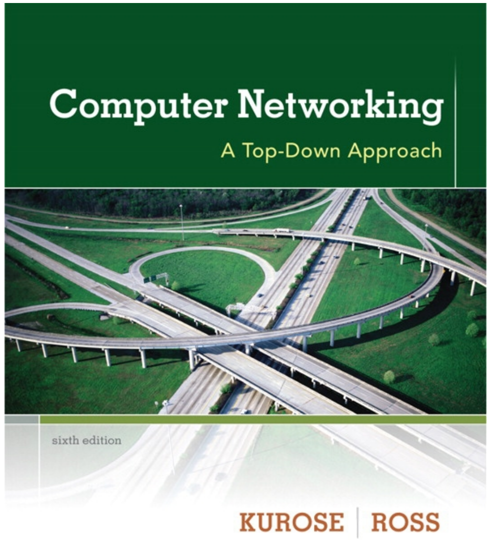
Chapter 7 Multimedia Networking

A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming *stored* video

7.3 voice-over-IP

7.4 protocols for *real-time* conversational applications

7.5 network support for multimedia

Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming *stored* video

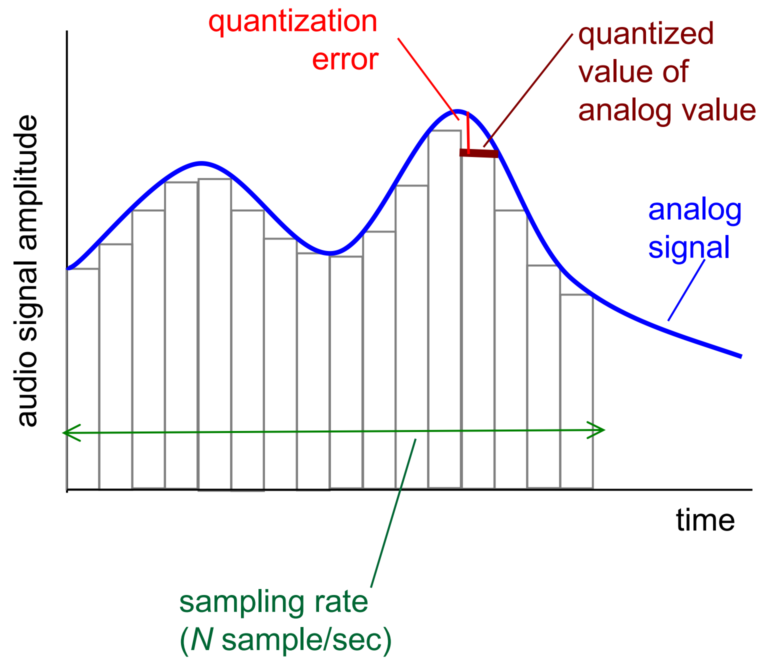
7.3 voice-over-IP

7.4 protocols for *real-time* conversational applications

7.5 network support for multimedia

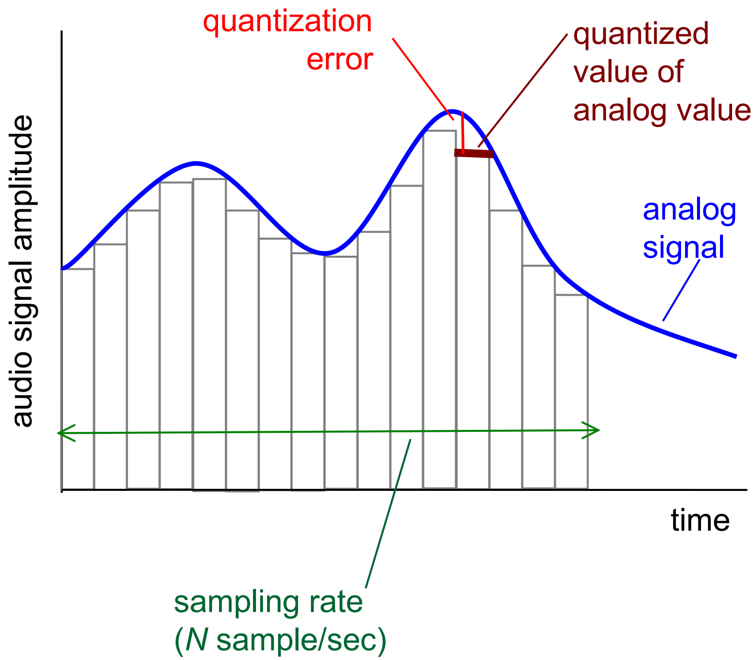
Multimedia: audio

- analog audio signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
  - e.g., 2^8=256 possible quantized values
  - each quantized value represented by bits, e.g., 8 bits for 256 values



Multimedia: audio

- example: 8,000 samples/sec, 256 quantized values: 64,000 bps
- receiver converts bits back to analog signal:
  - some quality reduction



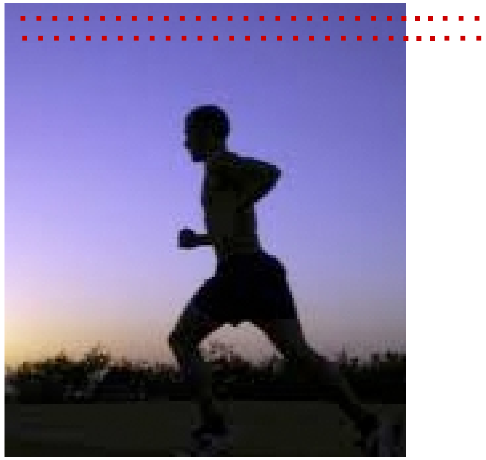
example rates

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

Multimedia: video

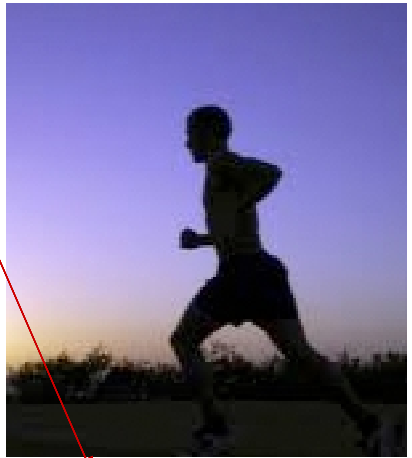
- video: sequence of images displayed at constant rate
  - e.g. 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

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



frame *i*

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

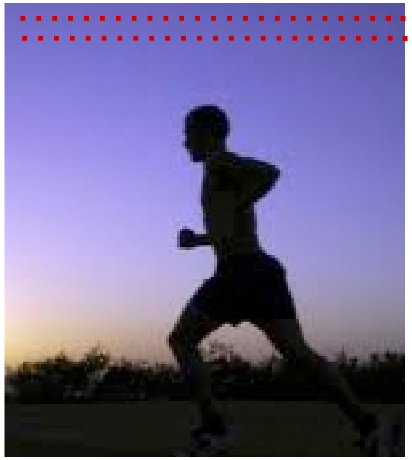


frame *i+1*

Multimedia: video

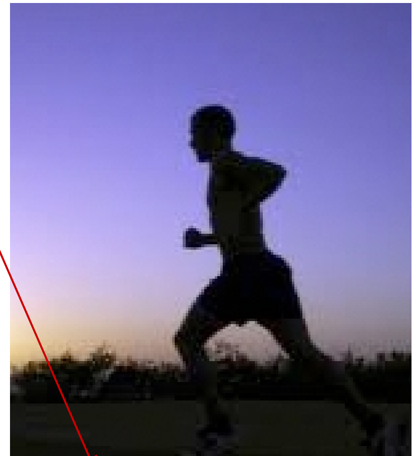
- *CBR: (constant bit rate):* video encoding rate fixed
- *VBR: (variable bit rate):* video encoding rate changes as amount of spatial, temporal coding changes
- *examples:*
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

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



frame *i*

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



frame *i+1*

Multimedia networking: 3 application types

- *streaming, stored* audio, video
  - *streaming:* can begin playout before downloading entire file
  - *stored (at server):* can transmit faster than audio/video will be rendered (implies storing/buffering at client)
  - e.g., YouTube, Netflix, Hulu
- *conversational* voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
  - e.g., Skype
- *streaming live* audio, video
  - e.g., live sporting event (futbol)

Multimedia networking: outline

7.1 multimedia networking applications

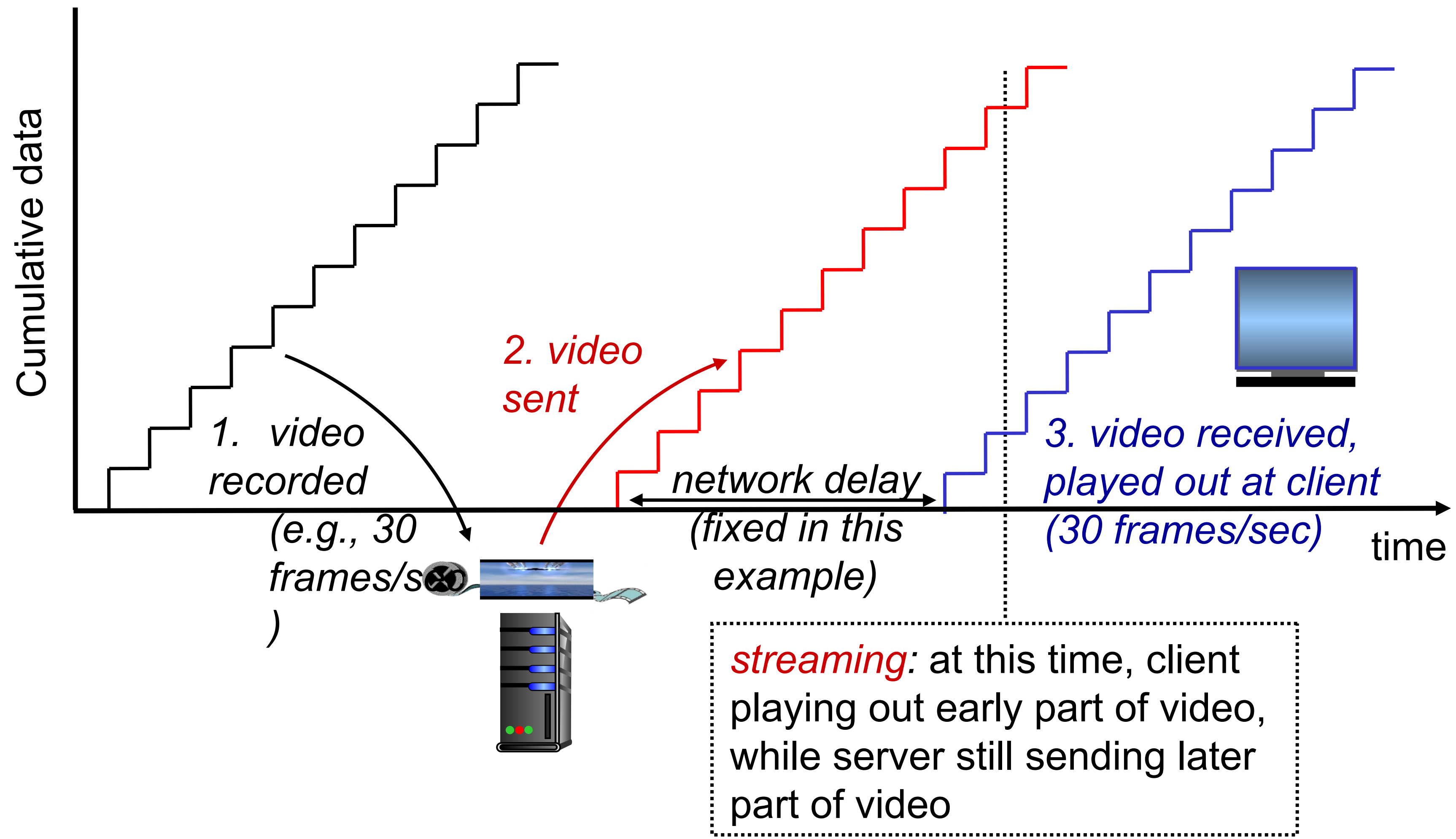
7.2 streaming *stored* video

7.3 voice-over-IP

7.4 protocols for *real-time* conversational applications

7.5 network support for multimedia

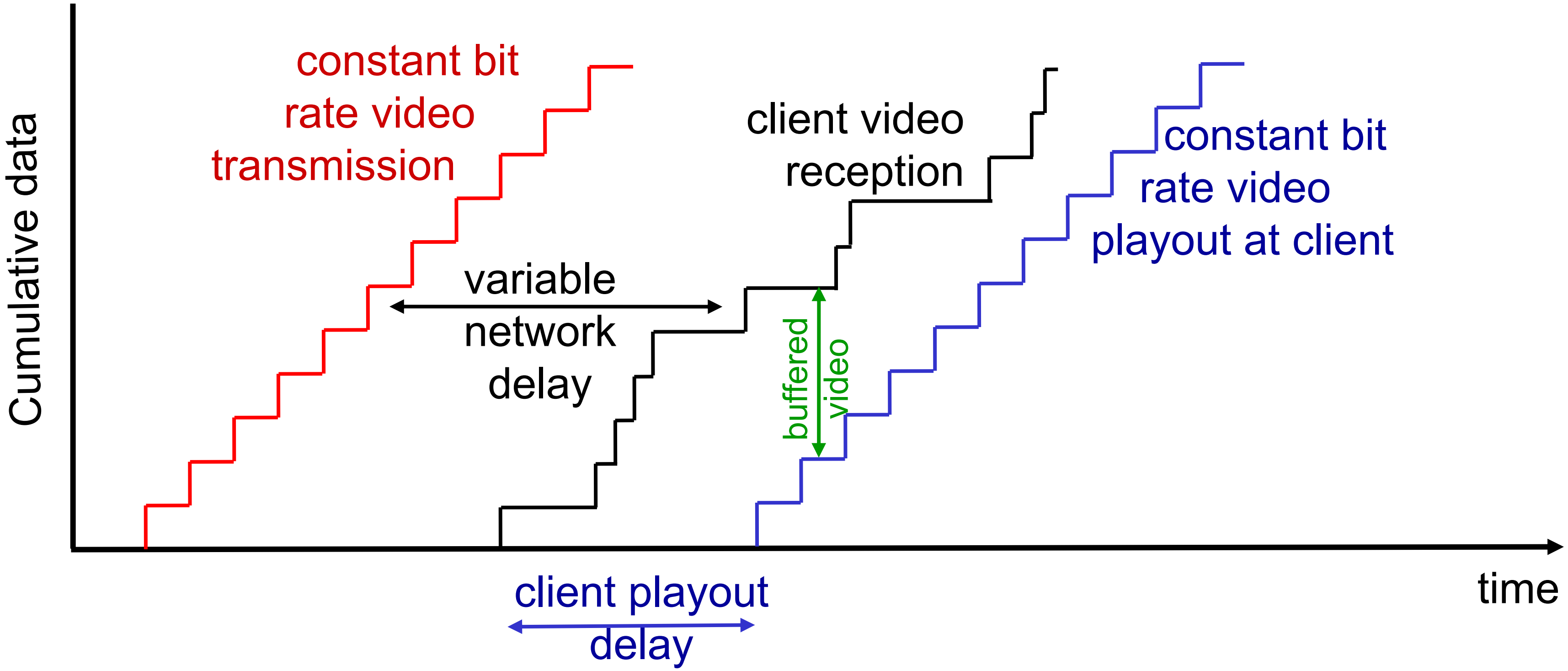
Streaming stored video:



Streaming stored video: challenges

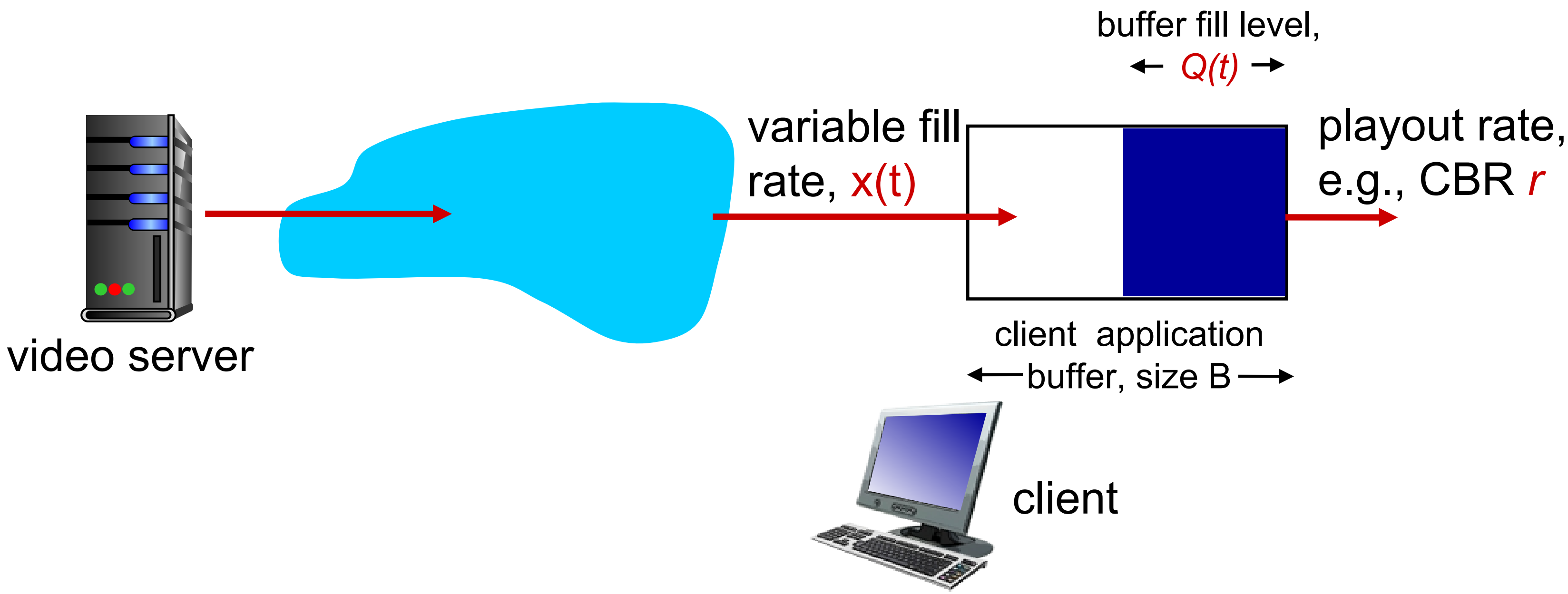
- *continuous playout constraint*: once client playout begins, playback must match original timing
  - … but *network delays are variable* (jitter), so will need *client-side buffer* to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted

Streaming stored video: revisted

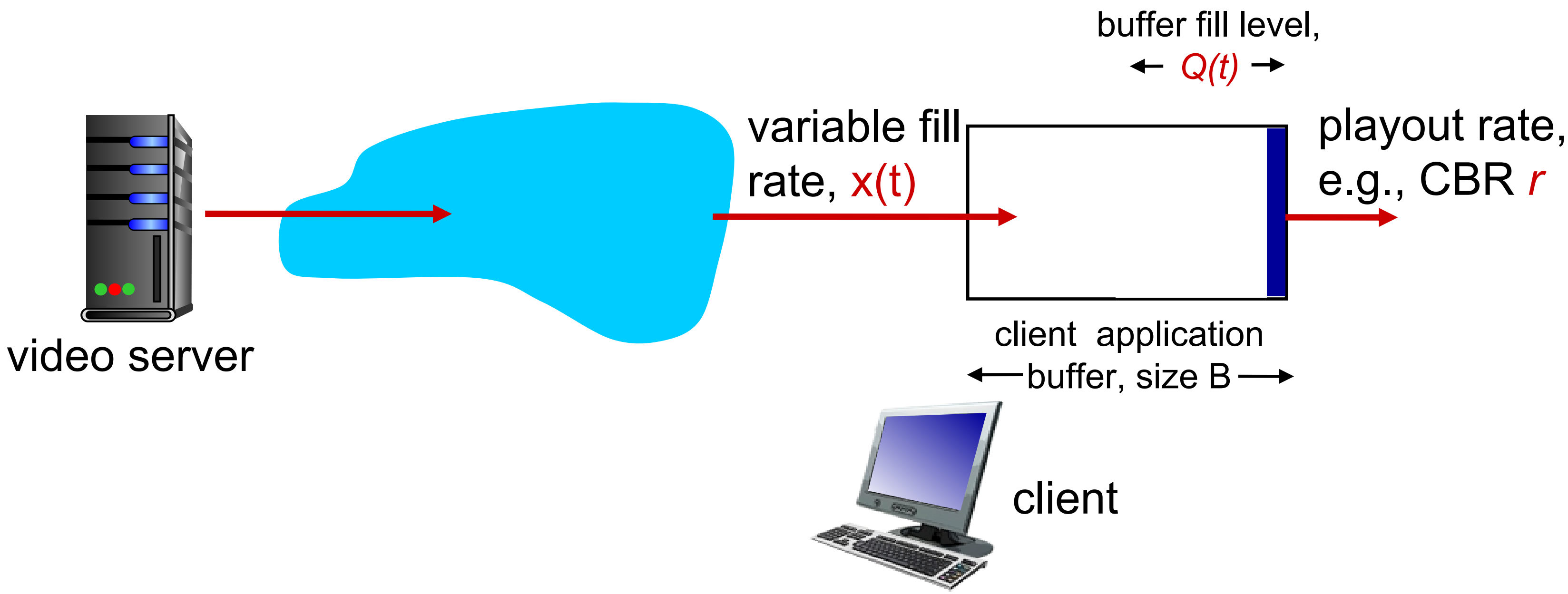


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout



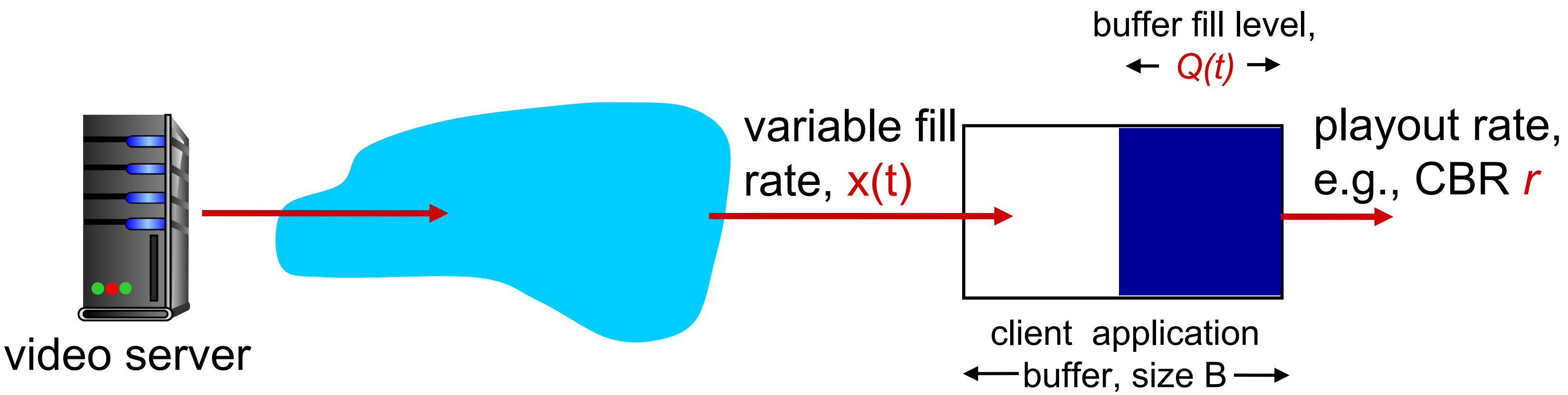
Client-side buffering, playout



1. Initial fill of buffer until playout begins at t_p

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

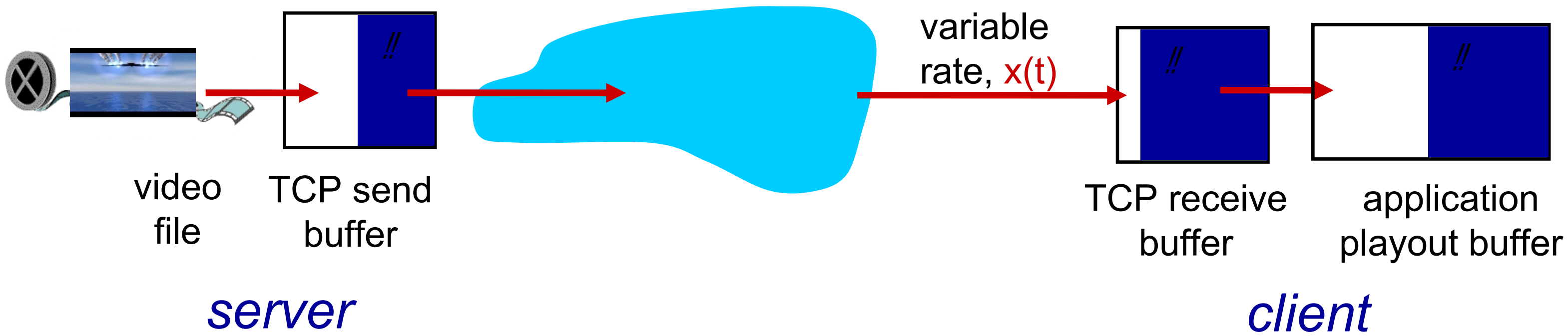
- $\bar{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x} > r$: buffer will not empty, provided initial playout delay is large enough to absorb variability in x(t)
  - *initial playout delay tradeoff:* buffer starvation less likely with larger delay, but larger delay until user begins watching

Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate
  - transmission rate can be oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter
- error recovery: application-level, timeipermitting
- RTP [RFC 2326]: multimedia payload types
- UDP may *not* go through firewalls

Streaming multimedia: HTTP

- 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 playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls

Streaming multimedia: DASH

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

Streaming multimedia: DASH

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

Content distribution networks

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

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

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

Content distribution networks

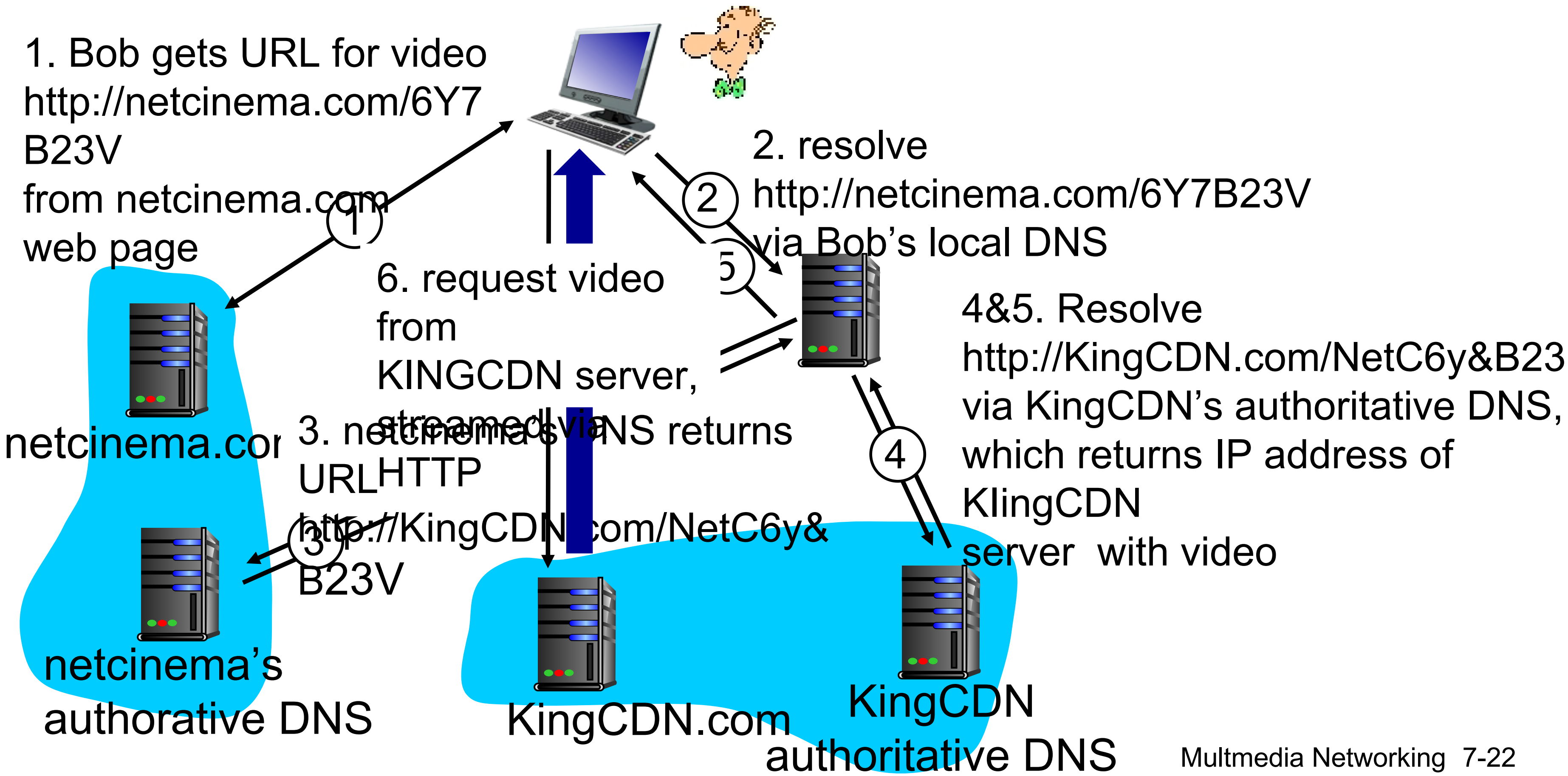
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

CDN: "simple" content access scenario

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

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



CDN cluster selection strategy

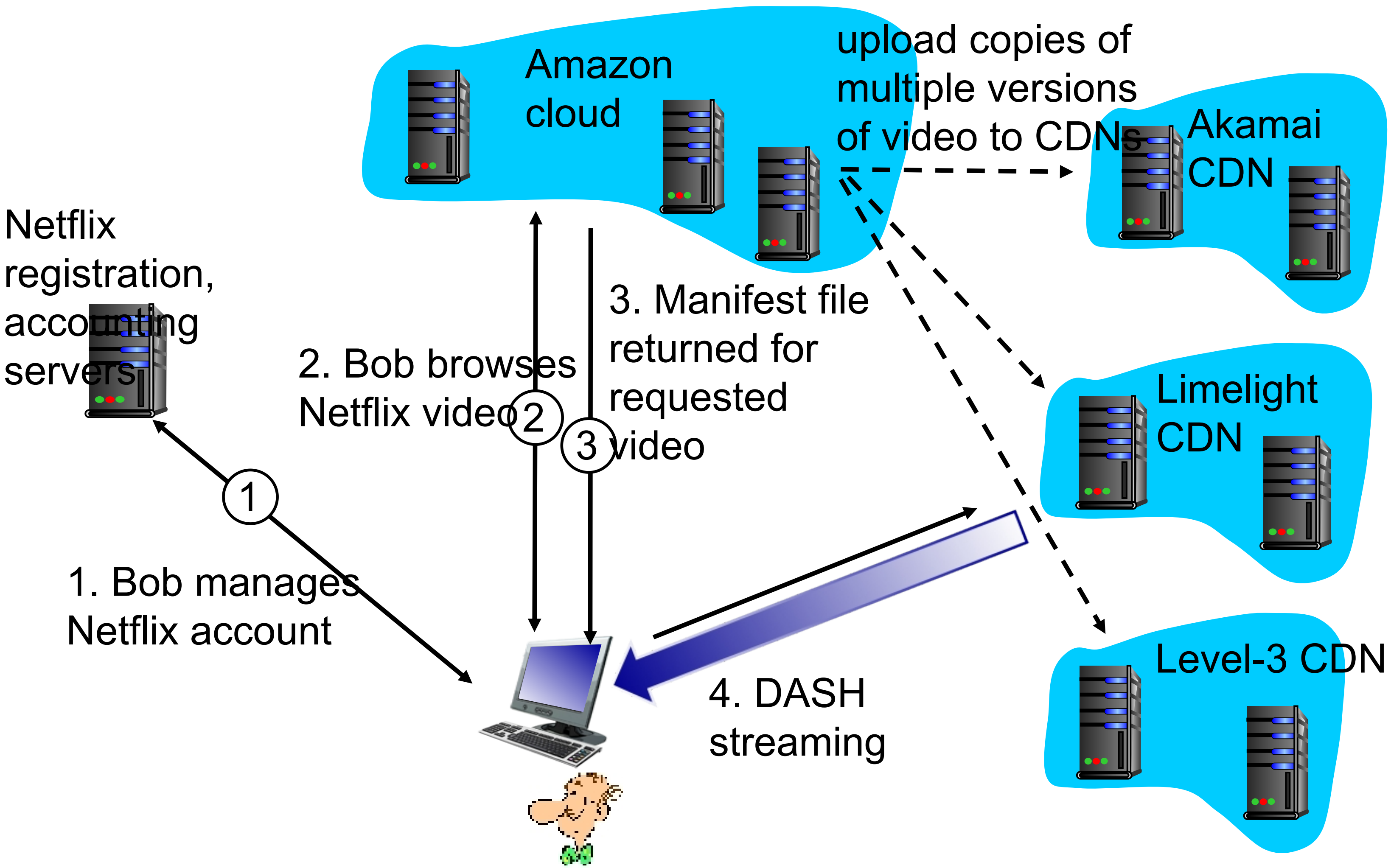
- *challenge:* how does CDN DNS select "good" CDN node to stream to client
  - pick CDN node geographically closest to client
  - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)
  - IP anycast

- *alternative:* let *client* decide - give client a list of several CDN servers
  - client pings servers, picks "best"
  - Netflix approach

Case study: Netflix

- 30% downstream US traffic in 2011
- owns very little infrastructure, uses 3rd party services:
  - own registration, payment servers
  - Amazon (3rd party) cloud services:
    - Netflix uploads studio master to Amazon cloud
    - create multiple version of movie (different endodings) in cloud
    - upload versions from cloud to CDNs
    - Cloud hosts Netflix web pages for user browsing
  - *three* 3rd party CDNs host/stream Netflix content: Akamai, Limelight, Level-3

Case study: Netflix



Multimedia networking: outline

7.1 multimedia networking applications
7.2 streaming *stored* video
7.3 voice-over-IP
7.4 protocols for *real-time* conversational applications
7.5 network support for multimedia

Voice-over-IP (VoIP)

- *VoIP end-end-delay requirement*: needed to maintain "conversational" aspect
  - higher delays noticeable, impair interactivity
  - $<$ 150 msec: good
  - $>$ 400 msec bad
  - includes application-level (packetization,playout), network delays
- *session initialization:* how does callee advertise IP address, port number, encoding algorithms?
- *value-added services:* call forwarding, screening, recording
- *emergency services:* 911

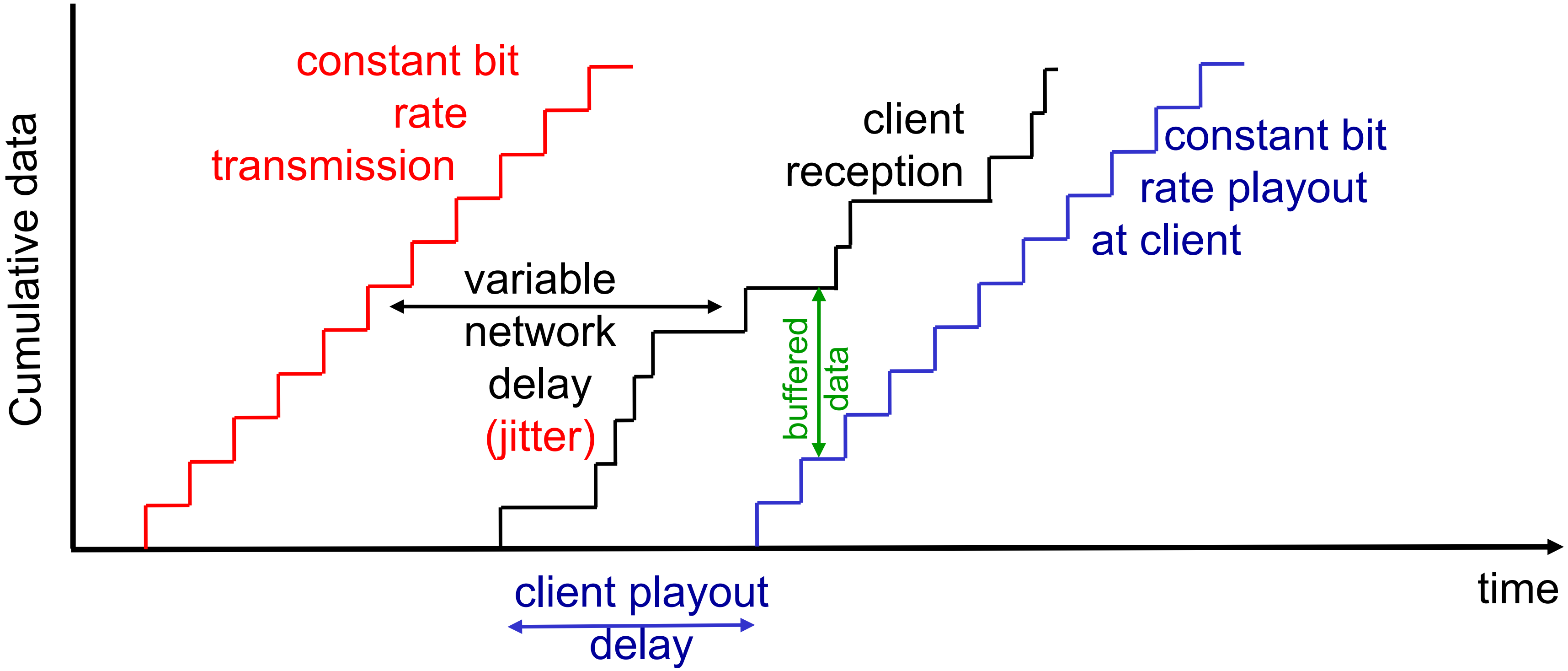
VoIP characteristics

- 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 of data
- application-layer header added to each chunk
- chunk+header encapsulated into UDP or TCP segment
- application sends segment into socket every 20 msec during talkspurt

VoIP: packet loss, 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, loss concealment, packet loss rates between 1% and 10% can be tolerated

Delay jitter



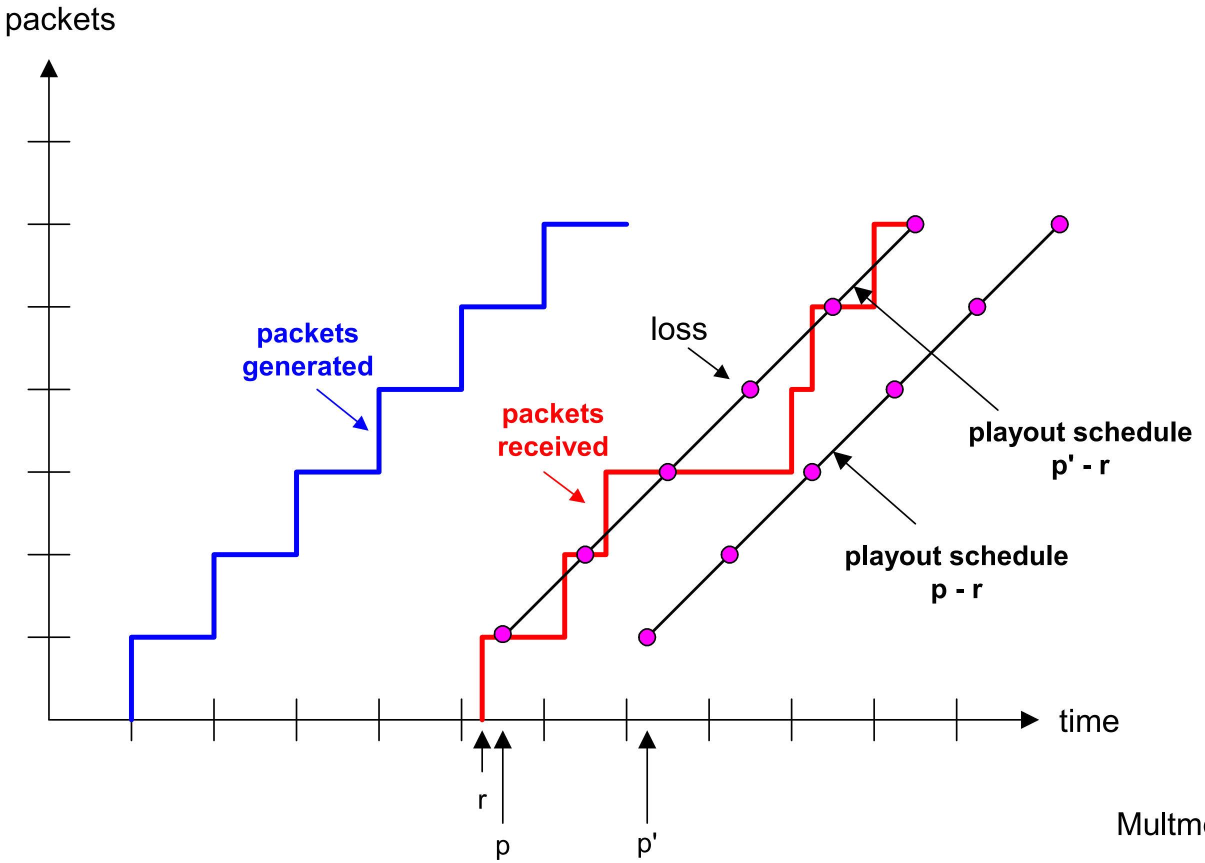
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

VoIP: 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 in choosing *q*:
  - *large q:* less packet loss
  - *small q:* better interactive experience

VoIP: 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'*



Adaptive playout delay (1)

- *goal:* low playout delay, low late loss rate
- *approach:* adaptive playout delay adjustment:
  - estimate network delay, adjust playout delay at beginning of each talk spurt
  - silent periods compressed and elongated
  - chunks still played out every 20 msec during talk spurt
- adaptively estimate packet delay: (EWMA - exponentially weighted moving average, recall TCP RTT estimate):

$$d_i = (1-\alpha)d_{i-1} + \alpha\ \underbrace{(r_i - t_i)}_{\text{measured delay of ith packet}}$$

delay estimate after ith packet

small constant, e.g. 0.1

time received - time sent (timestamp)

measured delay of ith packet

Adaptive playout delay (2)

- also useful to estimate average deviation of delay, v_i

$$v_i = (1-\beta)v_{i-1} + \beta\,|r_i - t_i - d_i|$$

- estimates d_i, v_i calculated for every received packet, but used only at start of talk spurt

- for first packet in talk spurt, playout time is:

$$playout\text{-}time_i = t_i + d_i + Kv_i$$

remaining packets in talkspurt are played out periodically

Adaptive playout delay (3)

Q: How does receiver determine whether packet is first in a talkspurt?

- if no loss, receiver looks at successive timestamps
  - difference of successive stamps > 20 msec -->talk spurt begins.
- with loss possible, receiver must look at both time stamps and sequence numbers
  - difference of successive stamps > 20 msec *and* sequence numbers without gaps --> talk spurt begins.

VoiP: recovery from packet loss (1)

Challenge:

recover from packet loss given small tolerable delay between original transmission and playout

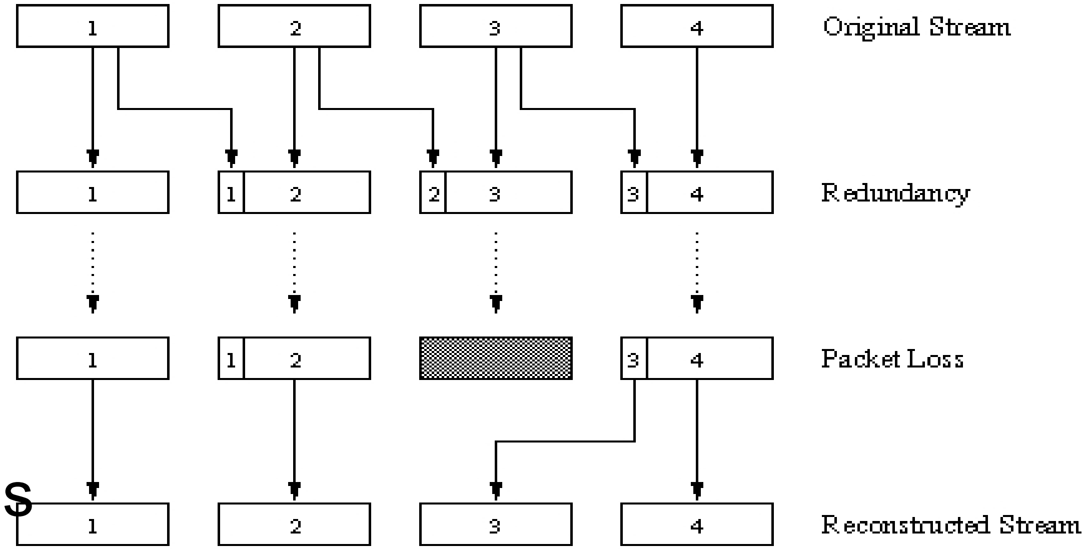
- each ACK/NAK takes ~ one RTT
- alternative: *Forward Error Correction (FEC)*
  - send enough bits to allow recovery without retransmission (recall two-dimensional parity in Ch. 5)

simple FEC

- for every group of *n* chunks, create redundant chunk by exclusive OR-ing *n* original chunks
- send *n+1* chunks, increasing bandwidth by factor *1/n*
- can reconstruct original *n* chunks if at most one lost chunk from *n+1* chunks, with playout delay

VoiP: recovery from packet loss (2)

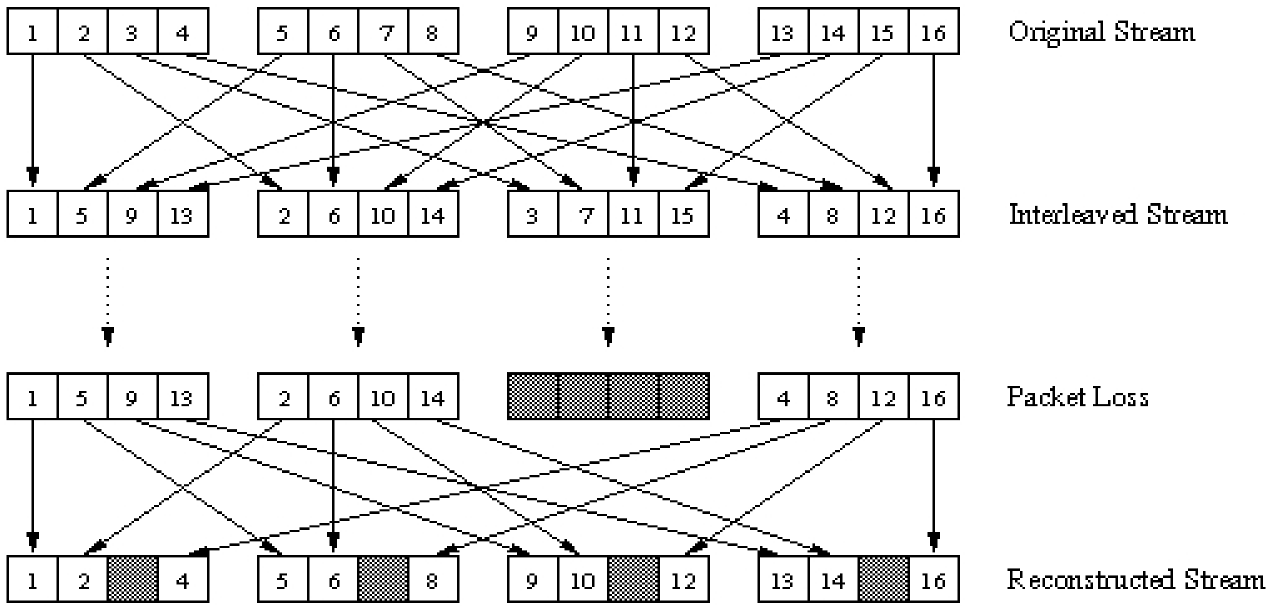
another FEC scheme:



- "piggyback lower quality stream"
- send lower resolution audio stream as redundant information
- e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps

- non-consecutive loss: receiver can conceal loss
- generalization: can also append (n-1)st and (n-2)nd low-bit rate chunk

VoiP: recovery from packet loss (3)



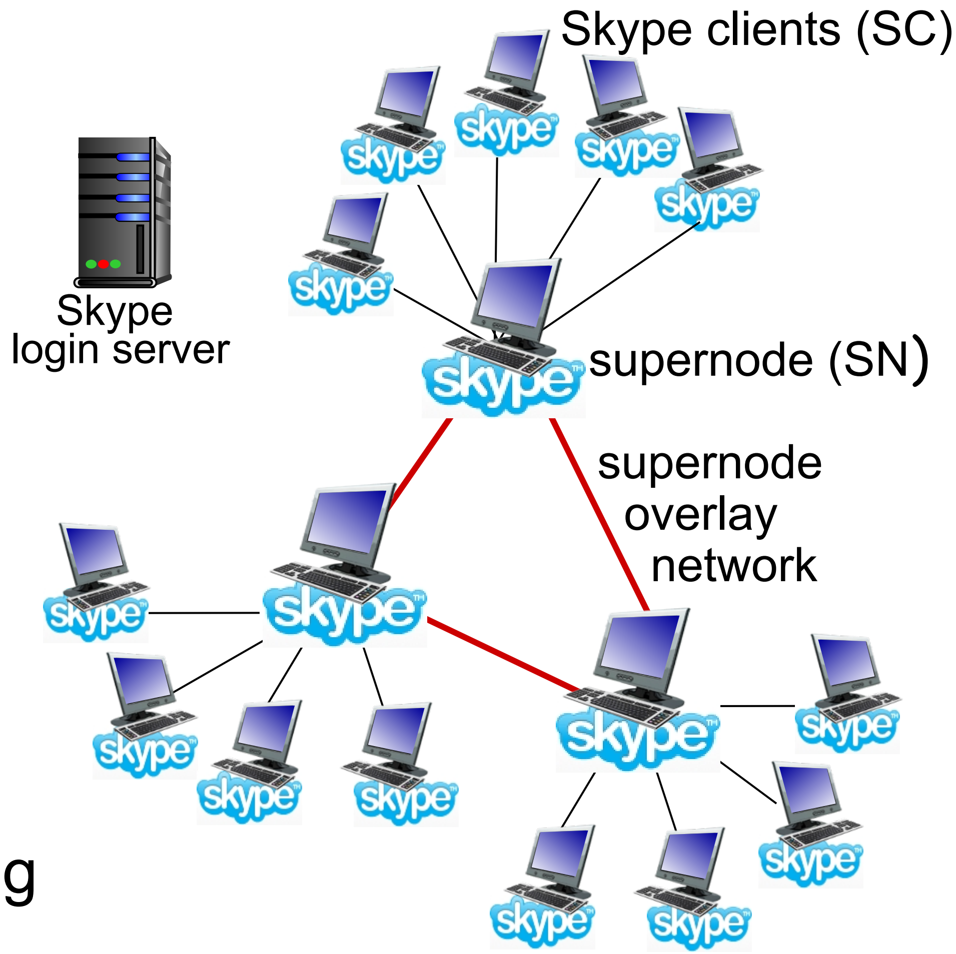
interleaving to conceal loss:

- audio chunks divided into smaller units, e.g. four 5 msec units per 20 msec audio chunk
- packet contains small units from different chunks

- if packet lost, still have *most* of every original chunk
- no redundancy overhead, but increases playout delay

Voice-over-IP: Skype

- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:
  - clients: skype peers connect directly to each other for VoIP call
  - super nodes (SN): skype peers with special functions
  - overlay network: among SNs to locate SCs
  - login server



P2P voice-over-IP: skype

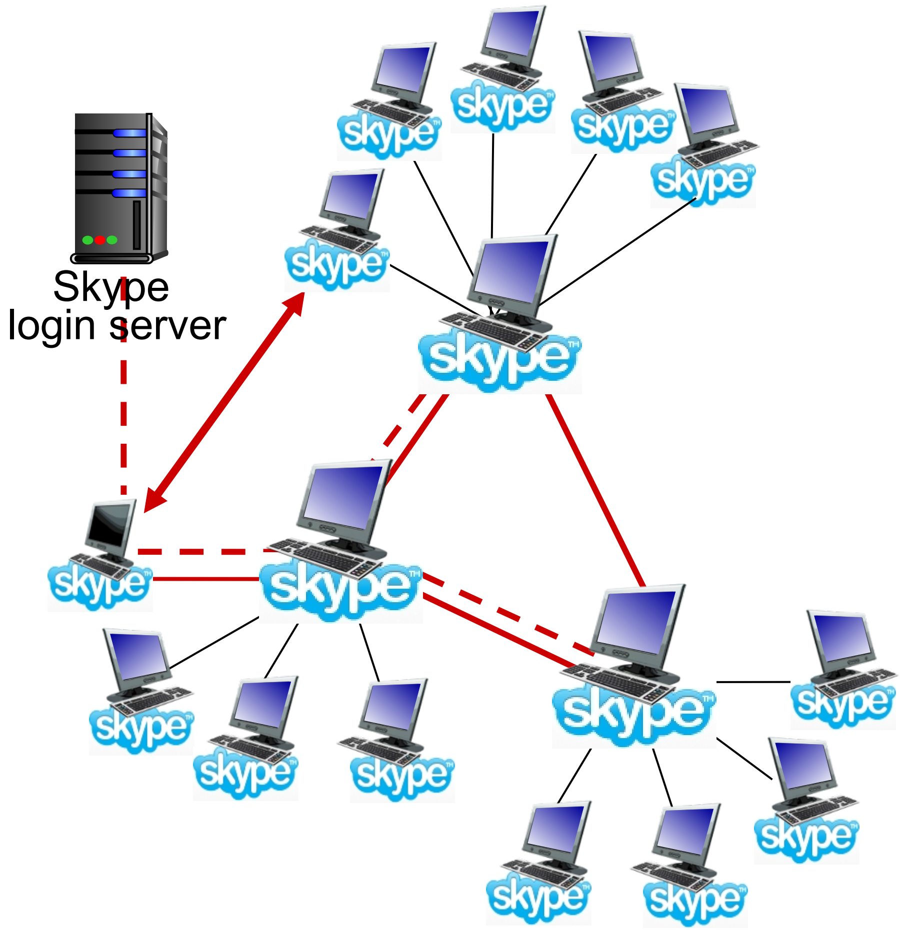
skype client operation:

1. joins skype network by contacting SN (IP address cached) using TCP

2. logs-in (usename, password) to centralized skype login server

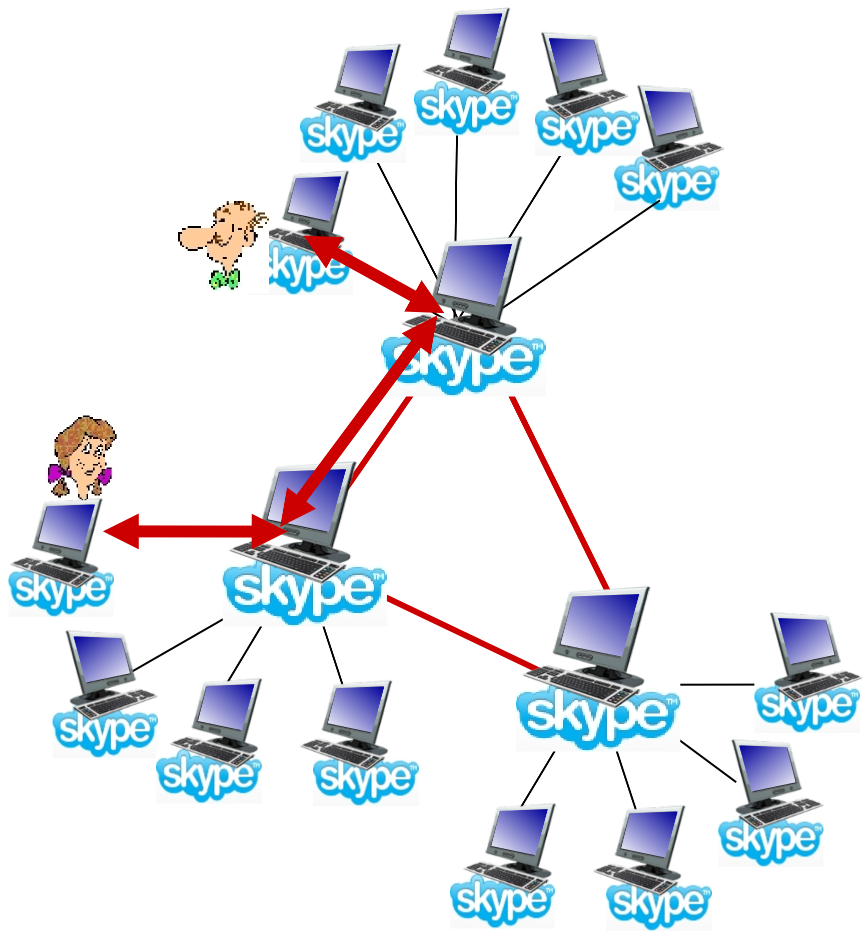
3. obtains IP address for callee from SN, SN overlay
   - or client buddy list

4. initiate call directly to callee



Skype: peers as relays

- *problem:* both Alice, Bob are behind “NATs”
  - NAT prevents outside peer from initiating connection to insider peer
  - inside peer *can* initiate connection to outside
- *relay solution: Alice, Bob maintain open connection to their SNs*
  - Alice signals her SN to connect to Bob
  - Alice’s SN connects to Bob’s SN
  - Bob’s SN connects to Bob over open connection Bob initially initiated to his SN



Multimedia networking: outline

7.1 multimedia networking applications
7.2 streaming stored video
7.3 voice-over-IP
7.4 protocols for real-time conversational applications: RTP, SIP
7.5 network support for multimedia

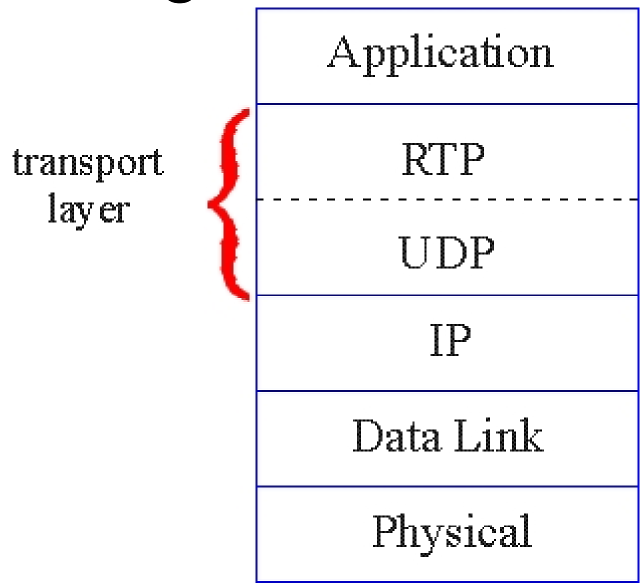
Real-Time Protocol (RTP)

- RTP specifies packet structure for packets carrying audio, video data
- RFC 3550
- RTP packet provides
  - payload type identification
  - packet sequence numbering
  - time stamping
- RTP runs in end systems
- RTP packets encapsulated in UDP segments
- interoperability: if two VoIP applications run RTP, they may be able to work together

RTP runs on top of UDP

RTP libraries provide transport-layer interface that extends UDP:

- port numbers, IP addresses
- payload type identification
- packet sequence numbering
- time-stamping



RTP example

example: sending 64 kbps PCM-encoded voice over RTP

- application collects encoded data in chunks, e.g., every 20 msec = 160 bytes in a chunk
- audio chunk + RTP header form RTP packet, which is encapsulated in UDP segment

- RTP header indicates type of audio encoding in each packet
  - sender can change encoding during conference
- RTP header also contains sequence numbers, timestamps

RTP and QoS

- RTP does *not* provide any mechanism to ensure timely data delivery or other QoS guarantees
- RTP encapsulation only seen at end systems (*not* by intermediate routers)
  - routers provide best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely matter

RTP header

payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields

payload type (7 bits): indicates type of encoding currently being
used. If sender changes encoding during call, sender informs receiver via payload type field

Payload type 0: PCM mu-law, 64 kbps
Payload type 3: GSM, 13 kbps
Payload type 7: LPC, 2.4 kbps
Payload type 26: Motion JPEG
Payload type 31: H.261
Payload type 33: MPEG2 video

sequence # (16 bits): increment by one for each RTP packet sent

- detect packet loss, restore packet sequence

RTP header

payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields

- *timestamp field (32 bits long):* sampling instant of first byte in this RTP data packet
  - for audio, timestamp clock increments by one for each sampling period (e.g., each 125 usecs for 8 KHz sampling clock)
  - if application generates chunks of 160 encoded samples, timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.
- *SSRC field (32 bits long):* identifies source of RTP stream. Each stream in RTP session has distinct SSRC