Network Applications – Part I

What are network applications? • Classes of network applications • Requirements of network applications • Peer-to-peer applications • Multimedia applications

• Real time versus non real time applications

What are Network Applications?

- Program with distributed components
  - ***Connect people*** – Telephony and instant messaging applications
  - ***Data access*** – Web applications
  - ***Remote processing*** – Web applets with server-side computing
  - ***Remote resource control*** – Remote sensing and control (mobile robots)
  - ***Distributed processing*** – High performance Grid computing

Classes of Network Applications

- Network apps can be grouped using following parameters
- ***Symmetry:***
  - Client-Server applications
  - Peer-to-peer applications
- ***Communicating:***
  - Dependent (communicating) component apps – distributed components are continuously passing messages among them
  - Independent (non communicating) components – pass few or no messages among them (e.g., SETI@Home)
- ***Granularity:***
  - Fine-grained applications – heavily communicating applications
  - Coarse-grained applications – loosely communicating applications (low inter-communication rate)
- ***Time sensitivity:***
  - Hard real-time – message communications among components have “hard” deadlines; missing deadlines can have catastrophic consequences
  - Soft real-time – missing deadlines degrades performance and reduced the user experience but does not create catastrophic consequences
  - Non real-time – no time based deadlines; want the best possible performance

Requirements of network apps

- Basic requirements (point-to-point messaging)
  - Send messages between end-points running components of the apps
  - Message transmissions subject to certain performance requirements
    - Maximum desired delay
    - Maximum variation in delay
    - Minimum bandwidth

Requirements of network...

- Advanced requirements (point-to-multipoint messaging)
  - Send messages to multiple components of an app (e.g., messaging in collaborative computing – whiteboards)
  - Quality of service and reliability in point-to-multipoint messaging (harder than in point-to-point messaging)

Requirements of network...

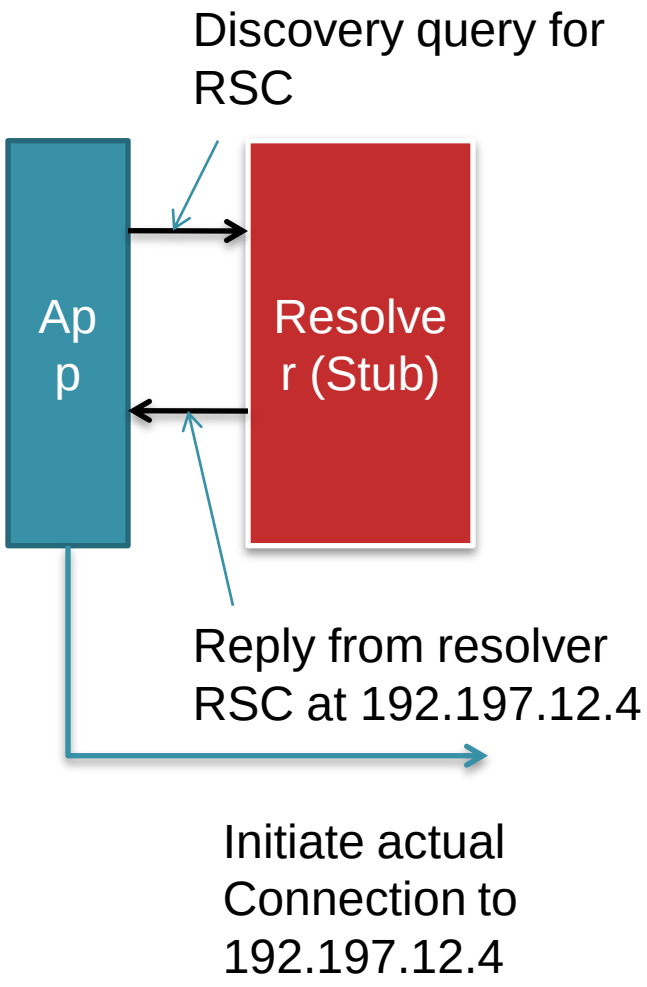
- Resource discovery requirements
  - Often one component needs to discover where another component is running to initiate a connection
- Discovery can be:
  - Name-based discovery
    - Most common – could be as simple as resolving the name to an address
    - Names are human friendly – usually hierarchical in nature
    - E.g., DNS on the Internet

Requirements of network...

- Discovery can be:
  - Attribute-based discovery
    - Give multiple parameters and ask for a resource that meets all or most of the requirements
    - Looking a particular file (given attributes such as name, version information, etc)
    - UDDI (universal description discovery and integration) is a proposed Web-services centric discovery mechanism that is attribute capable
- Name-based discovery is easier than attribute-based discovery

Resource Discovery Process

- Consider name-based discovery
- Once the name-to-addr binding is done, app contacts the machine
- How do we incorporate “late” information (up-to-the minute network information)?



Incorporating “Late” Information

- Fail and recover
  - Fail the application if the information is not current enough
  - Let the full discovery cycle take place again
- Use ***“late” binding***
  - RSC is used in the connection
  - Network finds RSC when the connection is initiated
  - Deep packet inspection is one way of doing late binding on legacy systems!

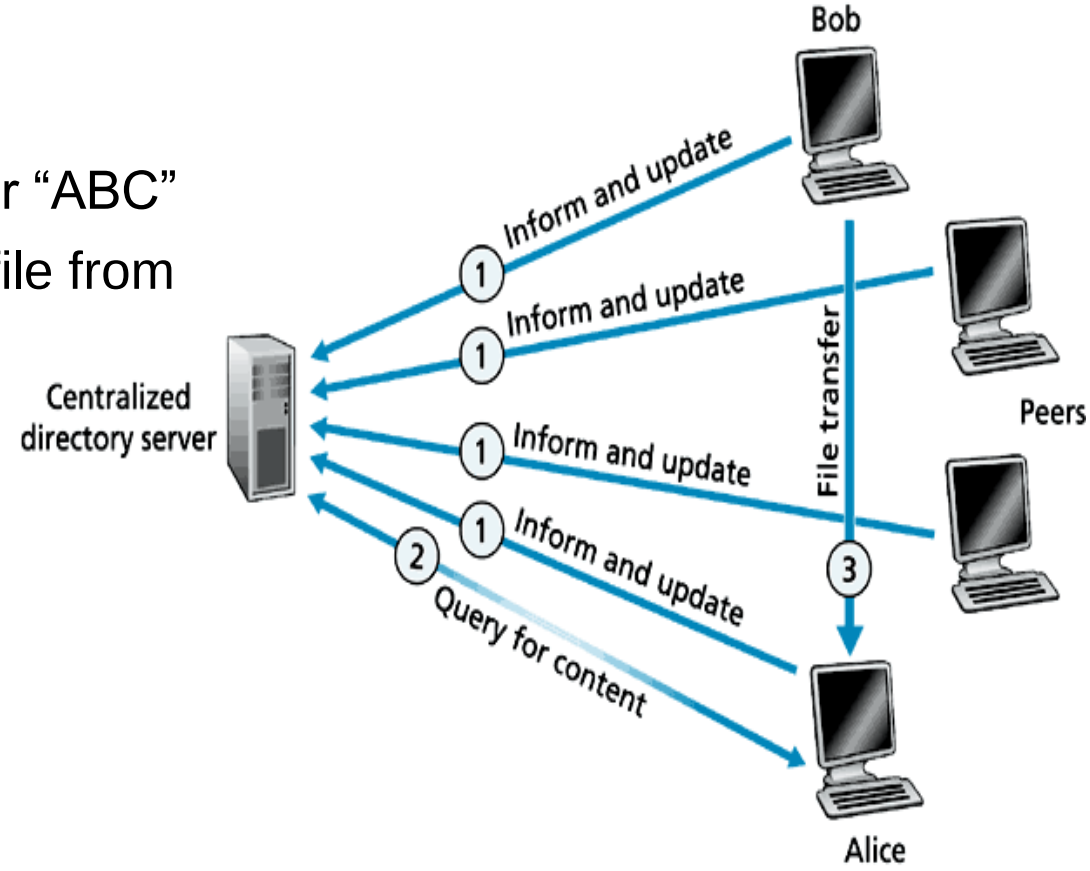
P2P file sharing

Example

- Alice runs P2P client application on her computer
- Intermittently connects to Internet; gets new IP address for each connection
- Asks for file “ABC”
- Application displays other peers that have copy of ABC
- Alice chooses one of the peers, Bob
- File is copied from Bob’s PC to Alice’s notebook: HTTP
- While Alice downloads, other users uploading from Alice
- Alice’s peer is both a Web client and a transient Web server
- All peers are servers = highly scalable!

P2P: centralized directory

- original “Napster” design
- 1) when peer connects, it informs central server:
  - IP address
  - content
- 2) Alice queries for “ABC”
- 3) Alice requests file from Bob



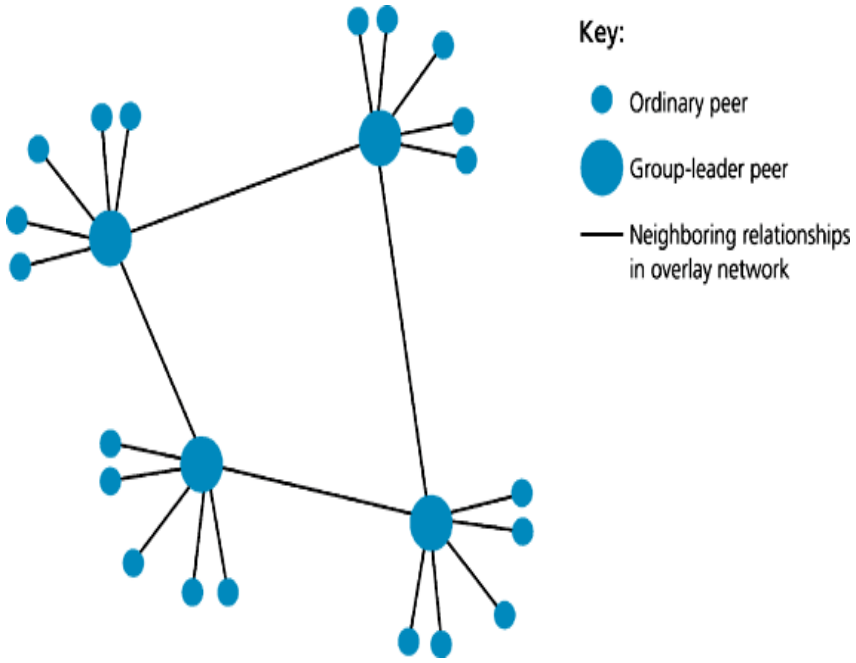
P2P: problems with centralized directory

- Single point of failure
- Performance bottleneck
- Copyright infringement

- file transfer is decentralized, but locating content is highly centralized

P2P: decentralized directory

- Each peer is either a group leader or assigned to a group leader.
- Group leader tracks the contents in all its children.
- Peer queries group leader; group leader may query other group leaders.



More on decentralized directory

overlay network

- peers are nodes
- edges between peers and their group leaders
- edges between some pairs of group leaders
- virtual neighbors

bootstrap node

- connecting peer is either assigned to a group leader or designated as leader

advantages of approach

- no centralized directory server
  - location service distributed over peers
  - more difficult to shut down

disadvantages of approach

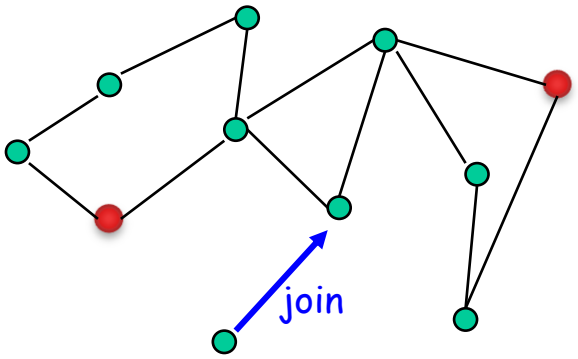
- bootstrap node needed
- group leaders can get overloaded

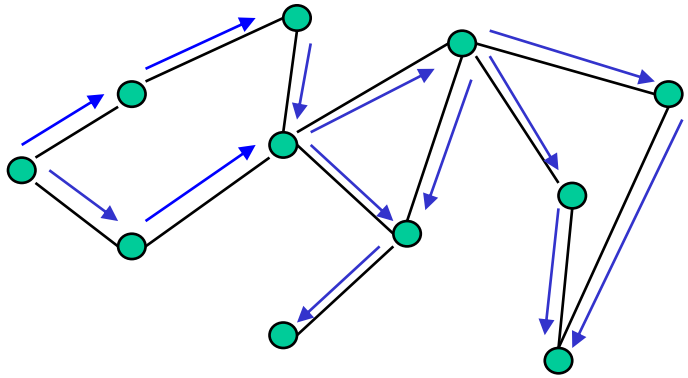
P2P: Query flooding

Gnutella type network

- no hierarchy
- use bootstrap node to learn about others
- join message

- Send query to neighbors
- Neighbors forward query
- If queried peer has object, it sends message back to querying peer





P2P: more on query flooding

Pros

- peers have similar responsibilities: no group leaders
- highly decentralized
- no peer maintains directory info

Cons

- excessive query traffic
- query radius: may not have content when present
- bootstrap node (points of failure or presence for an incoming node)
- maintenance of overlay network (need periodic message exchanges to remove dead nodes from overlay)

Structured P2P networks

- Traditional P2P file-sharing systems do not operate efficiently
  - Spend ***too many messages*** on constructing and maintaining the overlay network
  - Perform random ***global searches*** mostly by ***flooding*** the network
- One advantage of the traditional scheme is that the documents can be placed anywhere and the document will be found if at least one of the machines holding a copy is up and reachable.

Structured P2P networks

- Structured P2P networks (file sharing example) has two questions to consider:
  - How do we map objects onto nodes?
  - How do we route requests to the node that is responsible for the object?
- For the first question, simplest solution just uses hashing.

```
hash(x) -> n
```

- Where *x* is the object identifier and *n* is the node identifier onto which the object is placed.
- What properties do we need from the hash function?

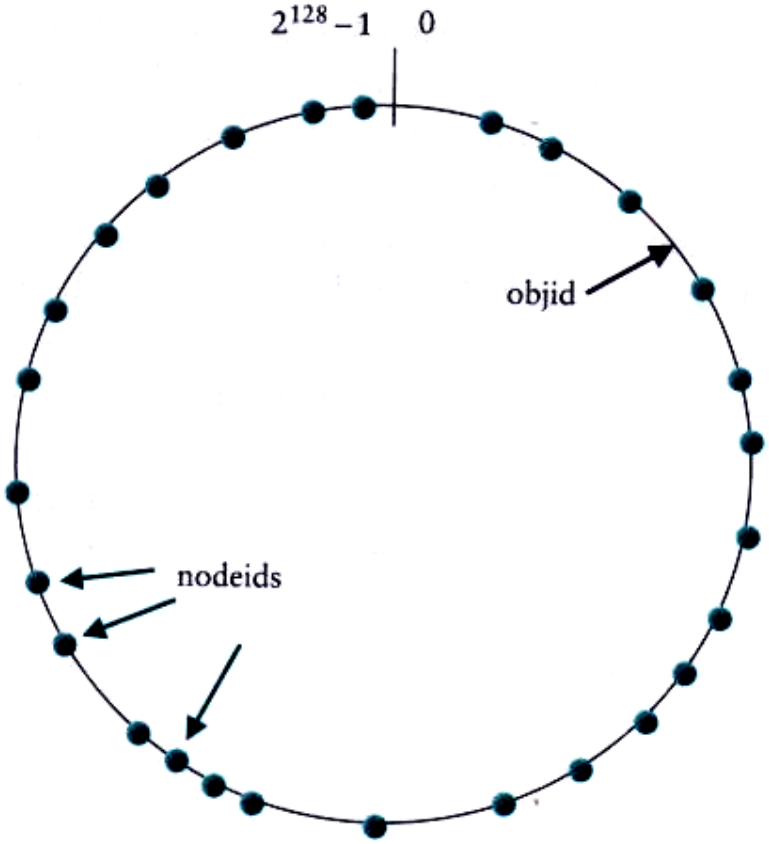
Structured P2P networks

- Problems with traditional hashing:
  - When nodes join and leave the hashing function will be affected
  - hash(x) { return x % 101 }
  - Need to know the exact number of hosts (in this example, 101)
- To address these issues, structured P2P networks use consistent hashing.

Consistent hashing

- Consistent hashing maps both objects and nodes onto a 128-bit ID space that is organized as a circle.
- Hash(object_name) -> objid
- Hash(IP_addr) -> nodeid

- Because the ID space is very large, an objid and nodeid would not (most likely) coincide.
- Select the node whose id is closest in this 128-bit space to the object id.

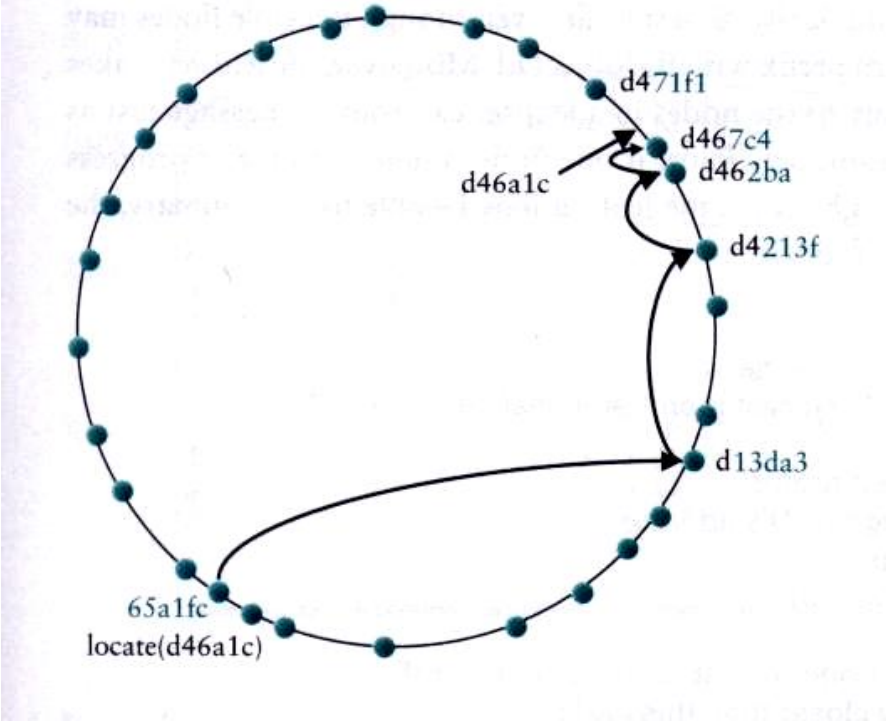


Consistent hashing…

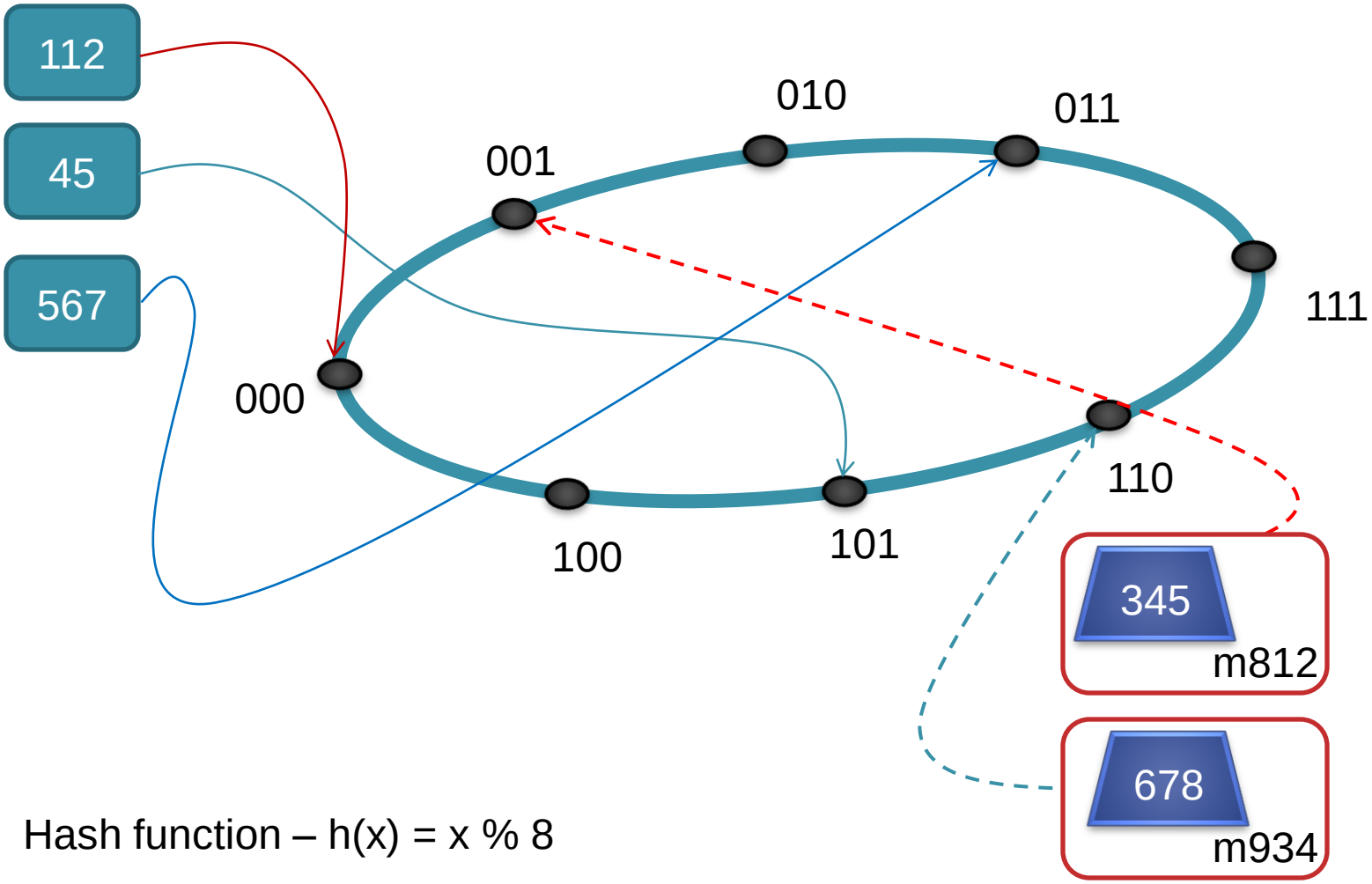
- Like ordinary hashing, distributes objects evenly across the nodes. However, unlike ordinary hashing, only a small number of objects have to move when a node (hash bucket) leaves or joins.
- How does a user who wants to access a object know which node holds the object?
  - Each node keeps a complete table of nodes IDs and associated IP addresses – search the list for the closest node ID and access the node!
  - Not practical for large networks (i.e., not scalable)
  - Another approach: route the request to the appropriate node

Distributed hash tables

- Suppose you are at node 65a1fc (hex) and trying to locate objid d46a1c
  - Your node does not share anything with the target object
  - You know a node that shared at least the prefix d – it is closer than you to this object
  - Ask this node to “locate object d46a1c” for you
  - Assuming node d13da3 knows another node with even longer prefix the message will be forwarded even further

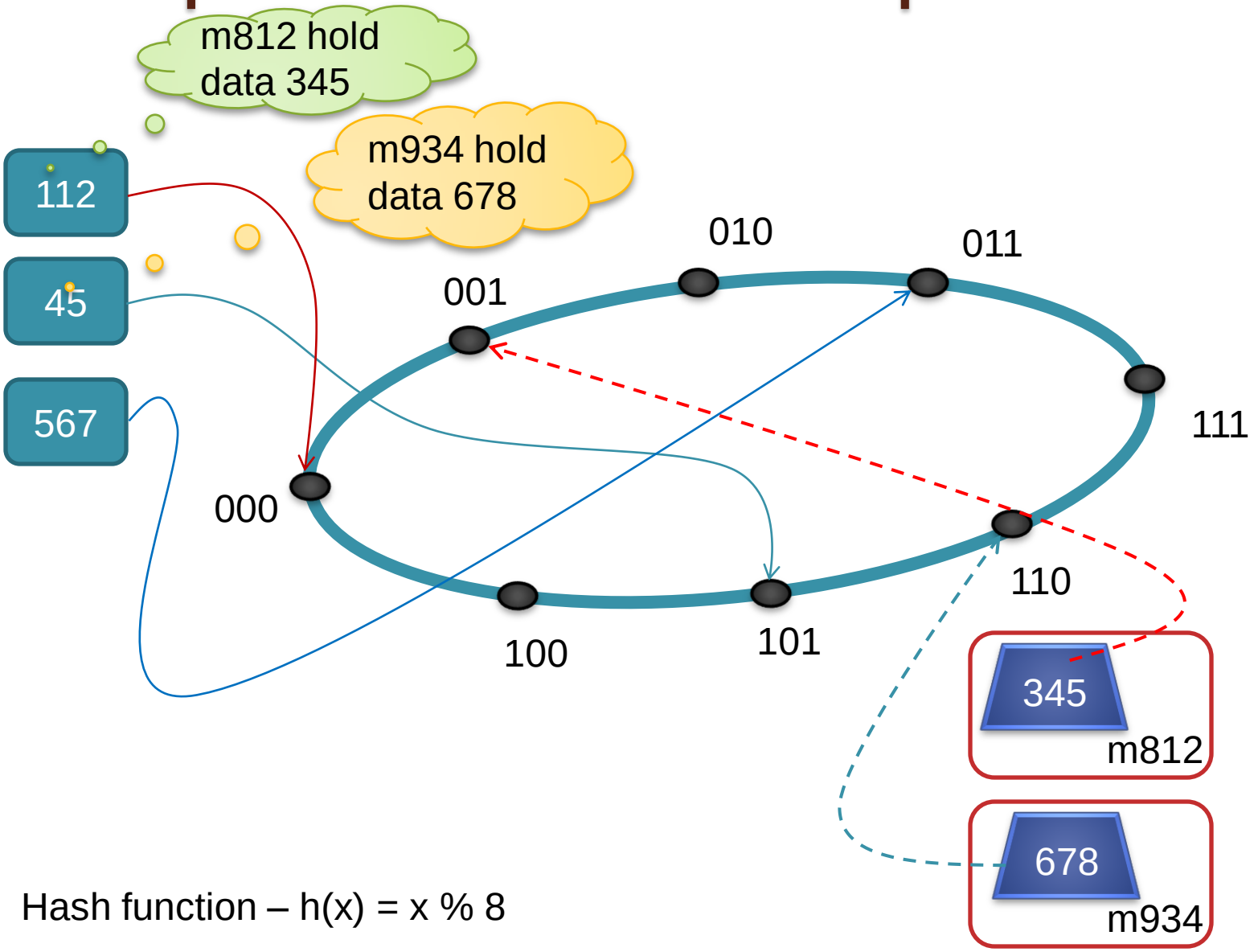


Simplified DHT Example



Hash function – h(x) = x % 8

Simplified DHT Example



Hash function – h(x) = x % 8

Distributed hash tables

- As the message moves through the ID space, the actual message moves through the Internet
- Each node maintains a route table as shown here
- Routing table is a 2-D array. Has a row for each hex digit in the ID (32 rows for a 128-bit ID)
- Entry in row *i* shares a prefix of length *i* with this node – the entry in the *j*-th column has hex value *j* at *i*+1-th position
- *x* denotes an unspecified suffix

Routing table at 65a1fc*x*

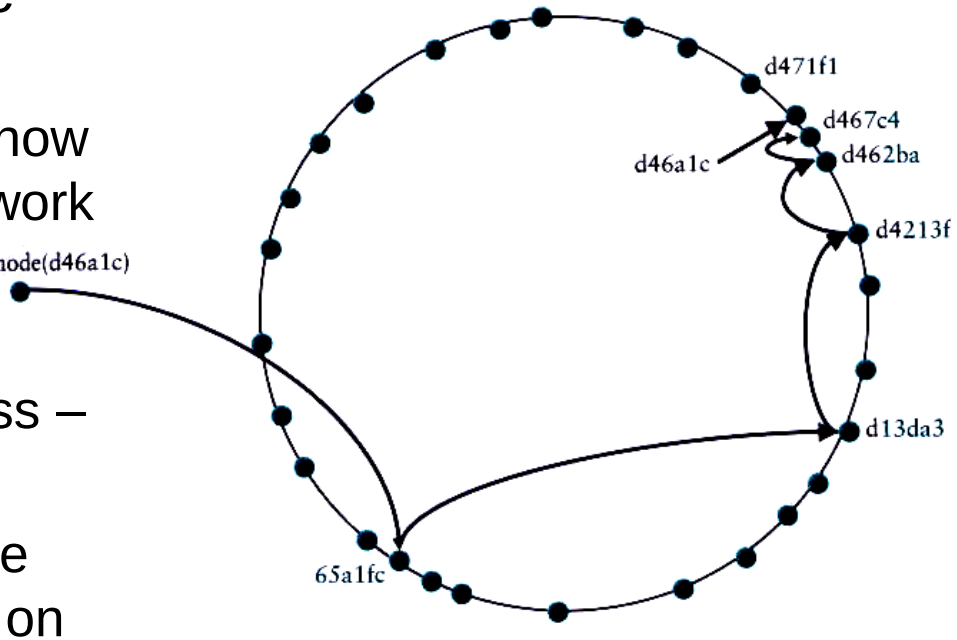
Distributed hash tables

- Each node maintains a leaf set – these are nodes that are numerically closest to the node.
- Leaf node peers with other leaf nodes within the same set of leafs. Suppose a leaf node is unable to do some operation because of some error condition that work may be offloaded onto another leaf node

```
Route(D)
  if D is within range of my leaf set
    forward to numerically closest member in leaf set
  else
    let l = length of shared prefix
    let d = value of lth digit in D's address
    if RouteTab[l,d] exists
      forward to RouteTab[l,d]
    else
      forward to known node with at least as long a prefix
      and is numerically closer than this node
```

Distributed hash tables

- Adding a node to overlay is much like routing a “locate object message”
- New node must at least know a member of the P2P network (preferable if the closest)
- Learns about other nodes through the routing process – fills out its routing able.
- Existing nodes also update their routing tables based on new arrivals



Comparison of P2P Approaches

	Unstructured P2P (Gnutella style)	Unstructured P2P (Super peer)	Structured P2P
Join	Join process might flood to from bootstrap nodes to find the best join node	Group leader for the region is the node to join with	Search/route process takes place from bootstrap node to find the join location
Search	Could lead to flooding the full network	Still flooding is possible; if group leaders have good indexing, flooding can be reduced to leader network	Search is very efficient; no flooding;
Locality	Locality is preserved on the actual network	Locality can be preserved easily	Locality only on logical space; on actual network locality can be destroyed
Fault tolerance (self organizing)	Self organizes well	Limited self organizing; group leader placement and load balancing is	Self organizes very well

Multimedia Networking Applications

Classes of MM applications:

1) Streaming stored audio and video
2) Streaming live audio and video
3) Real-time interactive audio and video

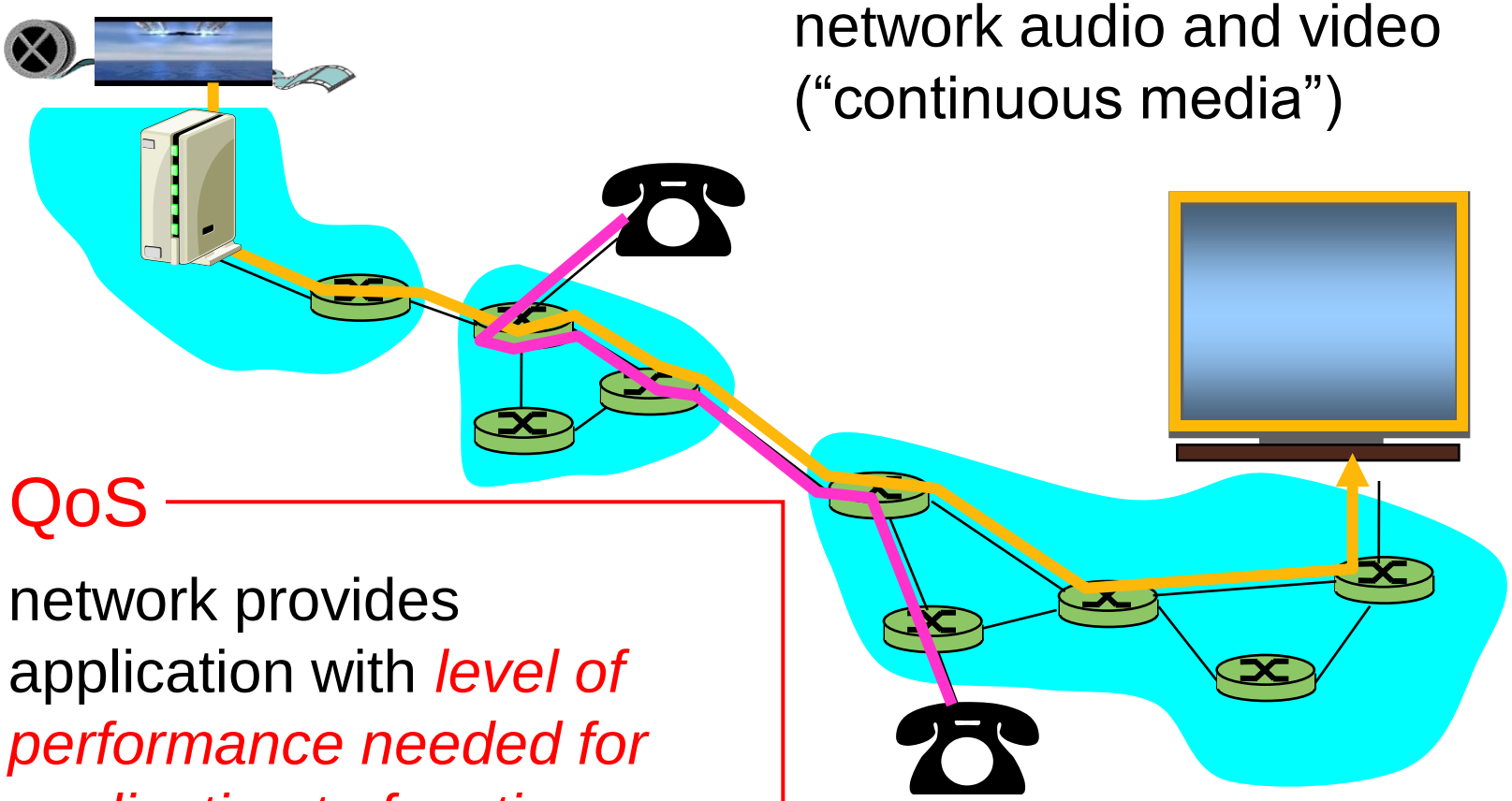
Jitter is the variability of packet delays within the same packet stream

Fundamental characteristics:

- Typically **delay sensitive**
  - end-to-end delay
  - delay jitter
- But **loss tolerant**: infrequent losses cause minor glitches
- Antithesis of data, which are loss intolerant but delay tolerant

Multimedia, Quality of Service: What is it?

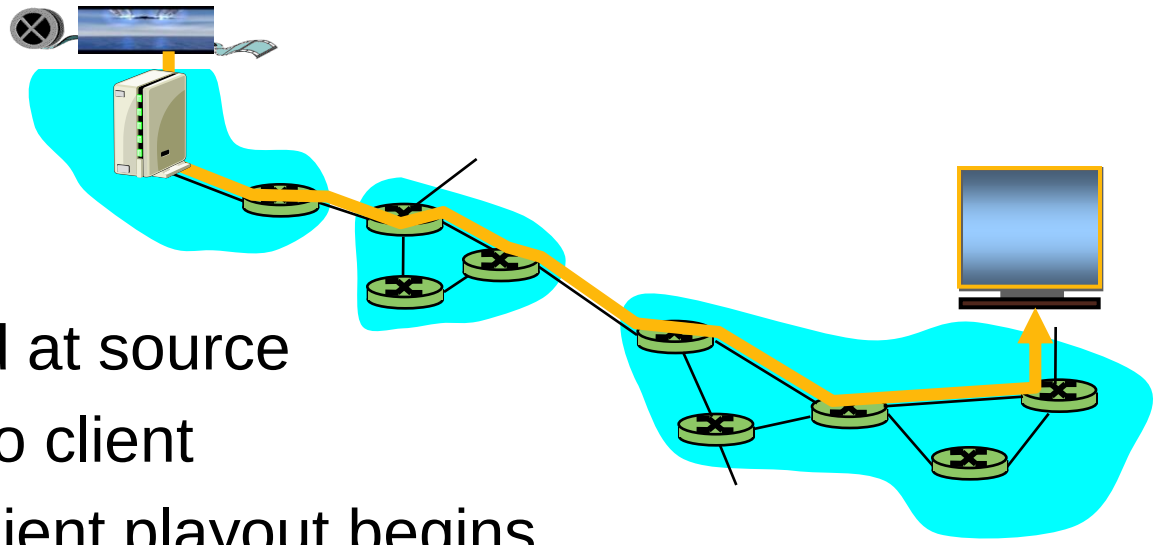
Multimedia applications:
network audio and video
(“continuous media”)



QoS

network provides application with *level of performance needed for application to function.*

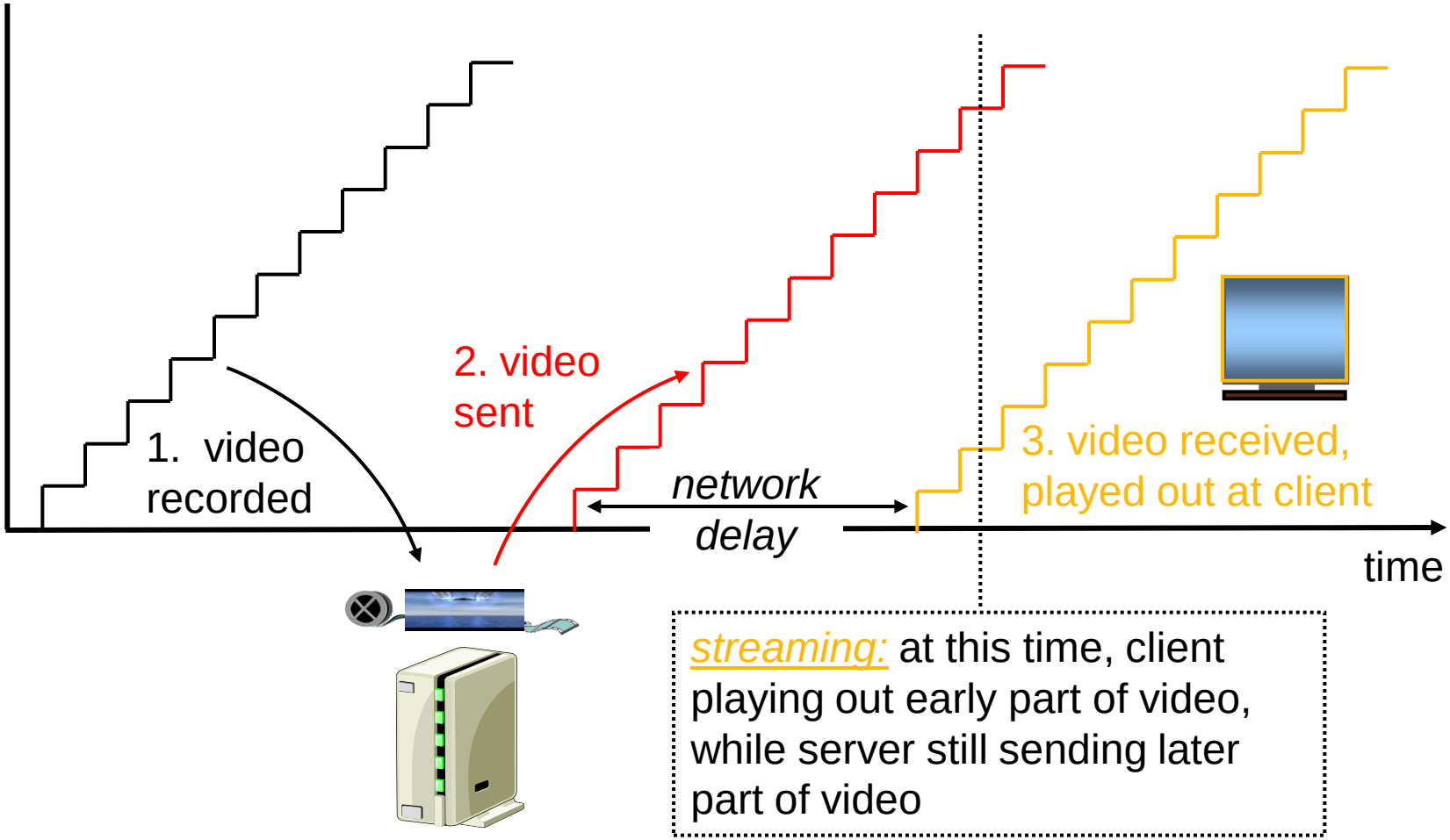
Streaming Stored Multimedia



Streaming:

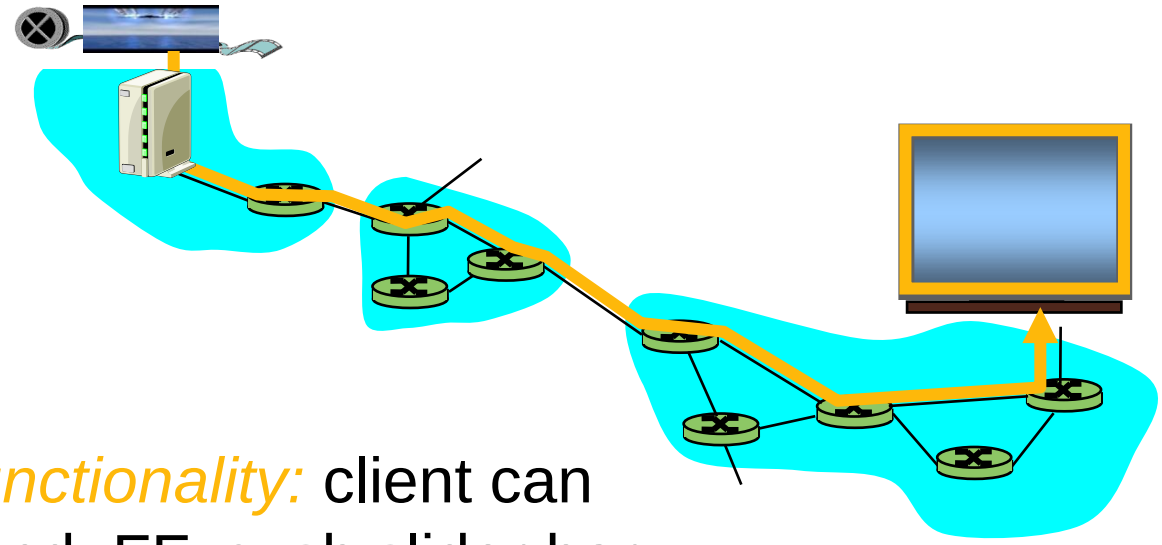
- media stored at source
- transmitted to client
- streaming: client playout begins *before* all data has arrived
  - timing constraint for still-to-be transmitted data: in time for playout

Streaming Stored Multimedia: What is it?



streaming: at this time, client playing out early part of video, while server still sending later part of video

Streaming Stored Multimedia: Interactivity



- *VCR-like functionality:* client can pause, rewind, FF, push slider bar
  - 10 sec initial delay OK
  - 1-2 sec until command effect OK
  - RTSP often used (more later)
- timing constraint for still-to-be transmitted data: in time for playout

Streaming Live Multimedia

Examples:

- Internet radio talk show
- Live sporting event

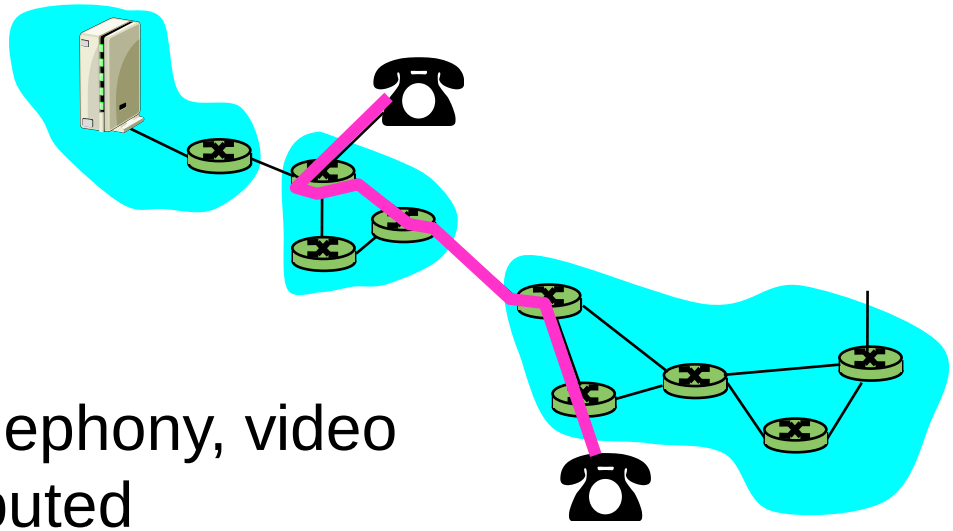
Streaming

- playback buffer
- playback can lag tens of seconds after transmission
- still have timing constraint

Interactivity

- fast forward impossible
- rewind, pause possible!

Interactive, Real-Time Multimedia



- applications: IP telephony, video conference, distributed interactive worlds
- end-end delay requirements:
  - audio: < 150 msec good, < 400 msec OK
    - includes application-level (packetization) and network delays
    - higher delays noticeable, impair interactivity
- session initialization
  - how does callee advertise its IP address, port number, encoding algorithms?

Multimedia Over Today's Internet

TCP/UDP/IP: "best-effort service"

- *no* guarantees on delay, loss

? ? ? ? ? ? ? ? ? ? ? ?

But you said multimedia apps requires QoS and level of performance to be effective!

Today's Internet multimedia applications use application-level techniques to mitigate (as best possible) effects of delay, loss

How should the Internet evolve to better support multimedia?

Integrated services philosophy:

- Fundamental changes in Internet so that apps can reserve end-to-end bandwidth
- Requires new, complex software in hosts & routers

Laissez-faire

- no major changes
- more bandwidth when needed
- content distribution, application-layer multicast
  - application layer

Differentiated services philosophy:

- Fewer changes to Internet infrastructure, yet provide 1st and 2nd class service.

What's your opinion?

A few words about audio compression

- Analog signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- Each sample quantized, ie, rounded
  - eg, 2^8=256 possible quantized values
- Each quantized value represented by bits
  - 8 bits for 256 values
- Example: 8,000 samples/sec, 256 quantized values --> 64,000 bps
- Receiver converts it back to analog signal:
  - some quality reduction

Example rates

- CD: 1.411 Mbps
- MP3: 96, 128, 160 kbps
- Internet telephony: 5.3 - 13 kbps

A few words about video compression

- Video is sequence of images displayed at constant rate
  - e.g. 24 images/sec
- Digital image is array of pixels
- Each pixel represented by bits
- Redundancy
  - spacial
  - temporal

Examples:

- MPEG 1 (CD-ROM) 1.5 Mbps
- MPEG2 (DVD) 3-6 Mbps
- MPEG4 (often used in Internet, < 1 Mbps)

Research:

- Layered (scalable) video
  - adapt layers to available bandwidth

Streaming Stored Multimedia

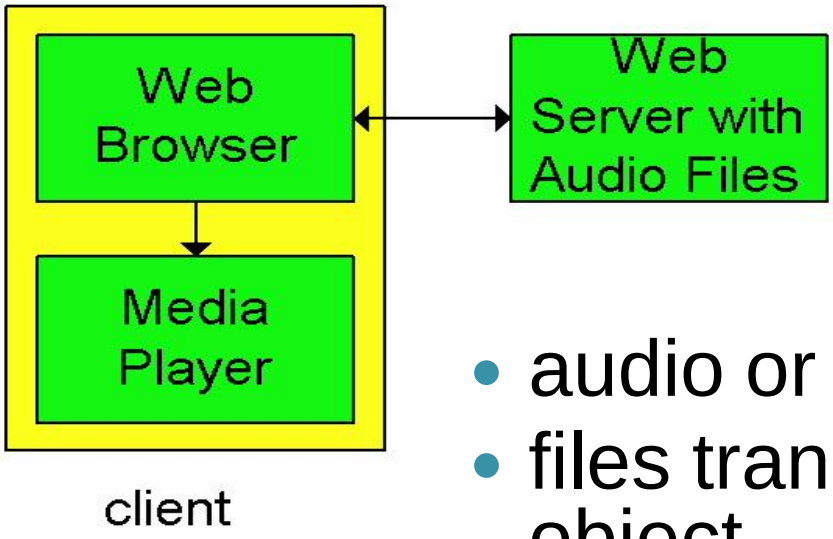
Application-level streaming techniques for making the best out of best effort service:

- client side buffering
- use of UDP versus TCP
- multiple encodings of multimedia

Media Player

- jitter removal
- decompression
- error concealment
- graphical user interface w/ controls for interactivity

Internet multimedia: simplest approach

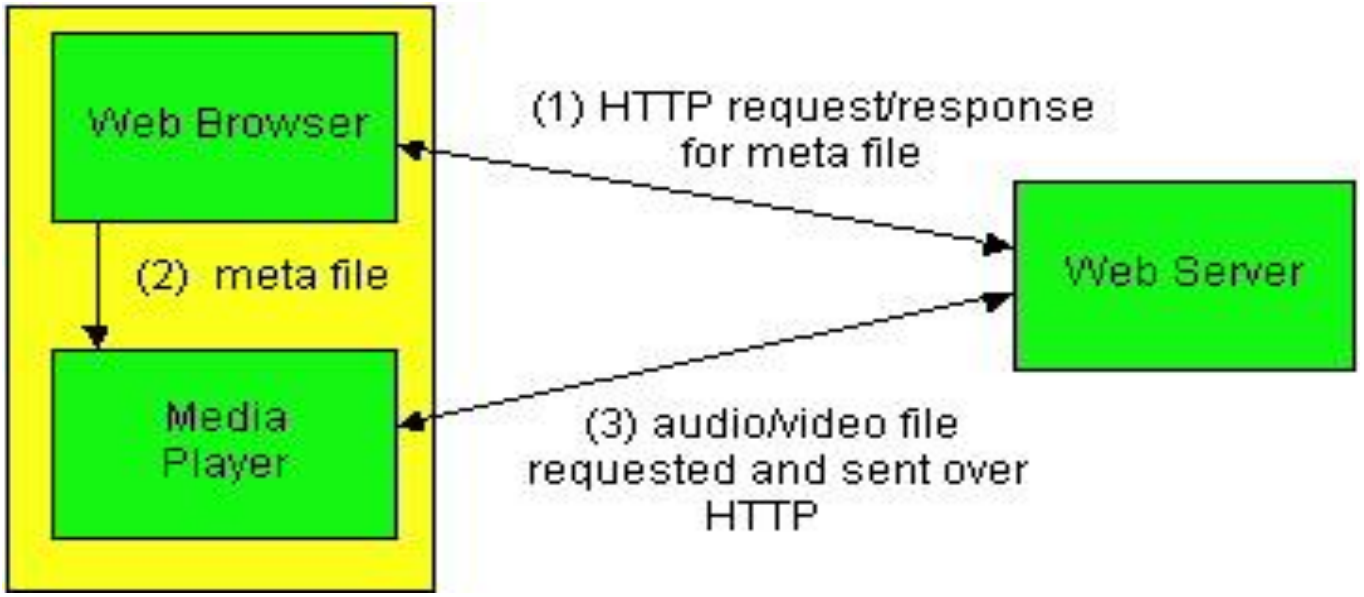


- audio or video stored in file
- files transferred as HTTP object
  - received in entirety at client
  - then passed to player

audio, video not streamed:

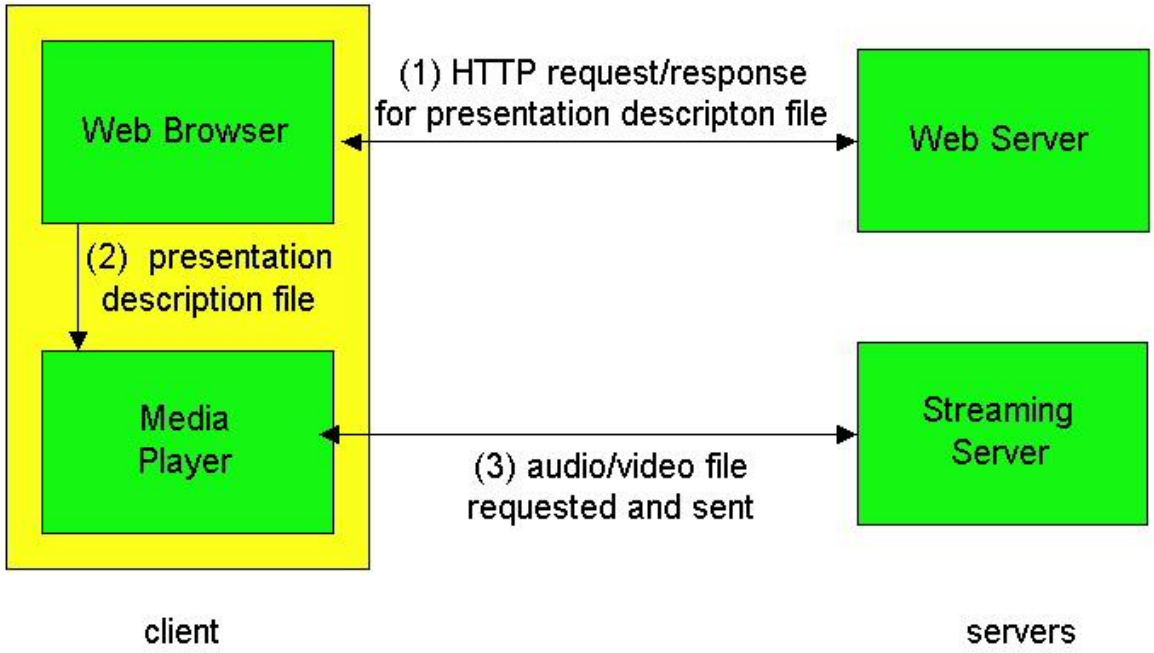
- no, “pipelining,” long delays until playout!

Internet multimedia: streaming approach



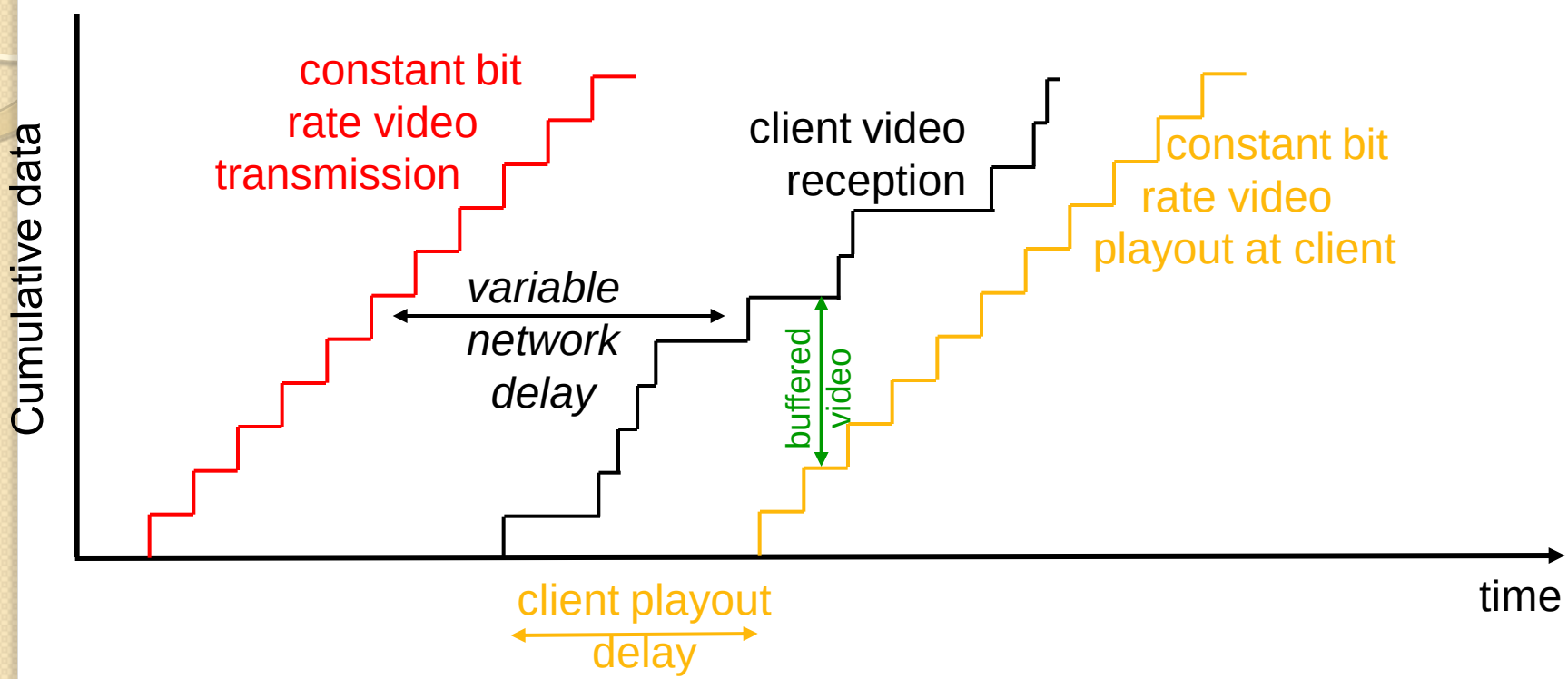
- browser GETs **metafile**
- browser launches player, passing metafile
- player contacts server
- server streams audio/video to player

Streaming from a streaming server



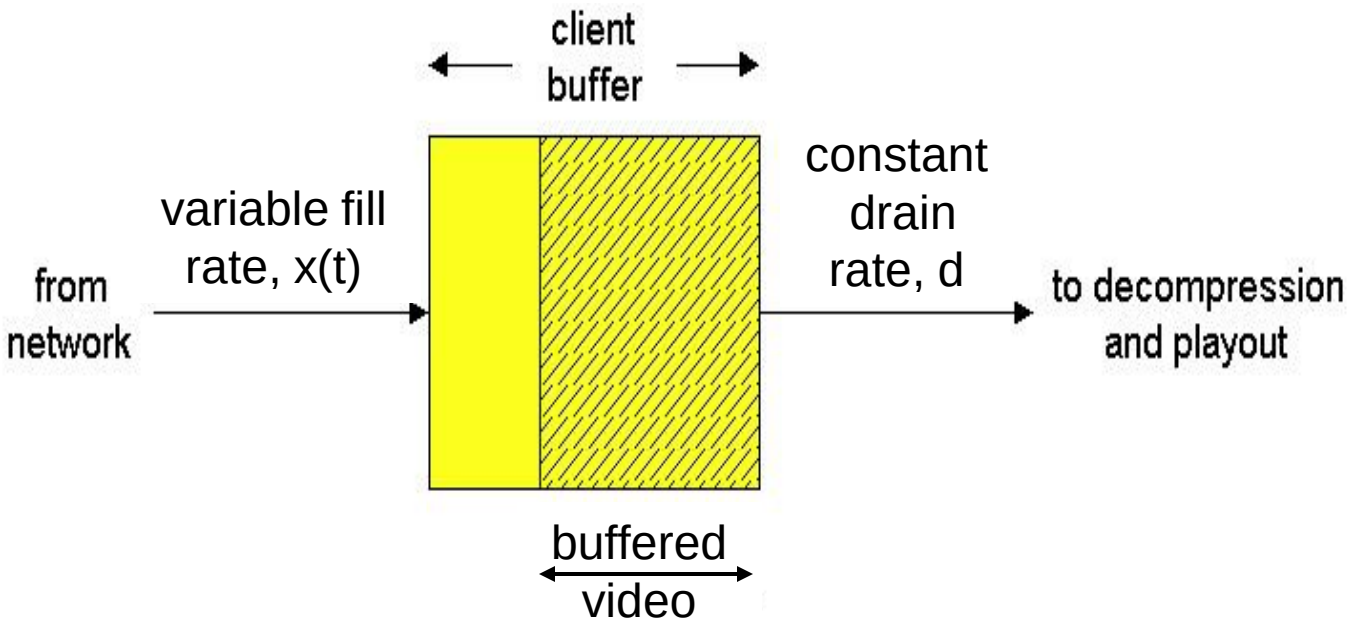
- This architecture allows for non-HTTP protocol between server and media player
- Can also use UDP instead of TCP.

Streaming Multimedia: Client Buffering



- Client-side buffering, playout delay compensate for network-added delay, delay jitter

Streaming Multimedia: Client Buffering



- Client-side buffering, playout delay compensate for network-added delay, delay jitter

Streaming Multimedia: UDP or TCP?

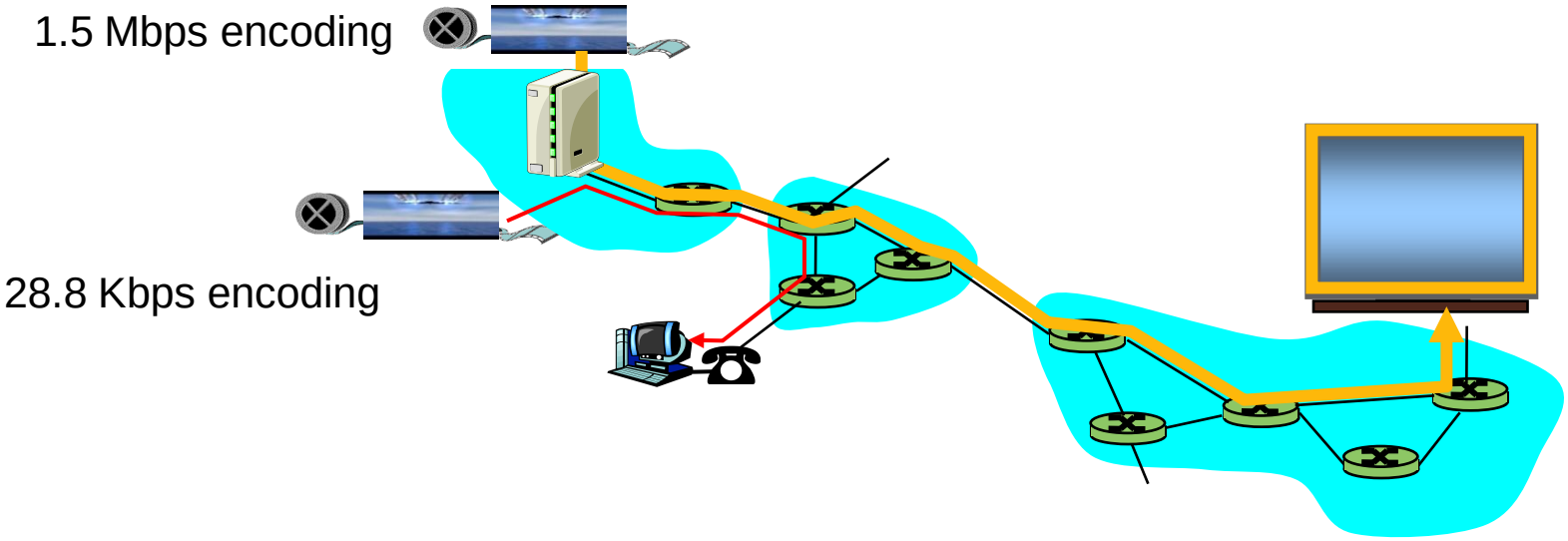
UDP

- server sends at rate appropriate for client (oblivious to network congestion !)
  - often send rate = encoding rate = constant rate
  - then, fill rate = constant rate - packet loss
- short playout delay (2-5 seconds) to compensate for network delay jitter
- error recover: time permitting

TCP

- send at maximum possible rate under TCP
- fill rate fluctuates due to TCP congestion control
- larger playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls

Streaming Multimedia: client rate(s)



Q: how to handle different client receive rate capabilities?

- 28.8 Kbps dialup
- 100Mbps Ethernet

A: server stores, transmits multiple copies of video, encoded at different rates

Real-time interactive applications

- PC-2-PC phone
  - instant messaging services are providing this
- PC-2-phone
  - Dialpad
  - Net2phone
- videoconference with Webcams

Going to now look at a PC-2-PC Internet phone example in detail

Interactive Multimedia: Internet Phone

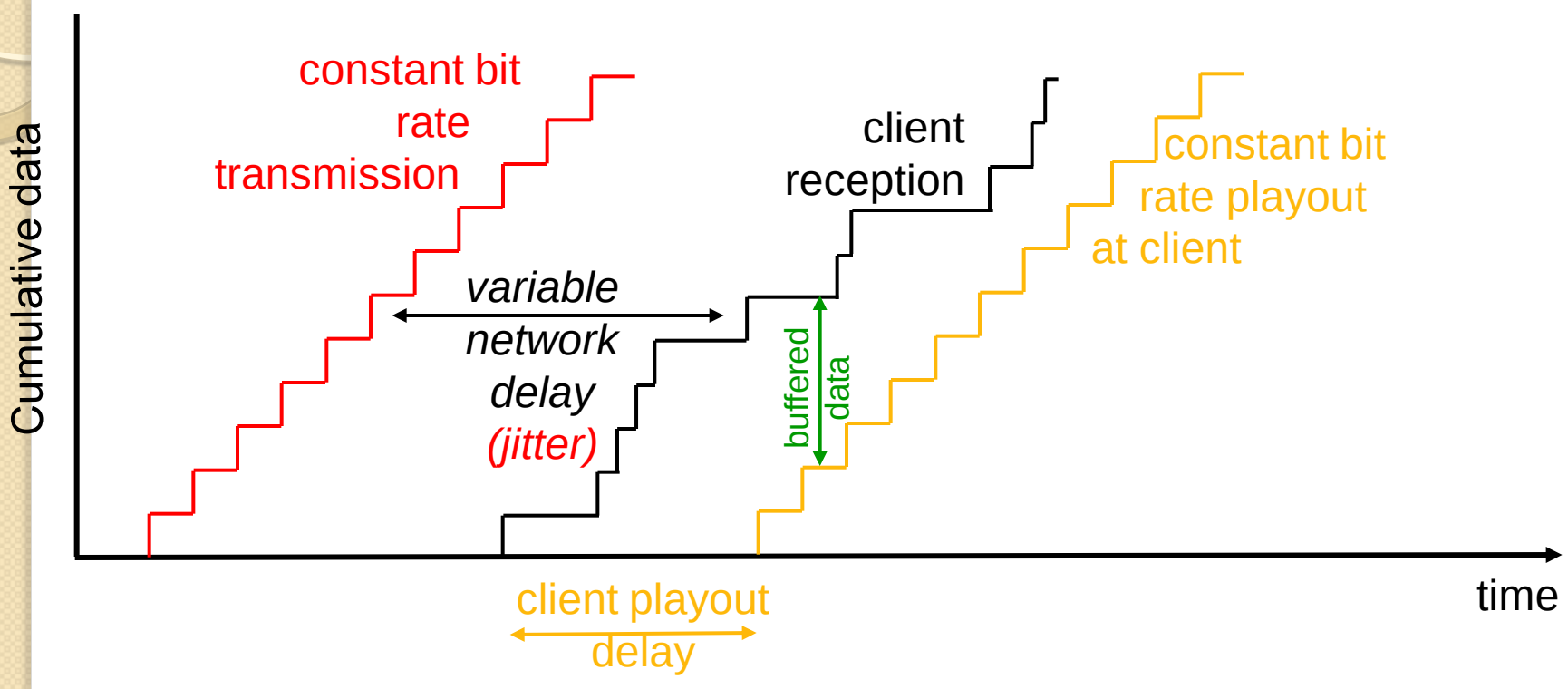
Introduce Internet Phone by way of an example

- speaker's audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
- pkts generated only during talk spurts
  - 20 msec chunks at 8 Kbytes/sec: 160 bytes data
- application-layer header added to each chunk.
- Chunk+header encapsulated into UDP segment.
- application sends UDP segment into socket every 20 msec during talkspurt.

Internet Phone: Packet Loss and Delay

- network loss: IP datagram lost due to network congestion (router buffer overflow)
- delay loss: IP datagram arrives too late for playout at receiver
  - delays: processing, queueing in network; end-system (sender, receiver) delays
  - typical maximum tolerable delay: 400 ms
- loss tolerance: depending on voice encoding, losses concealed, packet loss rates between 1% and 10% can be tolerated.

Delay Jitter



- Consider the end-to-end delays of two consecutive packets: difference can be more or less than 20 msec

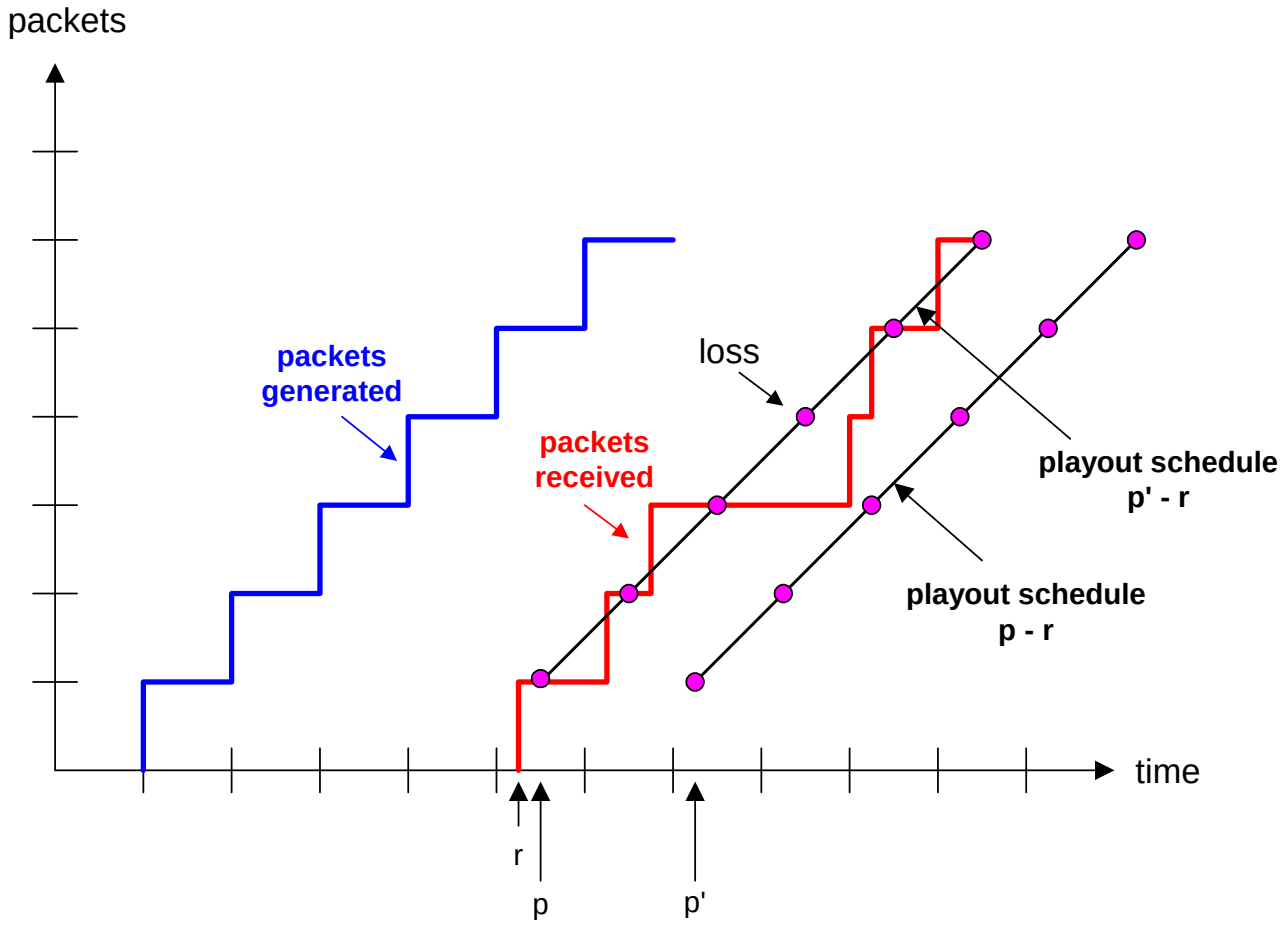
Internet Phone: Fixed Playout Delay

- Receiver attempts to playout each chunk exactly q msecs after chunk was generated.
  - chunk has time stamp t: play out chunk at t+q .
  - chunk arrives after t+q: data arrives too late for playout, data “lost”
- Tradeoff for q:
  - large q: less packet loss
  - small q: better interactive experience

- Routers could help. How?

Fixed Playout Delay

- Sender generates packets every 20 msec during talk spurt.
- First packet received at time r
- First playout schedule: begins at p
- Second playout schedule: begins at p’



SIP

- Session Initiation Protocol
- Comes from IETF

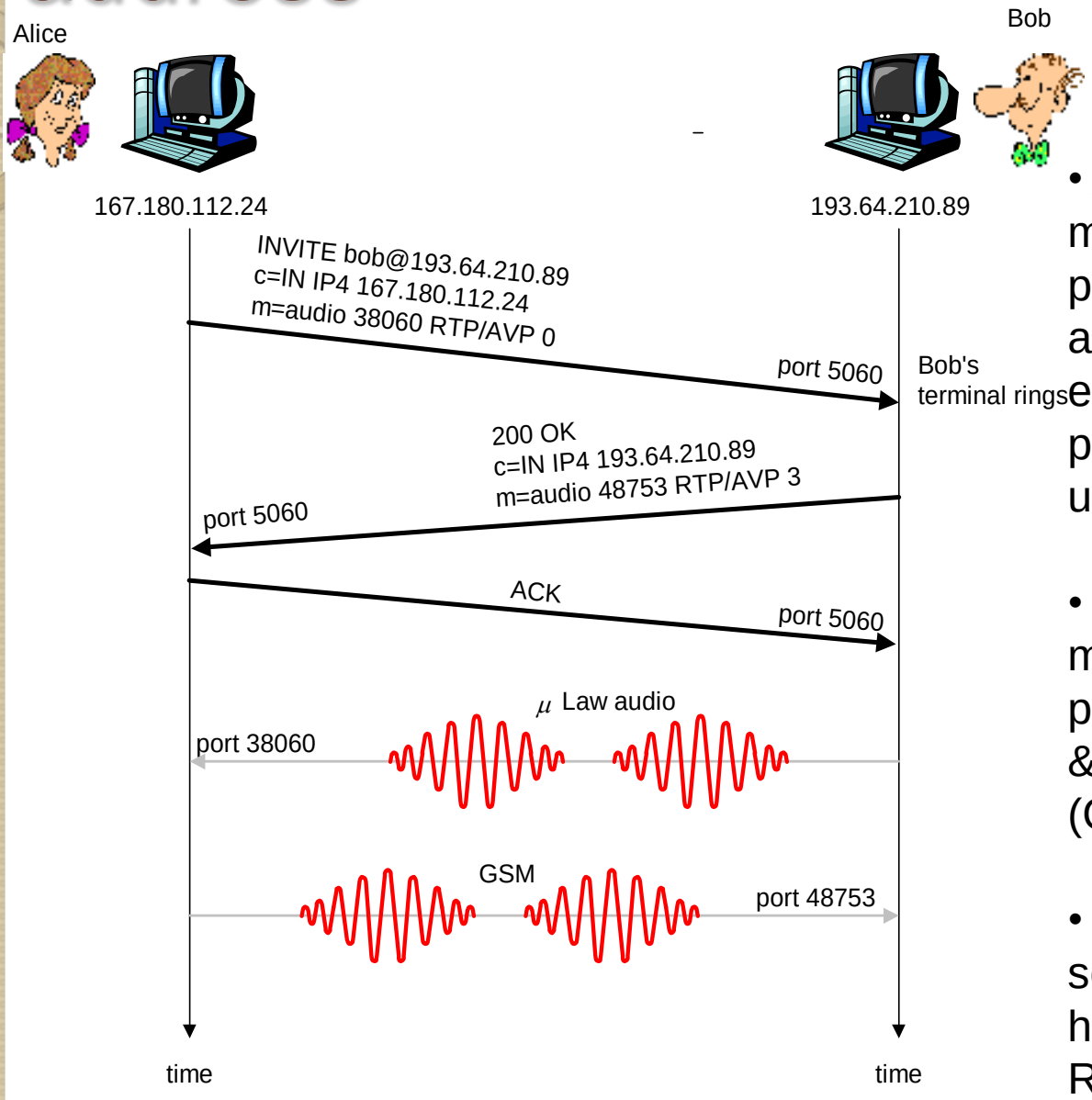
SIP long-term vision

- All telephone calls and video conference calls take place over the Internet
- People are identified by names or e-mail addresses, rather than by phone numbers.
- You can reach the callee, no matter where the callee roams, no matter what IP device the callee is currently using.

SIP Services

- Setting up a call
  - Provides mechanisms for caller to let callee know she wants to establish a call
  - Provides mechanisms so that caller and callee can agree on media type and encoding.
  - Provides mechanisms to end call.
- Determine current IP address of callee.
  - Maps mnemonic identifier to current IP address
- Call management
  - Add new media streams during call
  - Change encoding during call
  - Invite others
  - Transfer and hold calls

Setting up a call to a known IP address



- Alice's SIP invite message indicates her port number & IP address. Indicates encoding that Alice prefers to receive (PCM ulaw)

- Bob's 200 OK message indicates his port number, IP address & preferred encoding (GSM)

- SIP messages can be sent over TCP or UDP; here sent over RTP/UDP.

Setting up a call (more)

- Codec negotiation:
  - Suppose Bob doesn't have PCM ulaw encoder.
  - Bob will instead reply with 606 Not Acceptable Reply and list encoders he can use.
  - Alice can then send a new INVITE message, advertising an appropriate encoder.
- Rejecting the call
  - Bob can reject with replies "busy," "gone," "payment required," "forbidden".
- Media can be sent over RTP or some other protocol.

Example of SIP message

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

Notes:

- HTTP message syntax
- sdp = session description protocol
- Call-ID is unique for every call.

- Here we don't know Bob's IP address. Intermediate SIP servers will be necessary.

- Alice sends and receives SIP messages using the SIP default port number 506.

- Alice specifies in Via: header that SIP client sends and receives SIP messages over UDP

Name translation and user locataion

- Caller wants to call callee, but only has callee's name or e-mail address.
- Need to get IP address of callee's current host:
  - user moves around
  - DHCP protocol
  - user has different IP devices (PC, PDA, car device)

- Result can be based on:
  - time of day (work, home)
  - caller (don't want boss to call you at home)
  - status of callee (calls sent to voicemail when callee is already talking to someone)

Service provided by SIP servers:

- SIP registrar server
- SIP proxy server

SIP Registrar

- When Bob starts SIP client, client sends SIP REGISTER message to Bob's registrar server

Register Message:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

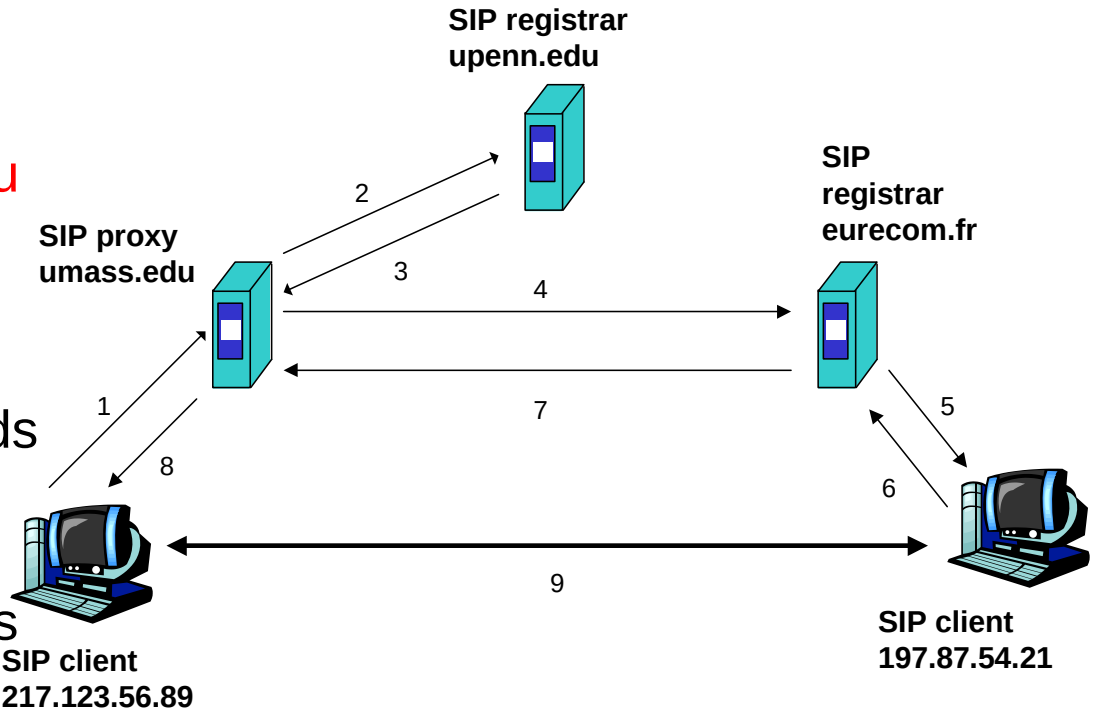
SIP Proxy

- Alice send's invite message to her proxy server
  - contains address sip:bob@domain.com
- Proxy responsible for routing SIP messages to callee
  - possibly through multiple proxies.
- Callee sends response back through the same set of proxies.
- Proxy returns SIP response message to Alice
  - contains Bob's IP address

- Note: proxy is analogous to local DNS server

Example

Caller jim@umass.edu
with places a
call to keith@upenn.edu



(1) Jim sends INVITE message to umass SIP proxy. (2) Proxy forwards request to upenn registrar server.
(3) upenn server returns redirect response, indicating that it should try keith@eurecom.fr

(4) umass proxy sends INVITE to eurecom registrar. (5) eurecom regrist
forwards INVITE to 197.87.54.21, which is running keith's SIP client. (6-8
SIP response sent back (9) media sent directly
between clients.