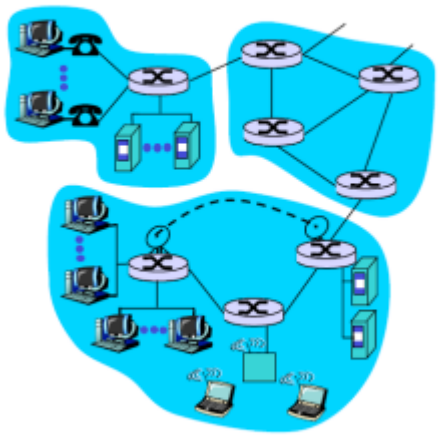
컴퓨터네트워크 기본

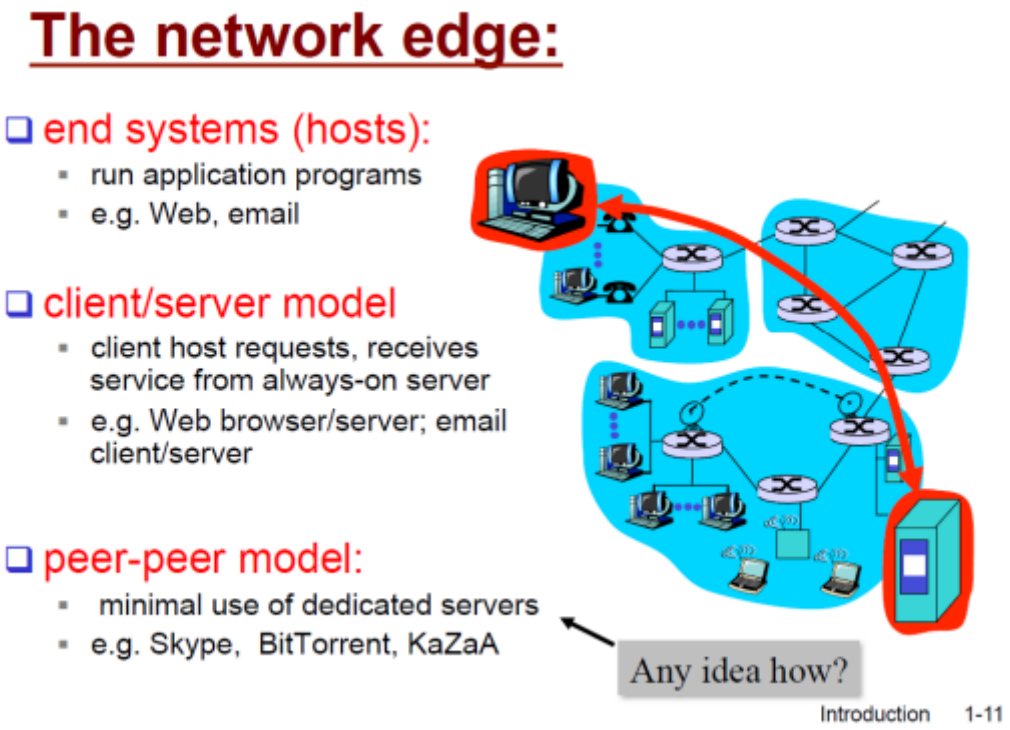
A closer look at network structure:

- network edge:
  applications and hosts
- network core:
  - routers
  - network of networks
- access networks,
  physical media:
  communication links



Introduction 1-10

- network edge: 네트워크 가장자리
  - 렙탑, 데스크탑, 웹서버 등 위치해있음(웹브라우저, 애플리케이션)
- network core: 네트워크 중심
  - 라우터(동그란거('X'표시))
- access networks, physical media:
  - communication links: 이어주는 링크들(인터넷 케이블, 전화선, 모뎀선, 무선링크(Wifi 등) 등..)



The network edge

end system(hosts):

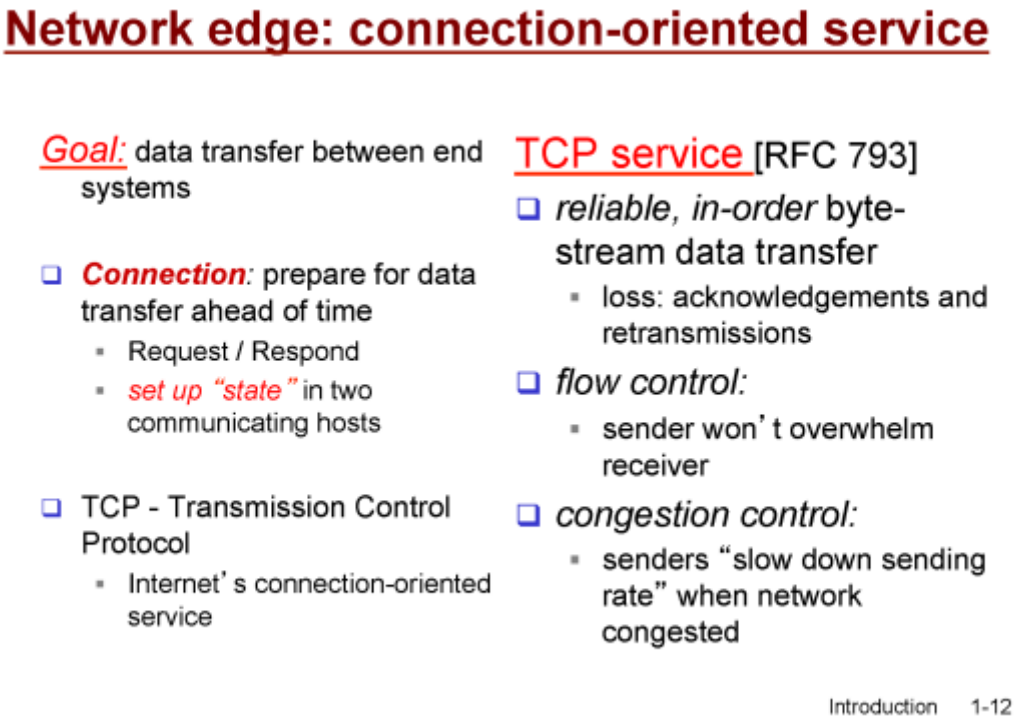
- run application programs
- e.g, Web, email

client/server model

- client : 자기가 원할 때 링크 연결을 해서 서버로부터 뭔가 정보를 가져오는 요소
- server : 항시 24시간 연결되어 있어서 클라이언트로부터 (언제들어올지모르는) 요청을 항상 기다리는 얘

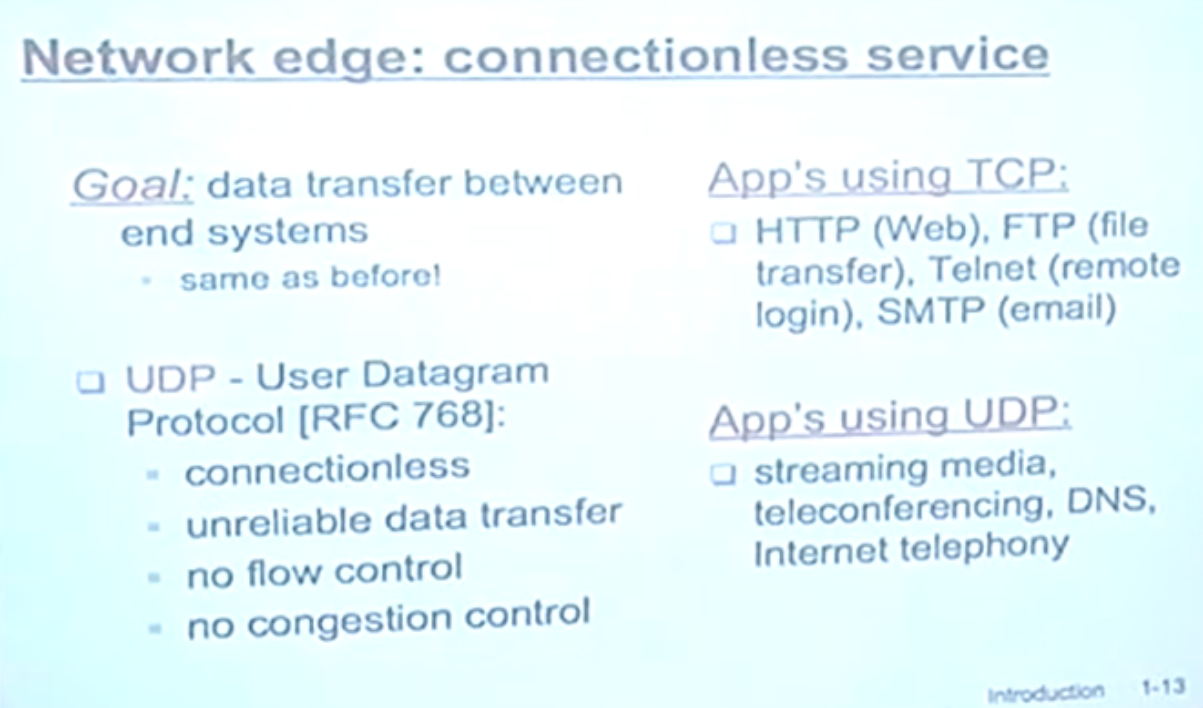
peer-peer model

- minimal use of dedicatied servers(전용 서버 최소한 사용)
- e.g, Skype, BitTorrent, KaZaA



TCP service

- Internet's connection-oriented service: 연결 지향 서비스
- reliable(믿을수있는), in-order(순서가 있는) byte stream data transfer : 클라이언트가 서버에 데이터를 보내면 신뢰성 있는 메세지가 유실되지 않고 그대로 감. 내가 보낸 메세지 순서 그대로 지키면서 도착 서버까지 도달.
- flow control: sender(보내는사람)이 receiver(받는 사람)에게 전달을 하는데, sender가 보내는 속도를 알맞게 조절해줌. receiver가 소화할 수 있는 속도에 맞춰서 그 속도로 전달.
    - 정보를 전달함에 있어서 받아들일 수 있는 사람의 소화할 수 있는 능력 속도에 맞춰서 전달.
- congestion control: sender(보내는 사람)와 receiver(받는 사람)의 처리속도가 동일하다고, 바로 보낼수 없음. 중간에 두 컴퓨터 사이를 연결해 주는 회선이 속도를 감당할수 없을 정도일 때(선이 가늘다거나) 둘 사이의 네트워크 상황에 맞춰서, 네트워크가 받아들일 수 있는 능력치만큼으로 보내줌



UDP- User Datagram Protocol

- connectionless : 비연결
- unreliable data transfer : 믿을수 없는 데이터 전달(데이터 유실 등)
- no flow control : 속도조절 없음
- no congestion control : 받는사람의 능력이 되든말든 그냥 전달

=⇒ TCP와 특징이 완전 반대.

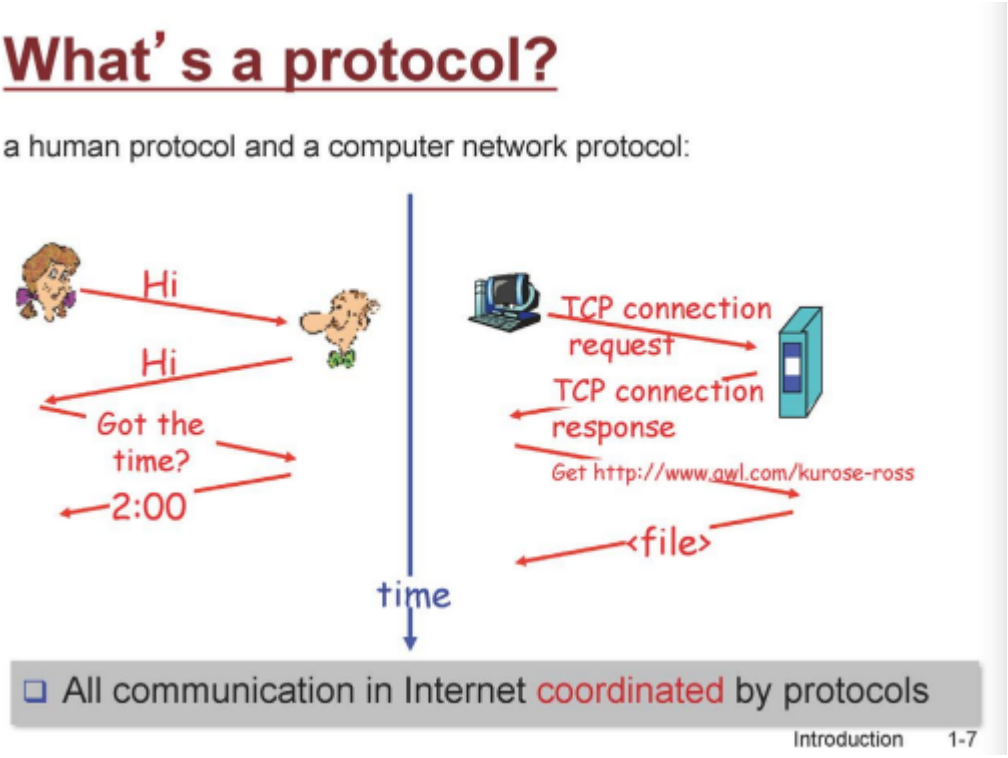
=⇒ 그래도 UDP가 사용되는 이유는?

=⇒ 속도가 빠르다는 것은 보내는 사람 입장에서 그냥 맘대로 막 쏟아 보내도 됨. 데이터 유실을 신경쓰지 않아도 되는 경우(예를 들어, 전화 통화) 오디오 패키지 몇개 유실되도 사람들은 감지를 못함. 실시간 오디오 같은 것

⇒그래도 데이터가 유실되지 않게 전달해야 하는 경우가 대부분이라 TCP(등기)사용.

TCP는 제공해주는 기능이 많지만 비용(컴퓨터 resource와 네트워크 resource)이 많이 듦.

TCP는 등기/ UDP는 일반 우편물 같은 느낌

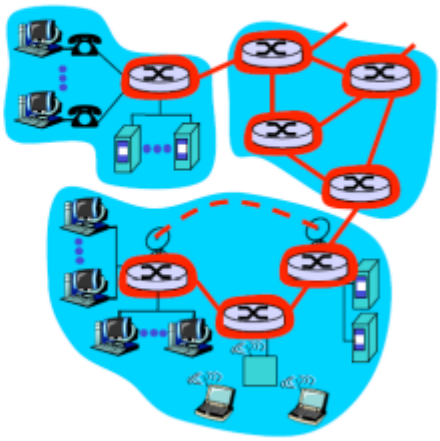


Protocol

- 복수의 컴퓨터 사이나 중앙 컴퓨터와 단말기 사이에서 데이터 통신을 원활하게 하기 위해 필요한 통신 규약
- 대화, 약속

The Network Core

- mesh of interconnected routers
- *the* fundamental question: how is data transferred through net?
  - circuit switching: dedicated circuit per call: telephone net
  - packet-switching: data sent thru net in discrete "chunks"



Introduction 1-15

- mesh of interconnected routers: 라우터들의 집합

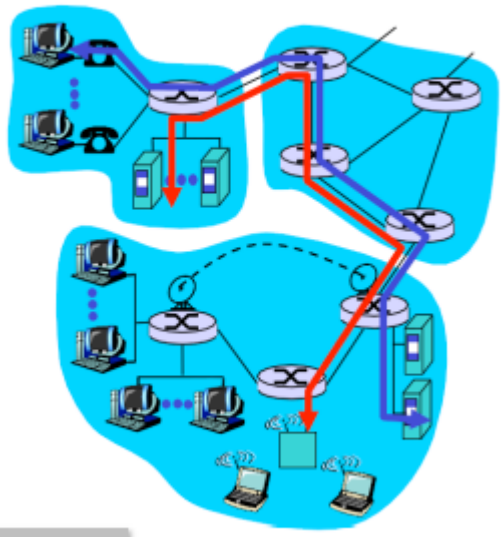
근본적인 질문: 데이터는 net을 통해 어떻게 전달이 될까?

- circuit switching (회로 스위칭): 전용 회로: 전화선
- packet-switching (패킷 스위칭): 개별 선을 통해 전송된 데이터 "청크?"

Network Core: Circuit Switching

End-end resources reserved for "call"

- link bandwidth, switch capacity
- dedicated resources: no sharing
- circuit-like (guaranteed) performance
- call setup required

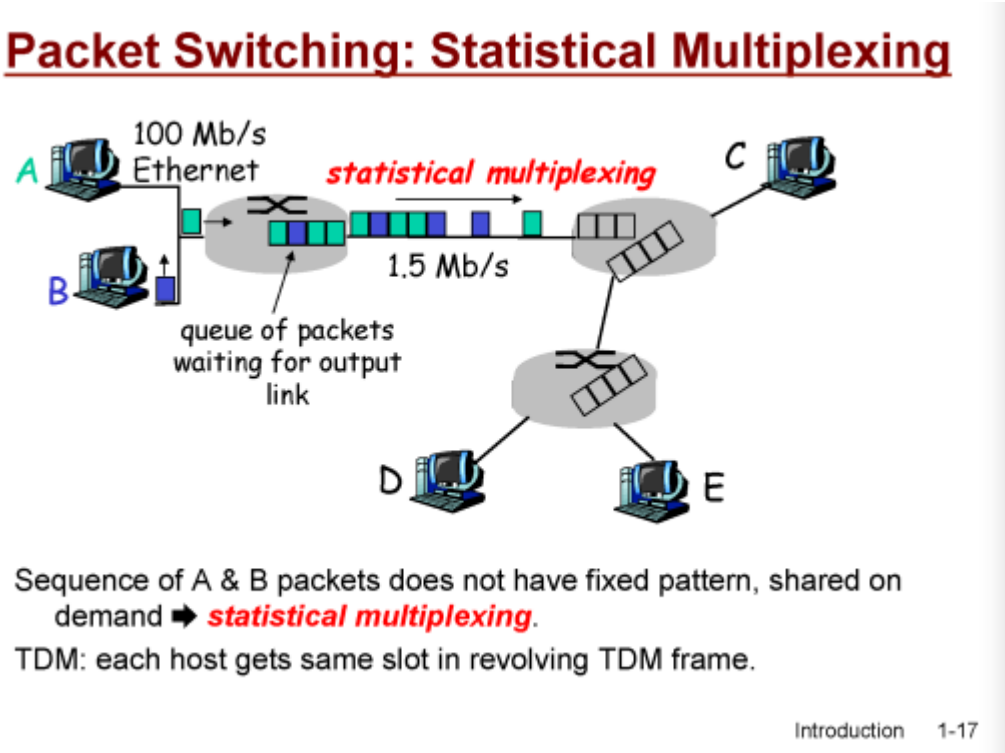


Analogy: When president travels, a CS path set up.

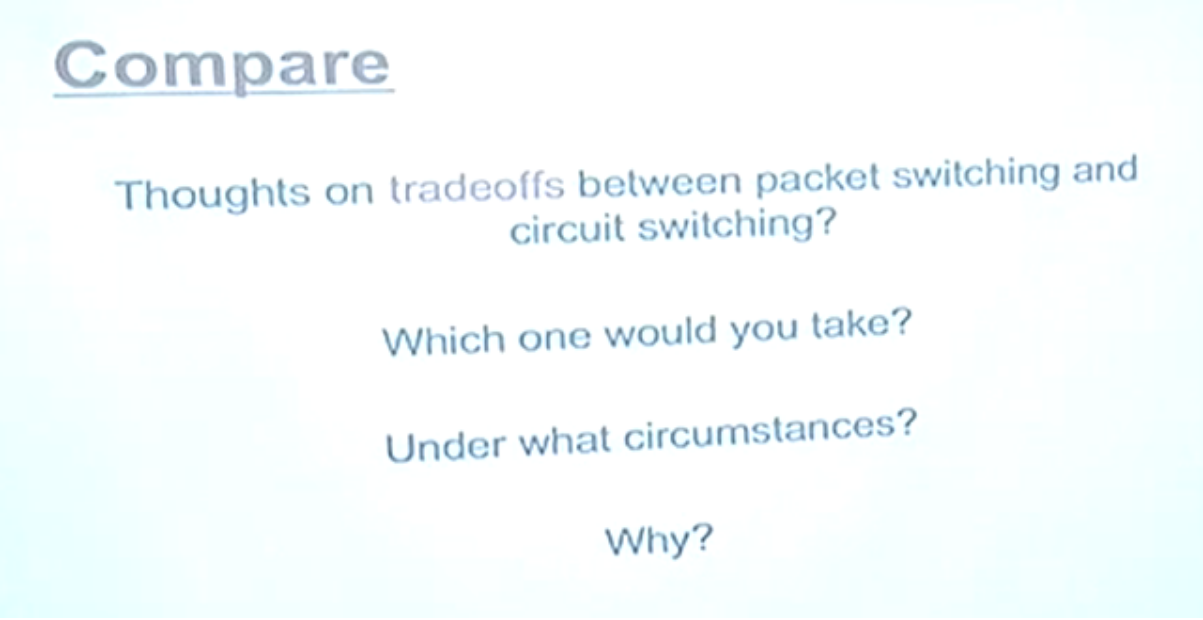
Introduction 1-1

Circuit Switching

- 출발지에서 부터 목적지까지 가는 길을 미리 예약을 해두고 특정 사용자만을 위해서 사용하기 위해 만들어 놓은 것.
- 유선전화망



- 단순히 유저가 보내는 메세지(패킷)를 패킷단위로 받아서 그때그때 올바른 방향으로 포워딩해둠(전달)



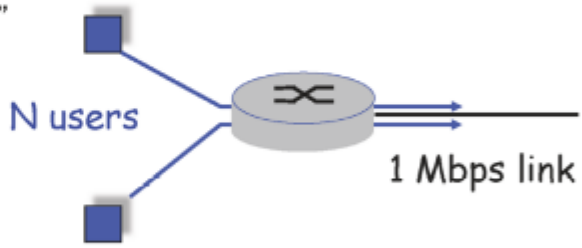
어떤 장단점들이 있길래 인터넷 초기 디자이너들은

Packet switching 을 선택했는가??

Packet switching versus circuit switching

Packet switching allows more users to use network!

- 1 Mb/s link
- each user:
  - 100 kb/s when "active"
  - active 10% of time
- circuit-switching:
  - 10 users
- packet switching:
  - with 35 users, probability > 10 active less than .0004



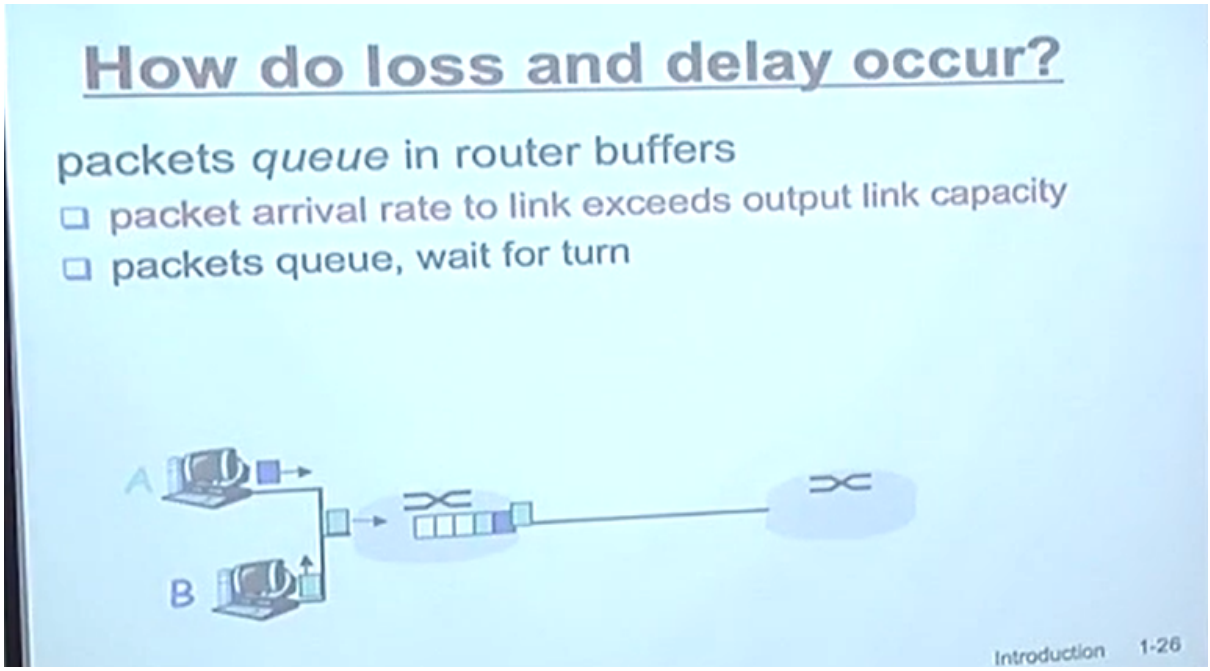
Q: how did we get value 0.0004?

Introduction 1-19

Packet VS Circuit

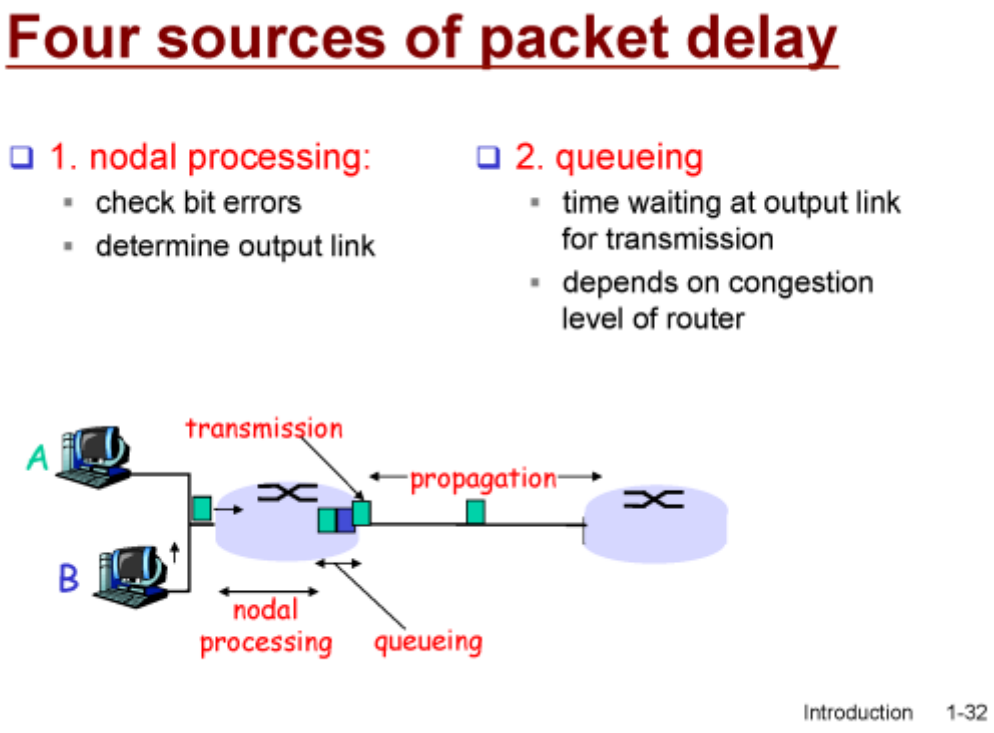
- 가정
  - 1 Mb/s link (초당)
  - each user : 1. 100kb/s | active 10% of time

- circuit-switching : 10명 가능 | 100kb/s * 10 = 1000kb/s ⇒ 1 Mb/s
- packet switching : 한계 없음 } 왜냐면? 들어오는대로 보내니까. 제약없음. 한꺼번에 10명이상 몰리지만 않으면 접속량이 분산되어 제약없이 사용할 수 있음. / 인터넷 사용 패턴에 더 적합.
  - (예를 들어, 네이버 뉴스기사를 볼때 접속시 클릭 한번만하고 글을 읽는 시간이 더 김. 글을 읽는 사이를 생각)



패킷 스위칭 사용시 생기는 문제들

- 딜레이
- loss(로스)

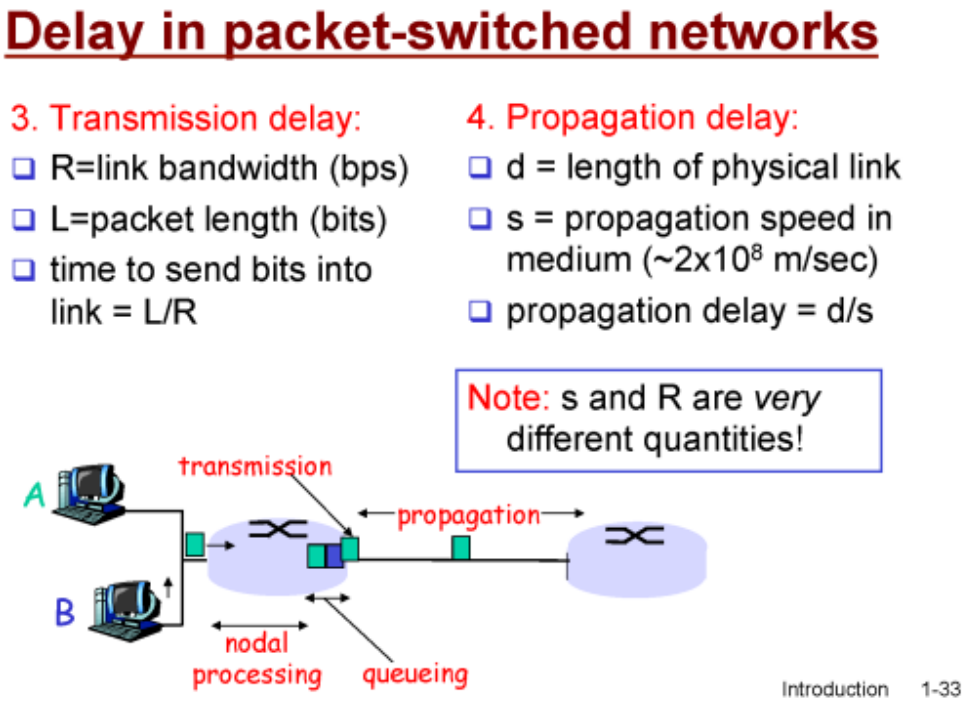


패킷 지연의 4가지 원인

1. nodal processing
    - 에러 체크
    - 출력할 링크 결정
2. queueing
    - 전송을 위해 출력 링크에 대기하는 시간
    - 라우터에 따라 혼잡 수준이 다름

나가기 전 딜레이 문제

나간 후 딜레이 문제



3. transmission delay
    - 첫번째 비트가 나가는 순간부터 시작해서 마지막 비트가 온전히 요 링크를 통로로 나갈때까지 걸리는 시간
    - [L = 패킷 길이(bits)] 를 [R = link bandwidth(bps, 대역폭)] 으로 나눈것 ⇒ L / R
    - bandwidth가 크면 클수록 transmission delay가 작아짐
4. propagation delay

- 마지막 비트가 링크에 올라와서 다음 라우터까지 도달할때까지 걸리는 시간
- [d = 링크 길이] 를 [s = 빛의 속도]로 나눈 것 ⇒ d / s

라우터에 패키지를 최초에 받았을때 검사시간 ⇒ nodal processing

큐에 집어넣고 큐에서 자기차례 올때까지 기다리는 시간 ⇒ queueing

제일 앞에서서 파이프에서 온전히 뿜어져 나올때까지 걸리는 시간 ⇒ transmission delay

마지막 비트가 그다음 라우터까지 도달할때까지 걸리는 시간 ⇒ propagation delay

딜레이를 줄일수 있는 방법

nodal processing ⇒ 라우터 성능 좋은거 사는것? 당연히 CPU 성능도 좋겠지 ⇒ 라우터 성능 개선

transmission delay ⇒ 케이블 공사? ⇒ 대역폭을 늘리면 딜레이가 줄어듬.(톨게이트 공사)

queueing ⇒ 가장 까다로운 딜레이, 사람이 몰리는 시간대(패턴) 같은 건드릴 수 없는 것.

- queue 의 크기는 무한대 이지 않다 ⇒ 최악의 경우 큐의 저장공간보다 더많이 들어오게 되면?⇒ 방법없음
- 그냥 버림 ⇒ 계속 몰리다보면 유실이 일어남 ⇒ 거의 90% 이상 큐가 터지는 바람에 일어남.

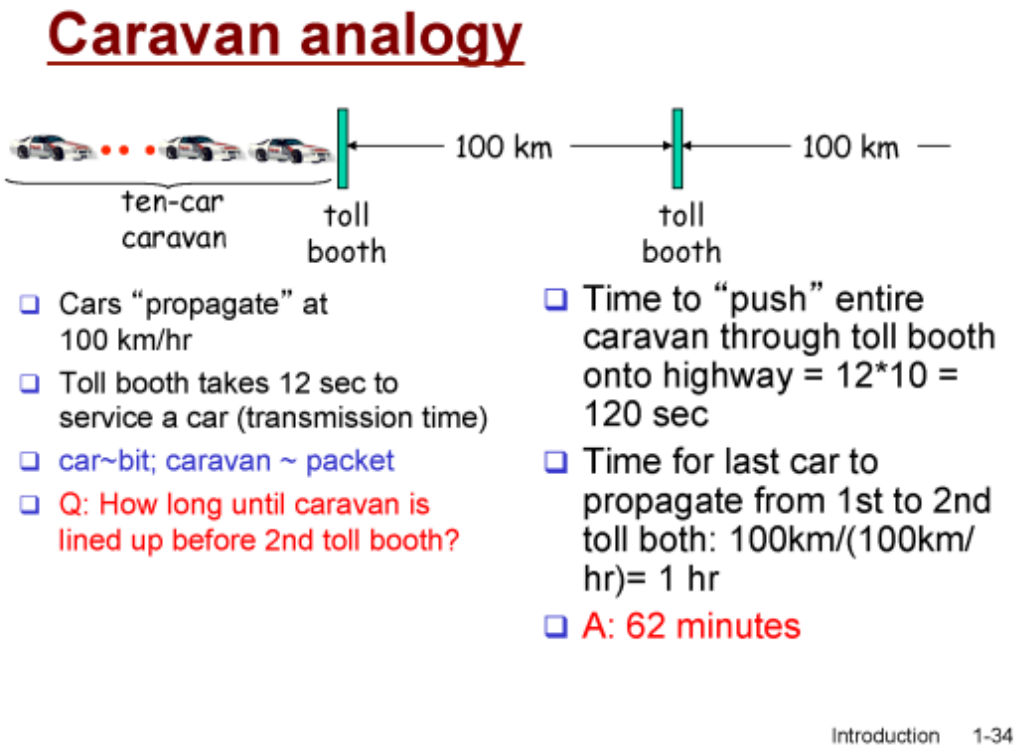
근데 UDP면 모를까 TCP는 유실이 없게 전달이 되는데 어떻게 가능한거지?

- 유실이 없기 위해서는 **재전송** 필요

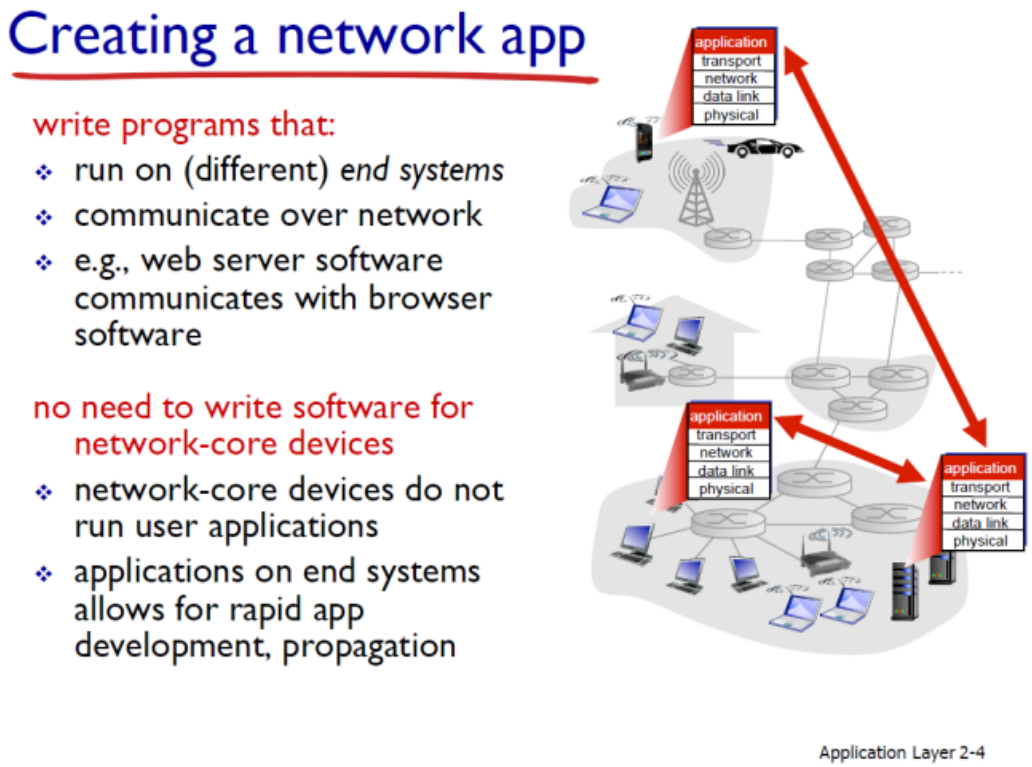
1. 직전 라우터가 재전송해주는 방법
2. 아예 처음부터 시작컴퓨터(클라이언트)가 재전송해주는 방법

⇒ 선택한 방법은? **2번**

- dump core: 최대한 데이터를 빨리 전송시키기 위한 기능 ⇒ 단순작업에 극대화 되어 있음 ⇒ 유실을 막기위해 재전송같은 기능을 넣어서 효율을 따지면 무지하게 느려짐. ⇒ 쓸모없어짐
- 재전송에 대한 기능은 A (TCP), D (TCP)에 몰아넣음. ⇒ 모든 기능적 매커니즘들은 edge에다가 밀어넣음.



- 패킷 을 자동차에 비유하여 딜레이 문제점을 보여줌.



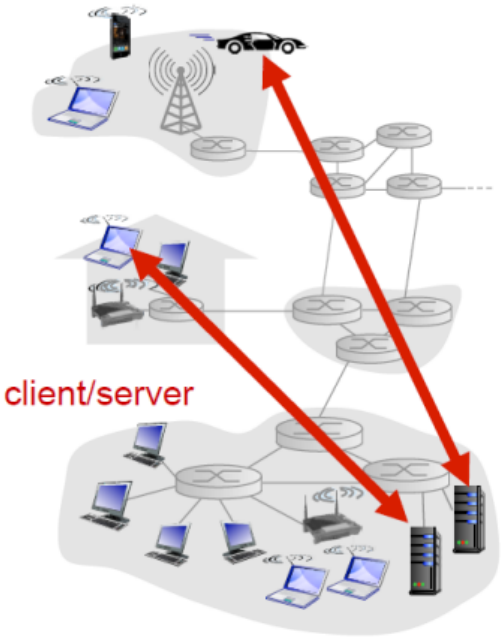
계층	예시
Application	HTTP
Transport	TCP UDP
Network	IP
Data link	Wifi LTE/3G Ethernet(유선 인터넷)
Physical	

- 네트워크 edge에만 존재하고 라우터에는 존재하지 않음.

네트워크 코어 장치용 소프트웨어를 작성할 필요가 없습니다.

- 네트워크 코어 장치는 사용자 응용 프로그램을 실행하지 않습니다.
- 엔드 시스템의 애플리케이션은 신속한 앱 개발, 전파를 허용합니다.

Client-server architecture



server:
- always-on host
- permanent IP address
- data centers for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Application Layer 2-6

클라이언트-서버 아키텍쳐

서버(웹서버)

- 항상 호스트가 켜져있음(24시간)
- 영구적인 IP 주소(인터넷 상에 존재하는 모든 컴퓨터들은 고유한 주소를 가지고 있음)
- 확장을 위한 데이터 센터

클라이언트(웹브라우저)

- 서버와 통신
- 간헐적으로 연결될 수 있음
- 동적 IP 주소일 수 있음
- 서로 직접 소통하지 않음

Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

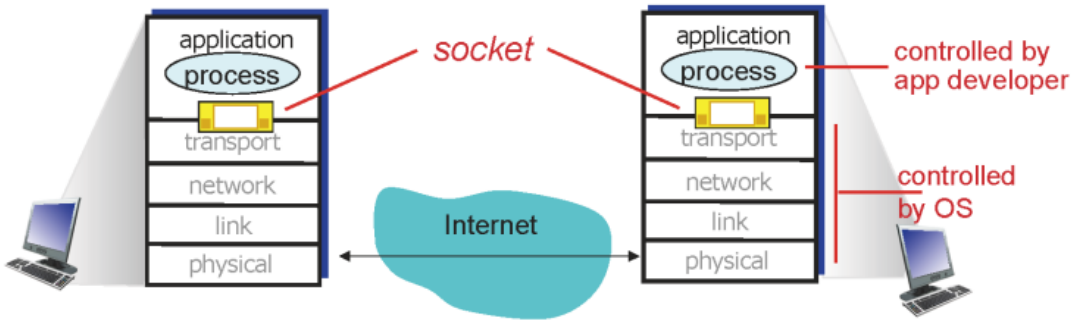
Application Layer 2-8

클라이언트 프로세스와 서버 프로세스와의 의사소통, 통신

프로세스들 사이의 통신

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Application Layer 2-9

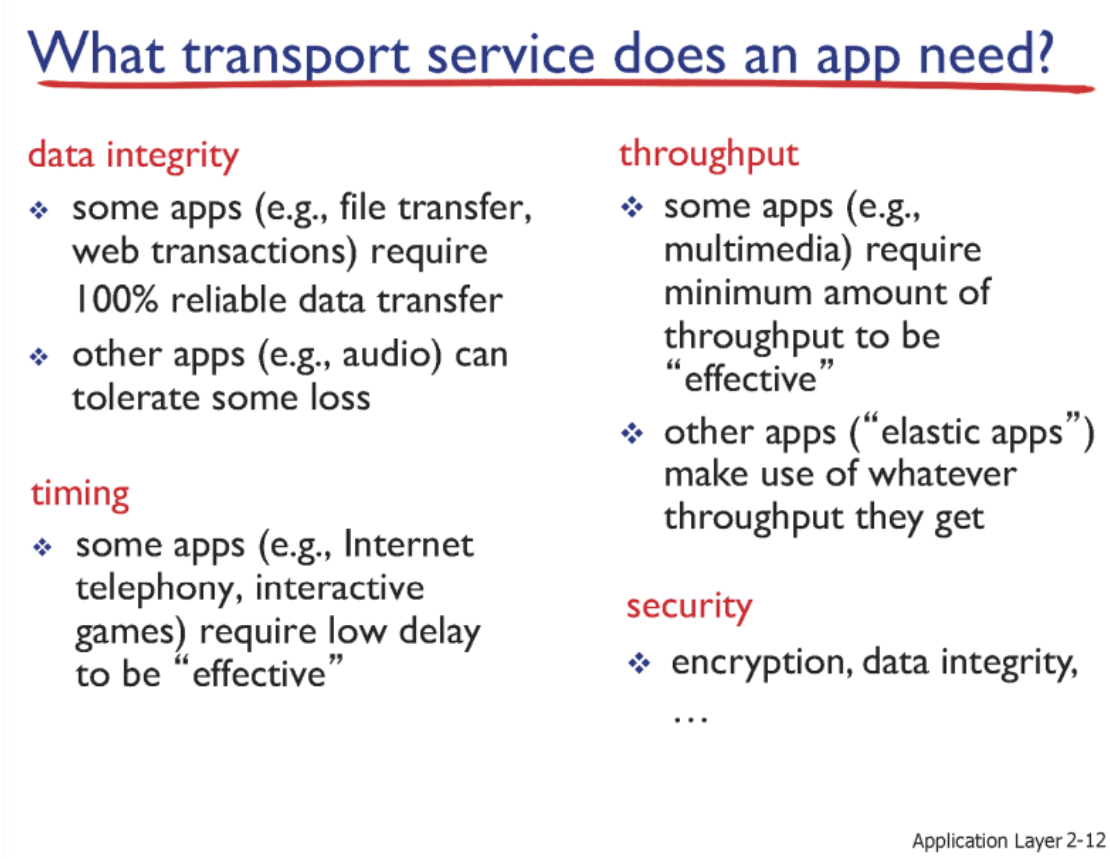
사전에 프로세스 간을 연결시켜야 하는데, 다른 프로세스의 소켓이랑 연결하고 싶다는 의사표현을 해야함. 즉, 다른 프로세스 소켓의 주소를 알아내야함. ⇒ 소켓을 인덱싱 하는==소켓의 주소 역할을 하는 뭔가가 인덱스가 필요함⇒ 그게 바로 IP address(인터넷 상에 존재하는 컴퓨터를 지칭하는 주소)와 Pote(하나의 컴퓨터 안에 수많은 프로세스 중에서 특정 프로세스를 지칭하는 역할)

웹브라우저를 실행시키고 다른 컴퓨터에 위치한 프로세스와 연결된 상태(구글, 네이버 등등) ⇒ 네이버 접속하고 싶을때, 네이버 프로세스에 해당하는 socket의 주소를 입력해야지 접속됨. www.naver.com 입력한다

하지만 네이버 컴퓨터의 IP주소와 네이버 웹서버 프로세스가 돌고 있는 socket에 해당하는 Pote number(80)를 입력해야지 실제로 연결이 됨. ⇒ 사람들은 이렇게 실행하지 않음 ⇒ 그래서 이용하는 사람들을 위해 그냥 알파벳 써라(www.naver.com)가 됨. ⇒ 내부 DNS라는 시스템에 의해서 네이버의 IP주소로 변환됨

웹서버를 운영하는 거의 모든 서버들이 Pote 80번을 쓰고 있음. ⇒ 왤까? ⇒ 서버는 24시간 켜져있어야하고, 주소가 일정해야함 ⇒ 그런데 네이버, 다음, 구글 등 각각틀린데, 그주소를 해석해주는게 DNS 인데, pote넘버까지 다 다르면 귀찮으니 다 동일하게 쓰자고 합의.

- 송신 프로세스는 수신 프로세스에서 소켓으로 메시지를 전달하기 위해 문의 반대편에 있는 전송 인프라에 의존합니다.



app에는 어떤 transport 서비스가 필요한가?

- data integrity(데이터 무결성): 내가 보낸 데이터가 유실되지않고 목적지까지 온전히 도착하는 것.
- throughput(처리량) : 내가 보낸 데이터가 최소 (1G) 이상 용량이 나오면 좋겠음.
- timing : 내가보낸 데이터가 제한시간 내에 도착했으면 하는 것.
- security(보안) : 데이터가 안전했으면 좋겠다.

⇒ 하지만 실제로 제공해주는 서비스는 data integrity 하나뿐.(TCP가 제공해주는 거지 UDP는 제공 안해줌)

만약에 추가적인 transport 서비스가 필요하면? ⇒ 알아서 기능 만들어서 사용(뭐 깔아라)

timing과 throughput는 같은 기능 아닌가? ⇒ timing은 내가보낸 packet이 제한시간 범위 안에 도착해야 한다는 것이고, throughput은 1초에 어느정도의 양이 도달해야 한다는 소리임. 모든 packet이 타이밍을 맞출 필요는 없음.

- timing(음성) / throughput(영화 다운로드) 를 예시로 들면 이해하기 편함

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Web and HTTP

First, a review…

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,

`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

Application Layer 2-18

HTTP(hypertext tansfer protocol)- hypertext를 전달하는 프로토콜

hypertext - text인데 중간중간에 다른 text를 지칭하는 link가 있음

- request : 내가 지금 원하는 hypertext 파일 이름 요청.
- response : 받으면 서버는 파일을 읽어서 response 에 담아서 전달.

HTTP overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

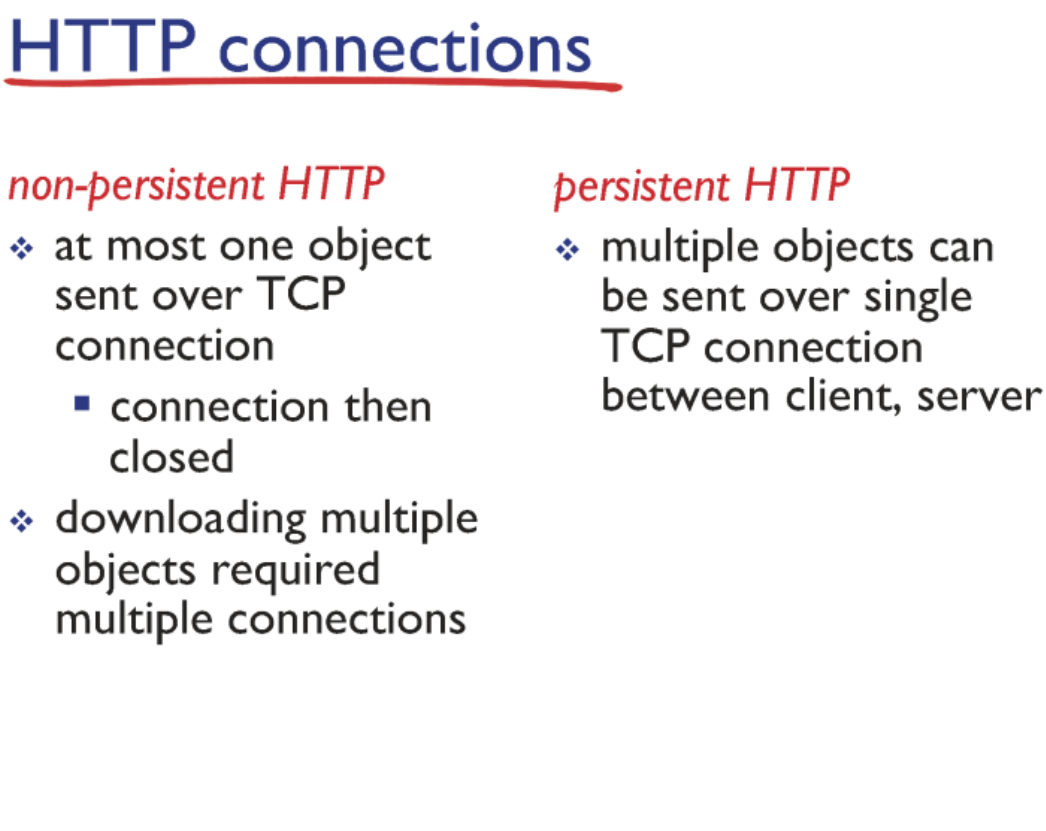
aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Application Layer 2-19

- uses TCP: HTTP는 application layer 프로토콜이기 때문에 당연히 transport layer 프로토콜을 사용함 ⇒ TCP를 사용하기 때문에 request/response 이전에 TCP connection 이라는 것을 해줘야함.
- HTTP is "stateless"(상태가 없음) : HTTP는 엄청 단순해서 request가 들어오면 단순히 request 해당하는 파일을 읽어서 reponse에 담아서 보내주면 끝. 더이상 기억안함. 요청이 들어오면 처리하고 처리하고 끝. ⇒ 상태를 기억하지 않음

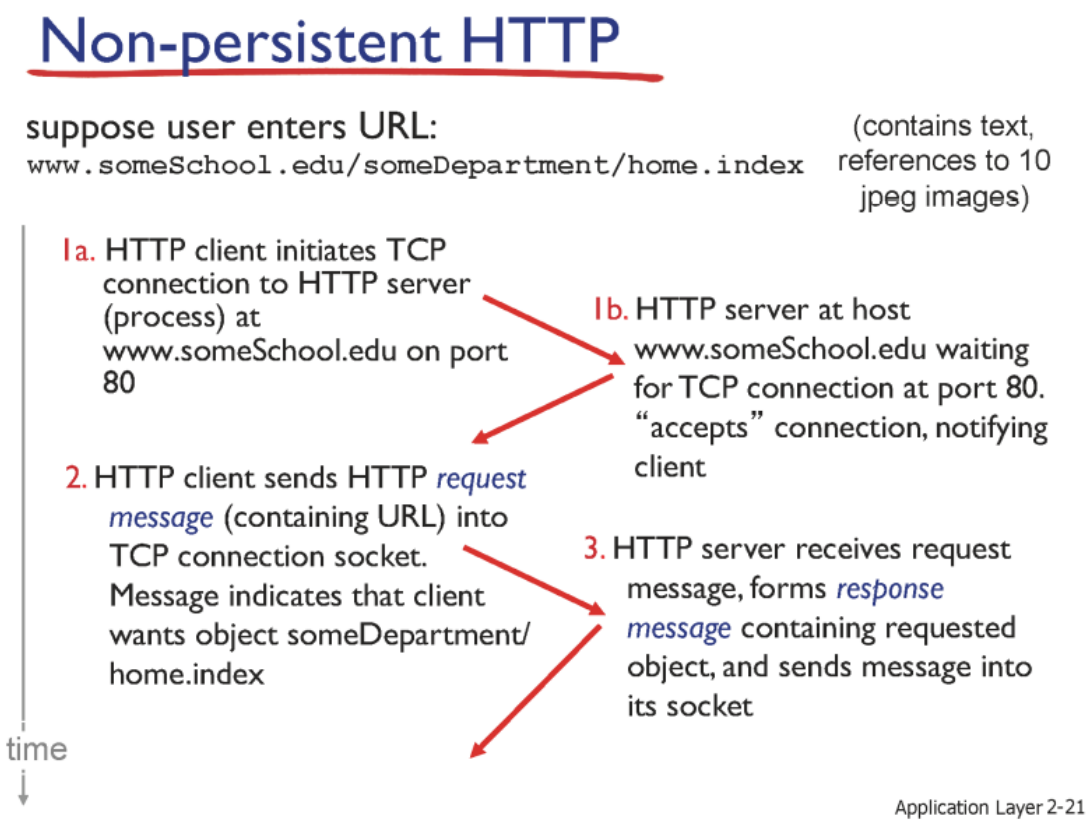


HTTP가 TCP를 사용하는 방식에 따라서 나뉘는 2가지 방식

프로세스와 프로세스 사이 HTTP 메세지를 request/reponse 주고받을려면 TCP 커넥션을 만들어 메세지를 주고받고 끊으면 non-persistent HTTP (지속성이 없는) / 끊지않고 유지하면서 재사용하면 persistent HTTP (지속성 있는)

- 예를들어 네이버, 웹브라우저가 웹서버한테 기사페이지(hypertext)을 읽어올려하는데, 그 웹페이지에는 다른 objects 그림파일 이라던가 여러개 있는 상태. 그러면 그 메인페이지와 다른 그림파일 여러개를 전부 가져와야하는데, ⇒ 우선 메인페이지를 요청하기 위해서 TCP 커넥션 생성하여 메인 페이지에 대한 request 를 보냄 ⇒ 그럼 메인페이지에 대한 reponse를 주고 ⇒ request/response 한번 왔다갔다 trnasgation? 끝나면 TCP커넥션 끊고 ⇒ 다시 TCP커넥션 만들어서 각각 그림파일을 가지고 오게되면 이게

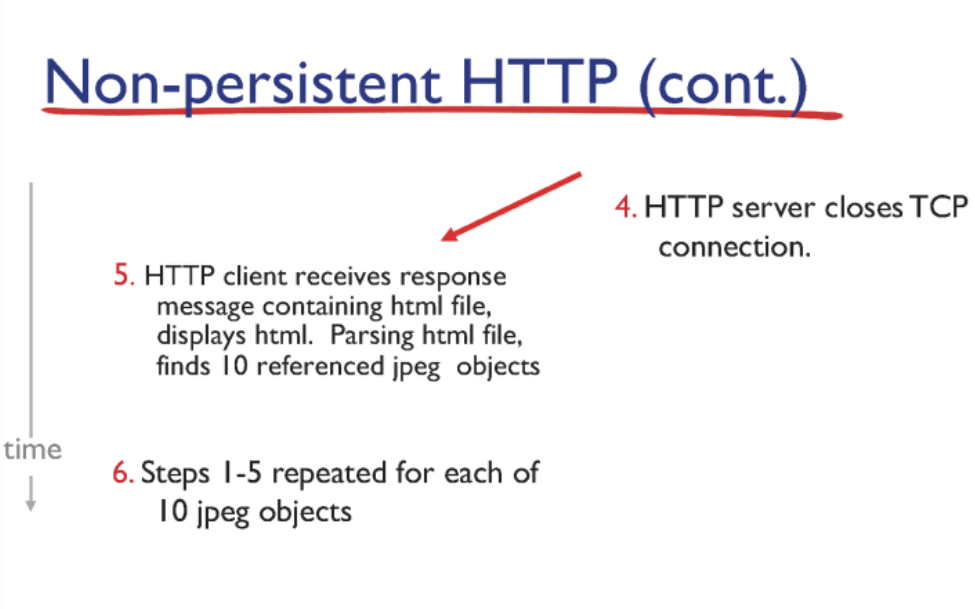
non-persistent



1. TCP 컨넥션 설치를 위한 메세지가 왔다갔다 하는 단계

2. TCP 커넥션 연결하고, 클라이언트가 Department/home.index 개체를 원한다는 메시지를 전달

3. request 메세지를 받으면, 요청된 객체를 포함하는 응답 메시지를 형성하고 소켓에 메시지를 보냅니다.



4. TCP 커넥션을 끊음. (완전히 끊어버린게 아니라 난 이미 내가 할일을 다했으니 끊을 준비를 하는것) 실제로 끊는 것을 client가 최종 판단

5. http 클라이언트는 html 파일이 포함된 response 메시지를 수신하고(받고) html을 표시합니다. html file을 분석하고 10개의 참조된 jpeg 객체를 찾습니다.

6. 1~5단계를 각 10개의 object 에 반복.

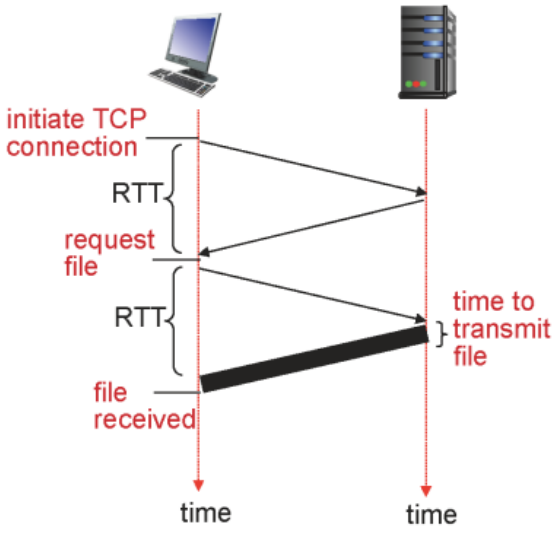
non-persisten 나 persistent나 HTTP 개념에서 봤을때는 동일한 건데 차이점이라면, TCP 자체를 다시 만드느냐 안 만드느냐의 차이

Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time:

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time =
  2RTT+ file transmission time



Application Layer 2-23