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, case studies: You Tube.

Network Support for Multimedia: Quality-of- Service (QoS) Guarantees: Resource Reservation and Call Admission.

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.

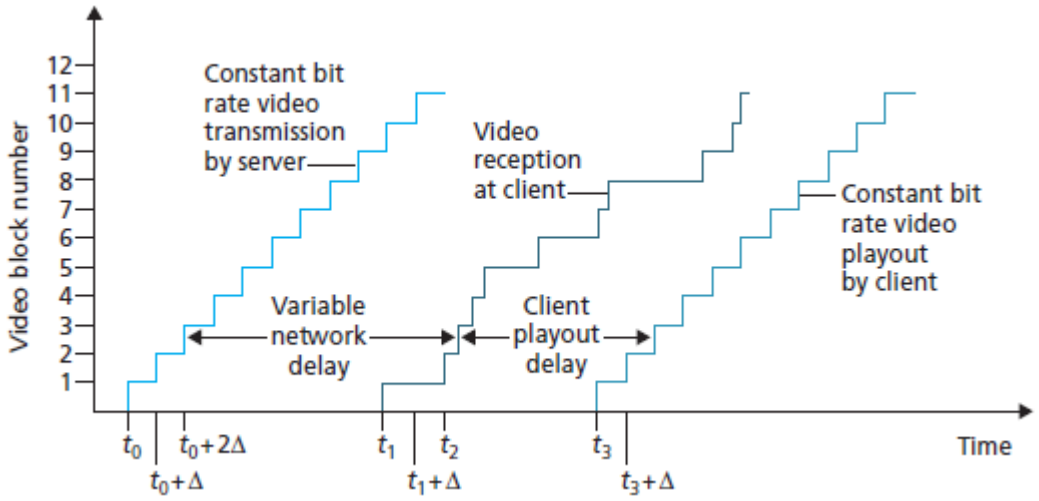


Figure above illustrates client-side buffering.

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

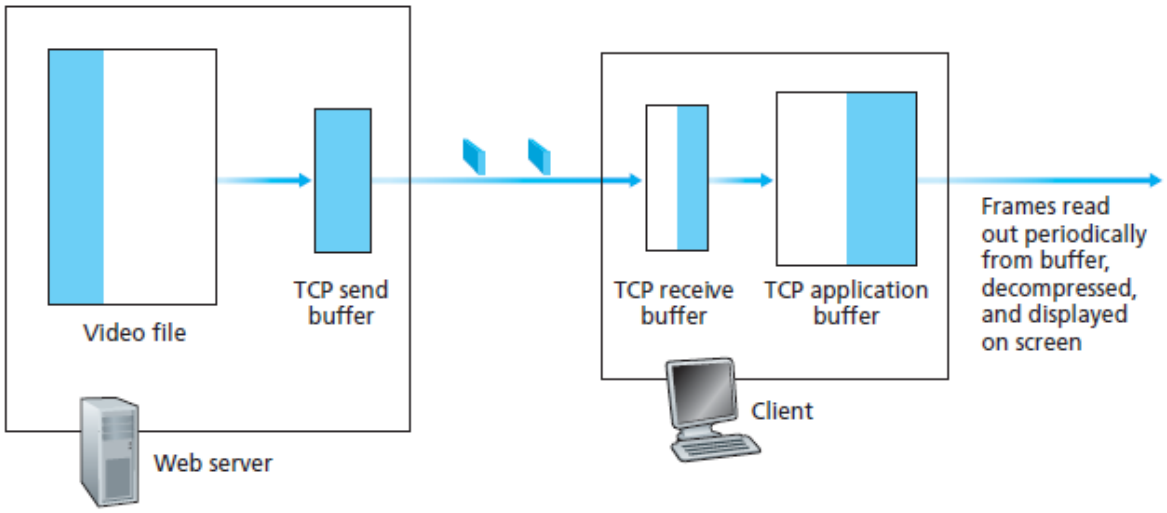


Figure 7.2 illustrates the interaction between client and server for HTTP streaming.

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.

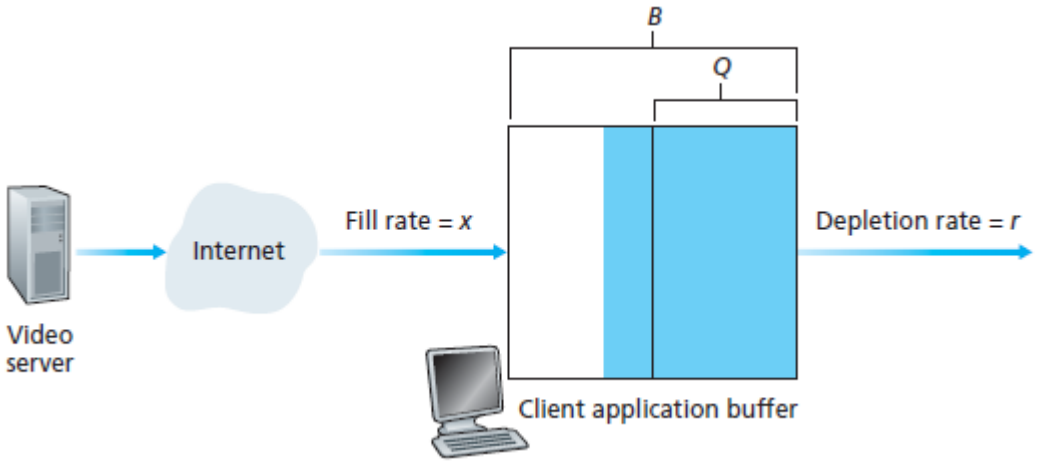
Therefore, *a full client application buffer indirectly imposes a limit on the rate that video can be sent from server to client when streaming over HTTP.*

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.

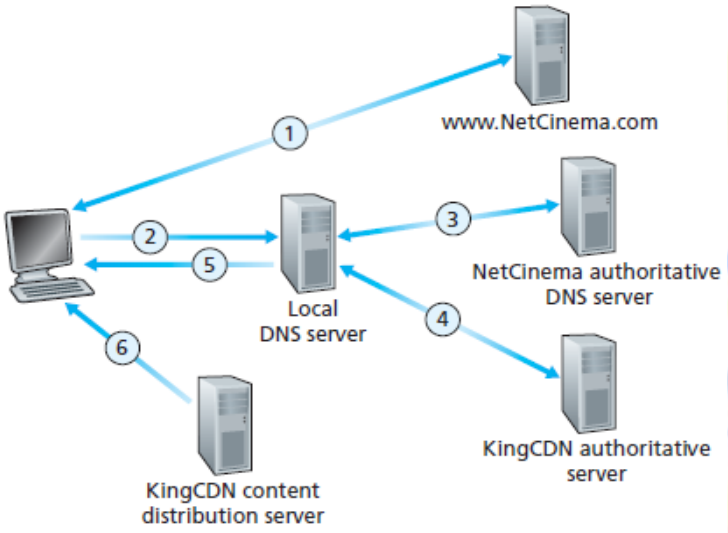
consider a example to illustrate how DNS is typically involved.

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.

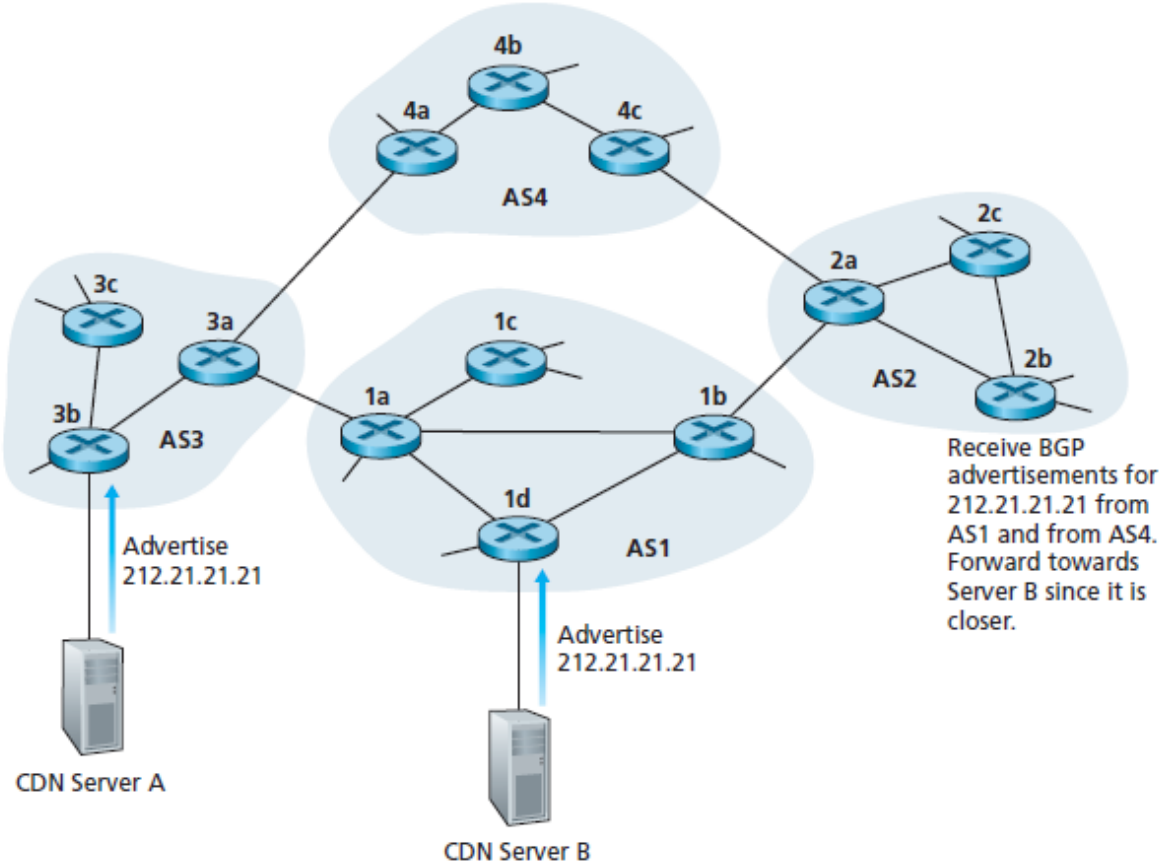
Such solutions, require redirecting clients to (possibly) suboptimal clusters from time to time in order to measure the properties of paths to these clusters.

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.

7.2.5 **Case Studies: YouTube**

YouTube

YouTube is the world's largest video-sharing site.

YouTube makes extensive use of CDN technology to distribute its videos. Unlike Netflix, however, Google does not employ third-party CDNs but instead uses its own private CDN to distribute YouTube videos.

Google has installed server clusters in many hundreds of different locations. From a subset of about 50 of these locations, Google distributes YouTube videos .

Google uses DNS to redirect a customer request to a specific cluster.

Most of the time, Google's cluster selection strategy directs the client to the cluster for which the RTT between client and cluster is the lowest; however, in order to balance the load across clusters, sometimes the client is directed (via DNS) to a more distant cluster.

Furthermore, if a cluster does not have the requested video, instead of fetching it from somewhere else and relaying it to the client, the cluster may return an HTTP redirect message, thereby redirecting the client to another cluster.

YouTube employs HTTP streaming. YouTube often makes a small number of different versions available for a video, each with a different bit rate and corresponding quality level.

YouTube does not employ adaptive streaming (such as DASH), but instead requires the user to manually select a version.

In order to save bandwidth and server resources that would be wasted by repositioning or early termination, YouTube uses the HTTP byte range request to limit the flow of transmitted data after a target amount of video is prefetched.

A few million videos are uploaded to YouTube every day.

YouTube videos streamed from server to client over HTTP and YouTube uploaders also upload their videos from client to server over HTTP. YouTube processes each video it receives, converting it to a YouTube video format and creating multiple versions at different bit rates.

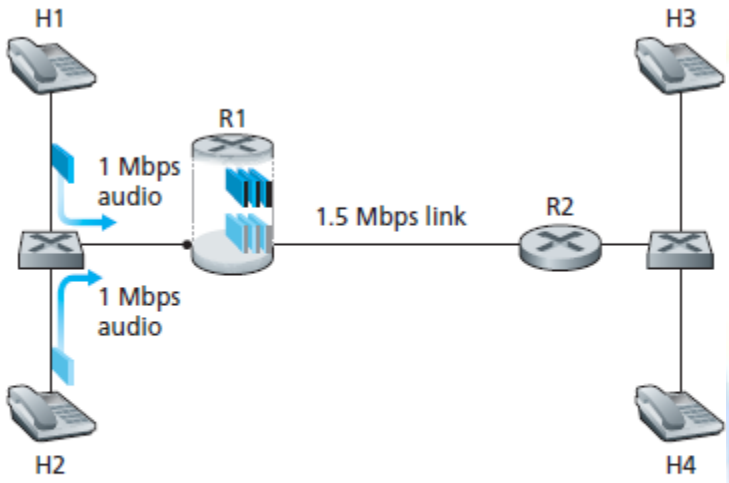
This processing takes place entirely within Google data centers. Thus, in stark contrast to Netflix, which runs its service almost entirely on third-party infrastructures, Google runs the entire YouTube service within its own vast infrastructure of data centers, private CDN, and private global network interconnecting its data centers and CDN clusters.

7.5.4 Per-Connection Quality-of-Service (QoS) Guarantees:Resource Reservation and Call Admission

- In the previous section, we have seen that packet marking and policing, traffic isolation, and link-level scheduling can provide one class of service with better performance than another.
- Under certain scheduling disciplines, such as priority scheduling, the lower classes of traffic are essentially "invisible" to the highest-priority class of traffic.

- With proper network dimensioning, the highest class of service can indeed achieve extremely low packet loss and delay—essentially circuit-like performance.
- But can the network guarantee that an ongoing flow in a high-priority traffic class will continue to receive such service throughout the flow's duration using only the mechanisms that we have described so far? It cannot.
- In this section, we'll see why yet additional network mechanisms and protocols are required when a hard service guarantee is provided to individual connections.

Consider two 1 Mbps audio applications transmitting their packets over the 1.5 Mbps link, as shown in Figure 7.27.



igure 7.27 ♦ Two competing audio applications overloading the R1-to-R2 link

The combined data rate of the two flows (2 Mbps) exceeds the link capacity.

- Even with classification and marking, isolation of flows, and sharing of unused bandwidth (of which there is none), this is clearly a losing proposition. There is simply not enough bandwidth to accommodate the needs of both applications at the same time.
- If the two applications equally share the bandwidth, each application would lose 25 percent of its transmitted packets. This is such an unacceptably low QoS that both audio applications are completely unusable; there's no need even to transmit any audio packets in the first place.
- Given that the two applications in Figure 7.27 cannot both be satisfied simultaneously, what should the network do?
- Allowing both to proceed with an unusable QoS wastes network resources on application flows that ultimately provide no utility to the end user.
- The answer is hopefully clear—one of the application flows should be blocked (that is, denied access to the network), while the other should be allowed to proceed on, using the full 1 Mbps needed by the application.

- The telephone network is an example of a network that performs such call blocking—if the required resources (an end-to-end circuit in the case of the telephone network) cannot be allocated to the call, the call is blocked (prevented from entering the network) and a busy signal is returned to the user.
- In our example, there is no gain in allowing a flow into the network if it will not receive a sufficient QoS to be considered usable.
- By explicitly admitting or blocking flows based on their resource requirements, and the source requirements of already-admitted flows, the network can guarantee that admitted flows will be able to receive their requested QoS.
- Implicit in the need to provide a guaranteed QoS to a flow is the need for the flow to declare its QoS requirements.
- This process of having a flow declare its QoS requirement, and then having the network either accept the flow (at the required QoS) or block the flow is referred to as the call admission process.

Insight 4: If sufficient resources will not always be available, and QoS is to be guaranteed, a call admission process is needed in which flows declare their QoS requirements and are then either admitted to the network (at the required QoS) or blocked from the network (if the required QoS cannot be provided by the network).

Our motivating example in Figure 7.27 highlights the need for several new network mechanisms and protocols if a call (an end-to-end flow) is to be guaranteed a given quality of service once it begins:

• **Resource reservation.**

- The only way to guarantee that a call will have the resources (link bandwidth, buffers) needed to meet its desired QoS is to explicitly allocate those resources to the call—a process known in networking parlance as resource reservation.
- Once resources are reserved, the call has on-demand access to these resources throughout its duration, regardless of the demands of all other calls.
- If a call reserves and receives a guarantee of x Mbps of link bandwidth, and never transmits at a rate greater than x, the call will see loss- and delay-free performance.

• Call admission.

- If resources are to be reserved, then the network must have a mechanism for calls to request and reserve resources. Since resources are not infinite, a call making a call admission request will be denied admission, that is, be blocked, if the requested resources are not available.
- Such a call admission is performed by the telephone network—we request resources when we dial a number.
- If the circuits (TDMA slots) needed to complete the call are available, the circuits are allocated and the call is completed.

- If the circuits are not available, then the call is blocked, and we receive a busy signal.
- A blocked call can try again to gain admission to the network, but it is not allowed to send traffic into the network until it has successfully completed the call admission process.
- Of course, a router that allocates link bandwidth should not allocate more than is available at that link.
- Typically, a call may reserve only a fraction of the link's bandwidth, and so a router may allocate link bandwidth to more than one call. However, the sum of the allocated bandwidth to all calls should be less than the link capacity if hard quality of service guarantees are to be provided.

• **Call setup signaling.**

- The call admission process described above requires that a call be able to reserve sufficient resources at each and every network router on its source-to-destination path to ensure that its end-to-end QoS requirement is met.
- Each router must determine the local resources required by the session, consider the amounts of its resources that are already committed to other ongoing sessions, and determine whether it has sufficient resources to satisfy the per-hop QoS requirement of the session at this router without violating local QoS guarantees made to an already-admitted session.
- A signaling protocol is needed to coordinate these various activities—the per-hop allocation of local resources, as well as the overall end-to-end decision of whether or not the call has been able to reserve sufficient resources at each and every router on the end-to-end path. This is the job of the call setup protocol, as shown in Figure 7.28.
- The RSVP protocol was proposed for this purpose within an Internet architecture for providing quality of-service guarantees.
- In ATM networks, the Q2931b protocol carries this information among the ATM network's switches and end point. Despite a tremendous amount of research and development, and even products that provide for per-connection quality of service guarantees, there has been almost no extended deployment of such services.

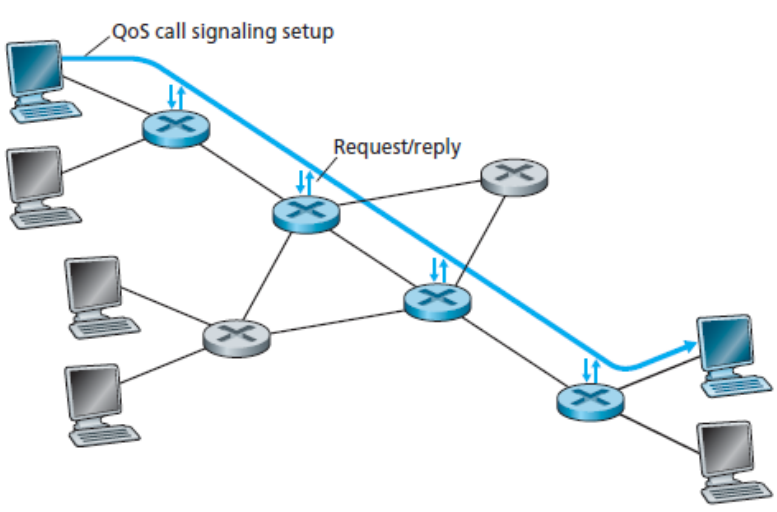


Figure 7.28 ♦ The call setup process

- There are many possible reasons.
- First and foremost, it may well be the case that the simple

application-level mechanisms that we studied in Sections 7.2 through 7.4, combined with proper network dimensioning provide "good enough" best-effort network service for multimedia applications.

- In addition, the added complexity and cost of deploying and managing a network that provides per-connection quality of service guarantees may be judged by ISPs to be simply too high given predicted customer revenues for that service.