

Network Service Model

What potential service model an application may ask from the “channel” transporting packets from sender to receiver?

Example services for individual packets:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

Example services for a *flow* of packets:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

What Transport Services Does an App Need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be “effective” and does not send more than a maximum rate
- other apps (“elastic apps”, e.g., file transfer, email, web browsing) make use of whatever bandwidth they can get

Transport Service Requirements of Common Apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	same day
e-mail	no	elastic	same day
Web documents	no	elastic	interactive
instant messaging	no	elastic	interactive
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-6Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps and up	yes, 100's msec

Multimedia Apps Quality of Service

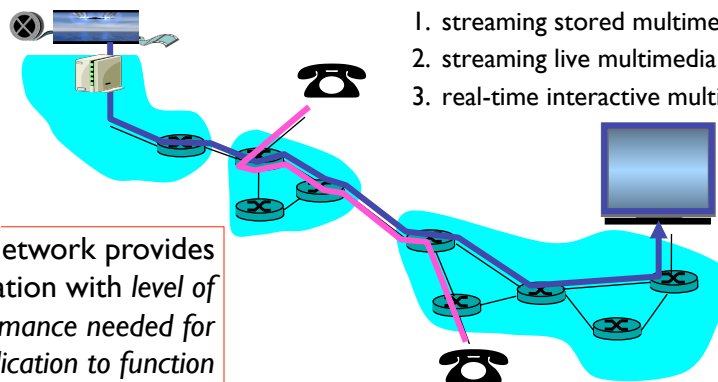
Multimedia applications: networked audio and video, distributed interactive worlds (multiplayer gaming)

Classes of MM applications:

1. streaming stored multimedia
2. streaming live multimedia
3. real-time interactive multimedia

QoS

network provides application with level of performance needed for application to function



Internet Transport Protocol Services

TCP service:

- *connection-oriented*: setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control*: sender won't overwhelm receiver
- *congestion control*: throttle sender when network overloaded
- *does not provide*: timing, minimum bandwidth guarantees

UDP service:

- “no frills,” “bare bones,” connectionless Internet transport protocol, does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee
- unreliable data transfer between sending and receiving process
 - each UDP segment handled independently of others, UDP segments may be lost or delivered out of order to app
 - “best effort” delivery: no reliability, no retransmission, no reordering of packets: no ACKs, no seq #'s, no need for connection establishment

Q: Why bother with UDP?

UDP: User Datagram Protocol

Advantages of using UDP:

- faster, no connection establishment/tear-down stages (1 vs. 2.5 rtts)
- simpler server: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired
- broadcast & multicast can only use UDP. Why?

Often used for streaming multimedia apps

- loss tolerant
- rate sensitive

Other UDP uses: DNS, SNMP

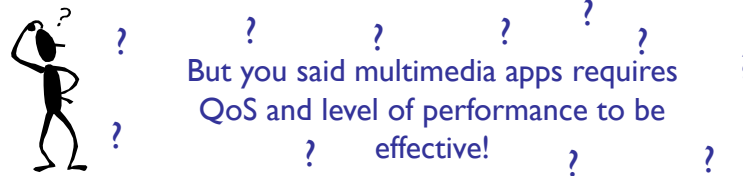
Reliable transfer over UDP: add reliability at app layer

- application-specific error recovery!

Multimedia Over Today's Internet

TCP/UDP/IP: "best-effort service"

no guarantees on delay, loss



Shouldn't multimedia traffic be given higher priority?
Why can't I make resource reservation and get quality of service (QoS) guarantees?

Whither Network Support for Multimedia Traffic?

Mechanisms needed to support QoS/VC:

- Reservation protocol (RSVP)
- Admission control
- Packet classification
- Realtime scheduling
- Pricing scheme
- Accounting and billing

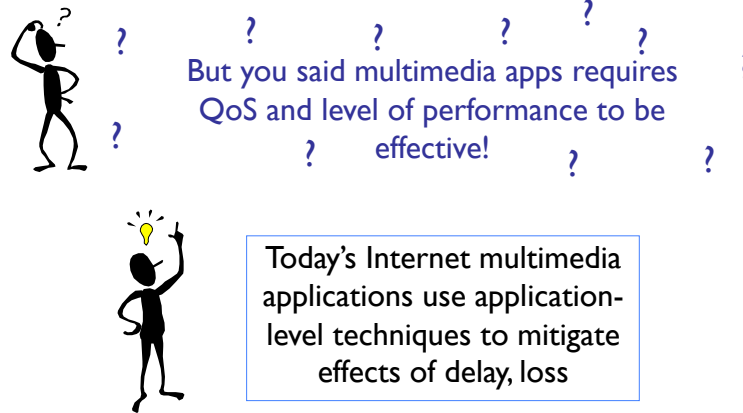
Proposed real-time services (defunct):

- Integrated Services: negotiated per flow
- Differentiated Services: negotiated per AS
- why not deployed?
 - over-provisioning of Internet backbone led to 2% utilization
 - no compelling case for deployment of real-time services
 - bottleneck is at the access networks, but no competitive choice

Multimedia Over Today's Internet

TCP/UDP/IP: "best-effort service"

no guarantees on delay, loss



Today's Internet multimedia applications use application-level techniques to mitigate effects of delay, loss

Multiplayer Gaming: In-game Networking Topics

Topology:

- client-server or peer-to-peer

Computing model:

- distributed object vs. message passing

Bandwidth requirement

Latency requirement

Latency effect: consistency

Which transport protocol to use?

- TCP, UDP, Reliable UDP

<http://www.mmogchart.com/charts>

Mutiplayer Gaming Traffic

What information is sent in a multiplayer game?

- depends on your computing model:
distributed object or message passing
- distributed object: game state, e.g., coordinates, status, action, facing, damage
- message passing: user keystrokes, e.g., commands/moves
For AoE: 1 every 1.5-2 sec, up to 3-4 commands/sec during battles (but some of these are redundant and can be filtered out)

Bandwidth Requirement

Bandwidth requirement has been **HIGHLY** optimized

- even with audio chat, takes up at most 8 Kbps
- so, bandwidth is not a big issue
 - but note the asymmetric nature:
for N players, you receive $N-1$ times the amount of bytes you send out
- must be continually vigilant against bloat

However, with player-created objects and worlds, bandwidth becomes an issue again: use streaming, levels of details, and pre-fetching

Latency Requirement

How is latency different from bandwidth?

Tolerable round-trip latency thresholds:

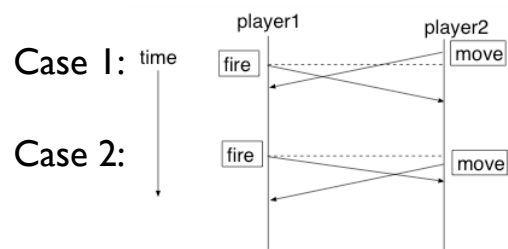
- RTS: ≤ 250 ms not noticeable,
250-500 ms playable, > 500 ms noticeable
- FPS: ≤ 150 ms preferred
- car racing: < 100 ms preferred,
100-200 ms sluggish, ≥ 500 ms, car out of control

Players' expectation can adapt to latency

- it is better to be slow but smooth than to be jittery

Latency Effect: Consistency

Problem statement:



In both cases:

- at player 1: player2's move arrives after player1's fire
- at player 2: player1's fire arrives after player2's move

Should player2 be considered shot in both cases?

Or only in the second case?

Synchronization

Synchronization ::
order moves by their times of occurrence

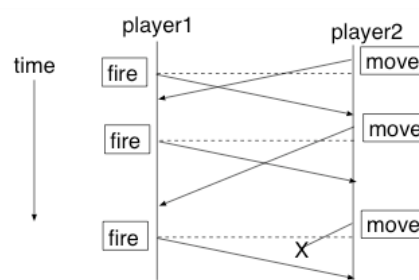
Assume globally synchronized clocks

Out-of-synch worlds are *inconsistent*

Small inconsistencies not corrected can lead to large compounded errors later on (deer not speared means one less villager means slower barrack build, etc.)

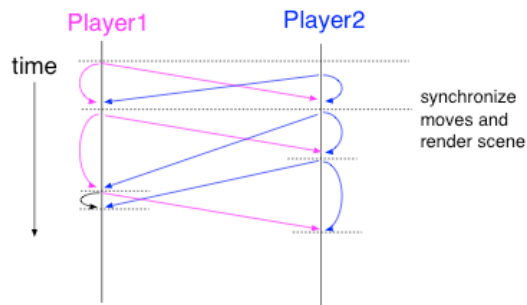
When to Render a Move?

How long do you have to wait for the other players' moves before rendering your world?



Lock-step Protocol

Algorithm: Each player receives all other players' moves before rendering next frame



Problems:

- long Internet latency
- variable latencies $\Rightarrow$ jittery game speed
- game speed determined by the slowest player

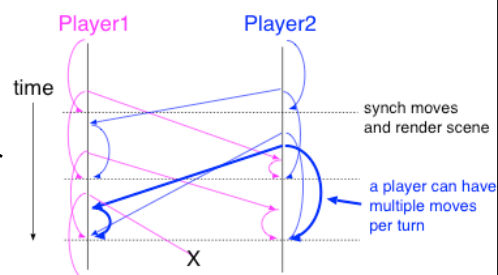
Bucket Synchronization

Algorithm:

- buffer both local and remote moves
- play them in the future
- each bucket is a round, say of about 200 ms
- bucket size can be adapted to measured rtt

Smoother player, but:

- game speed (bucket size) still determined by slowest player
- what if a move is lost or late?



Pessimistic Consistency

Every player must see the *exact* same world, e.g., AoE/AoK/AoM:

- each player simulates its own copy of the world
- all the worlds must be in synch
- uses bucket synchronization
- each player sends moves to all other players
- dropped packets are retransmitted
- a designated host collects measured RTTs from all players and set future bucket sizes

Problems (those of bucket synchronization):

- variable game speed if lost packets must be retransmitted
- speed determined by the slowest player

Dead Reckoning and Roll-back

Dead reckoning, a.k.a. client-side prediction

- extrapolate next move based on prior moves
- compute the velocity and acceleration of objects to dead reckon



- players can help by sending velocity and acceleration along
- obviously, only works if velocity and acceleration haven't changed

In case of inconsistency:

- server assumed to always have authoritative view
- when clients correct (roll-back) inconsistent views, players may experience “warping”

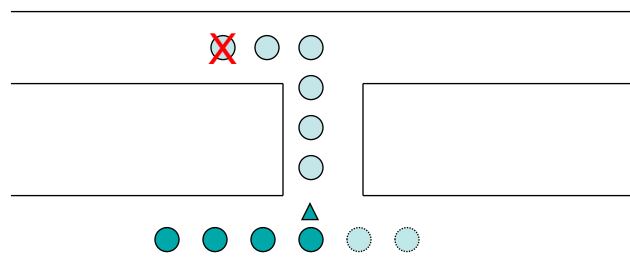
Optimistic Consistency with Roll-back

Observation: dead reckoning doesn't have to be limited to lost packets!

Half-Life:

- each client plays back its own moves immediately and sends the moves to server
- each client **also dead reckons the other players' moves**
- server computes world and sends its authoritative version to all clients
- clients reconcile dead reckoned world with server's version
- can result in some jerkiness and perception of “shooting around corner”
- only need to synchronize important events, but must be careful that dead reckoning error doesn't get compounded over time

Shooting Around Corner



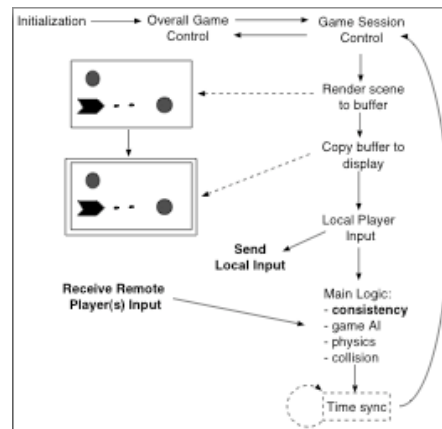
Consistency: Correctness

For consistency *all* user input *must* pass through the synchronization module

Be careful with random number generators: isolate the one used for game-state updating from other uses (ambient noise etc.)

Design for multiplayer from the start

- single-player becomes a special case of single-client multiplayer game



Consistency: Smoothness

For smoother playback, decouple bucket size from frame rate (even AoE does this)

Immediately render local moves

Modify game design to allow for latency and loss, e.g.,

- make players wait for elevator
- teleportation takes time
- require multiple hits per kill
- let bullet/missile have flying time
- build in inertia, don't allow sudden change in facing

Reducing Consistency Check

Do *area-of-interest* management (a.k.a. relevance filtering):

- *aura*: how far you can be sensed (ninja and cloaked ships have 0 aura)
- *nimbus*: how far you can sense (empath and quantum-sensor have large nimbus)

Perform consistency check only when B is within A 's nimbus and A is within B 's aura

Aura and nimbus are defined for a given set of “game technology” (e.g., cloaking device, quantum sensor, etc.)

Which Transport Protocol to Use?

Gaming requirements:

- late packets may not be useful anymore
- lost information can sometimes be interpolated
- (though loss statistics may still be useful)

Use UDP in game:

- can prioritize data
- can perform reliability if needed
- can filter out redundant data
- use soft-state
- send absolute values, not deltas
- or if deltas are used, send “baseline” data periodically
- *must* do congestion control if sending large amount of data

Multimedia Apps Quality of Service

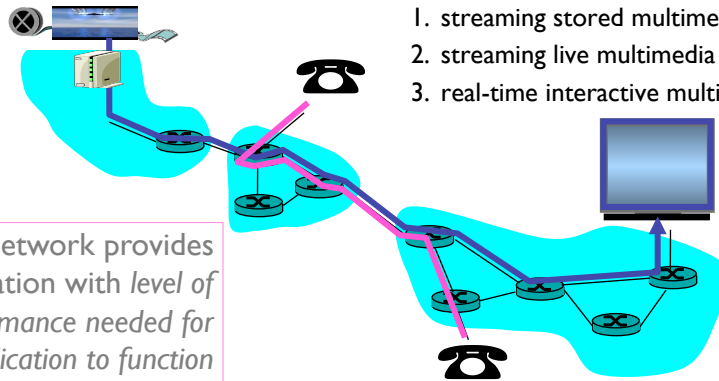
Multimedia applications:
networked audio and video,
distributed interactive
worlds (multiplayer gaming)

Classes of MM applications:

1. streaming stored multimedia
2. streaming live multimedia
3. real-time interactive multimedia

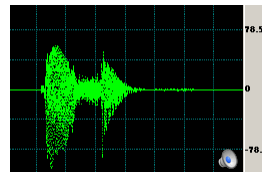
QoS

network provides
application with *level of
performance needed for
application to function*



Signal Digitization: Overview

Analog audio signal:



Sampling: reading the
signal at certain rate to
collect samples

Signal can be reconstructed
from the samples

A) ANALOG WAVEFORM



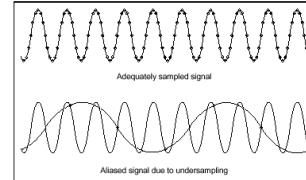
B) DIGITIZED SAMPLE



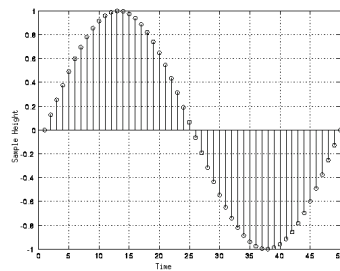
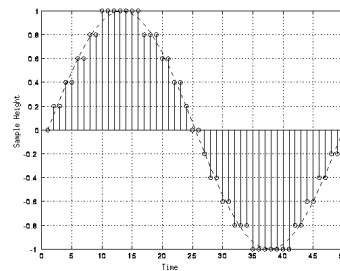
FIGURE 1: Analog waveform and digital equivalent

Signal Digitization: Overview

Inadequate sampling can lead to aliasing:



Quantization: partitioning potential signal values into levels and represent each level with a single number

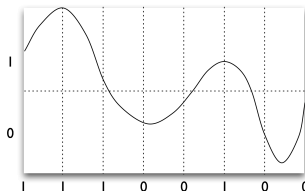
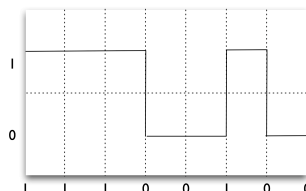


Bandwidth Requirement

Higher sampling rate and finer quantization levels result in better signal reconstruction at the cost of higher bandwidth requirement

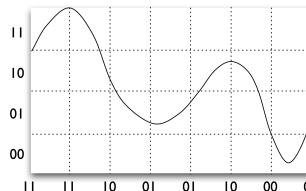
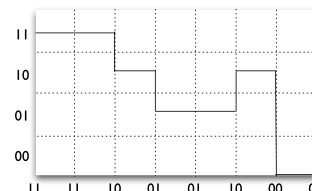
Example 1:

- sampling rate: 8 samples/sec
- quantization: 2 levels $\Rightarrow$ 1 bit per sample
- bandwidth requirement: 8 bps



Example 2:

- sampling rate: 8 samples/sec
- quantization: 4 levels $\Rightarrow$ 2 bit/sample
- bandwidth requirement: 16 bps



Audio Bandwidth Requirements

Bandwidth requirements of several popular encoding standards:

- PCM (telephone):
 - 8,000 samples/sec
 - 8 bits/sample (256 quantization levels)
 - ⇒ 64 Kbps data sent at the rate of 160 bytes/packet, once every 20 ms
- CD music:
 - 44,100 samples/sec
 - 16 bits/sample
 - ⇒ 705.6 Kbps mono, 1.411 Mbps stereo

Audio compression standards:

- GSM: 13 Kbps
- G.729: 8 Kbps
- G.723: 6.4 and 5.3 Kbps
- MPEG2 Layer 3 (MP3): 128 or 112 Kbps
- MPEG4/AAC: 96-128 Kbps, High-Efficiency AAC: 32-48 Kbps
- proprietary standards: Sorenson, Nellymoser

Video

Video is a sequence of images/frames displayed at a constant rate (moving pictures)

Digital image is an array of pixels, each pixel represented by bits

Examples:

- single frame image encoding:
 - 1024x1024 pixels, 24 bits/pixel ⇒ 3 MB/image
- movies: 24 frames/sec ⇒ 72 MB/sec
- TV: 30 frames/sec ⇒ 90 MB/sec
- Lower resolution requires less bandwidth, VHS quality 96 Mbps

Video Bandwidth Requirements

Sequence of images contains redundancy

- spatial redundancy within an image
- temporal redundancy across images

Video compression works by removing spatial and temporal redundancies

Video compression standards:

- MPEG1: 1 Mbps (VCD: VHS quality)
- MPEG2: 3-9 (usually 6) Mbps (DVD quality)
- MPEG4/H.264: 500 Kbps to 6 Mbps (HDTV quality)
- proprietary standards: On2, Windows Media
- upcoming standard: Scalable H.264: video encoded into several layers, cumulatively each additional layer gives higher resolution

Latency and Loss Requirements

Fundamental characteristics of multimedia applications:

Typically **delay sensitive**

- live audio < 150 msec end-to-end delay is not perceptible
- 150-400 msec end-to-end delay is tolerable
- > 400 msec end-to-end delay makes audio unintelligible
- streaming audio: can wait 5 secs or more before starts of playback
- low variability of packet delays (jitter) within the same packet stream

But **loss tolerant**: infrequent losses cause minor glitches

- 1% to 10% or even 20% loss is tolerable

Antithesis of data, which is loss intolerant but delay tolerant

Multimedia Apps Quality of Service

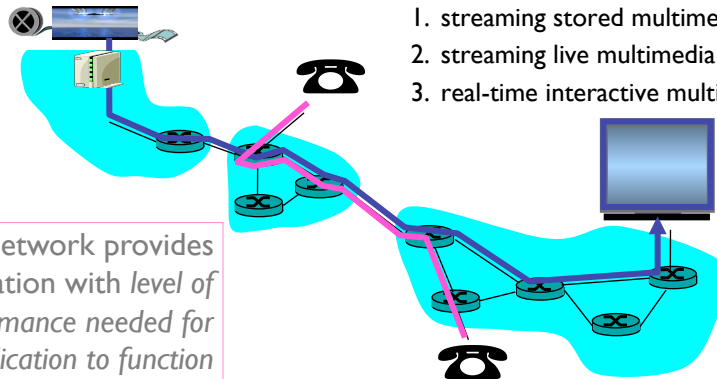
Multimedia applications:
networked audio and video,
distributed interactive
worlds (multiplayer gaming)

Classes of MM applications:

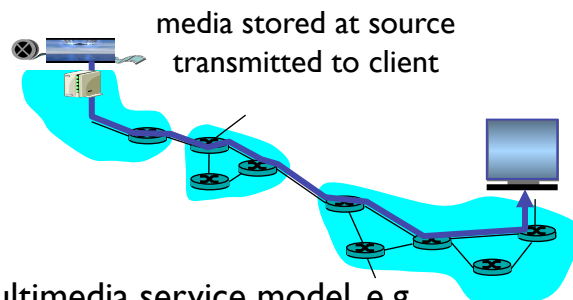
1. streaming stored multimedia
2. streaming live multimedia
3. real-time interactive multimedia

QoS

network provides
application with level of
performance needed for
application to function



Streaming Stored Multimedia

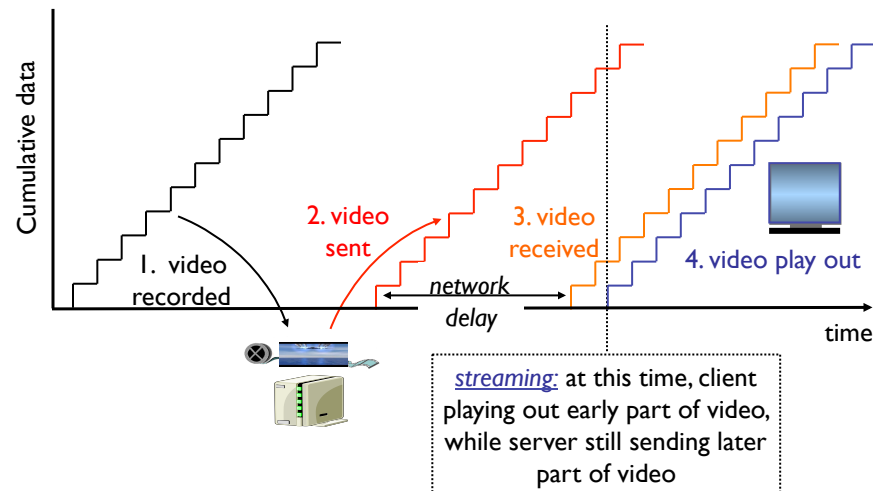


Most common multimedia service model, e.g.,
YouTube, Hulu, Video on Demand (VoD)

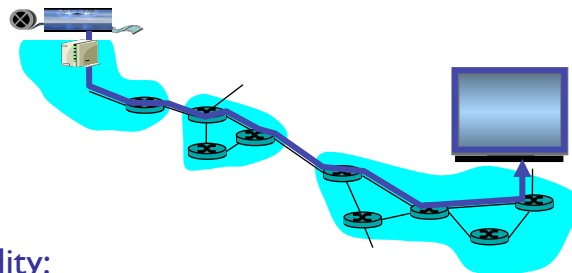
streaming:

- client playout begins *before* all data has arrived
- subsequent data must arrive in time for playout

Streaming Stored Multimedia



Streaming Stored Multimedia: Interactivity



VCR-like functionality:

client can pause, rewind, FF, push slider bar

User tolerance to interactive delay:

- 10 sec initial play out delay OK
- 1-2 sec until command effect OK

Streaming Live Multimedia

Examples:

- Internet radio talk show
- Live sporting event
- Breaking news
- Examples:
Zattoo (US), Octoshape (DK), Livestation (UK), Wilmaa (CH)

Streaming:

- playback cannot lag more than 30 seconds
after live transmission; Zattoo: ~8 secs, Livestation: ~13 secs

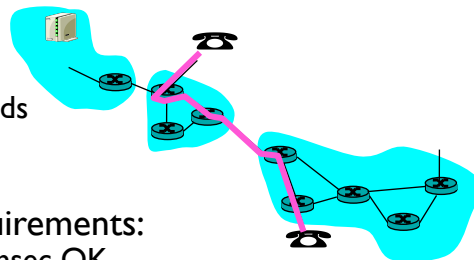
Interactivity:

fast forward impossible
rewind, pause technically possible (may not be licensed)

Interactive, Real-Time Multimedia

Sample applications:

- IP telephony
- video conferencing
- distributed interactive worlds
(multiplayer gaming)



End-end audio delay requirements:

< 150 msec good, < 400 msec OK
includes application-level (packetization) and network delays
higher delays noticeable, impair interactivity

Session initialization

how does callee advertise its IP address, port number, encoding algorithms?

Streaming Multimedia Techniques

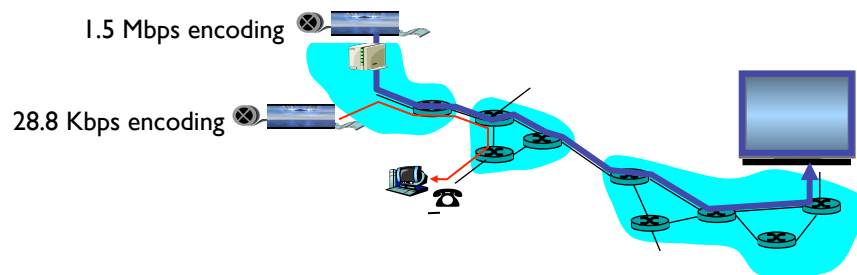
Application-level streaming techniques
for making the best out of best effort
service:

- multiple encodings of multimedia
- receiver-side buffering
- adaptive playback
- use of UDP versus TCP
- FEC

Media Player

jitter removal
decompression
adaptive playback
error concealment
graphical user interface
with controls for interactivity

Multiple Encodings

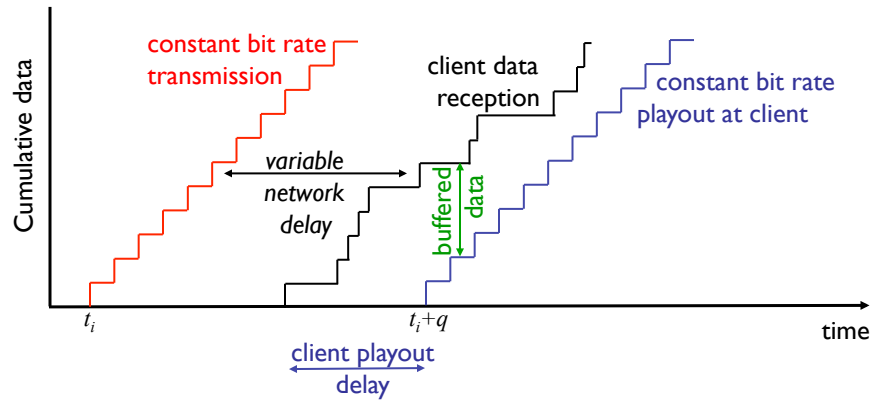


How to handle different client receive rate
capabilities, e.g., 236 Kbps EDGE, 100Mbps LAN?

Server stores and transmits multiple copies of video,
encoded at different rates
(or use scalable H.264!)

Delay Jitter

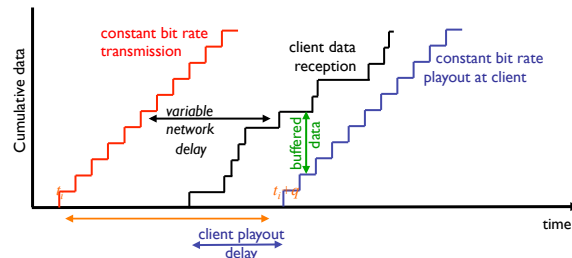
Jitter: variance in experienced delay



Observation: multimedia applications on the Internet are *playback application*, i.e. samples generated at time t_i are played back at time $t_i + q$

Receiver-side Buffering

Jitter removal:
receiver-side
buffering and
delayed playback



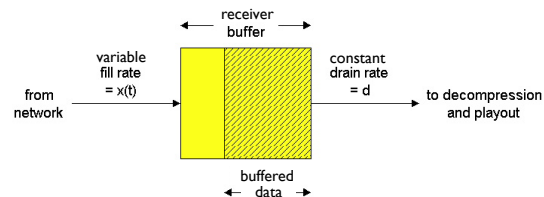
If q is large enough, by the time sampled must be played back, it would have arrived at the receiver

Playback point: transmitted time (t_i) + e2e delay (τ_i) + buffer time (b_i)

Receiver-side Buffering

Why not buffer/
download the
whole file before
playing it back?

-
-
-



Fixed Playout Delay

Receiver attempts to play out each sample exactly q msecs after it was generated

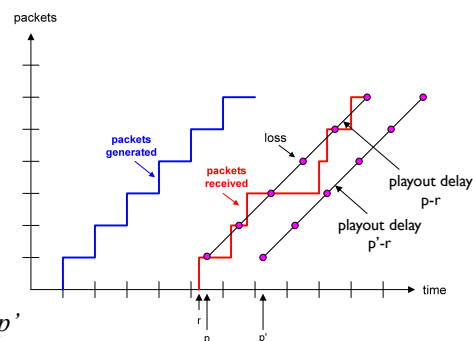
- sample has time stamp t : play out sample at $t+q$
- sample arrives after $t+q$: data arrives too late for play out, data "lost"

Tradeoff for q :

- large q : less packet loss
- small q : better interactive experience

For example, sender generates packets every 20 msec:

- first packet received at time r
- first playout schedule: begins at p
- second playout schedule: begins at p'



Adaptive Playout Delay

Goal: minimize playout delay while keeping missed-playout rate low

Observations:

- human speech can be deconstructed into talk spurts with intervening silent periods
- if playout buffer is empty, media player stalls playback and rebuffers

Approach: adaptive playout delay adjustment:

- estimate network delay, adjust playout delay at beginning of each talk spurt or buffer playback
- silent periods or rebuffering times compressed and elongated
- samples still played out without jitter during playback

Adaptive Playout Delay

Let:

t_i = timestamp of the i th packet

r_i = the time packet i is received by receiver

p_i = the time packet i is played at receiver

$\tau_i = r_i - t_i$ = network delay for i th packet

d_i = estimate of average network delay after receiving i th packet

Dynamic estimate of average delay at receiver:

$$d_i = (1 - u)d_{i-1} + u\tau_i$$

where u is a fixed constant (e.g., $u = .01$)

Adaptive Playout Delay

Also useful to estimate the average deviation of the delay, v_i :

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

The estimates d_i and v_i are calculated for every received packet, although they are only used to compute the playback time of the first packet in the playout buffer:

$$p_i = t_i + d_i + kv_i$$

where k is a positive constant, e.g., 4

Remaining packets in playout buffer are played out periodically, e.g., if the sample generation rate is Δt and p_0 is the first packet of the current playout buffer:

$$p_j = p_0 + j * \Delta t$$

UDP or TCP?

UDP

- server sends at rate appropriate for client (oblivious to network congestion!)
 - often send rate = encoding rate = constant rate
 - then, fill rate = constant rate - packet loss
- short playout delay (2-5 seconds) to compensate for network delay jitter
- ARQ: time permitting

TCP

- send at maximum possible rate under TCP
- fill rate fluctuates due to TCP congestion control
- larger playout delay: smooth TCP delivery rate
- HTTP/TCP passes more easily through firewalls (end up having to support both!)

Example: Internet Phone

Speaker's audio: alternating talk spurts, silent periods

- 64 kbps during talk spurt

Packets generated only during talk spurts

- 20 msec worth of samples at 8 Kbytes/sec: 160 bytes data

Application-layer header added to each packet

Samples+header encapsulated into UDP segment

Application sends UDP segment into socket every 20 msec during talkspurt

Start of Talkspurt

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 $\Rightarrow$ 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 $\Rightarrow$ talk spurt begins