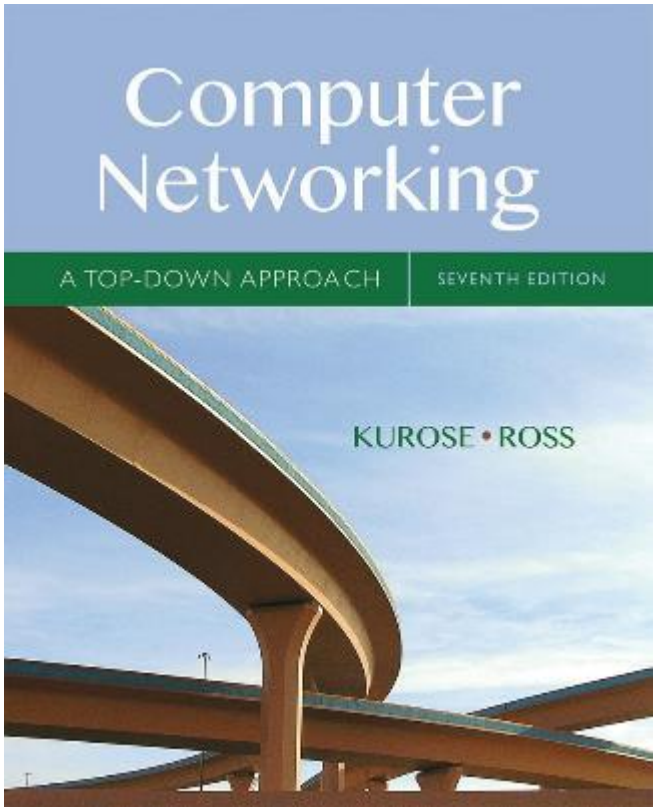
Τεχνολογίες Διαδικτύου 2025-26 (DIT315)

Δρ. Ειρήνη Λιώτου

eliotou@hua.gr

9/12/2025

Chapter 9 Multimedia Networking



Computer Networking: A Top Down Approach

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

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.

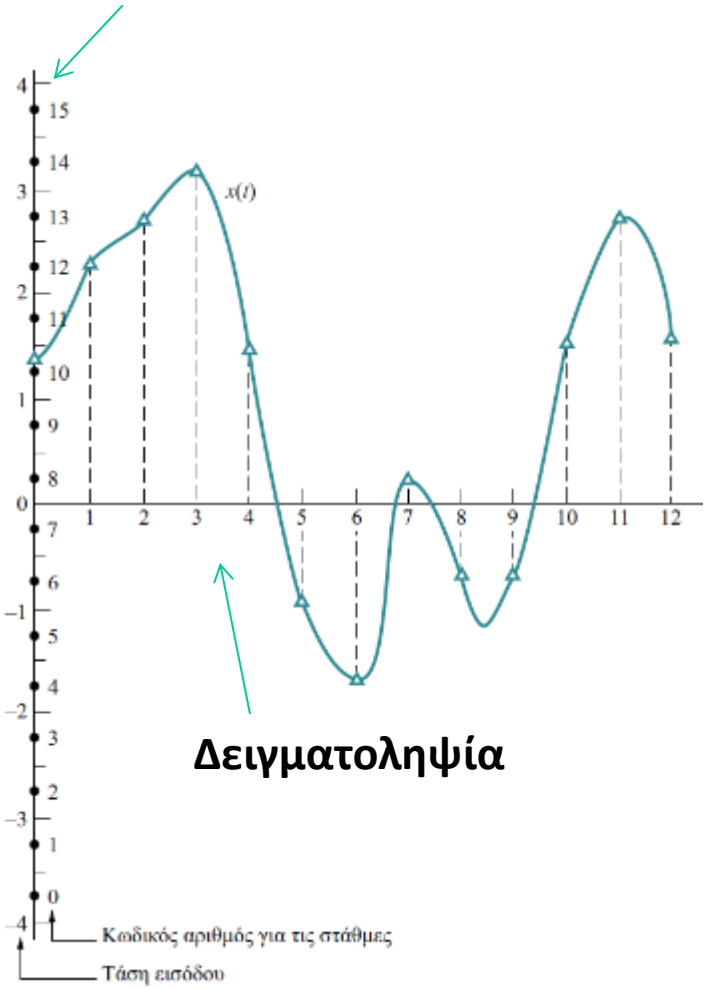
Thanks and enjoy! JFK/KWR

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

Παλμοκωδική διαμόρφωση (PCM)

Κβάντιση



Δειγματοληψία

Κωδικοποίηση

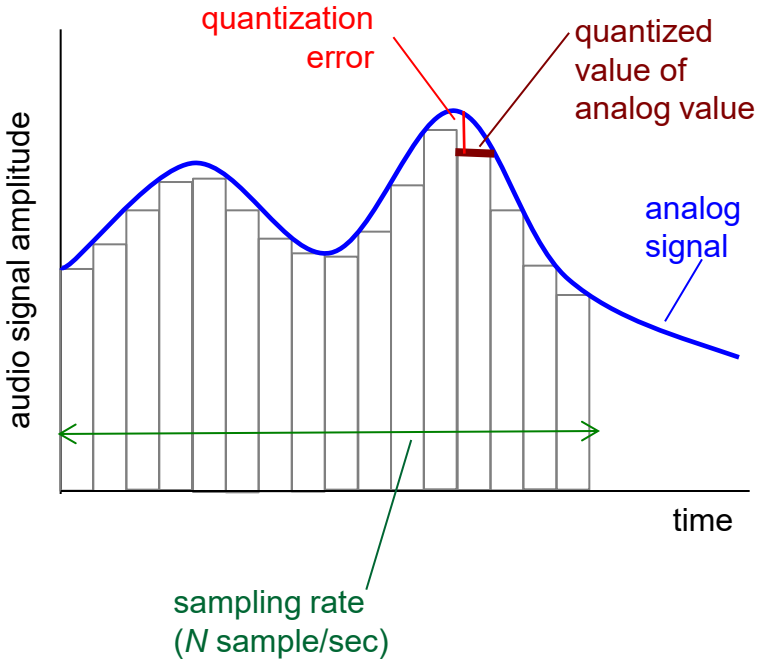
How to improve quality then?

Αριθμός δείγματος	$x_s(t)$	$x_q(t)$	Αριθμός στάθμης	Δυαδική τιμή αριθμού στάθμης
0	1,3	1,25	10	1010
1	2,3	2,25	12	1100
2	2,7	2,75	13	1101
3	3,2	3,25	14	1110
4	1,45	1,25	10	1010
5	– 0,9	– 0,75	6	0110
6	– 1,7	– 1,75	4	0100
7	0,3	0,25	8	1000
8	0,7	0,75	9	1001
9	0,7	0,75	9	1001
10	1,6	1,75	11	1011
11	2,8	2,75	13	1101
12	1,7	1,75	11	1011

Source: ΕΑΠ-ΠΛΗ22

Multimedia: audio

- analog audio signal sampled at constant rate
  - telephone: 8,000 samples/sec
  - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
  - e.g., 2^8=256 possible quantized values
  - each quantized value represented by bits, e.g., 8 bits for 256 values

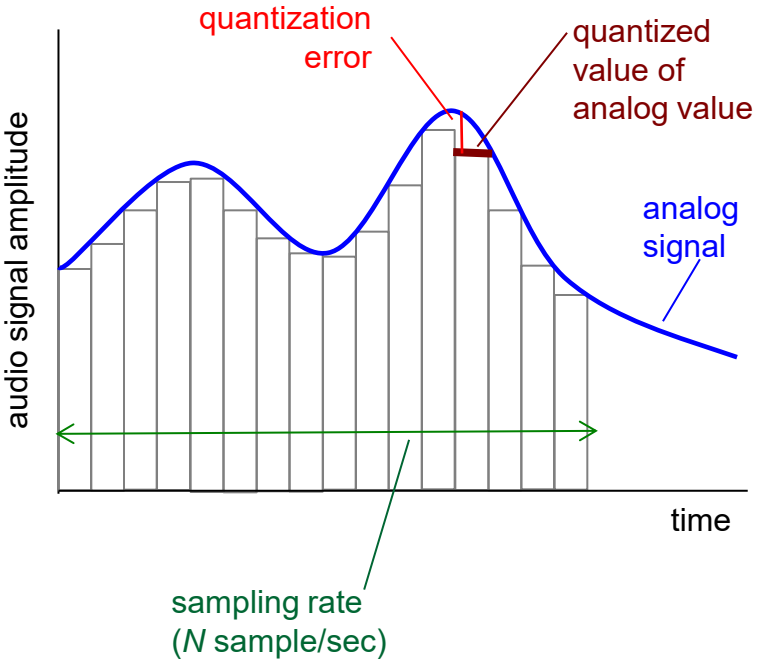


Multimedia: audio

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

example rates

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



Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

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

frame *i*

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

frame *i+1*

Multimedia: video

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes
- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

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

frame *i*

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

frame *i+1*

Multimedia networking: 3 application types

- *streaming, stored* audio, video
  - *streaming:* can begin playout before downloading entire file
  - *stored (at server):* can transmit faster than audio/video will be rendered (implies storing/buffering at client)
  - e.g., YouTube, Netflix, Hulu
- *conversational* voice/video over IP
  - interactive nature of human-to-human conversation limits delay tolerance
  - e.g., Skype
- *streaming live* audio, video
  - e.g., live sporting event (football)

Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

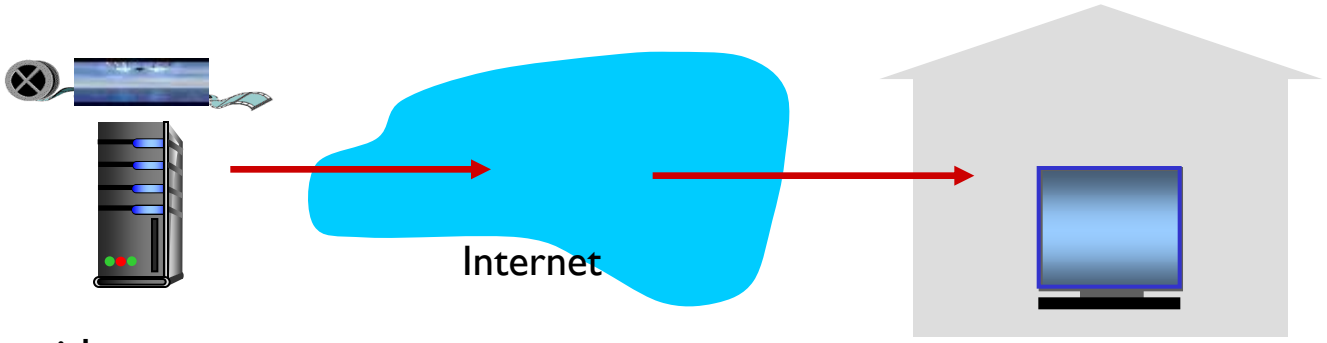
9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia

Streaming stored video:

UDP streaming,
HTTP streaming,
adaptive HTTP streaming

simple scenario:



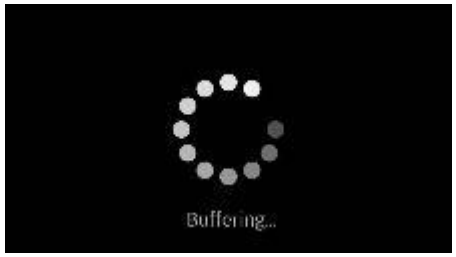
Characteristics

- Streaming
- Interactivity
- Continuous playout

The most important performance measure for streaming video is average end-to-end throughput

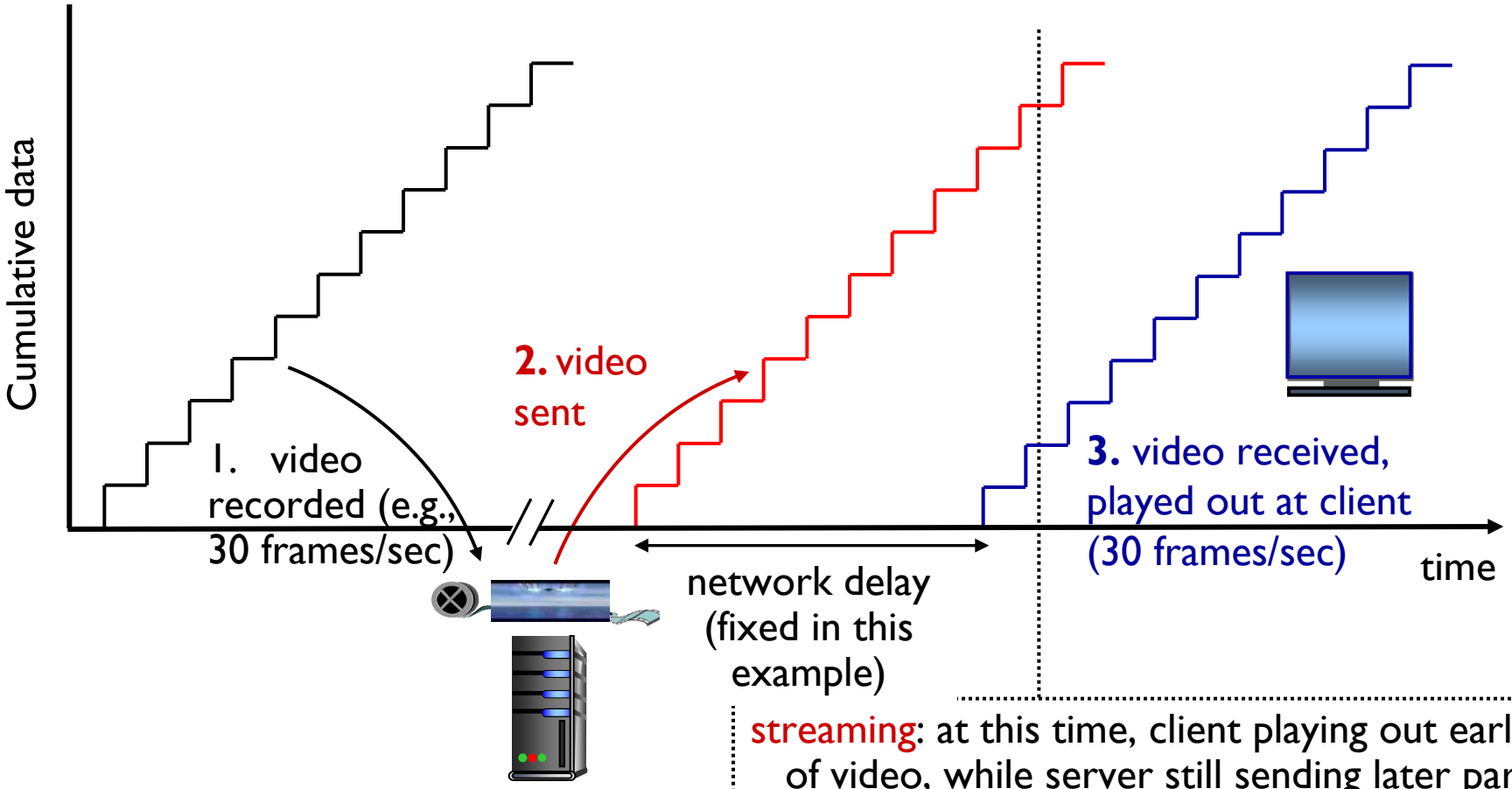
Streaming stored video: challenges

- continuous playout constraint: during client video playout, playout timing must match original timing
  - … but network delays are variable (jitter), so will need client-side buffer to match continuous playout constraint



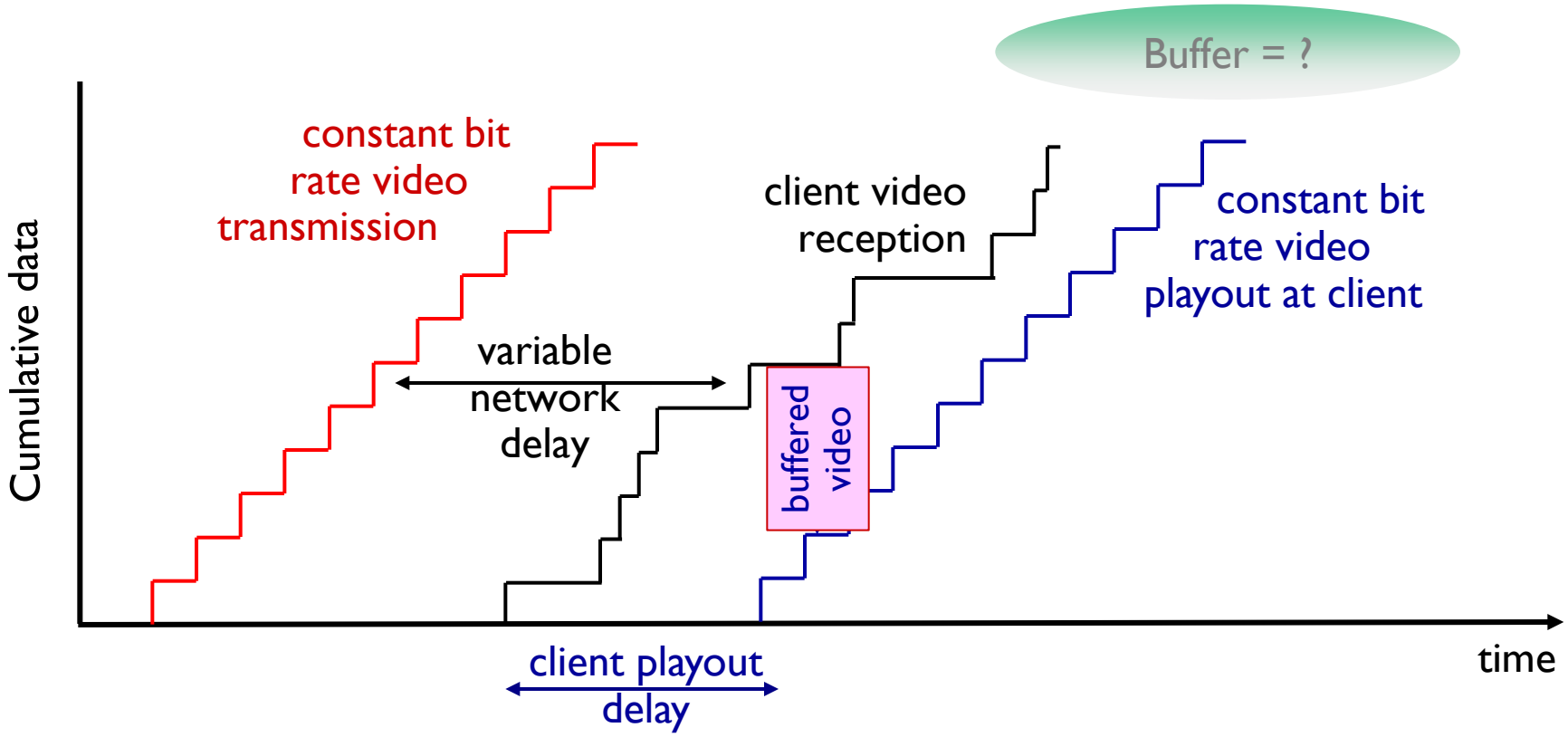
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through video
  - video packets may be lost, retransmitted

Streaming stored video



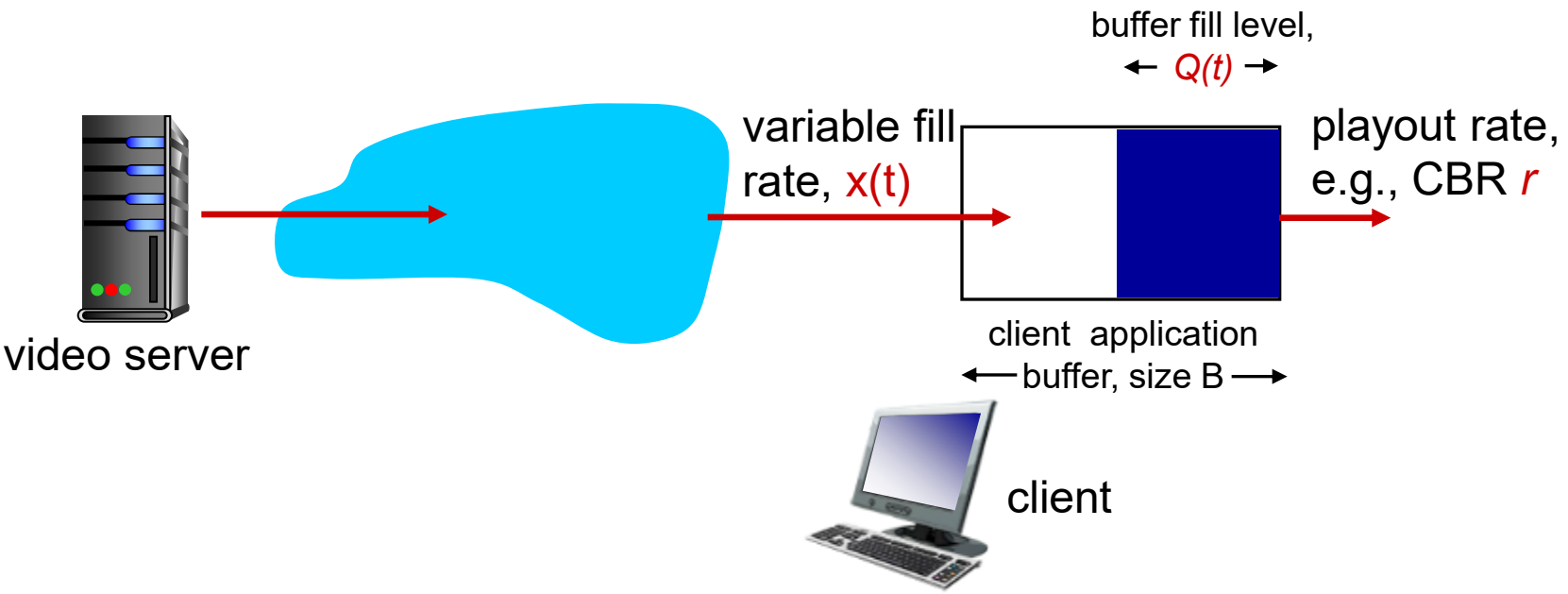
streaming: at this time, client playing out early part of video, while server still sending later part of video

Streaming stored video: playout buffering

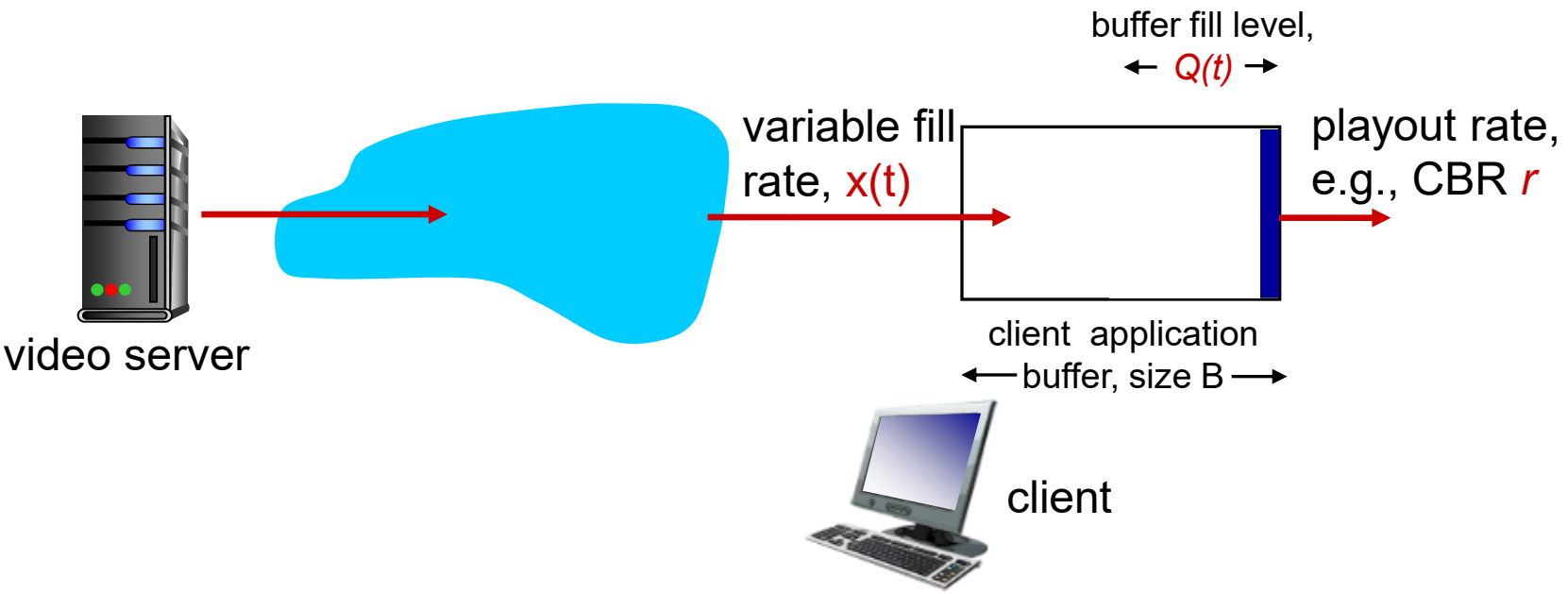


- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout

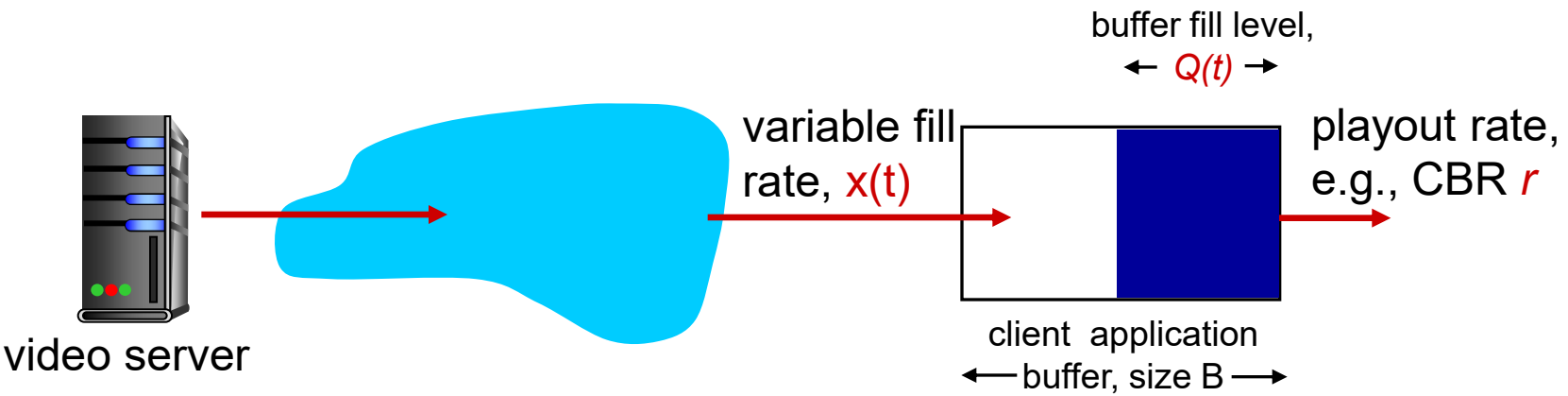


Client-side buffering, playout



1. Initial fill of buffer until playout begins at t_p
2. Playout begins at t_p,
3. Buffer fill level varies over time as fill rate x(t) varies and playout rate r is constant

Client-side buffering, playout

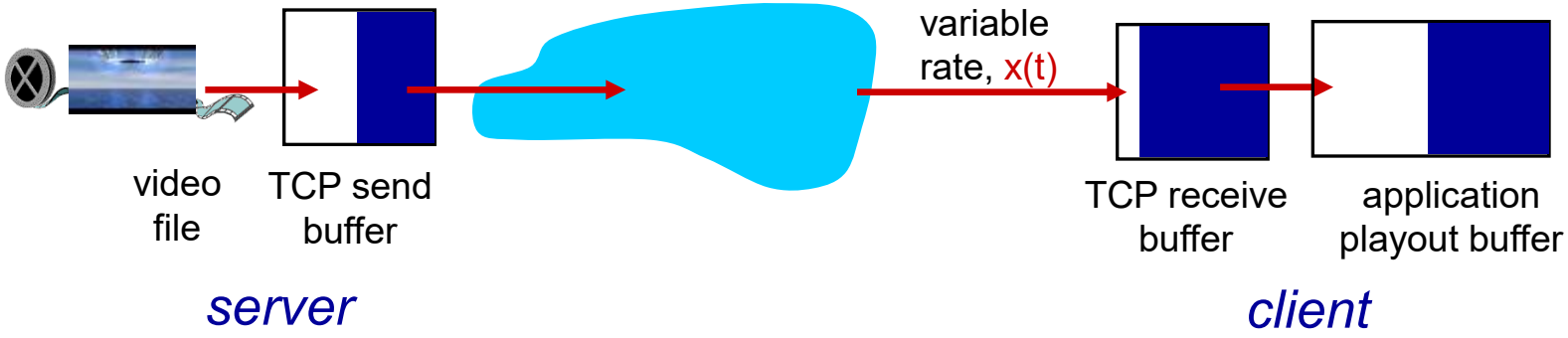


playout buffering: average fill rate ($\bar{x}$), playout rate (r):

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

Streaming multimedia: 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)
- Reposition / pause easier
- HTTP/TCP passes more easily through firewalls

Streaming multimedia: UDP

- server sends at rate appropriate for client
  - often: send rate = encoding rate = constant rate
  - transmission rate can be oblivious to congestion levels
- UDP may *not* go through firewalls
- UDP streaming can fail to provide continuous playout
- Separate control connection needed over which the client sends commands regarding session state changes (such as pause, resume, reposition, etc.)

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
- *value-added services:* call forwarding, screening, recording, teleconference

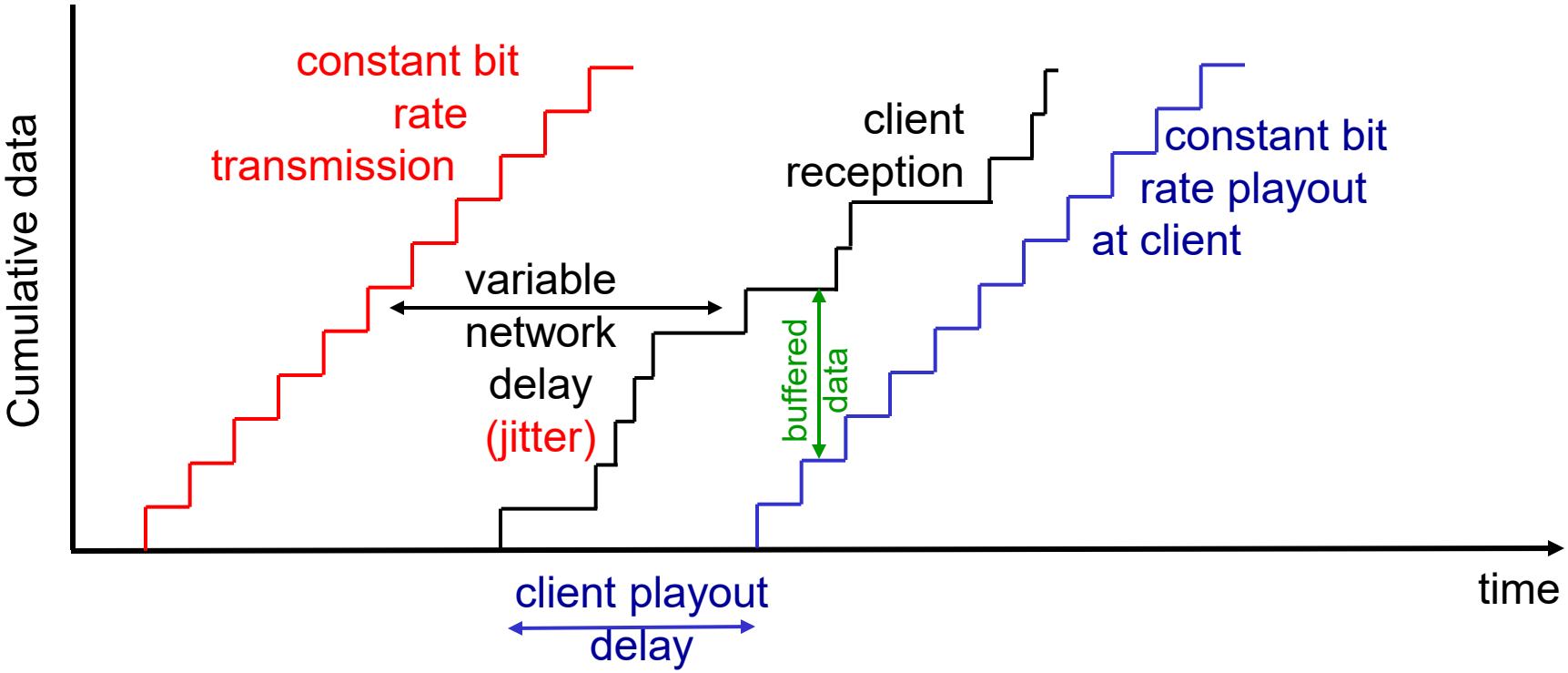
VoIP characteristics

- 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
- *End-to-end delay*

Delay jitter



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

VoIP: removing Jitter

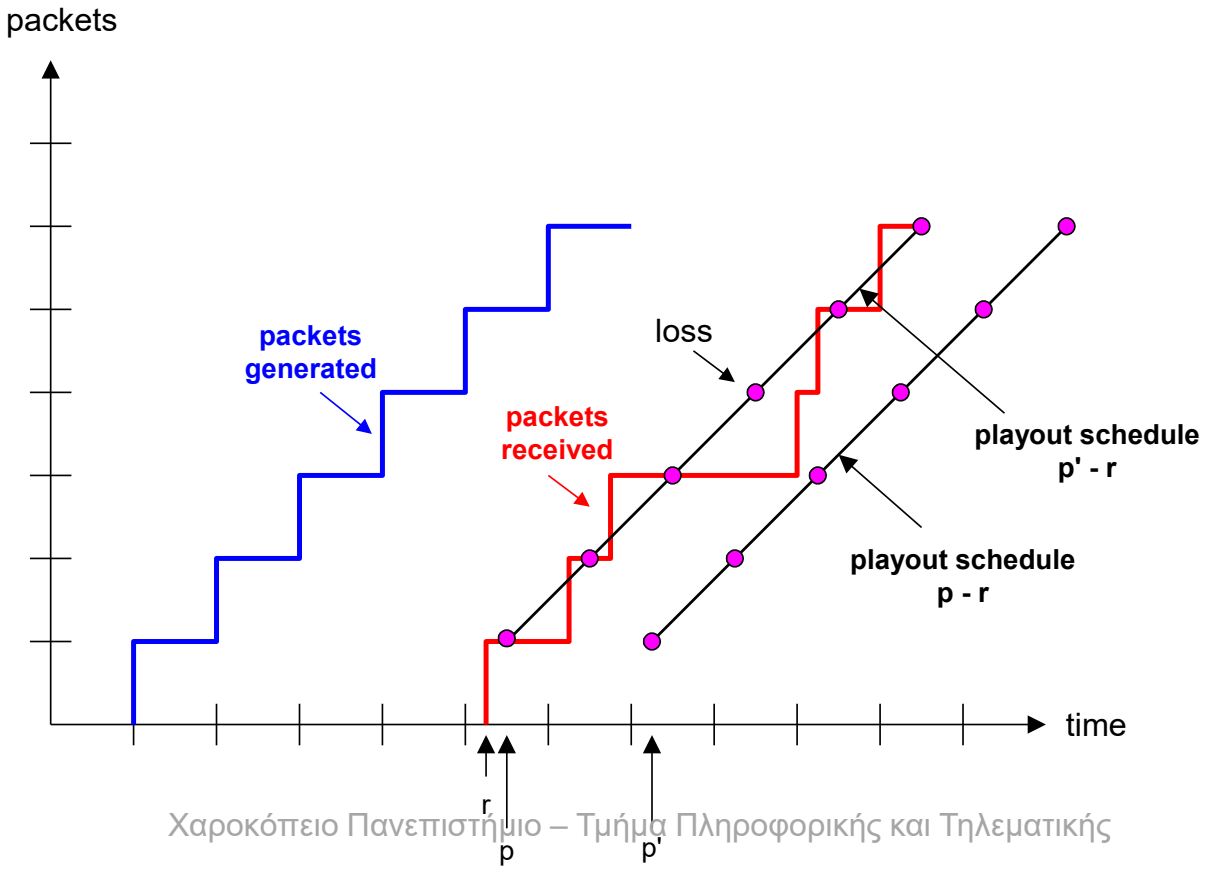
- Prepending each chunk with a timestamp.
  - The sender stamps each chunk with the time at which the chunk was generated.
- Delaying playout of chunks at the receiver.
  - The playout delay of the received audio chunks must be long enough so that most of the packets are received before their scheduled playout times.
  - This playout delay can either be fixed throughout the duration of the audio session or vary adaptively during the audio session lifetime.

VoIP: fixed playout delay

- receiver attempts to playout each chunk exactly *q* msecs after chunk was generated.
  - chunk has time stamp *t*: play out chunk at *t*+*q*
  - chunk arrives after *t*+*q*: data arrives too late for playout: data "lost"
- tradeoff in choosing *q*:
  - *large q:* less packet loss
  - *small q:* better interactive experience

VoIP: fixed playout delay

- sender generates packets every 20 msec during talk spurt.
- first packet received at time r
- first playout schedule: begins at p
- second playout schedule: begins at p’



Adaptive playout delay (1)

- *goal:* low playout delay, low loss rate
- *approach:* adaptive playout delay adjustment:
  - estimate network delay, adjust playout delay at beginning of each talk spurt
  - chunks still played out every 20 msec during talk spurt
- adaptively estimate packet delay: (EWMA - exponentially weighted moving average):

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

delay estimate after ith packet

small constant, e.g. 0.1

time received - time sent (timestamp)

measured delay of ith packet

Adaptive playout delay (2)

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

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

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

- for first packet in talk spurt, playout time is:

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

- remaining packets in talkspurt are played out periodically

Adaptive playout delay (3)

Q: How does receiver determine whether packet is first in a talkspurt?

- if no loss, receiver looks at successive timestamps
  - difference of successive stamps > 20 msec -->talk spurt begins.
- with loss possible, receiver must look at both time stamps and sequence numbers
  - difference of successive stamps > 20 msec *and* sequence numbers without gaps --> talk spurt begins.

Example (no loss)

Packet	Timestamp
100	0 ms
101	20 ms
102	40 ms
103	**200 ms** (big jump!)

Jump of **160 ms** → silence → *start of new talkspurt.*

Example (with loss handled correctly)

Packet	Seq #	Timestamp	Explanation
100	1000	0	
101	1001	20	
102	1002	**200**	Timestamp jump
			Sequence numbers continuous → **No loss** → **Start of talkspurt**

Example (loss scenario — *not* a talkspurt)

Packet	Seq #	Timestamp
105	1000	0
106	1010	200

Here:

- Timestamp jump > 20 ms ✔️
- **BUT sequence number gap exists** ❌

So this **is not** a new talkspurt — it was packet loss.

VoiP: recovery from packet loss (1)

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

- each ACK/NAK takes ~ one RTT
- alternative: *Forward Error Correction (FEC)*
  - send enough bits to allow recovery without retransmission

simple FEC

- for every group of *n* chunks, create redundant chunk by exclusive OR-ing *n* original chunks
- send *n+1* chunks, increasing bandwidth by factor *1/n*
- can reconstruct original *n* chunks if at most one lost chunk from *n+1* chunks, with playout delay

FEC example

Imagine tiny packets represented as 4-bit values:

```ini
P1 = 1100
P2 = 1010
```

Sender computes the XOR:

```yaml
FEC = P1 ⊕ P2
    = 1100 XOR 1010
    = 0110
```

Sender sends:

```scss
1100 (P1)
1010 (P2)
0110 (FEC)
```

Suppose P2 is lost.

Receiver has:

- P1 = 1100
- FEC = 0110

Recover P2:

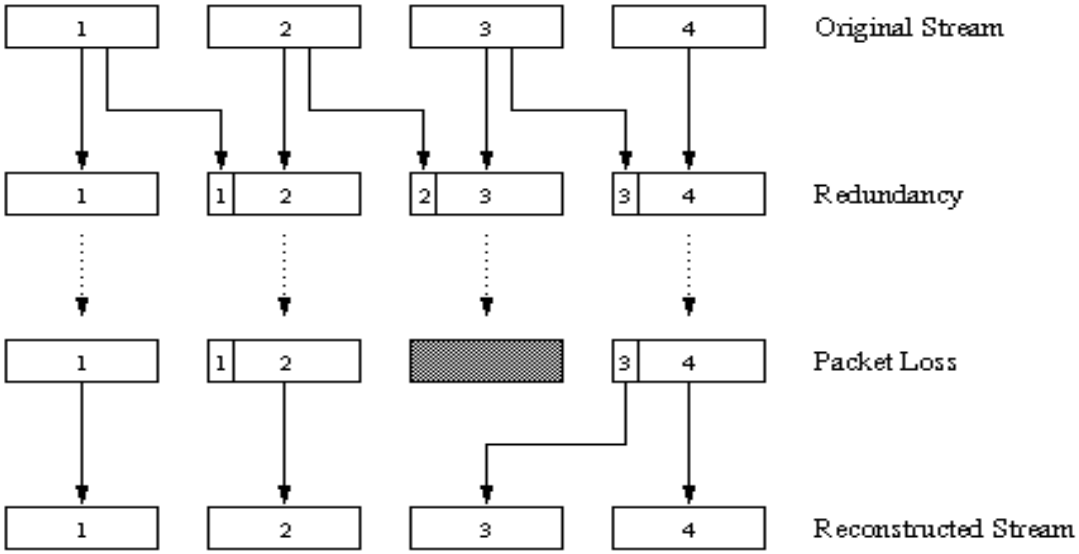
```yaml
P2 = FEC ⊕ P1
   = 0110 XOR 1100
   = 1010
```

Recovered exactly.

VoiP: recovery from packet loss (2)

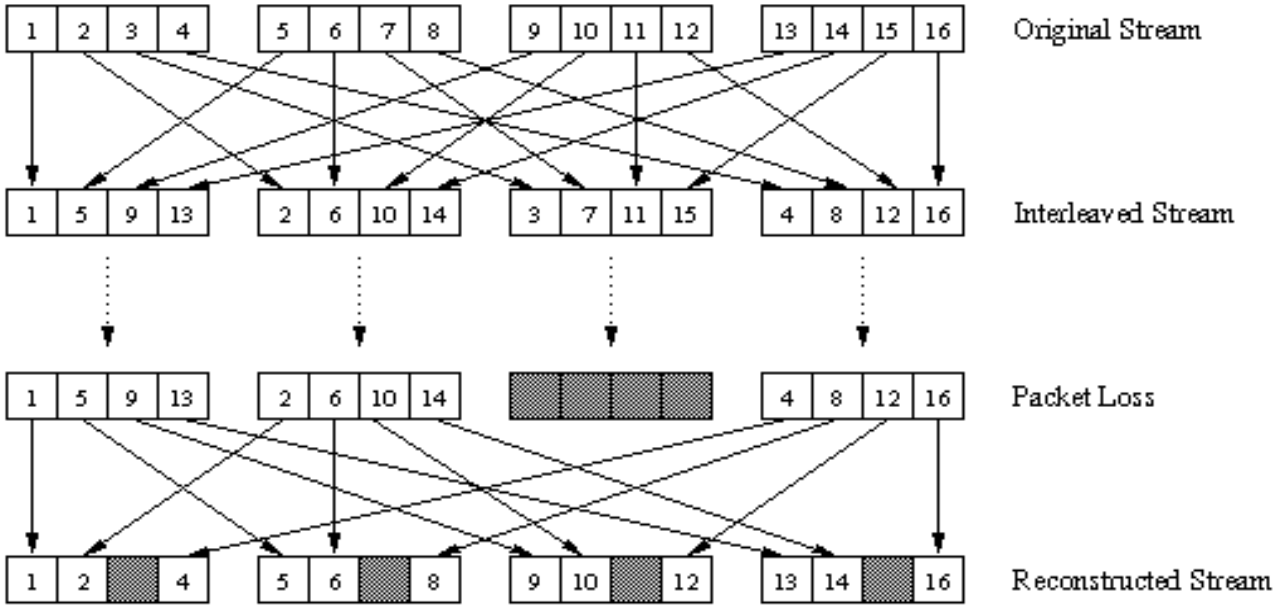
another FEC scheme:

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



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

VoiP: recovery from packet loss (3)



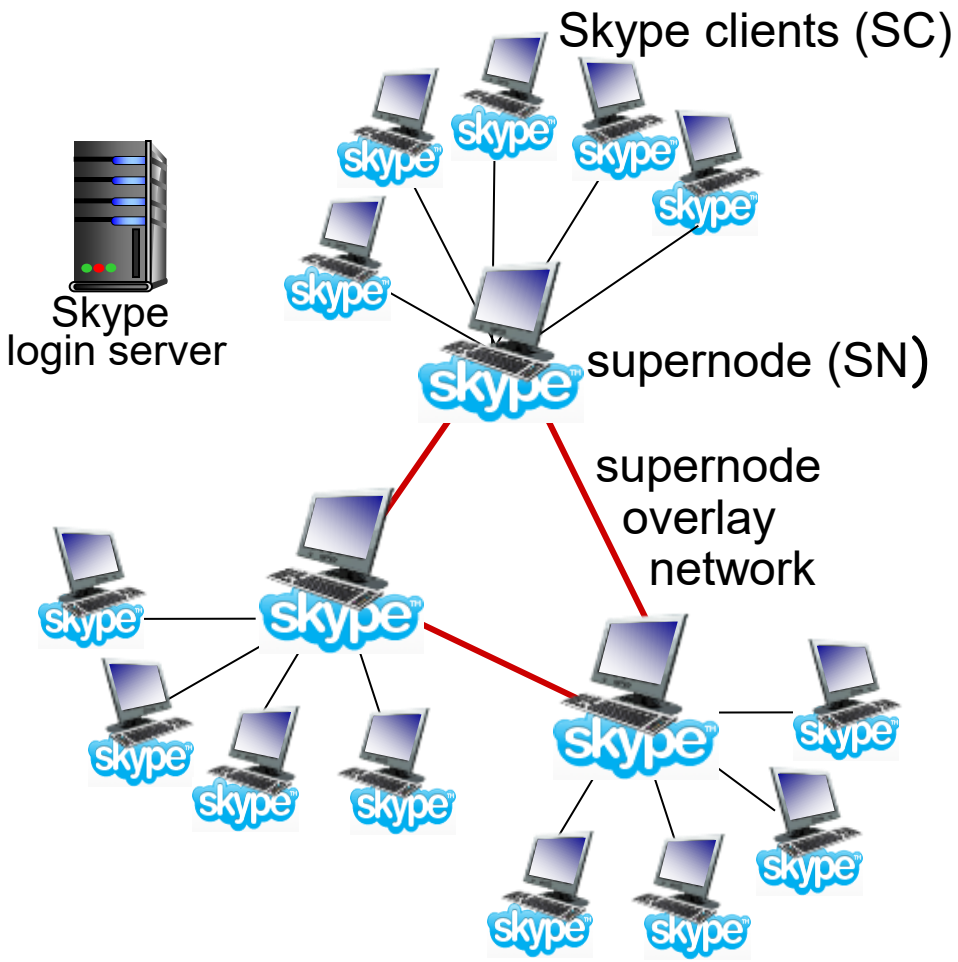
interleaving to conceal loss:

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

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

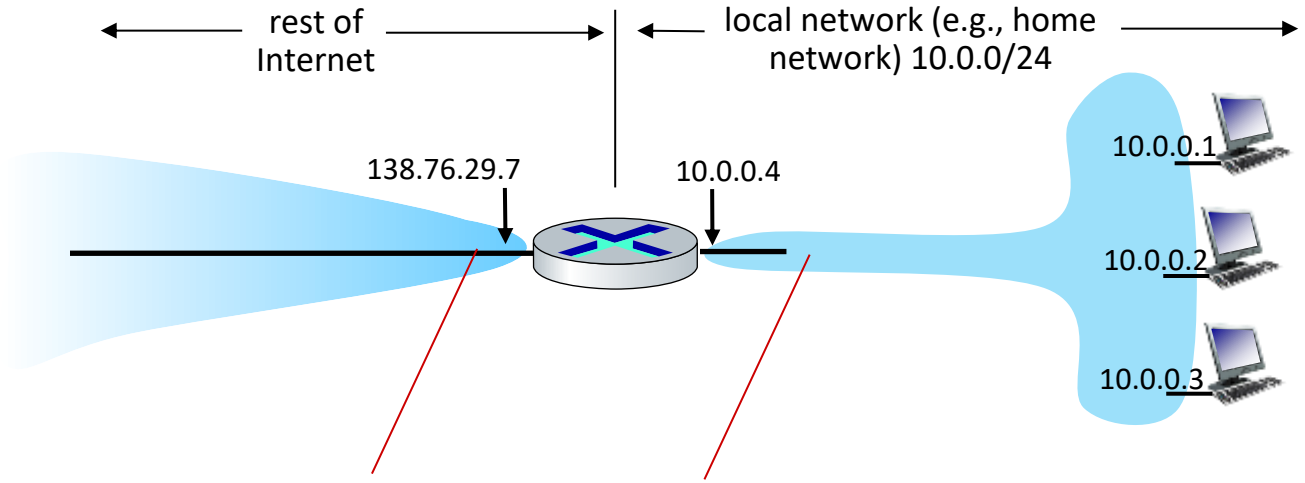
Voice-over-IP: Skype

- 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



NAT: network address translation

NAT: all devices in local network share just one IPv4 address as far as outside world is concerned

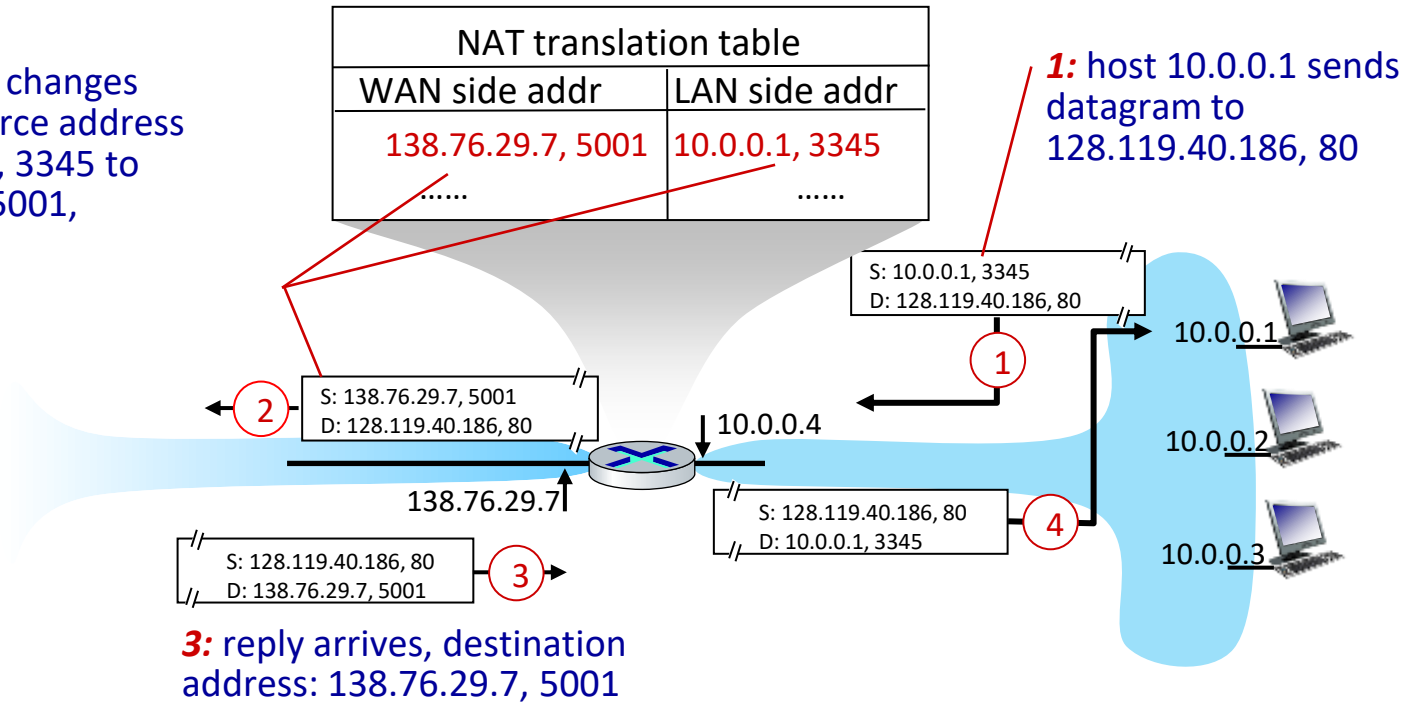


all datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: network address translation

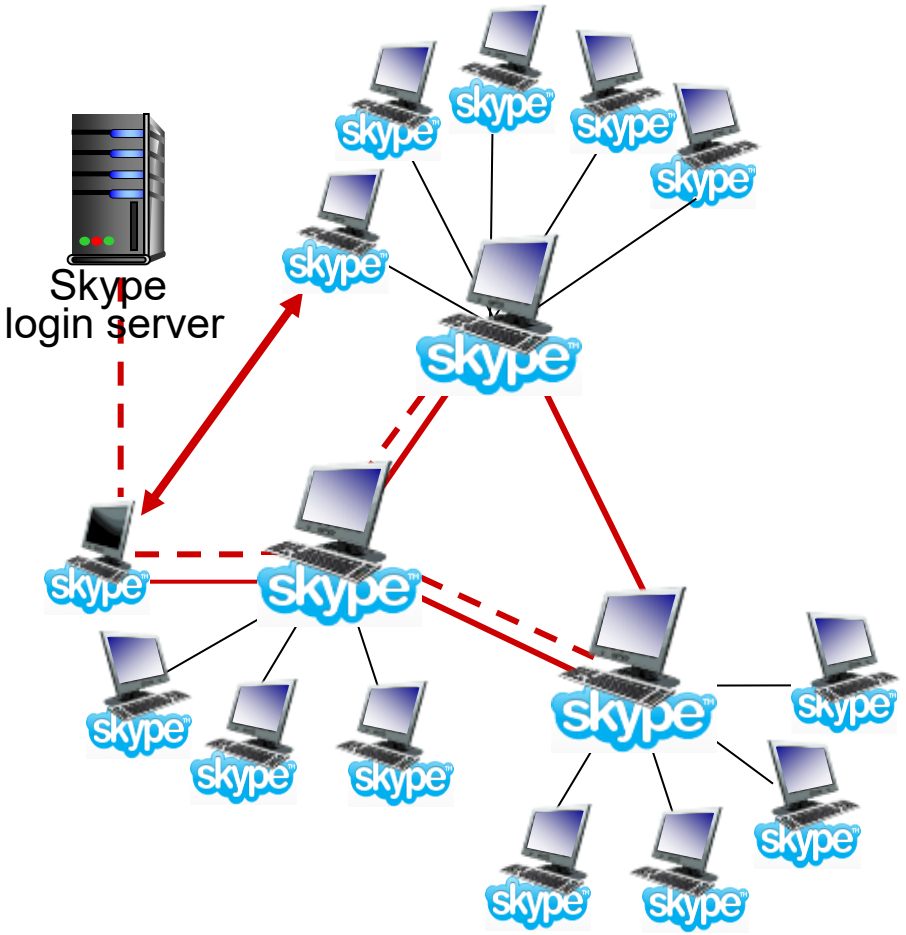
2: NAT router changes datagram source address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table



P2P voice-over-IP: Skype

Skype client operation:

1. joins Skype network by contacting SN using TCP
2. logs-in (username, password) to centralized Skype login server
3. obtains IP address for callee from SN
4. initiate call directly to callee



Skype: peers as relays

- *problem:* both Alice, Bob are behind "NATs"
  - NAT prevents outside peer from initiating connection to insider peer
  - inside peer *can* initiate connection to outside
- relay solution: Alice, Bob maintain open connection to their SNs
  - Alice signals her SN to connect to Bob
  - Alice's SN connects to Bob's SN
  - Bob's SN connects to Bob over open connection Bob initially initiated to his SN