Module – 5

Multimedia Networking Applications: Properties of video, properties of Audio, Types of multimedia Network Applications, Streaming stored video: UDP Streaming, HTTP Streaming, Adaptive streaming and DASH, content distribution Networks

Voice Over IP - Limitations of the Best Effort IP Service, Removing Jitter at the Receiver for Audio, Recovering from packet loss protocols for Real time conversational Applications, RTP, SIP

7.1.1 **Properties of Video**

The characteristic of video is its **high bit rate**.

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

Comparison of video with other Internet applications:

Consider three different users, each using a different Internet application.

- First user is going quickly through photos posted on his friends' Facebook

pages. User is looking at a new photo every 10 seconds, and that photos are on average 200 Kbytes in size.

- Second user is streaming music from the Internet to her smartphone. User is listening to many MP3 songs, one after the other, each encoded at a rate of 128 kbps.
- Third user is watching a video that has been encoded at 2 Mbps.

Suppose that the session length for all three users is 4,000 seconds.

Table below compares the bit rates and the total bytes transferred for these three users.

Video streaming consumes most bandwidth, having a bit rate of more than ten times greater than that of the Facebook and music-streaming applications.

	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

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, 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.

The two types of redundancy in video, both of which can be exploited by **video compression**.

Spatial redundancy is the redundancy within a given image. An image that consists of mostly white space has a high degree of redundancy and can be efficiently compressed without significantly sacrificing image quality.

Temporal redundancy reflects repetition from image to subsequent image. For example, an image and the subsequent image are exactly the same, reencoding the subsequent image is not required;

Higher is the bit rate, the better the image quality and the better the overall user viewing experience.

Compression technique can be used to create **multiple versions** of the same video, each at a different quality level.

For example, compression can be used to create, three versions of the same video, at rates of 300 kbps, 1 Mbps, and 3 Mbps.

Users can decide the download version they want to watch as a function of their current available bandwidth.

Users with high-speed Internet connections might choose the 3 Mbps version; users watching the video over 3G with a smartphone might choose the 300 kbps version.

7.1.2 **Properties of Audio**

Digital audio has significantly lower bandwidth requirements than video.

Following steps describes conversion of analog audio to a digital signal:

• The analog audio signal is sampled at some fixed rate, for example, at 8,000 samples per second. The value of each sample is an arbitrary real number.

• Each of the samples is then rounded to one of a finite number of values. This operation is referred to as **quantization**. The number of such finite values— called quantization values— is typically a power of two, for example, 256 quantization values.

• Each of the quantization values is represented by a fixed number of bits. For example, if there are 256 quantization values, then each value—and hence each audio sample—is represented by one byte.

• The bit representations of all the samples are then concatenated together to form the digital representation of the signal.

Example, if an analog audio signal is sampled at 8,000 samples per second and each sample is quantized and represented by 8 bits, then the resulting digital signal will have a rate of 64,000 bits per second.

The basic encoding technique that described above is called **pulse code modulation (PCM)**. Speech encoding often uses PCM, with a sampling rate of 8,000 samples per second and 8 bits per sample, resulting in a rate of 64 kbps.

MP3(MPEG 1Layer 3) encoders can compress to many different rates; 128 kbps is the most common encoding rate and produces very little sound degradation.

7.1.3 **Types of Multimedia Network Applications**

Multimedia applications are classified into three broad categories:

(i) streaming stored audio/video (ii) conversational voice/video-over-IP (iii) streaming live audio/video.

Streaming Stored Audio and Video

Streaming stored video combines video and audio components. Streaming stored audio is similar to streaming stored video, although the bit rates are typically much lower.

The underlying medium is pre recorded video, such as a movie, a television show, a prerecorded sporting event, or a pre recorded user generated video (such as YouTube).

These prerecorded videos are placed on servers, and users send requests to the servers to view the videos *on demand*.

Many Internet companies today provide streaming video, including YouTube (Google), Netflix, and Hulu.

Streaming stored video has three key features:

Streaming. In a streaming stored video application, the client begins video playout within a few seconds after it begins receiving the video from the server. This means that the client will be playing out from one location in the video while at the same time receiving later parts of the video from the server. This technique, known as **streaming**, avoids having to download the entire video file before playout begins.

• ***Interactivity***. As the media is prerecorded, the user may pause, reposition forward, reposition backward, fast-forward, and so on through the video content.

The time from when the user makes such a request until the action manifests itself at the client should be less than a few seconds for acceptable responsiveness.

• ***Continuous playout.*** Once playout of the video begins, it should proceed according to the original timing of the recording. Therefore, data must be received from the server in time for its playout at the client; otherwise, users experience video frame freezing or frame skipping.

An 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 the bit rate of the video itself.

Conversational Voice- and Video-over-IP

Real-time conversational voice over the Internet is referred as **Internet telephony.** It is also commonly called **Voice-over-IP (VoIP)**.

Conversational video is similar, except that it includes the video of the participants as well as their voices.

Eg. Skype, QQ, and Google Talk boasting hundreds of millions of daily users.

Two important constraints in this application are—**timing considerations and tolerance of data loss.**

Timing considerations are important because audio and video conversational applications are highly **delay-sensitive**.

For a conversation with two or more interacting speakers, the delay from when a user speaks or moves until the action is manifested at the other end should be less than a few hundred milliseconds.

For voice, delays smaller than 150 milliseconds are not perceived by a human listener, delays between 150 and 400 milliseconds can be acceptable, and delays exceeding 400 milliseconds can result in frustrating, if not completely unintelligible, voice conversations.

Conversational multimedia applications are **loss-tolerant**— occasional loss only causes occasional glitches in audio/video playback.

Streaming Live Audio and Video

It 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.

Live, broadcast-like applications often have many users who receive the same audio/video program at the same time. The distribution of live audio/video to many receivers can be efficiently accomplished using the IP multicasting techniques.

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.

Because the event is live, delay can also be an issue, although the timing constraints are much less stringent than those for conversational voice.

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.

7.2 **Streaming Stored Video**

Streaming 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 overcome the effects of varying end-to-end delays and varying amounts of available bandwidth between server and client.

For streaming video (both stored and live), users generally can tolerate a small several second

initial delay between when the client requests a video and when video playout begins at the client.

Consequently, when the video starts to arrive at the client, the client need not immediately begin playout, but can instead build up a reserve of video in an application buffer.

Once the client has built up a reserve of several seconds of buffered-but-not-yet-played video, the client can then begin video playout.

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.

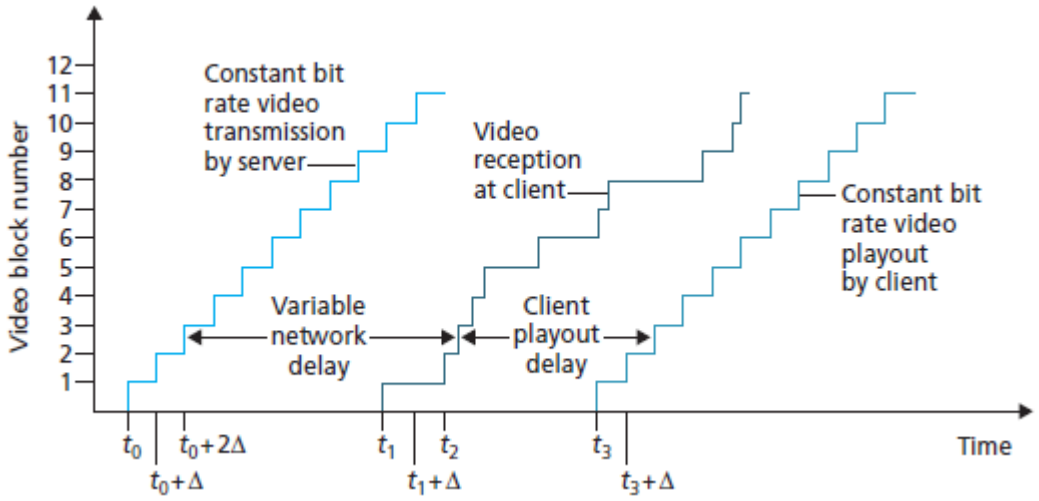


Figure above illustrates client-side buffering.

In this simple example, suppose that video is encoded at a fixed bit rate, and thus each video block contains video frames that are to be played out over the same fixed amount of time, Δ.

The server transmits the first video block at , the second block at t0 + Δ, the third block at t0 + 2Δ , and so on.

Once the client begins playout, each block should be played out Δ time units after the previous block in order to reproduce the timing of the original recorded video.

Because of the variable end-to-end network delays, different video blocks experience different delays.

The first video block arrives at the client at *t*1 and the second block arrives at t1+Δ.

The network delay for the *i*th block is the horizontal distance between the time the block was transmitted by the server and the time it is received at the client;

In this example, if the client were to begin playout as soon as the first block arrived at , then the second block would not have arrived in time to be played out at out at .

In this case, video playout would either have to stall (waiting for block 1 to arrive) or block 1 could be skipped—both resulting in undesirable playout impairments.

Instead, if the client were to delay the start of playout until , when blocks 1 through 6 have all arrived, periodic playout can proceed with *all* blocks having been received before their playout time.

7.2.1 **UDP Streaming**

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 bits)/(2 Mbps) = 4 msec.

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 within transport packets specially designed for transporting audio and video, using the Real-Time Transport Protocol (RTP)

Another distinguishing property of UDP streaming is the client and server also maintain, in parallel, a separate control connection over which the client sends commands regarding session state changes (such as pause, resume, reposition, and so on).

UDP streaming has 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.

For example, consider the scenario where 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.

- 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.

7.2.2 **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—specifically, it periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen.

Due to all of these advantages, most video streaming applications today—including YouTube and Netflix—use HTTP streaming (over TCP) as its underlying streaming protocol.

Prefetching Video

For streaming *stored* video, the client can attempt to download the video at a rate *higher* than the consumption rate, thereby **prefetching** video frames that are to be consumed in the future.

This prefetched video is naturally stored in the client application buffer.

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

Example. Suppose the video consumption rate is 1 Mbps but the network is capable of delivering the video from server to client at a constant rate of 1.5 Mbps.

Then the client will not only be able to play out the video with a very small playout delay, but will also be able to increase the amount of buffered video data by 500 Kbits every second.

In this manner, if in the future the client receives data at a rate of less than 1 Mbps for a brief period of time, the client will be able to continue to provide continuous playback due to the reserve in its buffer.

Client Application Buffer and TCP Buffers

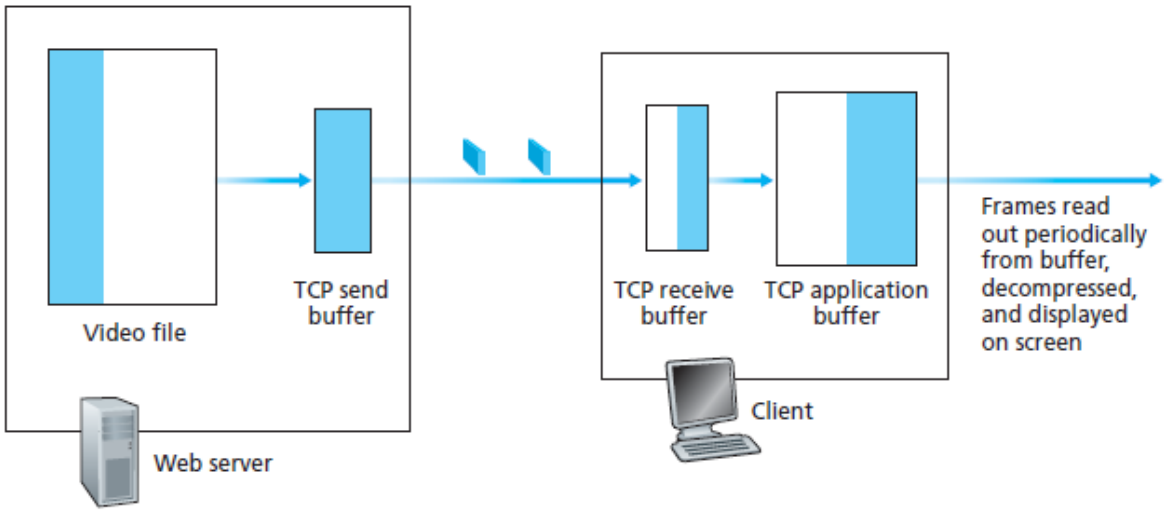


Figure 7.2 illustrates the interaction between client and server for HTTP streaming.

At the server side, the portion of the video file in white has already been sent into the server's socket, while the darkened portion is what remains to be sent.

After "passing through the socket door," the bytes are placed in the TCP send buffer before being transmitted into the Internet.

Because the TCP send buffer is shown to be full, the server is momentarily prevented from sending more bytes from the video file into the socket.

On the client side, the client application (media player) reads bytes from the TCP receive buffer (through its client socket) and places the bytes into the client application buffer.

At the same time, the client application periodically grabs video frames from the client application buffer, decompresses the frames, and displays them on the user's screen.

During the pause period, bits are not removed from the client application buffer, even though bits continue to enter the buffer from the server.

If the client application buffer is finite, it may eventually become full, which will cause "back pressure" all the way back to the server.

Specifically, once the client application buffer becomes full, bytes can no longer be removed from the client TCP receive buffer, so it too becomes full.

Once the client receive TCP buffer becomes full, bytes can no longer be removed from the client TCP send buffer, so it also becomes full.

Once the TCP send buffer becomes full, the server cannot send any more bytes into the socket.

Thus, if the user pauses the video, the server may be forced to stop transmitting, in which case the server will be blocked until the user resumes the video.

Even during regular playback (that is, without pausing), if the client application buffer becomes full, back pressure will cause the TCP buffers to become full, which will force the server to reduce its rate.

To determine the resulting rate, 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.

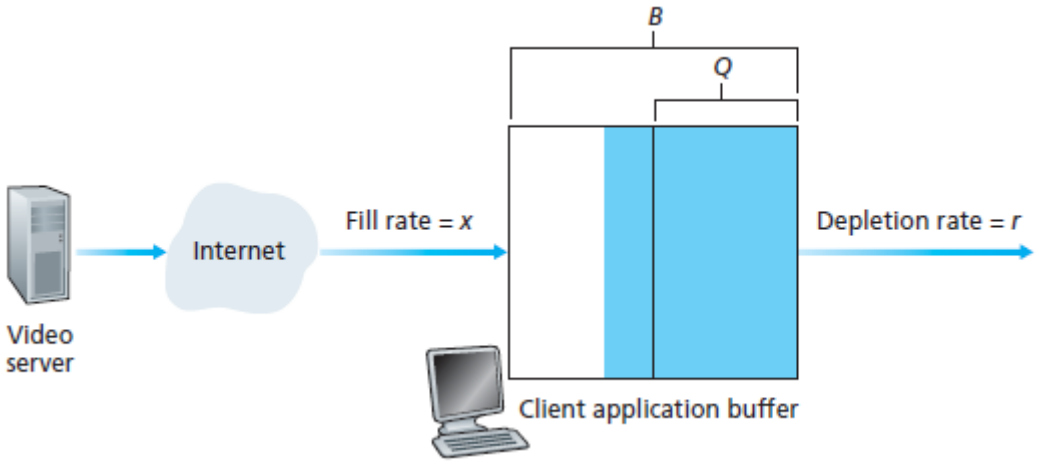
Therefore, *a full client application buffer indirectly imposes a limit on the rate that video can be sent from server to client when streaming over HTTP.*

Analysis of Video Streaming

As shown in Figure below, let *B* denote the size (in bits) of the client's application buffer, and let *Q* denote the number of bits that must be buffered before the client application begins playout. (i.e $Q < B$.)

Let *r* denote the video consumption rate—the rate at which the client draws bits out of the client application buffer during playback.

For example, if the video's frame rate is 30 frames/sec, and each (compressed) frame is 100,000 bits, then $r = 3$ Mbps.



Let's assume that the server sends bits at a constant rate *x* whenever the client buffer is not full.

Suppose at time $t = 0$, the application buffer is empty and video begins arriving to the client application buffer.

Bits arrive to the client application buffer at rate *x* and *no* bits are removed from this buffer before playout begins.

Thus, the amount of time required to build up *Q* bits (the initial buffering delay) is tp = Q / x.

If $x < r$ (that is, if the server send rate is less than the video consumption rate), then the client buffer will never become full!

Indeed, starting at time , the buffer will be depleted at rate *r* and will only be filled at rate

$x < r$.

Eventually the client buffer will empty out entirely, at which time the video will freeze on the screen while the client buffer waits another tp, seconds to build up *Q* bits of video.

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

Early Termination and Repositioning the Video

HTTP streaming systems often make use of the **HTTP byte-range header** in the HTTP GET request message, which specifies the specific range of bytes the client currently wants to retrieve from the desired video.

This is particularly useful when the user wants to reposition (that is, jump) to a future point in time in the video.

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.

During subject of repositioning, when a user repositions to a future point in the video or terminates the video early, some prefetched-but-not-yet-viewed data transmitted by the server will go unwatched—a 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 the bandwidth and server resources that were used to transmit those B bits have been completely wasted.

7.2.3 **Adaptive Streaming and DASH**

Drawback of HTTP Streaming is - 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.

In Dynamic Adaptive Streaming over HTTP (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 naturally selects chunks from a high-rate version; and when the available bandwidth is low, it naturally selects from a low-rate version.

The client selects different chunks one at a time with HTTP GET request messages

DASH allows clients with different Internet access rates to stream in video at different encoding rates.

Clients with low-speed 3G connections can receive a low bit-rate (and low-quality) version, and clients with fiber connections can receive a high-quality version.

DASH allows a client to adapt to the available bandwidth if the end-to-end bandwidth changes during the session.

This feature is particularly important for mobile users, who typically see their bandwidth availability fluctuate as they move with respect to the base stations.

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 various versions.

The client then selects one chunk at a time by specifying a URL and a byte range in an HTTP GET request message for each chunk.

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.

Since a sudden drop in bit rate by changing versions may result in visual quality degradation, the bit-rate reduction may be achieved using multiple intermediate versions to smoothly

transition to a rate where the client's consumption rate drops below its available receive bandwidth.

When the network conditions improve, the client can then later choose chunks from higher bit-rate versions.

By dynamically monitoring the available bandwidth and client buffer level, and adjusting the transmission rate with version switching, DASH can often achieve continuous playout at the best possible quality level without frame freezing or skipping.

Furthermore, since the client (rather than the server) maintains the intelligence to determine which chunk to send next, the scheme also improves server-side scalability.

Another benefit of this approach is that the client can use the HTTP byte-range request to precisely control the amount of prefetched video that it buffers locally.

Hence, the server not only stores many versions of the video but also separately stores many versions of the audio.

Each audio version has its own quality level and bit rate and has its own URL.

In these implementations, the client dynamically selects both video and audio chunks, and locally synchronizes audio and video playout.

7.2.4 **Content Distribution Networks**

For an Internet video company, providing streaming video service is to build a single massive data center, store all of its videos in the data center, and stream the videos directly from the data center to clients worldwide.

The three major problems with this approach.

1.If the client is far from the data center, server-to-client packets will cross many communication links and likely pass through many ISPs, with some of the ISPs possibly located on different continents.

If one of these links provides a throughput that is less than the video consumption rate, the end-to-end throughput will also be below the consumption rate, resulting in freezing delays for the user.

2.A popular video will be sent many times over the same communication links. This wastes network bandwidth, but the Internet video company will be paying its provider ISP (connected to the data center) for sending the *same* bytes into the Internet over and over again.

3.A single data center represents a single point of failure—if the data center or its links to the Internet goes down, it would not be able to distribute *any* video streams.

In order to meet the challenge of distributing massive amounts of video data to users distributed around the world, almost all major video-streaming companies make use of **Content Distribution Networks (CDNs)**.

A CDN manages servers in multiple geographically distributed locations, stores copies of the videos (and other types of Web content, including documents, images, and audio) in its servers and attempts to direct each user request to a CDN location that will provide the best user experience.

The CDN may be a **private CDN**, that is, owned by the content provider itself;

For example, Google's CDN distributes YouTube videos and other types of content.

The CDN may alternatively be a **third-party CDN** that distributes content on behalf of multiple content providers;

CDN Operation

When a browser in a user's host is instructed to retrieve a specific video (identified by a URL), the CDN must intercept the request so that it can :

(1) determine a suitable CDN server cluster for that client at that time.

(2) redirect the client's request to a server in that cluster.

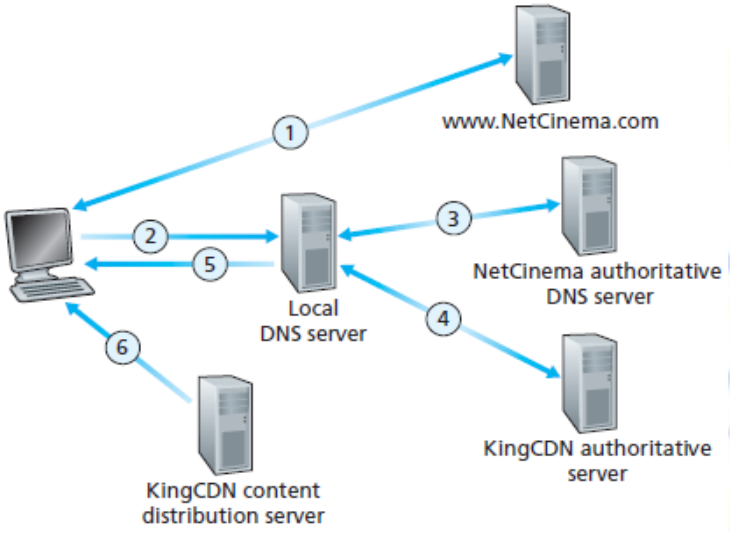
consider a example to illustrate how DNS is typically involved.

Suppose a content provider, NetCinema, employs the third-party CDN company, KingCDN, to distribute its videos to its customers.

On the NetCinema Web pages, each of its videos is assigned a URL that includes the string "video" and a unique identifier for the video itself;

For example, Transformers 7 might be assigned http://video.netcinema.com/6Y7B23V.

Six steps then occur, as shown in Figure 7.4:



1. The user visits the Web page at NetCinema.

2. When the user clicks on the link http://video.netcinema.com/6Y7B23V, the user's host sends a DNS query for video.netcinema.com.

3. The user's Local DNS Server (LDNS) relays the DNS query to an authoritative DNS server for NetCinema, which observes the string "video" in the hostname video.netcinema.com.

To "hand over" the DNS query to KingCDN, instead of returning an IP address, the NetCinema authoritative DNS server returns to the LDNS a hostname in the KingCDN's domain, for example, a1105.kingcdn.com.

4. From this point on, the DNS query enters into KingCDN's private DNS infrastructure. The user's LDNS then sends a second query, now for a1105.kingcdn.com, and KingCDN's DNS system eventually returns the IP addresses of a KingCDN content server to the LDNS. It is thus here, within the KingCDN's DNS system, that the CDN server from which the client will receive its content is specified.

5. The LDNS forwards the IP address of the content-serving CDN node to the user's host.

6. Once the client receives the IP address for a KingCDN content server, it establishes a direct TCP connection with the server at that IP address and issues an HTTP GET request for the video. If DASH is used, the server will first send to the client a manifest file with a list of URLs, one for each version of the video, and the client will dynamically select chunks from the different versions.

Cluster Selection Strategies

At the core of any CDN deployment is a **cluster selection strategy**, that is, a mechanism for dynamically directing clients to a server cluster or a data center within the CDN.

CDN learns the IP address of the client's LDNS server via the client's DNS lookup.

After learning this IP address, the CDN needs to select an appropriate cluster based on this IP address.

CDNs generally employ proprietary cluster selection strategies.

One simple strategy is to assign the client to the cluster that is **geographically closest**.

When a DNS request is received from a particular LDNS, the CDN chooses the geographically closest cluster, that is, the cluster that is the fewest kilometres from the LDNS "as the bird flies."

In order to determine the best cluster for a client based on the *current* traffic conditions, CDNs can instead perform periodic **real-time measurements** of delay and loss performance between their clusters and clients.

For instance, a CDN can have each of its clusters periodically send probes (for example, ping messages or DNS queries) to all of the LDNSs around the world.

One drawback of this approach is that many LDNSs are configured to not respond to such probes.

An alternative to sending extraneous traffic for measuring path properties is to 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.

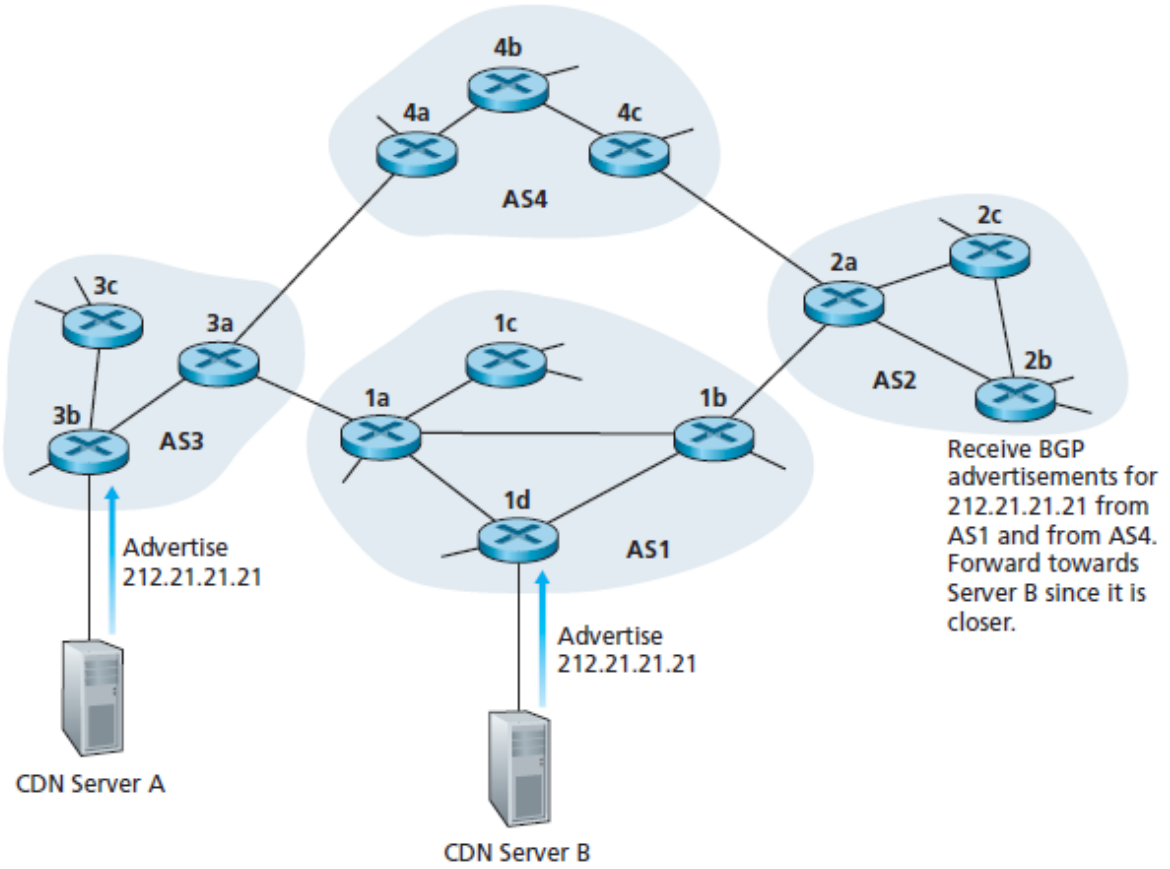
Such solutions, require redirecting clients to (possibly) suboptimal clusters from time to time in order to measure the properties of paths to these clusters.

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 (within Step 4 in previous Figure), 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.

In this scheme, the DNS servers continue to return the optimal cluster for the client, so that delivery of videos and other Web objects does not suffer.

A very different approach to matching clients with CDN servers is to use **IP anycast**. The idea behind IP anycast is to have the routers in the Internet route the client's packets to the "closest" cluster, as determined by BGP.



Specifically, as shown in Figure above, during the IP-anycast configuration stage, the CDN company assigns the *same* IP address to each of its clusters, and *uses standard BGP* to advertise this IP address from each of the different cluster locations.

When a BGP router receives multiple route advertisements for this same IP address, it treats these advertisements as providing different paths to the same physical location.

Following standard operating procedures, the BGP router will then pick the "best" (for example, closest, as determined by AS-hop counts) route to the IP address according to its local route selection mechanism.

For example, if one BGP route (corresponding to one location) is only one AS hop away from the router, and all other BGP routes (corresponding to other locations) are two or more AS hops away, then the BGP router would typically choose to route packets to the location that needs to traverse only one AS .

After this initial configuration phase, the CDN can do its main job of distributing content. When any client wants to see any video, the CDN's DNS returns the anycast address, no matter where the client is located.

When the client sends a packet to that IP address, the packet is routed to the "closest" cluster as determined by the preconfigured forwarding tables, which were configured with BGP as just described.

This approach has the advantage of finding the cluster that is closest to the client rather than the cluster that is closest to the client's LDNS.

VOICE – over – IP (VOIP) :

Real-time conversational voice over the Internet is often referred to as Internet telephony, since, from the user's perspective, it is similar to the traditional circuit-switched telephone service. It is called Voice-over-IP(VoIP).

Limitations of the Best – Effort IP Service :

Consider an example: The sender generates bytes at a rate of 8,000 bytes per second; every 20 msecs the sender gathers these bytes into a chunk. A chunk and a special header (discussed below) are encapsulated in a UDP segment, via a call to the socket interface. Thus,

the number of bytes in a chunk is (20 msecs)·(8,000 bytes/sec) = 160 bytes, and a UDP segment is sent every 20 msecs.

If each packet makes it to the receiver with a constant end-to-end delay, then packets arrive at the receiver periodically every 20 msecs. The receiver can simply play back each chunk as soon as it arrives. But unfortunately, some packets can be lost and most packets will not have the same end-to-end delay, even in a lightly congested Internet. For this reason, the receiver must take more care in determining (1) when to play back a chunk, and (2) what to do with a missing chunk.

Packet Loss

Consider one of the UDP segments generated by our VoIP application.

- The UDP segment is encapsulated in an IP datagram.
- As the datagram wanders through the network, it passes through router buffers (that is, queues) while waiting for transmission on outbound links.
- One or more of the buffers in the path from sender to receiver might be full, in which case the arriving IP datagram may be discarded, never to arrive at the receiving application.
- Loss could be eliminated by sending the packets over TCP (which provides for reliable data transfer) rather than over UDP.
- Retransmission mechanisms are often considered unacceptable for conversational real-time audio applications such as VoIP, because they increase end-to-end delay.

Furthermore, due to TCP congestion control, packet loss may result in a reduction of the TCP sender's transmission rate to a rate that is lower than the receiver's drain rate, possibly leading to buffer starvation. This can have a severe impact on voice intelligibility at the receiver. For these reasons, most existing VoIP applications run over UDP by default.

Packet loss rates between 1 and 20 percent can be tolerated, depending on how voice is encoded and transmitted, and on how the loss is concealed at the receiver.

For example, forward error correction (FEC) can help conceal packet loss.

If one or more of the links between sender and receiver is severely congested, and packet loss exceeds 10 to 20 percent (for example, on a wireless link), then there is really nothing that can be done to achieve acceptable audio quality.

End-to-End Delay

End-to-end delay is the accumulation of transmission, processing, and queuing delays in routers, propagation delays in links and end-system processing delays.

- For real-time conversational applications, such as VoIP, end-to-end delays smaller than 150 msecs are not perceived by a human listener;
- Delays between 150 and 400 msecs can be acceptable but are not ideal;
- Delays exceeding 400 msecs can seriously hinder the interactivity in voice conversations.
- The receiving side of a VoIP application will typically disregard any packets that are delayed more than a certain threshold, for example, more than 400 msecs. Thus, packets that are delayed by more than the threshold are effectively lost.

Packet Jitter

The time from when a packet is generated at the source until it is received at the receiver can fluctuate from packet to packet is called jitter.

Example: Consider two consecutive packets in our VoIP application. The sender sends the second packet 20 msecs after sending the first packet. But at the receiver, the spacing between these packets can become greater than 20 msecs.

Eg: Suppose the first packet arrives at a nearly empty queue at a router, but just before the second packet arrives at the queue a large number of packets from other sources arrive at the same queue. Because the first packet experiences a small queuing delay and the second packet suffers a large queuing delay at this router, the first and second packets become spaced by more than 20 msecs.

The spacing between consecutive packets can also become less than 20 msecs.

Eg: Again consider two consecutive packets. Suppose the first packet joins the end of a queue with a large number of packets, and the second packet arrives at the queue before this first

packet is transmitted and before any packets from other sources arrive at the queue. In this case, two packets find themselves one right after the other in the queue. If the time it takes to transmit a packet on the router's outbound link is less than 20 msecs, then the spacing between first and second packets becomes less than 20 msecs.

If the receiver ignores the presence of jitter and plays out chunks as soon as they arrive, then the resulting audio quality can easily become unintelligible at the receiver.

Jitter can be removed by using sequence numbers, timestamps and a playout delay.

Removing Jitter at the Receiver for Audio

For VoIP application, where packets are being generated periodically, the receiver should attempt to provide periodic playout of voice chunks in the presence of random network jitter. This is typically done by combining the following two mechanisms :•

- **Prepending each chunk with a timestamp**. The sender stamps each chunk with the time at which the chunk was generated.
- **Delaying playout of chunks at the receiver**. As in the Figure below, the playout delay of the received audio chunks must be long enough so that most of the packets are received before their scheduled playout times. This playout delay can either be fixed throughout the duration of the audio session or vary adaptively during the audio session life time.

Two playback strategies:

- Fixed play-out delay
- Adaptive playout delay
- Fixed Playout Delay

Fixed Playout Delay: Here The receiver attempts to play out each chunk exactly q msecs after the chunk is generated. So if a chunk is timestamped at the sender at time t, the receiver plays out the chunk at time t+q, assuming the chunk has arrived by that time. Packets that arrive after their scheduled playout times are discarded and considered lost.

VoIP can support delays up to about 400 msecs, although a more satisfying conversational experience is achieved with smaller values of q.

If q is made much smaller than 400 msecs, then many packets may miss their scheduled playback times due to the network-induced packet jitter.

If large variations in end-to-end delay are typical, it is preferable to use a large q; on the other hand, if delay is small and variations in delay are also small, it is preferable to use a small q, less than 150 msecs.

The trade-off between the playback delay and packet loss is illustrated in Figure. The figure below shows the times at which packets are generated and played out for a single talk spurt.

Two distinct initial playout delays are considered.

- As shown by the leftmost staircase, the sender generates packets at regular intervals—say, every 20 msecs.
- The first packet in this talk spurt is received at time r. As shown in the figure, the arrivals of subsequent packets are not evenly spaced due to the network jitter.
- For the first playout schedule, the fixed initial playout delay is set to p–r. With this schedule, the fourth packet does not arrive by its scheduled playout time, and the receiver considers it lost.
- For the second playout schedule, the fixed initial play-out delay is set to p –r. For this schedule, all packets arrive before their scheduled playout times, and there is therefore no loss.

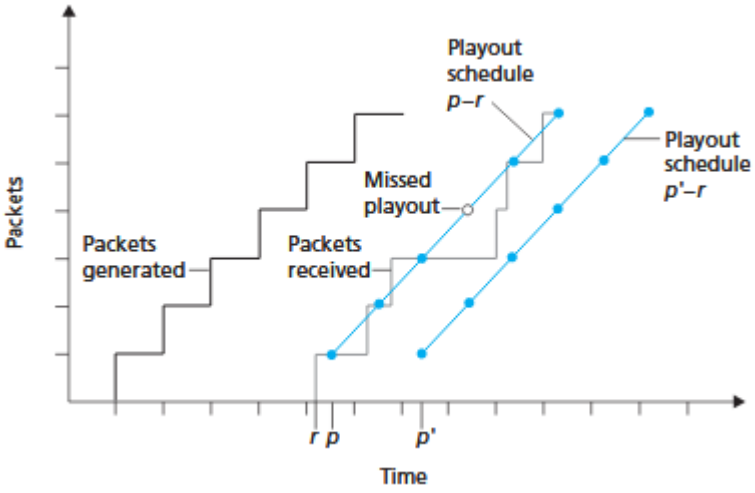


Fig: Packet loss for different fixed playout delays

Adaptive Playout Delay

By making the initial playout delay large, most packets will make their deadlines and there will therefore be negligible loss;

For conversational services such as VoIP, long delays can become bothersome if not intolerable. Ideally, the playout delay needs to be minimized such that the loss has to be below a few percent.

The natural way to deal with this trade-off is to estimate the network delay and the variance of the network delay and to adjust the playout delay accordingly at the beginning of each talk spurt.

This adaptive adjustment of playout delays at the beginning of the talk spurts will cause the sender's silent periods to be compressed and elongated; Compression and elongation of silence by a small amount is not noticeable in speech.

A generic algorithm is described such that the receiver can be used to adaptively adjust its playout delays.

Let ti= the timestamp of the ith packet = the time the packet was generated by the sender.

ri= the time packet i is received by receiver

pi= the time packet i is played at receiver

The end-to-end network delay of the ith packet is 'ri–ti'. Due to network jitter, this delay will vary from packet to packet.

Let di denote an estimate of the average network delay upon reception of the ith packet. This estimate is constructed from the timestamps as follows:

$di= (1 – u)di–1+u(ri–ti)$

where u is a fixed constant (for example, u= 0.01).

Thus di is a smoothed average of the observed network delays r1–t1, . . . , ri–ti.

The estimate places more weight on the recently observed network delays than on the observed network delays of the distant past.

Let vi denote an estimate of the average deviation of the delay from the estimated average delay.

This estimate is also constructed from the timestamps: $vi= (1 – u) vi–1+u |ri–ti–di|$

The estimates di and vi are calculated for every packet received, although they are used only to determine the playout point for the first packet in any talk spurt.

Once having calculated these estimates, the receiver employs the following algorithm for the playout of packets. If packet i is the first packet of a talk spurt, its playout time, pi, is computed as: $pi=ti+di+Kvi$

Where K is a positive constant (for example, K= 4).

The purpose of the Kvi term is to set the playout time far enough into the future so that only a small fraction of the arriving packets in the talk spurt will be lost due to late arrivals. The playout point for any subsequent packet in a talk spurt is computed as an offset from the point in time when the first packet in the talk spurt was played out.

In particular, let $qi=pi–ti$ be the length of time from when the first packet in the talk spurt is generated until itis played out. If packet j also belongs to this talk spurt, it is played out at time $pj=tj+qi$.

Recovering from Packet Loss

Schemes that attempt to preserve acceptable audio quality in the presence of packet loss are called loss recovery schemes.

A packet is lost either if it never arrives at the receiver or if it arrives after its scheduled playout time.

Retransmitting lost packets may not be feasible in a real-time conversational application such as VoIP. Retransmitting a packet that overflowed a router queue cannot normally be accomplished quickly enough. Because of these considerations, VoIP applications often use some type of loss anticipation scheme.

Two types of loss anticipation schemes are forward error correction (FEC)and interleaving.

Forward Error Correction (FEC)

The FEC is adding redundant information to the original packet stream.

For the cost of marginally increasing the transmission rate, the redundant information can be used to reconstruct approximations or exact versions of some of the lost packets.

The two simple FEC mechanisms are:

- The first mechanism sends a redundant encoded chunk after every n chunks. The redundant chunk is obtained by **exclusive OR-ing** the n original chunks .
    - If any one packet of the group of n+ 1 packets is lost, the receiver can fully reconstruct the lost packet.
    - 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.
    - The smaller the group size, the greater the relative increase of the transmission rate.
    - In particular, the transmission rate increases by a factor of 1/n, so that, if n= 3, then the transmission rate increases by 33 percent.

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.

- The second FEC mechanism is to send a lower-resolution audio stream as the redundant information.
  - The sender might create a nominal audio stream and a corresponding low-resolution, low-bit rate audio stream. (The nominal stream could be a PCM encoding at 64 kbps, and the lower-quality stream could be a GSM encoding at 13 kbps.)
  - The low-bit rate stream is referred to as the redundant stream. As shown in Figure, the sender constructs the nth packet by taking the nth chunk from the nominal stream and appending to it the (n– 1)st chunk from the redundant stream.
  - Whenever there is non consecutive packet loss, the receiver can conceal the loss by playing out the low-bit rate encoded chunk that arrives with the subsequent packet.
  - Low-bit rate chunks give lower quality than the nominal chunks. However, a stream of mostly high-quality chunks, occasional low-quality chunks, and no missing chunks gives good overall audio quality.
  - The receiver only has to receive two packets before playback, so that the increased playout delay is small.
  - If the low-bit rate encoding is much less than the nominal encoding, then the marginal increase in the transmission rate will be small.
  - In order to cope with consecutive loss, Instead of appending just the (n– 1)st low-bit rate chunk to the nth nominal chunk, the sender can append the (n– 1)st and (n– 2)nd low-bit rate chunk, or append the (n– 1)stand (n– 3)rd low-bit rate chunk, and so on.
  - 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.

Interleaving

As an alternative to redundant transmission, a VoIP application can send interleaved audio. As shown in Figure, the sender resequences units of audio data before transmission, so that originally adjacent units are separated by a certain distance in the transmitted stream.

Interleaving can mitigate the effect of packet losses. If, for example, units are 5 msecs in length and chunks are 20 msecs (that is, four units per chunk), then the first chunk could contain units 1, 5, 9, and 13;

The second chunk could contain units 2, 6, 10, and 14; and so on. Figure below shows that the loss of a single packet from an interleaved stream results in multiple small gaps in the reconstructed stream, as opposed to the single large gap that would occur in a non-interleaved stream.

Interleaving can significantly improve the perceived quality of an audio stream. It also has low overhead. The 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 inter-leaving is that it does not increase the bandwidth requirements of a stream.

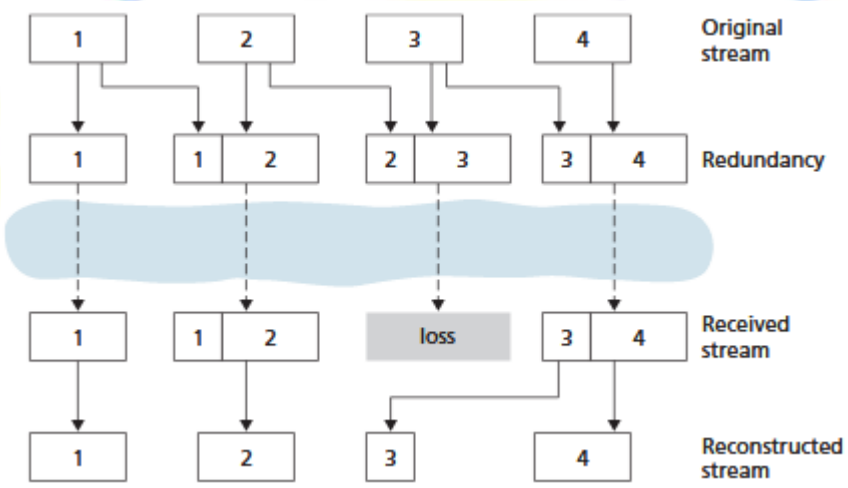


Fig: Piggybacking lower-quality redundant information

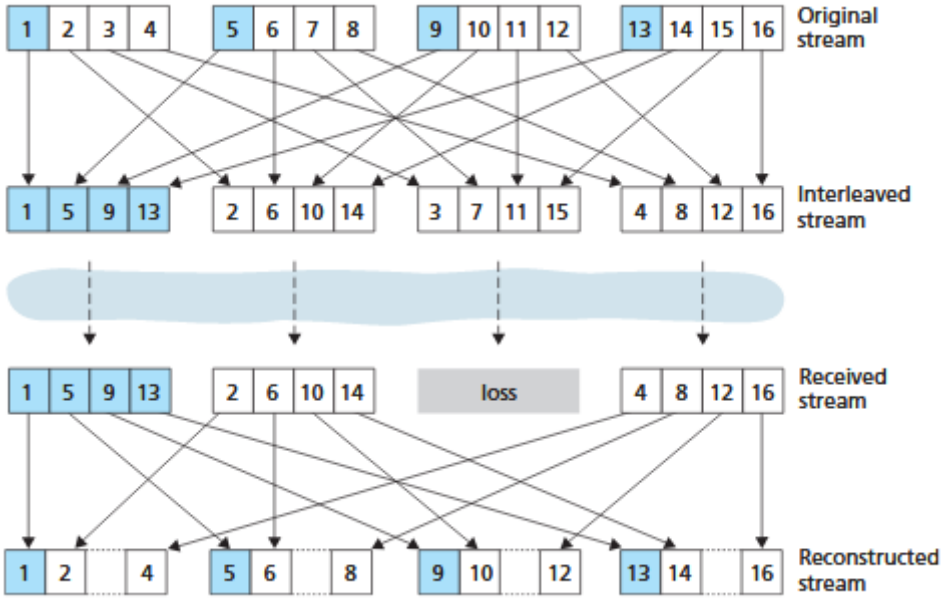


Fig: Sending interleaved audio

Error Concealment

Error concealment schemes attempt to produce a replacement for a lost packet that is similar to the original. Audio signals and in particular speech, exhibits large amounts of short-term self-similarity.

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.

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

Protocols for Real Time Conversational Applications

RTP (Real Time Transport Protocol):

RTP Basics

RTP typically runs on top of UDP. The sending side encapsulates a media chunk within an RTP packet, then encapsulates the packet in a UDP segment, and then hands the segment to IP.

The receiving side extracts the RTP packet from the UDP segment, then extracts the media chunk from the RTP packet, and then passes the chunk to the media player for decoding and rendering.

As an example, consider the use of RTP to transport voice. Suppose the voice source is PCM-encoded (that is, sampled, quantized, and digitized) at 64 kbps. Further suppose that the application collects the encoded data in 20-msec chunks, that is,160 bytes in a chunk.

The sending side precedes each chunk of the audio data with an RTP header that includes the type of audio encoding, a sequence number, and a timestamp.

The RTP header is normally 12 bytes. The audio chunk along with the RTP header form the RTP packet. The RTP packet is then sent into the UDP socket interface.

At the receiver side, the application receives the RTP packet from its socket interface.

The application extracts the audio chunk from the RTP packet and uses the header fields of the RTP packet to properly decode and play back the audio chunk.

RTP does not provide any mechanism to ensure timely delivery of data or provide other quality-of-service (QoS) guarantees;

It does not even guarantee delivery of packets or prevent out-of-order delivery of packets. RTP encapsulation is seen only at the end systems. Routers do not distinguish between IP datagrams that carry RTP packets and IP datagrams that don't.

RTP allows each source (for example, a camera or a microphone) to be assign edits own independent RTP stream of packets. For example, for a video conference between two

participants, four RTP streams could be opened—two streams for transmitting the audio (one in each direction) and two streams for transmitting the video (again, one in each direction).

Popular encoding techniques—including MPEG 1 and MPEG 2—bundle the audio and video into a single stream during the encoding process. When the audio and video are bundled by the encoder, then only one RTP stream is generated in each direction.

RTP packets are not limited to unicast applications. They can also be sent over one-to-many and many-to-many multicast trees. For a many-to-many multicast session, all of the session's senders and sources typically use the same multicast group for sending their RTP streams.

RTP multicast streams belonging together, such as audio and video streams emanating from multiple senders in a video conference application, belong to an RTP session.

RTP Packet Header Fields As shown in Figure below, the four main RTP packet header fields are the payload type, sequence number, timestamp, and source identifier fields.

The payload type field in the RTP packet is 7 bits long.

For an audio stream, the payload type field is used to indicate the type of audio encoding (for example, PCM, adaptive delta modulation, linear predictive encoding) that is being used.

If a sender decides to change the encoding in the middle of a session, the sender can inform the receiver of the change through this payload type field.

The sender may want to change the encoding in order to increase the audio quality or to decrease the RTP stream bitrate.

Table below lists some of the audio payload types currently supported by RTP.

For a video stream, the payload type is used to indicate the type of video encoding(for example, motion JPEG, MPEG 1, MPEG 2, H.261).

Again, the sender can change video encoding on the fly during a session. Table below lists some of the video payload types currently supported by RTP.

The other important fields are the following:•

Sequence number field.

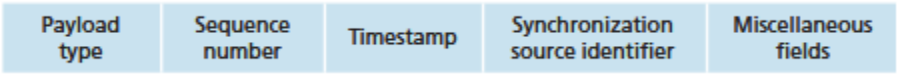


Fig: RTP header fields

Payload-Type Number	Audio Format	Sampling Rate	Rate
0	PCM μ-law	8 kHz	64 kbps
1	1016	8 kHz	4.8 kbps
3	GSM	8 kHz	13 kbps
7	LPC	8 kHz	2.4 kbps
9	G.722	16 kHz	48–64 kbps
14	MPEG Audio	90 kHz	—
15	G.728	8 kHz	16 kbps

Fig: Audio payload types supported by RTP

Payload-Type Number	Video Format
26	Motion JPEG
31	H.261
32	MPEG 1 video
33	MPEG 2 video

Fig : Some video payload types supported by RTP

The sequence number field is 16 bits long. The sequence number increments by one for each RTP packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.

Timestamp field.

The timestamp field is 32 bits long. It reflects the sampling instant of the first byte in the RTP data packet. The receiver can use timestamps to remove packet jitter introduced in the network and to provide synchronous playout at the receiver. The time-stamp is derived from a sampling clock at the sender.

Synchronization source identifier (SSRC)

The SSRC field is 32 bits long. It identifies the source of the RTP stream. Typically, each stream in an RTP session has a distinct SSRC. The SSRC is not the IP address of the sender, but instead is a number that the source assigns randomly when the new stream is started. The probability that two streams get assigned the same SSRC is very small.

SIP The Session Initiation Protocol (SIP)

SIP is an open and lightweight protocol that does the following:

It provides mechanisms for establishing calls between a caller and a callee overan IP network.

It allows the caller to notify the callee that it wants to start a call.

It allows the participants to agree on media encodings.

It also allows participants to end calls.

It provides mechanisms for the caller to determine the current IP address of the callee.

Users do not have a single, fixed IP address because they may be assigned addresses dynamically (using DHCP) and because they may have multiple IPdevices, each with a different IP address.

It provides mechanisms for call management, such as adding new media streams during the call, changing the encoding during the call, inviting new participants during the call, call transfer, and call holding.

Setting Up a Call to a Known IP Address

Example, Alice is at her PC and she wants to call Bob, who is also working at his PC.

Alice's and Bob's PCs are both equipped with SIP-based software for making and receiving phone calls.

In this example, we assume that Alice knows the IP address of Bob's PC. Refer Figure below illustrates the SIP call-establishment process.

SIP session begins when Alice sends Bob an INVITE message, which resembles an HTTP request message. This INVITE message is sent over UDP to the well-known port 5060 for SIP. (SIP messages can also be sent over TCP.)

The INVITE message includes an identifier for Bob(bob@193.64.210.89), an indication of Alice's current IP address, an indication that Alice desires to receive audio, which is to be encoded in format AVP 0 (PCM encoded-law) and encapsulated in RTP, and an indication that she wants to receive the RTP packets on port 38060.

After receiving Alice's INVITE message, Bob sends an SIP response message, which resembles an HTTP response message. This response SIP message is also sent to the SIP port 5060.

Bob's response includes a 200 OK as well as an indication of his IP address, his desired encoding and packetization for reception, and his port number to which the audio packets should be sent.

In the above example Alice and Bob are going to use different audio-encoding mechanisms: Alice is asked to encode her audio with GSM whereas Bob is asked to encode his audio with PCM -law. After receiving Bob's response, Alice sends Bob an SIP acknowledgment message. After this SIP transaction, Bob and Alice can talk.

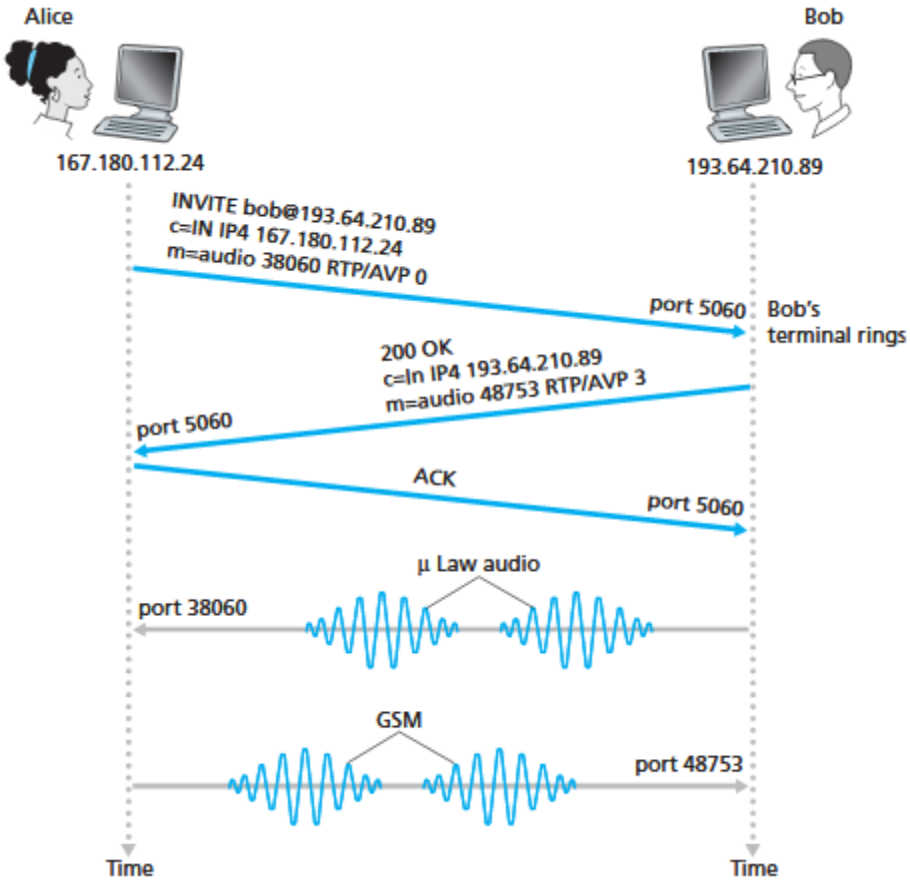


Fig: SIP call establishment when Alice knows Bob's IP address

Bob will encode and packetize the audio as requested and send the audio packets to port number 38060 at IP address167.180.112.24.

Alice will also encode and packetize the audio as requested and send the audio packets to port number 48753 at IP address 193.64.210.89

Key characteristics of SIP are:

First, SIP is an out-of-band protocol: The SIP messages are sent and received in sockets that are different from those used for sending and receiving the media data.

Second, the SIP messages themselves are ASCII-readable and resemble HTTP messages. Third, SIP requires all messages to be acknowledged, so it can run over UDP or TCP.

Assume, if Bob does not have a PCM-law codec for encoding audio, Then instead of responding with 200OK, Bob would likely respond with a 600 Not Acceptable and list in the message all the codecs he can use.

Alice would then choose one of the listed codecs and send another INVITE message, this time advertising the chosen codec.

Bob could also simply reject the call by sending one of many possible rejection reply codes.(There are many such codes, including “busy,” “gone,” “payment required,” and “forbidden.”)

SIP Addresses In the previous example, Bob’s SIP address is sip:bob@193.64.210.89. Forexample, Bob’s address might be sip:bob@domain.com. When Alice’s SIP device sends an INVITE message, the message would include this e-mail-like address;the SIP infrastructure would then route the message to the IP device that Bob is currently using.

Other possible forms for the SIP address could be Bob’s legacy phone number or simply Bob’s first/middle/last name (assuming it is unique).

SIP addresses can be included in Webpages, just as people’s e-mail addresses are included in Web pages with the mailtoURL.

For example, suppose Bob has a personal homepage, and he wants to pro-vide a means for visitors to the homepage to call him. URL could be sip:bob@domain.com.

When the visitor clicks on the URL, the SIP application in the visitor’s device is launched and an INVITE message is sent to Bob.

SIP Messages

Suppose that Alice wants to initiate a VoIP call to Bob, and this time Alice knows only Bob’s SIP address, bob@domain.com, and does not know the IP address of the device that Bob is currently using. Then her message might look something like this:

INVITE sip:bob@domain.com SIP/2.0

Via: SIP/2.0/UDP 167.180.112.24

From: sip:alice@hereway.com

To: sip:bob@domain.com

Call-ID: a2e3a@pigeon.hereway.com

Content-Type: application/sdp

Content-Length: 885c=IN IP4 167.180.112.24m=audio 38060 RTP/AVP 0

The INVITE line includes the SIP version, as does an HTTP request message.

Whenever an SIP message passes through an SIP device (including the device that orig-inates the message), it attaches a Via header, which indicates the IP address of the device.

Similar to an e-mail message, the SIP message includes a From header line and a To header line. The message includes a Call-ID, which uniquely identifies the call (similar to the message-ID in e-mail).

It includes a Content-Type header line, which defines the format used to describe the content contained in the SIP message. It also includes a Content-Length header line, which provides the length in bytes of the content in the message.

Finally, after a carriage return and line feed, the message contains the content. In this case, the content provides information about Alice's IP address and how Alice wants to receive the audio.

Name Translation and User Location

In the above example in Figure, we assumed that Alice's SIP device knew the IP address where Bob could be contacted. But this assumption is quite unrealistic, not only because IP addresses are often dynamically assigned with DHCP, but also because Bob may have multiple IP devices (for example, different devices for hishome, work, and car).

Suppose that Alice knows only Bob's e-mail address, bob@domain.com, and that this same address is used for SIP-based calls. In this case, Alice needs to obtain the IP address of the device that the userbob@domain.com is currently using.

Alice creates an INVITE message that begins with INVITE bob@domain.com SIP/2.0 and sends this message to an SIP proxy.

The proxy will respond with an SIP reply that might include the IP address of the device that bob@domain.com is currently using.

Alternatively, the reply might include the IP address of Bob's voicemail box, or it might include a URL of a Web page (that says "Bob is sleeping. Leave me alone!").

SIP registrar: Every SIP user has an associated registrar.

Whenever a user launches an SIP application on a device, the application sends an SIP register message to the registrar, informing the registrar of its current IP address.

For example, when Bob launches his SIP application on his PDA, the application would send a message along the lines of:

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

Bob's registrar keeps track of Bob's current IP address.

Whenever Bob switches to a new SIP device, the new device sends a new register message, indicating the new IP address.

Also, if Bob remains at the same device for an extended period of time, the device will send refresh register messages, indicating that the most recently sent IP address is still valid.

The registrar is analogous to a DNS authoritative name server: The DNS server translates fixed host names to fixed IP addresses;

The SIP registrar translates fixed human identifiers (for example, bob@domain.com) to dynamic IP addresses.

Often SIP registrars and SIP proxies are run on the same host.

Alice's SIP proxy server obtains Bob's current IPaddress by forwarding Alice's INVITE message to Bob's registrar/proxy.

The registrar/proxy could then forward the message to Bob's current SIP device.

Finally, Bob, having now received Alice's INVITE message, could send an SIP response to Alice.

Consider Figure below, jim@umass.edu, currently working on 217.123.56.89, wants to initiate a Voice-over-IP (VoIP) session with keith@upenn.edu, currently working on 197.87.54.21.

The following steps are taken:

(1) Jim sends an INVITE message to the umass SIP proxy.

(2) The proxy does a DNS lookup on the SIP registrar upenn.edu (not shown in diagram) and then forwards the message to the registrar server.

(3) Because keith@upenn.edu is no longer registered at the upenn registrar, the upenn registrar sends a redirect response, indicating that it should try keith@eurecom.fr.

(4) The umass proxy sends an INVITE message to the eurecom SIP registrar.

(5) The eurecom registrar knows the IP address of keith@eurecom.fr and forwards the INVITE message to the host197.87.54.21, which is running Keith's SIP client.

(6–8) An SIP response is sent back through registrars/proxies to the SIP client on 217.123.56.89.

(9) Media is sent directly between the two clients. (There is also an SIP acknowledgment message, which is not shown.)s

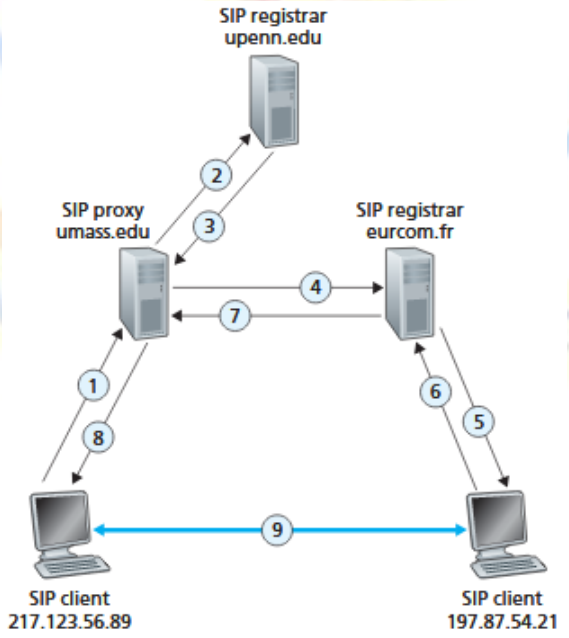


Fig : Session initiation, involving SIP proxies and registrars