

Added : Multimedia Networking

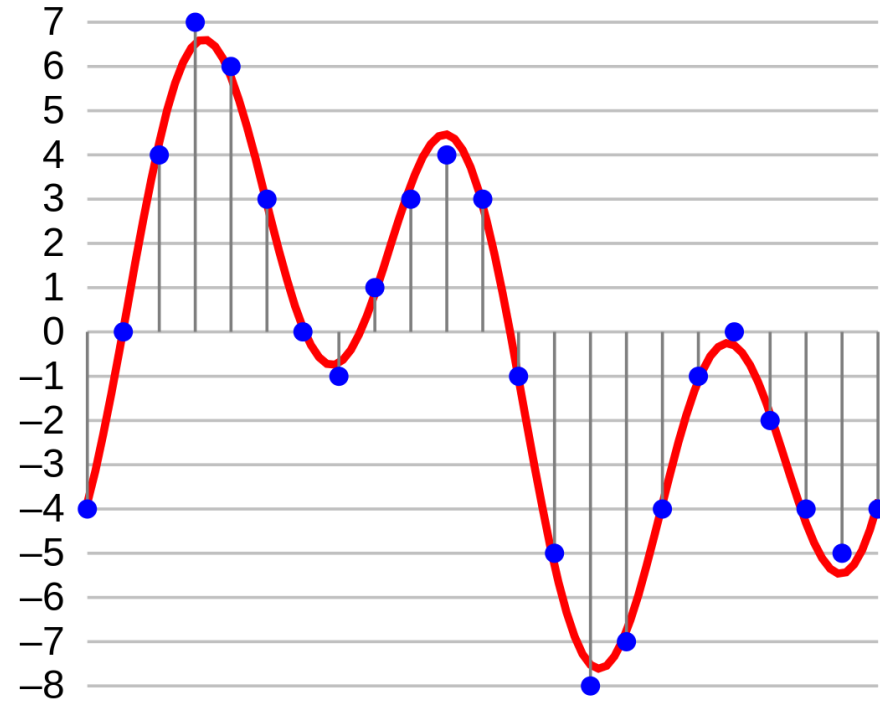
- ❑ Multimedia networking applications
- ❑ Streaming stored video
- ❑ Voice-over-IP
- ❑ Protocols for real-time conversational applications
- ❑ Network support for multimedia

Multimedia Networking

- multimedia networking applications
- streaming stored video
- voice-over-IP
- protocols for real-time conversational applications
- network support for multimedia

Multimedia: PCM encoding

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



Multimedia: audio

□ example: 8,000 samples/sec, 256

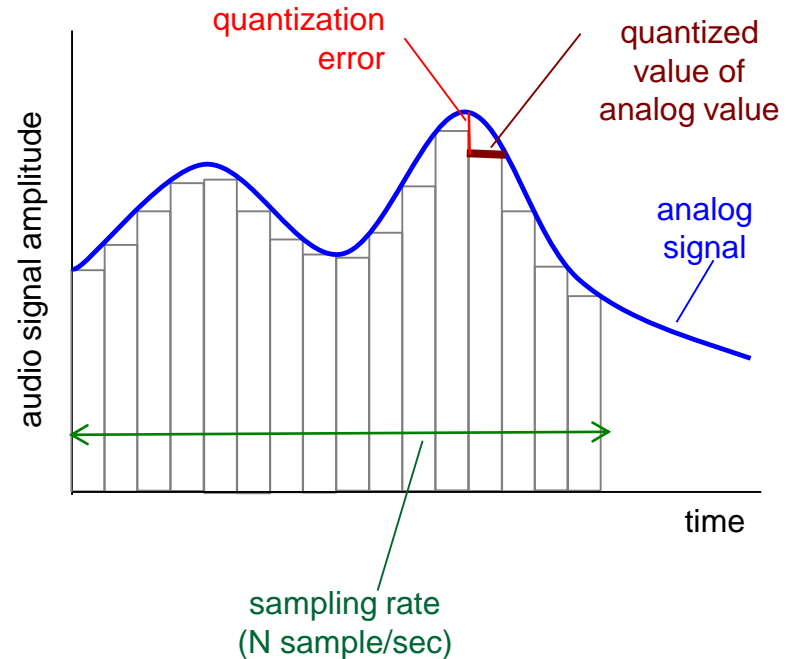
quantized values: 64,000 bps

□ receiver converts bits back to analog signal:

□ example rates

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

□ encoding error: sampling error, quantization error



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
 - resolution: 1920x1080 (full HD)
- ❖ coding: use redundancy **within** and **between** images to reduce # bits used to encode image
 - spatial (within image)
 - temporal (b/w images)

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



frame i

frame i+1

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



Multimedia: video

□ CBR: (constant bit rate): video

encoding rate fixed

□ VBR: (variable bit rate): video

encoding rate changes as amount of spatial, temporal coding changes

□ examples:

- MPEG 1 (CD-ROM): 1.5 Mbps
- MPEG2 (DVD): 3-6 Mbps
- MPEG4 (often used in Internet):
< 1 Mbps

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



frame i

frame i+1

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



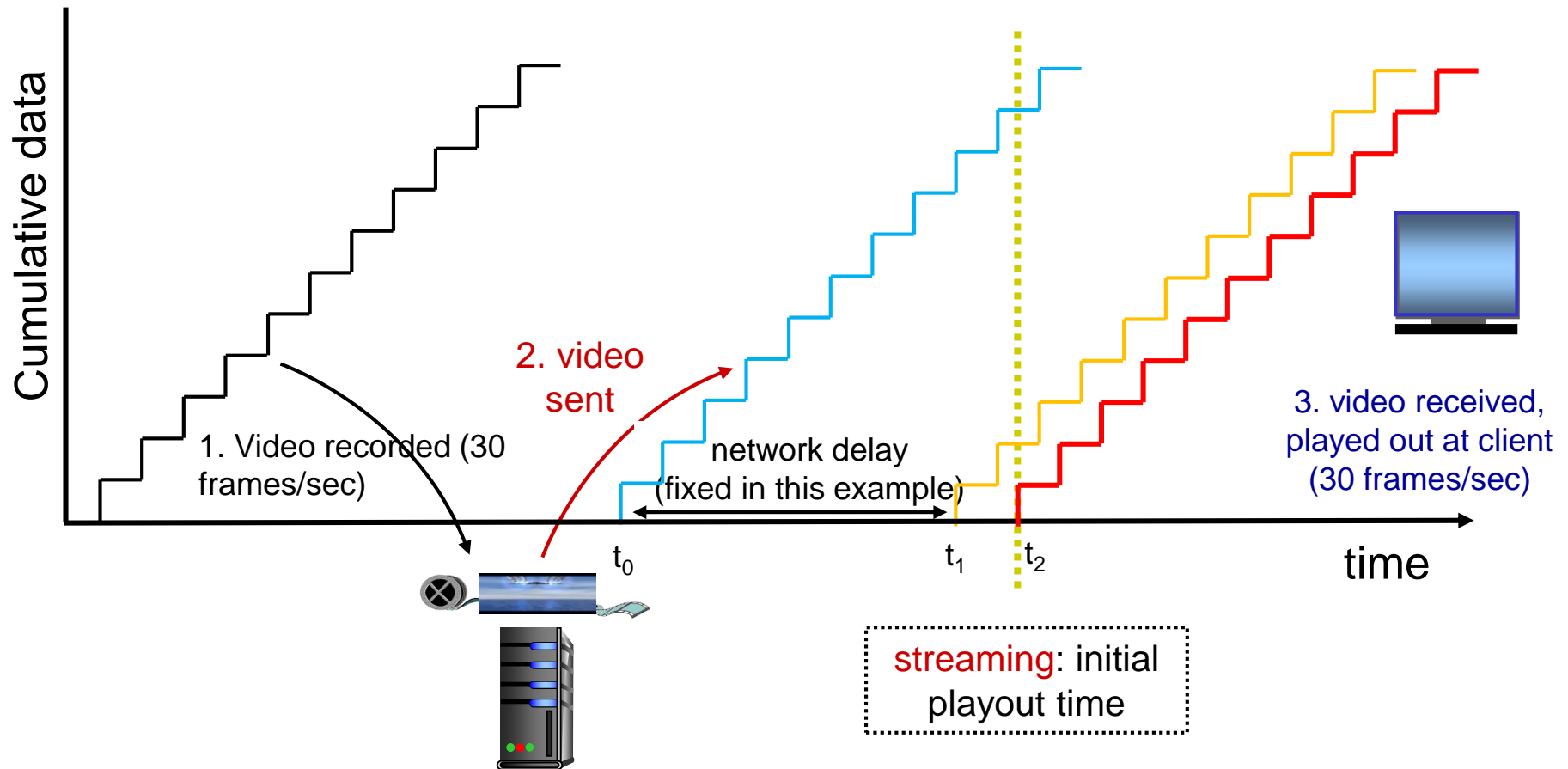
Multimedia networking: 3 application types

- streaming, stored audio, video
 - streaming: 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
- streaming live audio, video: e.g., live sporting event (football)
- conversational voice/video over IP
 - interactive nature of human-to-human conversation limits delay tolerance
 - e.g., Skype

Multimedia Networking

- multimedia networking applications
- streaming stored video
- voice-over-IP
- protocols for real-time conversational applications
- network support for multimedia

Streaming stored video



Streaming stored video: challenges

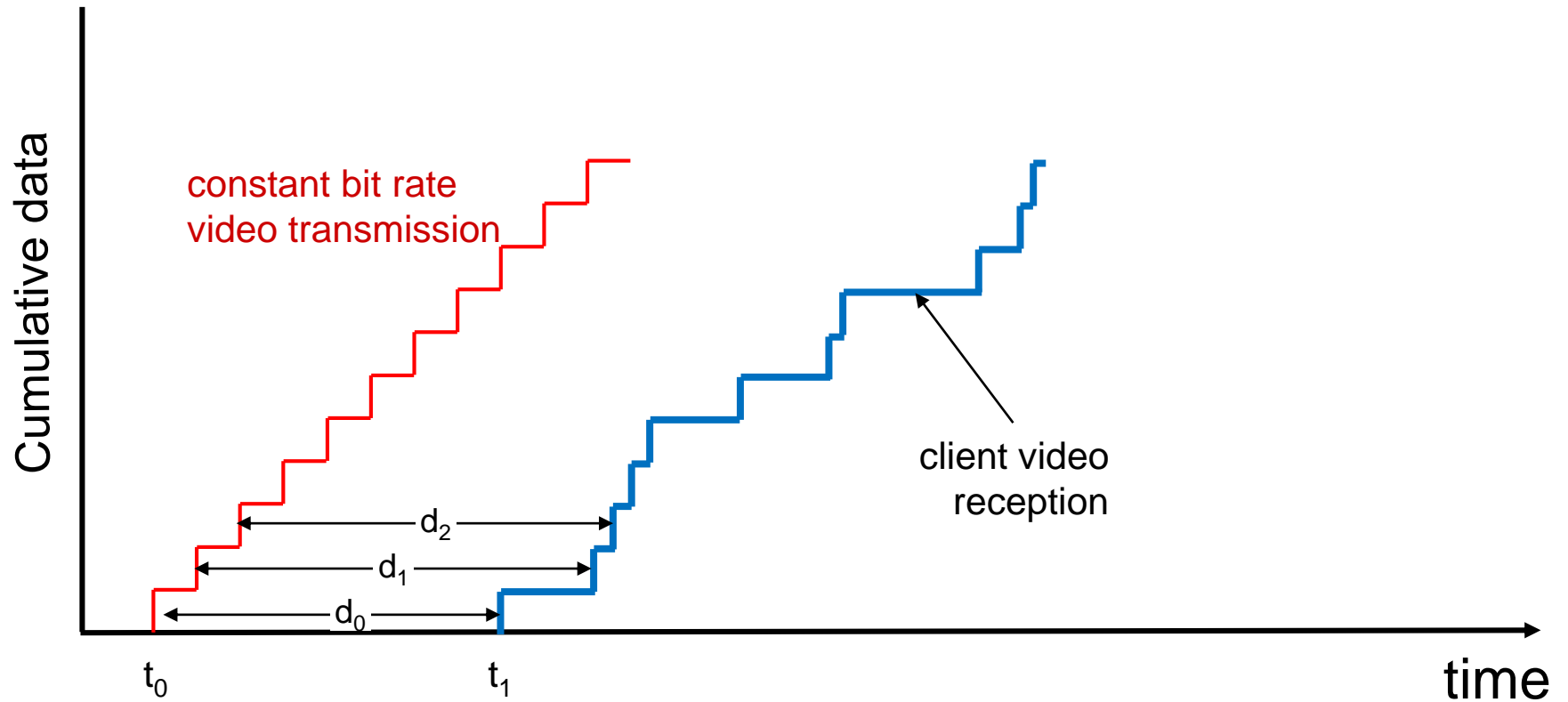
□ continuous playout constraint:

- once client playout begins, playback must match original timing
- ... but network delays are variable (**delay jitter**), so will need **client-side buffering** to match playout requirements

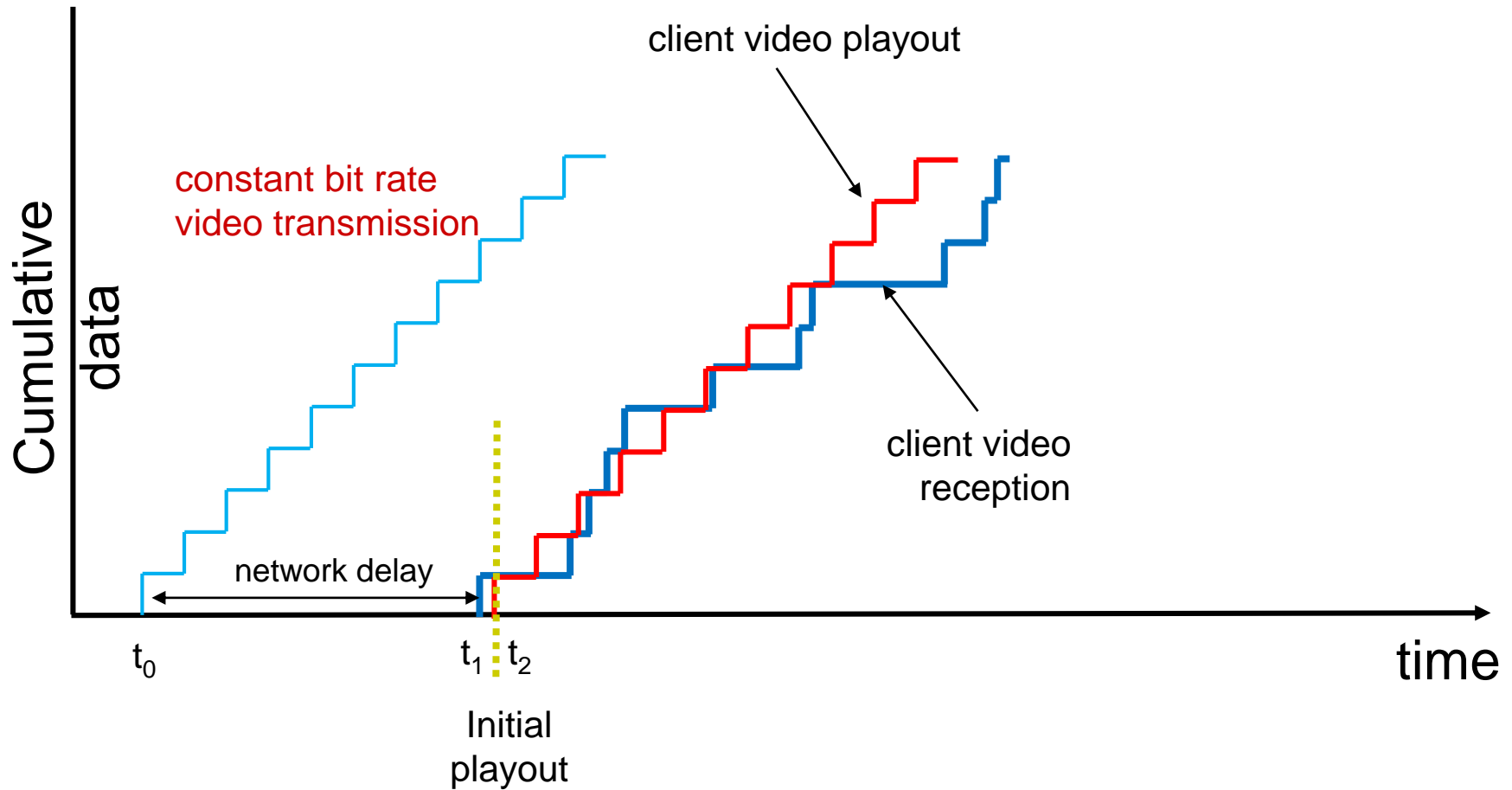
□ other challenges:

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

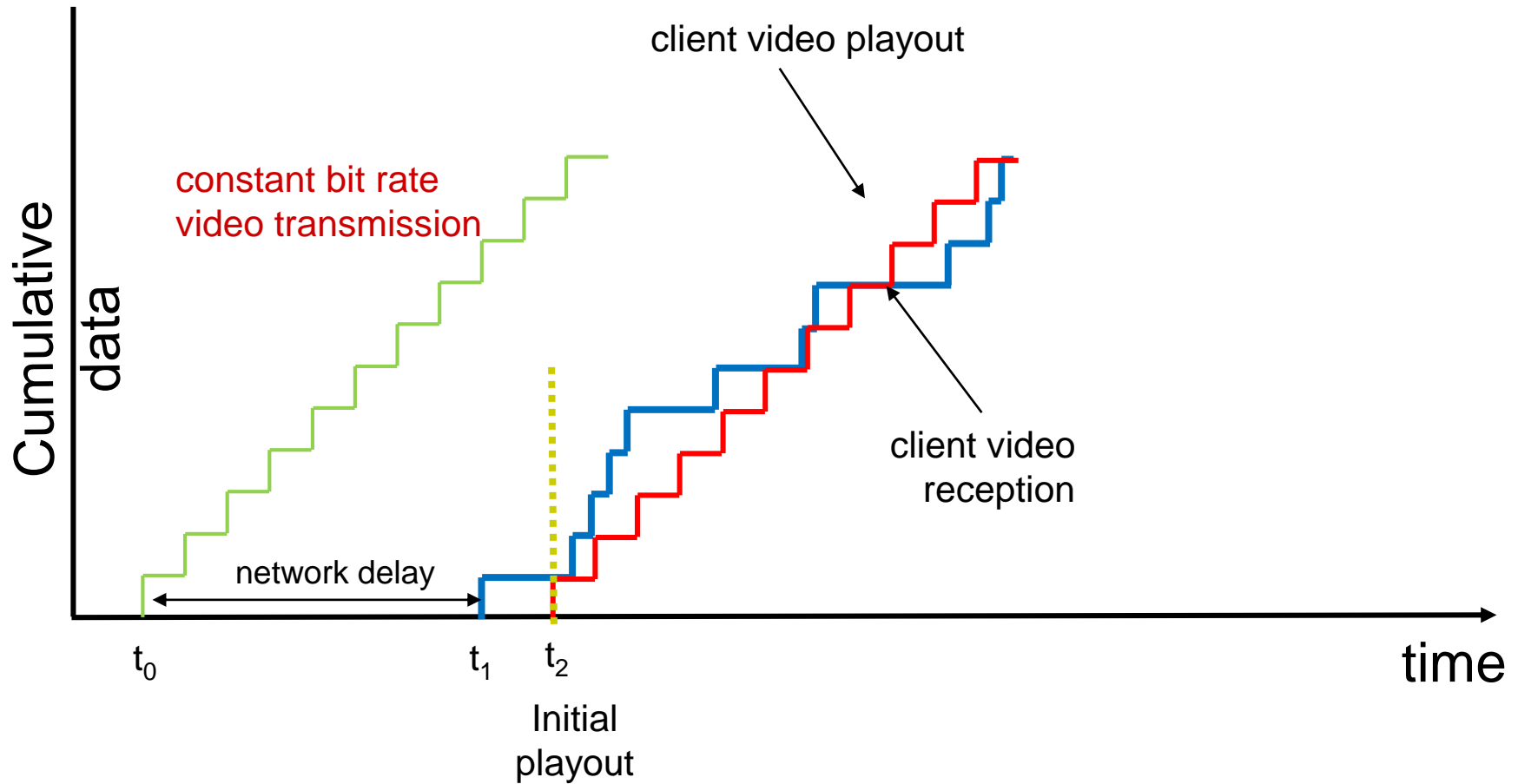
Streaming stored video: revisited



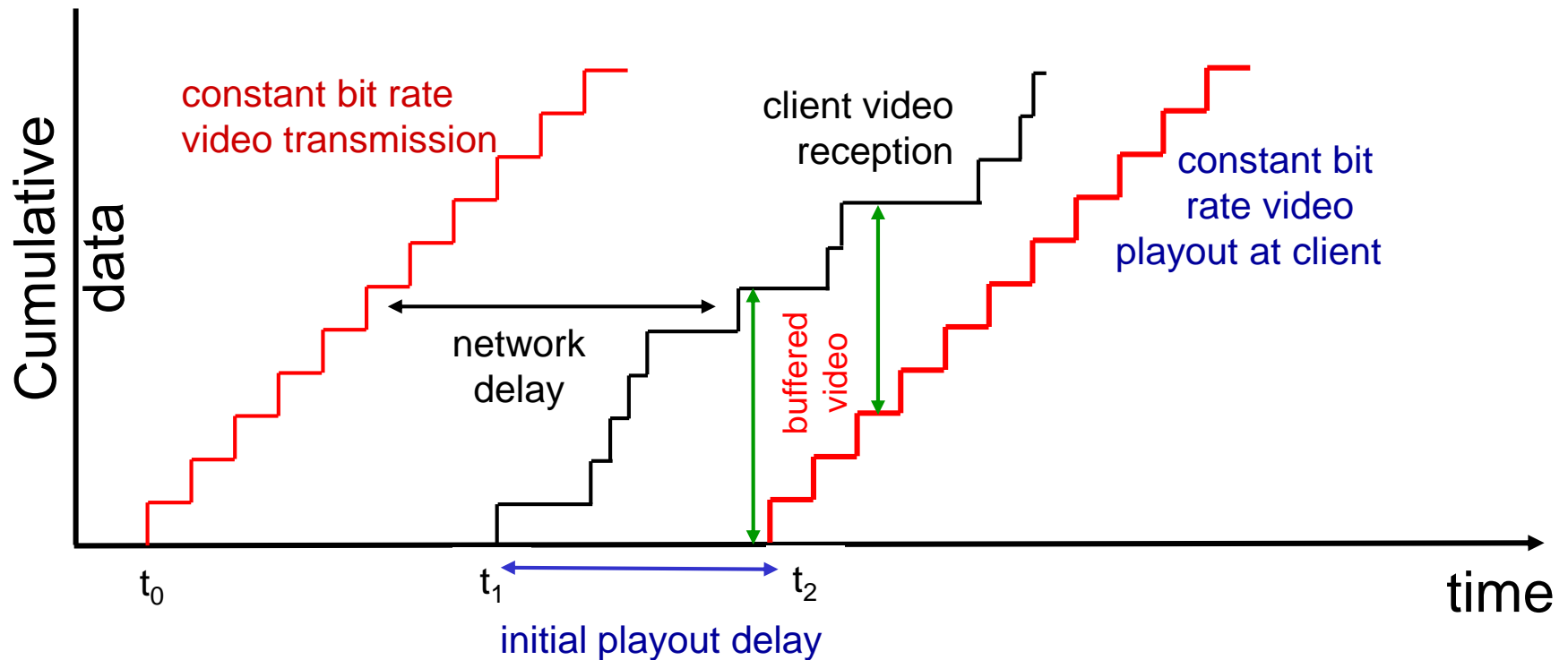
Streaming stored video: revisited



Streaming stored video: revisited

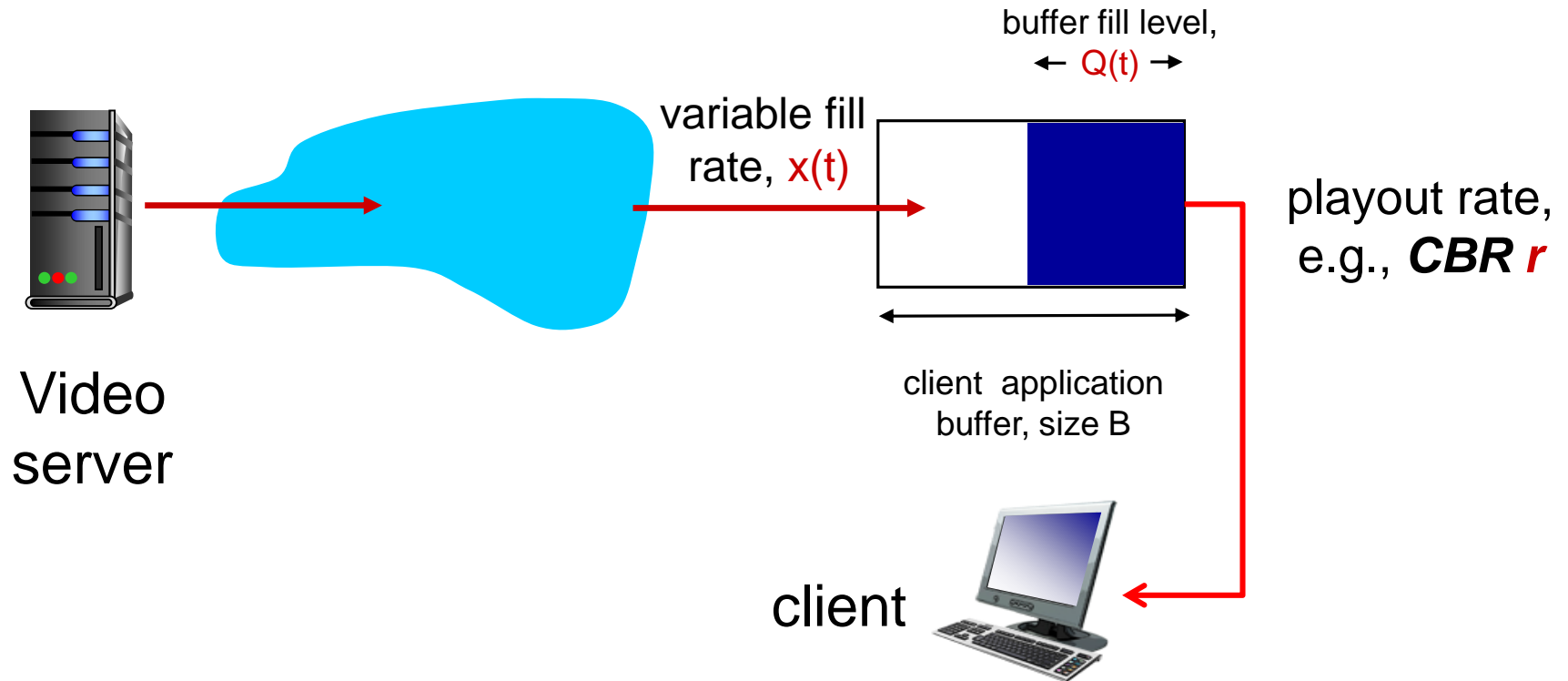


Streaming stored video: revisited

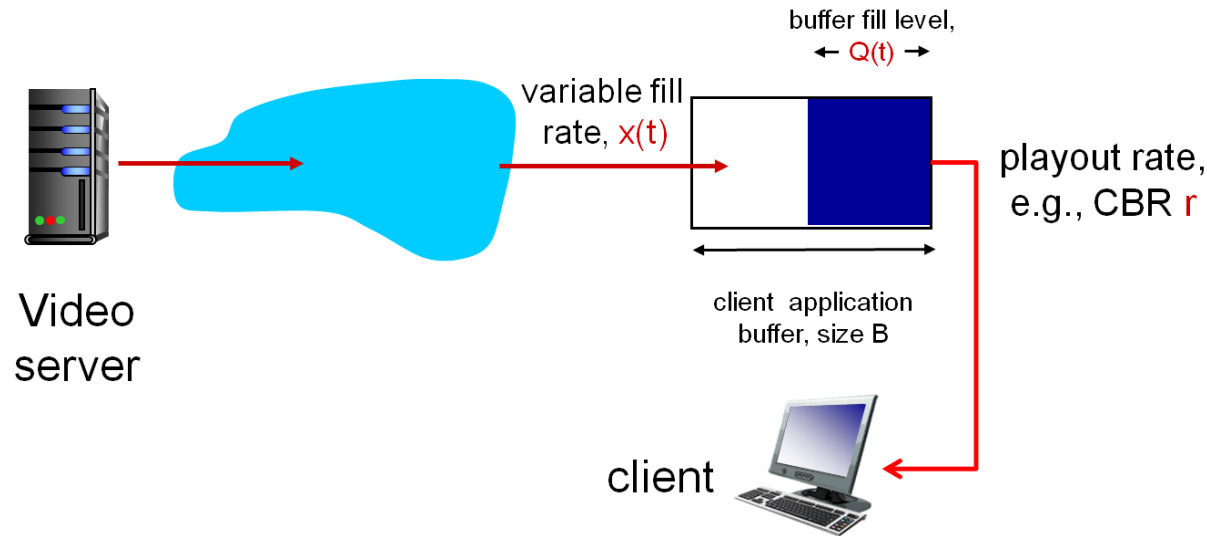


- *client-side buffering and playout delay*: compensate for network-added delay, delay jitter

Client-side buffering, playout



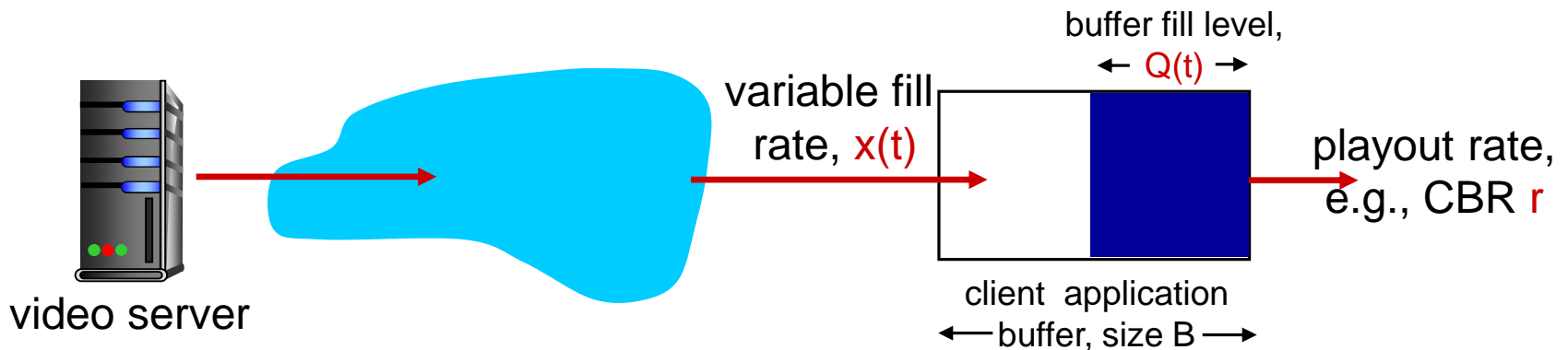
Client-side buffering, playout



- Initial fill of buffer until playout begins at t_p
- playout begins at t_p ($> t_p$)
- buffer fill level varies over time as fill rate $x(t)$ varies and playout rate r is constant

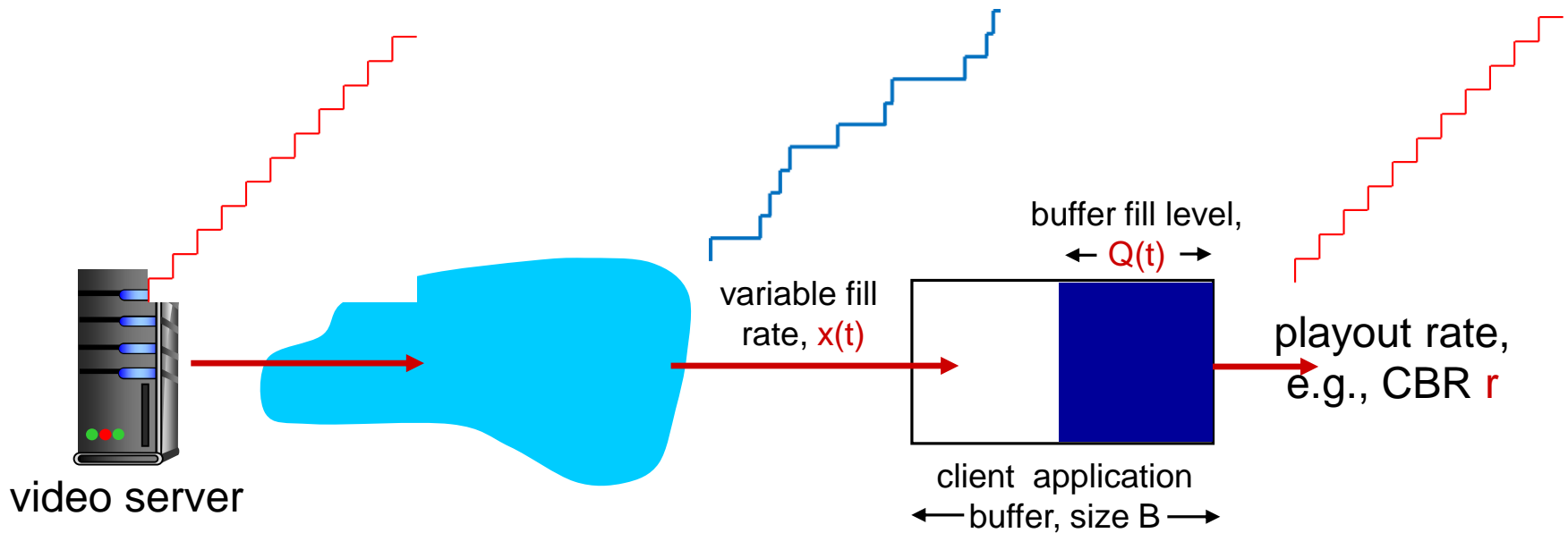
Client-side buffering, playout

- *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 $x(t)$



Client-side buffering, playout

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



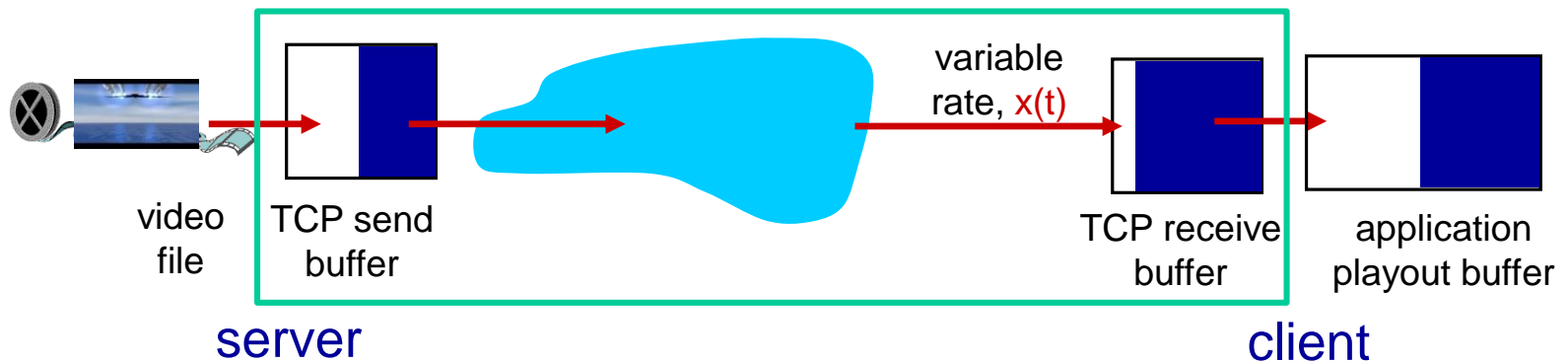
UDP Streaming

- server sends at rate appropriate for client
 - often: sending rate = encoding rate = constant rate
 - sender is oblivious to congestion levels
- short playout delay (2-5 seconds) to remove network jitter – client side buffering
- error recovery: application-level
- RTP [RFC 3550]: multimedia payload encapsulation
- RTSP [RFC 2326]: a network control protocol designed to control streaming media servers and process client-to-server interactivity
- UDP may not go through firewalls



HTTP Streaming

- multimedia file retrieved via **HTTP GET**
- **transmission rate under control of TCP**



- fill rate fluctuates due to **TCP congestion control**, retransmissions
- HTTP/TCP server: **client-side buffering and prefetching** enables HTTP streaming work well
- HTTP/TCP passes more easily through firewalls

Streaming multimedia: DASH

□ *DASH: Dynamic, Adaptive Streaming over HTTP*

- specification about manifest that includes information on multimedia streaming contents

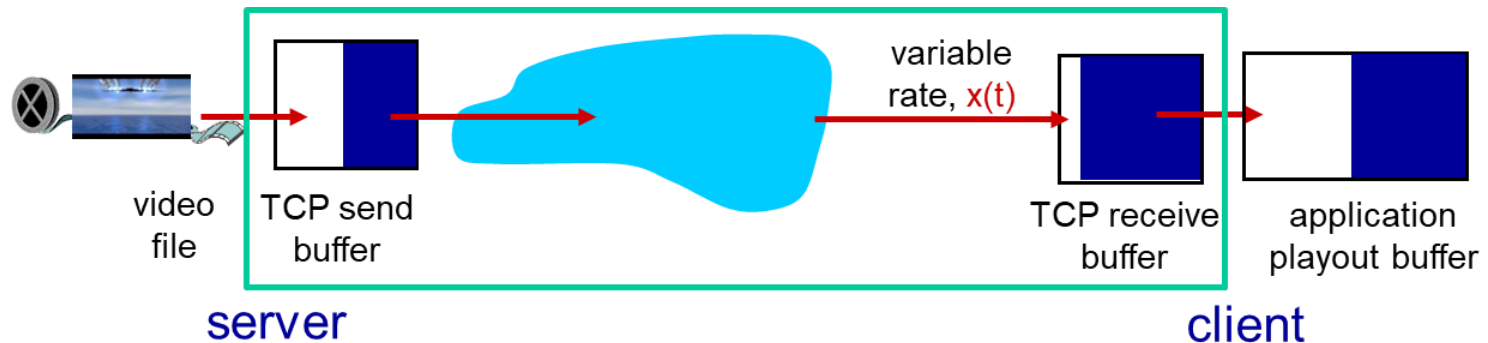
□ server:

- divides video file into multiple chunks
- each chunk stored, encoded at different rates
- manifest file: provides URLs for different chunks

Streaming multimedia: DASH

□ client:

- periodically measures server-to-client bandwidth
- consulting manifest, requests one chunk at a time
 - chooses maximum sustainable coding rate, given current bandwidth
 - can choose different coding rates at different points in time (depending on available bandwidth at time)



Streaming multimedia: DASH

- *“intelligence”* at client: client determines
 - *when to request chunk* (so that buffer starvation, or overflow does not occur)
 - *what encoding rate to request* (higher quality when more bandwidth available)
 - *where to request chunk* (can request from URL server that is “close” to client or has high available bandwidth)

Multimedia Networking

- multimedia networking applications
- streaming stored video
- voice-over-IP
- protocols for real-time conversational applications
- network support for multimedia

Voice-over-IP (VoIP)

□ *VoIP end-end-delay requirement:*

- higher delays noticeable, impair interactivity
- < 150 msec: good
- > 400 msec : bad
- includes application-level processing (packetization, playout) and network delays

□ *session initialization:* how does callee advertise IP address, port number, encoding algorithms?

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

□ *emergency services:* 119

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

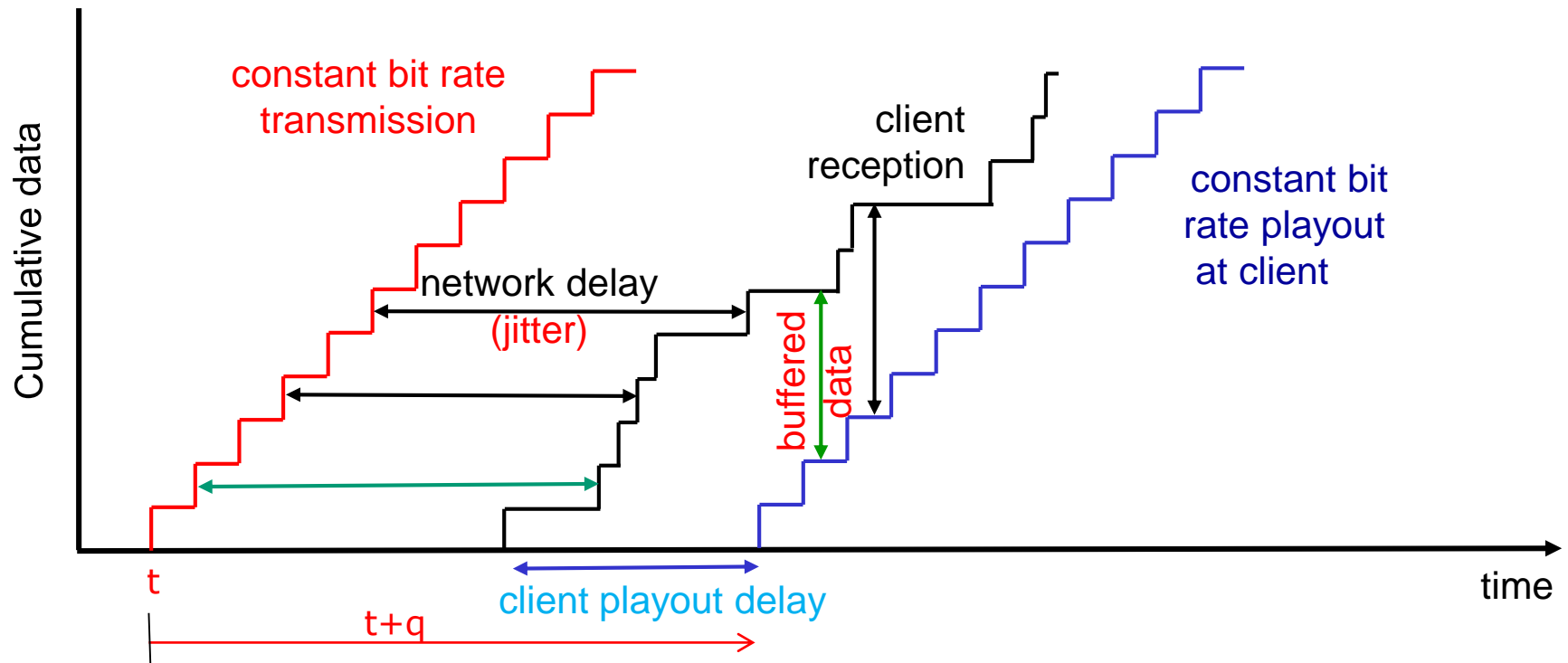


- application-layer header (AH) added to each chunk; (chunk+AH) encapsulated into UDP or TCP segment
- application sends segments thru socket every 20 msec during talk spurt

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, queuing in network; end-system (sender, receiver) delays
 - typical maximum tolerable delay: 400 ms
- *loss tolerance*:
 - depending on voice encoding and loss concealment
 - packet loss rates between 1% and 10% can be tolerated

Delay jitter



□ difference b/w end-to-end delays of two consecutive packets

VoIP: fixed playout delay

□ fixed playout delay: fixed q

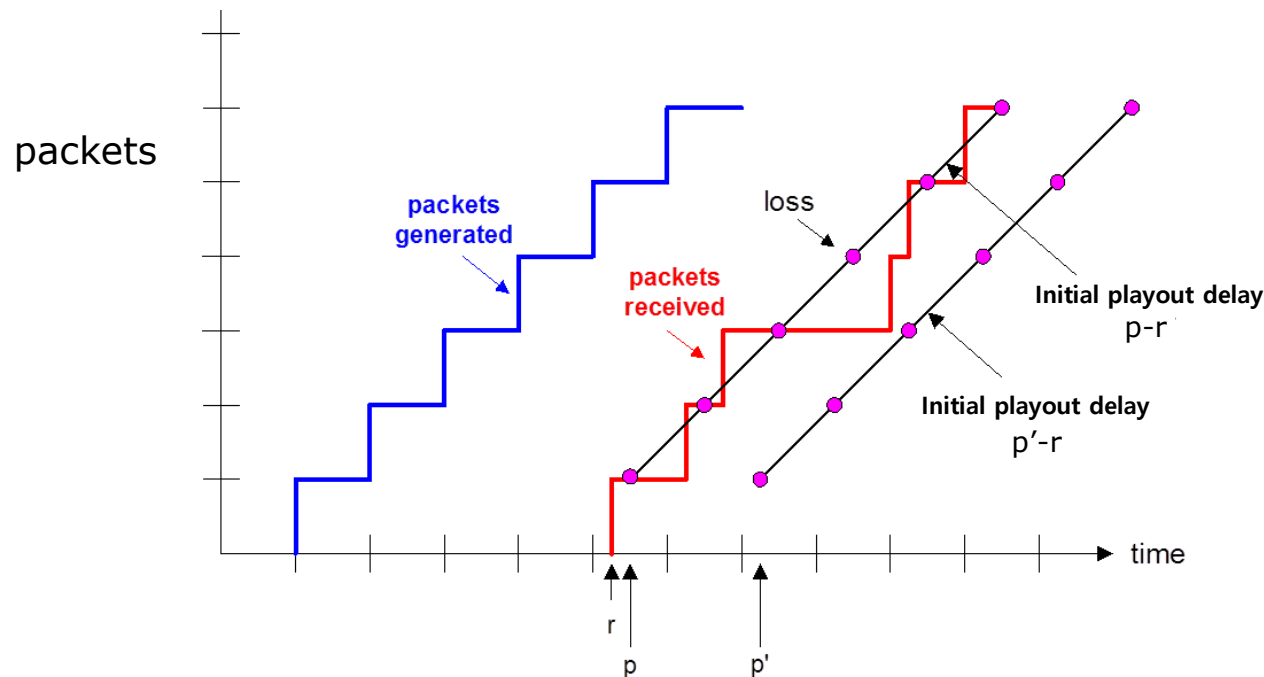
- receiver attempts to playout each chunk exactly q msecs after chunk was generated
- chunk has time-stamp t : play out chunk at $(t+q)$
- chunk arrives after $(t+q)$: data arrives too late for playout: data “lost”

□ 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
- **small playout delay**: first playout schedule: begins at p (small)
- **large playout delay**: second playout schedule: begins at p' (large)



Adaptive playout delay (1)

- *goal*: low playout delay, low late loss rate
- *approach*: adaptive playout delay adjustment:
 - estimate network delay during a talk spurt, adjust playout delay at the beginning of next talk spurt
 - silent periods reduced or elongated
 - chunks still played out every 20 msec during talk spurt



Adaptive playout delay (2)

- 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 $\underbrace{\text{time received} - \text{time sent}}_{\text{measured delay of i-th packet}}$

- also estimate average deviation of delay, v_i :

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

Adaptive playout delay (3)

- 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:

$$\text{playout-time}_i = t_i + d_i + K * v_i \quad (K: \text{constant, e.g. 4})$$

- remaining packets in talk-spurt are played out periodically



Adaptive playout delay (4)

Q: How to determine whether packet is first in a talk-spurt?

- uses time-stamps and sequence-no in the packet
- if no loss, receiver looks at successive time-stamps
 - difference of successive time-stamps $> 20 \text{ msec}$ --> talk spurt begins
- with loss possible, receiver must look at both time-stamps and sequence numbers
 - difference of successive time-stamps $> 20 \text{ msec}$ and sequence numbers without gaps --> talk spurt begins



RTP header

VoiP: recovery from packet loss (1)

Challenge: how to recover from packet loss

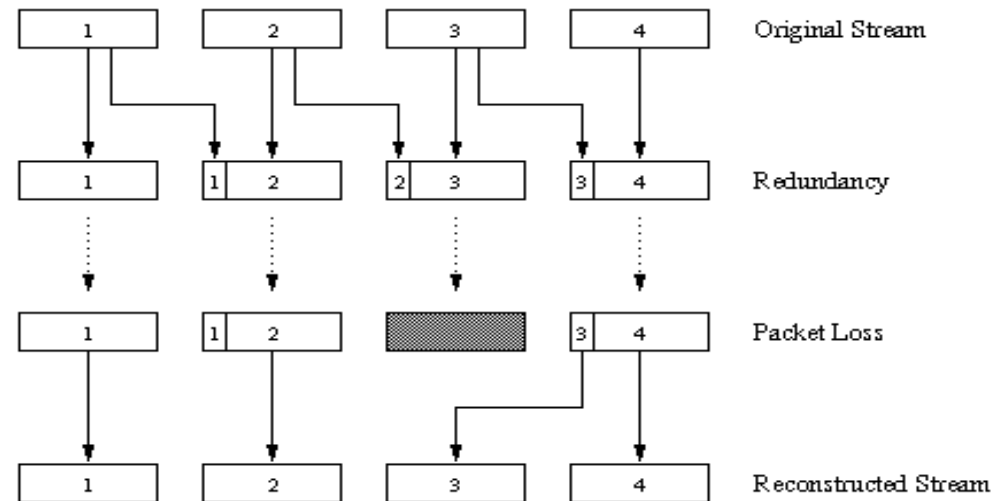
- retransmission after ACK/NAK takes more than one RTT
- *Forward Error Correction (FEC) using redundant chunks*
 - (e.g.) for every group of n chunks, create and send *one* redundant chunk by XOR-ing n original chunks → increasing bandwidth by factor $1/n$
 - can reconstruct original n chunks if at most one lost chunk from $n+1$ chunks, with small playout delay

C1	C2	C3	$C1 \oplus C2 \oplus C3$	C4	C5	C6	$C4 \oplus C5 \oplus C6$	C7
----	----	----	--------------------------	----	----	----	--------------------------	----

VoiP: recovery from packet loss (2)

□ another FEC scheme:

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



□ non-consecutive loss: receiver can conceal loss

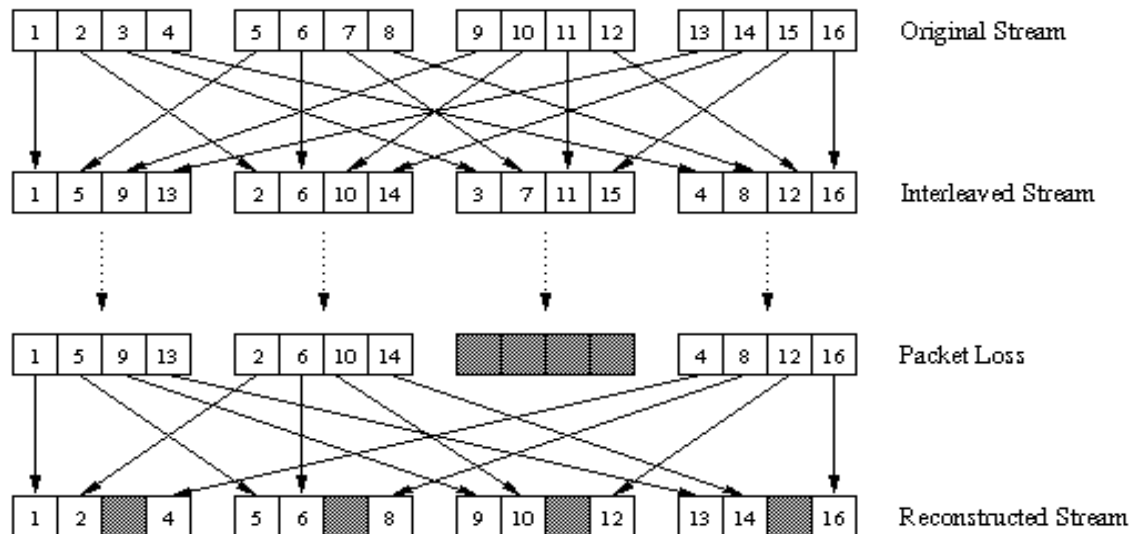
□ generalization: can also append (n-1)-st and (n-2)-nd low-bit rate chunk

VoiP: recovery from packet loss (3)

□ interleaving to conceal loss:

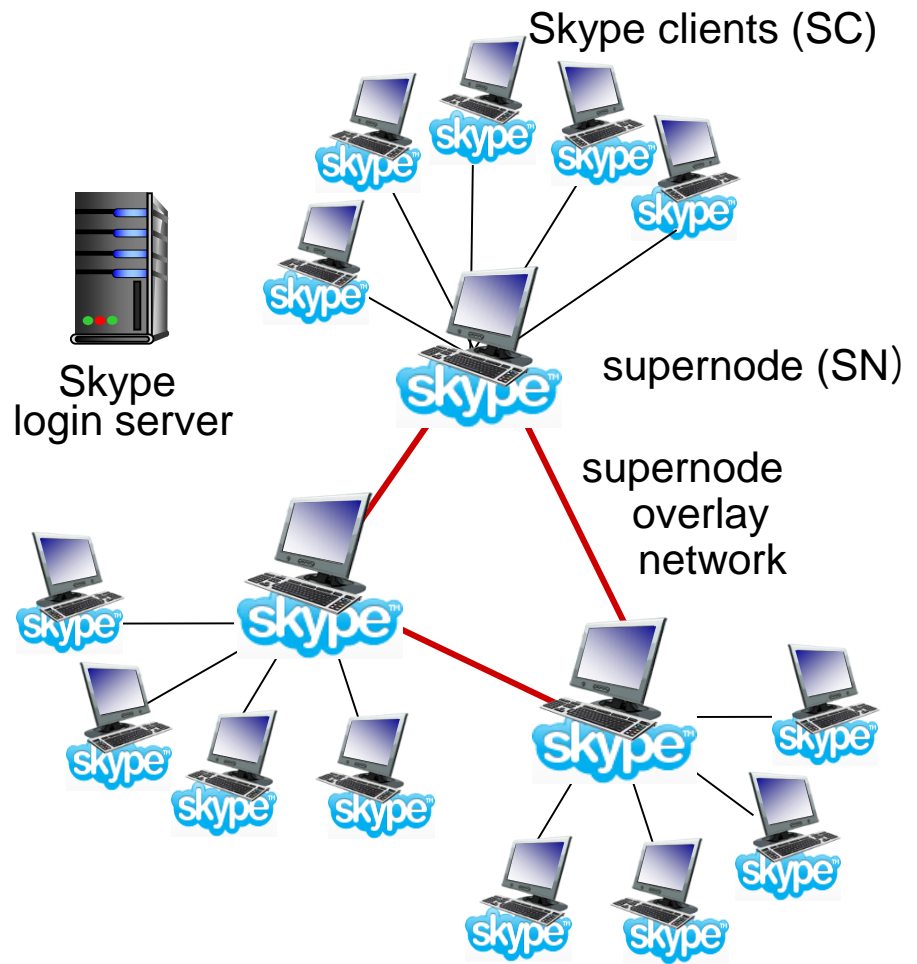
- audio chunks divided into smaller units, e.g. four 5 ms units per 20 ms audio chunk
- packet contains small units from different chunks

- if packet lost, still have most of every original chunk
- no redundancy overhead, but a little decrease in audio quality



Voice-over-IP: Skype

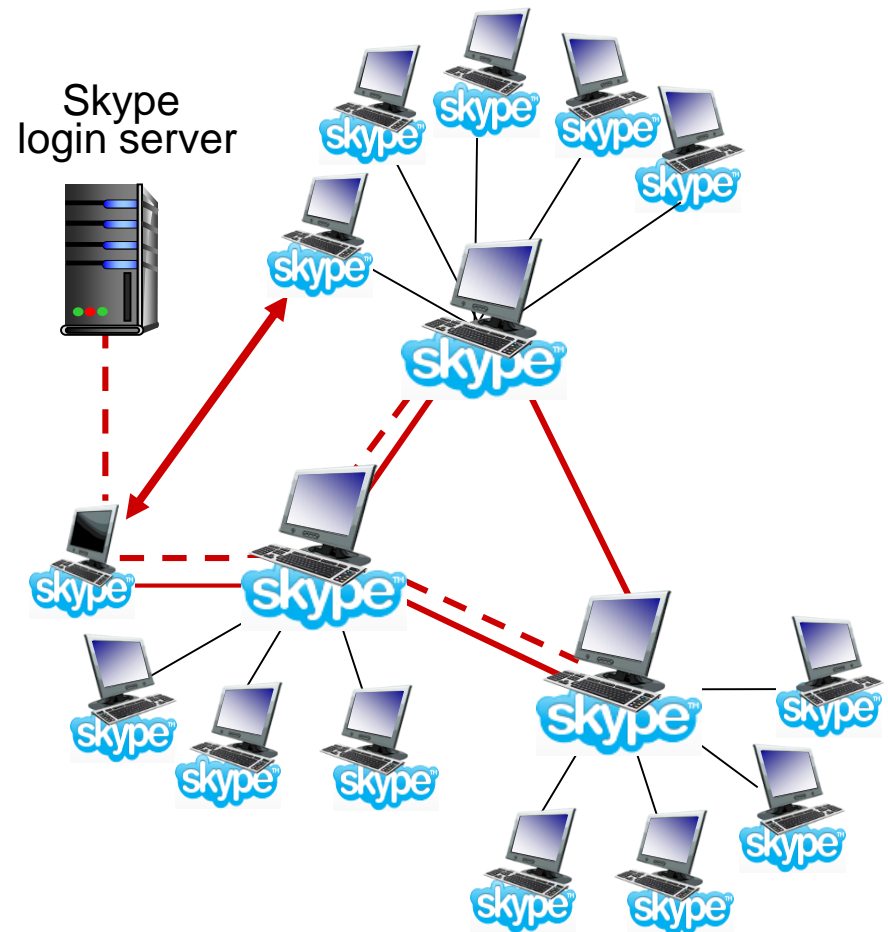
- proprietary application-layer protocol
- encrypted msgs
- P2P components:
 - **super nodes (SN):** skype peers with special functions
 - **overlay network:** among SNs to locate SCs
 - **login server**



P2P voice-over-IP: skype

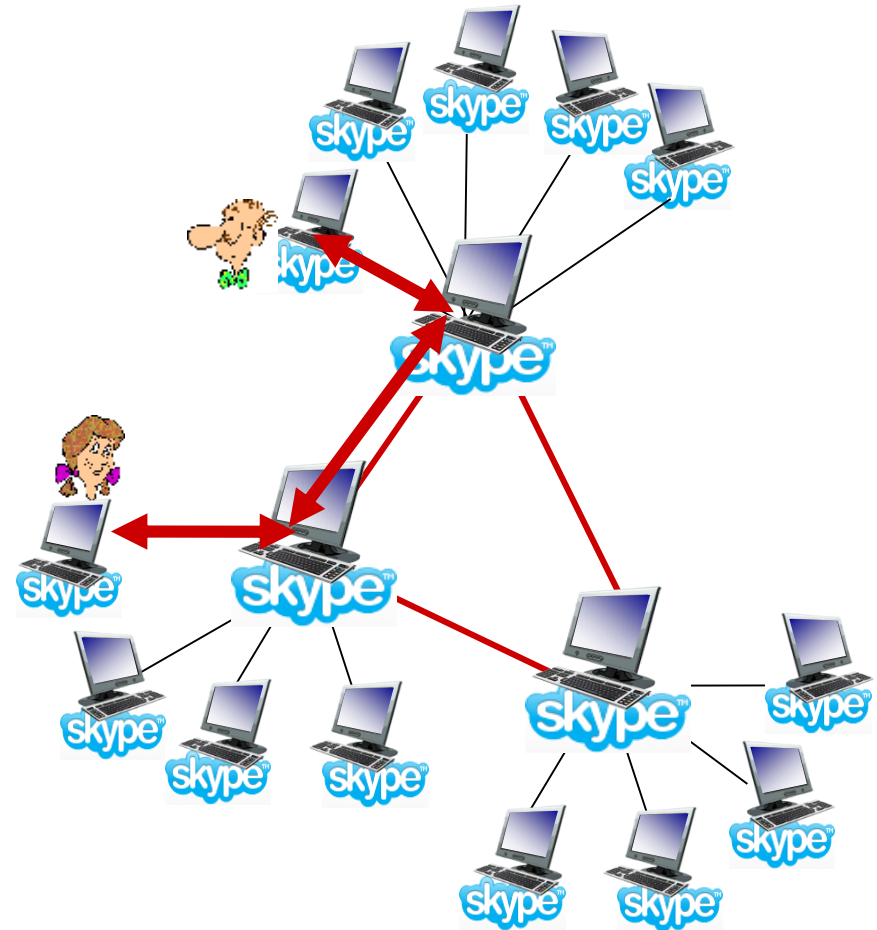
□ skype client operation

- joins skype network by contacting SN (IP address cached) using TCP
- logs-in (username, password) to centralized skype login server
- obtains IP address for callee from SN, SN overlay or client buddy list
- initiate call directly to callee



Skype: peers as relays

- *problem:* both Alice, Bob are behind “NATs”
- relay solution: maintain open connection using 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

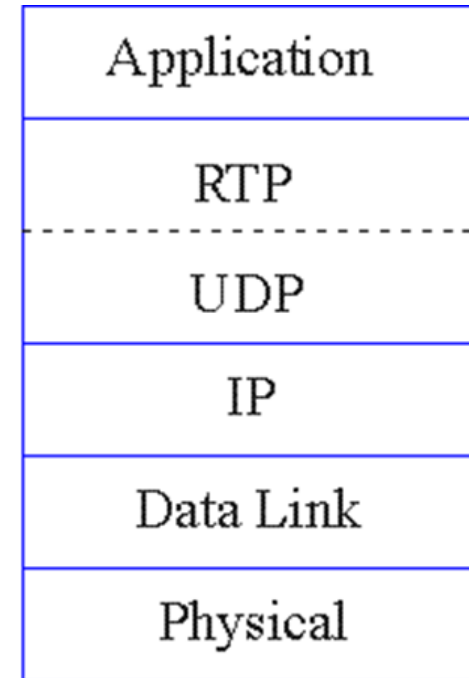


Multimedia Networking

- multimedia networking applications
- streaming stored video
- voice-over-IP
- protocols for real-time conversational applications:
RTP, SIP
- network support for multimedia

Real-Time Protocol (RTP)

- ❑ RTP specifies packet structure for carrying audio, video data
- ❑ RFC 3550
- ❑ RTP runs in end systems
- ❑ RTP packets encapsulated in UDP segments
- ❑ RTP header provides
 - 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
- (RTP header + audio chunk) form RTP packet, which is encapsulated in UDP segment

□ RTP header contains

- type of audio encoding in each packet; sender can change encoding during session
- sequence number
- time-stamps



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



□ payload type (7 bits):

- indicates **type of encoding** currently being used
- sender can change encoding during call
- type 0: PCM mu-law, 64 kbps, type 3: GSM, 13 kbps, type 7: LPC, 2.4 kbps, type 31: H.261, type 33: MPEG2 video

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

- detect packet loss, restore packet sequence

RTP header

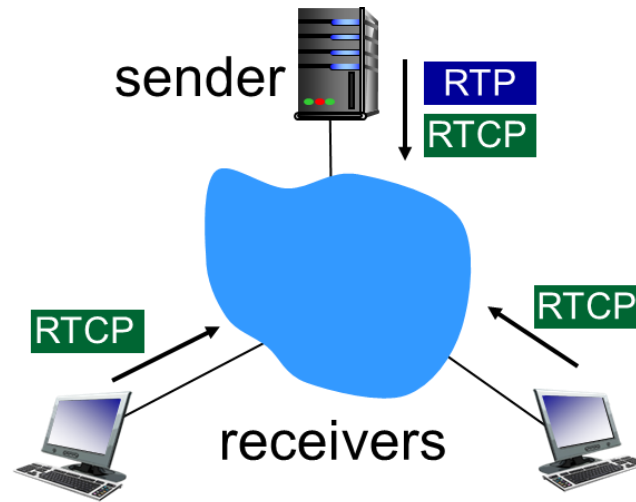
payload type	sequence number	time stamp	Synchronization Source ID	Miscellaneous fields
-----------------	--------------------	------------	------------------------------	-------------------------

- *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)
 - if appl 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 (silence period)
- *Synchronization source ID 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 sends periodically **RTCP control packets** to all other participants
- RTCP packet contains sender or receiver reports
 - **report statistics** : # packets sent, # packets lost, inter-arrival jitter
- **feedback used to control performance**
 - sender may modify its transmissions based on feedback

RTCP: multiple multicast senders



- ❑ **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** (RTP: even port number, RTCP: odd port number)
- ❑ 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 inter-arrival jitter

□ sender report packets:

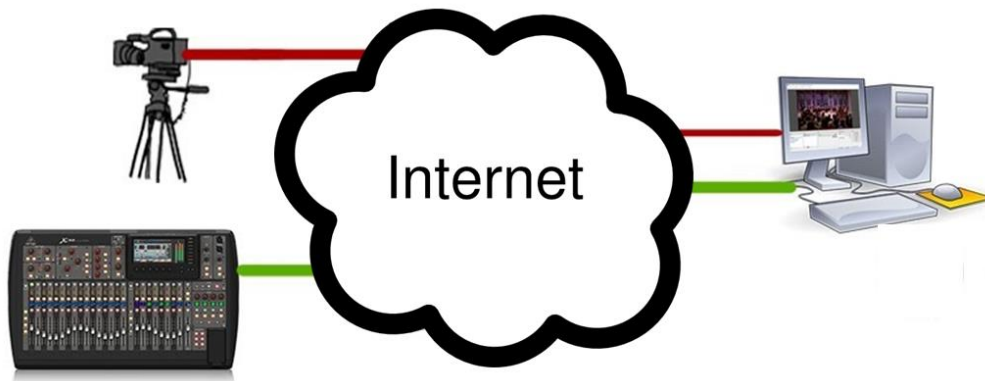
- SSRC of RTP stream, current time, number of packets sent, number of bytes sent

□ 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

- Synchronization of RTP streams within a RTP session using timestamps in RTP packets



- 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 timestamps in the associated stream to synchronize

RTCP: bandwidth scaling

- RTCP attempts to limit its traffic to 5% of session bandwidth
- RTCP gives 75% of rate to receivers; remaining 25% to sender
- example : one sender, sending video at 2 Mbps
 - RTCP attempts to limit RTCP traffic to 100 Kbps
- sender gets to send RTCP traffic at 25 kbps
- 75 kbps is equally shared among receivers:
 - with N receivers, each receiver gets to send RTCP traffic at $75/N$ kbps

SIP services

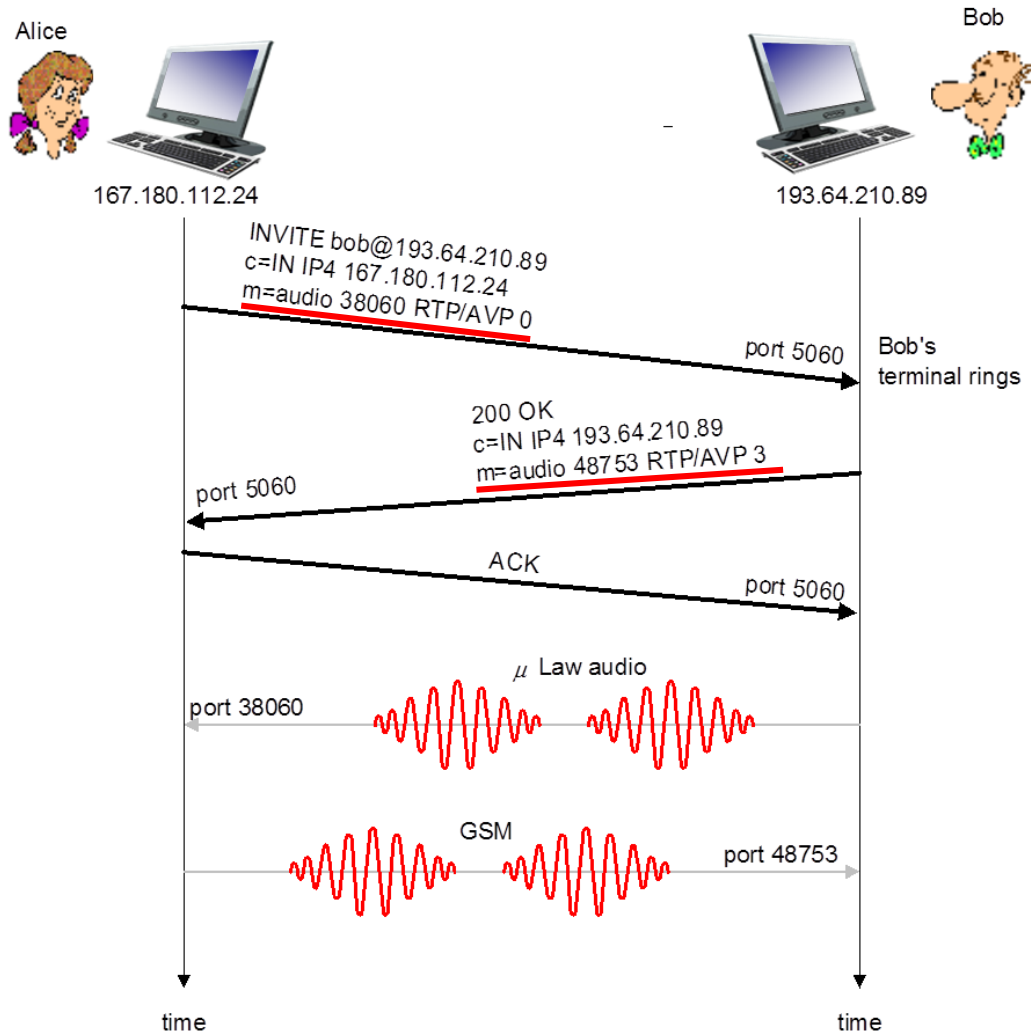
□ SIP provides mechanisms for call setup:

- for caller to initiate a call to callee
- determine current IP address of callee
- caller and callee can agree on media type, encoding
- to end call

□ call management:

- add new media streams during call
- change encoding during call
- invite others
- transfer, hold calls

SIP: setting up call to known IP address



- default SIP port number : 5060
- Alice's SIP **invite message** indicates her port number, IP address, encoding she prefers to receive (PCM mlaw)
- 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

과제 #3

□ SIP 프로토콜 조사

- SIP, SDP 프로토콜 기능
- SIP, SDP 프로토콜 메시지 포맷
- SIP 프로토콜 기반 VoIP 서비스 시나리오

□ 제출

- 파일 (hwp 또는 word)
- 파일이름: “hw3-학번-이름.hwp”

Multimedia Networking

- multimedia networking applications
- streaming stored video
- voice-over-IP
- protocols for real-time conversational applications
- network support for multimedia

Network support for multimedia

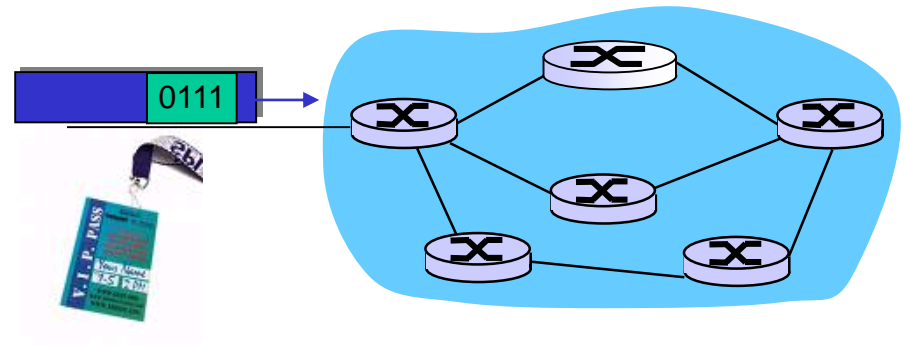
Approach	Granularity	Guarantee	Mechanisms	Complex	Deployed?
Making best of best effort service	All traffic treated equally	None or soft	No network support (all at application)	low	everywhere
Differentiated service (Diffserv)	Traffic “class”	None or soft	Packet marking scheduling, policing.	med	some
Per-connection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

Dimensioning best effort networks

- *traffic engineering*: deploy enough link capacity so that congestion doesn't occur, multimedia traffic flows without delay or loss
 - traditional approach: low complexity of network mechanisms (use current “best effort” network)
 - needs 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

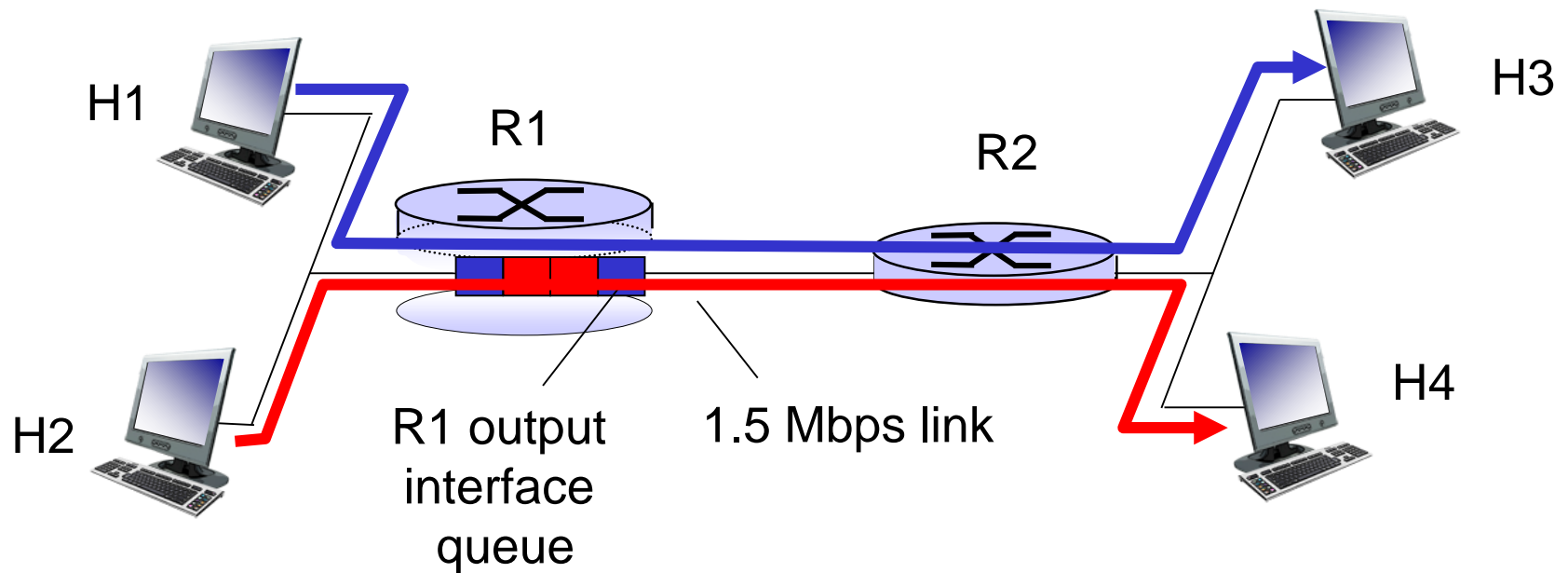
- alternative: multiple classes of service
 - partition traffic into classes
 - network treats different classes of traffic differently (analogy: VIP service versus regular service)
- granularity: differential service among multiple classes, not among individual connections
- history: ToS bits in IP header



7	6	5	4	3	2	1	0
(IP Precedence)			lowdelay	throughput	reliability	lowcost (RFC 1349)	(Must be zero)

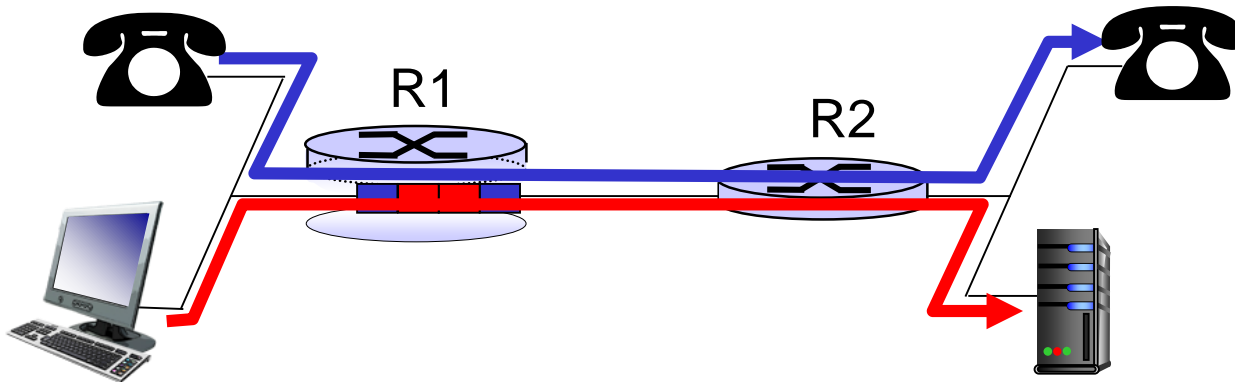
Multiple classes of service: scenario

Flows share communication links and interface queues



Scenario 1: mixed HTTP and VoIP

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

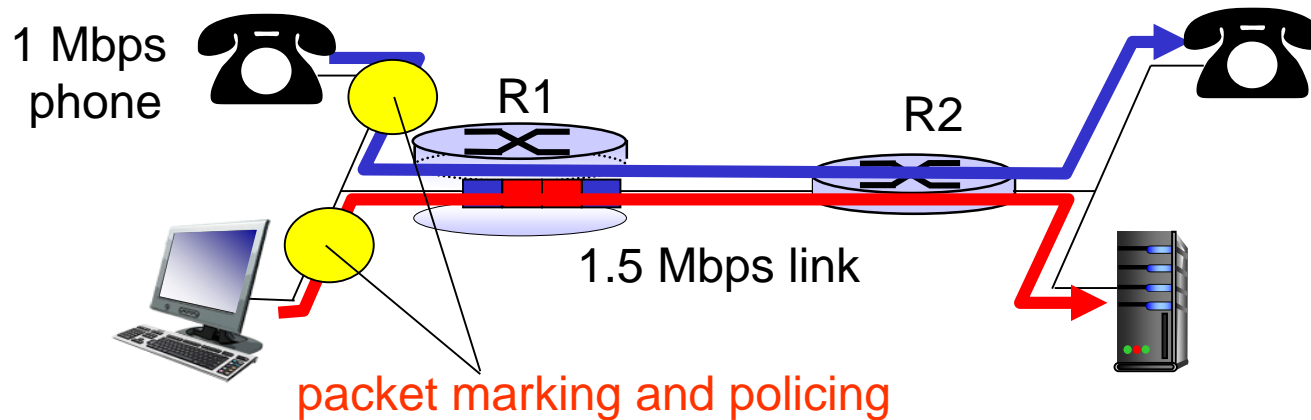


Principle 1

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

Principles for QOS guarantees (more)

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



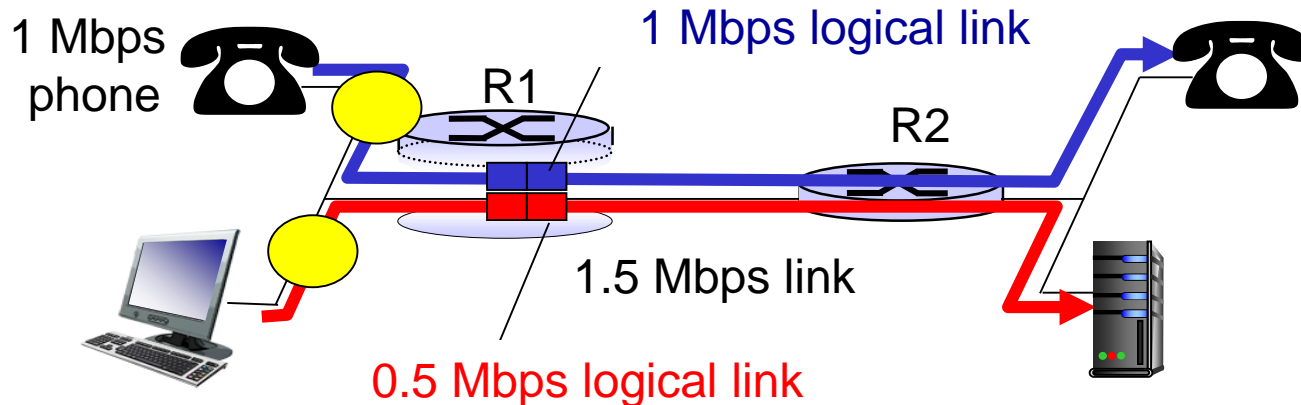
Principle 2

provide **protection (isolation)** for one class from others

Principles for QOS guarantees (more)

□ needs efficient use of bandwidth:

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

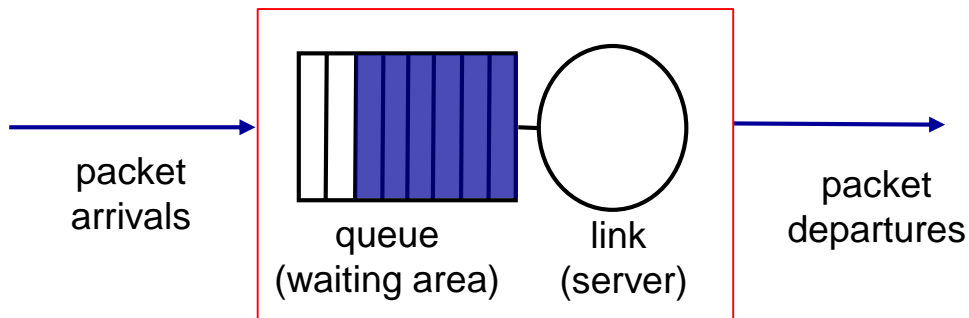


Principle 3

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

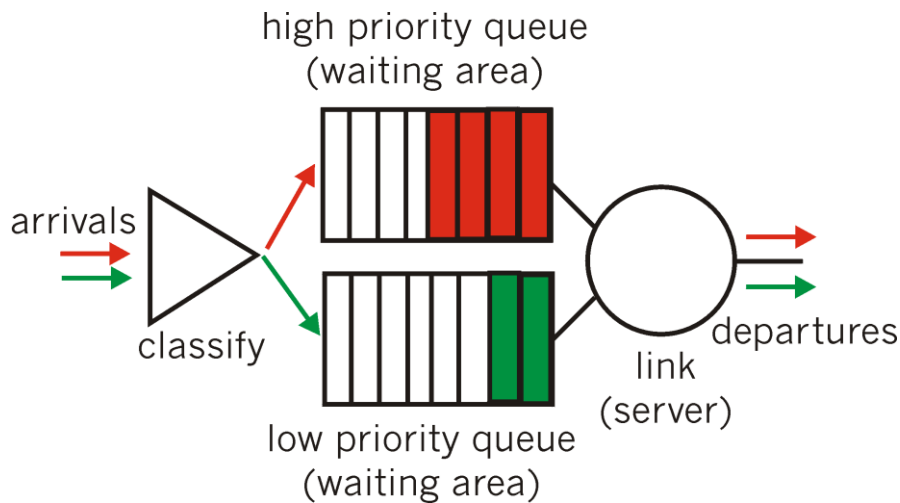


- Commonly used packet scheduling:
 - FCFS: first come first served
 - *priority-based scheduling*: multi-class priority, round robin, weighted fair queueing (WFQ)

Scheduling and policing mechanisms

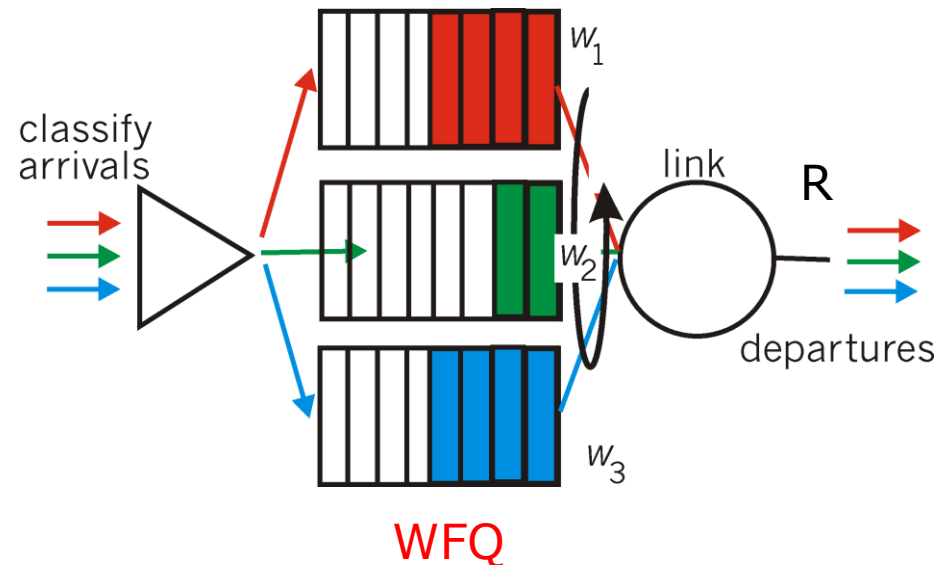
□ *Priority-based* packet scheduling:

- multi-class priority: high priority queue, low priority queue
- round robin
- *weighted fair queueing (WFQ)*



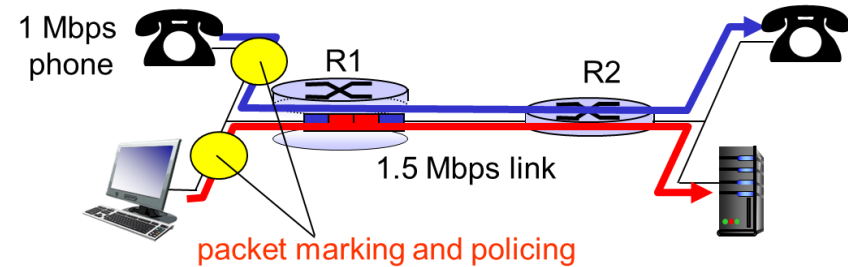
Rate assigned to flow i :

$$R_i = R * (W_i / \sum W_j)$$



Policing mechanisms

policing: limit traffic to not exceed declared parameters

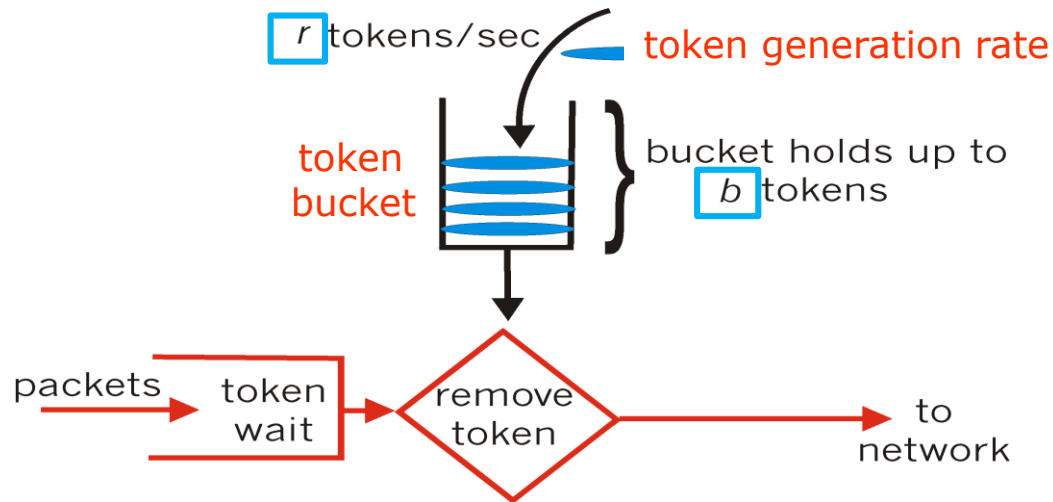


Traffic specification (TSPEC): Three common-used criteria:

- *(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 pps or 6000 ppm have same average!
- *peak rate*: e.g., 6000 ppm avg.; 500 pps peak rate
- *(max.) burst size*: max number of pkts sent consecutively (with no intervening idle)

Policing mechanisms: implementation

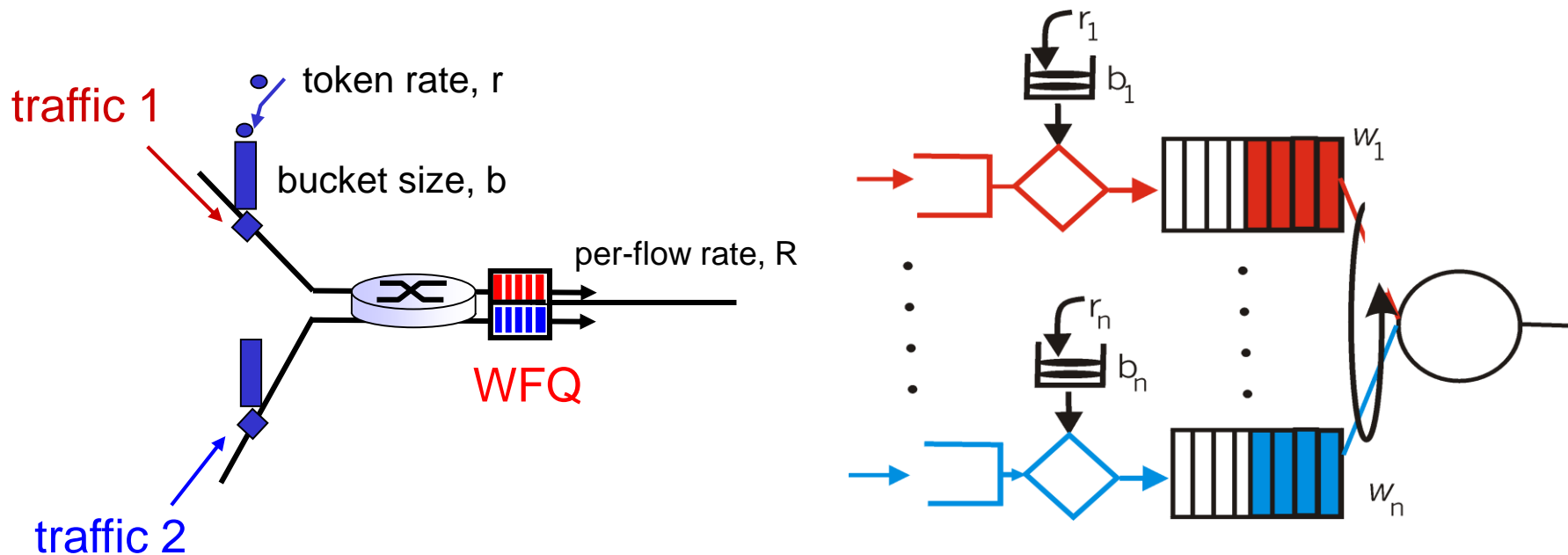
token bucket: limit input to specified *average rate* and *burst size*



- burst size : controlled by bucket size (b)
- average rate: controlled by token generation rate: r tokens/sec
- over interval of length t : number of packets transmitted is less than or equal to $(b + r * t)$

Policing and QoS guarantees

- token bucket combined with WFQ can provide guaranteed upper bound on delay → *QoS guarantee!*



$$D_i^{\max} = \frac{b_i}{R * (w_i / \sum w_k)}$$

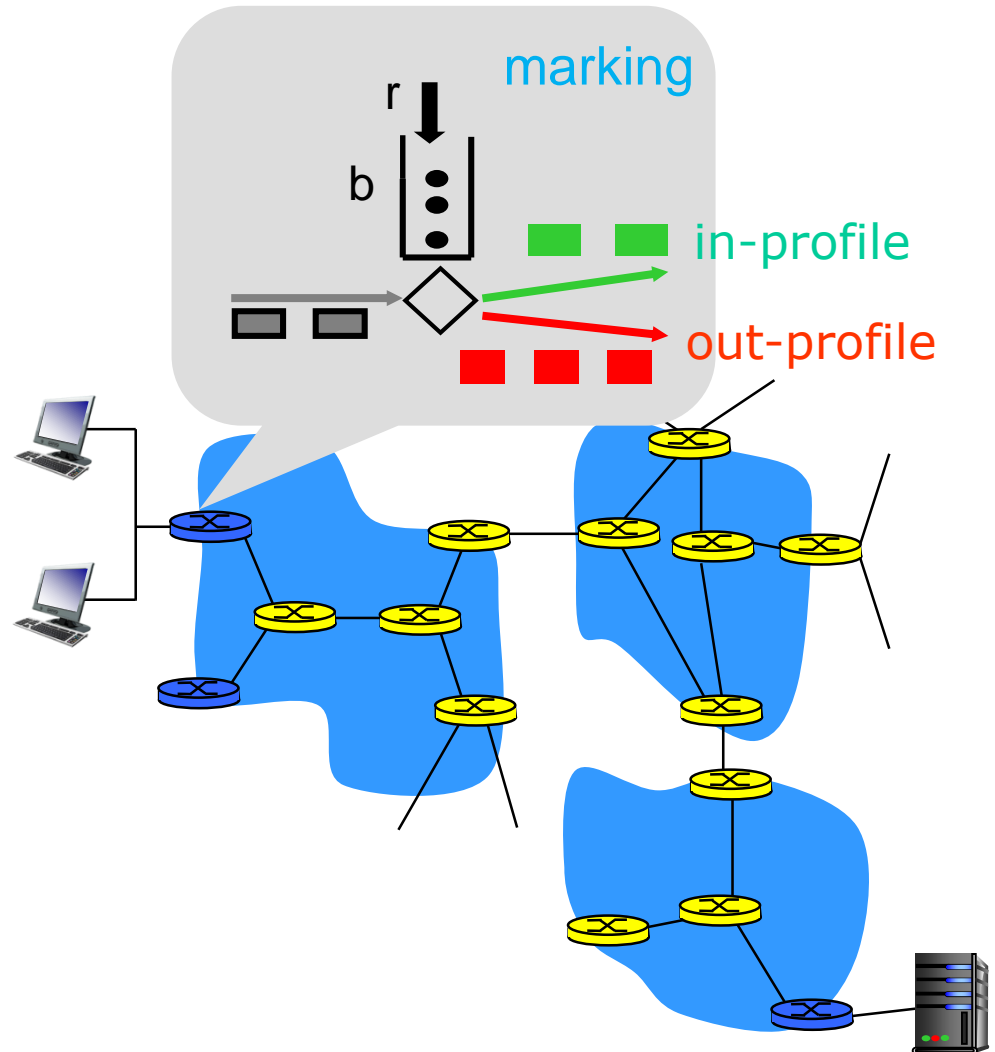
Differentiated services

- per-flow QoS: IntServ (Integrated Service)
 - signaling, maintaining per-flow router state difficult with large number of flows
- “qualitative” service classes: DiffServ (Differentiated Service)
 - relative service distinction: Platinum, Gold, Silver, Normal
 - *scalability: simple functions in network core*, relatively complex functions at edge routers (or hosts)
- DiffServ: define functional components to build service classes


Diffserv architecture

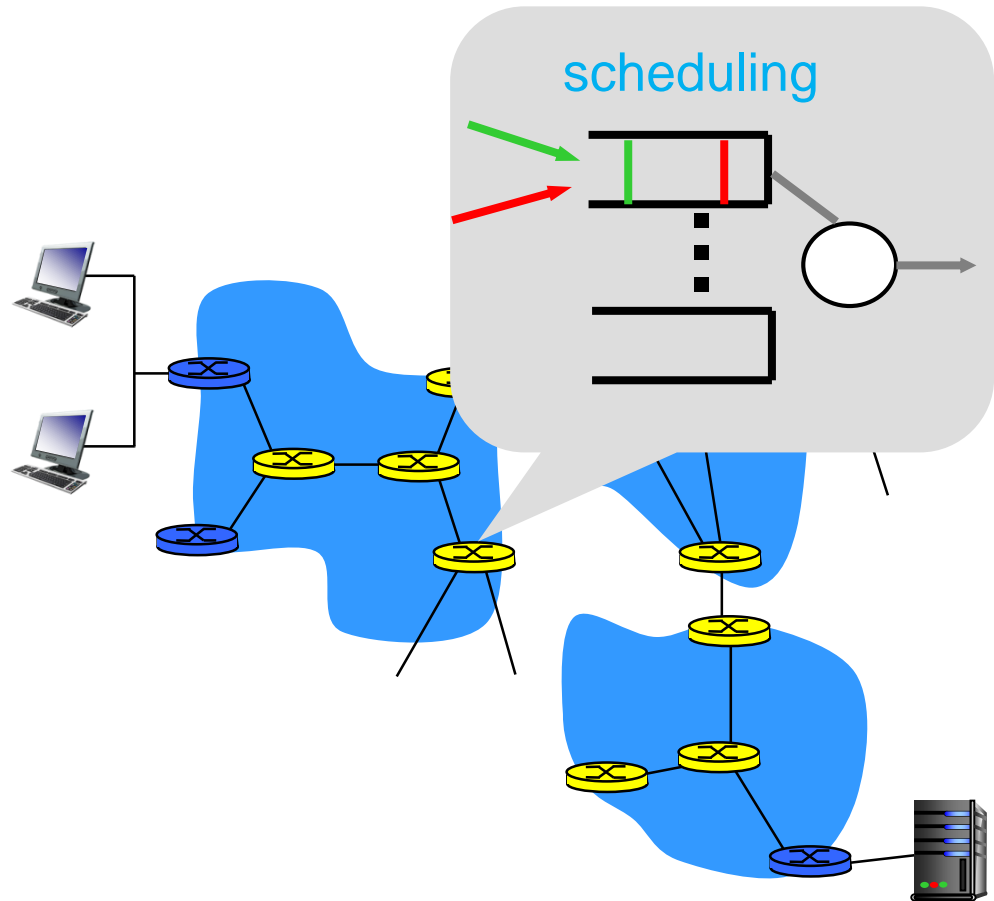
□ edge router: 

- **marking**: marks packets into classes and
- **policing**: marks packets into **in-profile** and **out-profile**



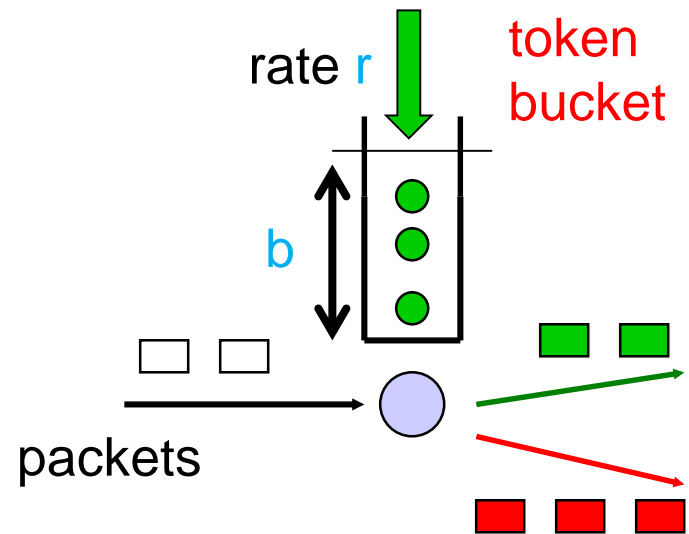
Diffserv architecture

- core router: 
 - per class traffic management
 - buffering and scheduling based on marking at edge
 - dropping: preference given to in-profile packets over out-of-profile packets



Edge-router packet marking

- token bucket profile: pre-negotiated rate r , bucket size b
- packet marking at edge based on per-flow profile
 - class-based marking: packets of different classes marked differently
 - intra-class marking: marks flow into conforming portion of flow (in-profile) and non-conforming one (out-profile)



Diffserv packet marking: details

- packet marking: in **Type of Service (TOS)** in IPv4, and **Traffic Class** in IPv6
- **Differentiated Service Code Point (DSCP):** 6 bits
 - determine **PHB (Per-Hop Behavior)** that the packet will be handled in core router (DiffServ router)
 - 2 bits currently unused



Forwarding Per-hop Behavior (PHB)

- PHB specifies 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

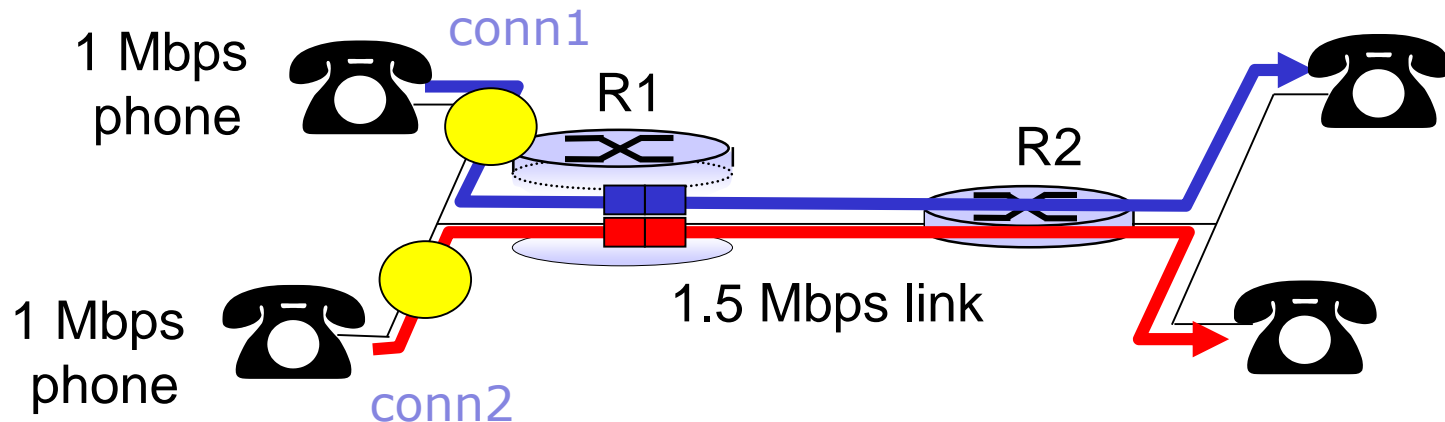
Forwarding PHB

PHBs proposed:

- *expedited forwarding*: packet departure rate of a class equals or exceeds specified rate
 - assigned to the highest priority class
 - logical link with a **minimum guaranteed rate**
- *assured forwarding*: 4 classes of traffic
 - for each class, guaranteed delivery rate of in-profile packets but not guaranteed to out-profile packets
 - define packet drop preferences for each class
- *Best Effort forwarding*: *does not guarantee QoS*

Per-connection QOS guarantees

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



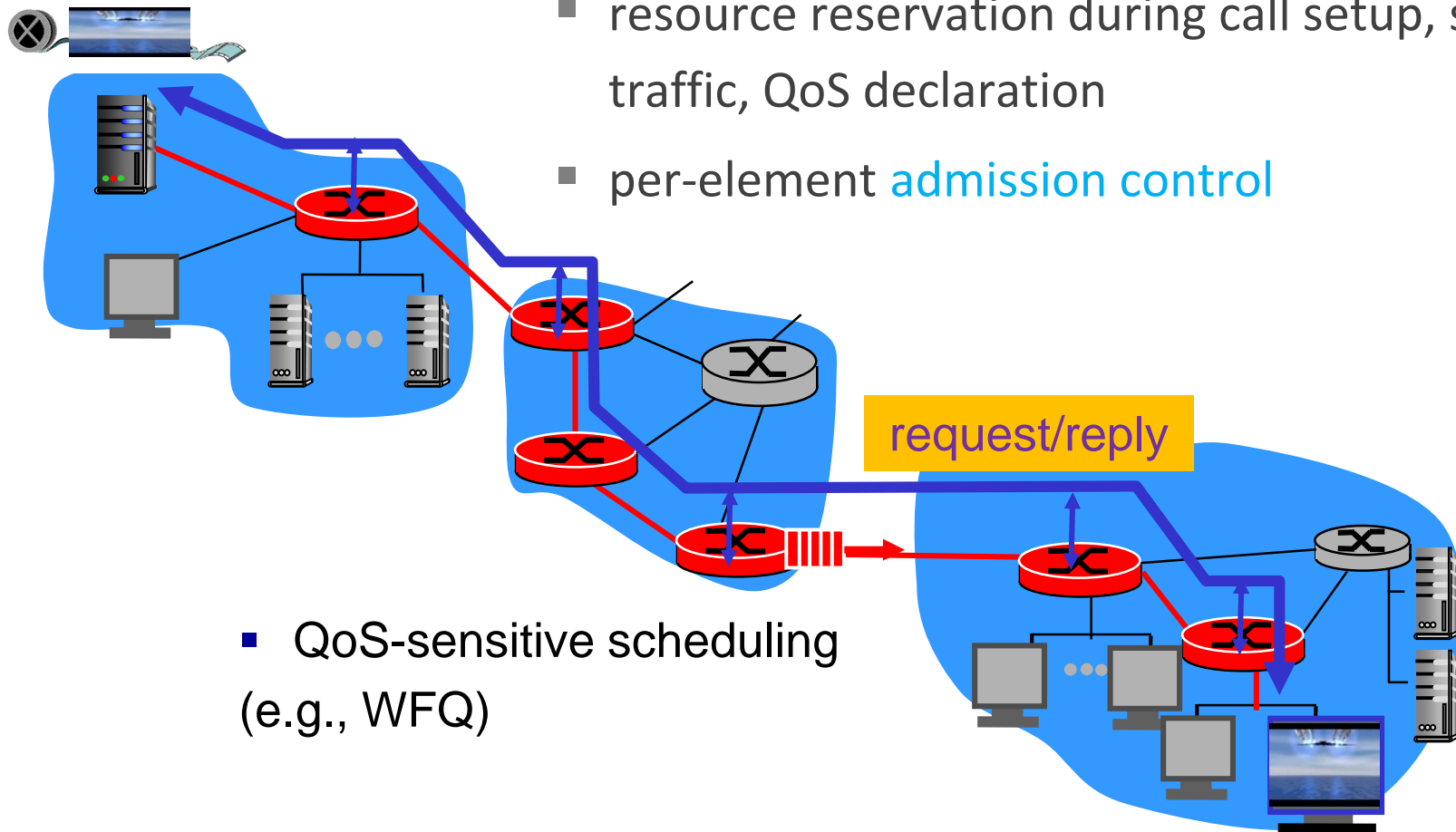
Principle 4

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

Per-connection QoS guarantees

□ *RSVP: call admission protocol*

- resource reservation during call setup, signaling traffic, QoS declaration
- per-element admission control



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