# Multimedia Networking

#2 Multimedia Networking

Semester Ganjil 2012

PTIIK Universitas Brawijaya

# Schedule of Class Meeting

1. Introduction
2. **Applications of MN**
3. **Requirements of MN**
4. Coding and Compression
5. RTP
6. IP Multicast
7. IP Multicast (cont'd)
8. Overlay Multicast
9. CDN: Solutions
10. CDN: Case Studies
11. QoS on the Internet: Constraints
12. QoS on the Internet: Solutions
13. Discussion
14. Summary

# Today's Outline

2. Multimedia Networking Applications
   - Streaming stored audio/video
   - Streaming live audio/video
   - Real-time interactive audio/video
3. Requirements of Multimedia Networking
   - Limitations of the Best-Effort IP Service
   - Removing Jitter at the Receiver for Audio
   - Recovering from Packet Loss
   - Case Study: Internet Telephony with Skype

# Multimedia networking: 3 application types

- *streaming, stored* audio, video
  - properties: prerecorded
  - e.g., YouTube, Netflix, Hulu, etc.

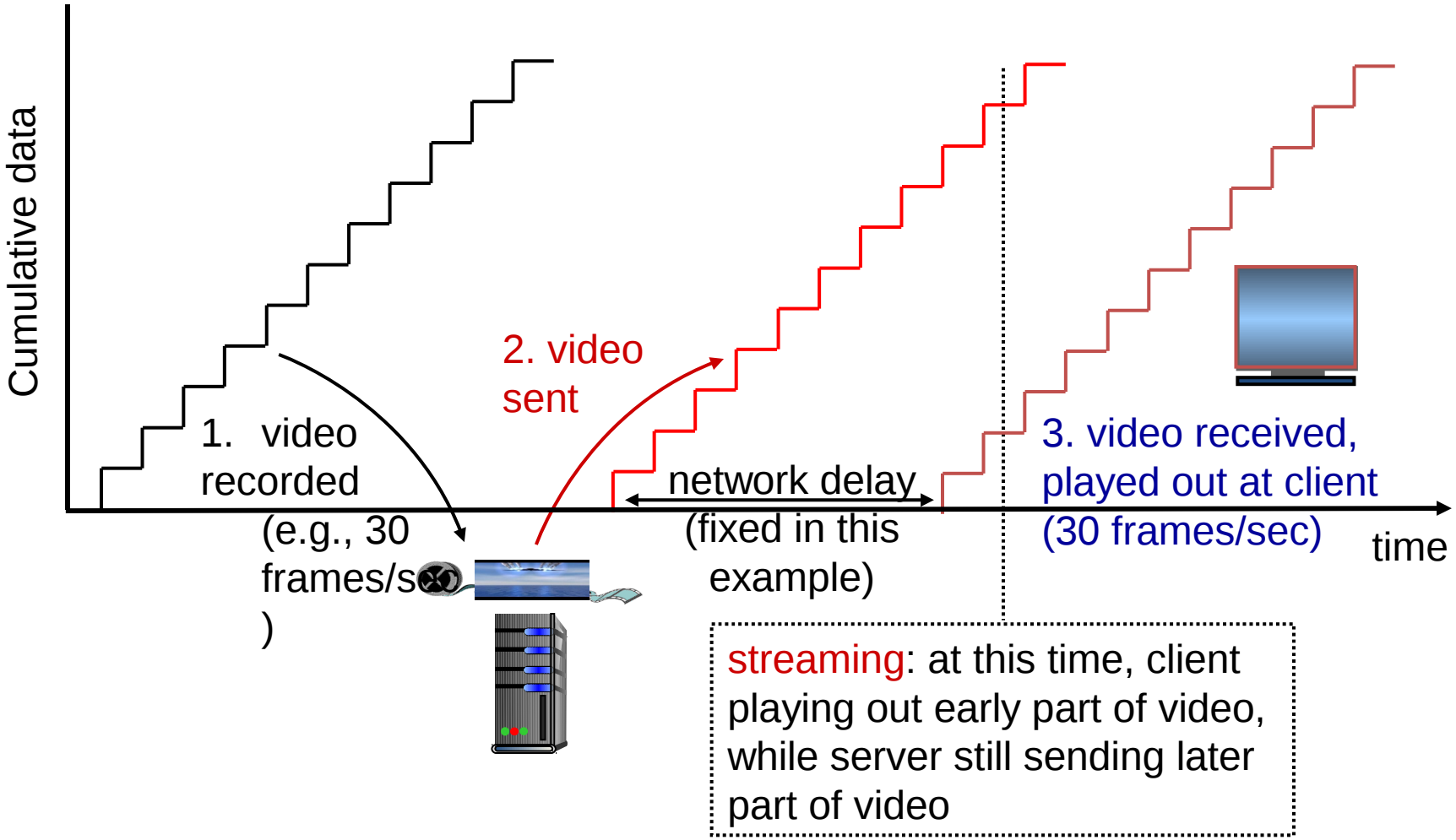
- *conversational* voice/video over IP
  - properties: real-time communication
  - e.g., Skype, GoogleTalk, etc.

- *streaming live* audio, video
  - properties: on-the-fly compression, broadcast
  - e.g., IPTV apps (e.g. Groovia), Ustream, Youtube Live, Livestream, etc.

# Streaming Stored Audio & Video

- Streaming:
  - playing out from one location of video/audio while at the same receiving the later parts of video/audio from the server

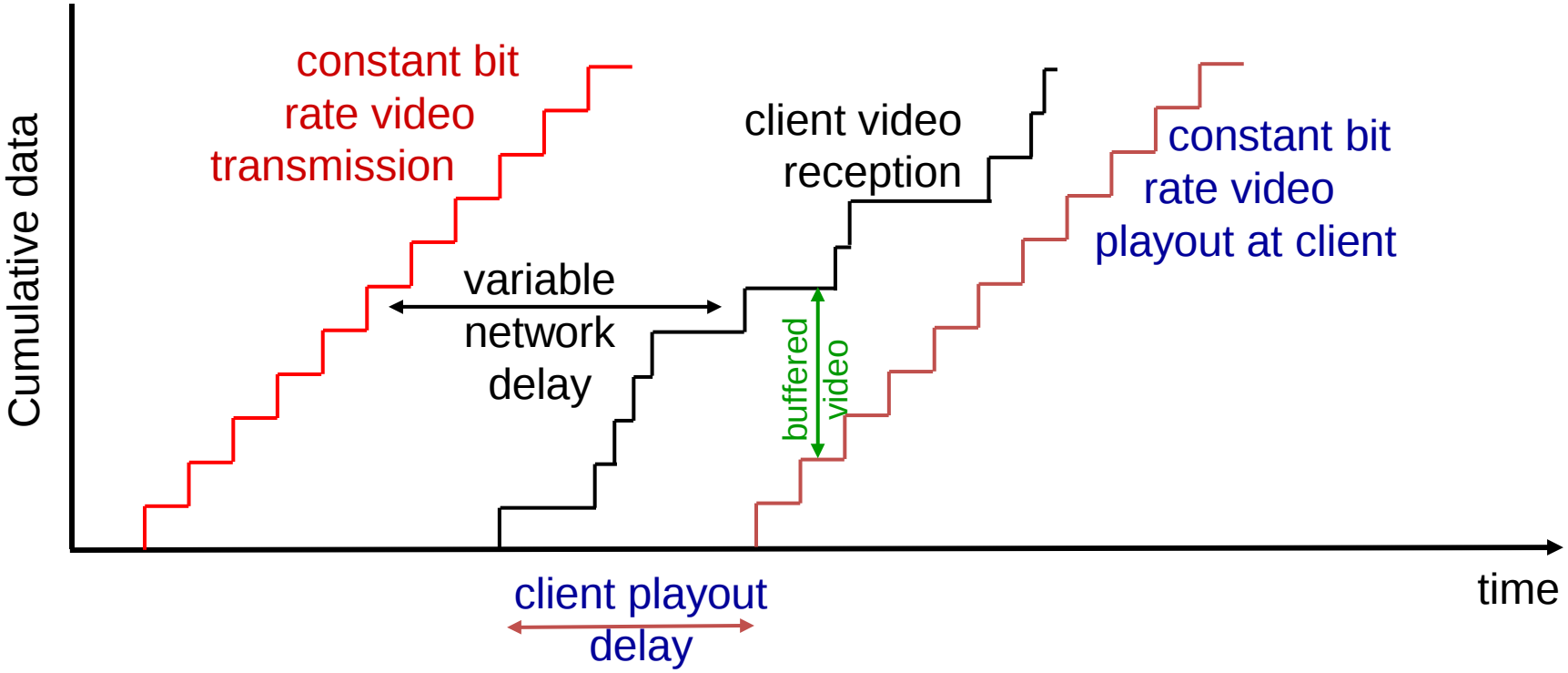
# Streaming stored video:



# Streaming stored video: challenges

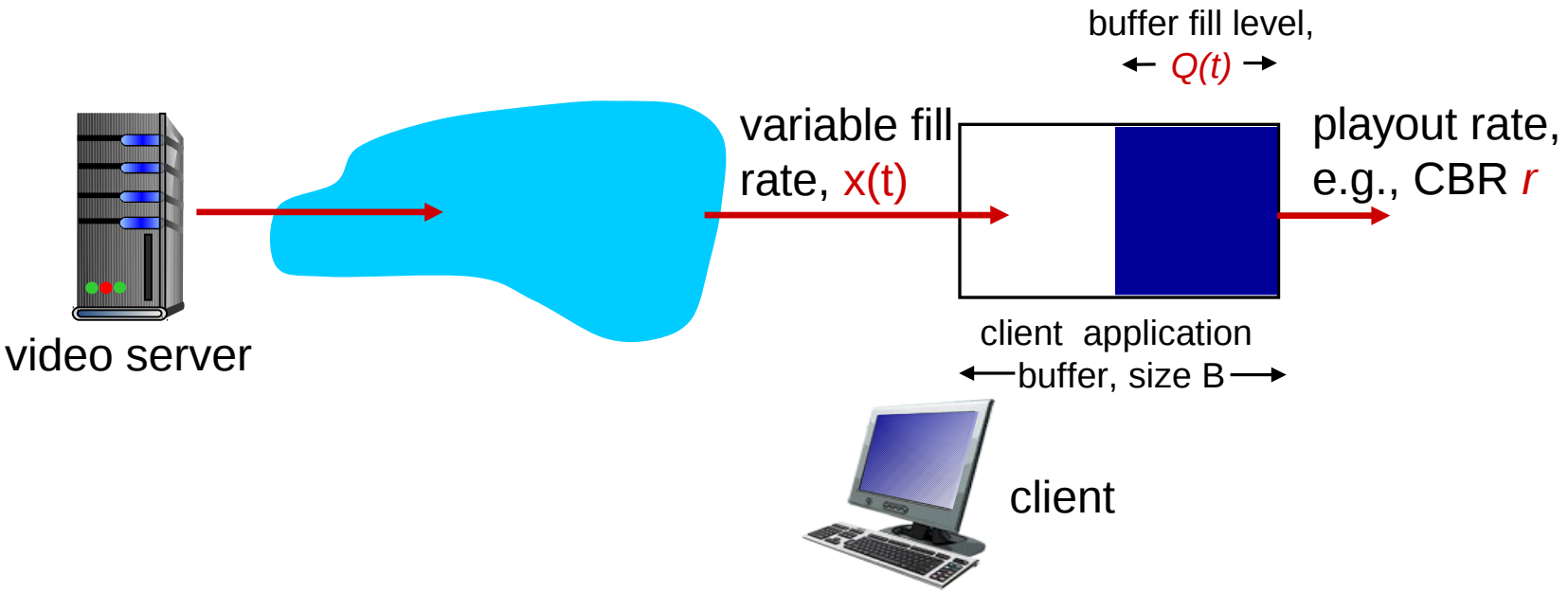
- continuous playout constraint: once client playout begins, playback must match original timing
  - ...
- but net

# Streaming stored video: revisted

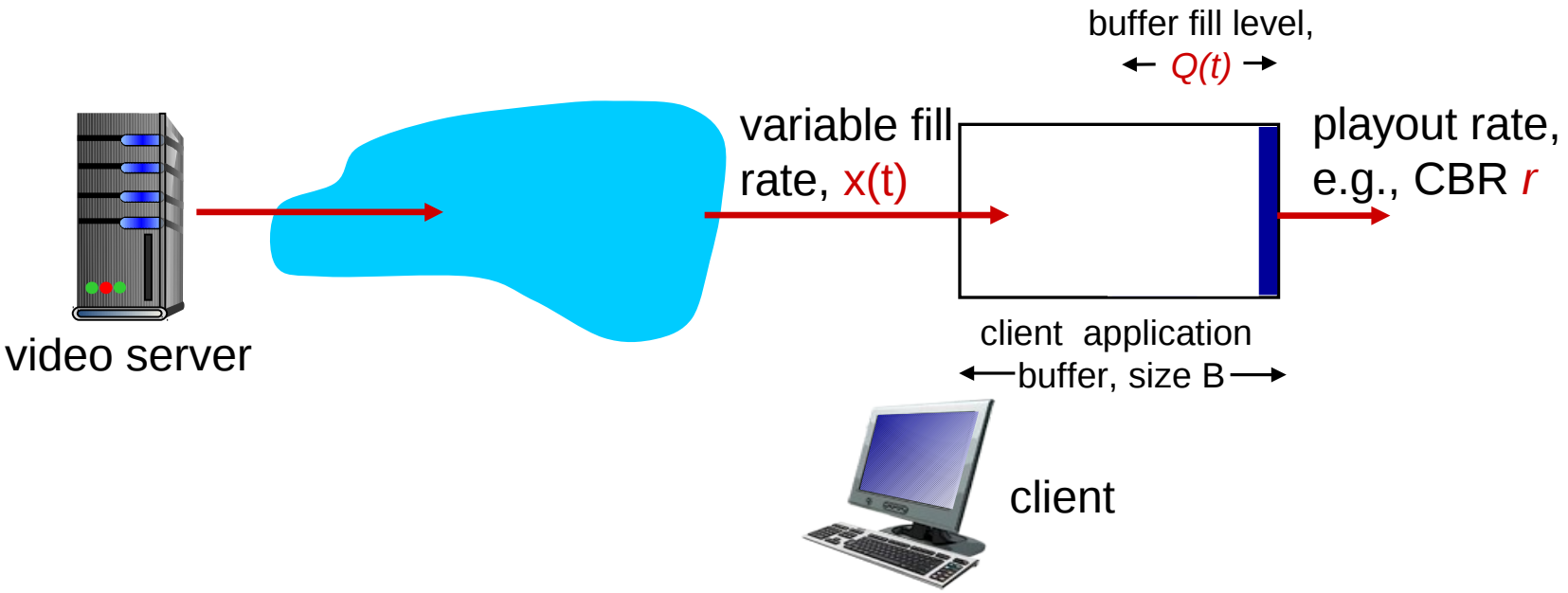



- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

# Client-side buffering, playout

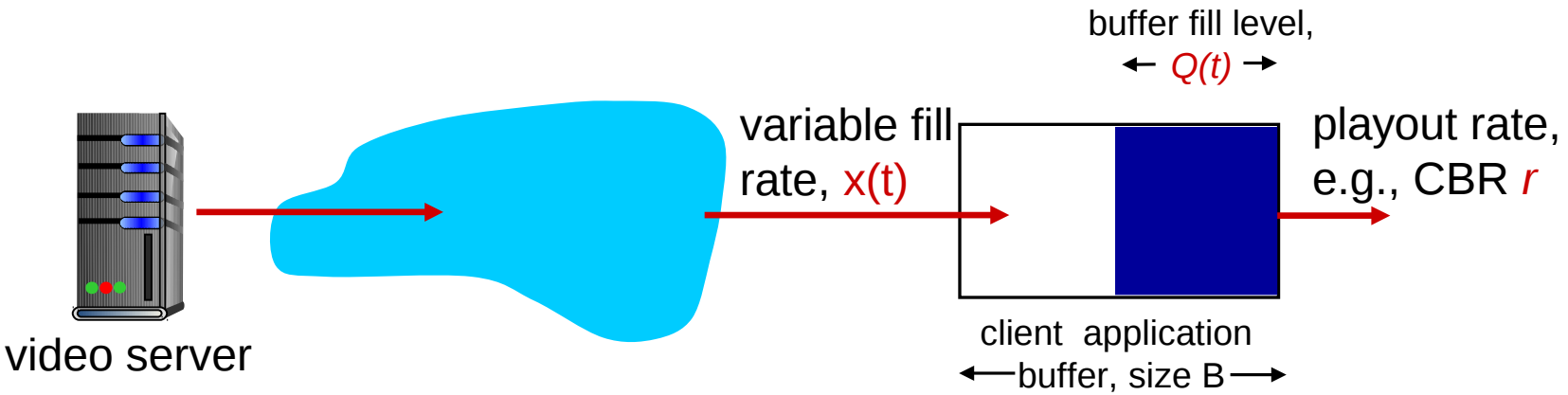


# Client-side buffering, playout




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 (x), playout rate (r):*

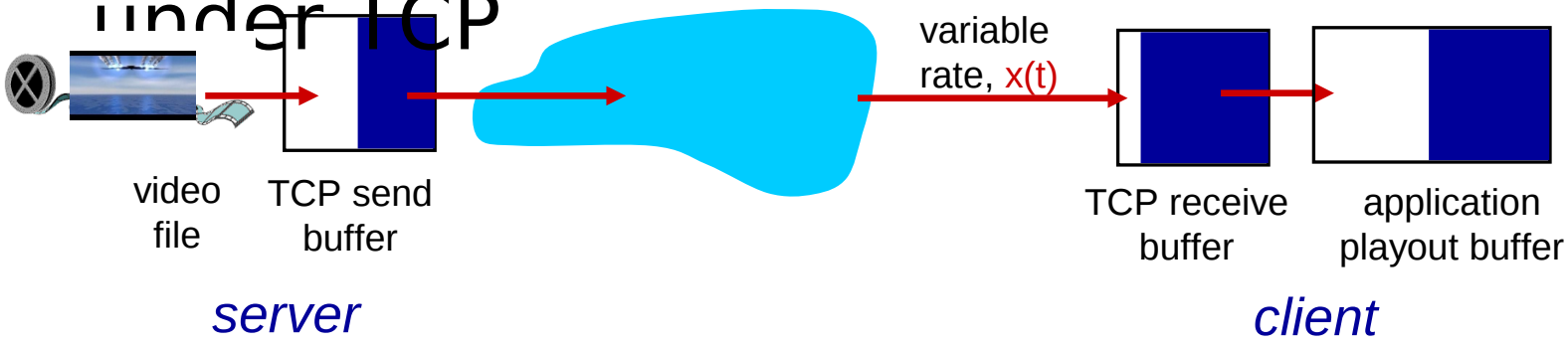
- x < r: buffer eventually empties (causing freezing of video playout until buffer again fills)
- x > r: buffer will not empty, provided initial playout delay is large enough to absorb variability in x(t)
  - *initial playout delay tradeoff:* buffer starvation less likely with larger delay, but larger 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, time permitting
- RTP [RFC 2326]: multimedia payload

# 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)

# 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

# Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *"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

# Today's Outline

2. Multimedia Networking Applications
   - Streaming stored audio/video
   - Streaming live audio/video
   - Real-time interactive audio/video
3. Requirements of Multimedia Networking
   - Limitations of the Best-Effort IP Service
   - Removing Jitter at the Receiver for Audio
   - Recovering from Packet Loss
   - Case Study: Internet Telephony with Skype

# Streaming live audio/video

- Live, broadcast-like
  - multiple receivers receive the same audio/video at the same time.
  - Multicasting is the natural approach of this type apps
  - cannot skip forward
  - same client buffering approach of streaming stored to deal with delay and jitter
  - In practice, a 10–15 second startup delay is usually adequate

# Today's Outline

2. Multimedia Networking Applications
   - Streaming stored audio/video
   - Streaming live audio/video
   - Real-time interactive audio/video
3. Requirements of Multimedia Networking
   - Limitations of the Best-Effort IP Service
   - Removing Jitter at the Receiver for Audio
   - Recovering from Packet Loss
   - Case Study: Internet Telephony with Skype

# Real-time interactive audio/video

- interactive nature of human-to-human conversation
- delay-sensitive
- loss-tolerant
- may include multi-party communication (conferencing)
  - e.g., Polycom, Tandberg, Google Hangouts, Skype, etc.

# 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

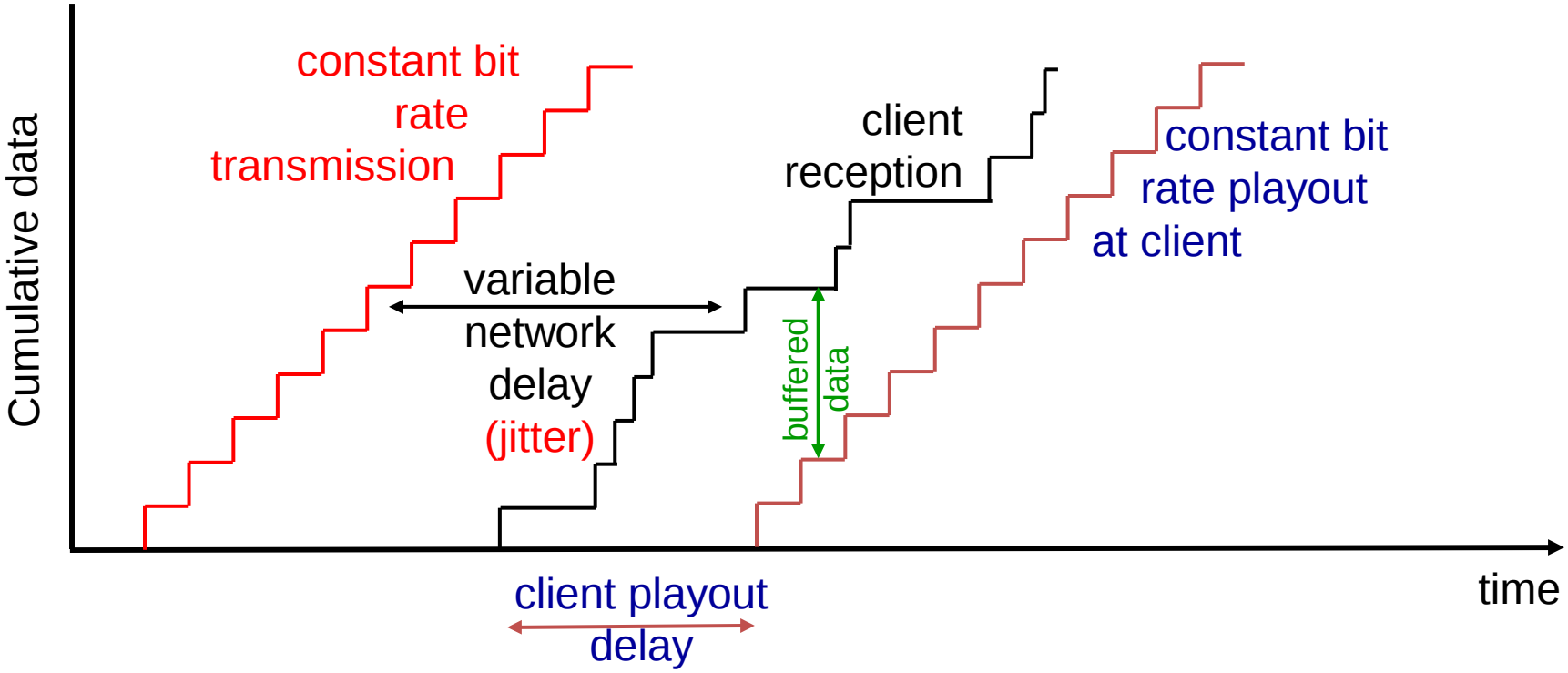
# VoIP characteristics

- 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

# Delay jitter



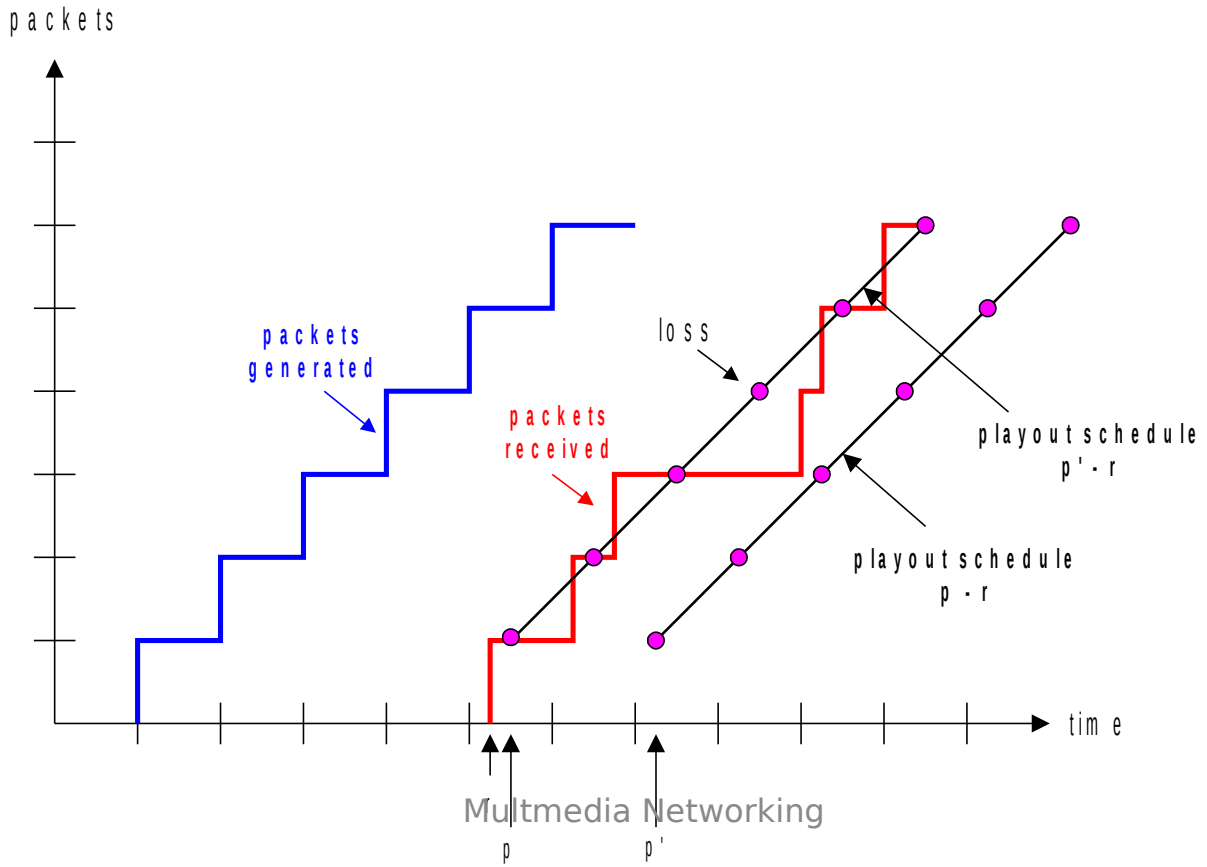

- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

# 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 spu
- 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):

$$d_i = (1-\alpha)d_{i-1} + \alpha(r_i - t_i)$$

*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

# 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

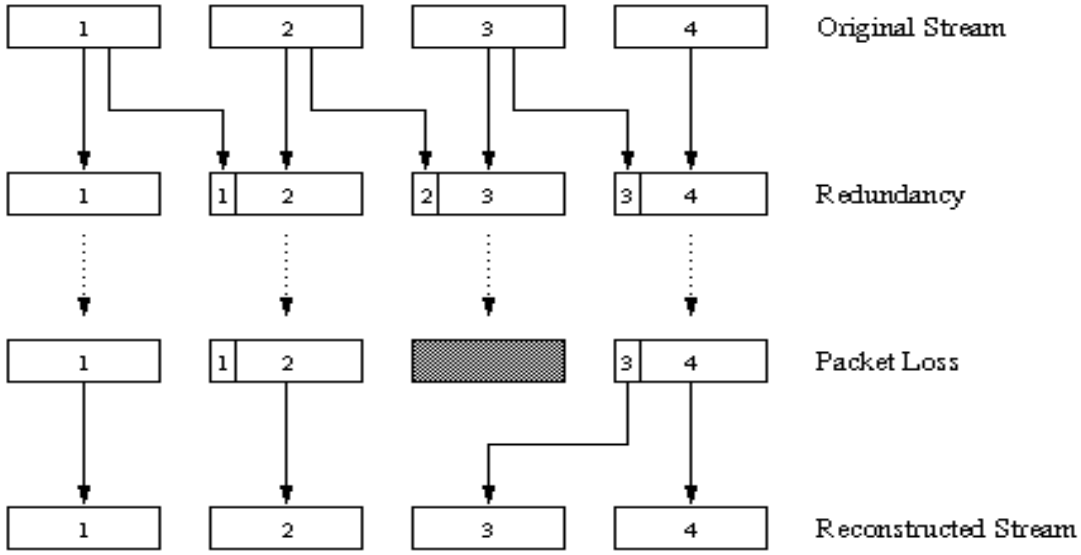
- each ACK/NAK takes ~ one RTT
- alternative: *Forward Error Correction (FEC)*
  - send enough bits to allow recovery without retransmission (recall two-dimensional parity in Ch. 5)

*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)

- "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

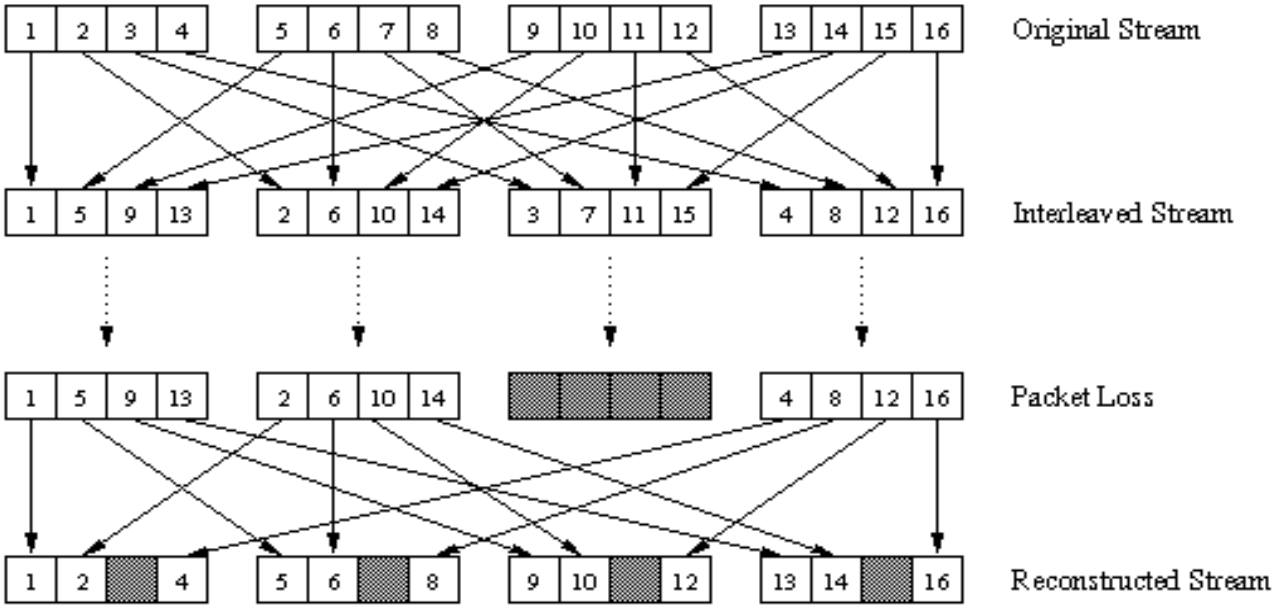



non-consecutive loss: receiver can conceal loss

generalization: can also append (n-1)st and (n-2)nd low-bit r

chunk

# VoIP: recovery from packet loss (3)



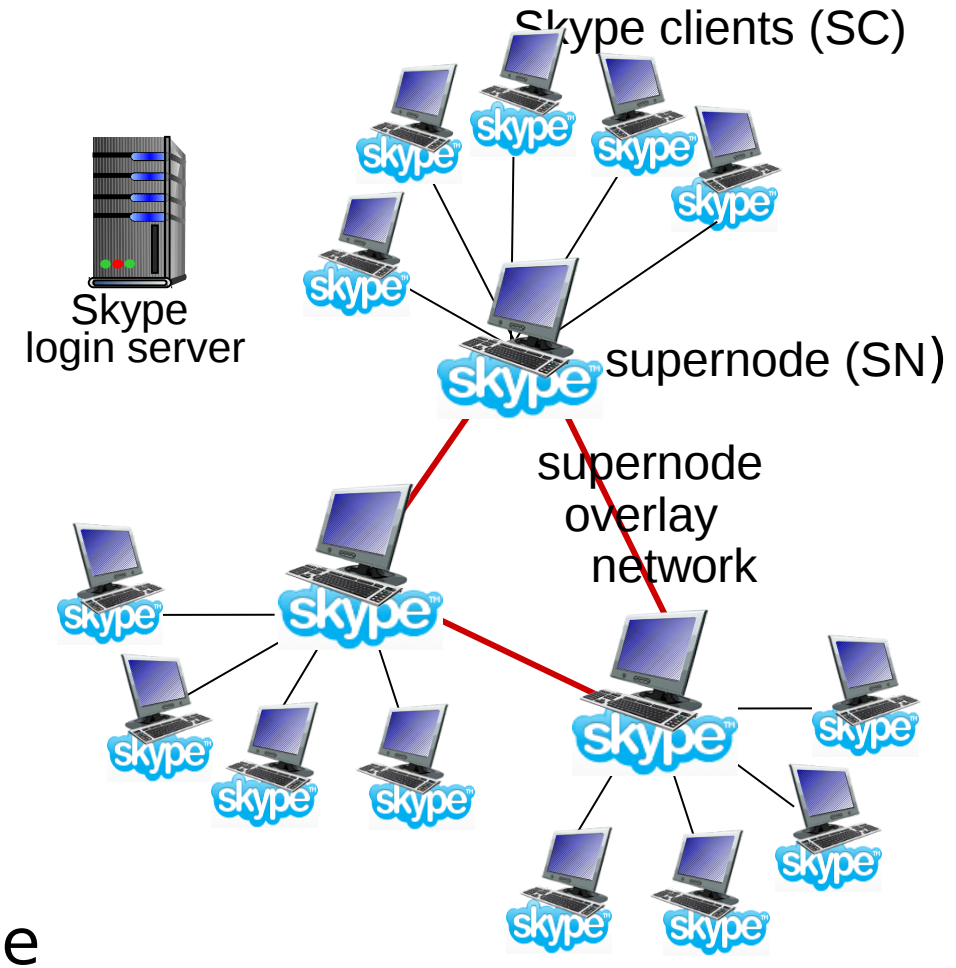

*interleaving to conceal loss:*

- 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

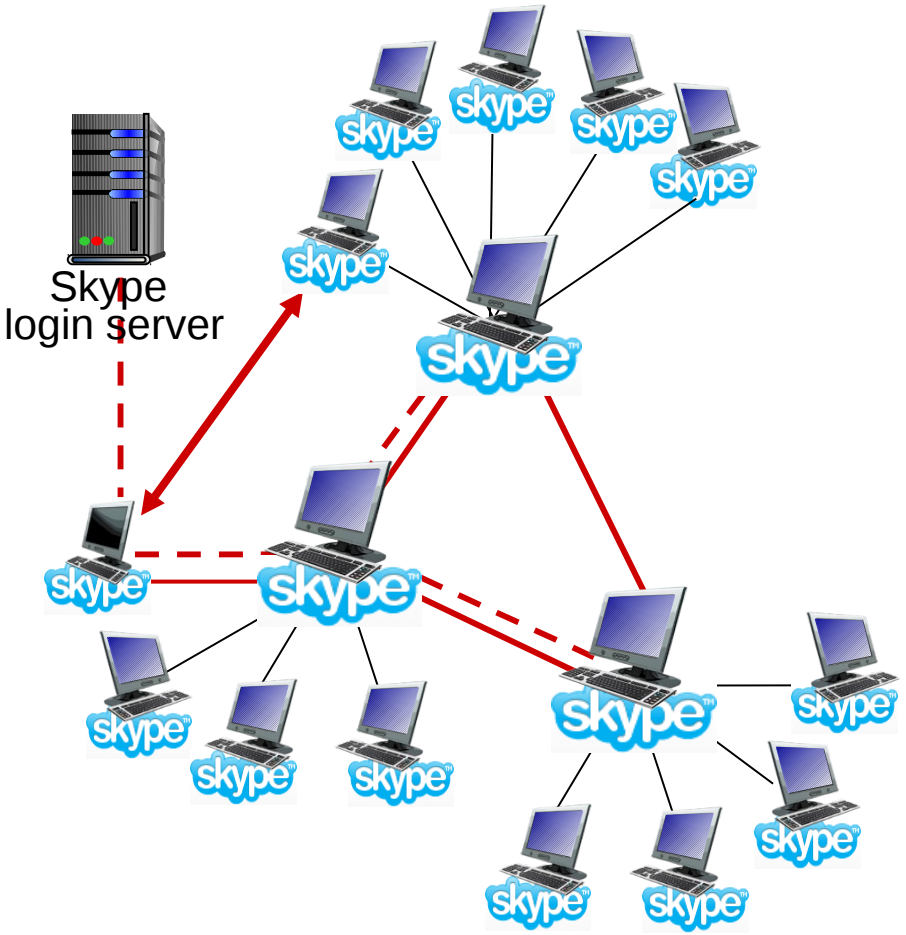
- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:
  - clients: skype peers connect directly to each other for VoIP call
  - super nodes (SN): skype peers with special functions
  - overlay network: among SNs to locate SCs
  - login server



# P2P voice-over-IP: skype

skype client operation:

1. joins skype network by contacting SN (IP address cached) using TCP
2. login (username, 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