50.012 Networks

Lecture 3: Multimedia networking applications

2021 Term 6

Assoc. Prof. CHEN Binbin

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Learning objectives today

- Understand the design of different types of multimedia networking applications
  - Their respective design objectives
  - The approaches to meet the design objectives
  - Their use of the transport-layer service

Outline

Multimedia networking applications

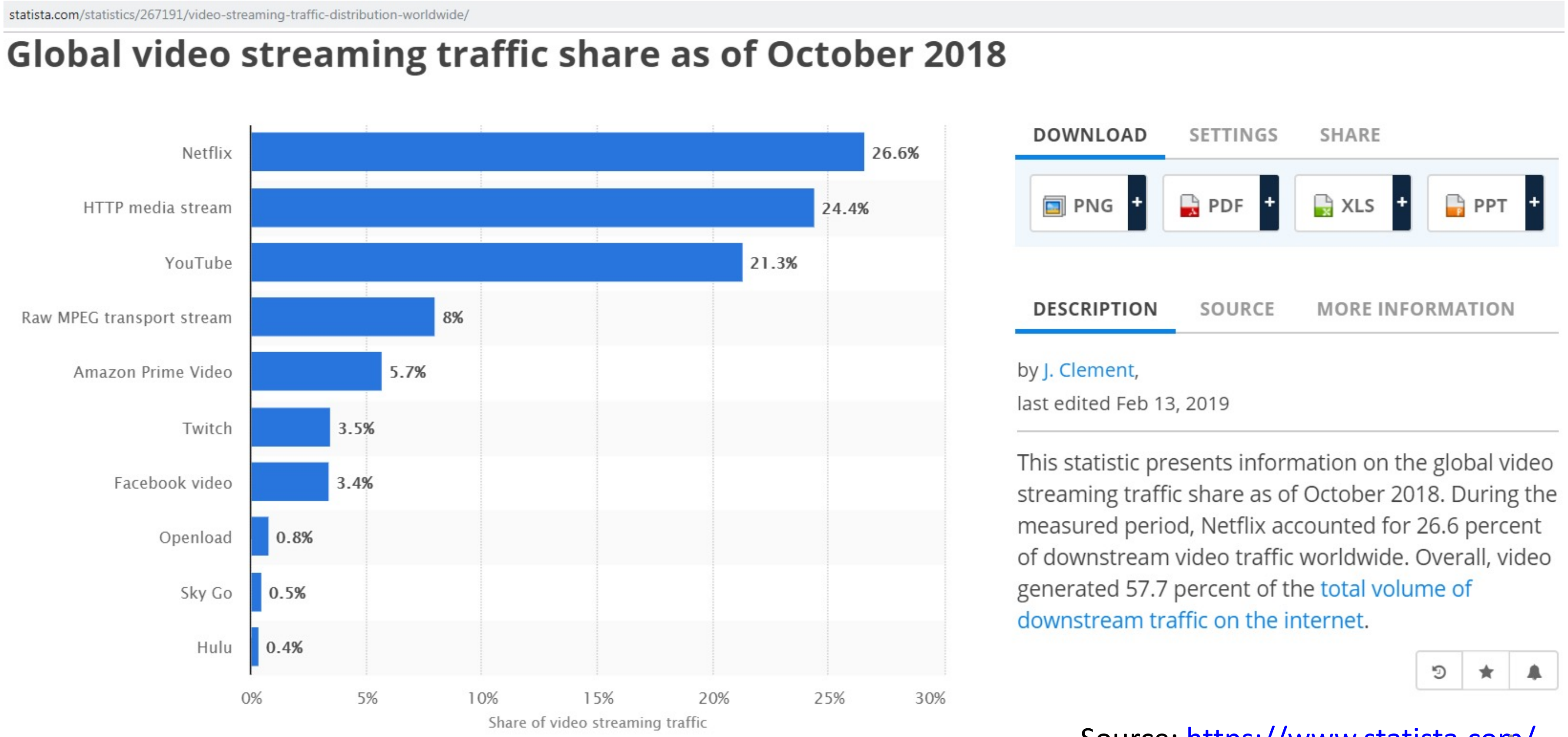
Streaming *stored* video

Voice-over-IP

Read: textbook Section 9.1, 9.2, 9.3

Context

- video traffic: major consumer of Internet bandwidth
- challenge: scale - how to serve > 1B users?
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile)



statista.com/statistics/267191/video-streaming-traffic-distribution-worldwide/

Global video streaming traffic share as of October 2018

by J. Clement,

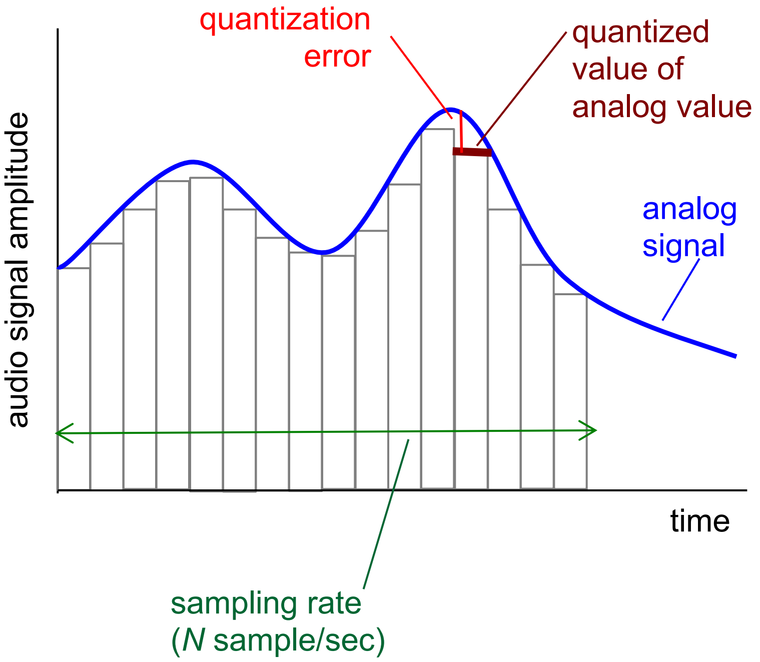
last edited Feb 13, 2019

This statistic presents information on the global video streaming traffic share as of October 2018. During the measured period, Netflix accounted for 26.6 percent of downstream video traffic worldwide. Overall, video generated 57.7 percent of the total volume of downstream traffic on the internet.

Source: https://www.statista.com/

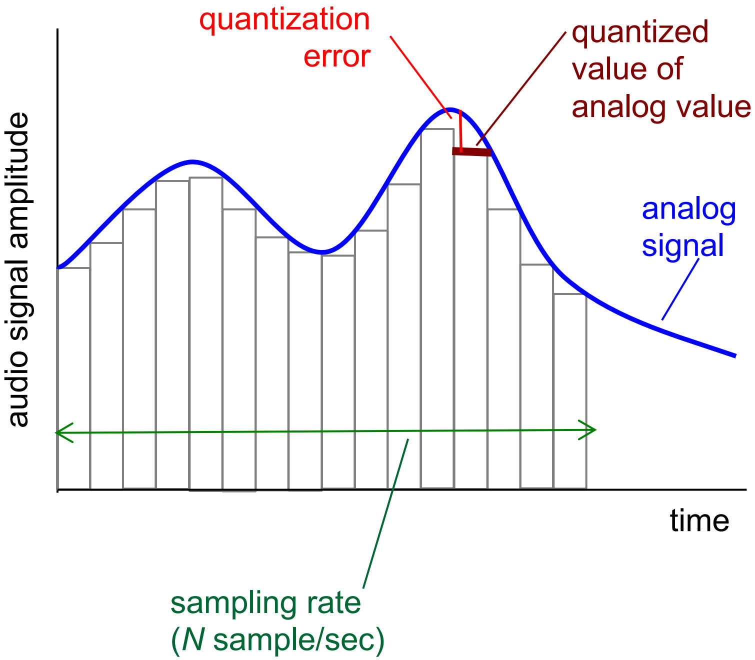
Multimedia: audio

- 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

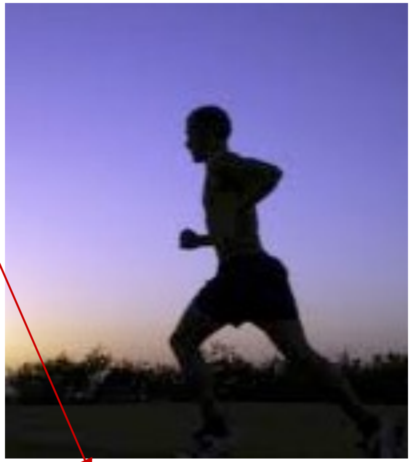
Multimedia: video

- 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*

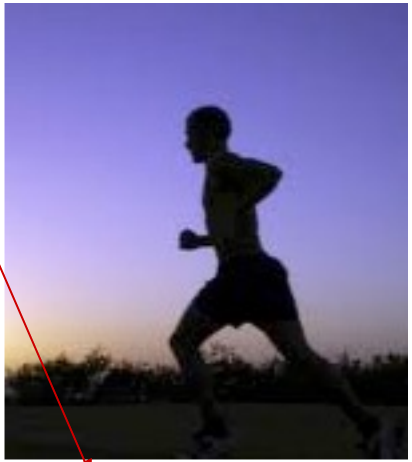
Multimedia: video

- 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

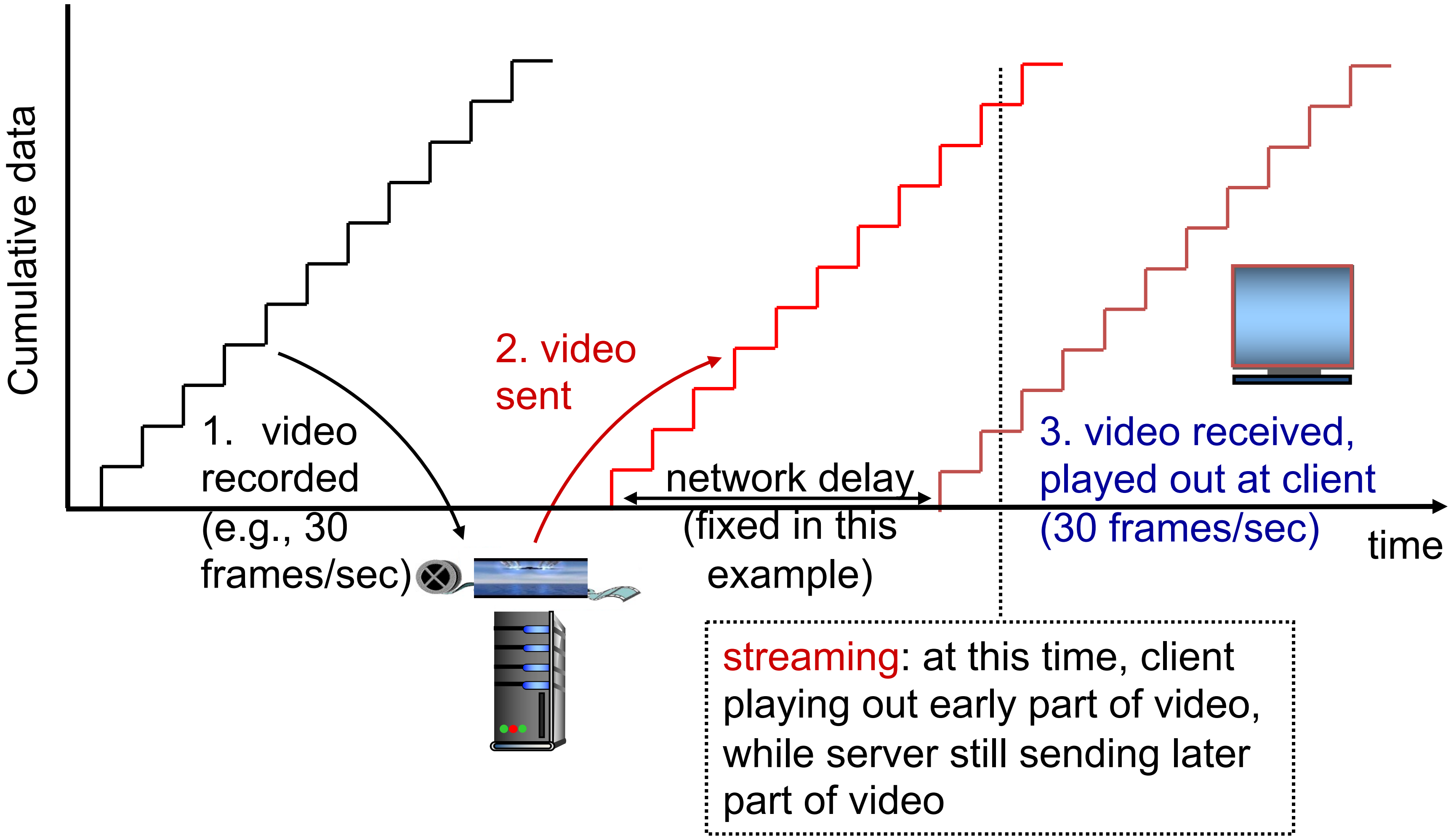
Outline

Multimedia networking applications

Streaming *stored* video

Voice-over-IP

Streaming stored video:

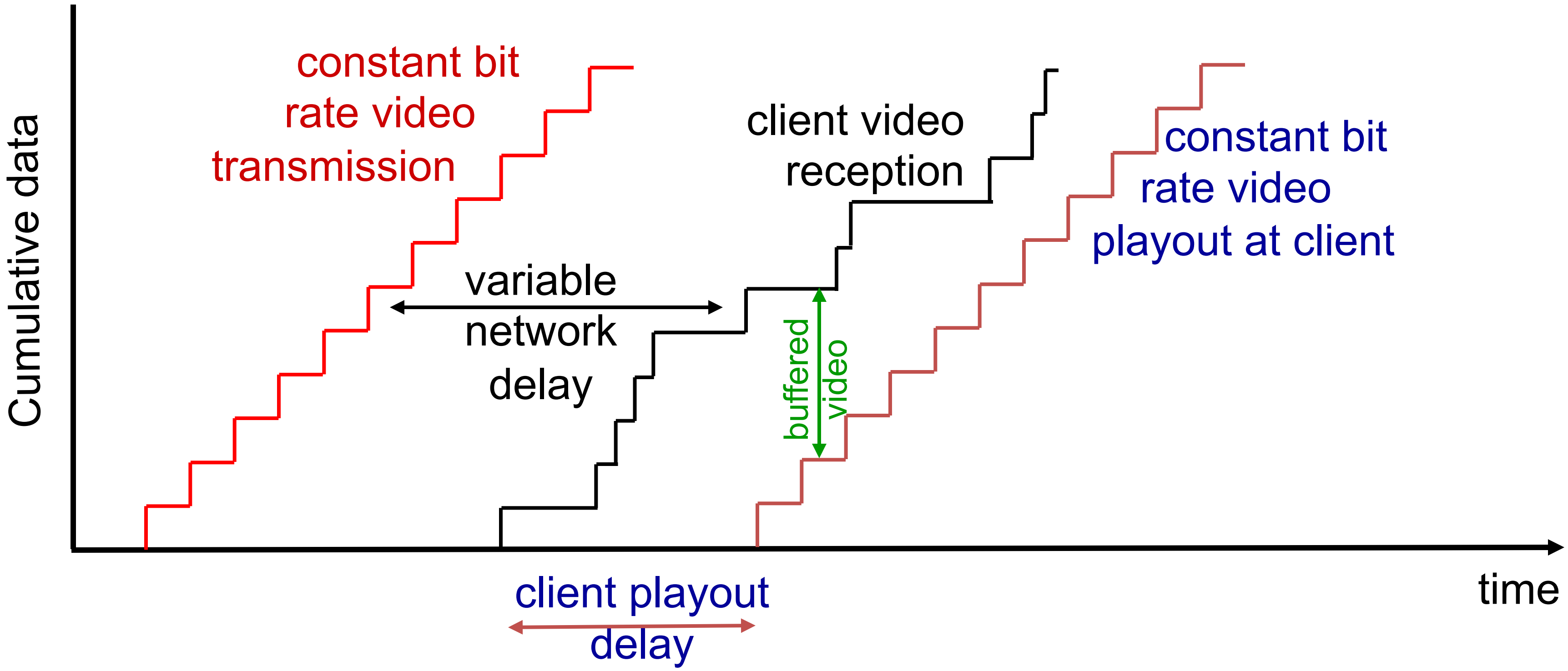


streaming: at this time, client playing out early part of video, while server still sending later part of video

Streaming stored video: challenges

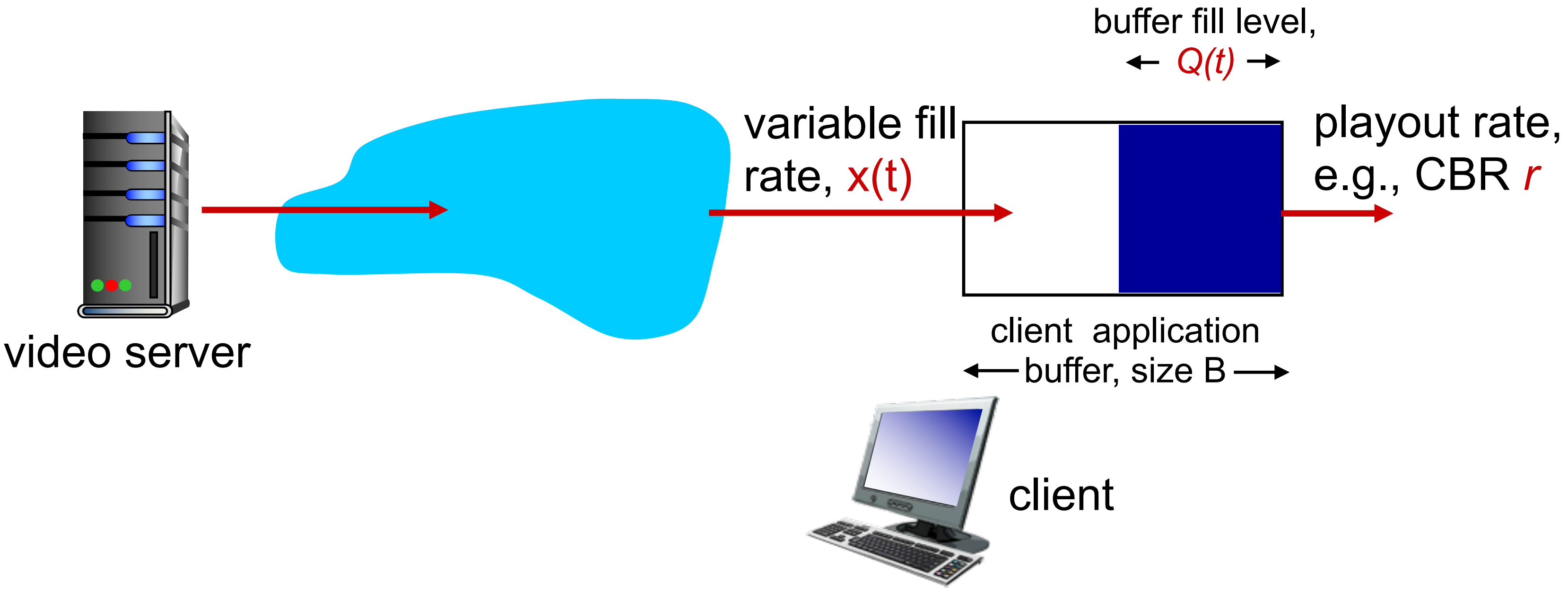
- continuous playout constraint: once client playout begins, playback must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video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

Streaming stored video: revisited

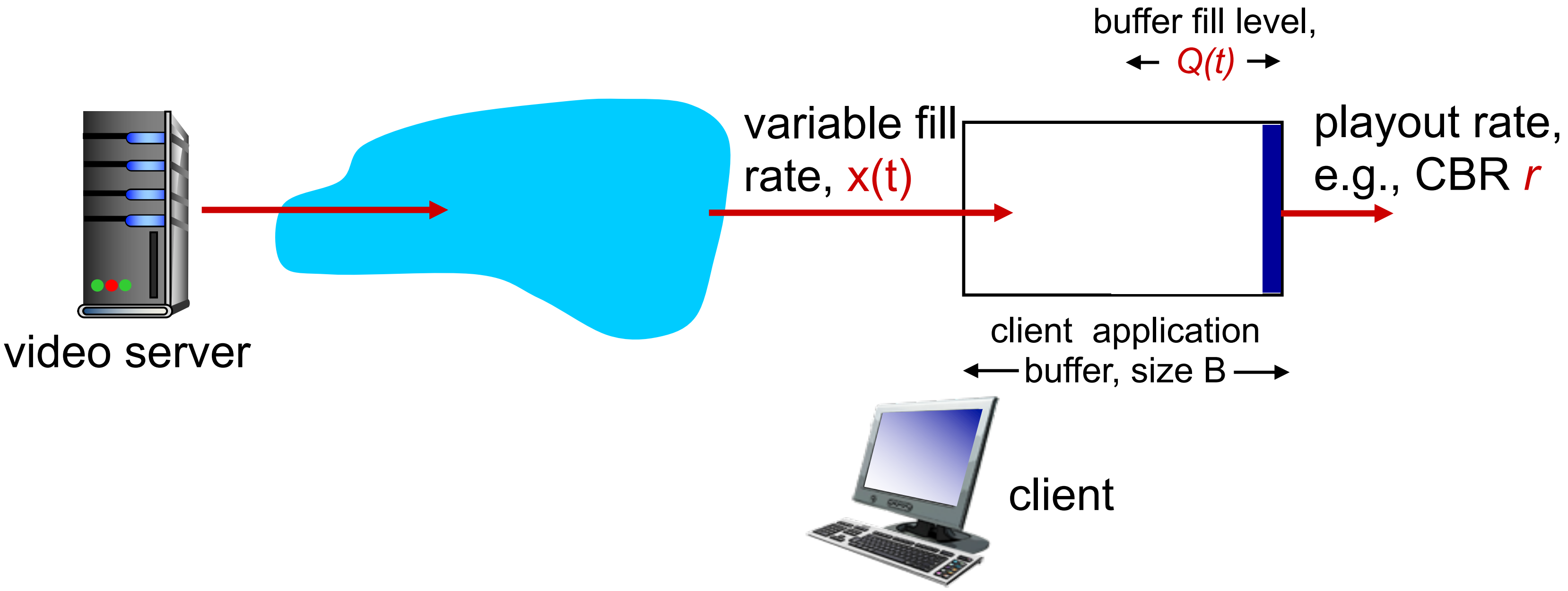


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout

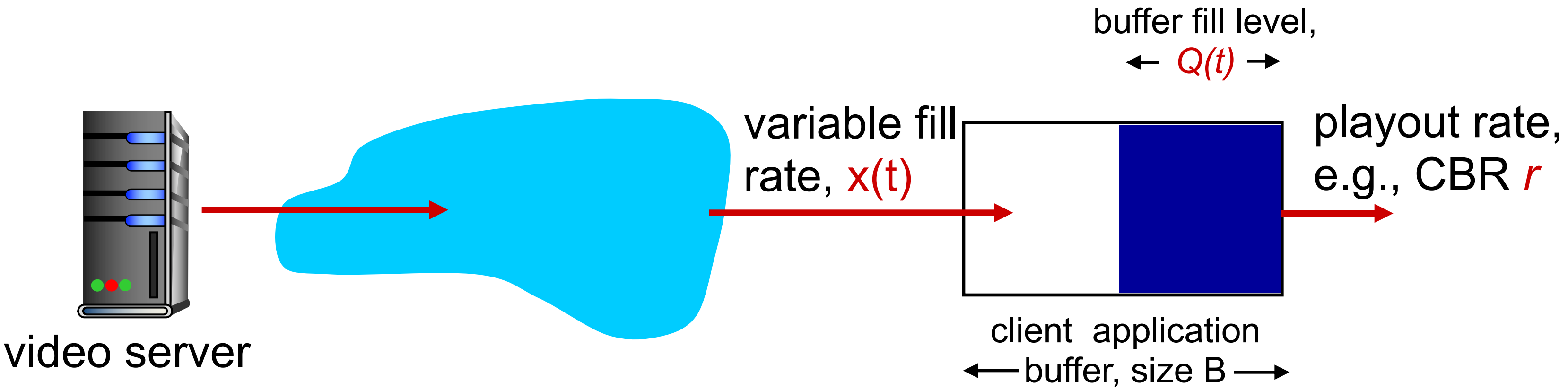


Client-side buffering, playout



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):

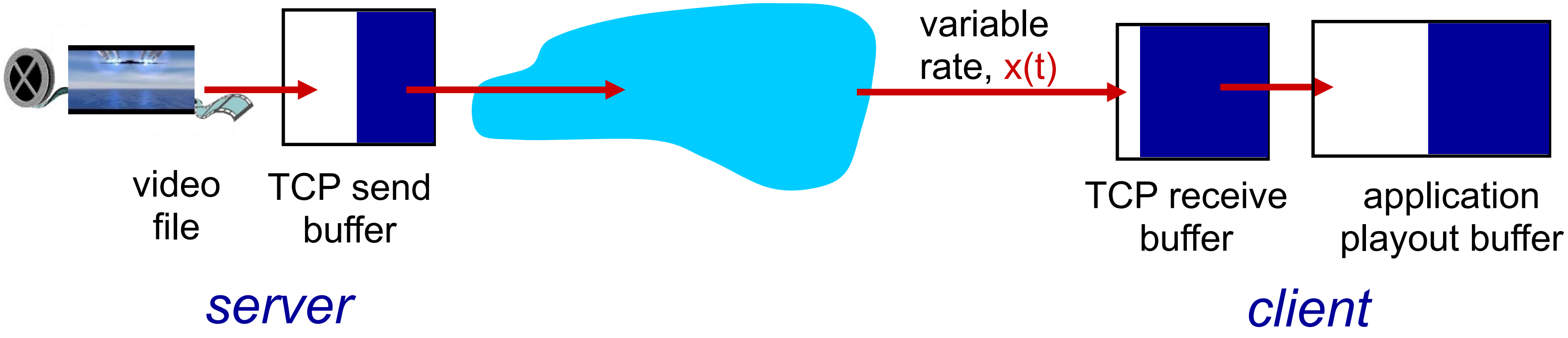
- $\bar{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x} > r$: buffer will not empty, provided initial playout delay is large enough to absorb variability in x(t)
  - *initial playout delay tradeoff:* buffer starvation less likely with larger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

Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = playback rate
  - send rate can be oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter
- error recovery: application-level, time permitting
- Real-time Transport Protocol (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
  - Other adaptive solutions: Apple's HTTP Live Streaming (HLS) solution, Adobe Systems HTTP Dynamic Streaming, Microsoft Smooth Streaming
- *Server:*
  - encodes video file into multiple versions
  - each version stored, encoded at a different rate
  - *manifest file:* provides URLs for different versions
- *Client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate (highest quality version) sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at the time)

Streaming multimedia: DASH

- *"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)
- Can leverage web and its existing infrastructure (proxy, caching…)

Outline

Multimedia networking applications

Streaming *stored* video

Voice-over-IP

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

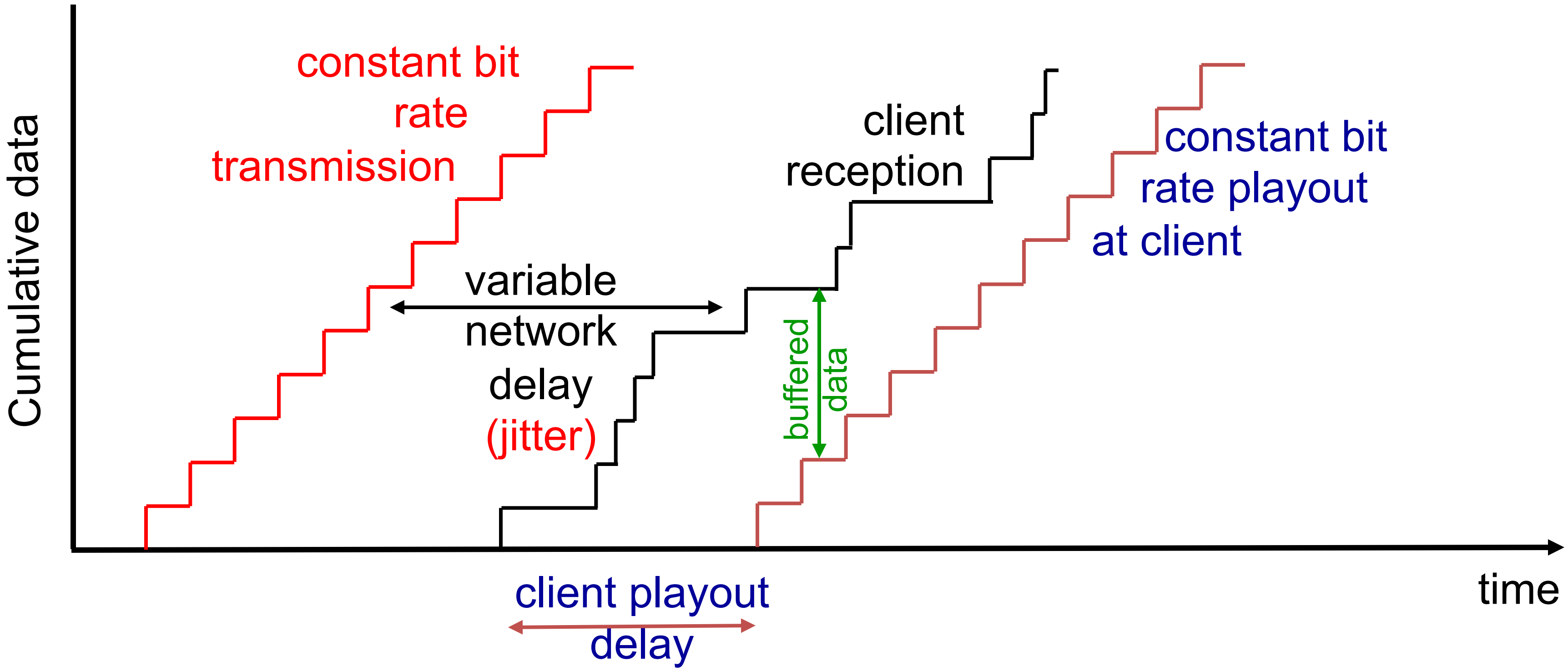
VoIP characteristics

- 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: vary due to 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 up to 10% may be tolerated

Delay jitter



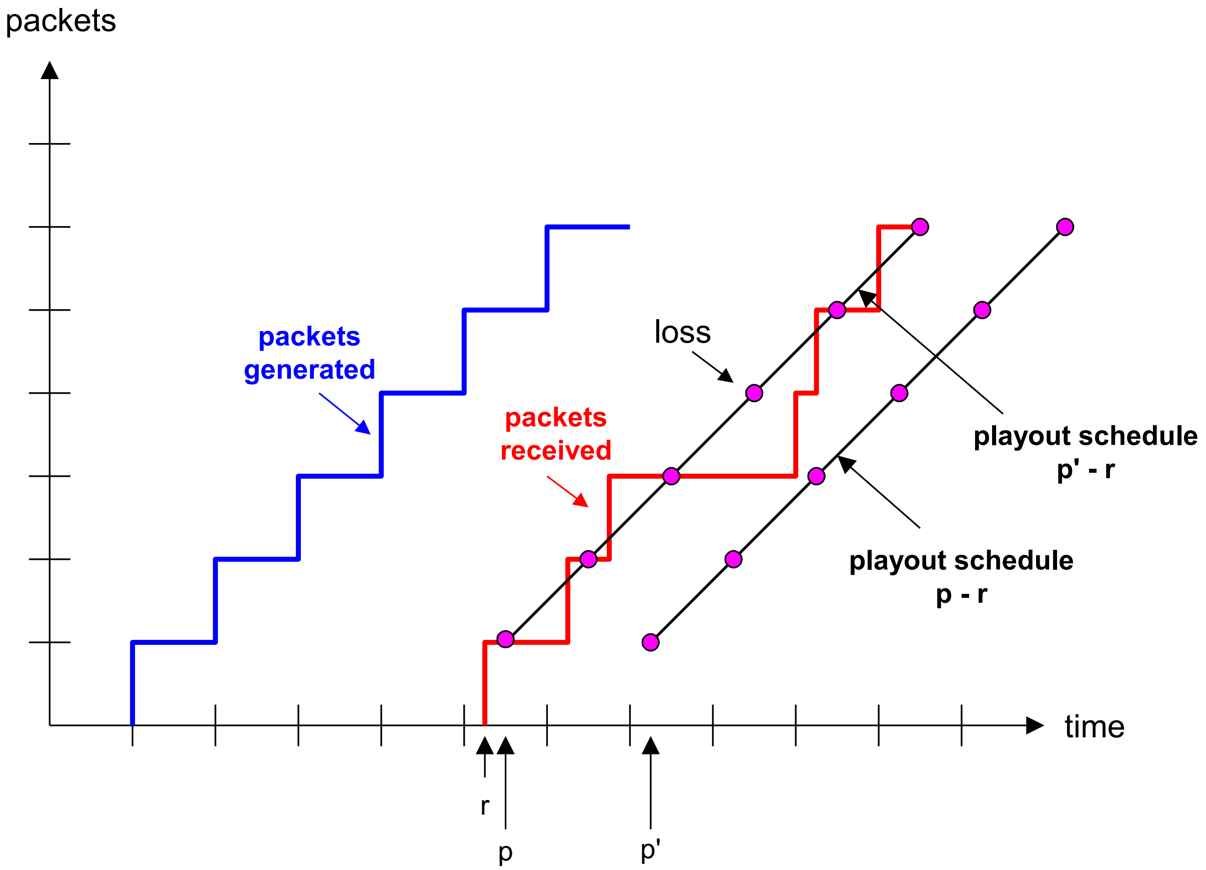
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

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):

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

delay estimate after ith packet

small constant, e.g. 0.1

$$\underbrace{\textit{time received} \;-\; \textit{time sent (timestamp)}}_{\textit{measured delay of ith packet}}$$

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:

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

- 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

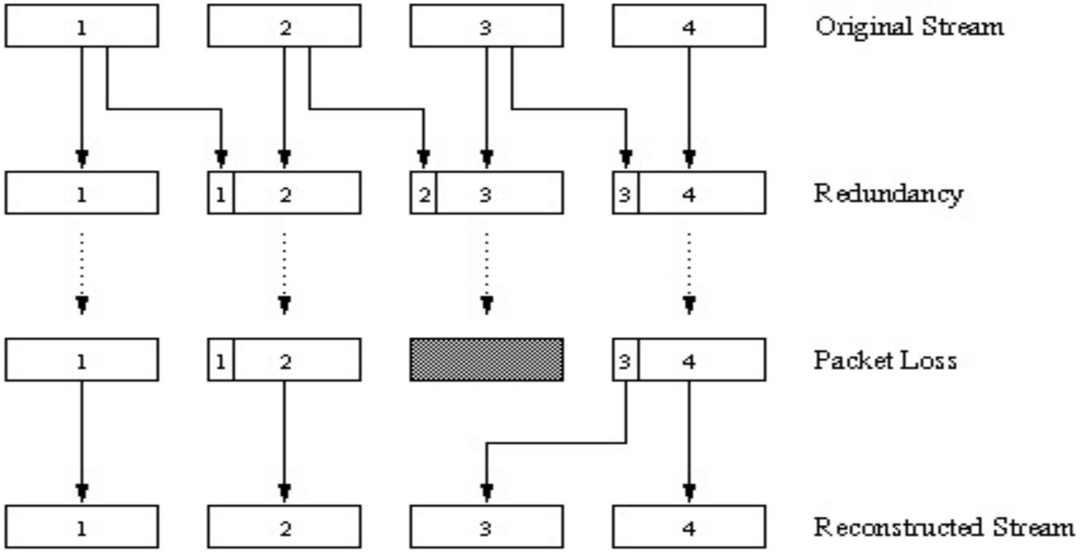
simple FEC

- 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
- Note: playout delay increases with *n*

VoIP: recovery from packet loss (2)

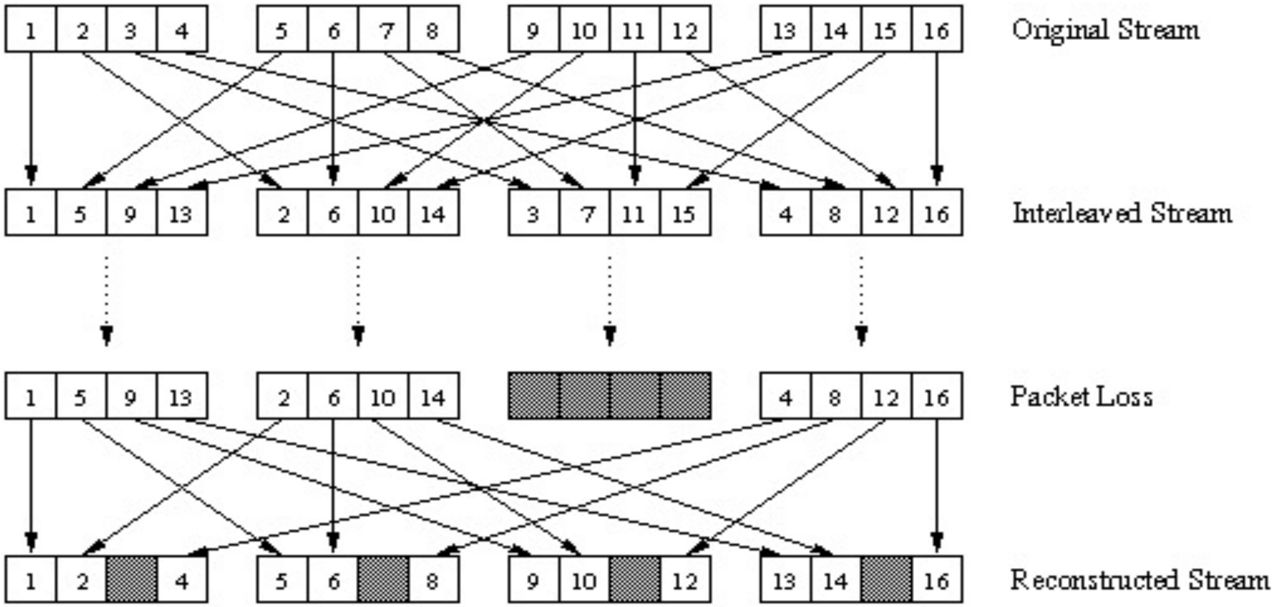
another FEC scheme:

- "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