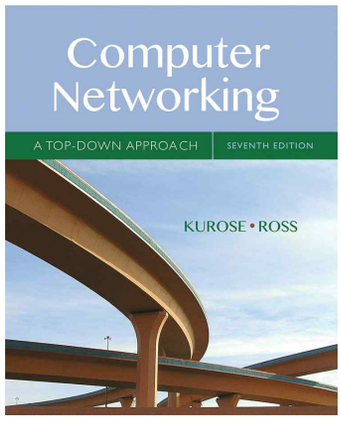
Chapter 2
Application Layer

Seongwook Youn
Department of Software
Korea National University of Transportation



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
April 2016

The slides are adapted from the publisher's material

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP
2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications
2.6 video streaming and content distribution networks (CDNs)

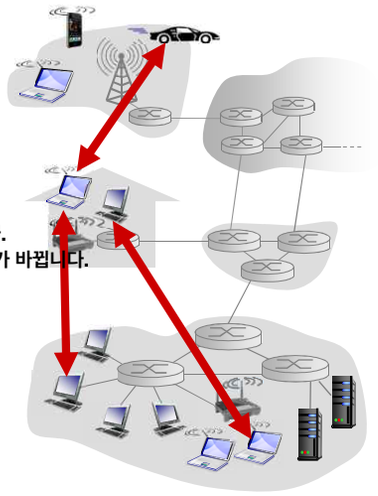
Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

항상 들어와 있는 서버는 없습니다
임의의 두 개의 p2p들이 통신을 합니다.
peers들은 쉬엄쉬엄 연결하고 IP주소가 바뀝니다.

examples:

- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



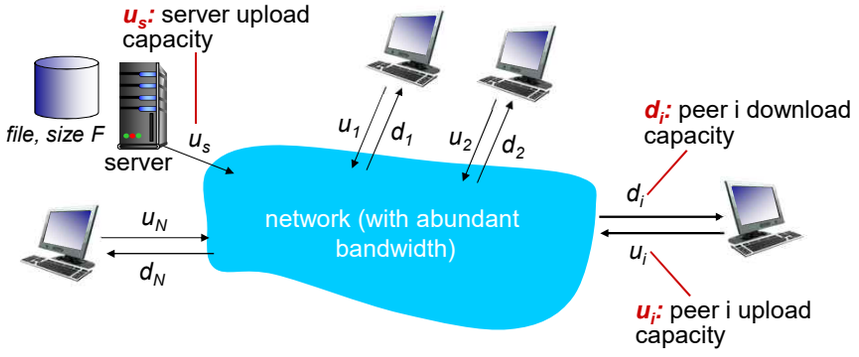
파일 분산 시스템에서 두 아키텍쳐를 비교해 봅시다.

File distribution: client-server vs P2P

F을 N명의 peer들에게 보내는 시간

Question: how much time to distribute file (size *F*) from one server to *N* peers?

- peer upload/download capacity is limited resource



P2P일 때와 함께 알아두세요

File distribution time: client-server

[0] 서버가 실제 네트워크 망에 올리는데 걸리는 시간: F(전송크기)/Us(전송속도)

N명 꺼를 해줘야 하기 때문에 NF/Us

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send N copies: NF/u_s

N개

전송크기/전송속도

N명의 사람들이 파일을 올리는데 걸리는 시간

- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}

[1] 각각의 사람들이 파일을 다운로드 받는데 걸리는 시간: F/d(min)

#min은 가장 오래걸리는 시간을 말합니다.

다운로드 받는데 제일 오래 걸리는 친구를 기준으로 잡습니다.
모든 사람들이 다 다운로드 받아야 하니깐요!

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

client <-> server 분배 시간은

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

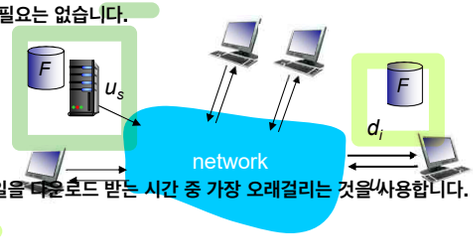
increases linearly in N

서버가 메일을 올리는 시간과
클라이언트들이 파일을 다운로드 받는 시간 중
더 오래 걸리는 것을 선택합니다.

File distribution time: P2P

[0] 서버가 네트워크 망에 올리는 시간은 client<->server와 동등하게 F/Us입니다.
임의의 peer들이 직접 통신하므로 N개를 올릴 필요는 없습니다.

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download *NF* bits
  - max upload rate (limting max download rate) is $u_s + \Sigma u_i$



[1] peer들이 파일을 다운로드 받는 시간 중 가장 오래걸리는 것을 사용합니다.

time to distribute F to N clients using P2P approach

하나만 업로드해도 되고, 파일이 필요한 사용자들은 다 다운로드 받으면 됩니다.

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

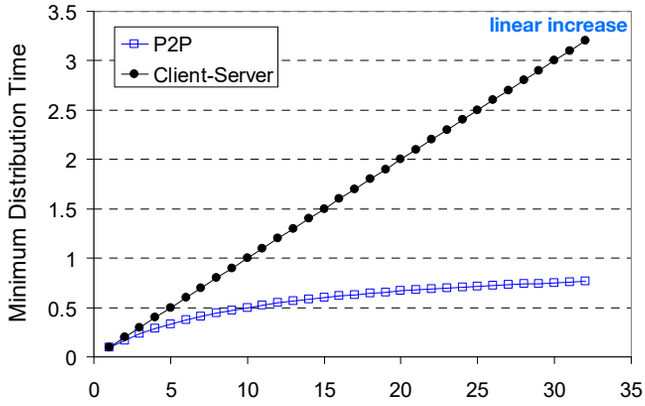
NF / (서버 업로드 + 사용자들이 업로드)

increases linearly in *N* …

… but so does this, as each peer brings service capacity

Client-server vs. P2P: example

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



기억해야 합니다.
client-server는 사용자가 늘수록 시간이 선형적으로 증가합니다.N
P2P는 어느 정도 늘어나지만, 이후로는 늘어나지 않습니다.
=>P2P 방식이 더 유리하다는 말입니다.

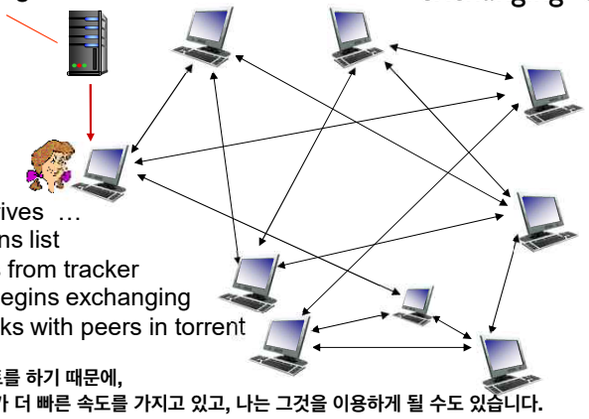
P2P file distribution: BitTorrent

Bit Torrent에서 파일을 256kb 덩어리로 잘라줍니다.

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

참여하는 peer들이 필요한 파일을 찾게 도와줍니다.
tracker: tracks peers participating in torrent

256kb 파일을 교환하는 피어그룹
torrent: group of peers exchanging chunks of a file

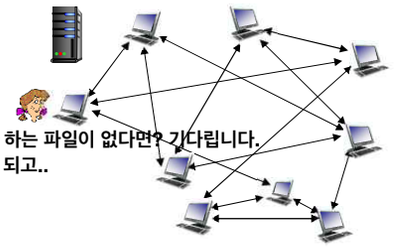


Alice arrives …
… obtains list of peers from tracker
… and begins exchanging file chunks with peers in torrent

주기적으로 업데이트를 하기 때문에,
다음번에는 다른 애가 더 빠른 속도를 가지고 있고, 나는 그것을 이용하게 될 수도 있습니다.

P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



축적하다

도착했는데 받고자 하는 파일이 없다면? 기다립니다.

tracker에 등록이 되고..

나는 다운로드 받으면서 동시에 다른 사람을 위해 내가 가지고 있는 파일을 업로드 할 수 있음

- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go 고객 변심의 의미지만, 피어들은 내 맘대로 나갔다 들어왔다 합니다.
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

내가 영화 파일 다운로드 받았다면, 떠나거나(이기적) 남아있기(이타적)

BitTorrent: requesting, sending file chunks

같은 시간이라도, 다른 피어들은 다른 서브셋을 가지고 있습니다. (네트워크 속도 등)
[0] 때문에, 주기적으로 각 peer들이 가지고 있는 chunks의 리스트를 요청합니다.
[1] 보기 어려운 파일부터 다운로드를 요청합니다.

requesting chunks:

- at any given time, different peers have different subsets of file chunks

주기적으로 피어가 가지고 있는 chunks들을 봅니다.

- periodically, Alice asks each peer for list of chunks that they have

드문 파일 부터 다운로드 받게 합니다.

- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

[0] chunks를 4명의 peer들에게 파일들을 보내고 있습니다.
[1] 10초마다 4명이 다른 4명에게 파일을 보내도록 합니다.
[2] 30초마다 무작위로 다른 peer를 선택합니다.

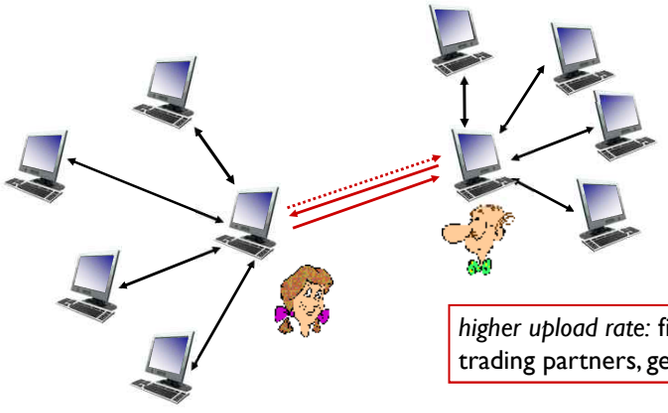
BitTorrent: tit-for-tat

(3) 까지 되었다면, 둘은 최고의 속도로 전송됩니다
그러니깐 서로에게 필요한 파일들을 서로가 가지고 있는 것이 베스트란 말입니다.

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

Distributed Hash Table (DHT)

- Hash table
- DHT paradigm
- Circular DHT and overlay networks
- Peer churn

Simple Database

Simple database with(key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	……
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		……
Lisa Kobayashi	9290124	177-23-0199

Hash Table을 통해 개념적으로 key value 값을 빨리 찾아갈 수 있게 한다는 말입니다.

Distributed Hash Table (DHT)

- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

[0] 분산시킨 (key, value) 쌍을 peer들에게 줍니다.
[1] peer들은 key를 통해 database에 접근할 수 있습니다.

Assign key-value pairs to peers

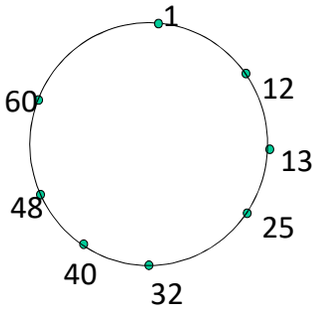
--> 가장 가까운 아이디를 가지고 있는 아이를 고르게 합니다.

- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1 끝까지 가면 다시 앞으로 간답니다.

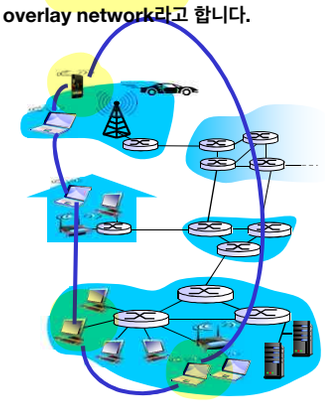
Circular DHT

각각의 피어들이 본인의 바로 뒤만 알고 있다고 가정 합시다.

- each peer *only* aware of immediate successor and predecessor.



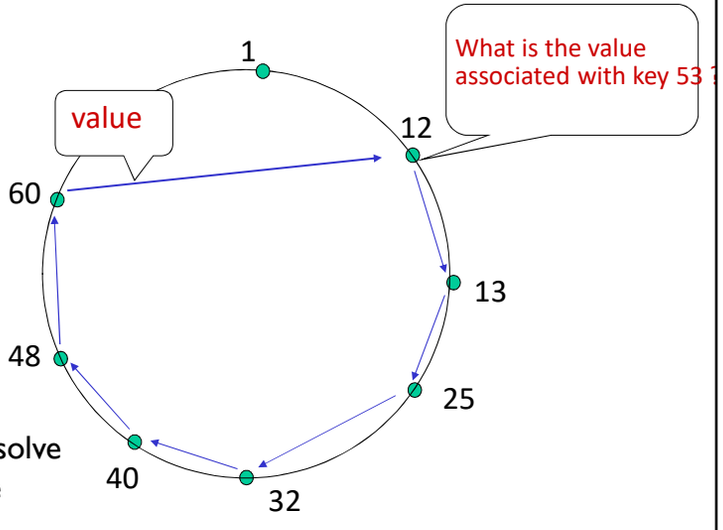
가상의 연결 네트워크를 만듭니다. 네트워크 위에 다른 연결 네트워크를 만드는 것을 **overlay network**라고 합니다.



"overlay network"

An **overlay network** is a computer **network** that is built on top of another **network**. Nodes in the **overlay network** can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying **network**.

Resolving a query

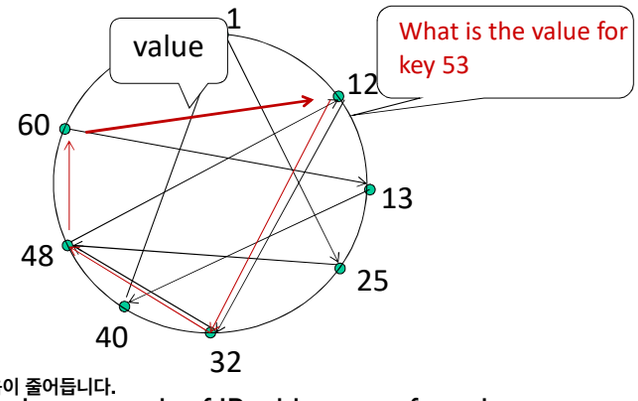


최대 N번의 검색을 합니다.

O(N) messages on avgerage to resolve query, when there are *N* peers

지름길을 만들어 놓으면, 좀 더 빨리 갈 것입니다.

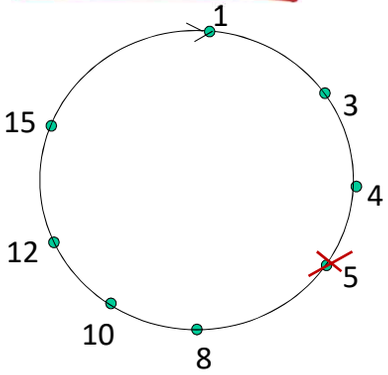
Circular DHT with shortcuts



아까보다 걸음이 줄어듭니다.

- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with *O(log N)* neighbors, *O(log N)* messages in query

Peer churn



example: peer 5 abruptly leaves

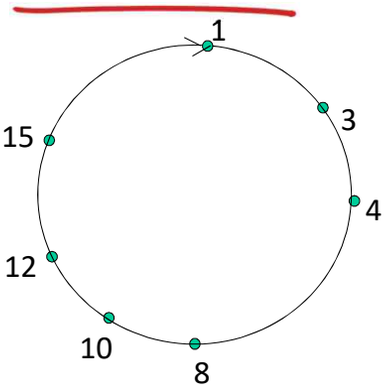
이때까지는 내 바로 뒤에 누가 있는지만 알고 있었습니다.
바로 뒤와 그 뒤까지 누가 있는지 알고 있습니다.

handling peer churn:

- ❖peers may come and go (churn)
- ❖each peer knows address of its two successors
- ❖each peer periodically pings its two successors to check aliveness
- ❖if immediate successor leaves, choose next successor as new immediate successor

5번이 없어졌습니다.
4번을 바로 뒤에 8번이 오니깐 5번이 없어진 걸 압니다.
4번은 8번을 바로 뒤에 오는 애로 정합니다. 8번에게 "너 뒤에는 누구니?" 라고 물어봅니다.
그리고 이 정보를 3번에게 알려줍니다.

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

Chapter 2: outline

2.1 principles of network applications
- app architectures
- app requirements

2.2 Web and HTTP

2.3 electronic mail
- SMTP, POP3, IMAP

2.4 DNS

2.5 P2P applications

2.6 video streaming and content distribution networks (CDNs)

넷플릭스와 같이 스트리밍 문화가 발전하면서 발달되었습니다. 2021년 기준 최신 1년 이내 기술

Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B users? 하나의 서버가 전 세계를 처리한다는 건 말이 안 됩니다.
  - single mega-video server won't work (why?)
- challenge: heterogeneity 다 다른 처리량을 가지고 있는데 어떻게 효율적으로 사용할 수 있을까요?
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor) 유선/무선 환경 등 사용자들이 갖고 있는 환경은 다 다릅니다.
- *solution:* distributed, application-level infrastructure
  분산된, application-level infrastructure을 제공하는 것입니다.
  인터넷이 느리다면? 화질을 낮게 해주는 서비스가 있습니다.

Q2 2019 – # of subscriber of Netflix is 151M
Apr 2019 - # of subscriber of YouTube is 1.9B

Multimedia: video

비디오는 일련의 이미지를 일정한 속도로 뿌려줍니다.

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits

이미지 사이에 중복이 있구나!! -> 데이터 중복은 좋지 않아요!!

- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

frame *i*

frame *i+1*

연속적인 이미지들 중 많은 부분은 같은 부분이니깐 바뀌는 부분만 redundancy(전송) 사용 합니다.

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

멀티미디어 강의에서 다룹니다.

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

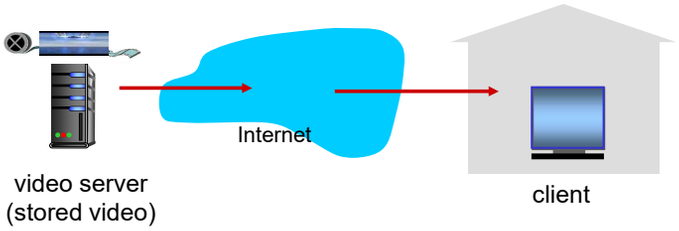
frame *i*

frame *i+1*

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

Streaming stored video:

simple scenario:



서버
[0] 서버는 비디오 파일을 다양한 chunks로 나눕니다.
[1] 각각의 chunk는 다른 속도로 저장되고, encoded 됩니다.
[2] manifest file이 다른 chunks에 대한 URL를 보여줍니다.

클라이언트
[0] 주기적으로 bandwidth를 측정합니다.
[1] manifest의 상담을 걸쳐 하나의 chunk를 다운 받습니다.

Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

인터넷 속도에 따라 최고화질~저화질로 보냅니다.
보이다
각각 다른 조각들에대한 URL이 정의되어 있습니다.
가능하면 현재 속도(bandwidth)에서 제공할 수 있는 가장 좋은화질을 보냅니다.
--> 당연한 말을 함
시간에 따라 당연히 처리량이 바뀌고 그에따라 화질도 바뀝니다.

Streaming multimedia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *"intelligence" at client:* client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

클라이언트가 결정할 수 있다고 합니다.
어디로, 언제 보내는지 클라이언트 쪽에서 detect가 됩니다.

Content distribution networks

동시에 사용하는 많은 사용자들에게 어떻게 stream 할 것인가요?

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

큰 서버 하나를 놓는 경우

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

….quite simply: this solution *doesn't scale*

별로 좋지 않음
사용자가 많아지면 감당 불가

Content distribution networks

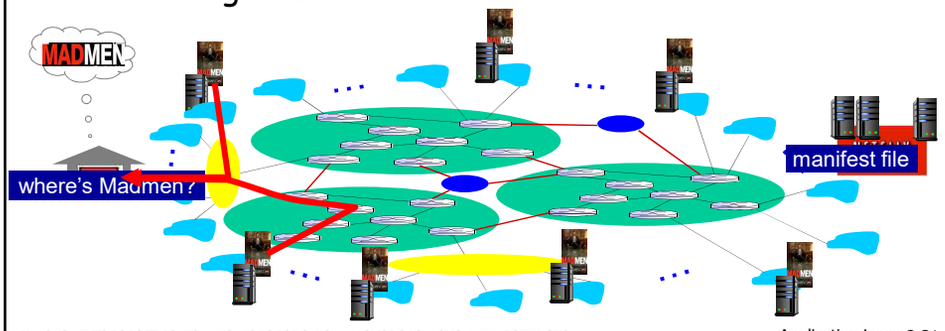
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

지역적으로 여러 군데 나눕니다.

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks 사용자와 가까이 놓습니다.
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight access network 근처에 큰 cluster를 둡니다. 비용이 많이 들겠습니다. :)
    - *lower maintenance and management overhead*

Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



요청한 클라이언트와 가능한 가까이에 있는 서버에서 서비스를 해줍니다.

Content Distribution Networks (CDNs)

Internet host-host communication as a service

OTT challenges: coping with a congested Internet

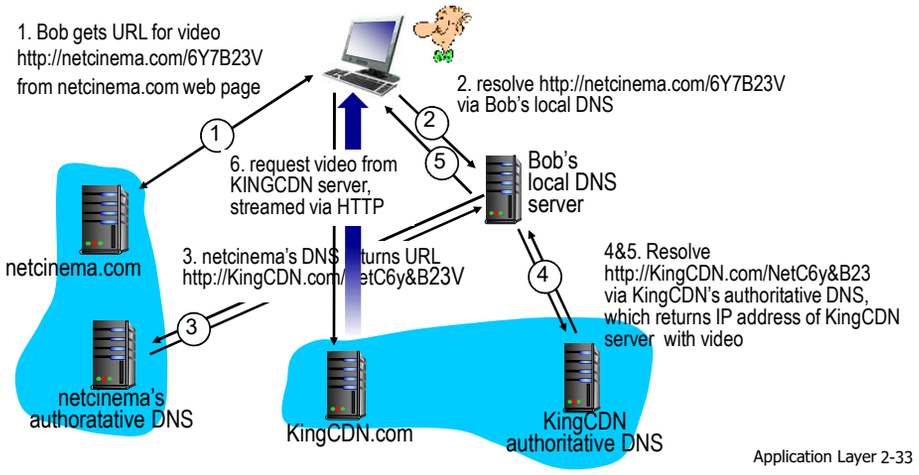
- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

통계에 기반하여
클라이언트가 좋아할 만한 것들을 주변에 배치시키는 것입니다.

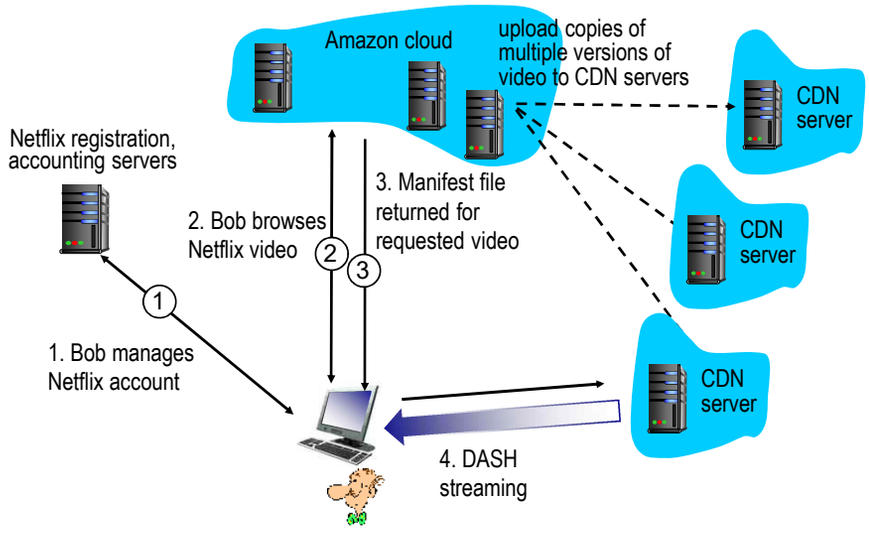
more .. in chapter 7

CDN content access: a closer look

Bob (client) requests video http://netcinema.com/6Y7B
- video stored in CDN at http://KingCDN.com/NetC6y&B23V



Case study: Netflix



Chapter 2: summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT

Marc Andreessen

- Co-creator of Mosaic (web browser),1993
- Co-founder of Netscape, 1994
- Developed SSL protocol
- Currently, co-founder and general partner of venture capital firm Andreessen Horowitz
- BS in Computer Science from UIUC