

# Chapter 9

## Multimedia

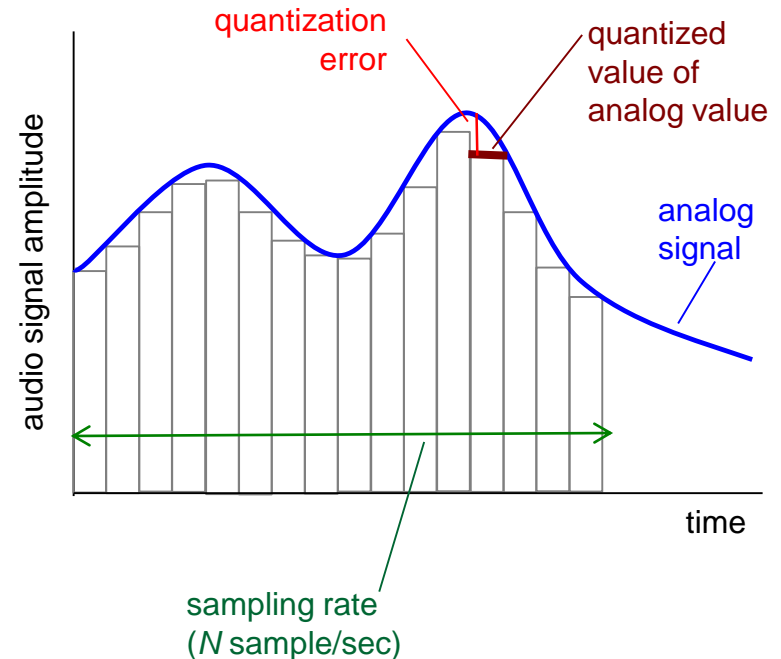
## Networking

# 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

# 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

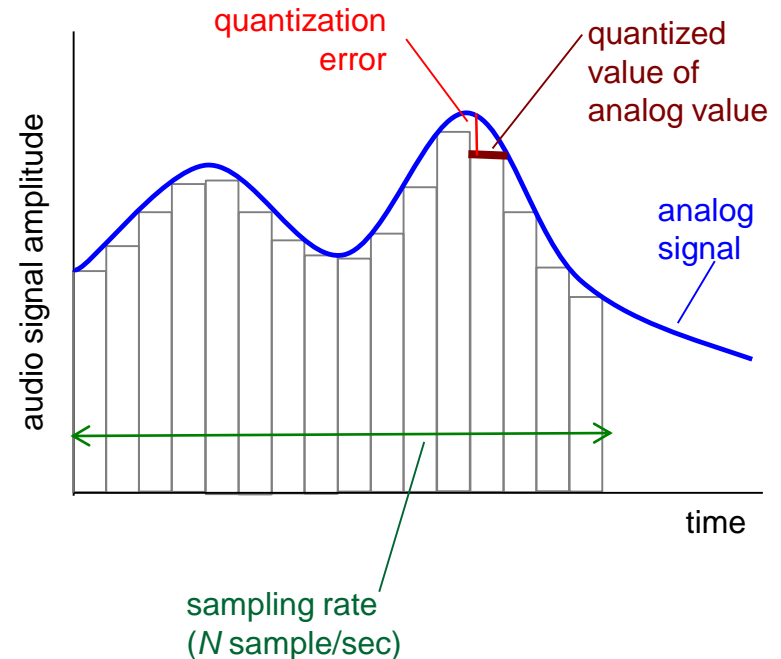


# Multimedia: audio

- 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

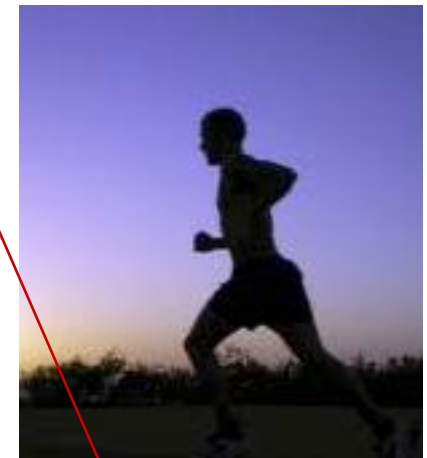
- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy *within* and *between* images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to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

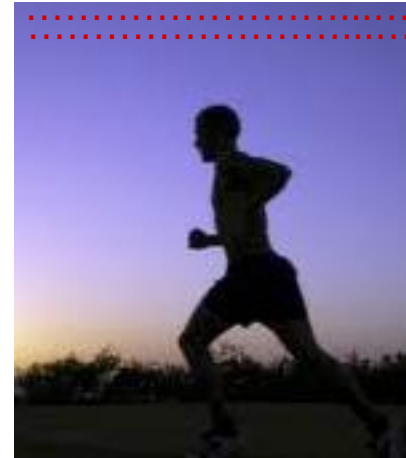


frame  $i+1$

# Multimedia: video

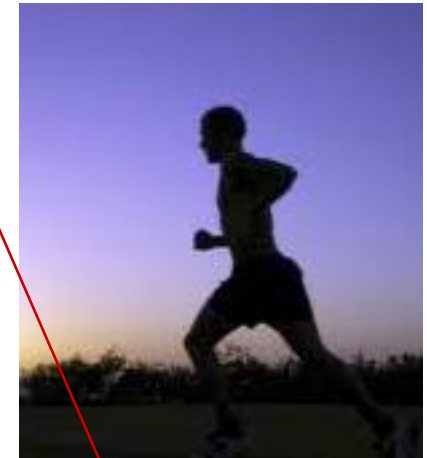
- **CBR: (constant bit rate):**  
video encoding rate fixed
- **VBR: (variable bit rate):**  
video encoding rate changes  
as amount of spatial,  
temporal coding changes
- **examples:**
  - MPEG I (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

*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 (futbol)

# Multimedia networking: outline

9.1 multimedia networking applications

9.2 *streaming stored video*

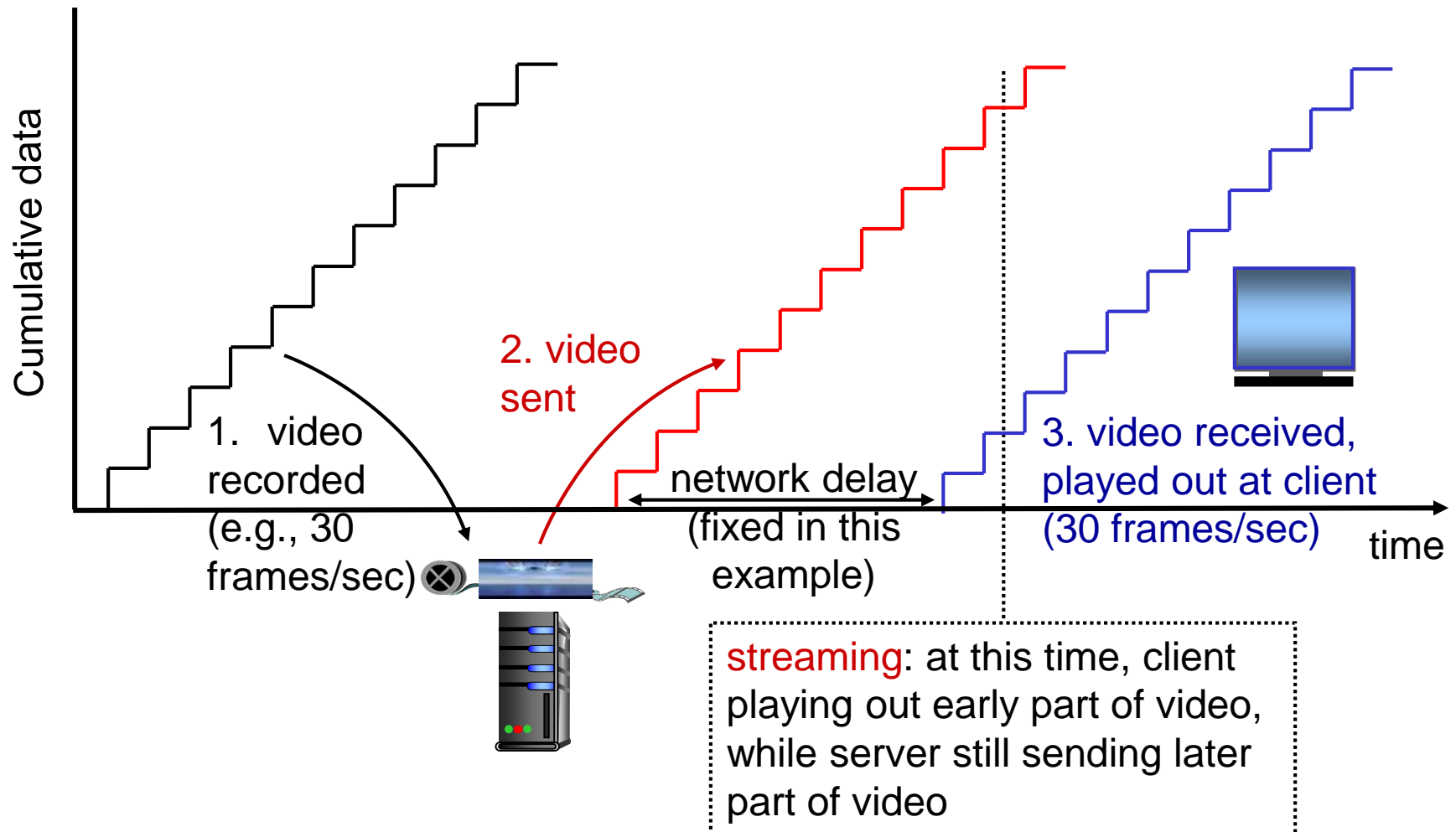
9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia



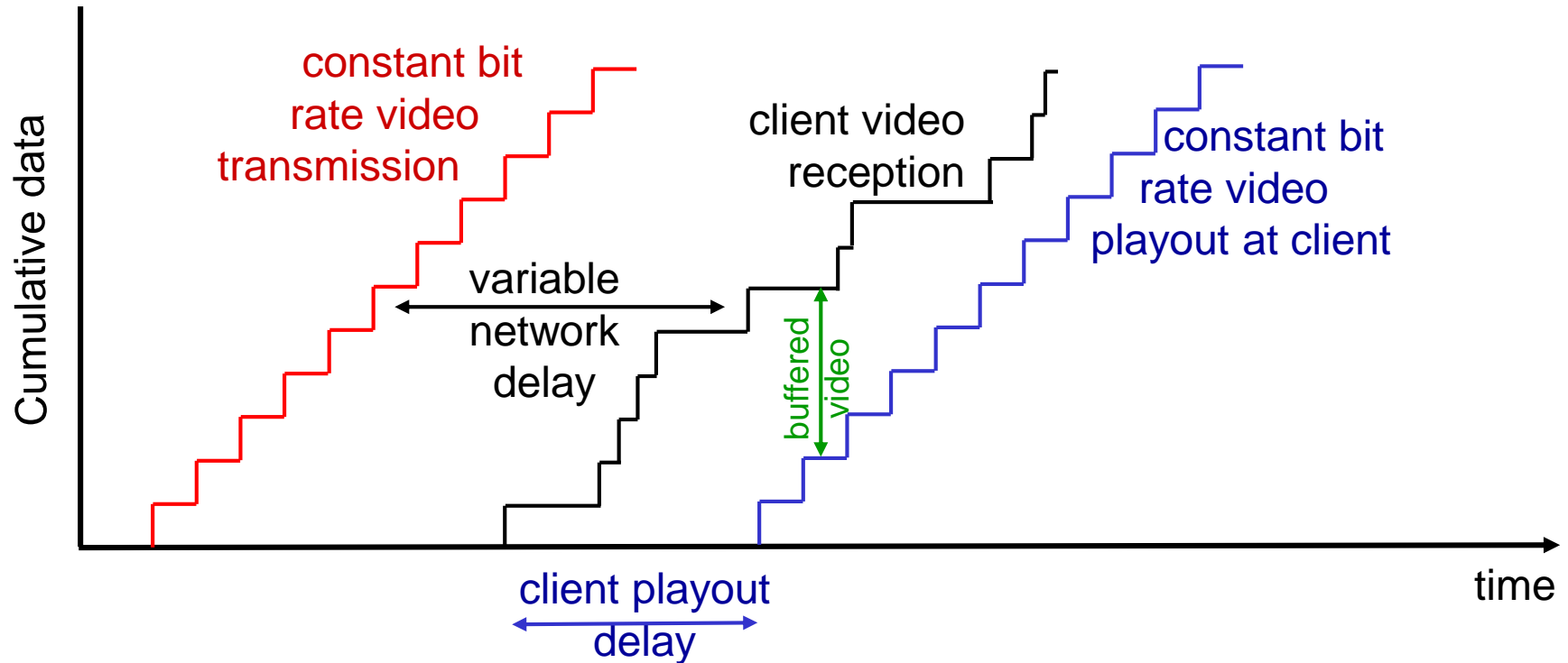
# Streaming stored video:



# Streaming stored video: challenges

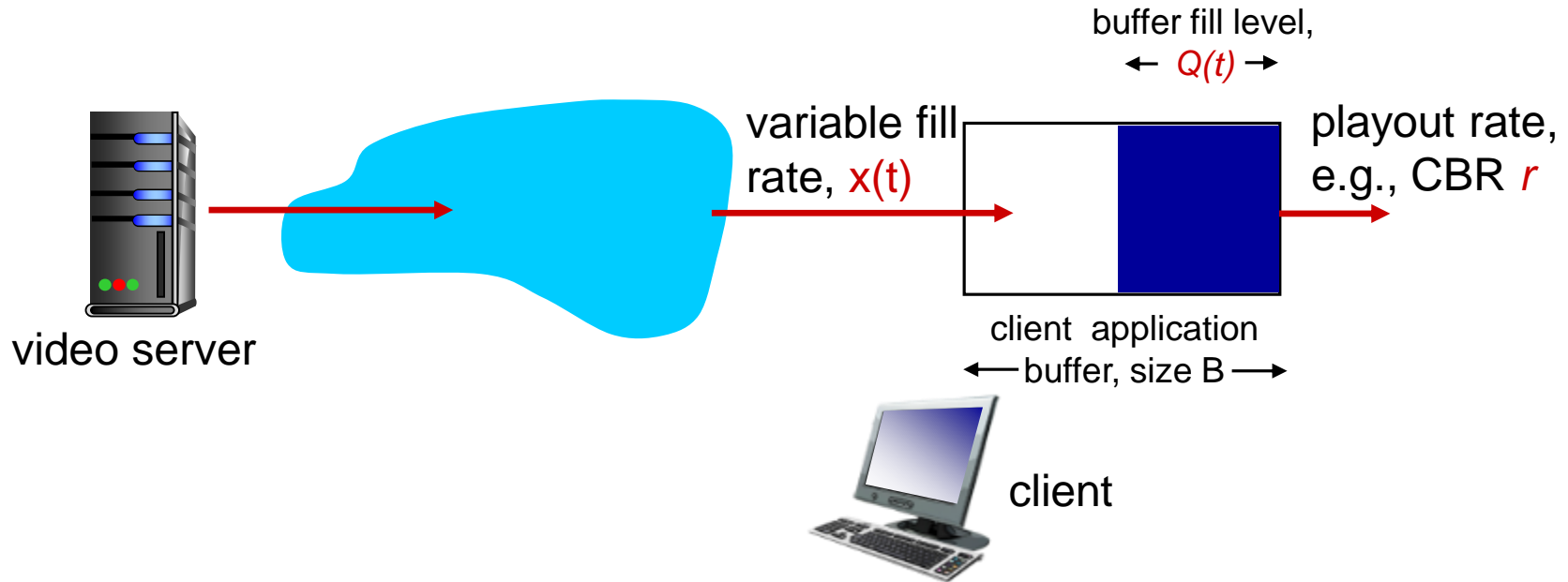
- **continuous playout constraint**: once client playout begins, playback must match original timing
  - ... but **network delays are variable** (jitter), so will need **client-side buffer** to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump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

# Streaming stored video: revisited

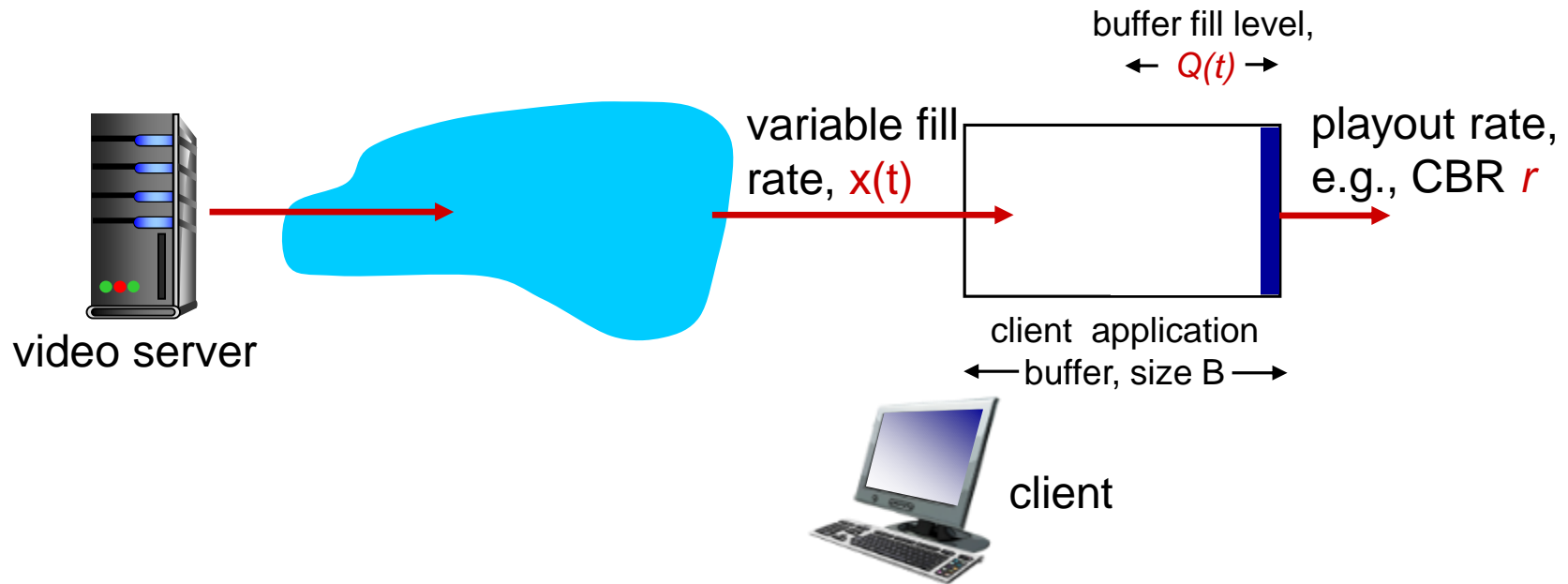


- *client-side buffering and playout delay*: compensate for network-added delay, delay jitter

# Client-side buffering, playout

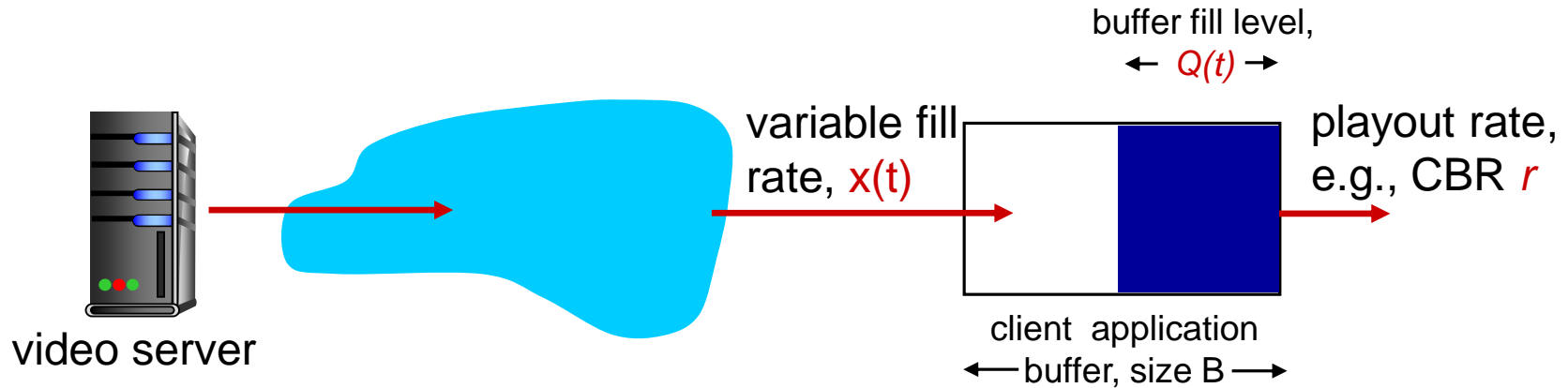


# Client-side buffering, playout



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 ( $r$ ):*

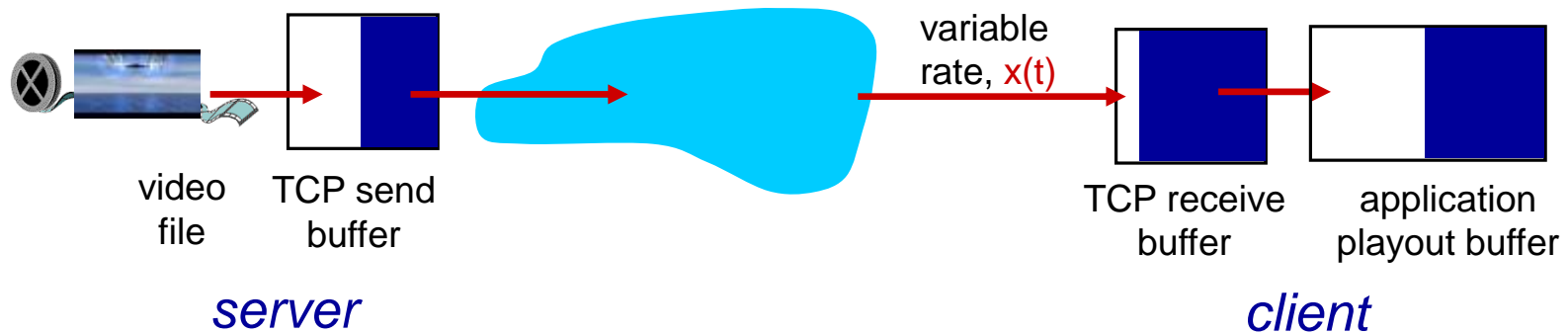
- $\bar{x} < r$ : buffer eventually empties (causing freezing of video playout until buffer again fills)
- $\bar{x} > r$ : buffer will not empty, provided initial playout delay is large enough to absorb variability in  $x(t)$ 
  - *initial playout delay tradeoff*: buffer starvation less likely with larger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



# Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia

# Voice-over-IP (VoIP)

- *VoIP end-end-delay requirement*: needed to maintain “conversational” aspect
  - higher delays noticeable, impair interactivity
  - < 150 msec: good
  - > 400 msec bad
  - includes application-level (packetization, playout), network delays
- *session initialization*: how does callee advertise IP address, port number, encoding algorithms?
- *value-added services*: call forwarding, screening, recording
- *emergency services*: 911

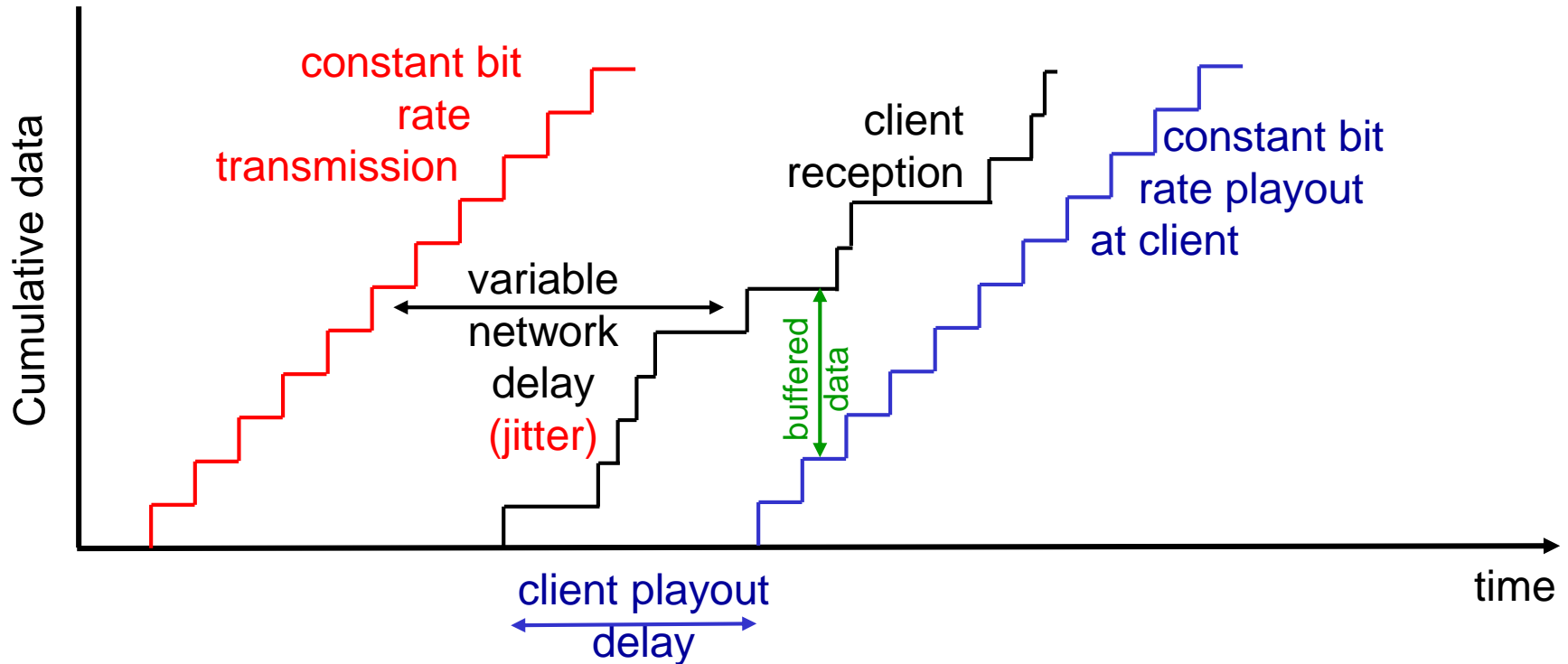
# VoIP characteristics

- speaker's audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
  - pkts generated only during talk spurts
  - 20 msec chunks at 8 Kbytes/sec: 160 bytes of data
- application-layer header added to each chunk
- chunk+header encapsulated into UDP or TCP segment
- application sends segment into socket every 20 msec during talkspurt

# VoIP: packet loss, delay

- **network loss:** IP datagram lost due to network congestion (router buffer overflow)
- **delay loss:** IP datagram arrives too late for playout at receiver
  - delays: processing, queueing in network; end-system (sender, receiver) delays
  - typical maximum tolerable delay: 400 ms
- **loss tolerance:** depending on voice encoding, loss concealment, packet loss rates between 1% and 10% can be tolerated

# Delay jitter



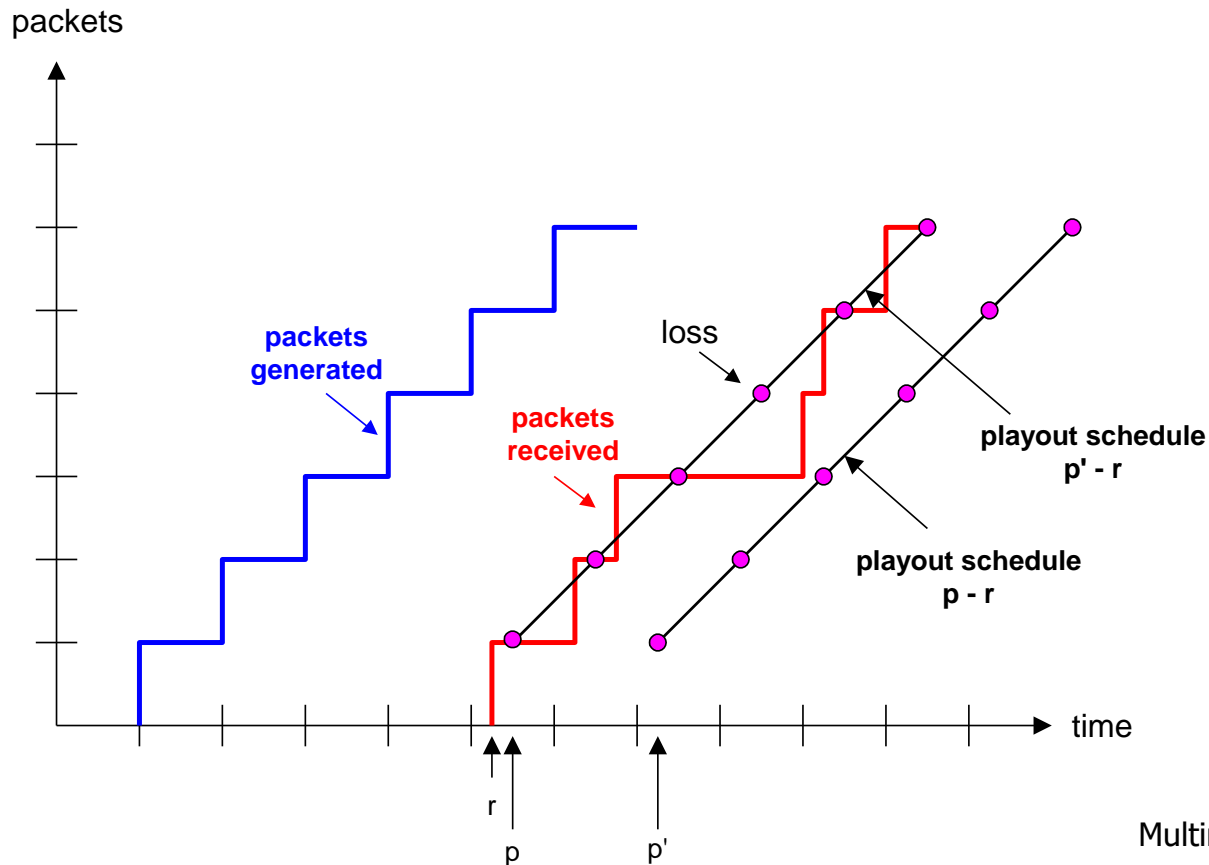
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

# VoIP: fixed playout delay

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

# 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.

# 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)

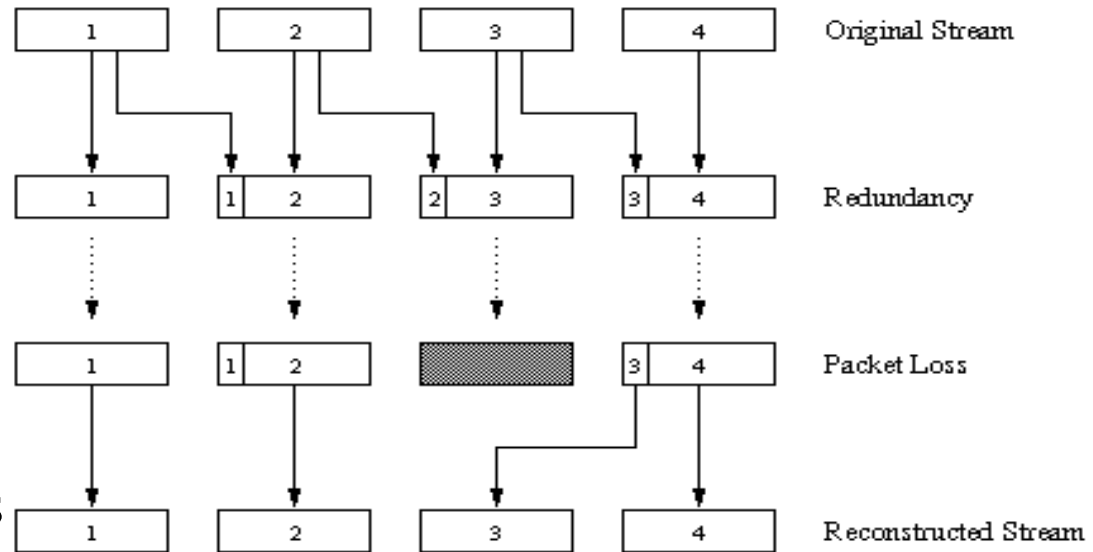
## *simple FEC*

- 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

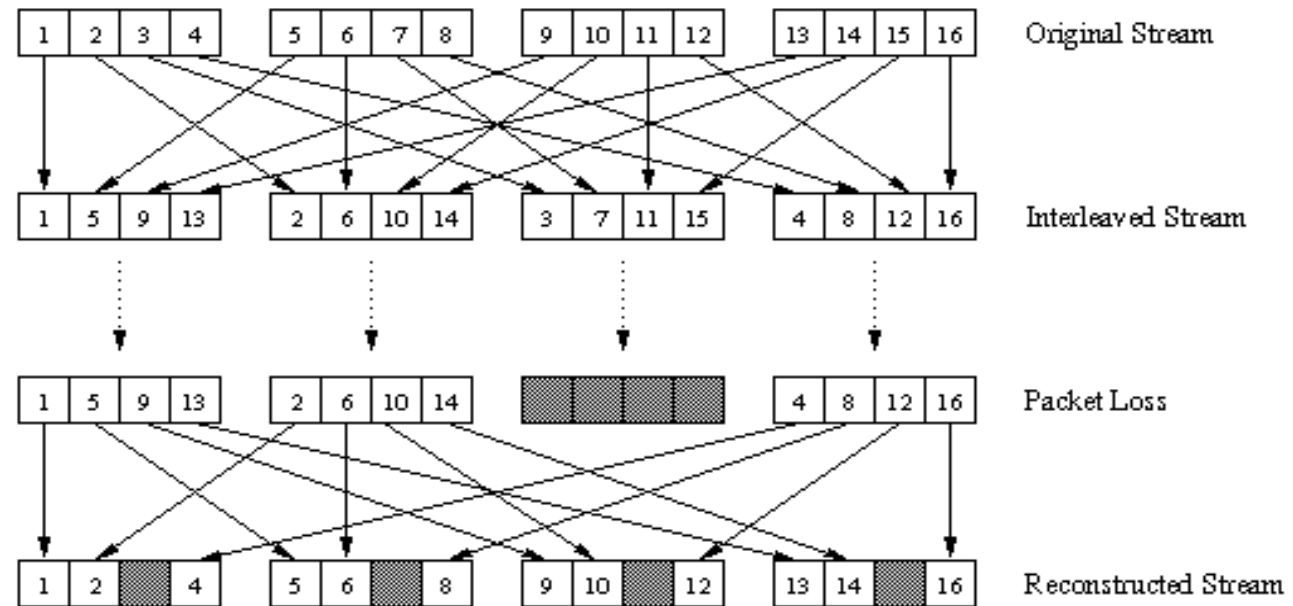
# VoiP: recovery from packet loss (2)

## 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



# VoiP: recovery from packet loss (3)

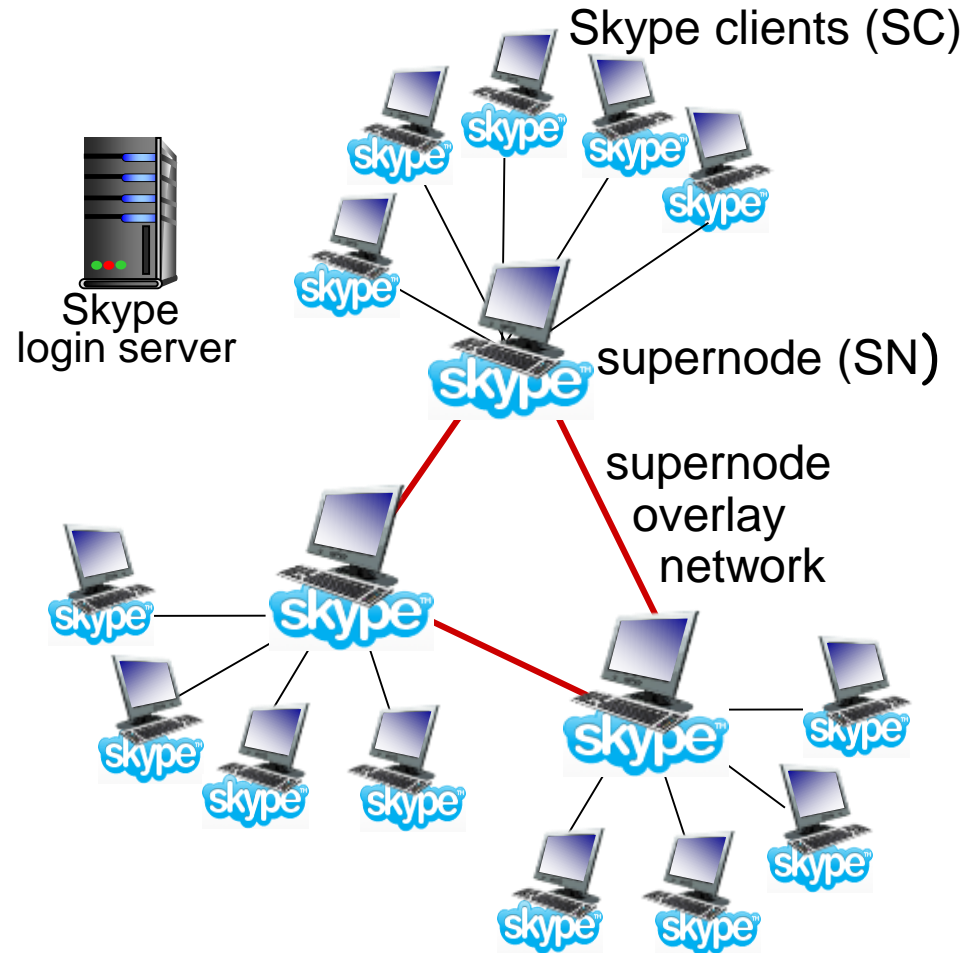


## *interleaving to conceal loss:*

- 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

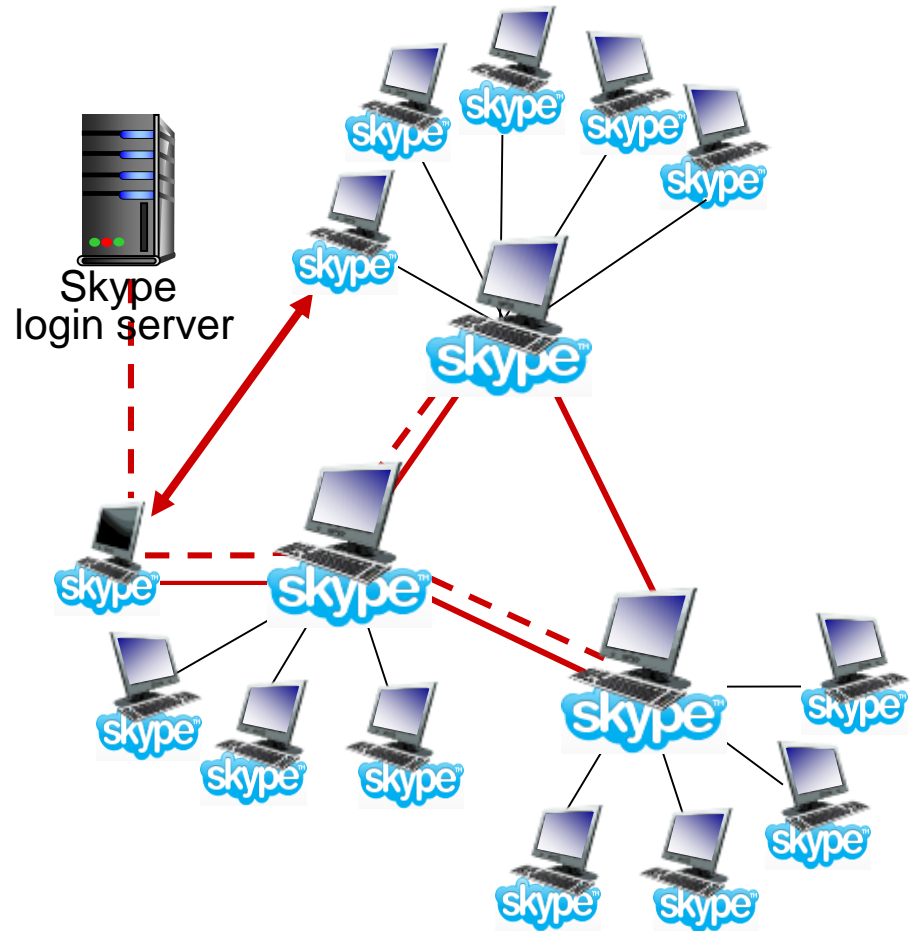
- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:
  - **clients:** Skype peers connect directly to each other for VoIP call
  - **super nodes (SN):** Skype peers with special functions
  - **overlay network:** among SNs to locate SCs
  -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



# P2P voice-over-IP: Skype

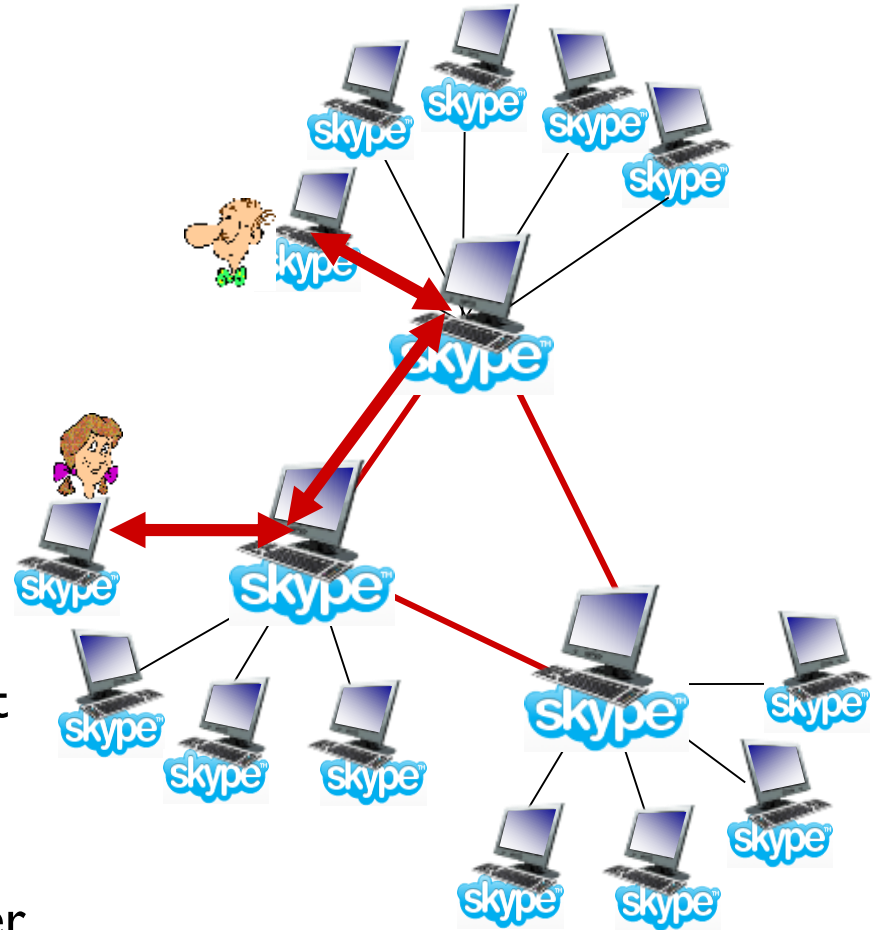
## Skype client operation:

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



# 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

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

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

9.5 network support for multimedia

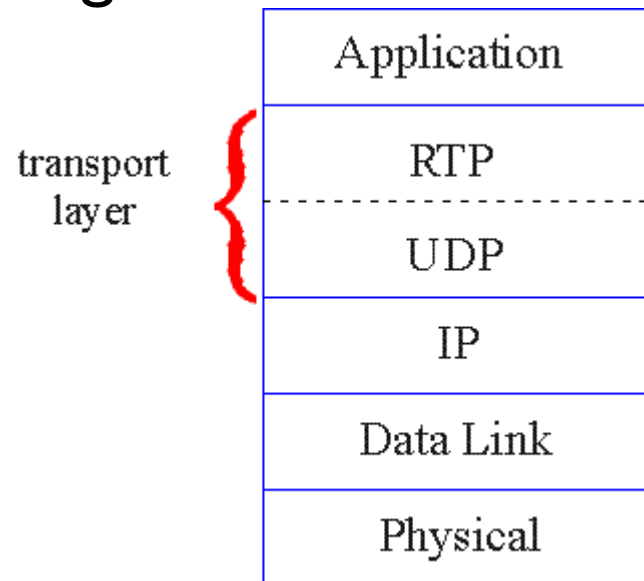
# Real-Time Protocol (RTP)

- 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 (e.g., each 125 usecs for 8 KHz sampling clock)
  - 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

# RTSP/RTP programming assignment

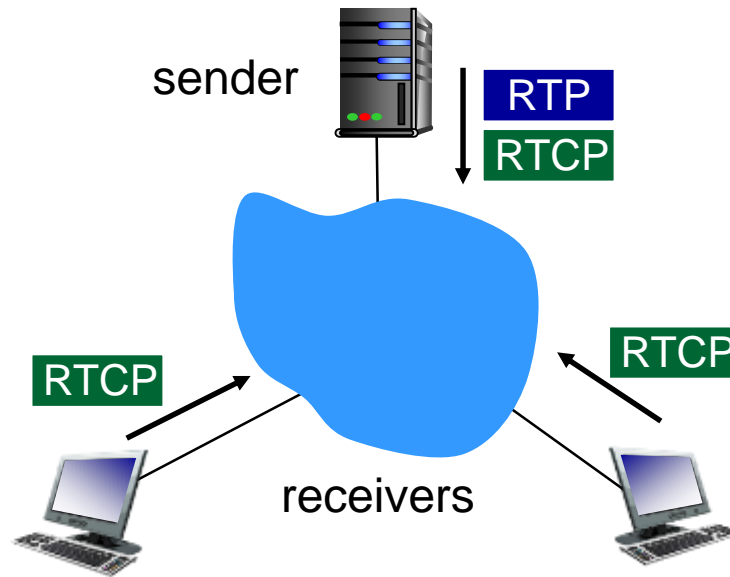
- build a server that encapsulates stored video frames into RTP packets
  - grab video frame, add RTP headers, create UDP segments, send segments to UDP socket
  - include seq numbers and time stamps
  - client RTP provided for you
- also write client side of RTSP
  - issue play/pause commands
  - server RTSP provided for you



# 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

# Multimedia networking: outline

9.1 multimedia networking applications

9.2 streaming *stored* video

9.3 voice-over-IP

9.4 protocols for *real-time* conversational applications

9.5 network support for multimedia

# Network support for multimedia

Approach	Granularity	Guarantee	Mechanisms	Complex	Deployed?
Making best of best effort service	All traffic treated equally	None or soft	No network support (all at application)	low	everywhere
Differentiated service	Traffic “class”	None 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

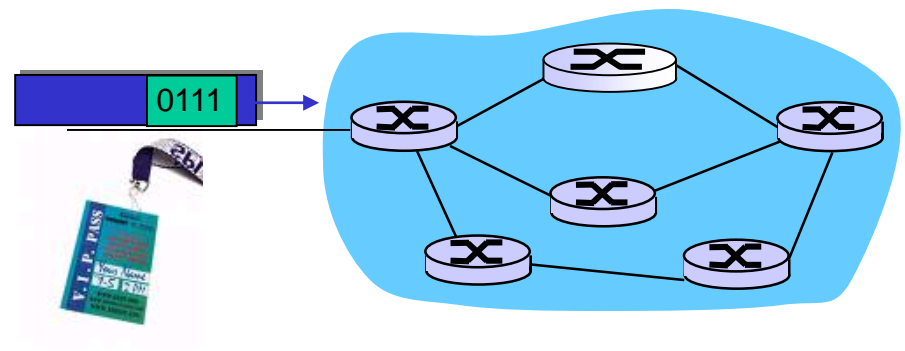
# Dimensioning best effort networks

- *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)

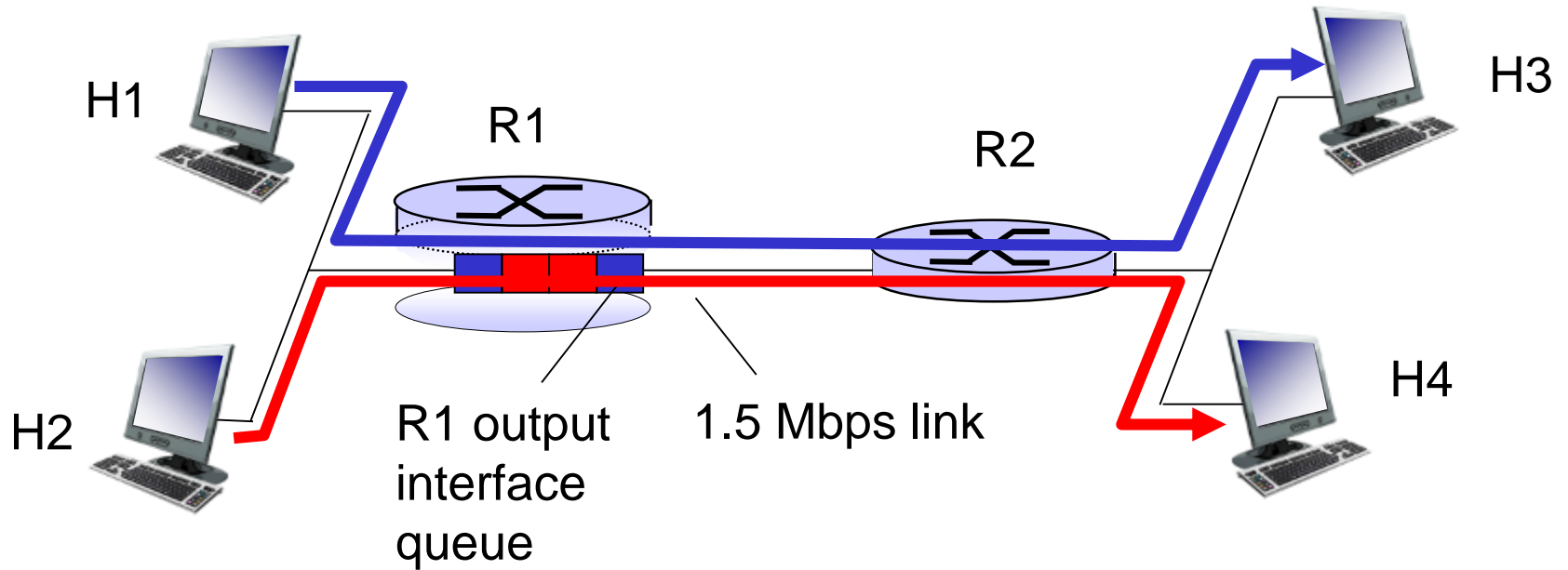


# Providing multiple classes of service

- 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

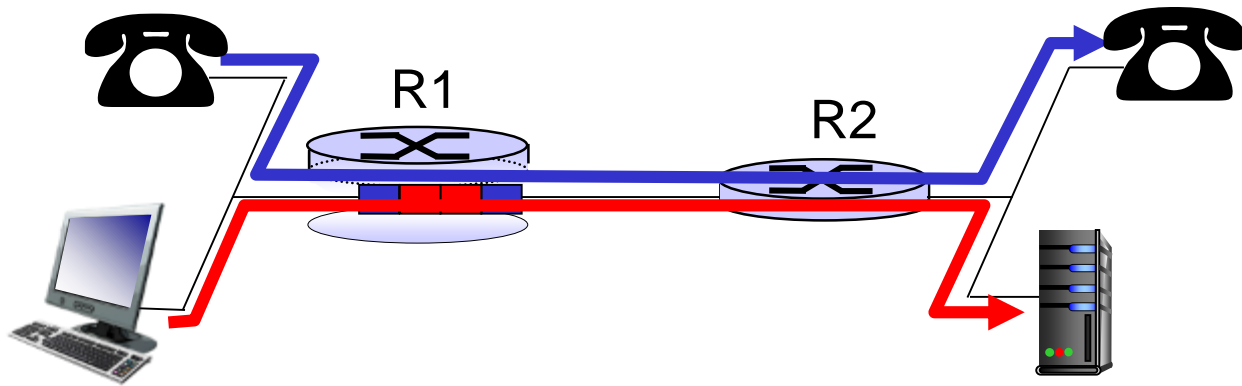


# Multiple classes of service: scenario



# Scenario I: mixed HTTP and VoIP

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

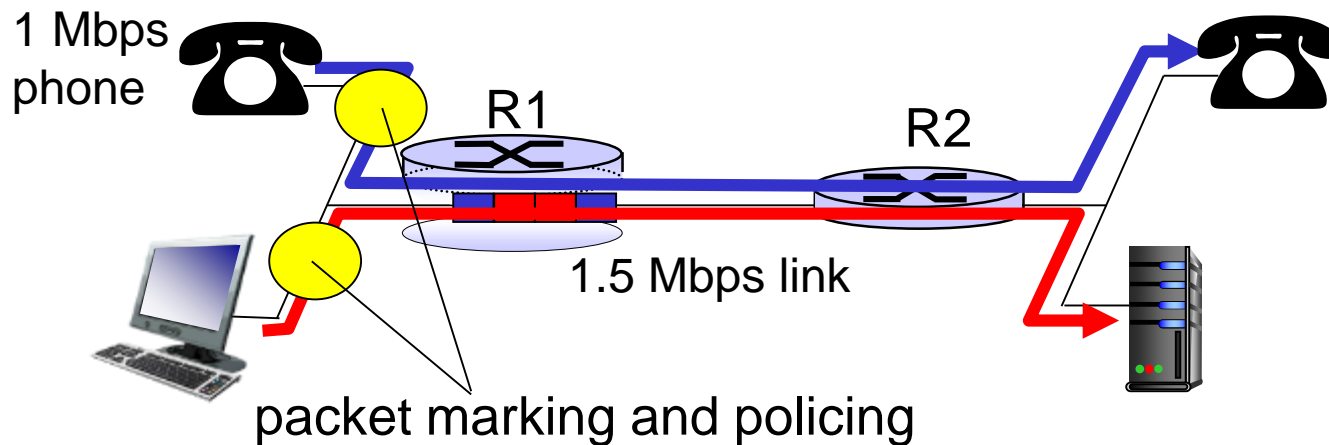


## Principle I

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

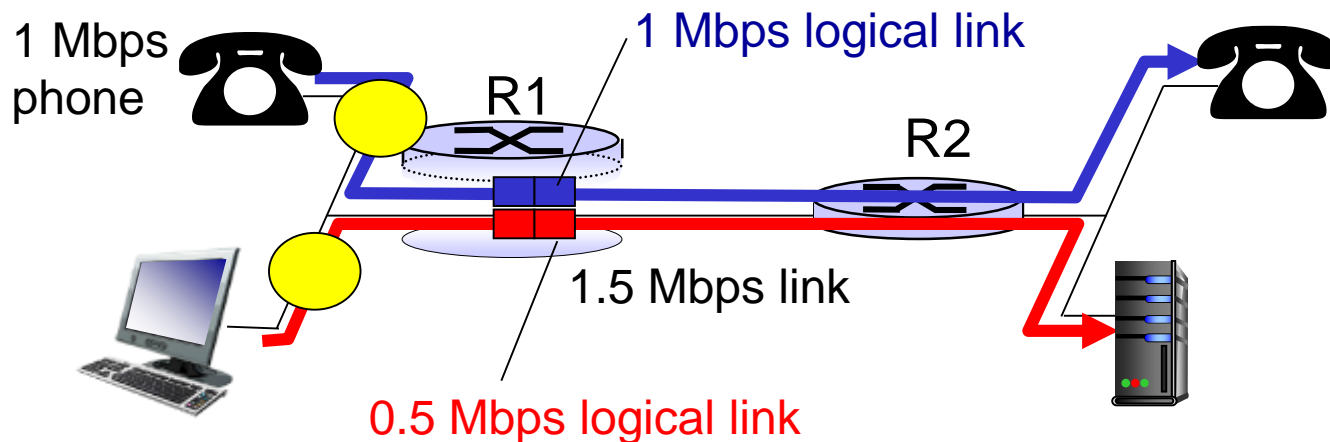


## Principle 2

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

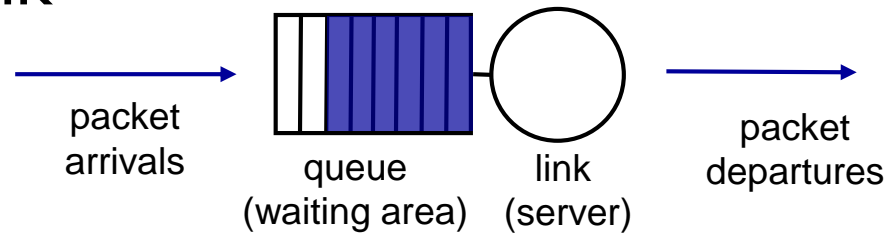


## Principle 3

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

# Scheduling and policing mechanisms

- *packet scheduling*: choose next queued packet to send on outgoing link



- previously covered in Chapter 4:
  - FCFS: first come first served
  - simply multi-class priority
  - round robin
  - weighted fair queueing (WFQ)

# Policing mechanisms

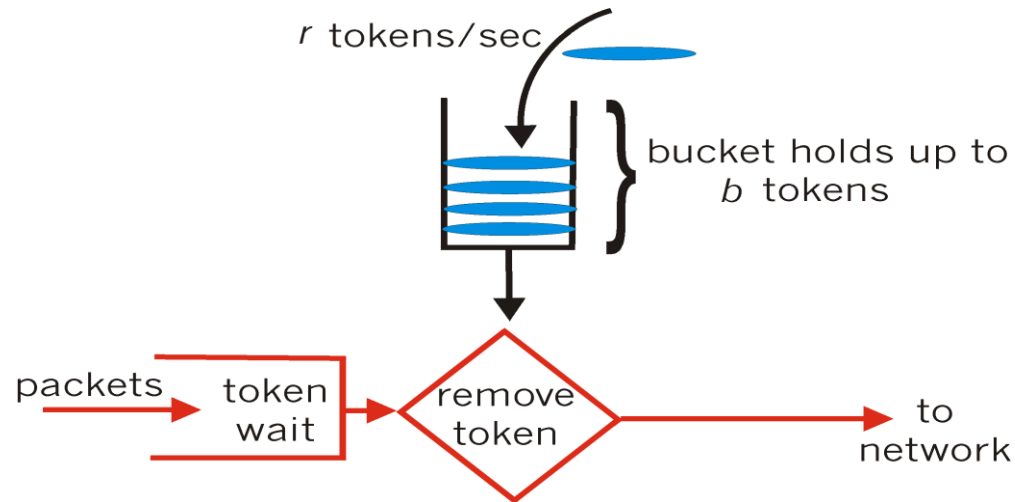
*goal:* limit traffic to not exceed declared parameters

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*

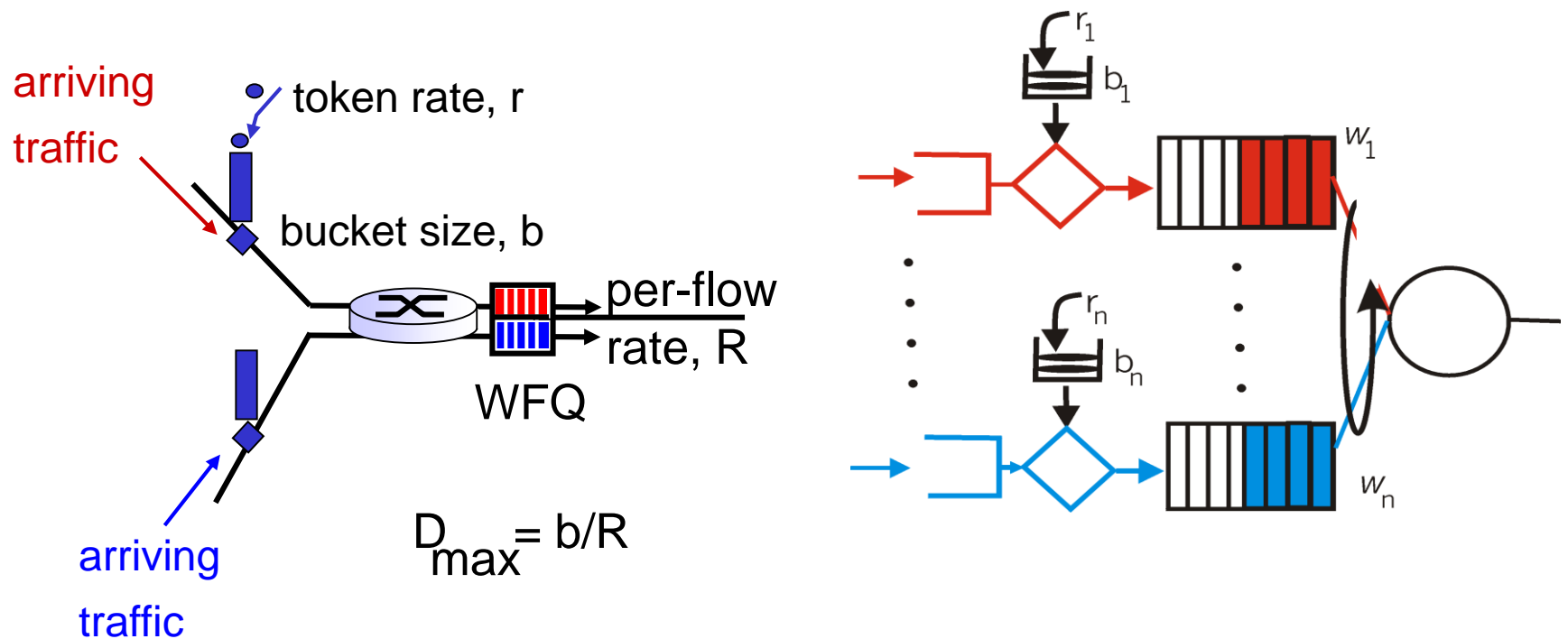


- bucket can hold  $b$  tokens
- tokens generated at rate  $r$  token/sec unless bucket full
- *over interval of length  $t$ : number of packets admitted less than or equal to  $(r t + b)$*



# Policing and QoS guarantees

- token bucket, WFQ combine to provide guaranteed upper bound on delay, i.e., *QoS guarantee!*



# Differentiated services

- want “qualitative” service classes
  - “behaves like a wire”
  - relative service distinction: Platinum, Gold, Silver
- *scalability*: simple functions in network core, relatively complex functions at edge routers (or hosts)
  - signaling, maintaining per-flow router state difficult with large number of flows
- don't define service classes, provide functional components to build service classes

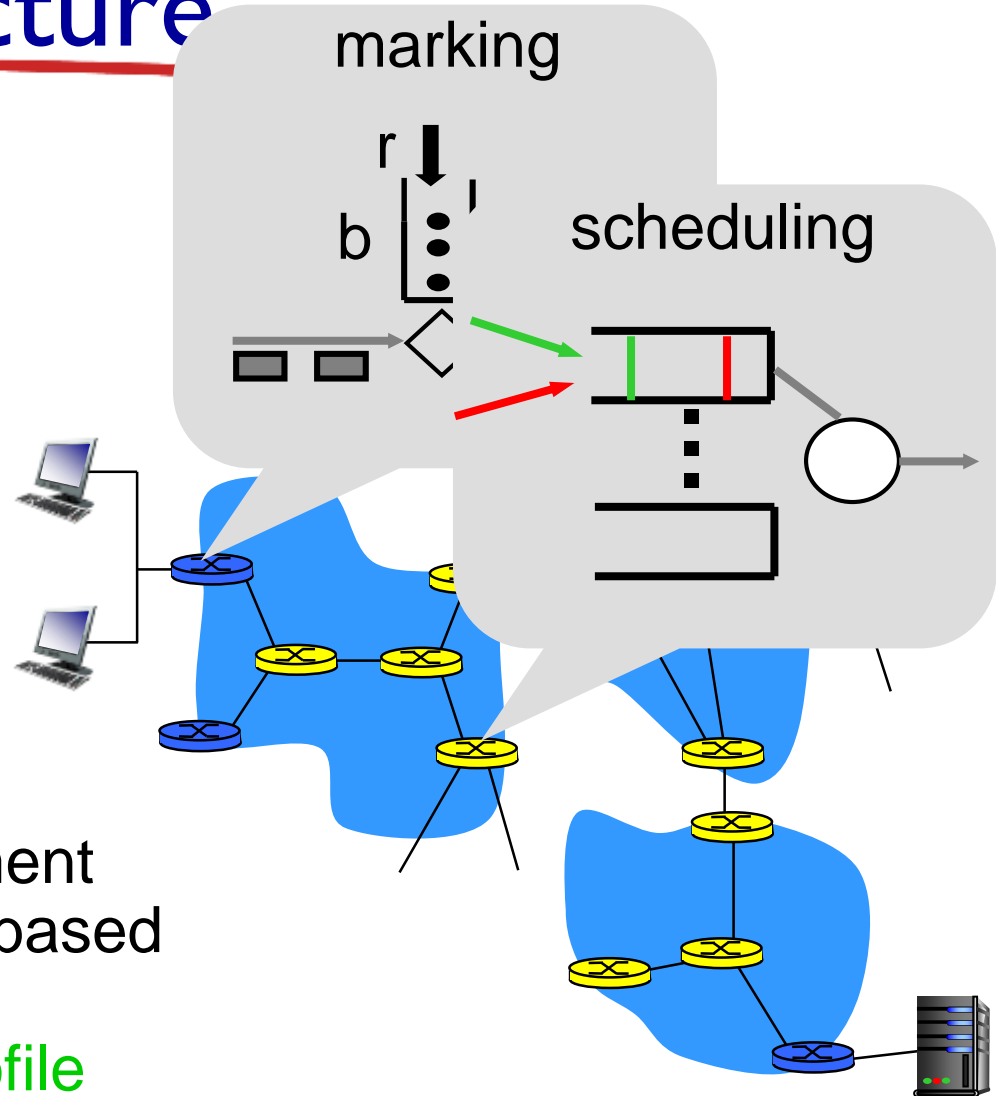
# Diffserv architecture

edge router: 

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

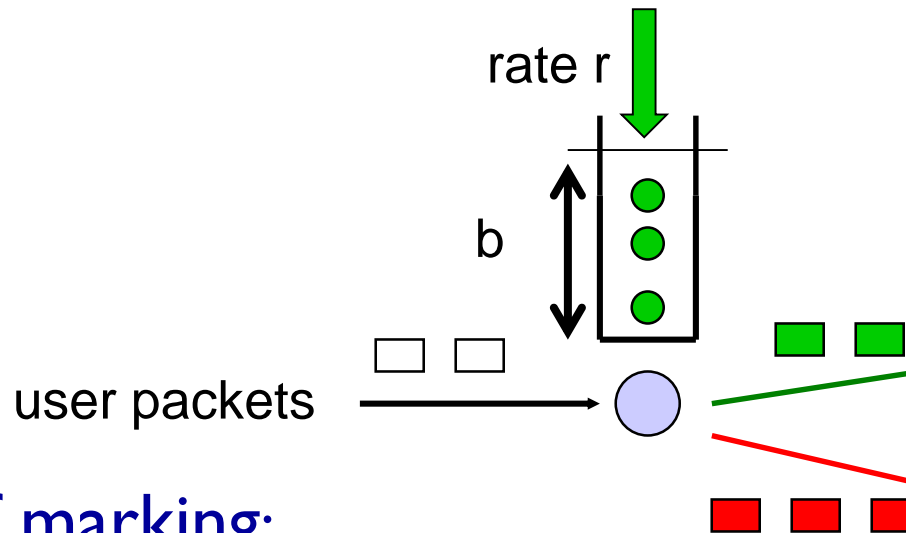
core router: 

- 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

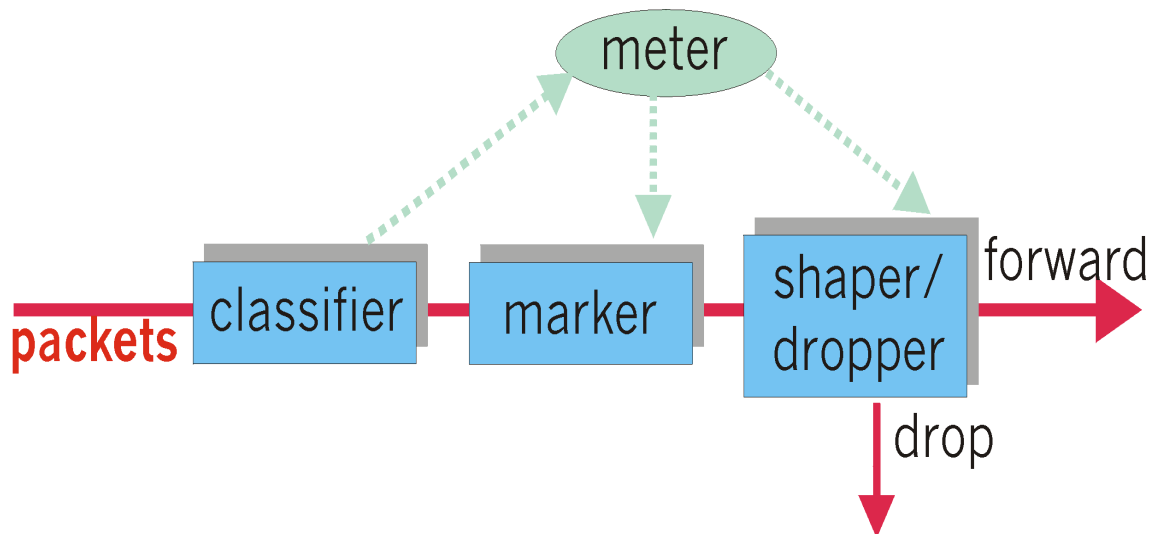
- packet is marked in the Type of Service (TOS) in IPv4, and Traffic Class in IPv6
- 6 bits used for Differentiated Service Code Point (DSCP)
  - determine PHB that the packet will receive
  - 2 bits currently unused



# 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

# Forwarding PHB

PHBs proposed:

- *expedited forwarding*: packet departure rate of a class equals or exceeds specified rate
  - logical link with a minimum guaranteed rate
- *assured forwarding*: 4 classes of traffic
  - each guaranteed minimum amount of bandwidth
  - each with three drop preference partitions



# 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