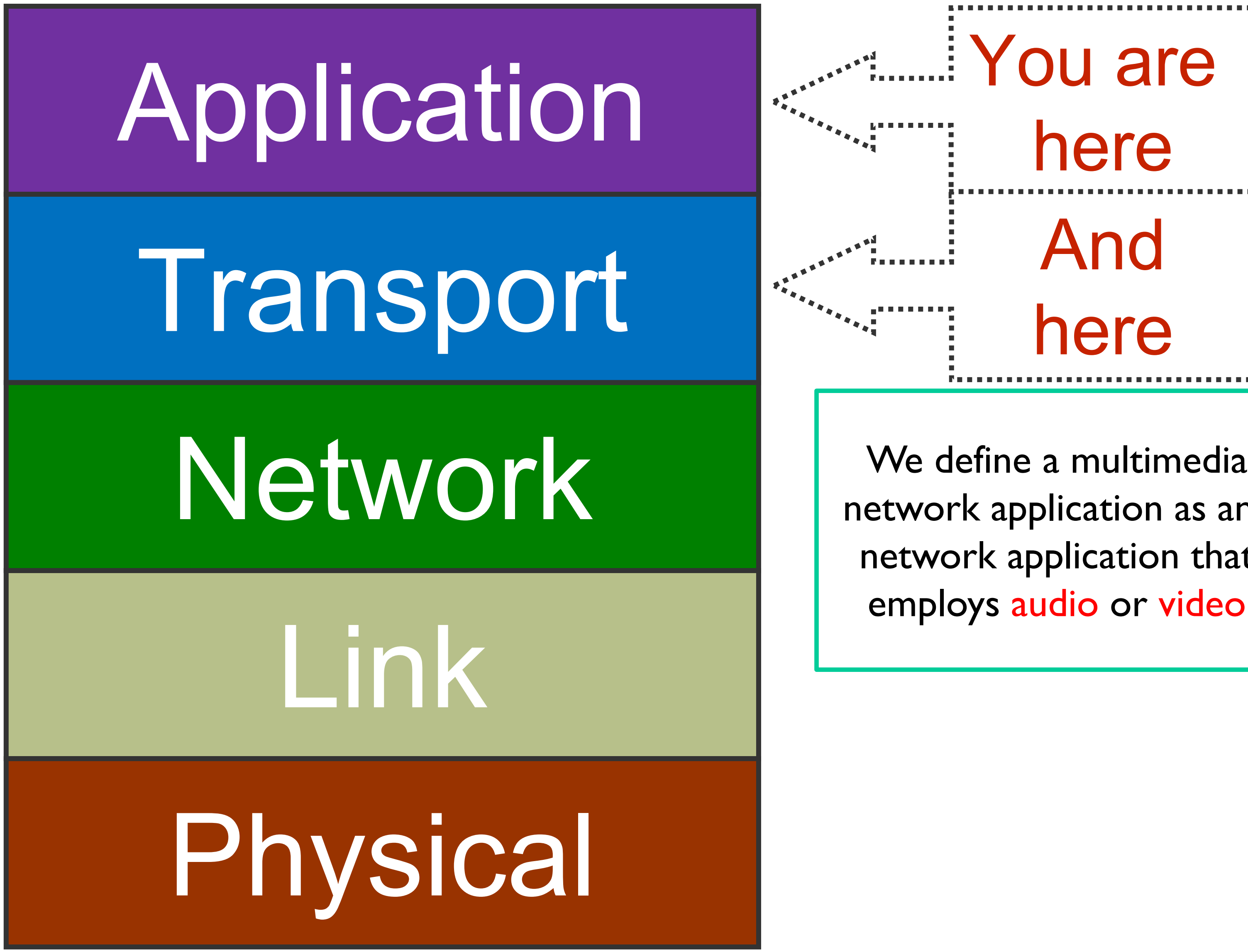
CS2105

An *Awesome* Introduction to Computer Networks

Multimedia Networking

Department of Computer Science
School of Computing



We define a multimedia network application as any network application that employs audio or video

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

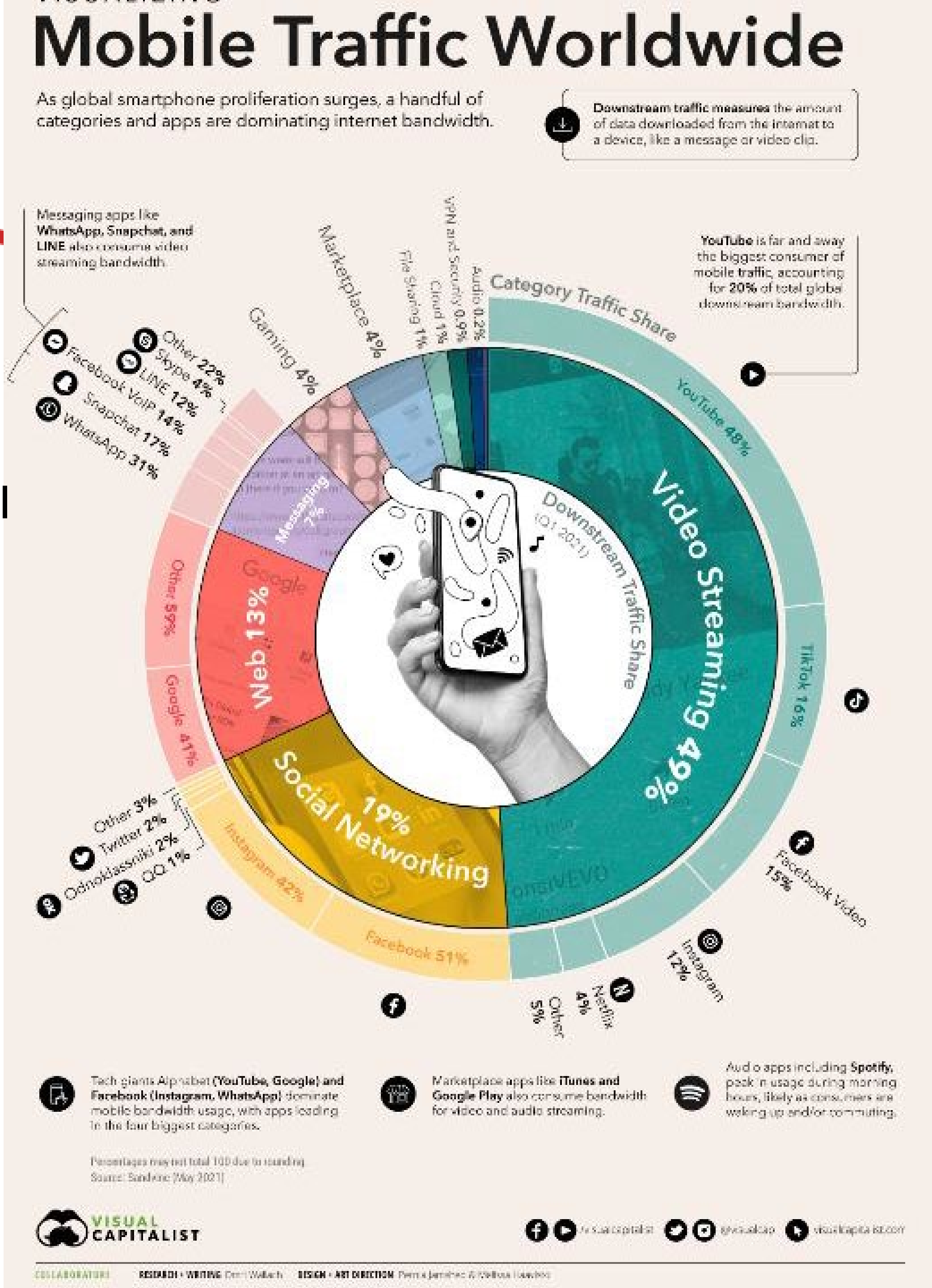
9.3 voice-over-IP

2.6 dynamic adaptive streaming over HTTP (DASH)

© CS2105

Motivation

- Sandvine, THE GLOBAL INTERNET PHENOMENA REPORT, JAN 2022:
  - In 2021, **53.7%** of the global Internet traffic was video
- Top users of internet:
  - YouTube (14.6%)
  - Netflix (9.39%)
- All these are delivered as OTT



Jargon Alert:

- OTT: over-the-top

Multimedia networking: 3 application types

- *Streaming stored* audio, video
  - *Streaming:* can begin playout before downloading entire file
  - *Stored (at server / CDNs):* 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
    - Delay more than 400 milli seconds, intolerable
  - e.g., Skype, Zoom, WhatsApp
- *Streaming live ("one-way live")* audio, video
  - Typically done with CDNs
  - e.g., live sporting event (soccer, football)

Jargon Alert:

- CDN: Content Distribution Network

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

frame *i*

- The most salient characteristic of video is its *high bit rate*
- To reduce data usage, we compress the video:
  - use redundancy *within* and *between* images to decrease # bits used to encode image

Multimedia: video

- Use redundancy *within* and *between* images to decrease # bits used to encode image
  - Spatial Coding (within image)
  - Temporal Coding (from one image to next)

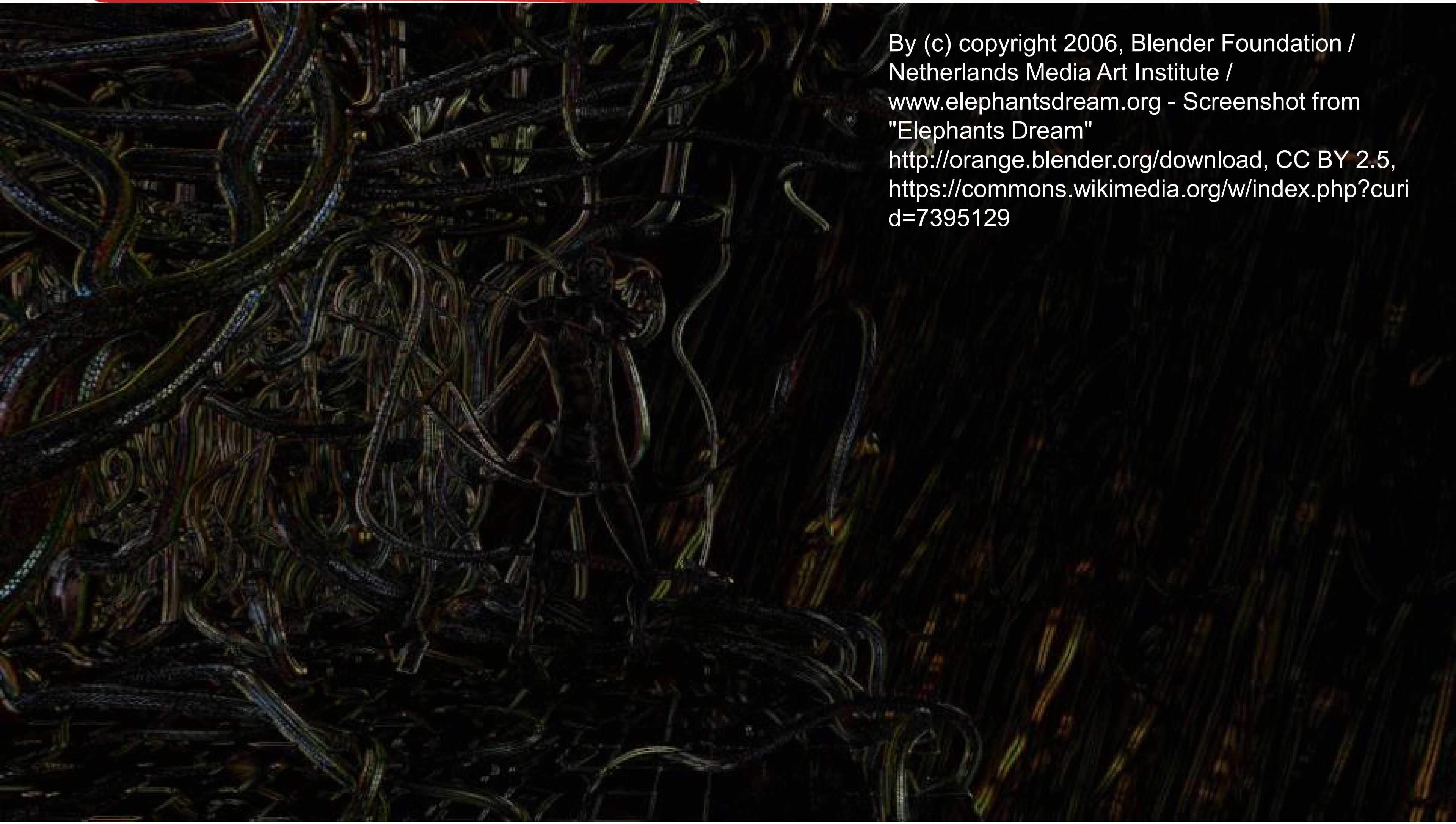
spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of repeated values* (N)

frame *i* → frame *i+1*

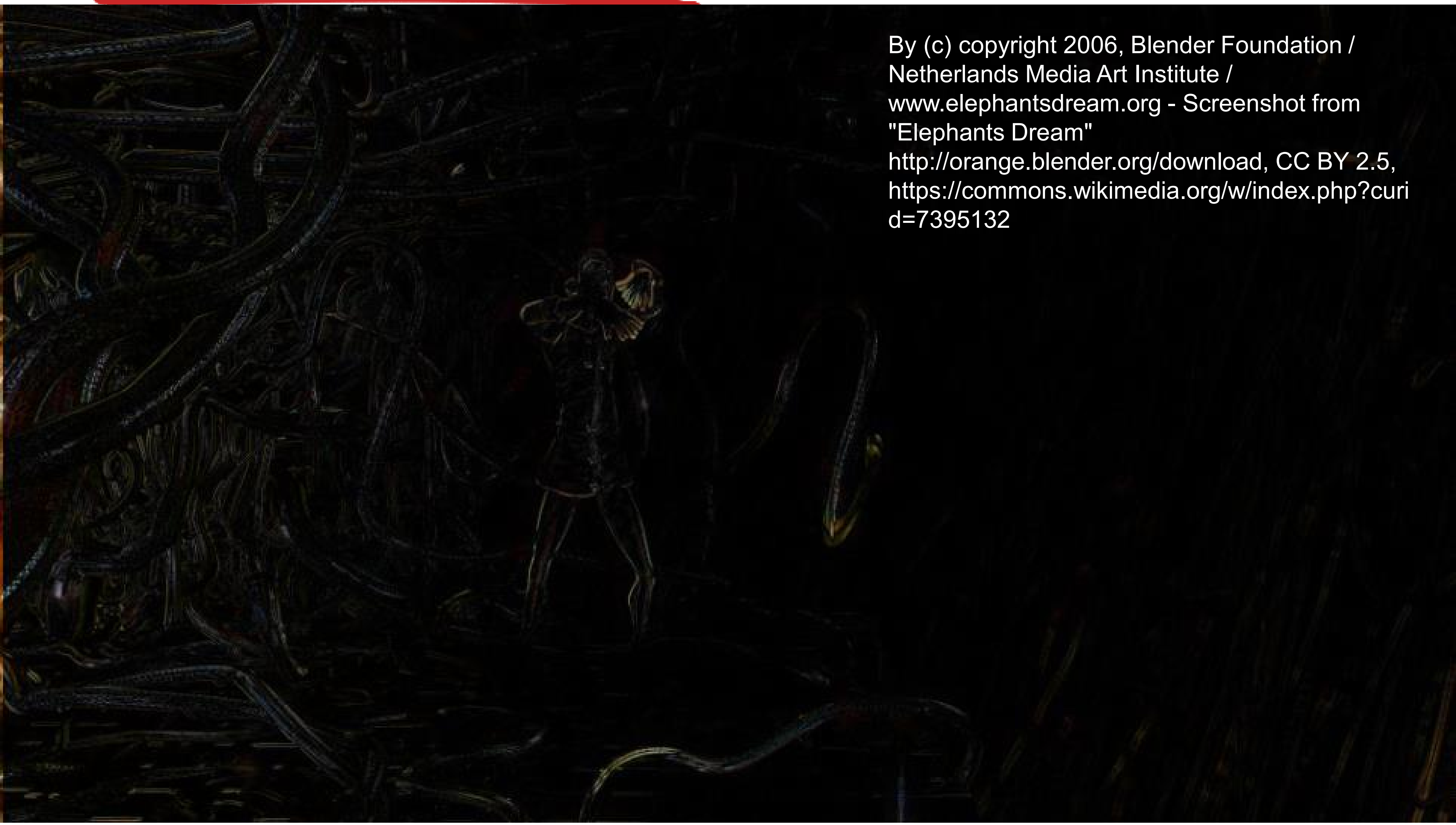
temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

Video: original frame *i*

Difference between 2 frames



Motion compensated difference



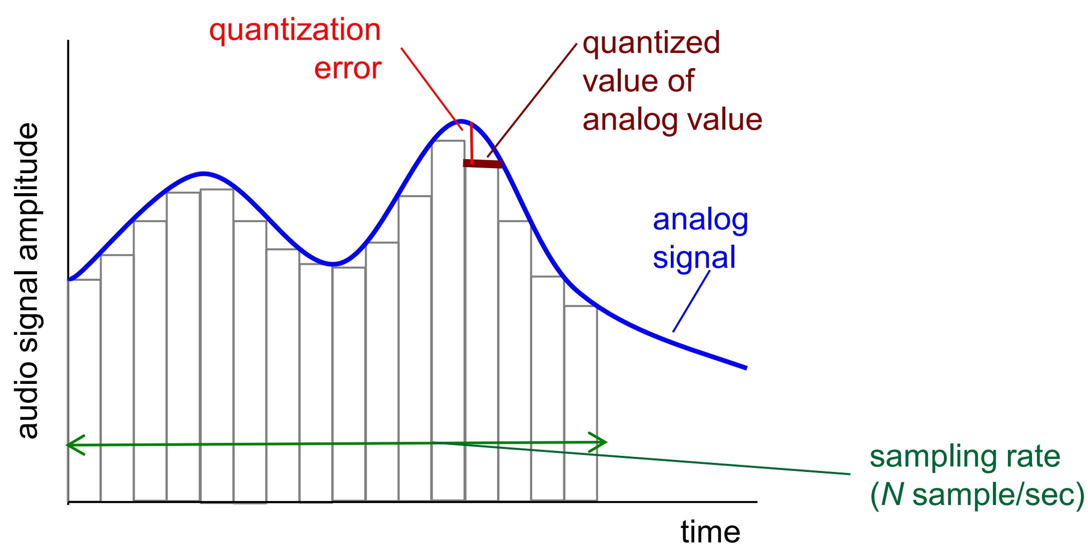
Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
  - Not responsive to the complexity of the video.
  - Need to set your bitrate relatively high to handle more complex segments of video.
  - The consistency of CBR makes it well-suited for real-time encoding.
    - For *real-time live streaming*
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
  - VBR best suited for *on-demand video* due to longer time to process the data.
- 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 > 10 Mbps

Multimedia: audio

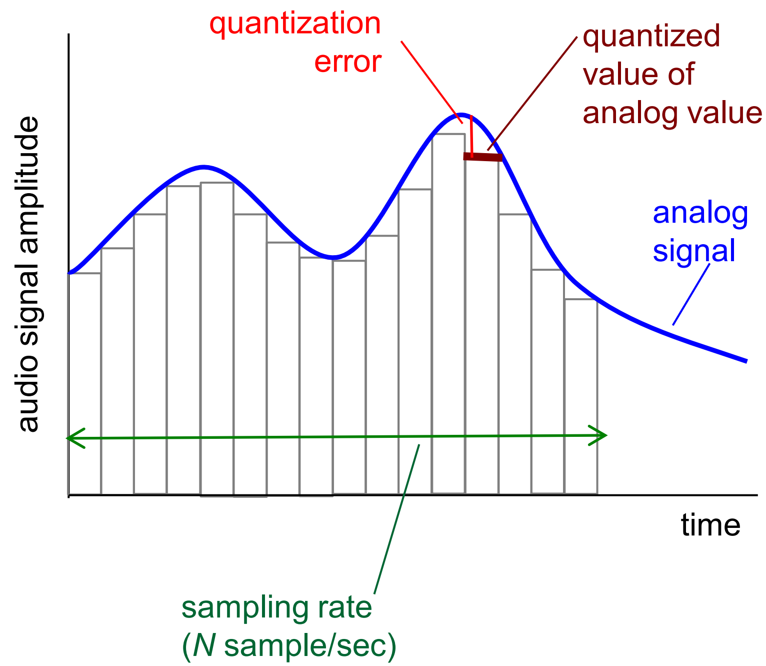
- Analog audio signal
  - sampled at constant rate
    - Telephone: 8,000 samples/sec
    - CD music: 44,100 samples/sec

- each sample quantized, i.e., rounded
  - each quantized value represented by bits, e.g., 8 bits for 256 (2^8) values



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

Jargon Alert:

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

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

2.6 dynamic adaptive streaming over HTTP (DASH)

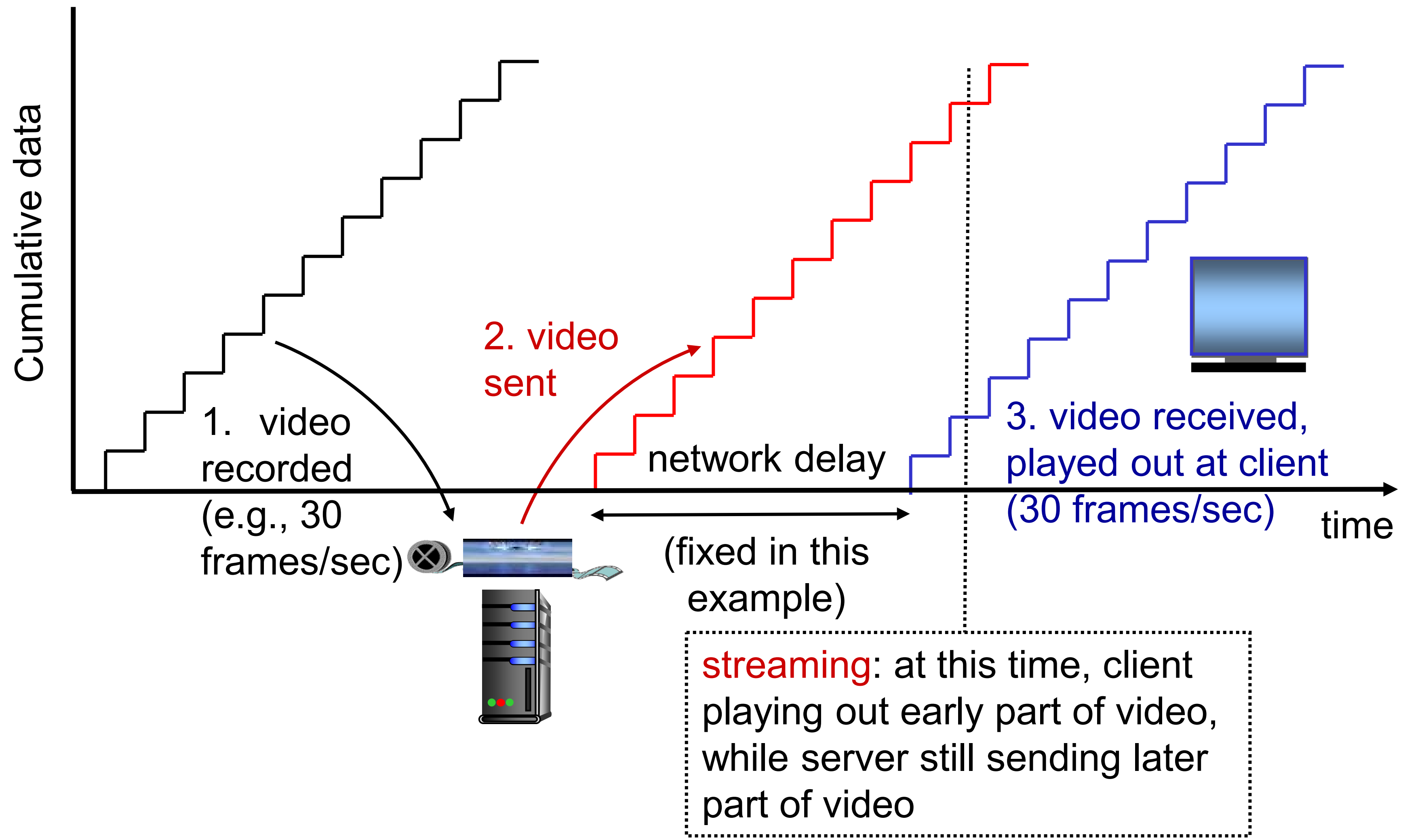
Streaming stored video:

- *Streaming stored* video
  - *Streaming:* Can begin playout *before* downloading entire file
  - *Stored (at server / CDNs):* can transmit faster than audio/video will be rendered (implies *storing/buffering* at client)

Jargon Alert:

- CDN: content distribution Network

Streaming stored video:

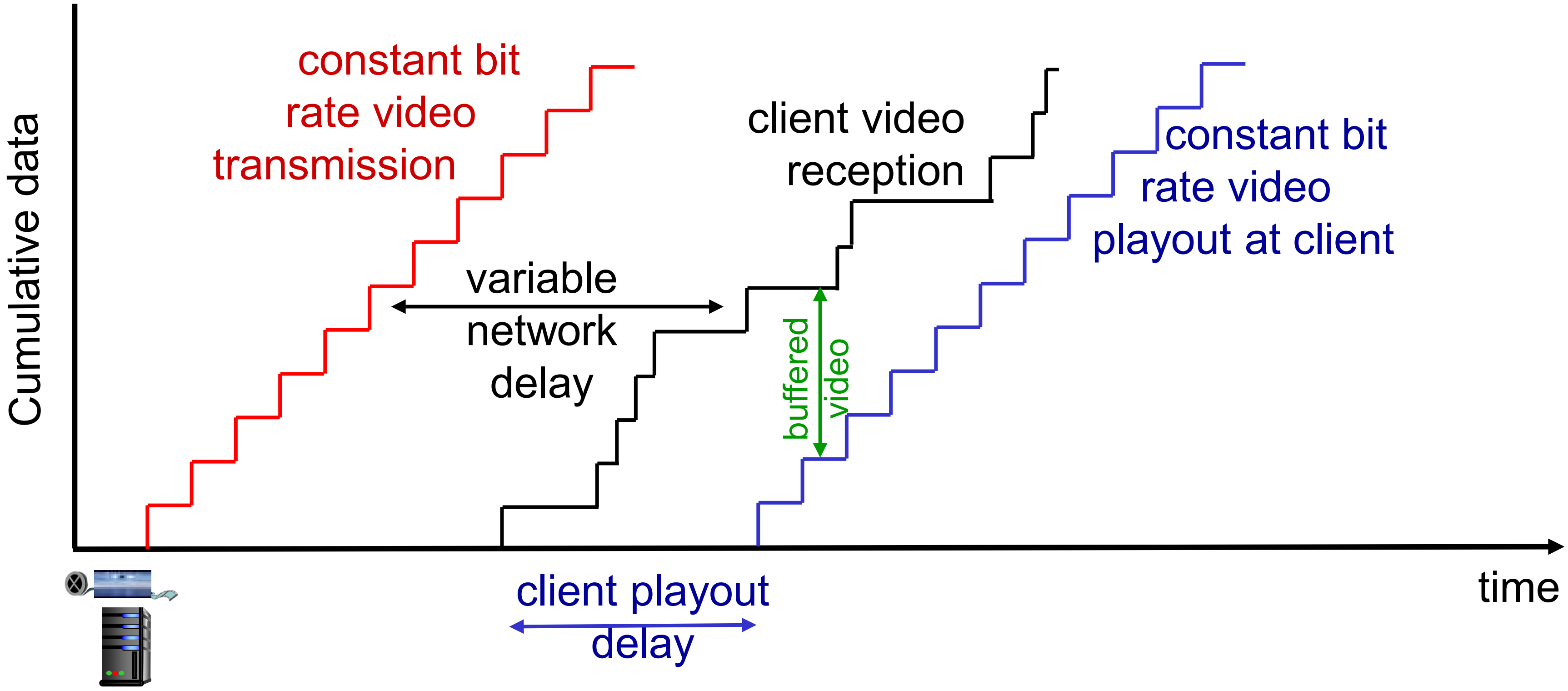


Streaming stored video: challenges

- Continuous playout constraint:
  - once client playout begins, playback *must match* original timing
  - … but network delays are variable (jitter),

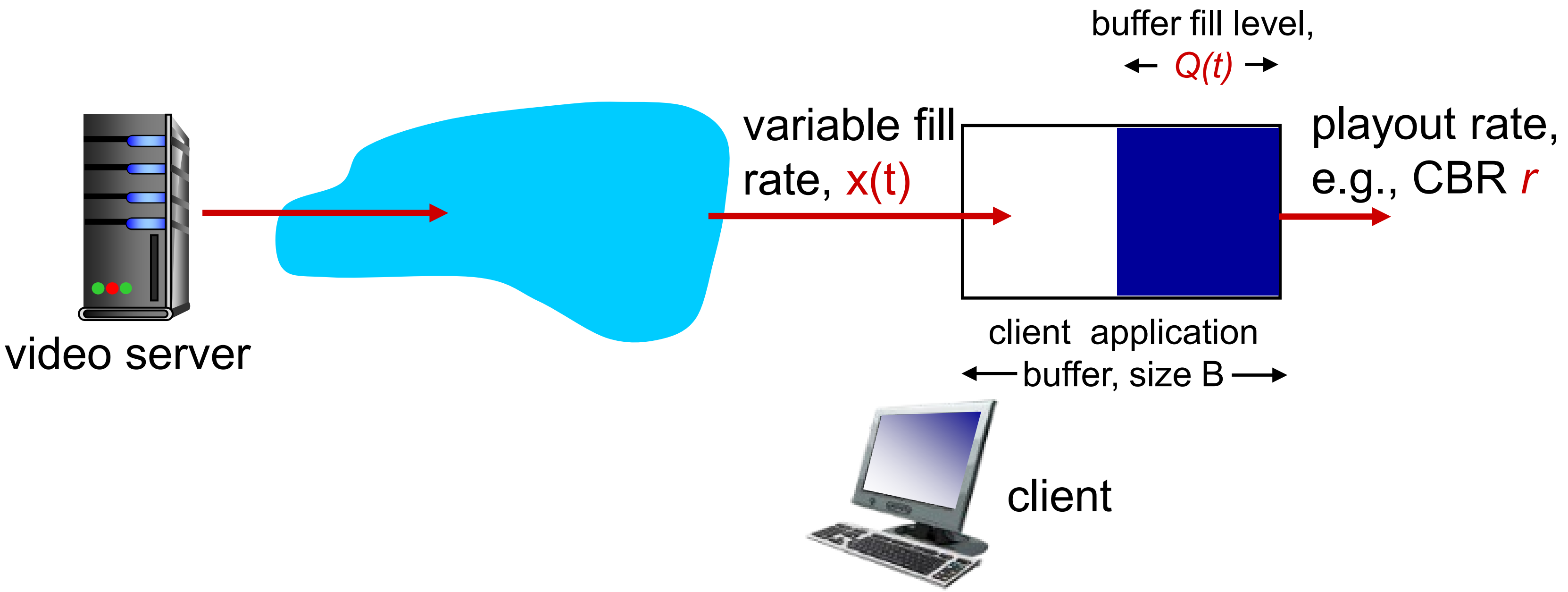
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted

Streaming stored video: revisited

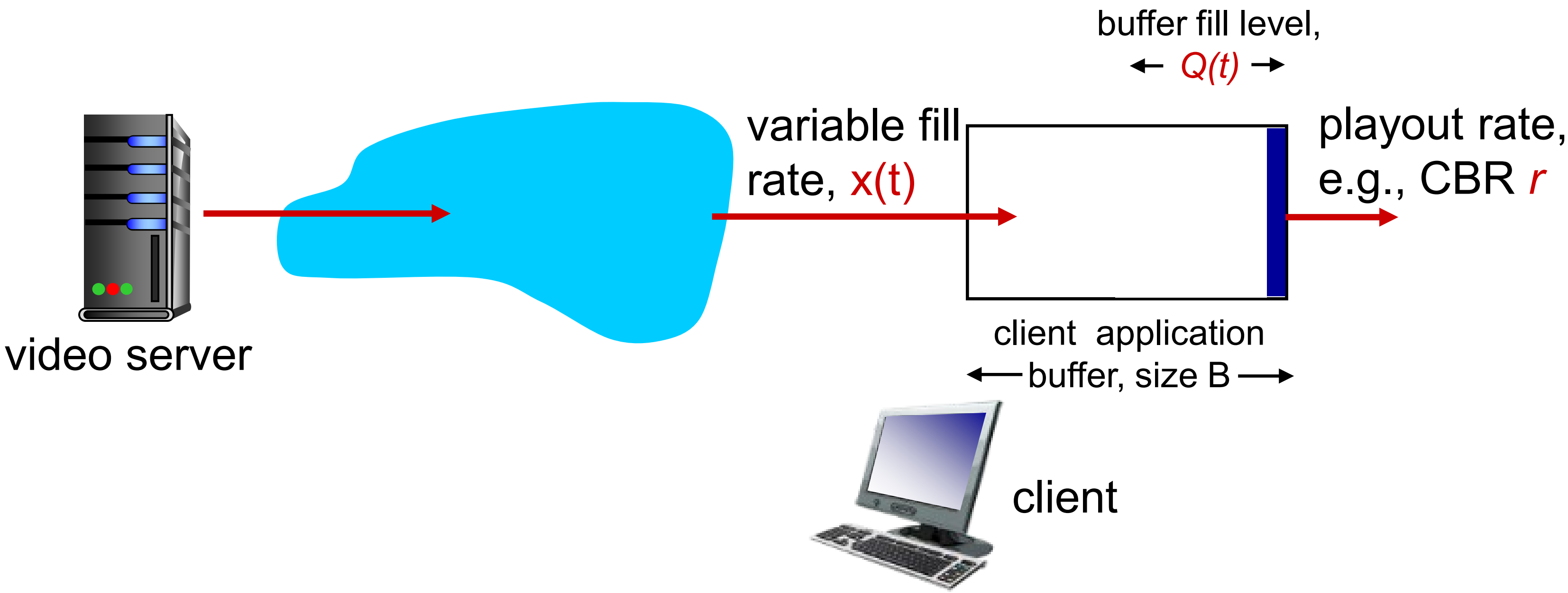


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout

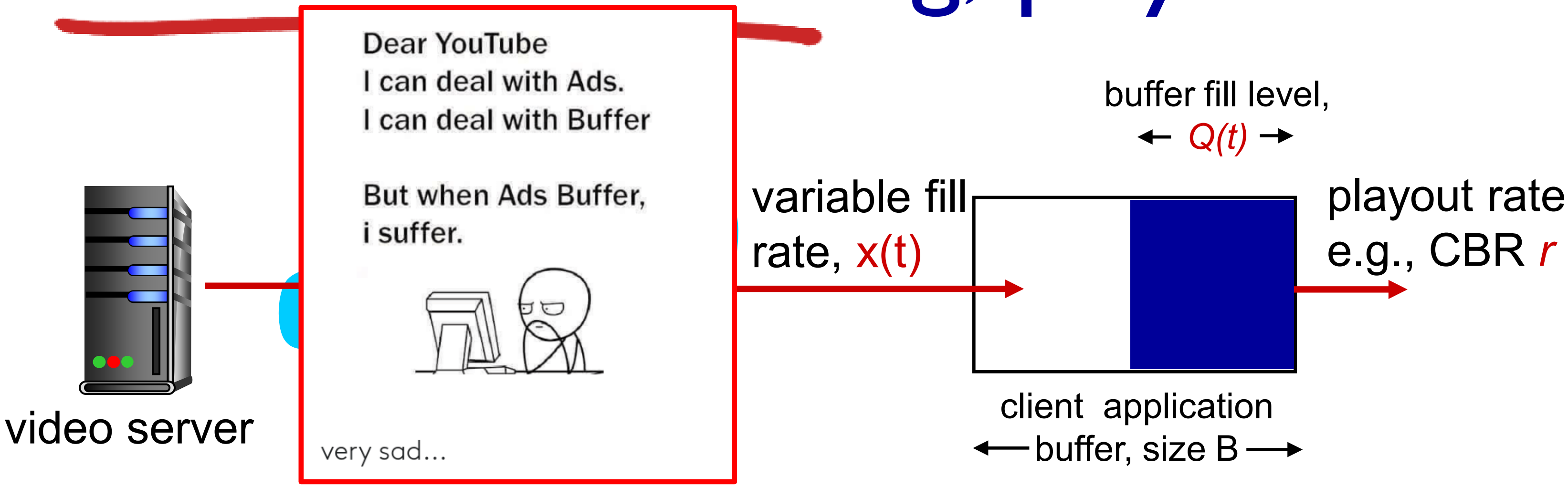


Client-side buffering, playout



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



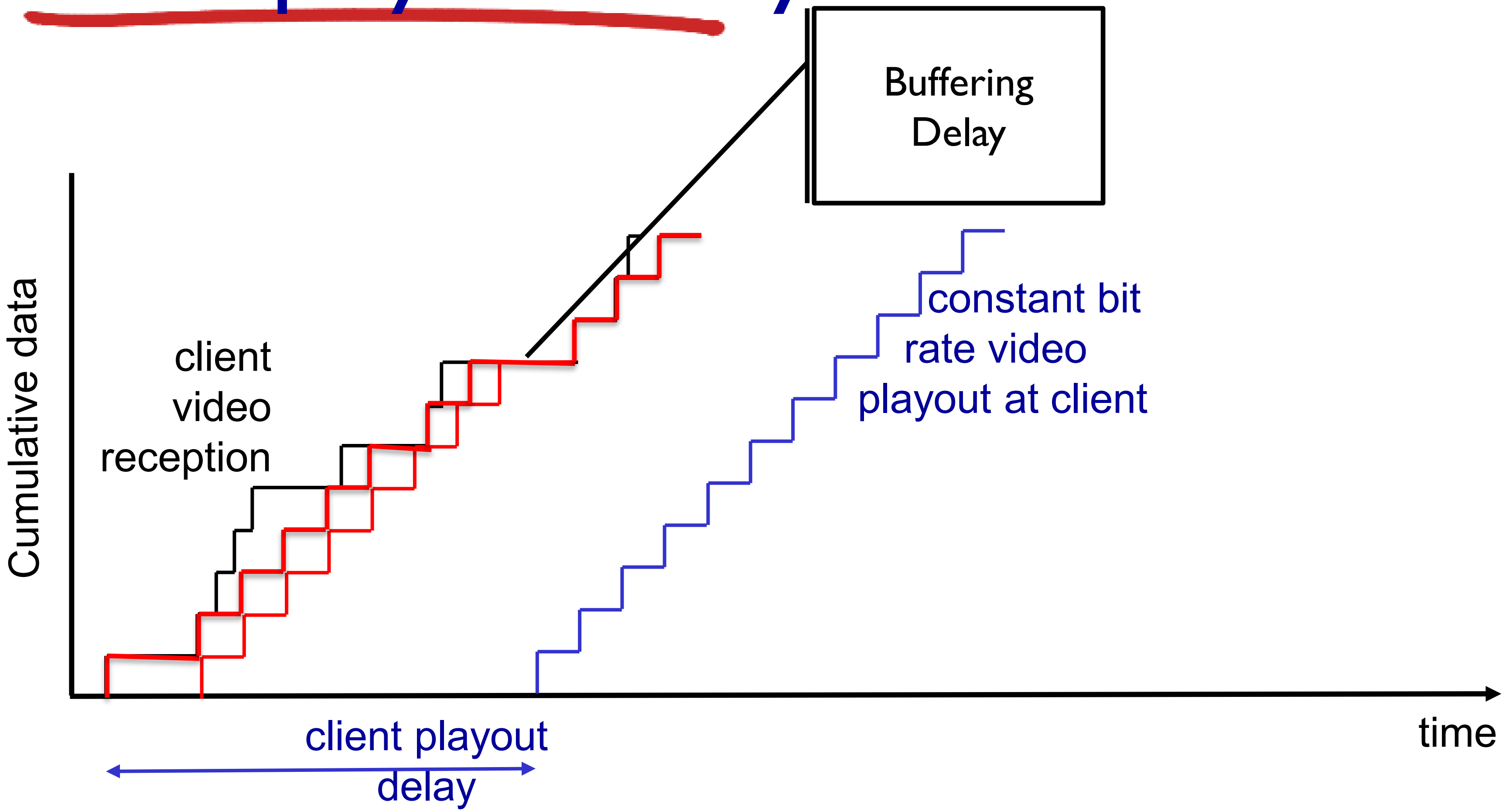
playout buffering: average fill rate ($\bar{x}$), playout rate (r):

- $\bar{x}$ < 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

Initial playout delay tradeoff



Buffering

- You are using a media player which has a playout buffer size of $B_{Playout}$ = 8 MB.
- The buffer is initially empty. The time when you press "play" for a video is t_0. It takes 4 seconds to fill the buffer $B_{Playout}$ to its mid-point, i.e., 4 MB of data.
- At that point (t_0 + 4 secs) the media player starts to play the video.
- The video has a size of 14 MB.
- The data continues to arrive after (t_0 + 4 secs) from the server at a constant rate of 8 Mb/s and the player plays (decodes) the media at a rate of 4 Mb/s until the video ends.
- Which of the following statements are **TRUE**? (Times are measured relative to t_0).

NUS National University of Singapore

Which of the following statements are TRUE? (Times are measured relative to t_0)

- The video will play normally for the whole duration of the video and will end at $t_0 + 28$ seconds.
- The $B_{Playout}$ buffer will overflow at $t_0 + 12$ seconds. (And the video may stall/stop.)
- The $B_{Playout}$ buffer will overflow at $t_0 + 8$ seconds. (And the video may stall/stop.)
- The $B_{Playout}$ buffer will underflow at $t_0 + 12$ seconds. (And the video may stall/stop.)
- The server will have delivered all the data of the video to the client at $t_0 + 12$ seconds.

Poll locked. Responses not accepted.

NUS National University of Singapore

Which of the following statements are TRUE? (Times are measured relative to t_0)

The video will play normally for the whole duration of the video and will end at t_0 + 28 seconds.

The $B_{Playout}$ buffer will overflow at t_0 + 12 seconds. (And the video may stall/stop.)

The $B_{Playout}$ buffer will overflow at t_0+ 8 seconds. (And the video may stall/stop.)

The $B_{Playout}$ buffer will underflow at t_0+ 12 seconds. (And the video may stall/stop.)

The server will have delivered all the data of the video to the client at t_0 + 12 seconds.

Poll locked. Responses not accepted.

Which of the following statements are TRUE? (Times are measured relative to t_0)

The video will play normally for the whole duration of the video and will end at t_0 + 28 seconds.

The $B_{Playout}$ buffer will overflow at t_0 + 12 seconds. (And the video may stall/stop.)

The $B_{Playout}$ buffer will overflow at t_0+ 8 seconds. (And the video may stall/stop.)

The $B_{Playout}$ buffer will underflow at t_0+ 12 seconds. (And the video may stall/stop.)

The server will have delivered all the data of the video to the client at t_0 + 12 seconds.

Buffering

- $Size(video) = 14MB = 112Mb \quad = 28\ chunks$
- $B_{Playout} = 8MB = 64Mb \quad = 16\ chunks$
- $Q(t_0 + 4) = 4MB = 32Mb \quad = 8\ chunks$
- $t_p = t_0 + 4$
- *fill rate,* $\bar{x} = 8Mbps \quad = 2\ chunks\ per\ sec$
- *playout rate,* $r = 4Mbps \quad = 1\ chunk\ per\ sec$
- *Rate of buffer growth* $= \bar{x} - r = 4Mbps \quad = 1\ chunk\ per\ sec$

- *Time taken by the buffer to fill* $= (t_0 + 4) + \frac{16-8}{1}$
  $= t_0 + 12$
- Video played till $(t_0 + 12) = (t_0 + 12) - (t_0 + 4) = 8\ chunks$
- Data delivered till $(t_0 + 12) = (8 + 16) \quad = 24\ chunks$
- *fill rate after* $(t_0 + 12)$, $\bar{x} = 1\ chunk\ per\ sec$
- Time to deliver the video in total $= (t_0 + 12) + (size(video) - data\ delivered)/\bar{x}$
  $= (t_0 + 12) + \frac{28 - 24}{1}$
  $= t_0 + 16$
- End of play time = $(t_0 + 4) + 28 = t_0 + 32$

Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate,
    - ☞ push-based streaming (*server push*)
  - UDP has no congestion control
    - Hence transmission without rate control restrictions
- short playout delay (2-5 seconds) to remove network jitter
- Error recovery: application-level, time permitting

Streaming multimedia: UDP

- Video chucks encapsulated using RTP
- Control Connection is maintained *separately* using RTSP
  - Is used for establishing and controlling media sessions between endpoints.
  - Clients issue commands such as *play*, *record* and *pause*
- Drawbacks
  - Need for a separate media control server like RTSP, increases cost and complexity
  - UDP may *not* go through firewalls

Seq num,
Time stamp,
Video encoding

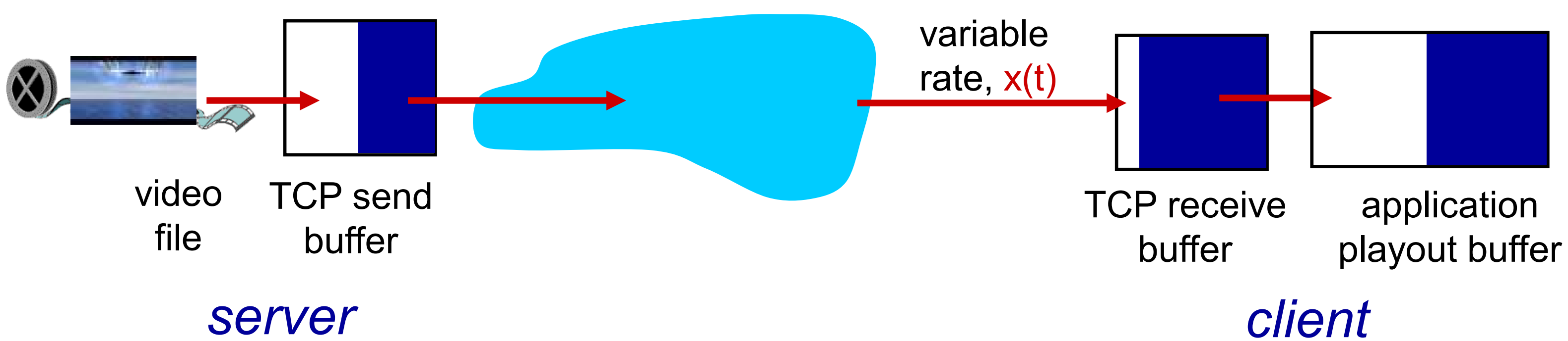
Jargon Alert:

- RTP: Real time Transport protocol
- RTSP: Real time Streaming protocol

Streaming multimedia: HTTP

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

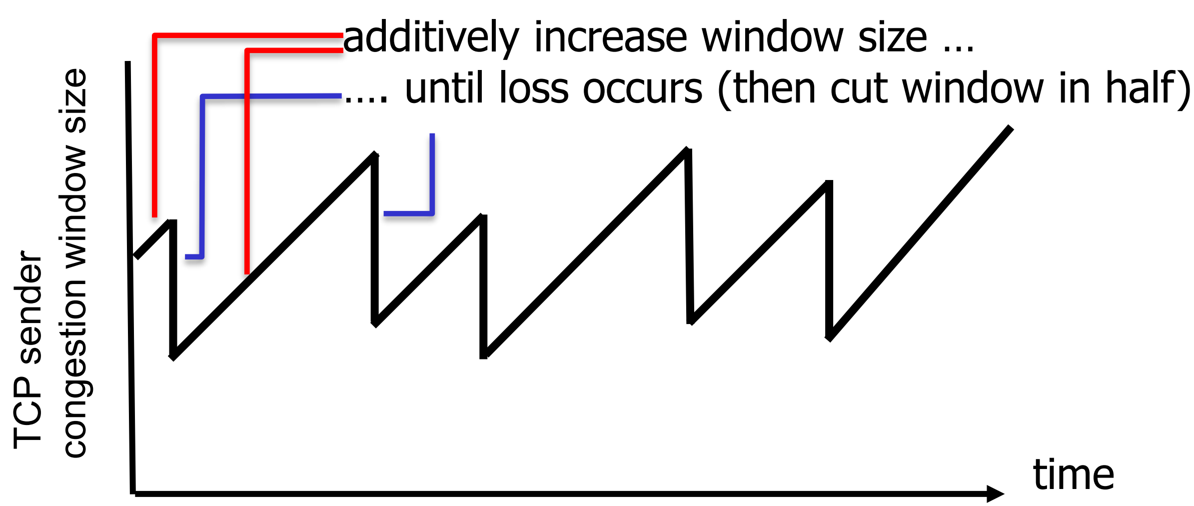
- send at maximum possible rate under TCP



Streaming multimedia: HTTP

- Advantages
  - HTTP/TCP passes more easily through *firewalls*
  - Network *infrastructure* (like CDNs and Routers) fine tuned for HTTP/TCP
- Drawbacks
  - fill rate *fluctuates* due to TCP congestion control, retransmissions (in-order delivery)
  - *larger* playout delay: smooth TCP delivery rate

Additive Increase Multiplicative Decrease *saw tooth* behavior: *probing for bandwidth*



Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

2.6 dynamic adaptive streaming over HTTP (DASH)

Conversational Multimedia: 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
  - Data loss over 10% makes conversation unintelligible.
- *Challenge:*
  - Internet (IP layer) is a *best-effort* service
    - No upper bound on *delay*
    - No upper bound on percentage of *packet loss*

Jargon Alert:
- VoIP: Voice over IP

VoIP characteristics

- Speaker's audio:
  - alternating talk spurts, silent periods.
  - 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 talk spurt
- *Challenge:*
  - No upper bound on *delay*
  - No upper bound on percentage of *packet loss*

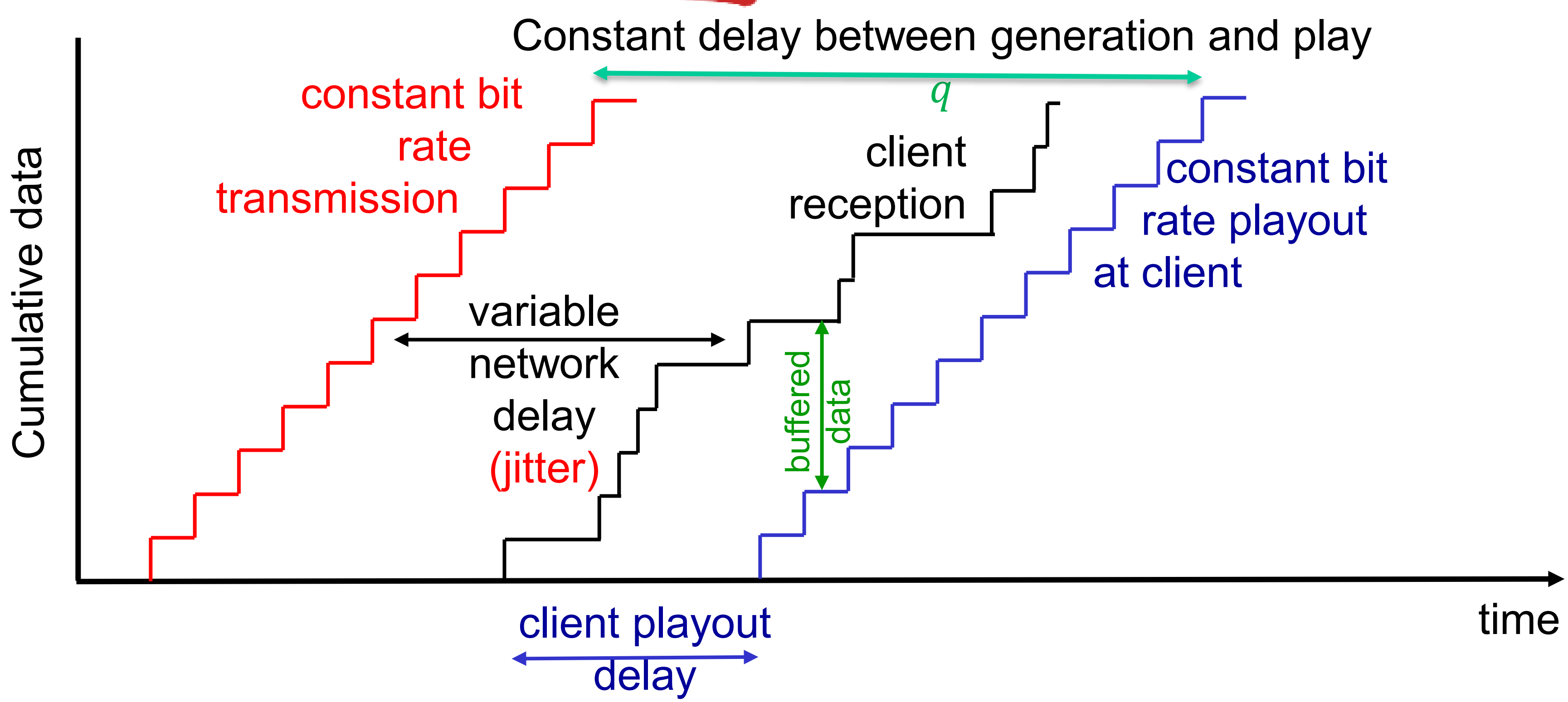
When to play back a chuck?

What to do with missing chuck?

VoIP: packet loss, delay

- *Network loss:* IP datagram lost due to network congestion (router buffer overflow, etc.)
- *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
  - VoIP Applications typically use UDP to avoid Congestion control.
- *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)

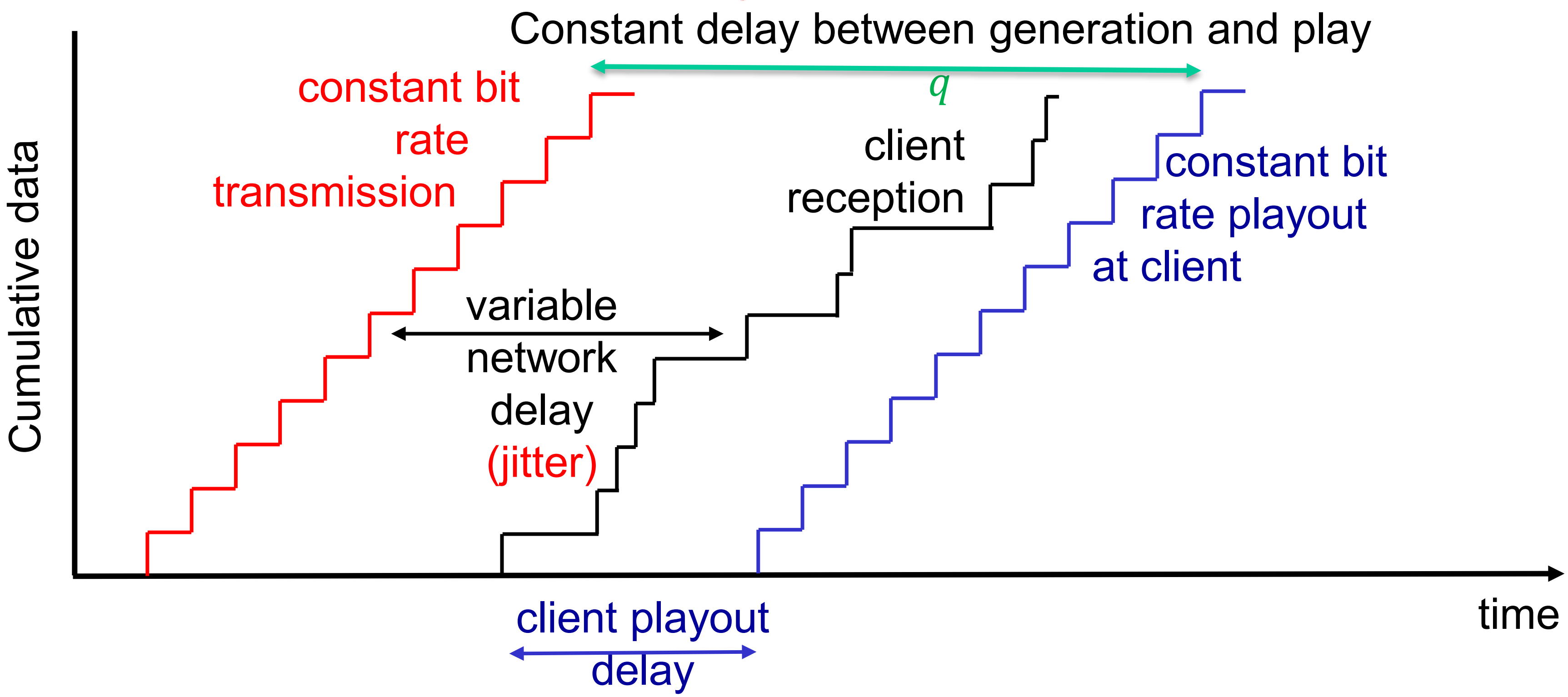
VoIP: fixed playout delay

- 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"

Every Chunk will have
- Sequence Number
- Timestamp

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

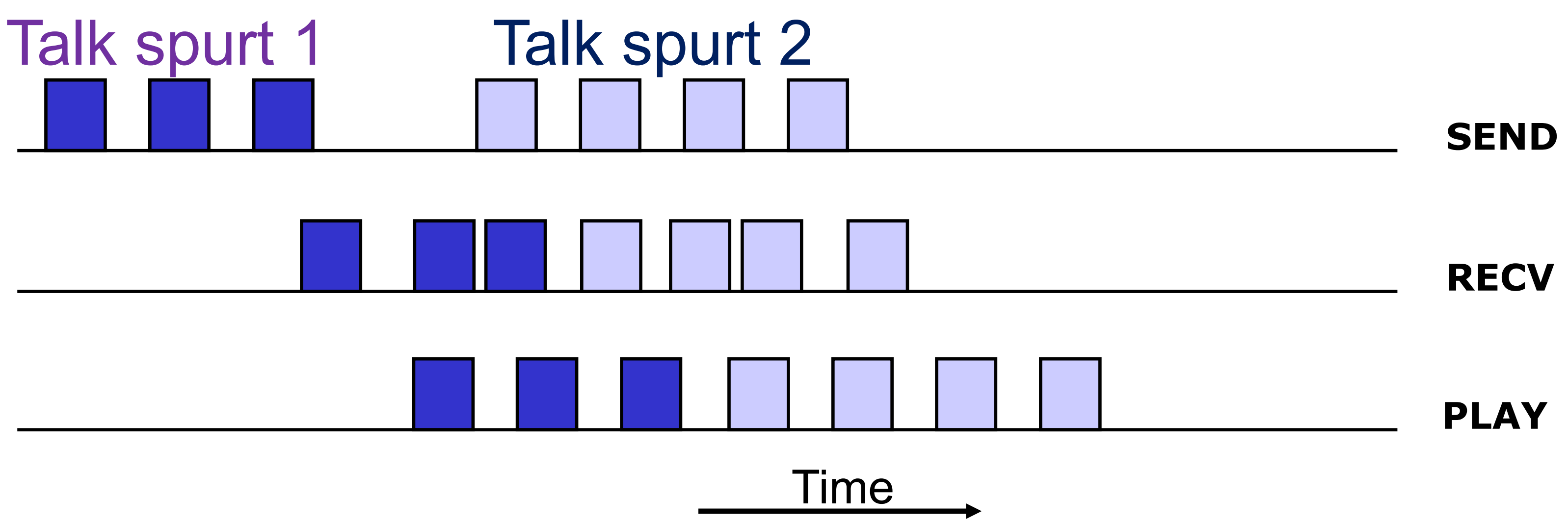
VoIP: fixed playout delay



- No value of q can guarantee an optimal performance
  - We will eventually have a packet loss, or
  - Waste a lot of playout time

Adaptive playout delay

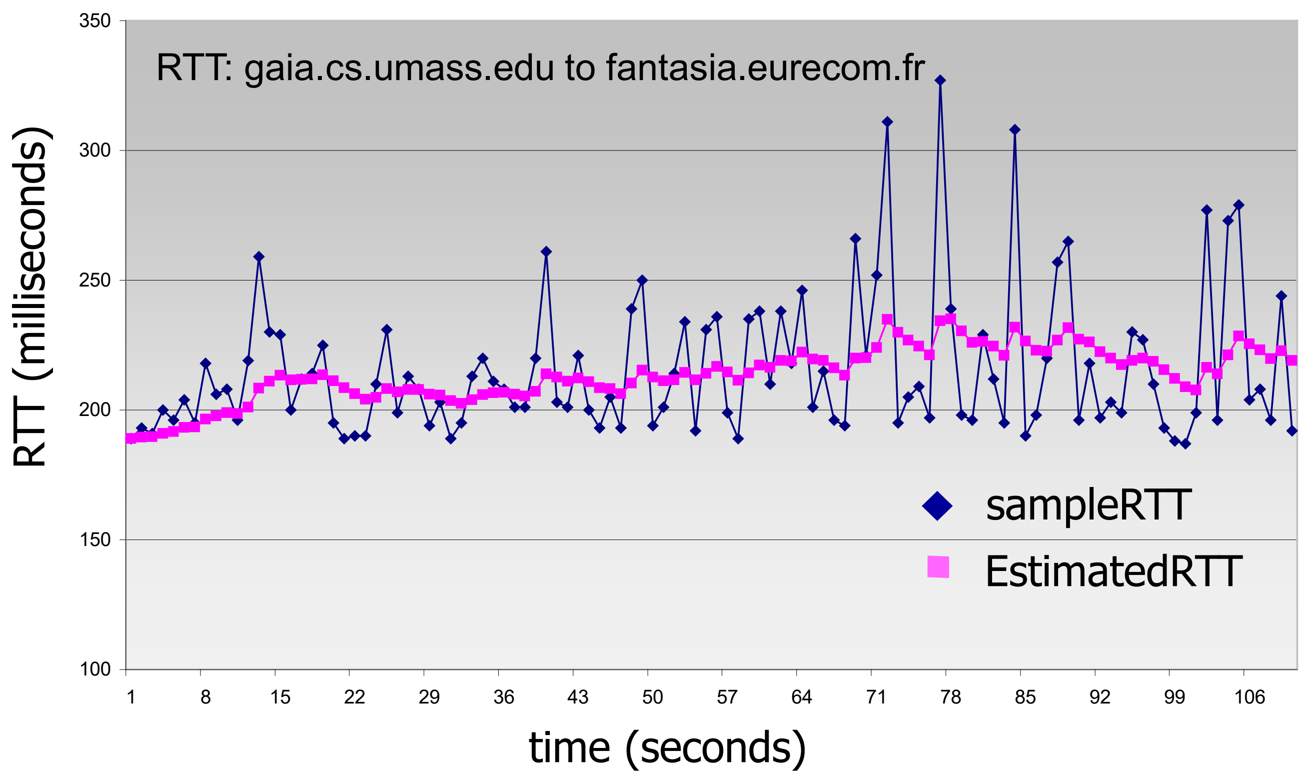
- *goal:* low playout delay, low late loss rate
- *approach:* adaptive playout delay adjustment
  - *estimate* network delay, *adjust* playout delay at beginning of each talk spurt
  - silent periods *compressed* and *elongated*
    - chunks still played out every 20 msec during talk spurt



Recall: TCP round trip time, timeout

$$\texttt{EstimatedRTT} = (1-\alpha)*\texttt{EstimatedRTT} + \alpha*\texttt{SampleRTT}$$

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$



Recall: TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT +
                β*|SampleRTT-EstimatedRTT|

                (typically, β = 0.25)
```

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

"safety margin"

Adaptive playout delay

Jargon Alert:
- EWMA: exponentially weighted moving average

- Adaptively estimate packet delay (EWMA):

$$\underbrace{d_i}_{\text{delay estimate after ith packet}} = (1-\underbrace{\alpha}_{\text{small constant, e.g. 0.1}})d_{i-1} + \alpha\,\underbrace{(r_i - t_i)}_{\text{measured delay of ith packet}}$$

time received - time sent (timestamp)

estimate of average deviation of delay after ith packet

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

- Estimates, d_i *and* v_i calculated for every received packet, but used only at start of talk spurt
  - for first packet in talk spurt, playout time is:

$$playout\text{-}time_i = t_i + d_i + 4v_i$$

  - remaining packets in talk spurt are played out periodically

VoIP: recovery from packet loss

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

- Use ACK/NAK
  - Each ACK/NAK takes ~ one RTT
  - Too slow
- Alternative: *Forward Error Correction (FEC)*
  - send enough bits to allow recovery without retransmission (recall two-dimensional parity)

VoIP: recovery from packet loss

Simple FEC

- for every group of n chunks
  - create redundant chunk by XOR-ing n original chunks
  - send $n + 1$ chunks
- can reconstruct original n chunks if at most one lost chunk from $n + 1$ chunks, with playout delay

$$\begin{array}{cccccc} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 \end{array}$$

- Drawback
  - Increasing bandwidth by factor $1/n$
  - Playout delay is increased during packet loss
    - Receiver waits for $n + 1$ chunks before playout

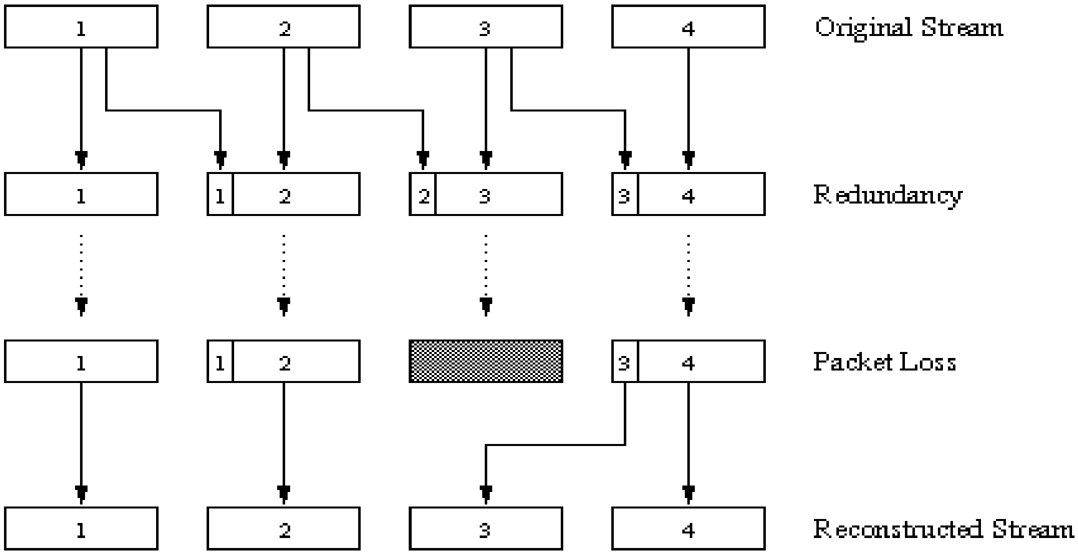
XOR (exclusive OR)

- Commutative
- Associative

VoIP: recovery from packet loss

Another cool 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

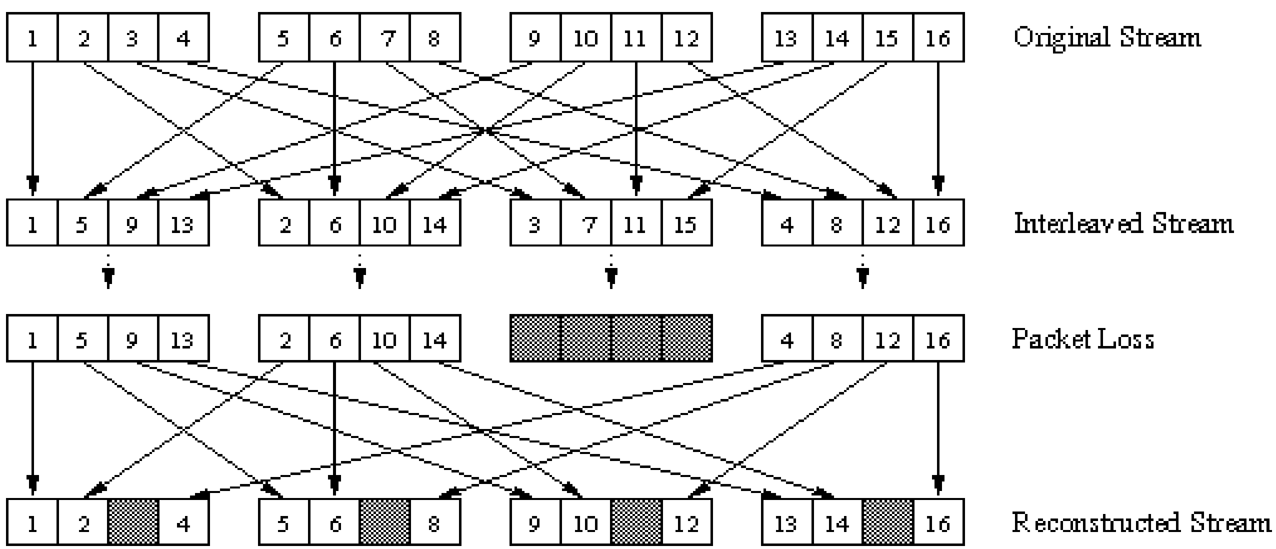


- *non-consecutive* loss: receiver can *conceal* loss
- generalization: can also append (n-1)st and (n-2)nd low-bit rate chunk

VoIP: recovery from packet loss

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
  - Concealed by packet repetition or interpolation
- no redundancy overhead, but increases playout delay, even without error



Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

2.6 dynamic adaptive streaming over HTTP (DASH)

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
- Drawbacks
  - can be wasteful, needs large client buffer
  - All client receive the *same* encoding of video, despite the variation in the device/network bandwidth

Solution:

Dynamic Adaptive Streaming over HTTP

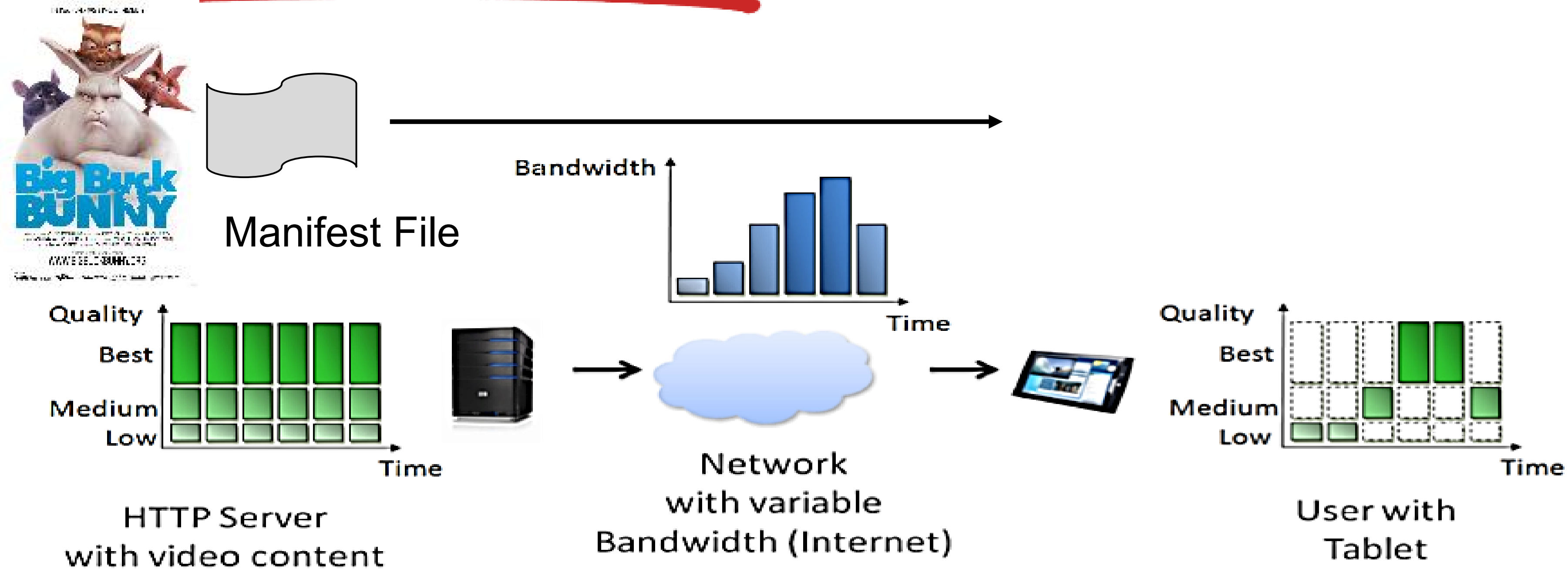
Streaming multimedia: DASH

- *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 encodings
- *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)

How DASH works



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

Streaming multimedia: DASH

- 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)

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

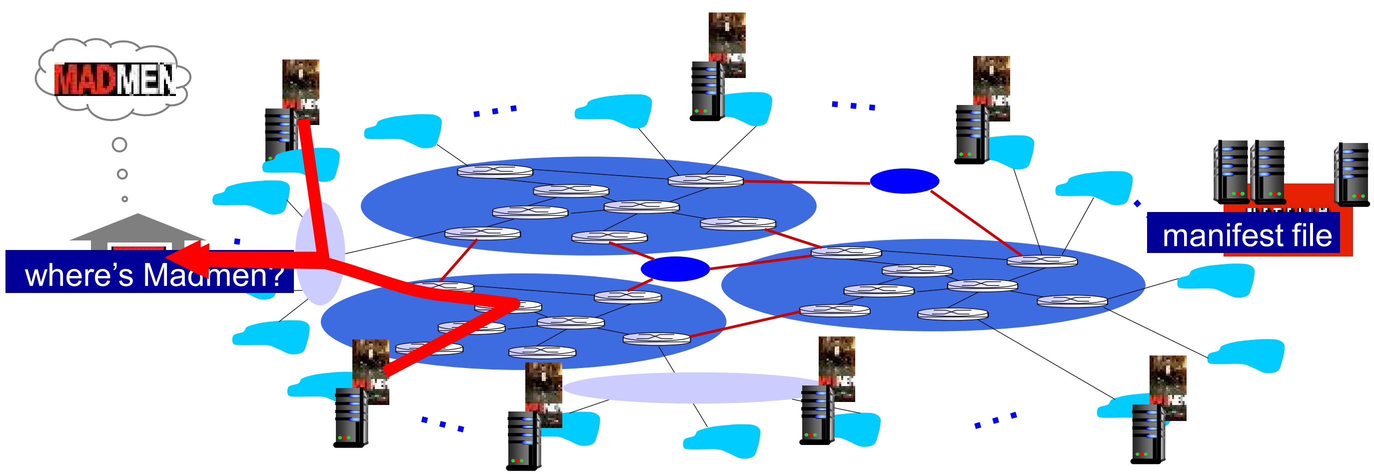
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

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

Jargon Alert:
- IXP: *Internet* exchange point

Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- Client requests content
  - service provider returns manifest
- using manifest, client retrieves content at highest supportable rate
- may choose different rate or copy if network path congested



Summary

- Encoding exploiting
  - Spatial redundancy
  - Temporal redundancy
- Client-side Buffering
  - Playout delay
  - Congestion Control
- VoIP
  - FEC
  - Error concealment
- Video Streaming
  - UDP
  - HTTP
  - DASH
  - CDN

There are various media applications on the Internet, even though it provides (few) guarantees