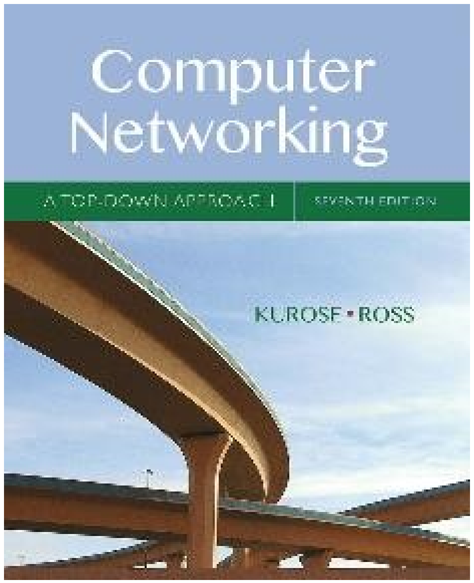
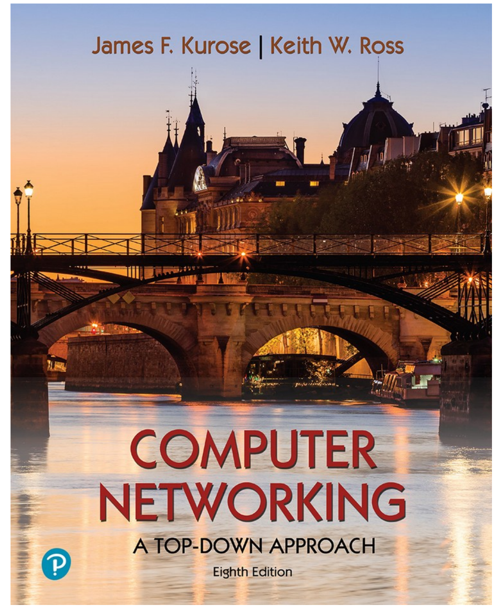
Computer Networking: A Top Down Approach



Chapter 9 Multimedia Networking

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016



Chapter 2 Application Layer

8th edition
Jim Kurose, Keith Ross
Pearson,
2020

A note on the use of these Powerpoint 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.

©anks and enjoy! JFK/KWR

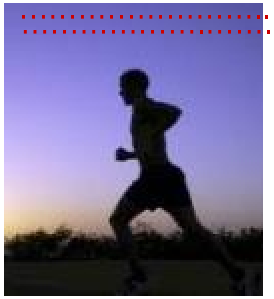
Video Streaming and CDNs: context

- stream video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube, Amazon Prime: 80% of residential ISP traffic (2020)
- *challenge:* scale - how to reach ~1B users?
- *challenge:* heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)
- *solution:* distributed, application-level infrastructure

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*

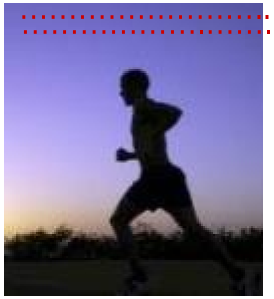
frame *i+1*

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

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, 64Kbps – 12 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

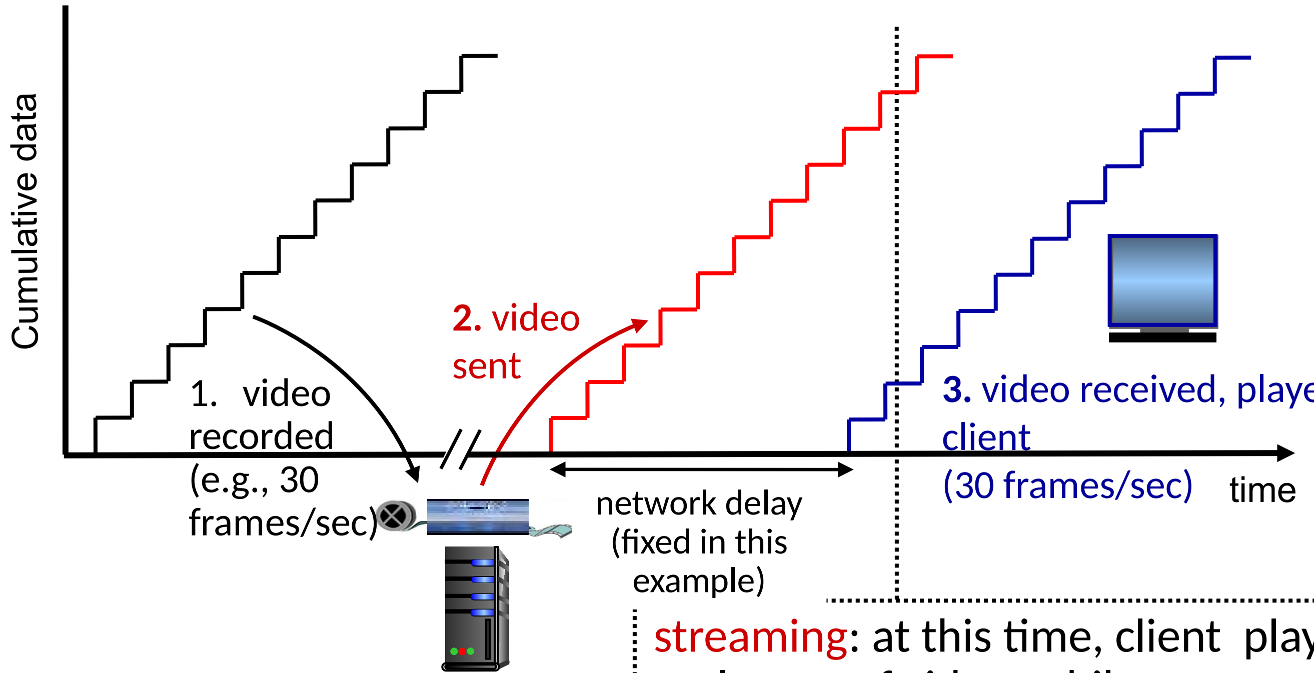
frame *i+1*

Multimedia networking: 3 example 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 (football, hockey, etc.)

And the future has more

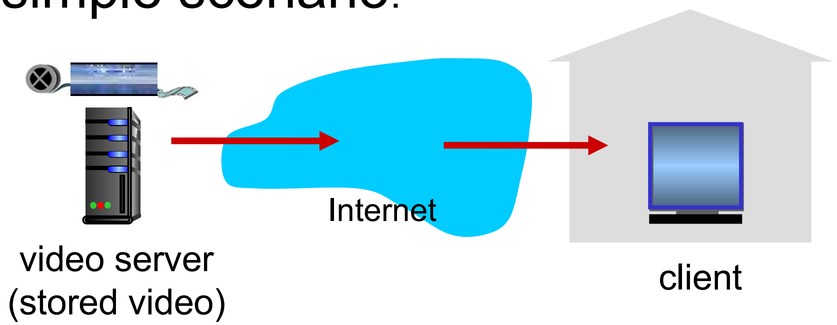
Streaming stored Video



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

Streaming stored Video

simple scenario:



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

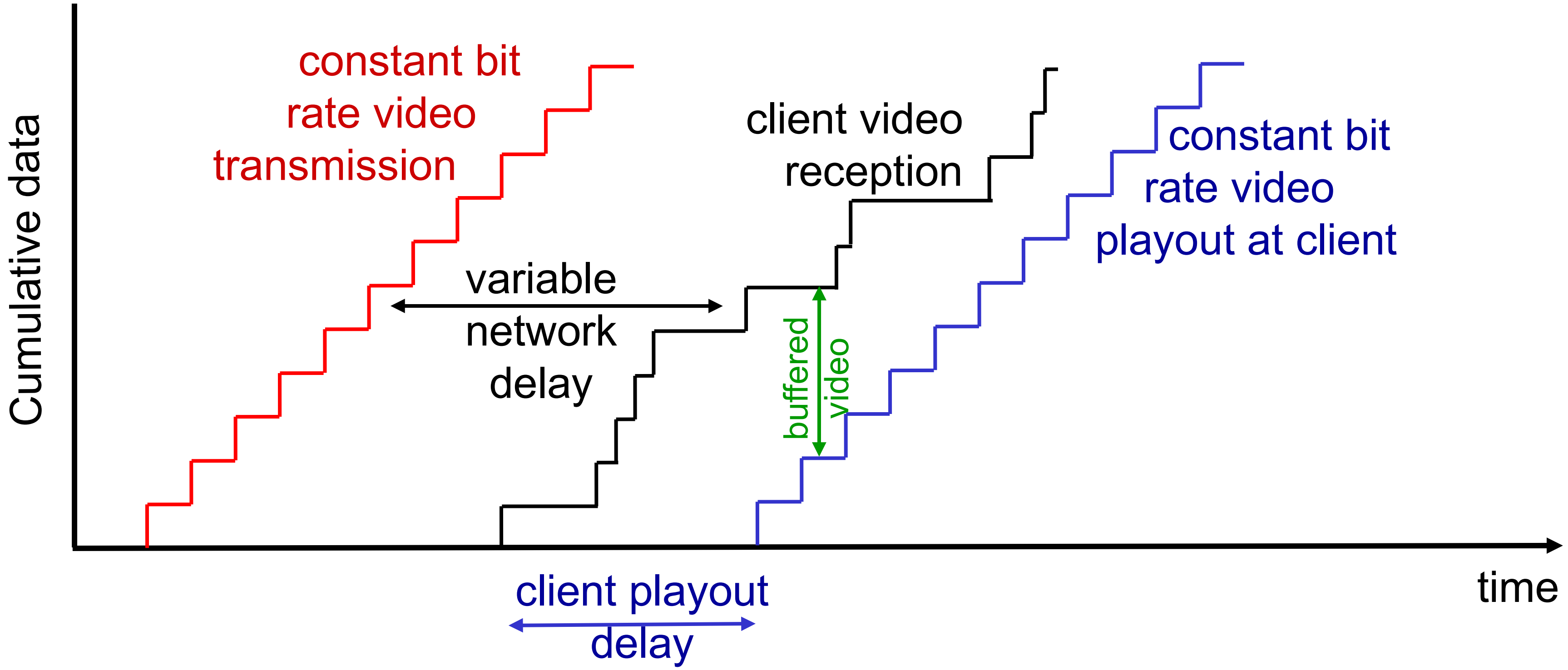
Main challenges:

- server-to-client bandwidth will *vary* over time, with changing network congestion levels (in house, access network, network core, video server)
- packet loss, delay due to congestion will delay playout, or result in poor video quality

Other challenges

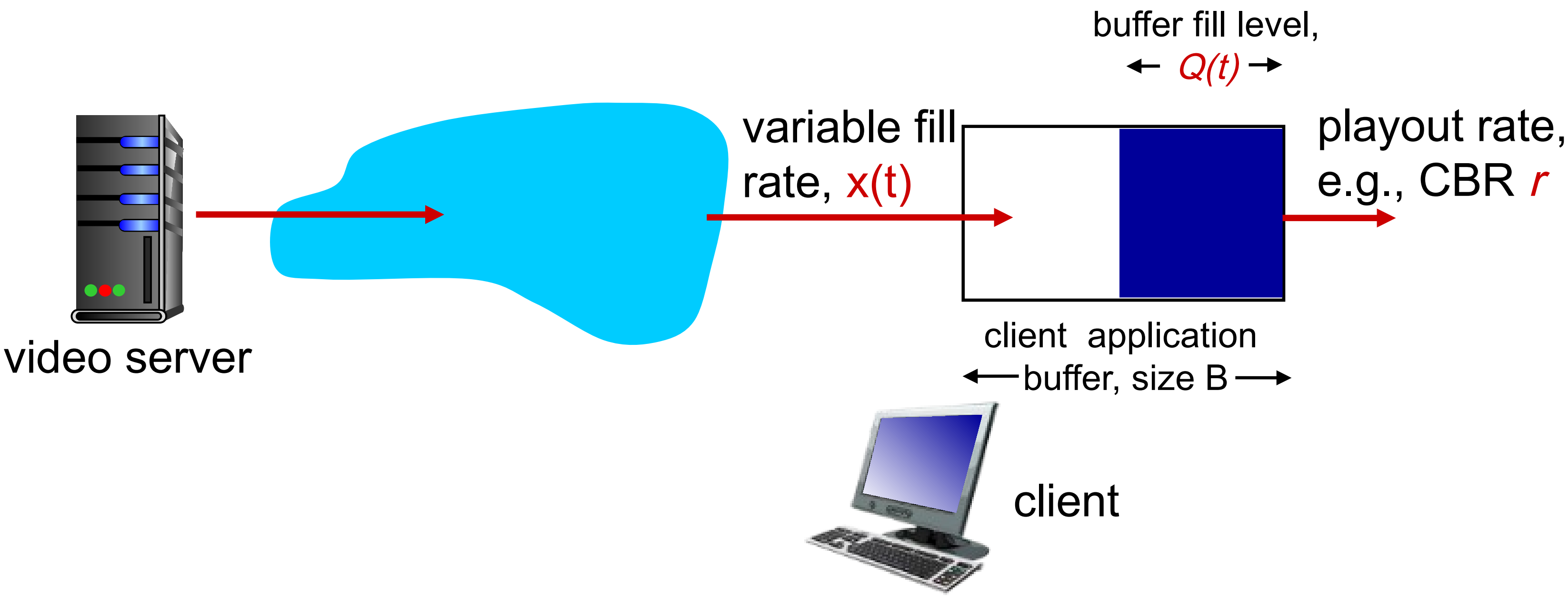
- client interactivity: pause, fast-forward, rewind, jump through video
- video packets may be lost, retransmitted

Streaming stored video: revisited

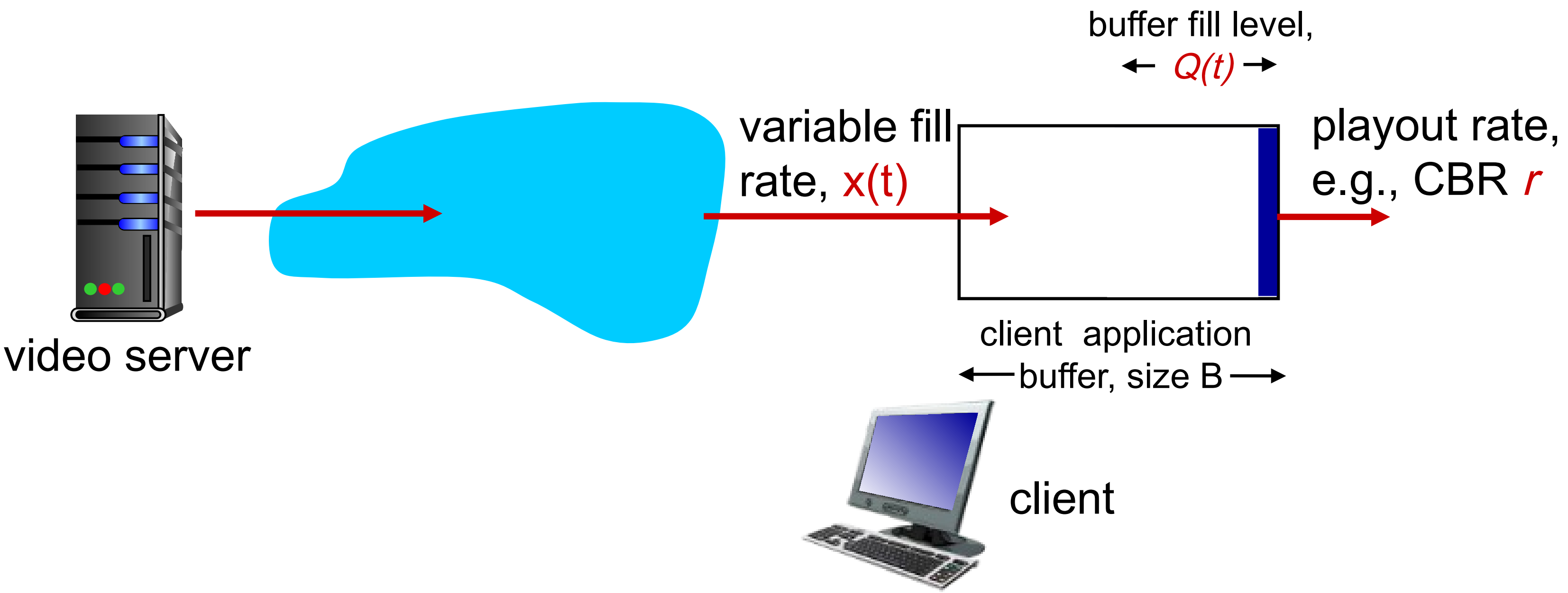


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, ~~playout~~



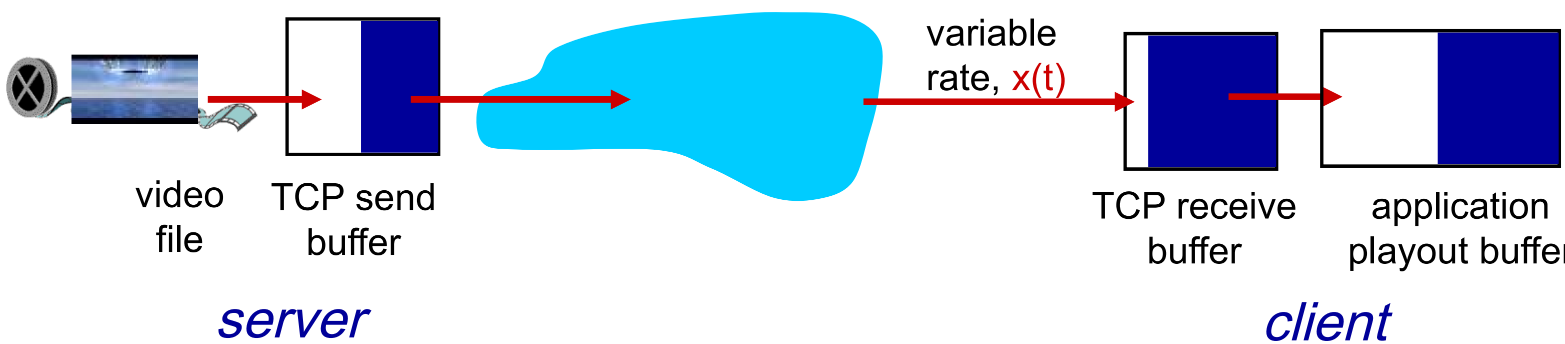
Client-side buffering, ~~playout~~



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

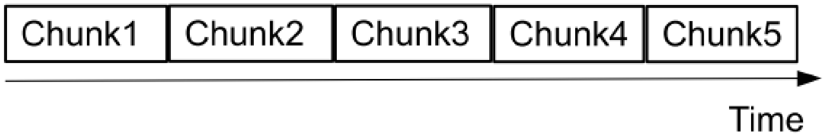
Streaming multimedia: ~~Basic HTTP~~

- multimedia file retrieved via HTTP GET
- send at maximum possible rate under TCP



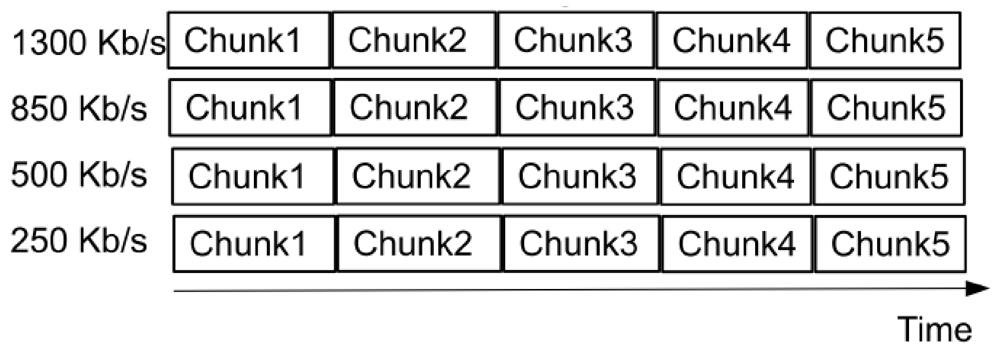
- fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)
- larger playout delay: smooth TCP delivery rate

HTTP-based Adaptive Streaming (HAS) or DASH …



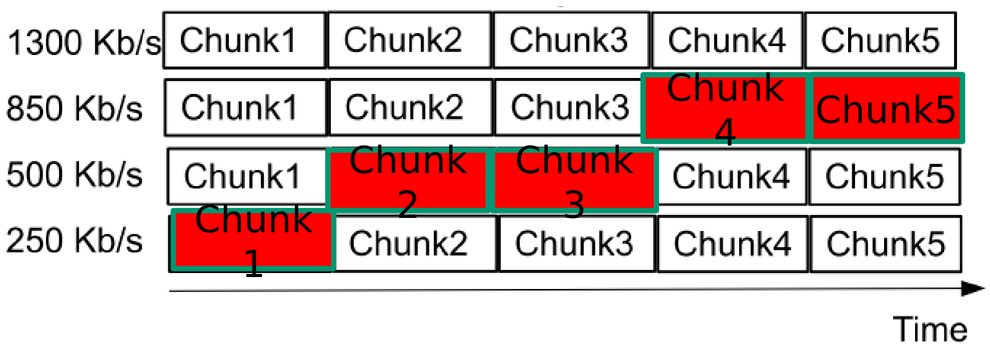
- HTTP-based ~~adaptive~~ streaming
  - Video is split into chunks

HTTP-based Adaptive Streaming (HAS) or DASH



- HTTP-based **adaptive** streaming
  - Video is split into chunks
  - Each chunk in multiple bitrates (qualities)

HTTP-based Adaptive Streaming (HAS) or DASH



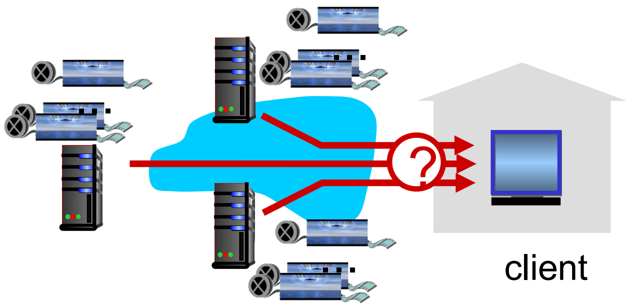
- HTTP-based **adaptive** streaming
  - Video is split into chunks
  - Each chunk in multiple bitrates (qualities)
  - Clients adapt quality encoding based on buffer/network conditions

Streaming multimedia: DASH (or HAS)

*D*ynamic, *A*daptive *S*treaming over *H*TTP

server:

- divides video file into multiple chunks
- each chunk encoded at multiple different rates
- different rate encodings stored in different files
- files replicated in various CDN nodes
- *manifest file:* provides URLs for different chunks

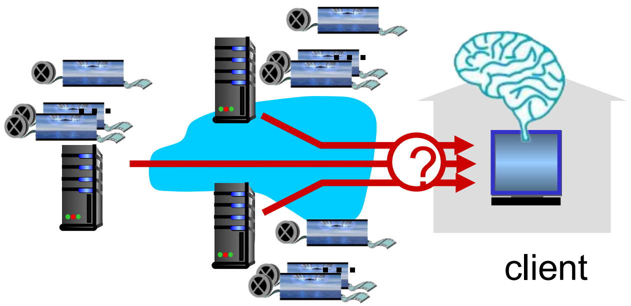


client:

- periodically estimates 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), and from different servers

Streaming multimedia: DASH (or HAS)

- *"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)



Streaming video = encoding + DASH + playout buffering

Content Distribution Networks (CDNs)

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 (and possibly congested) path to distant clients

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

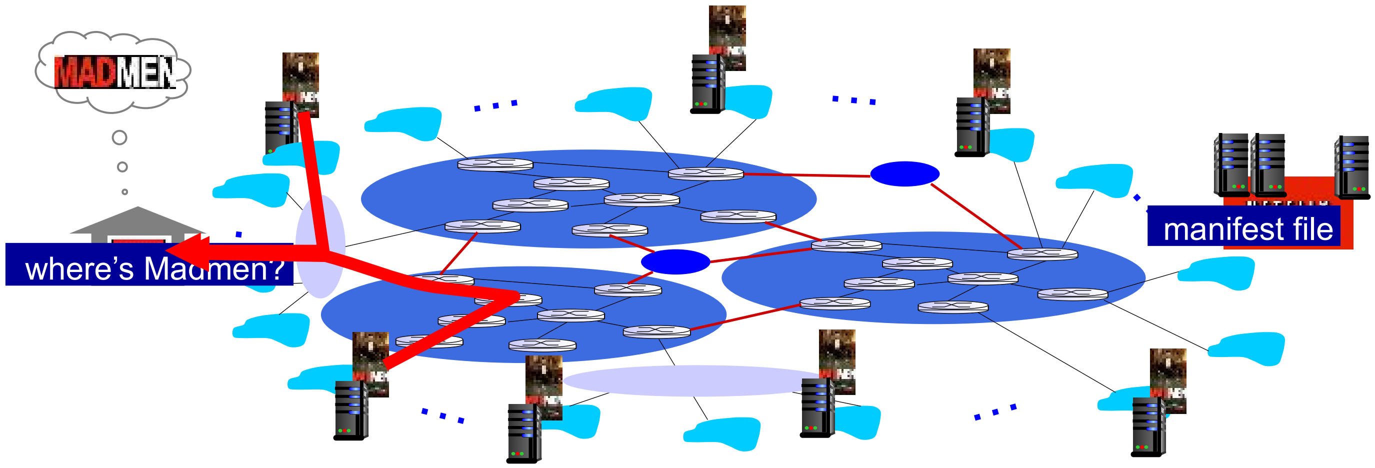
Content Distribution Networks (CDNs)

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
    - Akamai: 240,000 servers deployed in > 120 countries (2015)
  - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
    - used by Limelight

Content Distribution Networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes
- subscriber 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

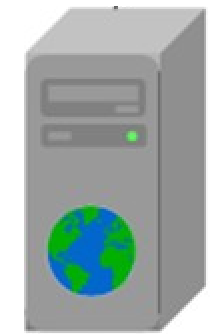


Content Distribution Networks (CDNs)

OTT challenges: coping with a congested Internet from the "edge"

- what content to place in which CDN node?
- from which CDN node to retrieve content? At which rate?

Example: HAS and proxy



Clients' want

- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

Example: HAS and proxy

Slides from: V. Krishnamoorthi et al. "Helping Hand or Hidden Hurdle: Proxy-assisted HTTP-based Adaptive Streaming Performance", Proc. IEEE MASCOTS, 2013

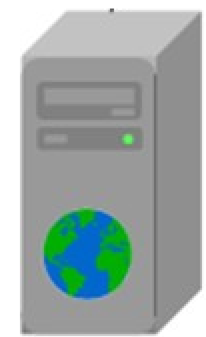
Clients' want

- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

HAS is increasingly responsible for larger traffic volumes
… proxies to reduce traffic??

Example: HAS and proxy

Slides from: V. Krishnamoorthi et al. "Helping Hand or Hidden Hurdle: Proxy-assisted HTTP-based Adaptive Streaming Performance", Proc. IEEE MASCOTS, 2013



Clients' want

- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality

Network providers' want

- High QoE of customers/clients

Example: HAS and proxy

Slides from: V. Krishnamoorthi et al. "Helping Hand or Hidden Hurdle: Proxy-assisted HTTP-based Adaptive Streaming Performance", Proc. IEEE MASCOTS, 2013

Clients' want

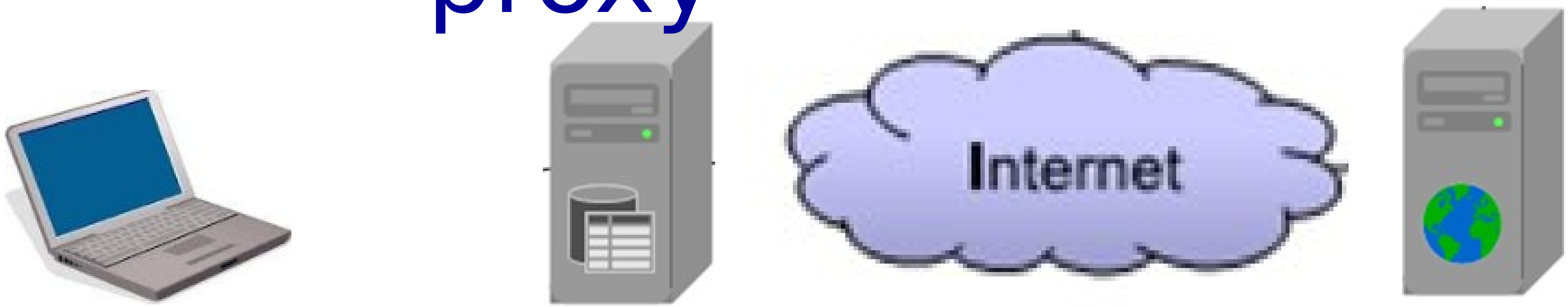
- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

Network providers' want

- High QoE of customers/clients
- Low bandwidth usage
- High hit rate

Example: HAS and proxy

Slides from: V. Krishnamoorthi et al. "Helping Hand or Hidden Hurdle: Proxy-assisted HTTP-based Adaptive Streaming Performance", Proc. IEEE MASCOTS, 2013



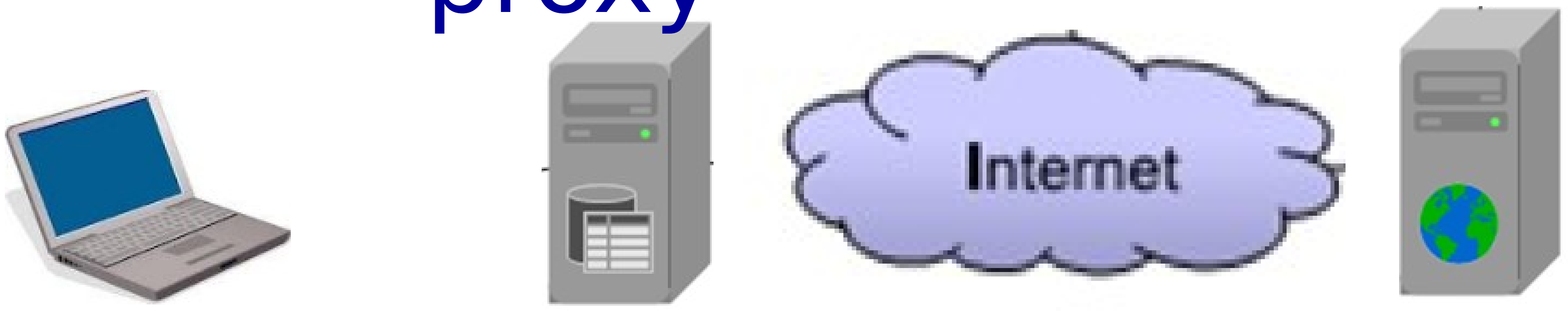
Clients' want

- High playback quality
- Small stall times
- Few buffer interruptions
- Few quality switches

Network providers' want

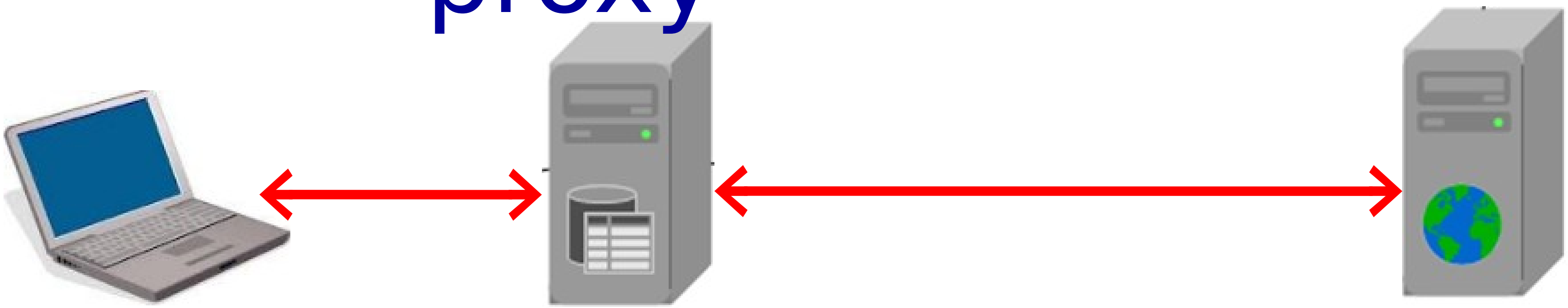
- High QoE of customers/clients
- Low bandwidth usage
- High hit rate

Example: HAS and proxy

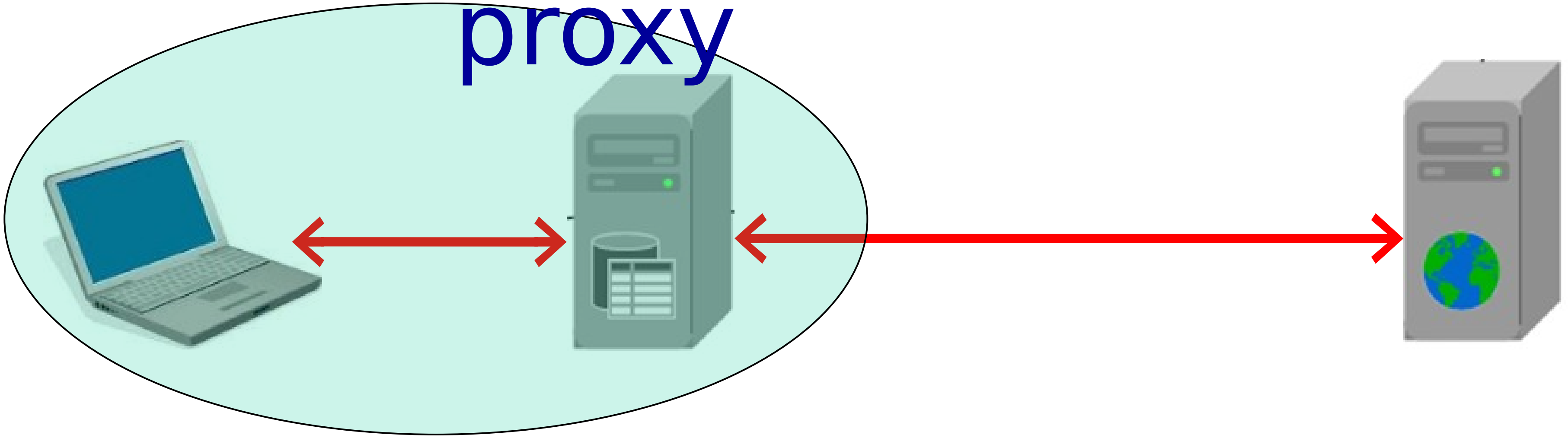


Proxy example …

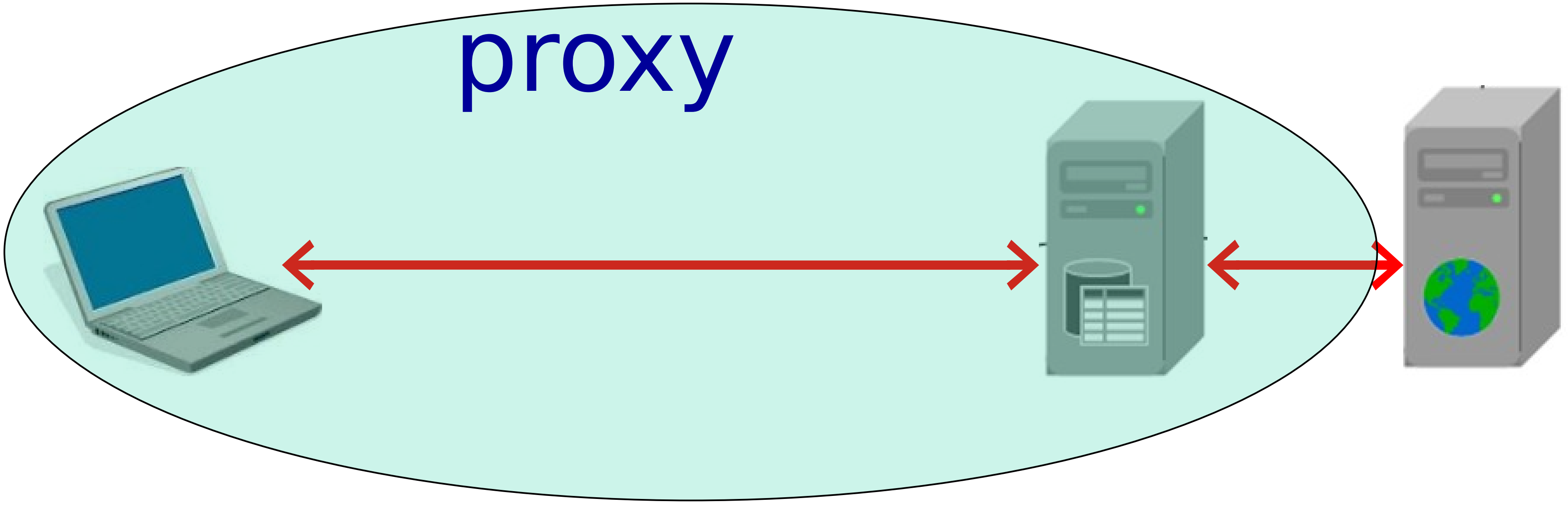
Example: HAS and proxy



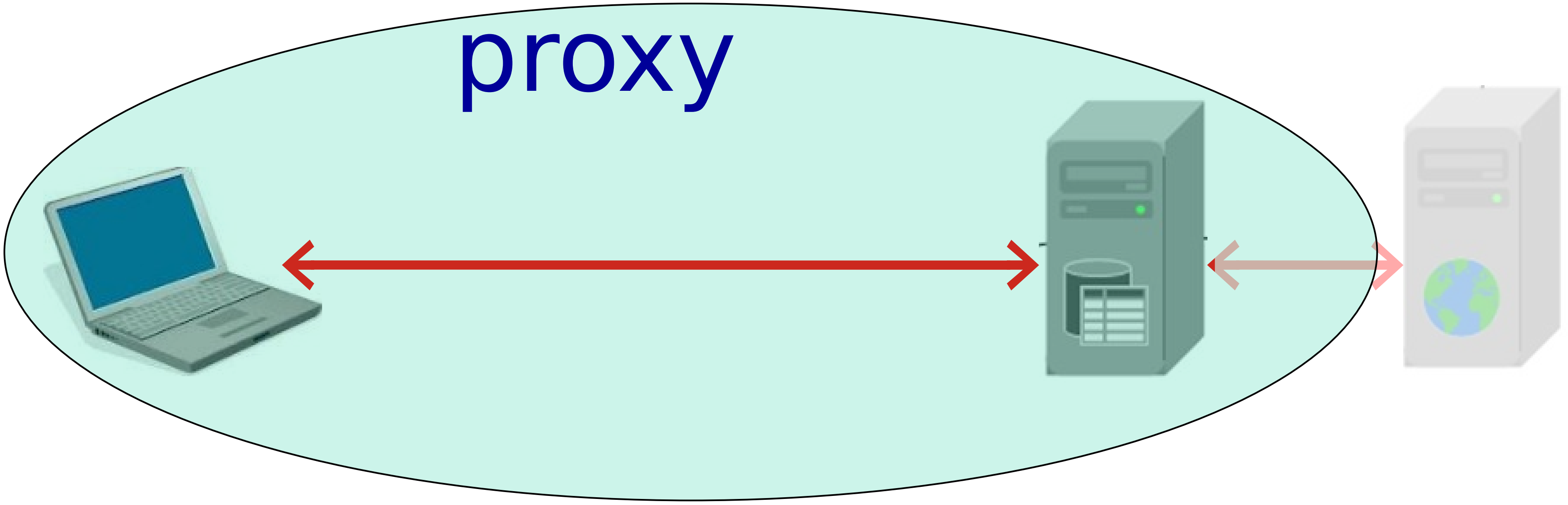
Example: HAS and proxy



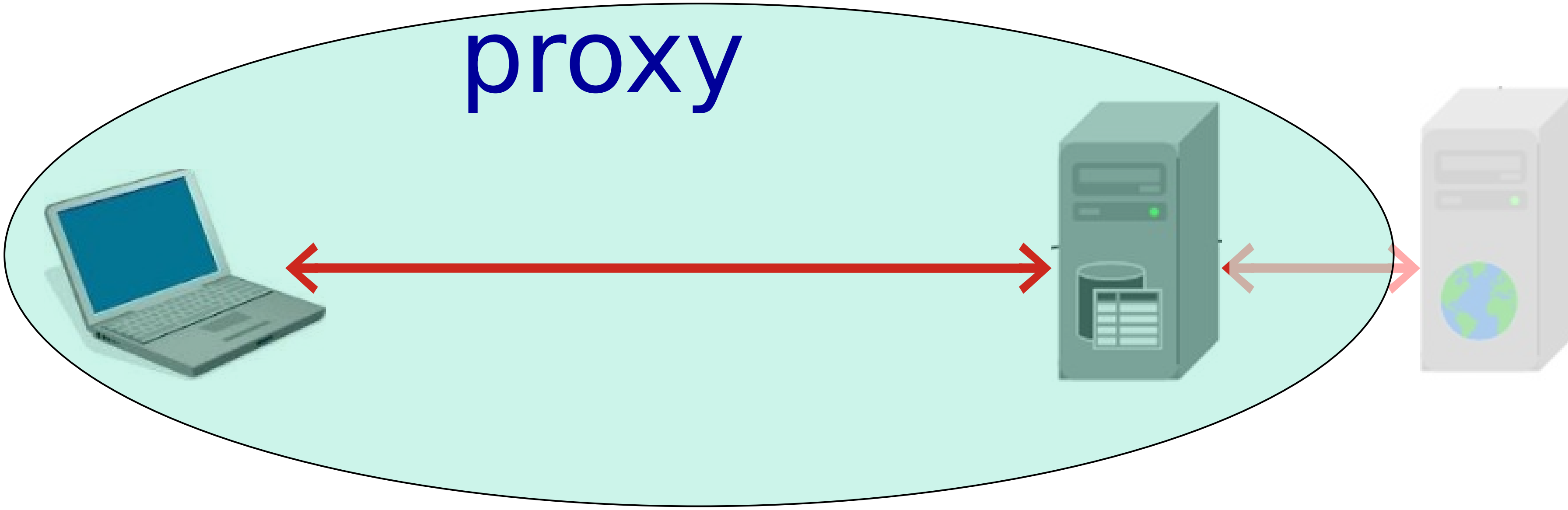
Example: HAS and proxy

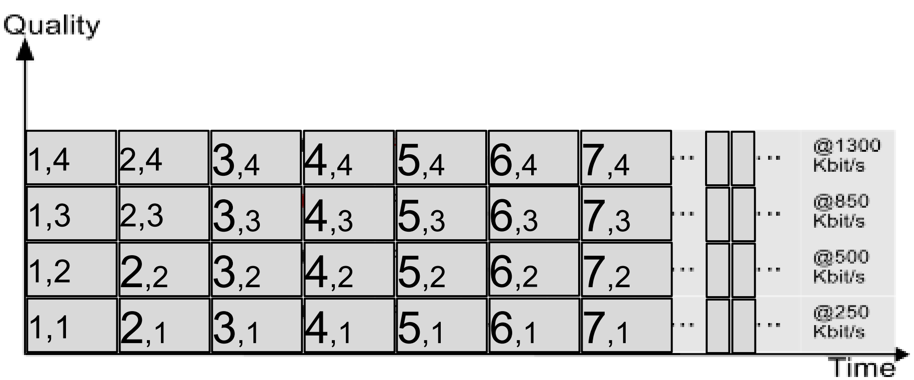


Example: HAS and proxy

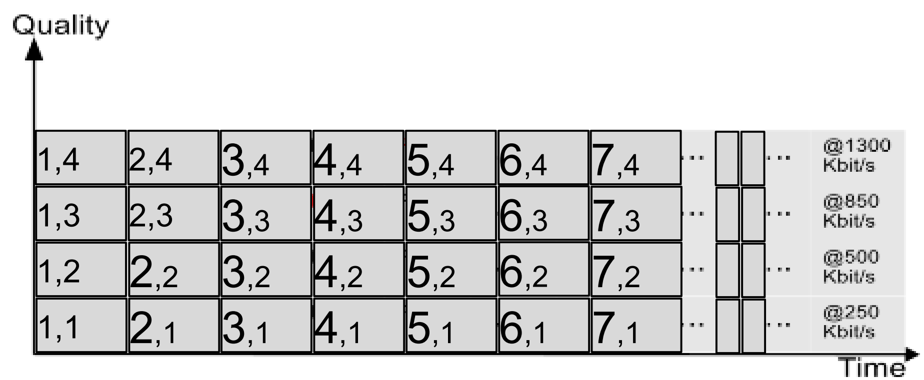


Example: HAS and proxy



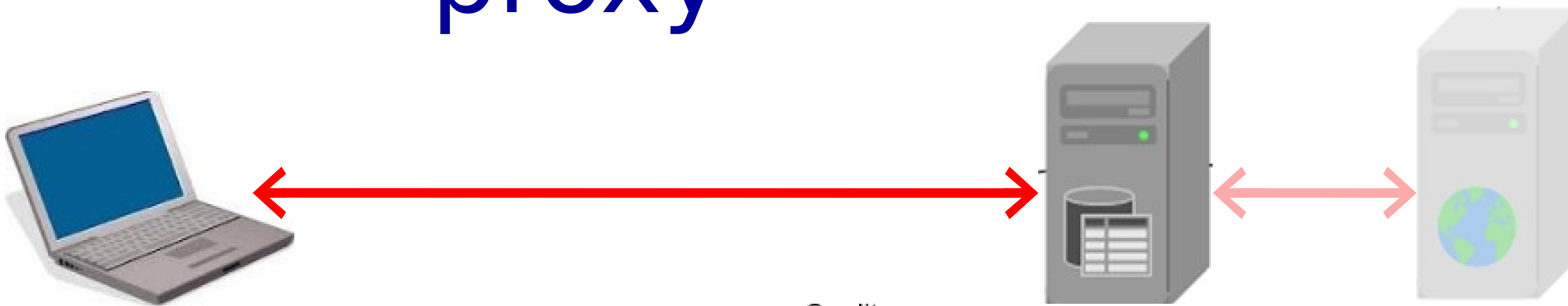


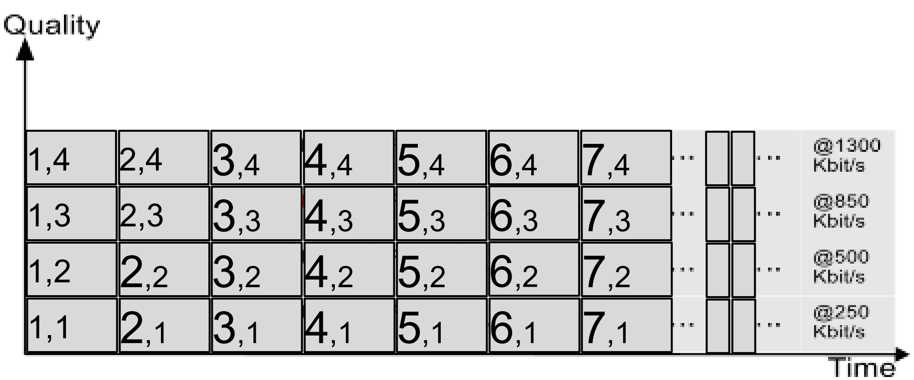
Client



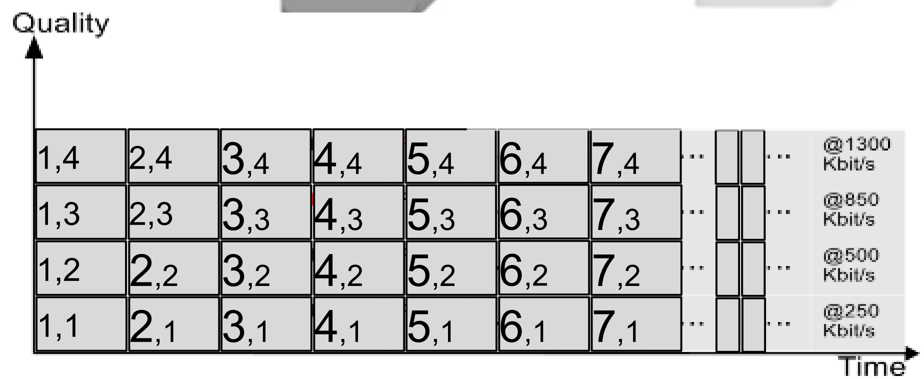
Proxy

Example: HAS and proxy

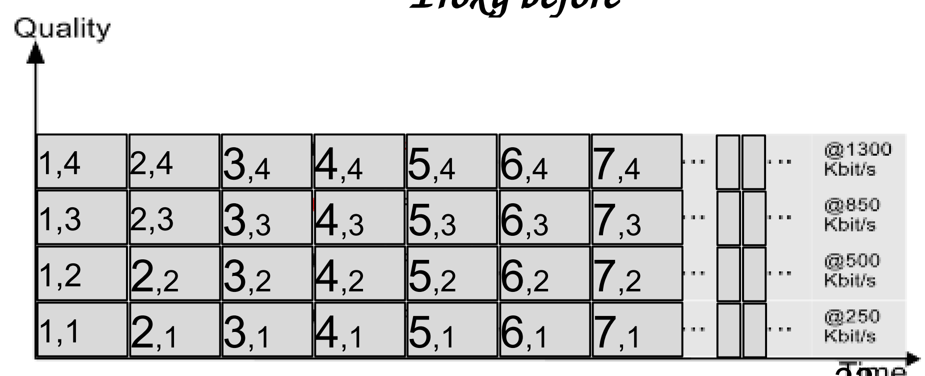




Client 1

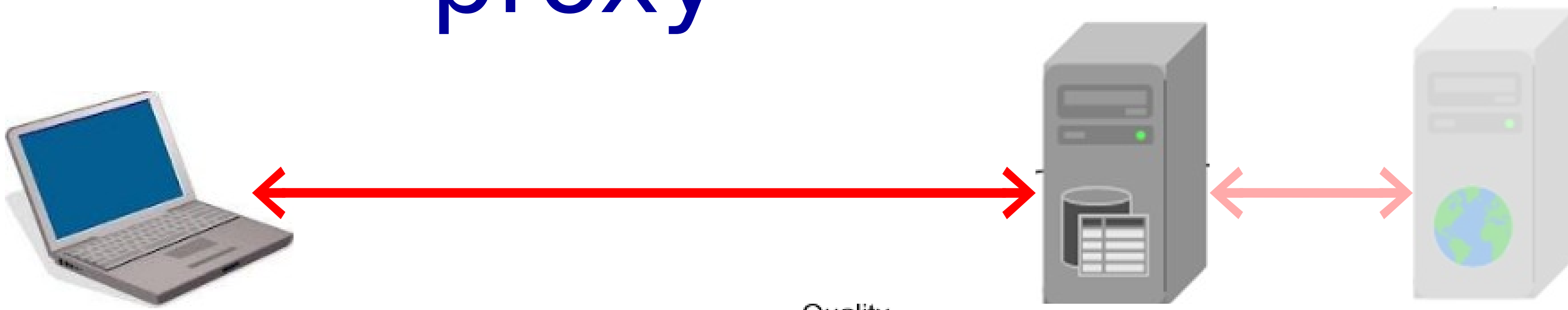


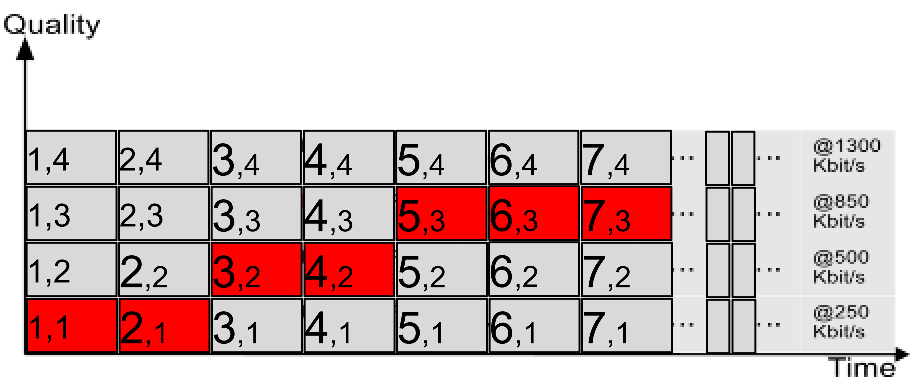
Proxy before



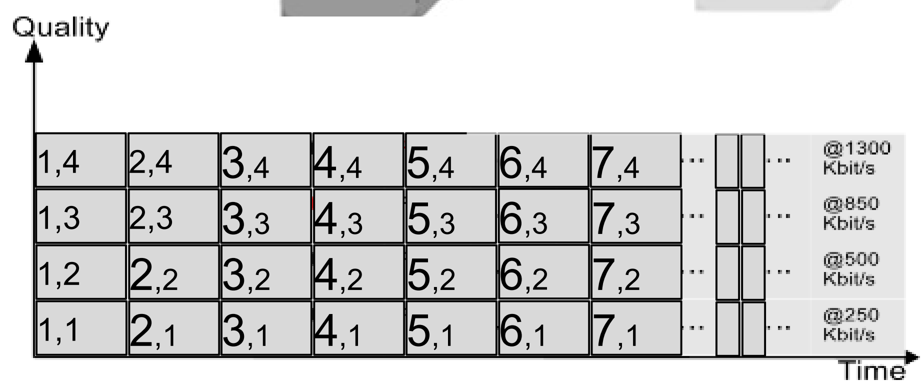
Proxy after

Example: HAS and proxy

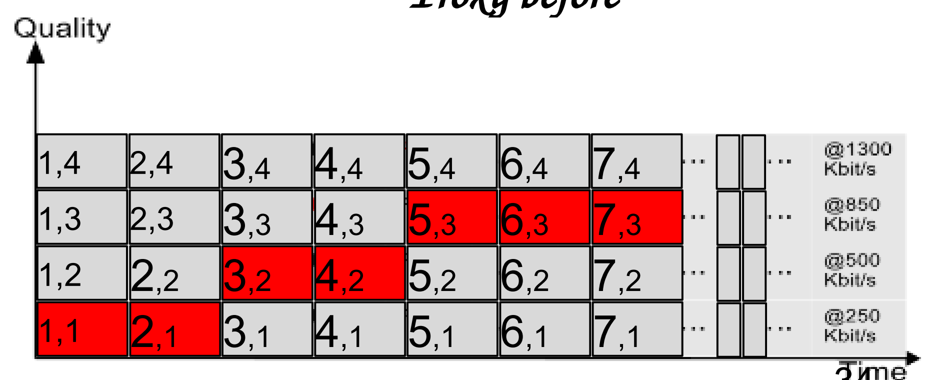




Client 1

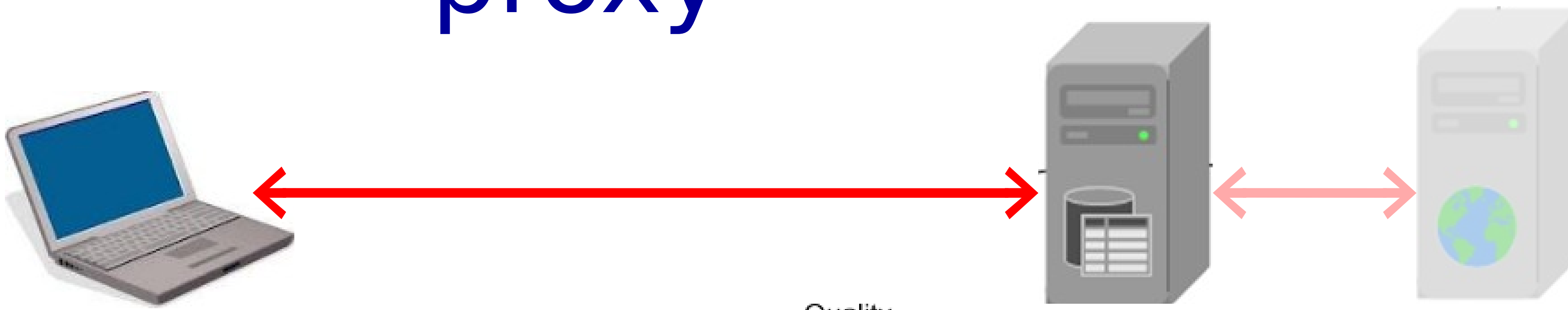


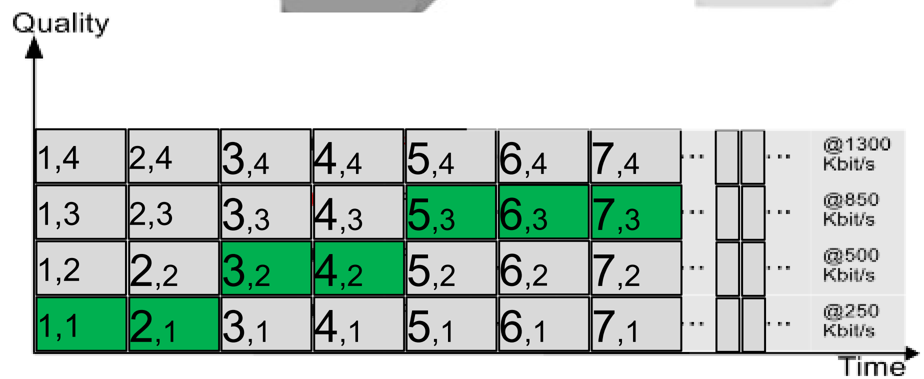
Proxy before



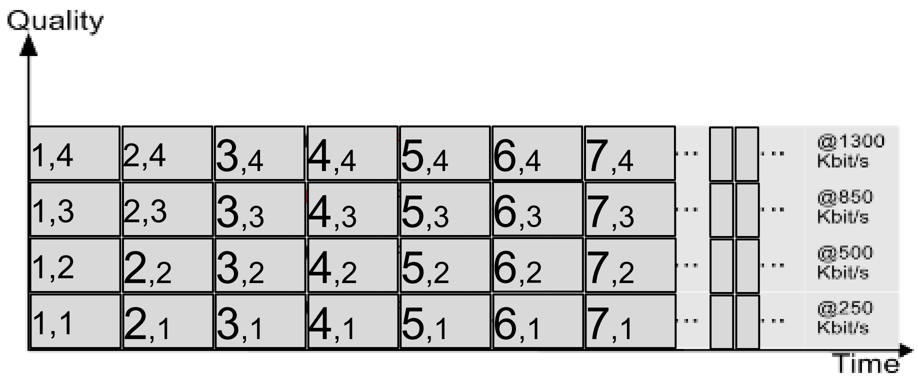
Proxy after

Example: HAS and proxy

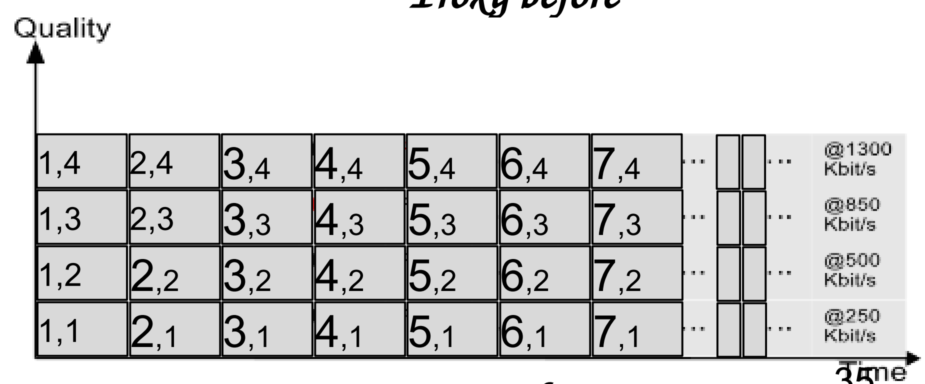




Proxy before

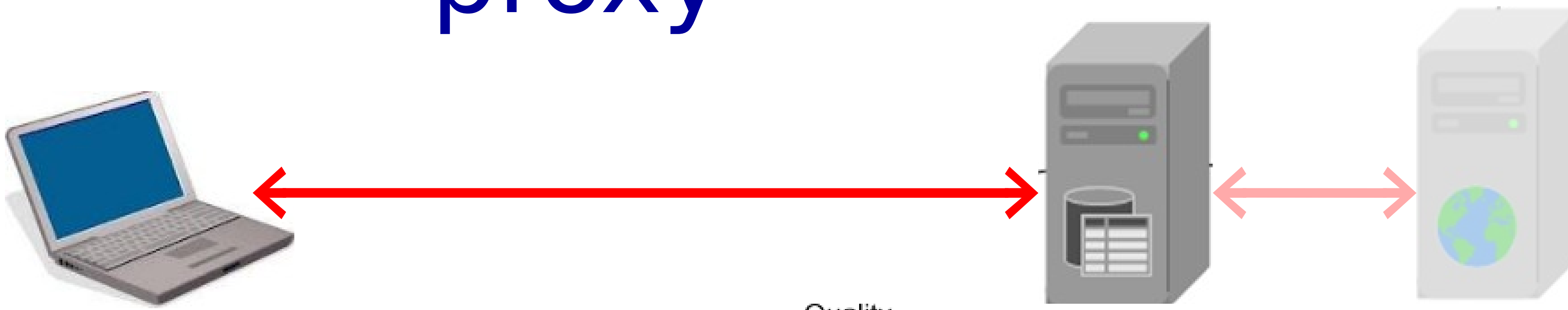


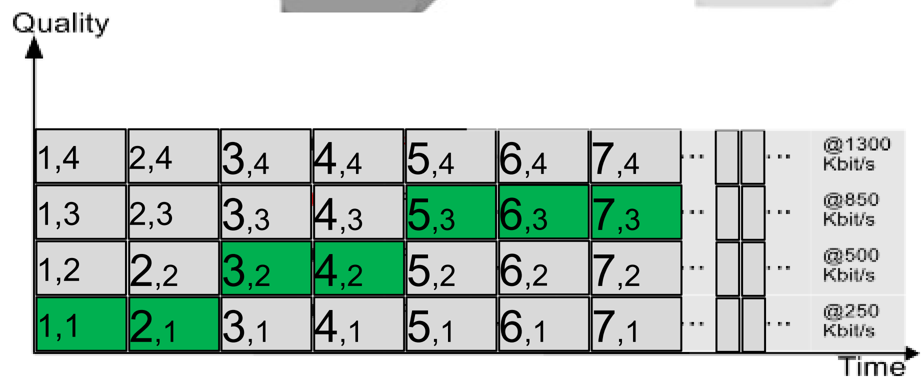
Client 2



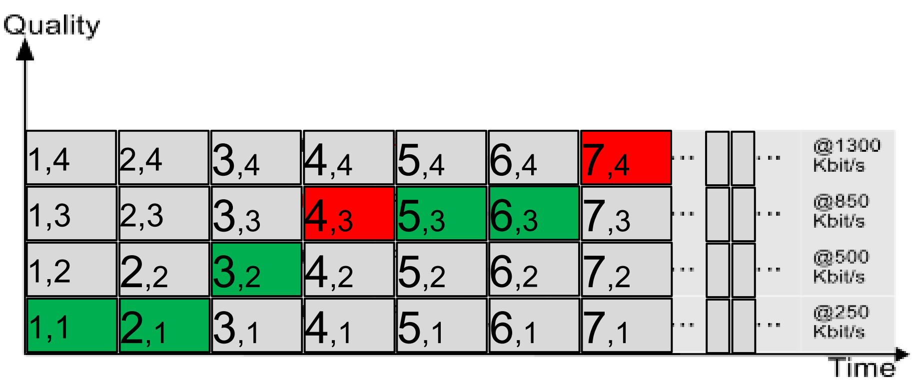
Proxy after

Example: HAS and proxy

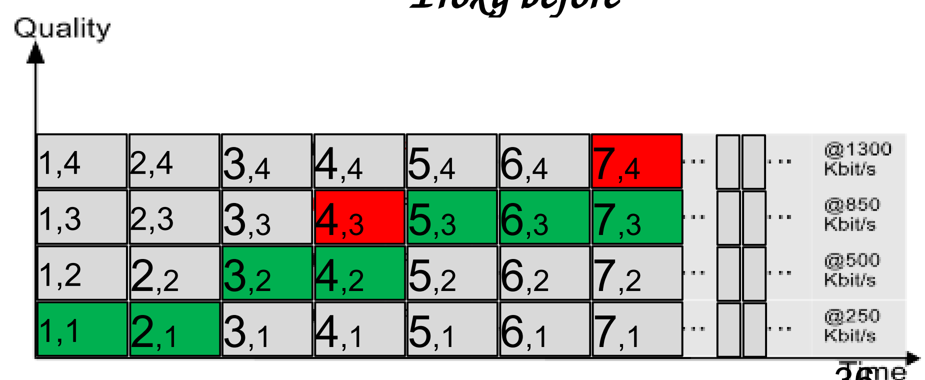




Proxy before

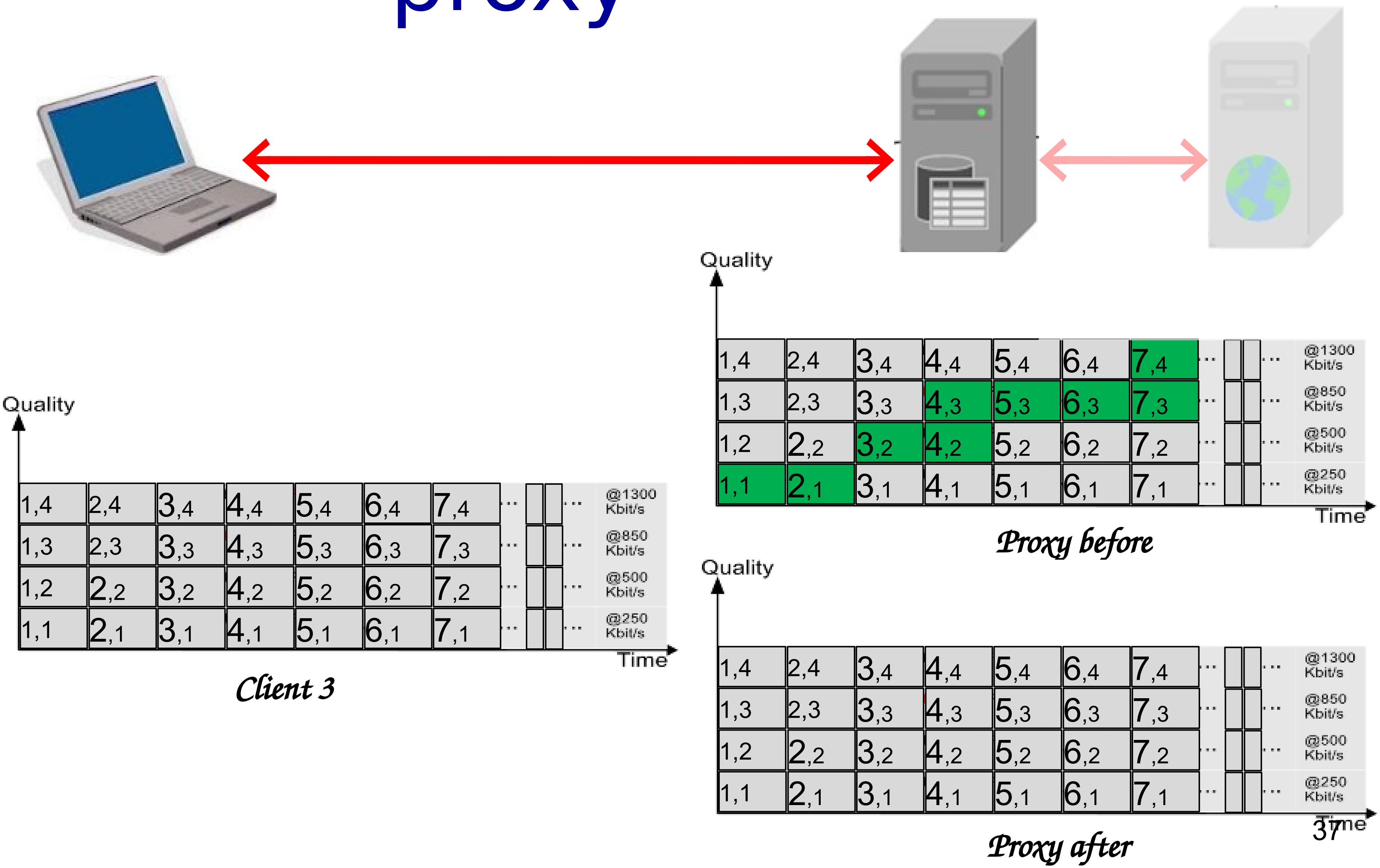


Client 2

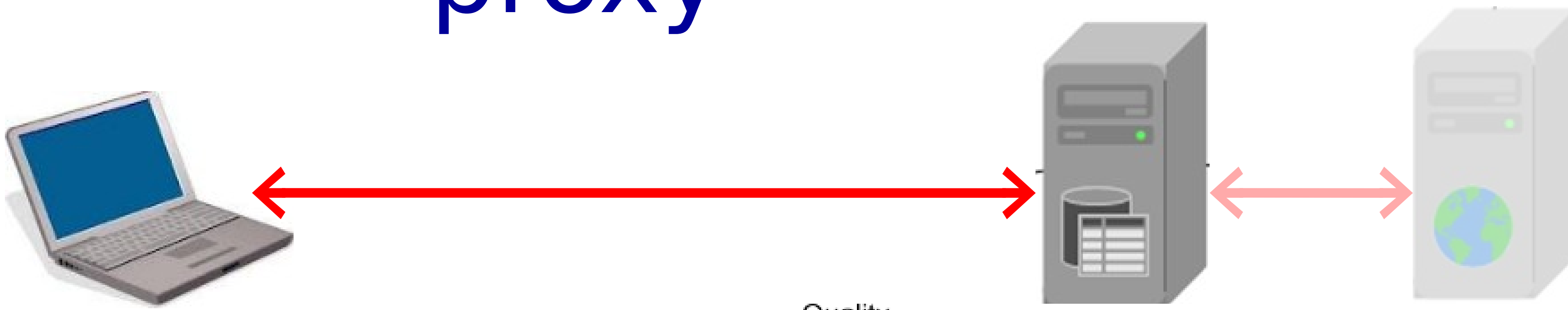


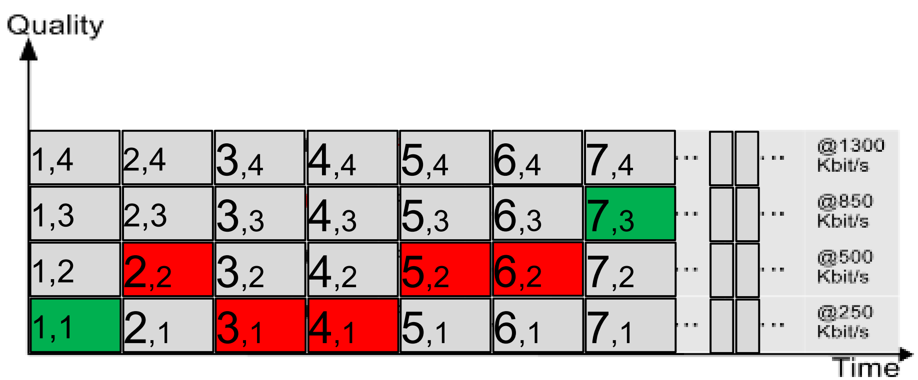
Proxy after

Example: HAS and proxy

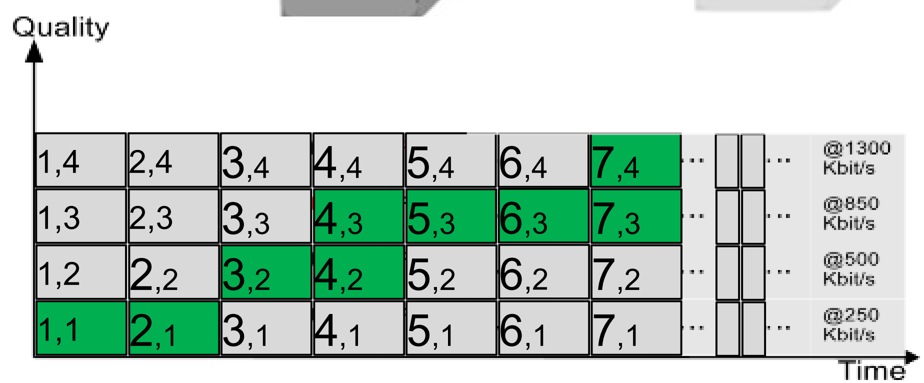


Example: HAS and proxy

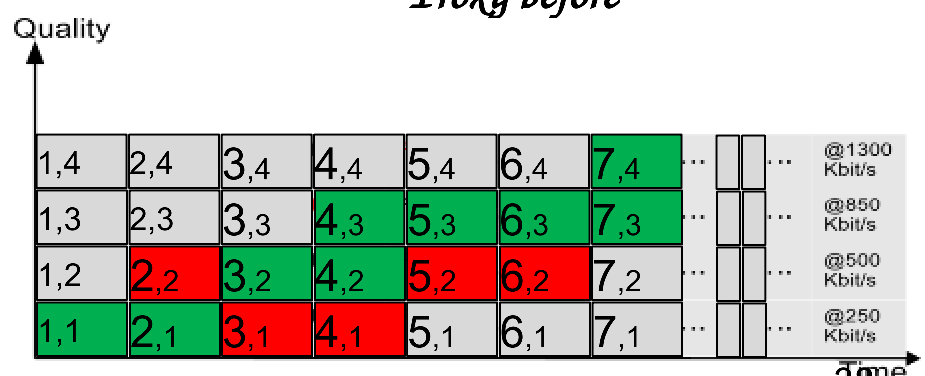




Client 3



Proxy before



Proxy after

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 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

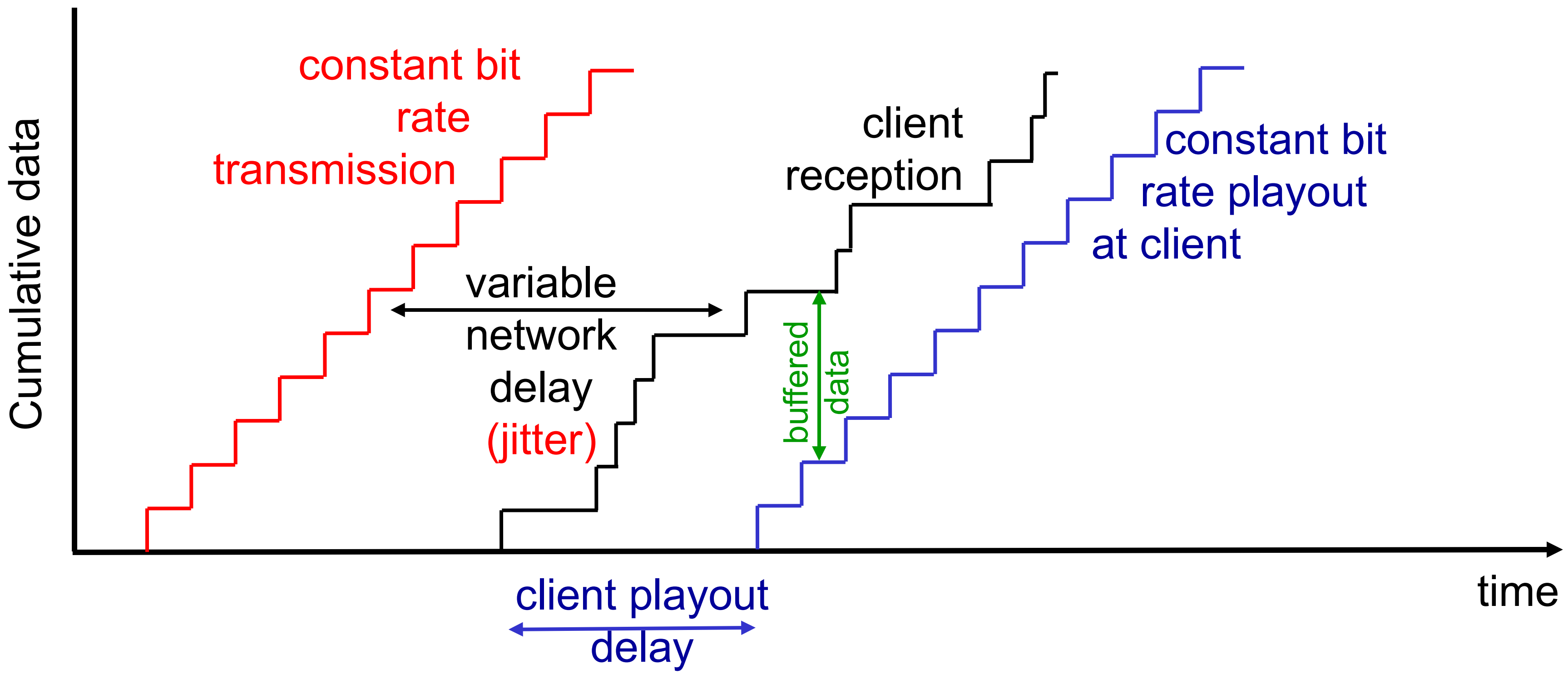
VoIP characteristics

- 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

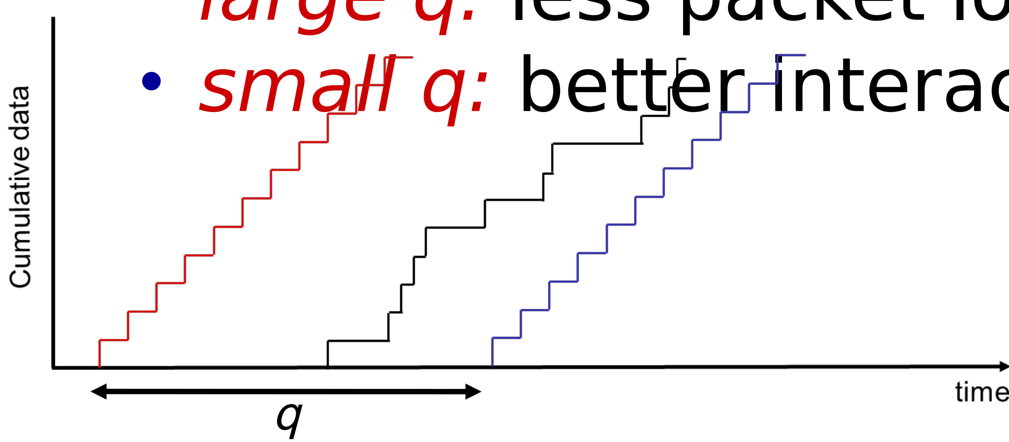
Delay jitter



- 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



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
- E.g., adaptively estimate packet delay: Exponentially weighted moving average (EWMA) [recall TCP RTT estimate]:

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

time received - time sent (timestamp)

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)

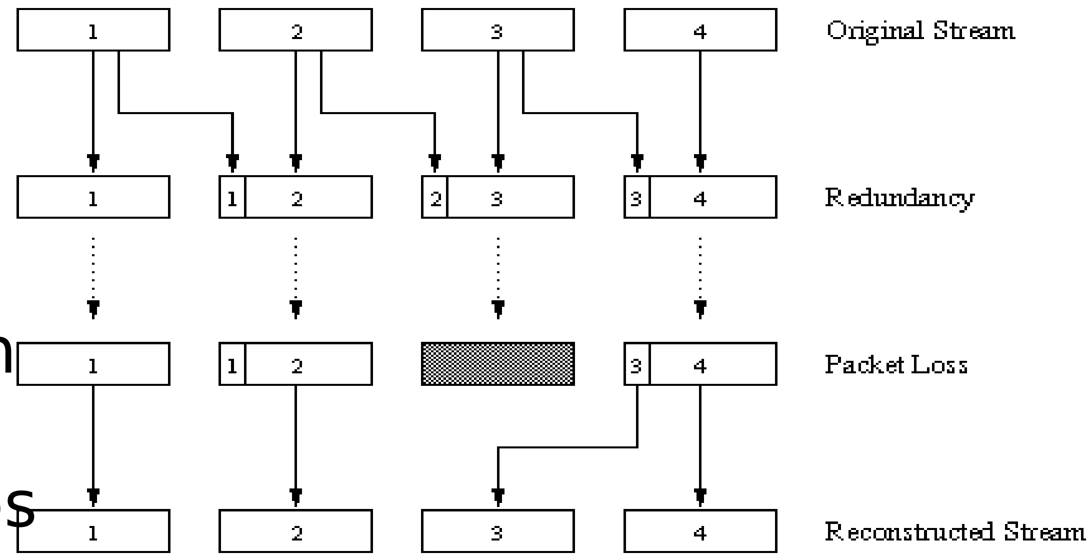
simple FEC

- 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:

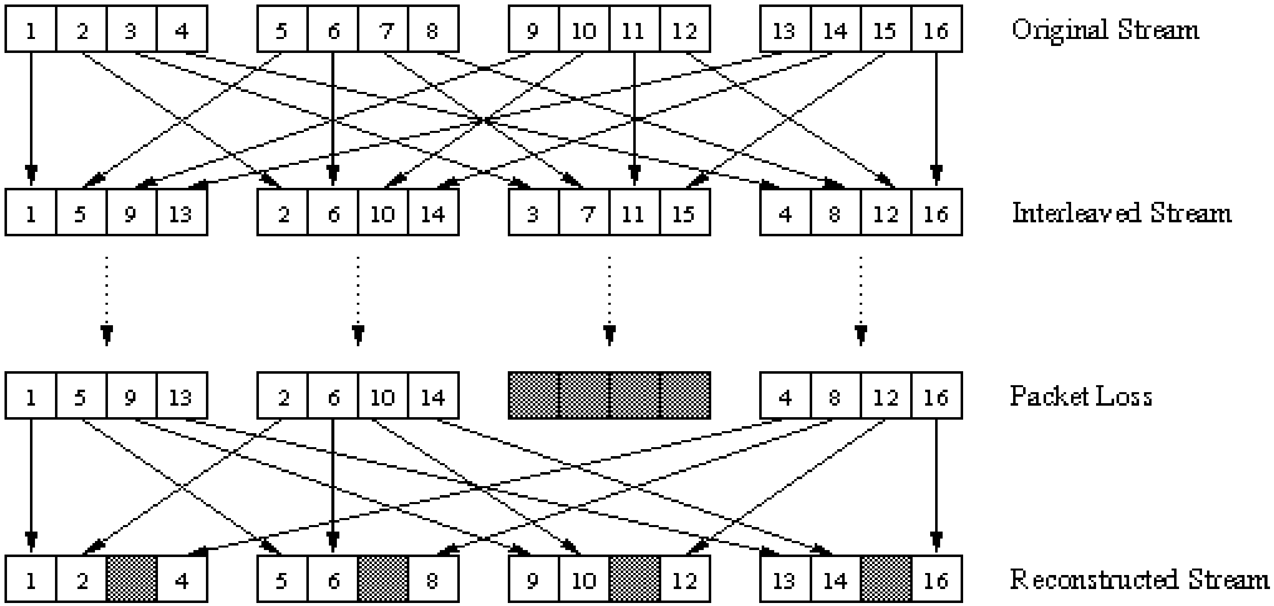
- “piggyback lower quality stream”
- send lower resolution audio stream as redundant information
- e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps



non-consecutive loss: receiver can conceal loss

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

VoIP: recovery from packet ~~loss (3)~~



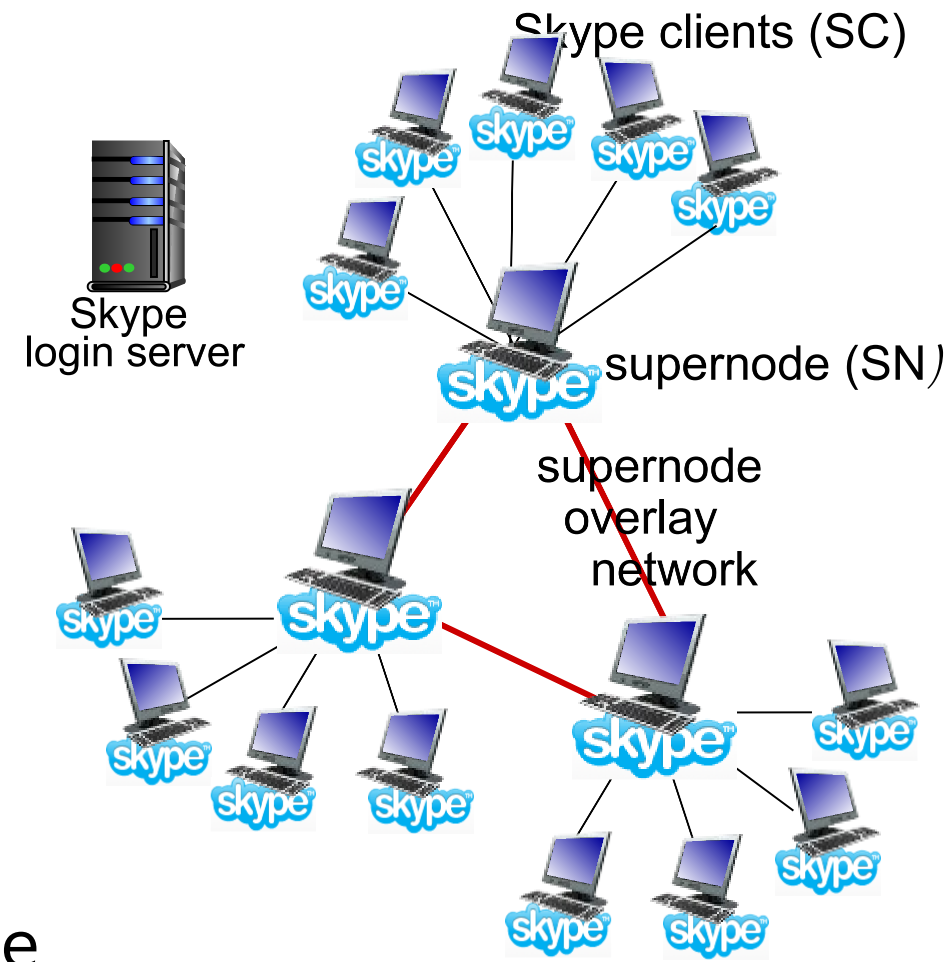
interleaving to conceal loss:

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

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

Voice-over-IP: Skype

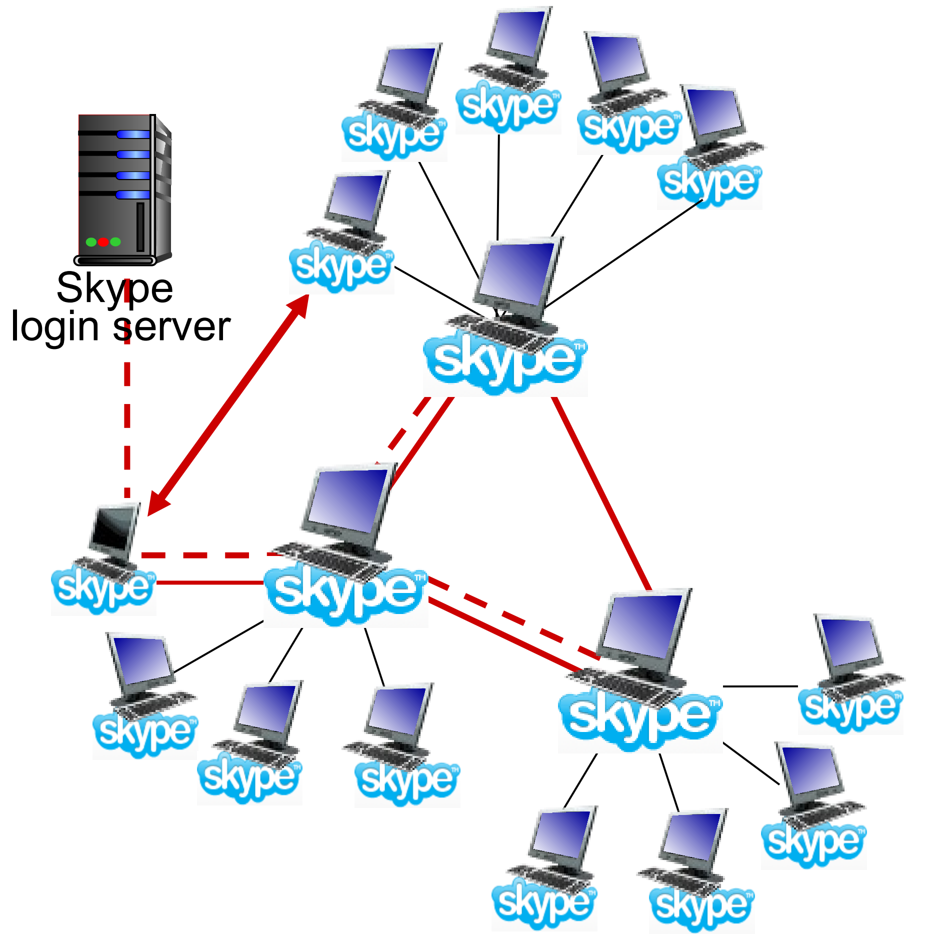
- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:
  - clients: Skype peers connect directly to each other for VoIP call
  - super nodes (SN): Skype peers with special functions
  - overlay network: among SNs to locate SCs
  - login server



P2P voice-over-IP: Skype

Skype client operation:

1. joins Skype network by contacting SN (IP address cached) using TCP
2. logs-in (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



Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

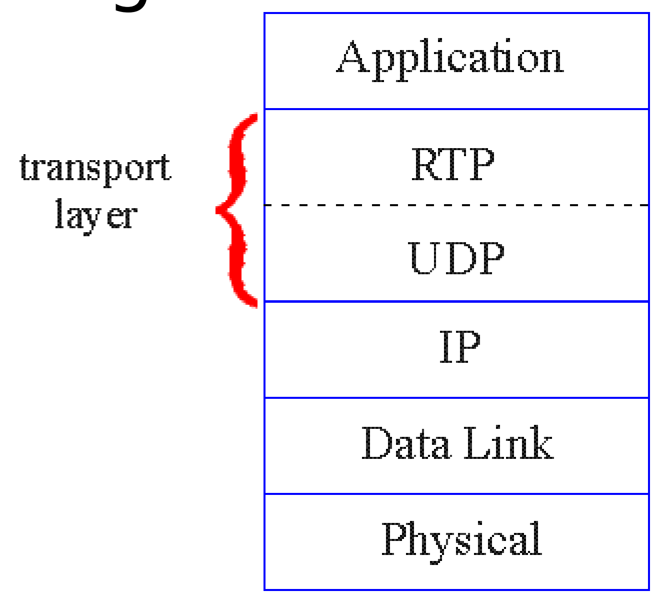
9.4 protocols for *real-time* conversational applications: RTP, SIP

9.5 network support for multimedia

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 header

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

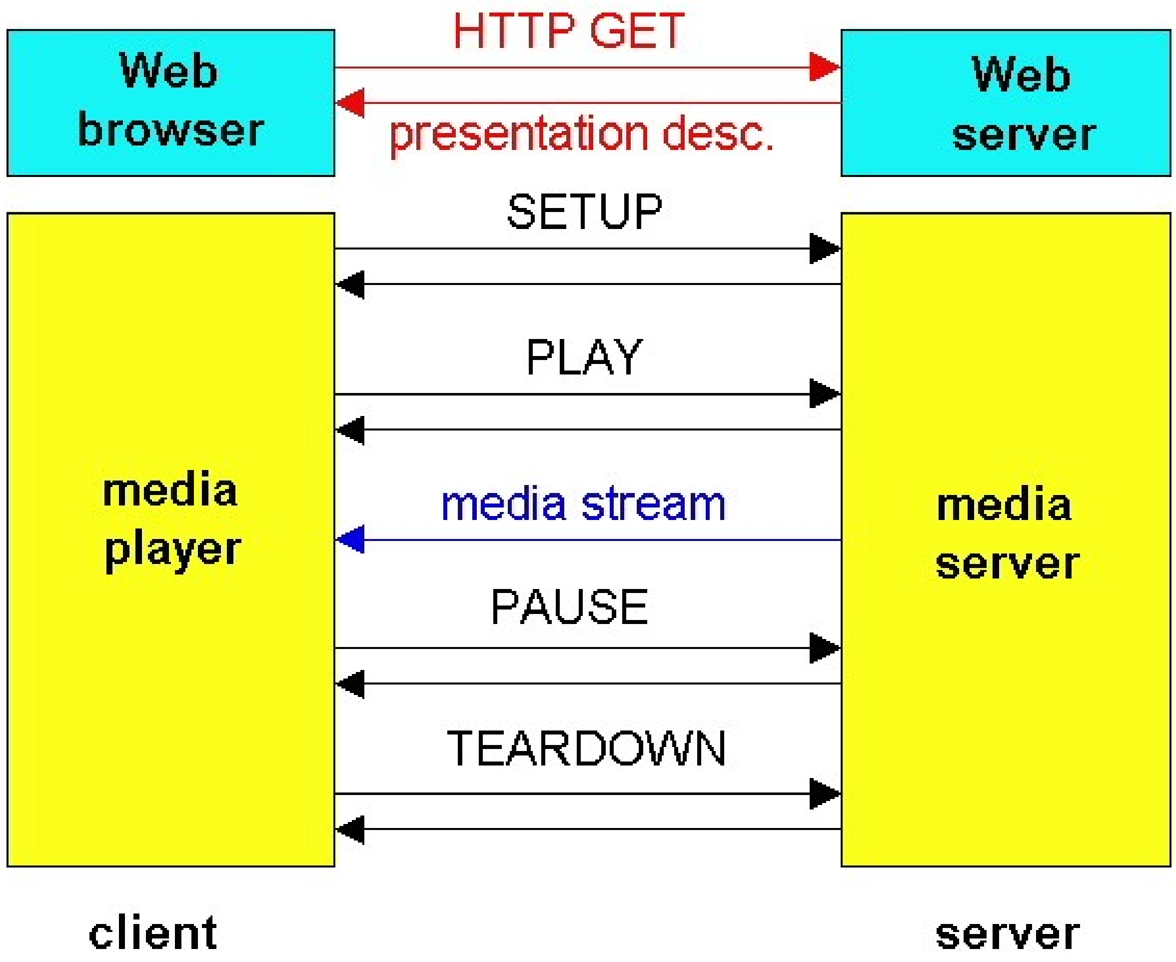
payload type (7 bits): indicates type of encoding currently being
used. If sender changes encoding during call, sender informs receiver via payload type field

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

- detect packet loss, restore packet sequence

timestamp field (32 bits long): sampling instant of first byte in this RTP data packet (e.g., for audio, timestamp clock increments by one for each sampling period)

RTSP Operation



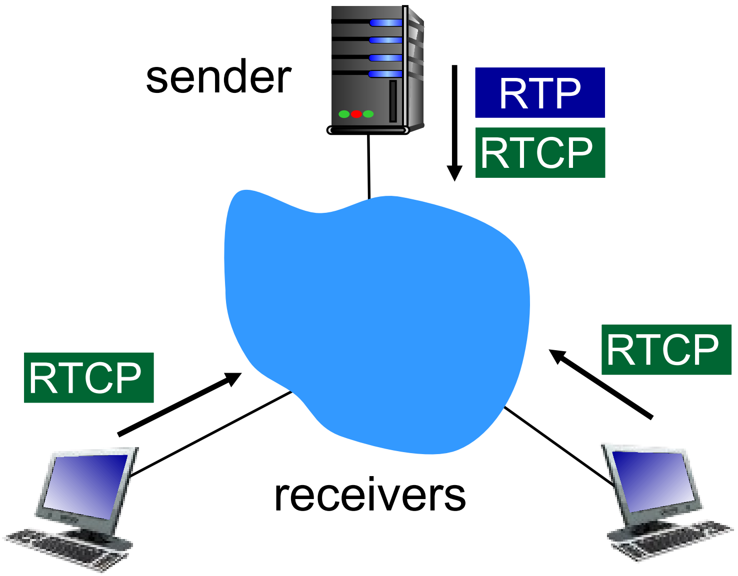
Control messages "out-of-band"
- port 554

Media stream "in-band".

Real-Time Control Protocol (RTCP)

- works in conjunction with RTP
- each participant in RTP session periodically sends RTCP control packets to all other participants
- each RTCP packet contains sender and/or receiver reports
  - report statistics useful to application: # packets sent, # packets lost, interarrival jitter
- feedback used to control performance
  - sender may modify its transmissions based on feedback

RTCP: multiple multicast senders



- E.g., RTCP attempts to limit its traffic to 5% of session bandwidth

receiver report packets:

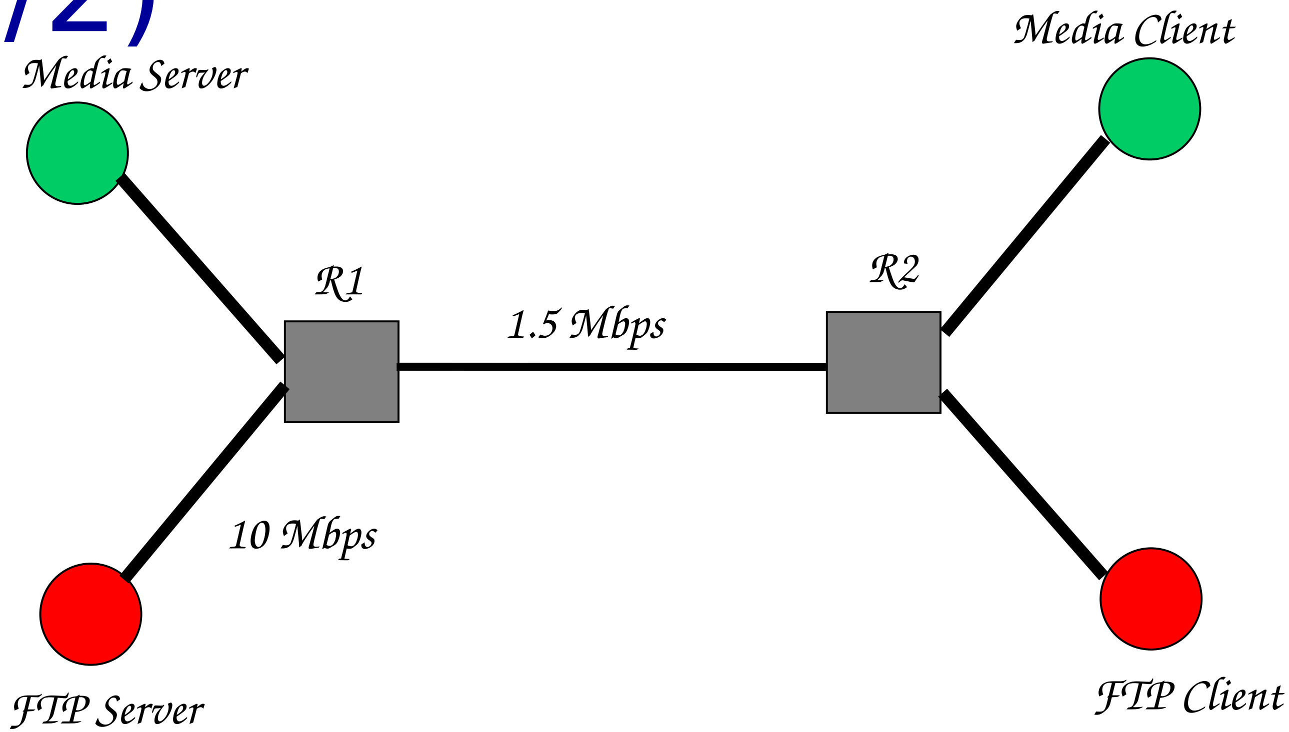
- fraction of packets lost, last sequence number, average interarrival jitter

sender report packets:

- SSRC of RTP stream, current time, number of packets sent, number of bytes sent

- each RTP session: typically a single multicast address; all RTP /RTCP packets belonging to session use multicast address
- RTP, RTCP packets distinguished from each other via distinct port numbers
- to limit traffic, each participant reduces RTCP traffic as number of conference participants increases

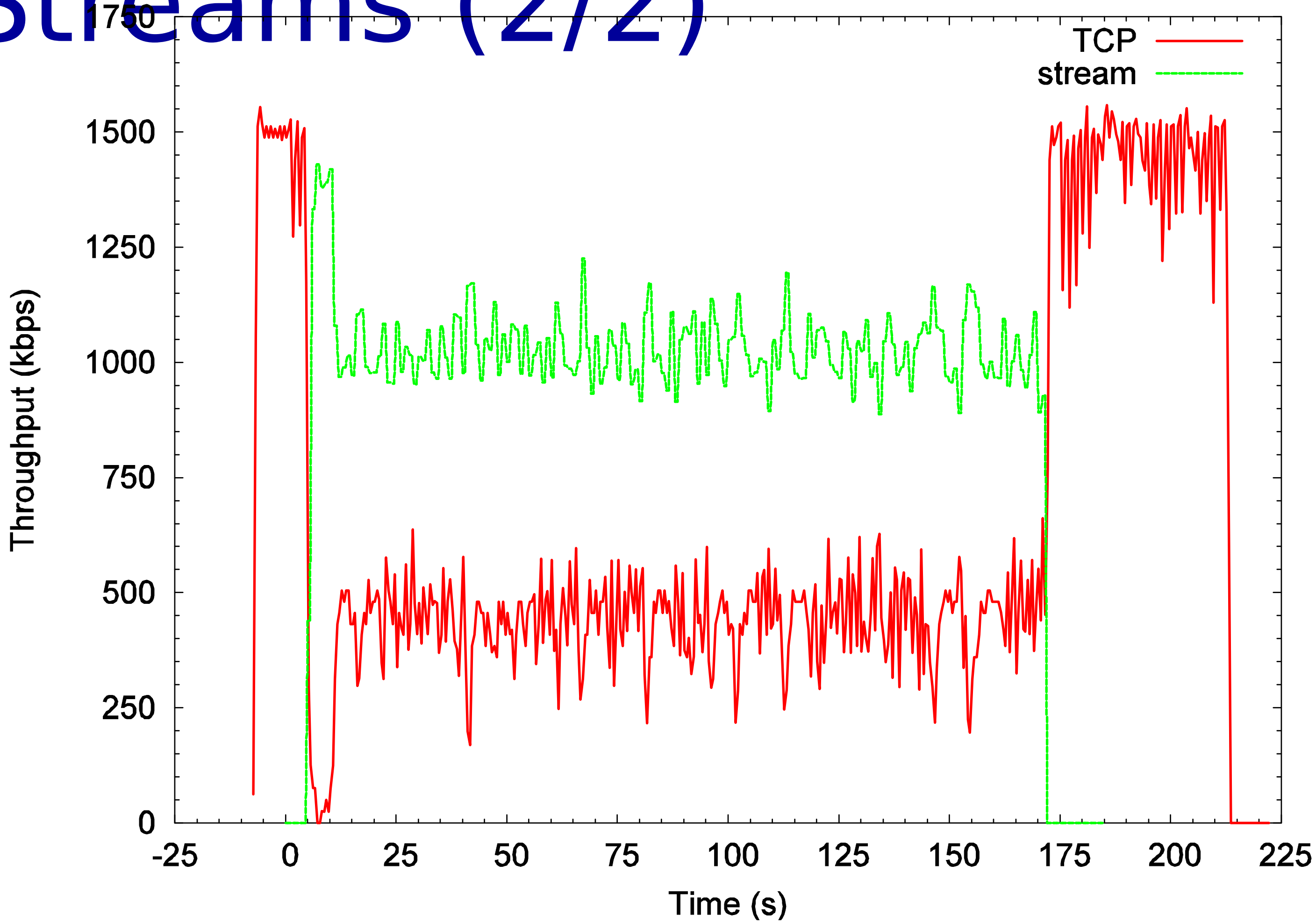
Fairness of UDP Streams (1/2)



- R1-R2 is the bottleneck link
- Streaming uses UDP at the transport layer; requested media encoded at 1 Mbps
- What fraction of the bottleneck is available to FTP?

Credit: MSc thesis work by Sean Boyden (2006)

Fairness of RealVideo Streams (2/2)



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 network support for multimedia

Network support for ~~multimedia~~

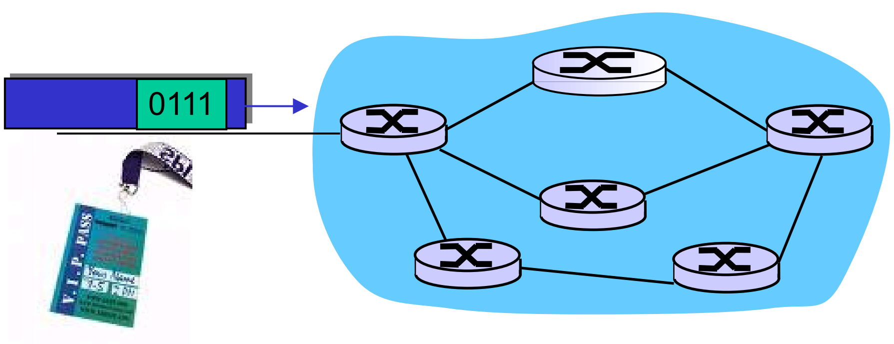
Approach	Granularity	Guarantee	Mechanisms	Complex	Deployed?
Making best of best effort service	All traffic treated equally	None or soft	No network support (all at application)	low	everywhere
Differentiated service	Traffic "class"	None of soft	Packet market, scheduling, policing.	med	some
Per-connection QoS	Per-connection flow	Soft or hard after flow admitted	Packet market, scheduling, policing, call admission	high	little to none

Dimensioning best effort networks

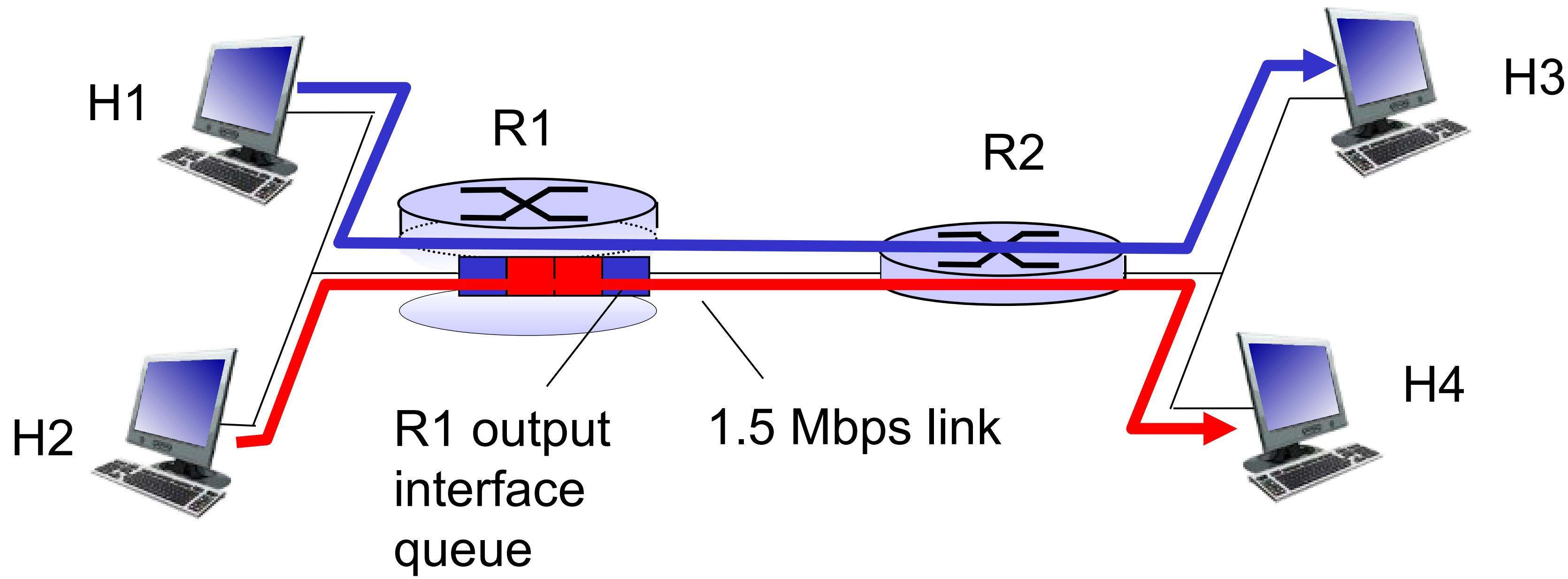
- *approach:* deploy enough link capacity so that congestion doesn't occur, multimedia traffic flows without delay or loss
  - low complexity of network mechanisms (use current "best effort" network)
  - high bandwidth costs
- challenges:
  - *network dimensioning:* how much bandwidth is "enough?"
  - *estimating network traffic demand:* needed to determine how much bandwidth is "enough" (for that much traffic)

Providing multiple classes of service

- thus far: making the best of best effort service
  - one-size fits all service model
- alternative: multiple classes of service
  - partition traffic into classes
  - network treats different classes of traffic differently (analogy: VIP service versus regular service)
- granularity: differential service among multiple classes, not among individual connections
- history: ToS bits

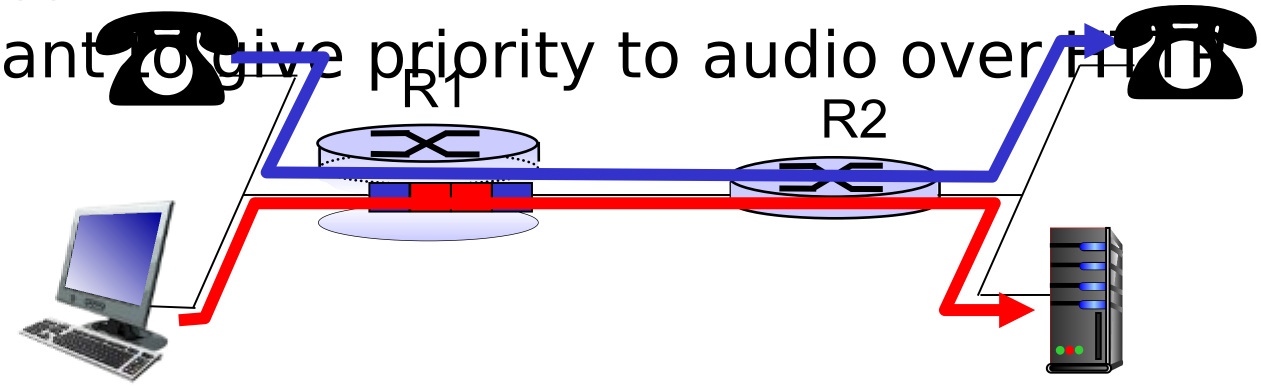


Multiple classes of service: scenario



Scenario 1: mixed HTTP and VoIP

- example: 1Mbps VoIP, HTTP share 1.5 Mbps link.
  - HTTP bursts can congest router, cause audio loss
  - want to give priority to audio over HTTP

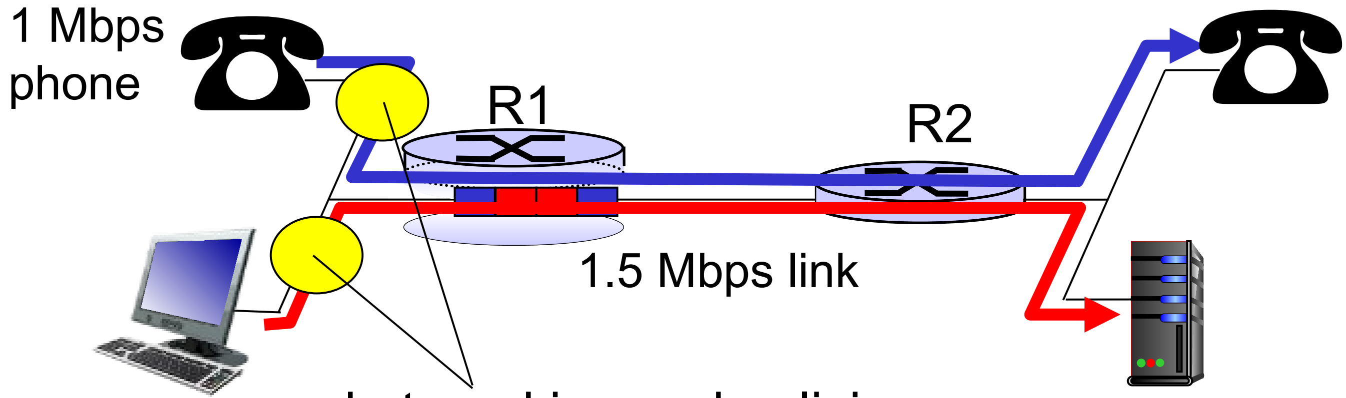


Principle 1

packet marking needed for router to distinguish between different classes; and new router policy to treat packets accordingly

Principles for QOS guarantees (more)

- what if applications misbehave (VoIP sends higher than declared rate)
  - policing: force source adherence to bandwidth allocations
- *marking*, *policing* at network edge

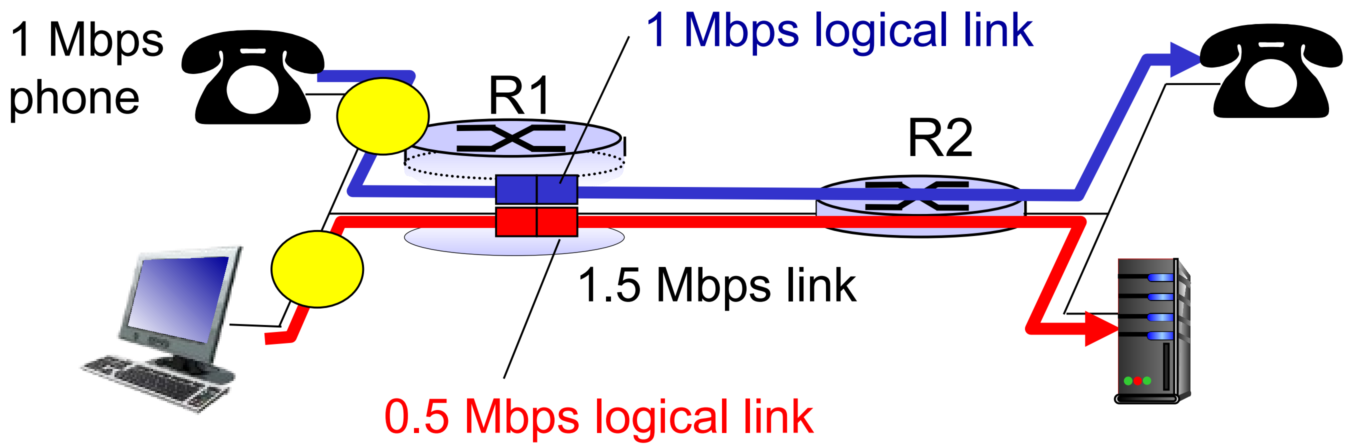


Principle 2

provide protection (isolation) for one class from others

Principles for QOS guarantees ~~(more)~~

- allocating *fixed* (non-sharable) bandwidth to flow: *inefficient* use of bandwidth if flows doesn't use its allocation



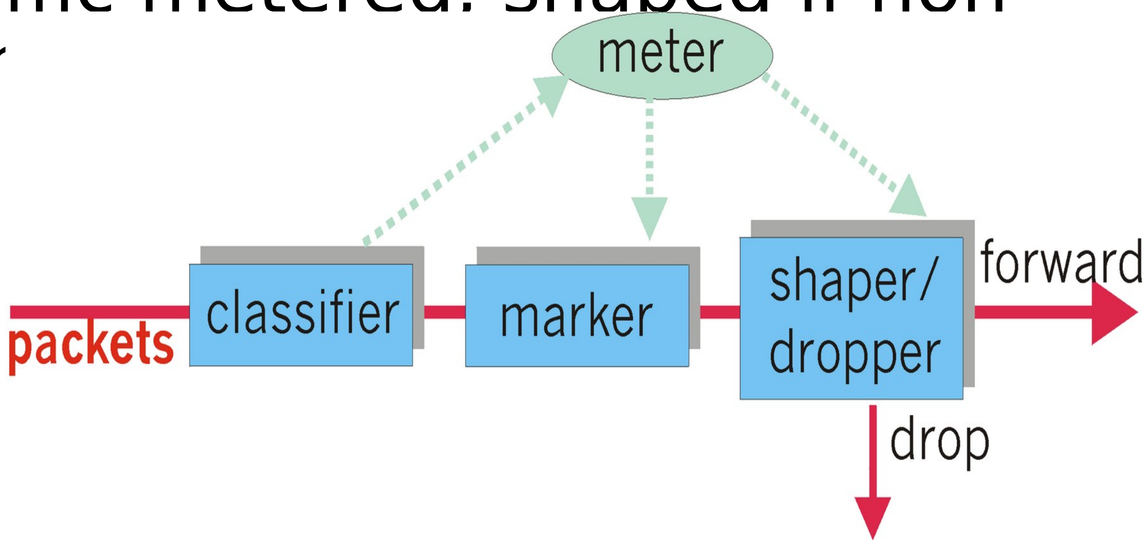
Principle 3

while providing isolation, it is desirable to use resources as efficiently as possible

Classification, conditioning

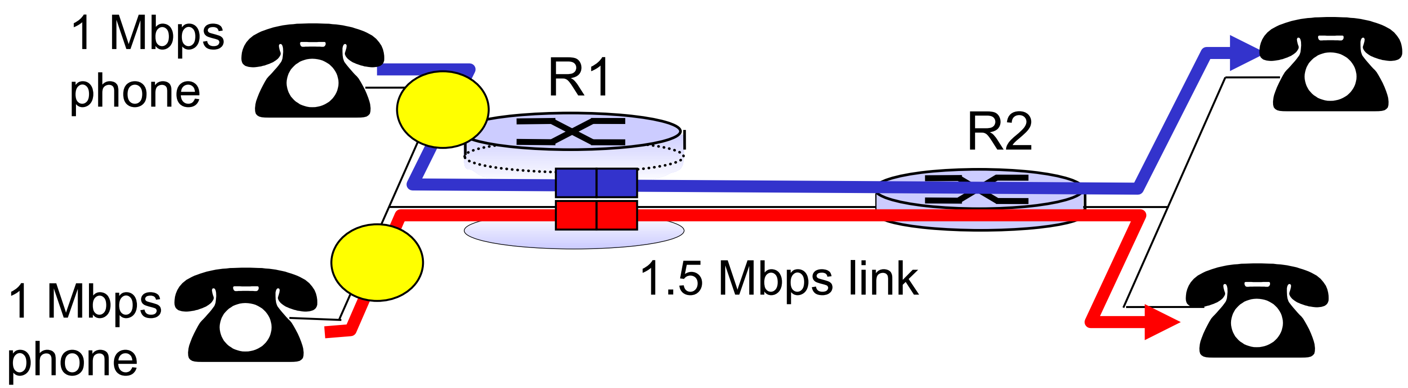
may be desirable to limit traffic injection rate of some class:

- user declares traffic profile (e.g., rate, burst size)
- traffic metered, shaped if non-
  cor



Per-connection QOS guarantees

- *basic fact of life:* can not support traffic demands beyond link capacity

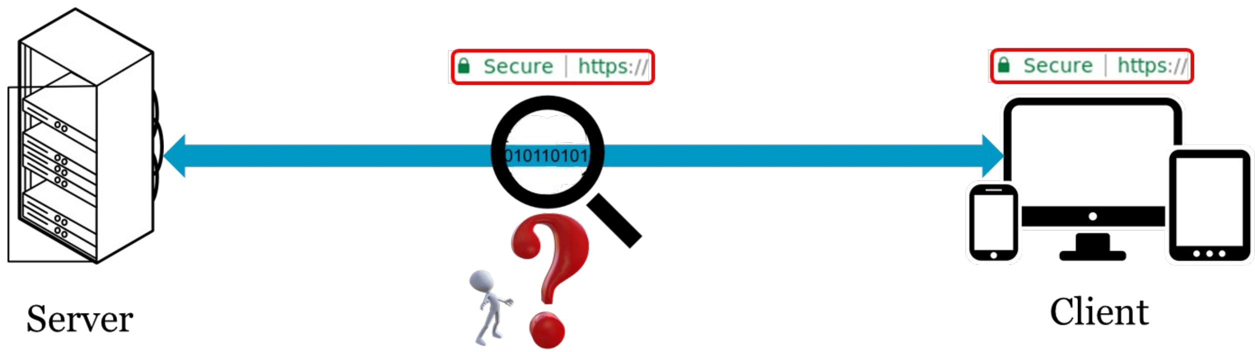


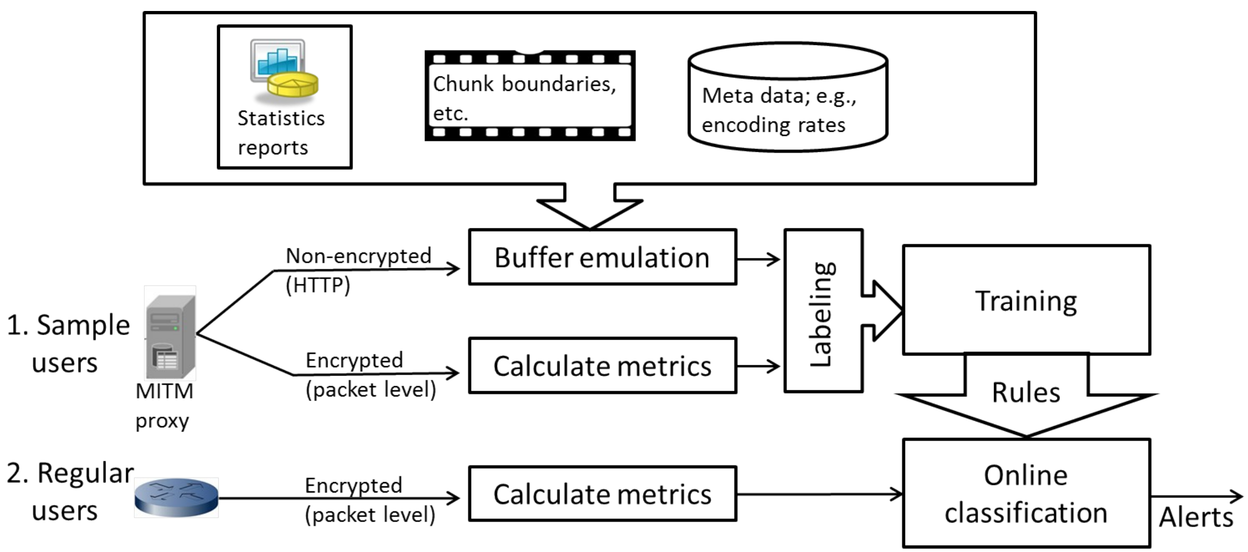
Principle 4

call admission: flow declares its needs, network may block call (e.g., busy signal) if it cannot meet needs

Scalable content delivery

… BUFFEST (per connection discussion)…





ACM MMSys 2017
(Krishnamoorthi et al.)

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 network support for multimedia