Lecture 10: Multimedia Networks

Acknowledgement: Materials presented in this lecture are predominantly based on slides from:

- *Computer Networking: A Top Down Approach,* J. Kurose, K. Ross, 7th ed., 2017, Addison-Wesley, Chapter 2, Chapter 4 and **Chapter 9**

Lecture 10: Multimedia Networks
Outline

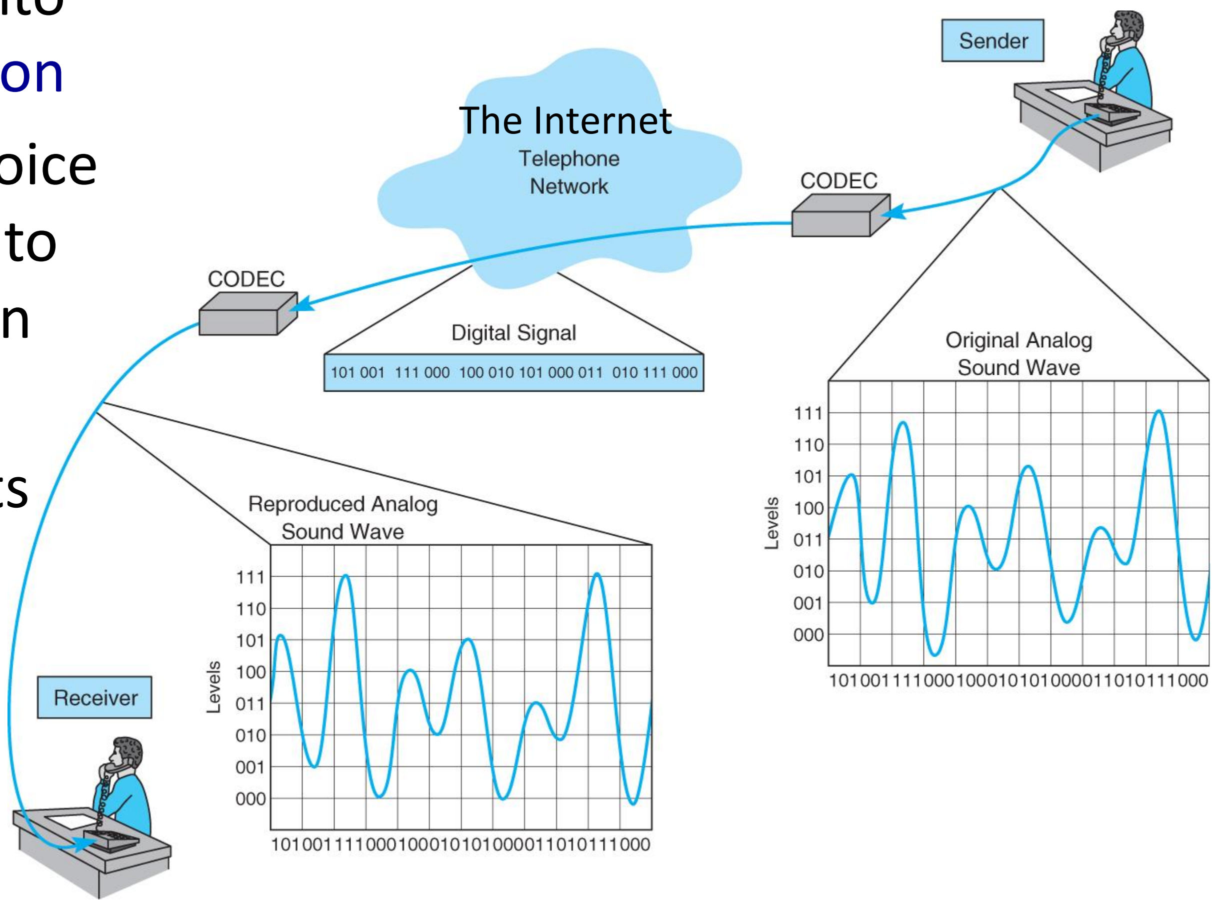
- Digital representation of audio and video data.
- Streaming stored video
- Dynamic Adaptive Streaming over HTTP (DASH)
- Content distribution networks
  - Netflix
- Voice-over-IP (VoIP)
- Voice-over-IP: Skype
- RTP: Real-Time Protocol (optional ?)

Multimedia

- **Multimedia** refers to representation of the information contents in a variety of forms including a combination of:
  - text,
  - audio (speech and music)
  - still images,
  - animation,
  - video,
  - interactivity (games)
- We will concentrate on audio and video forms
- A related term: "triple play" refers to making: the Internet, phone and TV available over a single broadband connection

Digital audio

- All forms of the multimedia information must be converted into a digital representation
- In the example the voice signal is converted into a string of bits used in computer networks
- Receiver converts bits back to analog signal with some quality reduction



Digitizing audio/voice signal

- Analog voice signal is sampled at constant rate of 8,000 samples/sec
- The sampling frequency is:

$$f_s = 8 \text{ kHz}$$

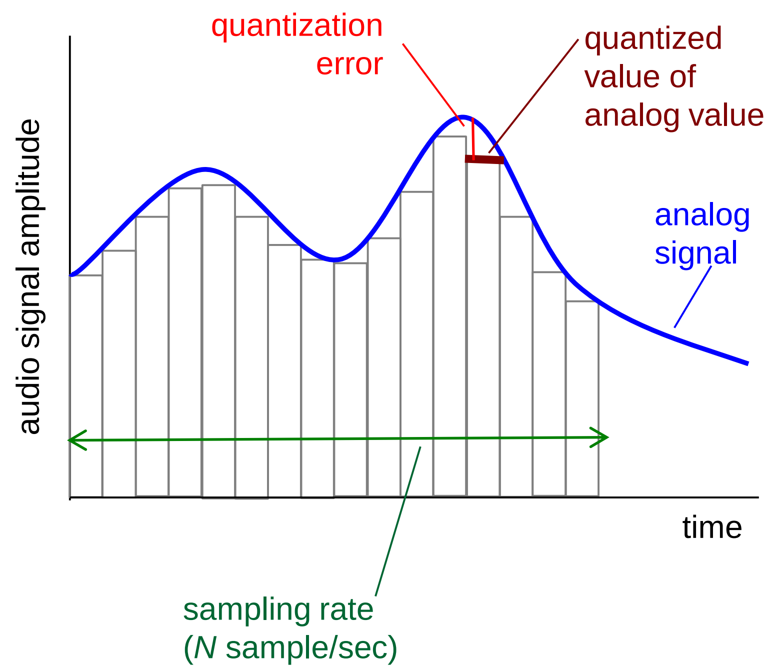
selected to be more than twice the maximum frequency component of the speech (3.4 kHz)

- The sampling time, that is the distance between samples is

$$t_s = {}^{1}/_{f_s} = 125 \ \mu\text{sec}$$

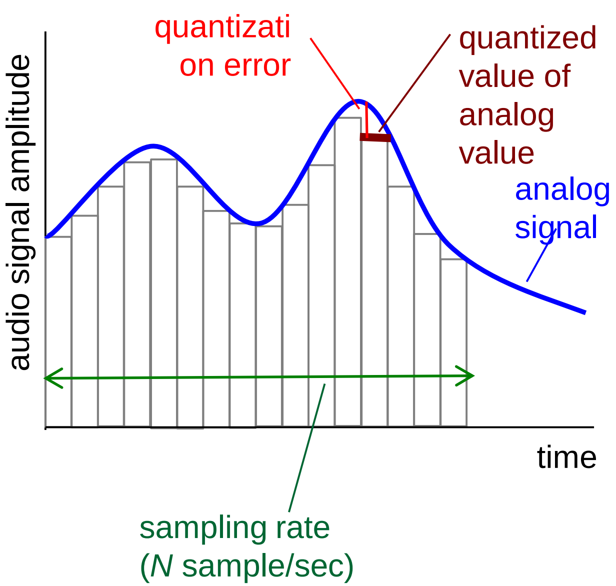
- Each sample is quantized into 2^8=256 possible quantized values
- Each quantized sample is represented by 8 bits.
- The resulted bit stream is

$$8 \times 8 = 64 \text{ kb/sec}$$



- The quantization error is the difference between the analog and the quantized value of the signal

Digitizing music



- Music frequency contents perceived by human ears is up to 20kHz
- Therefore the (CD) music is sampled at the sampling frequency:

$$f_s = 44.1 \text{ kHz}$$

- Each music sample is typically quantized into 2^{16}=65,536 possible quantized values – represented by 16-bit samples
- The resulted music (raw) bit stream is

$$16 \times 44.1 = 705.6 \text{ kb/sec}$$

- Using two (stereo) channels gives **1.411Mb/s**

- If your internet connection is not fast enough you would not be able to listen to high quality (uncompressed) music.

Audio compression

- Audio compression is based on the fact that it is possible to:
  - **predict** the value of the next sample from the previous samples
  - **calculate** the difference between the predicted and the real value of the signal,
  - transfer only this small residual value/difference.
- The receiver can recreate the original value typically with some (small) error
- The number of compression/coding methods is huge.
- The most popular **coding** (compression) methods include: MP3 standard and AAC (Advanced Audio Coding), … ?
- After compression the voice bit rate can be reduced from 64kbps to 5.3kbps
- The mp3 music bit rates can be: 96, 128, 160 kbps (from 1.411 Mbps)

Adaptive Multi Rate speech codec

- AMR and AAC compression methods are used also in our mobile phones
- The list of related 3GPP specifications can be found here
- In particular, the ANSI-C code for the AMR speech codec can be found in TS 26.073 (local copy, 2017)

Uncompressed digital video

- Consider the **High Definition Video**
- Each frame consists 1900 by 1080 pixels.
- Total of 2.052 Megapixels
- Each pixel is represented by 3×8 = 24 bits
- Total of 49.248 Megabits per frame (approx. 50Mb/fr)
- Typically the frames are being sent at the rate of 24 frames per second (or 30fps)
- The resulting **bit rate** to send uncompressed video at 24fps is:

  1,181.952Mps $\approx$ 1.2Gbps
- Typical broadband connection offers transmission rates around

  20Mbps
- 600 times less! (add audio to that number)

Compressing video

In order to compress video we take advantage of:

- Spatial redundancy within a frame: e.g. think about the blue sky spanning the top of an image
- Temporal redundancy, from one frame to the next e.g. if there are no fast moving cars, two adjacent frames are rather similar

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

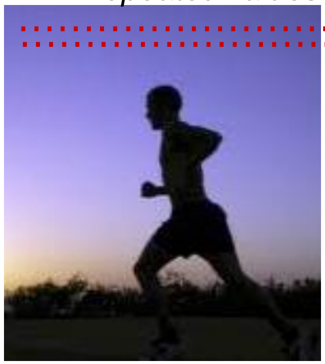
frame *i+1*

Multimedia: video

Two groups of methods of encoding 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 of coding methods:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in the 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

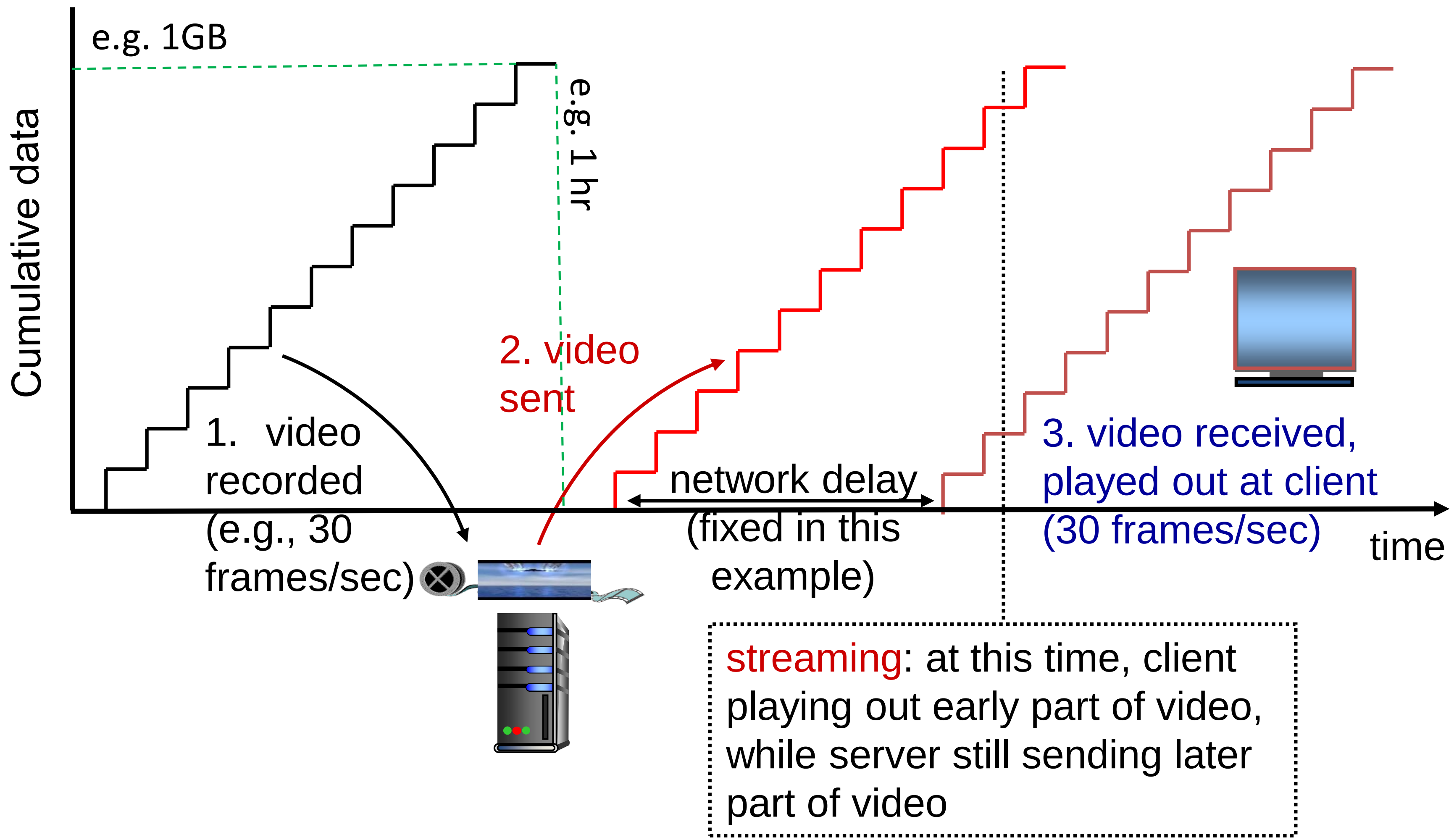
frame *i+1*

Video Formats

Multimedia networking: 3 application types

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

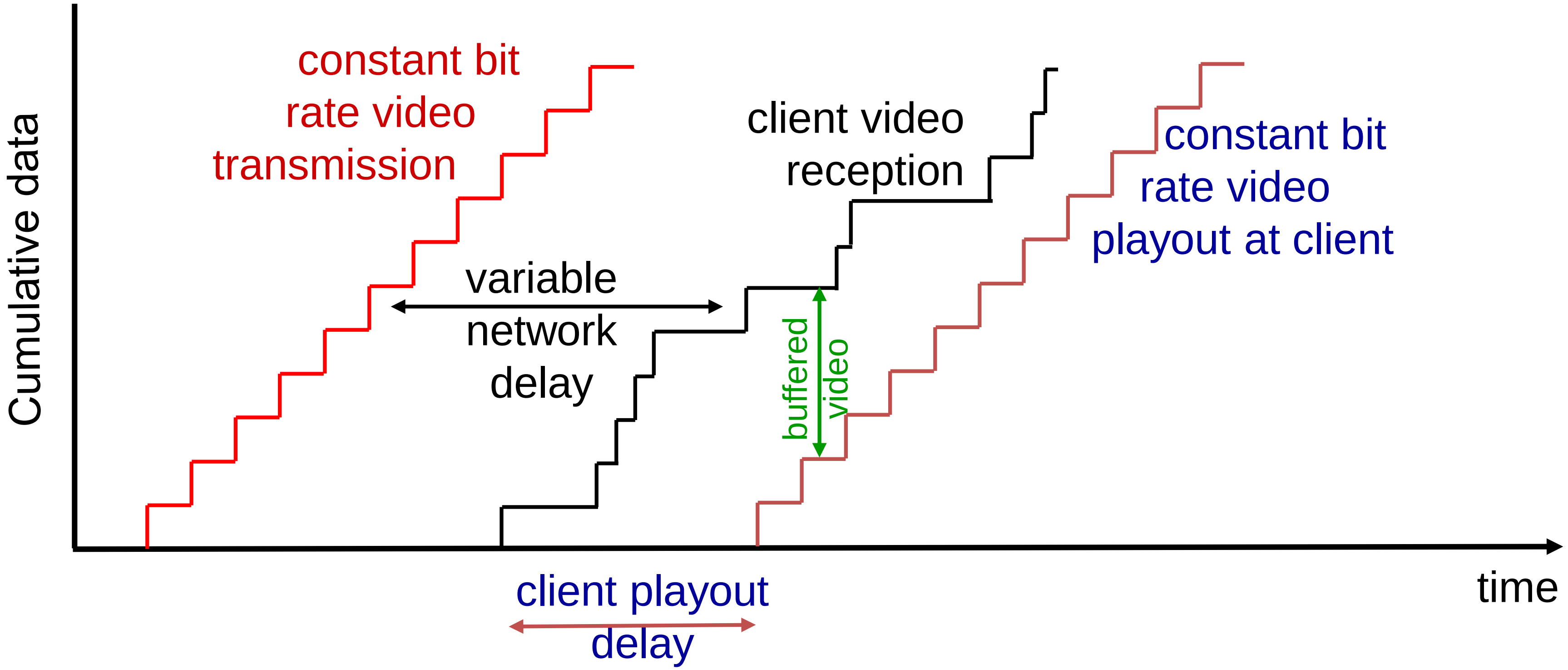
Streaming stored video:



Streaming stored video: challenges

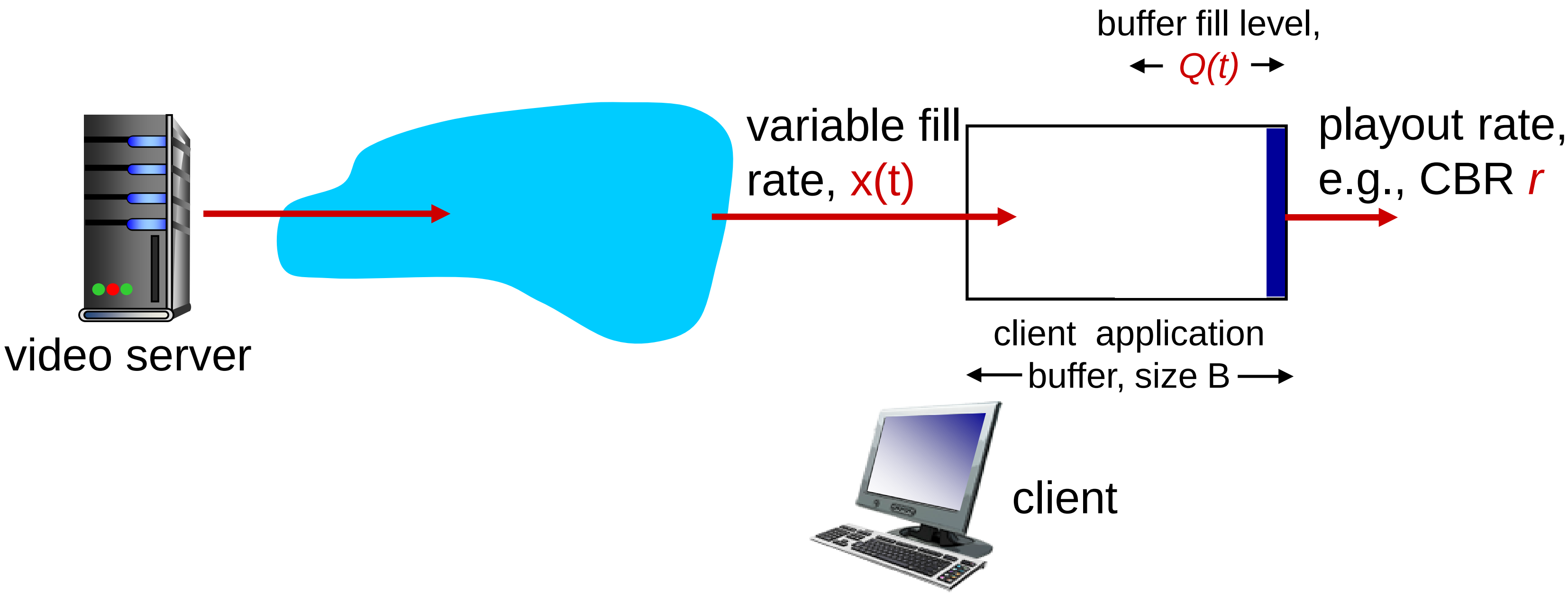
- *continuous playout constraint*: once client playout begins, playback must match original timing
  - … but *network delays are variable* (jitter), so will need *client-side buffer* to match playout requirements
- other challenges:
  - client interactivity: pause, fast-forward, rewind, jump through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

Streaming stored video: revisited



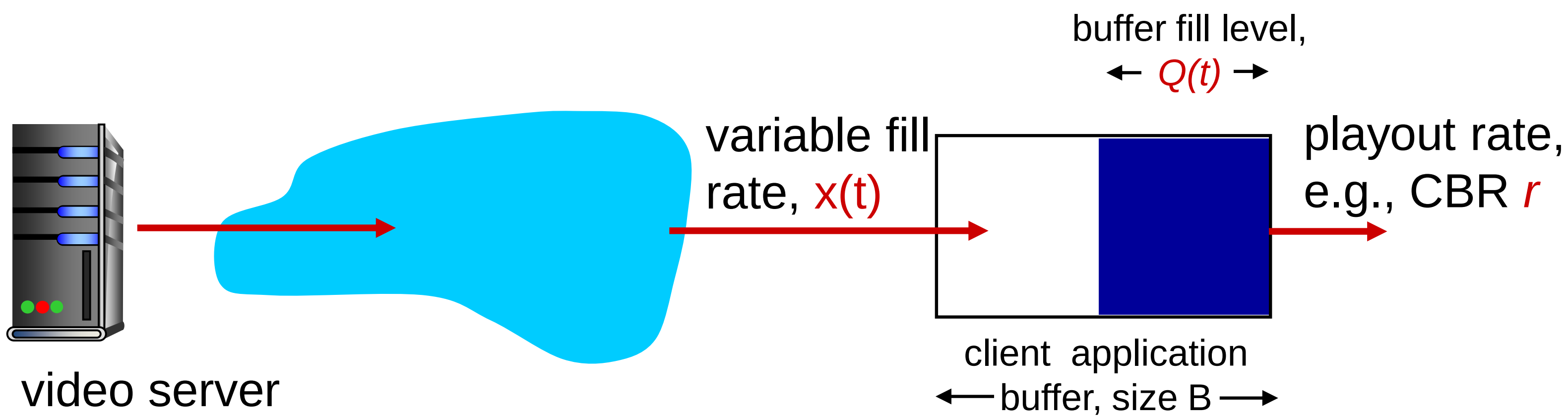
- *client-side buffering and playout delay:* compensate for network-added delay, delay jitter (what is this?)

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 Video Systems

- Streaming video systems allow a user to watch videos stored on the server with features as: pause, stop, fast forward, etc.
- Three basic categories:
  - UDP streaming (less popular)
  - HTTP streaming
  - Adaptive HTTP streaming
- All methods require the client-side buffering as explained before.

Streaming multimedia: UDP

- With UDP streaming, 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 into its socket every

$$(8000 \text{ bits})/(2 \text{ Mbps}) = 4 \text{ msec}$$

- UDP does not employ a congestion-control mechanism, hence 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 typically using the Real-Time Transport Protocol (RTP – RFC3550) designed for transporting audio and video

UDP Streaming Drawbacks 1

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

- e.g. the video consumption rate is 1 Mbps and the server-to-client available bandwidth is usually more than 1 Mbps, but every few minutes the available bandwidth drops below 1 Mbps for several seconds.
- In such a scenario, a UDP streaming system that transmits video at a constant rate of 1 Mbps over RTP/UDP would likely provide a poor user experience, with freezing or skipped frames soon after the available bandwidth falls below 1 Mbps.

UDP Streaming Drawbacks 2

2. The second drawback of UDP streaming is that it requires a media control server, such as an RTSP (Real-Time Streaming Protocol) server

- to process client-to-server interactivity requests and
- to track the client state
  - e.g., the client's playout point in the video,
  - whether the video is being paused or played, …

  for each ongoing client session.
- This increases the overall cost and complexity of deploying a large-scale video-on-demand system.

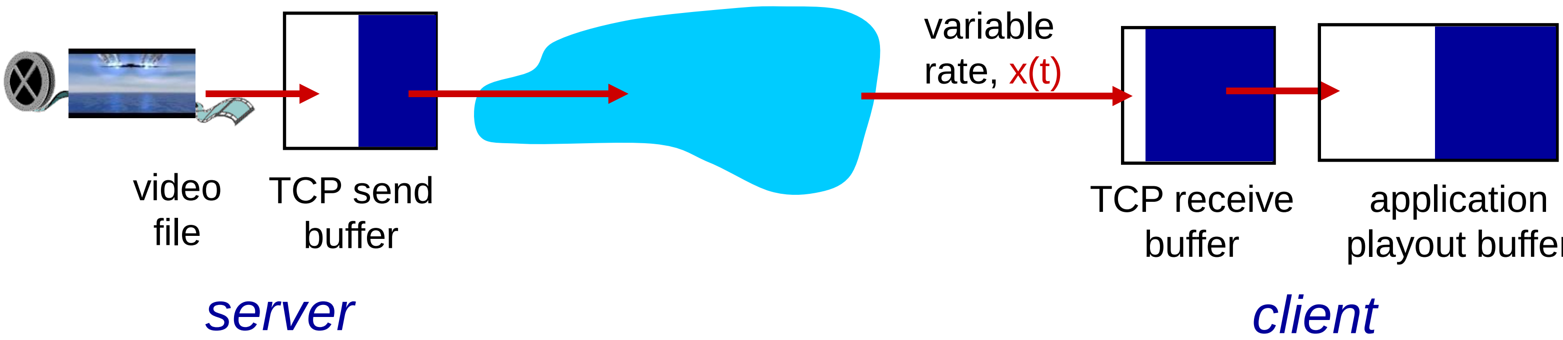
3. The third drawback is that many firewalls are configured to block UDP traffic, preventing the users behind these firewalls from receiving UDP video.

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 watch the video, the client issues an HTTP GET request for that URL and establishes a TCP connection with the server.
- 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 playout:
  - it periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen.

HTTP Streaming illustration:

- multimedia file retrieved via HTTP GET
- sent at a 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
- Used in YouTube, Netflix, …

DASH: Dynamic Adaptive Streaming over HTTP

- Basic HTTP streaming has a major shortcoming: All clients receive the **same encodin**g of the video, despite the large variations in the **amount of bandwidth available** to a client,
  - across different clients
  - over time for the same client.
- In DASH, the video is encoded into **several different versions**, with each version having a **different bit rate** and, correspondingly, a different quality level.
- The client dynamically requests chunks of video segments of a few seconds in length from the different versions:
- when the amount of available bandwidth is:
  - **high**, the client selects chunks from a high-rate version;
  - **low,** it selects from a low-rate version.
- The client selects different chunks **one at a time** with HTTP GET request messages

DASH Streaming: a manifest file

- The possibility to adapt the video quality depending on the available bandwidth is particularly important for mobile users, experiencing fluctuation in the bandwidth available.
- With DASH, each video version is stored in the HTTP server, each with a different URL.
- The HTTP server also has a manifest file, which provides a URL for each version along with its bit rate.
- The client first requests the **manifest file** and learns, about the available versions
  - then selects **one chunk at a time** by specifying a URL and a byte range in an HTTP GET request message for each chunk.

DASH Streaming: bandwidth adaptation

- While downloading chunks, the client also **measures** the received bandwidth and runs a **rate determination** algorithm to select the chunk to request next.
  - If the client has a lot of video buffered and if the measured receive bandwidth is high, it will choose a chunk from a high-rate version.
  - If the client has little video buffered and the measured received bandwidth is low, it will choose a chunk from a low-rate version.
- DASH therefore allows the client to freely switch among different quality levels.

Content Distribution Networks (CDN)

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

- *option 1:* single, large "mega-server".

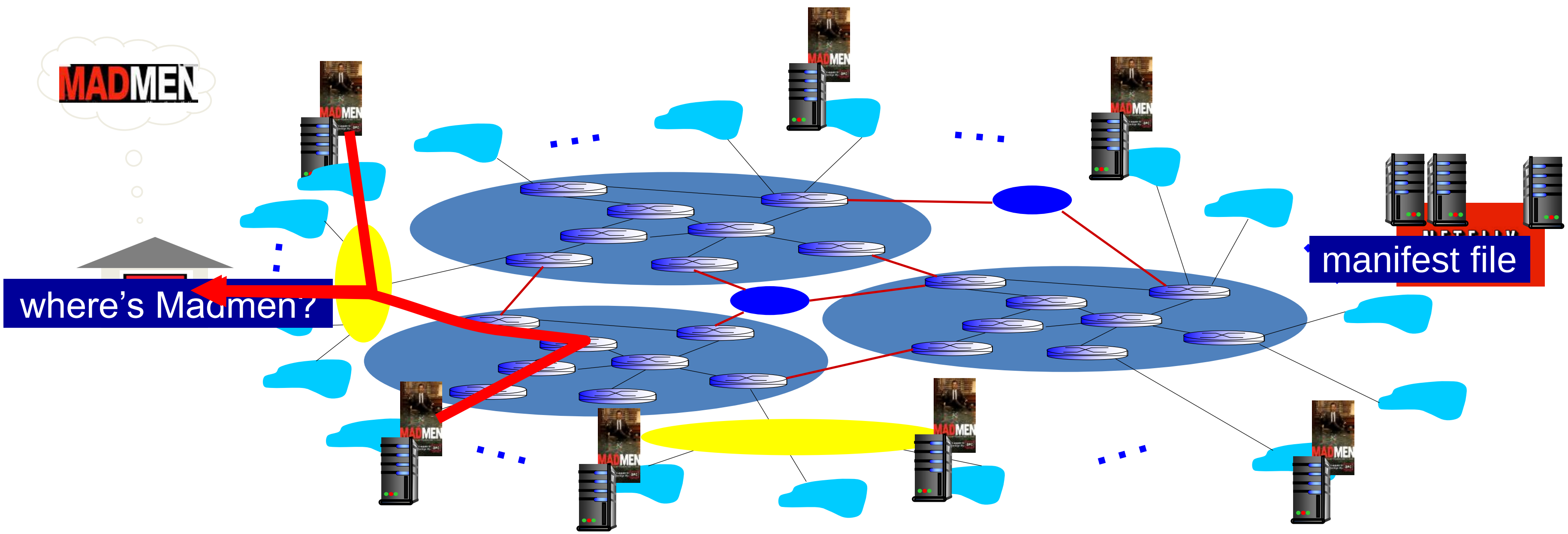
  No good since there is:
  - A single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link
- A single server solution is NOT scalable

Content Distribution Networks (CDN)

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?
- *option 2: Multiple CDN servers* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN):*
- *Two concepts:*
  1. *enter deep:* push CDN servers deep into many access networks
     - close to users
     - used by Akamai, 1700 locations
  2. *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
     - used by Limelight

Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested

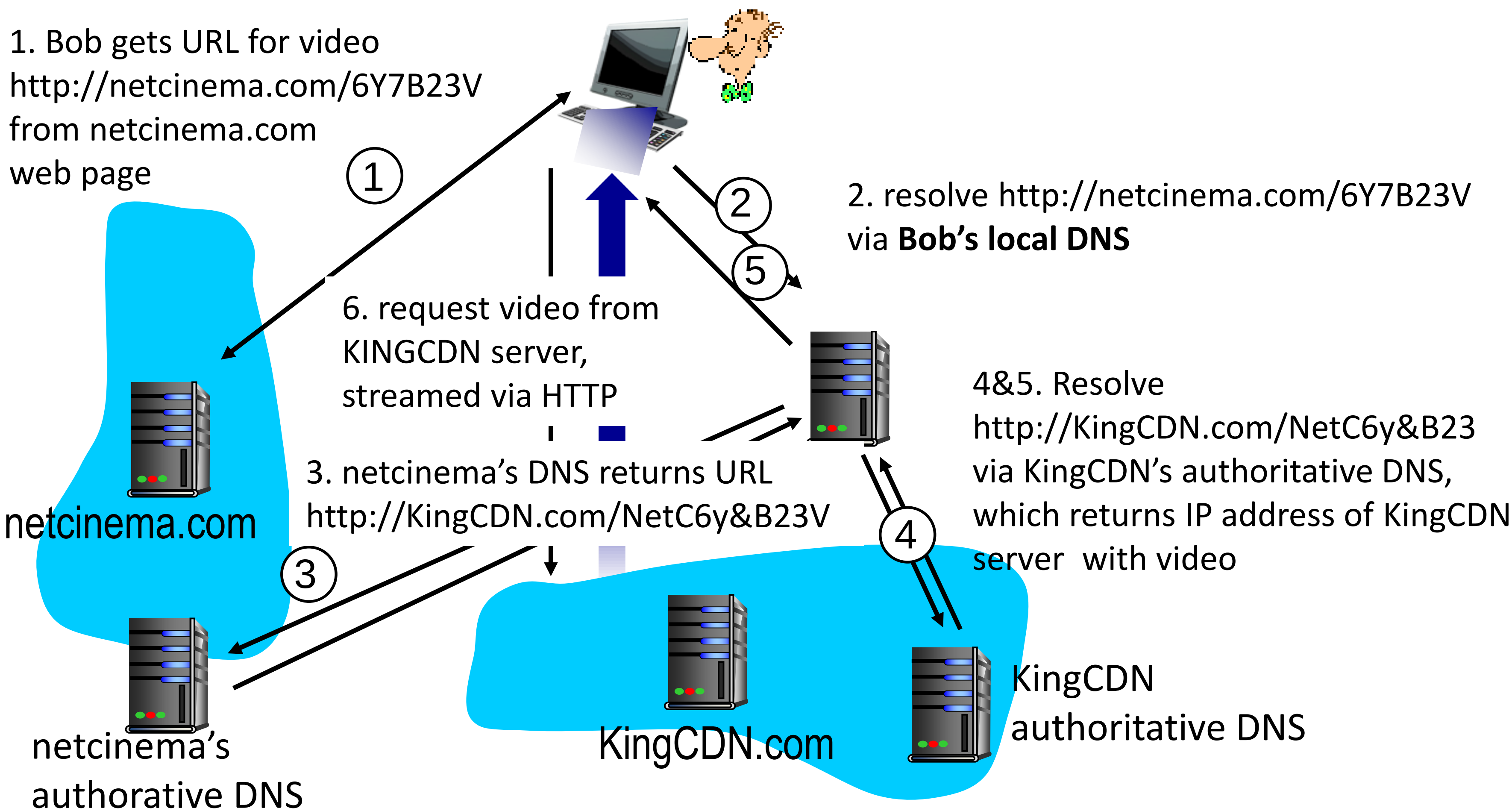


CDN: "simple" content access scenario

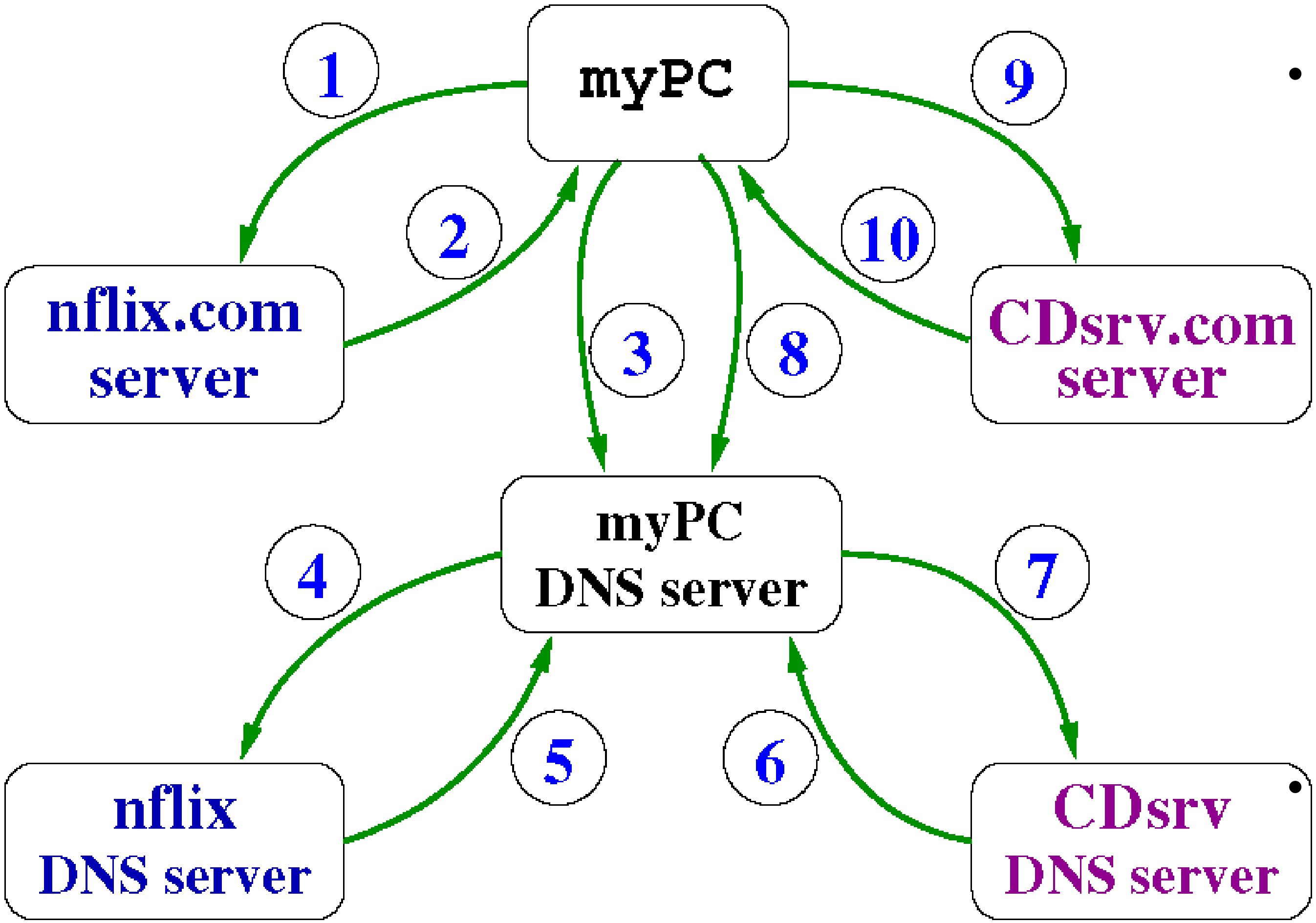
1. Bob (client) goes to the netcinema.com web page and gets the URL for a video: http://netcinema.com/6Y7B23V
2. When requesting the video, Bob's Local DNS resolves the URL by contacting the netcinama's DNS
3. The netcinama's authorative DNS returns: http://KingCDN.com/NetC6y&B23V where the video is stored
4. Bob's Local DNS resolves the new URL by contacting the KingCDN's authorative DNS
5. the KingCDN's DNS returns IP address of KingCDN server with video
6. requested video from KingCDN server, streamed via HTTP

CDN: "simple" content access scenario again

- Bob (client) requests video http://netcinema.com/6Y7B23V
- video stored in CDN at http://KingCDN.com/NetC6y&B23V



Content access example



- In the example **myPC** is searching for a movie on the **nflix.com** server.
- After the movie is selected and allowed to be screened (e.g. after paying the price of access to the movie) a PLAY app in myPC use DNS (Domain Name Services) to obtain the IP address of the selected movie to be screened from the **CDsrv.com** server.
- In the example the **DNS servers** use the iterative method of obtaining the IP address.

1. I register/login to the **nflix.com** video server and search for the movie. I selected “movie53” from the list offered by the server.
2. The netfix sends the URL for the selected movie, e.g. **nflix.com/movie53 .**
3. My PLAY app needs issue the HTTP request GET **nflix.com/movie53.** Since the IP for this URL is not known, PLAY sends the DNS request (what is the IP address of nflix.com/movie53) to myPC local DNS server.
4. Most likely, **myPC DNS server** does not know the requested IP address and consults the **nflix.com DNS server**
5. Now the nflix.com DNS server has an opportunity to select the server from which the video will be streamed, hence gives the DNS response: I do not know the requested IP address but CDsrv.com DNS server should know. Ask for CDsrv.com/Amovie53H
6. Now myPC DNS server sends the DNS request to the CDsrv.com DNS server
7. The response is the required IP address for CDsrv.com/Amovie53H is 134.22.55.33
8. myPC DNS server passes this information to the PLAY app
9. The PLAY app sends GET CDsrv.com/Amovie53H (IP = 134.22.55.33)
10. The HTTP response bring the requested video to myPC

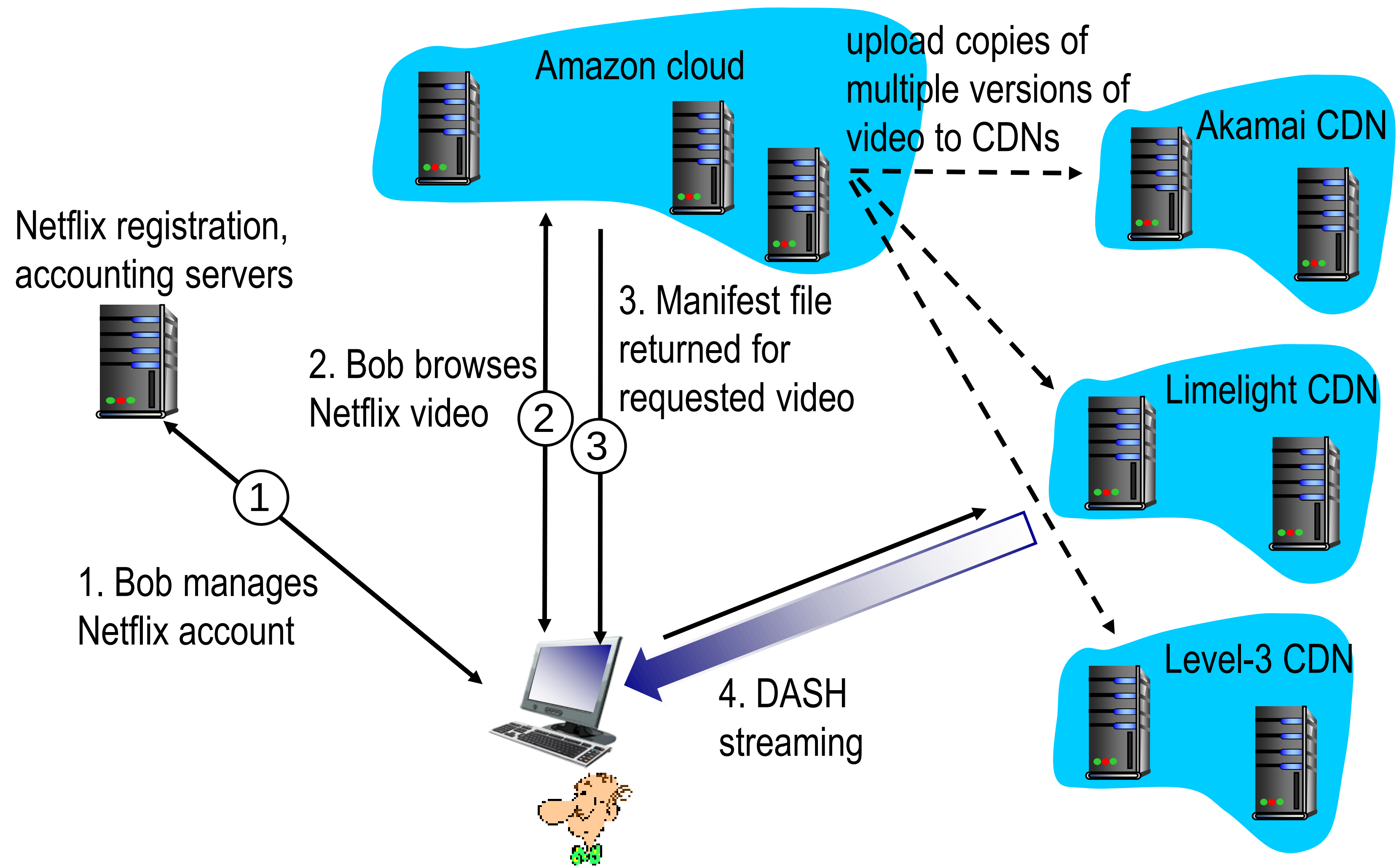
CDN cluster selection strategy

- *problem:* how does CDN DNS select "good" CDN node to stream to client
  1. pick CDN node geographically closest to client
  2. pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping the access ISPs, reporting results to CDN DNS)
  3. IP anycast: sending packets to a single member of a group of potential receivers that are all identified by the same destination address
- *alternative:* let *client* decide – give a client a list of several CDN servers
  - client pings servers, picks "best"
  - Netflix approach

Case study: Netflix

- an American provider of on-demand Internet streaming media
- 2014: 50 million global subscribers in 41 countries
- 32.3% video streaming market share in the United States.
- 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 encodings) 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



Voice-over-IP (VoIP)

- *VoIP end-end-delay requirement*: needed to maintain "conversational" aspect
  - higher delays are noticeable and impair interactivity
  - < 150 msec: good
  - > 400 msec: bad
  - includes application-level (packetization, playout), network delays

How to implement:

- *session initialization:* how does a callee advertise IP address, port number, encoding algorithms?
- *value-added services:* call forwarding, screening, recording
- *emergency services:* 911, 000

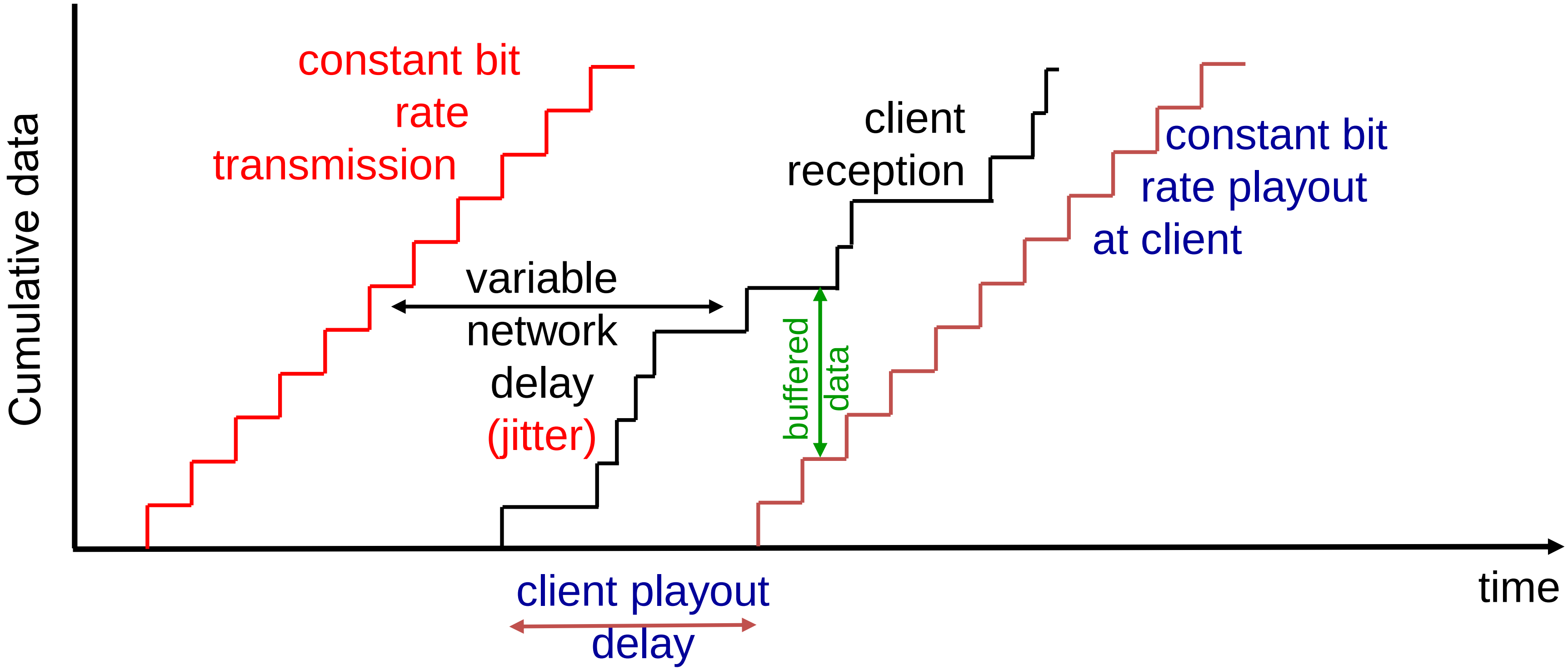
VoIP characteristics

- Speaker's audio: alternating talk spurts (bursts) with silent periods.
  - 64 kbps during talk spurt
  - packets 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 segments into the 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,
  - loss concealment,
- packet loss rates between 1% and 10% can be tolerated

Delay jitter



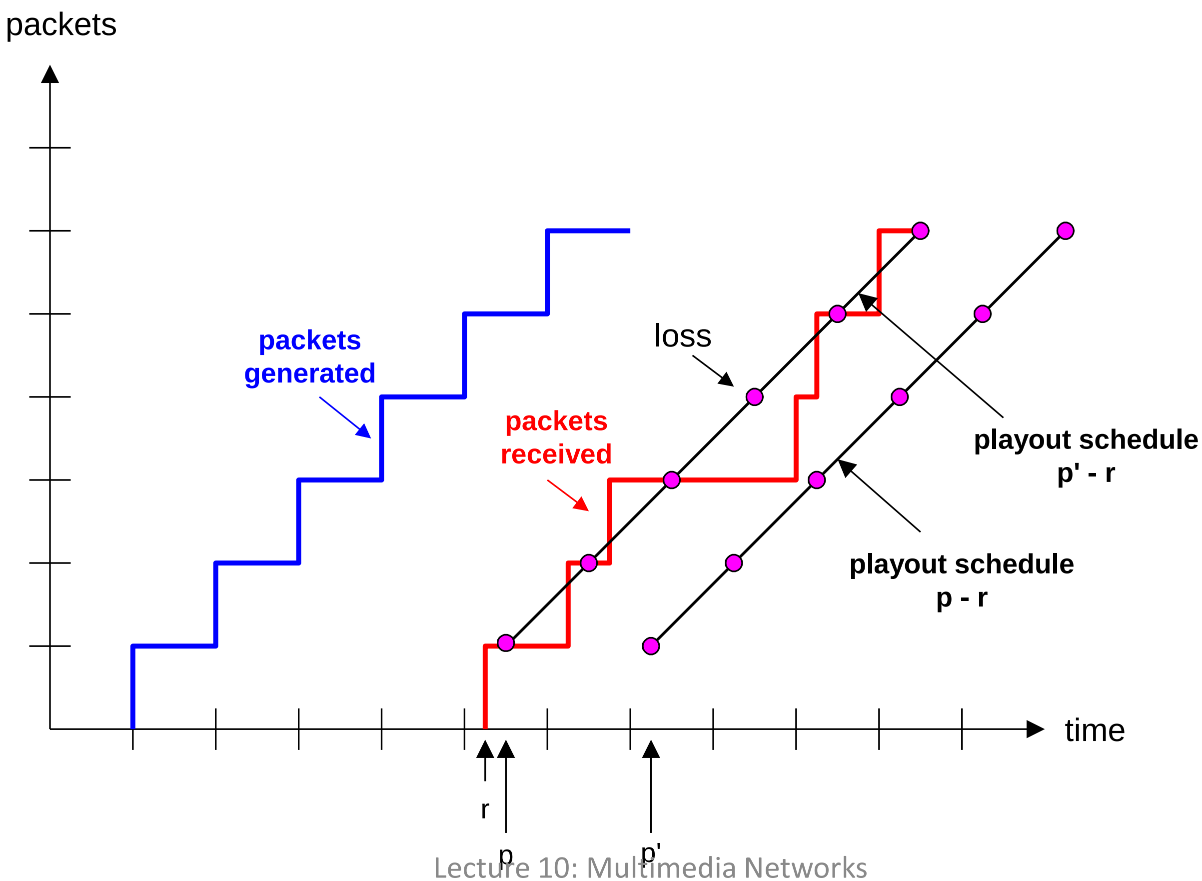
- end-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

VoIP: fixed playout delay

- receiver attempts to play out each chunk exactly *q* msec after chunk was generated.
  - chunk has time stamp *t:* play out chunk at *t+q*
  - chunk arrives after *t+q*: data arrives too late for playout: data considered "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'*



(Optional) 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 or elongated
  - chunks still played out every 20 msec during talk spurt
- adaptively estimate packet delay: (EWMA - exponentially weighted moving average, recall TCP RTT estimate):

$$\underbrace{d_i}_{\text{delay estimate after ith packet}} = (1-\underbrace{\alpha}_{\text{small constant, e.g. 0.1}})d_{i-1} + \alpha\,\underbrace{(r_i - t_i)}_{\text{measured delay of ith packet}}$$

delay estimate after ith packet

small constant, e.g. 0.1

time received - time sent (timestamp)

measured delay of ith packet

Adaptive playout delay (2)

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

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

- estimates d_i, v_i calculated for every received packet, but used only at start of talk spurt
- for the first packet in talk spurt, playout time is:

$$playout\_time_i = t_i + d_i + Kv_i$$

- K =4 (typically)
- remaining packets in talk spurt are played out periodically

Adaptive playout delay (3)

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

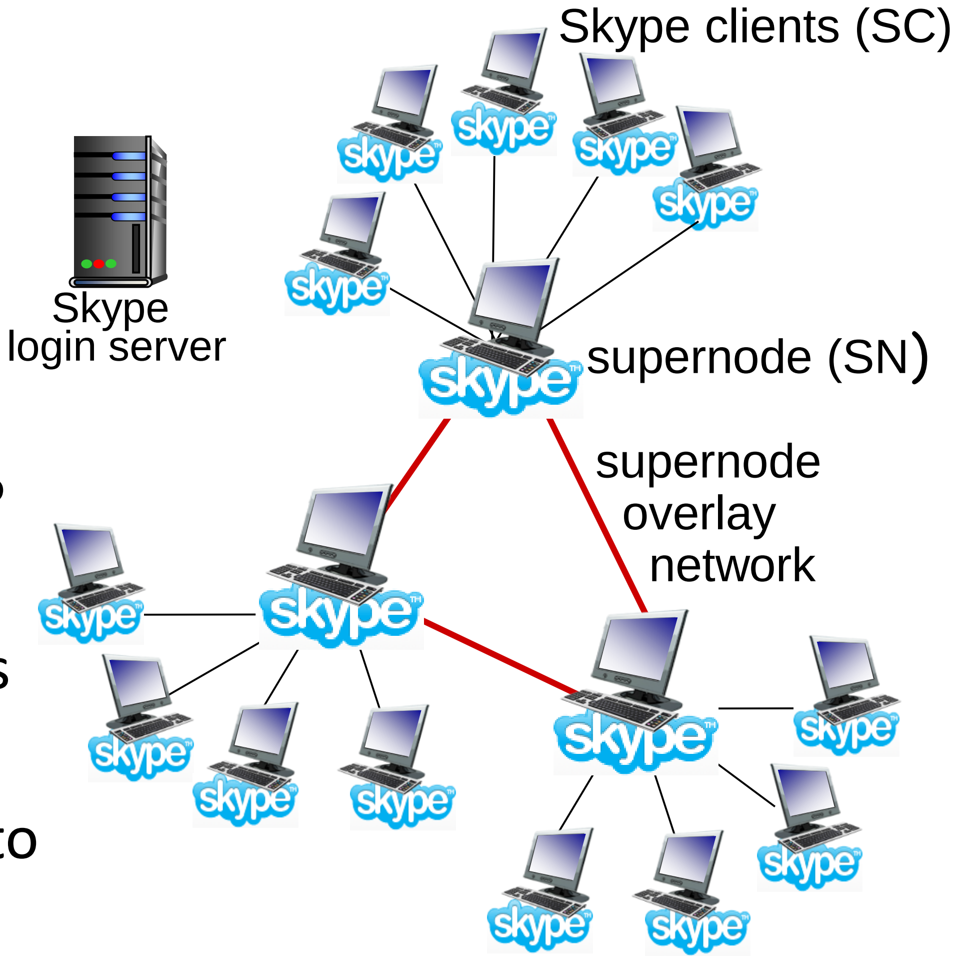
- if no loss, receiver looks at successive timestamps
  - difference of successive stamps > 20 msec --> talk spurt begins.
- with loss possible, receiver must look at both time stamps and sequence numbers
  - difference of successive stamps > 20 msec *and* sequence numbers without gaps --> talk spurt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

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



Skype Client operation

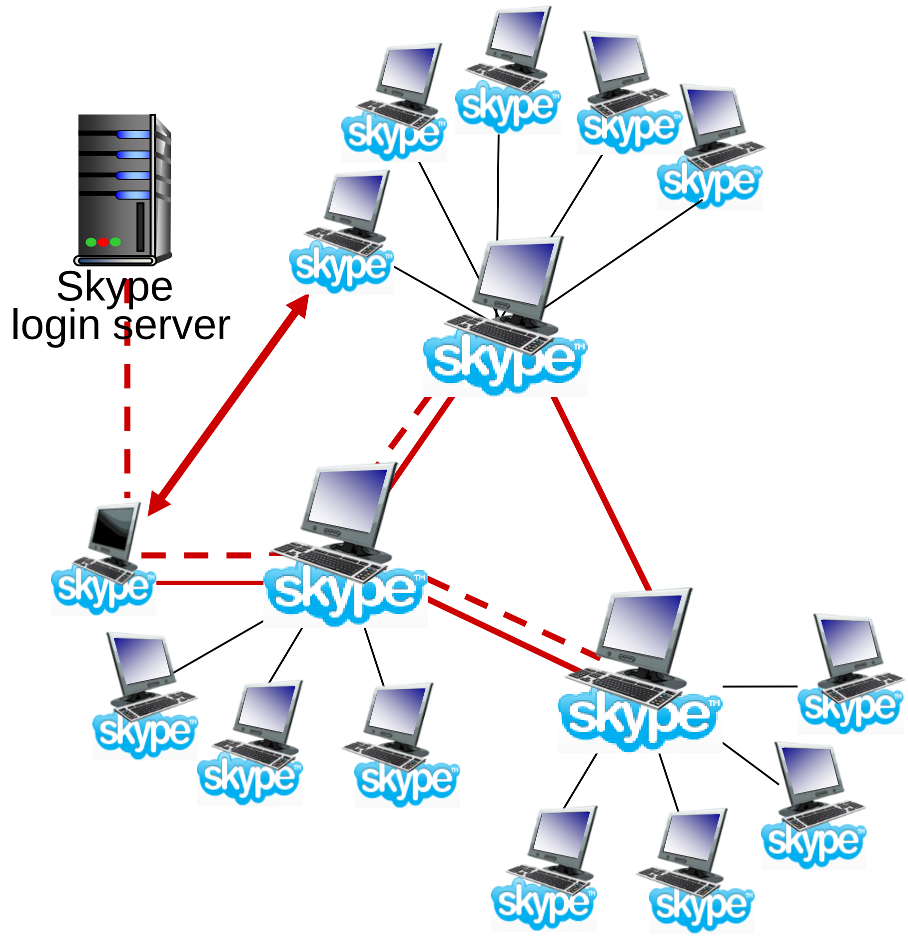
skype client operation:

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
   - or client buddy list

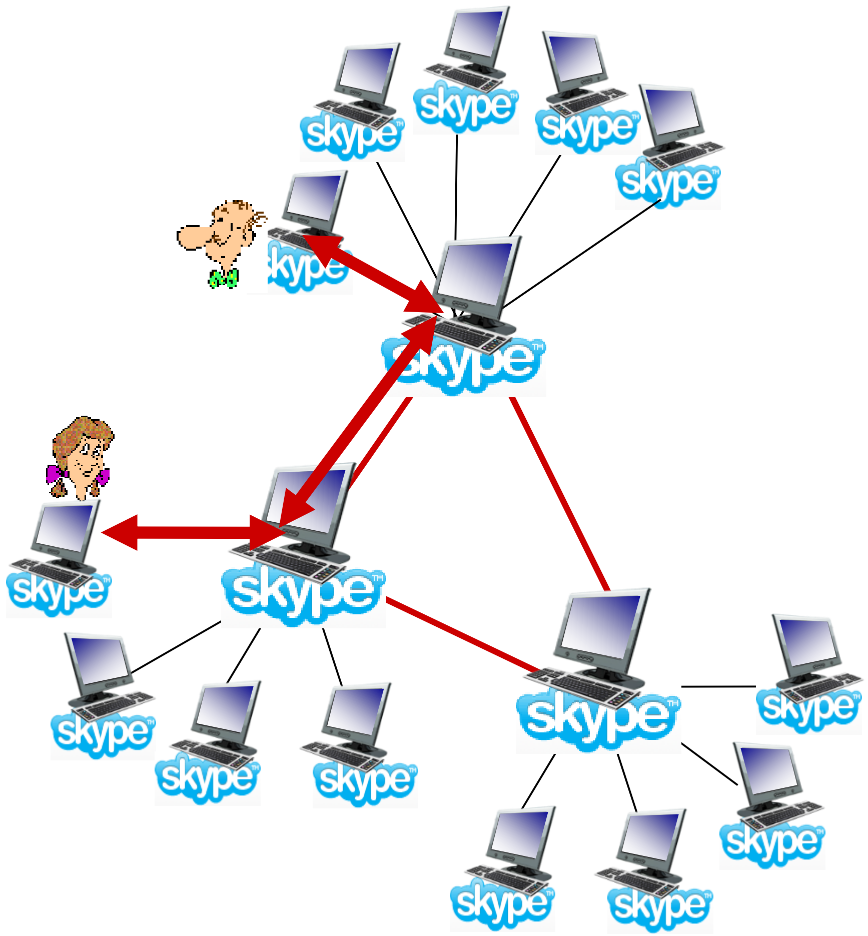
4. initiate call directly to callee



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



Real-Time Transport Protocol (RTP)

- RTP
- If the time permits