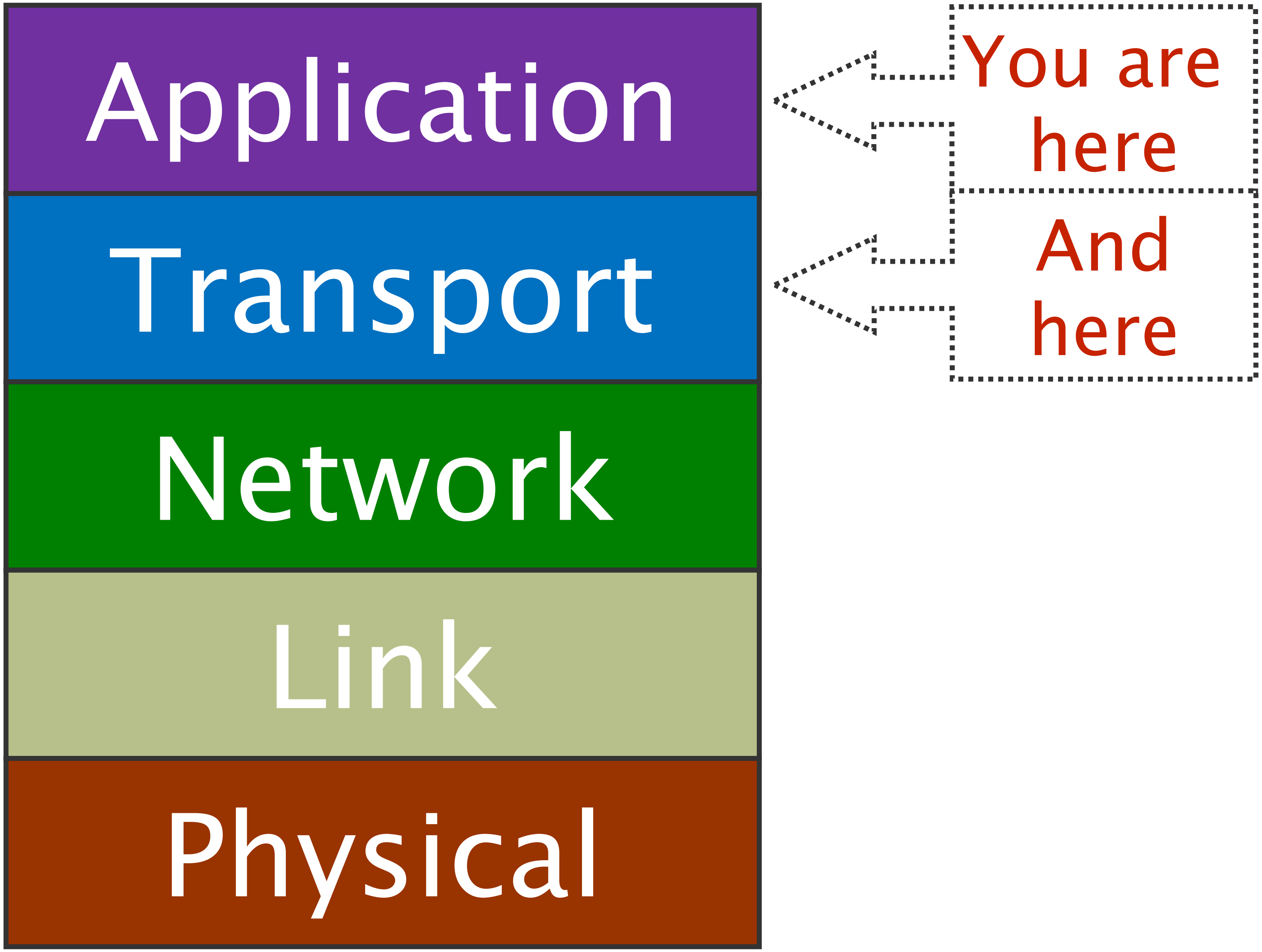
# CS2105

## An *Awesome* Introduction to Computer Networks

### Lecture 10: Multimedia Networking

NUS National University of Singapore | Department of Computer Science
School of Computing



Application

Transport

Network

Link

Physical

You are here

And here

# Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

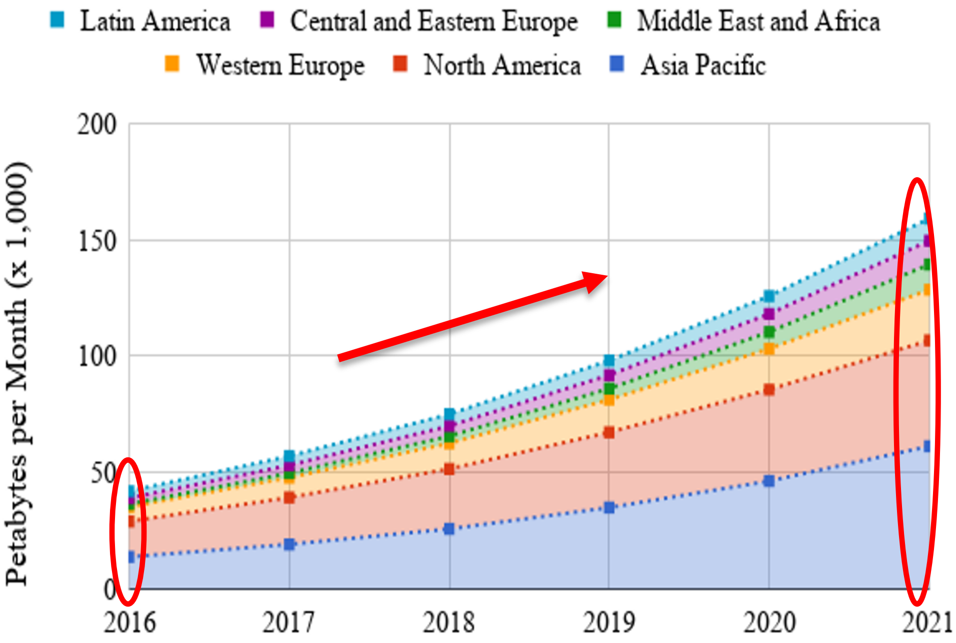
9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

# Why learn about multimedia networking?

## Video is predominant on the Internet!

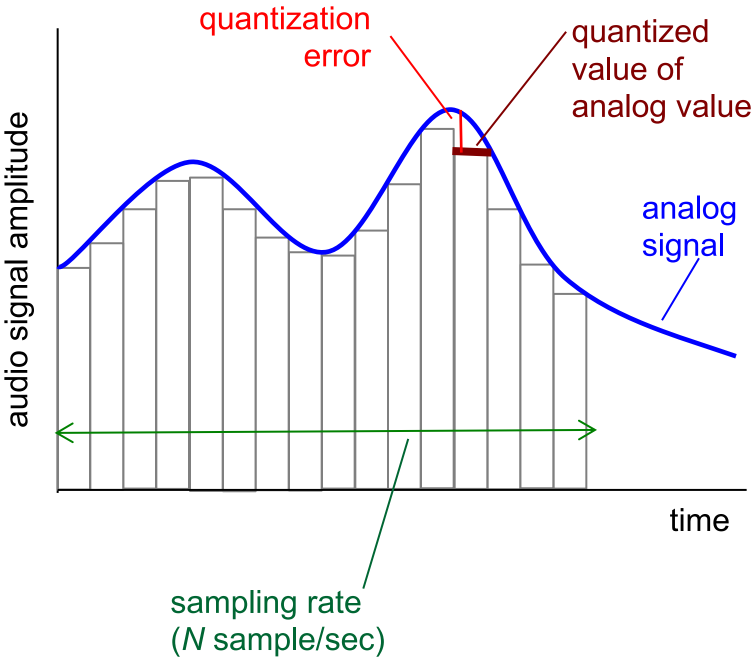
- Cisco reported in their annual VNI:
  - In 2016, **67%** of the global Internet traffic was video, with a projection to reach **80%** by 2021
- Popular services:
  - YouTube (14.0%)
  - Netflix (34.9%)
  - Amazon Video (2.6%)
  - Hulu (1.4%)
- All these are delivered as OTT



PhD Defense– Enabling Optimizations of Video Delivery in HAS; Abdelhak Bentaleb

# 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
  - $2^{16}$=65,536 for CD


quantization error
quantized value of analog value
analog signal
audio signal amplitude
time
sampling rate (*N* sample/sec)

- *Nyquist-Shannon sampling theorem:*

$$f_s \geq d \times 2$$

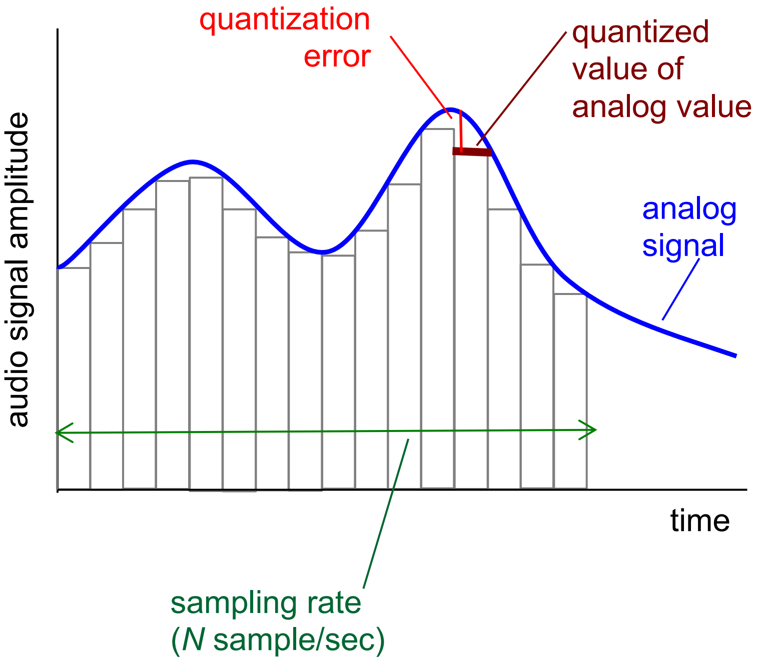
- $f_s$: sampling frequency
- $d$: highest signal frequency

# Multimedia: audio

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

## example rates

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



- ADC: analog-to-digital converter
- DAC: digital-to-analog converter

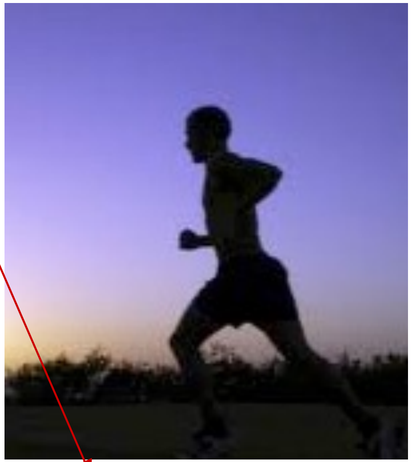
# Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 30 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*

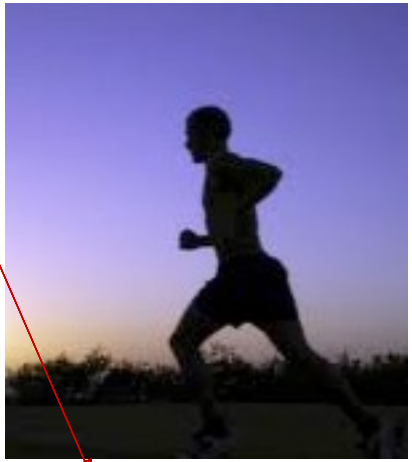
# Multimedia: video

- 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/H.264 (often used in Internet, < 2 Mbps)
  - H.265
  - 4K video < 85 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

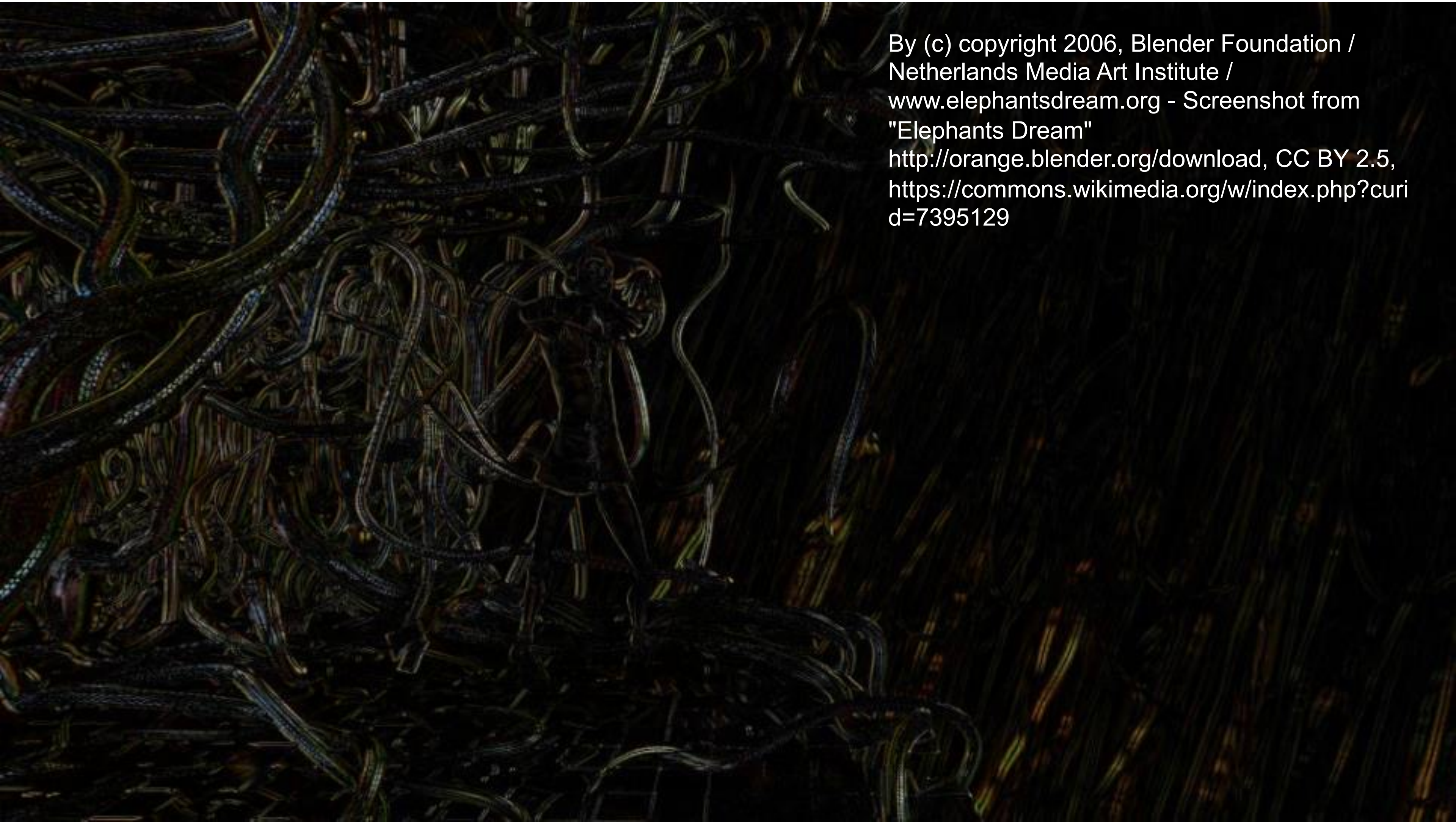


frame *i+1*

# Video: original frame *i*

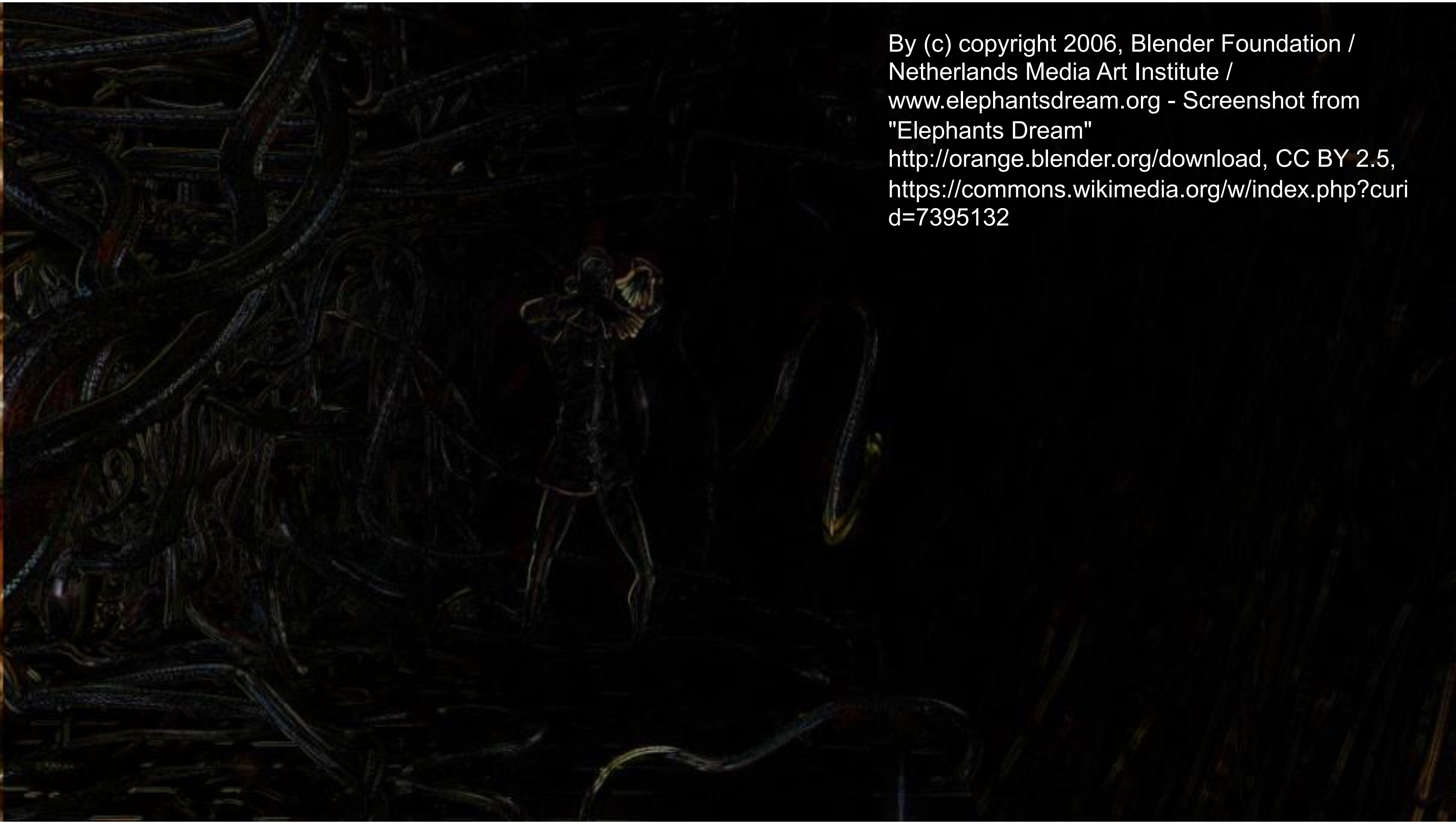
By (c) copyright 2006, Blender Foundation / Netherlands Media Art Institute / www.elephantsdream.org - Screenshot from "Elephants Dream" http://orange.blender.org/download, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=7395123

# Difference between 2 frames



By (c) copyright 2006, Blender Foundation / Netherlands Media Art Institute / www.elephantsdream.org - Screenshot from "Elephants Dream" http://orange.blender.org/download, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=7395129

# Motion compensated difference



By (c) copyright 2006, Blender Foundation / Netherlands Media Art Institute / www.elephantsdream.org - Screenshot from "Elephants Dream" http://orange.blender.org/download, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=7395132

# 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 ("two-way live")* voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
  - e.g., Skype, Zoom
- *streaming live ("one-way live")* audio, video
  - e.g., live sporting event (soccer, football)

# Multimedia networking: outline

9.1 multimedia networking applications

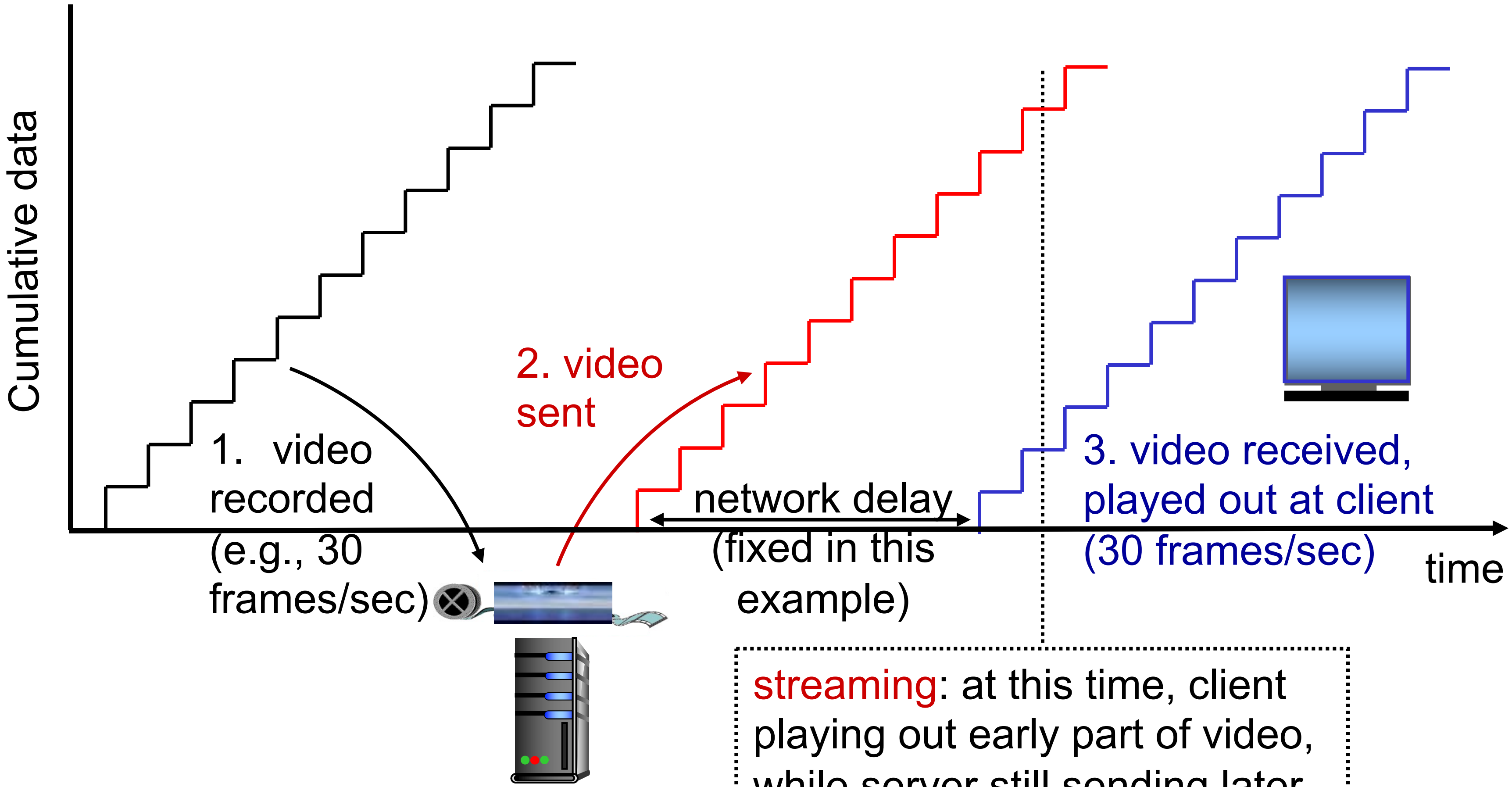
9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

# Streaming stored video:

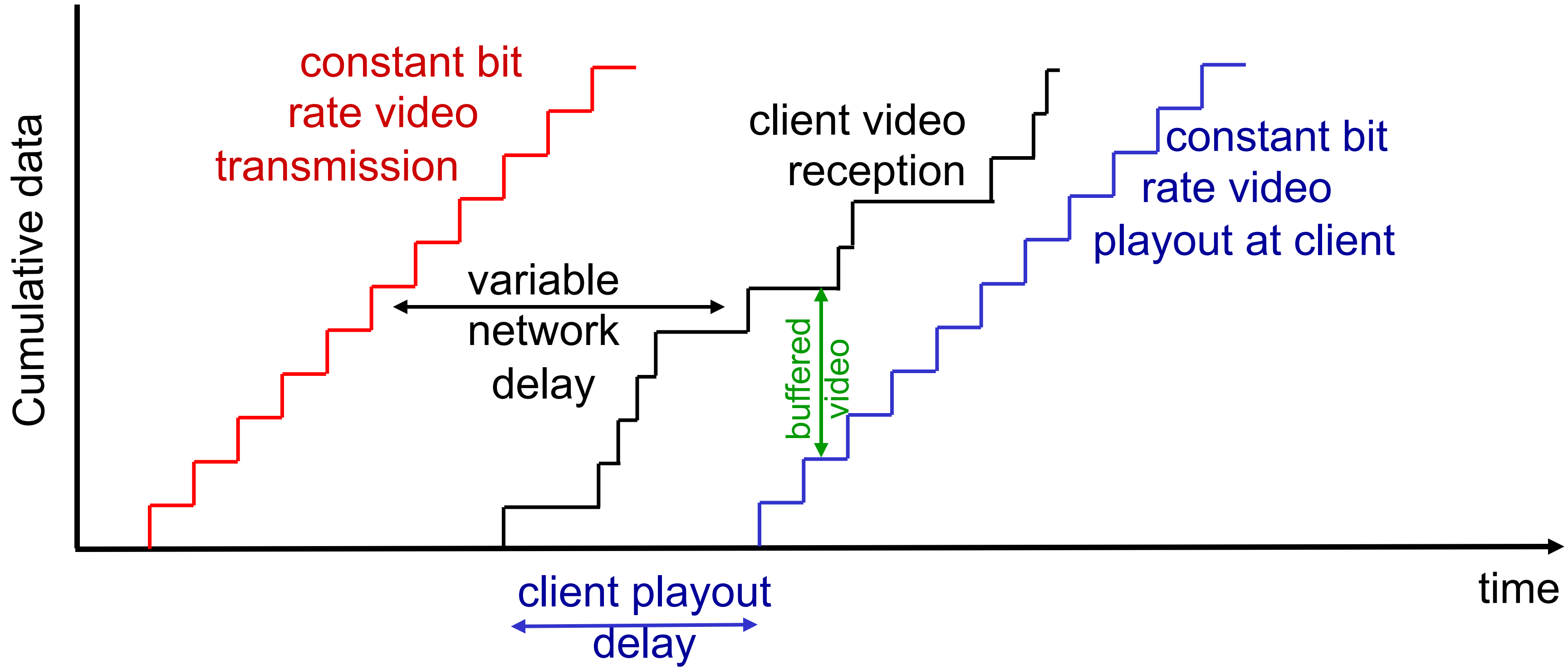

Cumulative data
1. video recorded (e.g., 30 frames/sec)
2. video sent
network delay (fixed in this example)
3. video received, played out at client (30 frames/sec)
time

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

# Streaming stored video: challenges

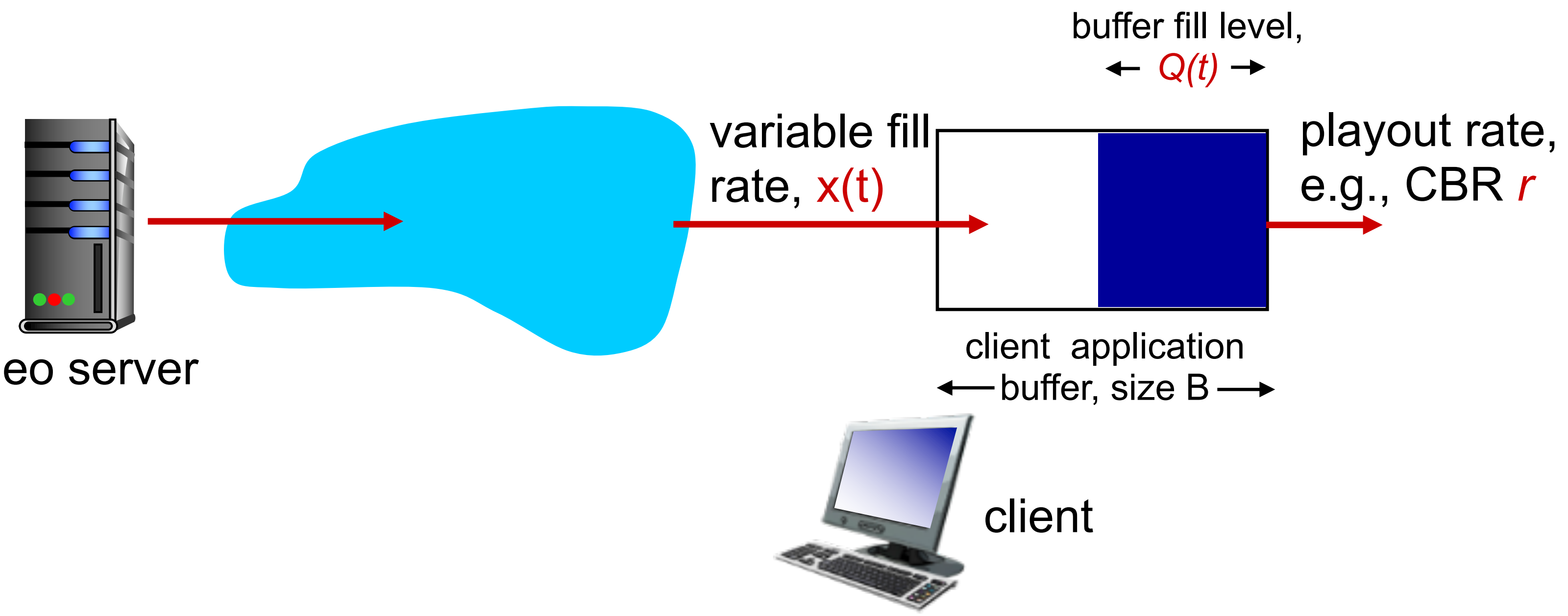
- continuous playout constraint: once client playout begins, playback must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video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

# Streaming stored video: revisited

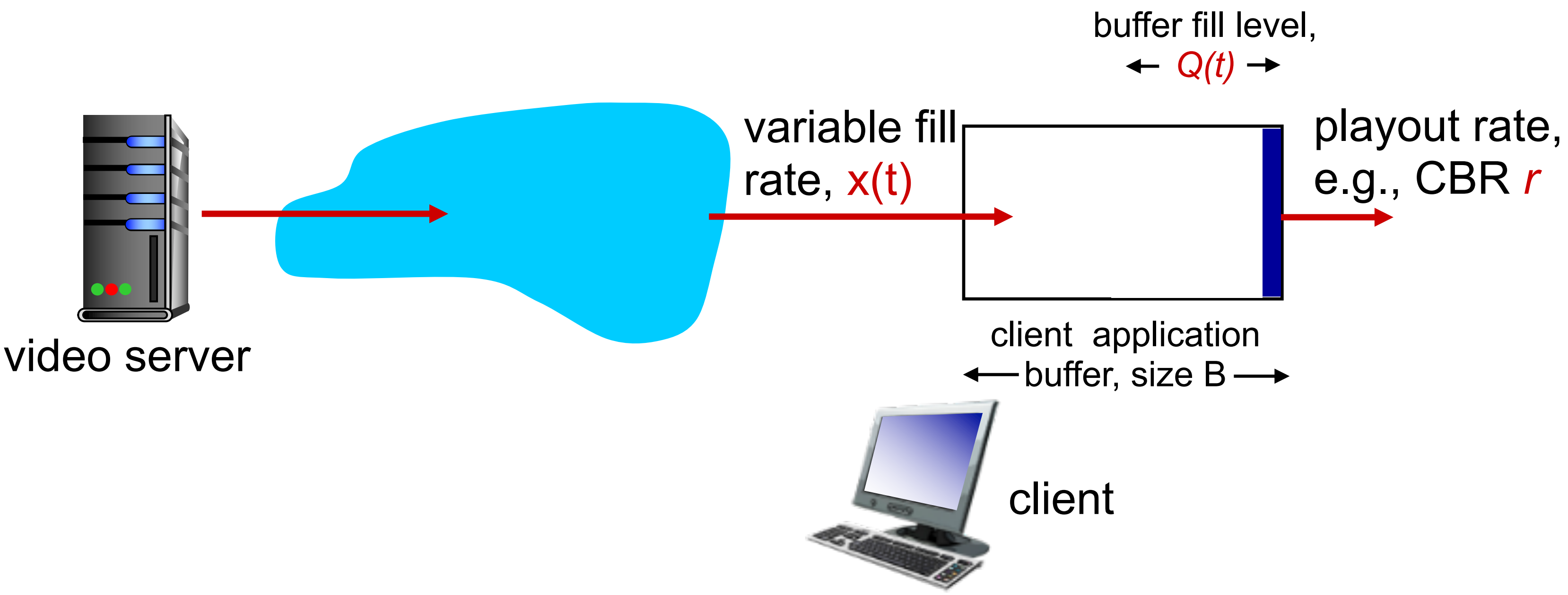


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

# Client-side buffering, playout

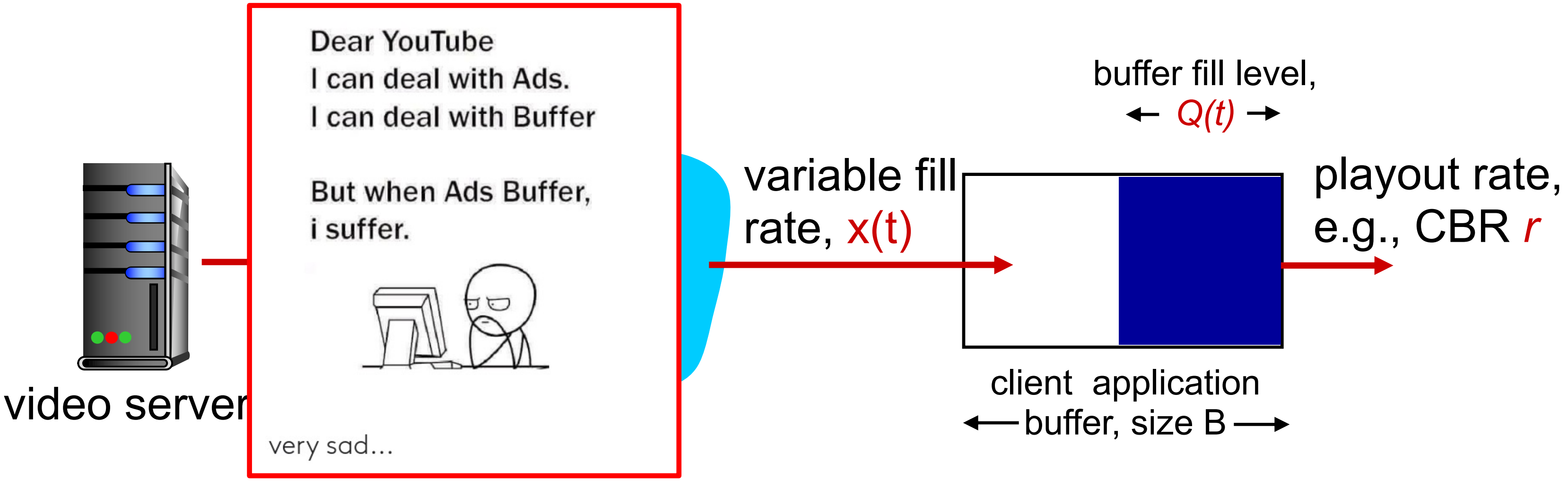


# Client-side buffering, playout



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):*

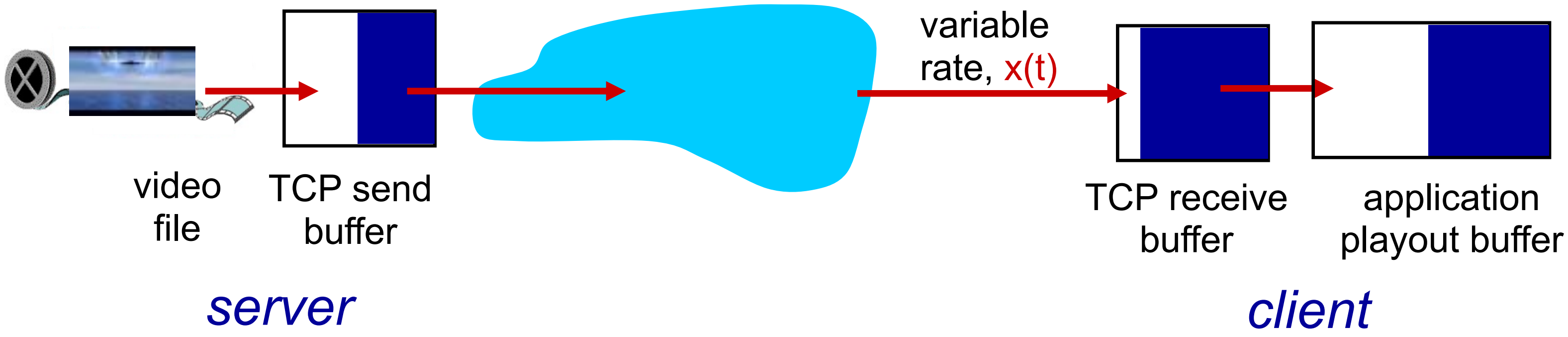
- $\bar{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x} > r$: buffer will not empty, provided initial playout delay is large enough to absorb variability in x(t)
  - *initial playout delay tradeoff:* buffer starvation less likely with larger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

# Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate, ☞ push-based streaming (*server push*)
  - 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 types
- UDP may *not* go through firewalls

# Streaming multimedia: HTTP

- multimedia file retrieved via HTTP GET,
  ☞ pull-based streaming (*client pull*)
- 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

# Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

# 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

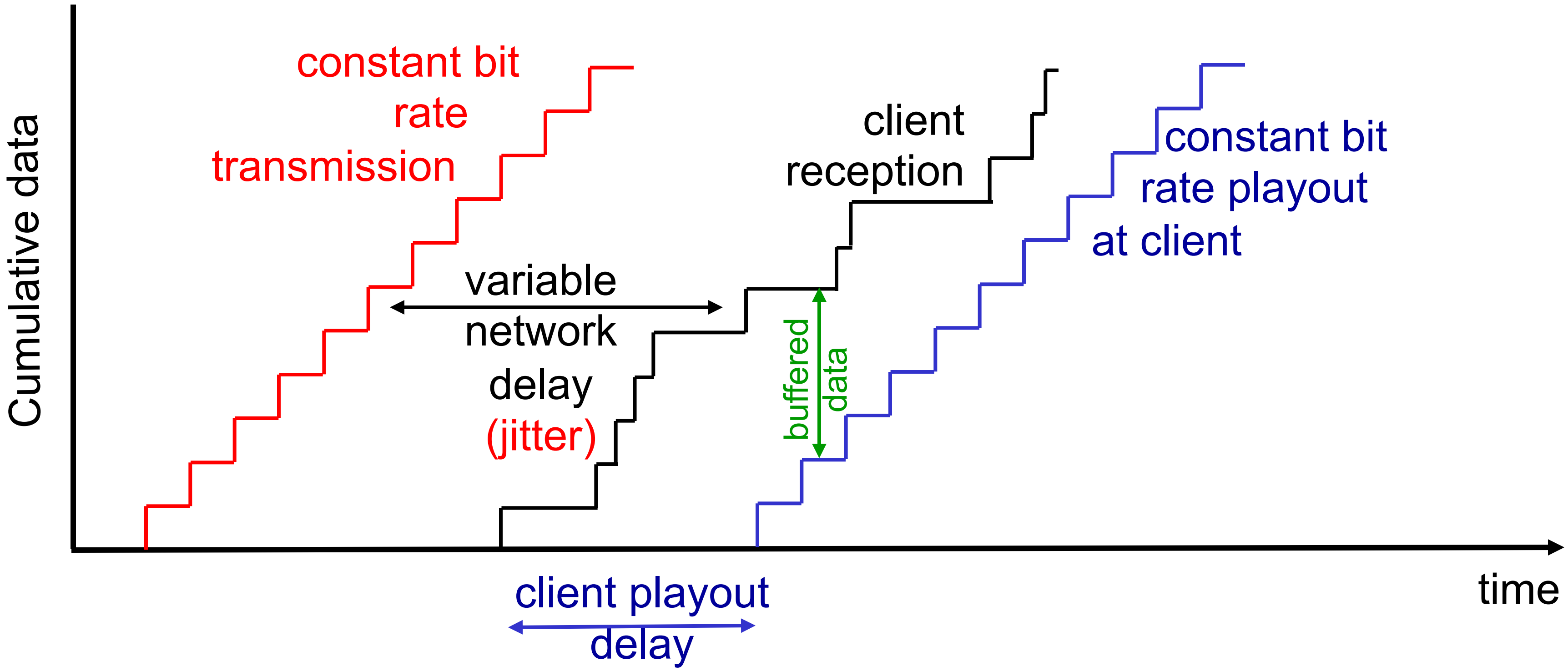
# VoIP characteristics

- 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


constant bit rate transmission
client reception
constant bit rate playout at client
variable network delay (jitter)
buffered data
Cumulative data
time
client playout delay

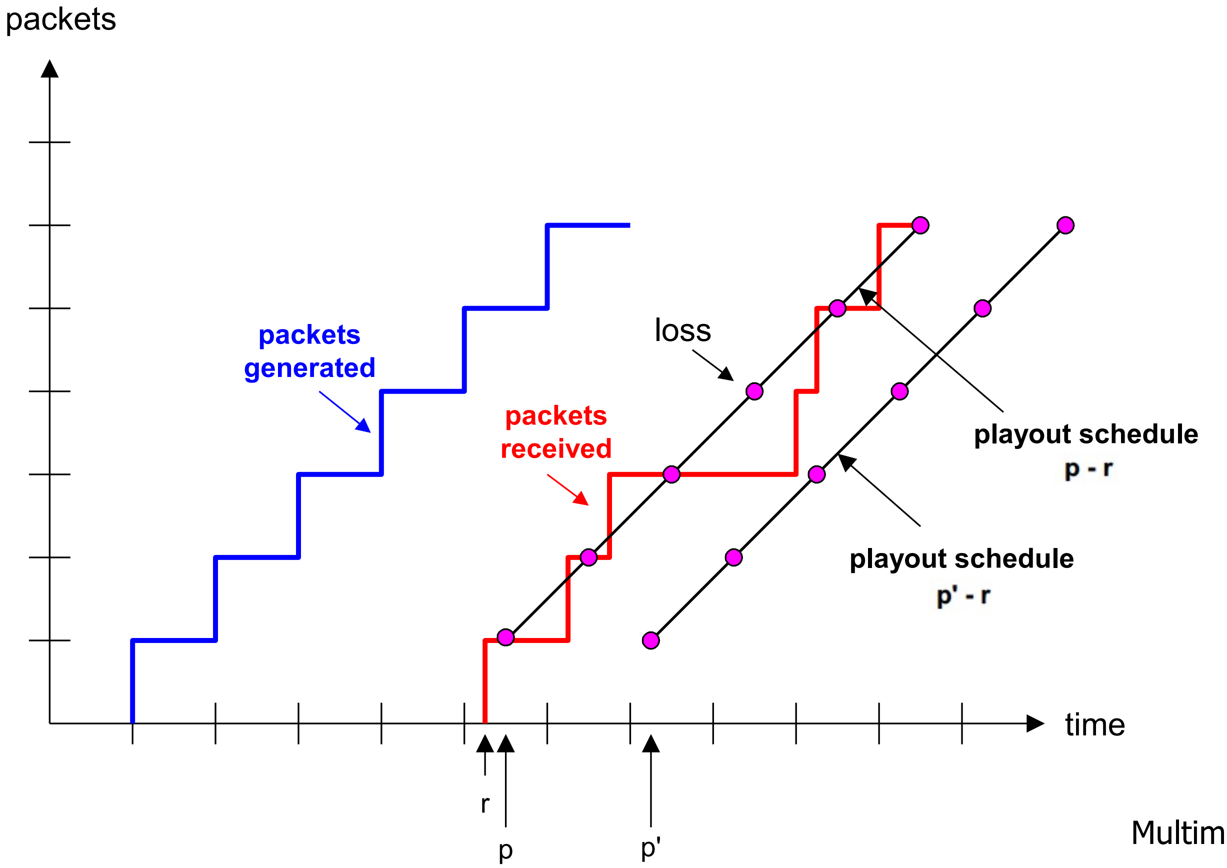
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

# 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 (I)

- *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):

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

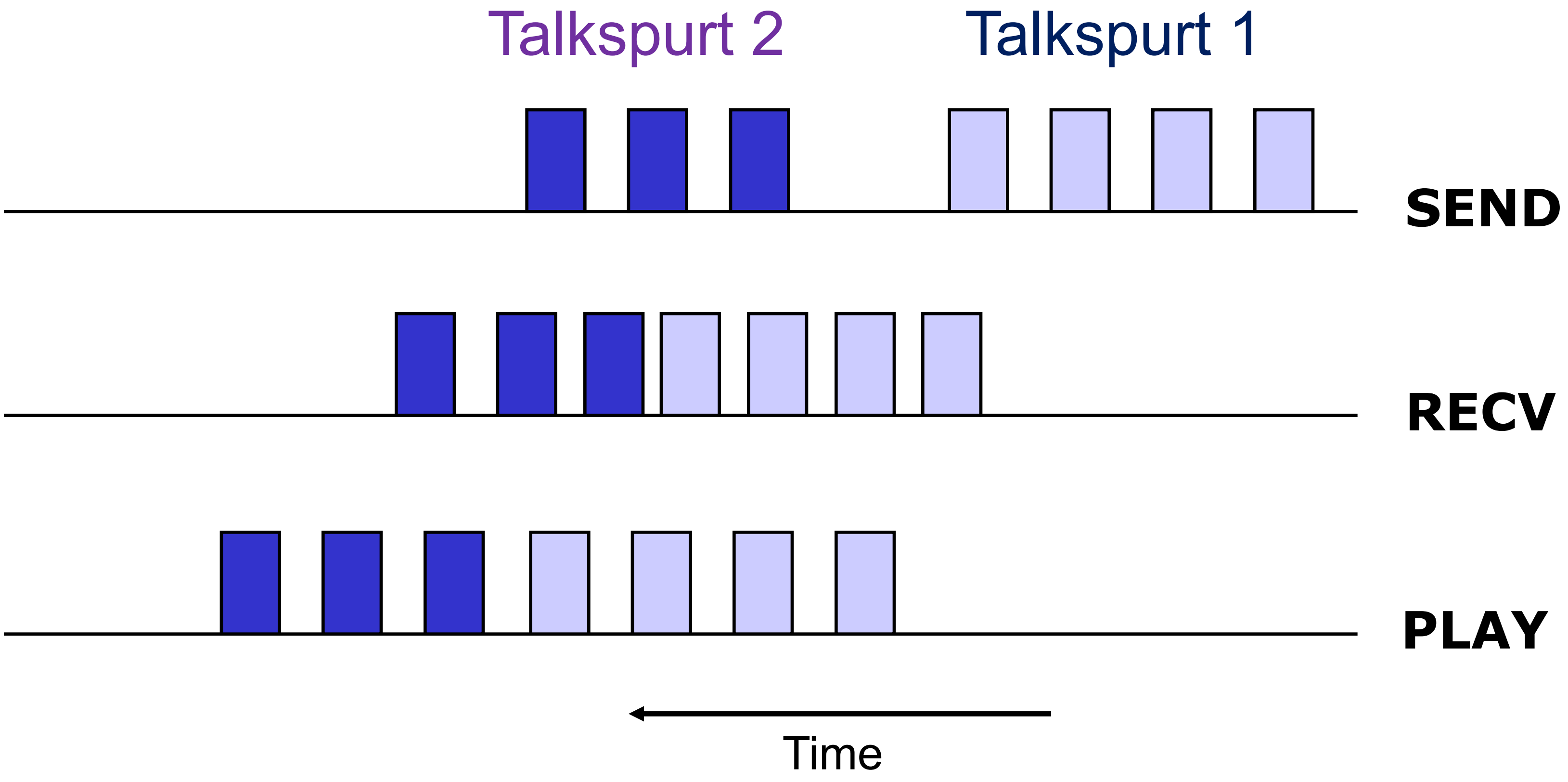
*delay estimate after ith packet*

*small constant, e.g. 0.1*

*time received - time sent (timestamp)*

*measured delay of ith packet*

# Adaptive playout delay (2)



# Adaptive playout delay (3)

- also useful to estimate average deviation of delay, $v_i$ :

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

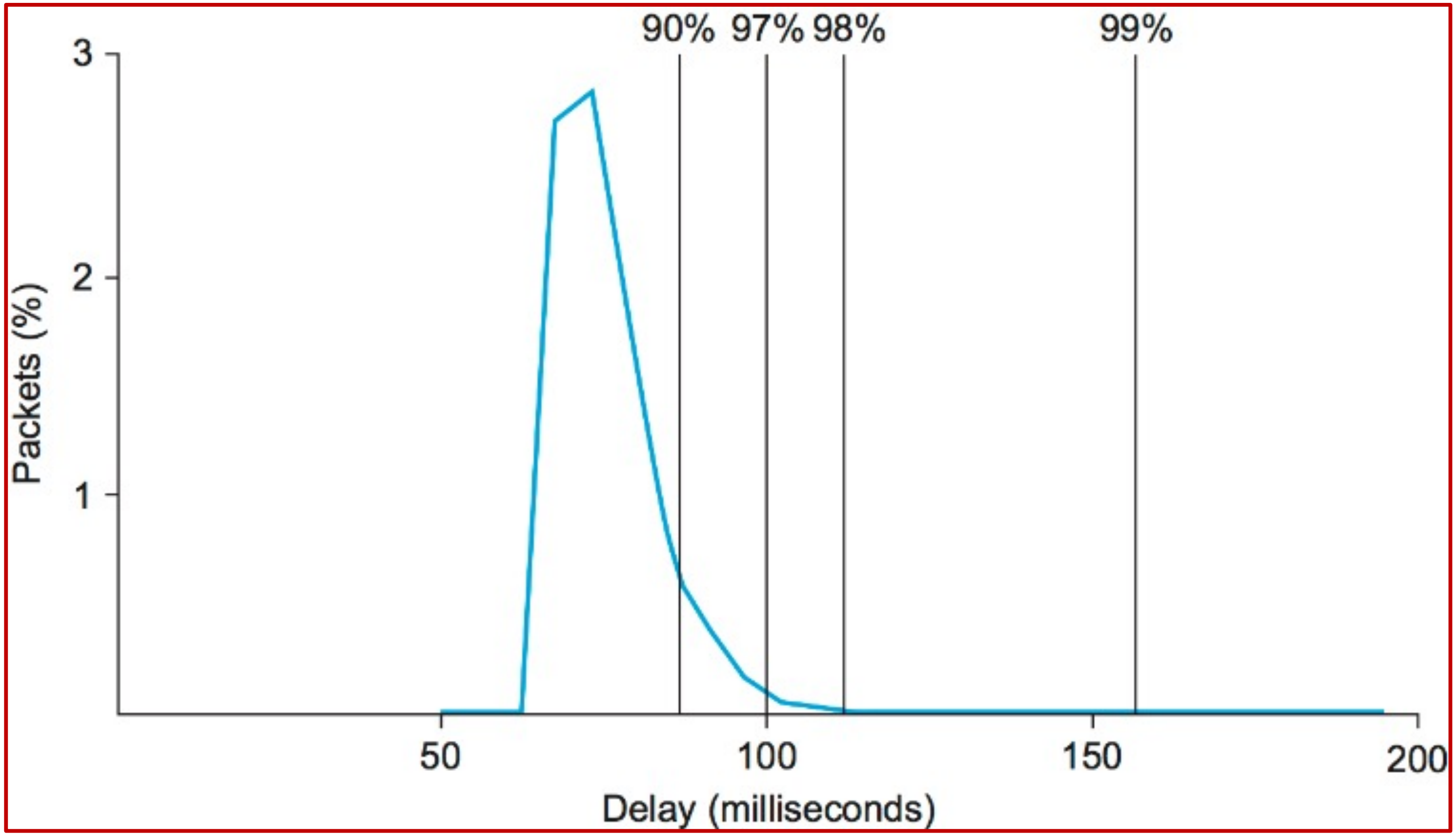
- estimates $d_i$, $v_i$ calculated for every received packet, but used only at start of talk spurt

- 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 (4)



# 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)

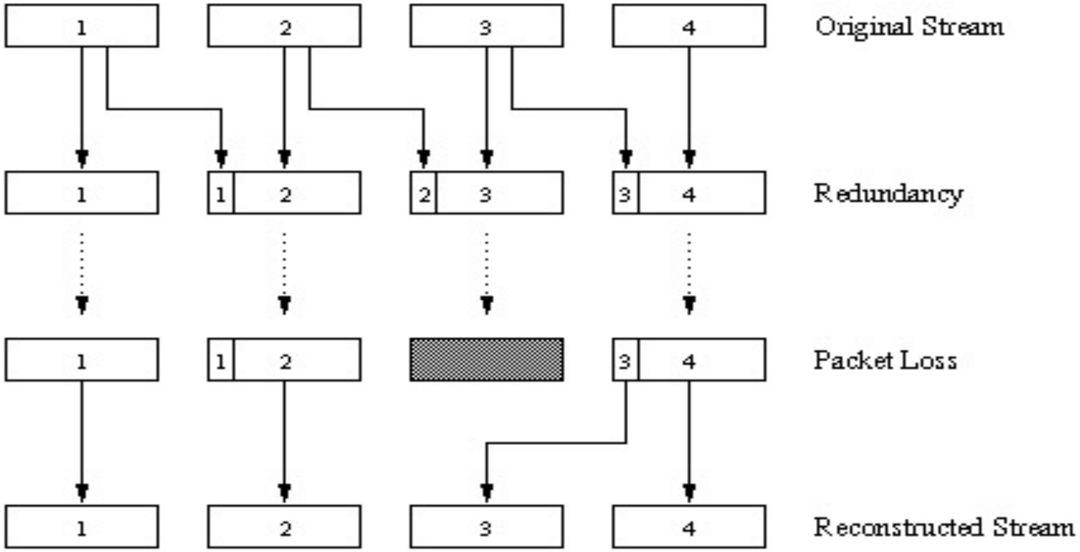
*simple FEC*

- 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)

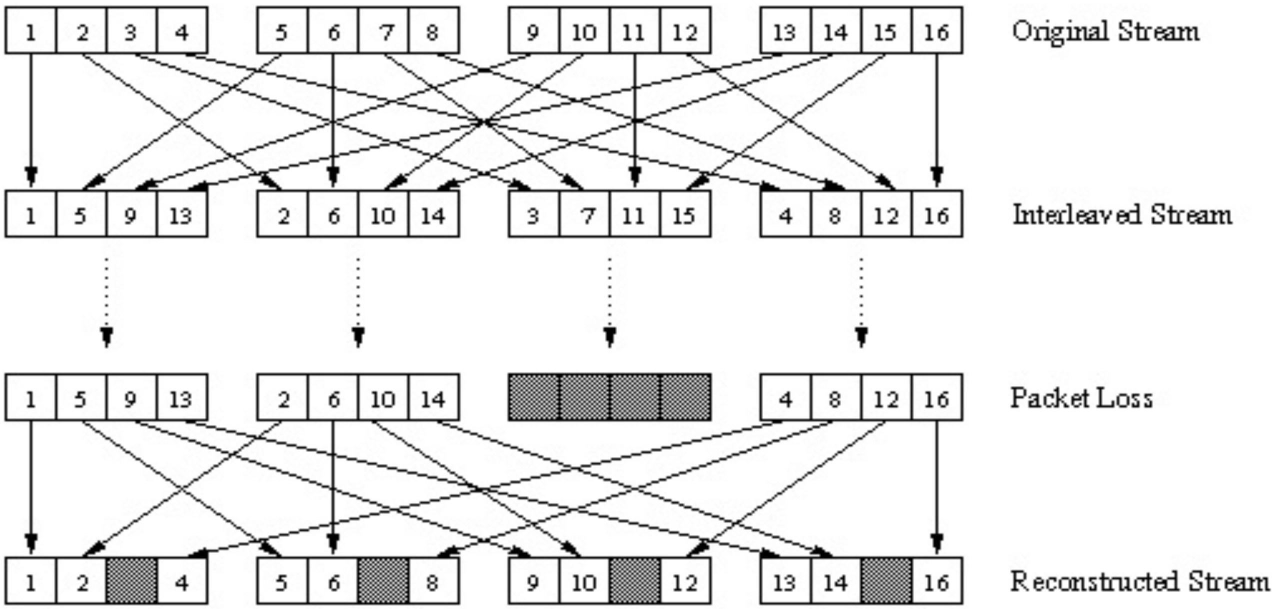
another FEC scheme:

- "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

# Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications: RTP, SIP

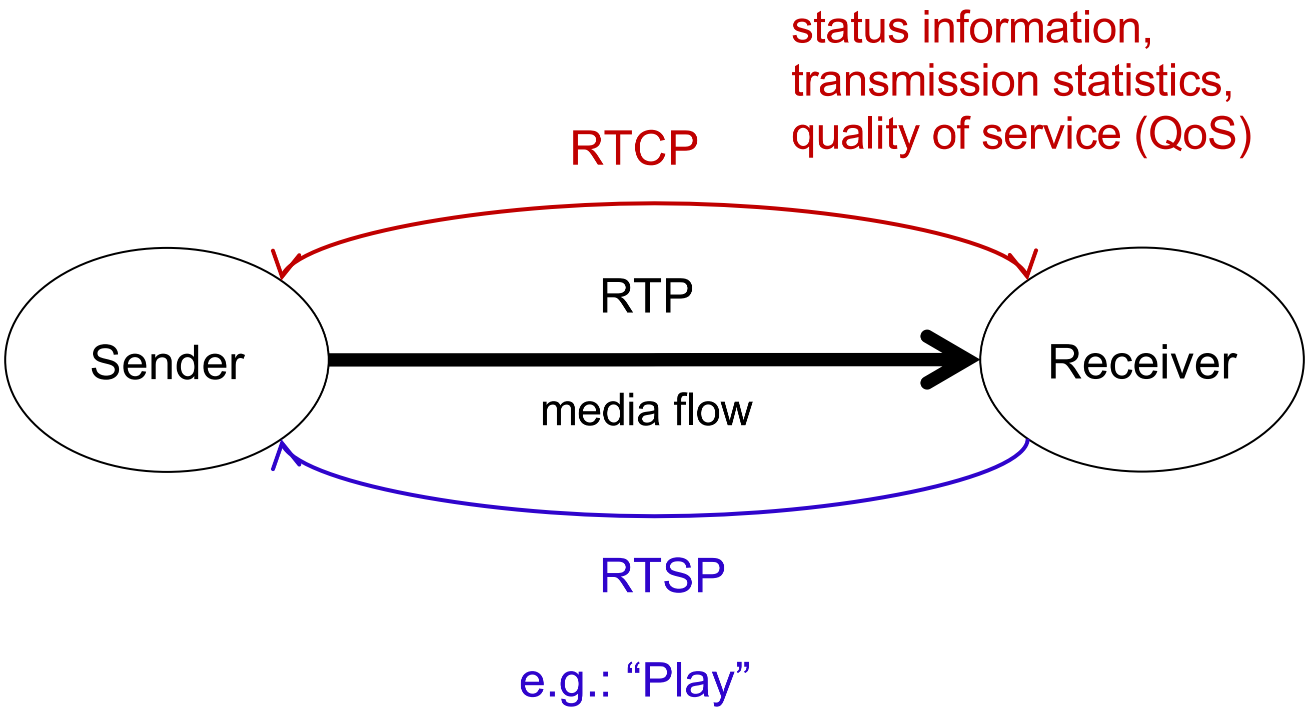
9.5 dynamic adaptive streaming over HTTP (DASH)

# 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

# Real-Time Protocol **Suite**

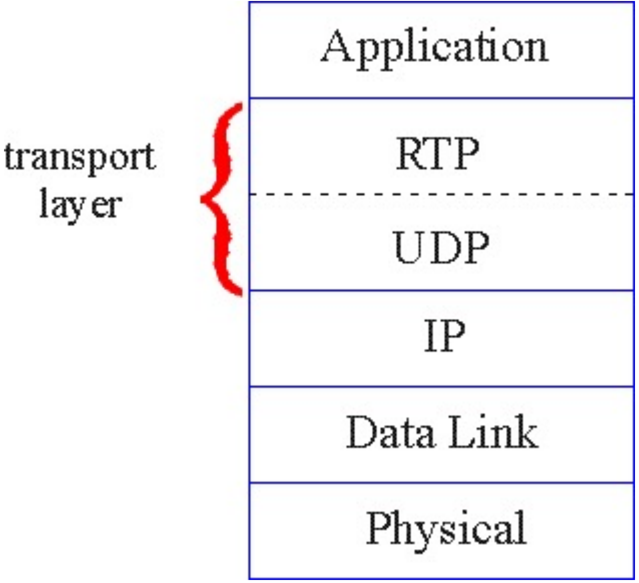
- Flow diagram: RTP, RTCP, RTSP



# 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* |

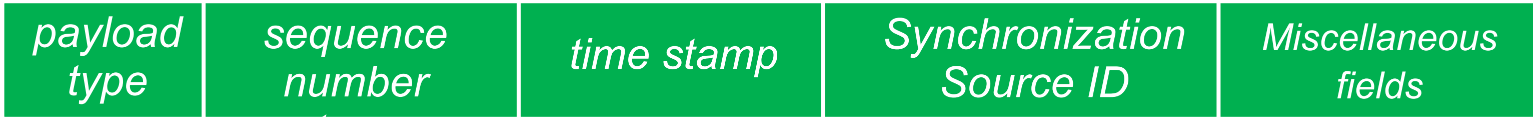
payload type (7 bits): indicates type of encoding currently being used. If sender changes encoding during call, sender informs receiver via payload type field

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



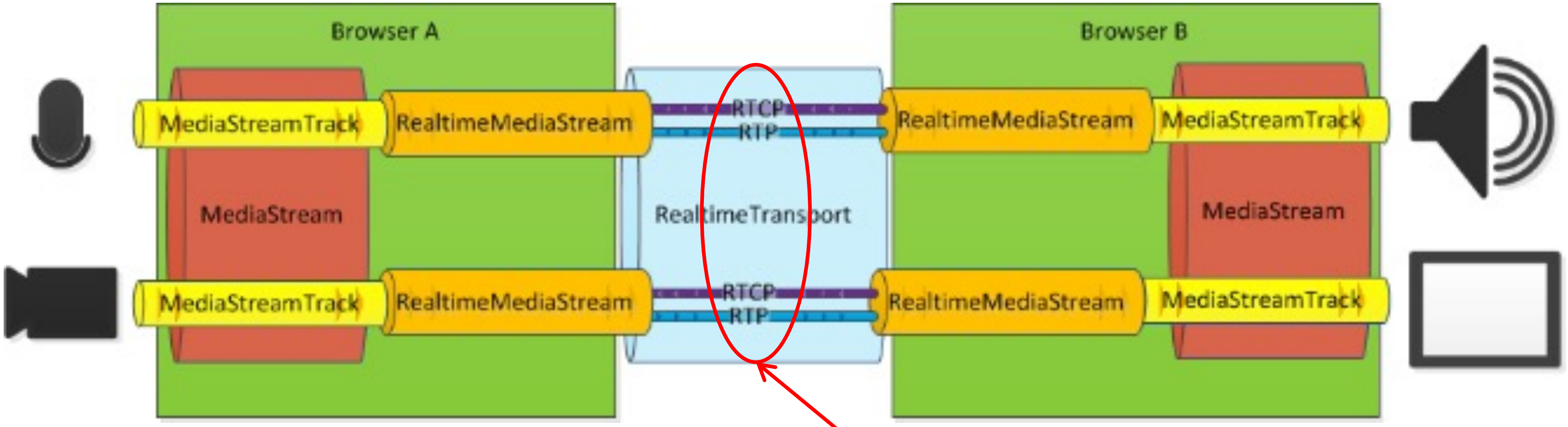
| payload type | sequence number | time stamp | Synchronization Source ID | Miscellaneous fields |
|---|---|---|---|---|

- *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

# WebRTC (www.webrtc.org)

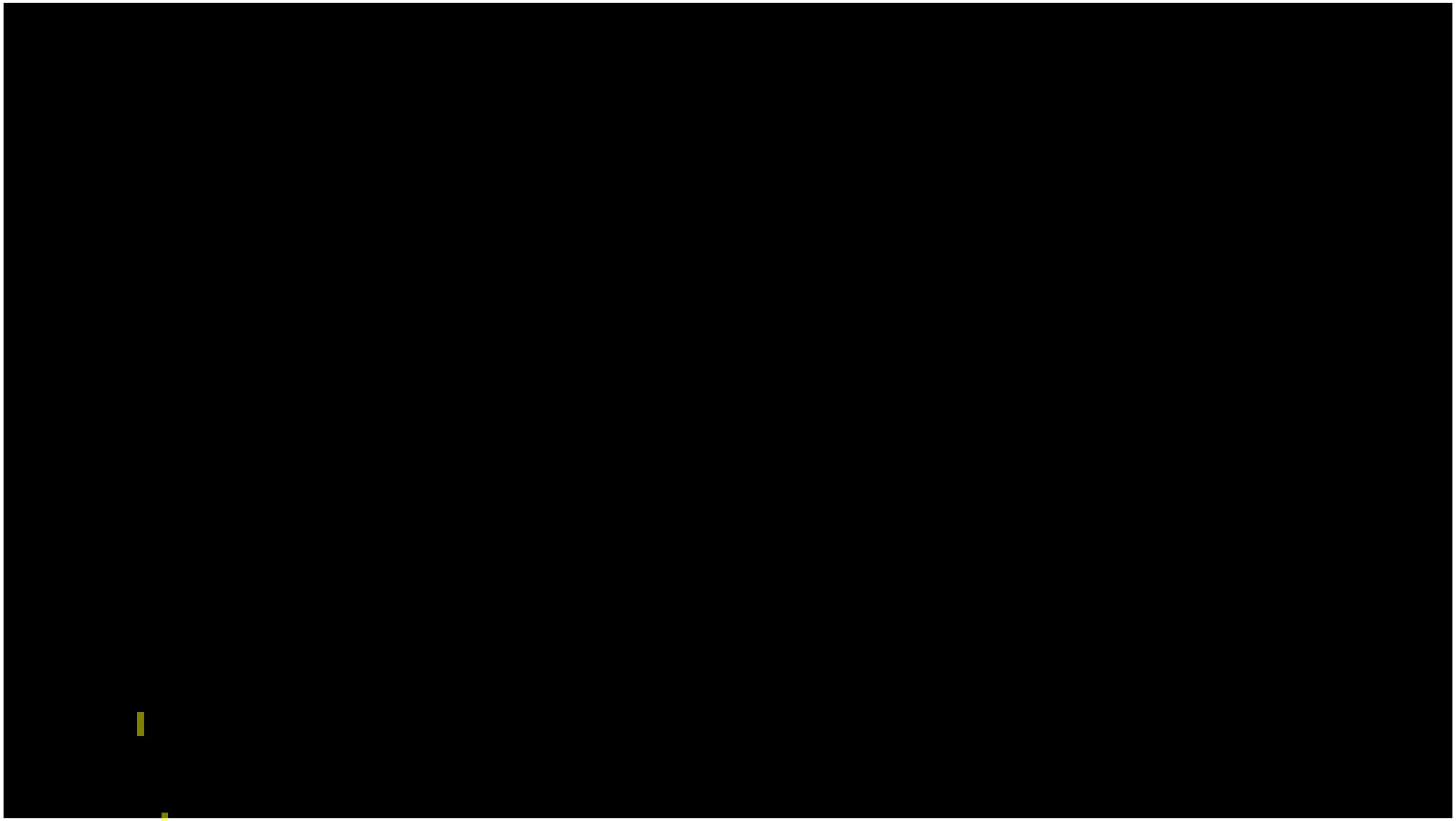
- Web browsers with Real-Time Communications (RTC) capabilities via simple JavaScript APIs.
- Pipeline for video conferencing in WebRTC (only one-way shown):



RTP/RTCP

© Muaz Khan

# WebRTC (Demo)



# SIP: Session Initiation Protocol [RFC 3261]

*long-term vision:*

- all telephone calls, video conference calls take place over Internet
- people identified by names or e-mail addresses, rather than by phone numbers
- can reach callee *(if callee so desires)*, no matter where callee roams, no matter what IP device callee is currently using

# SIP services

- SIP provides mechanisms for call setup:
  - for caller to let callee know she wants to establish a call
  - so caller, callee can agree on media type, encoding
  - 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, hold calls

# Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 dynamic adaptive streaming over HTTP (DASH)

# Notes on Streaming

- RTP/RTSP/RTCP streaming faces several challenges
  - Special-purpose server for media, e.g., fine-grained packet scheduling, keep state (complex)
  - Protocols use TCP and UDP transmissions (firewalls)
  - Difficult to cache data (no "web caching")
- Advantage
  - Short end-to-end latency (<100-500 milliseconds)
  - ☞ still used in WebRTC

# Notes on HTTP Streaming

- Video-on-Demand (VoD) video streaming increasingly uses HTTP streaming
- Simple HTTP streaming just GETs a (whole) video file from an HTTP server
  - ☞ can be wasteful, needs large client buffer

- Solution: Dynamic Adaptive **Streaming over HTTP**
- Main idea of DASH
  - Use HTTP protocol to "stream" media
  - Divide media into small chunks, i.e., streamlets

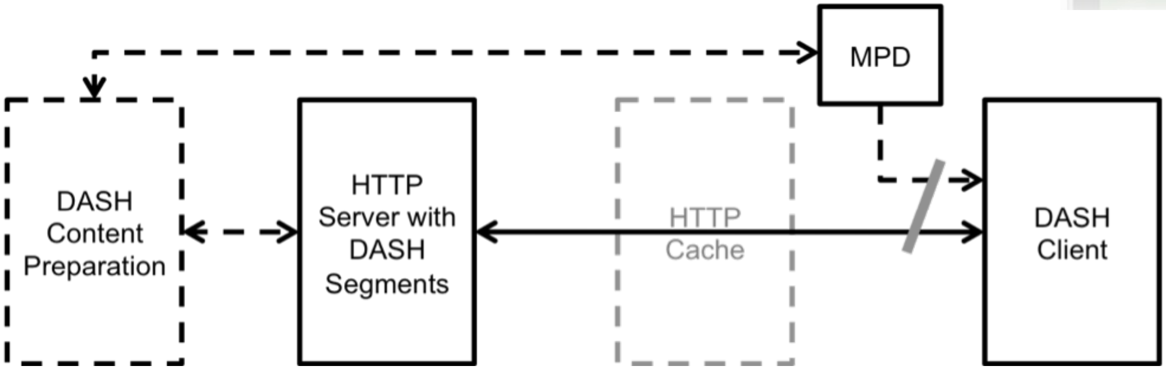
# Notes on HTTP Streaming

- Advantages of DASH
  - Server is simple, i.e., regular web server (no state, proven to be scalable)
  - No firewall problems (use port 80 for HTTP)
  - Standard (image) web caching works
- Disadvantages
  - DASH is based on media segment transmissions, typically 2-10 seconds in length
  - By buffering a few segments at the client side, DASH does not:
    - Provide low latency for interactive, two-way applications (e.g., video conferencing)

http://en.wikipedia.org/wiki/Dash_(disambiguation)

# How DASH works (1)

- Original DASH implementation by Move Networks
  - Introduced concept of streamlets
  - Additional idea: make playback adaptive
    - Encode media into multiple different streamlet files, e.g., a low, medium, and high quality version (different bandwidth)

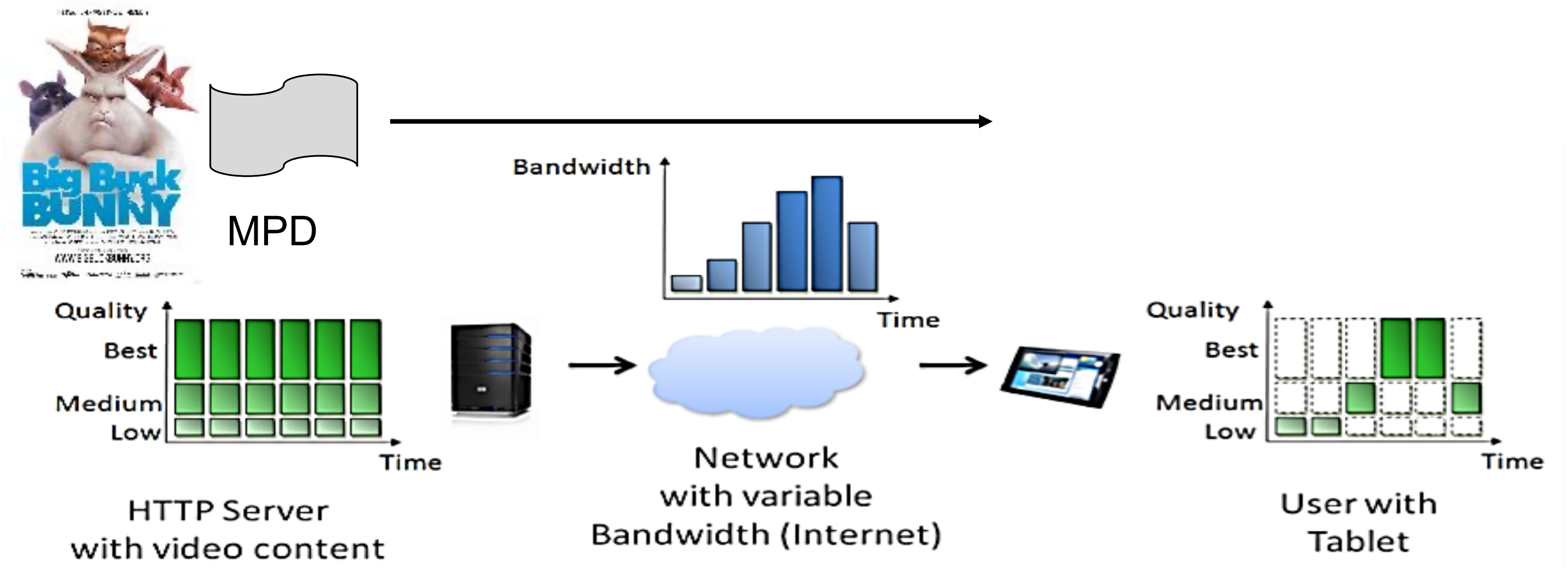


MPD: Media Presentation Description

# How DASH works (2)

- Web server provides a playlist
  - The playlist is a file in a specific format that lists all the available qualities and all the streamlets for each quality
  - Playlist file extension is .m3u8/.mpd
  - Content preparation:
    - Original media file needs to be split into streamlets
    - Streamlets need to be transcoded into different qualities
- ISO/IEC Standard:
  - "Information technology — MPEG systems technologies — Part 6: Dynamic adaptive streaming over HTTP (DASH)"
  - JTC 1/SC 29; FCD 23009-1

# How DASH works (3)



- Data is encoded into different qualities and cut into short segments (streamlets, chunks).
- Client first downloads MPD, which describes the available videos and qualities.
- Client/player executes an adaptive bitrate algorithm (ABR) to determine which segment do download next.

# How DASH works (4)

- DASH is very popular now; it has replaced almost all other VoD streaming protocols
- Used by YouTube, Netflix, Facebook, etc.
- Recent focus is on low(er) latency
  - If latency is too long for live events (e.g., sports) then social media (e.g., Twitter) is "out-of-sync"
- Lots of interesting work on how to build the best ABR algorithm
  - high avg. quality, no stalls, low latency

# Lecture 10: Summary

- There are various media applications on the Internet, even though it provides (few) guarantees
- Three types of media applications:
  - VoD: e.g., YouTube, Netflix
    - one-way, on-demand, media is stored, long latency okay
    - ☞ use DASH
  - Live streaming: e.g., Twitch.tv, Periscope
    - one-way, live source, latency should not be too long
    - ☞ use DASH
  - VoIP, Teleconferencing: e.g., Skype, Zoom, Webex
    - two-way, low latency critical
    - ☞ use RTP (WebRTC)

وشكرًا
Баярлалаа
рахмат
danke
謝謝
ngiyabonga
teşekkür ederim
spas
welalin
tack
mersi
kia ora
barka
спасибо
faafetai lava
vinaka
спасибі
blagodaram
dank je
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
kiitos
dankie
nanni
nandri
dhanyavad
mauruuru
thank you
gracias
хвала
asante
manana
obrigada
tenki
bedankt
bayarlalaa
gràçie
hvala
köszönöm
dankon
ačiū
akun
djiere dieuf
tau
mochchakkeram
murakoze
enkosi
dziękuję
дякую
mamnun
chokrane
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
go raibh maith agat
sobodi
děkuji
sagolun
sukriya
kop khun krap
obrigado
mèsi
didi madloba
najis tuke
kam sah hamnida
ありがとう
tanemirt
grazie
arigatô
takk
dakujem
trugarez
terima kasih
rahmat
rahmet
diolch
dhanyavadagalu
shukriya
mercé
мерси
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
merci