

# TDTS06: Computer Networks

Instructor: Niklas Carlsson

Email: [niklas.carlsson@liu.se](mailto:niklas.carlsson@liu.se)

Notes derived from "*Computer Networking: A Top Down Approach*", by Jim Kurose and Keith Ross, Addison-Wesley.

The slides are adapted and modified based on slides from the book's companion Web site, as well as modified slides by Anirban Mahanti and Carey Williamson.

# Scalable Content Delivery

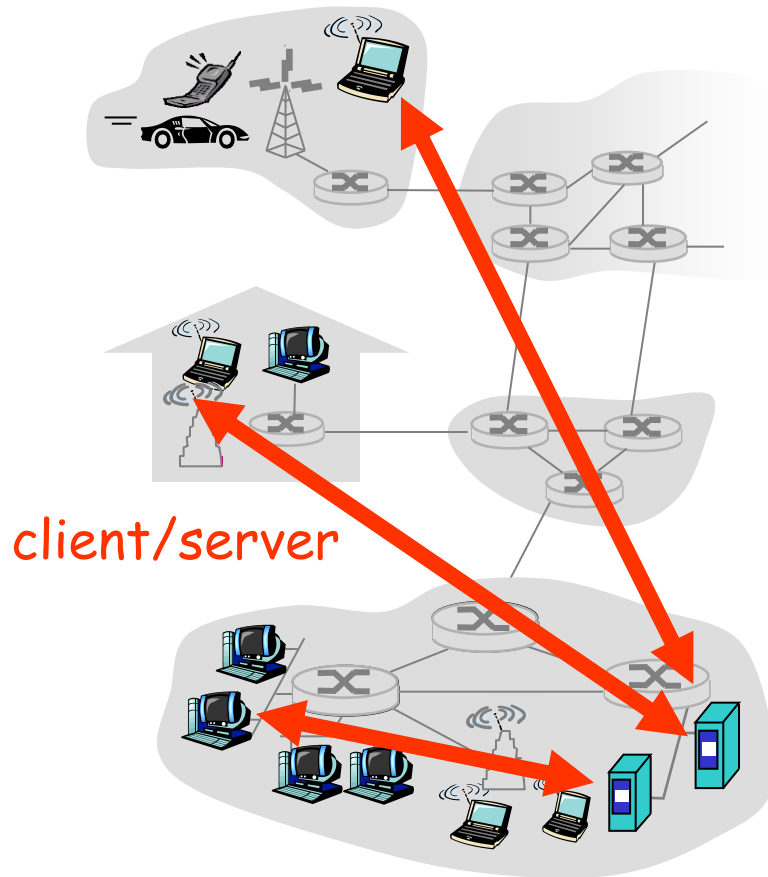
## Motivation

- **Use of Internet for content delivery is massive** ... and becoming more so (e.g., recent projection that by 2014, 90% of all IP traffic will be video content)
- **Variety of approaches:** HTTP-based Adaptive Streaming (HAS), broadcast/multicast, batching, replication/caching (e.g. CDNs), P2P, peer-assisted, ...
- In these slides, we only provide a few high-level examples

# Service models

# Client-server architecture

Client/server model has well-defined roles.



## server:

- always-on host
- permanent IP address
- server farms for scaling

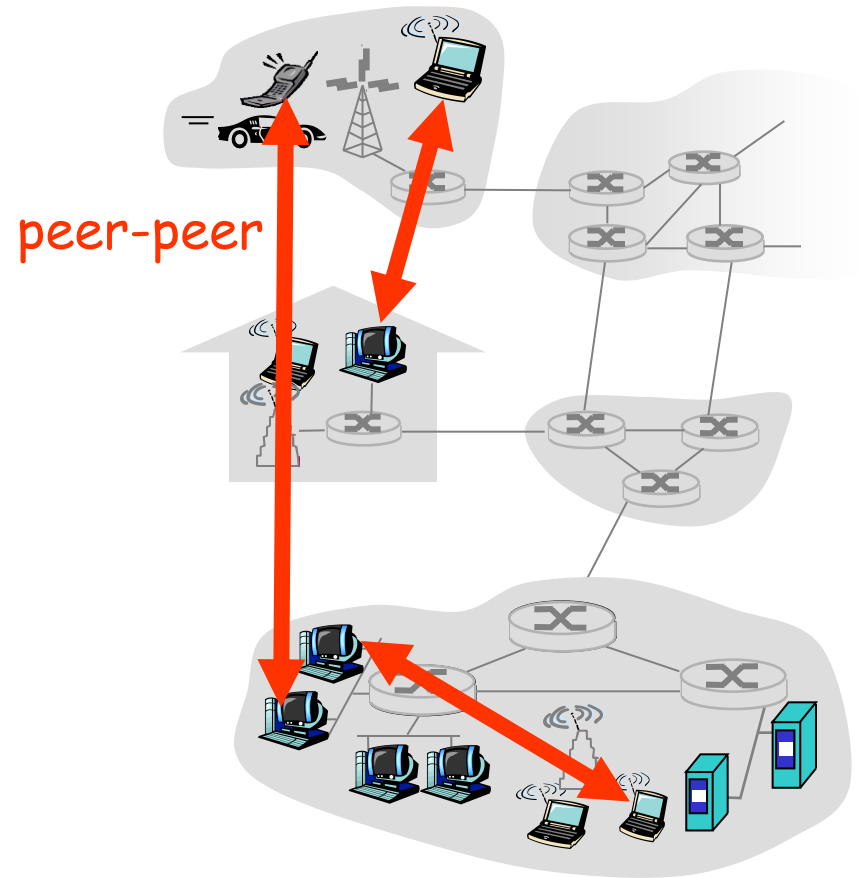
## clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

# Pure P2P architecture

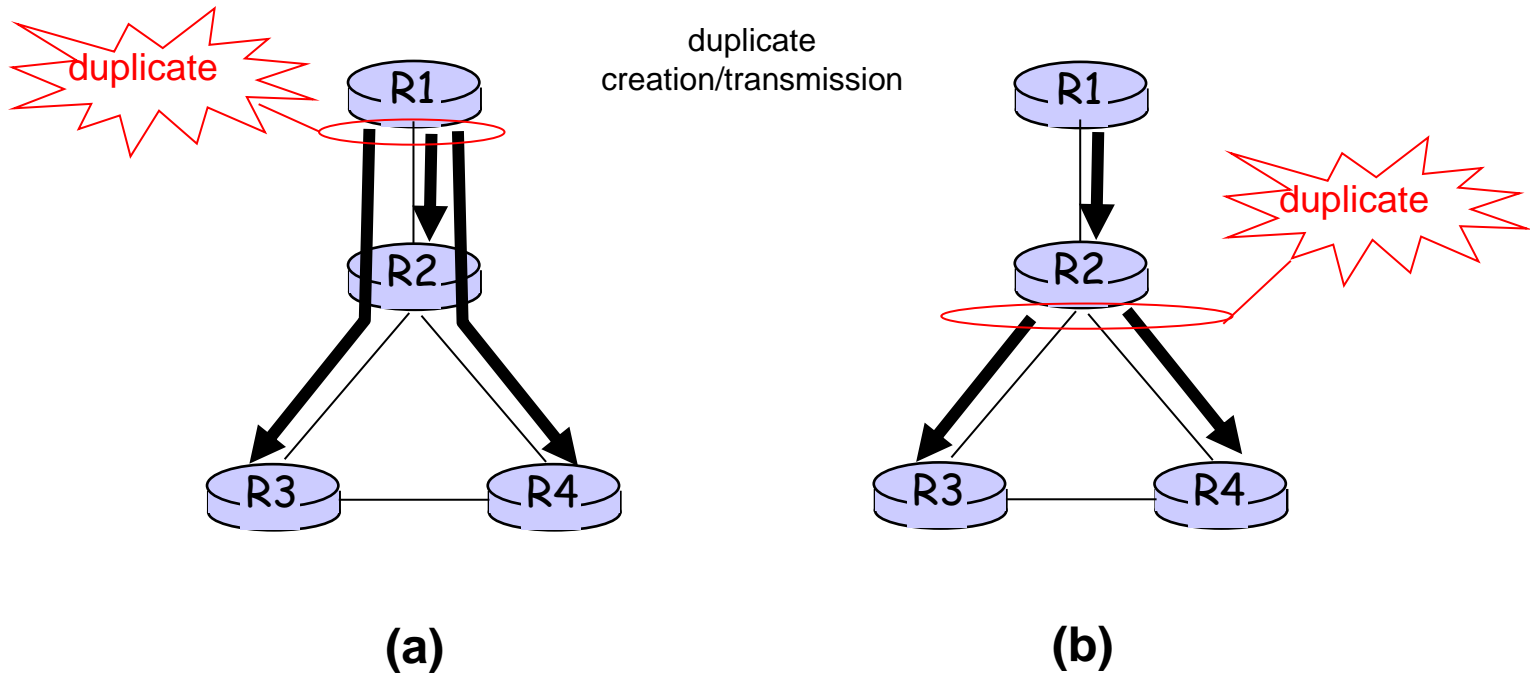
No fixed clients or servers: Each host can act as both client and server at any time

- ❑ *no* always-on server
- ❑ arbitrary end systems directly communicate
- ❑ peers are intermittently connected and change IP addresses



# Additional Multimedia Support

## Multicast/Broadcast



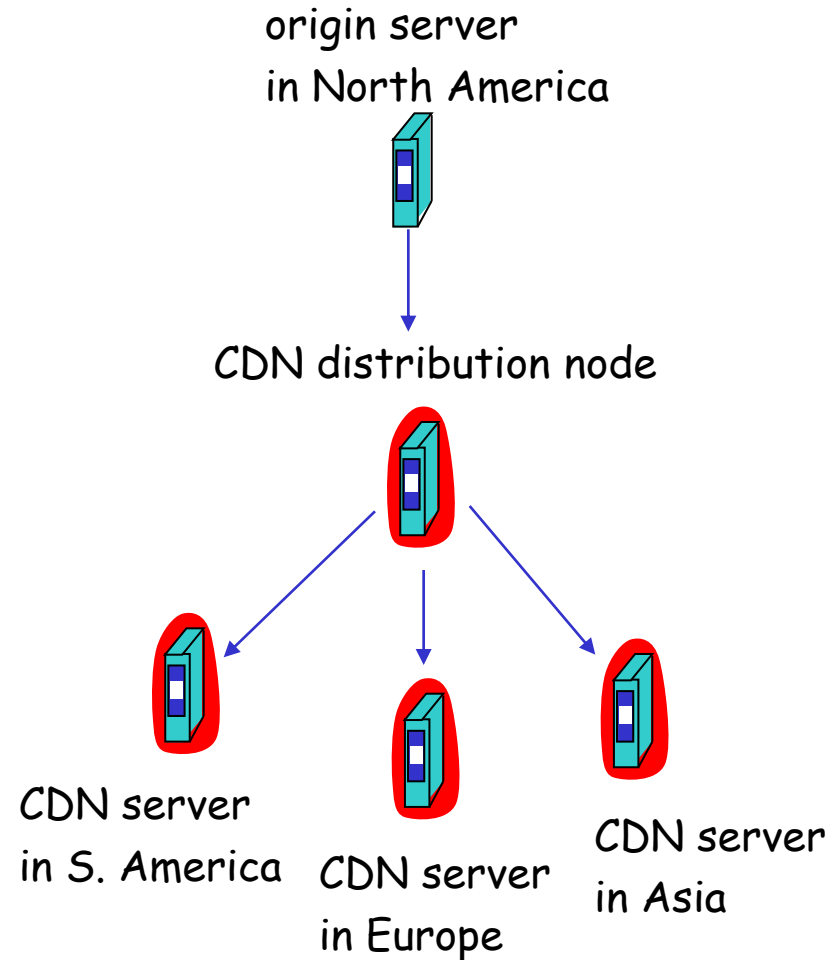
Source-duplication versus in-network duplication.  
(a) source duplication, (b) in-network duplication

Also, application-layer multicast ...

# Content distribution networks (CDNs)

## Content replication

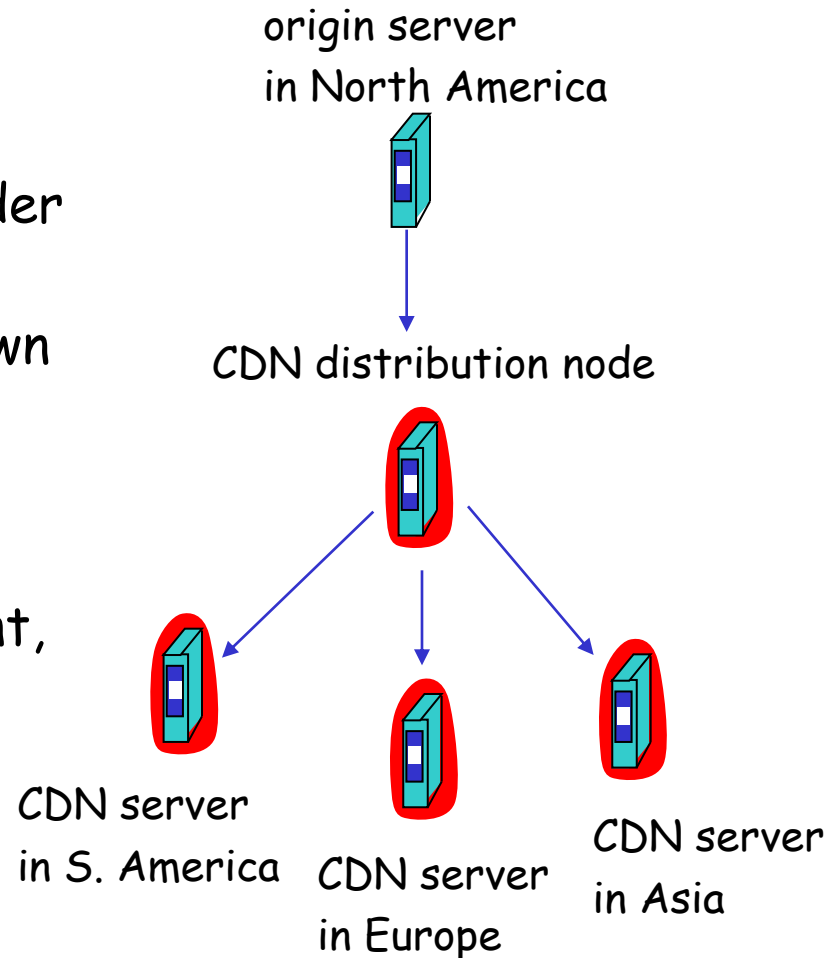
- ❑ replicate content at hundreds of servers throughout Internet
- ❑ content downloaded to CDN servers ahead of time
- ❑ content "close" to user reduce impairments (loss, delay) of sending content over long paths
- ❑ CDN server often in edge/access network



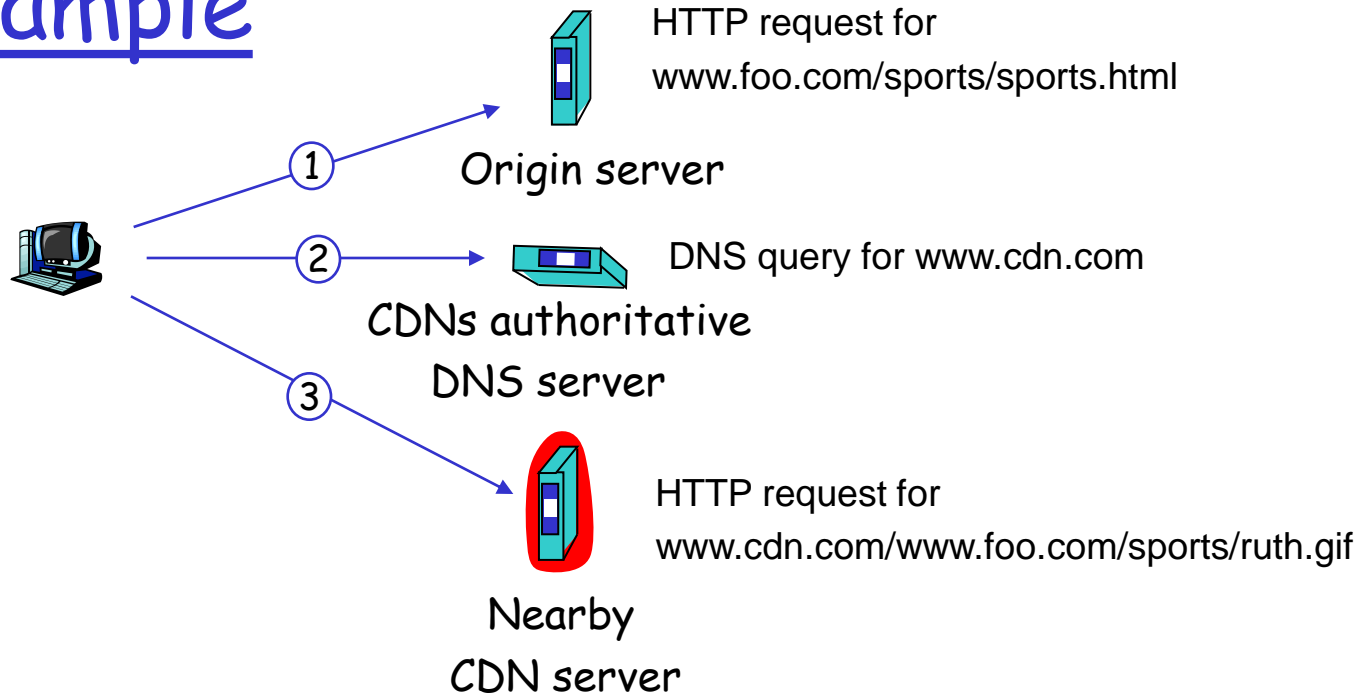
# Content distribution networks (CDNs)

## Content replication

- ❑ CDN (e.g., Akamai, Limewire) customer is the content provider (e.g., CNN)
- ❑ Other companies build their own CDN (e.g., Google)
- ❑ CDN replicates customers' content in CDN servers.
- ❑ When provider updates content, CDN updates servers



# CDN example



## origin server (www.foo.com)

- ❑ distributes HTML
- ❑ replaces:  
`http://www.foo.com/sports.ruth.gif`  
with  
`http://www.cdn.com/www.foo.com/sports/ruth.gif`

## CDN company (cdn.com)

- ❑ distributes gif files
- ❑ uses its authoritative DNS server to route redirect requests

# More about CDNs

## routing requests

- ❑ CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- ❑ when query arrives at authoritative DNS server:
  - server determines ISP from which query originates
  - uses “map” to determine best CDN server
- ❑ CDN nodes create application-layer overlay network



# Multimedia Networking

## Principles

- ❑ Classify multimedia applications
- ❑ Identify the network services the apps need
- ❑ Making the best of "best effort" service
- ❑ Mechanisms for providing QoS

## Protocols and Architectures

- ❑ Specific protocols for best effort delivery
- ❑ Architectures for QoS

# Why Study Multimedia Networking?

- ❑ Exciting and fun!
- ❑ Multimedia is everywhere
- ❑ Industry-relevant research topic
- ❑ Lots of open research problems



# Multimedia Networking Applications

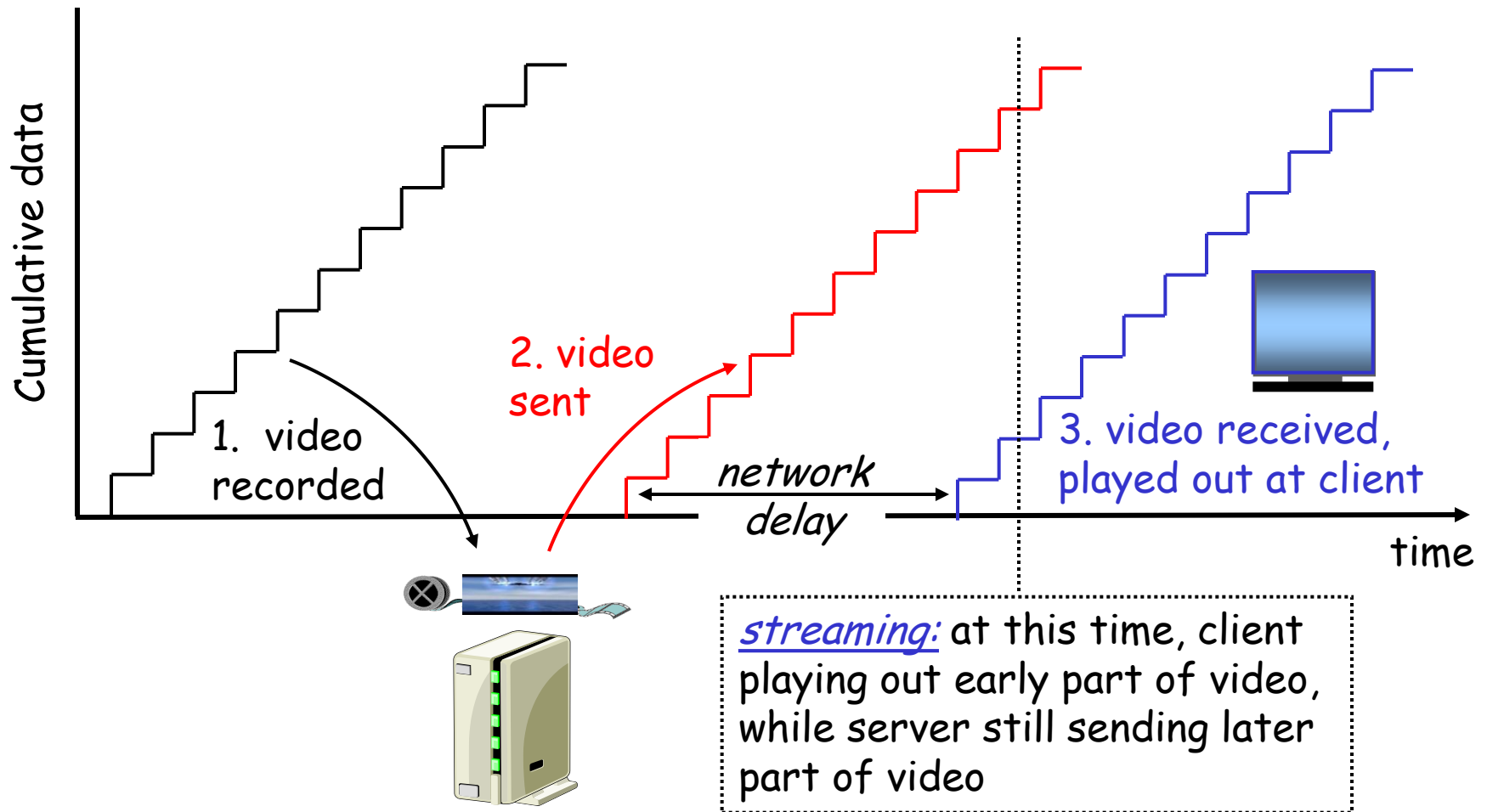
Classes of MM applications:

# Multimedia Networking Applications

## Classes of MM applications:

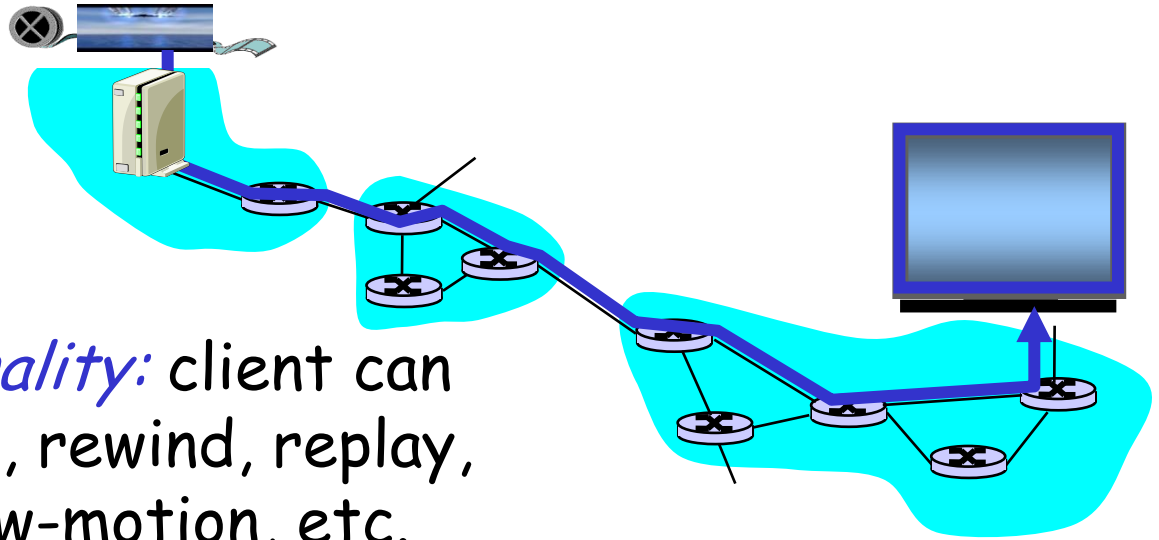
- 1) Streaming stored audio and video
- 2) Streaming live audio and video
- 3) Real-time interactive audio and video

# Streaming Stored Multimedia (1/2)

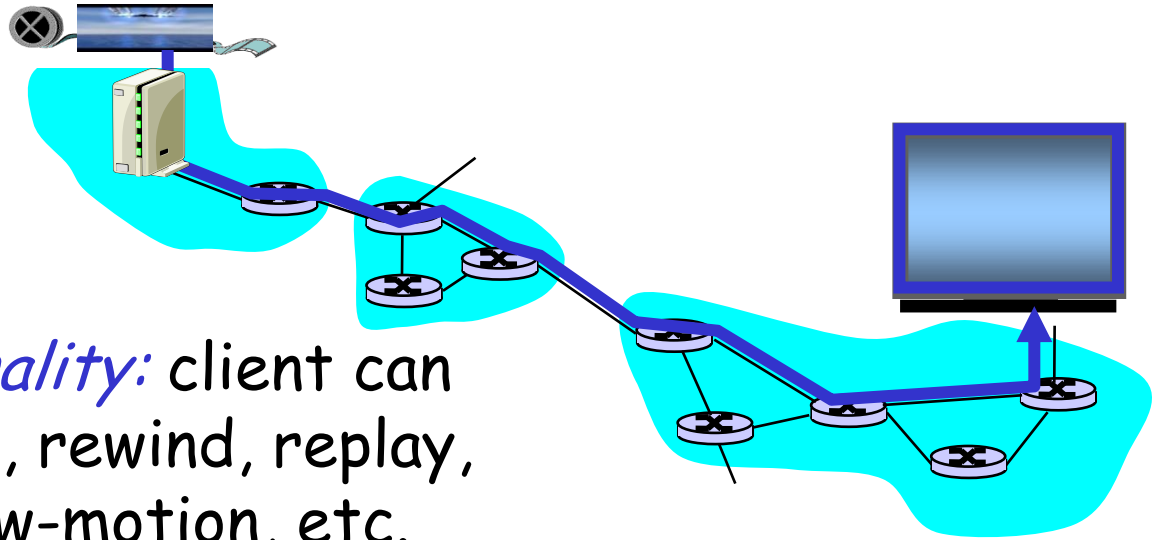


# Streaming Stored Multimedia (2/2)

- *VCR-like functionality*: client can start, stop, pause, rewind, replay, fast-forward, slow-motion, etc.



# Streaming Stored Multimedia (2/2)



- ❑ *VCR-like functionality*: client can start, stop, pause, rewind, replay, fast-forward, slow-motion, etc.
  - 10 sec initial delay OK
  - 1-2 sec until command effect OK
  - need a separate control protocol?
- ❑ timing constraint for data that is yet to be transmitted: must arrive in time for playback

# Streaming Live Multimedia

## Examples:

- ❑ Internet radio talk show
- ❑ Live sporting event

# Streaming Live Multimedia

## Examples:

- ❑ Internet radio talk show
- ❑ Live sporting event

## Streaming

- ❑ playback buffer
- ❑ playback can lag tens of seconds after transmission
- ❑ still have timing constraint

## Interactivity

- ❑ fast-forward is not possible
- ❑ rewind and pause possible!

# Interactive, Real-time Multimedia

- **applications:** IP telephony, video conference, distributed interactive worlds

# Interactive, Real-time Multimedia

- ❑ **applications:** IP telephony, video conference, distributed interactive worlds
- ❑ **end-end delay requirements:**
  - audio: < 150 msec good, < 400 msec OK
    - includes application-layer (packetization) and network delays
    - higher delays noticeable, impair interactivity
- ❑ **session initialization**
  - callee must advertise its IP address, port number, frame rate, encoding algorithms



# Multimedia Networking Applications

Fundamental characteristics:

# Multimedia Networking Applications

## Fundamental characteristics:

- Inherent frame rate

# Multimedia Networking Applications

## Fundamental characteristics:

- ❑ Inherent frame rate
- ❑ Typically **delay-sensitive**
  - end-to-end delay
  - delay jitter

# Multimedia Networking Applications

## Fundamental characteristics:

- ❑ Inherent frame rate
- ❑ Typically **delay-sensitive**
  - end-to-end delay
  - delay jitter
- ❑ But **loss-tolerant**: infrequent losses cause minor transient glitches
- ❑ Unlike data apps, which are often delay-tolerant but loss-sensitive.

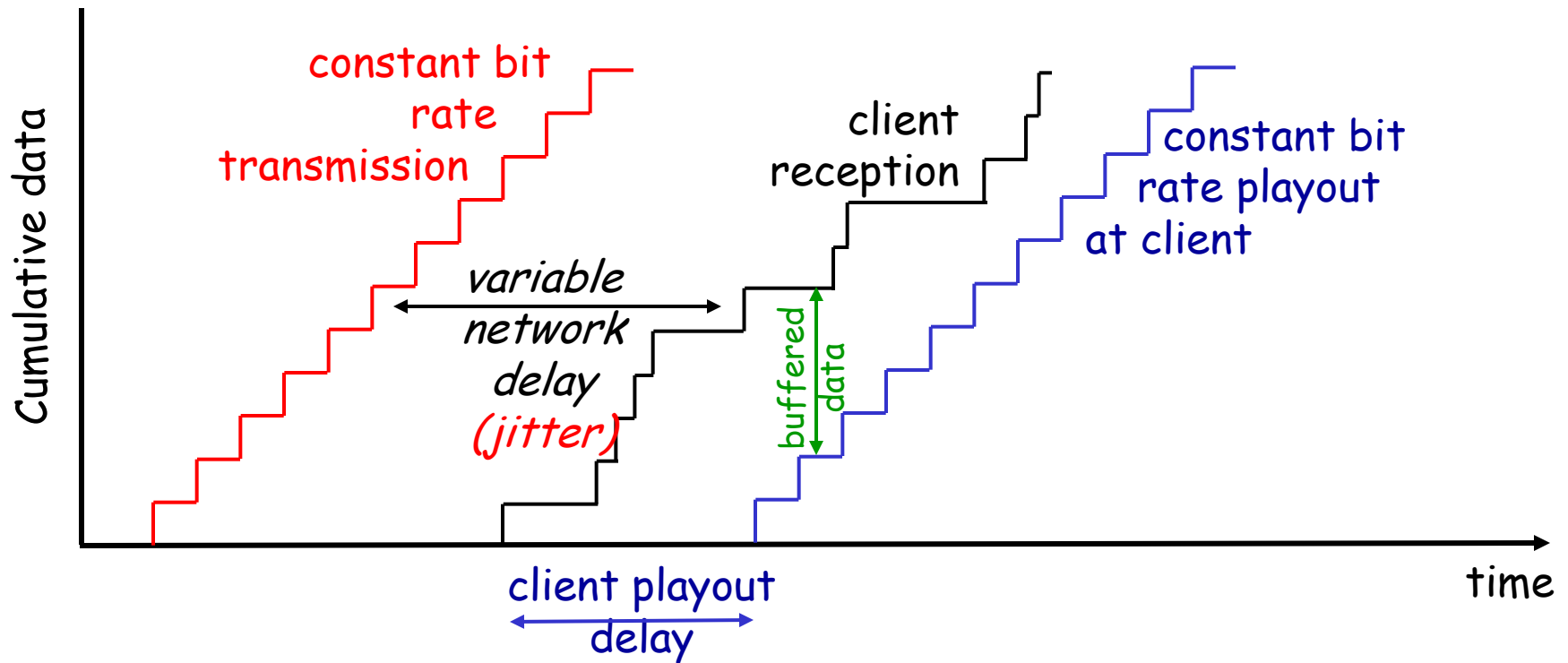
# Multimedia Networking Applications

## Fundamental characteristics:

- ❑ Inherent frame rate
- ❑ Typically **delay-sensitive**
  - end-to-end delay
  - delay jitter
- ❑ But **loss-tolerant**: infrequent losses cause minor transient glitches
- ❑ Unlike data apps, which are often delay-tolerant but loss-sensitive.

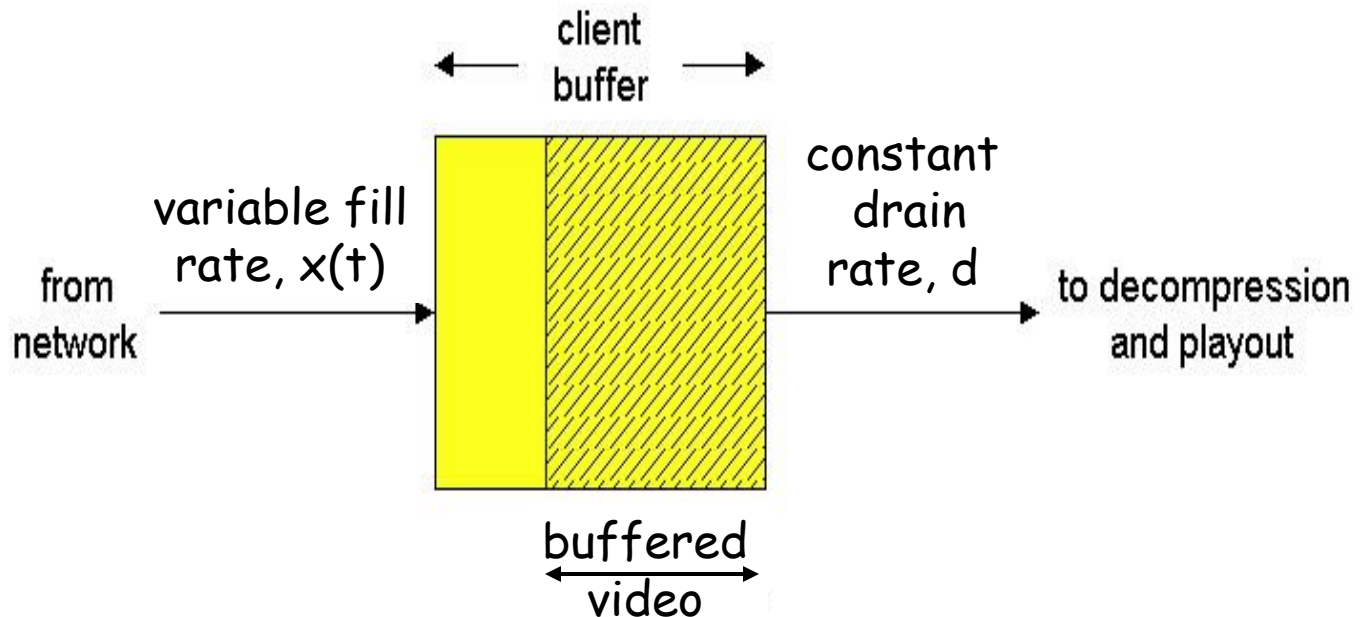
**Jitter** is the variability of packet delays within the same packet stream

# Delay Jitter



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

# Streaming Multimedia: Client Buffering



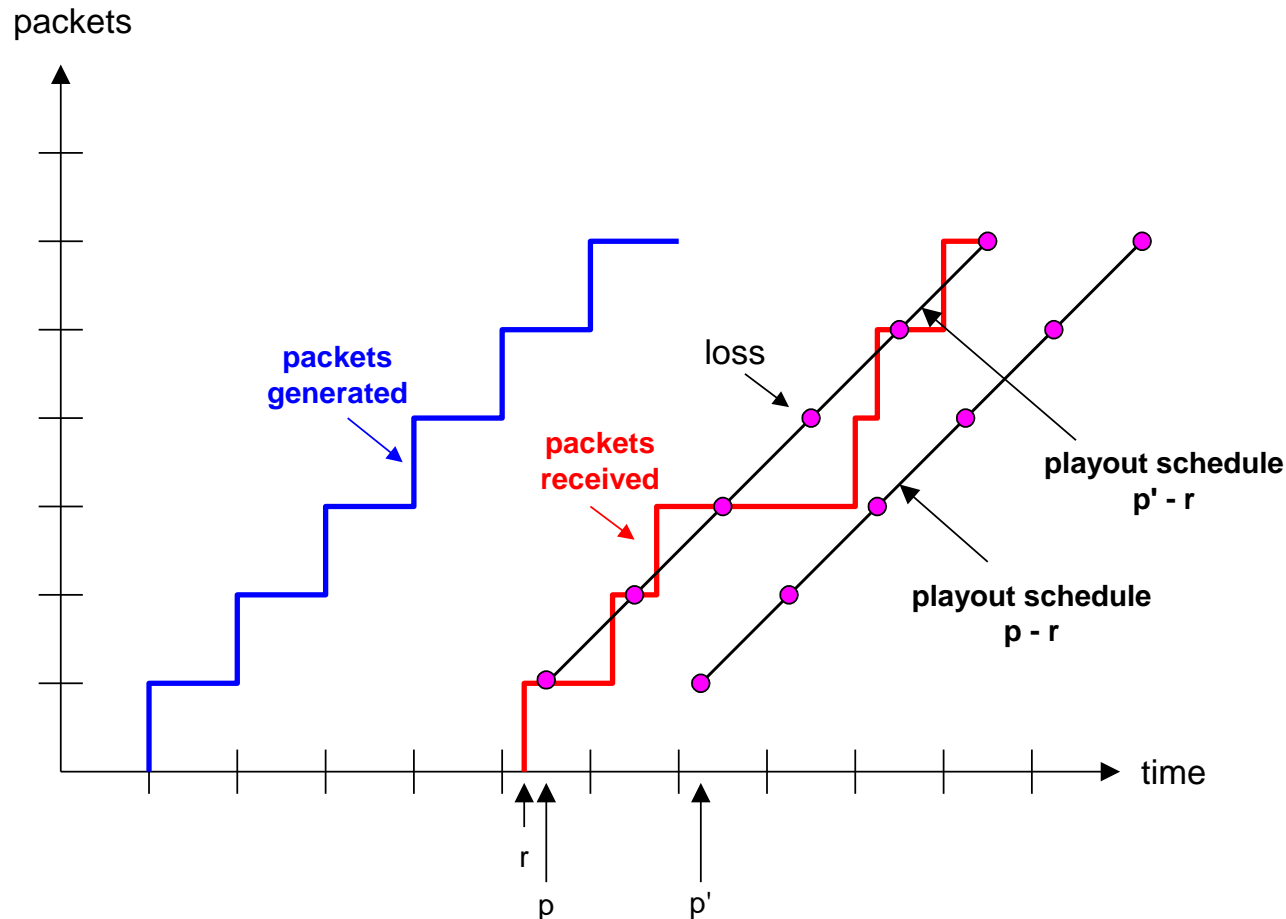
- Client-side buffering, playout delay compensate for network-added delay, delay jitter

# Example: Fixed Playout Delay

- ❑ receiver attempts to playout each chunk exactly  $q$  msec 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

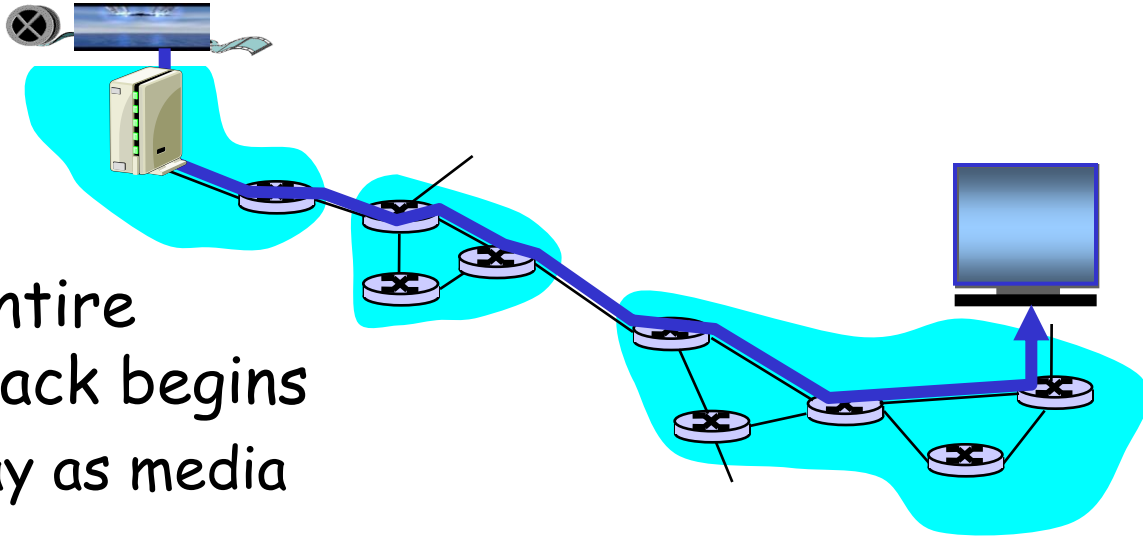
# Example: Fixed Playout Delay

- sender generates packets every X msec
- first packet received at time  $r$
- first playout schedule: begins at  $p$
- second playout schedule: begins at  $p'$



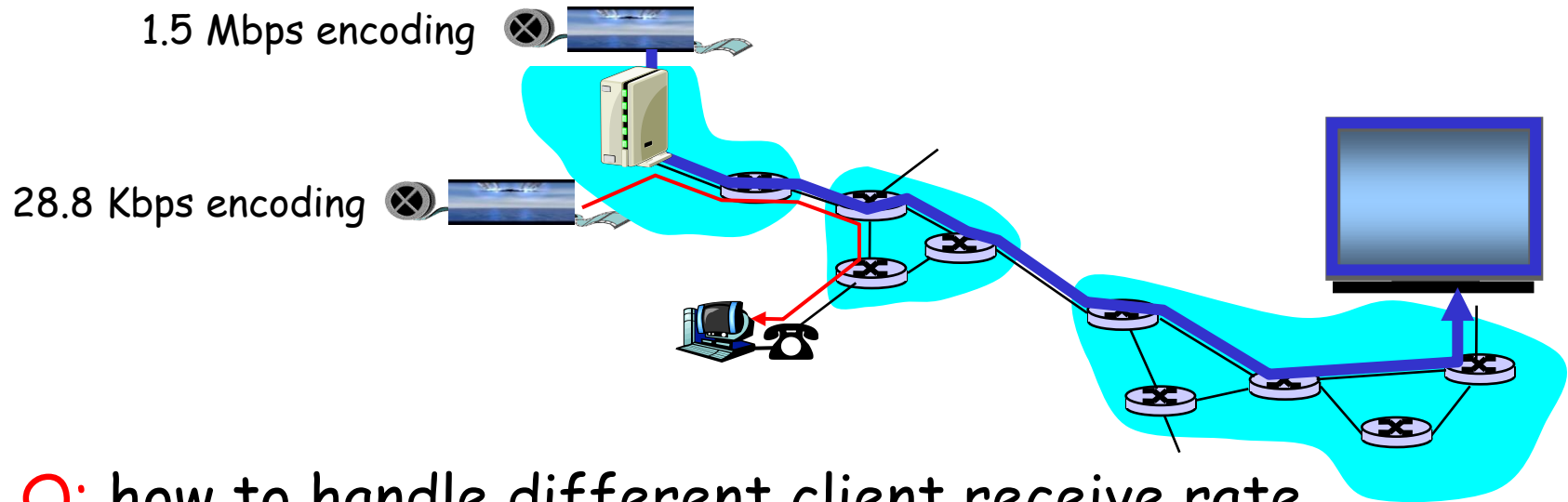


# Streaming vs. Download (Stored MM Content)



- ❑ **Download:** Receive entire content before playback begins
  - High "start-up" delay as media file can be large
  - ~ 4GB for a 2 hour MPEG II movie
- ❑ **Streaming:** Play the media file while it is still being received
  - Reasonable "start-up" delays
  - Assumes reception rate exceeds playback rate. (Why?)

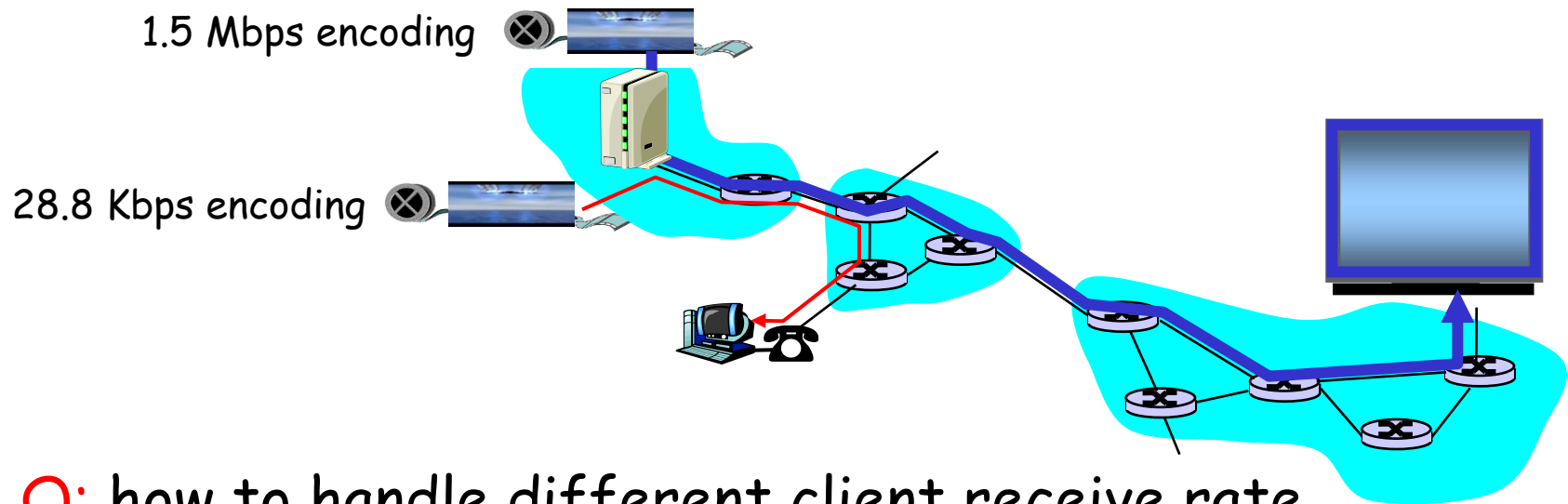
# Streaming Multimedia: client rate(s)



Q: how to handle different client receive rate capabilities?

- 28.8 Kbps dialup
- 100 Mbps Ethernet

# Streaming Multimedia: client rate(s)



Q: how to handle different client receive rate capabilities?

- 28.8 Kbps dialup
- 100 Mbps Ethernet

A1: server stores, transmits multiple copies of video, encoded at different rates

A2: layered and/or dynamically rate-based encoding



# Consider first ...

## Streaming Stored Multimedia

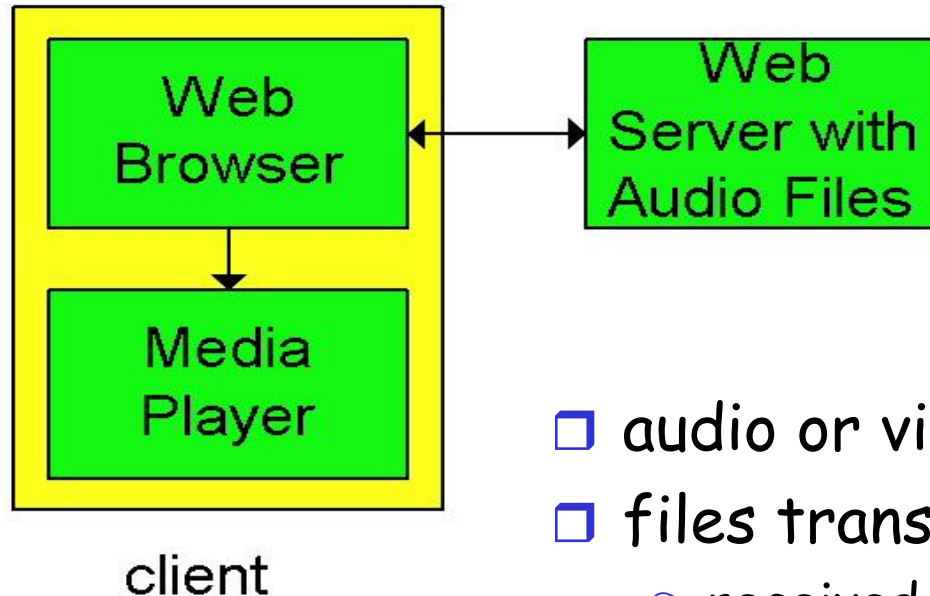
application-level  
streaming techniques for  
making the best out of  
best effort service:

- client-side buffering
- use of UDP versus TCP
- multiple encodings of multimedia

### Media Player

- jitter removal
- decompression
- error concealment
- graphical user interface  
w/ controls for  
interactivity

# Internet multimedia: simplest approach

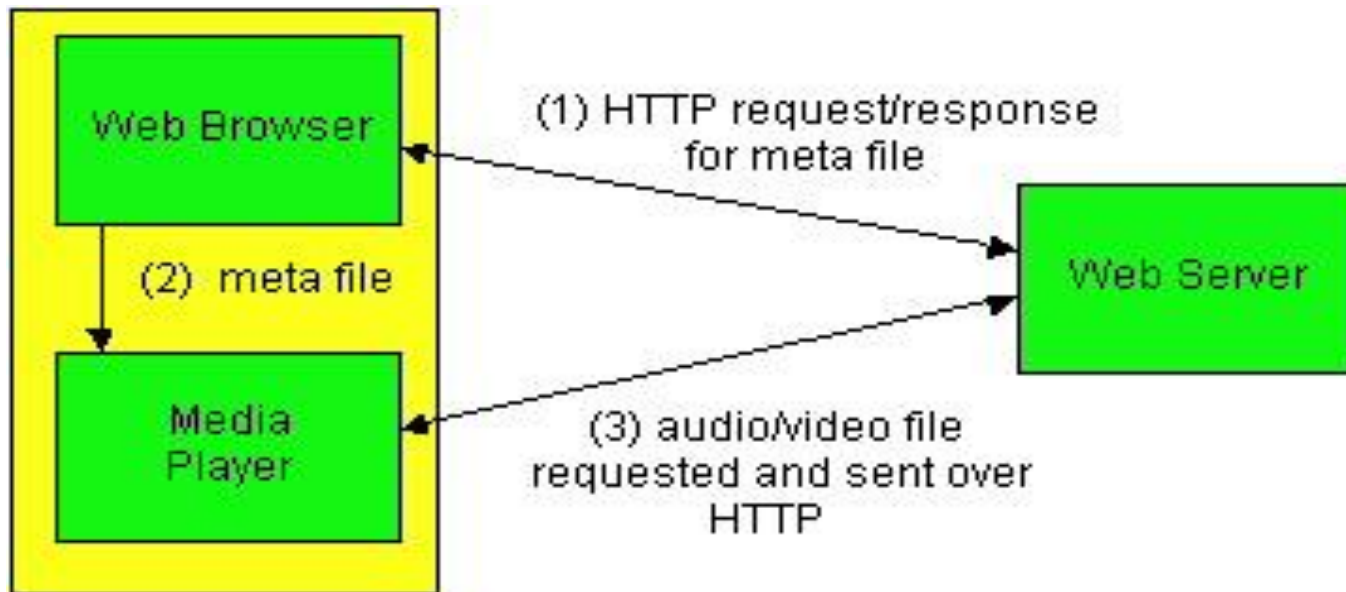


- ❑ audio or video stored in file
- ❑ files transferred as HTTP object
  - received in entirety at client
  - then passed to player

audio, video is downloaded, not streamed:

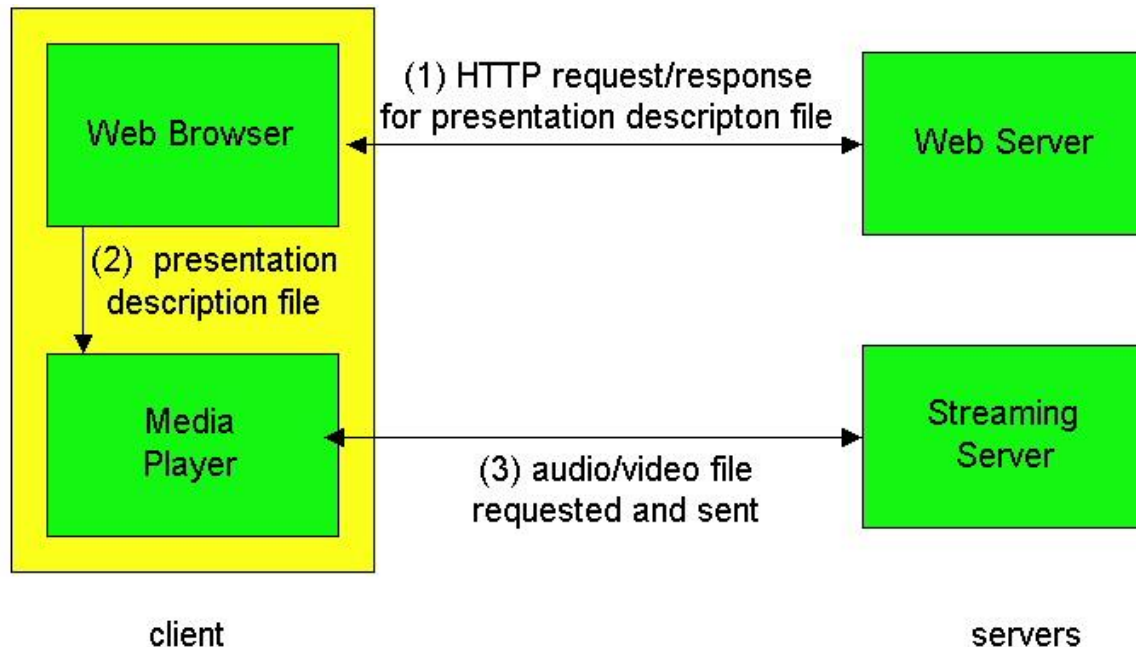
- ❑ long delays until playout, since no pipelining!

# Progressive Download



- ❑ browser retrieves **metafile** using HTTP GET
- ❑ browser launches player, passing metafile to it
- ❑ media player contacts server directly
- ❑ server **downloads** audio/video to player

# Streaming from a Streaming Server



- ❑ This architecture allows for non-HTTP protocol between server and media player
- ❑ Can also use UDP instead of TCP.



# Streaming Multimedia: UDP or TCP?

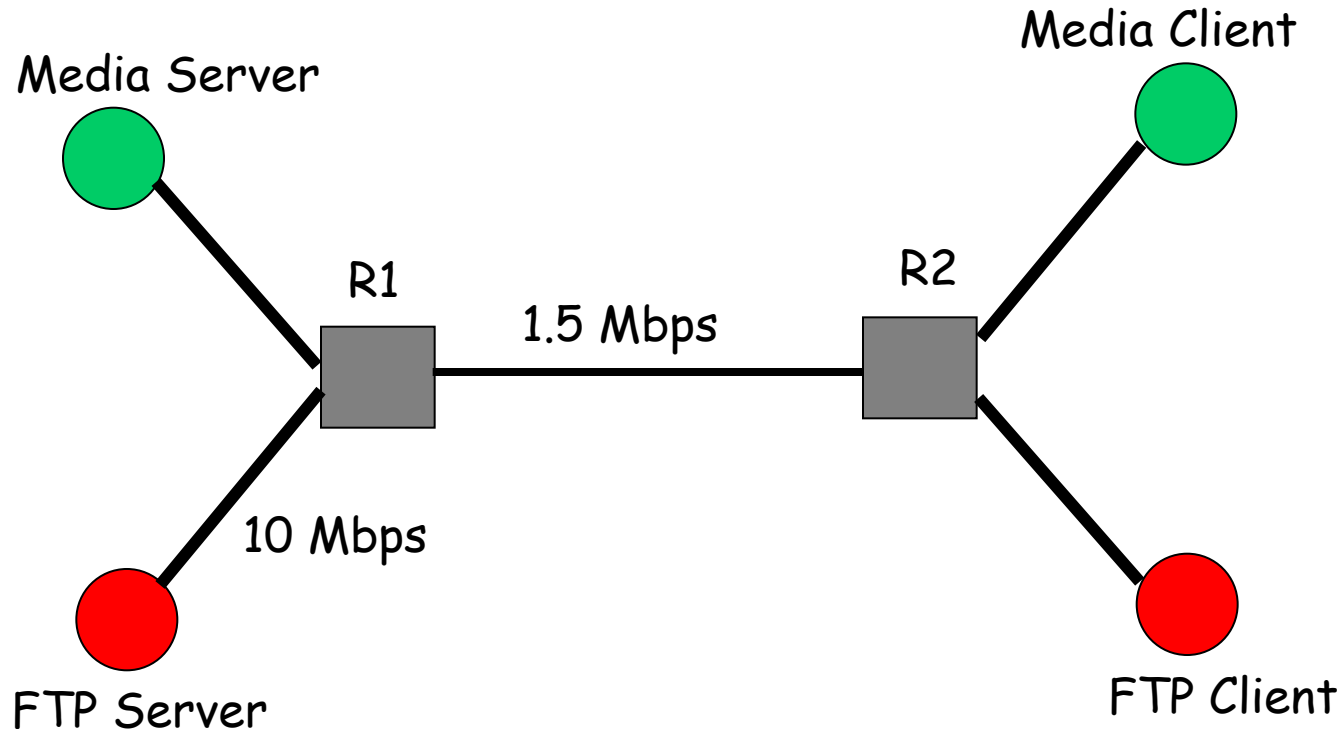
## UDP

- ❑ server sends at rate appropriate for client (oblivious to network congestion !)
  - often send rate = encoding rate = constant rate
  - then, fill rate = constant rate - packet loss
- ❑ short playout delay (2-5 seconds) to compensate for network delay jitter
- ❑ error recover: time permitting

## TCP

- ❑ send at maximum possible rate under TCP
- ❑ fill rate fluctuates due to TCP congestion control
- ❑ larger playout delay: smooth TCP delivery rate
- ❑ HTTP/TCP passes more easily through firewalls

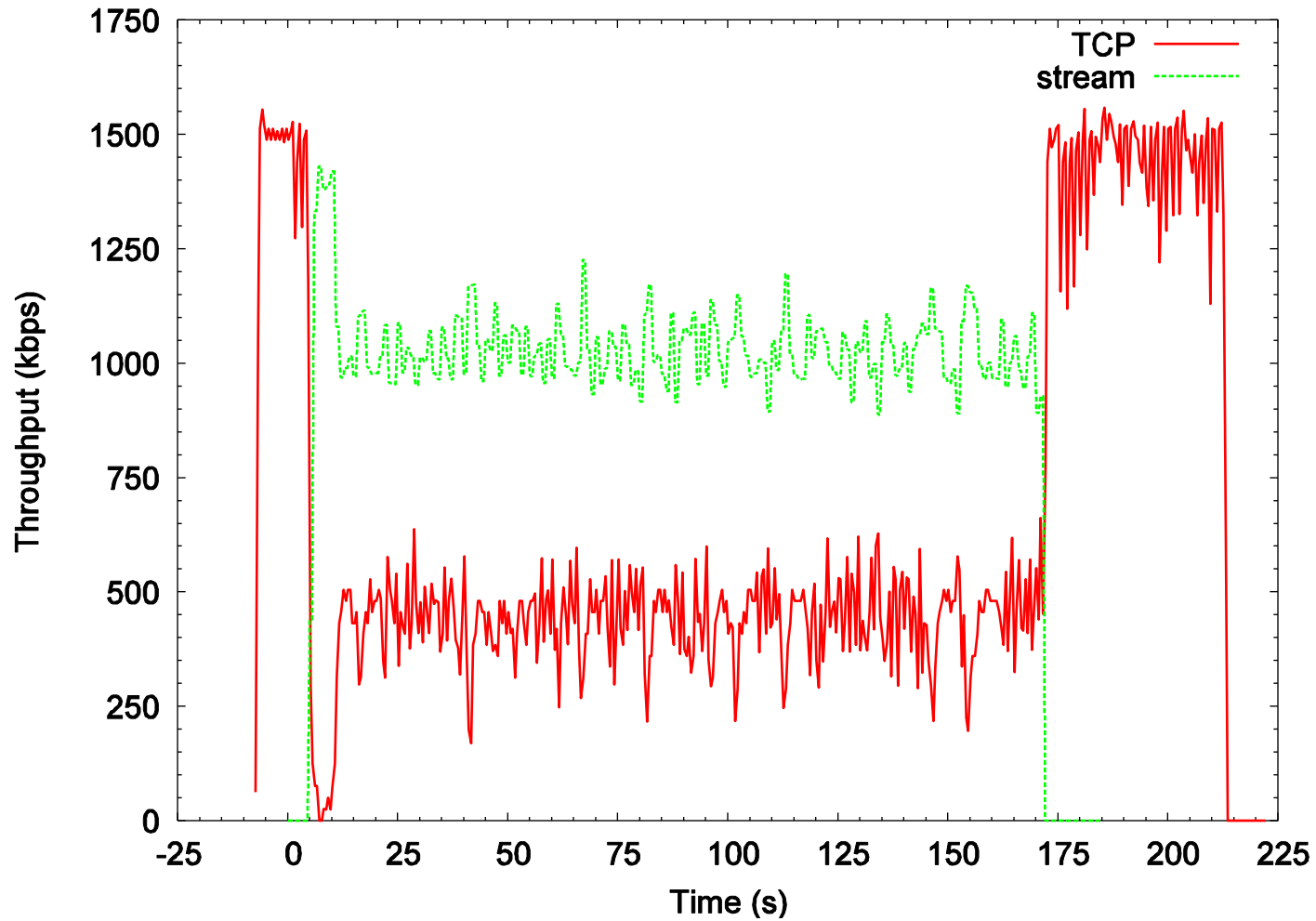
# Fairness of UDP Streams (1/2)



- 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)





# A protocol family for streaming

- ❑ RTSP
- ❑ RTP
- ❑ RTCP

# User Control of Streaming Media: RTSP

## RTSP: RFC 2326

- ❑ client-server application layer protocol
- ❑ user control: rewind, fast forward, pause, resume, repositioning, etc.

## What it doesn't do:

- ❑ doesn't define how audio/video is encapsulated for streaming over network
- ❑ doesn't restrict how streamed media is transported (UDP or TCP possible)
- ❑ doesn't specify how media player buffers audio/video

# RTSP: out-of-band control

FTP uses an "out-of-band" control channel:

- ❑ file transferred over one TCP connection.
- ❑ control info (directory changes, file deletion, rename) sent over separate TCP connection
- ❑ "out-of-band", "in-band" channels use different port numbers

RTSP messages (also) sent out-of-band:

- ❑ RTSP control messages use different port numbers than media stream: out-of-band.
  - port 554
- ❑ media stream is considered "in-band".

# RTSP Example

## Scenario:

- ❑ metafile communicated to web browser
- ❑ browser launches player
- ❑ player sets up an RTSP control connection, data connection to streaming server

# Metafile Example

<title>Twister</title>

<session>

  <group language=en lipsync>

    <switch>

      <track type=audio

        e="PCMU/8000/1"

        src = "rtsp://audio.example.com/twister/audio.en/lofi">

      <track type=audio

        e="DVI4/16000/2" pt="90 DVI4/8000/1"

        src="rtsp://audio.example.com/twister/audio.en/hifi">

    </switch>

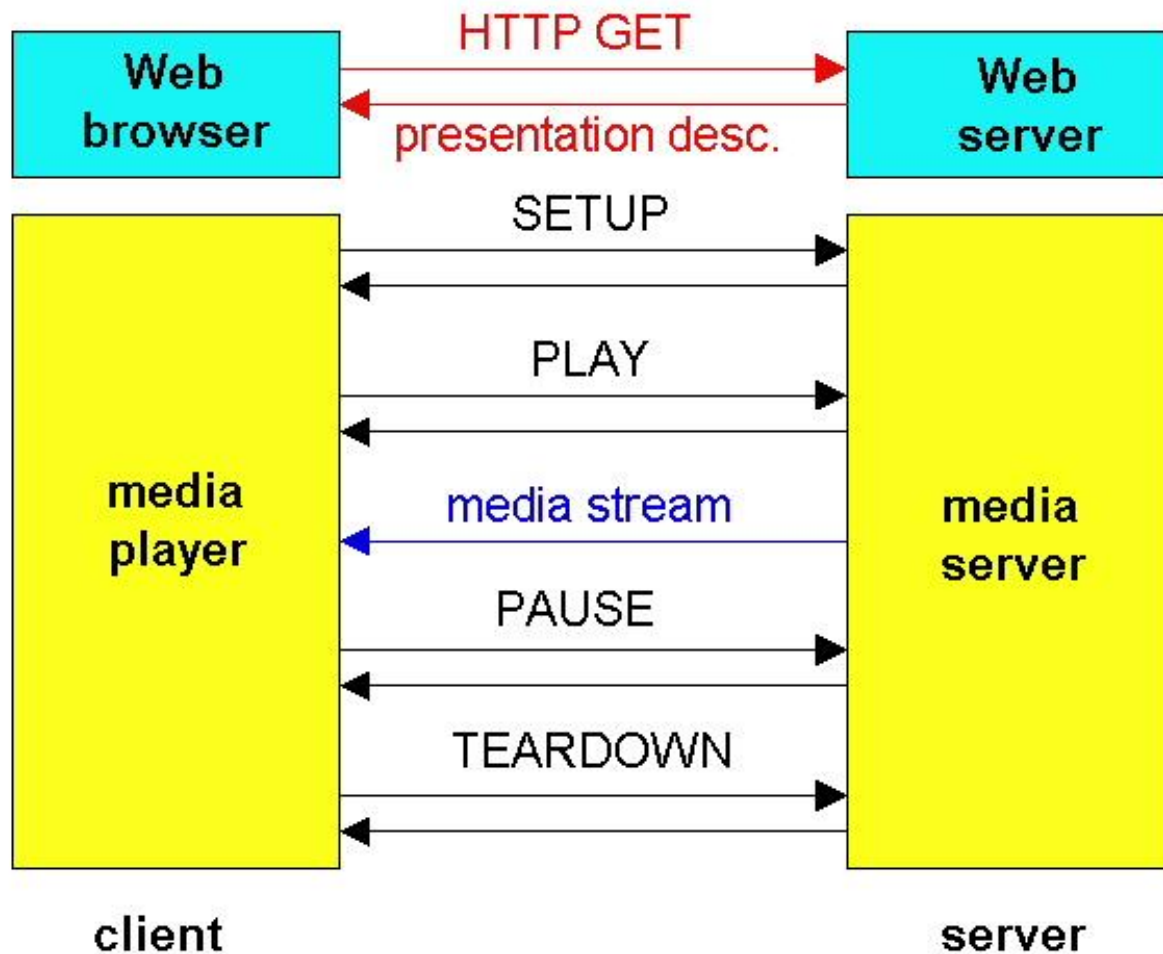
  <track type="video/jpeg"

    src="rtsp://video.example.com/twister/video">

  </group>

</session>

# RTSP Operation



# RTSP Exchange Example

C: SETUP rtsp://audio.example.com/twister/audio RTSP/1.0  
Transport: rtp/udp; compression; port=3056; mode=PLAY

S: RTSP/1.0 200 1 OK  
Session 4231

C: PLAY rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0  
Session: 4231  
Range: npt=0-

C: PAUSE rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0  
Session: 4231  
Range: npt=37

C: TEARDOWN rtsp://audio.example.com/twister/audio.en/lofi RTSP/1.0  
Session: 4231

S: 200 3 OK

# Real-Time Protocol (RTP)

- ❑ RTP specifies packet structure for packets carrying audio, video data
- ❑ RFC 3550
- ❑ RTP runs in end systems
- ❑ RTP packets encapsulated in UDP segments

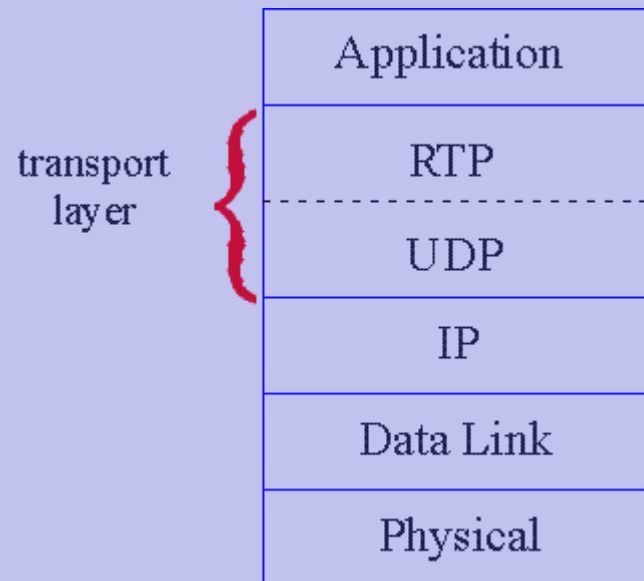
# Real-Time Protocol (RTP)

- ❑ 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 Internet phone applications run RTP, then they may be able to work together

# 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

- ❑ consider 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 is only seen at end systems (not) by intermediate routers.
  - routers providing best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely matter.

# RTP Header



RTP Header

*Payload Type (7 bits)*: Indicates type of encoding currently being used. If sender changes encoding in middle of conference, 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 Number (16 bits)*: Increments by one for each RTP packet sent, and may be used to detect packet loss and to restore packet sequence.

## RTP Header (2)

- ❑ *Timestamp field (32 bytes long):* sampling instant of first byte in this RTP data packet
  - for audio, timestamp clock typically increments by one for each sampling period (for example, each 125 usecs for 8 KHz sampling clock)
  - if application generates chunks of 160 encoded samples, then 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 should have distinct SSRC.

# Real-Time Control Protocol (RTCP)

- ❑ works in conjunction with RTP.
- ❑ each participant in RTP session periodically transmits 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, etc.
- ❑ feedback can be used to control performance
  - sender may modify its transmissions based on feedback

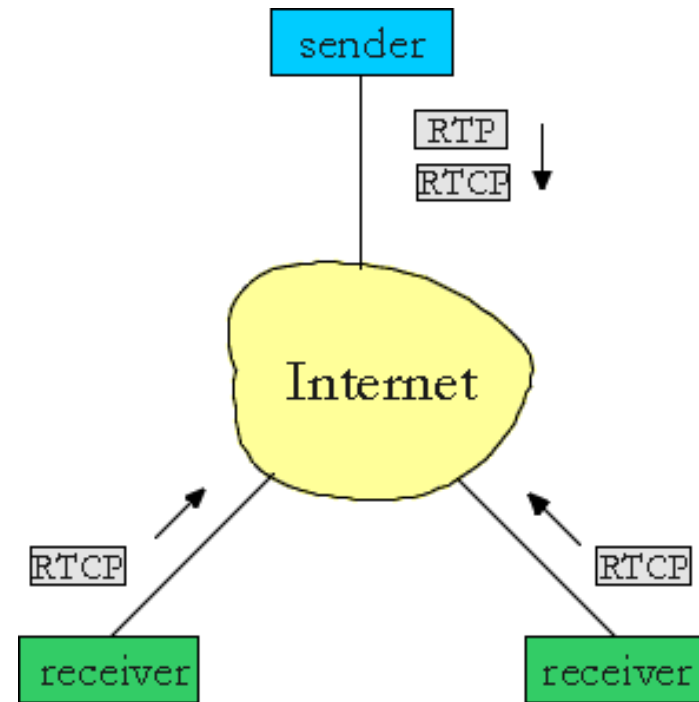
# Real-time Control Protocol (RTCP)

## Receiver report packets:

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

## Sender report packets:

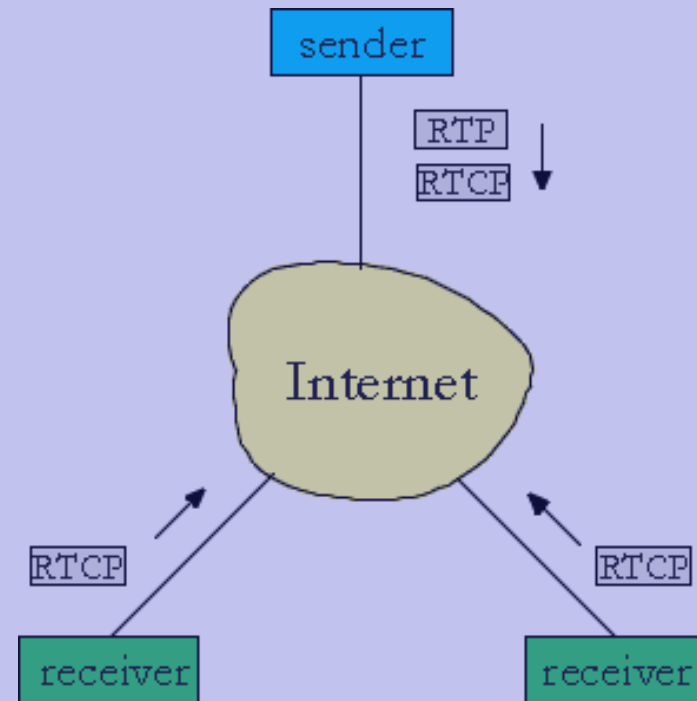
- ❑ SSRC of RTP stream, current time, number of packets sent, number of bytes sent



- ❑ RTCP attempts to limit its traffic to 5% of session bandwidth

# Real-time Control Protocol (RTCP)

- RTCP attempts to limit its traffic to 5% of session bandwidth



- 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

# RTCP Packets

## 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

## Source description packets:

- ❑ e-mail address of sender, sender's name, SSRC of associated RTP stream
- ❑ provide mapping between the SSRC and the user/host name

# Synchronization of Streams

- ❑ RTCP can synchronize different media streams within a RTP session
- ❑ consider videoconferencing app for which each sender generates one RTP stream for video, one for audio.
- ❑ timestamps in RTP packets tied to the video, audio sampling clocks
  - *not* tied to wall-clock time
- ❑ each RTCP sender-report packet contains (for most recently generated packet in associated RTP stream):
  - timestamp of RTP packet
  - wall-clock time for when packet was created.
- ❑ receivers uses association to synchronize playout of audio, video

# RTCP Bandwidth Scaling

- ❑ RTCP attempts to limit its traffic to 5% of session bandwidth.

## Example

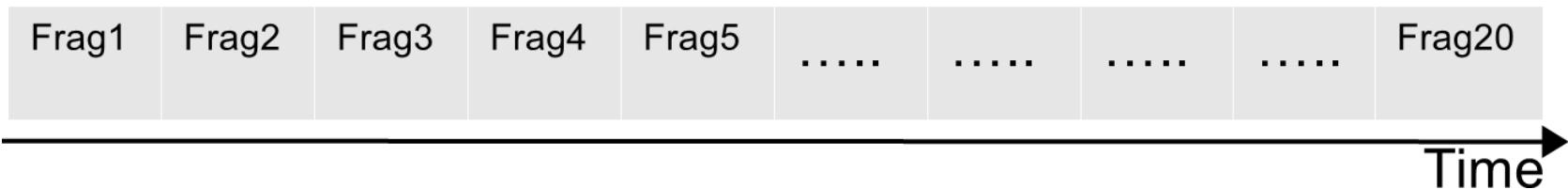
- ❑ Suppose one sender, sending video at 2 Mbps. Then RTCP attempts to limit its traffic to 100 Kbps.
- ❑ RTCP gives 75% of rate to receivers; remaining 25% to sender
- ❑ 75 kbps is equally shared among receivers:
  - with  $R$  receivers, each receiver gets to send RTCP traffic at  $75/R$  kbps.
- ❑ sender gets to send RTCP traffic at 25 kbps.
- ❑ participant determines RTCP packet transmission period by calculating avg RTCP packet size (across entire session) and dividing by allocated rate



# HTTP-based Adaptive Streaming (HAS)

- Please see notes  
from previous  
lecture ...

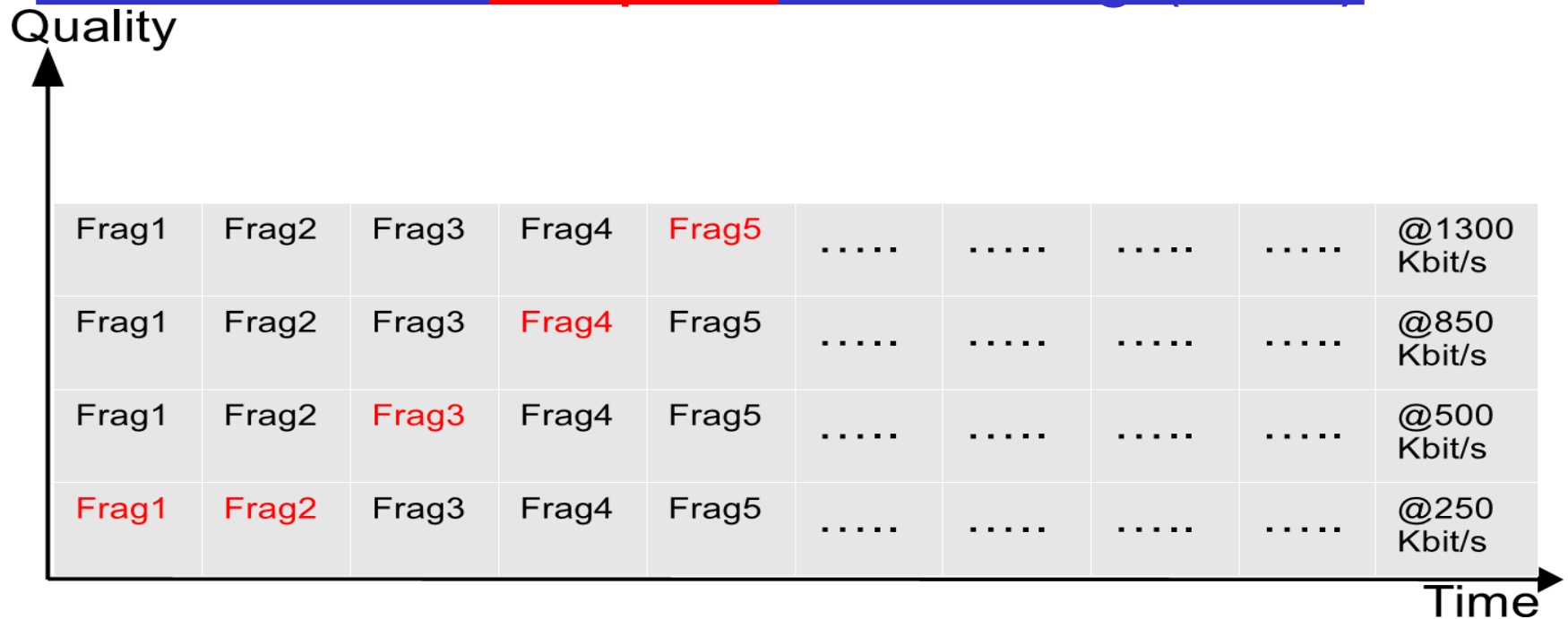
# HTTP-based streaming



## □ HTTP-based streaming

- Allows easy caching, NAT/firewall traversal, etc.
- Use of TCP provides natural bandwidth adaptation
- Split into fragments, download sequentially
- Some support for interactive VoD

# HTTP-based **adaptive** streaming (HAS)



## □ HTTP-based **adaptive** streaming





- Multiple encodings of each fragment (defined in manifest file)
- Clients adapt quality encoding based on (buffer and network) conditions

# HTTP-based Adaptive Streaming (HAS)

- ❑ Other terms for similar concepts: Adaptive Streaming, Smooth Streaming, HTTP Chunking
- ❑ Probably most important is *return to stateless server and TCP basis* of 1st generation
- ❑ Actually a series of small progressive downloads of chunks (or range requests)
- ❑ No standard protocol.
  - Apple HLS: HTTP Live Streaming
  - Microsoft IIS Smooth Streaming: part of Silverlight
  - Adobe: Flash Dynamic Streaming
  - DASH: Dynamic Adaptive Streaming over HTTP

# Example players

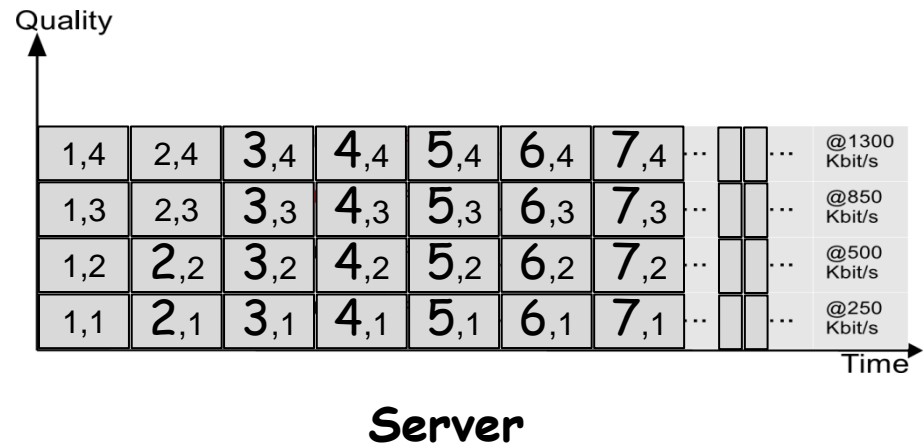
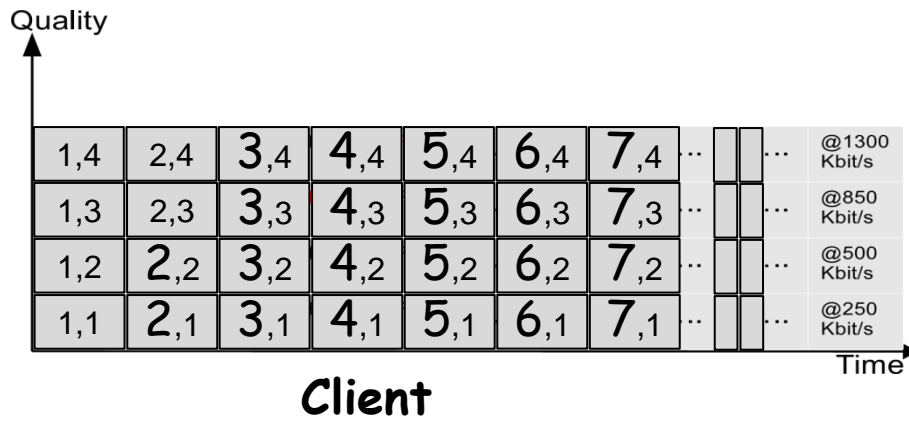


	Player	Container	Type	Open Source
 Microsoft Silverlight™	Microsoft Smooth Streaming	Silverlight	Chunk	✗
	Netflix player	Silverlight	Range	✗
	Apple HLS	QuickTime	Chunk	✗
	Adobe HDS	Flash	Chunk	✓

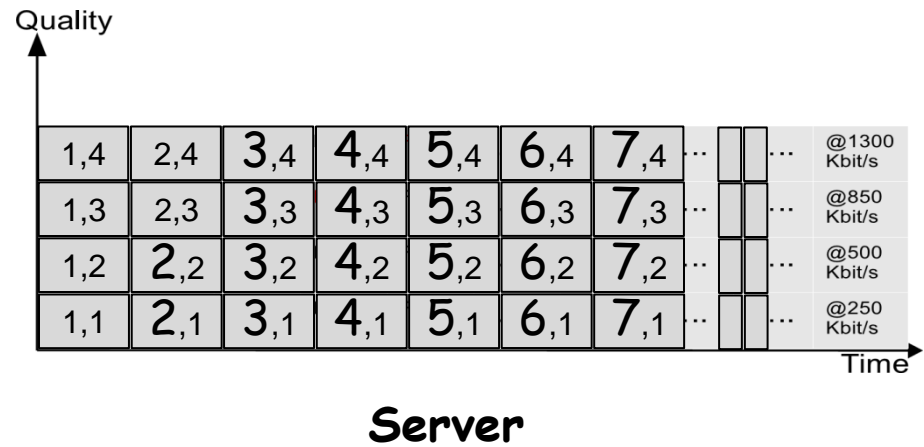
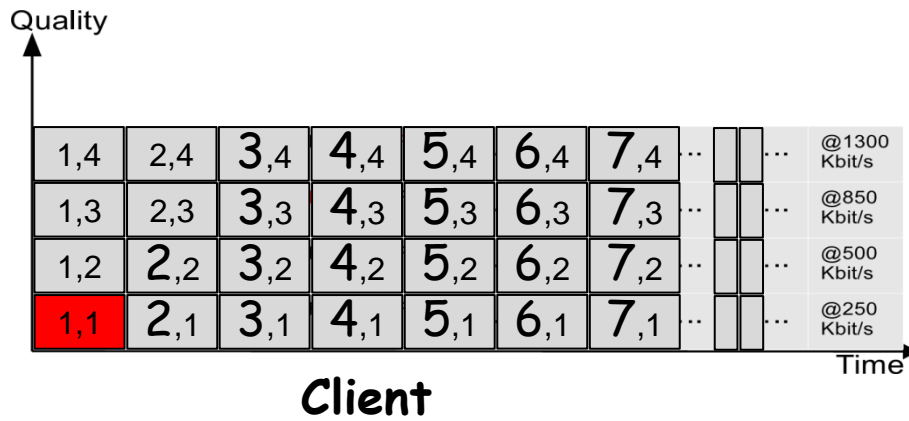
# Chunk-based streaming

- ❑ Chunks begin with keyframe so independent of other chunks
- ❑ Playing chunks in sequence gives seamless video
- ❑ Hybrid of streaming and progressive download:
  - Stream-like: sequence of small chunks requested as needed
  - Progressive download-like: HTTP transfer mechanism, stateless servers

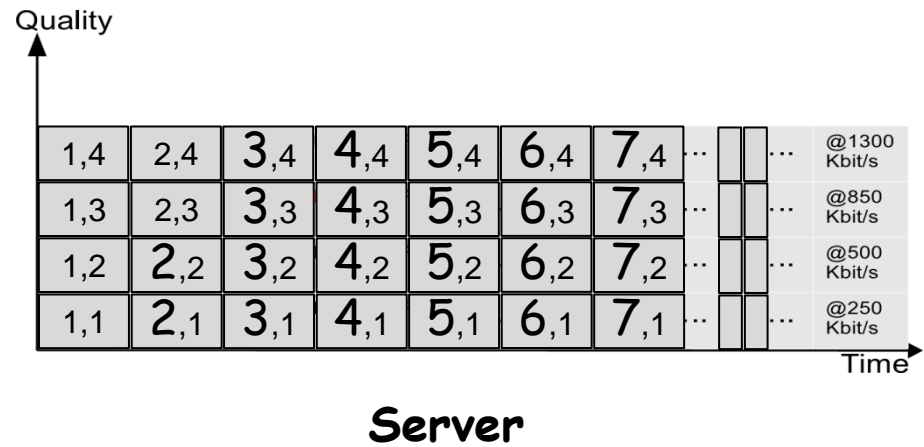
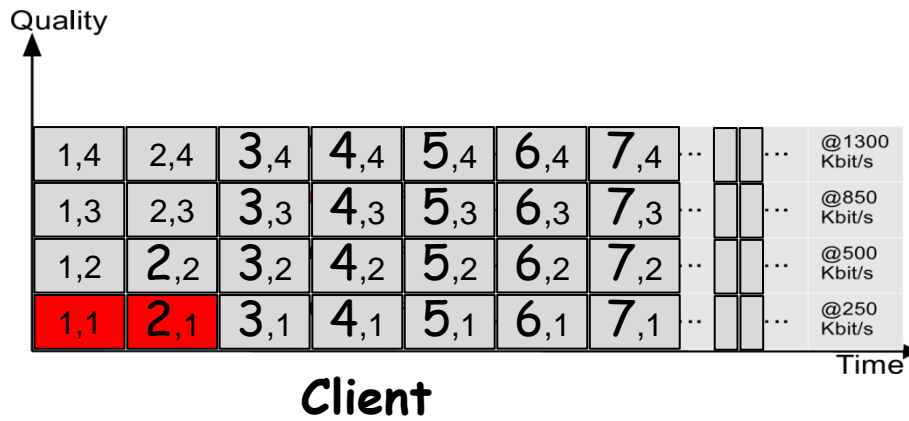
# Example



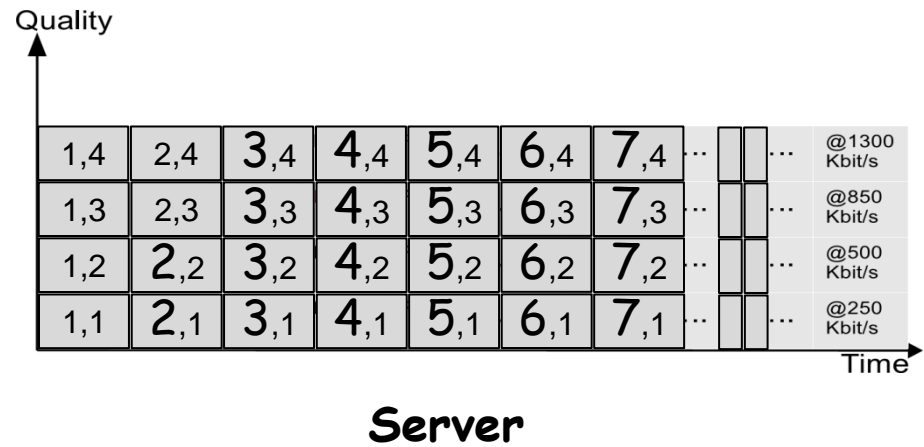
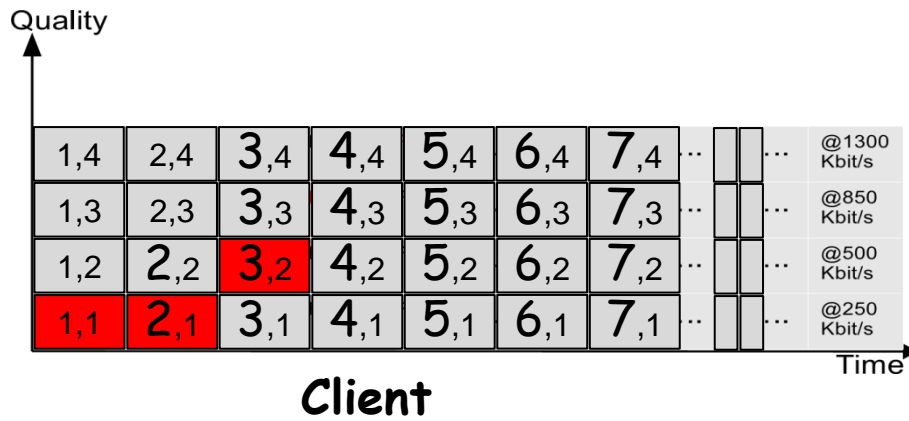
# Example



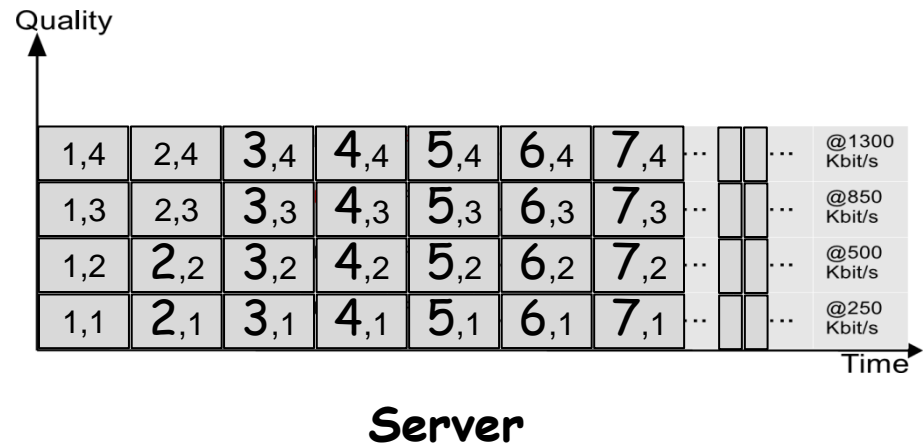
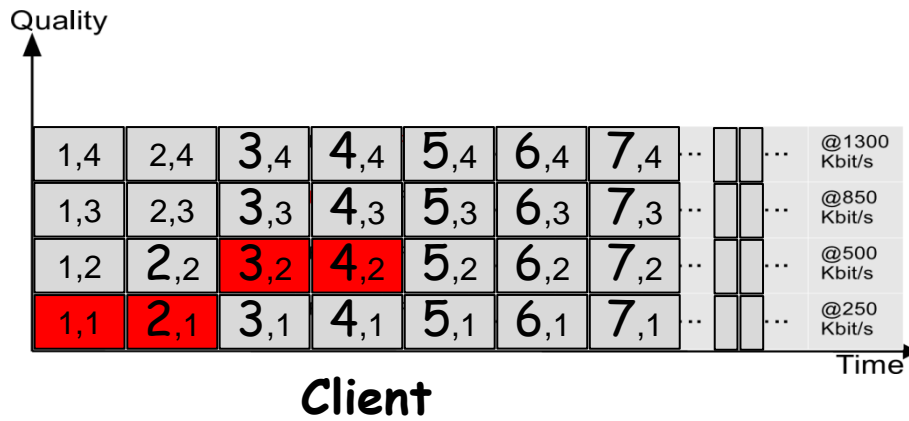
# Example



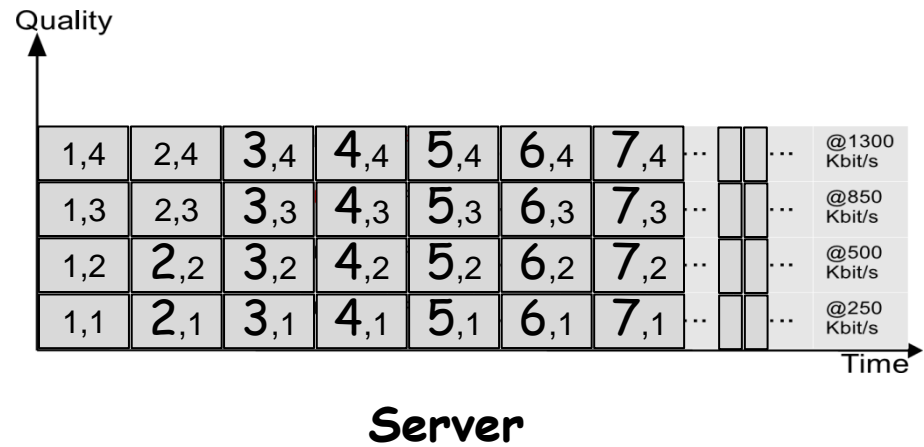
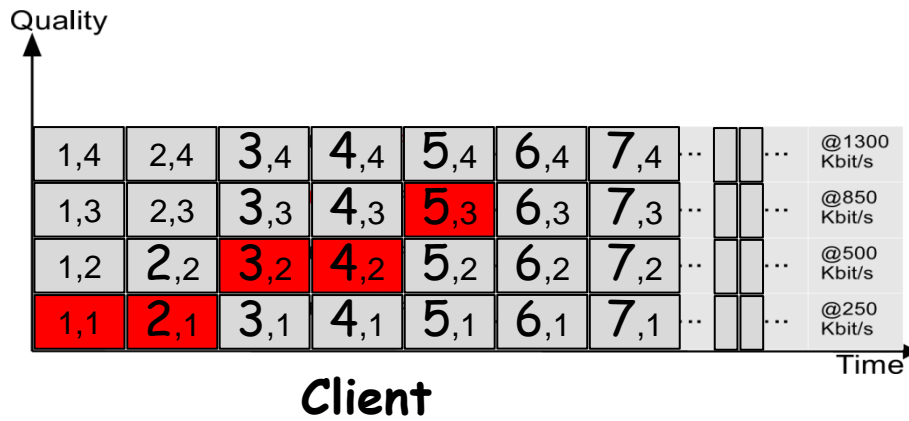
# Example



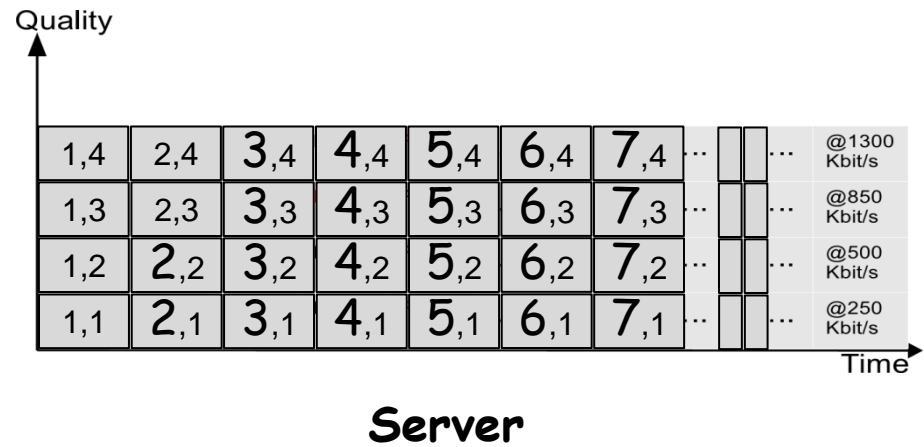
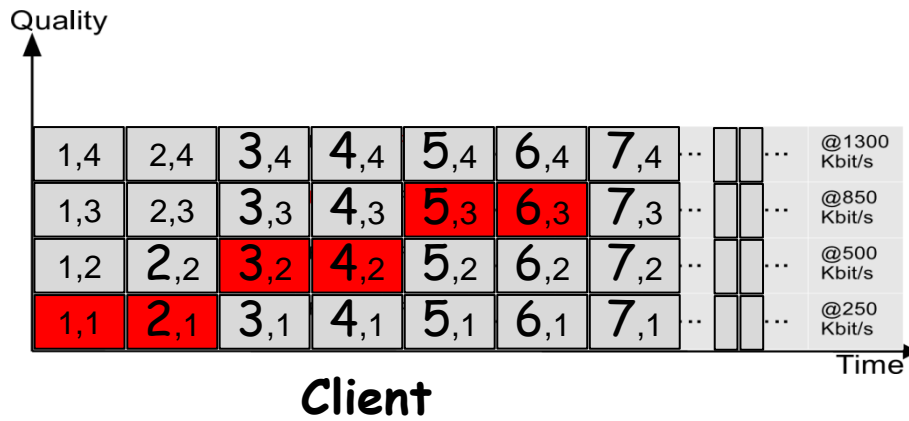
# Example



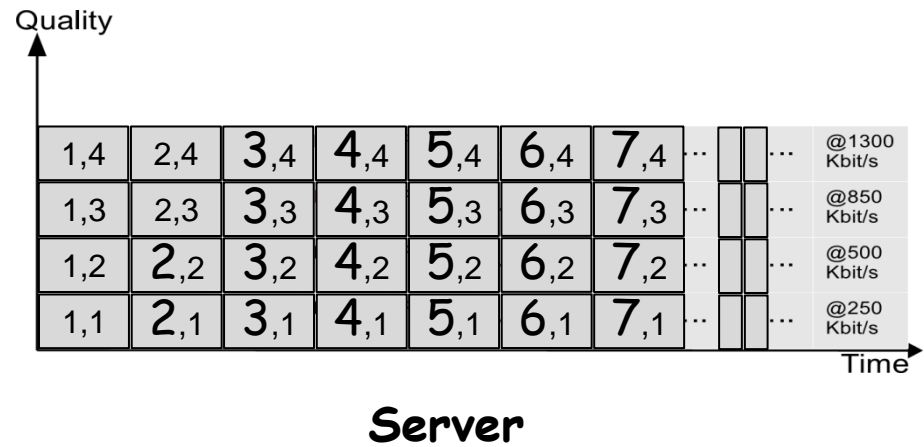
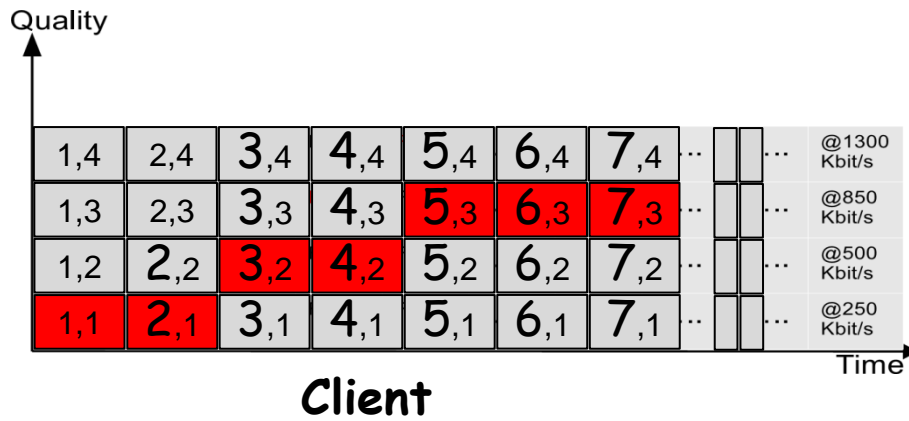
# Example



# Example



# Example



# Example: HAS and proxy



## **Clients' want**

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

# HTTP Streaming (2)

## □ Adaptation:

- Encode video at different levels of quality/bandwidth
- Client can adapt by requesting different sized chunks
- Chunks of different bit rates must be synchronized: All encodings have the same chunk boundaries and all chunks start with keyframes, so you can make smooth splices to chunks of higher or lower bit rates

## □ Evaluation:

- Easy to deploy: it's just HTTP, caches/proxies/CDN all work
- Fast startup by downloading lowest quality/smallest chunk
- Bitrate switching is seamless
- Many small files

## □ Chunks can be

- Independent files -- many files to manage for one movie
- Stored in single file container -- client or server must be able to access chunks, e.g. using range requests from client.

# Examples: Netflix & Silverlight

- ❑ Netflix servers allow users to search & select movies
- ❑ Netflix manages accounts and login
- ❑ Movie represented as an XML encoded "manifest" file with URL for each copy of the movie:
  - Multiple bitrates
  - Multiple CDNs (preference given in manifest)
- ❑ Microsoft Silverlight DRM manages access to decryption key for movie data
- ❑ CDNs do no encryption or decryption, just deliver content via HTTP.
- ❑ Clients use "Range-bytes=" in HTTP header to stream the movie in chunks.

# Example: HAS and proxy

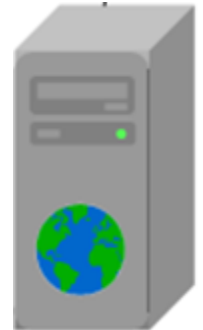


## **Clients' want**

- ☐ 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



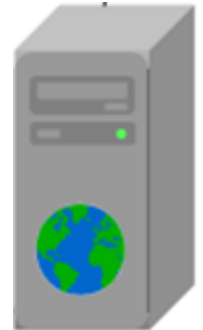
## **Clients' want**

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

## **Network providers' want**

- High QoE of customers/clients

# Example: HAS and proxy



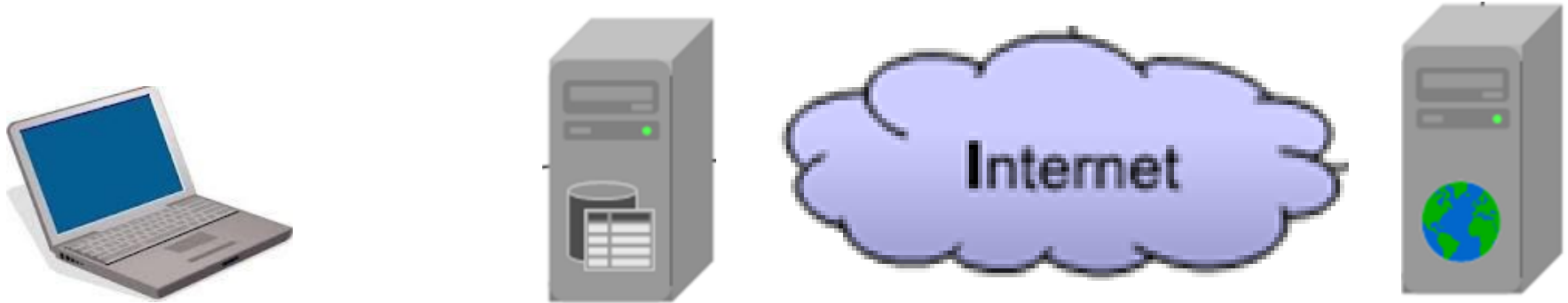
## **Clients' want**

- ☐ 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



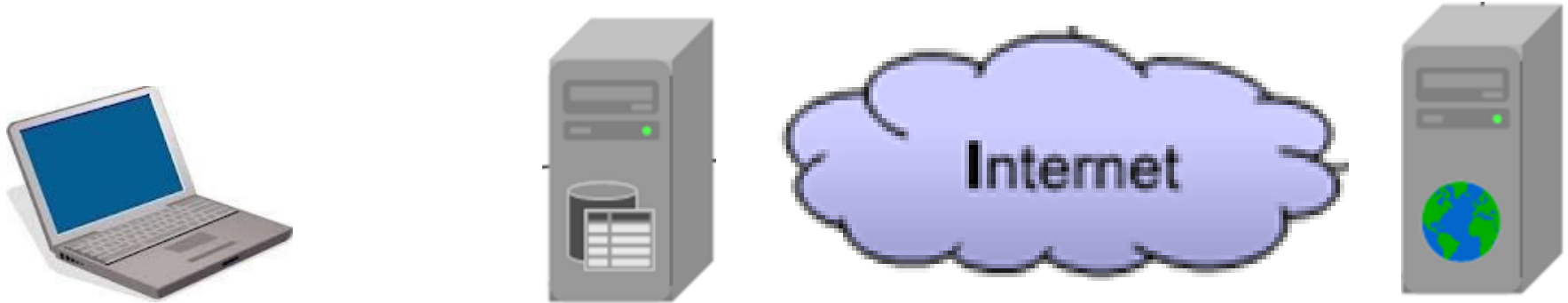
## **Clients' want**

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

## **Network providers' want**

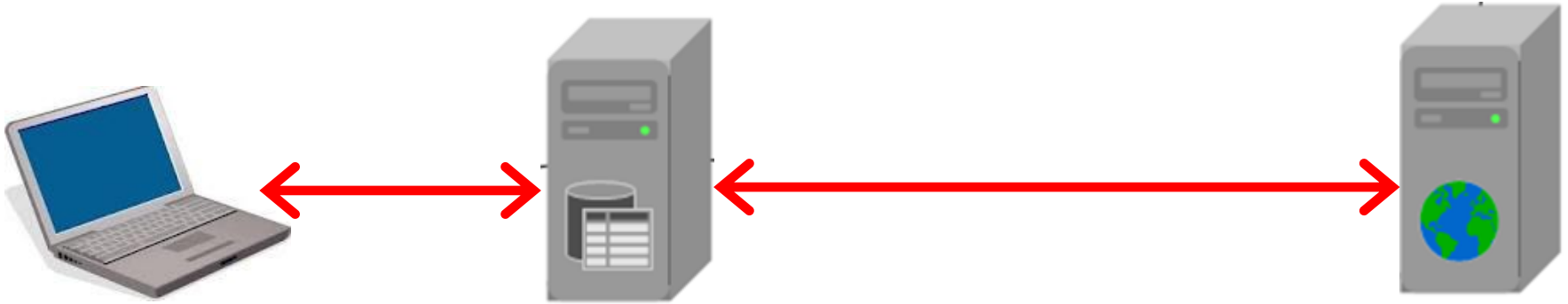
- High QoE of customers/clients
- Low bandwidth usage
- High hit rate

## Example: HAS and proxy

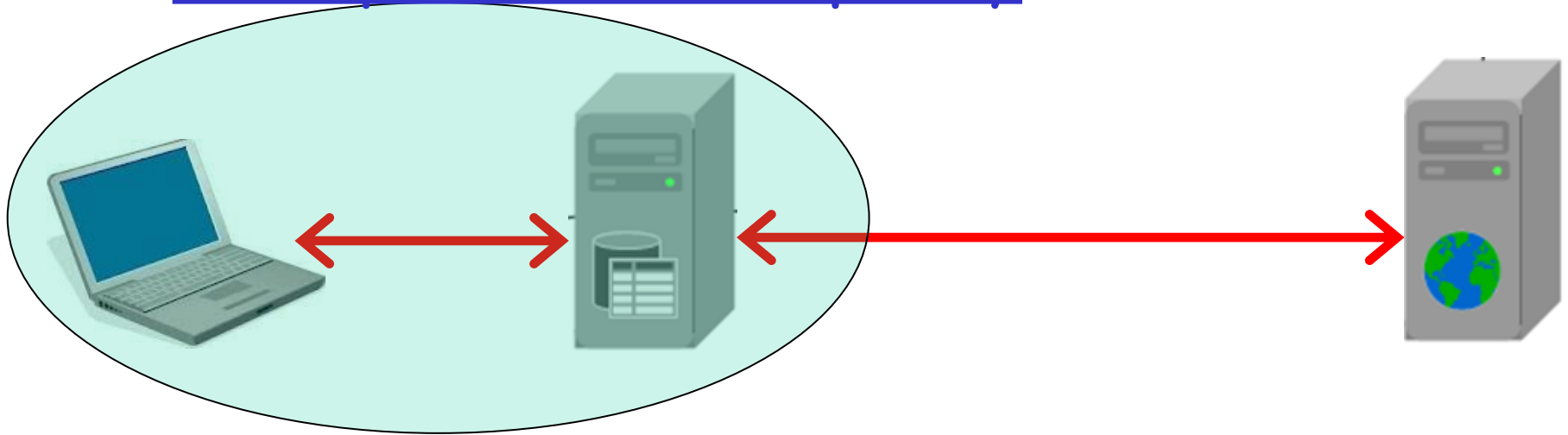


Proxy example ...

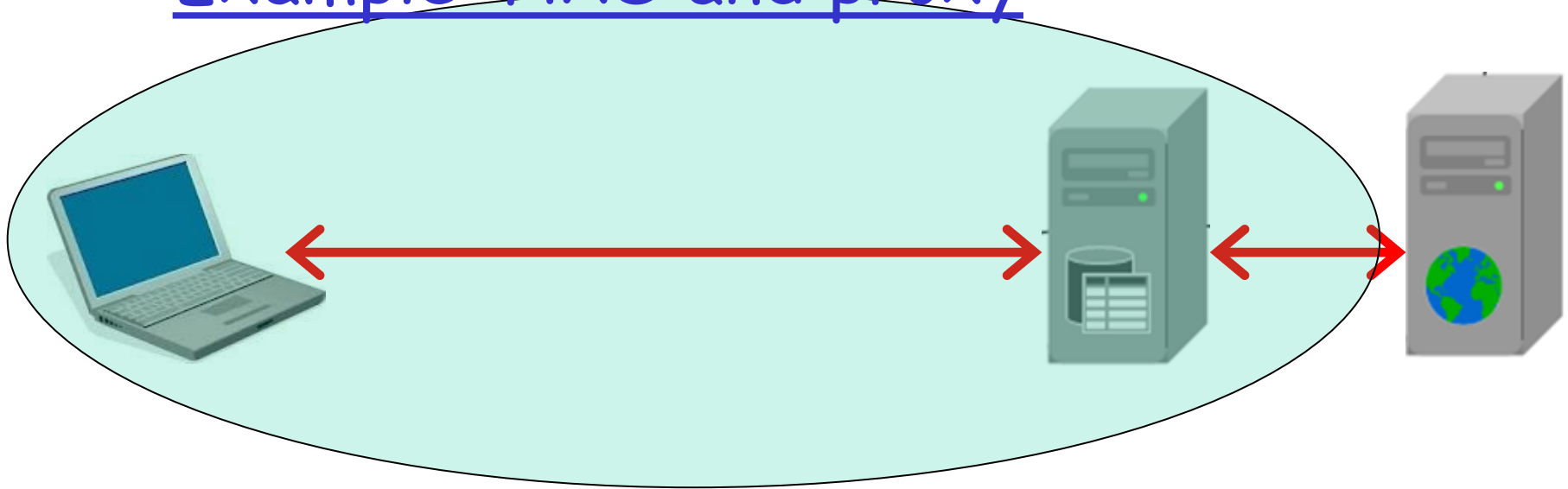
## Example: HAS and proxy



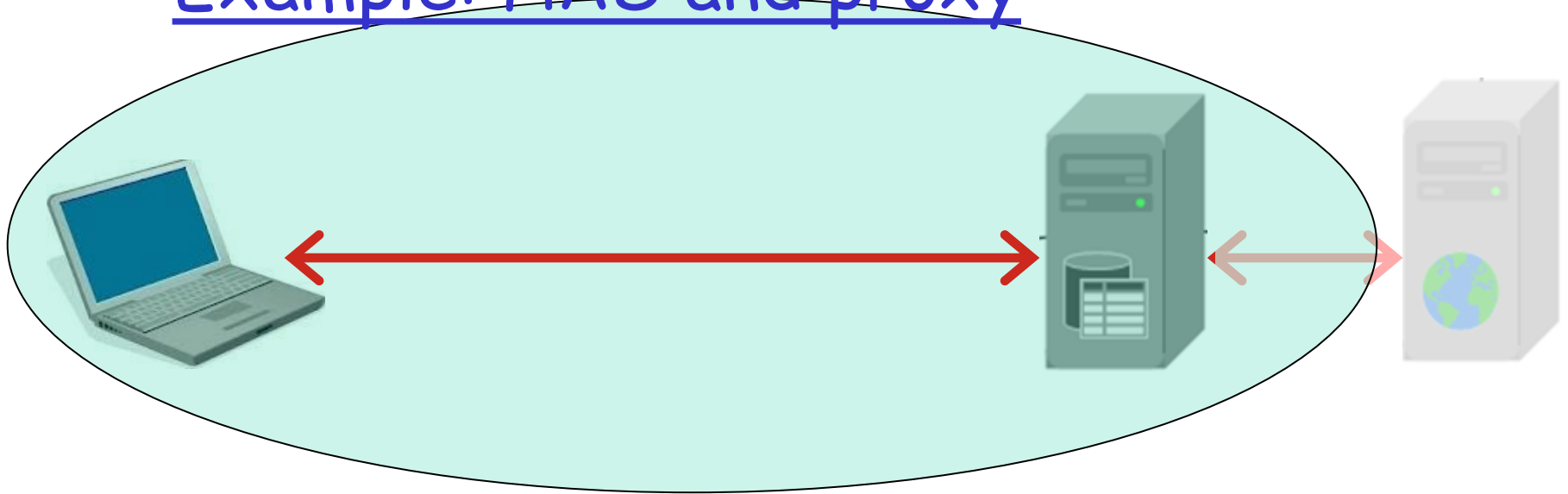
## Example: HAS and proxy



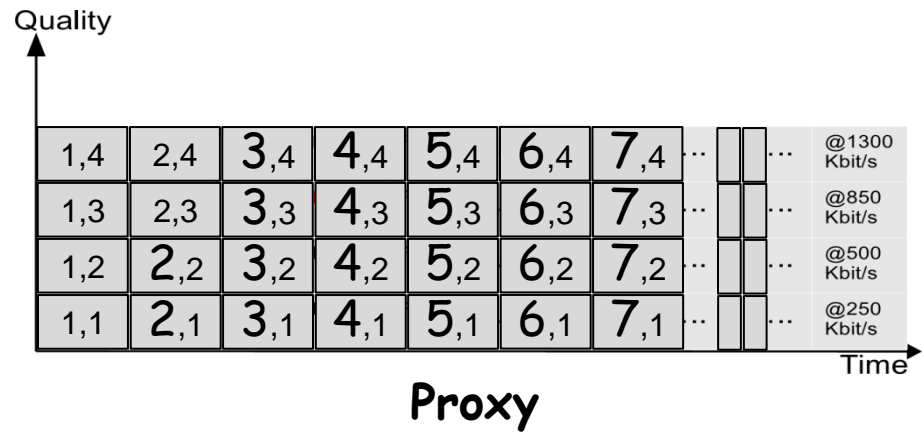
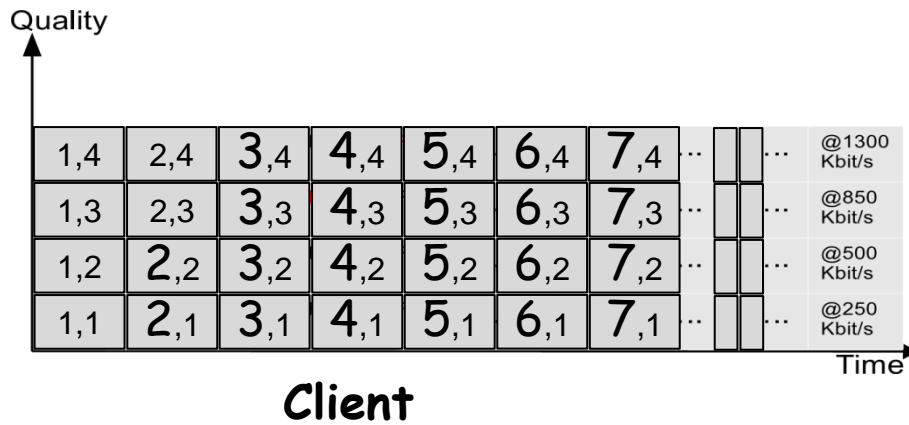
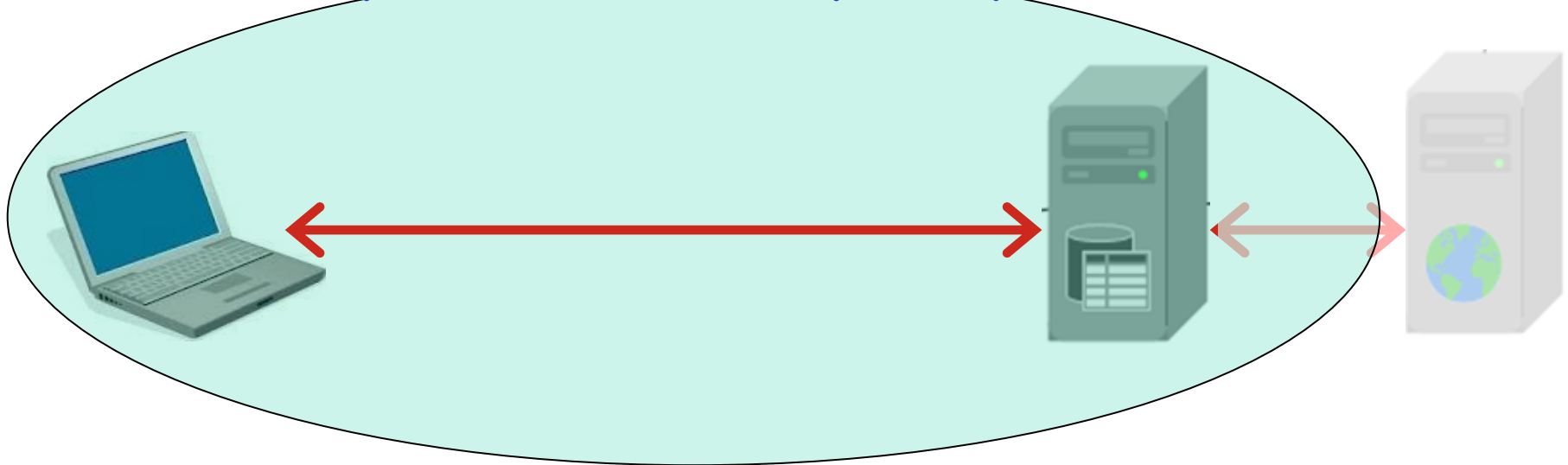
## Example: HAS and proxy



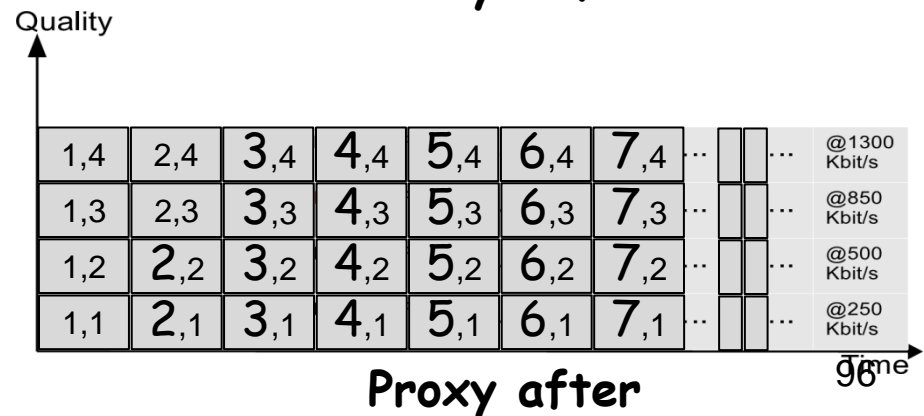
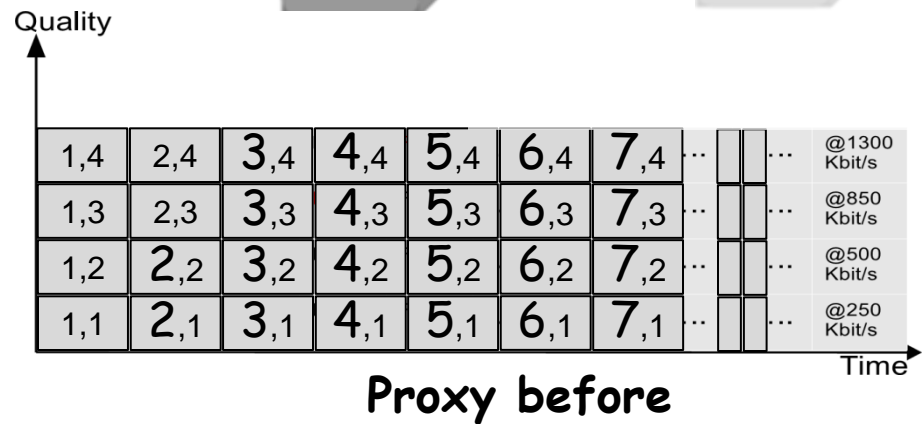
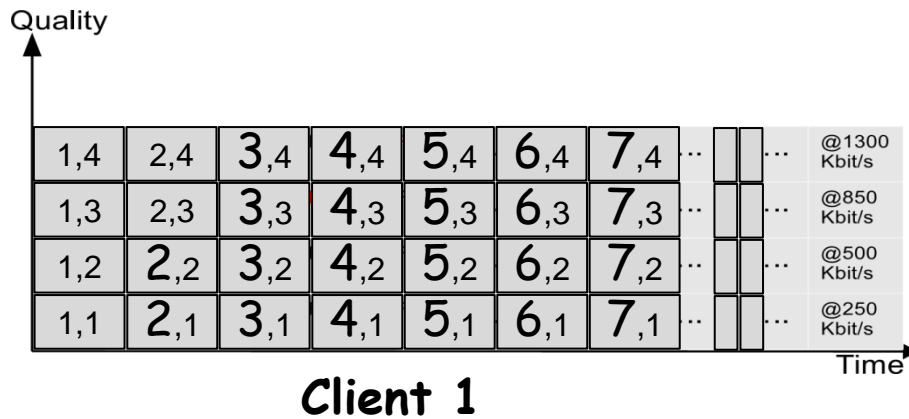
## Example: HAS and proxy



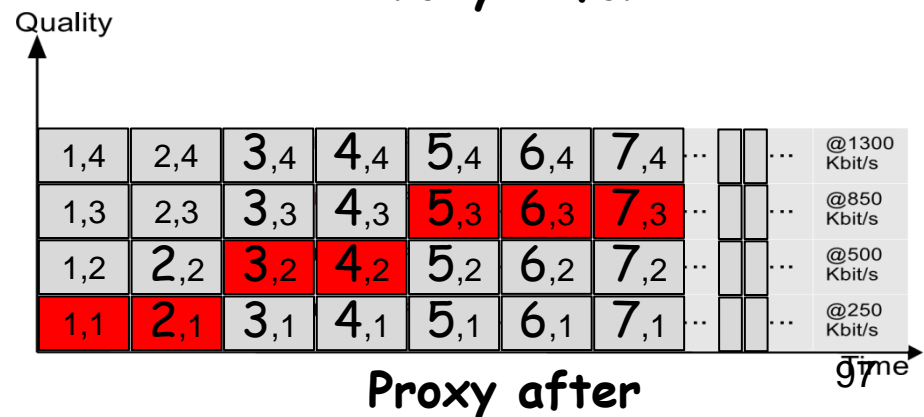
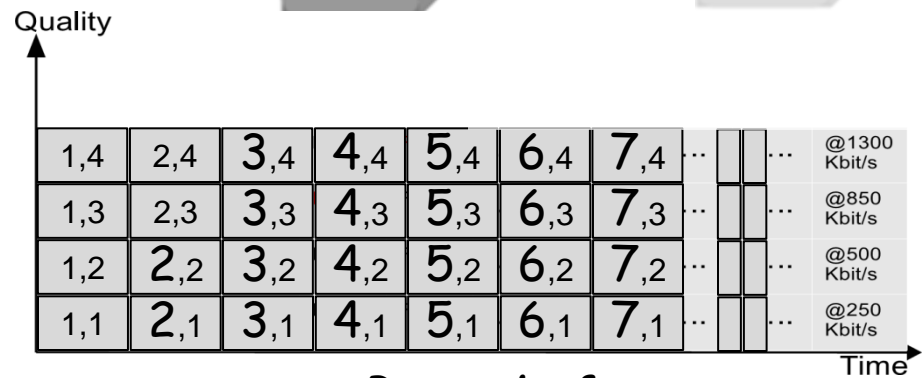
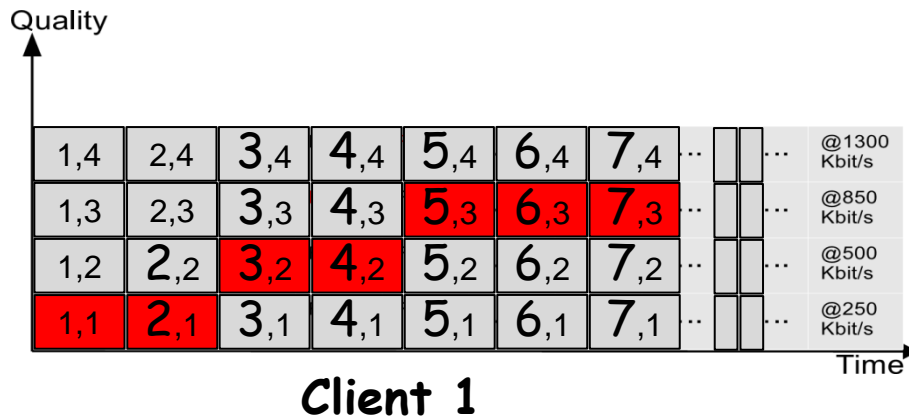
# Example: HAS and proxy



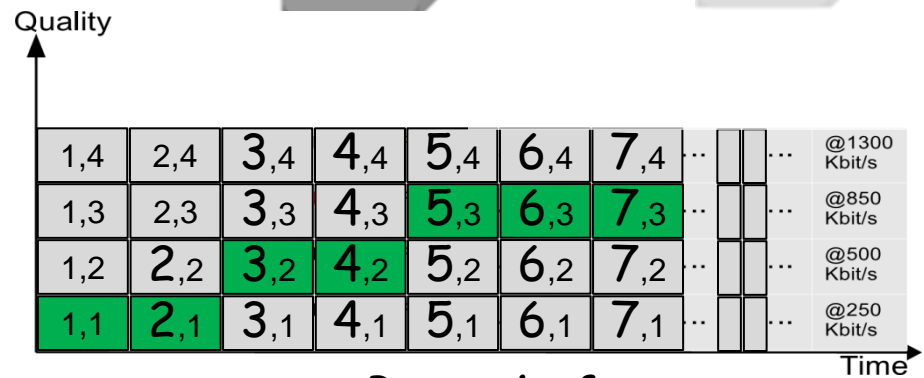
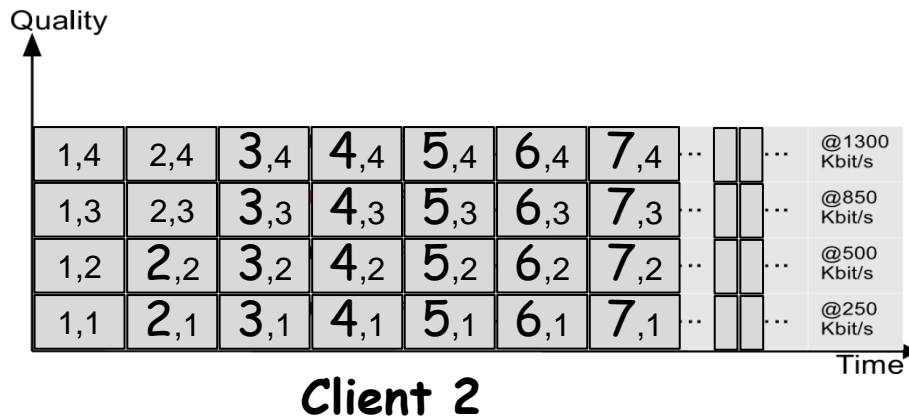
# Example: HAS and proxy



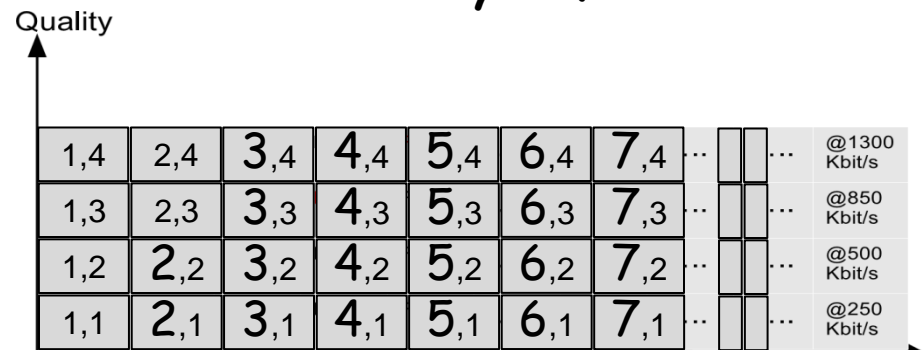
# Example: HAS and proxy



# Example: HAS and proxy

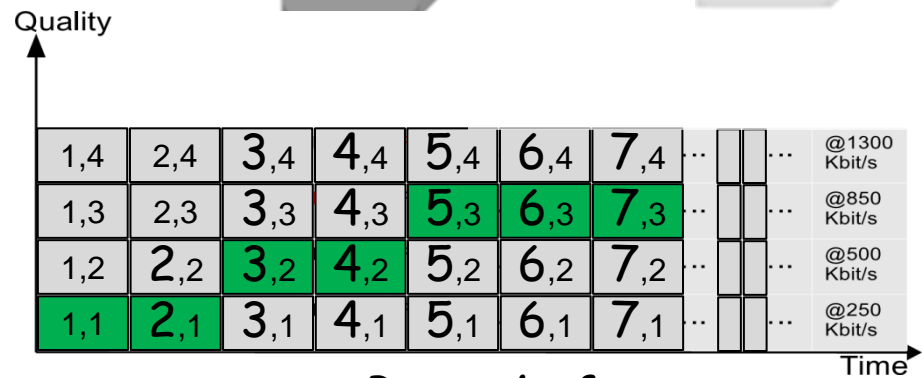
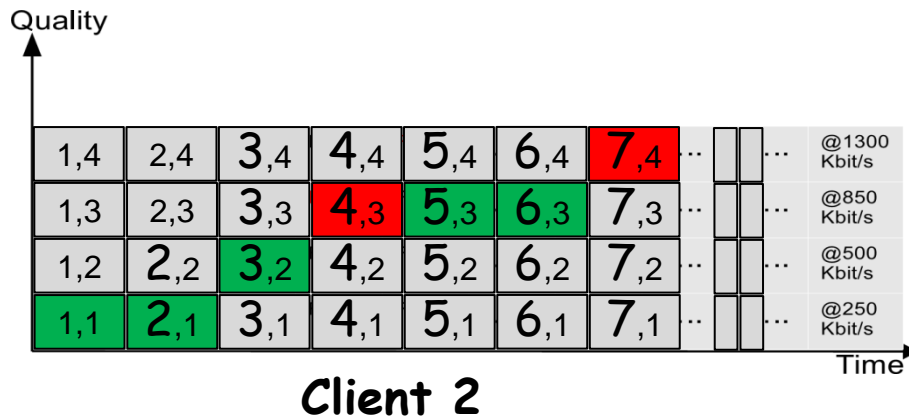


**Proxy before**

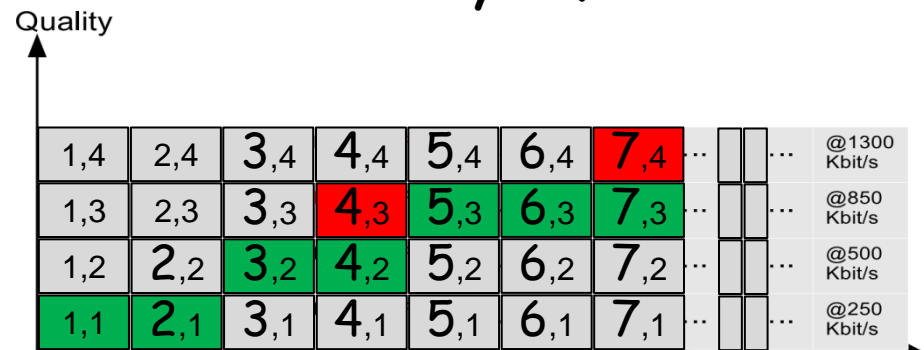


**Proxy after**

# Example: HAS and proxy

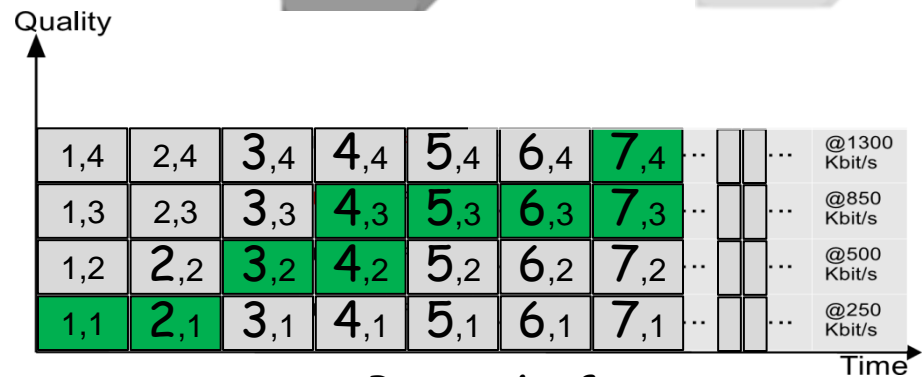
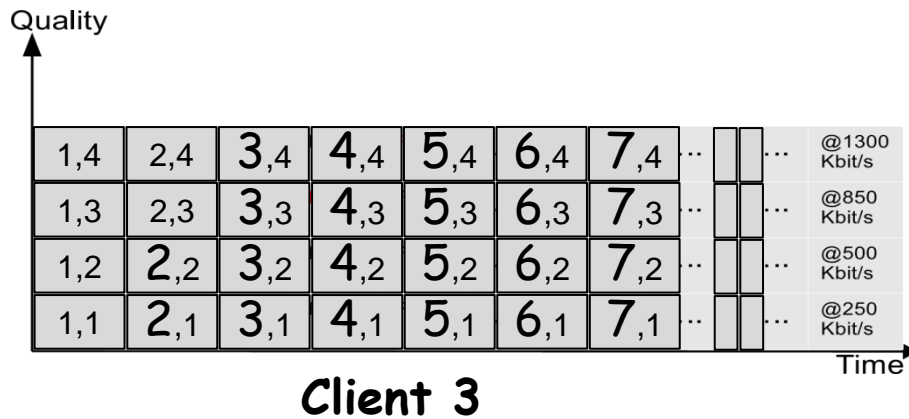


**Proxy before**

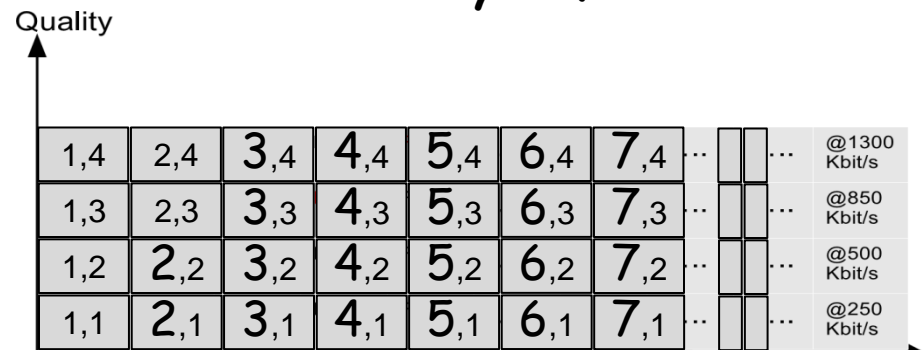


**Proxy after**

# Example: HAS and proxy

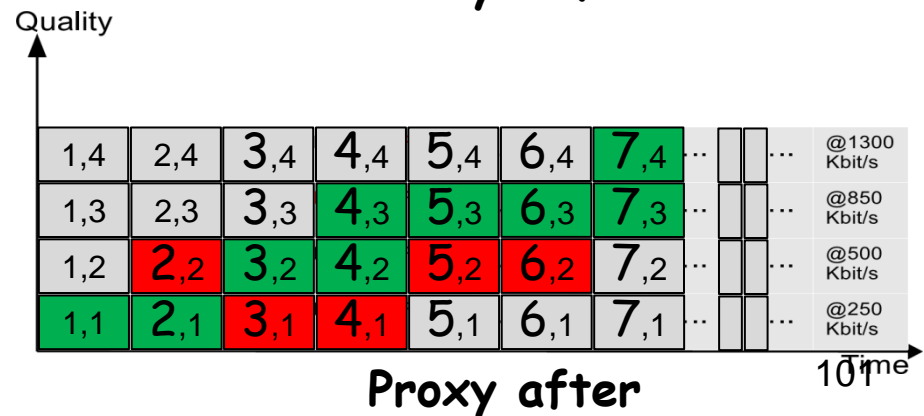
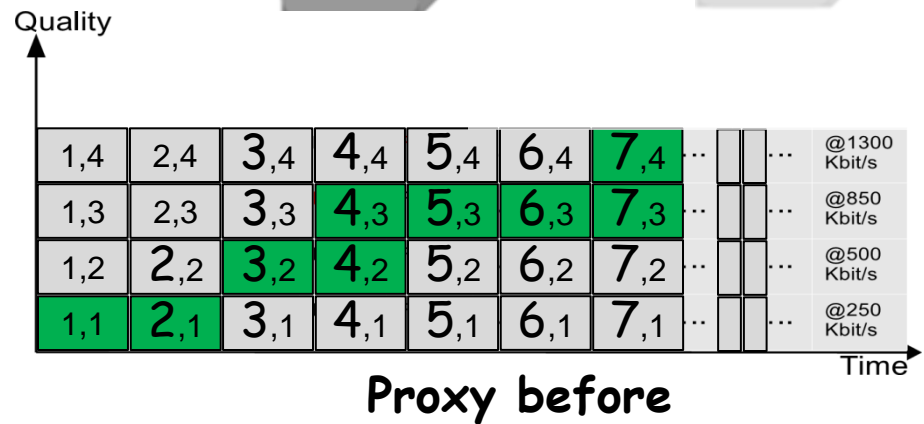
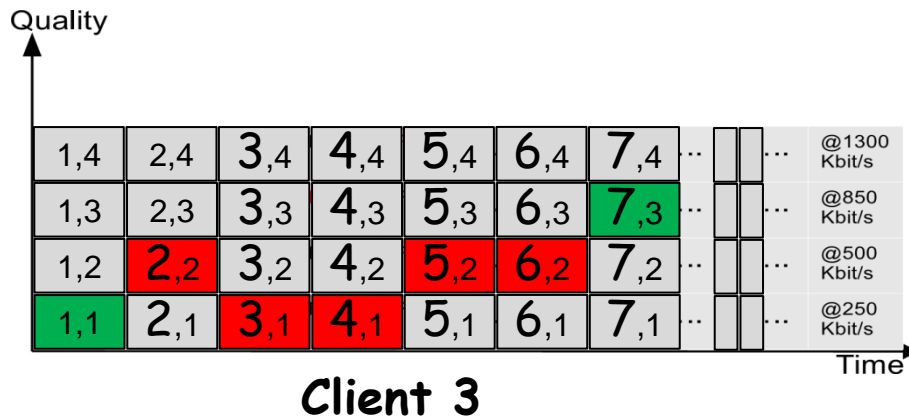


**Proxy before**



**Proxy after**

# Example: HAS and proxy





# Multimedia Over "Best Effort" Internet

- **TCP/UDP/IP:** *no guarantees on delay, loss*



? ? ? ? ?  
But you said multimedia apps requires ?  
QoS and level of performance to be  
? effective! ? ?



Today's multimedia applications implement functionality at the app. layer to mitigate (as best possible) effects of delay, loss

# Packet Loss

- ❑ **network loss:** IP datagram lost due to network congestion (router buffer overflow) or losses at wireless link(s)
- ❑ **delay loss:** IP datagram arrives too late for playout at receiver (effectively the same as if it was lost)
  - delays: processing, queueing in network; end-system (sender, receiver) delays
  - Tolerable delay depends on the application
- ❑ How can packet loss be handled?
  - We will discuss this next ...

# Receiver-based Packet Loss Recovery

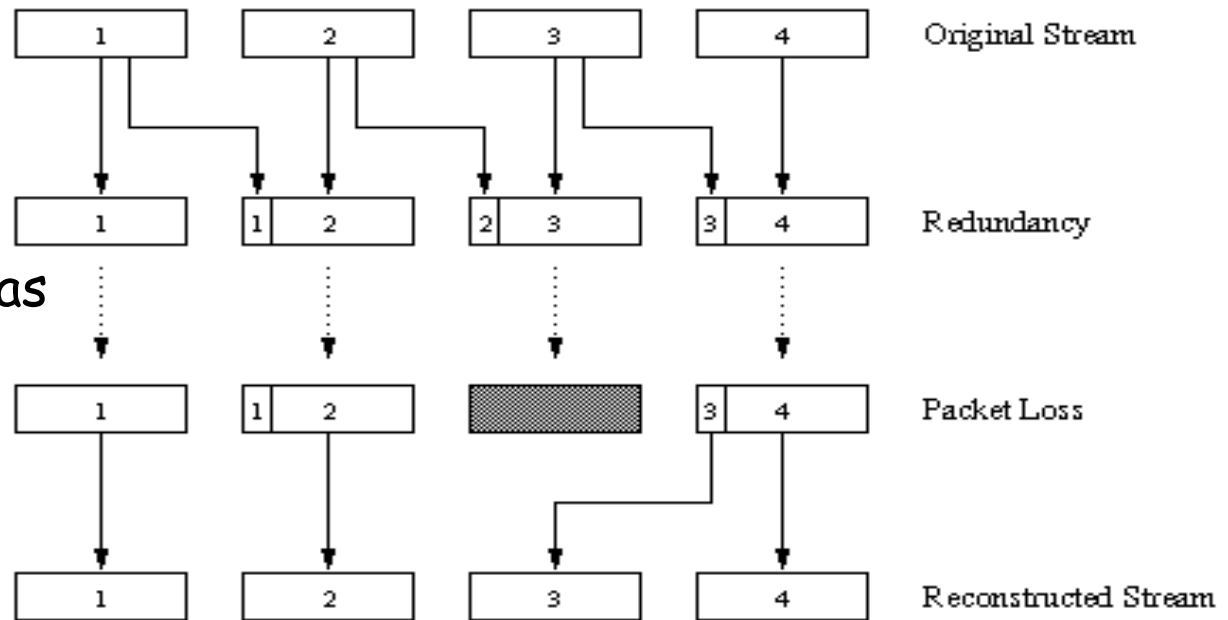
- ❑ Generate replacement packet
  - Packet repetition
  - Interpolation
  - Other sophisticated schemes
- ❑ Works when audio/video streams exhibit short-term correlations (e.g., self-similarity)
- ❑ Works for relatively low loss rates (e.g.,  $< 5\%$ )
- ❑ Typically, breaks down on “bursty” losses

# Forward Error Correction (FEC)

- ❑ For every group of  $n$  actual media packets, generate  $k$  additional redundant packets
- ❑ Send out  $n+k$  packets, which increases the bandwidth consumption by factor  $k/n$ .
- ❑ Receiver can reconstruct the original  $n$  media packets provided at most  $k$  packets are lost from the group
- ❑ Works well at high loss rates (for a proper choice of  $k$ )
- ❑ Handles "bursty" packet losses
- ❑ Cost: increase in transmission cost (bandwidth)

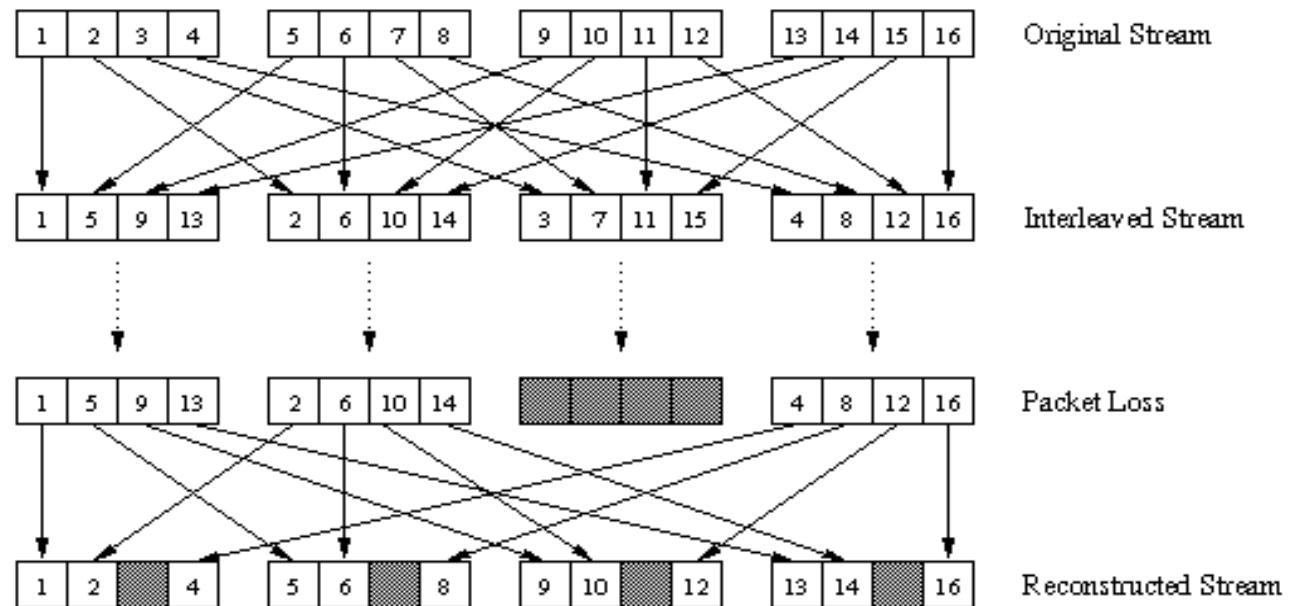
# Another FEC Example

- “piggyback lower quality stream”
- Example: send lower resolution audio stream as the redundant information
- 



- Whenever there is non-consecutive loss, the receiver can conceal the loss.
- Can also append (n-1)st and (n-2)nd low-bit rate chunk

# Interleaving: Recovery from packet loss



## Interleaving

- ❑ Intentionally alter the sequence of packets before transmission
- ❑ Better robustness against "burst" losses of packets
- ❑ Results in increased playout delay from inter-leaving



More slides ....

# Outline

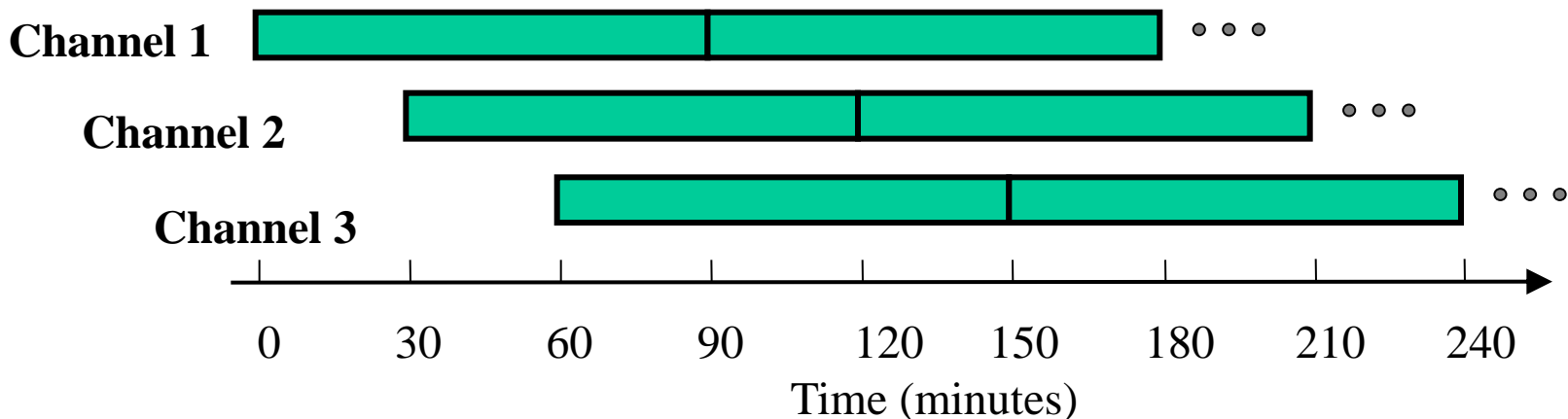
- ❑ Multimedia Networking Applications
- ❑ Streaming stored audio and video
- ❑ Scalable Streaming Techniques
- ❑ Content Distribution Networks
- ❑ Beyond Best Effort

# Streaming Popular Content

- ❑ Consider a popular media file
  - Playback rate: 1 Mbps
  - Duration: 90 minutes
  - Request rate: once every minute
- ❑ How can a video server handle such high loads?
  - **Approach 1:** Start a new "stream" for each request
  - Allocate server and disk I/O bandwidth for each request
  - Bandwidth required at server =  $1 \text{ Mbps} \times 90$

# Streaming Popular Content using Batching

- ❑ **Approach 2:** Leverage the multipoint delivery capability of modern networks
- ❑ Playback rate = 1 Mbps, duration = 90 minutes
- ❑ Group requests in non-overlapping intervals of 30 minutes:
  - Max. start-up delay = 30 minutes
  - Bandwidth required = 3 channels = 3 Mbps

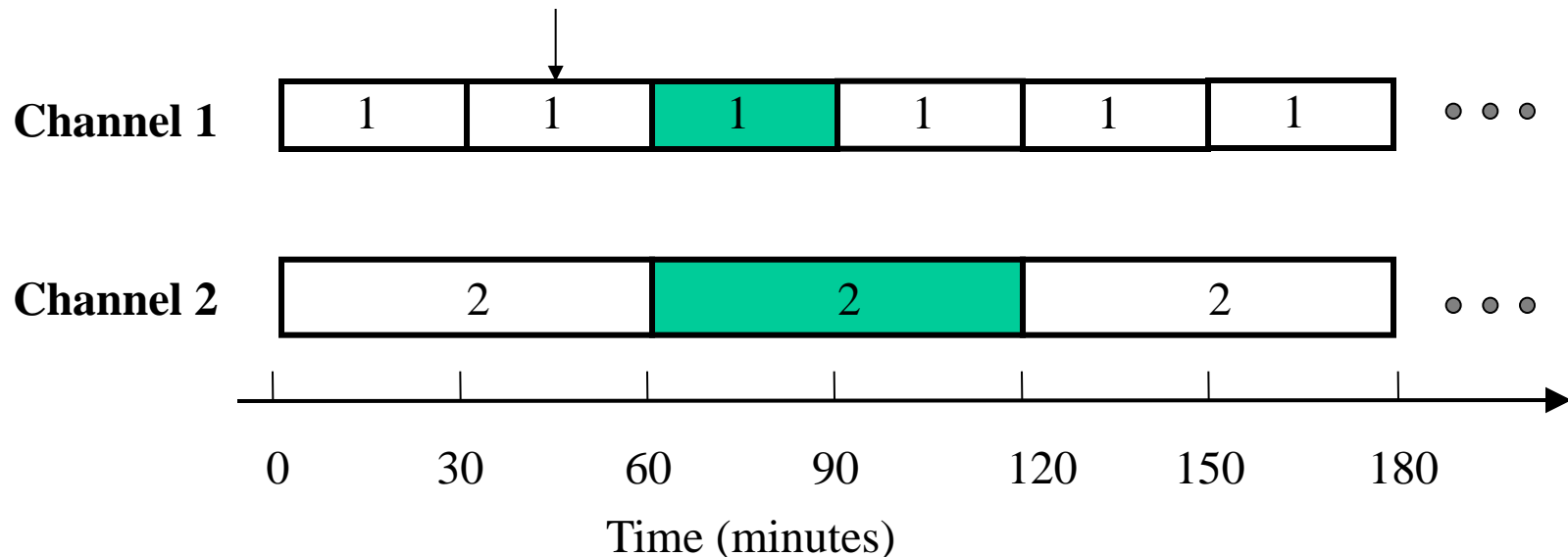


# Batching Issues

- ❑ Bandwidth increases linearly with decrease in start-up delays
- ❑ Can we reduce or eliminate “start-up” delays?
  - Periodic Broadcast Protocols
  - Stream Merging Protocols

# Periodic Broadcast Example

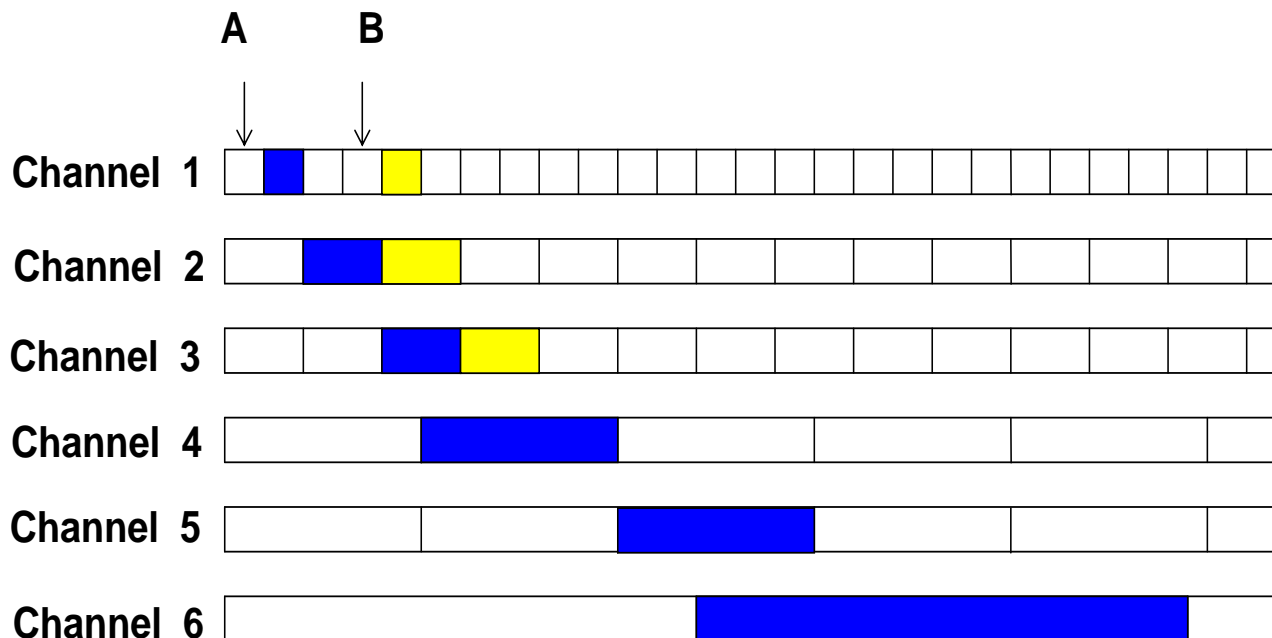
- ❑ Partition the media file into 2 segments with relative sizes {1, 2}. For a 90 min. movie:
  - Segment 1 = 30 minutes, Segment 2 = 60 minutes
- ❑ Advantage:
  - Max. start-up delay = 30 minutes
  - Bandwidth required = 2 channels = 2 Mbps
- ❑ Disadvantage: Requires increased client capabilities



# Skyscraper Broadcasts (SB)

[Hua & Sheu 1997]

- Divide the file into *K* segments of increasing size
  - Segment size progression: 1, 2, 2, 5, 5, 12, 12, 25, ...
- Multicast each segment on a separate channel at the playback rate
- Aggregate rate to clients: *2 x playback rate*



# Comparing Batching and SB

Server Bandwidth	Start-up Delay	
	Batching	SB
1 Mbps	90 minutes	90 minutes
2 Mbps	45 minutes	30 minutes
6 Mbps	15 minutes	3 minutes
10 Mbps	9 minutes	30 seconds

❑ Playback rate = 1 Mbps, duration = 90 minutes

❑ Limitations of Skyscraper:

- Ad hoc segment size progress
- Does not work for low client data rates

# Reliable Periodic Broadcasts (RPB)

[Mahanti *et al.* 2001, 2003, 2004]

- ❑ **Optimized PB protocols** (no packet loss recovery)
  - client fully downloads each segment before playing
  - required server bandwidth near minimal
  - Segment size progression is *not* ad hoc
  - Works for client data rates  $< 2 \times \text{playback rate}$
- ❑ extend for packet loss recovery
- ❑ extend for “bursty” packet loss
- ❑ extend for client heterogeneity

# Reliable Periodic Broadcasts (RPB)

[Mahanti *et al.* 2001, 2003, 2004]

- ❑ **Optimized PB protocols** (no packet loss recovery)
  - client fully downloads each segment before playing
  - required server bandwidth near minimal
  - Segment size progression is *not* ad hoc
  - Works for client data rates  $< 2 \times \text{playback rate}$
- ❑ extend for packet loss recovery
- ❑ extend for “bursty” packet loss
- ❑ extend for client heterogeneity

# Optimized Periodic Broadcasts



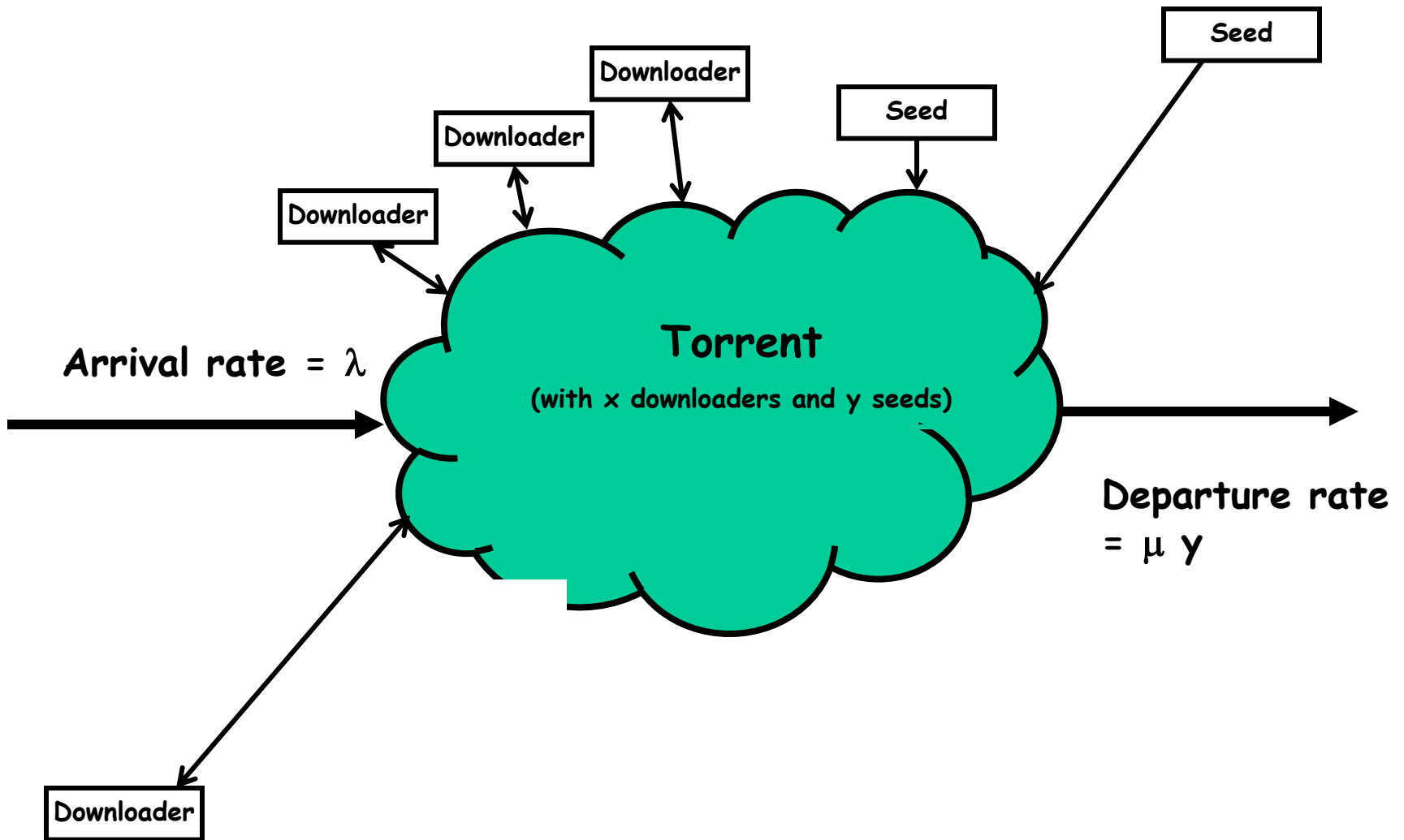
- $r$  = segment streaming rate = 1
- $s$  = maximum # streams client listens to concurrently = 2
- $b$  = client data rate =  $s \times r = 2$

□ length of first  $s$  segments:  $\frac{1}{r}l_k = \frac{1}{r}l_1 + \sum_{j=1}^{k-1} l_j$

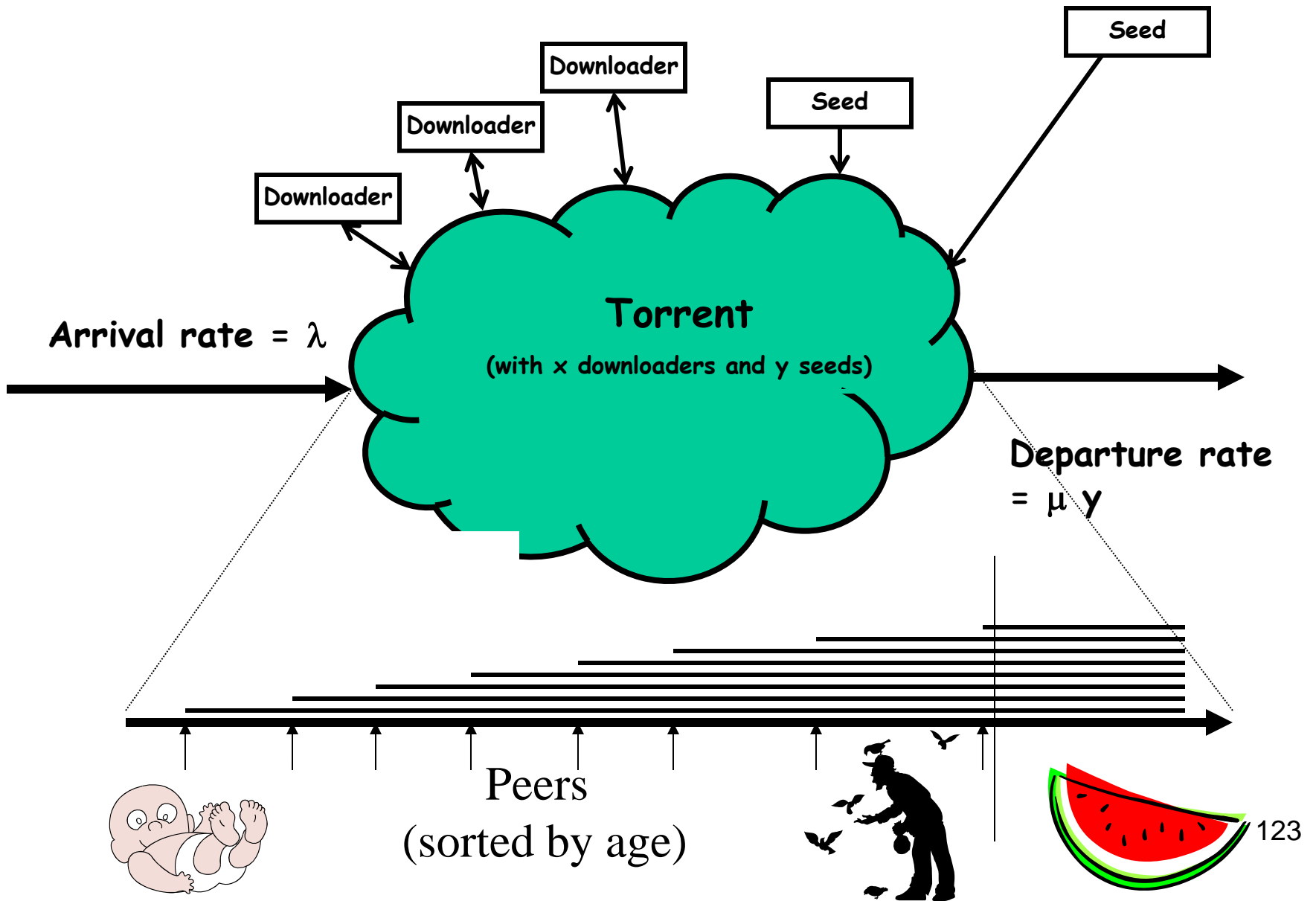
□ length of segment  $k > s$ :  $\frac{1}{r}l_k = \sum_{j=k-s}^{k-1} l_j$



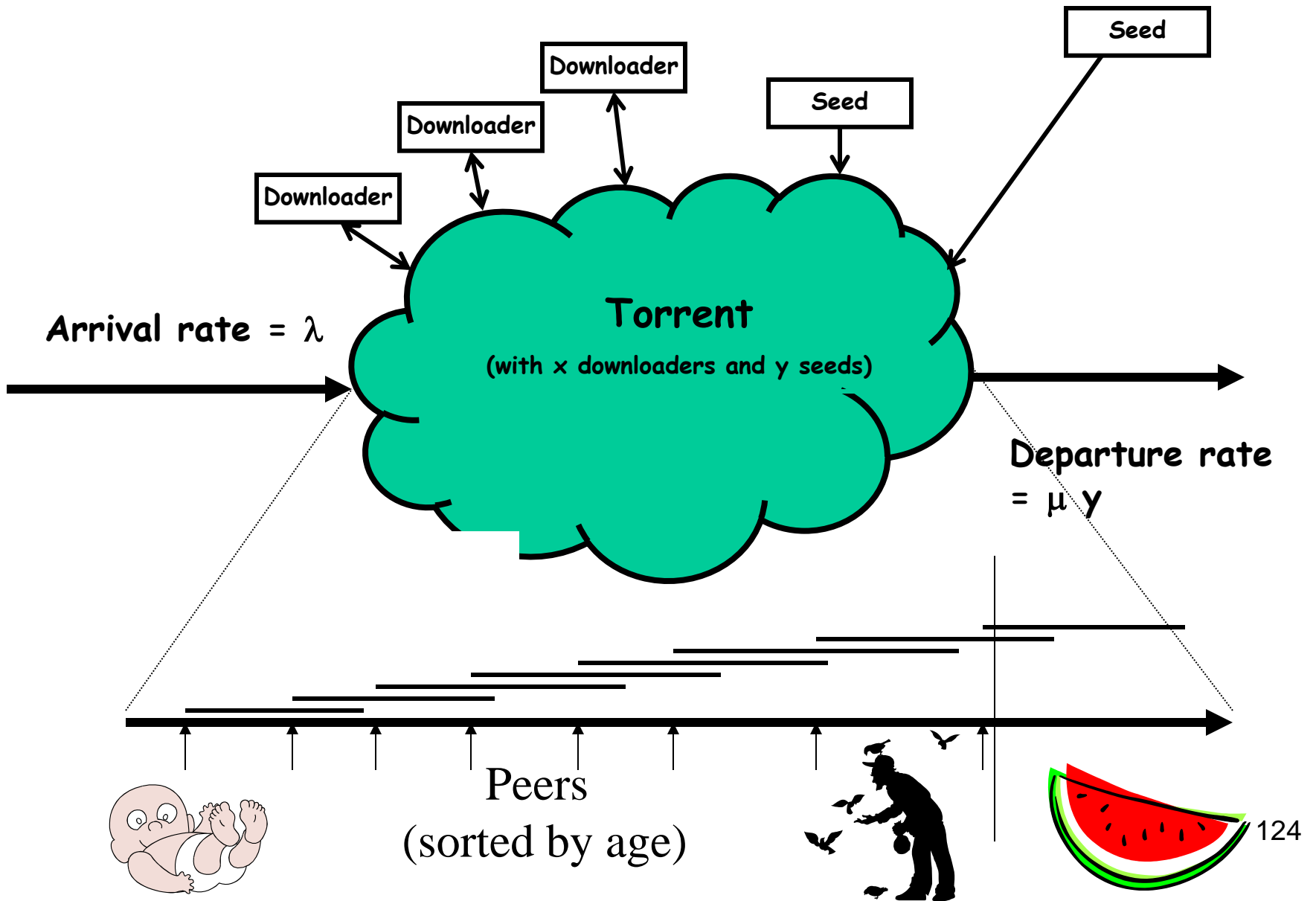
# BitTorrent Model



# BitTorrent Model (random)



# BitTorrent Model (chaining)

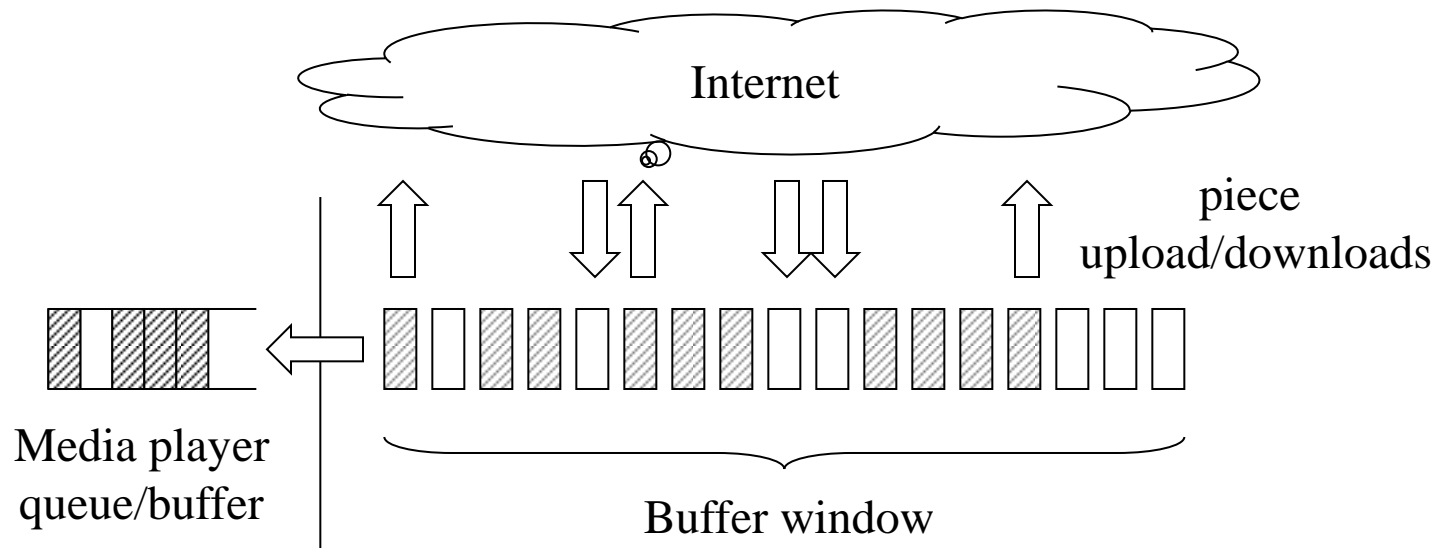


# Peer-assisted VoD streaming

## Some research questions ...

- ❑ Can BitTorrent-like protocols provide scalable on-demand streaming?
- ❑ How sensitive is the performance to the application configuration parameters?
  - Piece selection policy (rarest vs. in-order tradeoff)
  - Peer selection policy
  - Upload/download bandwidth
- ❑ What is the user-perceived performance?
  - Start-up delay
  - Probability of disrupted playback

# Live Streaming using BT-like systems



## ❑ Live streaming (e.g., CoolStreaming)

- All peers at roughly the same play/download position
  - High bandwidth peers can easily contribute more ...
- (relatively) Small buffer window
  - Within which pieces are exchanged





# Outline

- ❑ Multimedia Networking Applications
- ❑ Streaming stored audio and video
- ❑ Scalable Streaming Techniques
- ❑ Content Distribution Networks
- ❑ Beyond Best Effort

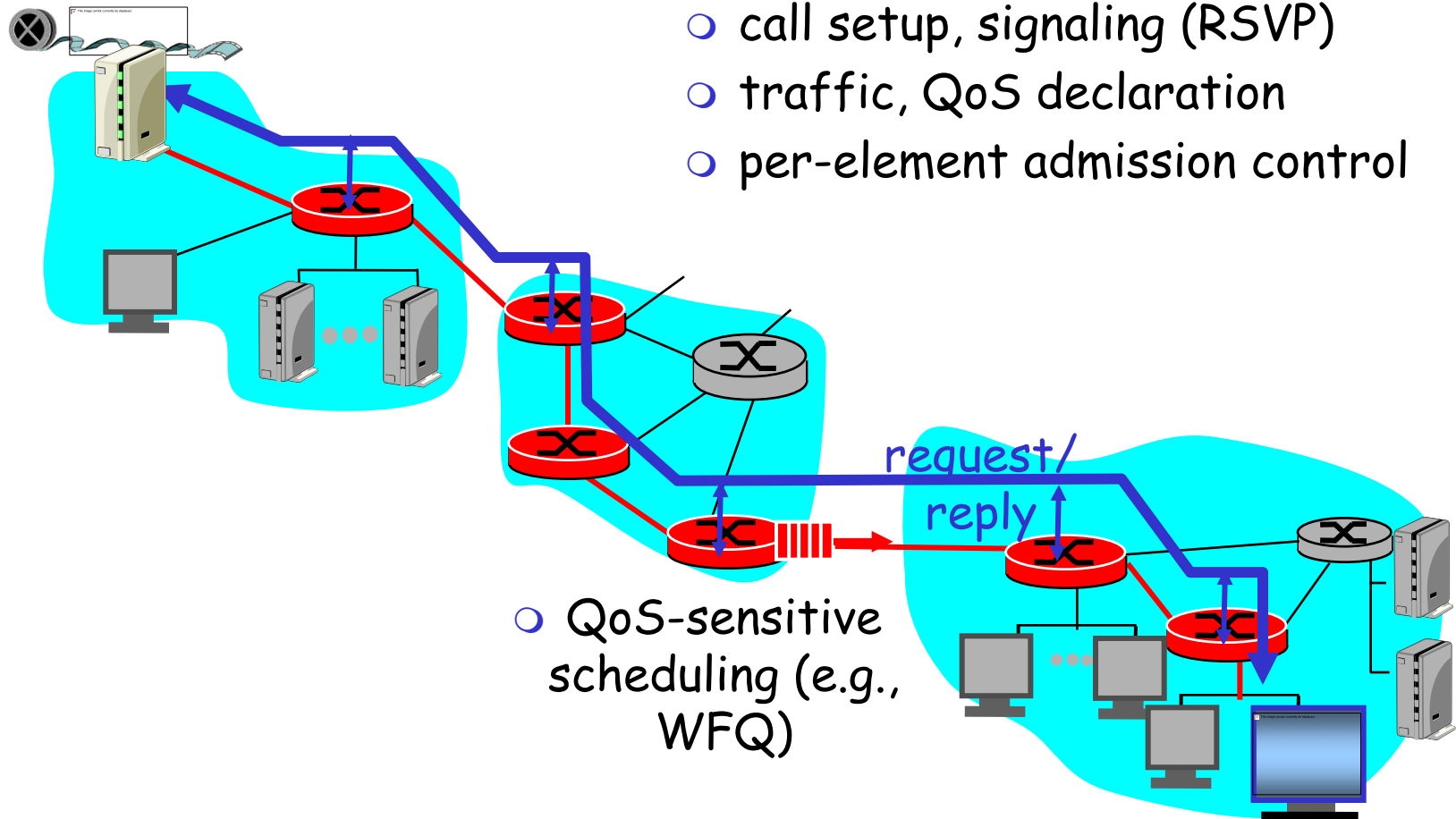
# Integrated Services (IntServ) Architecture

- ❑ architecture for providing QOS guarantees in IP networks for individual flows
- ❑ flow: a distinguishable stream of distinct IP datagrams
  - Unidirectional
  - Multiple recipient
- ❑ Components of this architecture:
  - Admission control
  - Reservation protocol
  - Routing protocol
  - Classifier and route selection
  - Packet scheduler

# Intserv: QoS guarantee scenario

## □ Resource reservation

- call setup, signaling (RSVP)
- traffic, QoS declaration
- per-element admission control



# Call Admission

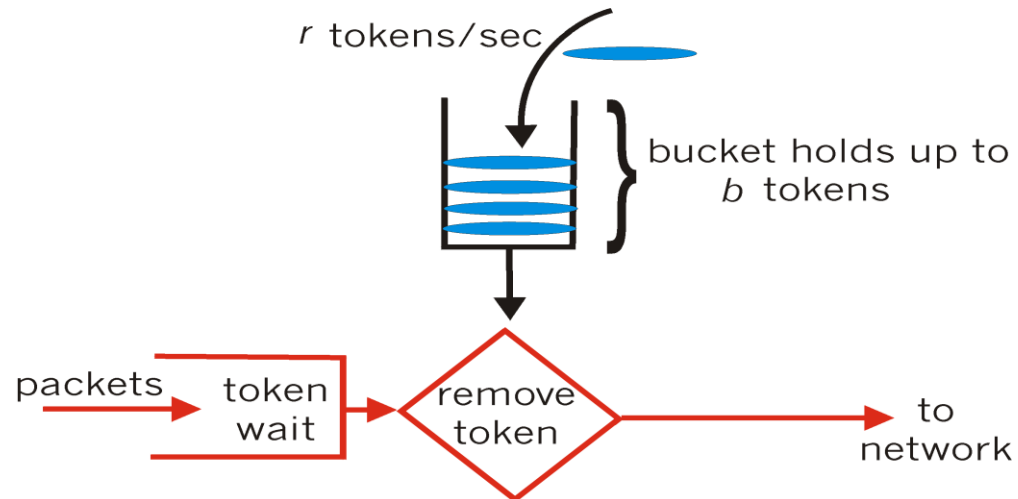
Arriving session must :

- ❑ declare its QoS requirement
  - **R-spec**: defines the QoS being requested
- ❑ characterize traffic it will send into network
  - **T-spec**: defines traffic characteristics
- ❑ signaling protocol: needed to carry R-spec and T-spec to routers (where reservation is required)
  - **RSVP**

Need Scheduling and Policing Policies to provide QoS

# Policing: Token Bucket

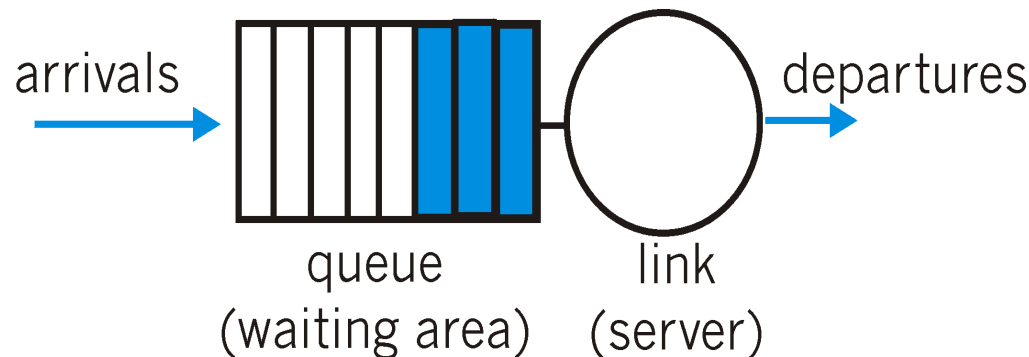
Token Bucket: limit input to specified Burst Size and Average Rate.



- ❑ bucket can hold  $b$  tokens
- ❑ tokens generated at rate  $r$  token/sec unless bucket full
- ❑ *over interval of length  $t$ : number of packets admitted less than or equal to  $(r t + b)$ .*

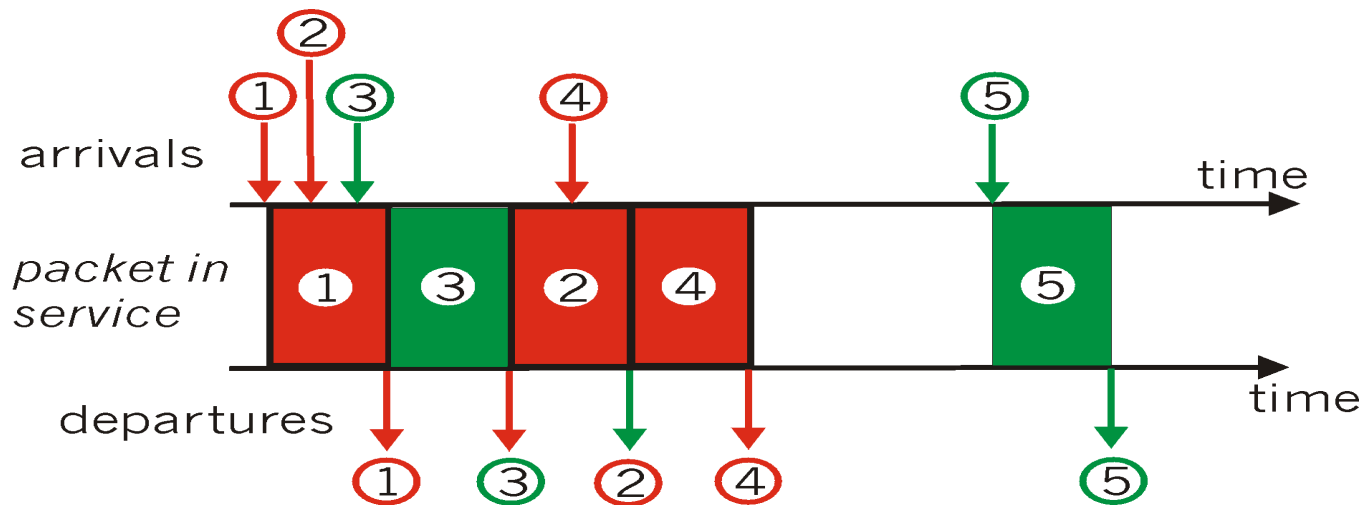
# Link Scheduling

- ❑ **scheduling**: choose next packet to send on link
- ❑ **FIFO (First In First Out) scheduling**: send in order of arrival to queue
  - **discard policy**: if packet arrives to full queue: who to discard?
    - DropTail: drop arriving packet
    - Priority: drop/remove on priority basis
    - Random: drop/remove randomly (e.g., RED)



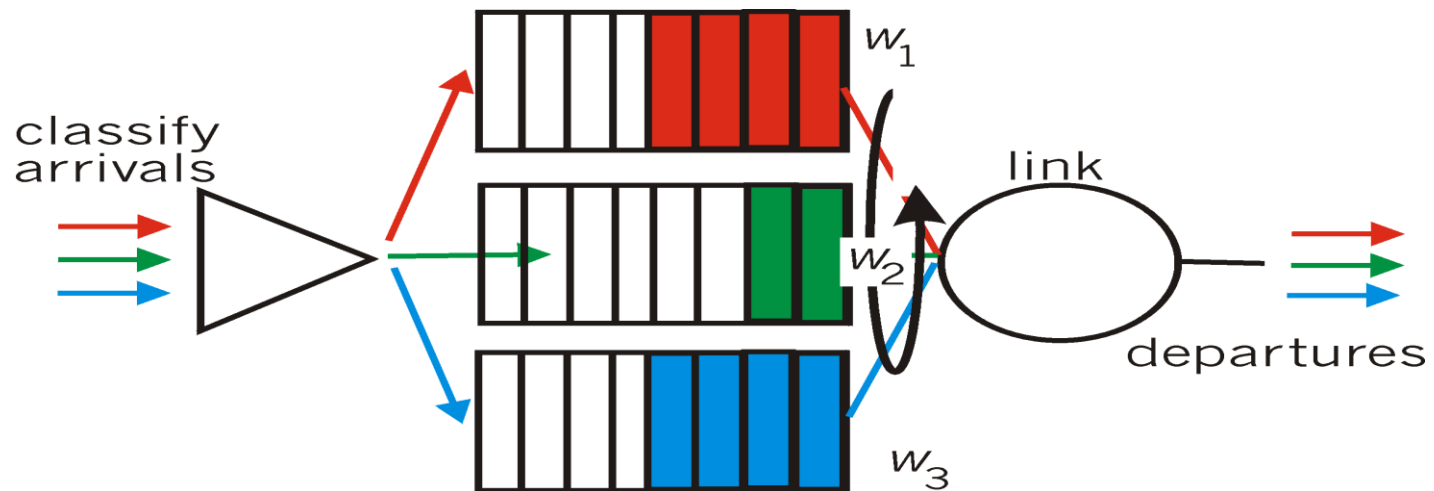
# Round Robin

- multiple classes
- cyclically scan class queues, serving one from each class (if available)



# Weighted Fair Queuing

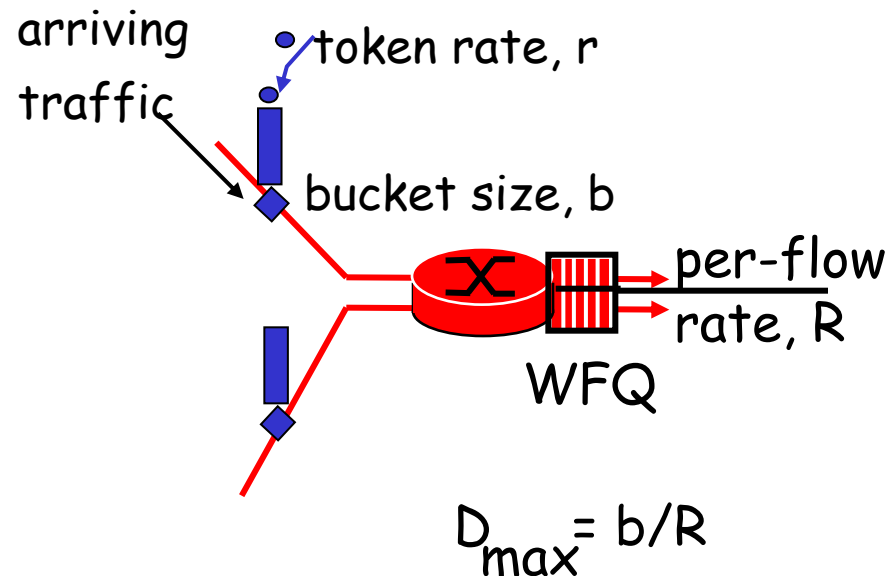
- generalized Round Robin
- each class gets weighted amount of service in each cycle



# IntServ QoS: Service models [rfc2211, rfc 2212]

## Guaranteed service:

- ❑ Assured data rate
- ❑ A specified upper bound on queuing delay



## Controlled load service:

- ❑ "a quality of service closely approximating the QoS that same flow would receive from an unloaded network element."
- ❑ Similar to behavior best effort service in an unloaded network

# Differentiated Services

## Concerns with IntServ:

- ❑ **Scalability:** signaling, maintaining per-flow router state difficult with large number of flows
- ❑ **Flexible Service Models:** Intserv has only two classes. Desire “qualitative” service classes
  - E.g., Courier, xPress, and normal mail
  - E.g., First, business, and cattle class ☺

## DiffServ approach:

- ❑ simple functions in network core, relatively complex functions at edge routers (or hosts)
- ❑ Don't define service classes, just provide functional components to build service classes

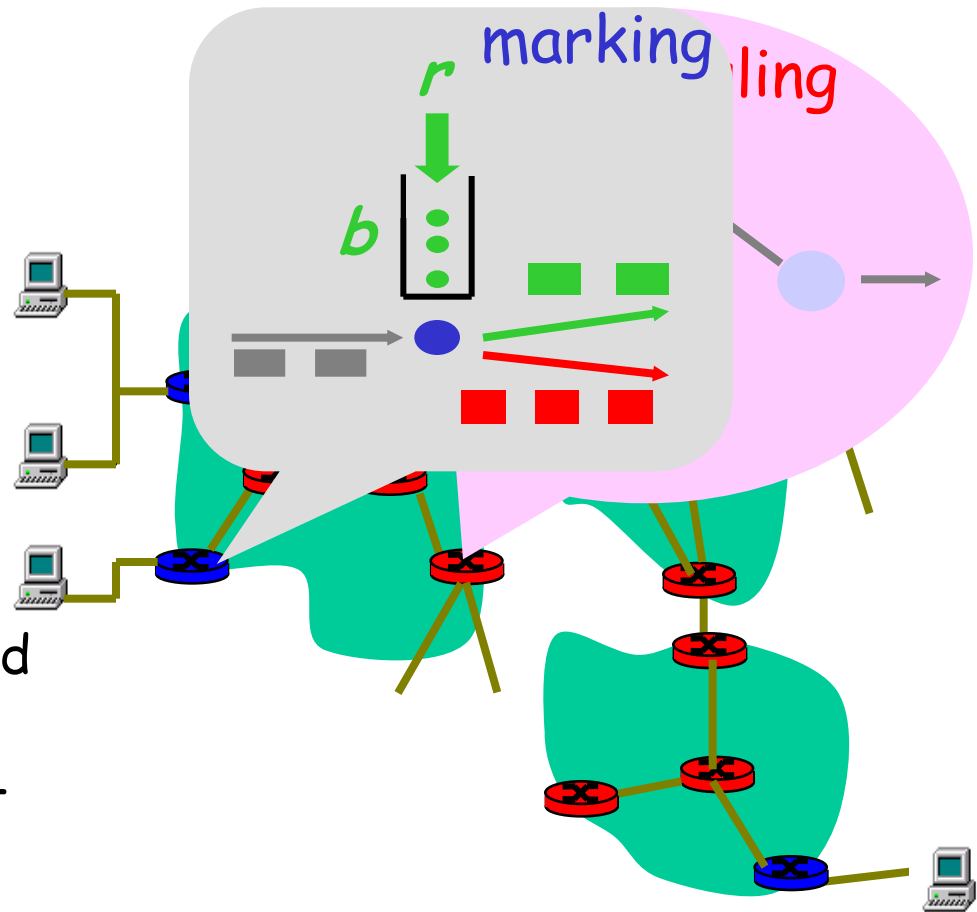
# DiffServ Architecture

## Edge router:

- per-flow traffic management
- Set the DS field; value determines type of service (PHB: Per-Hop Behavior)

## Core router:

- buffering and scheduling based on **marking** at edge
- per-class traffic management



# Traffic Classification/Conditioning

- ❑ How can packet markings be carried in IPv4 datagrams?
- ❑ Sender may agree to conform to a “traffic profile”, thus a leaky bucket policer may be used at the network edge to enforce
  - Peak rate
  - Average rate
  - Burst size
- ❑ What happens when traffic profile is violated?
  - Employ traffic shaping?

# Deployment Issues

- ❑ Single administrative domain
- ❑ Incremental deployment
- ❑ Traffic policing/shaping complexity
- ❑ Charging models

# Signaling in the Internet

connectionless  
(stateless)  
forwarding by IP  
routers      +      best effort  
service      =      no network  
signaling protocols  
in initial IP  
design

- ❑ **New requirement:** reserve resources along end-to-end path (end system, routers) for QoS for multimedia applications
- ❑ **RSVP:** Resource reSerVation Protocol [RFC 2205]
  - “ ... allow users to communicate requirements to network in robust and efficient way.” i.e., signaling !
- ❑ earlier Internet Signaling protocol: ST-II [RFC 1819]

# RSVP Design Goals

1. accommodate **heterogeneous receivers** (different bandwidth along paths)
2. accommodate different applications **with different resource requirements**
3. make **multicast a first class service**, with adaptation to multicast group membership
4. **leverage existing multicast/unicast routing**, with adaptation to changes in underlying unicast, multicast routes
5. **control protocol overhead** to grow (at worst) linear in # receivers
6. **modular design** for heterogeneous underlying technologies

## RSVP: does not...

- ❑ specify how resources are to be reserved
  - ❑ rather: a mechanism for communicating needs
- ❑ determine routes packets will take
  - ❑ that's the job of routing protocols
  - ❑ signaling decoupled from routing
- ❑ interact with forwarding of packets
  - ❑ separation of control (signaling) and data (forwarding) planes

# Multimedia Networking: Summary

- ❑ multimedia applications and requirements
- ❑ making the best of today's "best effort" service
- ❑ scheduling and policing mechanisms
- ❑ next generation Internet: IntServ, RSVP, DiffServ, IPv6, IP-QoS