Course on Computer Communication and Networks

Lecture 10

Continuously evolving Internet-working

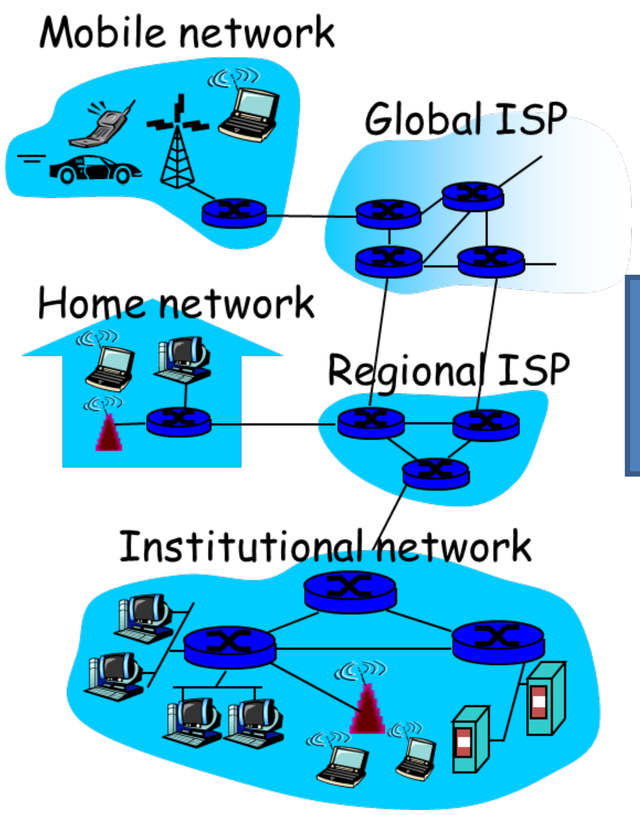
Part A: p2p networking, media streaming, CDN

(*TBC in part B: QoS, traffic engineering, SDN, IoT*)

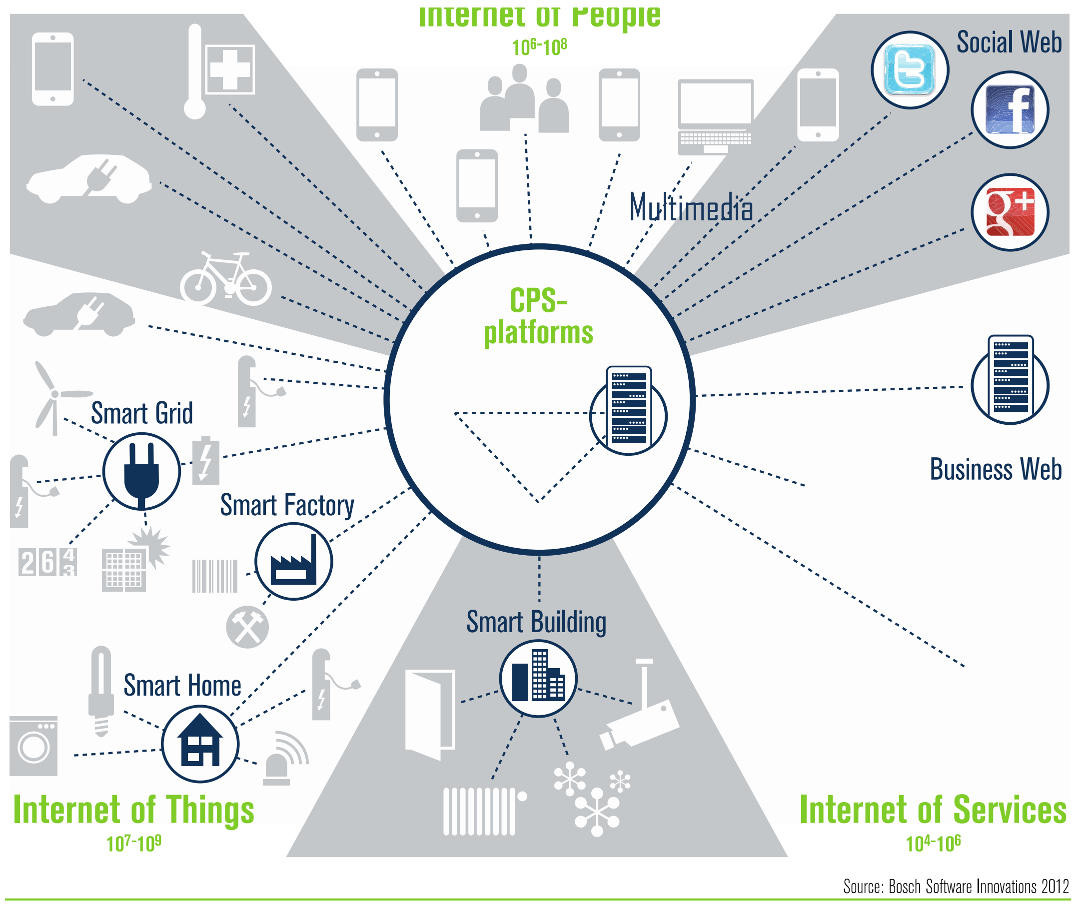
EDA344/DIT 423, CTH/GU

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

Internet & its context….

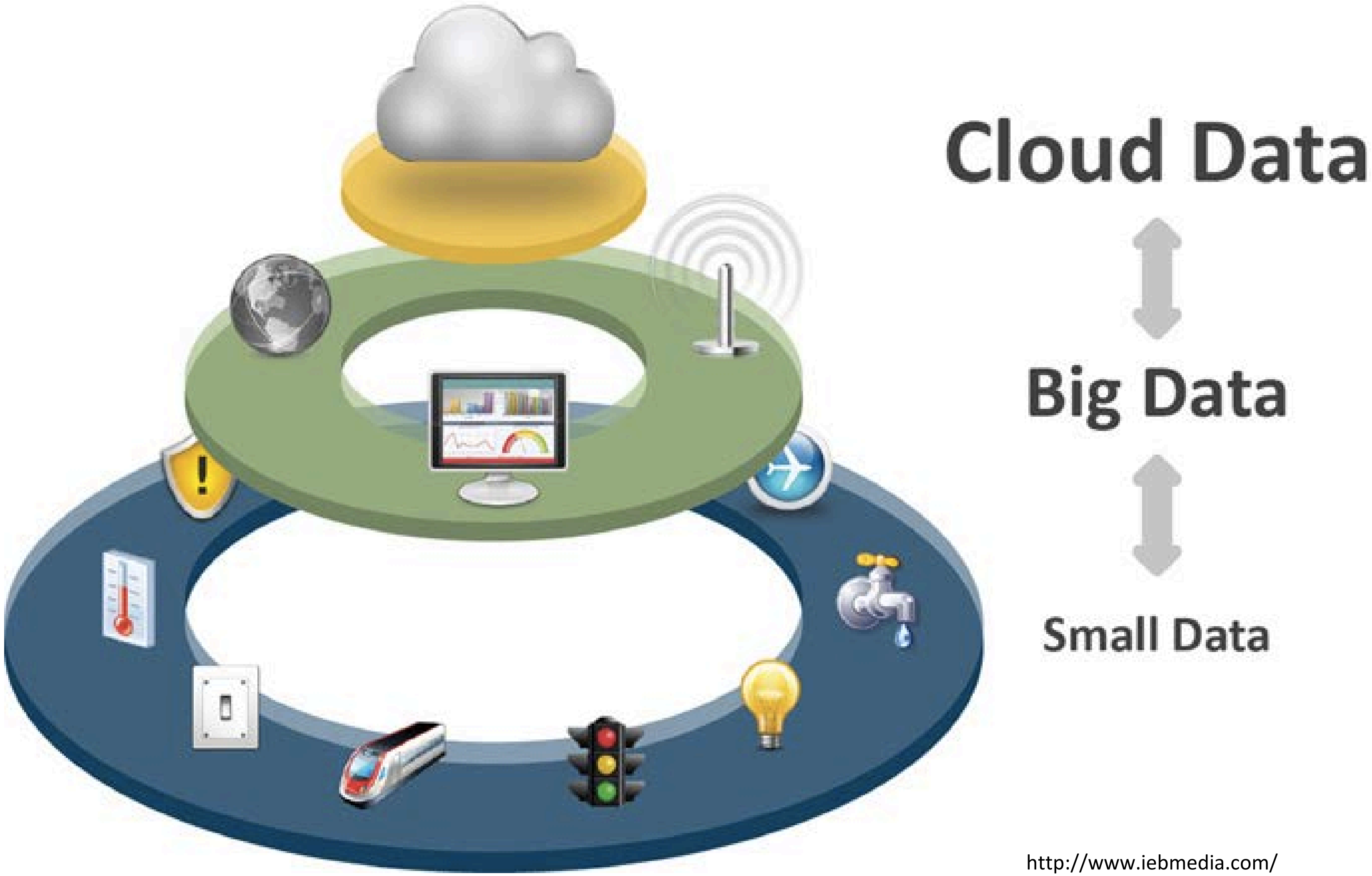


approx 10 yrs ago



continuous evolution ….

Internet, Data processing and Distributed Computing in interplay: IoT



http://www.iebmedia.com/

Internet protocol stack layers&protocols

Application: protocols supporting *network applications*

http (*web*), smtp (*email*), p2p, streaming, CDN, …

transport: process2process (end2end) data transfer

UDP, TCP

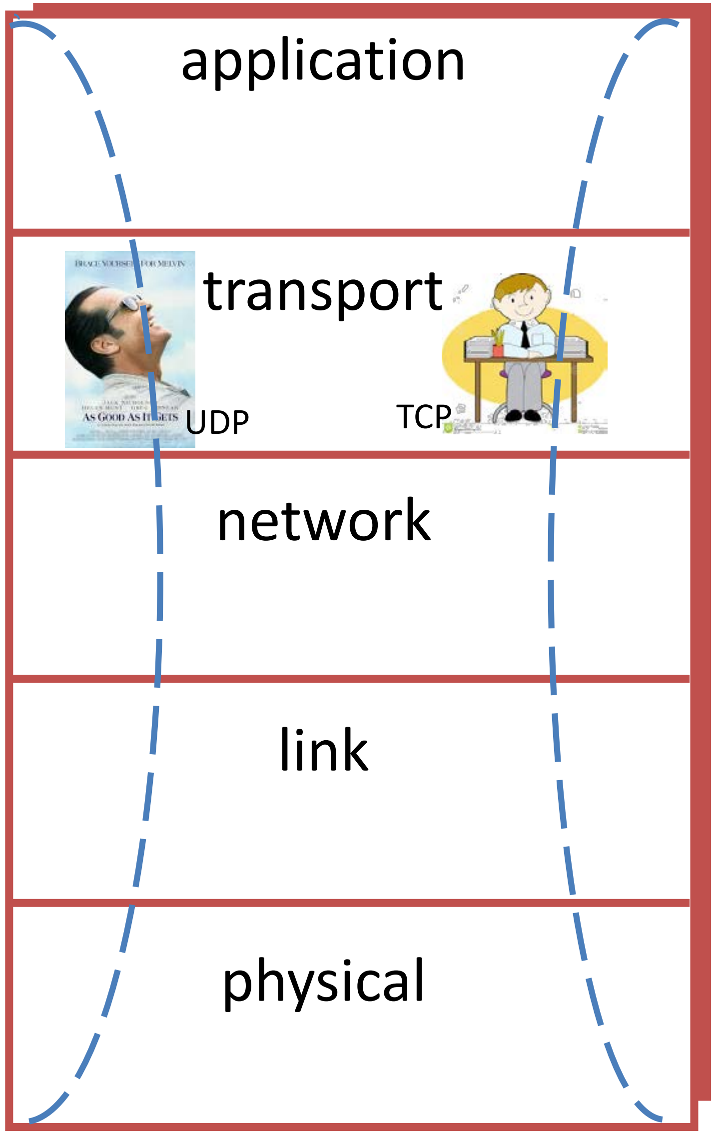
network: routing of datagrams (independent data-packets), connecting different physical networks

IP addressing, routing protocols, virtualization, virtualization, …

link: data transfer between neighboring (physically connected) hosts

Ethernet, WiFi, …

physical: bit-transmission on the physical medium between neighboring nodes



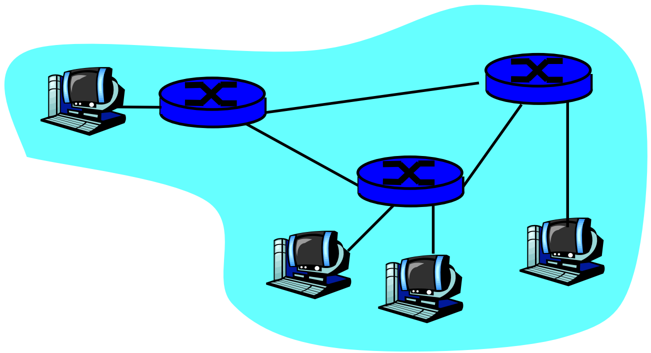
Recall:
the Internet concept: virtualizing networks

1974: multiple unconnected nets

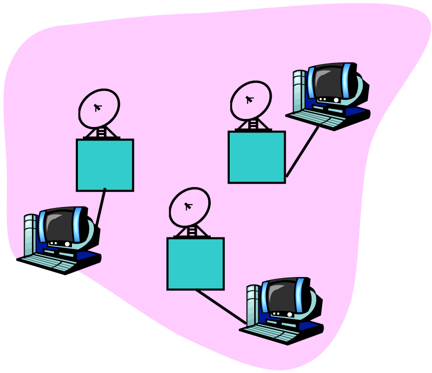
- ARPAnet
- data-over-cable networks
- packet satellite network (Aloha)
- packet radio network

… 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.

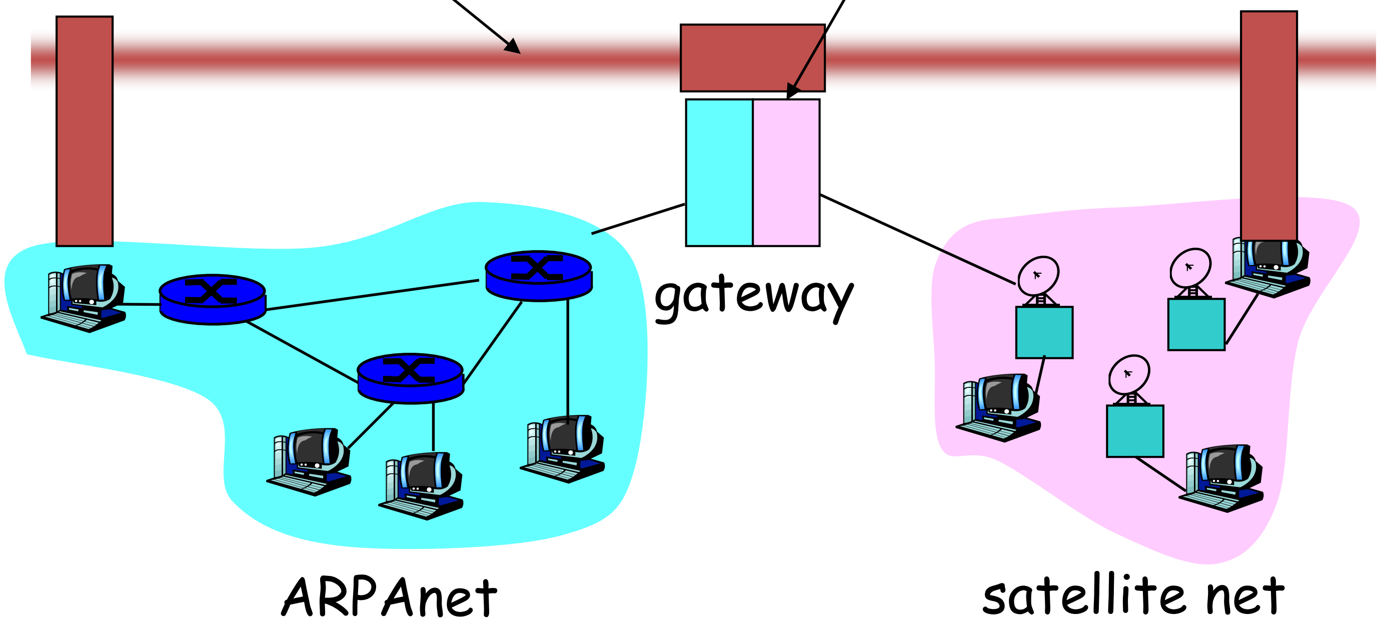
The Internet: virtualizing networks

Internetwork layer (IP):

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

Gateway:

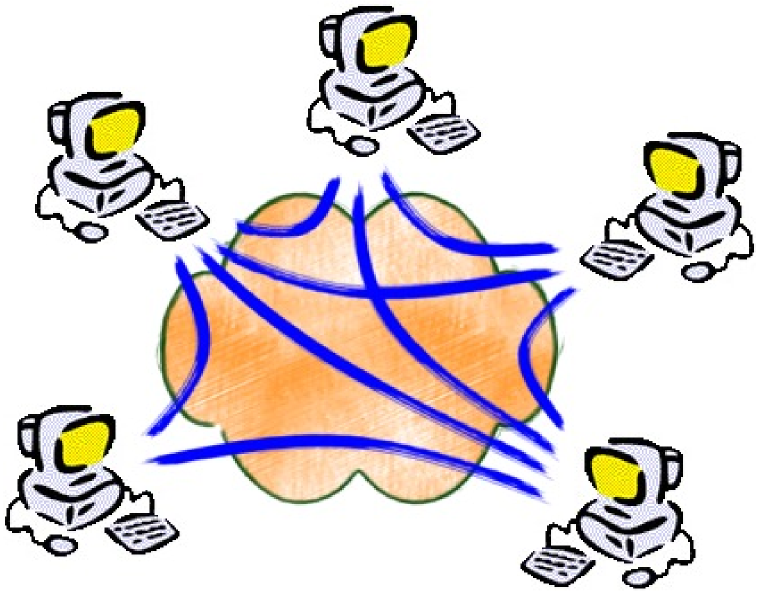
- "embed internetwork packets in local packet format"
- route (at internetwork level) to next gateway



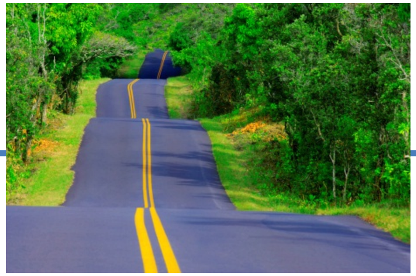
Notice: virtualization & network overlays

Overlay: a network implemented on top of a network

What else to do with this?



Roadmap



P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

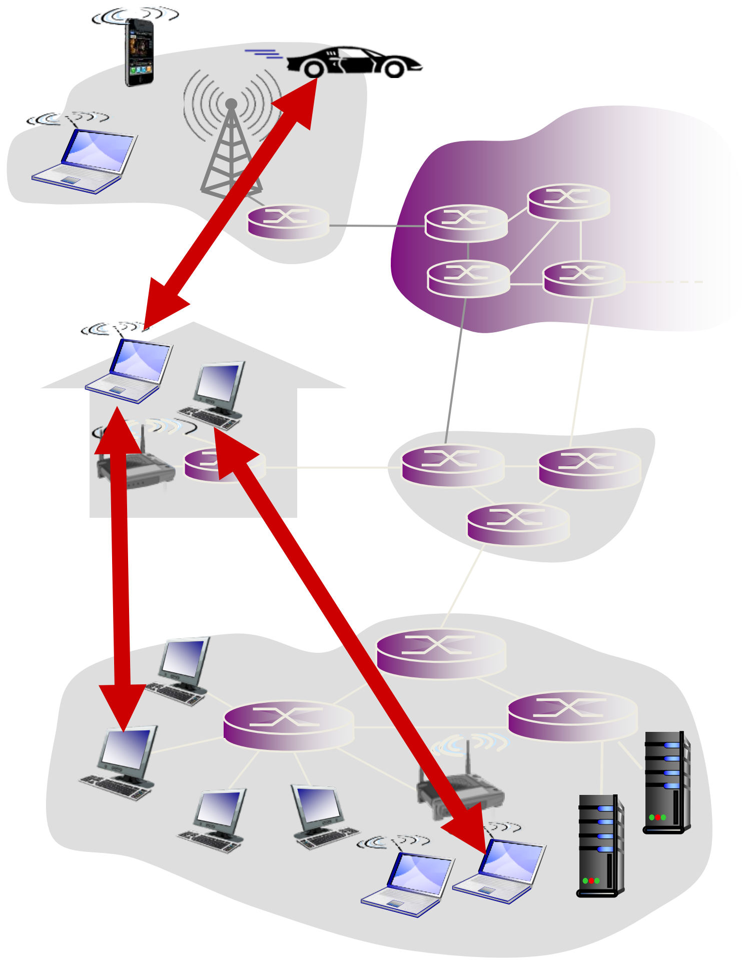
- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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

Pure P2P architecture

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



examples:

- **file distribution/sharing**
- Streaming multimedia (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?
- **Fetch**: how to I retrieve a file?

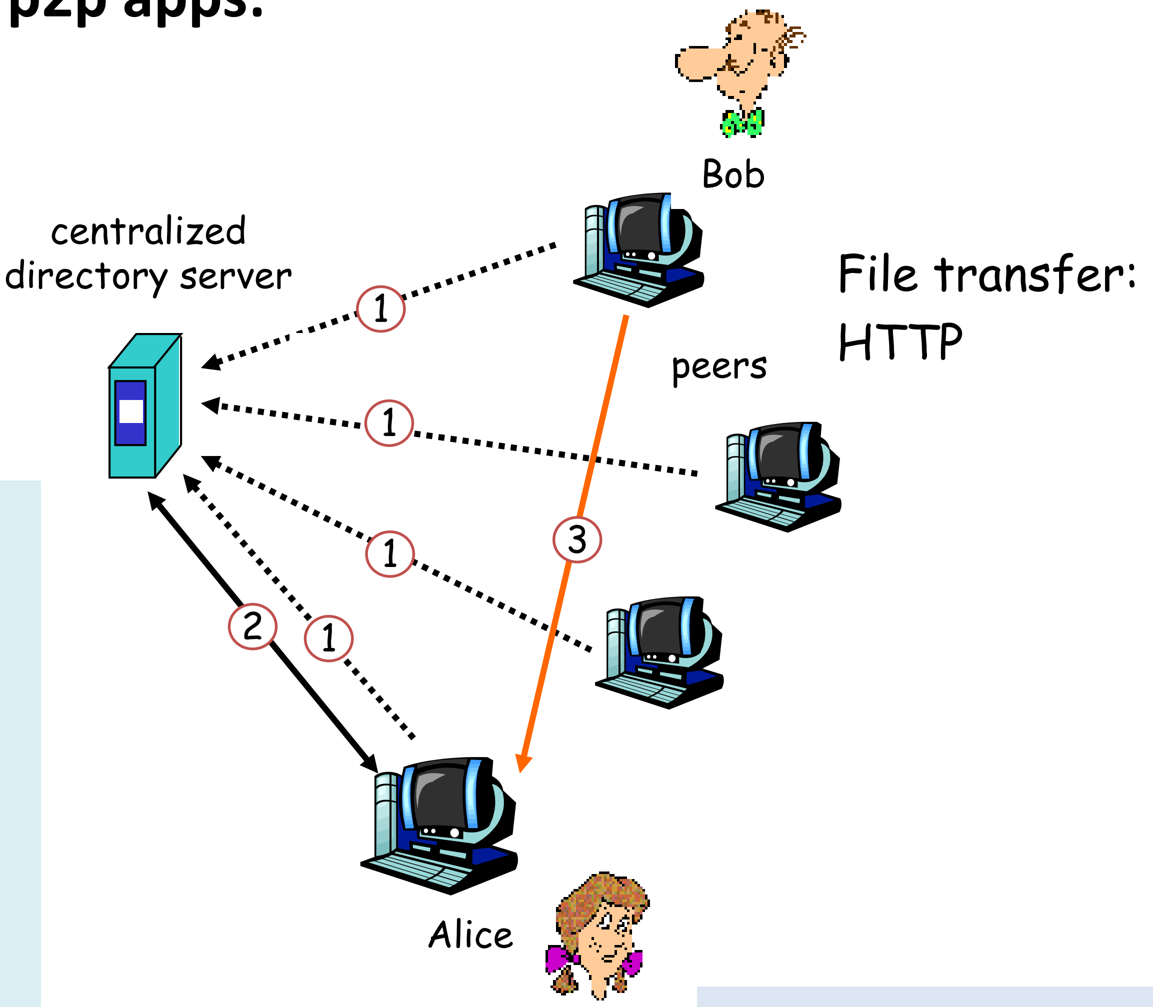
P2P: centralized directory

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

1) when peer connects, it informs central server:
   - IP address, content

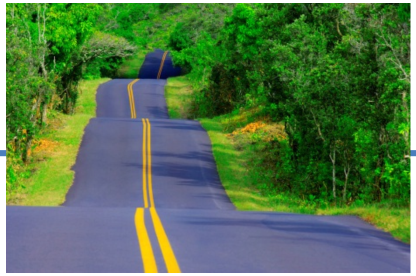
2) Alice queries directory server for “LetItBe”

3) Alice requests file from Bob



Q: What is p2p in this?

Roadmap



P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- **Collaborate/form-*overlays* to *find* content:**
  - **Unstructured overlays**
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

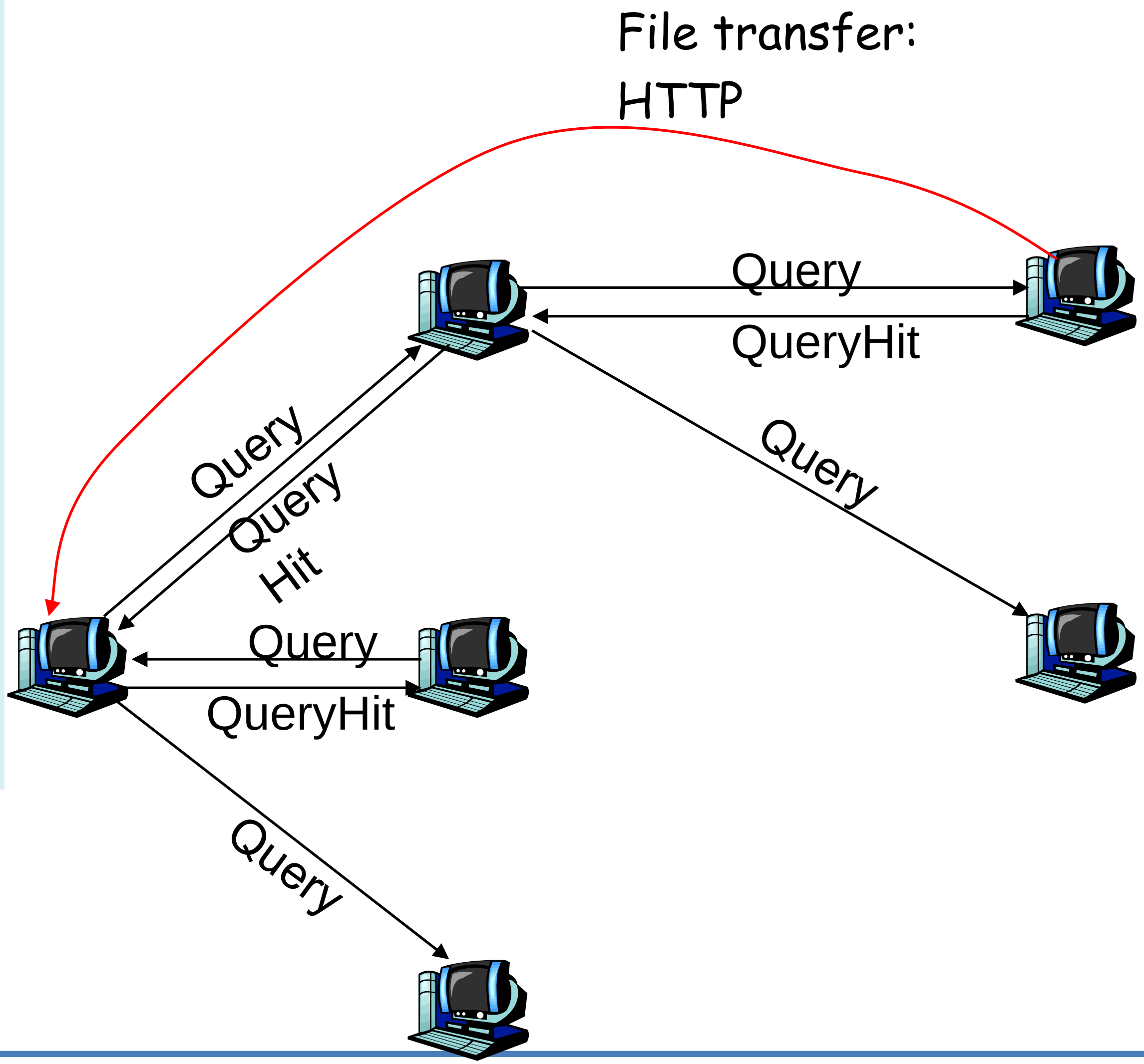
Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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

P2p Gnutella (no directory): protocol

Query Flooding:

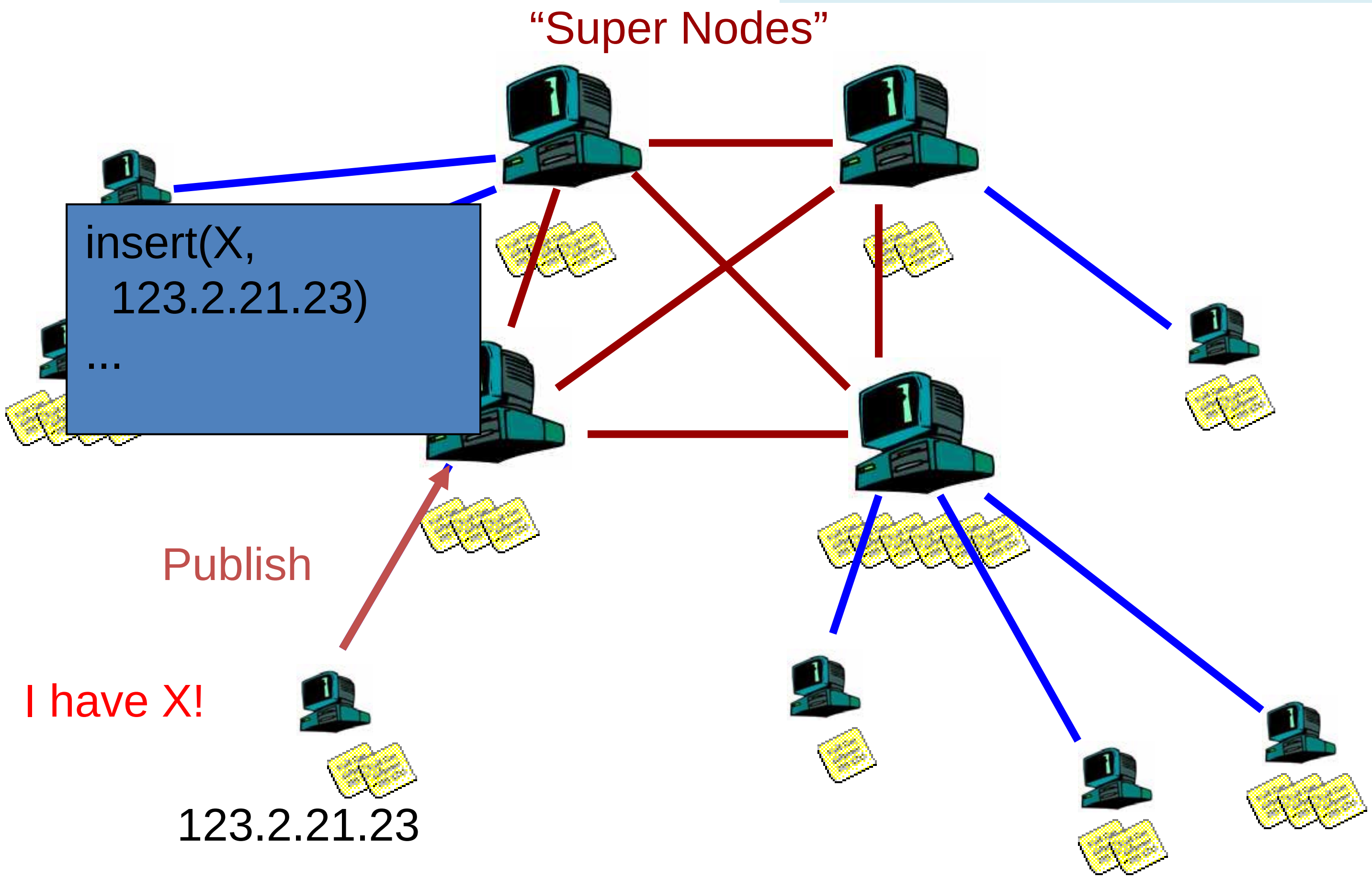
- **Join**: on startup, client connects to a few other nodes (learn from bootstrap-node); these become its "neighbors" (overlay!! ☺)
- **Publish**: no need
- **Search**: ask *"neighbors",* who ask their neighbors, and so on... when/if found, reply to sender.
- **Fetch**: get the file directly from peer



KaZaA : distributed directory

"Smart" Query Flooding:

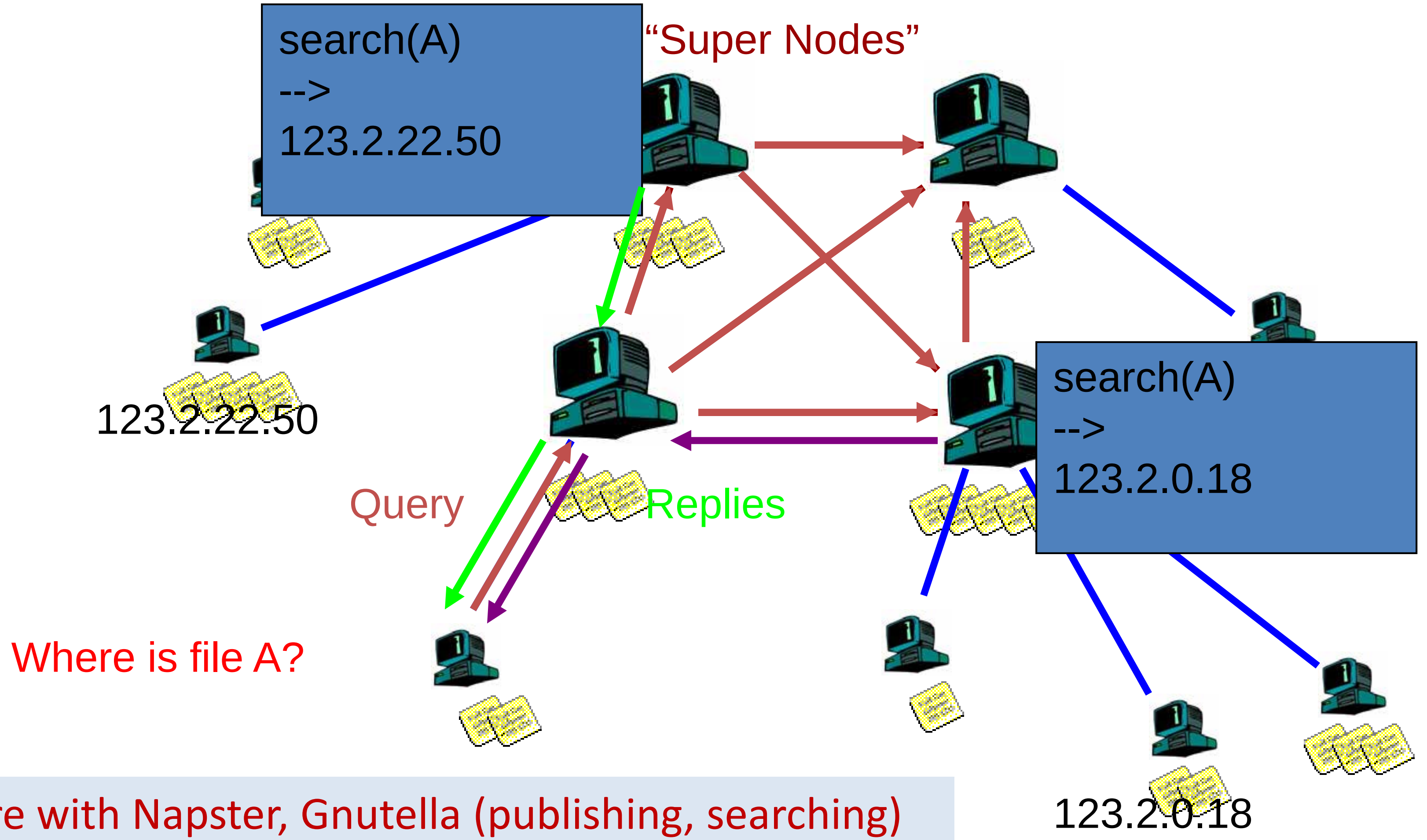
- **Join**: on startup, client contacts a "supernode" ... may at some point become one itself
- **Publish**: send list of files to supernode



KaZaA: Search

"Smart" Query Flooding:

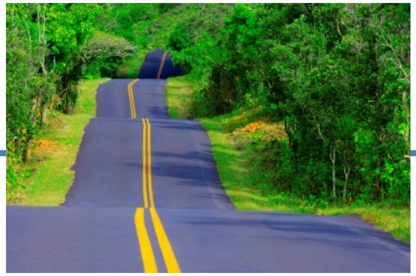
- **Search**: send query to supernode, supernodes flood query amongst themselves.
- **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers



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

Skype's architecture (so far…)

Roadmap



P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - **Structured Overlays/DHT**
- Collaborate/form-*overlays to fetch* content

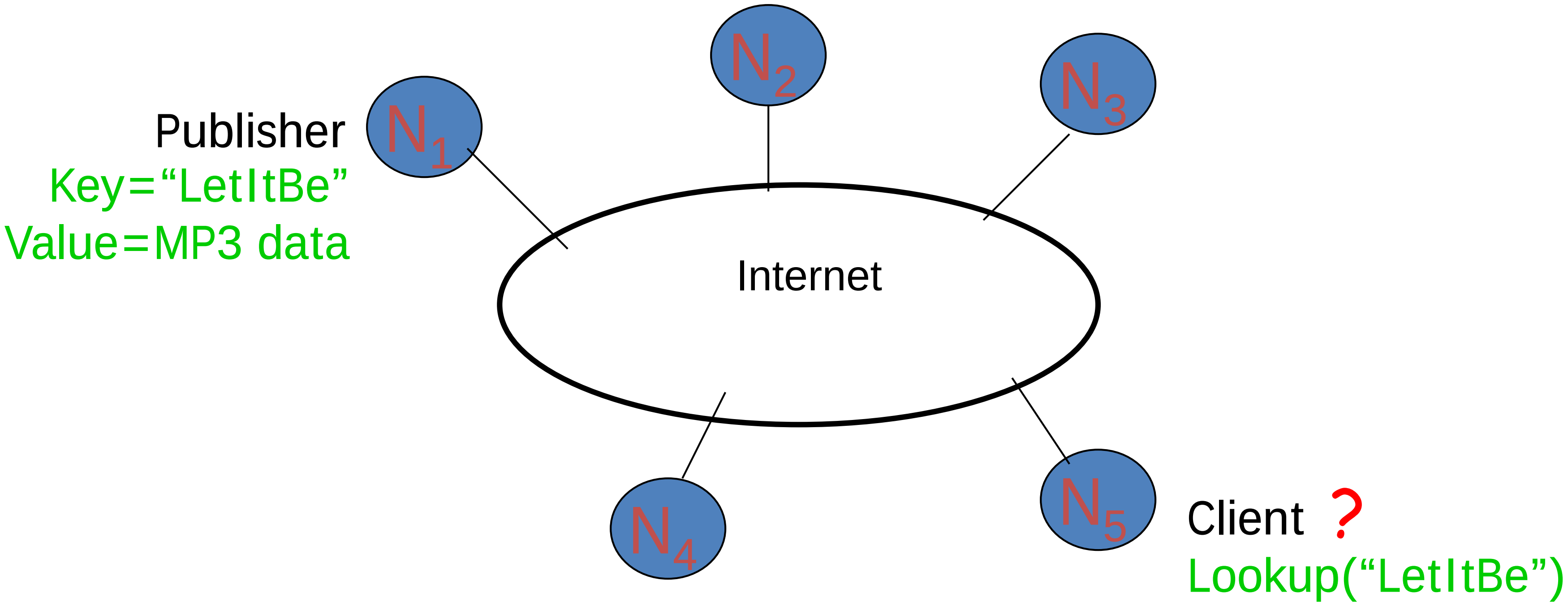
Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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

Problem revisited: formulation

How to find data in a distributed file sharing system?

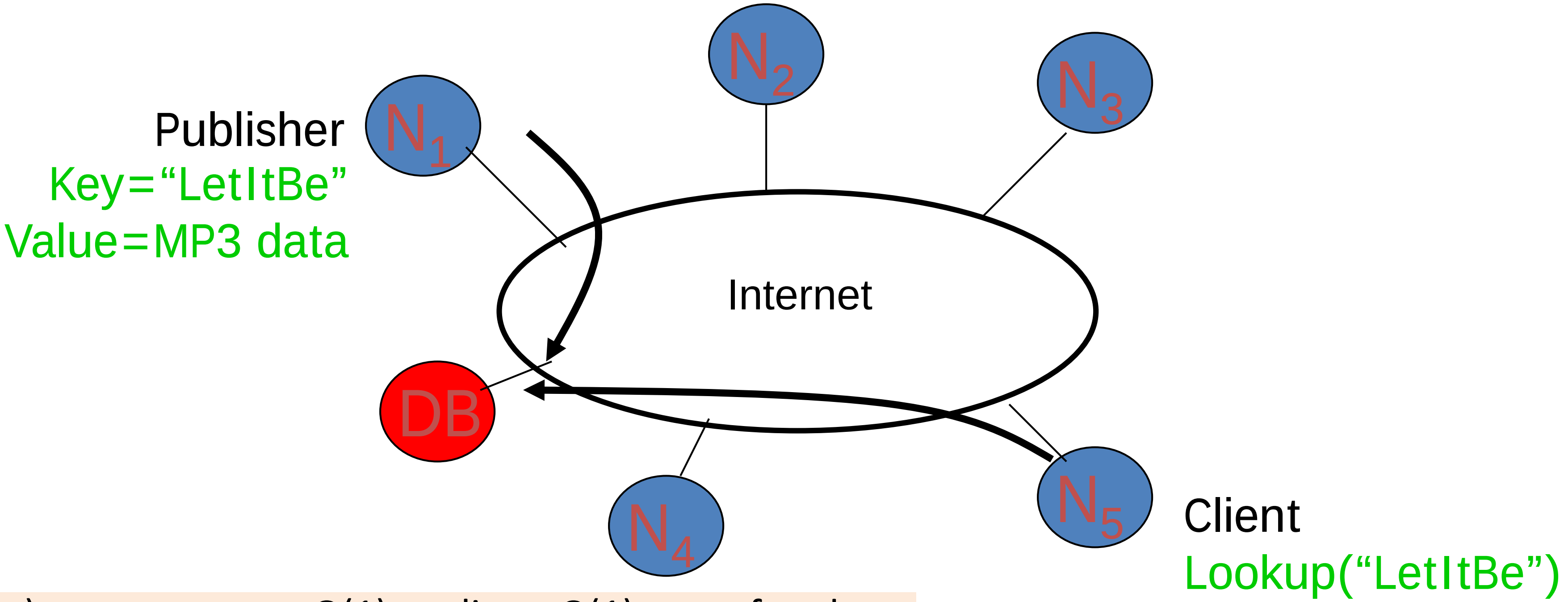
(aka "routing" to the data, i.e. content-oriented routing)



How to do Lookup?

Centralized Solution

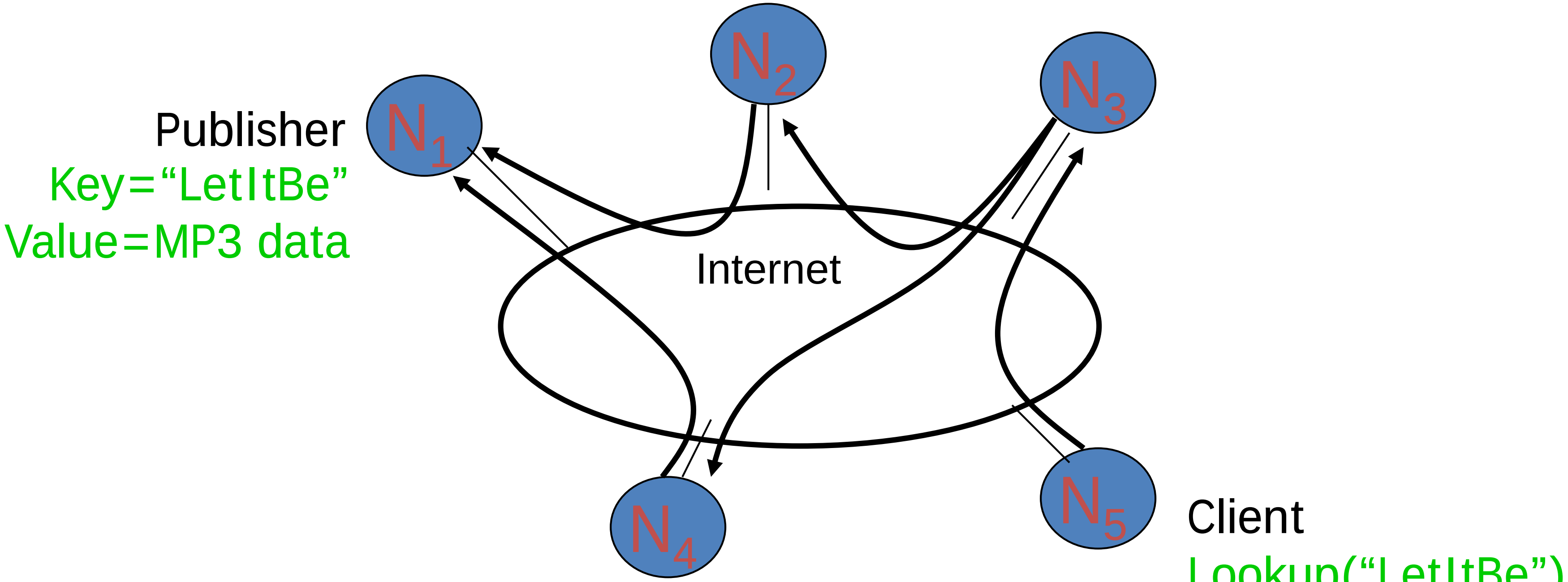
Central server (Napster)



- O(peers,items) state at server, O(1) at client, O(1) cost of update
- O(1) search communication overhead
- Single point of failure

Fully decentralized (elementary distributed) solution

Flooding (Gnutella, etc.)



- O(1) state per node, O(1) cost of update
- 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) :

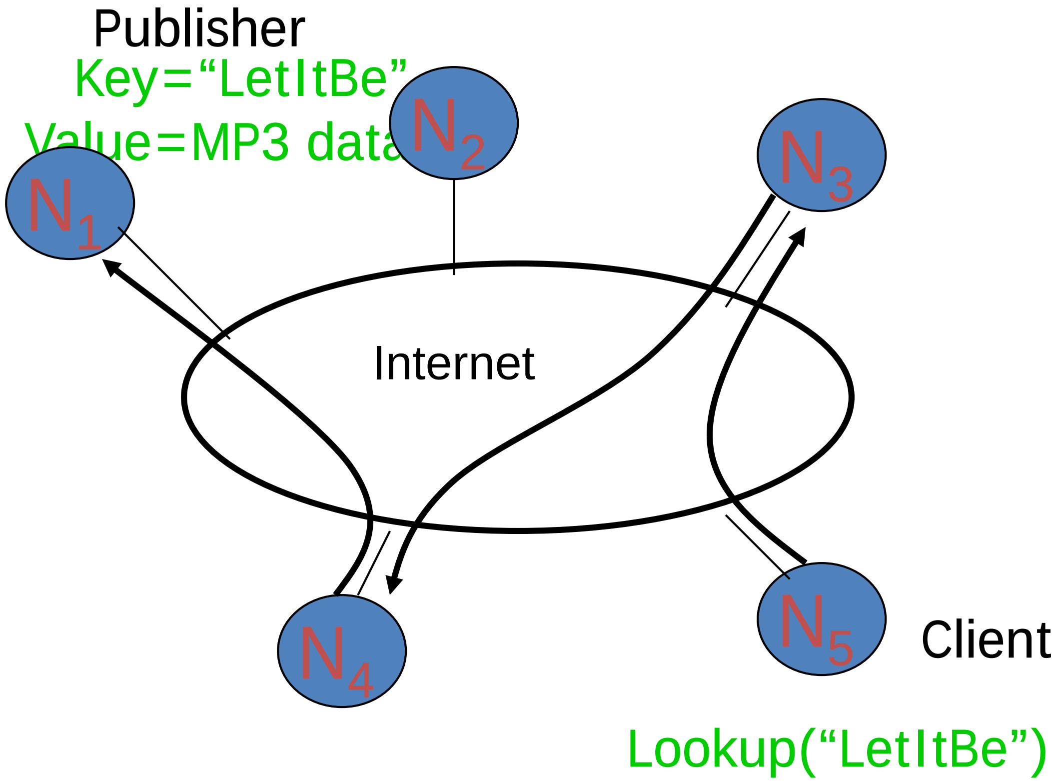
insert(key, item);

item = get(key);

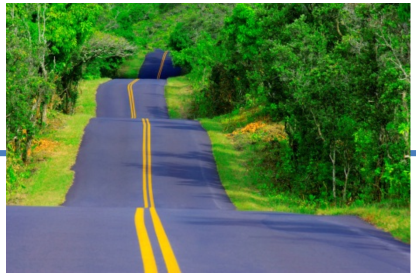
Each node (peer) is **responsible for**

- **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 (used for propagating queries) 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: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- **Collaborate/form-*overlays to fetch* content**

Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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

Second generation p2p: focus on fetching

Motivation:

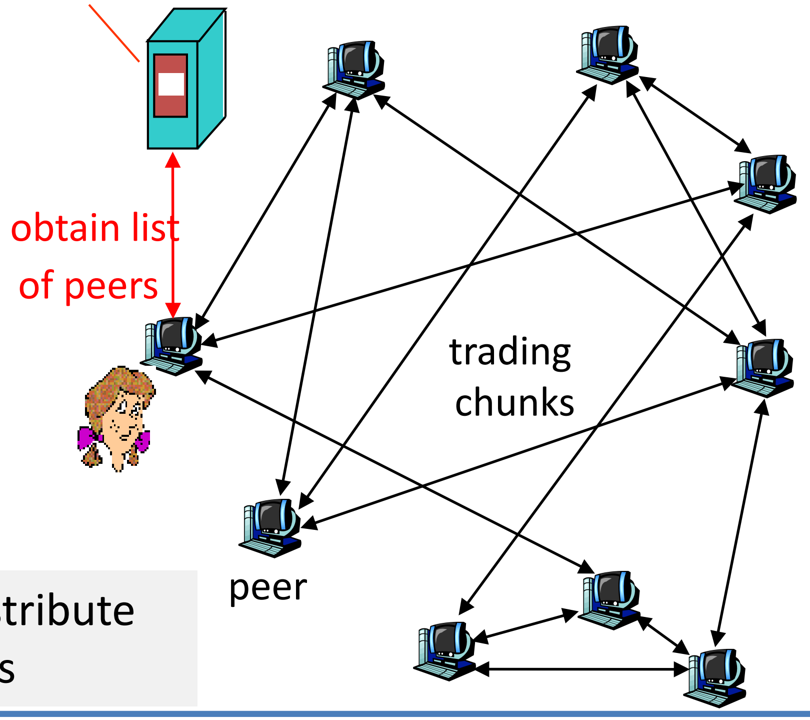
- Popularity exhibits temporal locality (Flash Crowds)
- Can bring file "provider" to "its knees"

Idea:

- Files are "chopped" in chunks; fetch from many sources (**swarming**)
- Overlay: nodes "hold hands" with those who share chunks at similar rates

torrent: group of peers exchanging chunks of a file

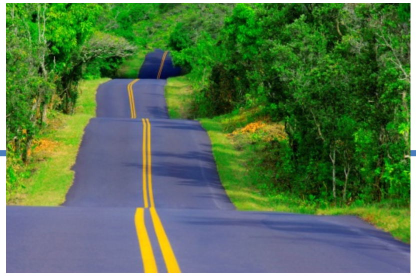
tracker: tracks peers participating in torrent



- **Join**: contact a server, aka "tracker" get a list of peers.
- **Publish**: can run a tracker server.
- **Search**: Out-of-band. E.g., use search-engine, or DHT, … to find a tracker, which gives list of peers to contact
- **Fetch**: Download chunks from several of peers. Upload chunks you have to them.

Used by publishers to distribute software, other large files

Roadmap



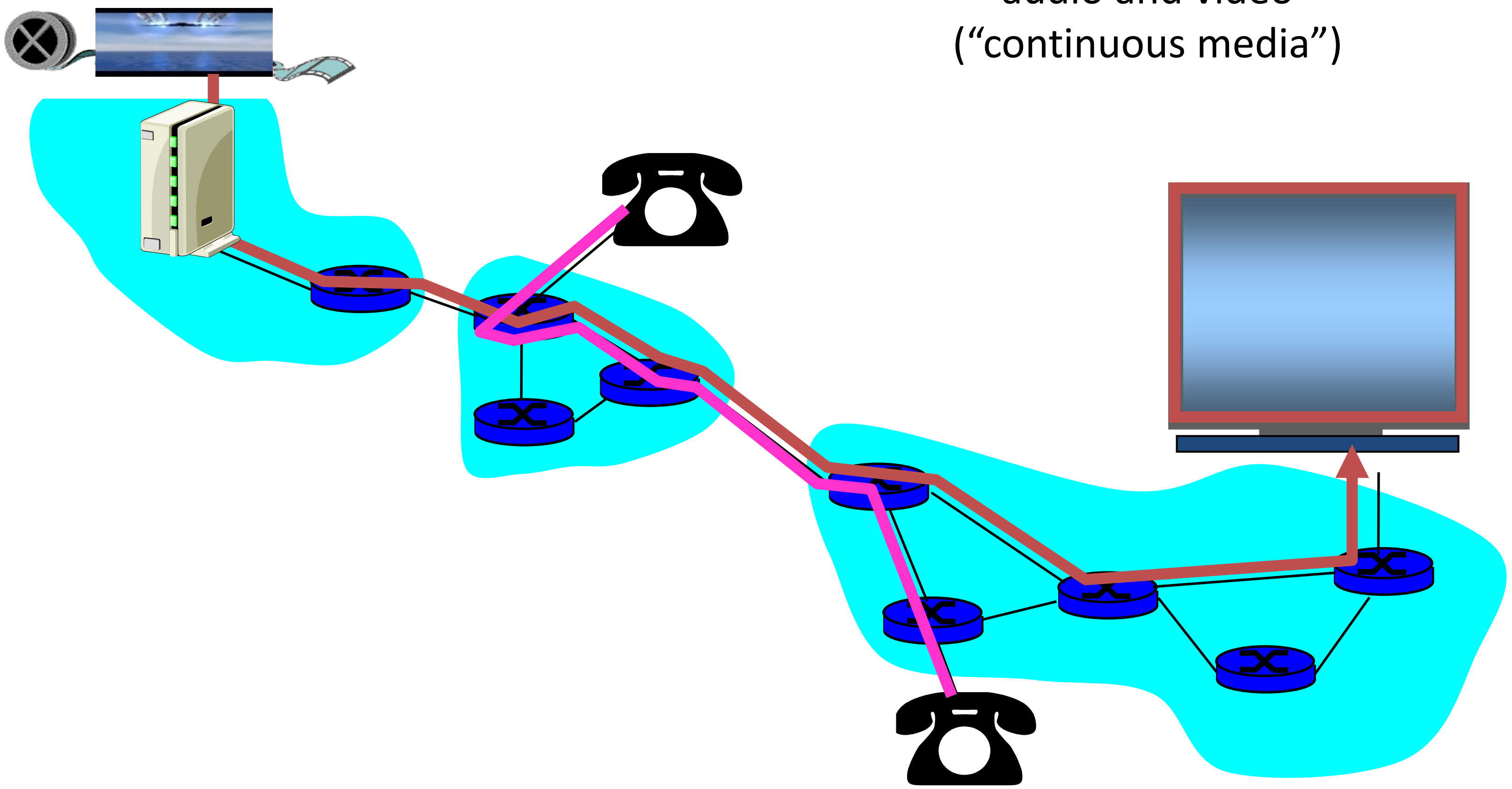
P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

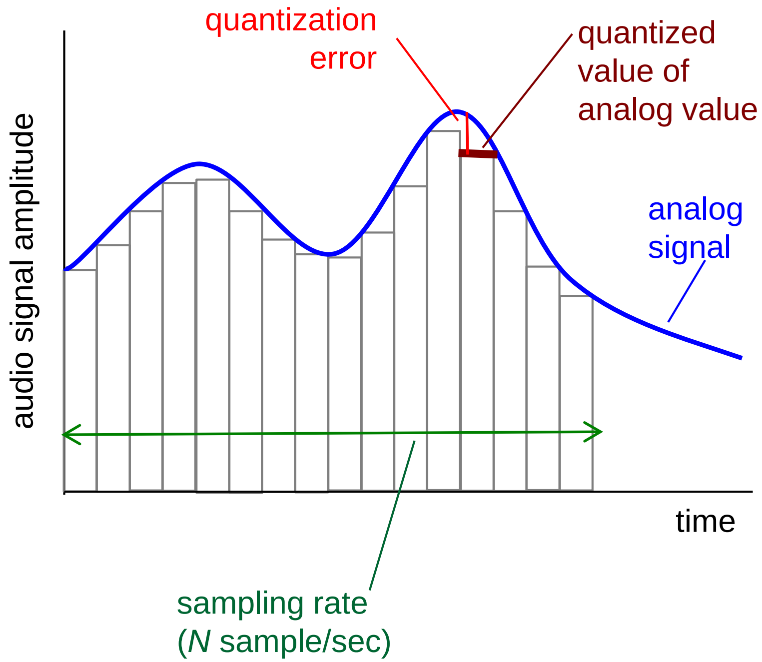
- Application classes, challenges
- Today's applications representative technology
  - Recovery from jitter and loss
  - Streaming protocols and new proposals
  - CDN: overlays infrastructure for content delivery

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
  - eg: 8,000 samples/sec, 256 quantized levels: 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

- 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)

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*

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

entire

kype

ity of packet delays
cket stream

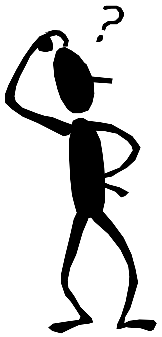
Fundamental characteristics:

- typically **delay sensitive**
  - end-to-end delay
  - delay *jitter*
- **loss tolerant**: infrequent losses cause only minor glitches

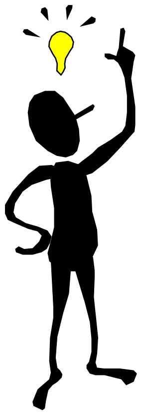
- In contrast to traditional data-traffic apps, which are loss *intolerant* but delay *tolerant*.

Recall Internet's transport services: TCP, UDP

no guarantees on delay….



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



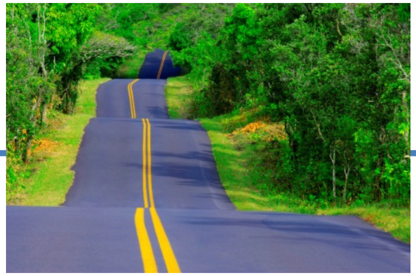
Internet multimedia applications
use application-level techniques to mitigate
(as best possible) effects of delay, loss;
(Also complementing with "traffic engineering"
& Software-Defined Networking *in NW core*: discussion next lectures)

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: 2.5 7e (2.6 6/e)

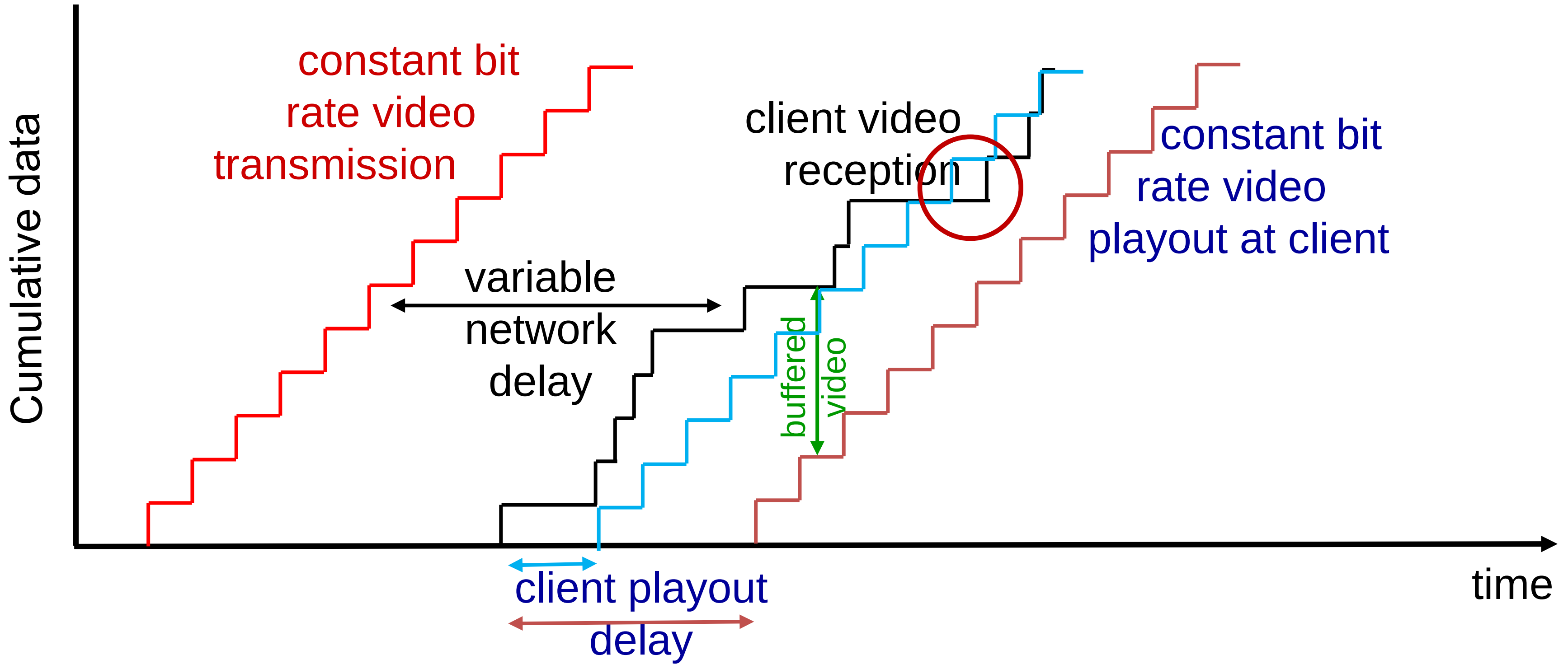
- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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

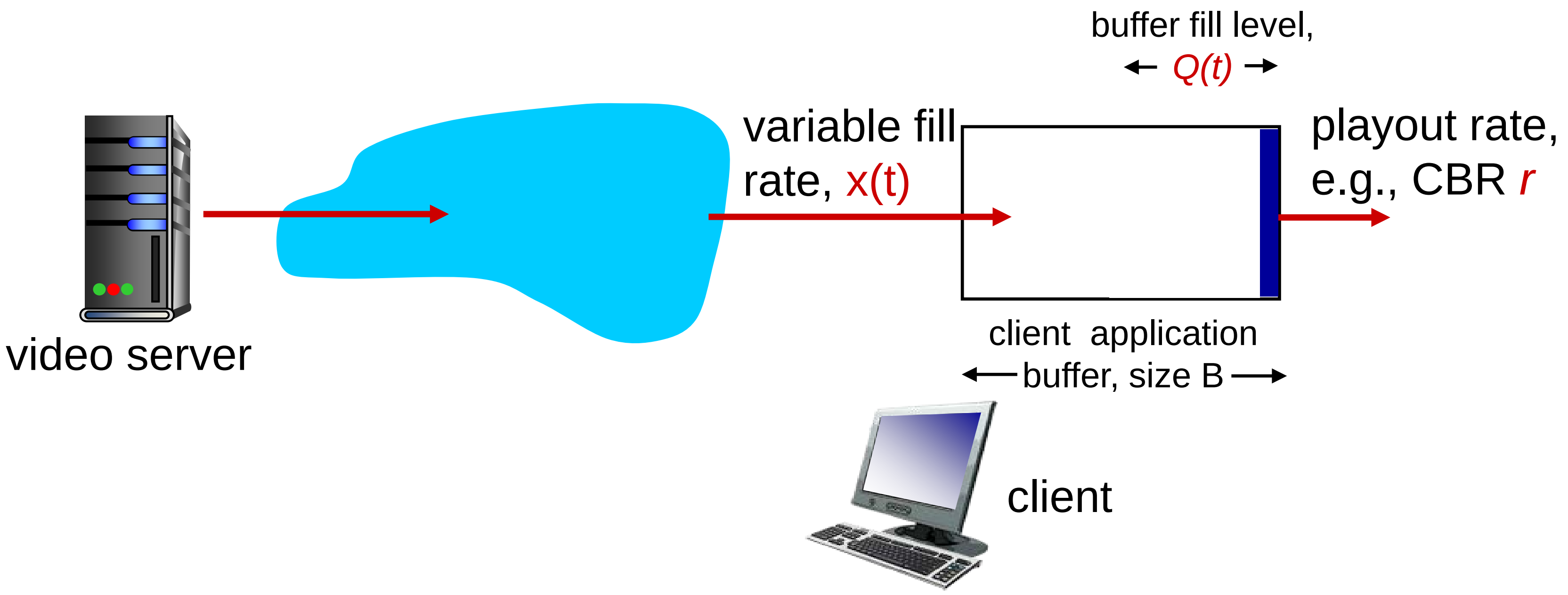
Streaming: recovery from jitter

playout delay small => higher loss rate



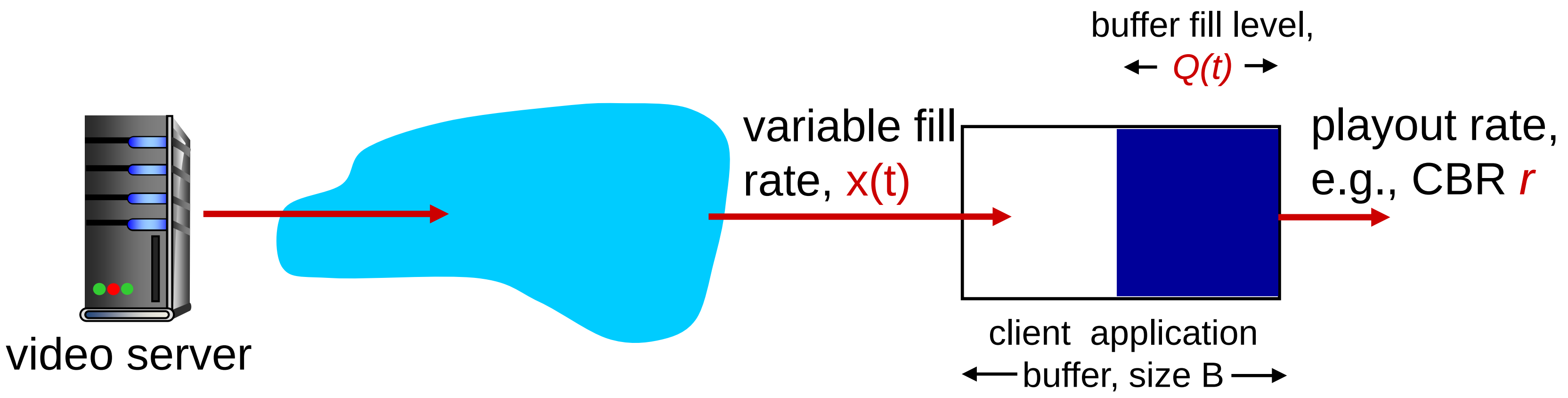
- ❖ *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout



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

Client-side buffering, playout



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

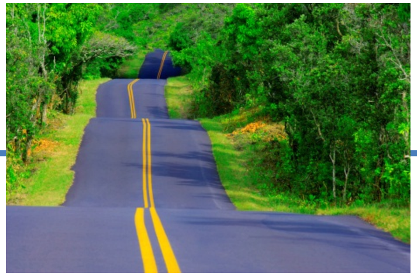
- $\bar{x}$ < r: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x}$ > r: need to have enough buffer-space to absorb variability in x(t)

initial playout delay tradeoff:

\+ buffer empty less likely with larger delay, but

\- larger delay until user begins watching

Roadmap



P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

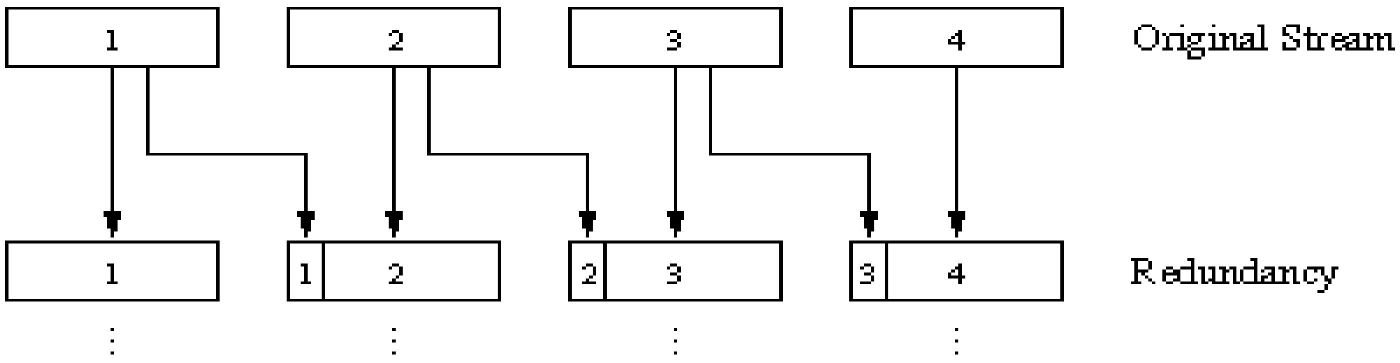
Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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

Recovery From Packet Loss: Forward Error Correction (FEC)

Eg. 1. through piggybacking Lower Quality Stream

sender

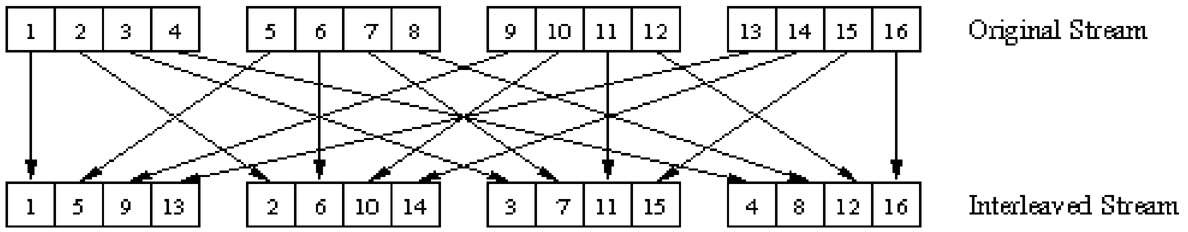


Recovery From Packet Loss/FEC (cont)

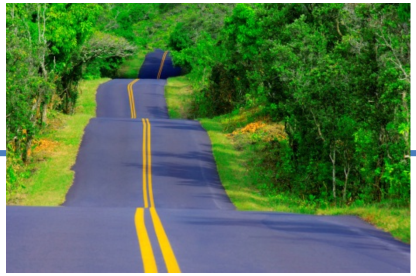
2. Interleaving:

– Upon loss, have a set of partially filled chunks
– Playout time must adapt to receipt of group (risk wrt Real-Time requirements)

sender



Roadmap



P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

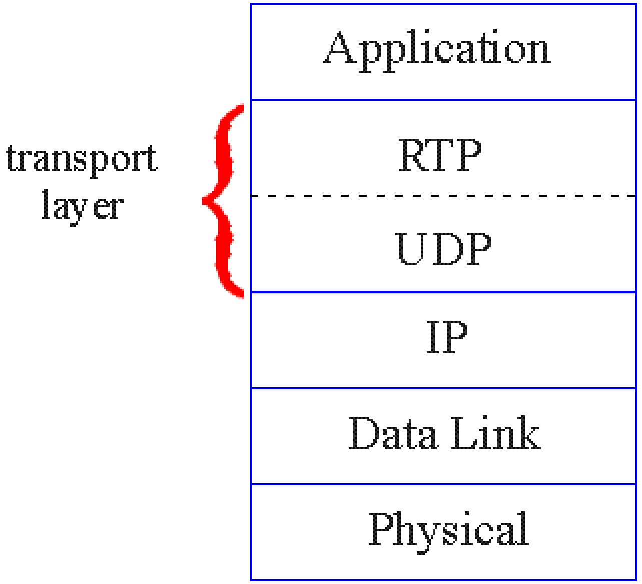
- Application classes, challenges
- Today's applications representative technology
  - Recovery from jitter and loss
  - Streaming protocols and new proposals
  - CDN: overlays infrastructure for content delivery

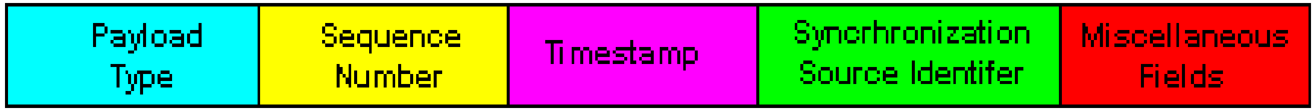
Real-Time Protocol (RTP) RFC 3550

- 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





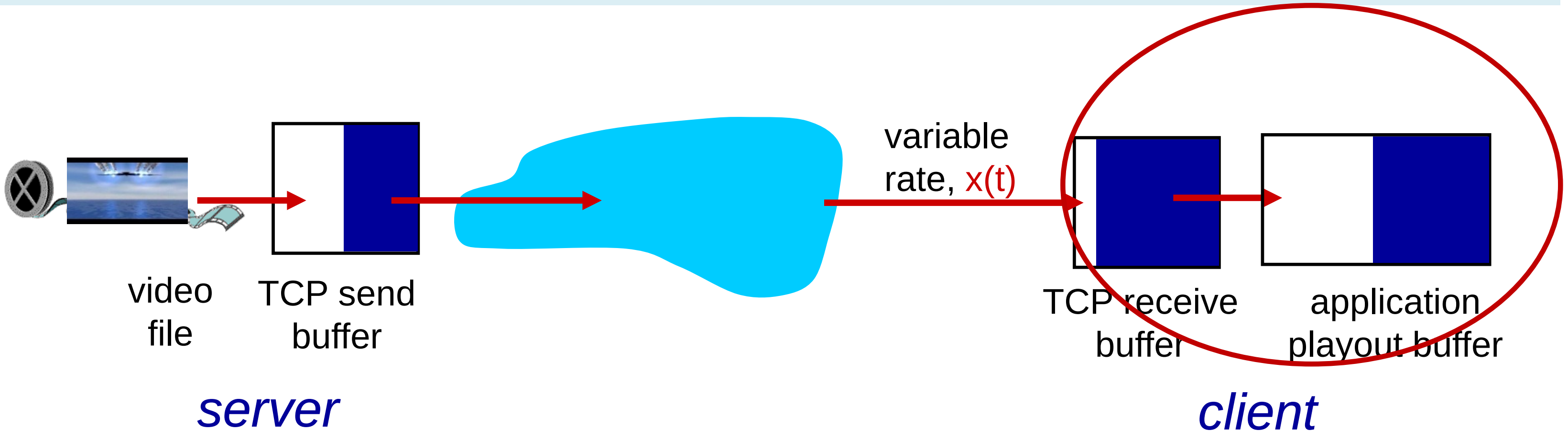
RTP Header

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

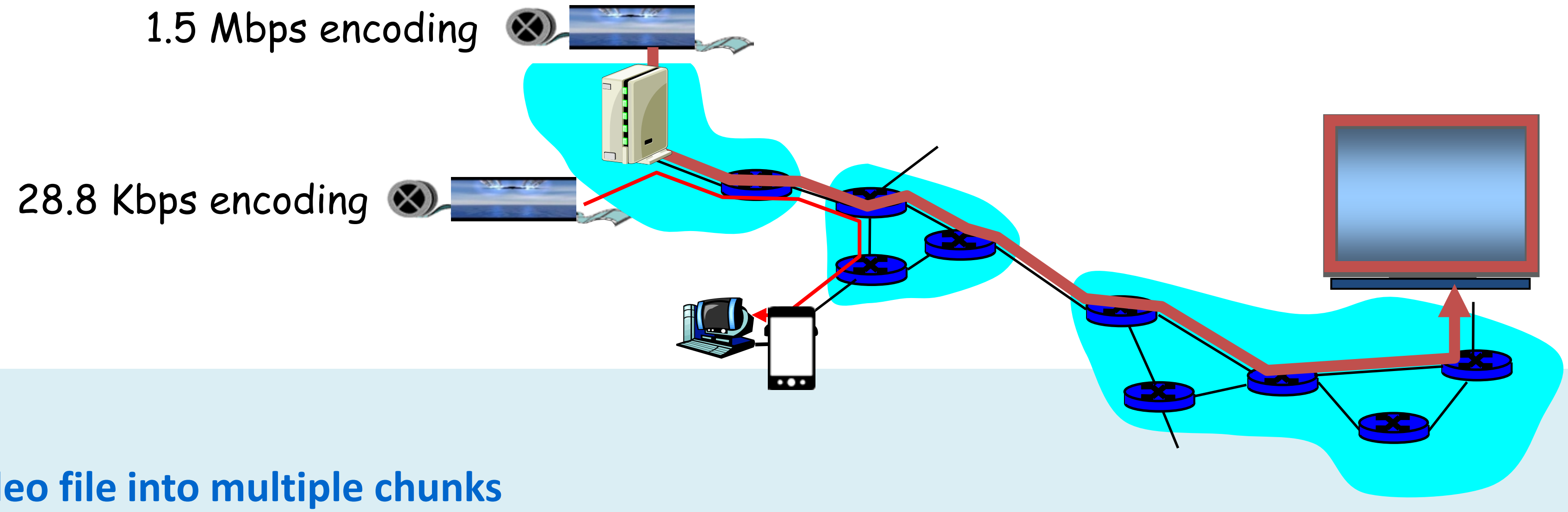
Streaming multimedia: HTTP (ie through TCP)

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

HTTP/2 (HTTP/2.0, *RFC 7540*)

- Derived from SPDY (introduced by Google)
- Does not require changes to existing web apps
- New: how data is framed and transported, e.g.:
  - header compression + websites can "minify" resources e.g. images, scripts
  - server can "push" content, i.e. respond with more data than the client requested
  - prioritization of requests, multiplexing

Some criticism

- violates the protocol layering principle, e.g. duplicates flow control (transport layer issue)
- overwhelming complexity *[P.H. Kamp, ACM queue 2015]*
- Common client implementations (Firefox, Chrome, Safari, Opera, IE, Edge) only support HTTP/2 over encrypted channels (TLS)

QUIC (Quick UDP Internet Connections)

Announced publicly in 2013 *[Google], …* to improve performance of connection-oriented web apps that used TCP

- supports MUXed connections over UDP
- designed for **security** protection equivalent to TLS/SSL
- bandwidth estimation to avoid congestion (congestion avoidance into the application space)

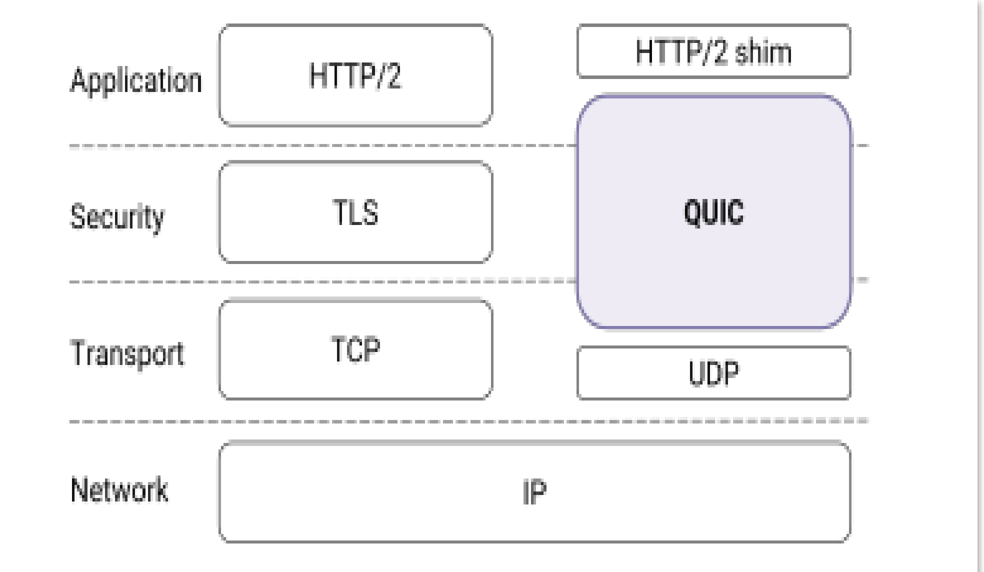


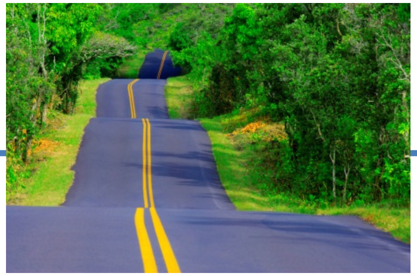
Figure 1: QUIC in the traditional HTTPS stack.

Langley et-al; Quic; ACM SIGCOMM '17, DOI: https://doi.org/10.1145/3098822.3098842

- *2015: Internet Draft* of a specification for QUIC submitted to the IETF for standardization
- *QUIC working group*: multipath support & forward error correction (FEC) as next steps

See also: Kohler et-al DCCP:. *SIGCOMM Comp. Commun. 2006,* DOI=10.1145/1151659.1159918 http://doi.acm.org/10.1145/1151659.1159918

Roadmap

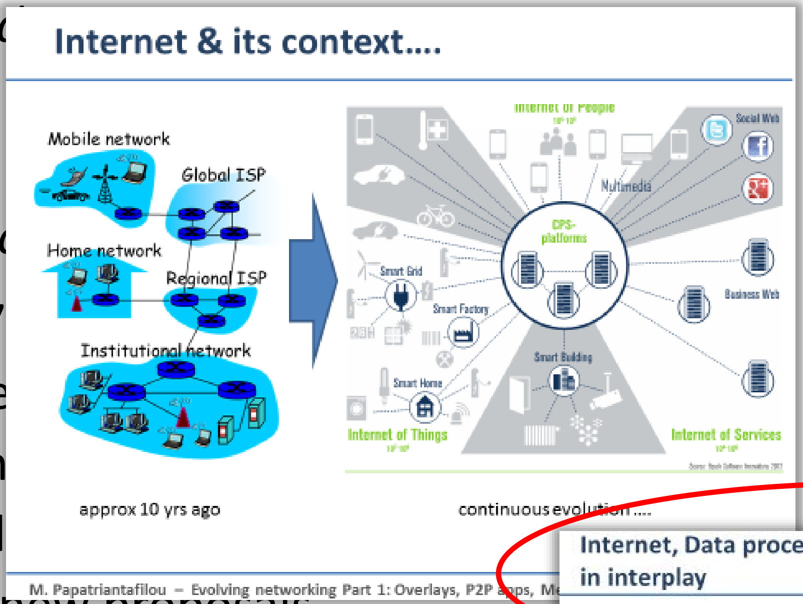


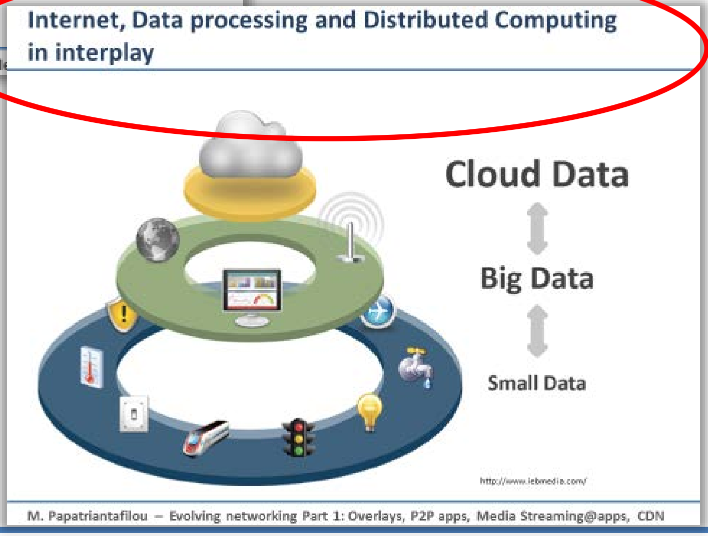
P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find*
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch*

Media Streaming Ch 9.1-9.3 & 2.6 7

- Application classes, challenge
- Today's applications represen
  - Recovery from jitter and l
  - Streaming protocols and new proposals
  - **CDN: overlays infrastructure for content delivery**





Content distribution networks (CDNs)

- Scalability big problem to stream large files from single origin
-

ush

s, delay) of

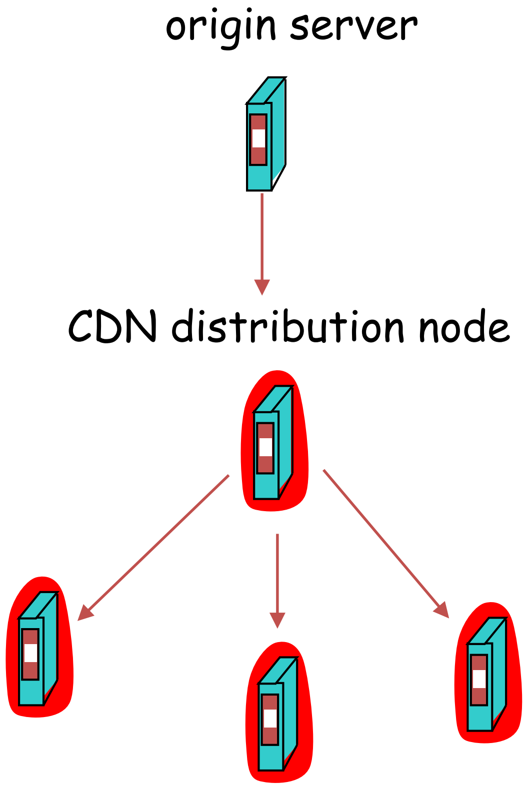
ny access

clusters in

e with

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

KanKan does use a mixed p2p &CDN approach

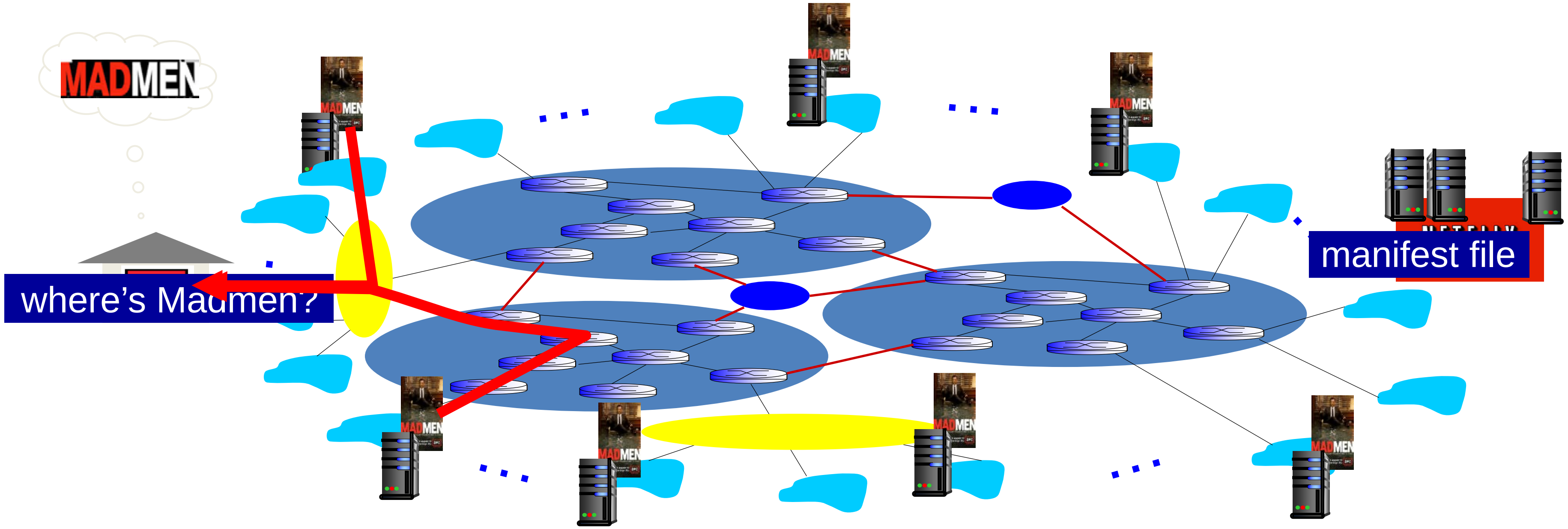


Video link: http://vimeo.com/26469929

Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen

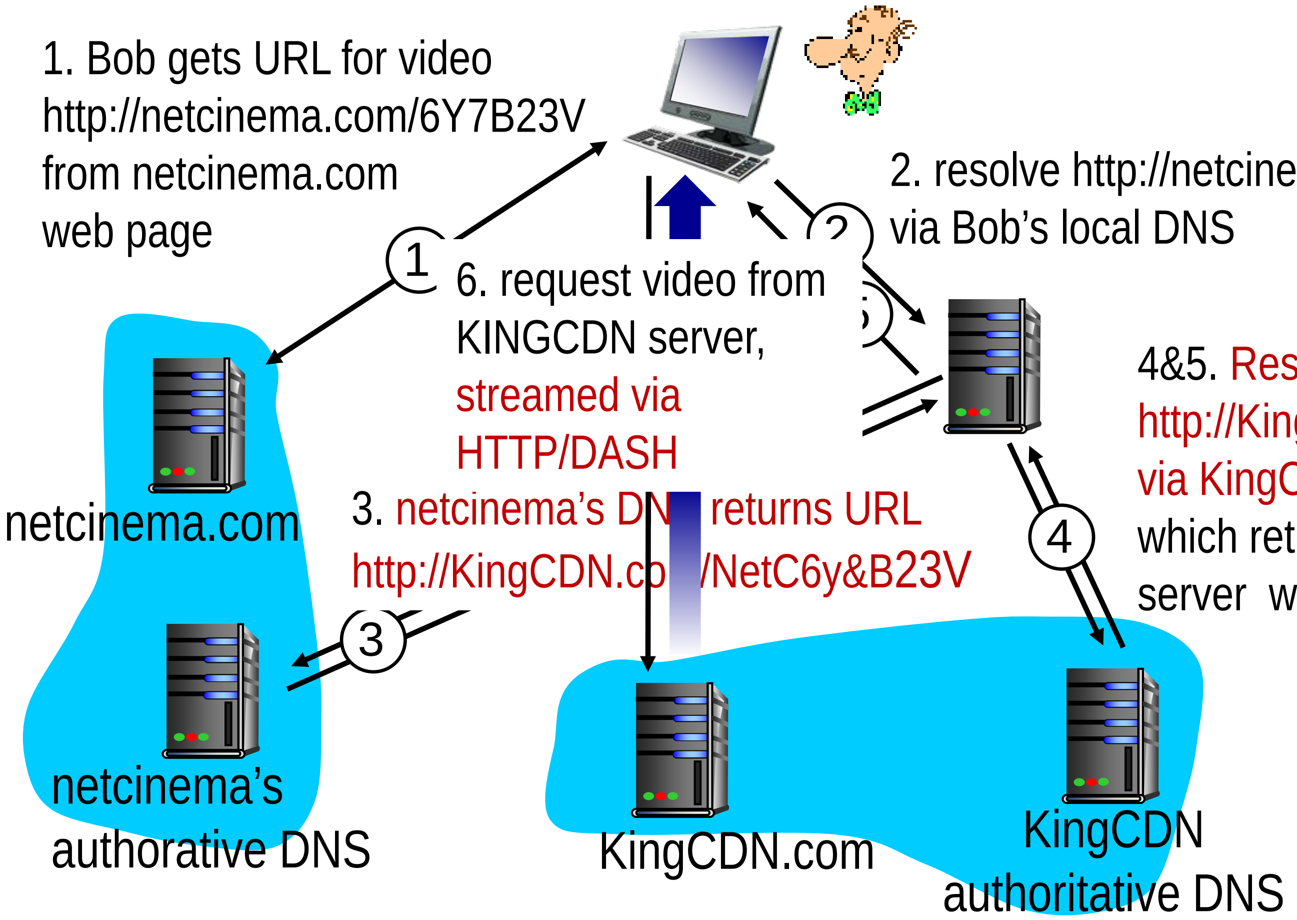
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy *if network path congested*



CDN: "simple" content access scenario

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

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



Eg Netflix' CDN approach

youtube.com/watch?v=LkLLpYdDINA

youtube.com/watch?v=tbqcsHg-Q_o

Up to this point summary Internet Multimedia

- use UDP to avoid TCP congestion control (delays) for time-sensitive traffic; or multiple TCP connections (DASH + new http/2, ….)
  - 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: could this be simpler with Network (layer) support?

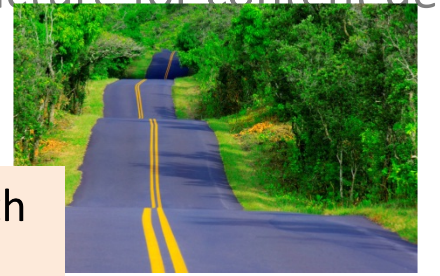
Roadmap

P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)

- Collaborate/form-*overlays* to *find* content:
  - Unstructured overlays
  - Structured Overlays/DHT
- Collaborate/form-*overlays to fetch* content

Media Streaming Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

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



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

Reading instructions and pointers for further study

P2P apps and overlays for info sharing Ch: 2.5 7e (2.6 6/e)
Media Streaming & support from applications Ch 9.1-9.3 & 2.6 7e (7.1-7.3 6/e)

- Upkar Varshney, Andy Snow, Matt McGivern, and Christi Howard. 2002.
  DOI=10.1145/502269.502271
- Jussi Kangasharju, James Roberts, Keith W. Ross, Object replication strat
  Volume 25, Issue 4, 1 March 2002, Pages 376-383, ISSN 0140-3664, http
- K.L Johnson, J.F Carr, M.S Day, M.F Kaashoek, The measured performanc
  Volume 24, Issue 2, 1 February 2001, Pages 202-206, ISSN 0140-3664, ht
- Eddie Kohler, Mark Handley, and Sally Floyd. 2006. Designing DCCP: Con
  Rev. 36, 4 (August 2006), 27-38. DOI=10.1145/1151659.1159918 http://
- *Langley et-al; Quic; ACM SIGCOMM '17, DOI: https://doi.org/10.1145/3*

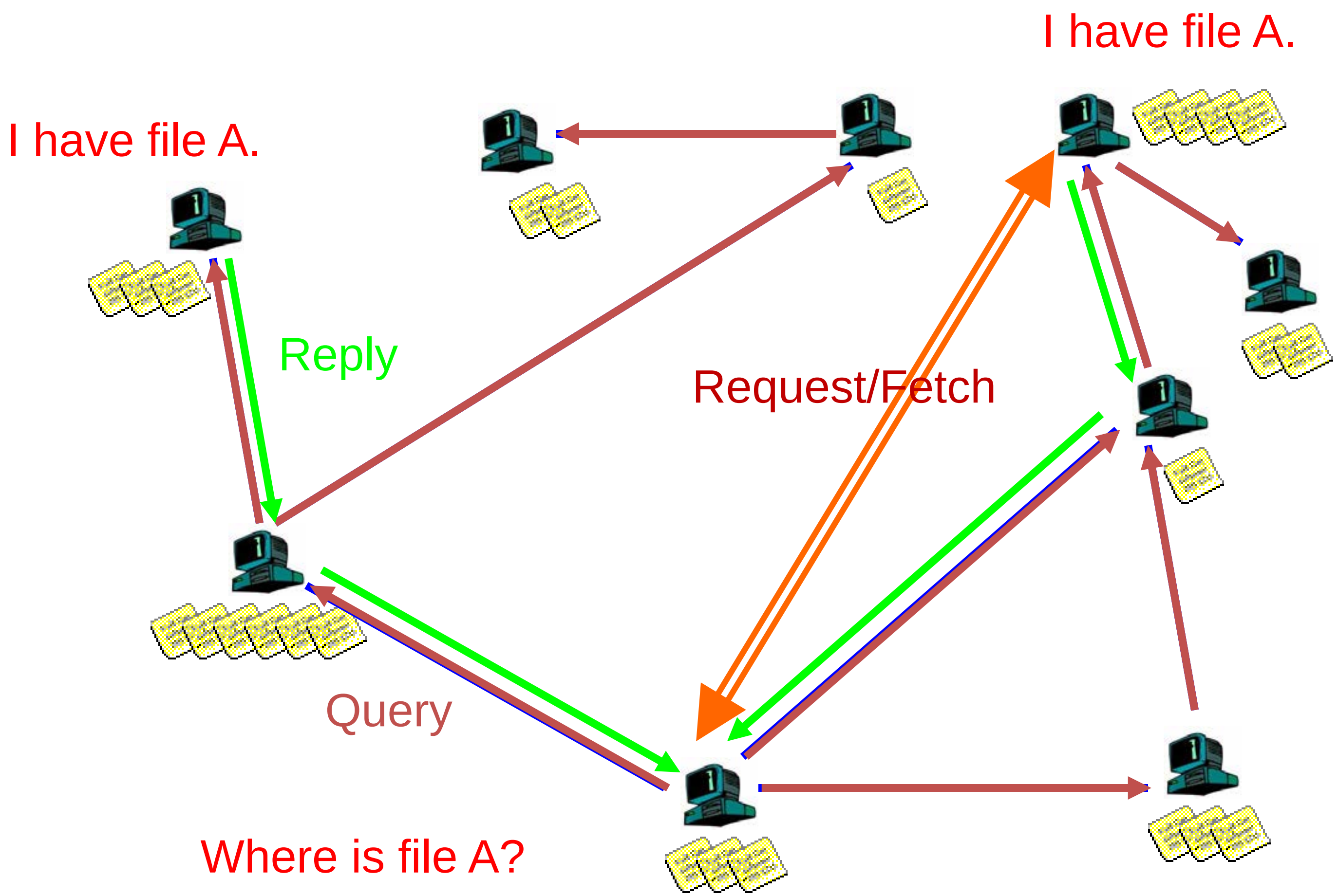
- **Applications in p2p sharing, eg dissemination and media streaming**
  – J. Mundinger, R. R. Weber and G. Weiss. Optimal Scheduling of Pee
  2, 2008. [**arXiv**] [**JoS**]
  – Christos Gkantsidis and Pablo Rodriguez, Network Coding for Large
  (Avalanche swarming: combining p2p + media streaming)

Review questions

1. R2.21, R2.24, R9.5, R9.7, R9.8, R9.9, R9.11 (7e)
2. n for introducing them and outline
3. ɔ http://netcinema.com/6Y7B23V;
DN.com/NetC6y&B23V. Describe the
) that make it possible for Bob to

Extra notes / for further study

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**)

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)

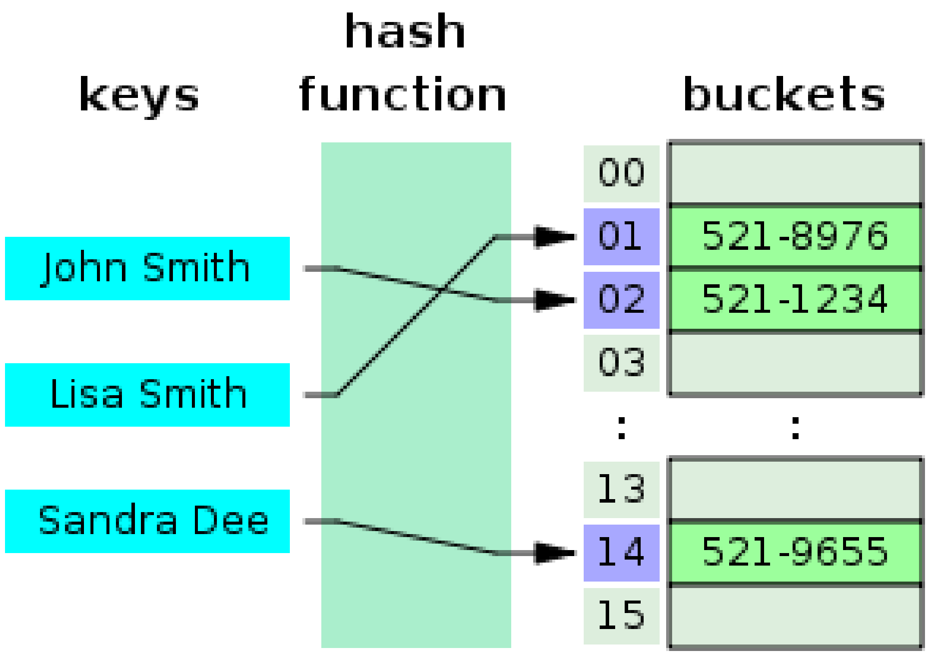


figure source: wikipedia; "Hash table 3 1 1 0 1 0 0 SP" by Jorge Stolfi - Own work. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg#/media/File:Hash_table_3_1_1_0_1_0_0_SP.svg

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"

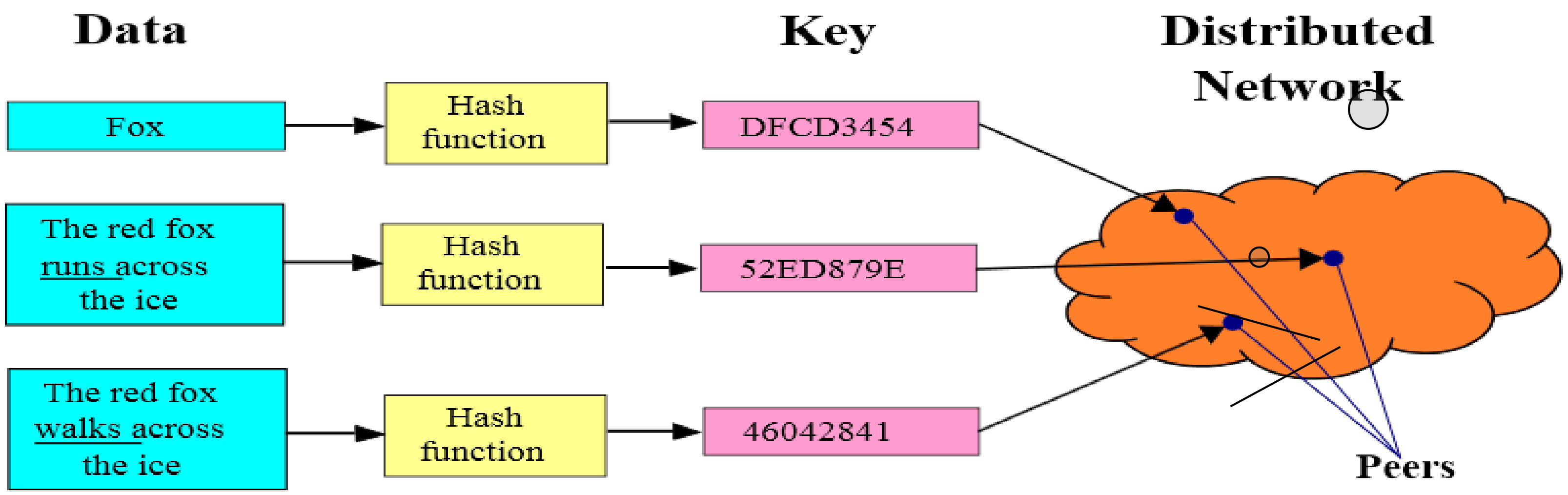
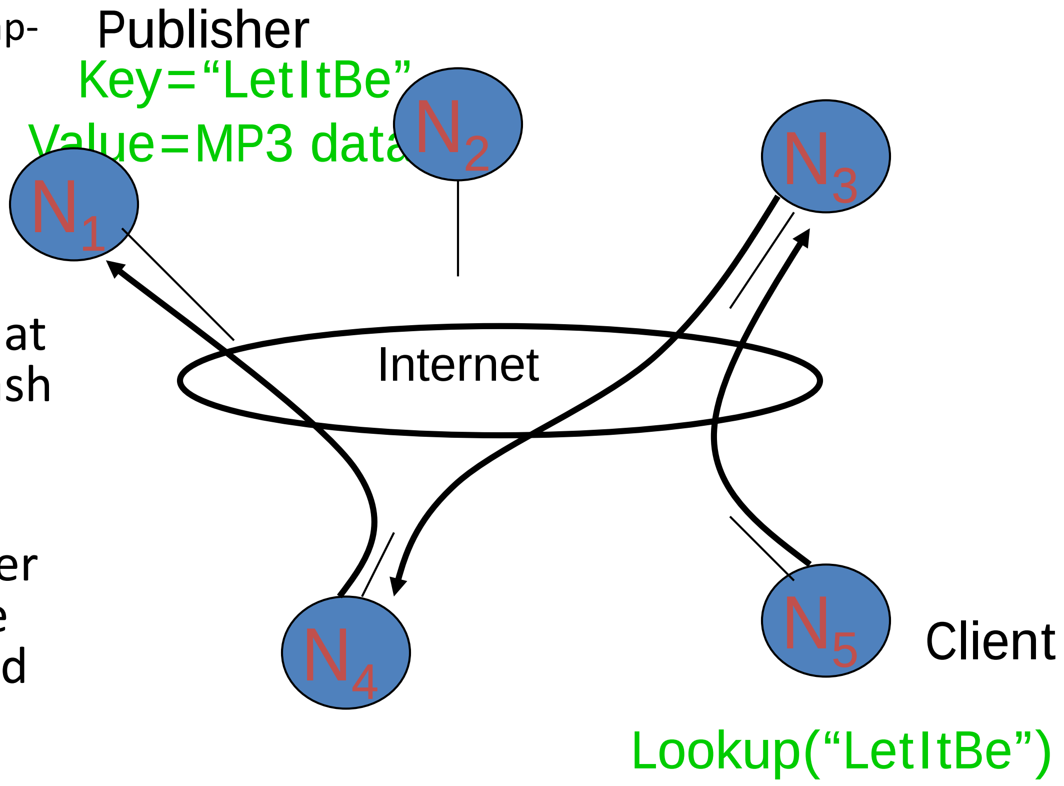


figure source: wikipedia

Distributed-Hash-Table-Based p2p sharing

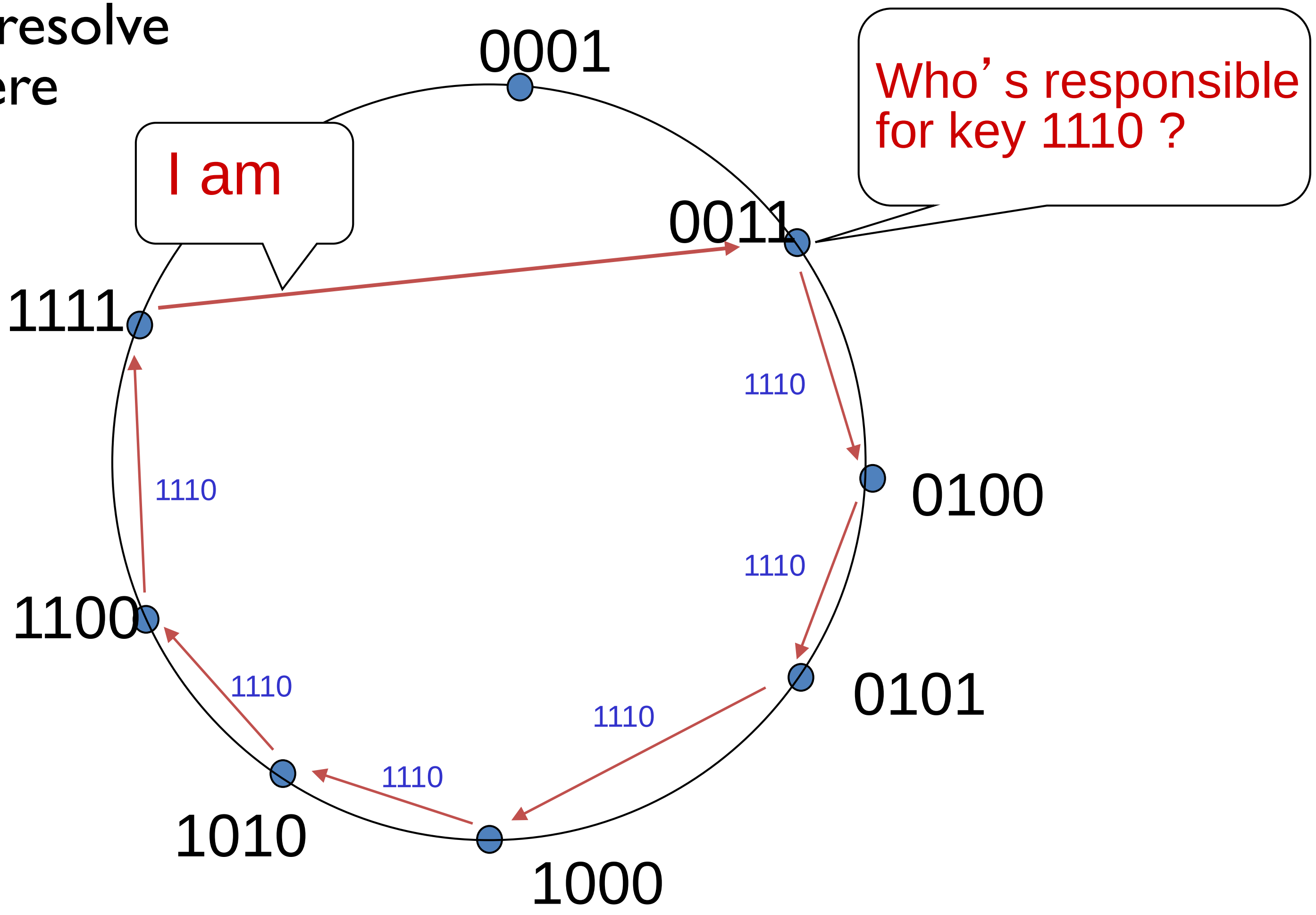
- **Join**:
  - get connected in the overlay through info from bootstrap-node & using the specific DHT algorithm (eg Chord)
  - Start maintaining of files that you are responsible for (following the hash function)
  - NOTE: upon leaving DHT needs restructuring!
- **Publish**: tell which files you have,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



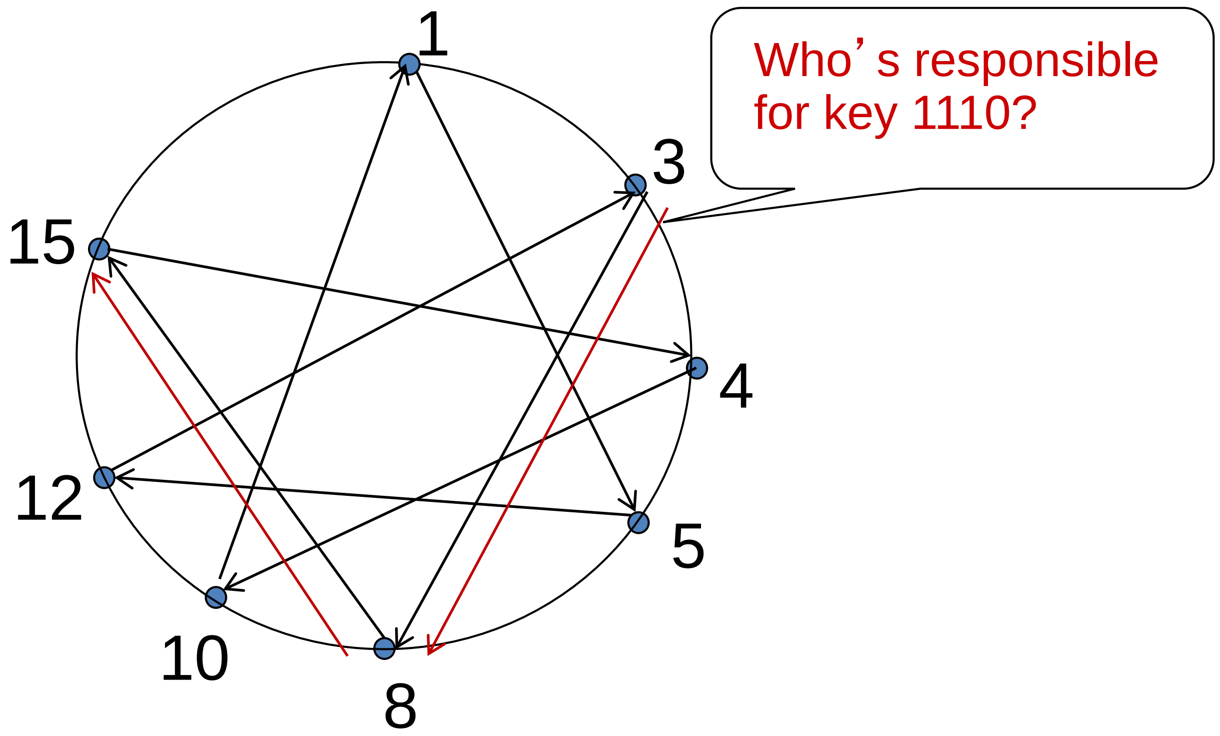
Challenges *[cf related literature@end of notes]:*
- Keep the hop count (asking chain) small
- Keep the routing tables (#neighbours) "right size"
- Stay robust despite rapid changes in membership (churn)

e.g. Circular DHT (I)

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

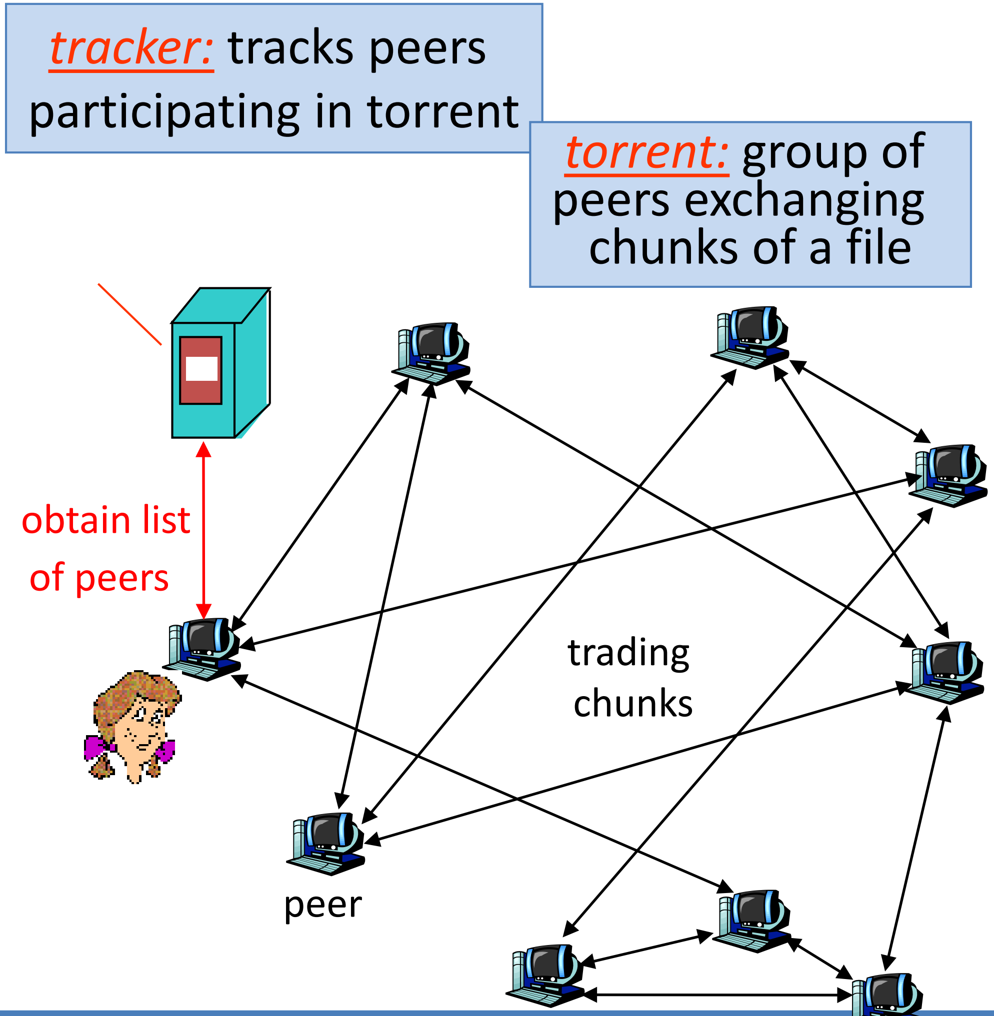


Circular DHT with shortcuts



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

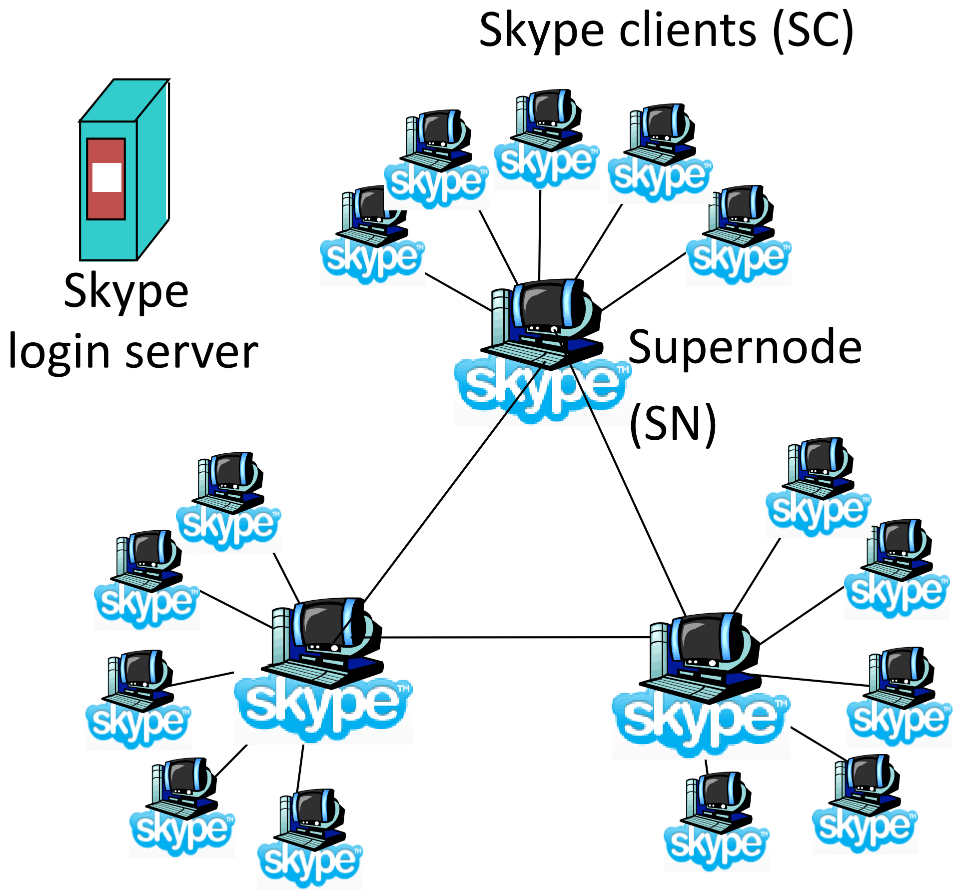
Swarming: File distribution



- 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

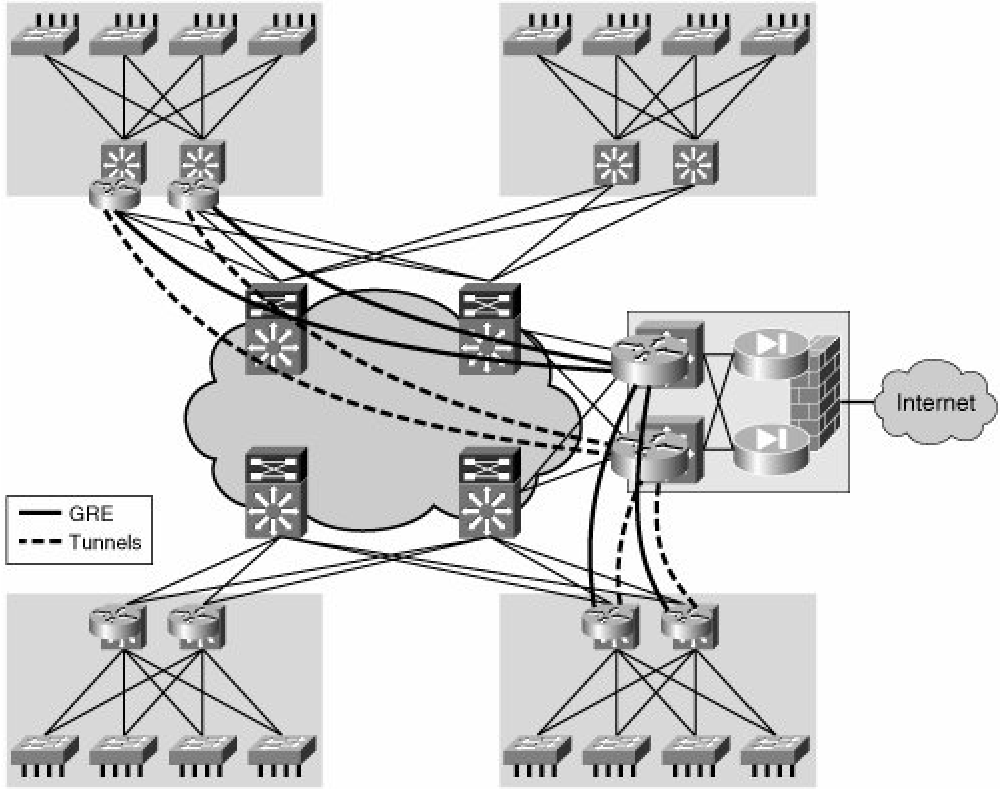
e.g. P2P & streaming Case study: Skype

- 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



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.