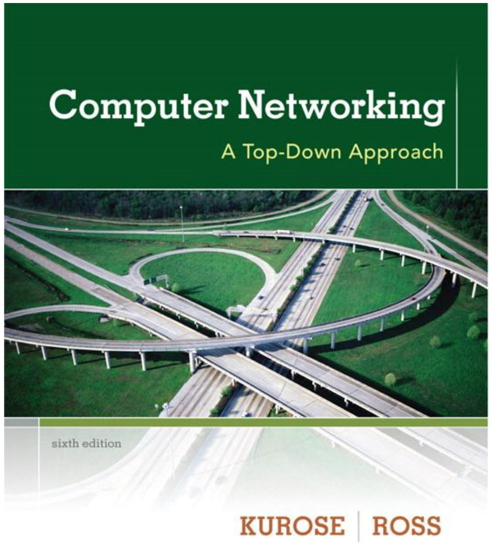
Chapter 7
Multimedia Networking



A note on the use of these ppt slides:

We' re making these slides freely available to all (faculty, students, readers). They' re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we' d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

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

A multimedia network application is a network application that uses audio or video.

Before diving into an in-depth discussion of Internet multimedia applications, it is useful to consider the intrinsic characteristics of the audio and video media themselves.

Properties of Video

Perhaps the most salient characteristic of video is its high bit rate.

Video distributed over the Internet typically ranges from 100 kbps for low-quality video conferencing to over 3 Mbps for streaming high-definition movies.

To get a sense of how video bandwidth demands compare with those of other Internet applications, let's briefly consider three different users, each using a different Internet application.

User 1:

Browsing photos: Flipping through photos every 10 seconds, and that photos are on average 200 Kbytes in size.

User2:

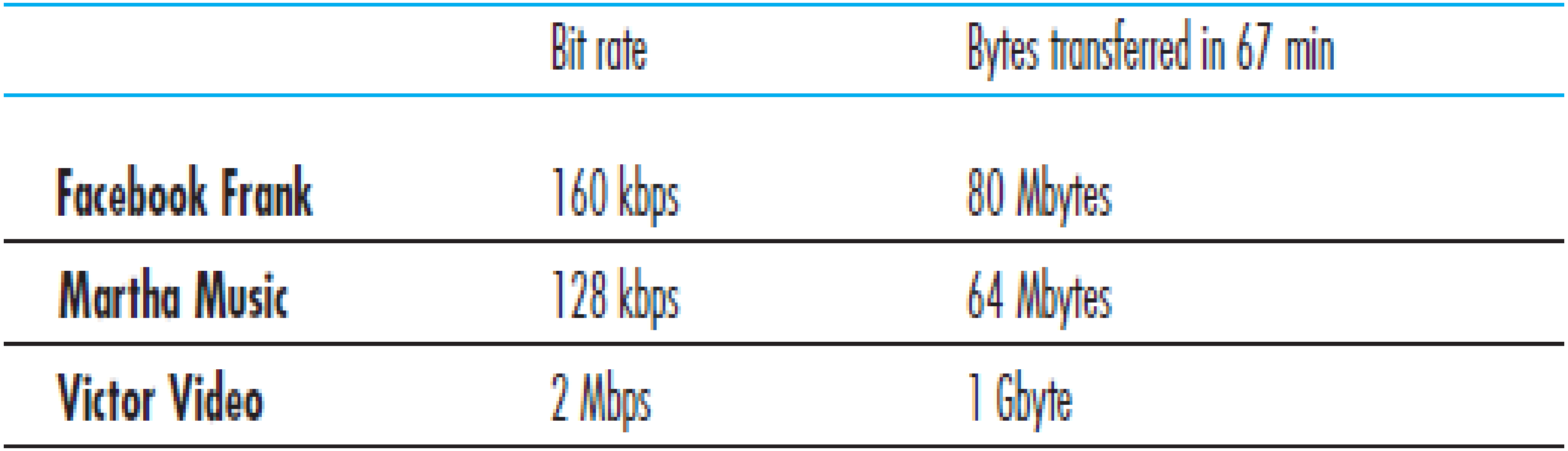
Streaming music from the Internet to the system: Listening to many MP3 songs, one after the other, each encoded at a rate of 128 kbps.

User3:

Watching a video that has been encoded at 2 Mbps.

Finally, let’s suppose that the session length for all three users is 4,000 seconds (approximately 67 minutes).

Table 7.1 compares the bit rates and the total bytes transferred for these 3 users.



	Bit rate	Bytes transferred in 67 min
Facebook Frank	160 kbps	80 Mbytes
Martha Music	128 kbps	64 Mbytes
Victor Video	2 Mbps	1 Gbyte

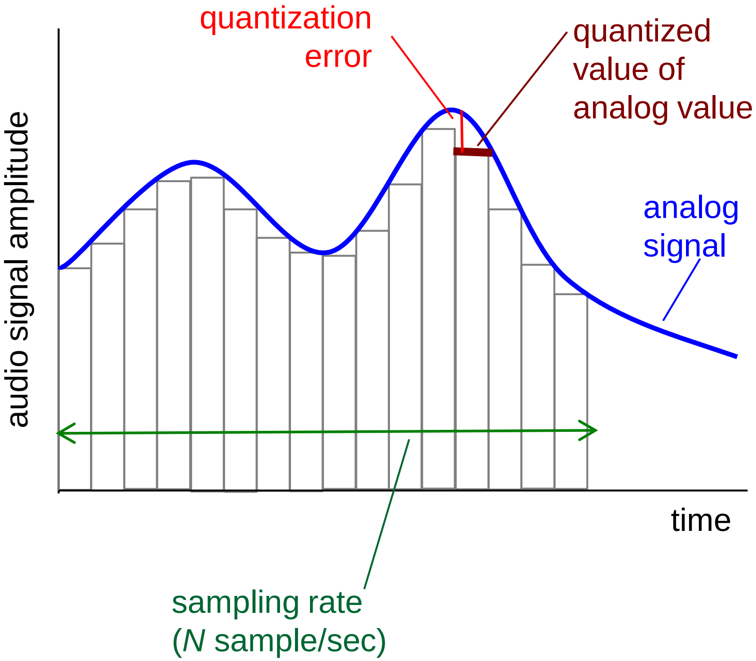
Table 7.1 ♦ Comparison of bit-rate requirements of three Internet applications

Therefore, when designing networked video applications, the first thing we must keep in mind is the high bit-rate requirements of video.

Given the popularity of video and its high bit rate, it was predicted that streaming live and stored video will be approximately 90 percent of global consumer Internet traffic by 2015.

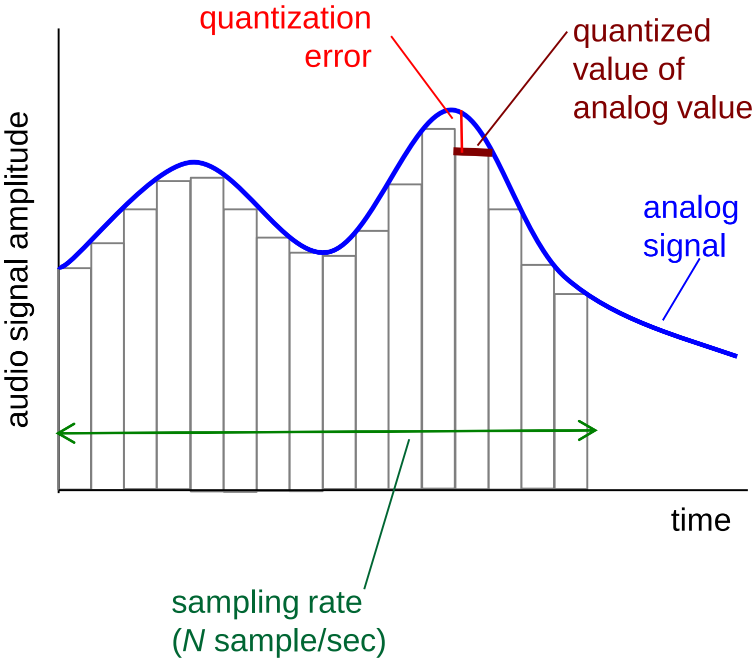
Multimedia: audio

- 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


quantization error
quantized value of analog value
analog signal
audio signal amplitude
time
sampling rate (*N* sample/sec)

Multimedia: audio

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


quantization error
quantized value of analog value
analog signal
audio signal amplitude
time
sampling rate (*N* sample/sec)

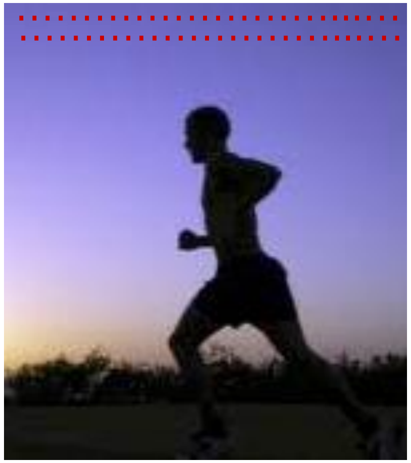
example rates

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

Multimedia: video

- **Another important property of video is that it can be compressed.**
  - 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*

An important characteristic of video is that it can be compressed, thereby trading off video quality with bit rate.

A video is a sequence of images, typically being displayed at a constant rate, for example, at 24 or 30 images per second.

An uncompressed, digitally encoded image consists of an array of pixels, with each pixel encoded into a number of bits to represent luminance and color.

There are two types of redundancy in video, both of which can be exploited by video compression.

i. Spatial redundancy is the redundancy within a given image.
ii. Temporal redundancy reflects repetition from image to subsequent image.

Today's off-the-shelf compression algorithms can compress a video to essentially any bit rate desired.

Of course, the higher the bit rate, the better the image quality and the better the overall user viewing experience. Multiple versions of same video are available.

Multimedia: video

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

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

frame *i*

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

frame *i+1*

Multimedia networking: 3 application types

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

Streaming stored Audio or Video

It has three key distinguishing features:

- *Streaming.*
- *Interactivity.*
- *Continuous playout.*

The most important performance measure for streaming video is average throughput.

In order to provide continuous playout, the network must provide an average throughput to the streaming application that is at least as large as the bit rate of the video itself.

By using buffering and prefetching, it is possible to provide continuous playout even when the throughput fluctuates, as long as the average throughput (averaged over 5–10 seconds) remains above the video rate.

For many streaming video applications, prerecorded video is stored on, and streamed from, a CDN or P2P network.

Conversational Voice- and Video-over-IP

Also called as **Internet telephony or Voice-over-IP (VoIP)**.

Conversational voice and video are widely used in the Internet today, with the Internet companies Skype, QQ, and Google Talk.

Application requirements: Timing considerations and tolerance of data loss.

The audio and video conversational applications are highly **delay-sensitive**.
On the other hand, conversational multimedia applications are **loss-tolerant**—occasional loss only causes occasional glitches in audio/video playback, and these losses can often be partially or fully concealed.

These delay-sensitive but loss-tolerant characteristics are clearly different from those of elastic data applications such as Web browsing, e-mail, social networks, and remote login.

Streaming Live Audio and Video

This class of applications is similar to traditional broadcast radio and television, except that transmission takes place over the Internet.

These applications allow a user to receive a live radio or television transmission—such as a live sporting event or an ongoing news event—transmitted from any corner of the world.

Can be accomplished through multicasting techniques or multiple unicasting techniques. May use CDN or P2P networks.

As with streaming stored multimedia, the network must provide each live multimedia flow with an average throughput that is larger than the video consumption rate. Techniques used are similar to streaming stored content.

Because the event is live, delay can also be an issue.

Delays of up to ten seconds or so from when the user chooses to view a live transmission to when playout begins can be tolerated.

Streaming stored video systems can be classified into three categories:

UDP streaming, HTTP streaming, and adaptive HTTP streaming.

A common characteristic of all three forms of video streaming is the extensive use of client-side application buffering to mitigate the effects of varying end-to-end delays and varying amounts of available bandwidth between server and client.

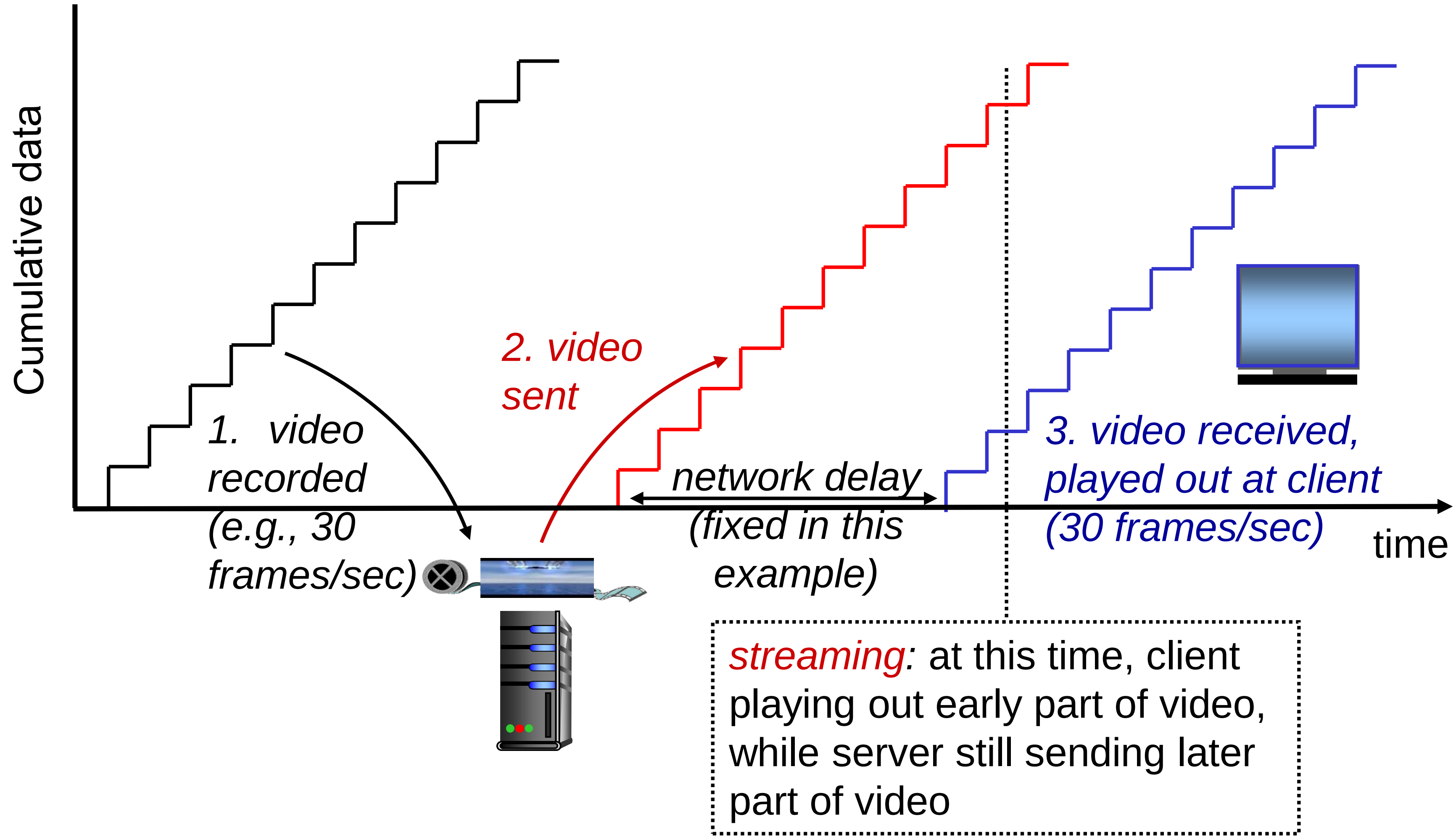
There are two important advantages provided by such client buffering.

First, client side buffering can absorb variations in server-to-client delay.

If a particular piece of video data is delayed, as long as it arrives before the reserve of received-but-not yet- played video is exhausted, this long delay will not be noticed.

Second, if the server-to-client bandwidth briefly drops below the video consumption rate, a user can continue to enjoy continuous playback, again as long as the client application buffer does not become completely drained.

Streaming stored video:


Cumulative data
1. video recorded (e.g., 30 frames/sec)
2. video sent
network delay (fixed in this example)
3. video received, played out at client (30 frames/sec)
time

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

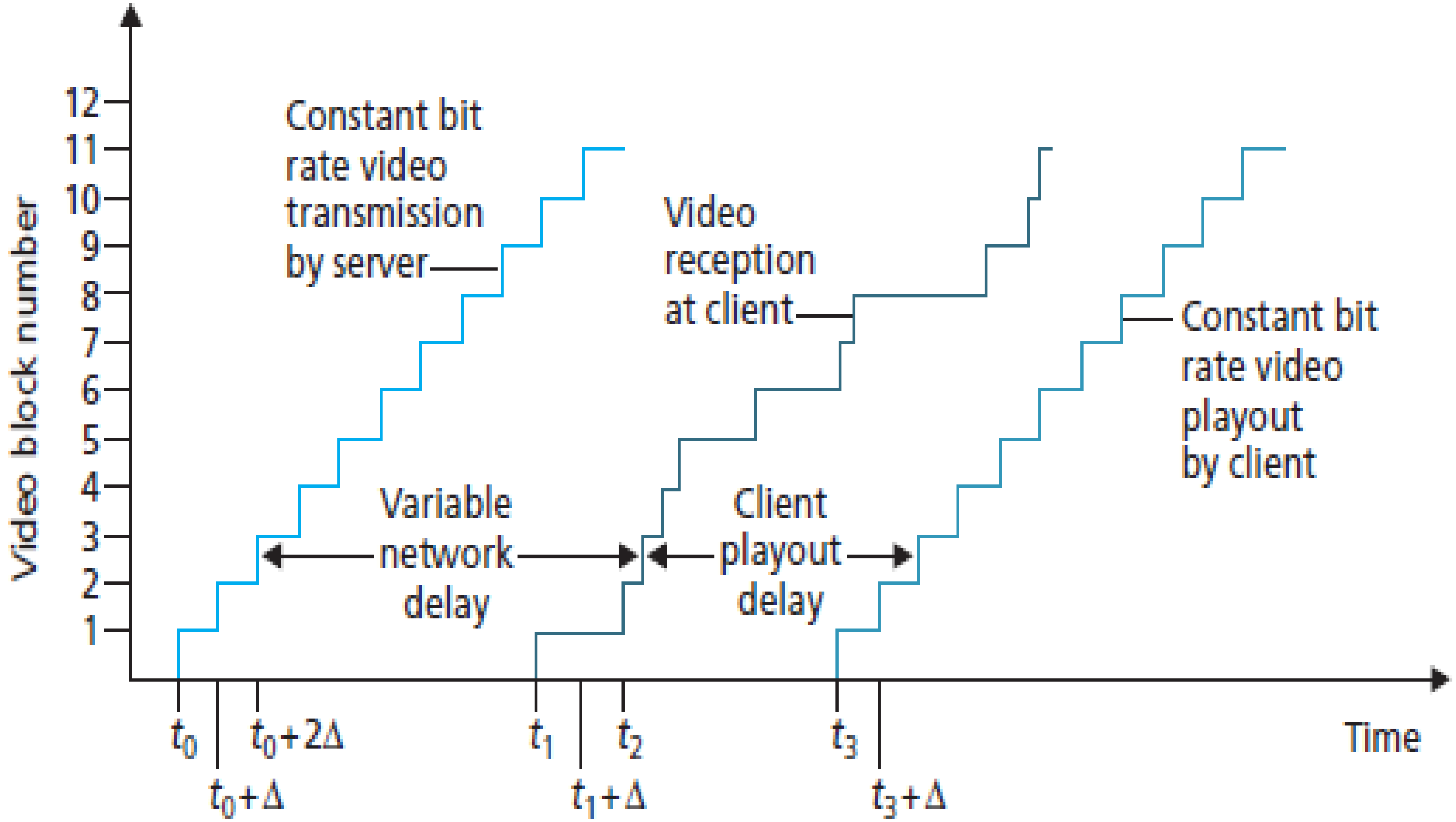
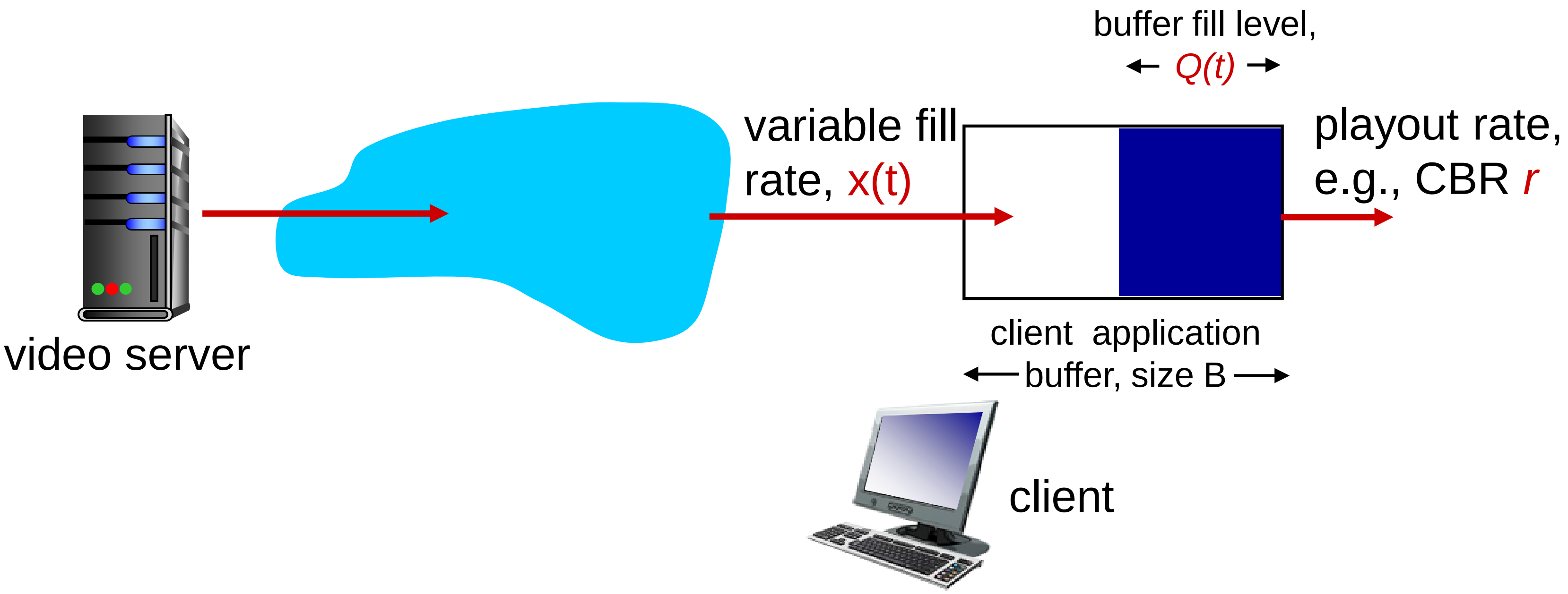


Figure 7.1 ♦ Client playout delay in video streaming

Client-side buffering, playout



Streaming multimedia: UDP

- The server transmits video at a rate that matches the client’s video consumption rate by clocking out the video chunks over UDP at a steady rate.

- For example, if the video consumption rate is 2 Mbps and each UDP packet carries 8,000 bits of video, then the server would transmit one UDP packet every 4 msec.

- As UDP does not employ a congestion-control mechanism, the server can push packets into the network at the consumption rate of the video without the rate-control restrictions of TCP.

- UDP streaming typically uses a small client-side buffer, big enough to hold less than a second of video.

- Before passing the video chunks to UDP, the server will encapsulate the video chunks using the Real-Time Transport Protocol (RTP).

- Another distinguishing property of UDP streaming is that the client and server maintain a separate control connection over which the client sends commands regarding session state changes (such as pause, resume, reposition etc.,). This control connection is in many ways analogous to the FTP control connection.

- The Real-Time Streaming Protocol (RTSP) is a popular open protocol for such a control connection.

- Although UDP streaming has been employed in many open-source systems and proprietary products, it suffers from three significant drawbacks.

- First, due to the unpredictable and varying amount of available bandwidth between server and client, constant-rate UDP streaming can fail to provide continuous playout.

- The second drawback of UDP streaming is that it requires a media control server, such as an RTSP server, to process client-to-server interactivity requests and to track client state (e.g., the client’s playout point in the video, whether the video is being paused or played, and so on) for each ongoing client session. This increases the overall cost and complexity of deploying a large-scale video-on-demand system.

- The third drawback is that many firewalls are configured to block UDP traffic, preventing the users behind these firewalls from receiving UDP video.

HTTP Streaming

In HTTP streaming, the video is simply stored in an HTTP server as an ordinary file with a specific URL.

When a user wants to see the video, the client establishes a TCP connection with the server and issues an HTTP GET request for that URL.

The server then sends the video file, within an HTTP response message, as quickly as possible, that is, as quickly as TCP congestion control and flow control will allow.

On the client side, the bytes are collected in a client application buffer. Once the number of bytes in this buffer exceeds a predetermined threshold, the client application begins playback.

- In HTTP streaming the transmission rate may vary in a “saw-tooth” manner and the packets can also be significantly delayed.

Advantages:

- The use of HTTP over TCP allows the video to traverse through firewalls and NATs.

- There is no need for RTSP server, thereby reducing the cost.

- Most video streaming applications today—including YouTube and Netflix—use HTTP streaming (over TCP) as its underlying streaming protocol.

- Video prefetching on client side occurs naturally with TCP streaming, since TCP's congestion avoidance mechanism will attempt to use all of the available bandwidth between server and client.

- [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.

Client Application Buffer and TCP Buffers

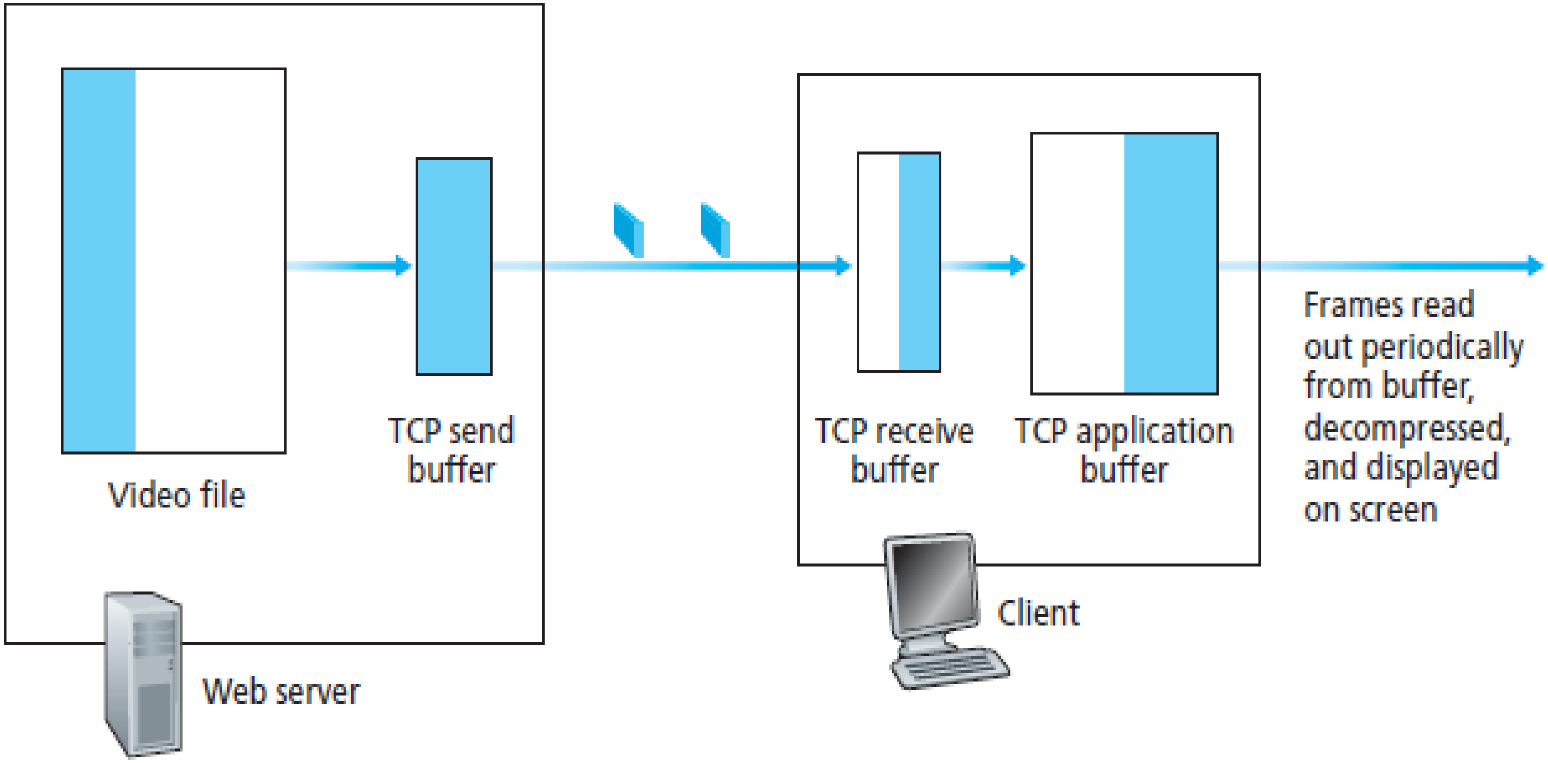


Figure 7.2 ♦ Streaming stored video over HTTP/TCP

Back pressure occurs if the user pauses the video, if the client application buffer becomes full.

Note that when the client application removes *f* bits, it creates room for *f* bits in the client application buffer, which in turn allows the server to send *f* additional bits.

Thus, the server send rate can be no higher than the video consumption rate at the client.

Therefore, *a full client application buffer indirectly imposes a limit on the rate at which the video can be sent from server to client when streaming over HTTP.*

Analysis of Video Streaming

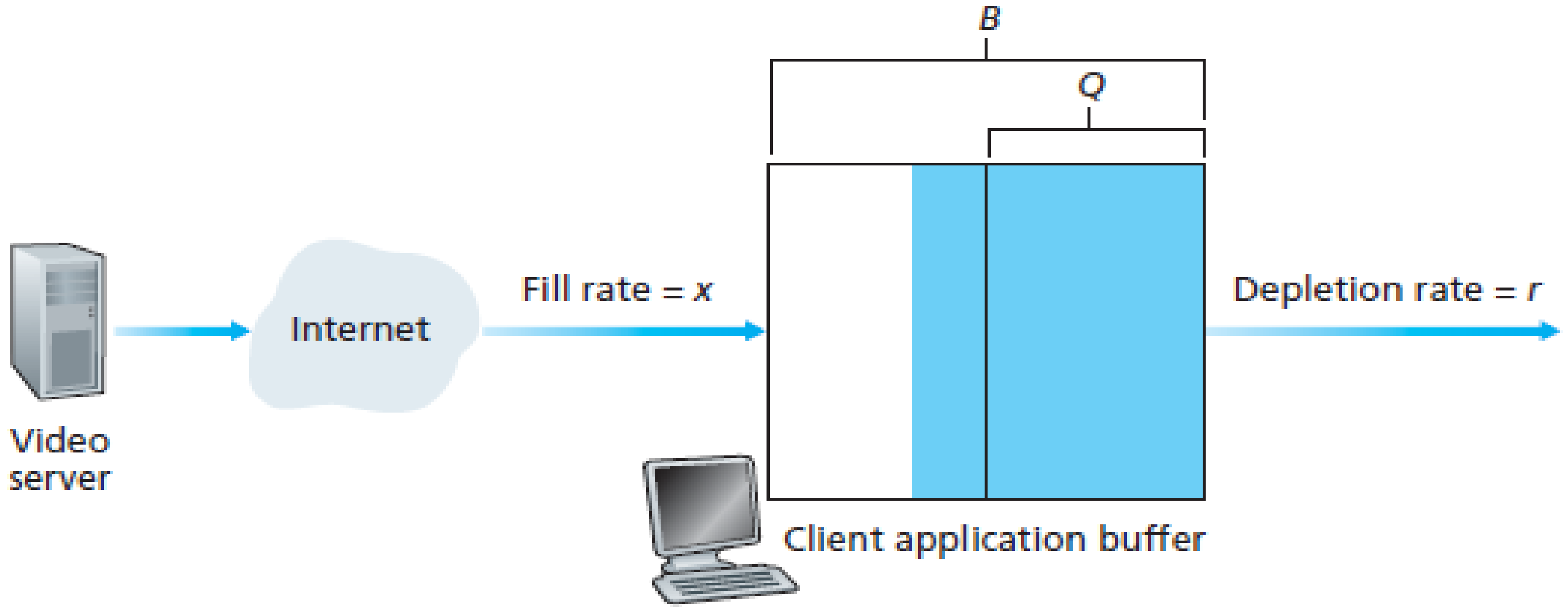


Figure 7.3 ♦ Analysis of client-side buffering for video streaming

The amount of time required to build up Q bits (the initial buffering delay) is $t_p = Q/x$ where t_p is the playout time.

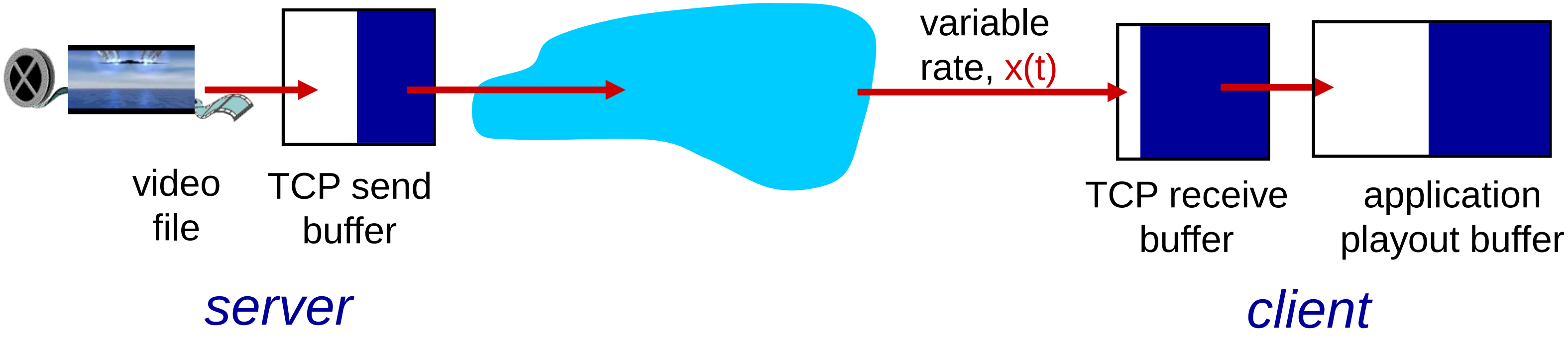
If $x < r$, that is the available rate in the network is less than the video rate, playout will alternate between periods of continuous playout and periods of freezing.

- When x > r. In this case, starting at time t_p , the buffer increases from Q to B at the rate x-r since bits are being depleted at rate r but are arriving at rate x.

- ***Thus when the available rate in the network is more than the video consumption rate, after the initial buffering delay, the user will enjoy continuous playout until the video ends.***

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

Early Termination and Repositioning the Video

HTTP byte-range header in the HTTP GET request message.

When the user repositions to a new position, the client sends a new HTTP request, indicating with the byte-range header from which byte in the file should the server send data.

When the server receives the new HTTP request, it can forget about any earlier request and instead send bytes beginning with the byte indicated in the byte range request.

Repositioning leads to waste of network bandwidth and server resources.

For example, suppose that the client buffer is full with B bits at some time t0 into the video, and at this time the user repositions to some instant t > t0 + B/r into the video, and then watches the video to completion from that point on.

- In this case, all B bits in the buffer will be unwatched and there is significant wasted bandwidth in the Internet due to early termination, which can be quite costly, particularly for wireless links.

- For this reason, many streaming systems use only a moderate-size client application buffer, or will limit the amount of prefetched video using the byte-range header in HTTP Requests.

Adaptive Streaming and DASH

Although HTTP streaming, has been extensively deployed in practice , it has a major shortcoming:

All clients receive the same encoding of the video, despite the large variations in the amount of bandwidth available to a client, both across different clients and also over time for the same client.

This has led to the development of a new type of HTTP-based streaming, often referred to as Dynamic Adaptive Streaming over HTTP (DASH).

Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *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
  - consults 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*

- *"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)

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

….quite simply: this solution *doesn't scale*

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in PoPs near (but not within) access networks
    - used by Limelight

CDN: "simple" content access scenario

Bob (client) requests video http://netcinema.com/6Y7B23V

- video stored in CDN at http://KingCDN.com/NetC6y&B23V

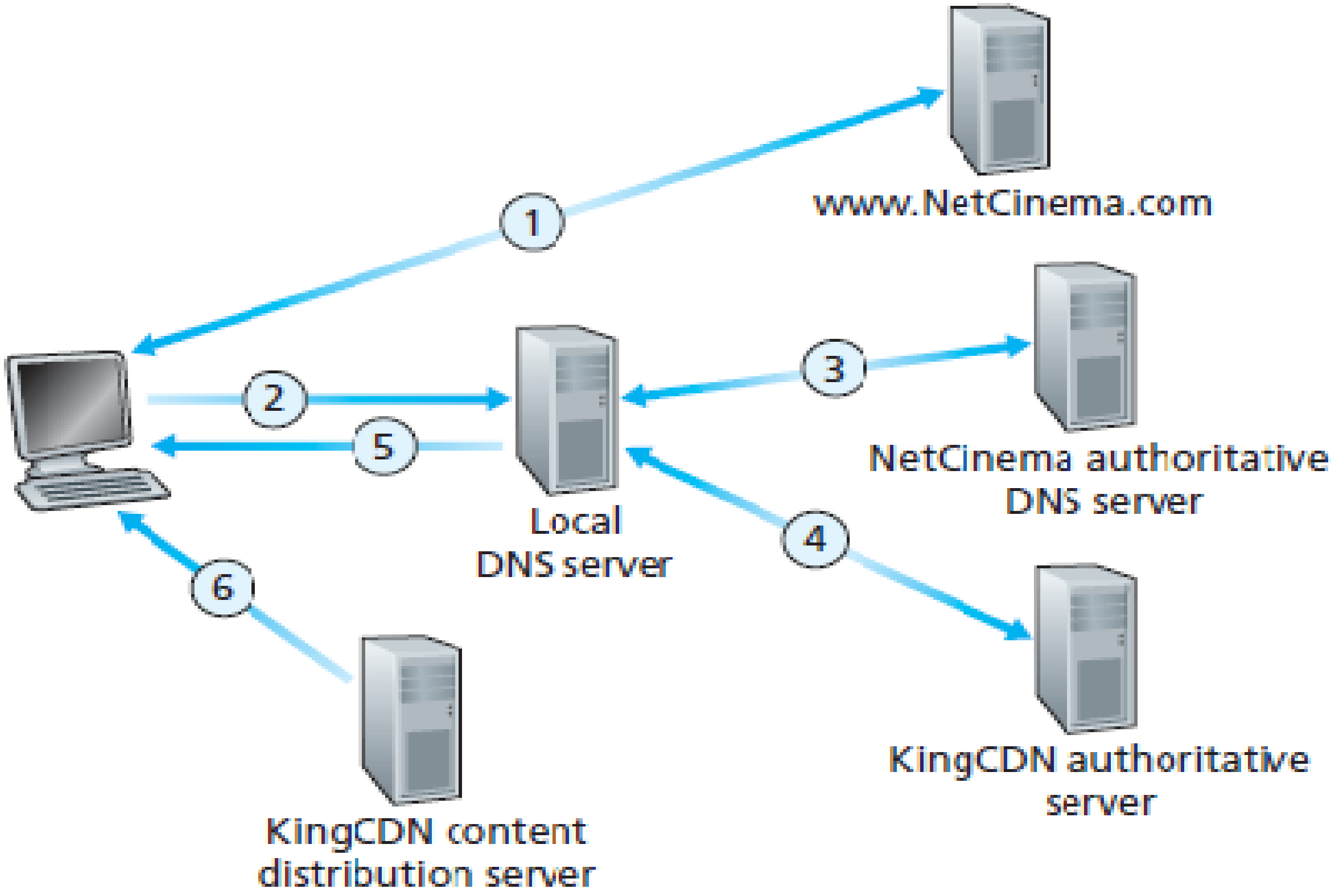

www.NetCinema.com
1
2
5
3
4
6
Local DNS server
NetCinema authoritative DNS server
KingCDN authoritative server
KingCDN content distribution server

Figure 7.4 ♦ DNS redirects a user's request to a CDN server

Google’s CDN infrastructure

It has three tiers of server clusters:

• Eight “mega data centers,” with six located in the United States and two located in Europe, with each data center having on the order of 100,000 servers. These mega data centers are responsible for serving dynamic (and often personalized) content, including search results and gmail messages.

• About 30 “bring-home” clusters, with each cluster consisting on the order of 100–500 servers. The cluster locations are distributed around the world, with each location typically near multiple tier-1 ISP PoPs. These clusters are responsible for serving static content, including YouTube videos.

• Many hundreds of “enter-deep” clusters , with each cluster located within an access ISP. Here a cluster typically consists of tens of servers within a single rack. These enter-deep servers perform TCP splitting and serve static content, including the static portions of Web pages that embody search results.

All of these data centers and cluster locations are networked together with Google’s own private network, as part of one enormous AS (AS 15169).

When a user makes a search query, often the query is first sent over the local ISP to a nearby enter-deep cache, from where the static content is retrieved.

While providing the static content to the client, the nearby cache also forwards the query over Google’s private network to one of the mega data centers, from where the personalized search results are retrieved.

For a YouTube video, the video itself may come from one of the bring-home caches, whereas portions of the Web page surrounding the video may come from the nearby enter-deep cache, and the advertisements surrounding the video come from the data centers.

In summary, except for the local ISPs, the Google cloud services are largely provided by a network infrastructure that is independent of the public Internet.

CDN cluster selection strategy

- *challenge:* how does CDN DNS select "good" CDN node to stream to client

  - pick CDN node geographically closest to client

  Drawback: ignores the variation in delay and available bandwidth over time of Internet paths, always assigning the same cluster to a particular client.

  - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN. DNS-makes periodic real-time measurements of delay and loss performance between their clusters and clients.)

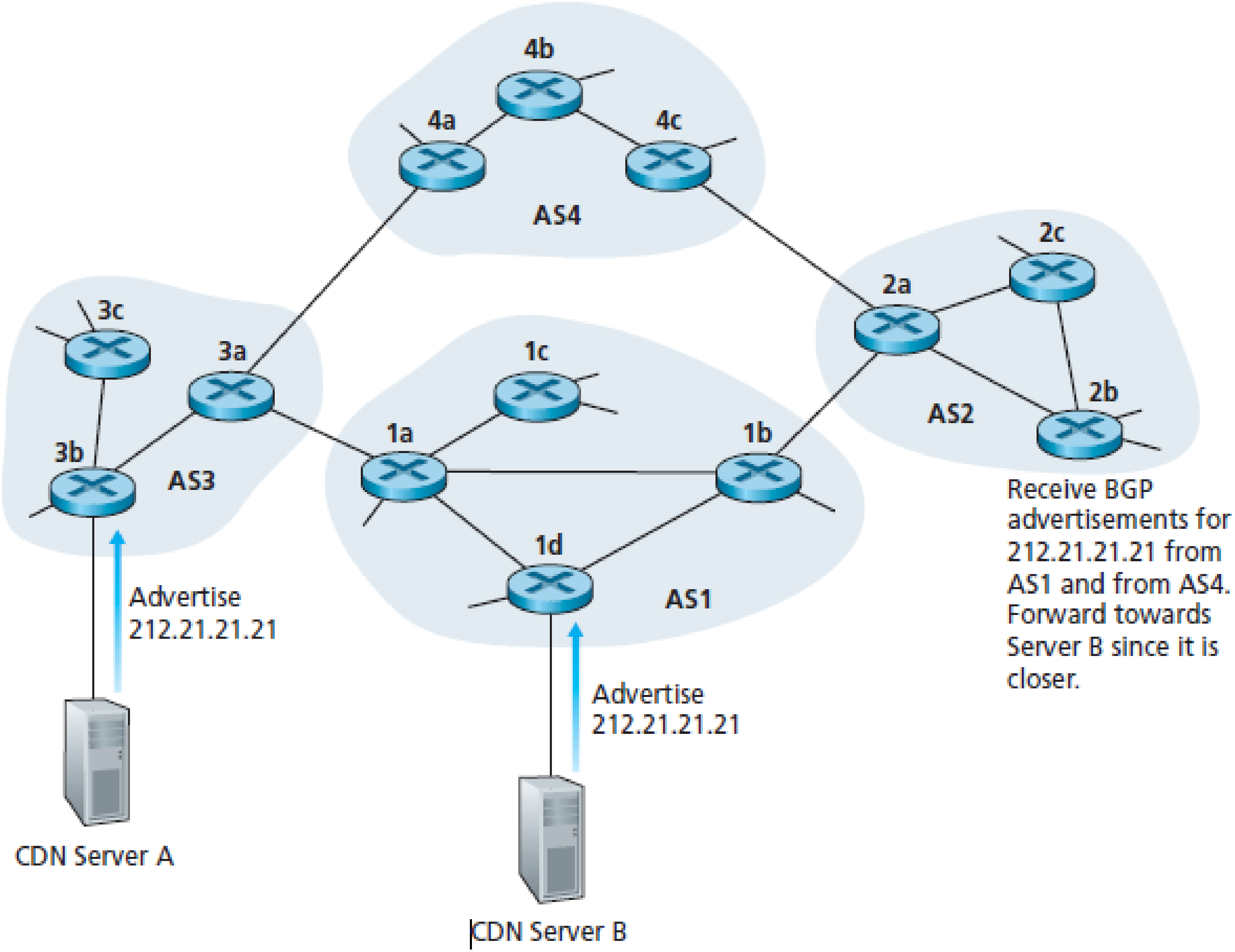
  Drawback: Many LDNSs are configured to not respond to such probes.

- Use the characteristics of recent and ongoing traffic between the clients and CDN servers. For instance, the delay between a client and a cluster can be estimated by examining the gap between server-to-client SYNACK and client-to-server ACK during the TCP three-way handshake.

- Another alternative for cluster-to-client path probing is to use DNS query traffic to measure the delay between clients and clusters. Specifically, during the DNS phase the client’s LDNS can be occasionally directed to different DNS authoritative servers installed at the various cluster locations, yielding DNS traffic that can then be measured between the LDNS and these cluster locations.

- IP anycast

❖ *alternative:* let *client* decide - give client a list of several CDN servers client pings servers, picks “best”



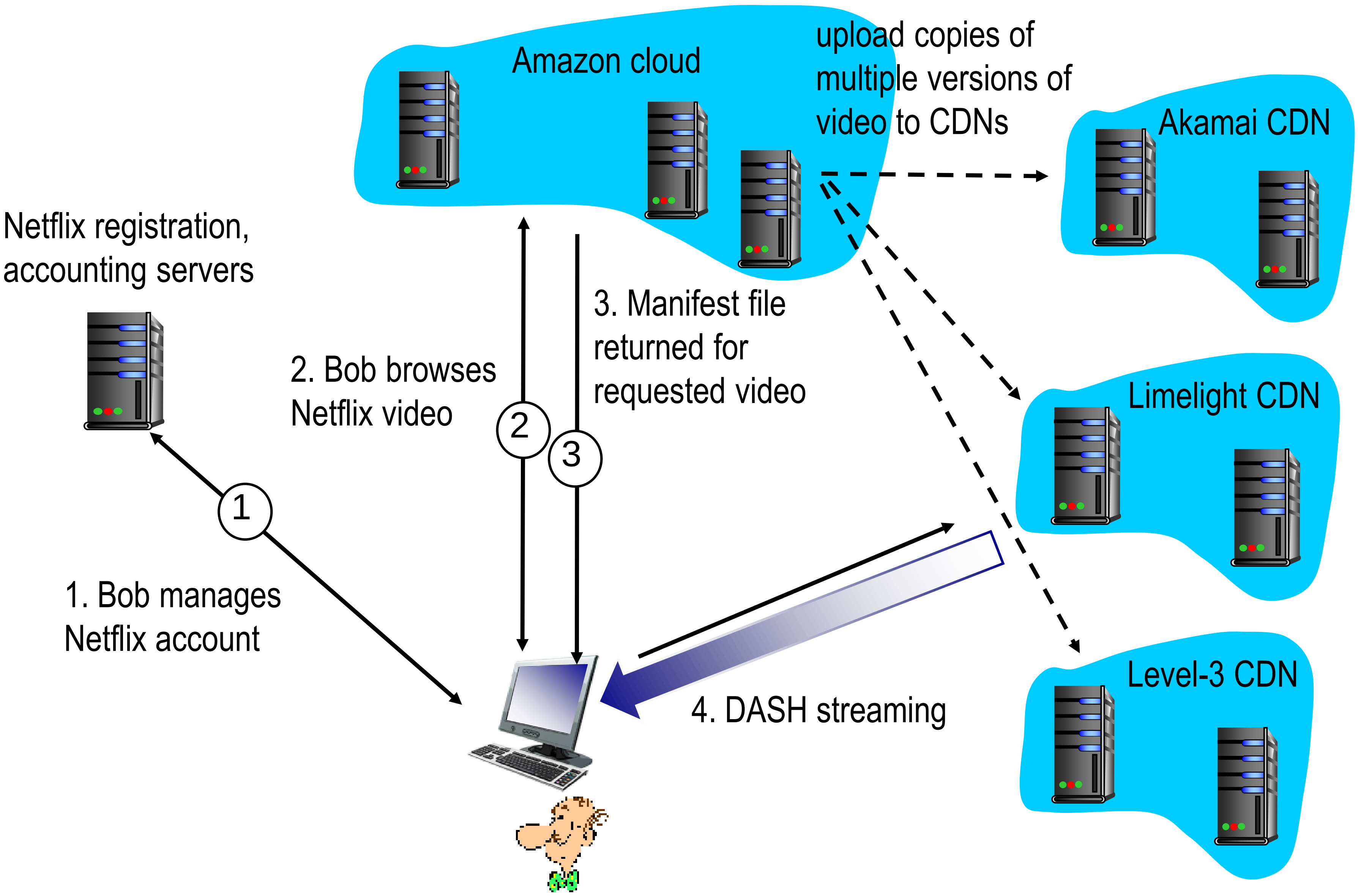
Receive BGP advertisements for 212.21.21.21 from AS1 and from AS4. Forward towards Server B since it is closer.

Figure 7.5 ♦ Using IP anycast to route clients to closest CDN cluster

Case study: Netflix

- 30% downstream US traffic in 2011
- owns very little infrastructure, uses 3rd party services:
  - own registration, payment servers
  - Amazon (3rd party) cloud services:
    - Netflix uploads studio master to Amazon cloud
    - create multiple version of movie (different endodings) in cloud
    - upload versions from cloud to CDNs
    - Cloud hosts Netflix web pages for user browsing
  - *three* 3rd party CDNs host/stream Netflix content: Akamai, Limelight, Level-3

Case study: Netflix


Amazon cloud
upload copies of multiple versions of video to CDNs
Akamai CDN
Limelight CDN
Level-3 CDN
Netflix registration, accounting servers
2. Bob browses Netflix video
3. Manifest file returned for requested video
1
2
3
1. Bob manages Netflix account
4. DASH streaming

Some of the functions taking place in the Amazon cloud include:

- *Content ingestion.*
- *Content processing.*
- *Uploading versions to the CDNs.*
- **Limelight Networks** is a company that provides global (CDN) services that enable organizations to deliver their digital content (e.g. videos, operating system updates, online games, etc.) to any device, anywhere in the world.
- As of December 2014, the company's network has over 80 points-of-presence and 11 Terabits per second of egress capacity across the globe. The company is based in Tempe, Arizona**,** U.S.A.

- **Level3 communications** is a big company that has been providing a wide range of content delivery services for over 20 years. Level3 owns a Tier 1 backbone and has over 5.6 Tbps of peering capacity.

- The Level3 CDN services allow for token-based authentication, traffic prioritization, resource popularity tracking etc.

- **Akamai Technologies, Inc.** is a CDN and cloud service provider headquartered in Cambridge,USA. Akamai's CDN is one of the world's largest distributed computing platforms.

- The company operates a network of servers around the world and rents capacity on the servers to customers who want their websites to work faster by distributing content from locations close to the user.

- Some of its customers are apple, facebook, bing, google, ebay etc.

- The company was founded in 1998 by Daniel M Lewin (then a graduate student at MIT) and MIT applied maths professor Tom Leighton.

Case study: YouTube

- YouTube employs HTTP streaming.
- YouTube uses the HTTP byte range header in request to limit the flow of transmitted data after a target amount of video is prefetched.
- Supports Uploading.
- Google runs the entire YouTube service within its own vast infrastructure of data centers, private CDN, and private global network interconnecting its data centers and CDN clusters.

Case study: Kankan

- Kankan uses P2P delivery instead of client-server (via CDNs) delivery.

- This allows the service provider to significantly reduce its infrastructure and bandwidth costs.

- P2P video delivery approach is used with great success by several companies in China, including Kankan (owned and operated by Xunlei), PPTV (formerly PPLive), and PPs (formerly PPstream).

- Kankan, is currently the leading P2P-based video-on-demand provider in China, with over 20 million unique users viewing its videos every month.

- ✓ *The Kankan design employs a tracker and its own DHT for tracking content.*

- ✓ *Swarm sizes for the most popular content involve tens of thousands of peers, typically larger than the largest swarms in BitTorrent.*

- ✓ *The Kankan protocols—for communication between peer and tracker, between peer and DHT, and among peers—are all proprietary.*

- ✓ *Interestingly, for distributing video chunks among peers, Kankan uses UDP .*

Voice-over-IP (VoIP)

- Real-time conversational voice over the Internet is often referred to as **Internet telephony**, or **Voice-over-IP (VoIP)**.
- *Limitations of Best Effort IP Service.*
- *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
  - real-time conversational applications, are acutely sensitive to packet delay, jitter, and loss.
- the receiver must take more care in determining (1) when to play back a chunk, and (2) what to do with a missing chunk.

VoIP characteristics

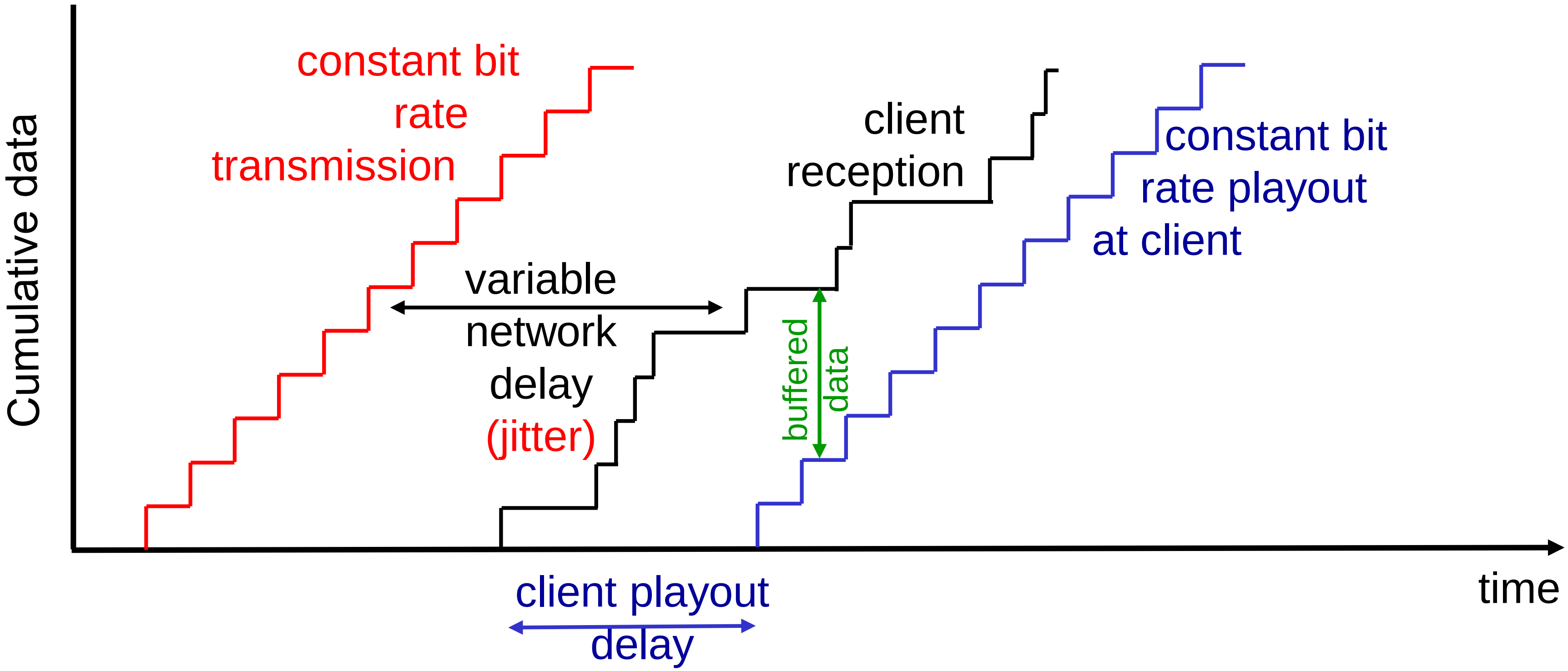
- speaker' s audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
  - pkts generated only during talk spurts
  - 20 msec chunks at 8 Kbytes/sec: 160 bytes of data
- application-layer header added to each chunk
- chunk+header encapsulated into UDP segment
- application sends segment 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: transmission, processing, queueing, propagation; 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 20% can be tolerated

- UDP Vs TCP for Packet Transmission

Delay jitter


Cumulative data
constant bit rate transmission
variable network delay (jitter)
client reception
buffered data
constant bit rate playout at client
client playout delay
time

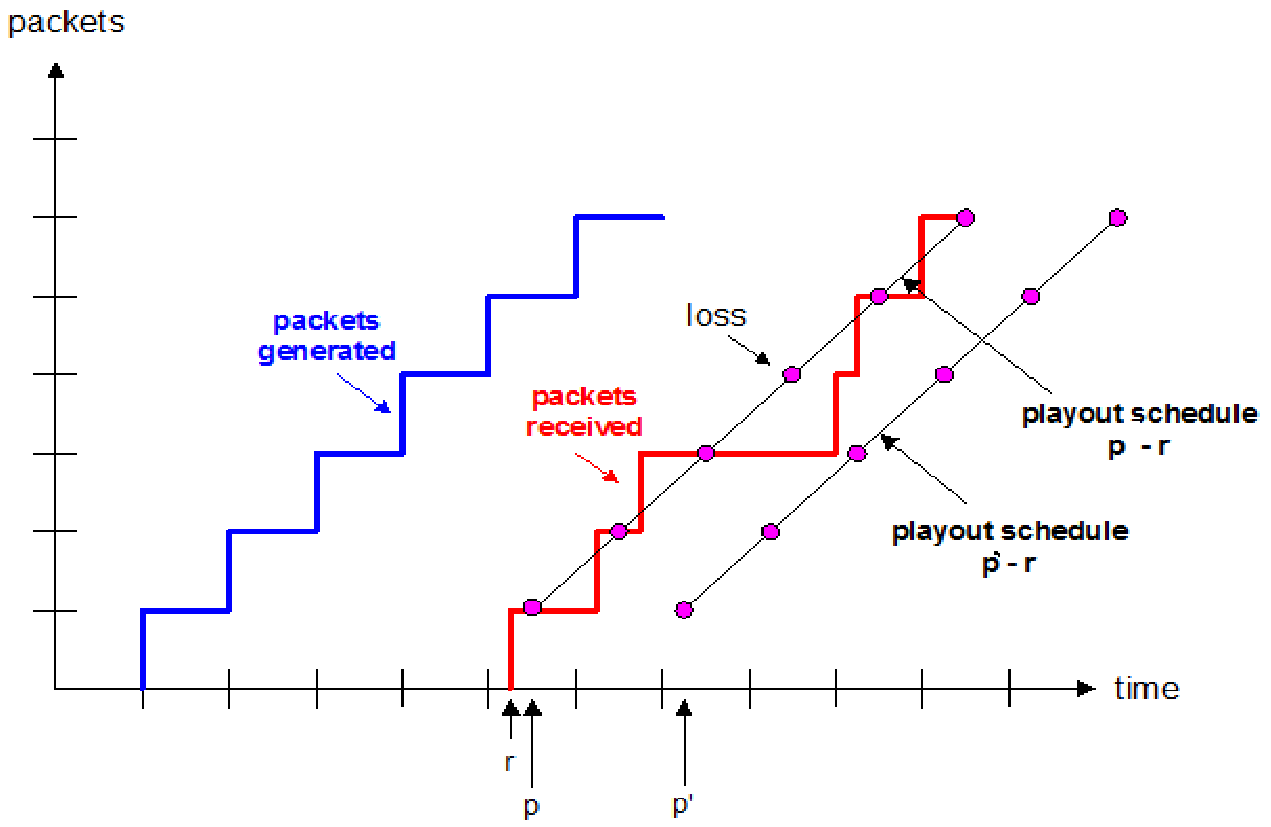
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

VoIP: fixed playout delay

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

VoIP: fixed playout delay

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



Adaptive playout delay (1)

- *goal:* low playout delay, low 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):

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

delay estimate after ith packet

small constant, e.g. 0.01

time received - time sent (timestamp)

measured delay of ith packet

Adaptive playout delay (2)

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

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

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

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

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

where K=4(a +ve constant)

- If packet *j* also belongs to this talk spurt, it is played out at time
- $p_j = t_j + q_i$ *where* $q_i = p_i - t_i$

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.

Recovering from Packet Loss(1)

- There is a need to maintain acceptable audio quality in the presence of packet loss.
- Two types of loss anticipation schemes are **forward error correction (FEC)** and **interleaving**.

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 chunk if at most one is lost
- But if two or more packets in a group are lost, the

receiver cannot reconstruct the lost packets.

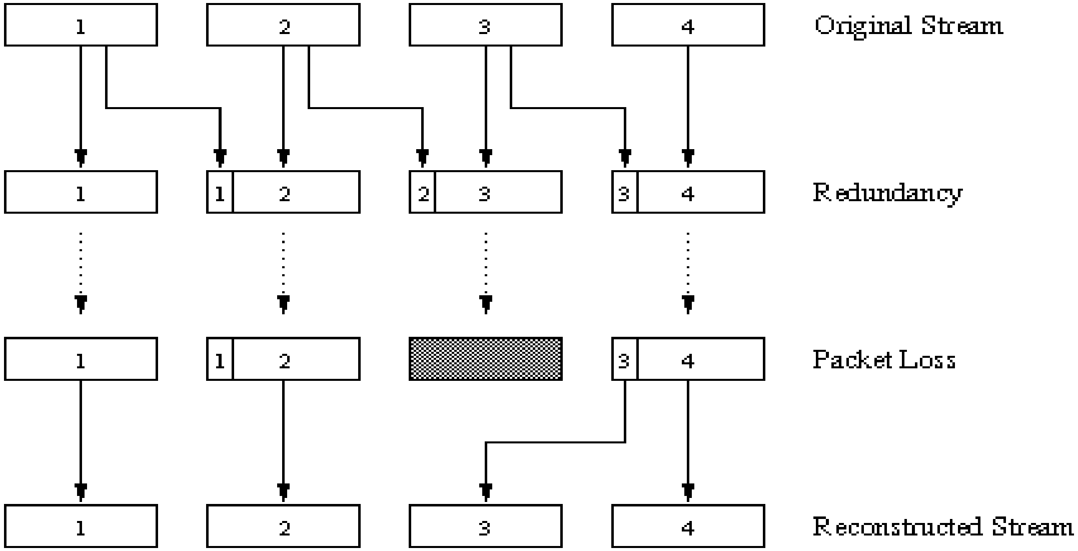
- By keeping *n* + 1, the group size, small, a large fraction of the lost packets can be recovered when loss is not excessive.

- Furthermore, this simple scheme increases the playout delay, as the receiver must wait to receive the entire group of packets before it can begin playout

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

consecutive loss : can also append (n-1)st and (n-2)nd low-bit rate chunk or append the (n – 1)st and (n – 3)rd low-bit rate chunk, and so on.

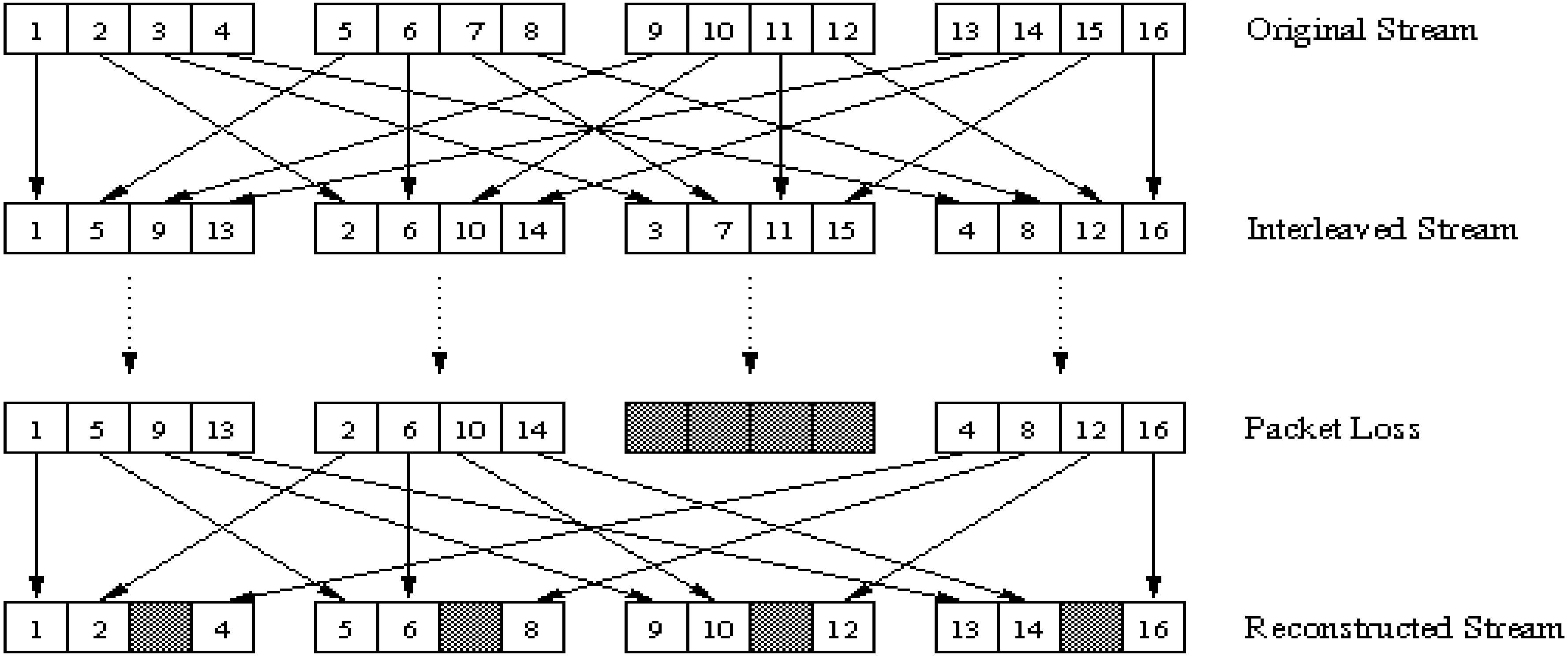
- Note that in this scheme, the receiver only has to receive two packets before playback, so that the increased playout delay is small.

- However, a stream of mostly high-quality chunks,
- occasional low-quality chunks, and no missing chunks gives good overall audio quality.

- By appending more low-bit rate chunks to each nominal chunk, the audio quality at the receiver becomes acceptable for a wider variety of harsh best-effort environments.

- On the other hand, the additional chunks increase the transmission bandwidth and the playout delay.

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

- Interleaving can significantly improve the perceived quality of an audio stream. It also has low overhead.

- The obvious disadvantage of interleaving is that it increases latency.

- This limits its use for conversational applications such as VoIP, although it can perform well for streaming stored audio.

- A major advantage of interleaving is that it does not increase the bandwidth requirements of a stream.

Error Concealment

- Error concealment schemes attempt to produce a replacement for a lost packet that is similar to the original.

- Perhaps the simplest form of receiver-based recovery is packet repetition. Packet repetition replaces lost packets with copies of the packets that arrived immediately before the loss. It has low computational complexity and performs reasonably well.

Another form of receiver-based recovery is interpolation, which uses audio before and after the loss to interpolate a suitable packet to cover the loss.

Interpolation performs somewhat better than packet repetition but is significantly more computationally intensive.

As such, these techniques work for relatively small loss rates (less than 15 percent), and for small packets (4–40 msecs).

When the loss length approaches the length of a phoneme (5–100 msecs) these techniques break down, since whole phonemes may be missed by the listener.

Case Study: Voice-over-IP with Skype

- ***Skype is an immensely popular VoIP application with over 50 million accounts active on a daily basis.***

- ***Skype was acquired by Microsoft in 2011 for over $8 billion.***

- ***Provides host-to-host VoIP service.***

- ***Skype offers host-to-phone services, phone-to-host services, and multi-party host-to-host video conferencing services***

- ***For both voice and video, the Skype uses codecs.***

- ***Video rates for Skype have been measured to be as low as 30 kbps for a low-quality session up to almost 1 Mbps for a high quality session.***

- ***Typically, Skype's audio quality is better than the "POTS".***

Case Study: Voice-over-IP with Skype

- *By default, Skype sends audio and video packets over UDP. However, control packets are sent over TCP.*

- *Skype uses FEC for loss recovery for both voice and video streams sent over UDP.*

- *Skype uses P2P techniques in a number of innovative ways, for file sharing, instant messaging, for user location and for NAT traversal.*

Case Study: Voice-over-IP with Skype

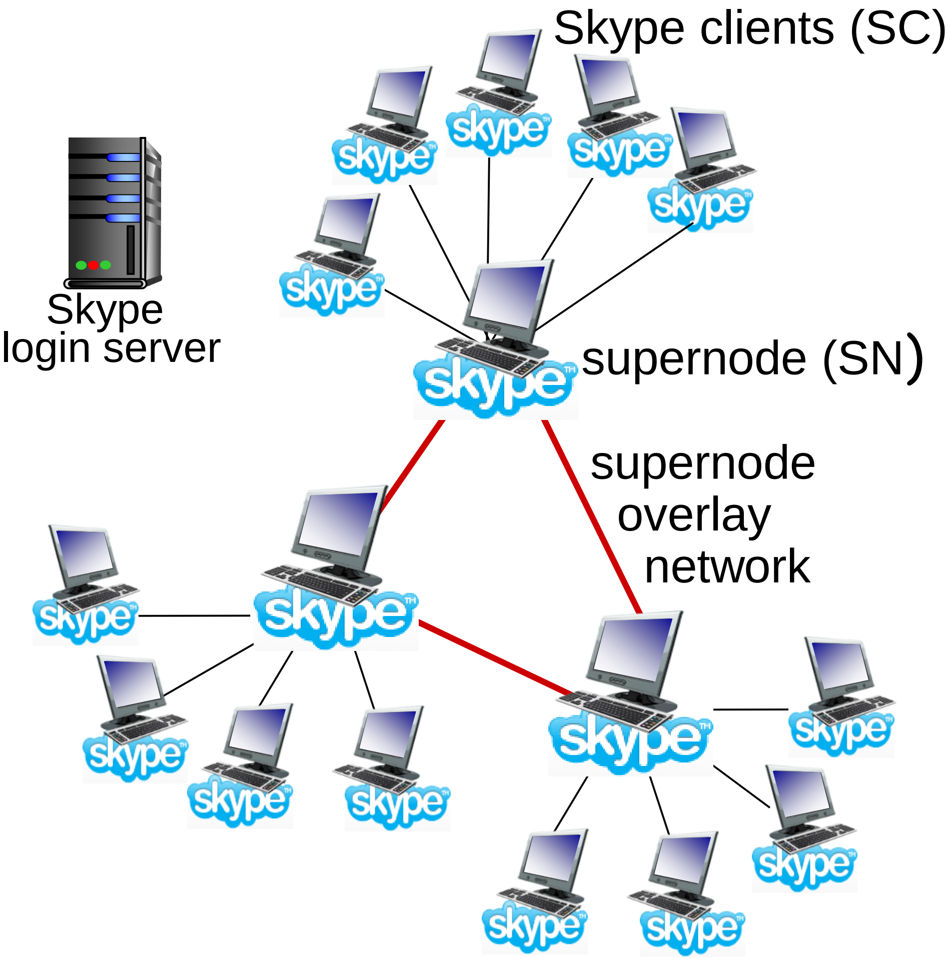
- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:

- clients: skype peers connect directly to each other for VoIP call

- super nodes (SN): skype peers with special functions(indexing)

- overlay network: among SNs to locate SCs

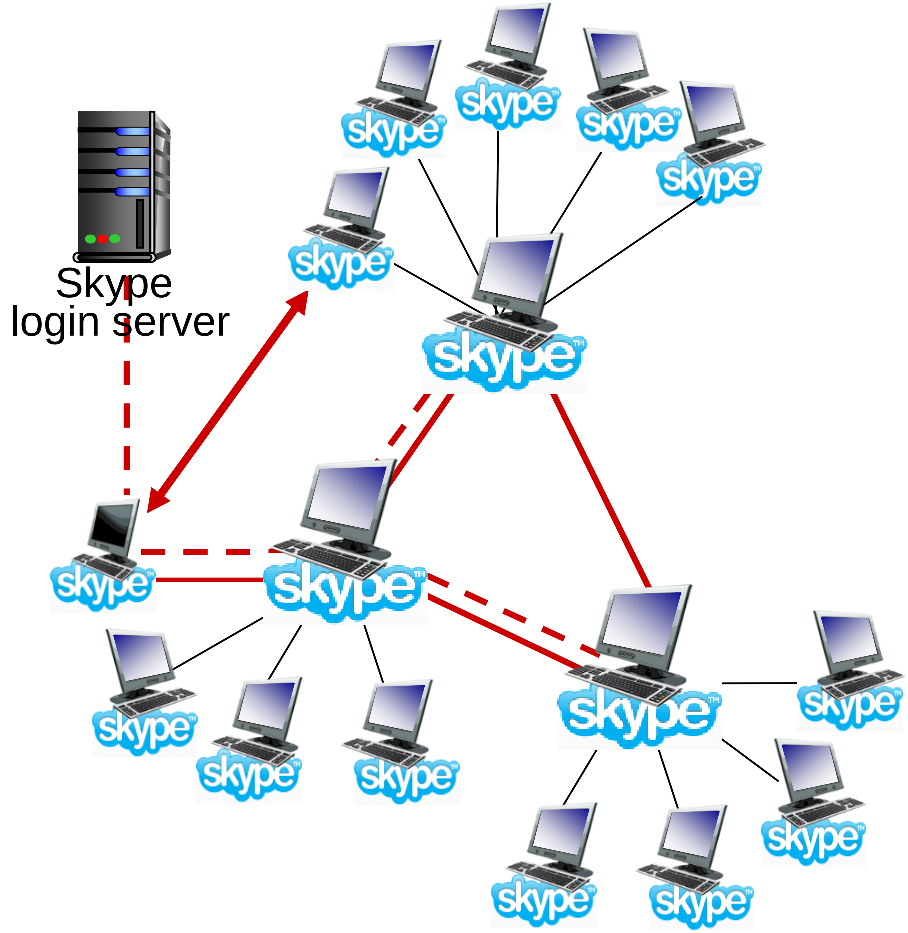
- login server



P2P voice-over-IP: skype

skype client operation:

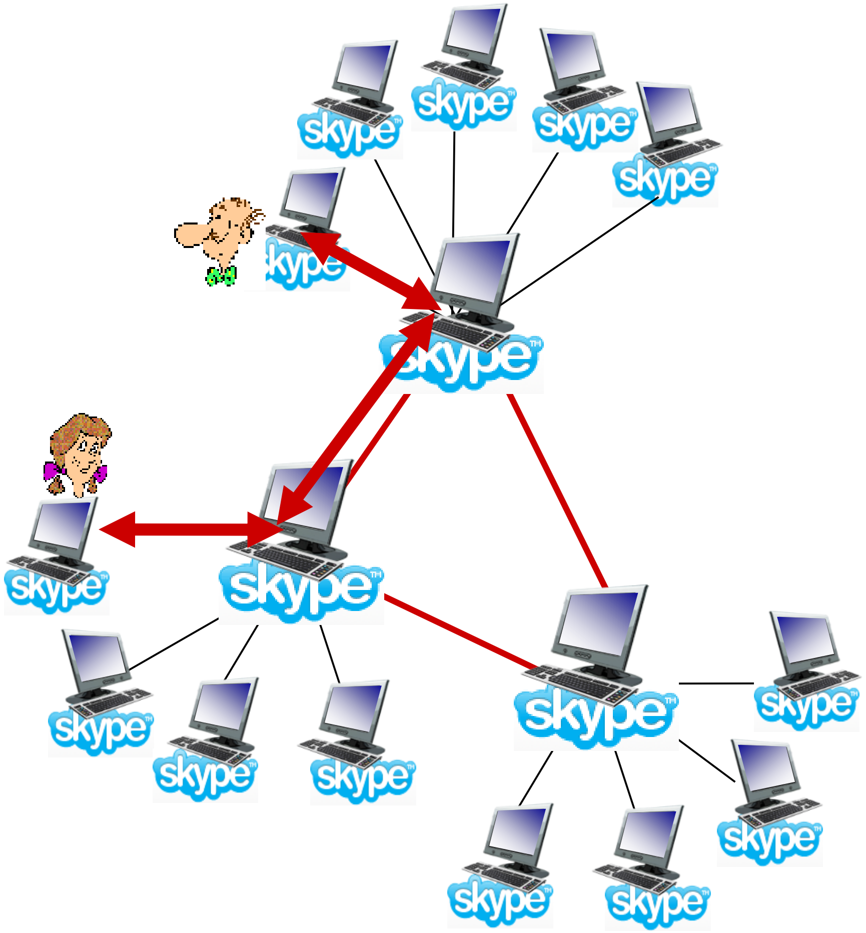
1. joins skype network by contacting SN (IP address cached) using TCP
2. logs-in (usename, password) to centralized skype login server
3. obtains IP address for callee from SN, SN overlay
4. initiate call directly to callee



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



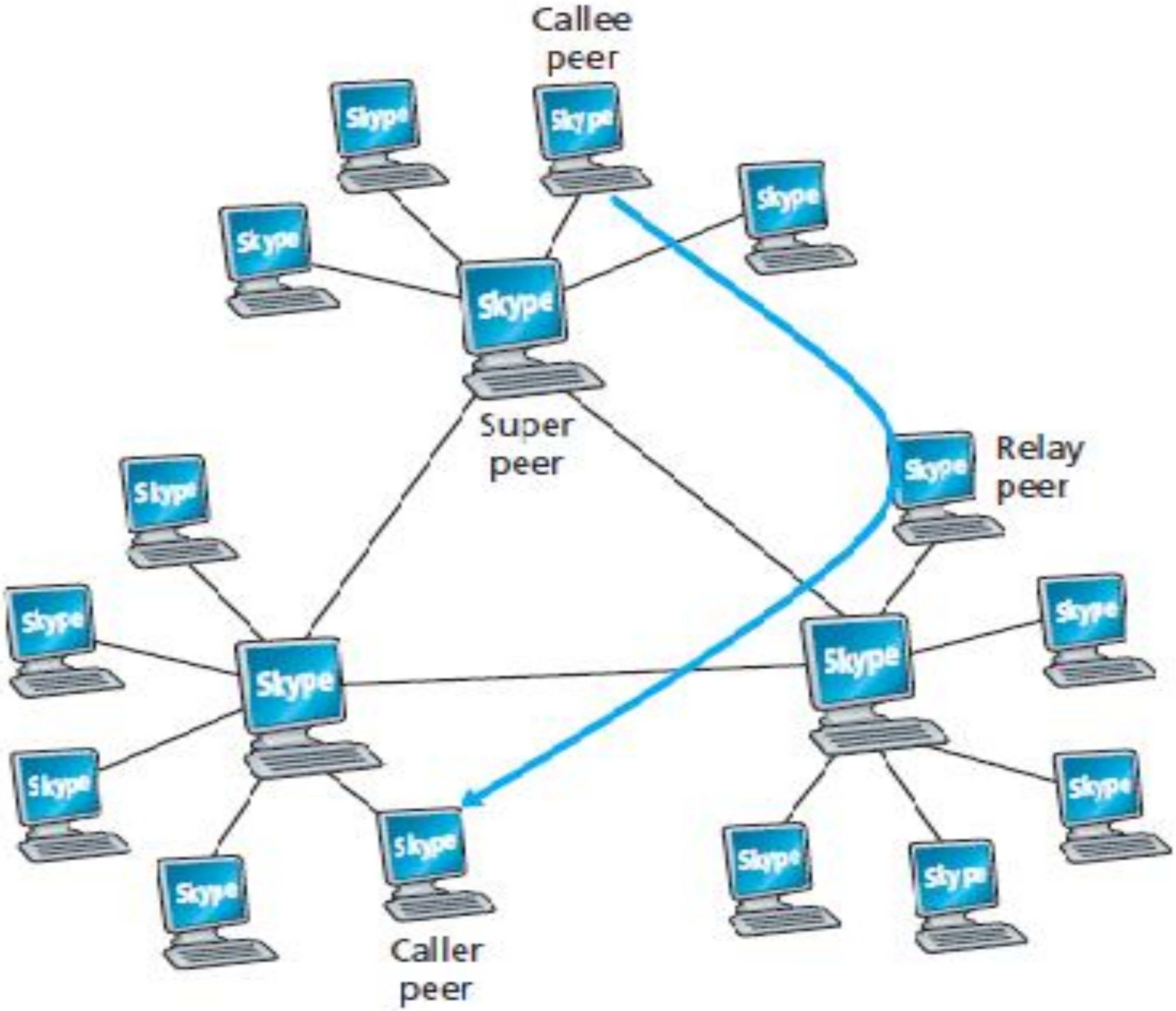


Figure 7.10 ◆ Skype peers

- Multi-party audio conference calls

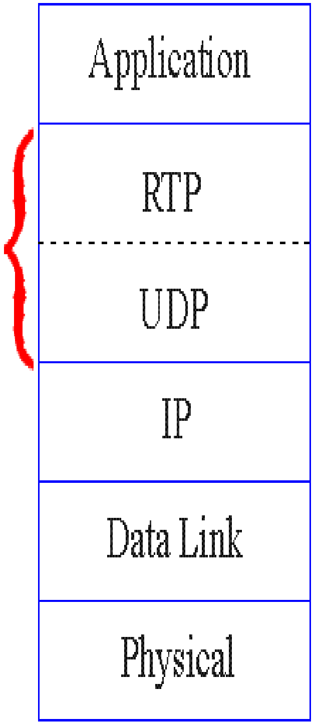
- Two party video conference calls

- Multi-party video conference calls

- VoIP systems such as Skype, QQ, and Google Talk introduce new privacy concerns.

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

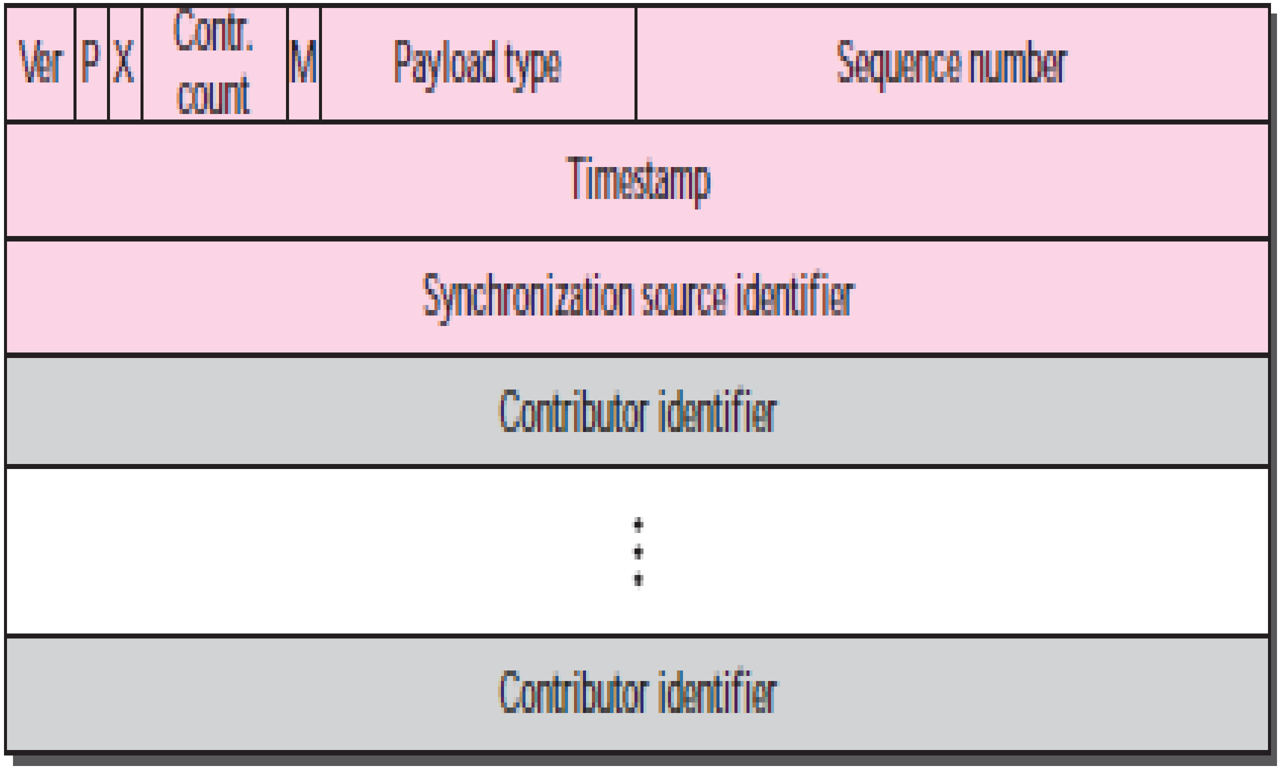


- RTP runs in end systems
- RTP packets encapsulated in UDP segments
- interoperability: if two VoIP applications run RTP, they may be able to work together

RTP 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

Figure 25.19 *RTP packet header format*



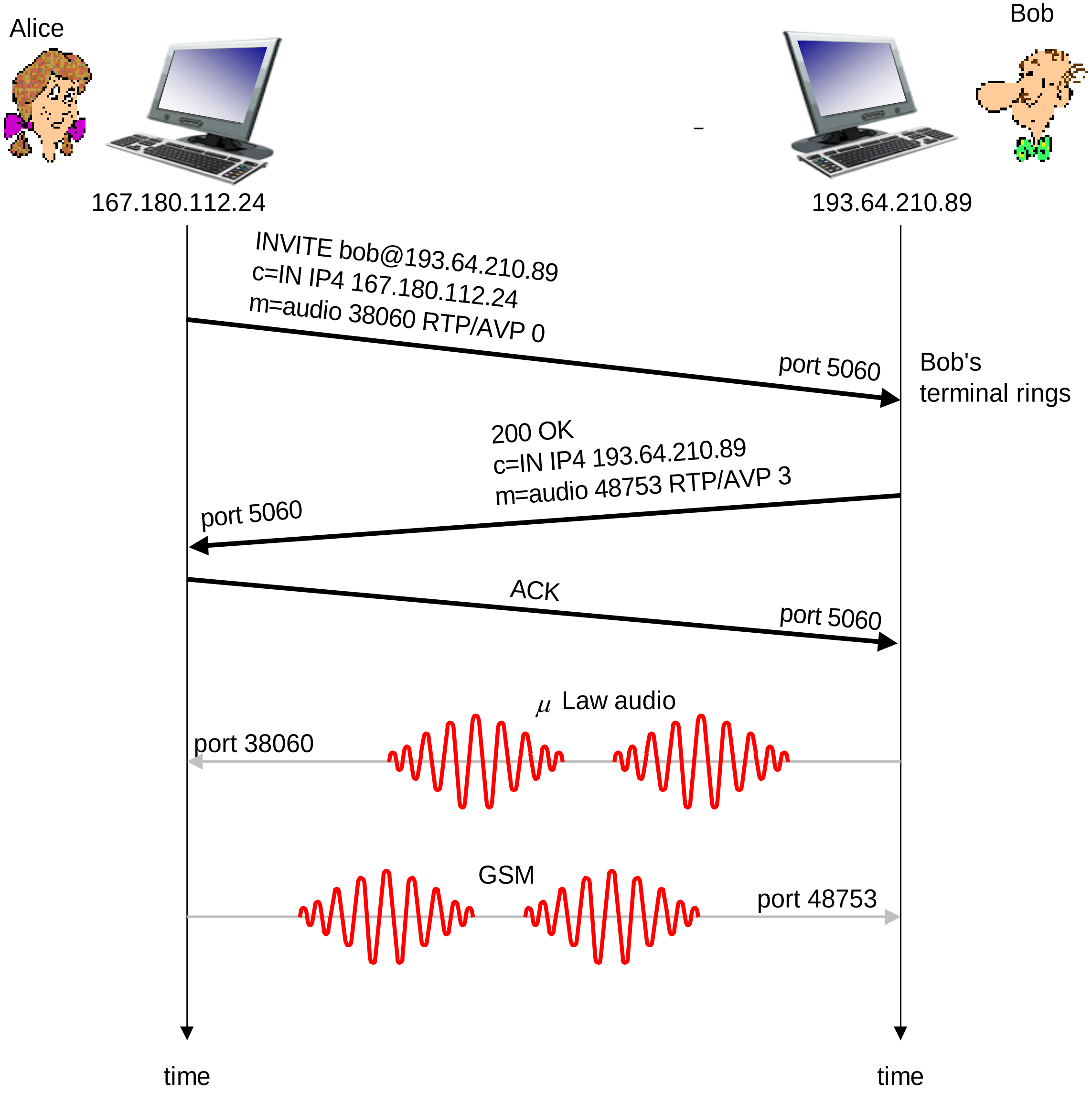
RTP uses a temporary even-numbered UDP port.

SIP: Session Initiation Protocol

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 μlaw)
- 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

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

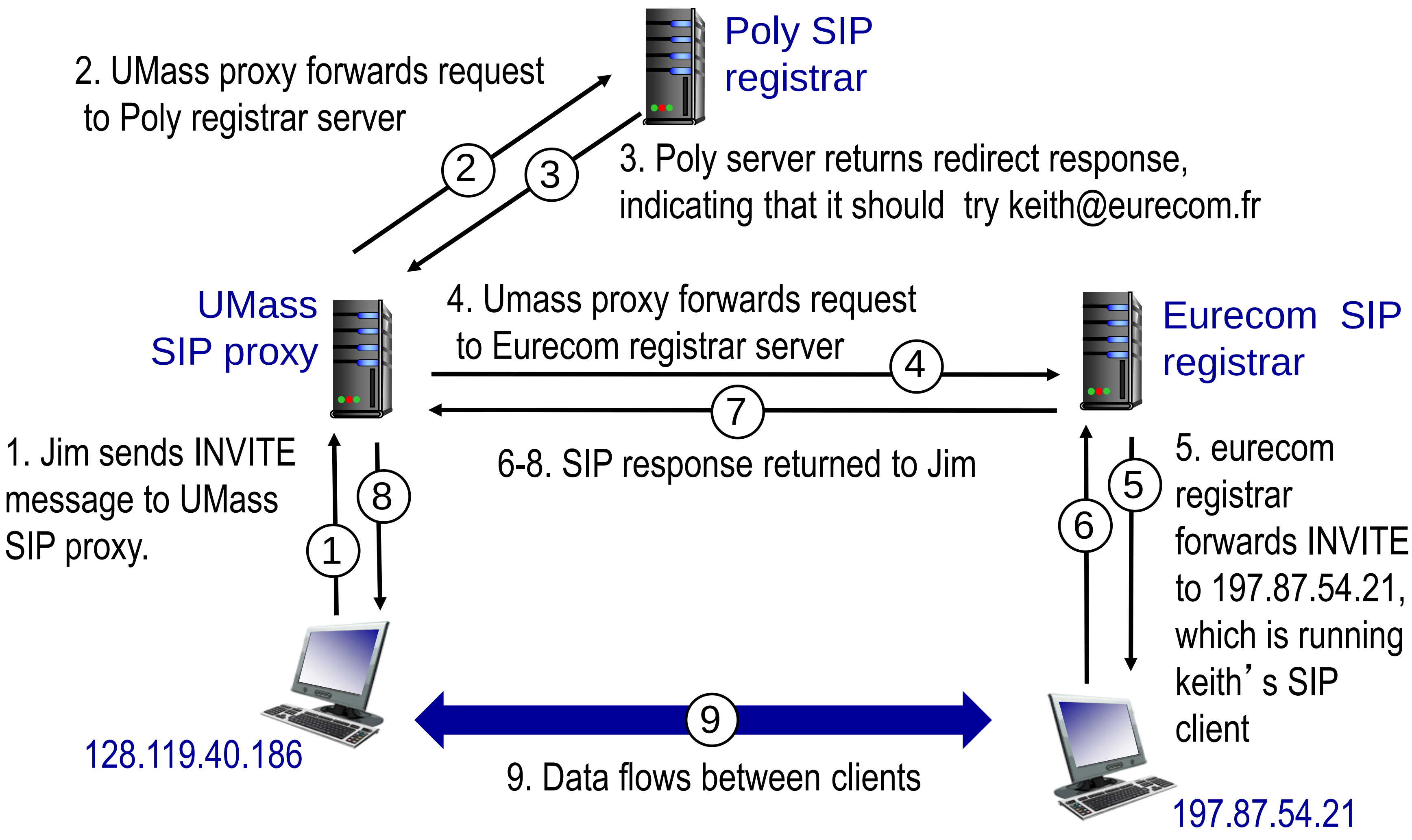
register message:

```
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



1. Jim sends INVITE message to UMass SIP proxy.

2. UMass proxy forwards request to Poly registrar server

3. Poly server returns redirect response, indicating that it should try keith@eurecom.fr

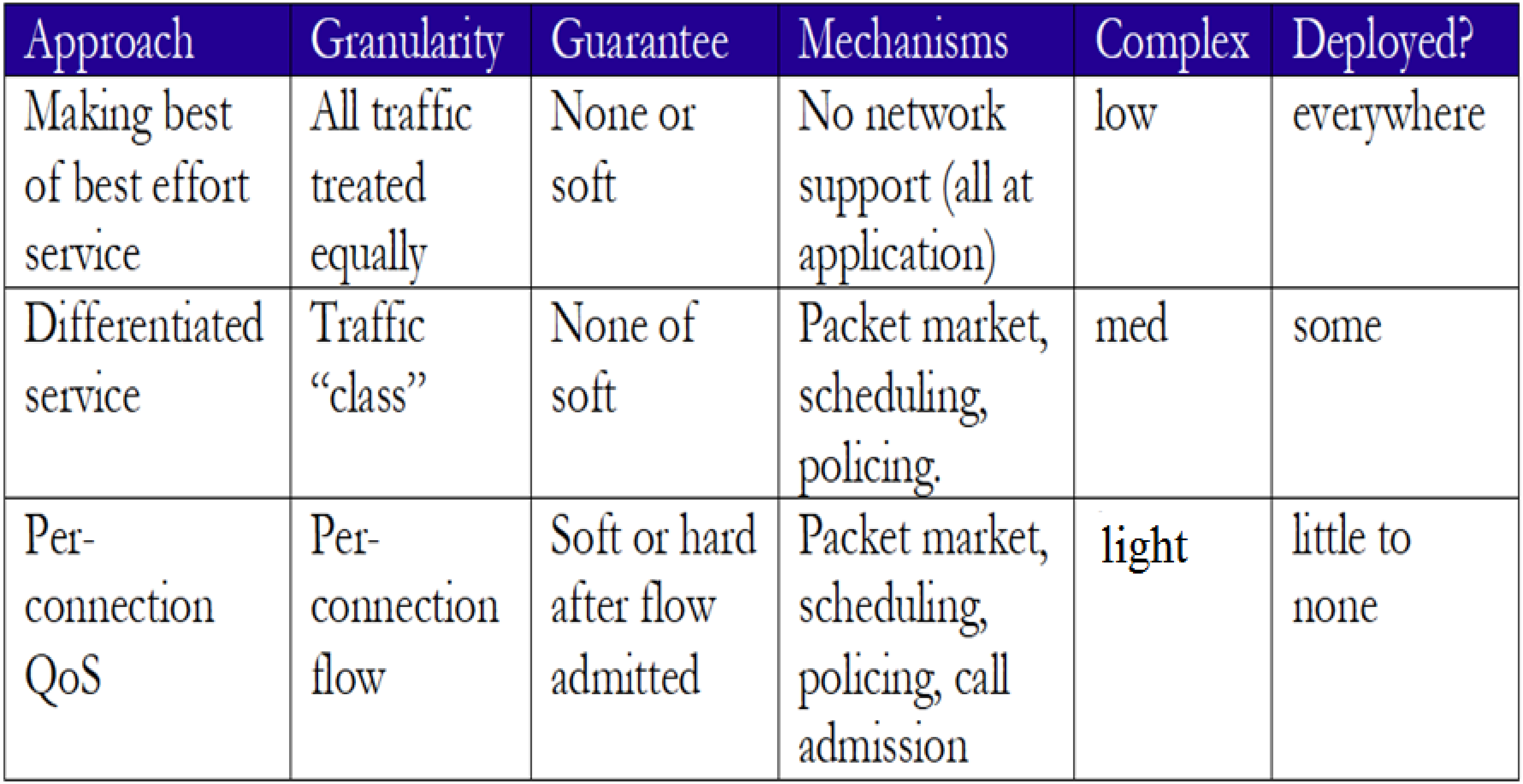
4. Umass proxy forwards request to Eurecom registrar server

5. eurecom registrar forwards INVITE to 197.87.54.21, which is running keith’s SIP client

6-8. SIP response returned to Jim

9. Data flows between clients

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 of 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	light	little to none

Dimensioning Best-Effort Networks

*The question of how much capacity to provide at network links in a given topology to achieve a given level of performance is often known as **bandwidth provisioning**.*

*The even more complicated problem of how to design a network topology (where to place routers, how to interconnect routers with links, and what capacity to assign to links) to achieve a given level of end-to-end performance is a network design problem often referred to as **network dimensioning**.*

The following issues must be addressed in order to predict application-level performance between two network end points, and thus provision enough capacity to meet an application's performance requirements:

- *Models of traffic demand between network end points.*

Models may need to be specified at both the call level and at the packet level. Note that workload may change over time.

- *Well-defined performance requirements.*

- *Models to predict end-to-end performance for a given workload model, and techniques to find a minimal cost bandwidth allocation that will result in all user requirements being met.*

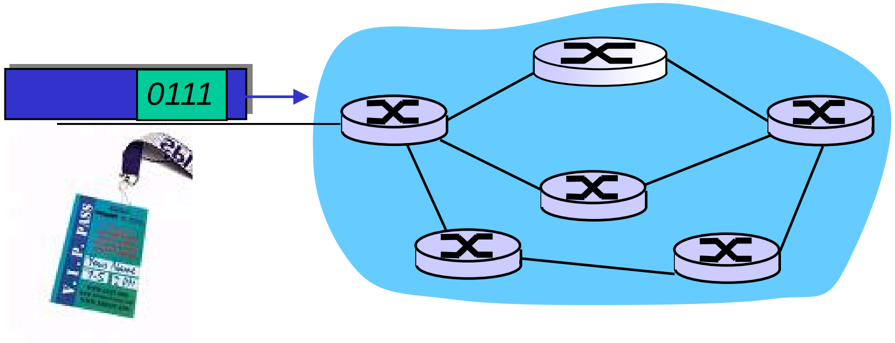
Why today's Internet doesn't provide NETWORK SUPPORT FOR MULTIMEDIA?

The answers are primarily economic and organizational.

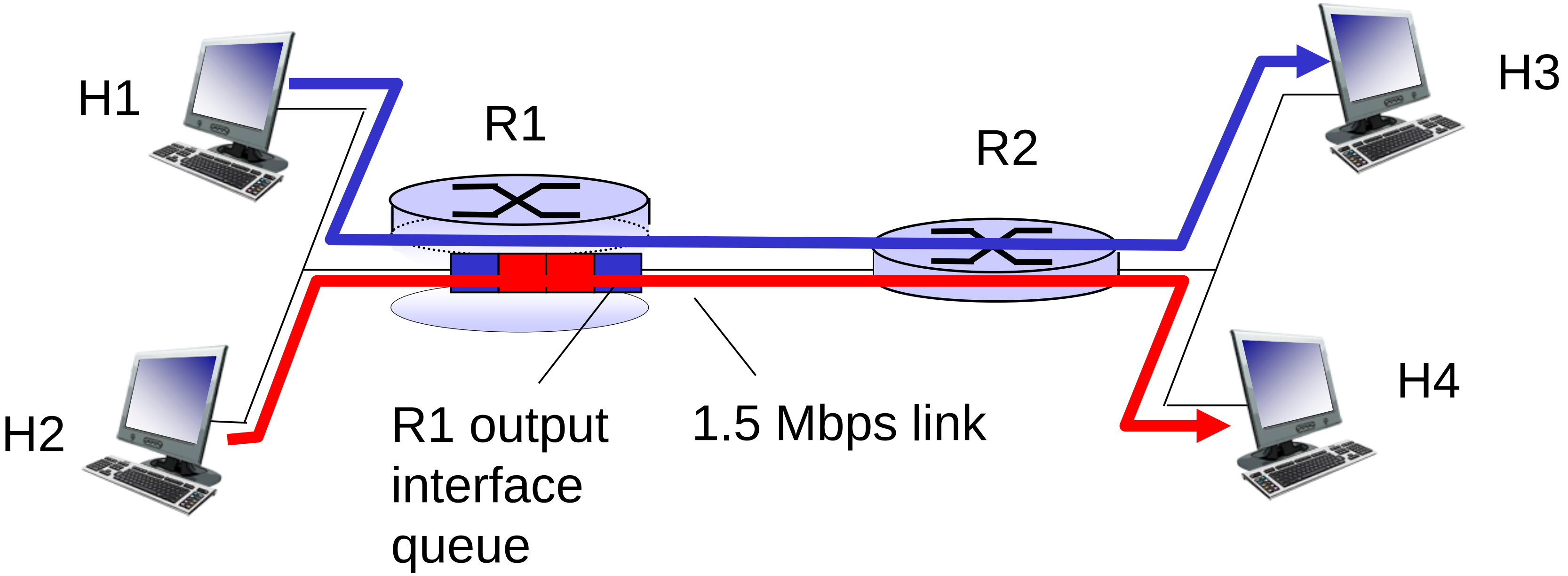
Providing multiple classes of service

- 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

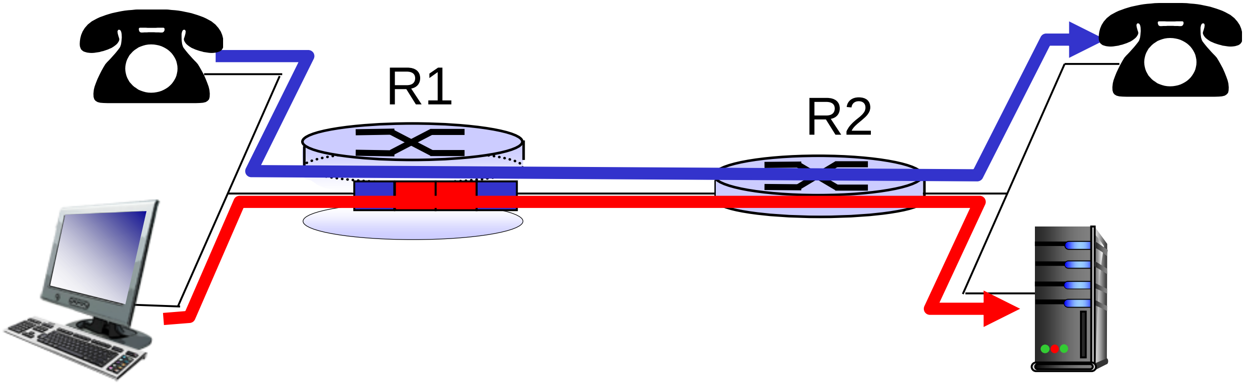


Multiple classes of service: scenario



Scenario 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

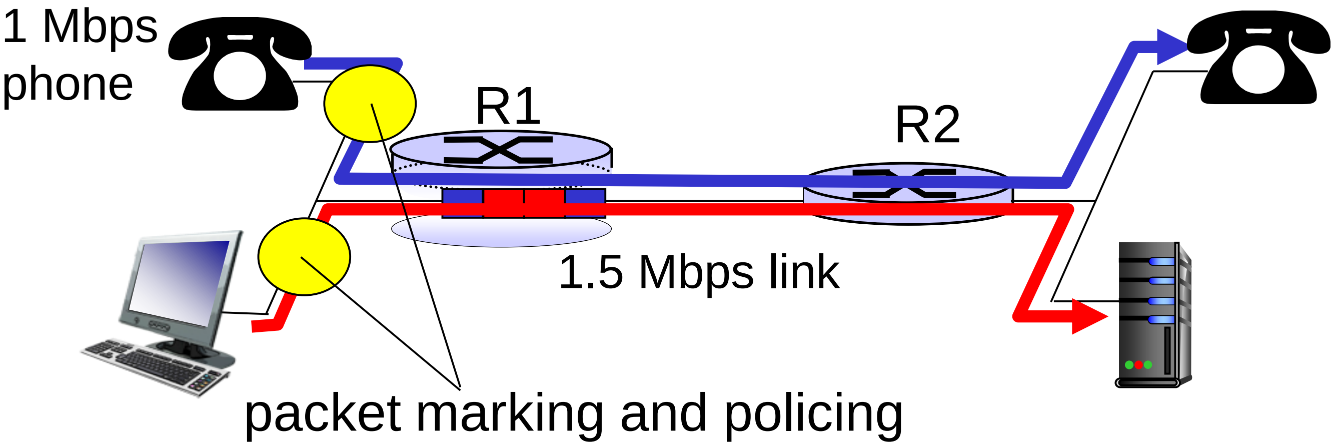


Principle 1

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 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

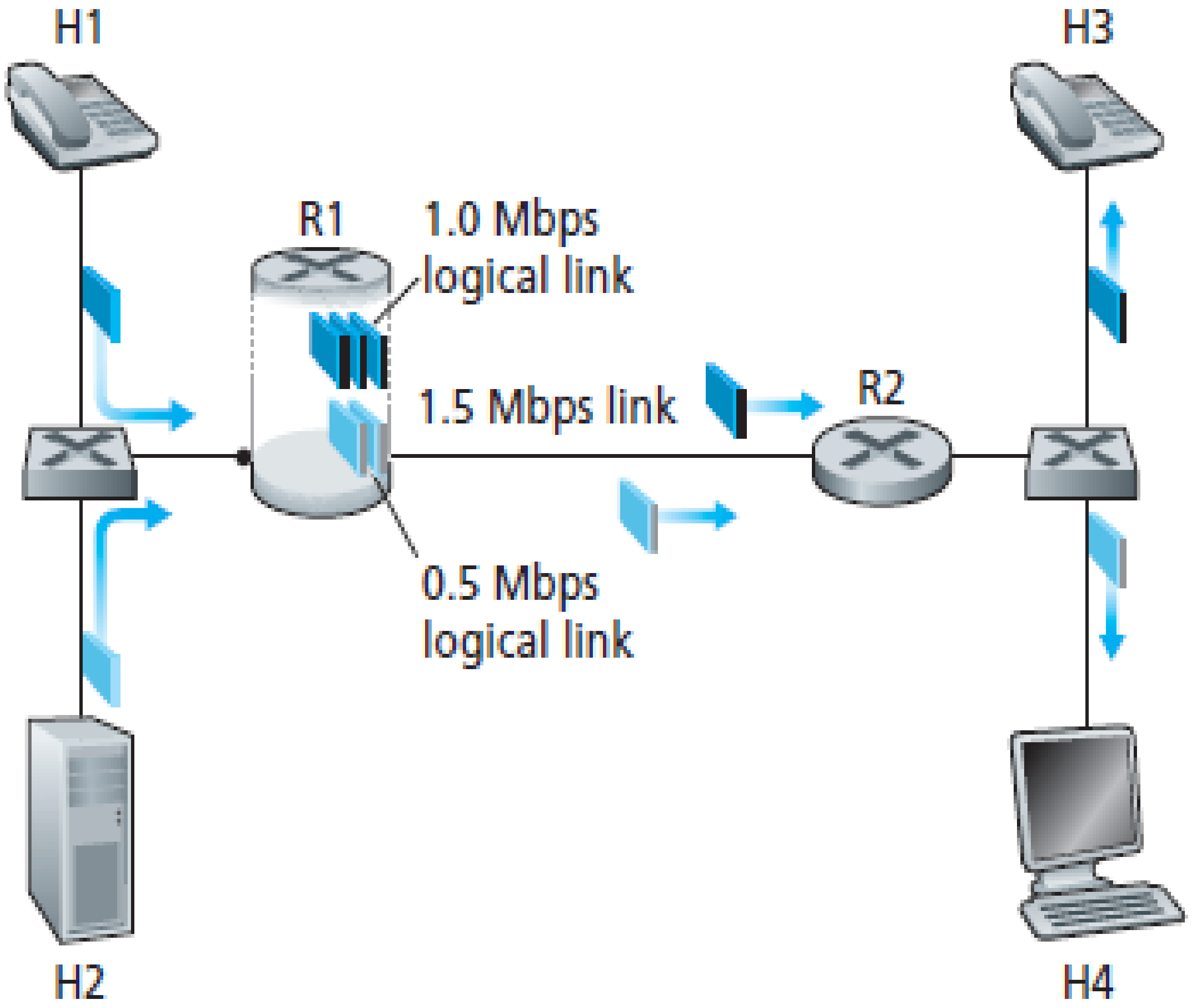
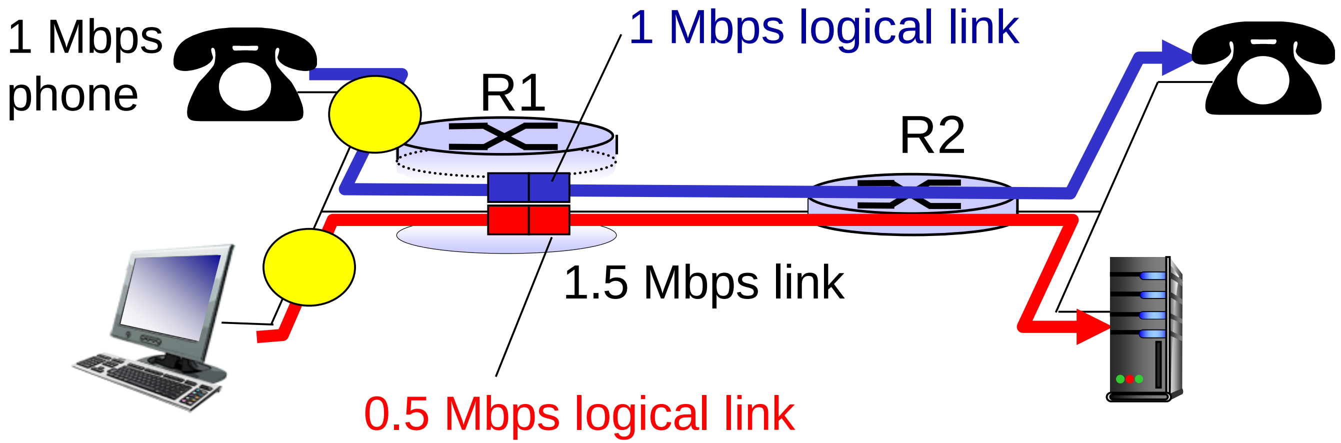


Figure 7.16 ♦ Logical isolation of audio and HTTP traffic classes

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 mechanisms

- *scheduling:* The manner in which queued packets are selected for transmission on the link is known as the link-scheduling discipline.

- *FIFO (first in first out) scheduling:* send in order of arrival to queue

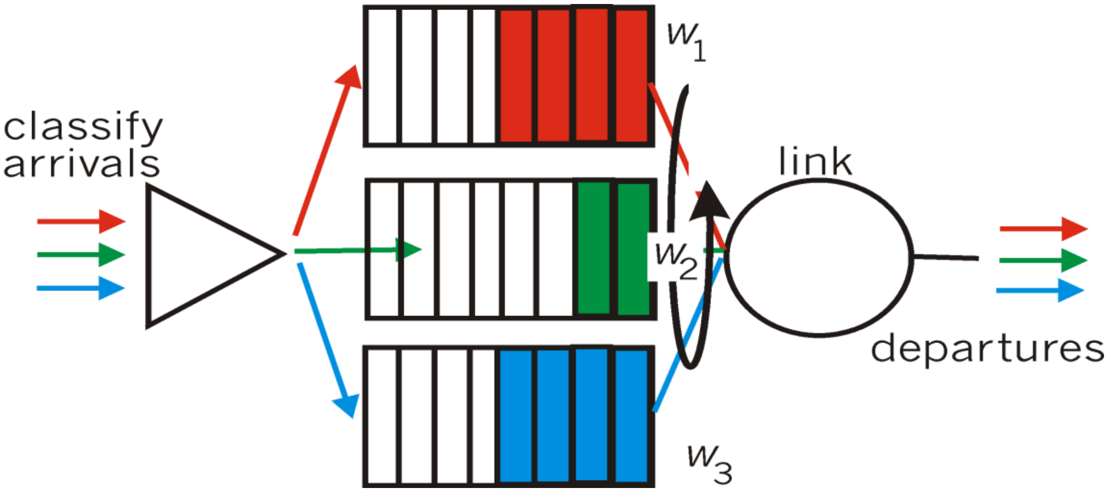
- *priority scheduling:* send highest priority queued packet

- *Round Robin (RR) scheduling:* multiple classes, cyclically scan class queues, sending one complete packet from each class (if available)

Scheduling policies: still more

Weighted Fair Queuing (WFQ):

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



Policing mechanisms

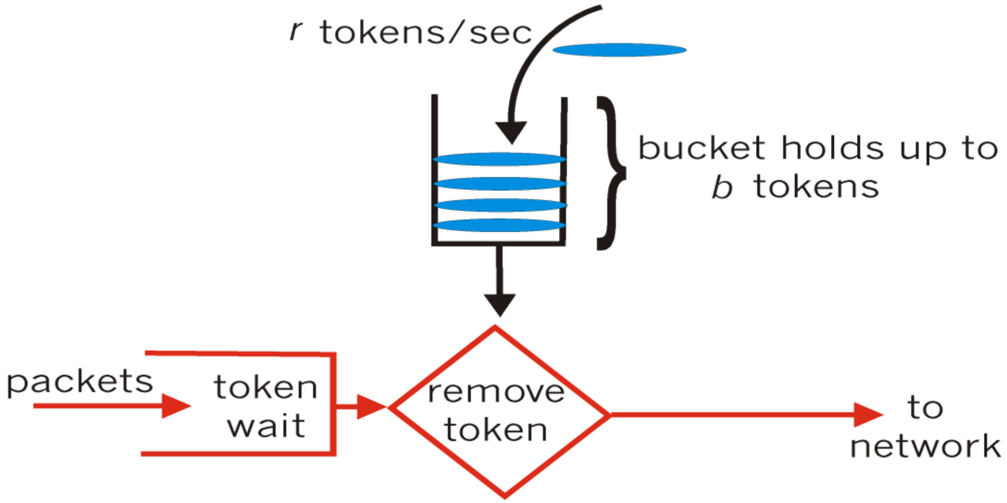
goal: limit traffic to not exceed declared parameters

Three commonly-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 in a very short time(instantaneously)

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 is less than or equal to (r t + b)*

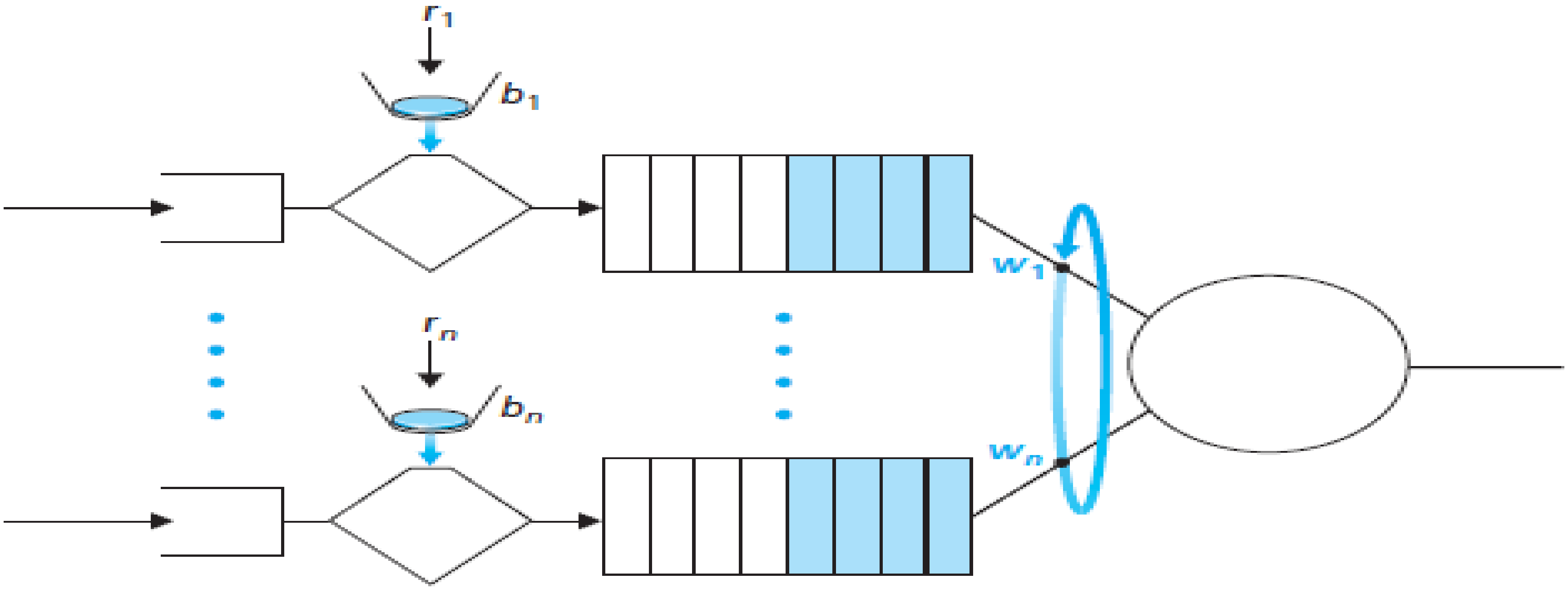


Figure 7.24 ♦ *n* multiplexed leaky bucket flows with WFQ scheduling

Let us focus on flow 1. Suppose that flow 1's token bucket is initially full. A burst of b1 packets then arrives to the leaky bucket policer for flow 1. These packets remove all of the tokens (without wait) from the leaky bucket and then join the WFQ waiting area for flow 1. Since these b1 packets are served at a rate of at least R · wi /(Σwj) packet/sec, the last of these packets will then have a maximum delay, dmax, until its transmission is completed, where

$$d_{\max} = \frac{b_1}{R \cdot w_1 / \sum w_j}$$

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

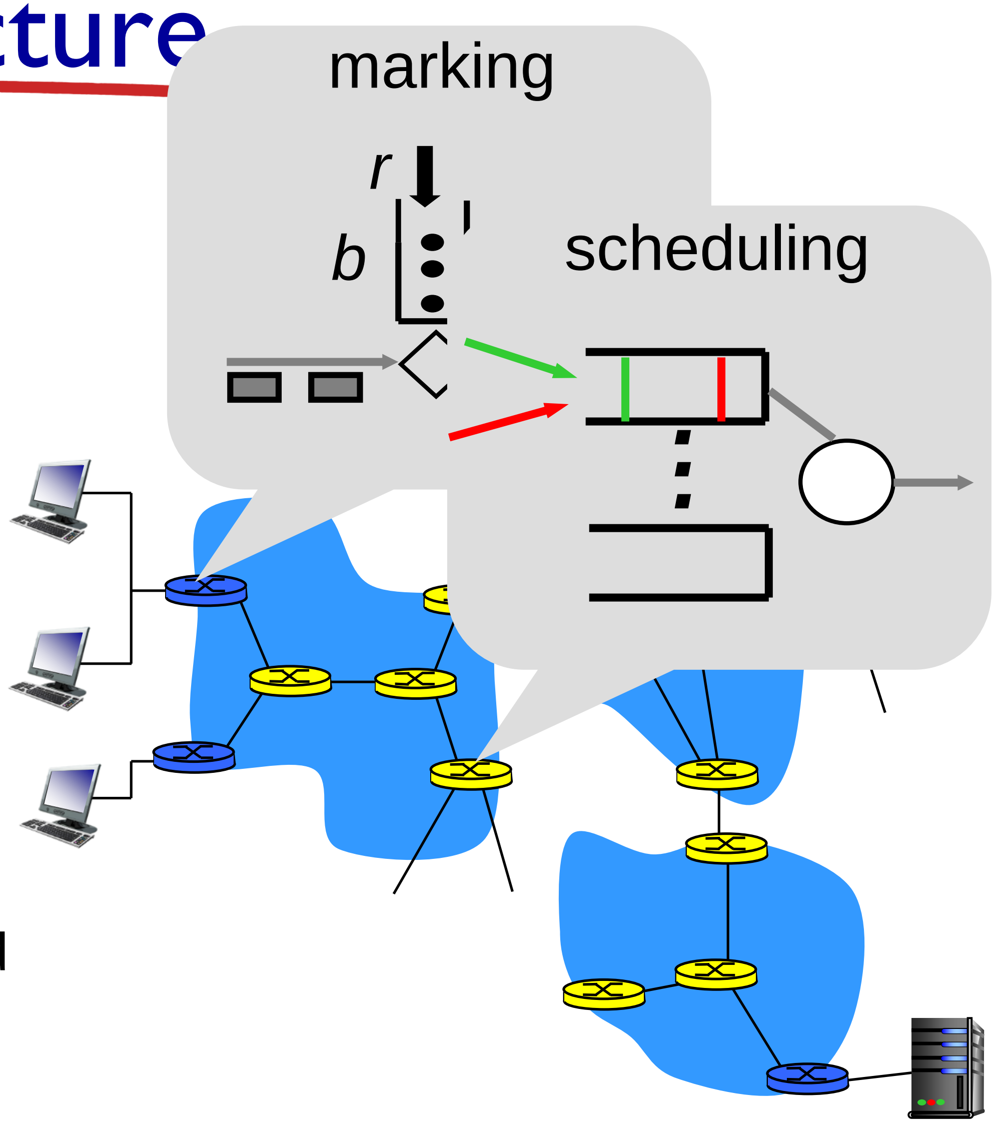
Diffserv architecture

edge router:

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

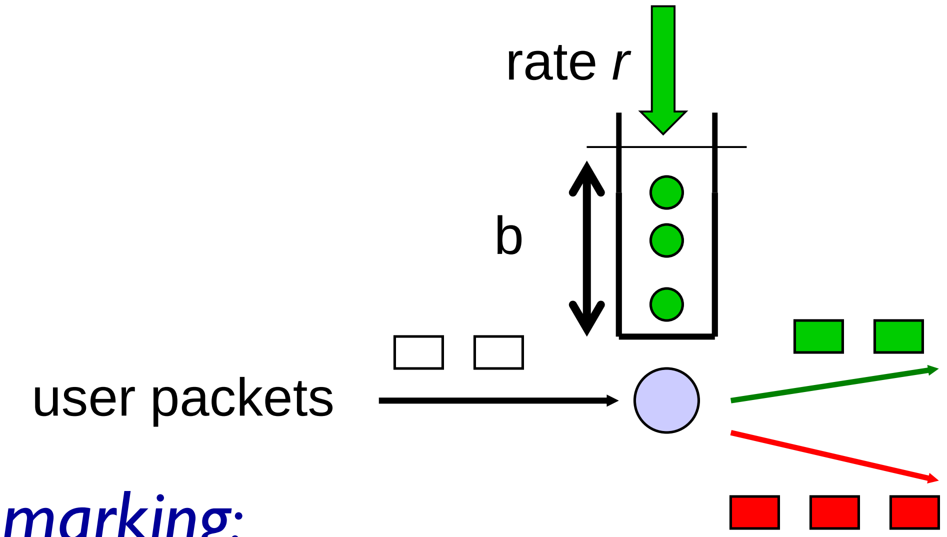
core router:

- *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

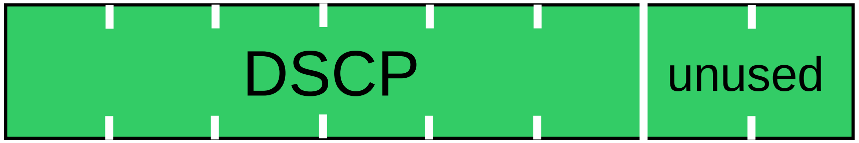


possible use of marking:

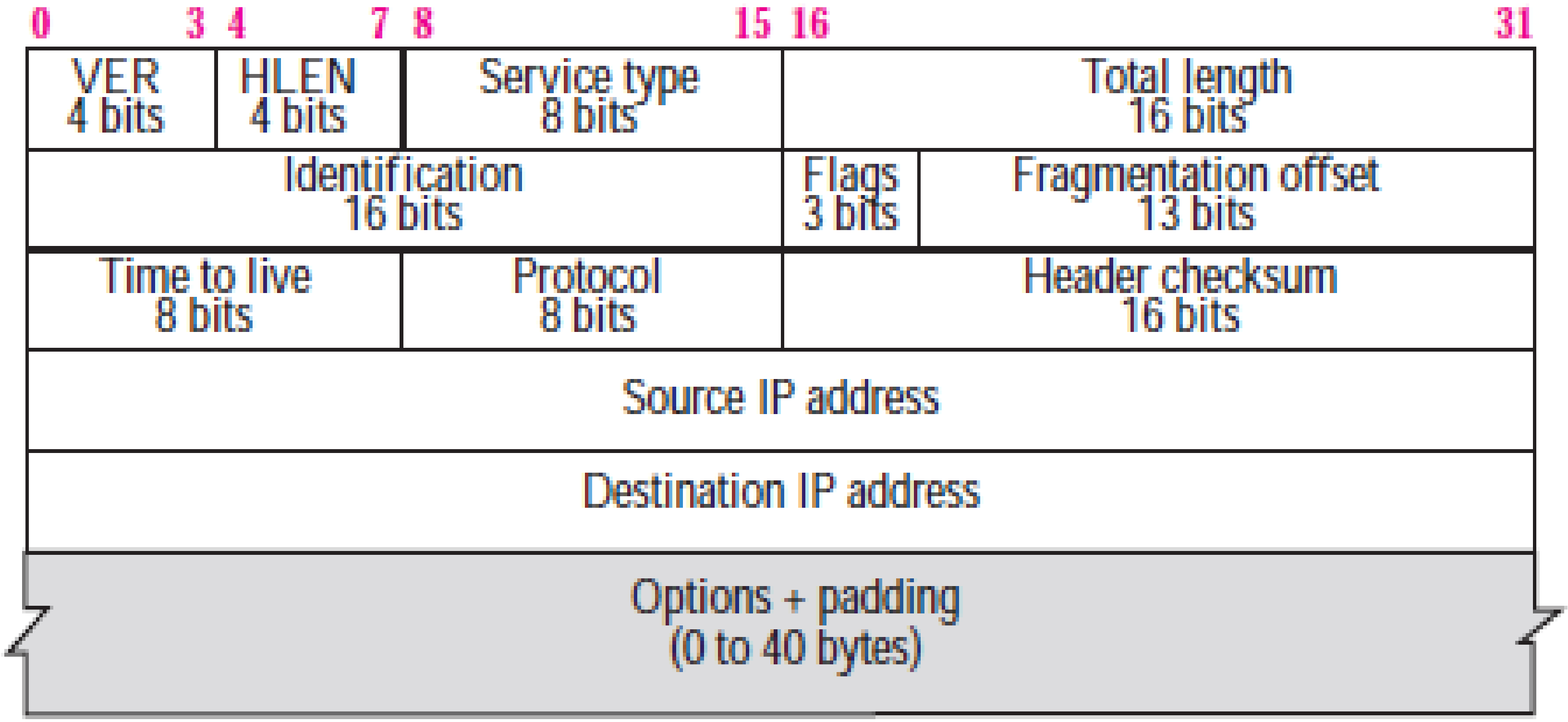
- class-based marking: packets of different classes marked differently
- intra-class marking: conforming portion of flow marked differently than non-conforming one

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



IP Datagram



0 3 4 7 8 15 16 31

VER 4 bits

HLEN 4 bits

Service type 8 bits

Total length 16 bits

Identification 16 bits

Flags 3 bits

Fragmentation offset 13 bits

Time to live 8 bits

Protocol 8 bits

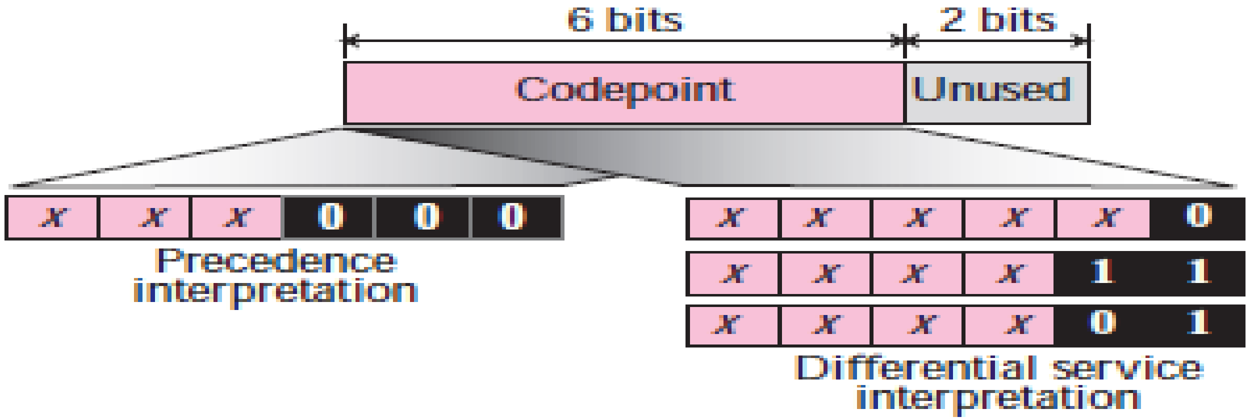
Header checksum 16 bits

Source IP address

Destination IP address

Options + padding (0 to 40 bytes)

Service type



6 bits

2 bits

Codepoint

Unused

x x x 0 0 0

Precedence interpretation

x x x x x 0

x x x x 1 1

x x x x 0 1

Differential service interpretation

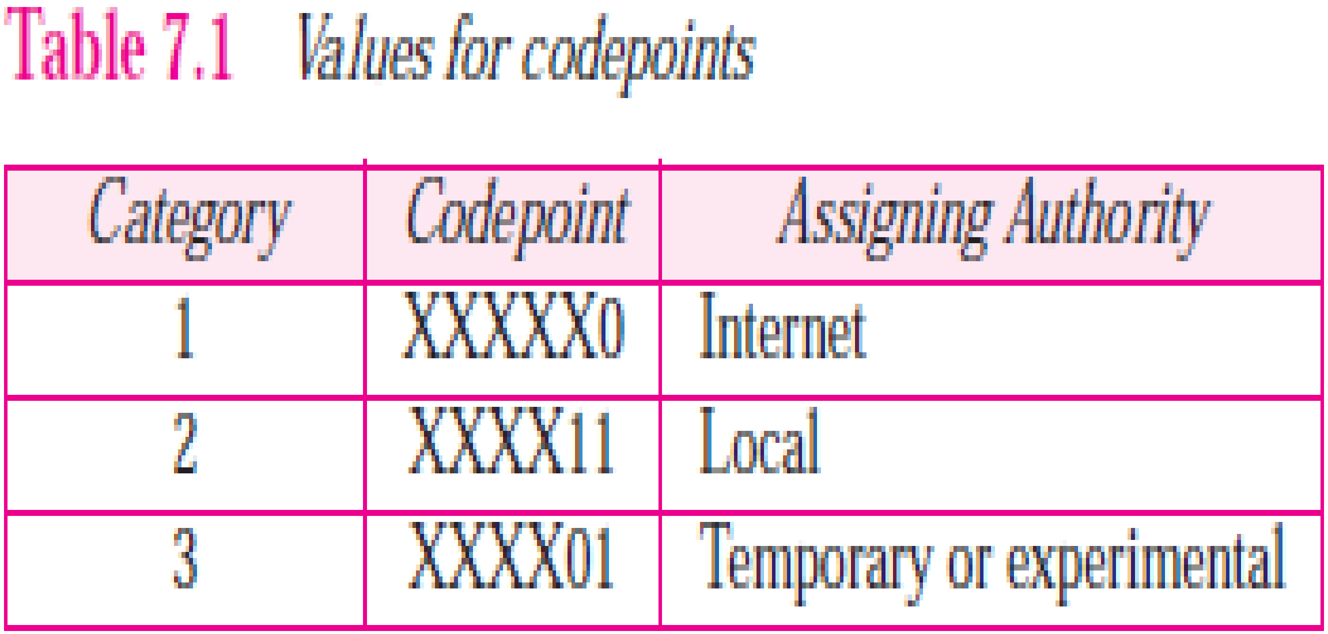


Table 7.1 Values for codepoints

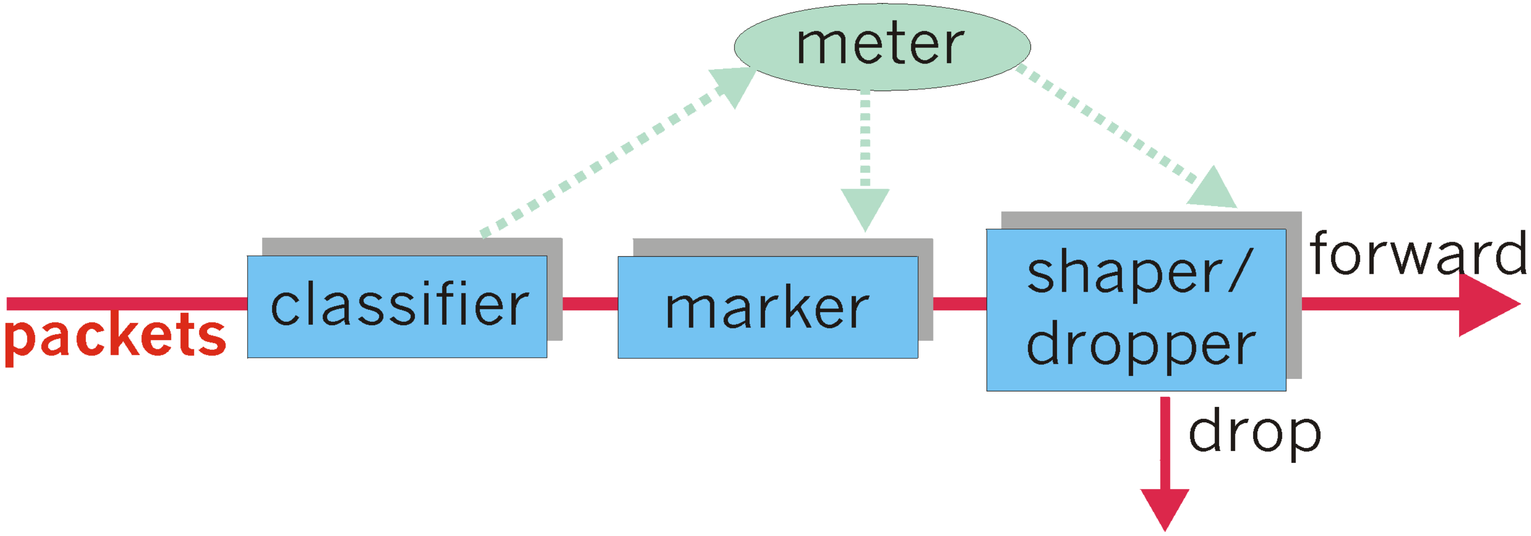
Category	Codepoint	Assigning Authority
1	XXXXX0	Internet
2	XXXX11	Local
3	XXXX01	Temporary or experimental

In IPV6 traffic class and flow label fields together take over the function of the service type field.

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 results 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

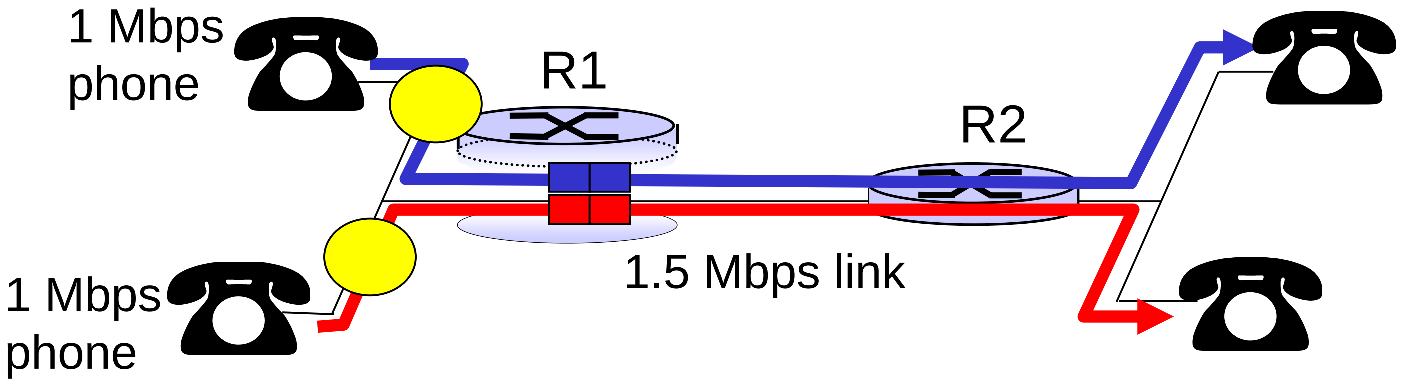
Two PHBs have been defined: an expedited forwarding and an assured forwarding.

The **expedited forwarding** PHB specifies that the departure rate of a class of traffic from a router must equal or exceed a configured rate.

The **assured forwarding** PHB divides traffic into four classes, where each AF class is guaranteed to be provided with some minimum amount of bandwidth and buffering.

Per-connection QOS guarantees

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

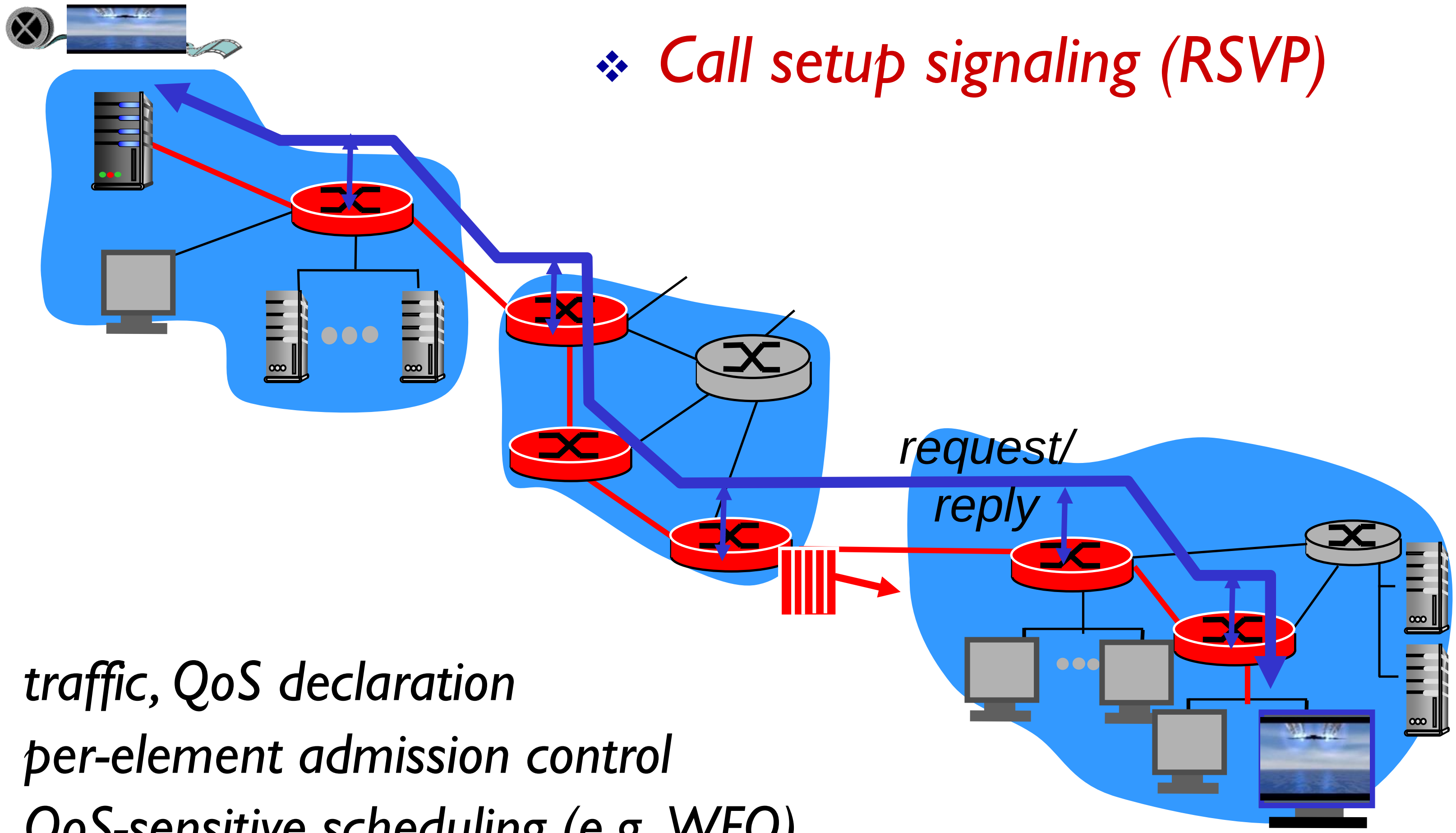


Principle 4

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 Admission*
- *Call setup signaling (RSVP)*



- *traffic, QoS declaration*
- *per-element admission control*
- *QoS-sensitive scheduling (e.g., WFQ)*

Thank You