 일정할 때>



⇒ 사용자가 앞부분을 재생할때, 서버는 뒷부분을 전송

<딜레이가 일정하지 않을 때> ← continuous playout해야함 but delay는 가변적임(Jitter) ⇒ buffer에 저장 필요



⇒ client가 일정하게 video를 받지 않으므로 buffer에 저장 → client playout delay

- DASH (Dynamic Adaptive Streaming over HTTP) : 사용자의 통신 상태에 맞춰 최적의 품질로 video를 전송하는 프로토콜
  - Server :

    비디오 파일을 여러개의 chunk로 나눔

    각각의 chunk는 다른 비트율로 인코딩되어 저장된다.

    manifest file : 청크들의 URL을 제공
  - Client :

    주기적으로 server-client 대역폭 측정

    manifest file을 참고하여 하나의 청크 요청

    현재 bandwidth를 고려하여 최대 인코딩 비트율 선택

⇒ Streaming Video = encoding + DASH + Playout buffering

- CDNs(Content distribution networks)

- Mega-server

  single point of failure

  네트워크 혼잡지점

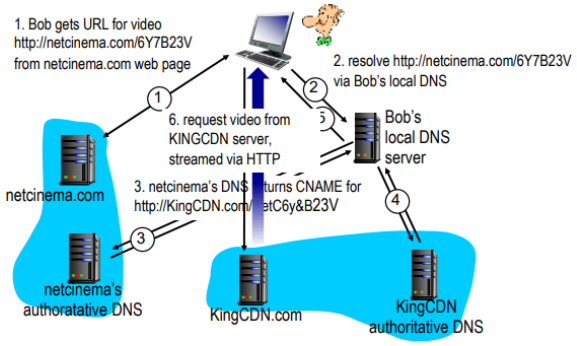
  client와의 긴 거리

  동영상의 여러 복사본이 발신 링크를 통해 전송

  ⇒사용자가 많아지면 성능이 떨어지므로 doesn't scale

- store/serve multiple copies of video at multiple geographically distributed sites(CDNs)

  - subscriber requests content from CDN



  3번에서 client와 가까운 서버 도메인 네임을 알려줌

- Socket Programming : application 프로세스와 transport protocol 사이의 문 (정보를 주고받을 수 있는 통로)
  - UDP(User Datagram Protocol)

no connection between client & server

sender : attach IP destination and port number

receiver : extracts sender IP address and port number

server		client
create socket, port = x serverSocket = socket(AF_INET, SOCK_DGRAM)		create socket clientSocket = socket(AF_INET, SOCK_DGRAM)
read datagram from serverSocket	←	Create datagram with server IP and port = x send datagram via clientSocket
write reply to serverSocket specifiying client address, port number	→	read datagram from clientSocket
		close clientSocket

<client>

```
from socket import*
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence') ← get user keyboard input
clientSocket.sendto(message,endcode(), (serverName, serverPort)) ← attach server name, port to message; send into socket
modifiedMessage, server Address = clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

<server>

```
from socket import*
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print("The server is ready to receive")
while True:
  message, clientAddress = serverSocket.recvfrom(2048)
  modifiedMessage = message.decode().upper()
  serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

- TCP(Transmission Control Protocol)

client must contact server

server		client
create socket port = x, for incoming request: serverSocket = socket()		
wait for incoing connection request connectionSocket = serverSocket.accept()	←TCP conection setup	create socket, connect to hostid, port=x clientSocket = socket()
read request from connectionSocket	←	send request using clientSocket
wrtie reply to connectionSocket	→	read reply from clientSocket
close connectionSocket		close clientSocket

<client>

```
from socket import*
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect(serverName, serverPort)
sentence = raw_input('Input lowercase sentense:')
clientSocket.send(sentence.encode()) #연결되어있으므로 receiver 지정할 필요 없음
modifiedSentence = clientSocket.recv(1024)
print('From Server:', modifiedSentence.decode())
clientSocket.close()
```

<server>

```
from socket import*
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1) #요청을 받아들이겠다 선언
print('The server is ready to receive')
while True:
  connectionSocket, addr = serverSocket.accept()
  sentence = connecctionSocket.recv(1024).decode()
  capitalizedSentence = sentence.upper()
  connectionSocket.send(capitalizedSentence.encode())
  connectionSocket.close()
```

▼ Transport Layer

1. Source부터 Destination까지 패킷이 제대로 전송될 수 있도록 함
2. Application layer에서 만든 데이터를 일정한 크기로 자름

sender : break application messages into segments, pass to network layer

receiver : reassemble segments into message, pass to application layer

- Multiplexing : 여러 어플리케이션의 Socket들로부터 데이터를 수집하여 패킷으로 만들어 하위 레이어로 전송하는 것 (Application → Transport)

  input : socket

  output : segment

- Demultiplexing : 하위 레이어로부터 수신된 패킷을 올바른 Socket으로 전송하여 올바른 어플리케이션에게 전송하는 것, Port Number 사용 (Transport → Application)

  UDP : demultiplexing using destination port number

  TCP : demultiplexing using 4-tuple

  input : segment

  output : socket

  - Connectionless demultiplexing → UDP ⇒ 1ㄷ1 커넥션x

    호스트는 IP datagram을 수신

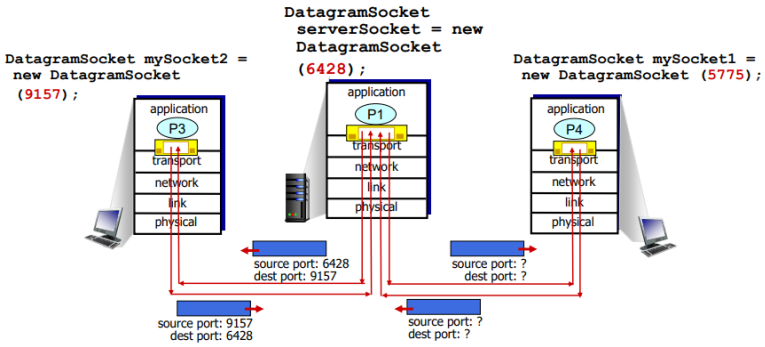
    Socket 생성 시 host-local port number 필요

    ex) DatagramSocket mySocket1 = new DatagramSocket(12345); ← 알아서 시스템에서 할당

    Datagram 생성 시 destinaton IP address & destination port number 필요(없으면 통신 불가)

    UDP segment 수신 시 destination port number 체크

    UDP 데이터그램 portnumber가 같을 수 있음, source IP가 다를 수 있음 ⇒ 동일한 소켓으로 지정(client가 여러 서버와 통신)



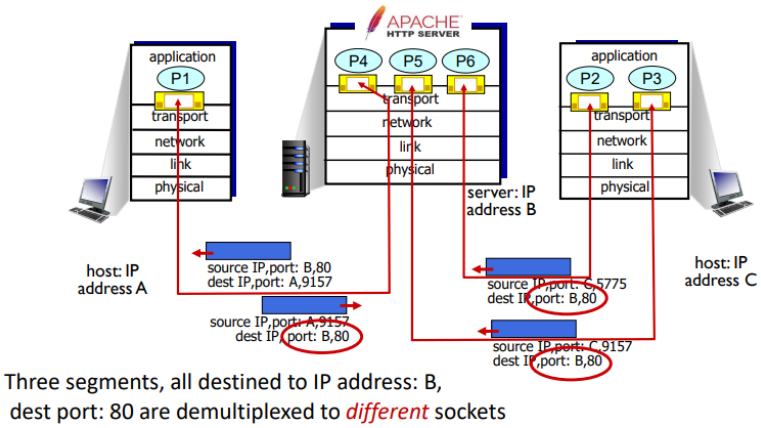
  - Connection-oriented demultiplexing → TCP ⇒ 1ㄷ1 커넥션

    4-튜플로 식별되는 TCP 소켓

    1. source IP address
    2. source port number
    3. dest IP addres
    4. dest port number

    Demux : 4-튜플정보를 가지고 정보가 모두 일치하는 포트로 전달

    각 연결된 클라이언트에 서로 다른 소켓 갖음



새로운 소켓을 만들때 같은 포트 써도 됨. 클라이언트별로 ip와 포트번호가 다르기때문에 구분 가능

- UDP : User Datagram Protocol

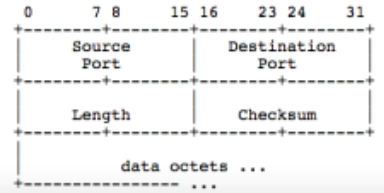
  connectionless : no —handshaking between UDP sender, receiver

  small header size

  no congestion control

  속도↑

  realiable transfer 필요 ⇒ add reliability & congestion control at application layer



  <sender>

  1. application layer에서 온 message 받음
  2. UDP segment header field값 결정
  3. UDP segment 생성
  4. segment IP로 전송

  <receiver>

  1. IP에서 segment 받음
  2. UDP chekcum header값 체크
  3. application layer message로 추출
  4. 소켓을 통해 메세지를 application layer로 demux

  - UDP checksum : 전송된 segment에서 에러 detect 목적

    <sender>

    treat contents of UDP segment(soruce port, dest port, length, data) as equence of 16bit integers

    checksum : 모두 더한 뒤 1의 보수를 취함 ⇒ receiver가 sender로 보내는 checksum값이 모두 1이 나오도록 하기 위함

    <receiver>

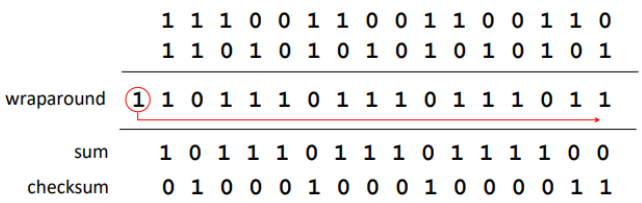
compute checksum of received segment

check computed checksum equals checksum field value

- 같으면 no error detected

- 다르면 error detected

example: add two 16-bit integers



각각의 field를 16bit로 바꾸어서 더함(만약 carry생기면 + 1) =sum

checksum = sum의 1의 보수값

ex) <sender> : checksum - 0001

<receiver> : 받은 데이터 값을 모두 더함 ⇒ 1110 (1110 + 0001) = 1111 ⇒ 데이터 손실x

<receiver> : 받은 데이터 값을 모두 더함 ⇒ 1010 (1010 + 0001) = 1011 ⇒ 데이터 손실o