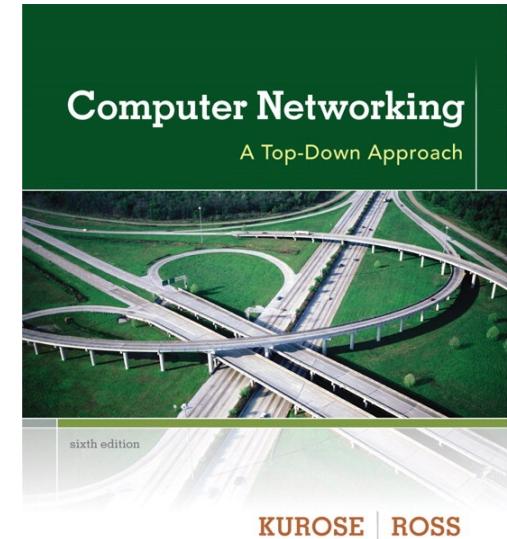


Chapter 7 Multimedia Networking & QoS support over the Internet

*Reti di Calcolatori
Prof.ssa Chiara Petrioli
a.a. 2023/2024*

*We thank for the support material Prof. Kurose-Ross
All material copyright 1996-2012
J.F Kurose and K.W. Ross, All Rights Reserved*



*Computer
Networking: A Top
Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012*

©

Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming stored video

7.3 voice-over-IP

7.4 protocols for *real-time* conversational
applications

7.5 network support for multimedia

Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming stored video

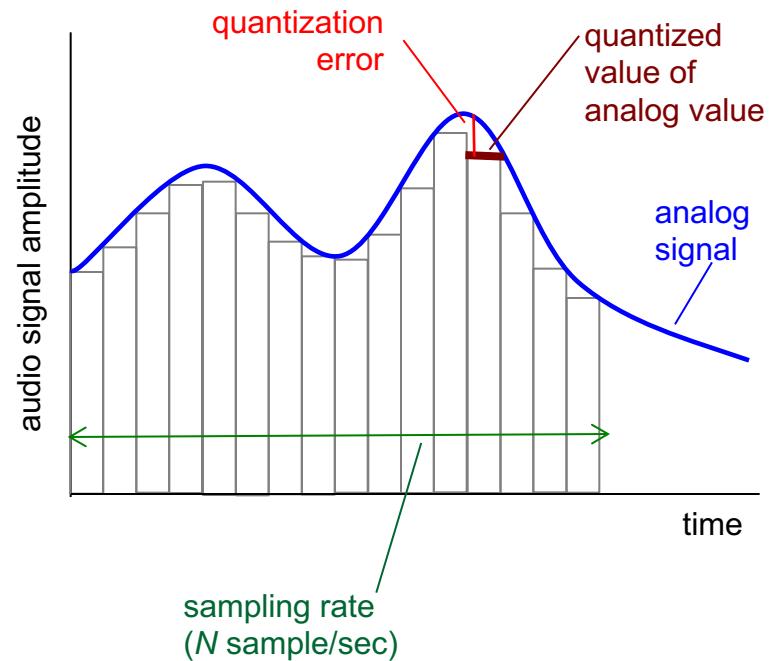
7.3 voice-over-IP

7.4 protocols for *real-time* conversational
applications

7.5 network support for multimedia

Multimedia: audio

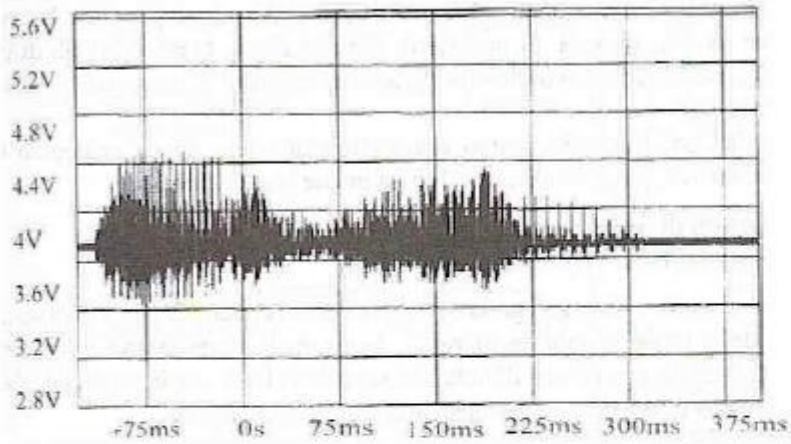
- ❖ analog audio signal sampled at constant rate
 - telephone: 8,000 samples/sec
 - CD music: 44,100 samples/sec
- ❖ each sample quantized, i.e., rounded
 - each quantized value represented by bits, e.g., 8 bits for 256 values
 - e.g., $2^8=256$ possible quantized values



Codifica della voce: Caratteristiche tempo/frequenza

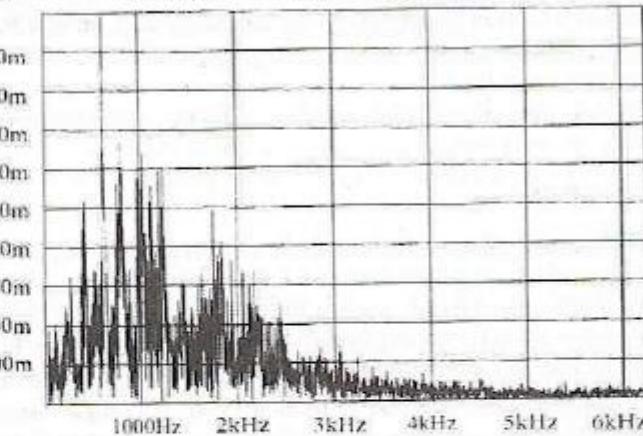
Parola: effe

Data file : EFFE.SCT Number : 16384
date : 26.02.96 16:49:40 Y-Step : 50 mV
Y-Axis : 12.75V X-Step : 40 ms
X-Axis : 655.32. ms From : 55 to 13326



Andamento del segnale nel tempo

Data file : EFFE1.FFT Number : 2048
date : 26.02.96 16:49:40 Y-Step : 3.906 m
Y-Axis : 996.094 m X-Step : 3.052 Hz
X-Axis : 6.247 kHz From : 0 to 2047



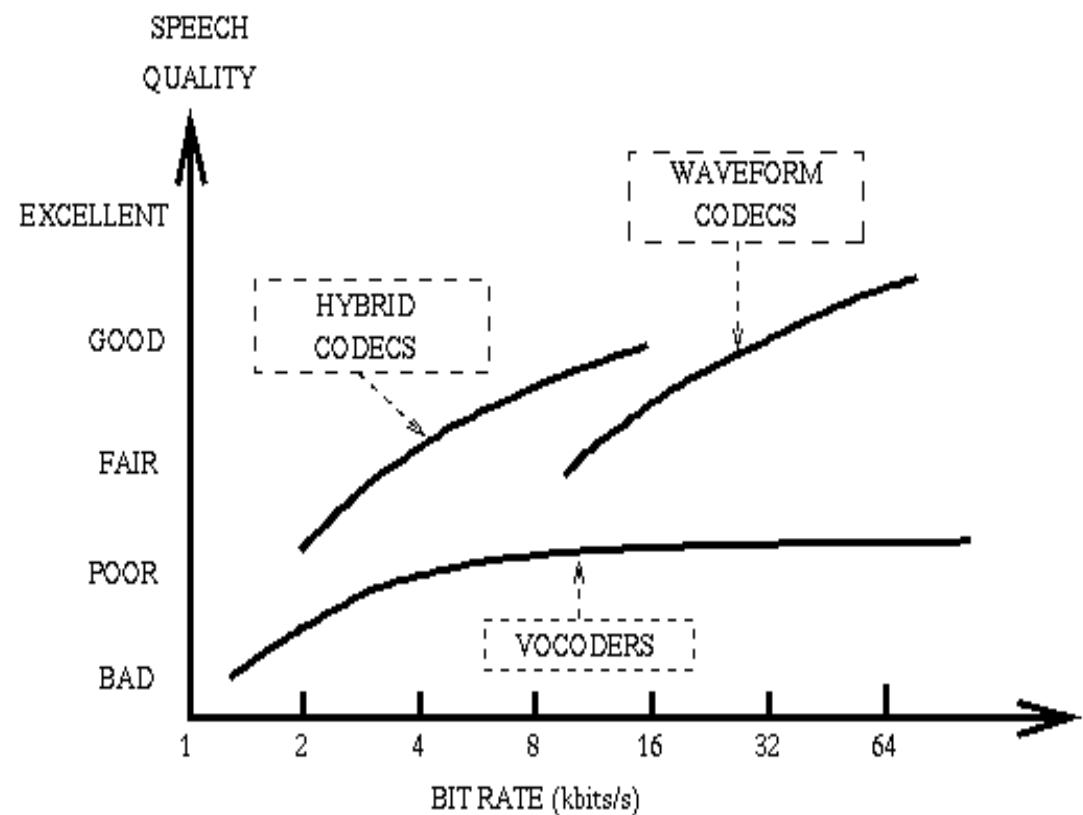
Spettro del segnale (diverse componenti in frequenza)

*Si nota come consonanti e vocali abbiano ampiezze diverse e come le componenti di frequenza piu' significative si trovino tra i 300Hz e i 3400Hz
Con componenti spettrali (piccole) fino a 6 KHz*

Codificatori vocali

Trasformano la voce in un flusso di bit

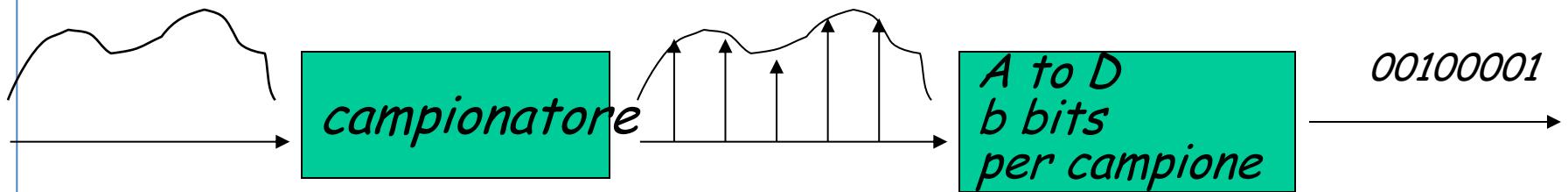
Digitalizzazione di un
Segnale analogico ←
Realizzano una descrizione
esplicita
della forma d'onda in
ingresso (es. PCM)



- ❖ Waveform codecs
- ❖ Source codecs (vocoders)
- ❖ Hybrid codecs

Waveform codecs

- ❖ nessuna conoscenza a priori di come il segnale sia stato generato
- ❖ informazione necessaria:
 - banda del segnale B (telefonia classica < 4 KHz)
 - massimo rumore di quantizzazione tollerabile



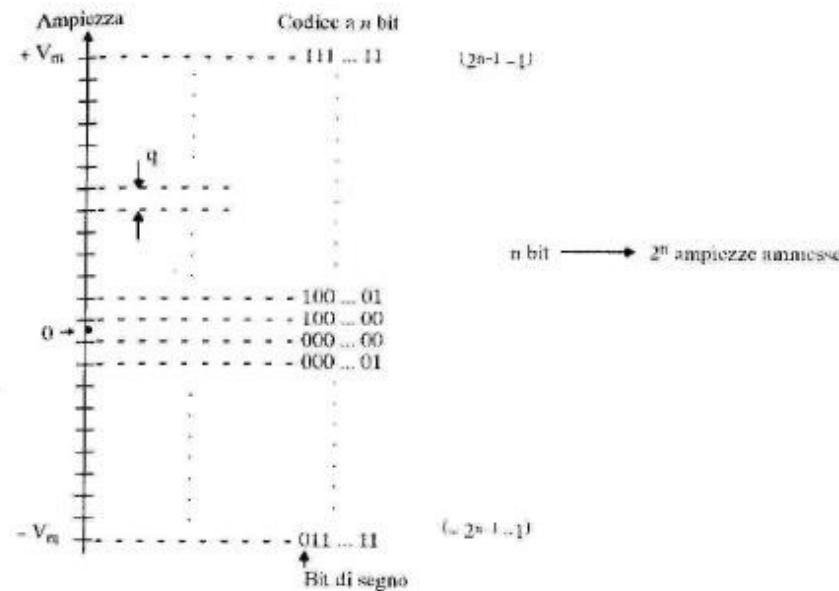
alta qualita', bassa complessita', basso ritardo (1 campione), robustezza agli errori e al rumore di fondo

Codifica della voce: Pulse Code Modulation (PCM)

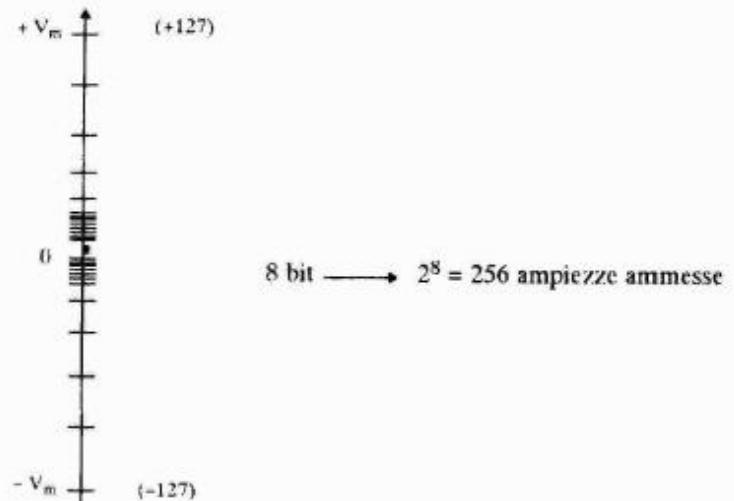
- ❖ standardizzato da ITU nel 1960: G.711
- ❖ si assume $B=4$ kHz, e la frequenza di campionamento $B_c=8$ kHz, 8 bit/campione, 64 kb/s
- ❖ due differenti regole di quantizzazione (logaritmica)
 - per America (μ -law) e
 - Europa (A-law)
 - regole di conversione standard

Tipi di quantizzazione

Quantizzazione uniforme



Quantizzazione non uniforme (o non lineare)



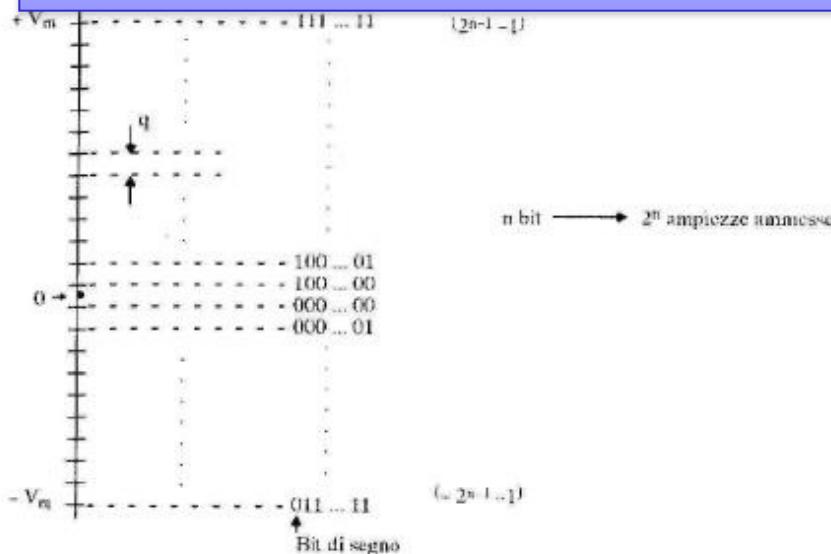
- L'errore di quantizzazione e' fisso ($< q/2$, con q passo di quantizzazione)
- Servono 12 bit per campione per riuscire a ottenere un errore di quantizzazione suff. Basso nel caso di valori piccoli

- Valori grandi possono sopportare errori maggiori
- Sono sufficienti 8 bit per campione in questo caso

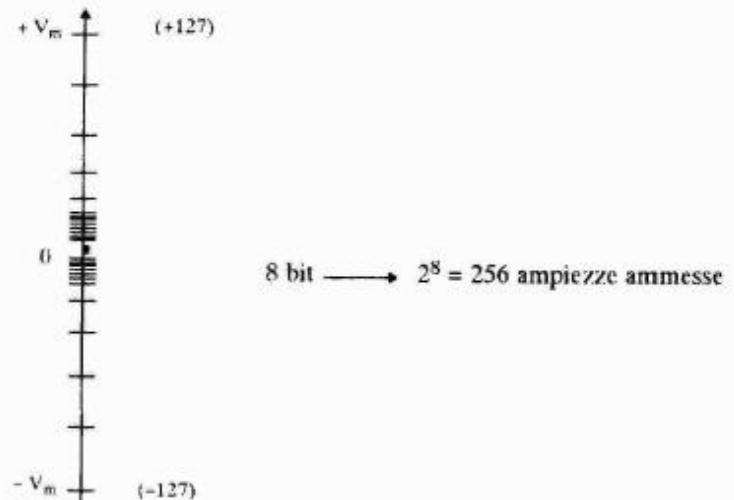
Tipi di quantizzazione

Quantizzazione uniforme

Asse delle ampiezze suddiviso in intervalli uguali



Quantizzazione non uniforme (o non lineare)

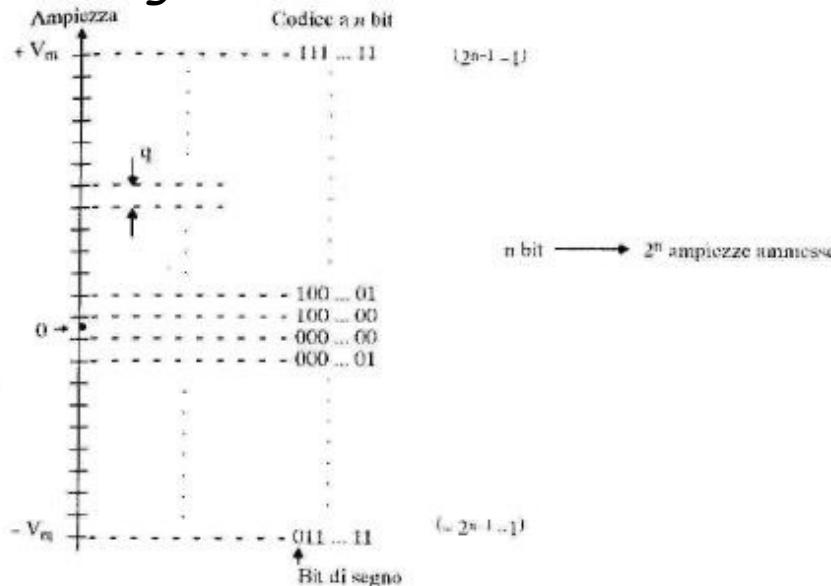


- L'errore di quantizzazione e' fisso ($< q/2$, con q passo di quantizzazione)
- Servono 12 bit per campione per riuscire a ottenere un errore di quantizzazione suff. Basso nel caso di valori piccoli
- Valori grandi possono sopportare errori maggiori
- Sono sufficienti 8 bit per campione in questo caso

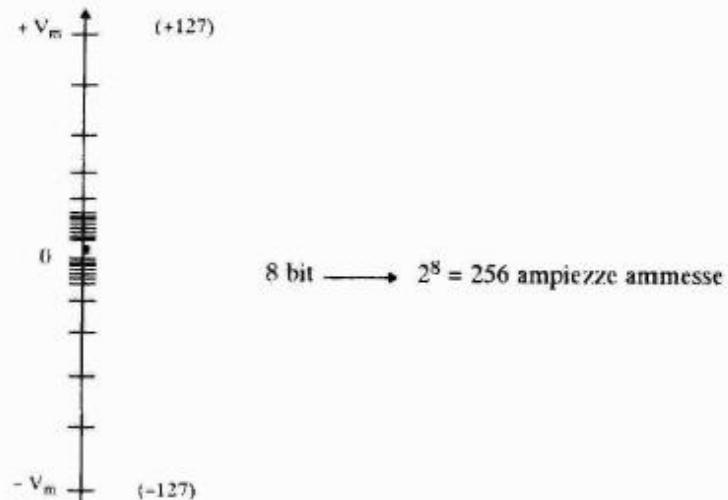
Tipi di quantizzazione

Quantizzazione uniforme

Asse delle ampiezze suddiviso in intervalli uguali



Quantizzazione non uniforme (o non lineare)

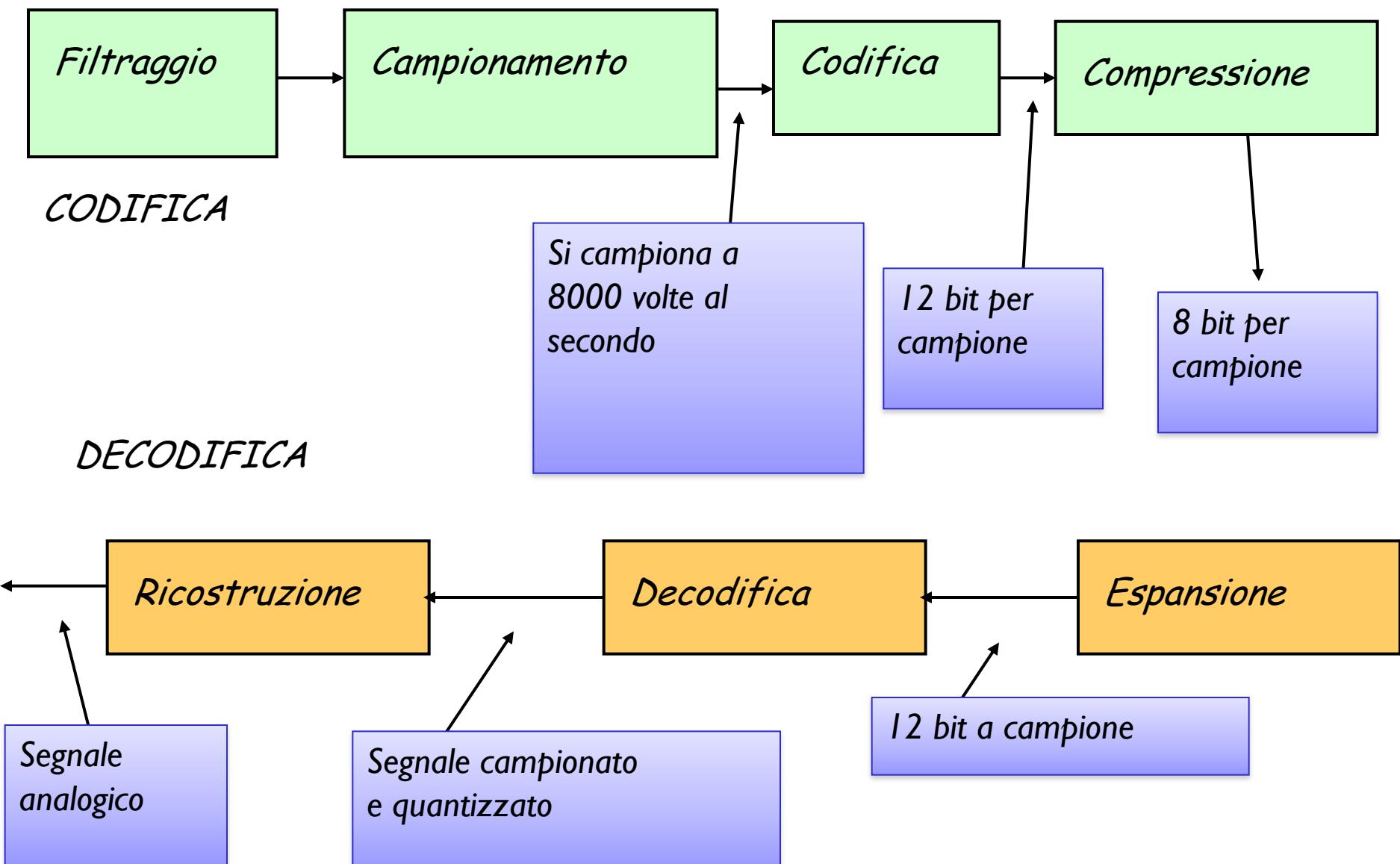


- L'errore di quantizzazione e' fisso ($< q/2$, con q passo di quantizzazione)
- Servono 12 bit per campione per riuscire a ottenere un errore di quantizzazione suff. Basso nel caso di valori piccoli

- Valori grandi possono sopportare errori maggiori
- Sono sufficienti 8 bit per campione in questo caso

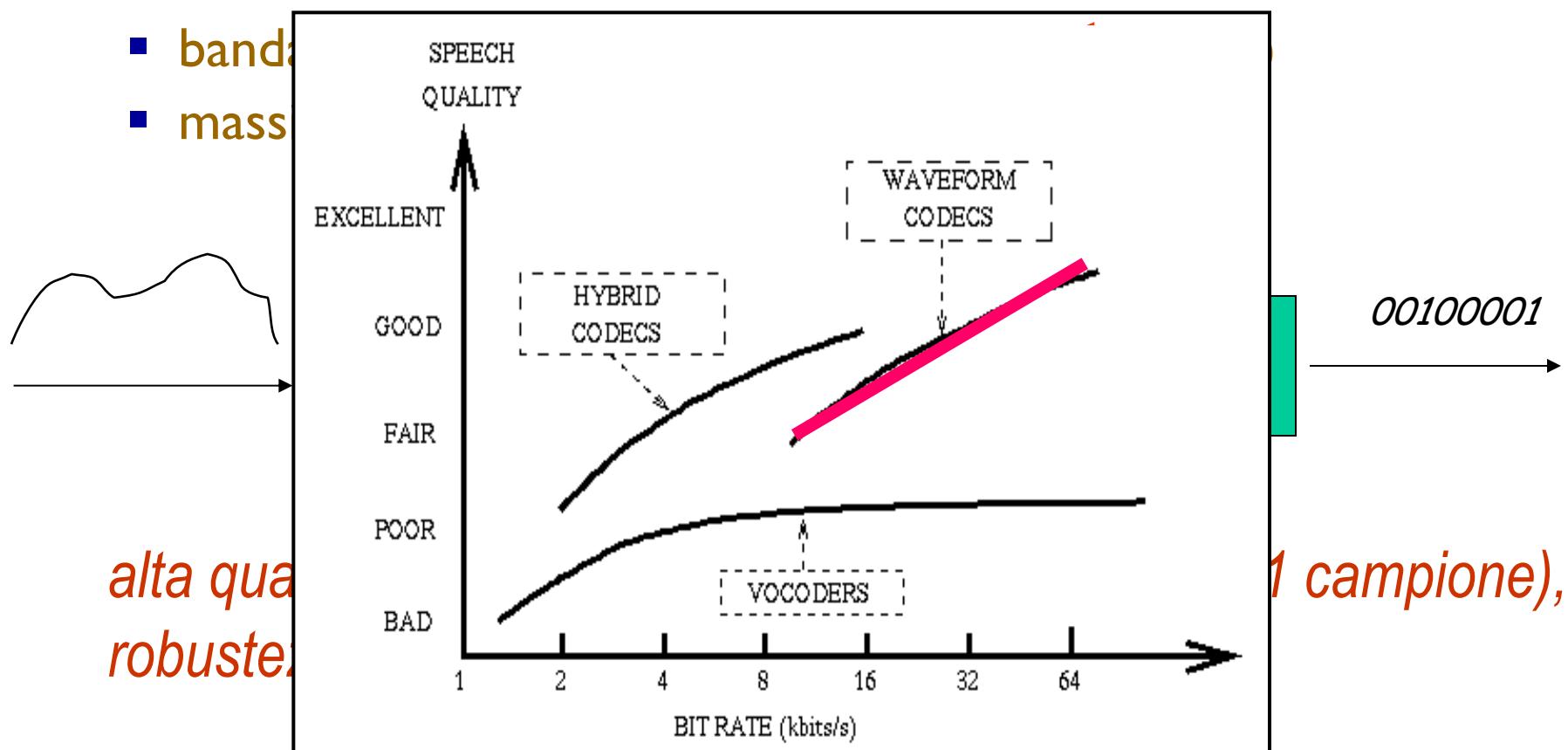
compressione

Fasi della codifica/decodifica



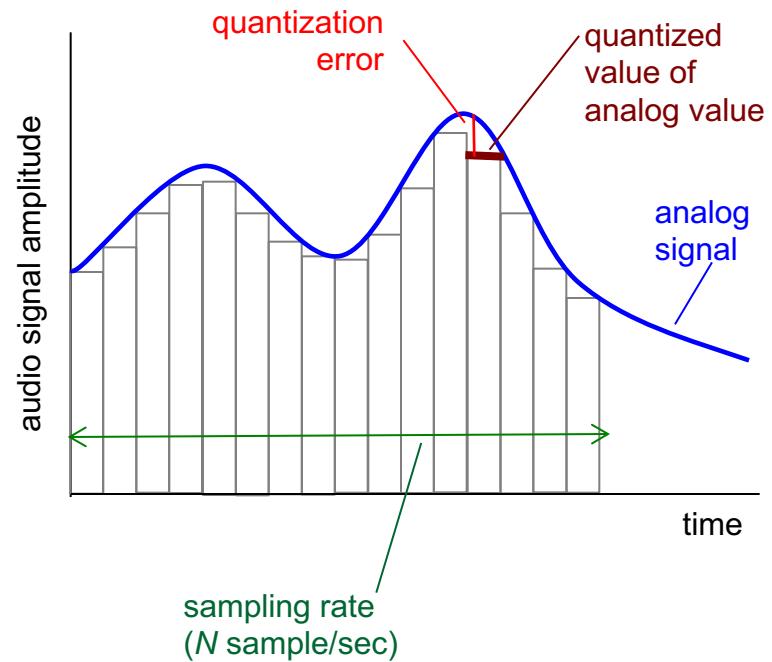
Waveform codecs

- ❖ nessuna conoscenza a priori di come il segnale sia stato generato
- ❖ informazione necessaria:
 - banda
 - massa



Multimedia: audio

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

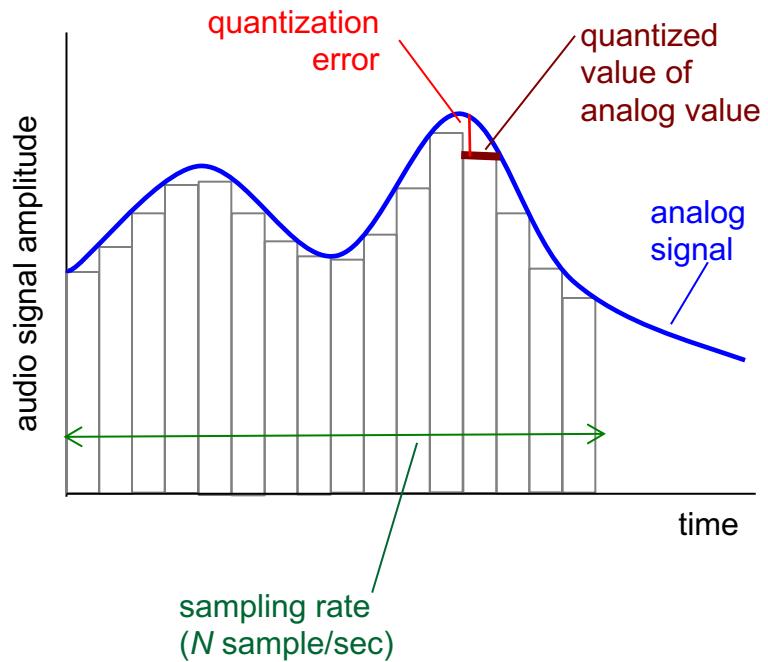


Multimedia: audio

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

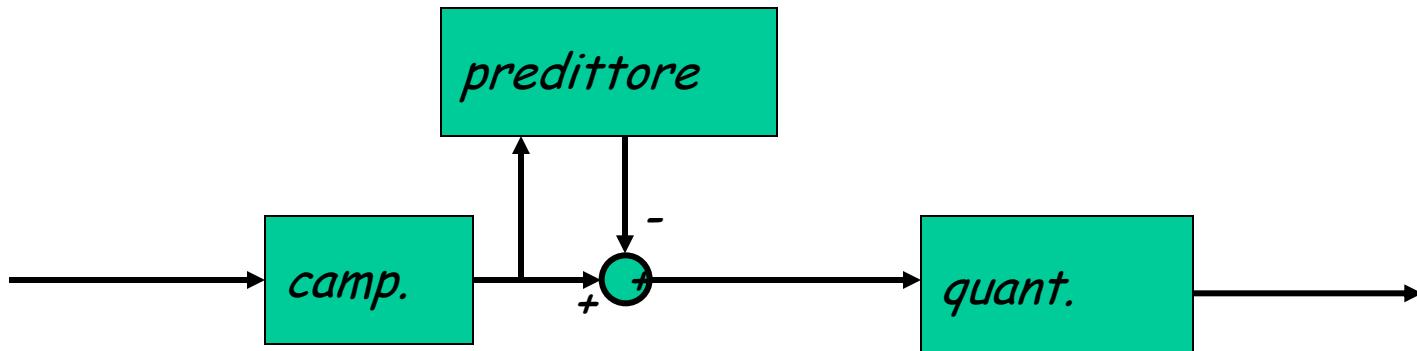
example rates

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



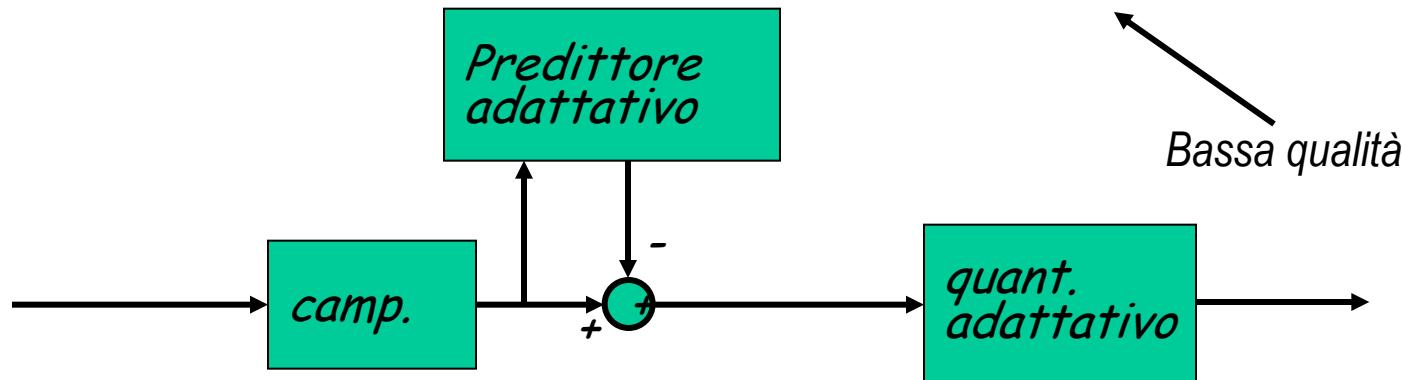
Codifica della voce: Differential PCM (DPCM)

- ❖ i campioni vocali successivi presentano della correlazione
- ❖ è possibile utilizzare metodi di predizione per valutare il campione successivo noti i precedenti
- ❖ si trasmette solo la differenza tra valore predetto e valore reale
- ❖ a causa della correlazione la varianza della differenza è minore ed è possibile codificarla con un minor numero di bit



Codifica della voce: Adaptive DPCM (ADPCM)

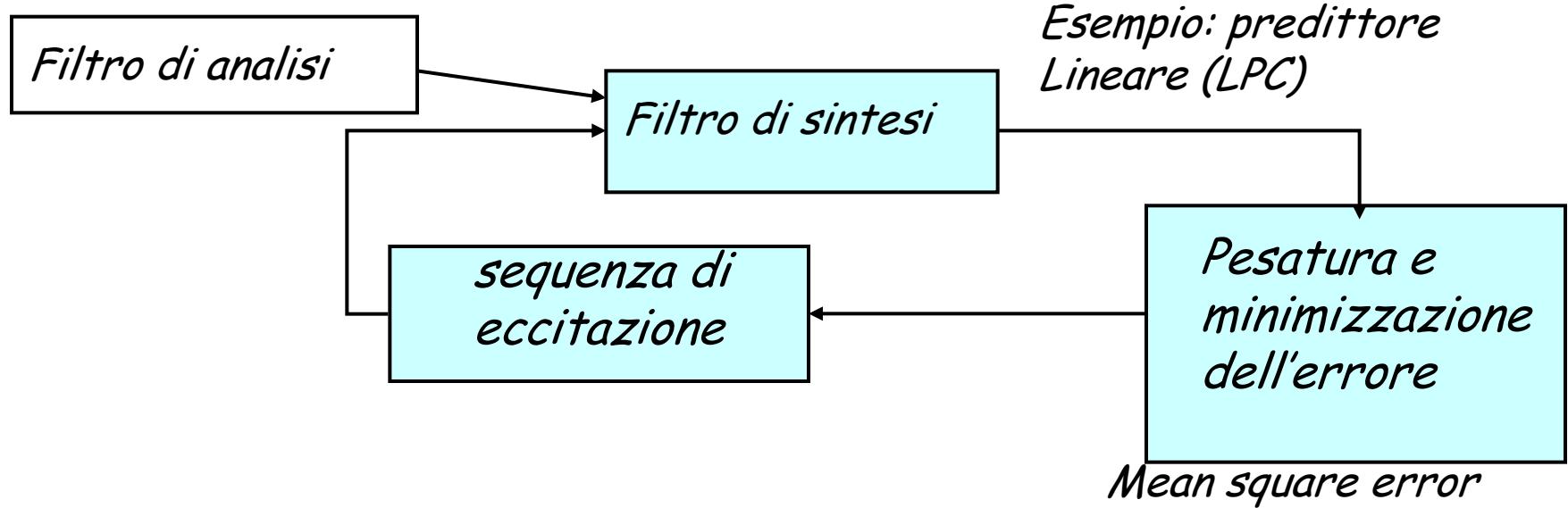
- ❖ le prestazioni migliorano se predittore e quantizzatore sono *Cordless DECT* adattativi
- ❖ standardizzato nel 1980 da ITU ADPCM a 32 kbit/s: G.721 ↗
- ❖ successivamente ADPCM a 40, 32, 24, 16 kbit/s: G.726 e G.727



Vantaggi: riduzione della velocità di emissione a parità di qualità
(da 64Kbps a 32 Kbps) 2) consentire una maggiore qualità a
parità di data rate disponibile per ogni canale vocale)

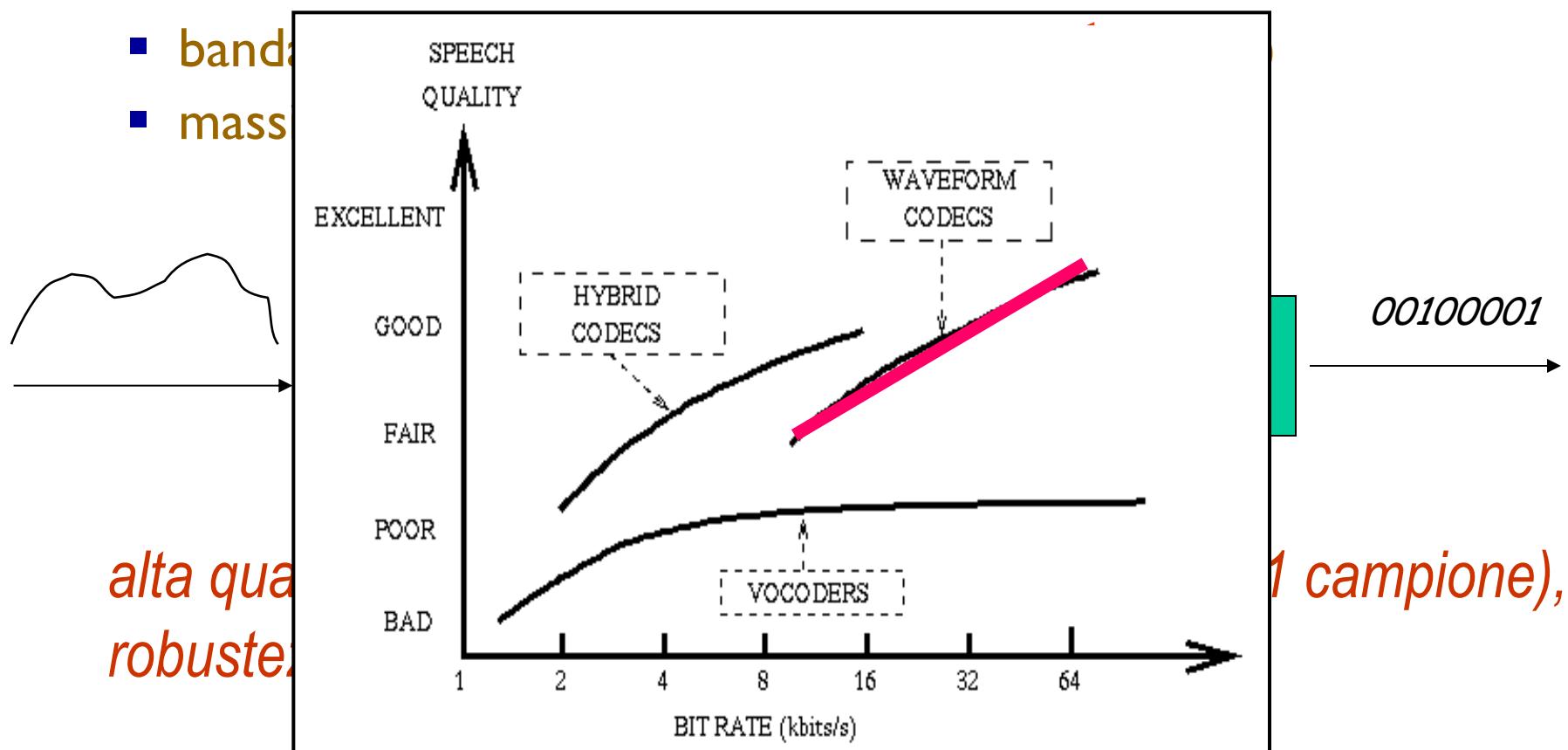
Source codecs (vocoders)

- Si basano su modelli di generazione della voce umana
- i modelli permettono di "togliere la ridondanza" da segmenti vocali fino a un'informazione base sufficiente a riprodurre la voce (Idea: se conosciamo la struttura del segnale poche informazioni caratteristiche saranno sufficienti a ricostruirlo)
- elevata complessità
- ritardi mediamente elevati
- sensibili a errori, rumore di fondo e suoni non umani



Waveform codecs

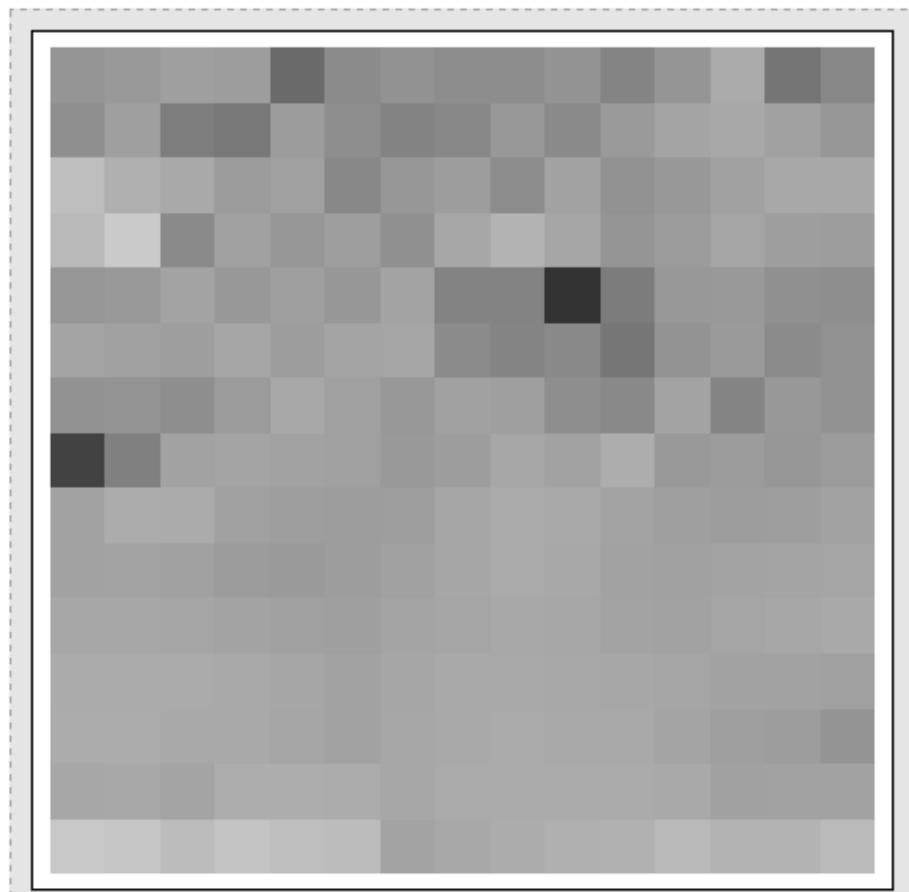
- ❖ nessuna conoscenza a priori di come il segnale sia stato generato
- ❖ informazione necessaria:
 - banda
 - massa



Images



The range of intensity values from 0 (black) to 255 (white).



The lower left 15 x 15 pixel portion of the image.

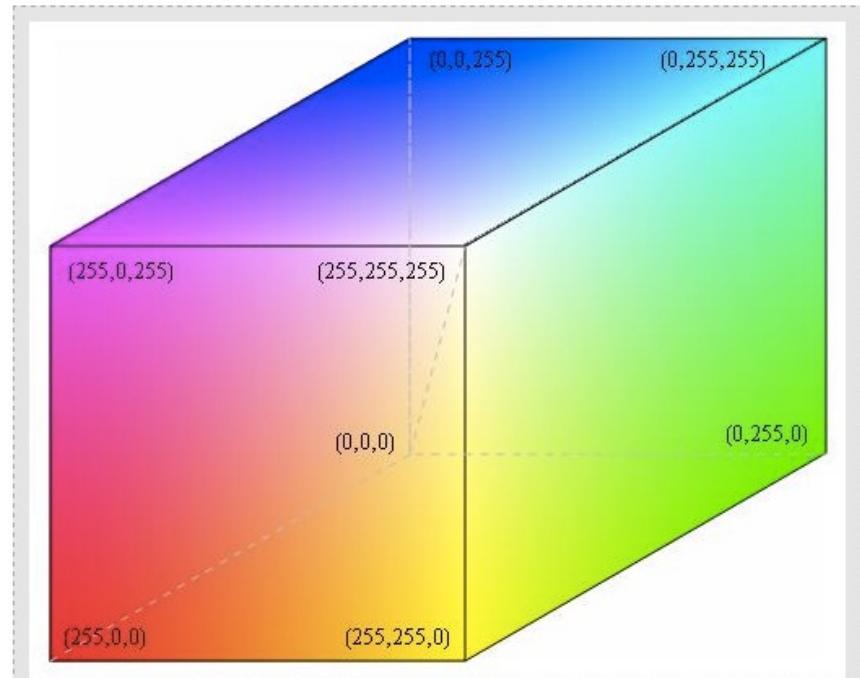


150 154 160 157 106 140 147 142 141 147 132 150 171 117 136
144 159 125 121 157 143 132 136 153 138 155 164 169 162 152
190 175 169 155 161 136 152 158 141 162 147 153 161 168 169
185 203 139 161 151 159 145 167 179 167 150 155 165 159 158
151 153 163 152 160 152 164 131 131 51 124 152 154 145 143
164 162 158 167 157 164 166 139 132 138 119 148 154 139 146
147 148 143 155 169 160 152 161 159 143 138 163 132 152 146
66 129 163 165 163 161 154 157 167 162 174 153 156 151 156
162 173 172 161 158 158 159 167 171 169 164 159 158 159 162
163 164 161 155 155 158 161 167 171 168 162 162 163 164 166
167 167 165 163 160 160 164 166 169 168 164 163 165 167 170
172 171 170 170 166 163 166 170 169 168 167 165 163 163 160
173 172 170 169 166 163 167 169 170 170 170 165 160 157 148
167 168 165 173 173 172 167 170 170 171 171 169 162 163 162
200 198 189 196 191 188 163 168 172 177 177 186 180 180 188

Pixel intensity values of the lower left 15 x 15 pixel portion of the image.

Colored images

Luminance and Chrominance



The RGB colorspace cube. Two-dimensional version

The first step to convert a red, green, and blue triple (r,g,b) to YCbCr space is to divide each intensity by 255 so that the resulting triple has values in the interval $[0,1]$. Let's define $(\bar{r}, \bar{g}, \bar{b}) = (r/255, g/255, b/255)$.

We obtain the luminance value y using the formula

$$y = .299\bar{r} + .587\bar{g} + .114\bar{b}$$

$$Cb = (\bar{b} - y)/1.772$$

$$Cr = (\bar{r} - y)/1.402$$

Multimedia: video

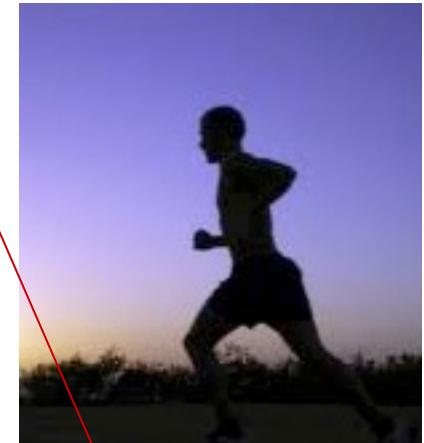
- ❖ video: sequence of images displayed at constant rate
 - e.g. 24 images/sec
- ❖ digital image: array of pixels
 - each pixel represented by bits
- ❖ coding: use redundancy *within* and *between* images to decrease # bits used to encode image
 - spatial (within image)
 - temporal (from one image to 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 $i+1$

Compression rate over time

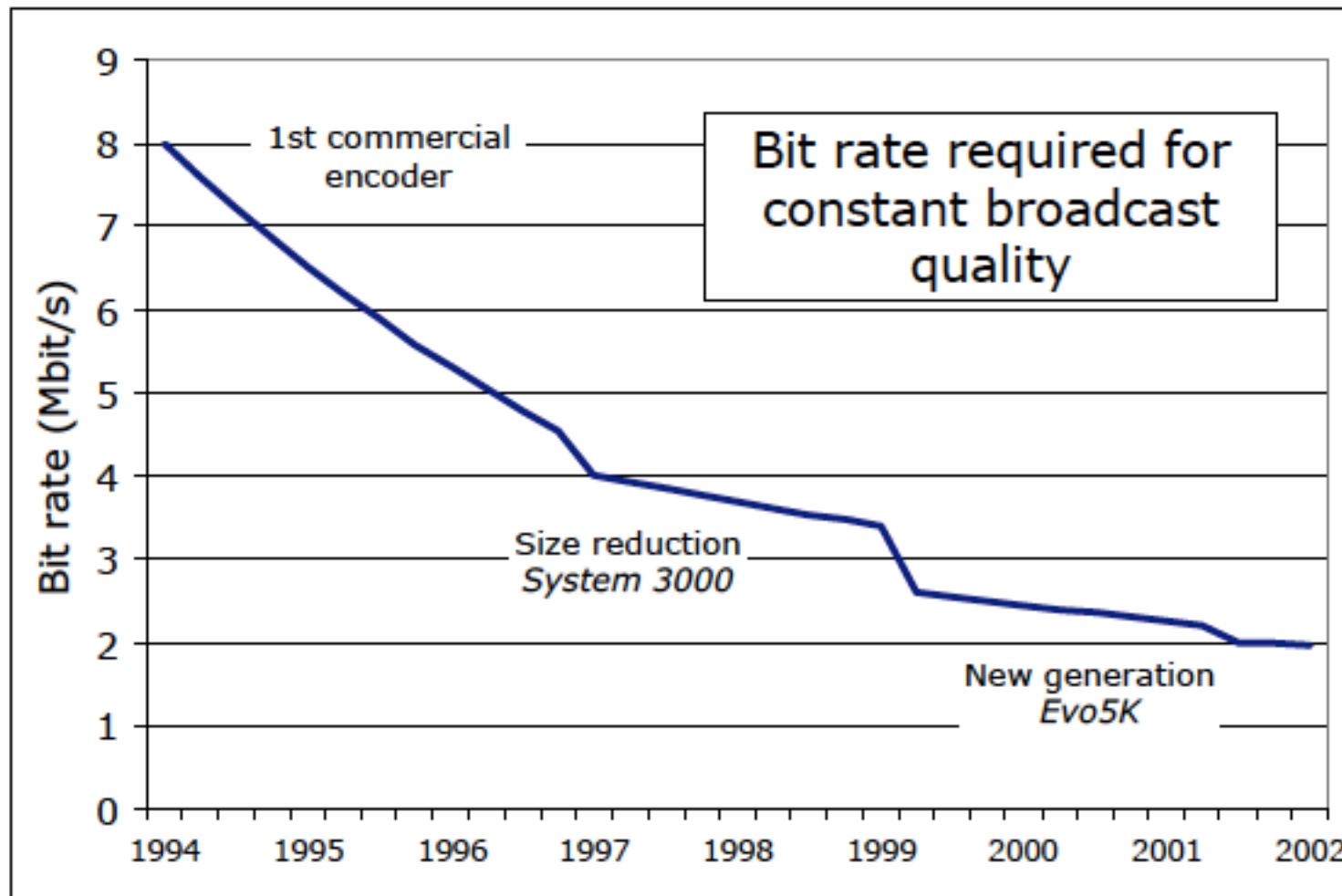


Figure 4. Improvements in coding efficiency are facilitated by MPEG-based competition. Courtesy Tandberg TV.

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

Difference wrt elastic traffic

- QoS requirements
 - *Bounded latency and delay jitter*
 - *Data rate*
 - *Limited packet loss can be tolerated*

Multimedia networking: outline

7.1 multimedia networking applications

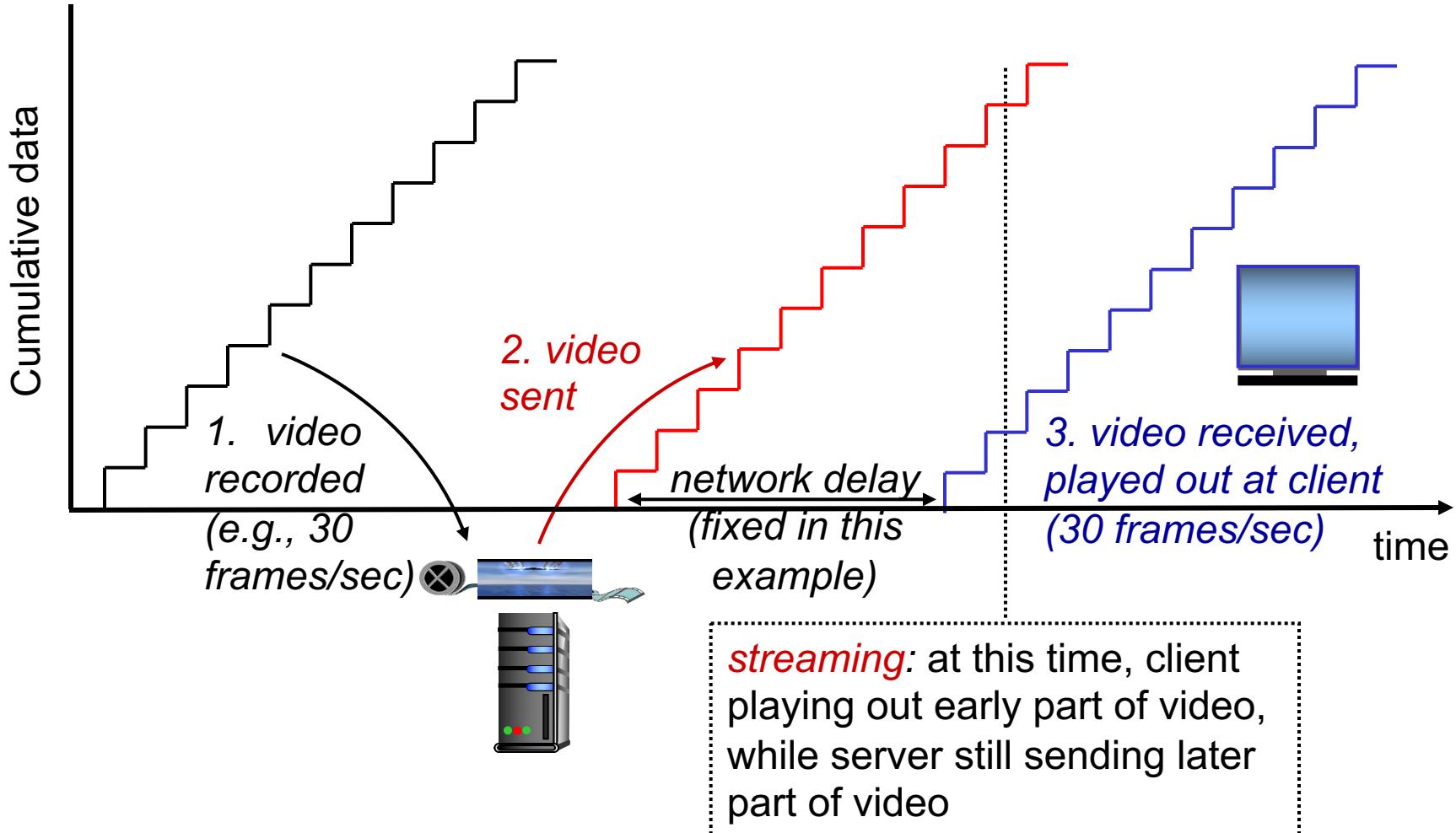
7.2 streaming *stored* video

7.3 voice-over-IP

7.4 protocols for *real-time* conversational
applications

7.5 network support for multimedia

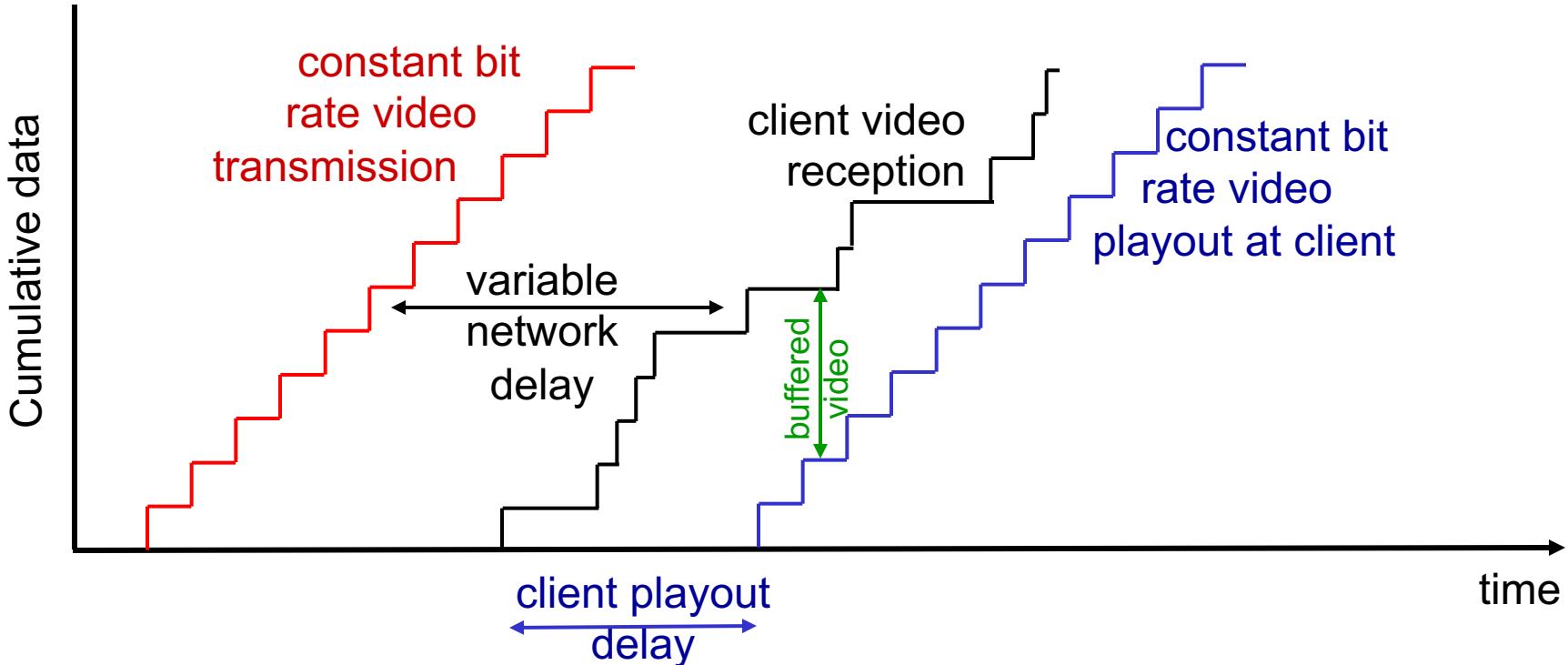
Streaming stored video:



Streaming stored video: challenges

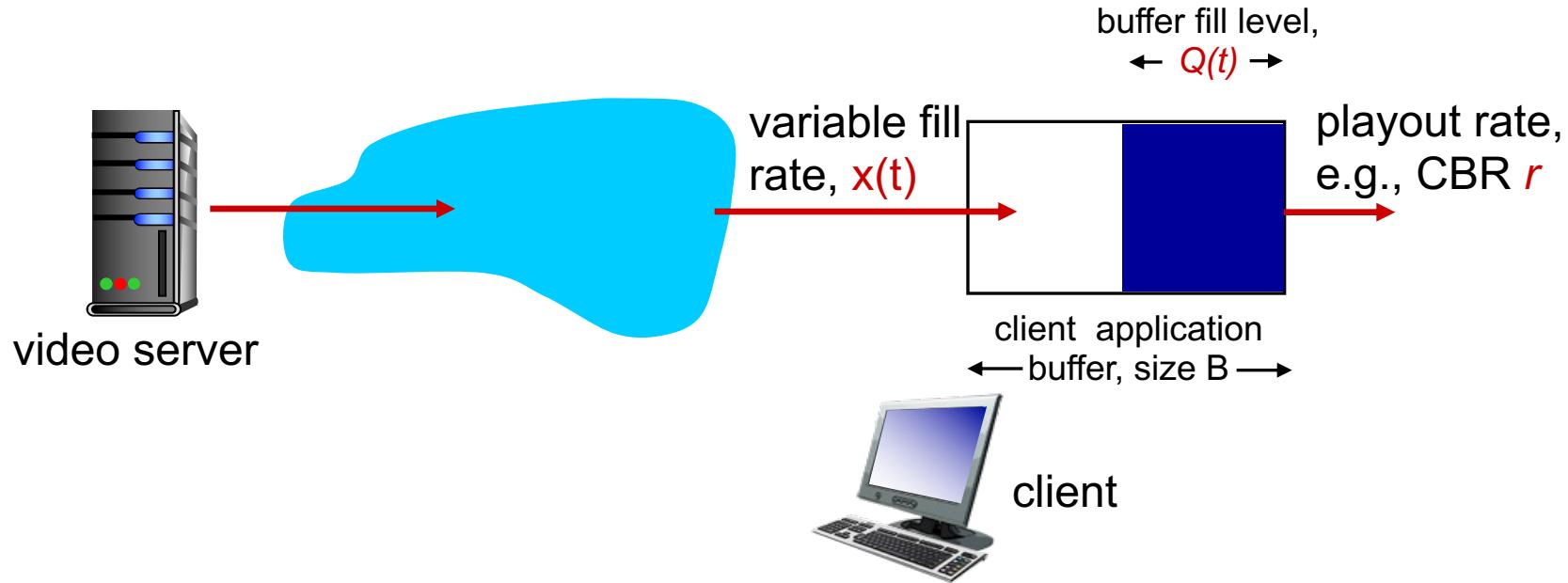
- ❖ *continuous playout constraint*: once client playout begins, playback must match original timing
 - ... but *network delays are variable* (jitter), so will need *client-side buffer* to match playout requirements
- ❖ other challenges:
 - client interactivity: pause, fast-forward, rewind, jump through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

Streaming stored video: revisited

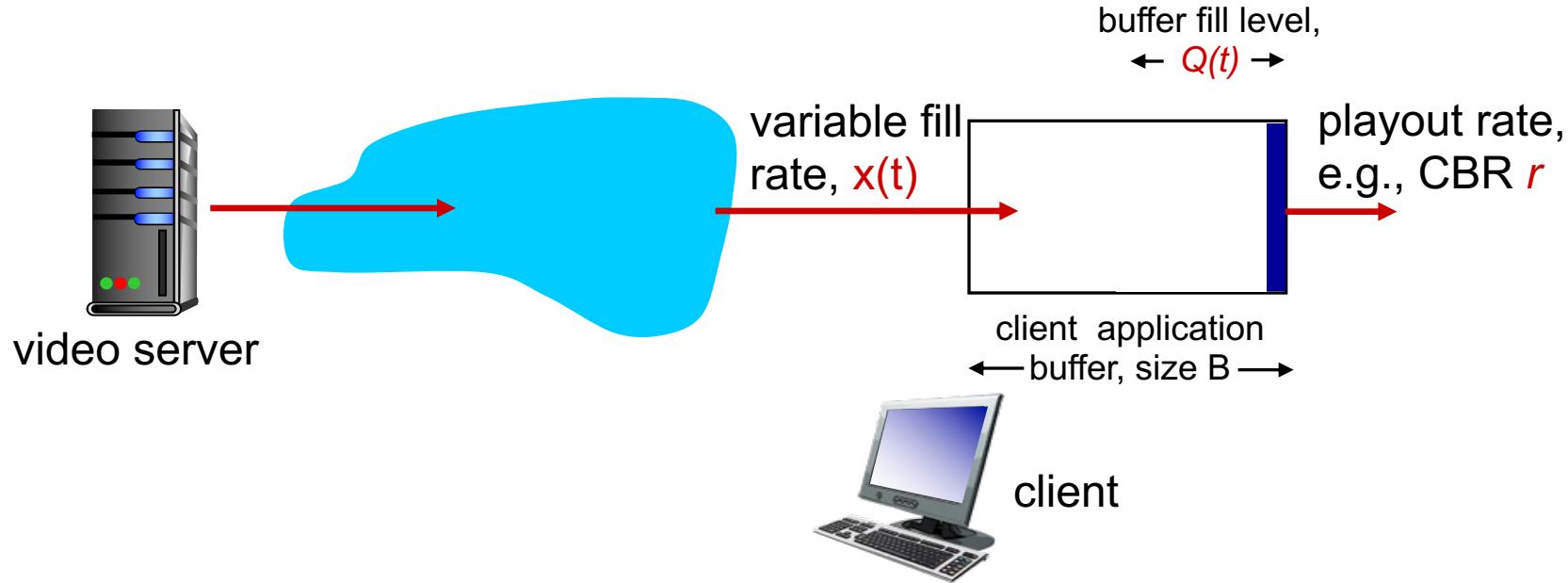


- ❖ *client-side buffering and playout delay:* compensate for network-added delay, delay jitter

Client-side buffering, playout

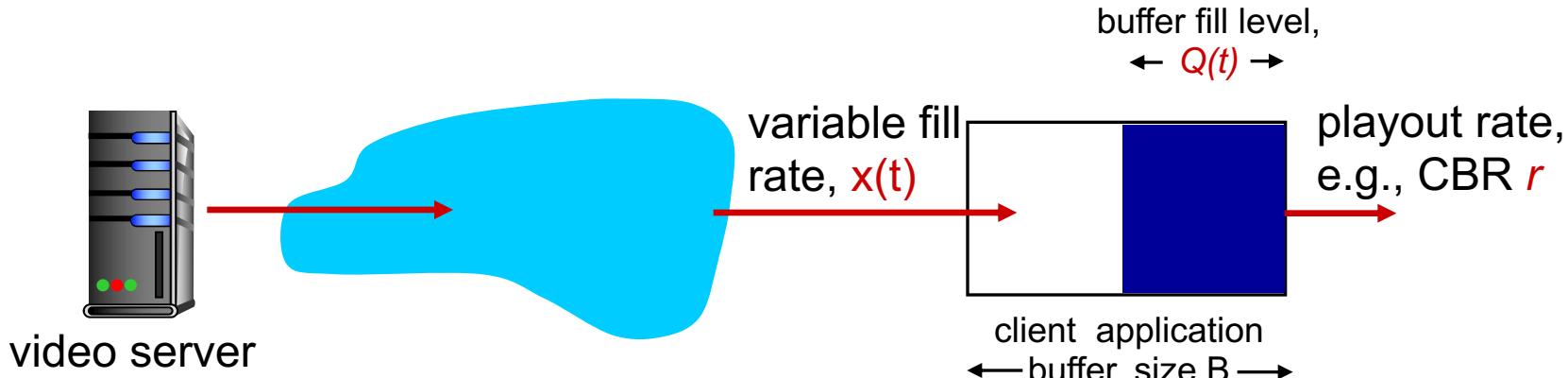


Client-side buffering, playout



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

Client-side buffering, playout



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

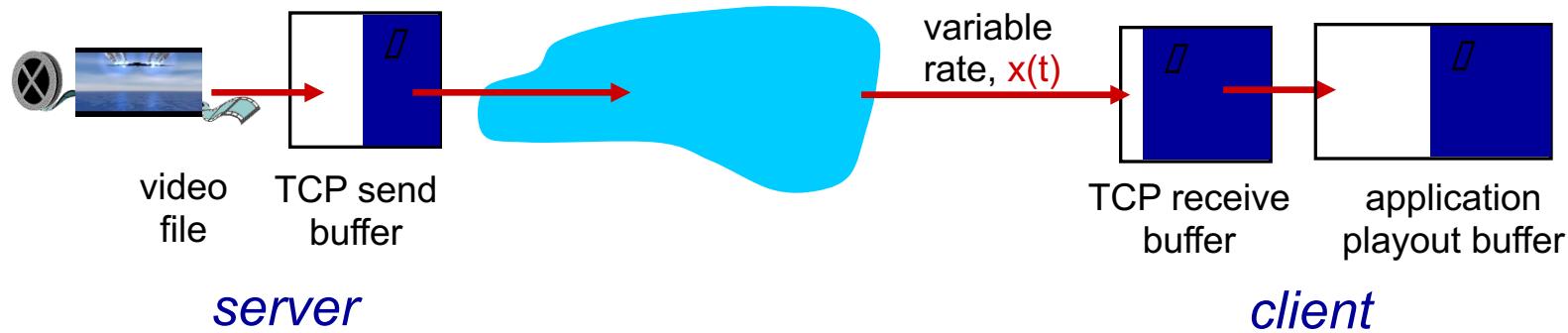
- ❖ $\bar{x} < r$: buffer eventually empties (causing freezing of video playout until buffer again fills)
- ❖ $\bar{x} > r$: buffer will not empty, provided initial playout delay is large enough to absorb variability in $x(t)$
 - *initial playout delay tradeoff*: buffer starvation less likely with larger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

Streaming multimedia: UDP

- ❖ server sends at rate appropriate for client
 - often: send rate = encoding rate = constant rate
 - transmission rate can be oblivious to congestion levels
- ❖ short playout delay (2-5 seconds) to remove network jitter
- ❖ error recovery: application-level, time-permitting
- ❖ RTP [RFC 2326]: multimedia payload types
- ❖ UDP may *not* go through firewalls

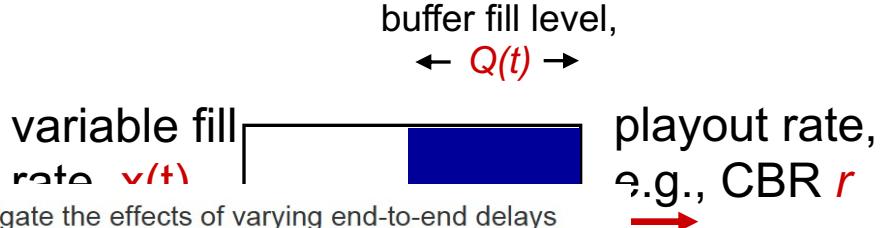
Streaming multimedia: HTTP

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



- ❖ fill rate fluctuates due to TCP congestion control, retransmissions (in-order delivery)
- ❖ larger playout delay: smooth TCP delivery rate
- ❖ HTTP/TCP passes more easily through firewalls

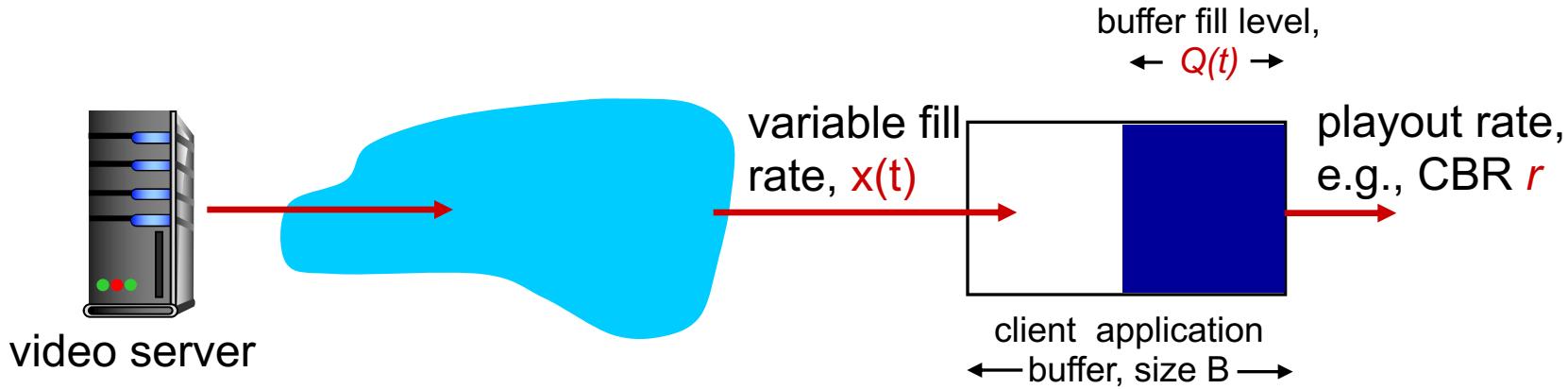
Prefetching



As we just learned, client-side buffering can be used to mitigate the effects of varying end-to-end delays and varying available bandwidth. In our earlier example in [Figure 9.1](#), the server transmits video at the rate at which the video is to be played out. However, for streaming *stored* video, the client can attempt to download the video at a rate *higher* than the consumption rate, thereby **prefetching** video frames that are to be consumed in the future. This prefetched video is naturally stored in the client application buffer. Such prefetching occurs naturally with TCP streaming, since TCP's congestion avoidance mechanism will attempt to use all of the available bandwidth between server and client.

To gain some insight into prefetching, let's take a look at a simple example. Suppose the video consumption rate is 1 Mbps but the network is capable of delivering the video from server to client at a constant rate of 1.5 Mbps. Then the client will not only be able to play out the video with a very small playout delay, but will also be able to increase the amount of buffered video data by 500 Kbits every second. In this manner, if in the future the client receives data at a rate of less than 1 Mbps for a brief period of time, the client will be able to continue to provide continuous playback due to the reserve in its buffer. [\[Wang 2008\]](#) shows that when the average TCP throughput is roughly twice the media bit rate, streaming over TCP results in minimal starvation and low buffering delays.

Prefetching



HTTP streaming systems often make use of the [HTTP byte-range header](#) in the HTTP GET request message, which specifies the specific range of bytes the client currently wants to retrieve from the desired video.

The problem of prefetching is in case of early termination or reposition requested by the client.

*Amount of prefetched video is therefore limited to avoid wasting resources.
It is also possibly limited by the buffer space.*

But then what to do to cope with variable throughput?

Streaming multimedia: DASH

- ❖ *DASH: Dynamic, Adaptive Streaming over HTTP*
- ❖ *server:*
 - divides video file into multiple chunks
 - each chunk stored, encoded at different rates
 - *manifest file:* provides URLs for different chunks
- ❖ *client:*
 - periodically measures server-to-client bandwidth
 - consulting manifest, requests one chunk at a time
 - chooses maximum coding rate sustainable given current bandwidth
 - can choose different coding rates at different points in time (depending on available bandwidth at time)

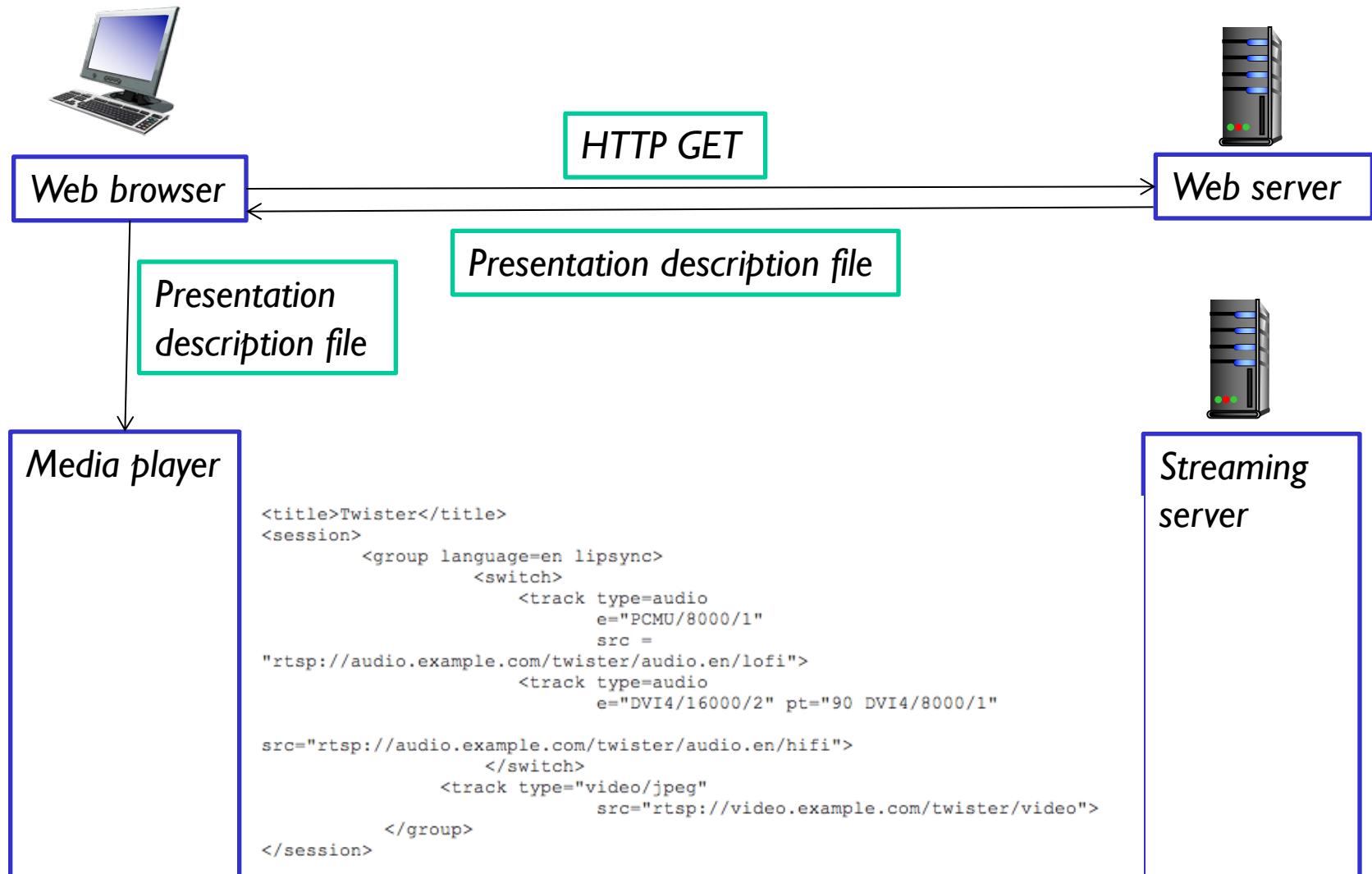
Streaming multimedia: DASH

- ❖ DASH: Dynamic, Adaptive Streaming over HTTP
 - ❖ server:
 - divides video file
 - each chunk stored
 - manifest file: provides URLs
 - ❖ client:
 - periodically measures bandwidth
 - consulting manifest file
 - chooses manifest based on current bandwidth
 - can choose different points in timeline at time)
- + Use of CDN
Already covered when we have studied the Netflix case
- ## CDN: “simple” content access scenario
- Bob (client) requests video <http://netcinema.com/6Y7B23V>
- video stored in CDN at <http://KingCDN.com/NetC6y&B23V>
-
- The diagram illustrates the flow of a video request from Bob's client to a KingCDN server. It shows three main components: netcinema.com, KingCDN.com, and Bob's client. The process is numbered as follows:
1. Bob gets URL for video <http://netcinema.com/6Y7B23V> from netcinema.com web page.
 2. Bob's local DNS resolves <http://netcinema.com/6Y7B23V>.
 3. netcinema's authoritative DNS returns URL <http://KingCDN.com/NetC6y&B23V>.
 4. KingCDN's authoritative DNS resolves <http://KingCDN.com/NetC6y&B23V> via KingCDN's authoritative DNS, which returns IP address of KlingCDN server with video.
 5. Bob's local DNS performs a reverse lookup to find the IP address of the KlingCDN server.
 6. Bob's client sends a request to the KlingCDN server to stream the video via HTTP.

Real time streaming protocol (RTSP)

- ❖ Allows a media player to control the transmission of a media stream
- ❖ Out of band protocol (port number 544 different from that of the media stream)
- ❖ Over TCP
- ❖ Control actions:
 - Pause, resume, repositioning of playback, fast forward, rewind

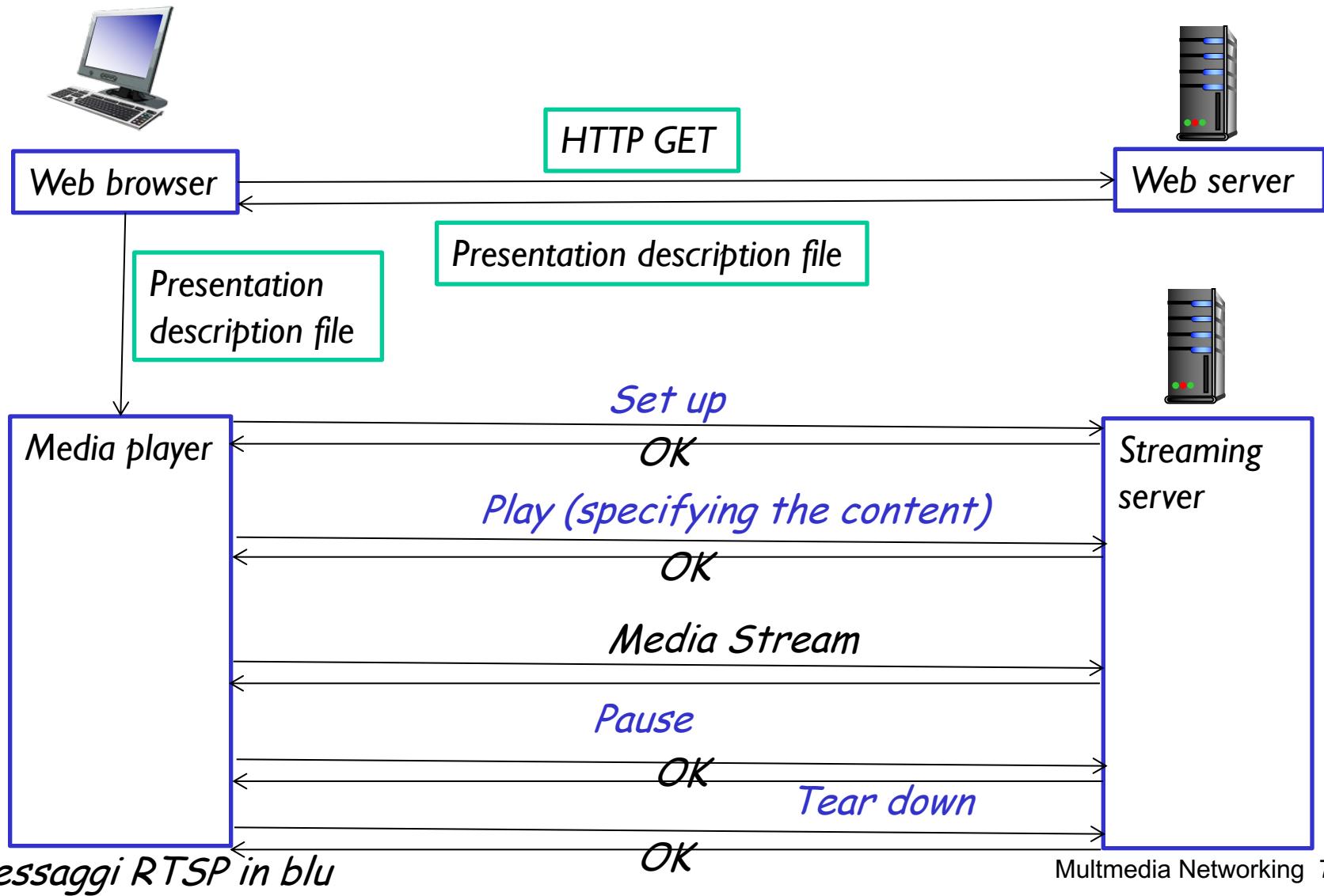
Streaming multimedia: RTSP



Presentation description file

```
<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>
```

Streaming multimedia: RTSP



Real time streaming protocol (RTSP)

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

Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming stored video

7.3 voice-over-IP

7.4 protocols for *real-time* conversational
applications

7.5 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?

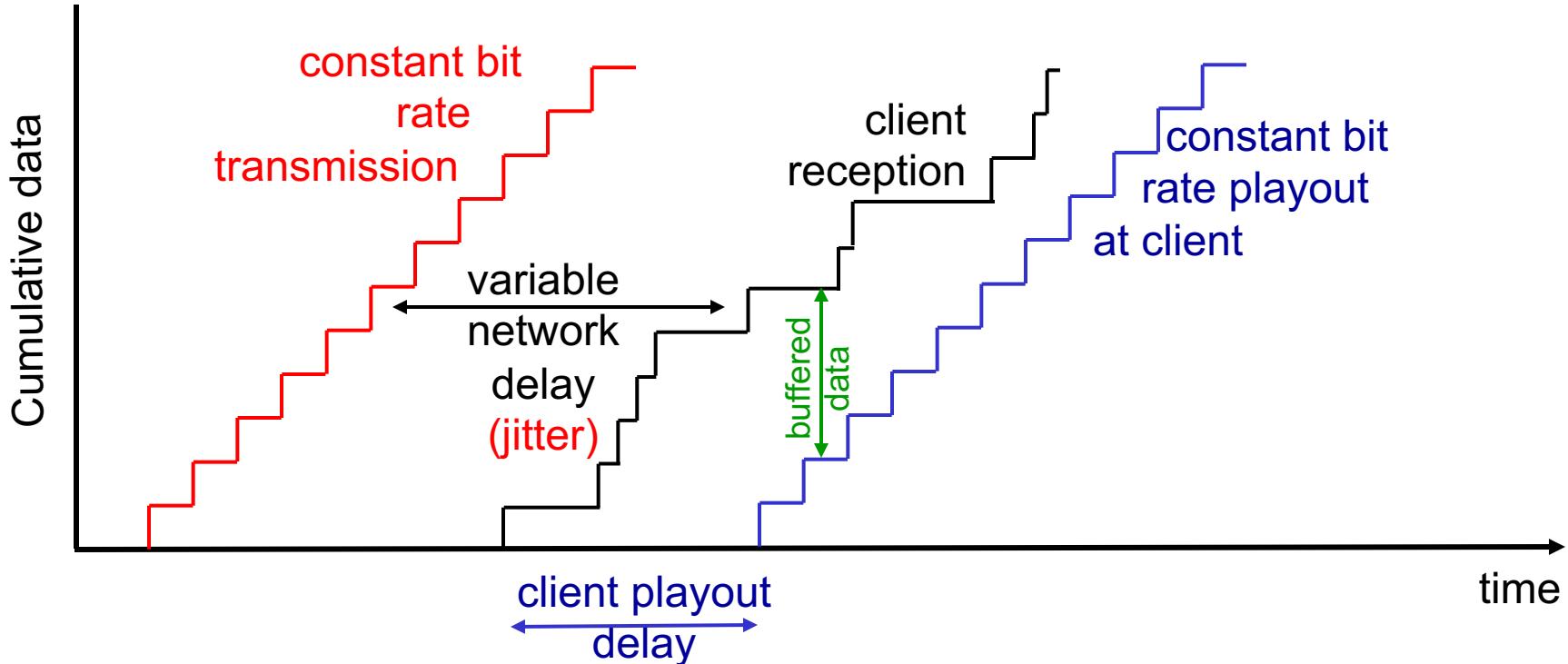
VoIP characteristics

- ❖ 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



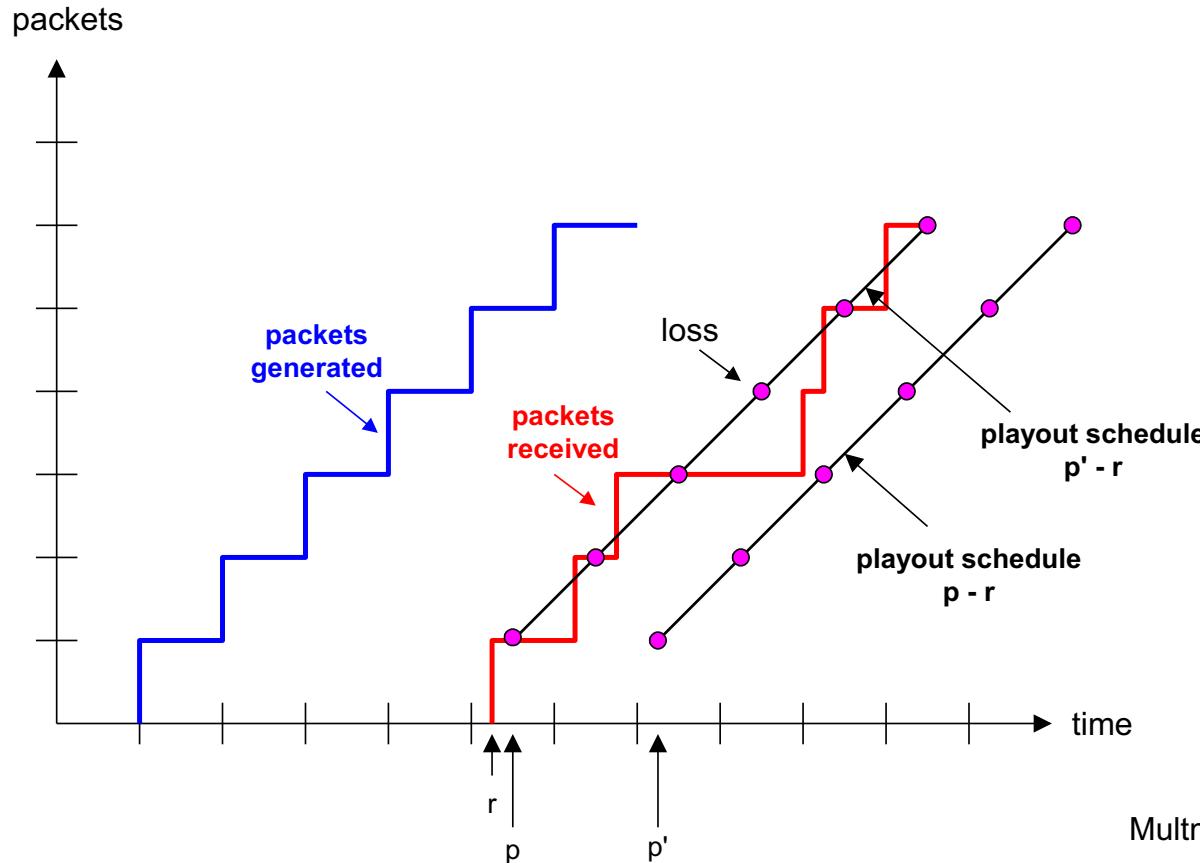
- ❖ end-to-end delays of two consecutive packets:
difference can be more or less than 20 msec
(transmission time difference)

VoIP: fixed playout delay

- ❖ receiver attempts to playout each chunk exactly q 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*

VoIP: fixed playout delay

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



Adaptive playout delay (I)

- ❖ **goal:** low playout delay, low late loss rate
 - ❖ **approach:** adaptive playout delay adjustment:
 - estimate network delay, adjust playout delay at beginning of each talk spurt
 - silent periods compressed and elongated
 - chunks still played out every 20 msec during talk spurt
 - ❖ adaptively estimate packet delay: (EWMA - exponentially weighted moving average, recall TCP RTT estimate):

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

delay estimate
after i th packet
small constant,
e.g. 0.1
time received - time sent
(timestamp)
measured delay of i th 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:

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

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.
 - Energy on the channel

*Multimedia packets must contain
Sequence number
Timestamping*

VoIP: recovery from packet loss (I)

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

- ❖ each ACK/NAK takes \sim one RTT
- ❖ alternative: *Forward Error Correction (FEC)*
 - send enough bits to allow recovery without retransmission (recall two-dimensional parity in Ch. 5)

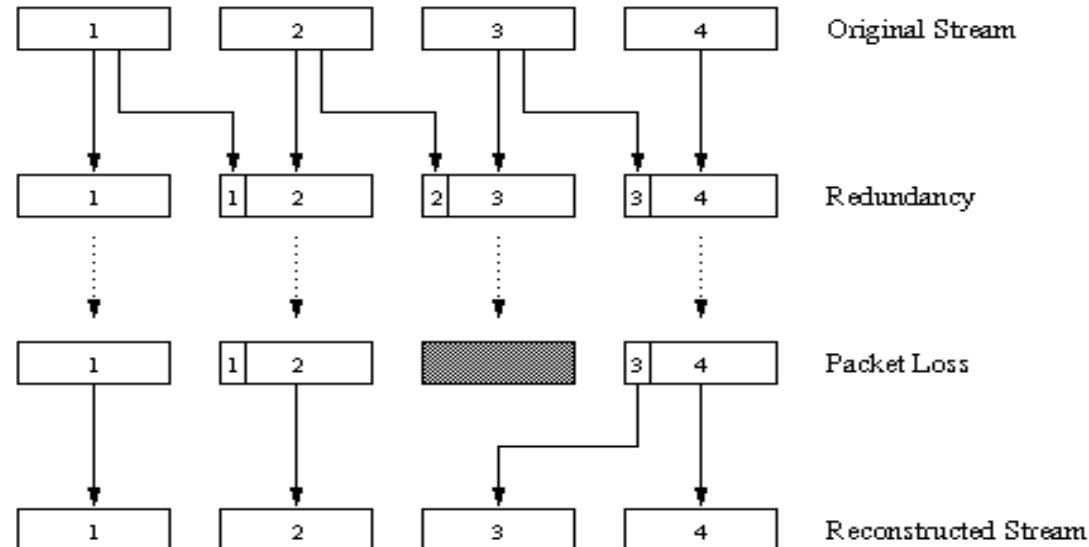
simple FEC

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

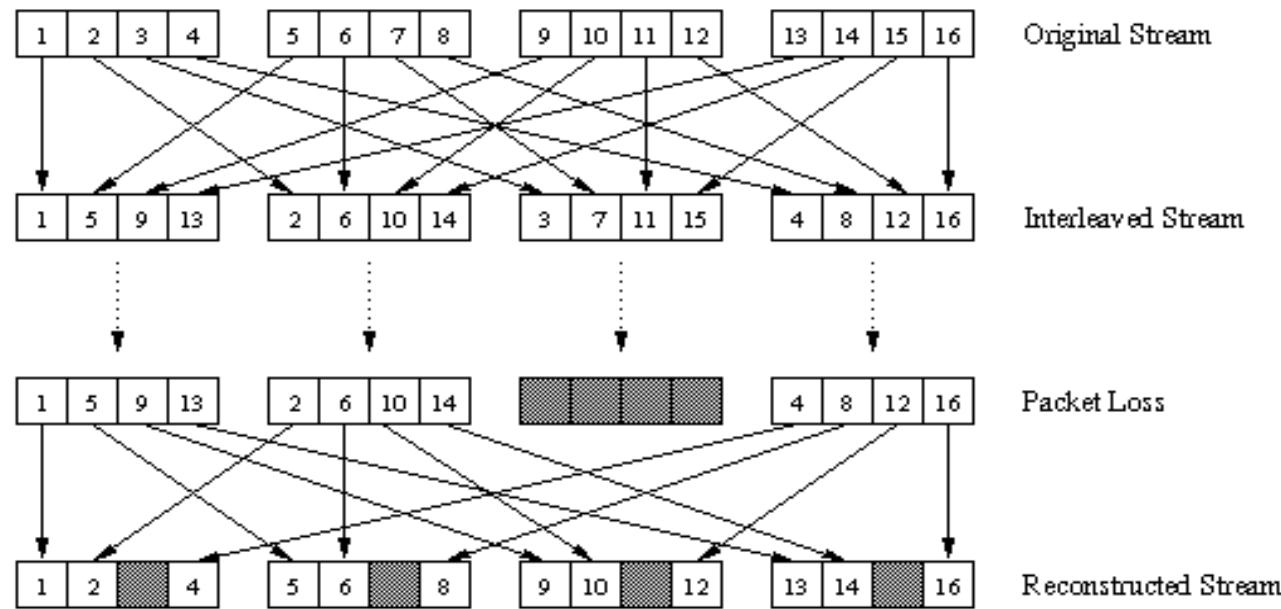
VoIP: recovery from packet loss (2)

another FEC scheme:

- ❖ “piggyback lower quality stream”
- ❖ send lower resolution audio stream as redundant information
- ❖ e.g., nominal stream PCM at 64 kbps and redundant stream GSM at 13 kbps
- ❖ non-consecutive loss: receiver can conceal loss
- ❖ generalization: can also append (n-1)st and (n-2)nd low-bit rate chunk (or more)



VoIP: recovery from packet loss (3)

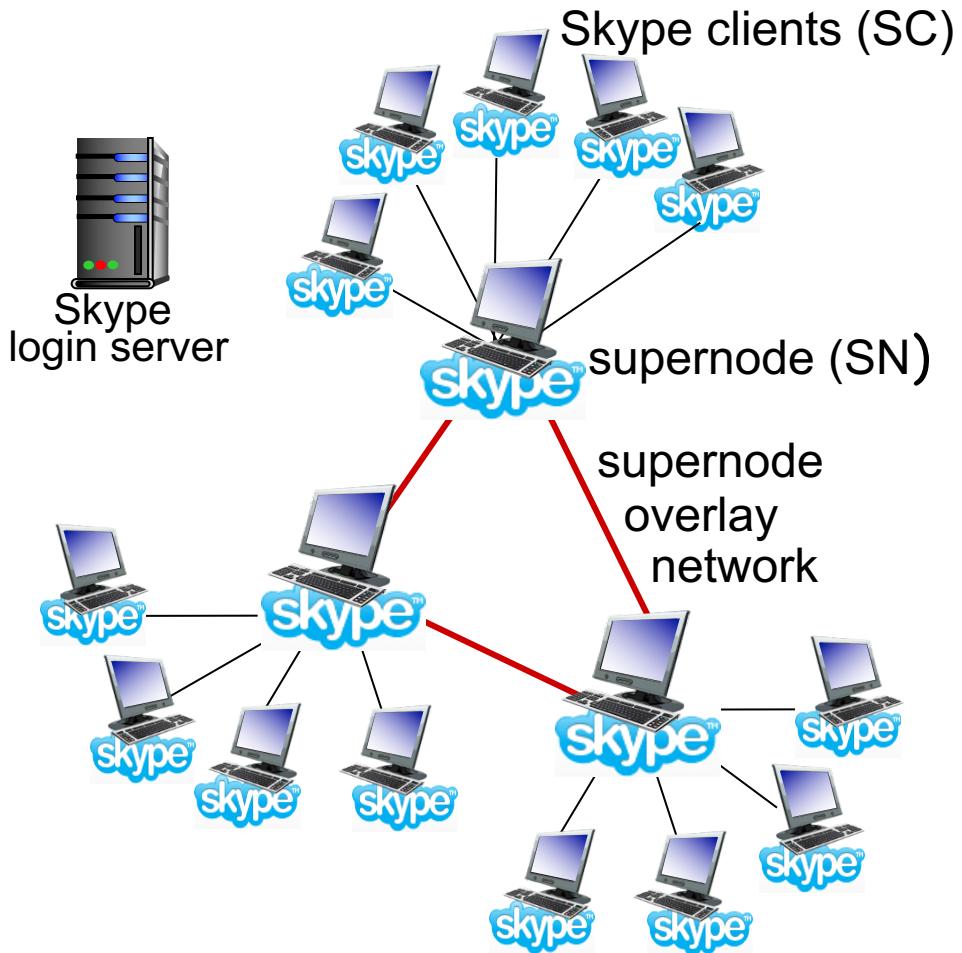


interleaving to conceal loss:

- ❖ 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

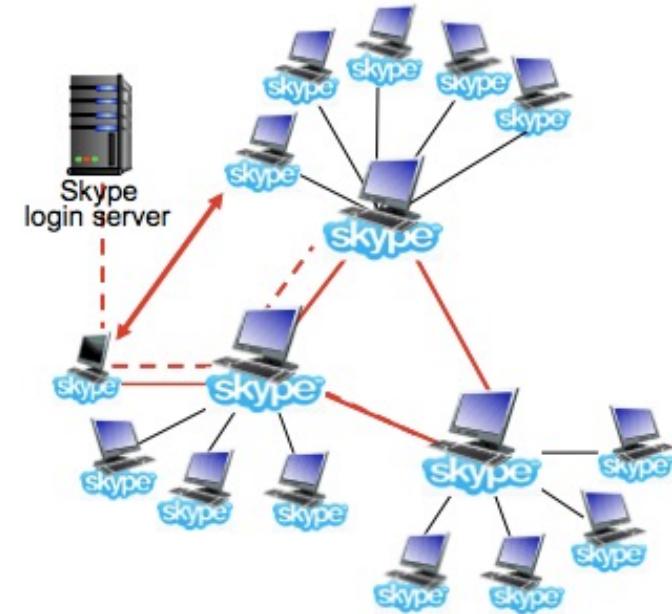
- ❖ proprietary application-layer protocol (inferred via reverse engineering)
 - encrypted msgs
- ❖ P2P components:
 - **clients:** skype peers connect directly to each other for VoIP call
 - **super nodes (SN):** skype peers with special functions
 - **overlay network:** among SNs to locate SCs
 - **login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



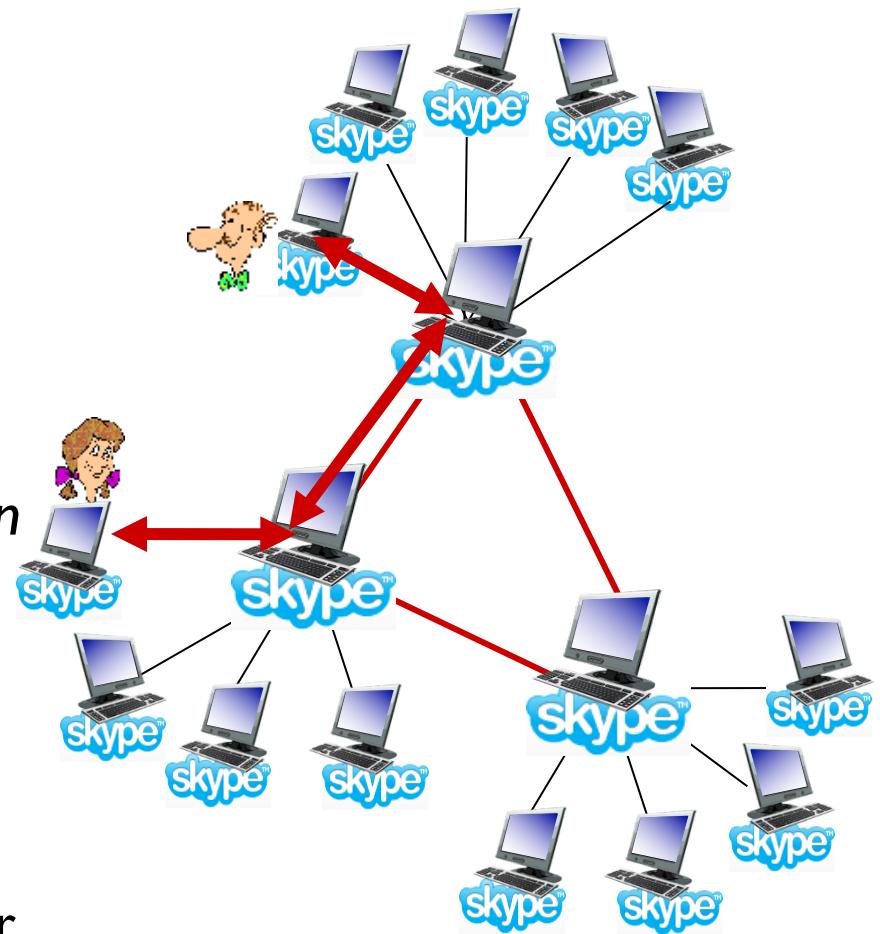
*Proprietary scheme. General rules:
Audio/video packets by default over UDP
Control packets over TCP
data (in case Firewalls block UDP) over TCP*

*Use of FEC; different codecs supported
FEC/codec adapted based on net. Conditions*

IP/port numbers maintained in a distributed indexing across SNs

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



Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming stored video

7.3 voice-over-IP

7.4 protocols for real-time conversational
applications: RTP, SIP

7.5 network support for multimedia

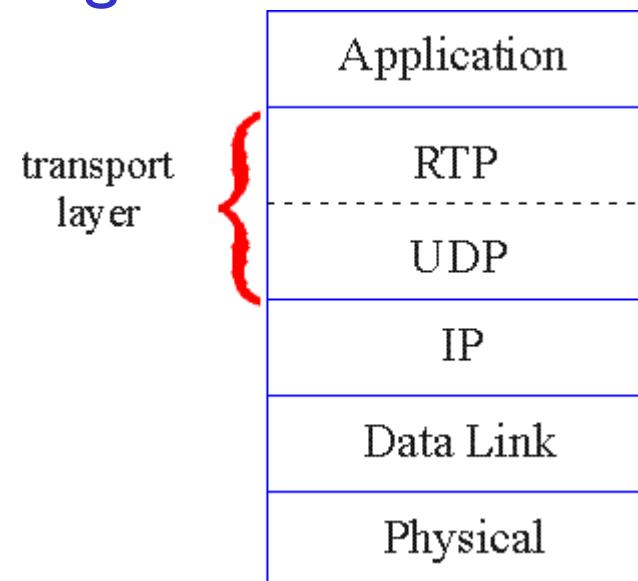
Real-Time Protocol (RTP)

- ❖ RTP specifies packet structure for packets carrying audio, video data
- ❖ RFC 3550
- ❖ RTP packet provides
 - payload type identification
 - packet sequence numbering
 - time stamping
- ❖ RTP runs in end systems
- ❖ RTP packets encapsulated in UDP segments
- ❖ interoperability: if two VoIP applications run RTP, they may be able to work together

RTP runs on top of UDP

RTP libraries provide transport-layer interface that extends UDP:

- port numbers, IP addresses
- payload type identification
- packet sequence numbering
- time-stamping



RTP example

example: sending 64 kbps PCM-encoded voice over RTP

- ❖ application collects encoded data in chunks, e.g., every 20 msec = 160 bytes in a chunk
- ❖ audio chunk + RTP header form RTP packet, which is encapsulated in UDP segment

- ❖ RTP header indicates type of audio encoding in each packet
 - sender can change encoding during conference
- ❖ RTP header also contains sequence numbers, timestamps

RTP and QoS

- ❖ RTP does *not* provide any mechanism to ensure timely data delivery or other QoS guarantees
- ❖ RTP encapsulation only seen at end systems (*not* by intermediate routers)
 - routers provide best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely matter

RTP header

<i>payload type</i>	<i>sequence number</i>	<i>time stamp</i>	<i>Synchronization Source ID</i>	<i>Miscellaneous fields</i>
---------------------	------------------------	-------------------	----------------------------------	-----------------------------

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

Payload type 0: PCM mu-law, 64 kbps

Payload type 3: GSM, 13 kbps

Payload type 7: LPC, 2.4 kbps

Payload type 26: Motion JPEG

Payload type 31: H.261

Payload type 33: MPEG2 video

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

- ❖ detect packet loss, restore packet sequence

RTP header

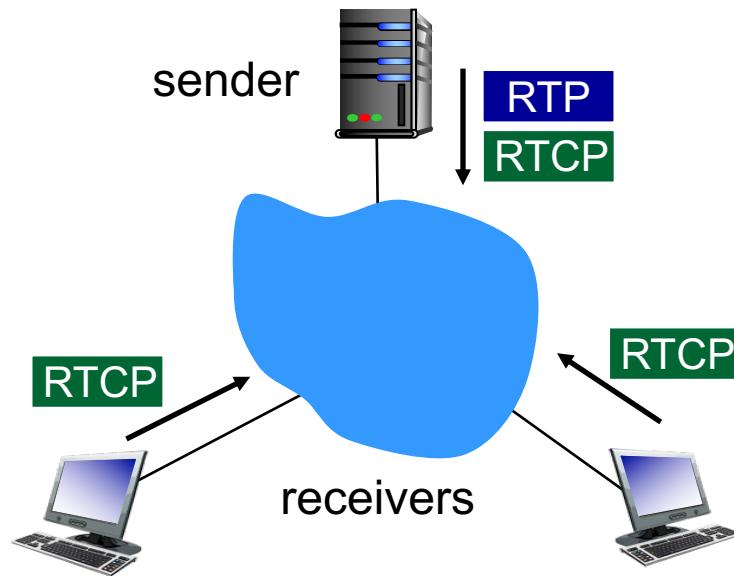
<i>payload type</i>	<i>sequence number</i>	<i>time stamp</i>	<i>Synchronization Source ID</i>	<i>Miscellaneous fields</i>
---------------------	------------------------	-------------------	----------------------------------	-----------------------------

- ❖ ***timestamp field (32 bits long):*** sampling instant of first byte in this RTP data packet
 - for audio, timestamp clock increments by one for each sampling period (each 125 usecs for 8 KHz sampling clock for audio) ; frequency changes depending on type of content
 - if application generates chunks of 160 encoded samples, timestamp increases by 160 for each RTP packet when source is active. Timestamp clock continues to increase at constant rate when source is inactive.
- ❖ ***SSRC field (32 bits long):*** identifies source of RTP stream. Each stream in RTP session has distinct SSRC

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



- ❖ 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: packet types

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

RTCP: stream synchronization

- ❖ RTCP can synchronize different media streams within a RTP session
- ❖ e.g., videoconferencing app: 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 : one sender, sending video at 2 Mbps

- ❖ RTCP attempts to limit RTCP 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

SIP: Session Initiation Protocol [RFC 3261]

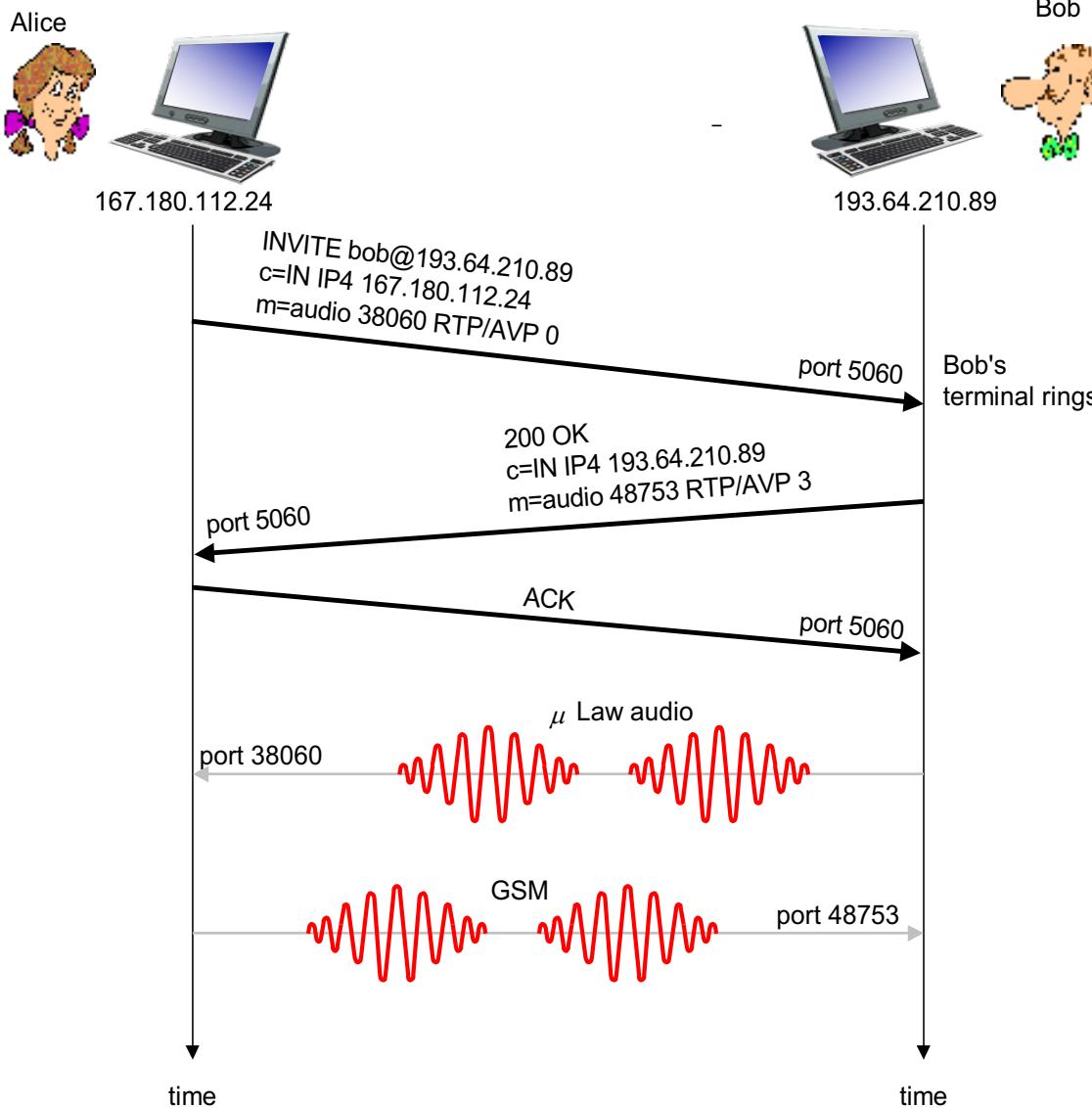
long-term vision:

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

SIP services

- ❖ SIP provides mechanisms for call setup:
 - for caller to let callee know she wants to establish a call
 - so caller, callee can agree on media type, encoding
 - to end call
- ❖ determine current IP address of callee:
 - maps mnemonic identifier to current IP address
- ❖ call management:
 - add new media streams during call
 - change encoding during call
 - invite others
 - transfer, hold calls

Example: setting up call to known IP address



- ❖ Alice's SIP invite message indicates her port number, IP address, encoding she prefers to receive (PCM μ law)

- ❖ Bob's 200 OK message indicates his port number, IP address, preferred encoding (GSM)

- ❖ SIP messages can be sent over TCP or UDP; here sent over RTP/UDP

- ❖ default SIP port number is 5060

Setting up a call (more)

- ❖ codec negotiation:
 - suppose Bob doesn't have PCM µlaw encoder
 - Bob will instead reply with 606 Not Acceptable Reply, listing his encoders. Alice can then send new INVITE message, advertising different encoder
- ❖ rejecting a call
 - Bob can reject with replies "busy," "gone," "payment required," "forbidden"
- ❖ media can be sent over RTP or some other protocol

Example of SIP message

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

Notes:

- ❖ HTTP message syntax
- ❖ sdp = session description protocol
- ❖ Call-ID is unique for every call

- ❖ Here we don't know Bob's IP address
 - intermediate SIP servers needed
- ❖ Alice sends, receives SIP messages using SIP default port 506
- ❖ Alice specifies in header that SIP client sends, receives SIP messages over UDP

Name translation, user location

- ❖ caller wants to call callee, but only has callee's name or e-mail address.
- ❖ need to get IP address of callee's current host:
 - user moves around
 - DHCP protocol
 - user has different IP devices (PC, smartphone, car device)
- ❖ result can be based on:
 - time of day (work, home)
 - caller (don't want boss to call you at home)
 - status of callee (calls sent to voicemail when callee is already talking to someone)

SIP registrar

- ❖ one function of SIP server: *registrar*
- ❖ when Bob starts SIP client, client sends SIP REGISTER message to Bob's registrar server

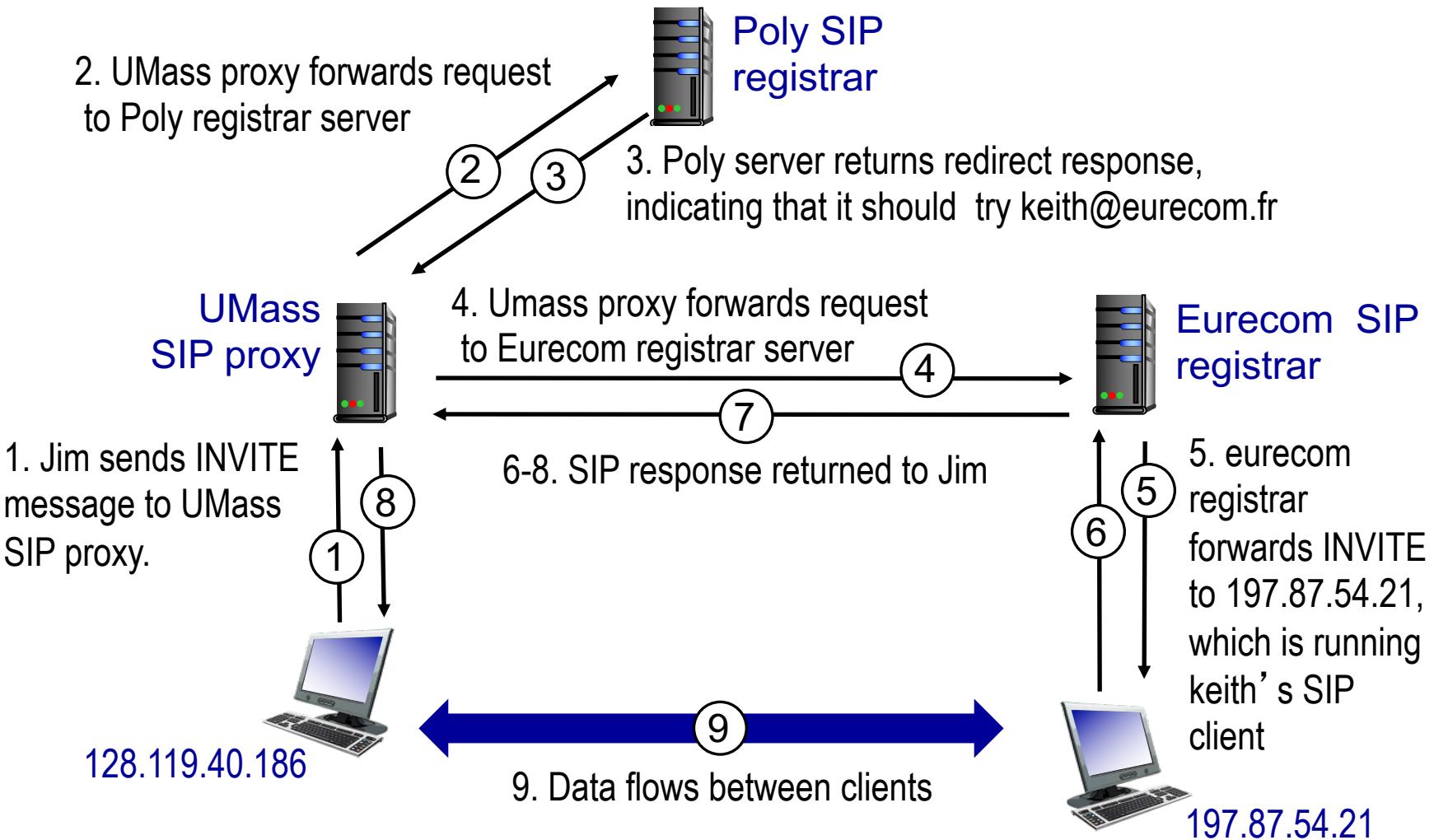
register message:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

SIP proxy

- ❖ another function of SIP server: *proxy*
- ❖ Alice sends invite message to her proxy server
 - contains address `sip:bob@domain.com`
 - proxy responsible for routing SIP messages to callee, possibly through multiple proxies
- ❖ Bob sends response back through same set of SIP proxies
- ❖ proxy returns Bob's SIP response message to Alice
 - contains Bob's IP address
- ❖ SIP proxy analogous to local DNS server plus TCP setup

SIP example: `jim@umass.edu` calls `keith@poly.edu`



Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming stored video

7.3 voice-over-IP

7.4 protocols for real-time conversational
applications

7.5 network support for multimedia

Network support for multimedia

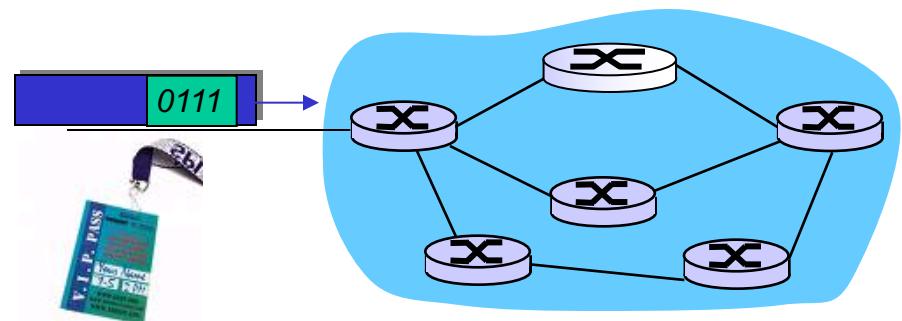
Approach	Granularity	Guarantee	Mechanisms	Complex	Deployed?
Making best of best effort service	All traffic treated equally	None or soft	No network support (all at application)	low	everywhere
Differentiated service	Traffic “class”	None or soft	Packet market, scheduling, policing.	med	some
Per-connection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

Dimensioning best effort networks

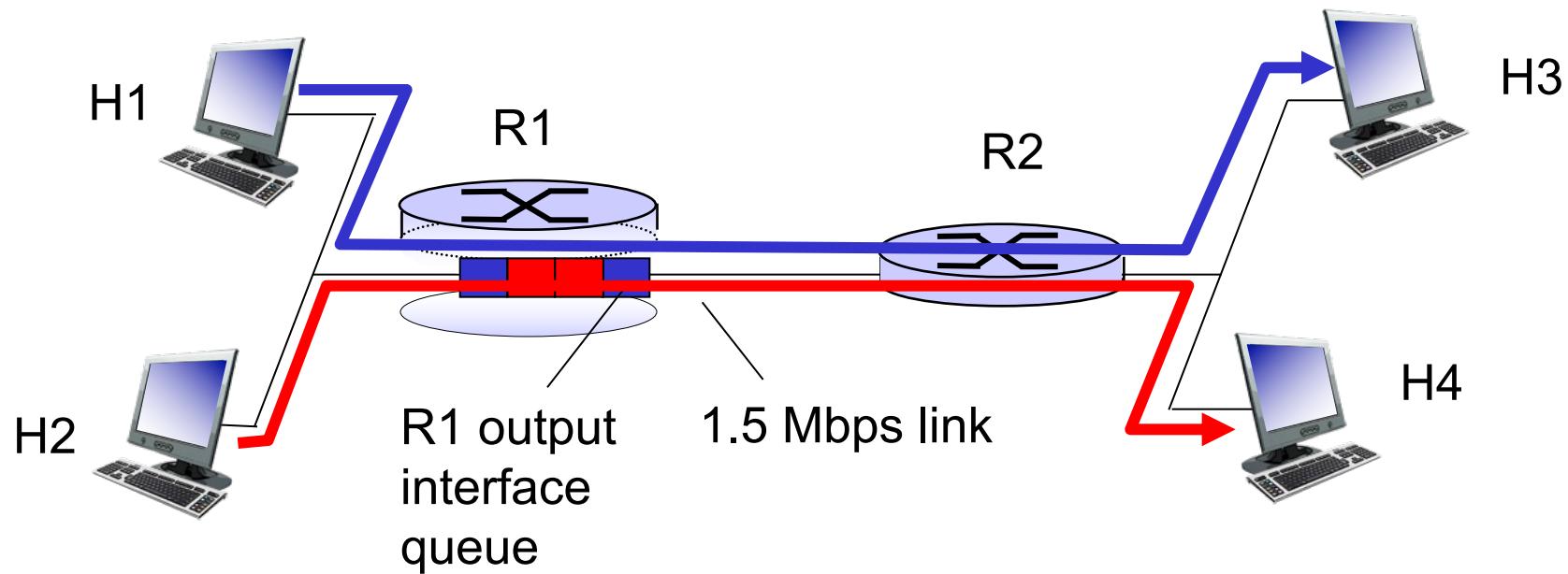
- ❖ *approach:* deploy enough link capacity so that congestion doesn't occur, multimedia traffic flows without delay or loss
 - low complexity of network mechanisms (use current “best effort” network)
 - high bandwidth costs
- ❖ challenges:
 - *network dimensioning:* how much bandwidth is “enough?”
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

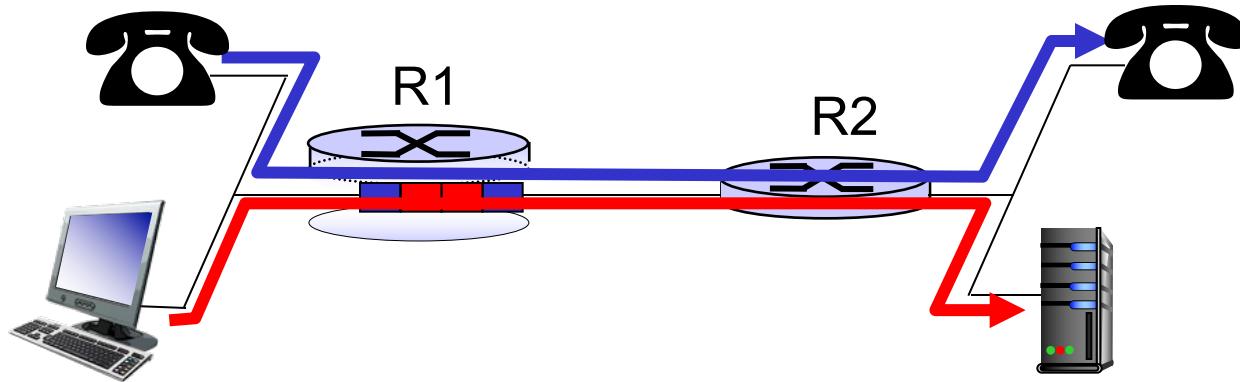


Multiple classes of service: scenario



Scenario I: 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

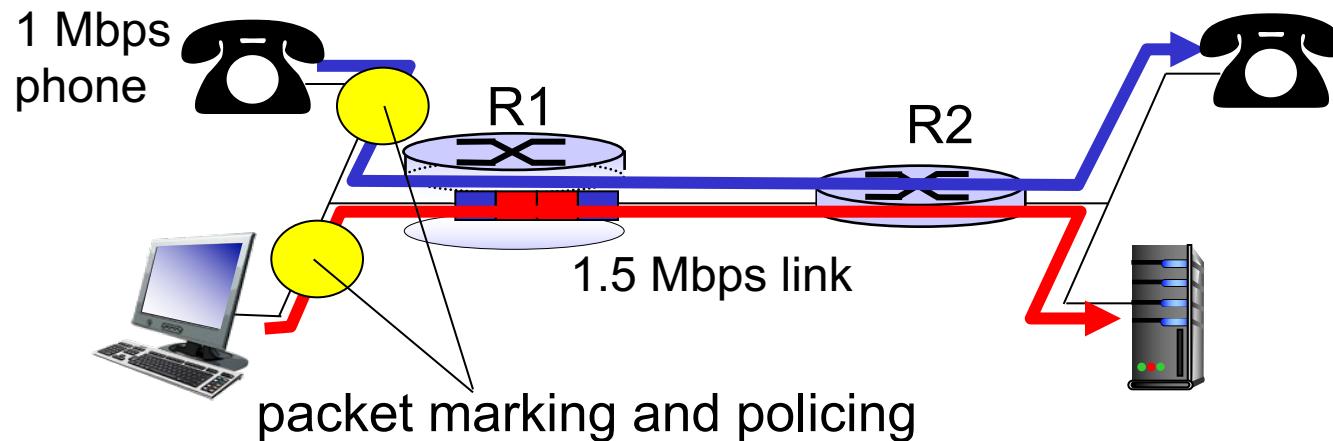


Principle I

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

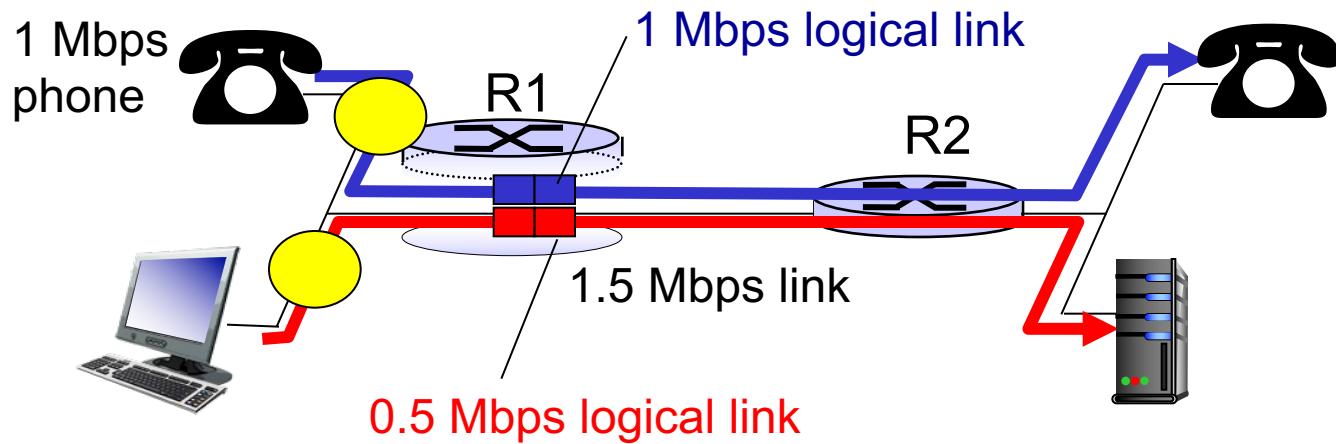


Principle 2

provide protection (isolation) for one class from others

Principles for QOS guarantees (more)

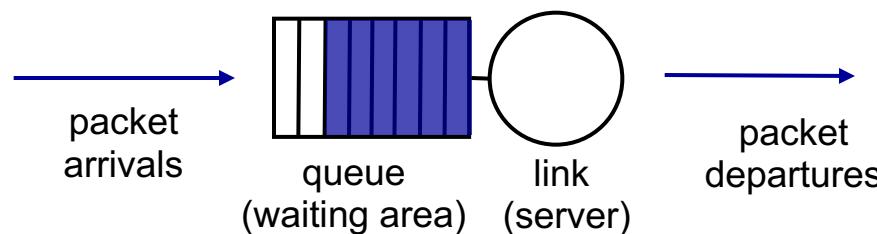
- ❖ allocating *fixed* (non-sharable) bandwidth to flow:
inefficient use of bandwidth if flows doesn't use its allocation



Principle 3
while providing isolation, it is desirable to use resources as efficiently as possible

Scheduling and policing mechanisms

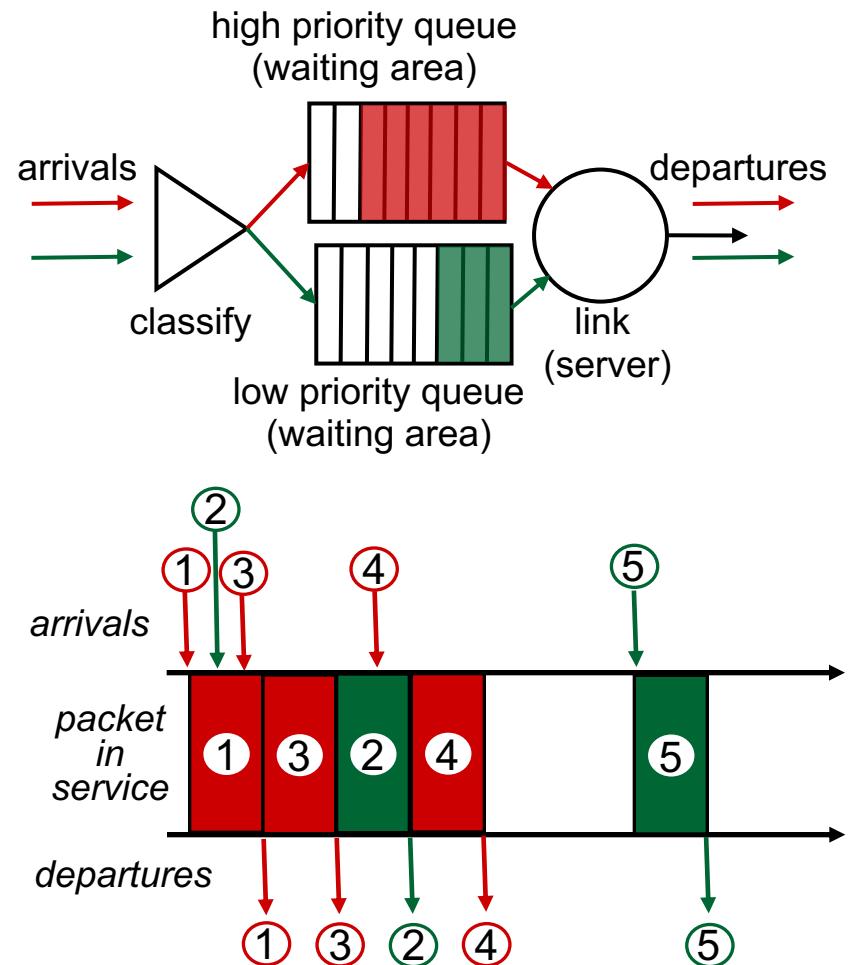
- ❖ *scheduling*: choose next packet to send on link
- ❖ *FIFO (first in first out) scheduling*: send in order of arrival to queue
 - real-world example?
 - *discard policy*: if packet arrives to full queue: who to discard?
 - *tail drop*: drop arriving packet
 - *priority*: drop/remove on priority basis
 - *random*: drop/remove randomly



Scheduling policies: priority

priority scheduling: send highest priority queued packet

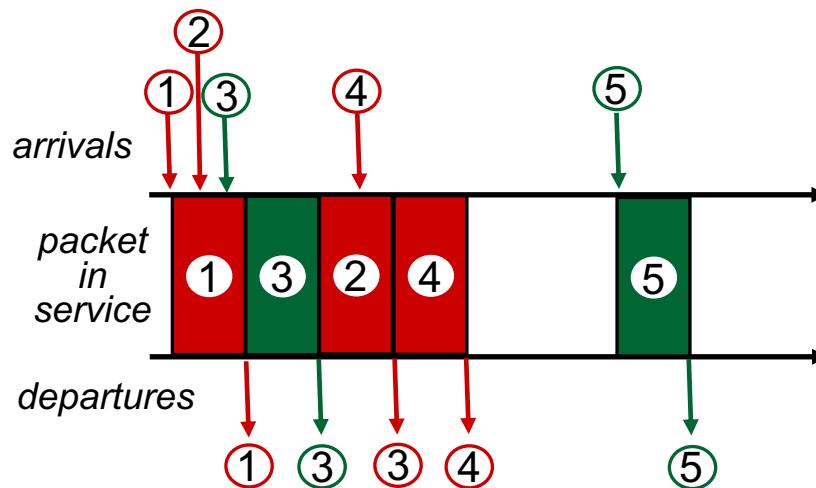
- ❖ multiple classes, with different priorities
 - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
 - real world example?



Scheduling policies: still more

Round Robin (RR) scheduling:

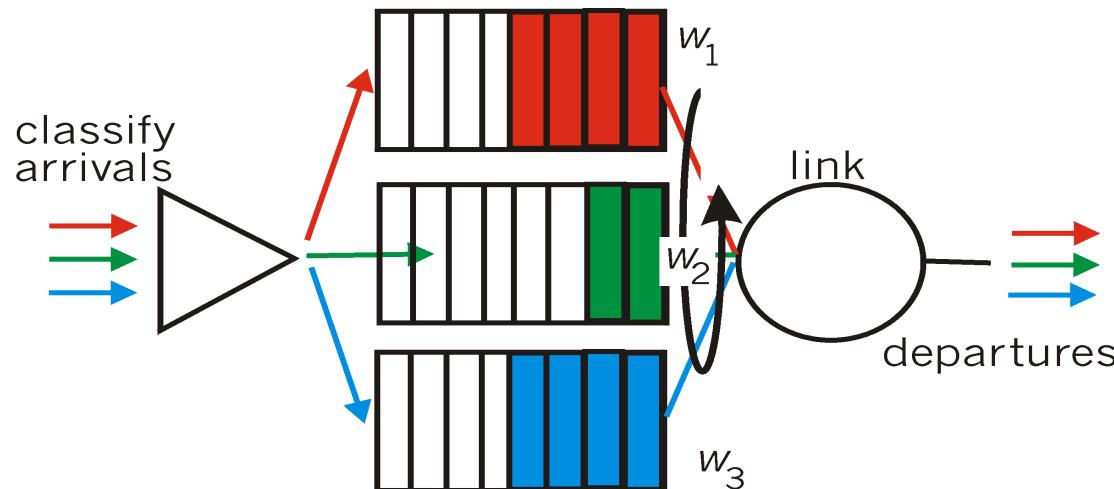
- ❖ multiple classes
- ❖ cyclically scan class queues, sending one complete packet from each class (if available)
- ❖ real world example?



Scheduling policies: still more

Weighted Fair Queuing (WFQ):

- ❖ generalized Round Robin
- ❖ each class gets weighted amount of service in each cycle
- ❖ real-world example?



Policing mechanisms

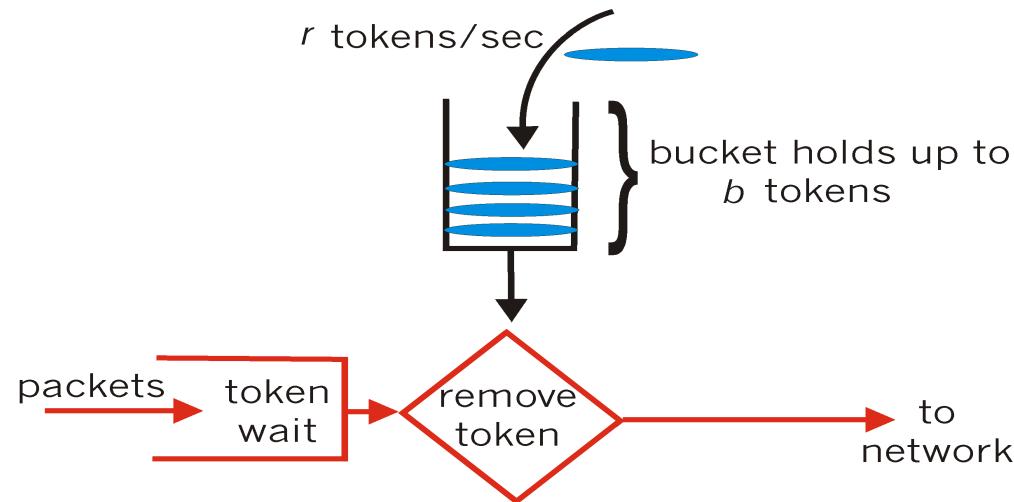
goal: limit traffic to not exceed declared parameters

Three common-used criteria:

- ❖ *(long term) average rate:* how many pkts can be sent per unit time (in the long run)
 - crucial question: what is the interval length: 100 packets per sec or 6000 packets per min have same average!
- ❖ *peak rate:* e.g., 6000 pkts per min (ppm) avg.; 1500 ppm peak rate
- ❖ *(max.) burst size:* max number of pkts sent consecutively (with no intervening idle)

Policing mechanisms: implementation

token bucket: limit input to specified *burst size* and *average rate*

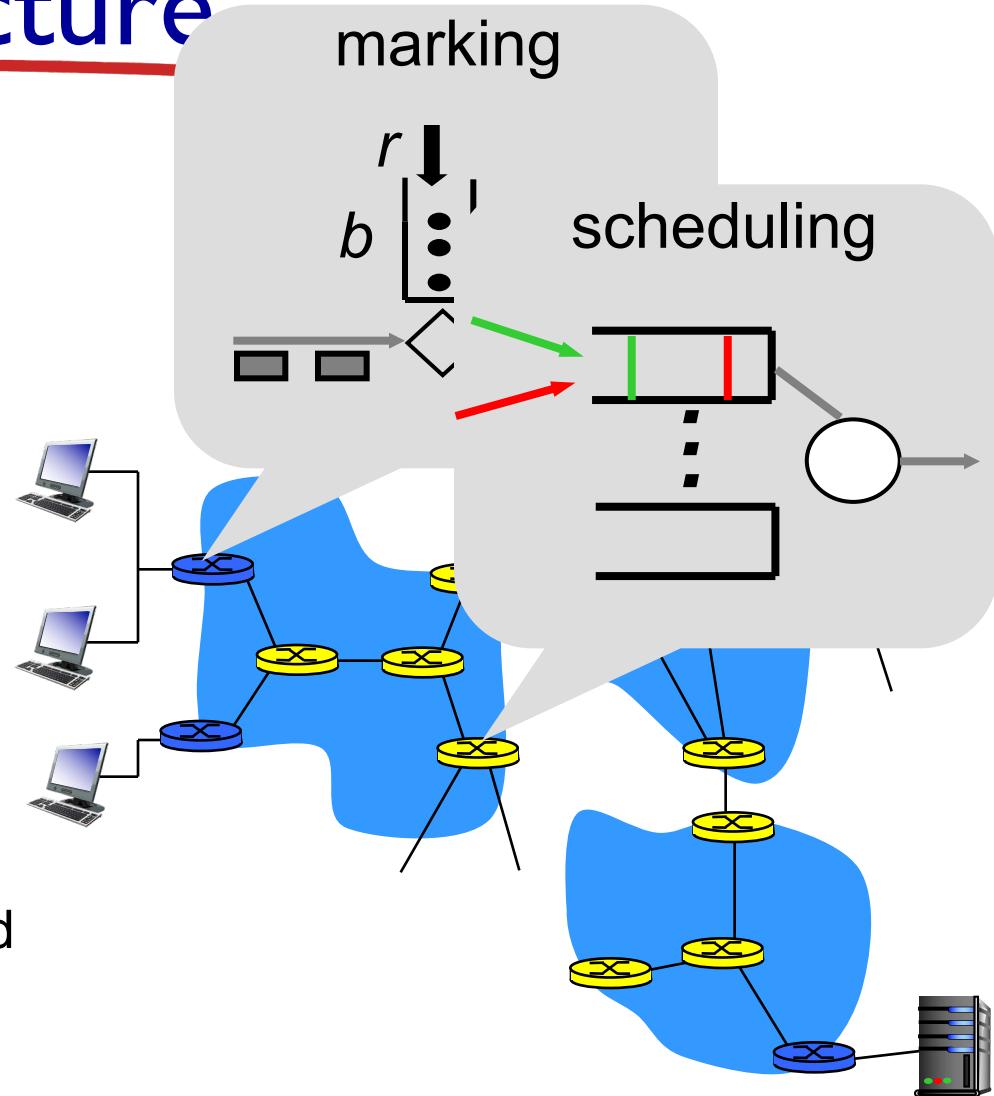


- ❖ bucket can hold b tokens (max burst size)
- ❖ tokens generated at rate r token/sec unless bucket full
- ❖ *over interval of length t: number of packets admitted less than or equal to $(r t + b)$*

Diffserv architecture

edge router: 

- ❖ *per-flow* traffic management
- ❖ marks packets as **in-profile** and **out-profile**

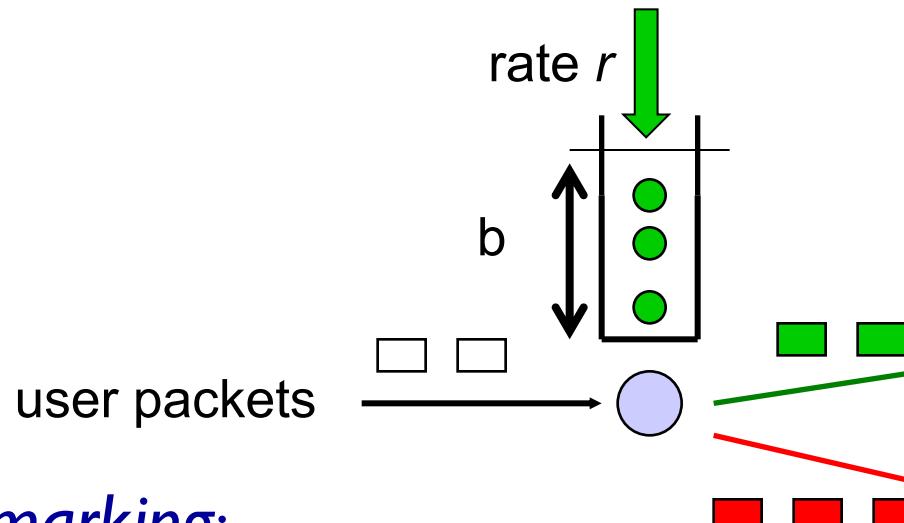


core router: 

- ❖ *per class* traffic management
- ❖ buffering and scheduling based on **marking** at edge
- ❖ preference given to **in-profile** packets over **out-of-profile** packets

Edge-router packet marking

- ❖ *profile*: pre-negotiated rate r , bucket size b
- ❖ packet marking at edge based on *per-flow* profile

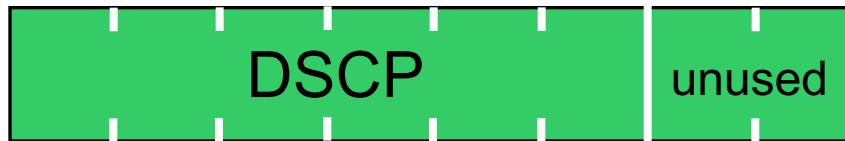


possible use of marking:

- ❖ class-based marking: packets of different classes marked differently
- ❖ intra-class marking: conforming portion of flow marked differently than non-conforming one

Diffserv packet marking: details

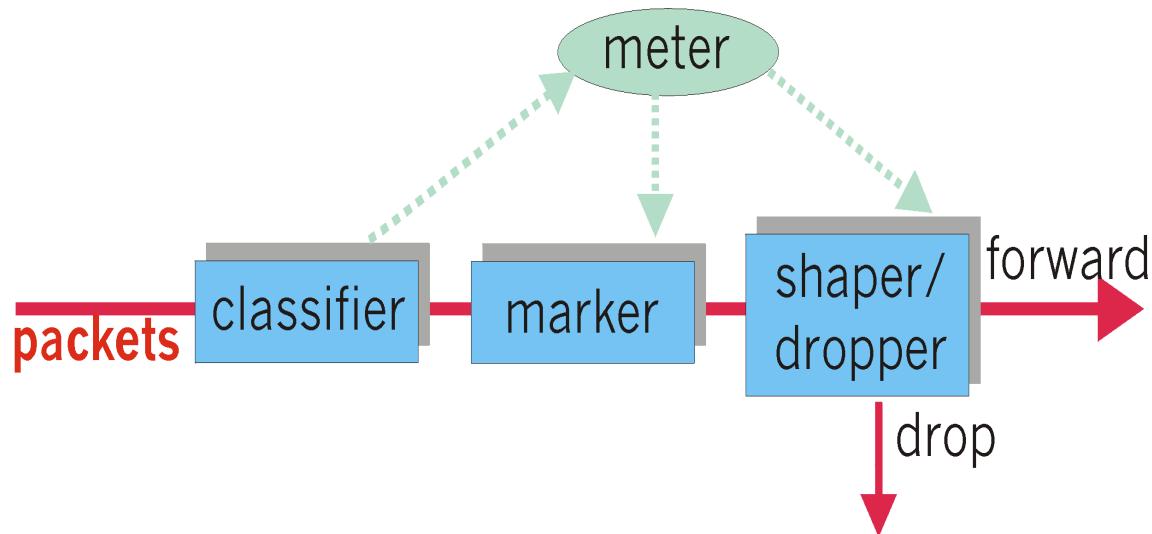
- ❖ packet is marked in the Type of Service (TOS) in IPv4, and Traffic Class in IPv6
- ❖ 6 bits used for Differentiated Service Code Point (DSCP)
 - determine PHB that the packet will receive
 - 2 bits currently unused



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-conforming

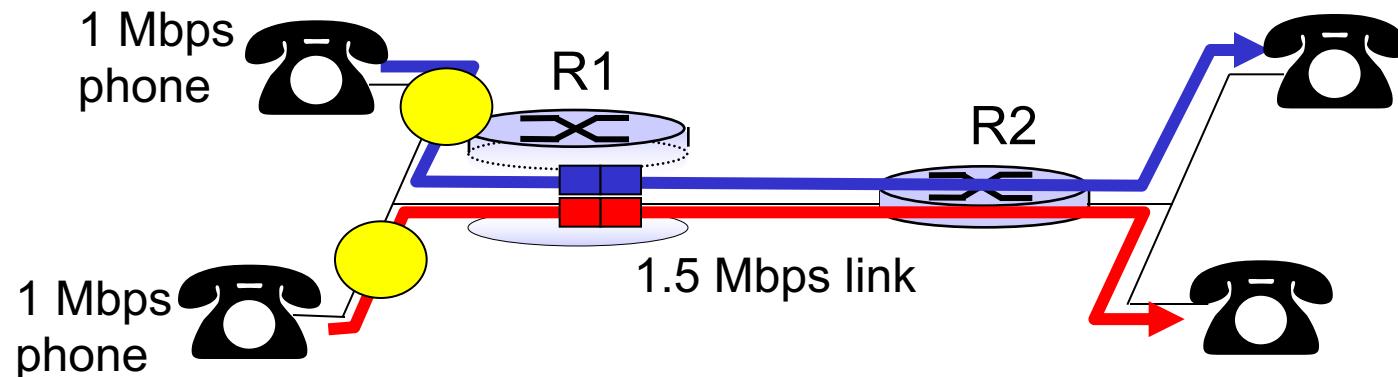


Forwarding Per-hop Behavior (PHB)

- ❖ PHB result in a different *observable* (*measurable*) forwarding performance behavior
- ❖ PHB does *not* specify what mechanisms to use to ensure required PHB performance behavior
- ❖ examples:
 - class A gets x% of outgoing link bandwidth over time intervals of a specified length
 - class A packets leave first before packets from class B

Per-connection QOS guarantees

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



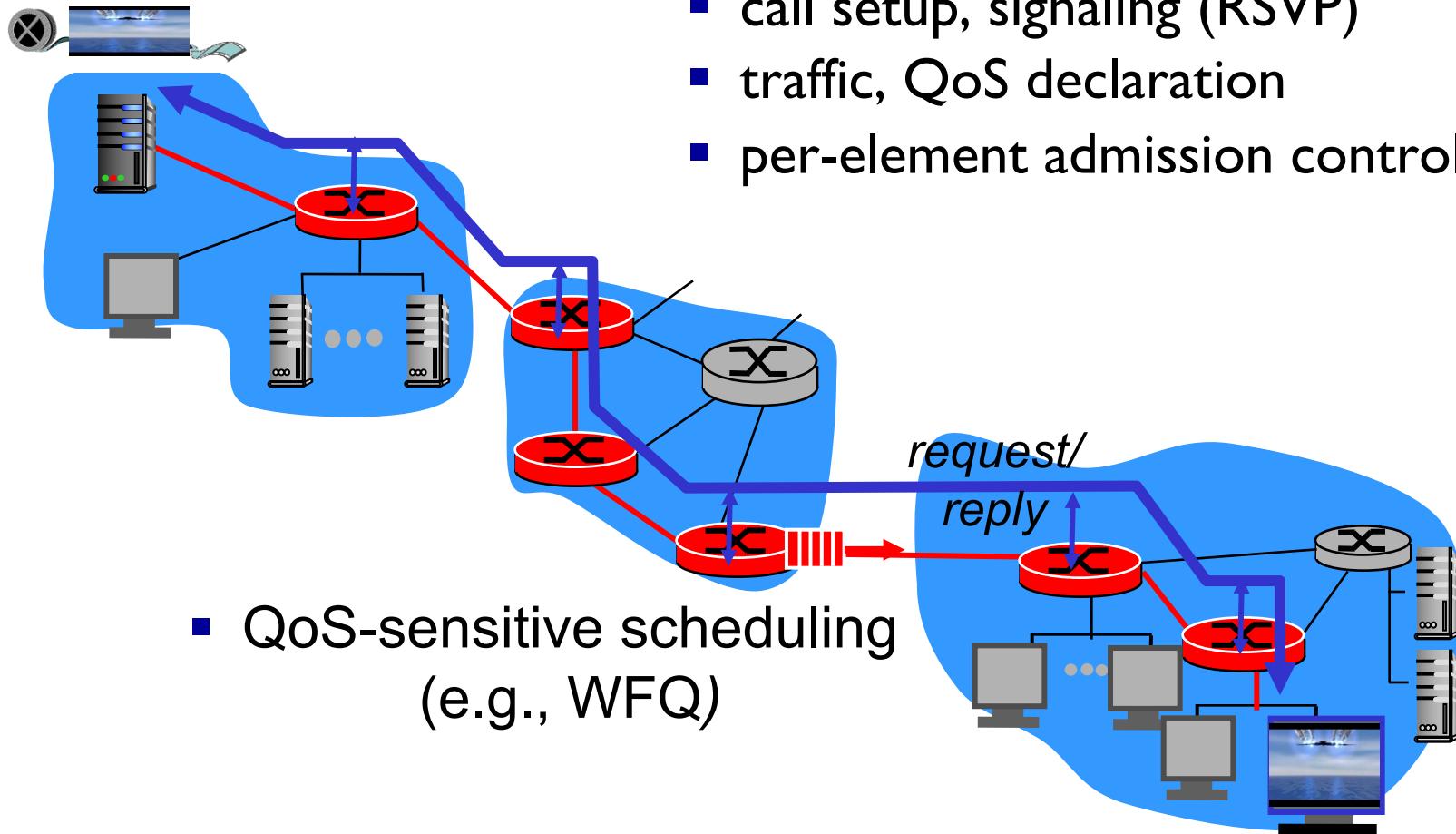
Principle 4

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

QoS guarantee scenario

❖ *resource reservation*

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

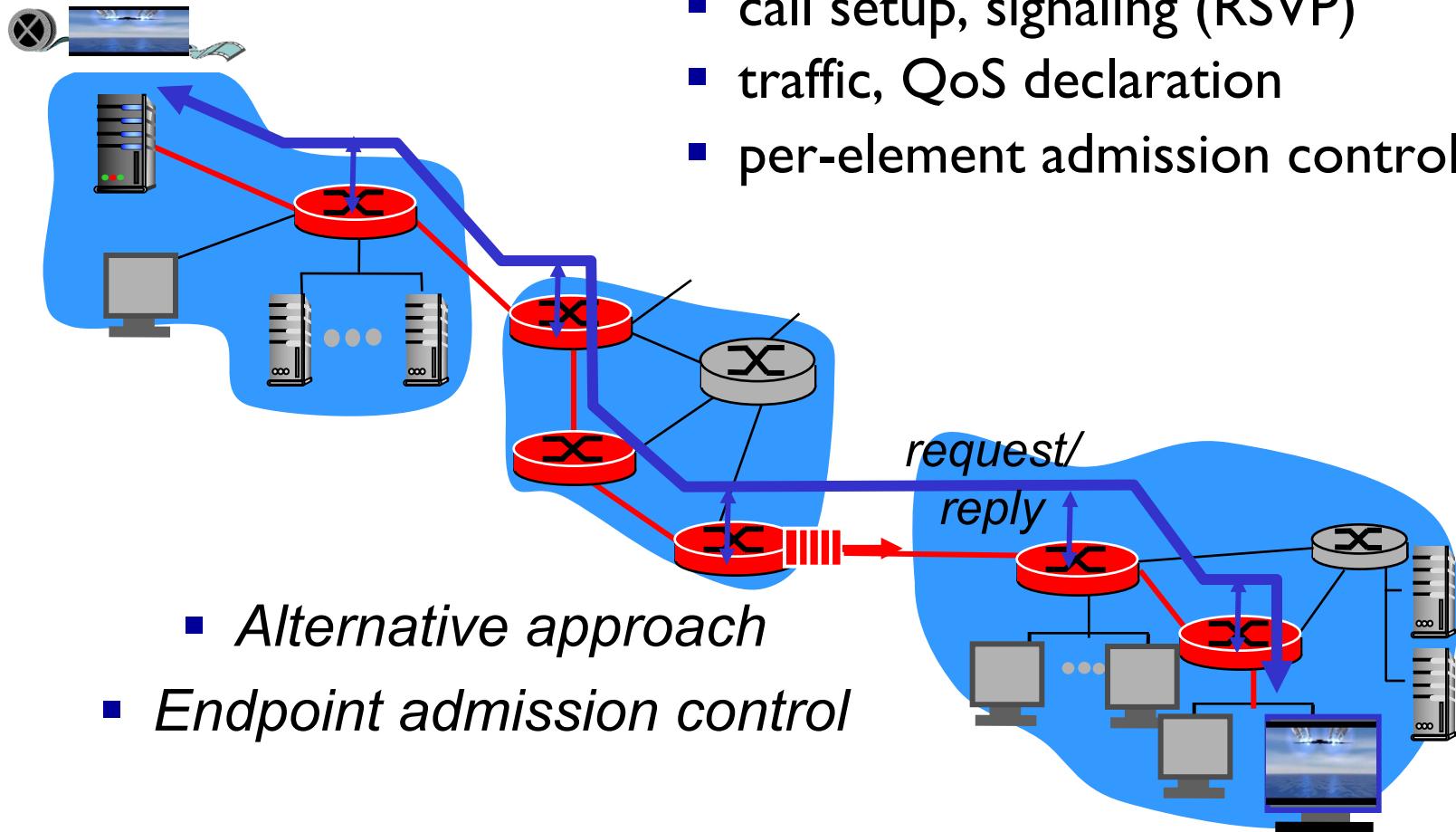


- QoS-sensitive scheduling
(e.g., WFQ)

QoS guarantee scenario

❖ *resource reservation*

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



Multimedia networking: outline

7.1 multimedia networking applications

7.2 streaming stored video

7.3 voice-over-IP

7.4 protocols for *real-time* conversational
applications

7.5 network support for multimedia