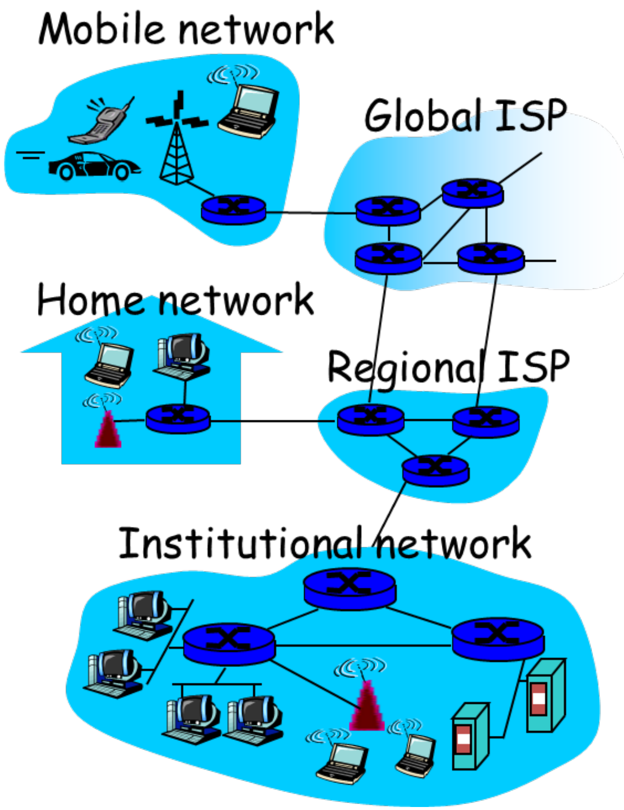
Course on Computer Communication and Networks

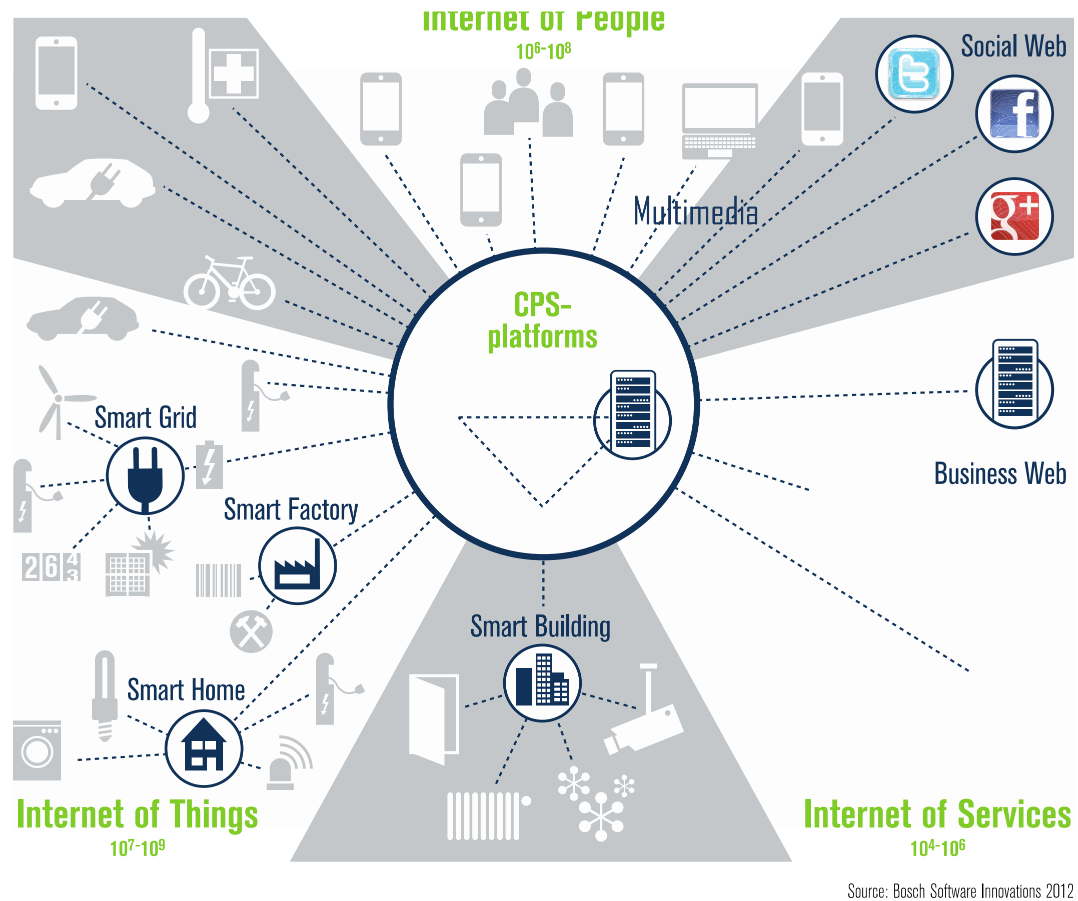
Lecture 11
Continuously evolving Internet-working
Part 1: p2p networking, media streaming, CDN
(*TBC in part 2: QoS, traffic engineering, SDN, IoT*)

EDA344/DIT 420, CTH/GU

Based on the book Computer Networking: A Top Down Approach, Jim Kurose, Keith Ross, Addison-Wesley.

Internet & its context….

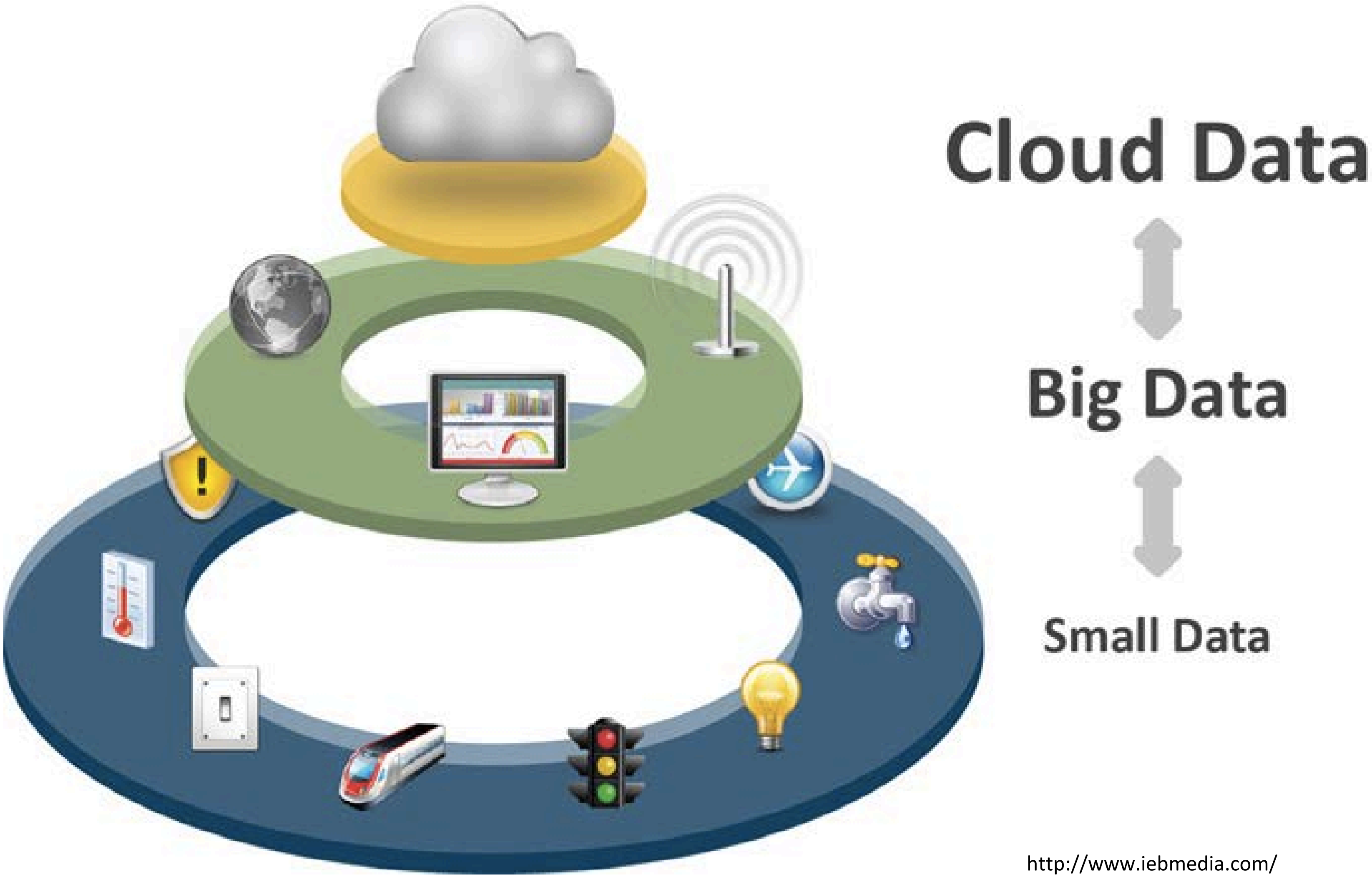




approx 10 yrs ago

continuous evolution ….

Internet, Data processing and Distributed Computing in interplay



http://www.iebmedia.com/

Recall: Internet protocol stack layers

Application: protocols supporting *n/W applications*

http (*web*), smtp (*email*),

p2p, media streaming, CPS/IoT apps

Transport: end2end data transfer protocols

UDP, TCP

Network: routing of datagrams, connecting different physical networks

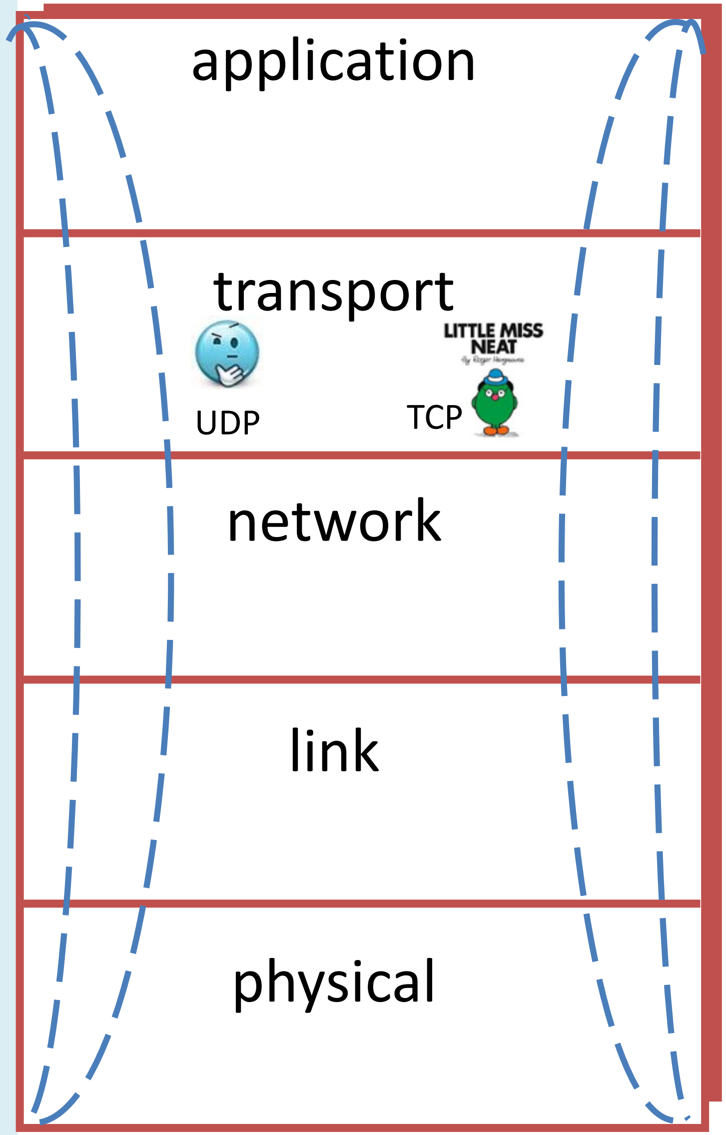
IP addressing, routing,

Virtualization, traffic engineering, Software Defined Networks

link: data transfer between neighboring nodes

Ethernet, WiFi, …

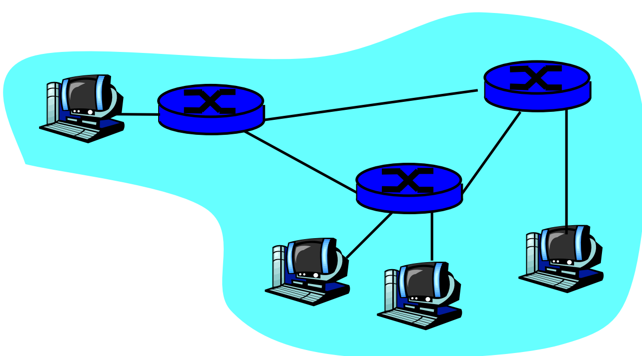
physical: protocols for bit-transmission/receipt on the physical medium between neighboring nodes



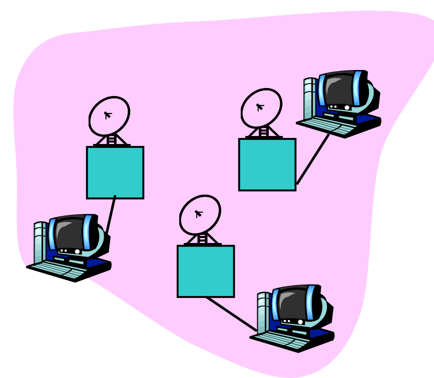
Recall:
the Internet concept: virtualizing networks

1974: multiple unconnected nets
- ARPAnet
- data-over-cable networks
- packet satellite network (Aloha)
- packet radio network

… differing in:
- addressing conventions
- packet formats
- error recovery
- routing



ARPAnet



satellite net

"A Protocol for Packet Network Intercommunication", V. Cerf, R. Kahn, IEEE Transactions on Communications, May, 1974, pp. 637-648.

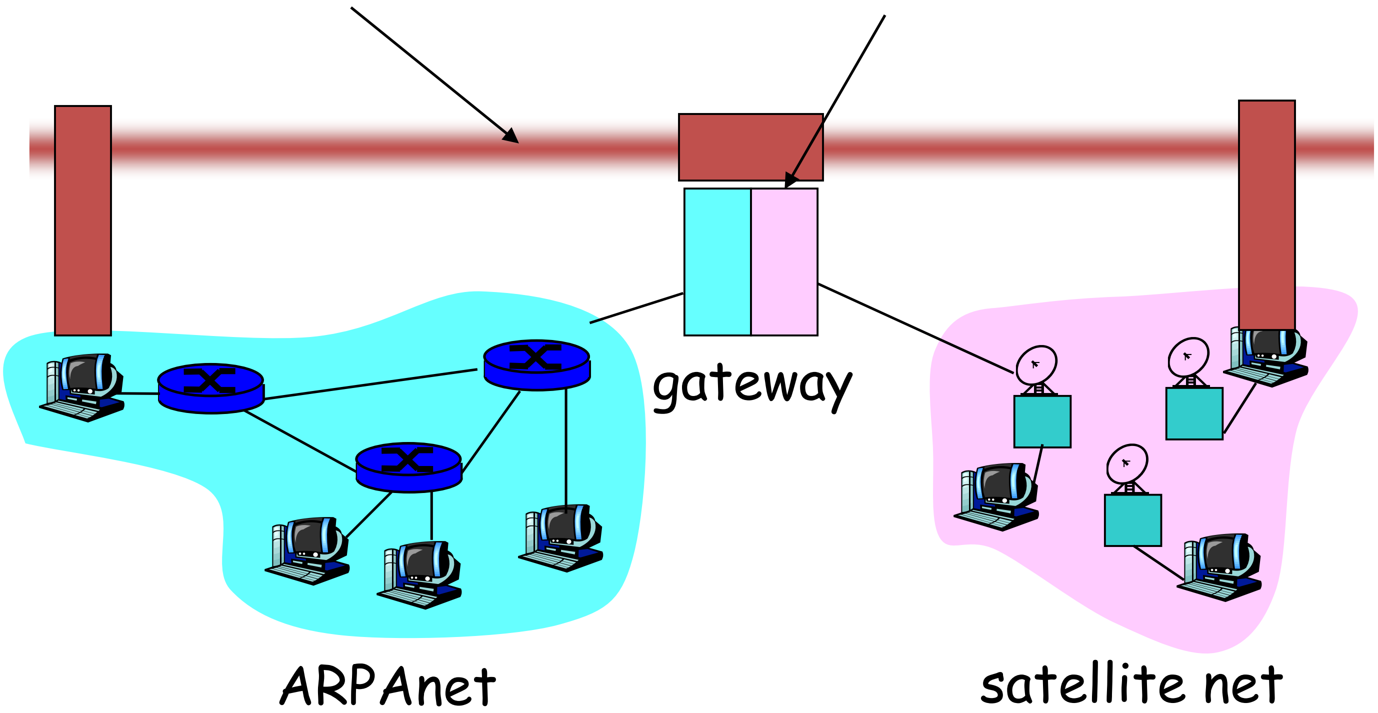
The Internet: virtualizing networks

Internetwork layer (IP):

- addressing: internetwork appears as single, uniform entity, despite underlying local network heterogeneity
- network of networks

Gateway:

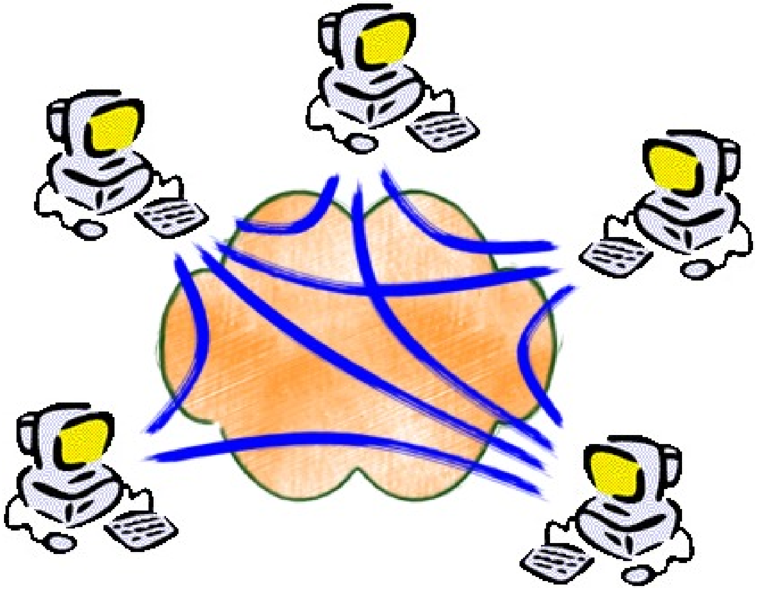
- "embed internetwork packets in local packet format"
- route (at internetwork level) to next gateway



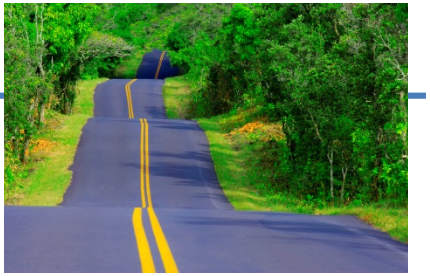
Notice: network overlays

Overlay: a network implemented on top of a network

What else to do with this?



Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

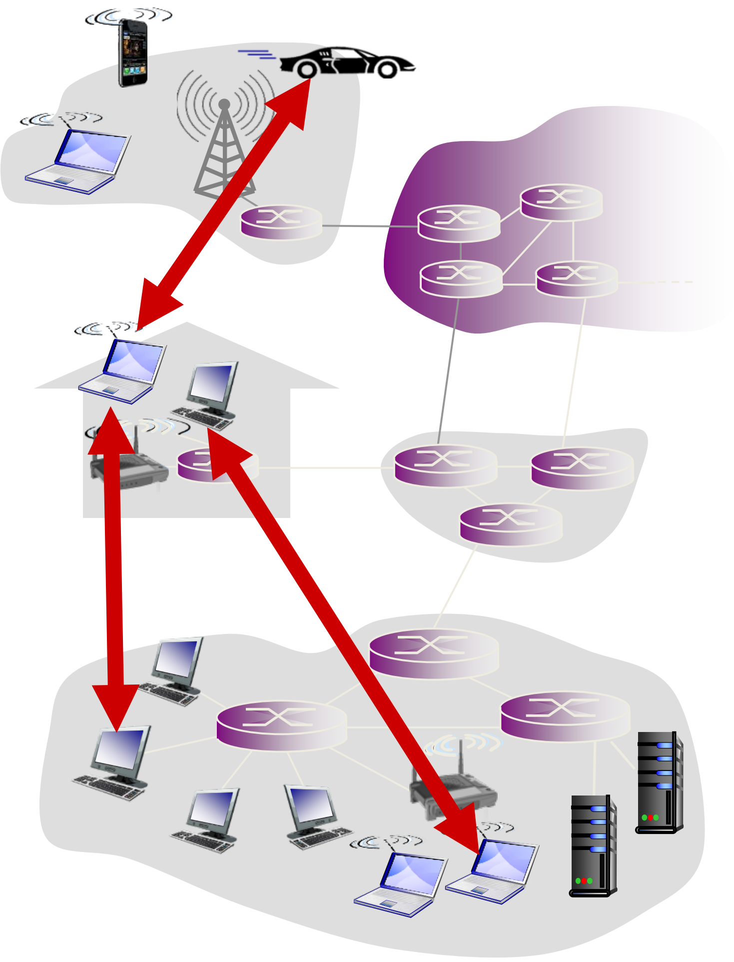
- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

Pure P2P architecture

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

examples:

- **file distribution/sharing** (BitTorrent, …)
- Streaming (media KanKan)
- VoIP (Skype)

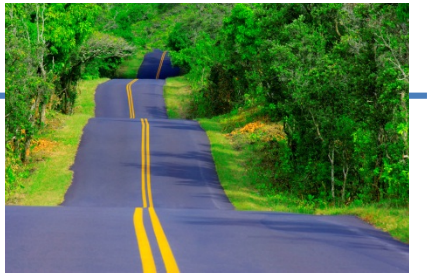


file-sharing peer-to-peer (p2p) applications: preliminaries

Background: Common Primitives in file-sharing p2p apps:

- **Join**: how do I begin participating?
- **Publish**: how do I advertise my file?
- **Search**: how to I find a file/service?
- **Fetch**: how to I retrieve a file/use service?

Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

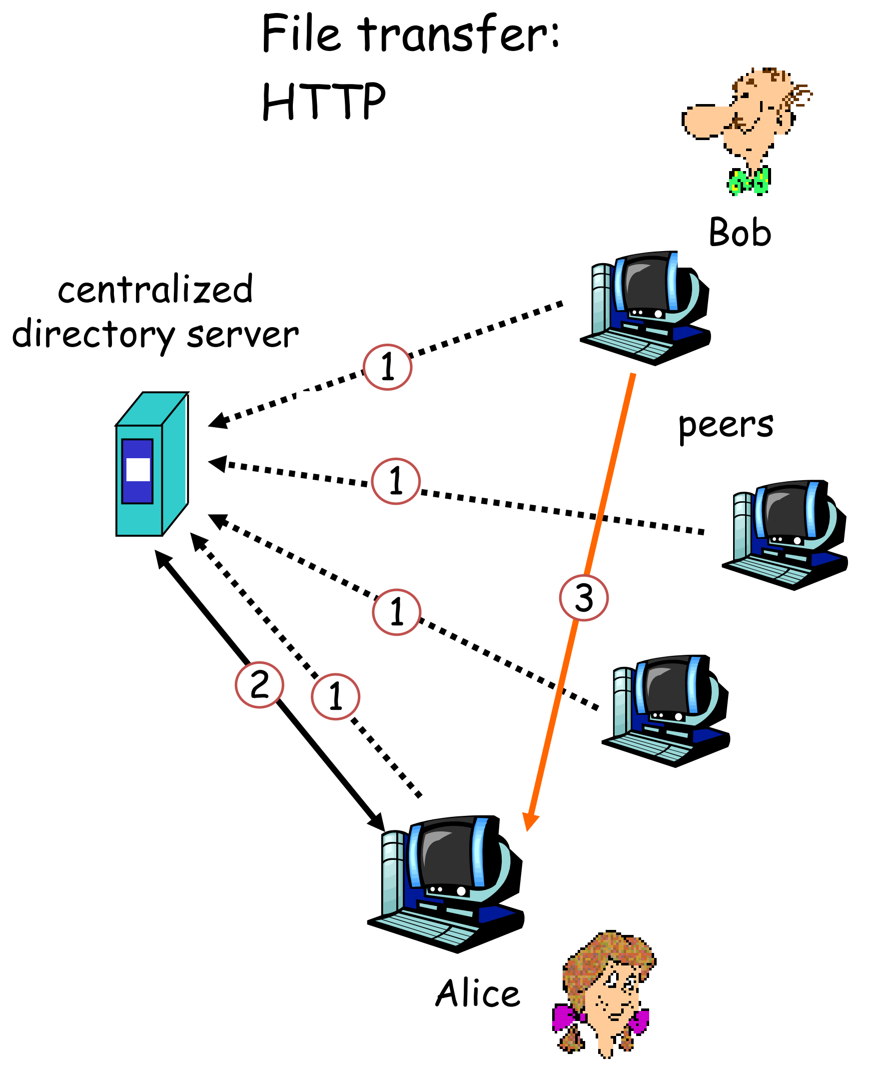
- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

P2P: centralized directory

original “Napster” design (1999, S. Fanning)

1) when peer connects, it informs central server:
   - IP address, content
2) Alice queries directory server for “Boulevard of Broken Dreams”
3) Alice requests file from Bob

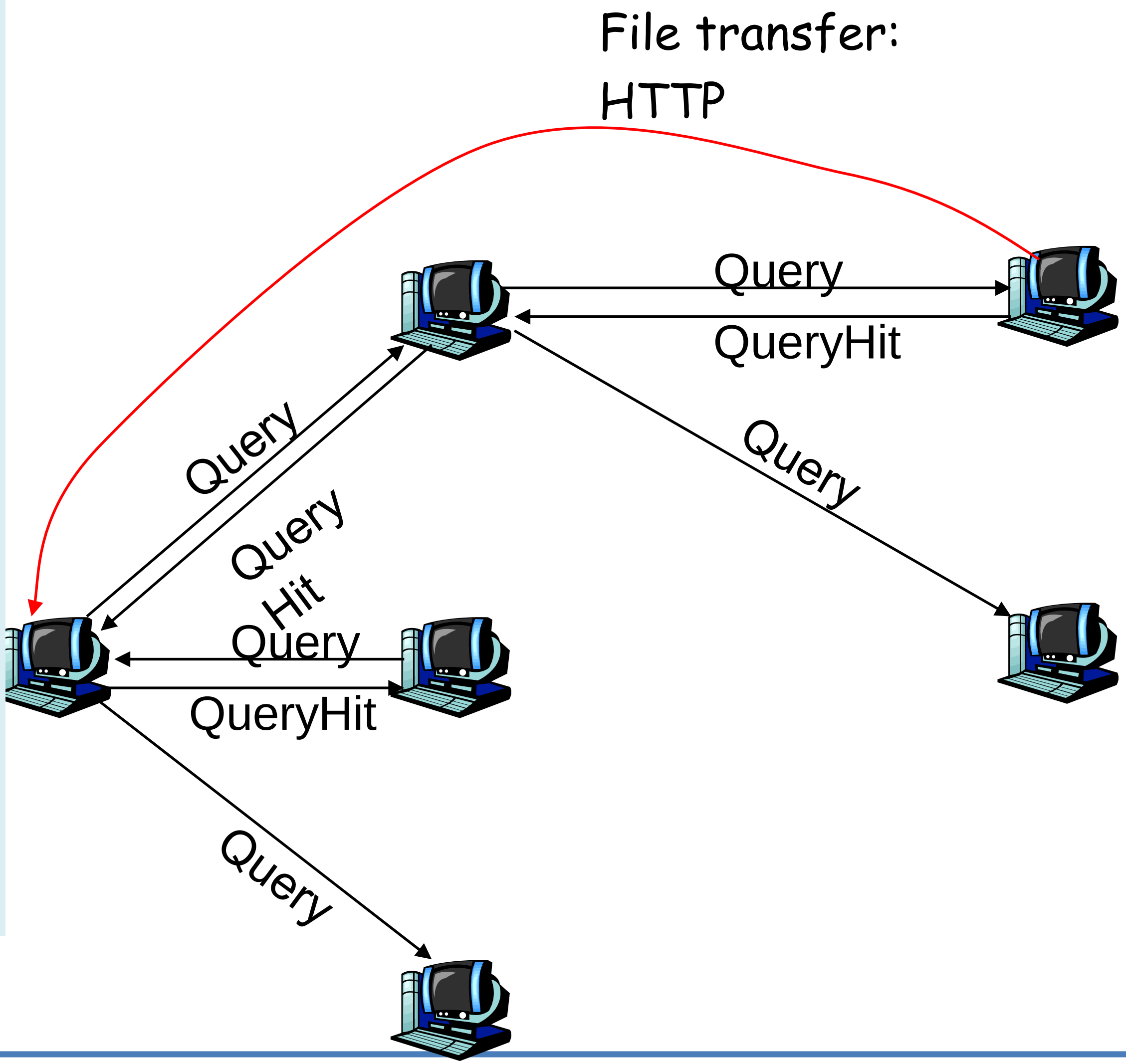
Q: What is p2p in this?



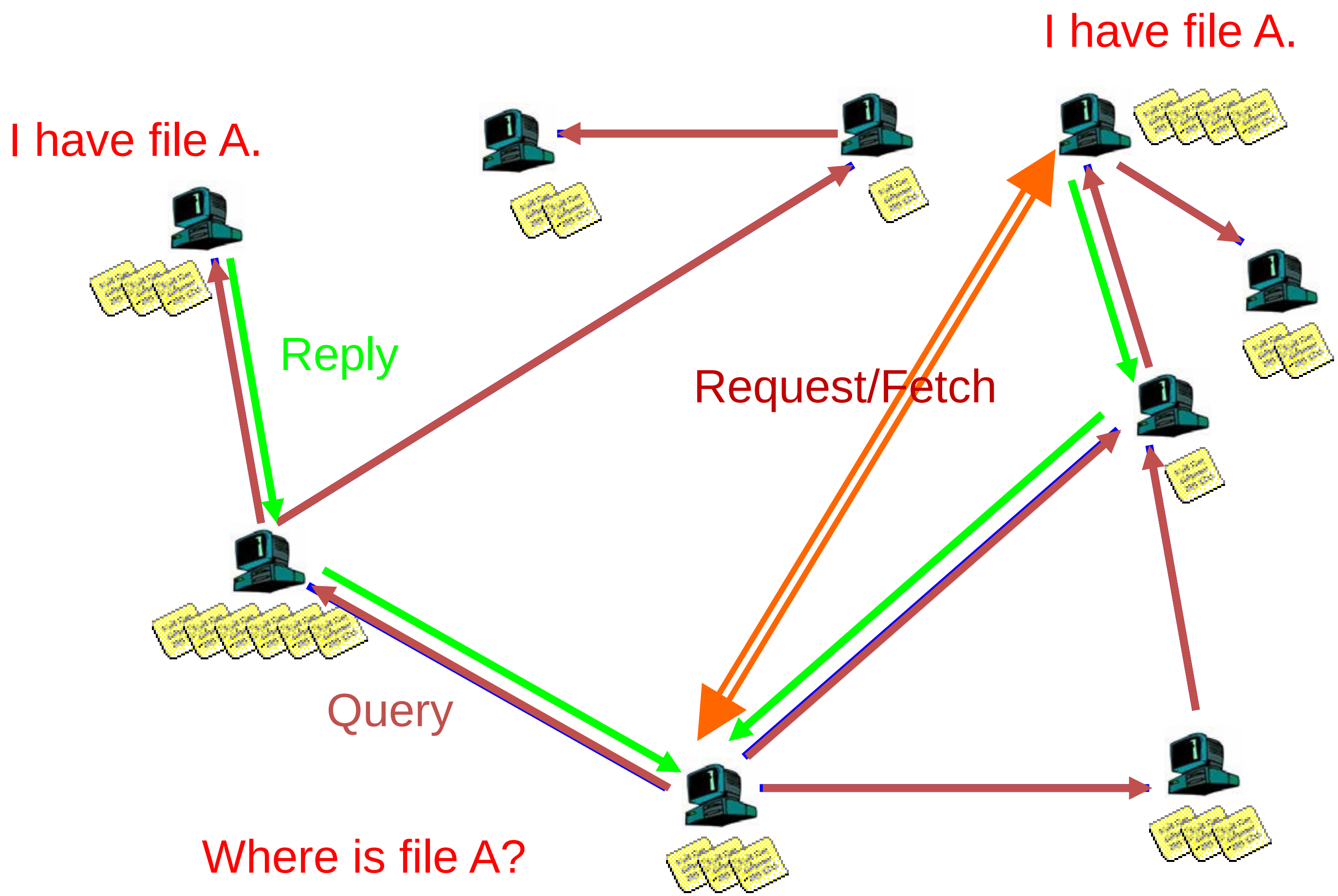
P2p Gnutella (no directory): protocol

Query Flooding:

- **Join**: on startup, client connects to a few other nodes (learn from bootstrap-node); these become its "neighbors" (overlay!! ☺)
- **Publish**: no need
- **Search**: ask *"neighbors"*, who ask their neighbors, and so on... when/if found, reply to sender.
- **Fetch**: get the file directly from peer



Gnutella: Search



Q: Compare with Napster

Discussion +, -?

Napster

- Pros:
  - Simple
  - Search scope is O(1)
- Cons:
  - Server maintains O(peers,items) State
  - Server performance bottleneck
  - Single point of failure

Gnutella:

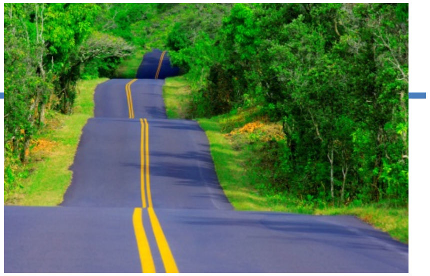
- Pros:
  - Simple
  - Fully de-centralized
  - Search cost distributed
- Cons:
  - Search scope is O(peers, items)
  - Search time is O(???)

Synch questions:

1. how are the ”neighbors” connected?
2. what is the overlay here useful for?

- Edge is not a single physical link E.g. edge between peer X and Y if they *know each-other’s IP addresses* or *there’s a TCP connection*
- Used for supporting the **search** operation (**aka routing in p2p networks**)

Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

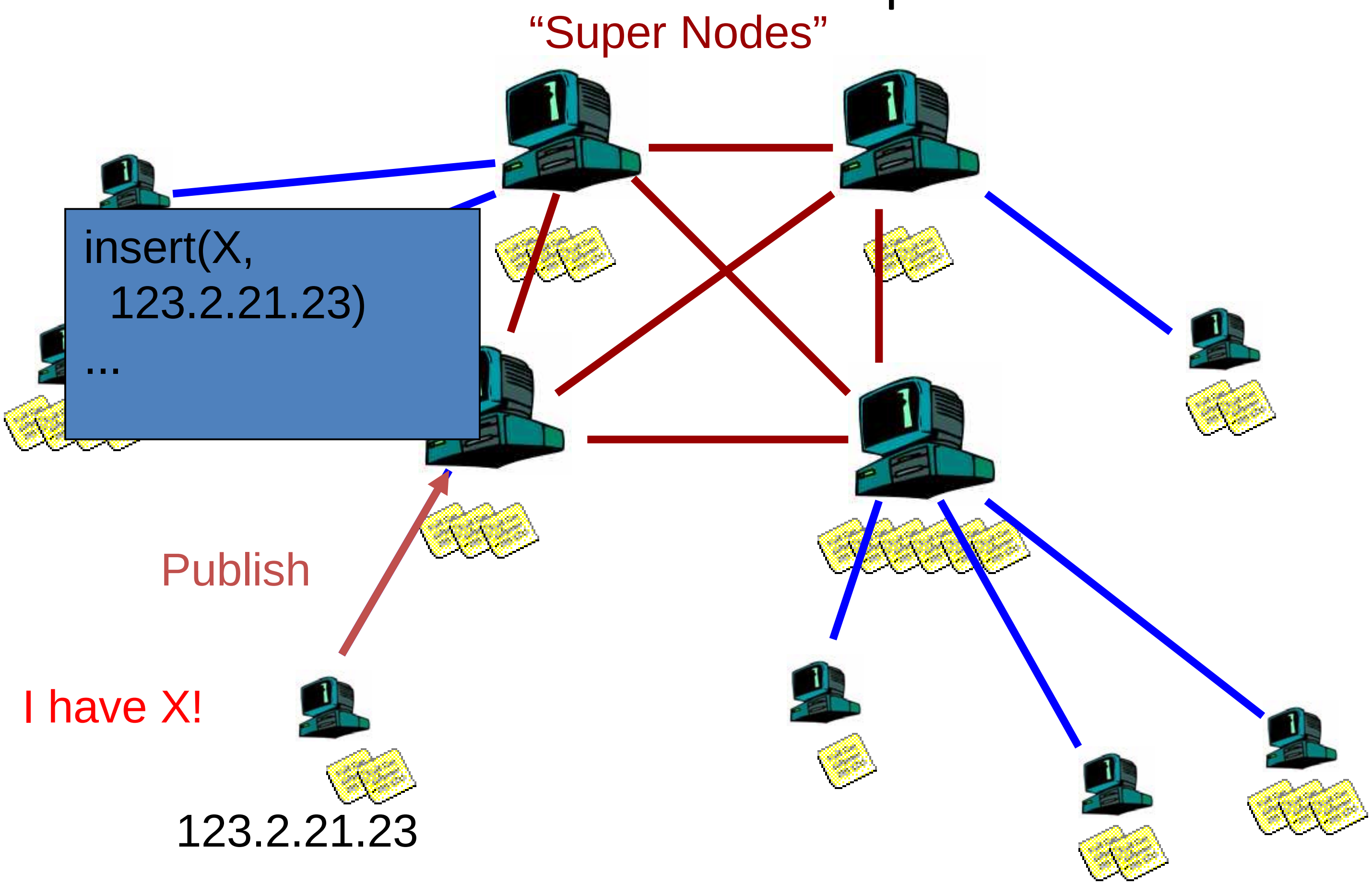
Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

KaZaA : distributed directory

"Smart" Query Flooding:

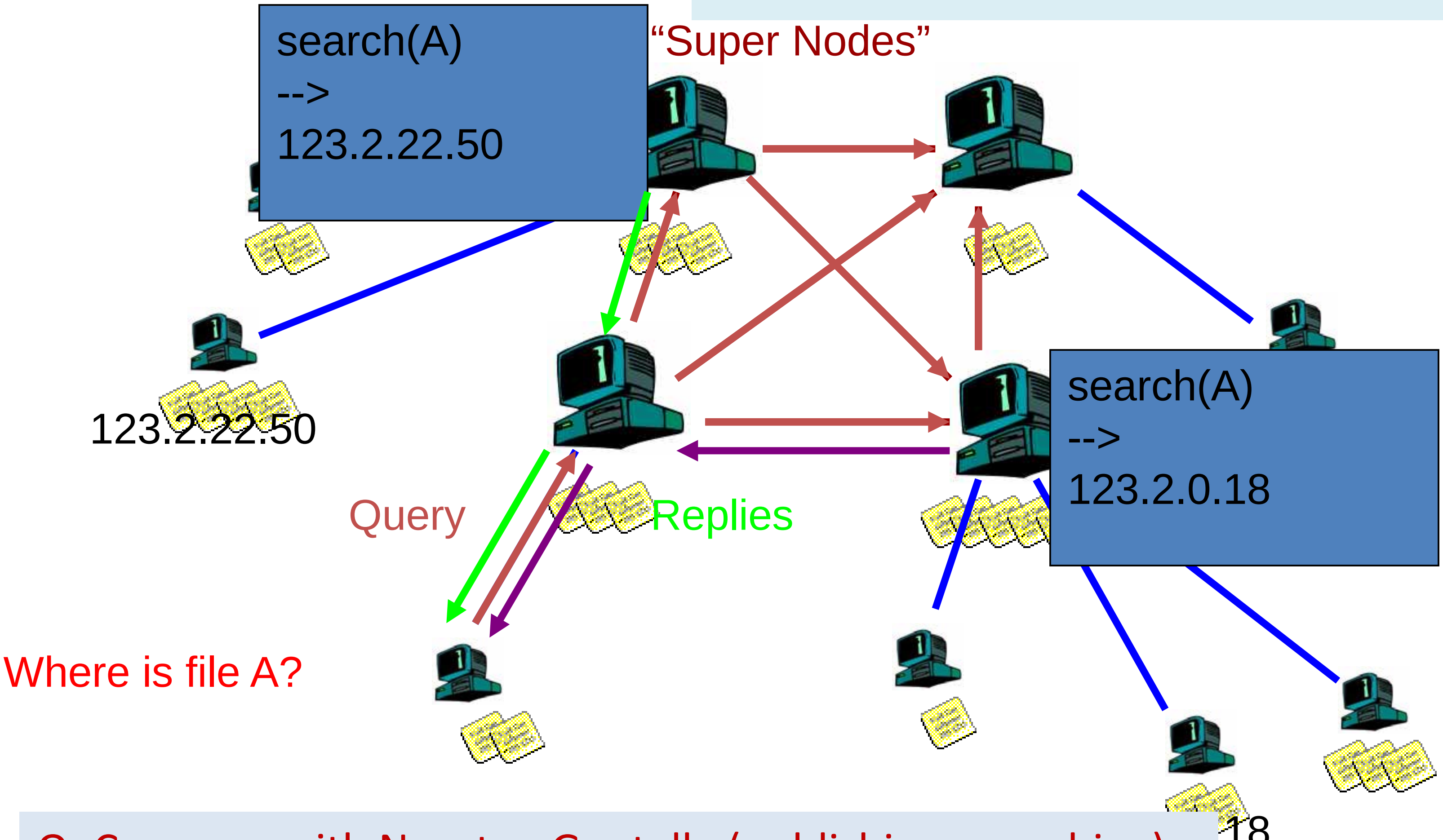
- **Join**: on startup, client contacts a "supernode" ... may at some point become one itself
- **Publish**: send list of files to supernode



KaZaA: Search

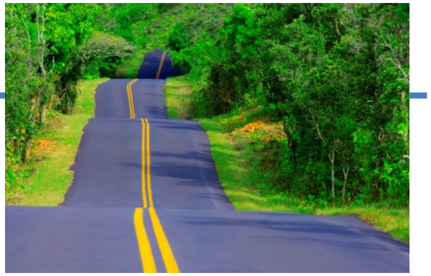
"Smart" Query Flooding:

- **Search**: send query to supernode, supernodes flood query amongst themselves.
- **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers



Q: Compare with Napster, Gnutella (publishing, searching)

Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

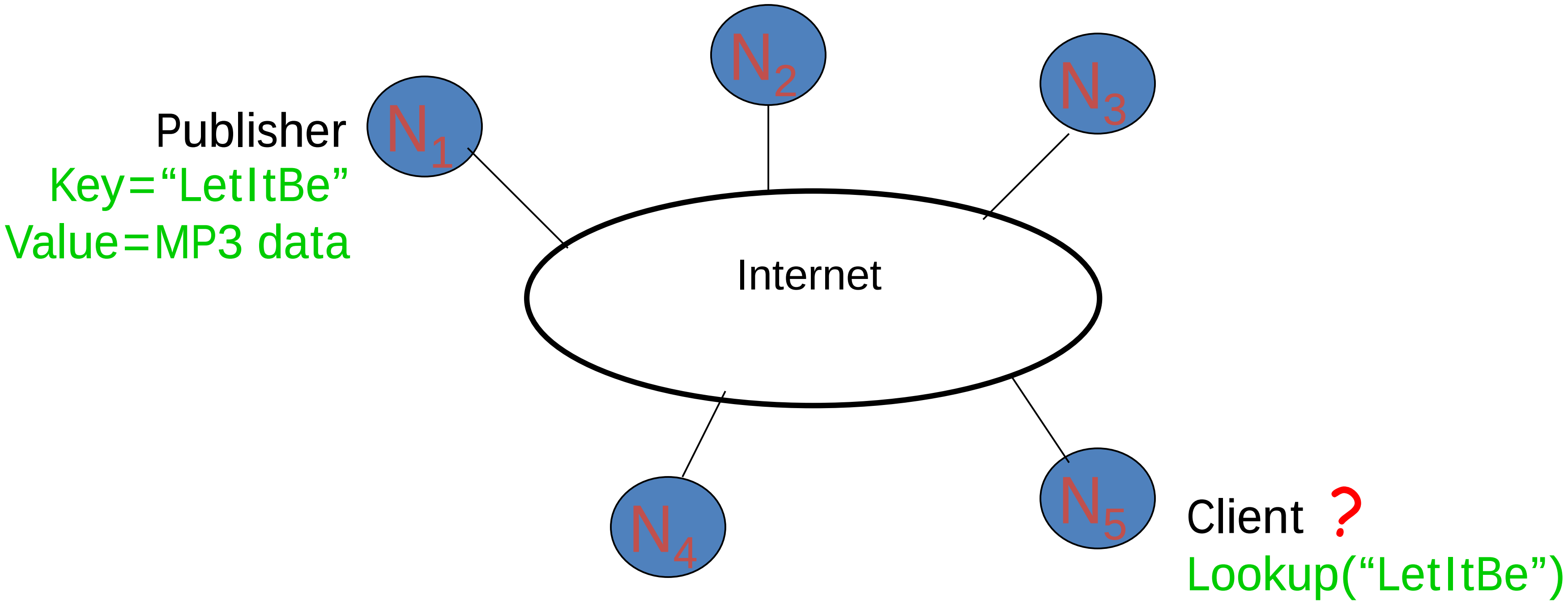
Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

Problem from this perspective

How to find data in a distributed file sharing system?

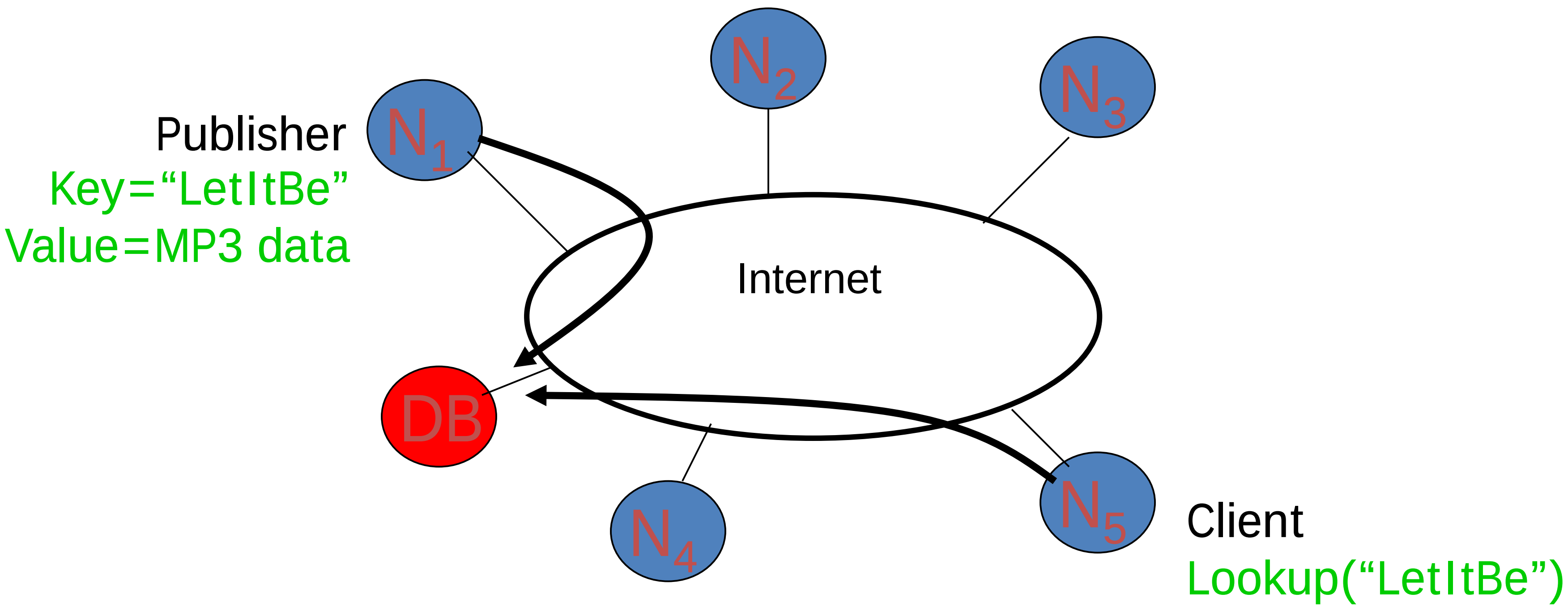
(aka "routing" to the data, i.e. content-oriented routing)



How to do Lookup?

Centralized Solution

Central server (Napster)



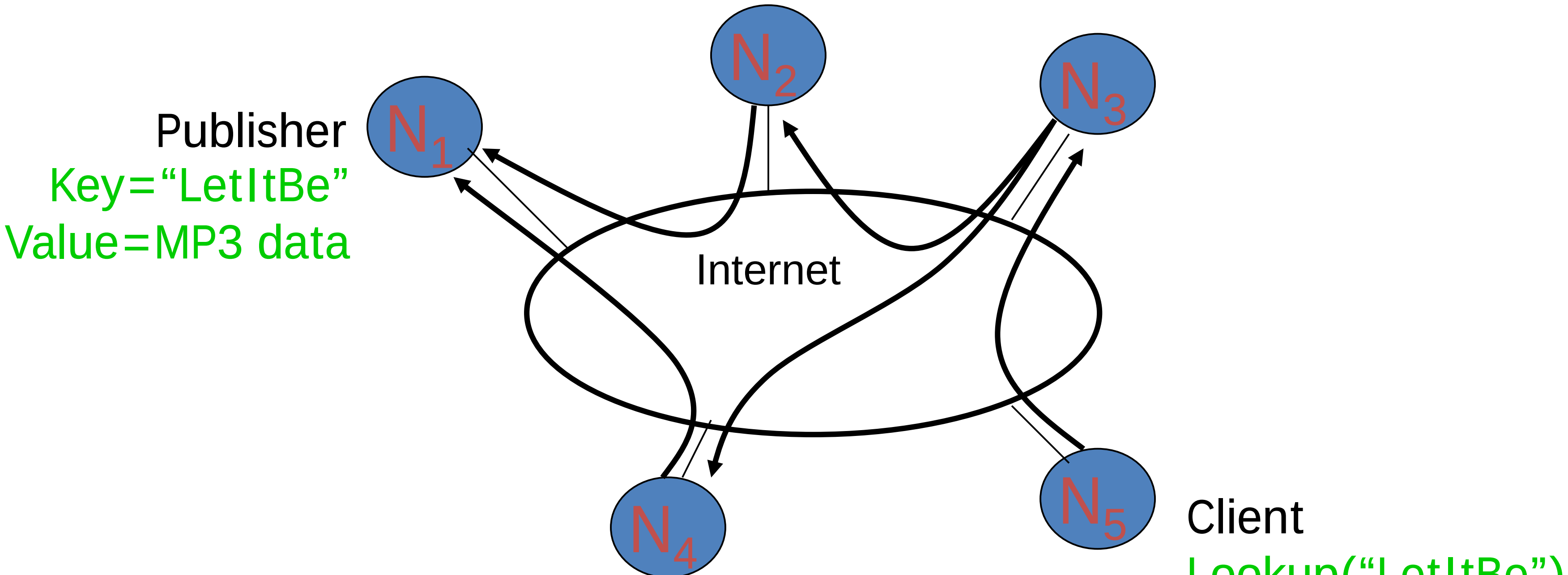
O(peers,items) state at server, O(1) at client

O(1) search communication overhead

Single point of failure

Fully decentralized (distributed) solution

Flooding (Gnutella, etc.)



O(1) state per node

Worst case O(peers, items) complexity per lookup

Better Distributed Solution?

(with some more structure? In-between the two? Yes)

balance the update/lookup complexity;

Abstraction: a lookup data structure (distributed "hash-table" DHT) :

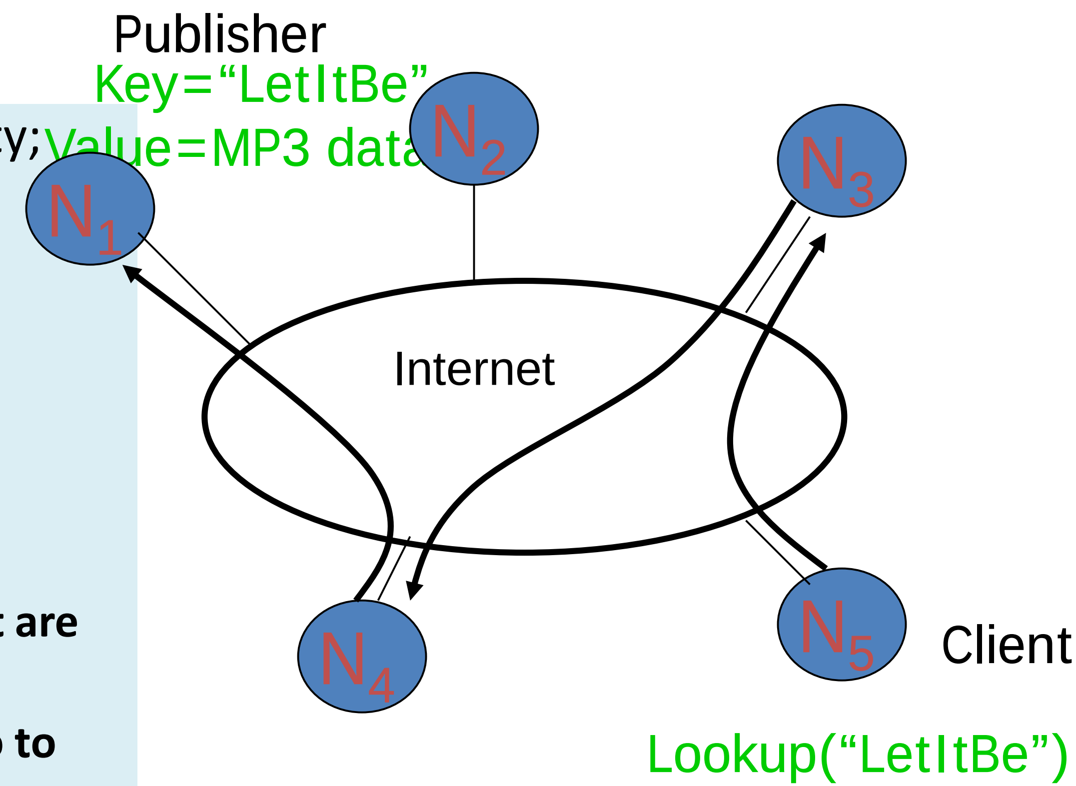
insert(key, item);

item = get(key);

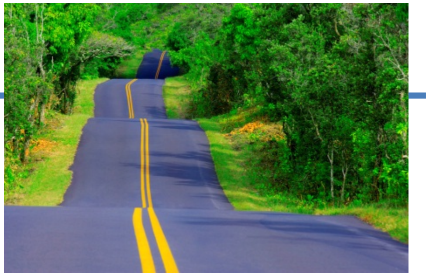
Each node **responsible for**

- **Maintaining part of the database in a structured manner (ie the entries that are hash-mapped to it)**
- **Knowing its overlay neighbours & who to start asking for what**

Eg. overlay can be a ring (eg Chord, also having shortcuts for binary serach) or cube, tree, butterfly network, etc



Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

BitTorrent: Next generation fetching

- Key Motivation:
  – Popularity exhibits temporal locality (Flash Crowds)
  – Can bring file “provider” to “its knees”

- **Idea: Focused on Efficient *Fetching*, not *Searching*:**
  – Files are “chopped” in chunks, fetching is done from many sources
  – Overlay: nodes “hold hands” with those who share (send chunks) at similar rates

- Method used by publishers to distribute software, other large files

- http://vimeo.com/15228767

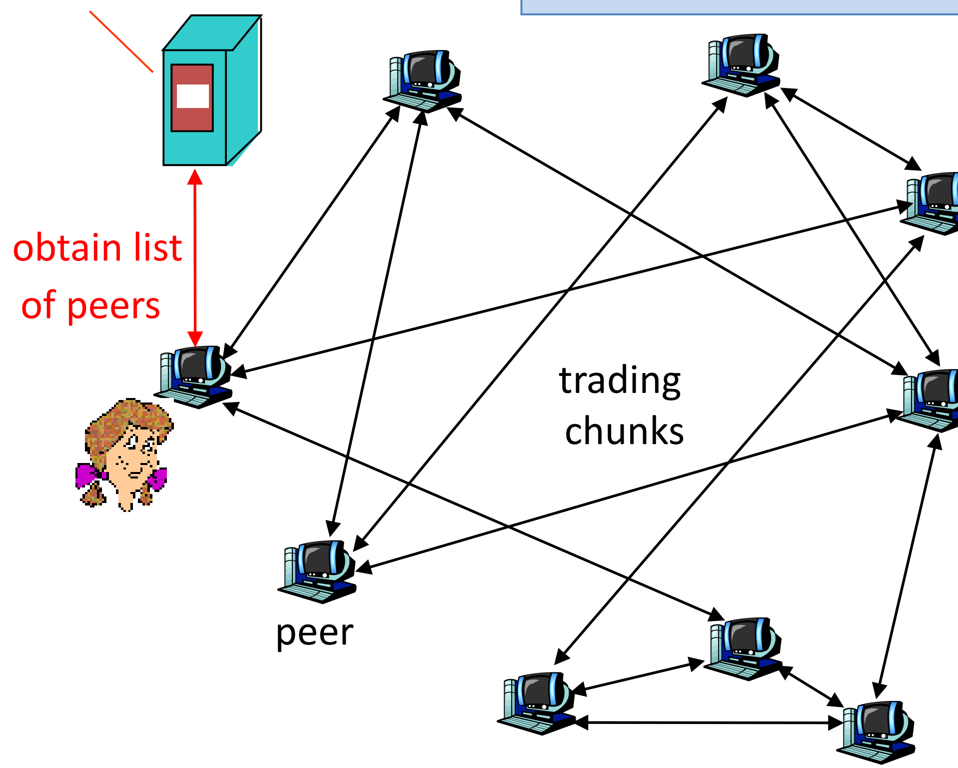
BitTorrent: Overview

Swarming:

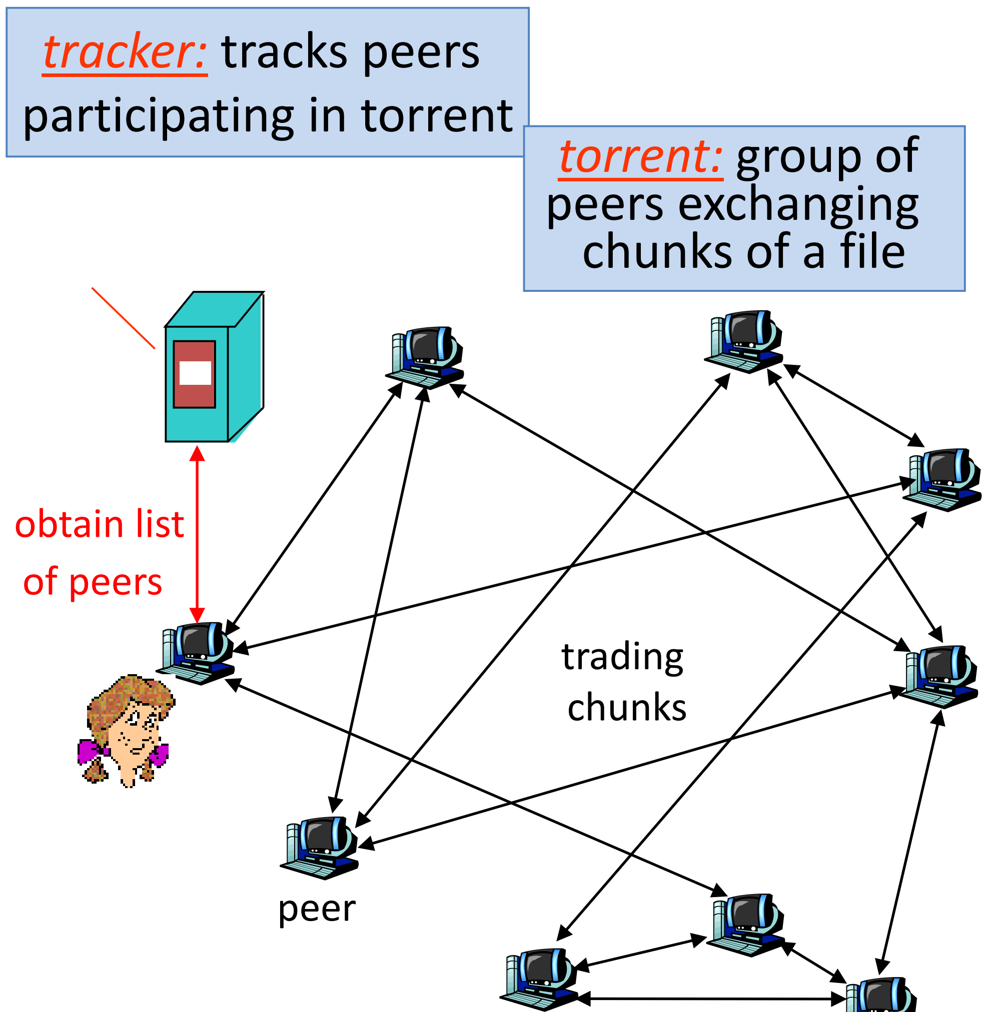
- **Join**: contact some server, aka "tracker" get a list of peers.
- **Publish**: can run a tracker server.
- **Search**: Out-of-band. E.g., use search-engine, some DHT, etc to find a tracker for the file you want. Get list of peers to contact for assembling the file in chunks
- **Fetch**: Download chunks of the file from several of your peers. Upload chunks you have to them.

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

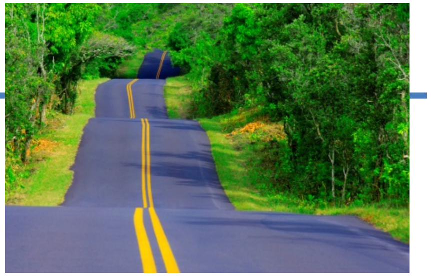


File distribution: BitTorrent



- Peer joining torrent:
  - has no chunks, but will accumulate over time
  - gets list of peers from tracker, connects to subset of peers ("neighbors") who share at *similar rates* (tit-for-tat)
- while downloading, peer uploads chunks to other peers.
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

Roadmap



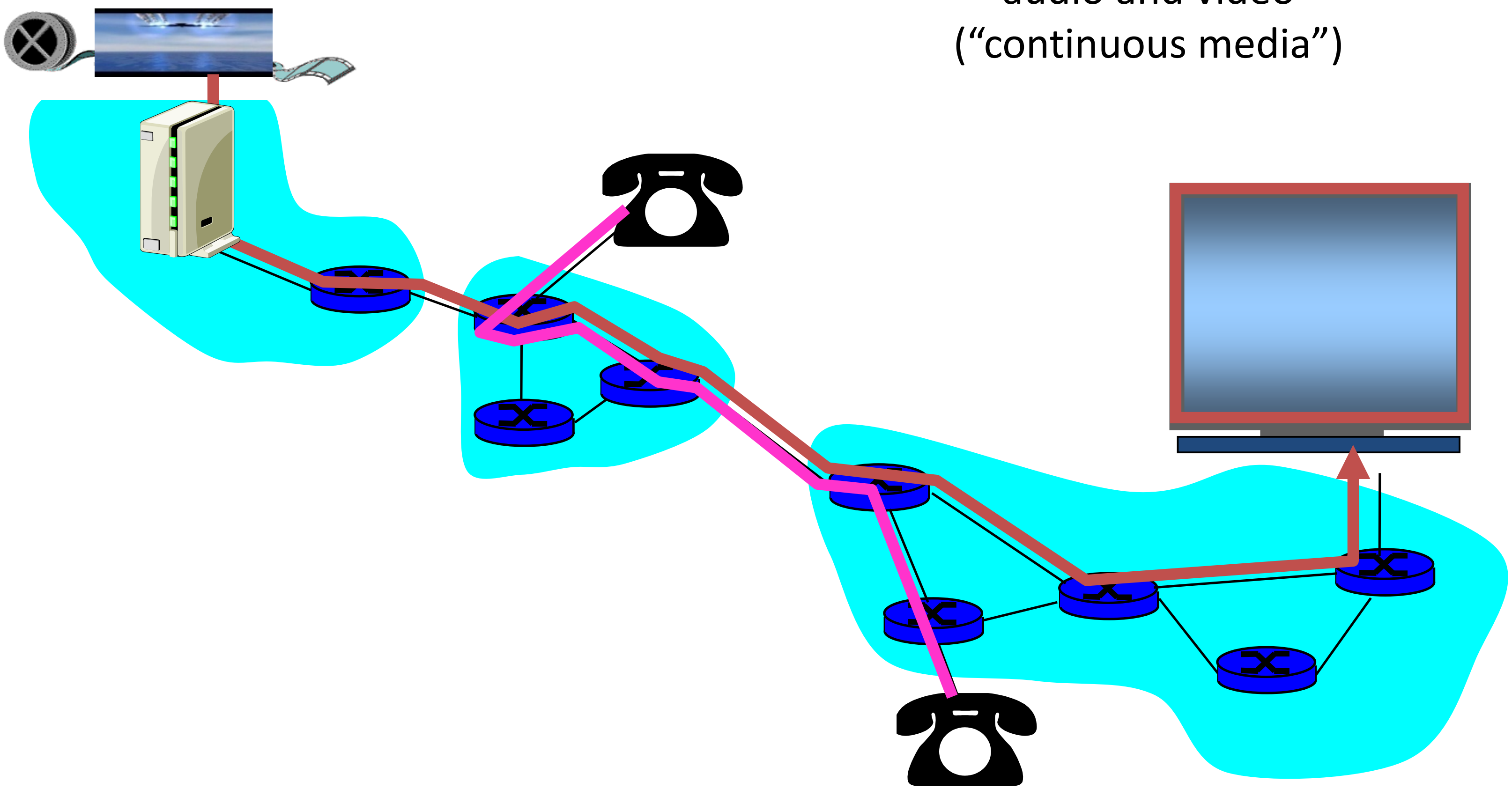
P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

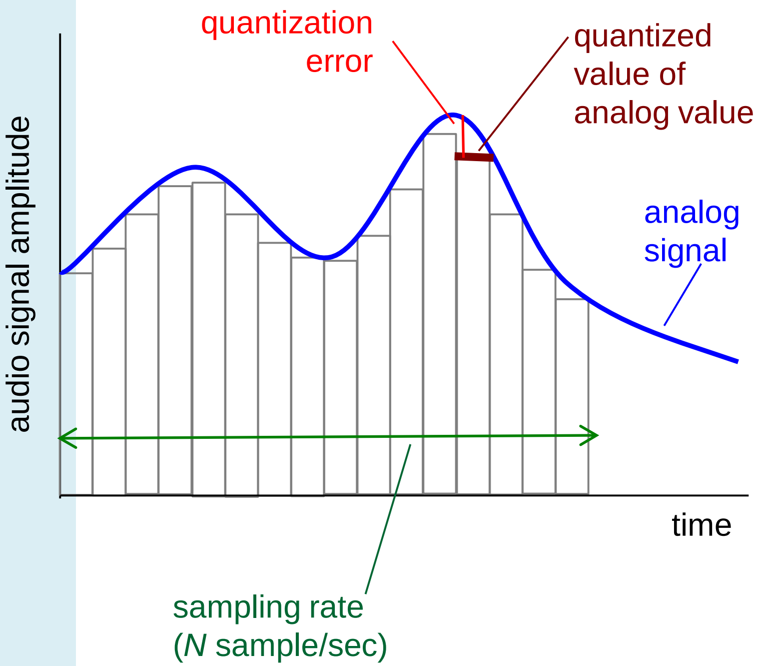
- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

multimedia applications: network audio and video ("continuous media")



Multimedia: audio

- analog audio signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- example: 8,000 samples/sec, 256 quantized values: 64,000 bps
- receiver converts bits back to analog signal:



example rates

- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 kbps and up

Multimedia: video

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)

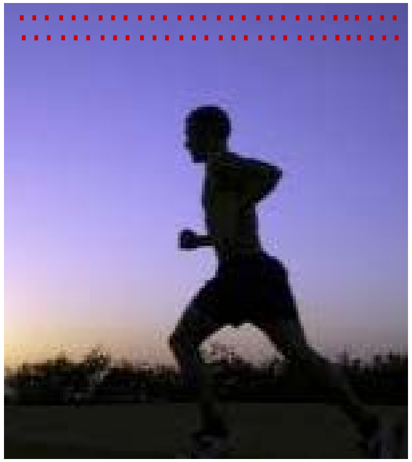
- video: sequence of images (arrays of pixels) displayed at constant rate
  - e.g. 24 images/sec

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)



frame *i*

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

frame *i+1*

Multimedia networking: application types

- *streaming stored* audio, video

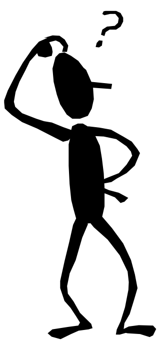
Fundamental characteristics:

- typically **delay sensitive**
  - end-to-end delay
  - delay *jitter*
- **loss tolerant**: infrequent losses cause minor glitches
- In contrast to traditional data-traffic apps, which are loss *intolerant* but delay *tolerant*.

Jitter is the variability of packet delays within the same packet stream

Recall Internet's services

no guarantees on delay….



But you said multimedia apps require
Delay/jitter (ie bandwidth) guarantees to be
effective!

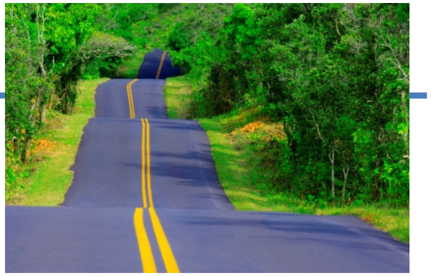
Internet multimedia applications
use application-level techniques to mitigate
(as best possible) effects of delay, loss;
(Also complementing with "traffic engineering"
& Software-Defined Networking in NW core: coming soon)

Solution Approaches in Internet

To mitigate impact of "best-effort" protocols:

- Several applications use UDP to avoid TCP's ack-based progress (and slow start)…
- Buffer content at client and control playback to remedy jitter
- Different error control methods (no ack)
- Exhaust all uses of caching, proxys, etc
- Adapt compression level to available bandwidth
- add more bandwidth
- Traffic engineering, SDN

Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

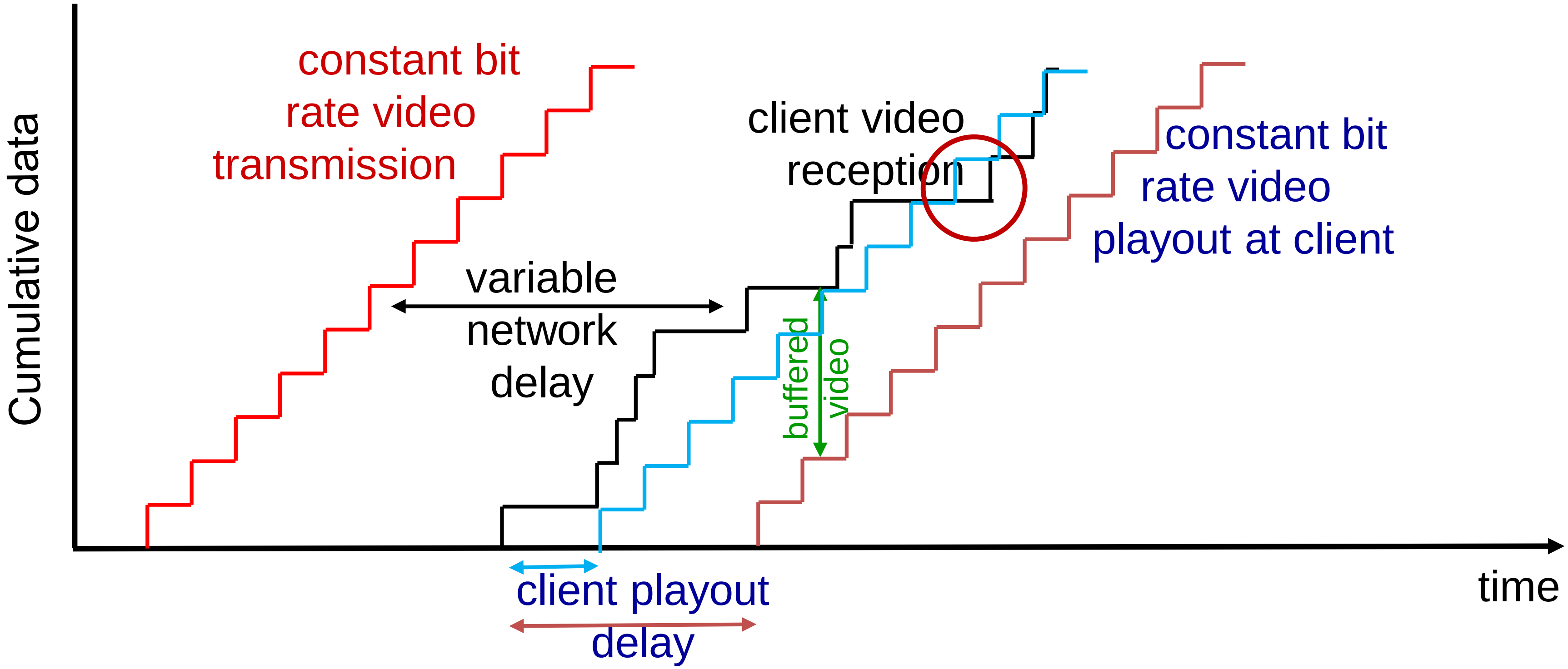
- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

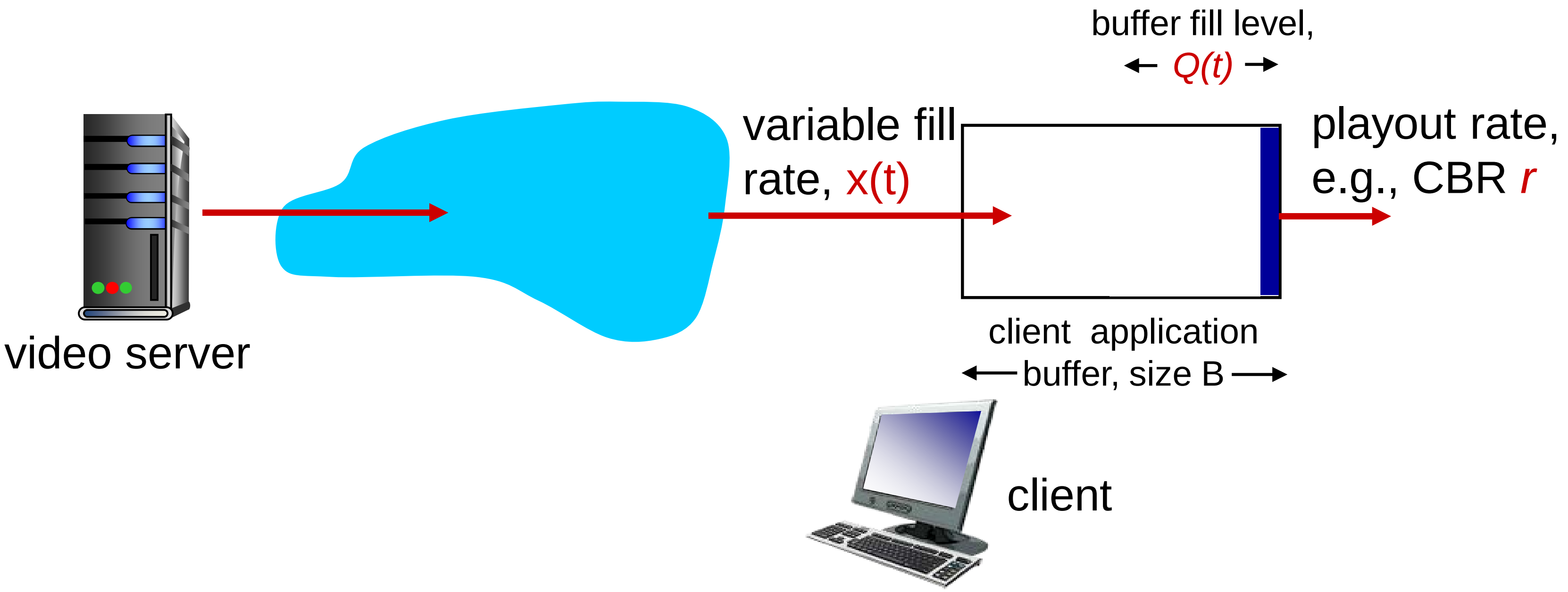
Streaming: recovery from jitter

playout delay small => higher loss rate



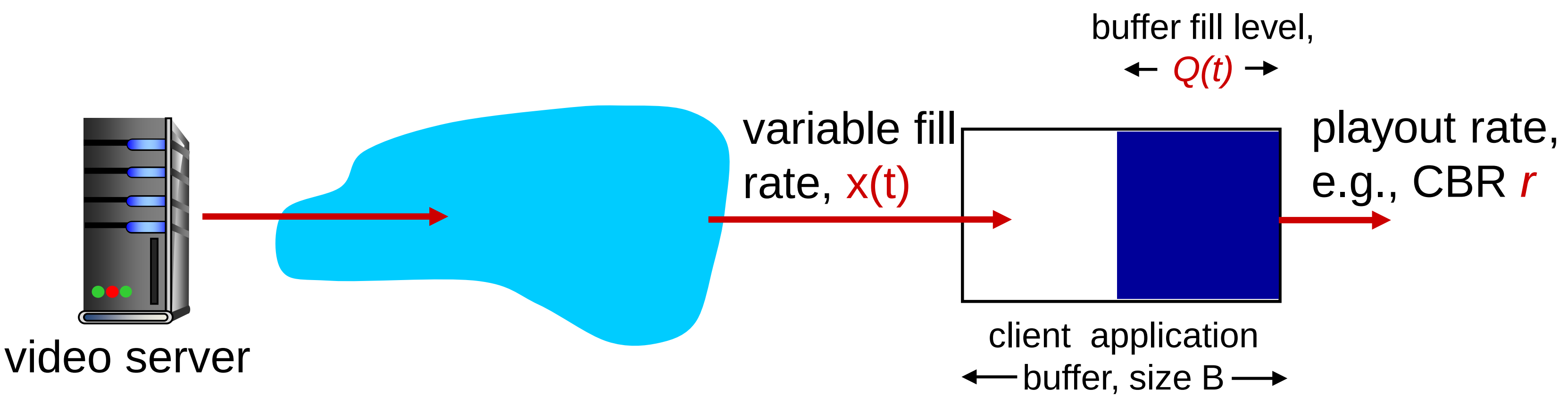
❖ *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout



1. Initial fill of buffer until …
2. … playout begins at t_p,
3. buffer fill level varies over time as fill rate *x(t)* varies and playout rate *r* is constant

Client-side buffering, playout

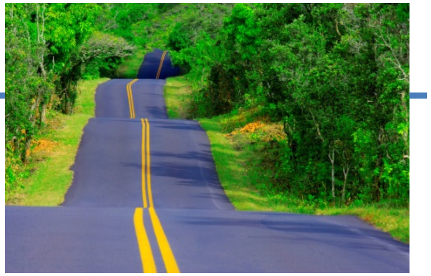


playout buffering: average fill rate ($\overline{x}$), playout rate (r):

- $\overline{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\overline{x} > r$: need to have enough buffer-space to absorb variability in x(t)

initial playout delay tradeoff: buffer empty (aka starvation) less likely with larger delay, but larger delay until user begins watching

Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

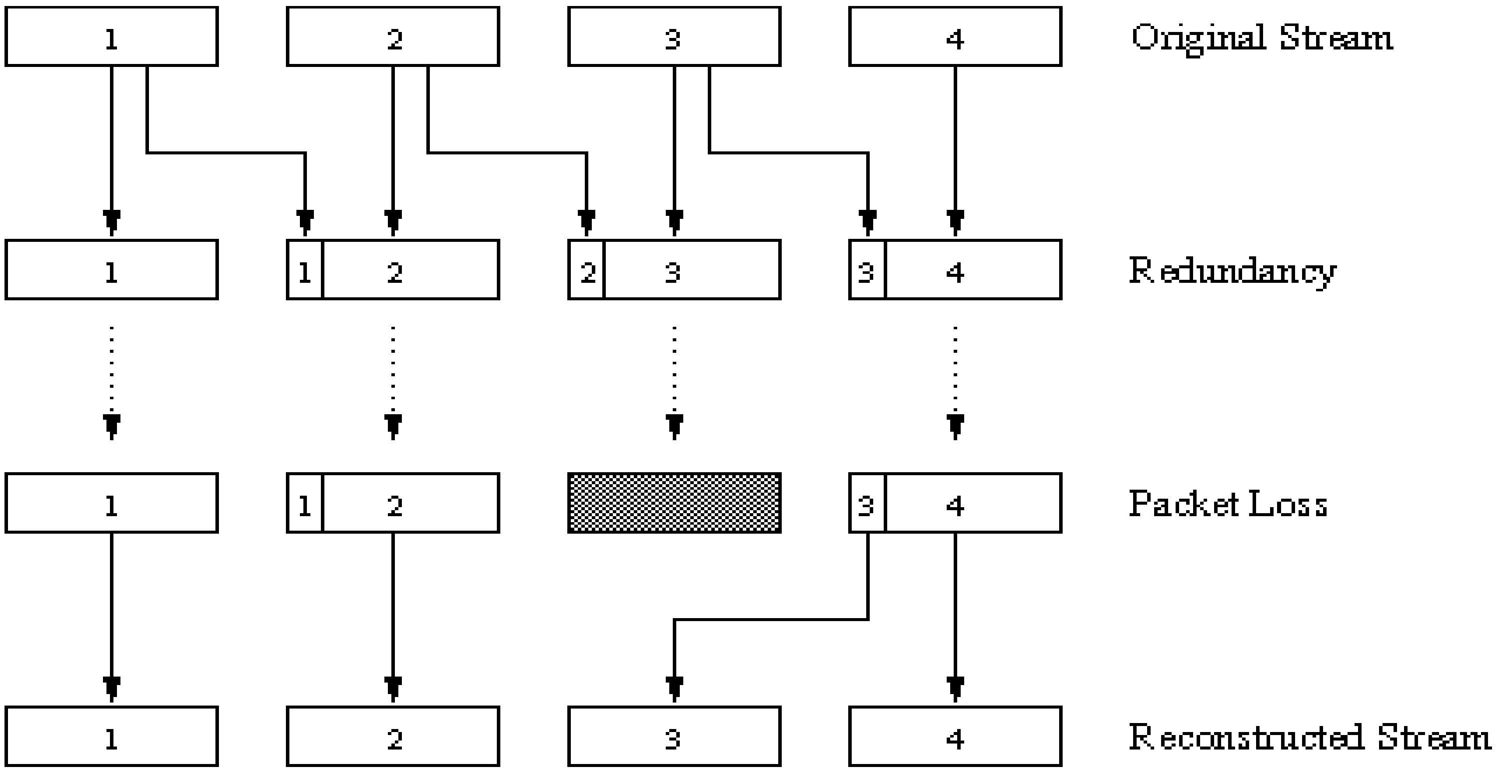
- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and *loss*
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

Recovery From Packet Loss
Forward Error Correction

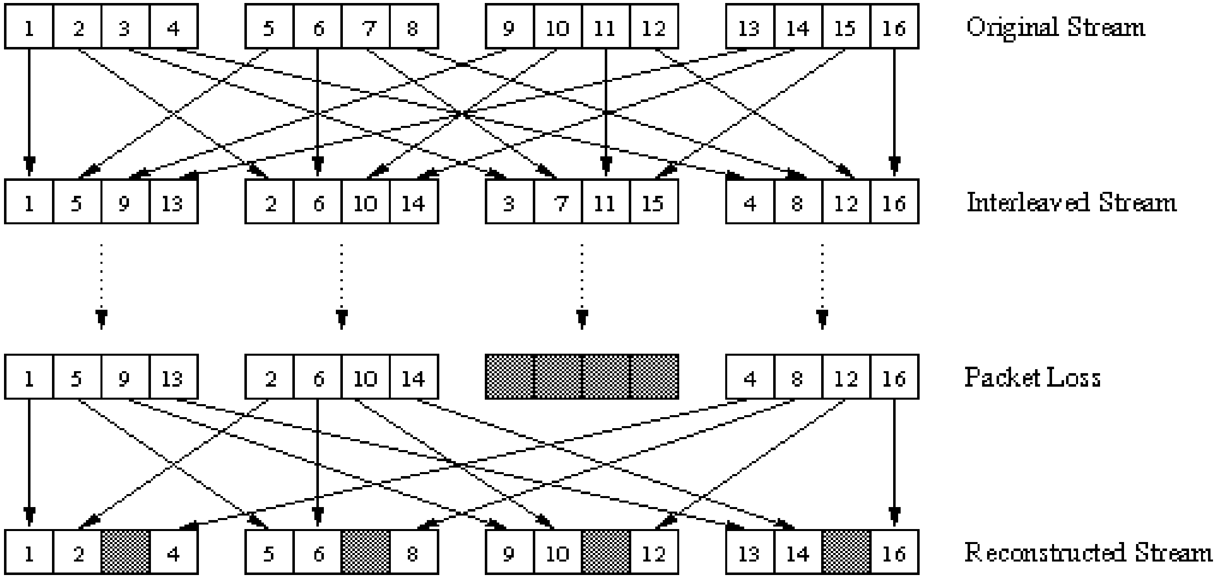
Eg. 1. through piggybacking Lower Quality Stream



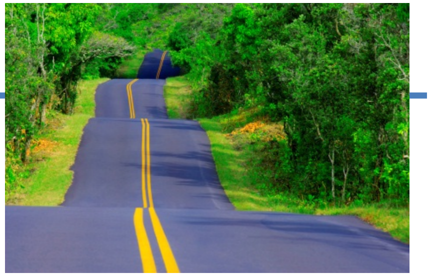
Why care about FEC?

Recovery From Packet Loss/FEC (cont)

2. Interleaving: no redundancy, but can cause delay in playout beyond Real Time requirements
   – Upon loss, have a set of partially filled chunks
   – playout time must adapt to receipt of group



Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

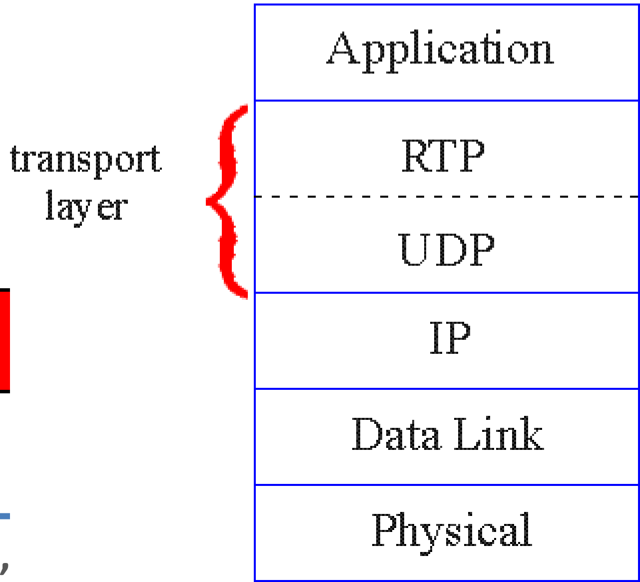
- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

Real-Time Protocol (RTP) RFC 3550

- RTP specifies packet structure for carrying audio, video data
  - payload type (encoding)
  - sequence numbering
  - time stamping
- RTP does not provide any mechanism to ensure timely data delivery or other guarantees
- RTP encapsulation only seen at end systems

RTP packets encapsulated in UDP segments

- interoperability: e.g. if two Internet phone applications run RTP, then they may be able to work together

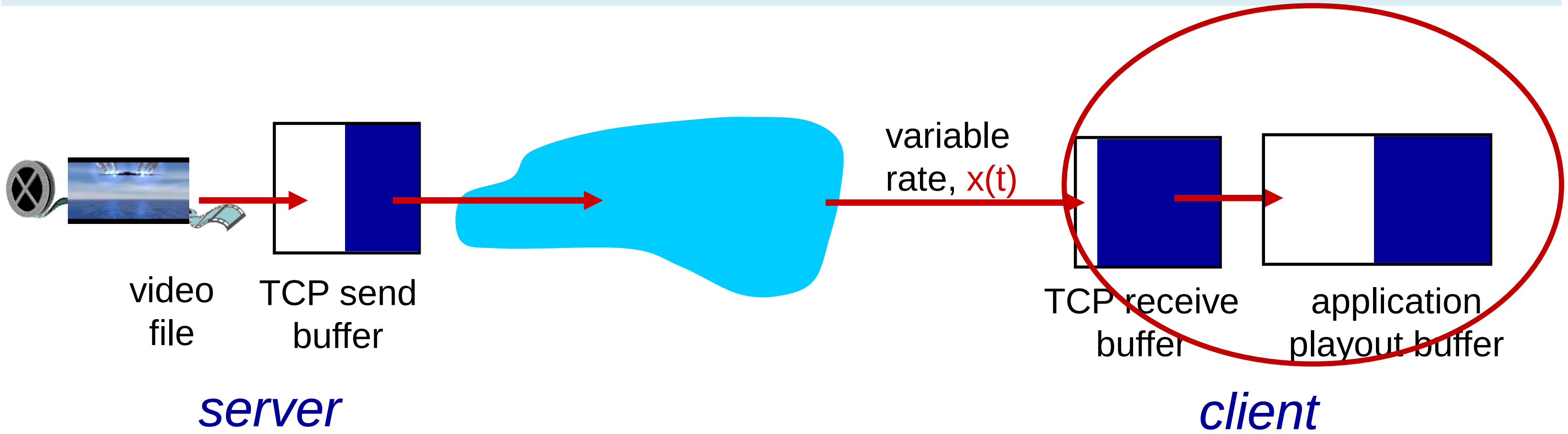


Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate
  - send rate can be oblivious to congestion levels (is this good? selfish?)
- short playout delay to remove network jitter
- BUT: UDP may *not* go through firewalls

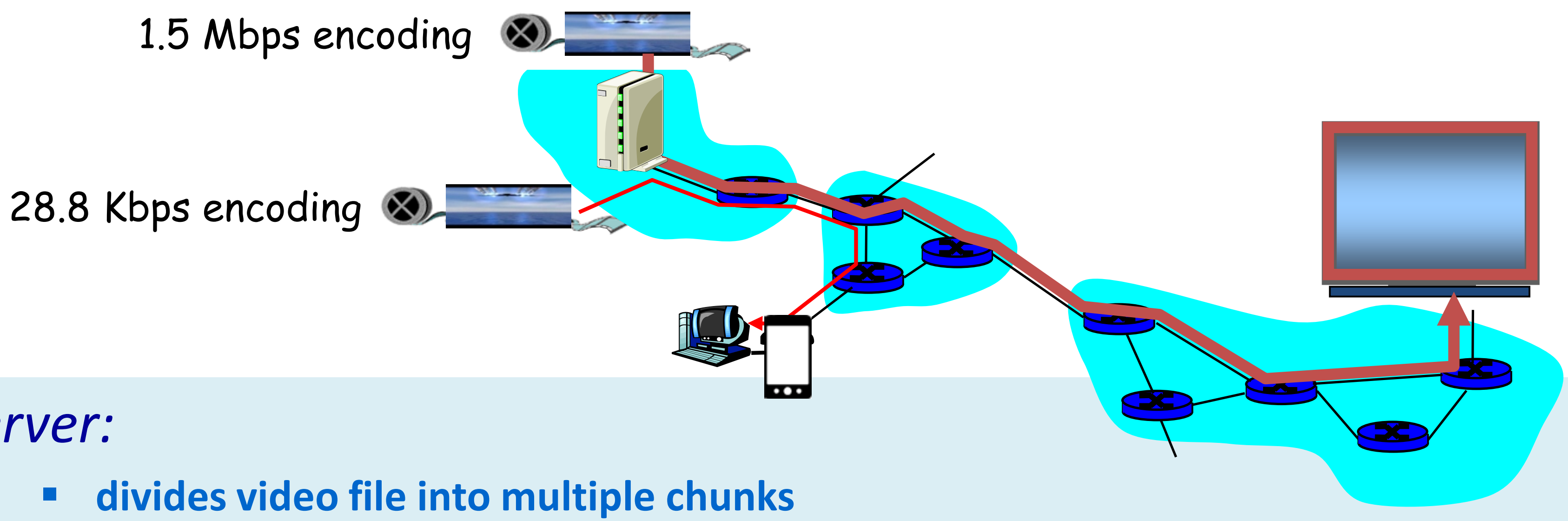
Streaming multimedia: HTTP (ie through TCP)

- ❖ multimedia file retrieved via HTTP GET
- ❖ send at maximum possible rate under TCP



- ❖ fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)
- ❖ larger/adaptive playout delay: to smooth TCP saw-tooth delivery rate
- ❖ HTTP/TCP passes easier through firewalls

Streaming multimedia: DASH: Dynamic, Adaptive Streaming over HTTP



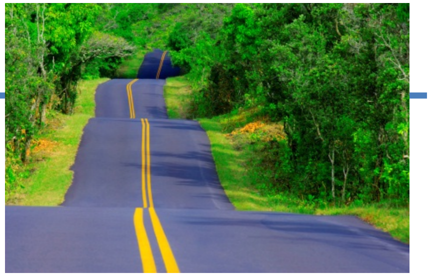
server:

- **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, at appropriate coding rate
  - can choose different coding rates at different points in time (depending on available bandwidth at time)

Roadmap



P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

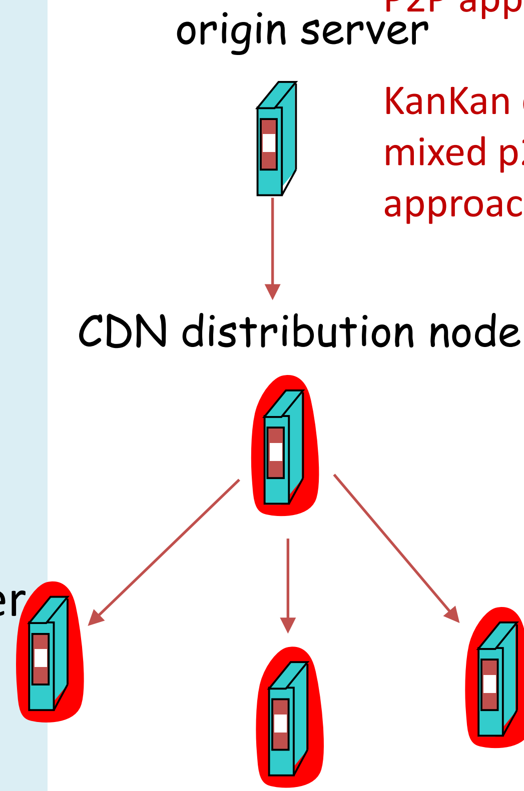
- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**

Content distribution networks(CDNs)

- Scalability big problem to stream large files from single origin server in real time to millions end-
- servers
  in pull or
  rments
  eep into
  's) of larger
  in) access
  ions of

Resembles DNS and **overlay networks** in P2P applications!

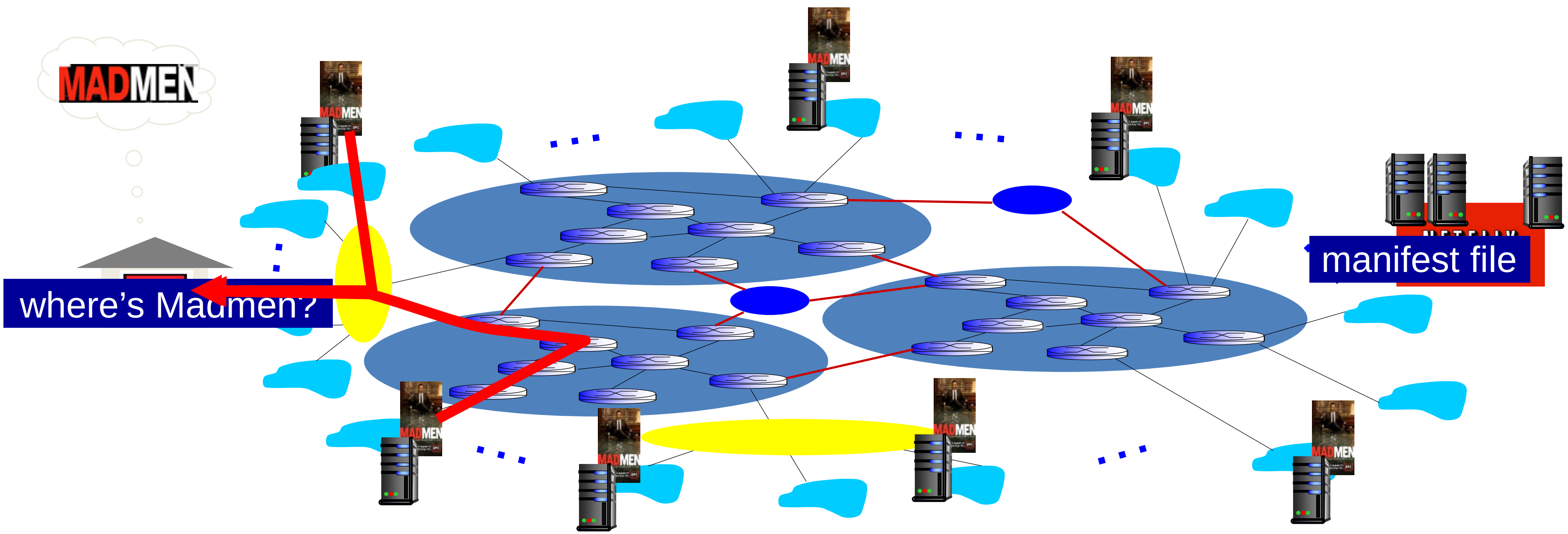
KanKan does use a mixed p2p &CDN approach



Video link: http://vimeo.com/26469929

Content Distribution Networks (CDNs)

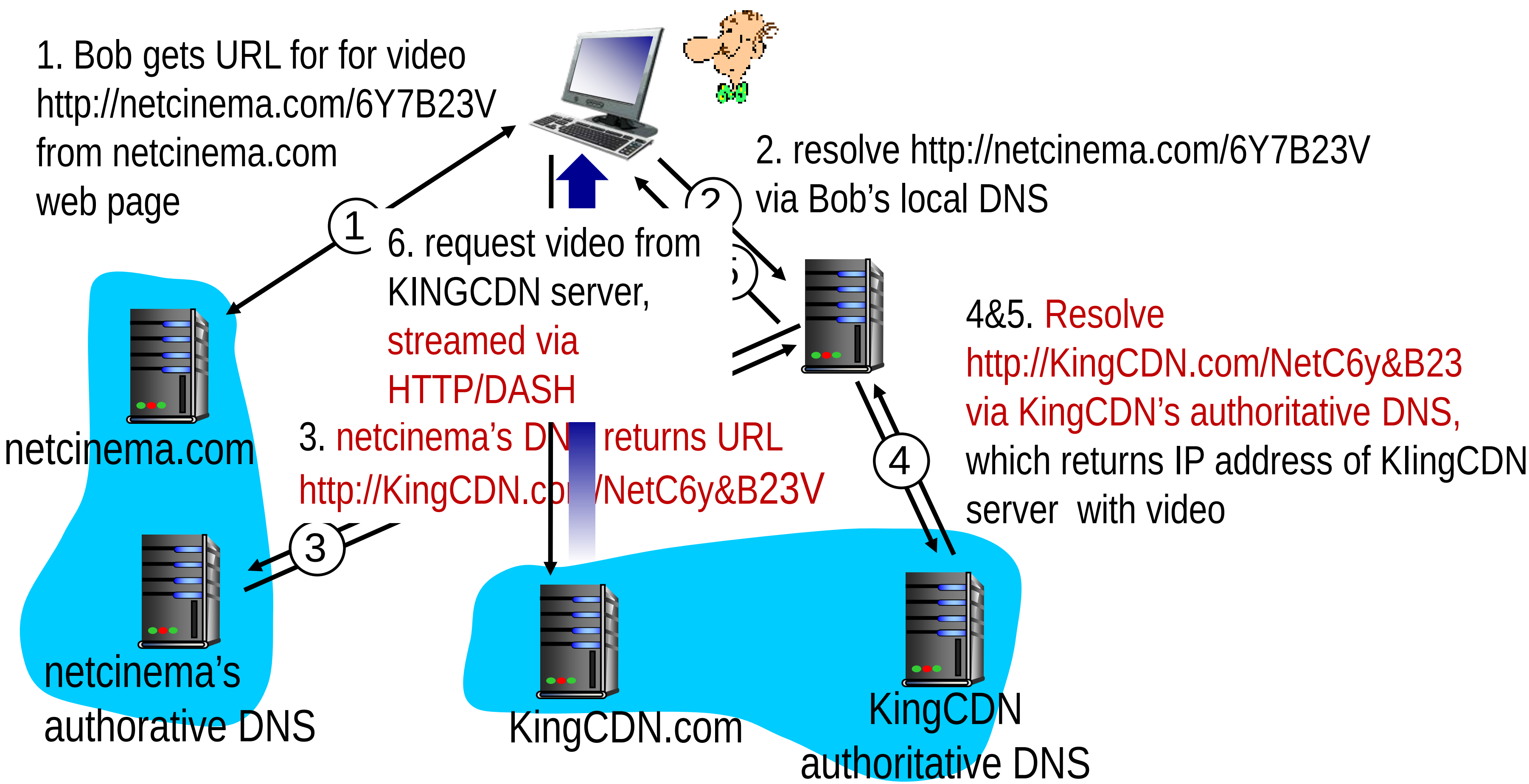
- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy *if network path congested*



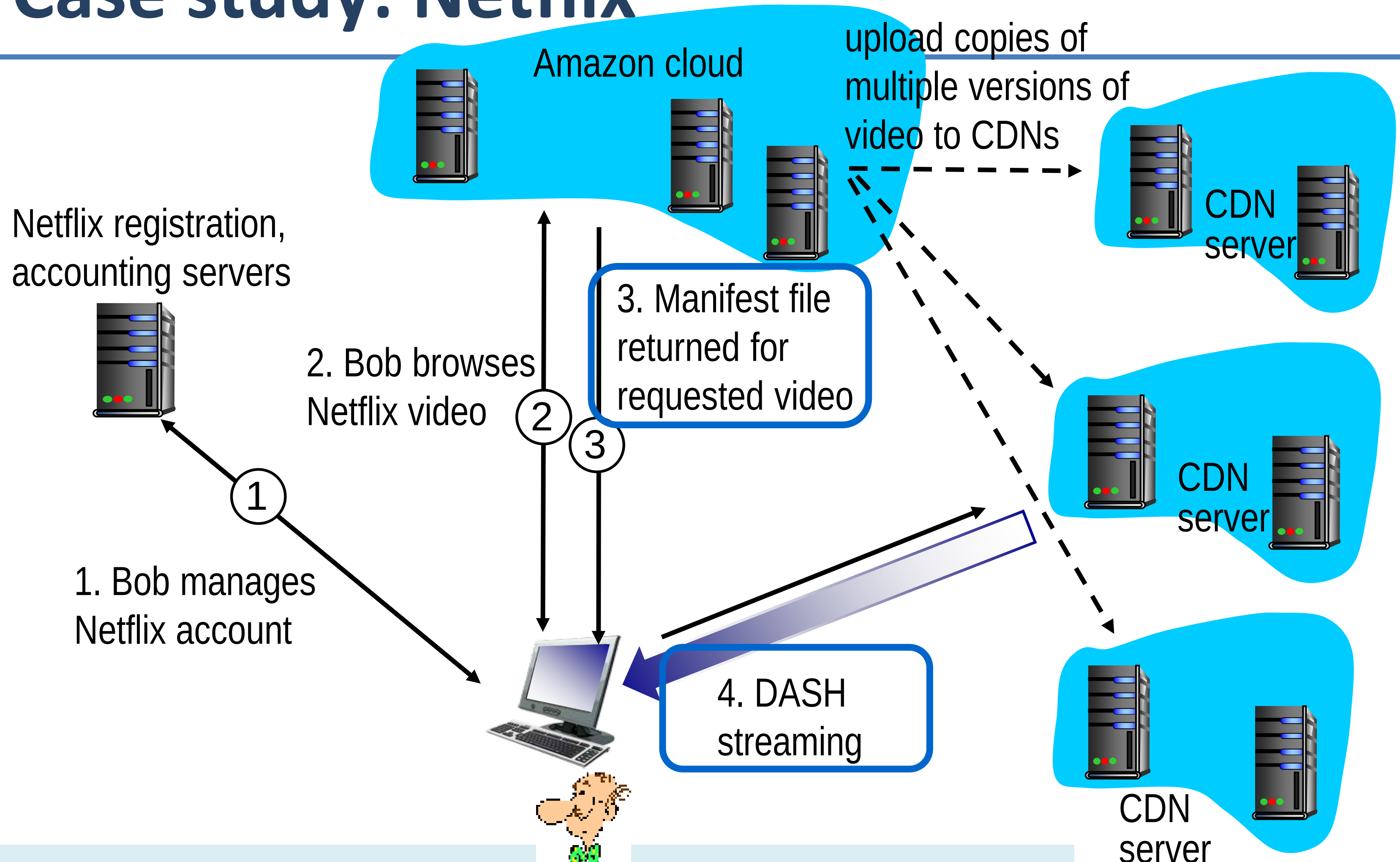
CDN: “simple” content access scenario

Bob (client) requests video http://netcinema.com/6Y7B23V

- video stored in CDN at http://KingCDN.com/NetC6y&B23V



Case study: Netflix



- Netflix uploads copy of video to 3rd party cloud, which
  - creates multiple versions of movie (different encodings) in cloud
  - uploads versions from cloud to CDN servers;
- user downloads the suitable encoding from them

youtube.com/watch?v=LkLLpYdDINA

youtube.com/watch?v=tbqcsHg-Q_o

Up to this point summary Internet Multimedia: bag of tricks

- use UDP to avoid TCP congestion control (delays) for time-sensitive traffic; or multiple TCP connections (DASH)
  - Buffering and client-side adaptive playout delay: to compensate for delay
  - error recovery (on top of UDP)
    - FEC, interleaving, error concealment
- CDN: bring content closer to clients
- server side matches stream bandwidth to available client-to-server path bandwidth
  - chose among pre-encoded stream rates
  - dynamic server encoding rate

Q: would all this be simpler with Network (layer) support?

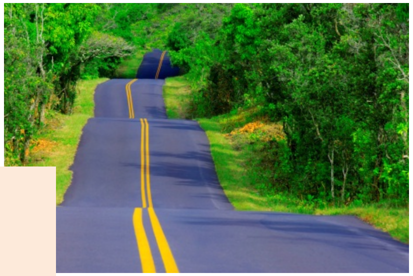
Roadmap

P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)

- Collaborate/form-*overlays* to *find* content:
  - No overlay: centralized DB (Napster)
  - Unstructured overlays:
    - Query Flooding (Gnutella)
    - Hierarchical Query Flooding (KaZaA)
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Application classes, challenges
- Today's applications representative technology
  - recovery from jitter and loss
  - Streaming protocols
  - **CDN: overlays infrastructure for content delivery**



Next: would all this be simpler with Network (layer) support?

Reading nstructions and pointers for further study

P2P apps and overlays for info sharing Ch: 2.6 (2.5 7e)
Media Streaming & support from applications Ch 7.1-7.3 (9.1-9.3 & 2.6 7e)

- Upkar Varshney, Andy Snow, Matt McGivern, and Chr
  1 (January 2002), 89-96. DOI=10.1145/502269.50227
- Jussi Kangasharju, James Roberts, Keith W. Ross, O
  networks, Computer Communications, Volume 25, Iss
  3664, http://dx.doi.org/10.1016/S0140-3664(01)00409
- K.L Johnson, J.F Carr, M.S Day, M.F Kaashoek, The r
  networks, Computer Communications, Volume 24, Iss
  3664, http://dx.doi.org/10.1016/S0140-3664(00)00315
- Eddie Kohler, Mark Handley, and Sally Floyd. 2006. D
  reliability. *SIGCOMM Comput. Commun. Rev.* 36, 4 (A
  http://doi.acm.org/10.1145/1151659.1159918
- **Applications in p2p sharing, eg dissemination and me**
  - J. Mundinger, R. R. Weber and G. Weiss. Optimal
    of Scheduling, Volume 11, Issue 2, 2008. [**arXiv**] [
  - Christos Gkantsidis and Pablo Rodriguez, Networl
    INFOCOM, March 2005 (Avalanche swarming: co

Extra notes / for further study

KaZaA: Discussion

- Pros:
  - Tries to balance between search overhead and space needs (trading-off Napster's & Gnutella's extremes)
  - Tries to take into account node heterogeneity:
    - Peer's Resources (eg bandwidth)
- Cons:
  - No real guarantees on search scope or search time
  - Super-peers may "serve" a lot!

- P2P architecture used by Skype, Joost (communication, video distribution p2p systems)

(Recalling hash tables)

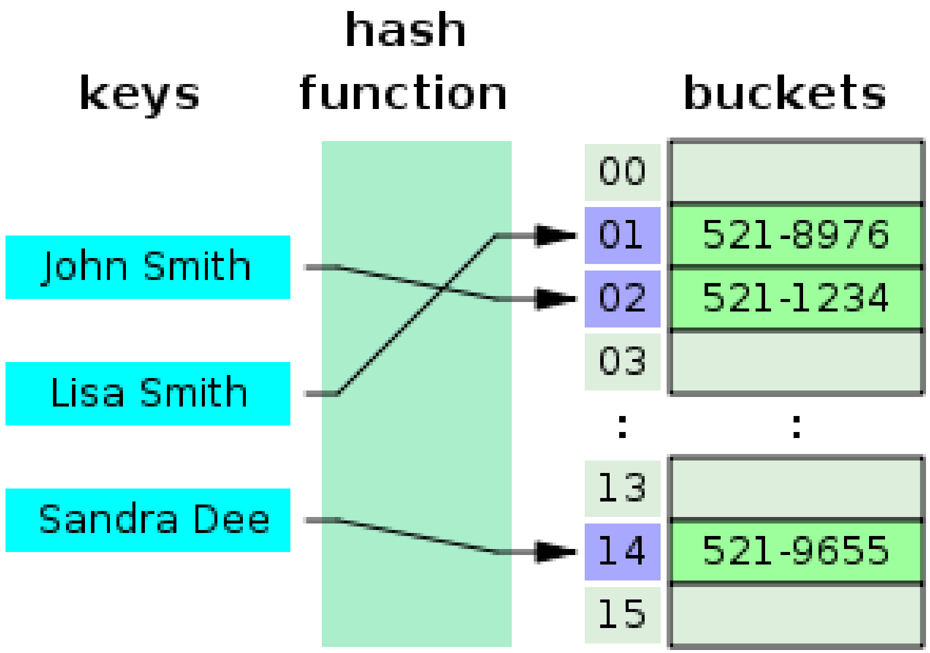


figure source: wikipedia; "Hash table 3 1 1 0 1 0 0 SP" by Jorge Stolfi - Own work. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg#/media/File:Hash_table_3_1_1_0_1_0_0_SP.svg

Distributed Hash Tables (DHT)

Implementation:

- Hash function maps entries to nodes (**insert**)
- Node-overlay has *structure* (Distributed Hash Table ie a distributed data structure, eg. Ring, Tree, cube) using it, do:
  - **Lookup/search**: find the node responsible for item; that one knows where the item is

Upon being queried:
"I do not know DFCD3454 but can ask some Neighbour/s in the DHT and propagate the search to find the owner"

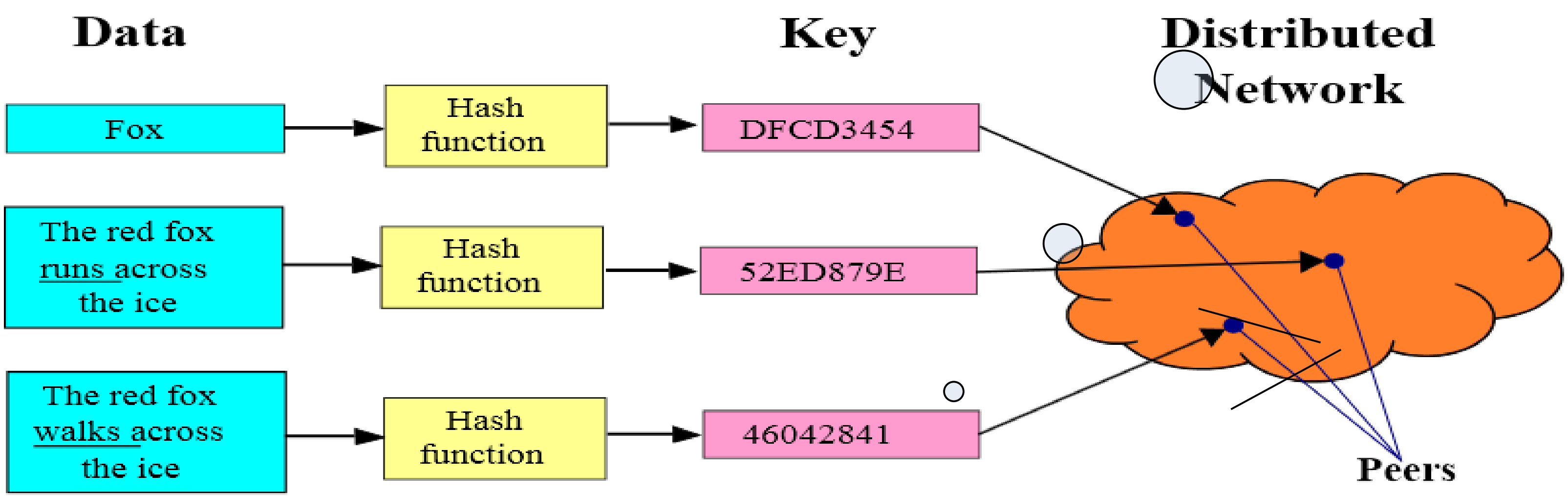


figure source: wikipedia

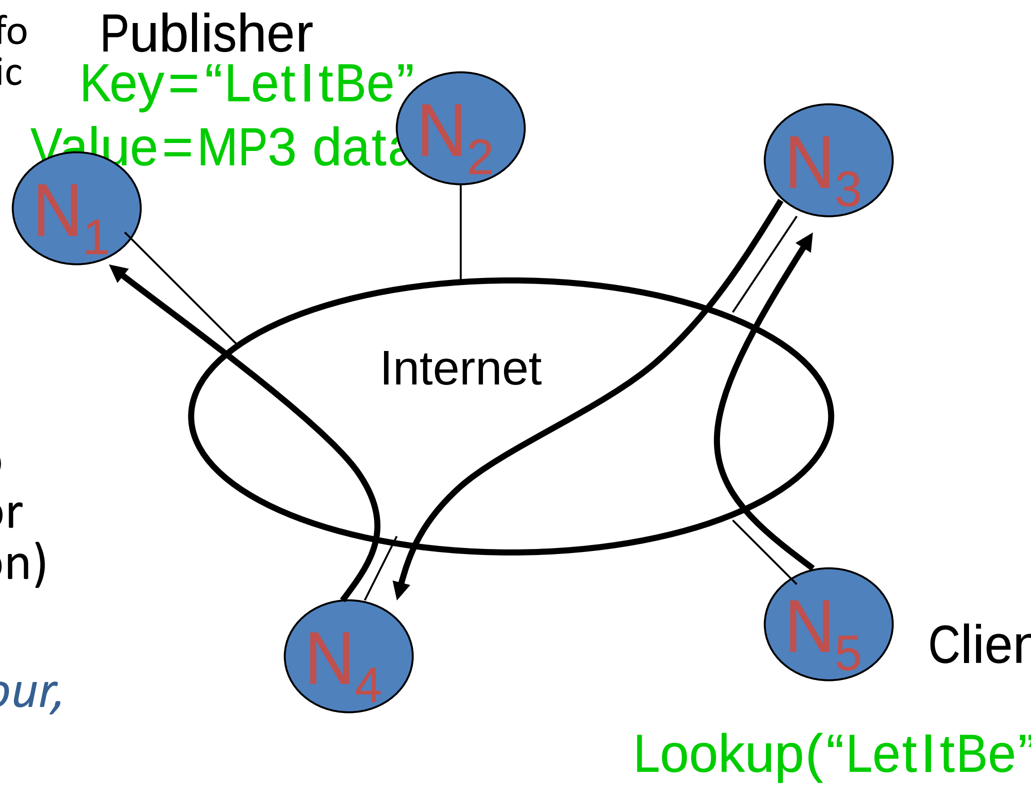
Distributed-Hash-Table-Based p2p sharing

- **Join**:
  - get connected in the overlay through info from bootstrap-node & using the specific DHT algorithm (eg Chord)
  - Start maintaining of files that you are responsible for (following the hash function)
  - NOTE: upon leaving DHT needs restructuring!

- **Publish**: tell which files you have, to the peers that will be responsible for them (according to the hash function)

- **Search**: ask *the appropriate neighbour,* who either is responsible for the searched file or will ask the next appropriate neighbor, and so on; guaranteed search time; commonly in O(logNodes)

- **Fetch**: get the file directly from peer

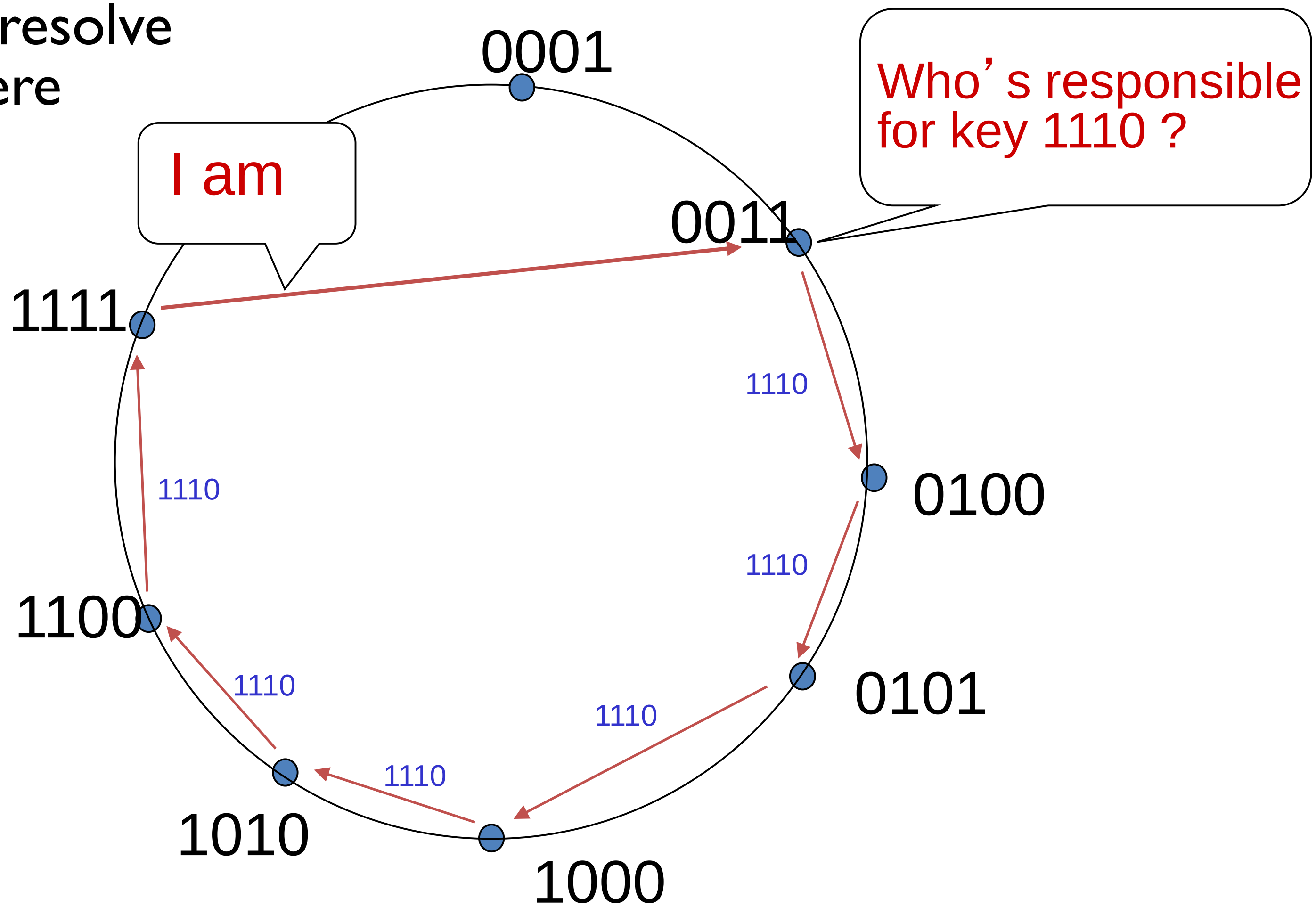


Challenges *[cf related literature@end of notes]:*

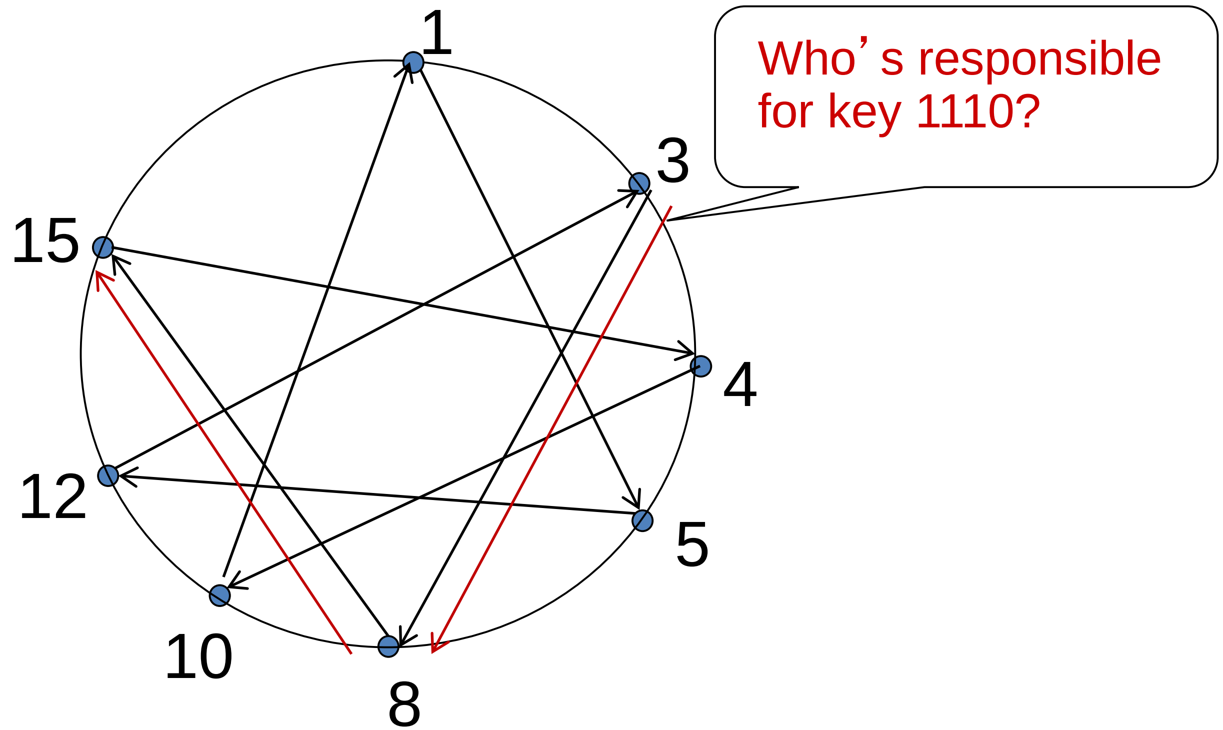
- Keep the hop count (asking chain) small
- Keep the routing tables (#neighbours) "right size"
- Stay robust despite rapid changes in membership (churn)

e.g. Circular DHT (I)

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



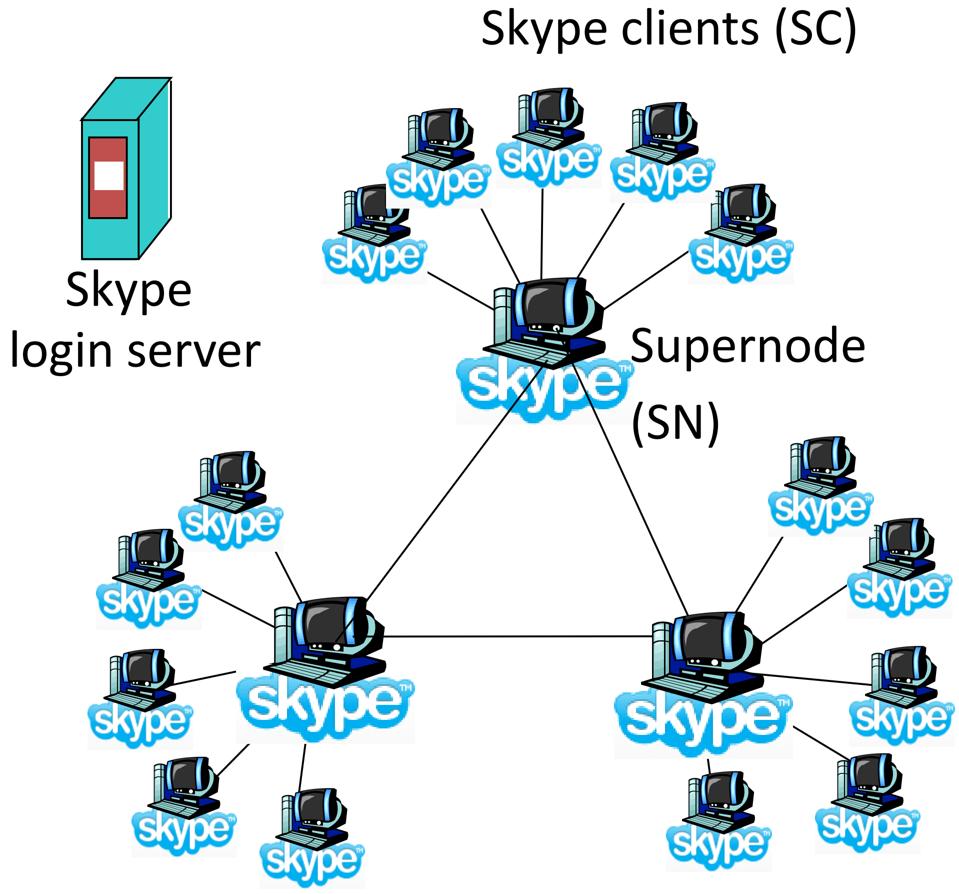
Circular DHT with shortcuts



- Here: reduced from 6 to 2 messages.
- possible to design shortcuts so *O(log N)* neighbors, *O(log N)* messages in query

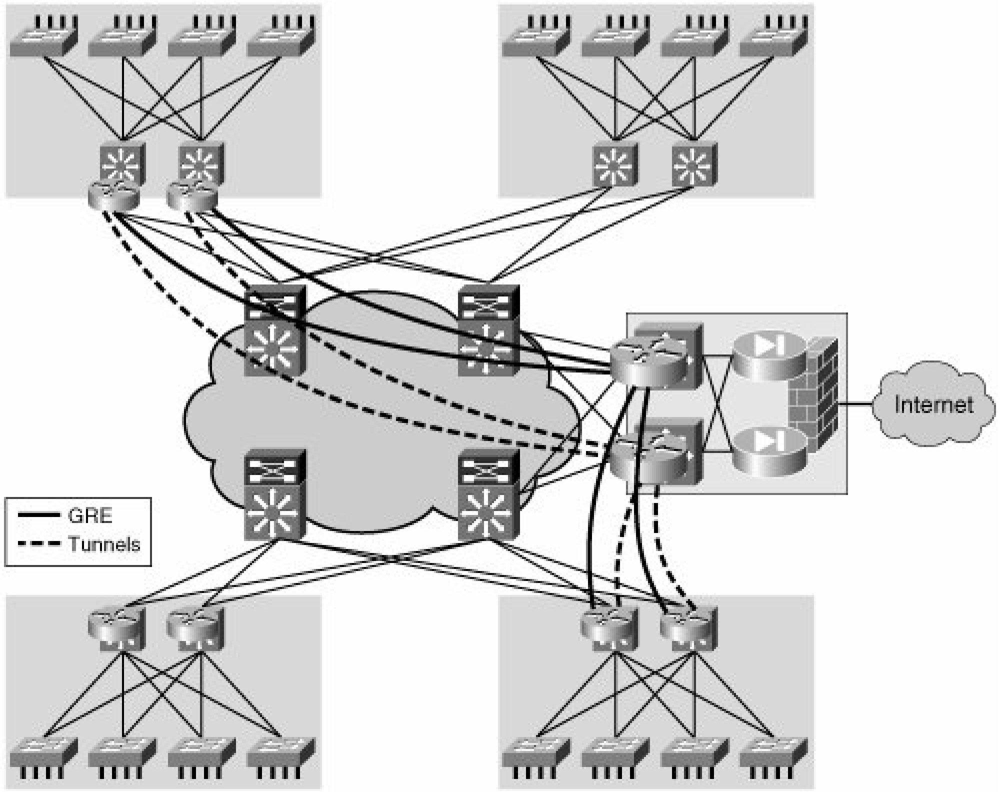
e.g. P2P & streaming Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



eg (2) Router Overlays – in support of Software Defined Networks

for e.g.
- distributing responibility of control and routing (5G)
- protection/mitigation of flooding attacks, collaborate for filtering flooding packets



Cf eg: Fu, Z., & Papatriantafilou, M. Off the Wall: Lightweight Distributed Filtering to Mitigate Distributed Denial of Service Attacks. In IEEE SRDS 2012.