Chapter 7 - Overview of Streaming Media

We now turn our attention to delivering audio and video across networks. There are a number of ways to think about how multimedia data is delivered across networks. In terms of network latency, the multimedia data can be interactive, stored, or delay tolerant. In addition, the data can be delivered over reservation-based or best-effort networks. In the remainder of this chapter, we will describe each area at a high-level focusing on the differences in solutions for each. We will then describe each area in more detail in later chapters.

7.1 Best-Effort Networking

Best-effort multimedia streaming is the delivery of continuous multimedia streams over Internet-like networks. The key problem with best-effort multimedia networking is the fact that the end-to-end bandwidth and latency can be highly variable over time. Because no end-to-end coordination is necessary to reserve bandwidth for a particular multimedia session, the system is inherently more scalable. Of course, this means the application needs to deal with a number of potential problems including missing packets and network delay jitter.

7.1.1 Best-effort Interactive Video Streaming

For best-effort interactive video streaming, not only does the application need to deal with variable end-to-end bandwidth and jitter, it has to do so under extreme time pressure. For interactive video services, the end-to-end latency requirements are typically less than 200 milliseconds. This means that the system needs to capture a frame, compress it, transmit it across the network, decompress it, and display it. Doing so continuously over time, without interrupt! While all of these constraints may make it seem impossible to implement an interactive video streaming application that actually works, there are a number of things that make interactive video streaming somewhat simpler. First, in interactive video streaming sessions, the video is being generated on-the-fly. As a result, the data is typically being generated from a single camera. Because the contents of the scene do not change that much, this means the compressed data will be relatively stable over time. Second, because the video is being compressed on-the-fly, the video can be coded directly into its target size. In contrast, with a frame that has already been compressed (e.g. stored video), the video needs to be partially decompressed and recompressed to meet the target size.

7.1.2 Best-effort Stored Video Streaming

For best-effort stored video streaming, the tight end-to-end latency constraints do not exist. That is, stored video delivery requires "just in time" delivery, meaning that any frame of the video could be sent *arbitrarily* early. While this mitigates the end-to-end latency problem, in its place, stored video streams have already been compressed. This means that in the delivery of the video, the stream may potentially need to be adapted to the underlying network bandwidth. In some cases, where the stream is relatively small and no adaptation needs to be performed, this means that the sender of the data does not need to encode or re-encode the video data, making it somewhat easier than the interactive streaming case. In other cases, where the stream needs to be downsized to fit network bandwidth, the server either needs to spend extra computation and storage up front to make the video amenable to

adaptation or needs to spend extra computation during transmission to downsize the video stream on-the-fly. To further complicate matters, because the video is stored, the video typically consists of more than one distinct scene. This means that the video will have a much more variable bit-rate requirement over time, especially for variable-bit-rate encoded video streams. Finally, because the video is stored, the users may require the ability to scan and rewind through the video. Scans, particularly those in the forward direction, make planning of the delivery of video harder.

7.2 Reservation-based Multimedia Networking

Reservation-based multimedia networking assumes the ability to reserve bandwidth between the two entities streaming the multimedia data. Such reservations require that the entire path between the client and the server be reserved. For traditional telephony networks, where the "bandwidth" has been traditionally managed with circuit switching, reserving bandwidth involves allocating sufficient channels to deliver the required resources. For virtually any other type of networking technology strict reservation-based resources becomes more complicated. One of the main reasons is that most end hosts use either Ethernet-based or 802.11-based technologies. Neither of these technologies allows bandwidth reservations. For wireless networking, it is further complicated by random interference in the air. If the Ethernet LAN or 802.11 network are not heavily used (that is, the bandwidth available is always greater than the reservation bandwidth), it is possible to deliver the bandwidth reservation to the client but may still suffer from bursty delivery of the data. The second problem with traditional networking technologies is that they often do not support reservations directly. Rather, a trained networking support person may need to configure the routers to provide the reservation across the network.

The ability to deliver a reservation for bandwidth using traditional networking technologies is heavily dependent upon what entities "own" the network for which the bandwidth reservation is required. For networks that are under the control of a single entity (e.g. within a company), setting up a reservation is fairly easy and has the proper incentive (that is, it only helps the company). For networks that have multiple controlling interests such as the wide area Internet, reservations are hard, if not impossible, to provide. The main reason is that every controlling interest along the way will be trying to make as much money on the reservation as possible.

At this point there are two types of reservations worth mentioning. Obviously allocation network resources equivalent to the *peak* bandwidth requirement for an application will provide the best service to the application. This, however comes with a huge cost. Consider the following scenario. Suppose we want to deliver MPEG video data using the frame pattern: IBBPBBPBBIBBPBBPBB… Further, suppose the I, P, and B frames are compressed with compression ratios 20:1, 50:1, and 100:1. This means that if we allocate at the peak bandwidth (i.e. the I-frame bandwidth), then the network reservation will be only 1/3rd utilized. That is 66% of the time, the network will be carrying *no data.* Clearly, the network bandwidth has been way over provisioned. If the network reservation is reduced, then the client may need to adapt its video to the underlying reservation. Of course, having some reservation that you can count on all the time is easier to deal with than no reservations at all, as in the best-effort networking case. We will describe the interactive video and stored

video delivery over reservation-based networks using both over provisioned allocation and a reduced bandwidth allocation.

7.2.1 Reservation-based Interactive Video Streaming

Over provisioned, reservation-based interactive video (ORIV) streaming is perhaps the easiest to implement from a systems perspective. It is also the most costly. For over-provisioned, reservation-based interactive video streaming, network resources are allocated at the peak bandwidth requirement of the application. While the actual peak bandwidth is not known ahead of time in an interactive video streaming session, minor adjustment in the quantization can make this easy to adhere to. ORIV streaming is also simplified by the fact that the video is being generated on-the-fly. As a result, a single camera is typically being used to generate the data so that the bandwidth requirement will be relatively stable over time.

As previously mentioned, allocating bandwidth at the peak bandwidth requirement the data is typically being generated from a single camera. Assuming one wants to have a smaller than peak target, there are two options available: reducing the compressed bit stream being created or use some buffering in the system to reduce the peak bandwidth requirement of the stream. These will be discussed in a later chapter.

7.2.2 Reservation-based Stored Video Streaming

As with the best-effort stored video streaming, the tight end-to-end latency constraints do not exist for reservation-based stored video streaming. The "just in time" delivery semantics also apply here. In addition, many of the other points apply including (i) the video streaming may be complicated by forward and reverse scans, (ii) the video stream will probably have higher bandwidth variation requirements due to the different scenes typical of prerecorded content, (iii) buffering is necessary to support data that is sent before it is going to be played back.

For simple delivery, one can allocate the reservation such that it is equivalent to the peak rate of the entire stream. As previously mentioned, this will undoubtedly lead to massive underutilization of the network. The key, of course, is the use of the buffer in reducing the bandwidth requirements. Because data can be sent arbitrarily early, if one can plan the bandwidth usage ahead of time, the bandwidth reservation necessary to support continuous playback can be reduced.

Chapter 8 - Interactive Streaming Applications

We break our discussion of interactive streaming applications into delivering media over reservation-based and best-effort interactive video applications.

8.1 Interactive Streaming over Reservation-Based Networks

Of the four streaming application types, interactive, reservation-based streaming is perhaps one of the easiest applications to implement. At the same time, it is perhaps one of the most expensive, because of the cost of reserving bandwidth.

If the bandwidth reserved is greater than the peak bandwidth requirement for every frame of video then we are guaranteed of two things. First, the application will not have to adapt its video at all. This means that the application only needs to capture the video data and send it directly across the network. Second, this also means that the network resources are being wasted due to bandwidth that has been allocated but used only for the peak bandwidth frames.

If the bandwidth reserved is at a rate less than the peak, than the application needs to adapt the video stream, perhaps only slightly, to fit within the reserved bandwidth. Fortunately, however, the application typically has a target rate at which to code the video. If you recall, the H.261 video coding technique has a rate control algorithm that allows it to code video to a target bit rate. The same algorithm can be used to adapt the video stream for interactive streaming in the same exact way. In addition, because the coding is done on-the-fly, it is relatively easy to manage the adaptation of the video.

Another alternative is to add a little buffering at the client. This will allow the larger independently coded video frames to be smoothed over several smaller predictive coded frames.

8.2 Interactive Streaming over Best-Effort Networks

8.2.1 Best Effort Basics

For best-effort streaming, there are a number of problems that the network can introduce. These include: delay jitter, packet loss, out-of-order arrivals, and duplicate arrivals.

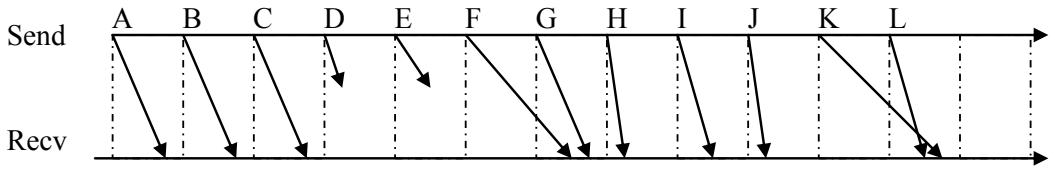
Delay jitter is the variation in delay experienced by the stream of packets. In an ideal system, all packets for a particular stream would have exactly the same delay. In this way, it is very easy to schedule the display of incoming packets. That is, just display them as they come in because you are guaranteed that the next packet will arrive in time for the next display interval. With delay jitter, some buffering, and hence delay, need to be added to the system in order to provide continuous playback. Suppose one packet arrives with small delay and the next packet arrives with a slightly larger delay. In this case, without buffering, the first packet will be displayed and an interruption will appear while waiting for the second packet to arrive. The root cause of delay jitter is that the routers in the network are processing packets as they arrive. As the router's queues fill up, the packets get processed in the order that they arrive. Ensuring that all packets from a particular stream receive the same delay through the router is arbitrarily hard and complex to implement in practical systems.

Packet loss is due to packets being dropped within the network. Most of the time, this is due to congestion in the network causing a router's buffer to overflow. The cause, of course, is the fact that most systems use TCP, which by its nature, is trying to probe the network so that it is always sending at the highest rate. Packet loss can also be attributed to other factors such as transient network loops causing the time to live field in the IP packet to expire or can be attributed to a bad wireless network transmission.

Out of order arrival does not occur that often but can occur. This will manifest itself in two ways. If the packets are being buffered for delay jitter mitigation purposes, the packets can be simply reordered. If the packets are being played back with little buffering, then the out of order packet will appear to have arrived late. In this case, it is typically discarded.

Duplicate arrivals occur because of retransmission-based transport layer techniques such as TCP retransmitting packets, when they haven't been lost but have not yet been acknowledged. This is the easiest case as it can be discarded on the receiver.

The way we will describe delay jitter, packet loss, and out of order arrivals are shown below:

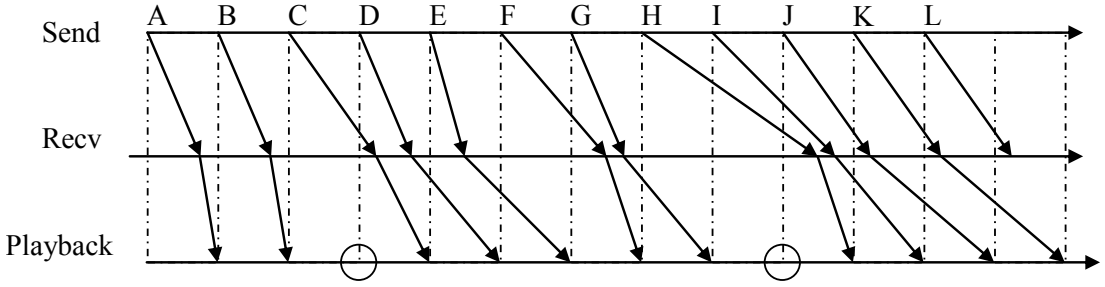


In this figure, packets A through C are delivered with ideal delay jitter. That is, there is no jitter and the packets are delivered as they were spaced when sending. Packets D and E demonstrate packets that have been lost in the network (dropped due to congestion). Packets F through J demonstrate the effect of delay jitter. Note how the arrival of packets is different than the rate at which they were sent. Finally K and L show an out of order packet sequence.

8.2.1.1 Managing Delay Jitter

There are two fundamental ways in which to deal with delay jitter in terms of packet arrival. When a packet arrives, we can either playback the packet immediately or we can buffer the packet. Without loss of generality, we assume that each packet contains a displayable unit of multimedia.

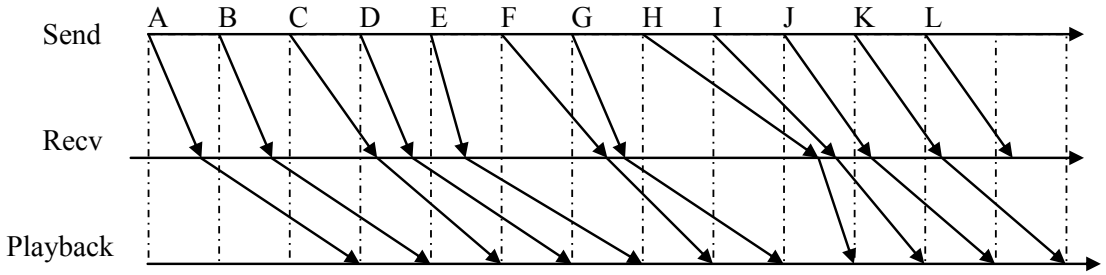
Suppose we decide to playback the packets as soon as they arrive.



In the above example, playing packets as soon as they arrive results in disruptions in playback (as shown with circles on the playback line). In addition, after a long delay (e.g.

packet H), packets will start to be queued up and delayed because the system has samples to play. The delay will continue to grow depending upon the worst case delay latency. The only way to "catch" up is if packets are lost in the network.

At the other extreme, when a new packet arrives, we can enqueue the packet in order to mitigate the effects of delay jitter. As shown in the following example:



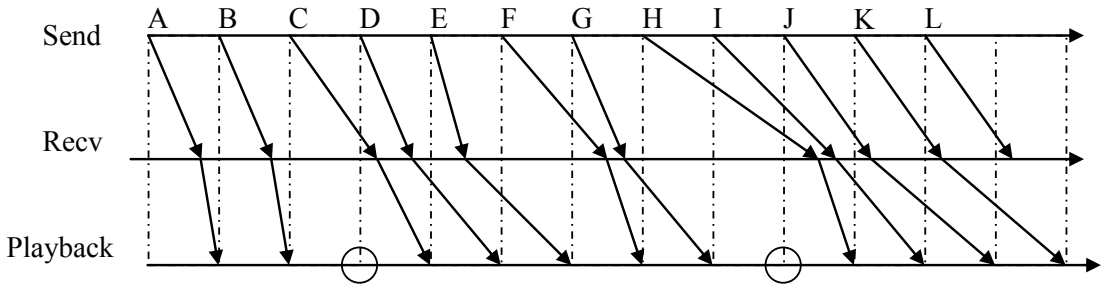
The initial packet A has been held for two display periods. This allows for continuous playback because it allows packets with large delay enough time to arrive. In this example, it allows, for example, packets F and H to get to the destination. This, of course, is at the expense of delay.

The key to all of this is that some delay is necessary to deal with delay jitter in order to allow continuous playback. Too much delay, however, introduces too much delay for interactive playback.

8.2.1.2 Managing Delay

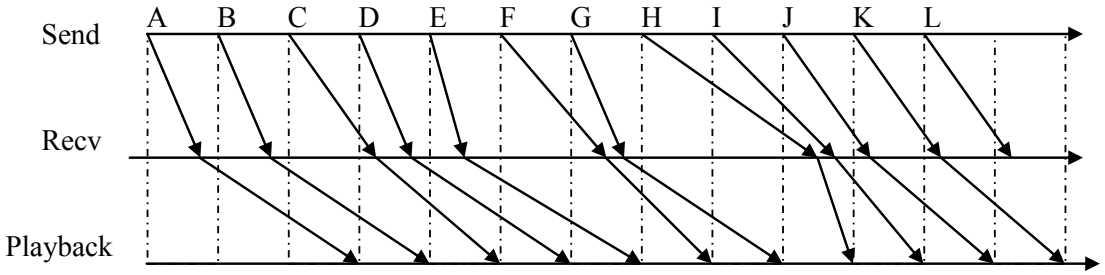
In order to ensure interactive playback of video, the client needs to be able to distinguish when a packet should be enqueued for playback in order to provide continuous playback versus causing unnecessary delay in playback. There are two basic ways in which this can be accomplished.

There are three fundamental approaches to deal with delay. The first is let the naturally occurring network delay determine the playout latency. Thus, a scenario similar to the first graph in the previous subsection happens where delay naturally increases.

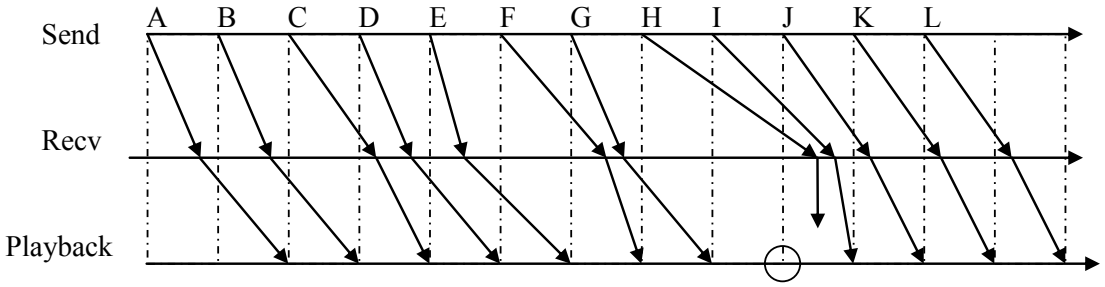


The second approach is to select an initial playout latency and set the playout delay to it. Typically, the way to measure network delay is to timestamp packets on the sender side and

record when they arrive at the client. As an example, if we pick the delay to be large, then we can ensure continuous playback.



The third way to deal with delay is to set a delay (hopefully, based upon the actual network delay) and to discard "late" packets in order to maintain low-latency. As an example, suppose we take the previous example and set the playout latency to 2 frames. Here we see that packet H arrives late and is discarded. By discarding this, we can keep the delay to 2 frames. Otherwise, after packet H, the delay will increase to 3 if all packets are played back.



In practice, it is difficult to experimentally measure the delay. In the following section, we describe a Queue Monitoring technique that allows us to indirectly measure the network delay.

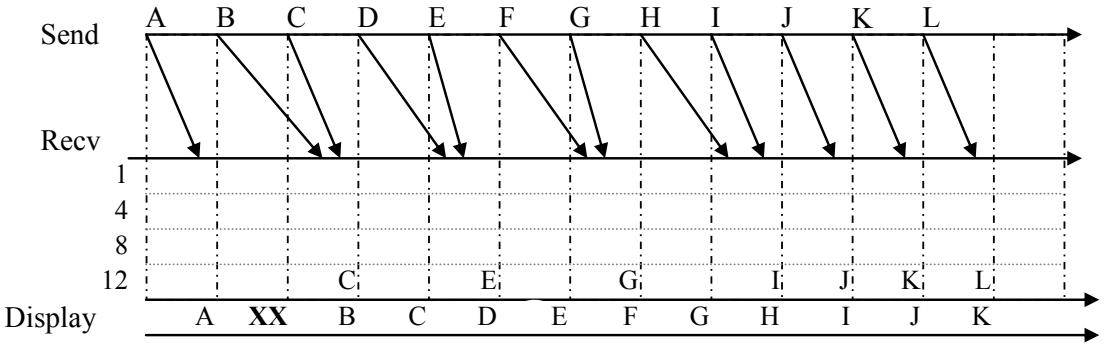
8.2.2 Queue Monitoring

Queue monitoring is a technique that uses the queue of multimedia data at the client to indirectly measure network delay. If the queue is growing in size, this means that the number of packets arriving at the client is greater than is being consumed by the delay (i.e. removed from the queue). The only way that this can happen is if the network delay is decreasing, and therefore, delivering packets to the client at a higher rate. If the queue size remains constant, then that means the network delay is constant. That is, packets are arriving in the queue at the same rate other packets are being removed for display. Finally, If the queue is shrinking in size, this means that the packet arrival rate is slower than the rate at which packets are being removed for display. This means that the network delay is increasing.

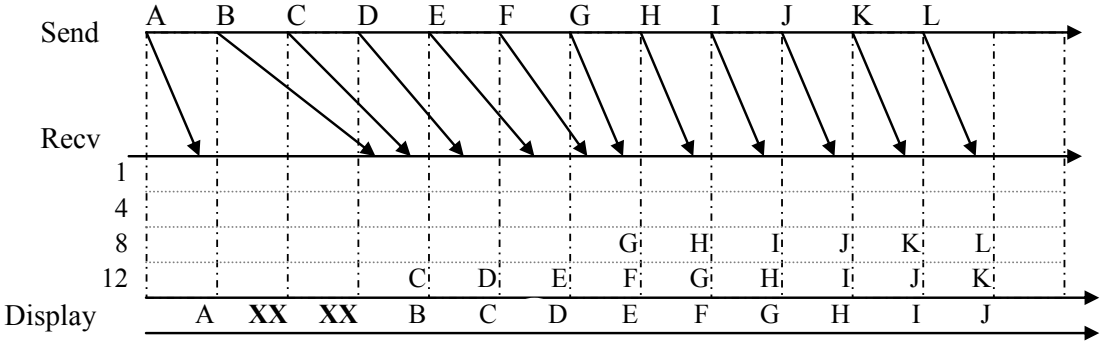
In the queue monitoring approach, every position in the queue for display is assigned a threshold. This threshold indicates the number of consecutive display intervals in which the corresponding queue position is allowed to have a sample in it. If this threshold is crossed, this indicates that the network delay has been stable (or growing) because packets are starting to be queued up. As a result, removing samples can decrease the delay in playback. Thus,

the queue is used to deal with variations in delay as well as a way to potentially minimize playout latency.

For the delay jitter case, consider the following example:



In this example, we see that the queue is used to deal with delay jitter effectively by storing a packet so that if some of the packets have longer delay (e.g. D, F, and H), that there is sample to be played back. Now, consider the following example:

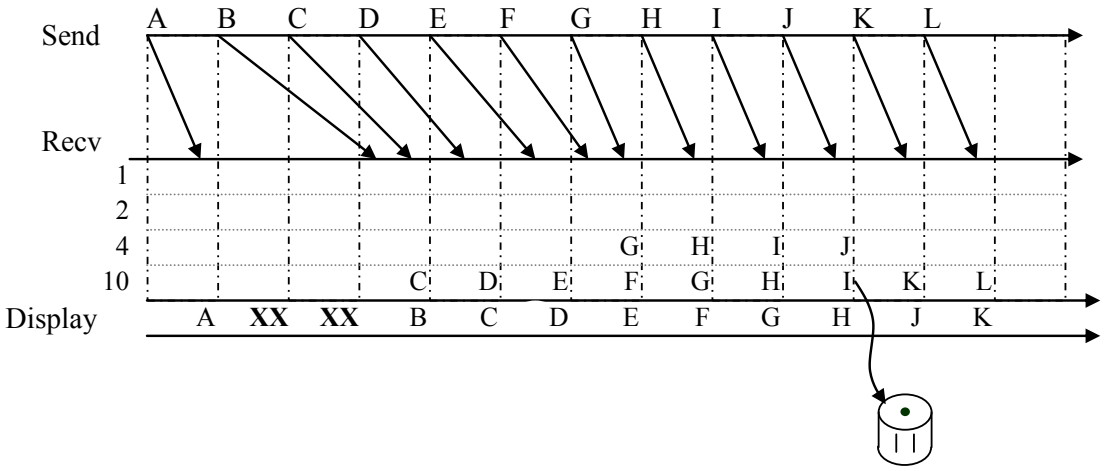


In this example, we see that after some packets get delayed (packet B), that the following packet (C), gets queued up because it arrives in the same interval. During the delivery of packets F and G, the network delay decreases further. As shown in the packets end up getting queued up. Because the delay has become constant with no network jitter, we see that without doing anything, we have introduced unnecessary delay into the system. This is exactly what queue monitoring does.

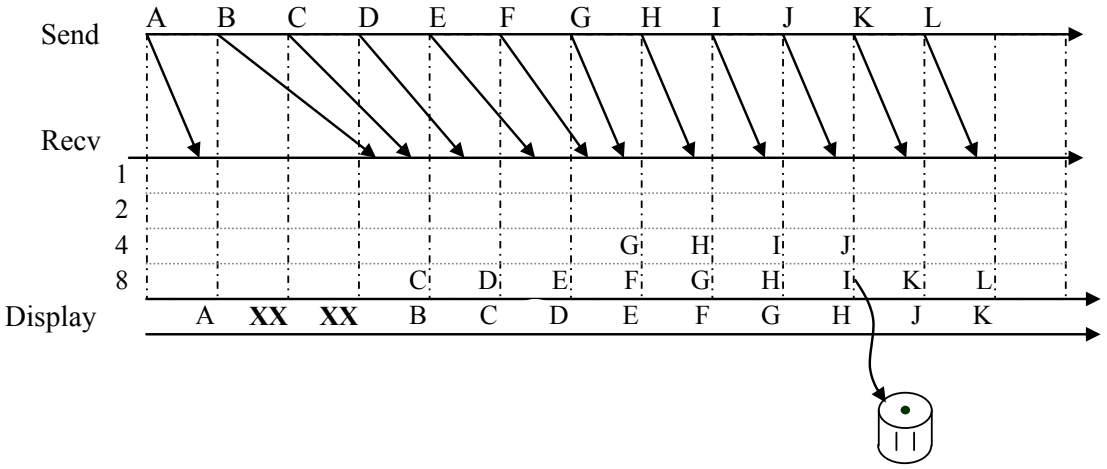
The queue monitoring algorithm has a number of steps. These are outlined below:

- For each queue entry, keep a threshold that indicated the number of consecutive frame intervals in which the entry is allowed to have an item in it.
- Start with all entrys to 0 and begin streaming
- Each time there is an entry in the queue increment a counter. If there is no entry in the queue, reset the counter to 0.
- If the counter for an entry meets the threshold, remove the oldest packet in order to reduce delay.

Now, consider the following example:



In this example, the second queue entry (the row labeled 4) reaches its threshold after the arrival of packet J. That is, the second entry in the queue has had 4 consecutive frame intervals in which a packet has been queued. Thus, the oldest sample is removed. The oldest sample, in this case, is I. Generally, the queue thresholds follow a geometric sequence. This allows the system to quickly return to low latency following the delivery of a large number of packets and at the same time slowly approach minimal latency in order to avoid discontinuity in playback. As another example:



8.2.3 Adaptive Interactive Video Streaming

Queue monitoring works well for managing audio packets for an interactive streaming session. This is mostly due to the relatively constant encoding size of the packets, as well as the fact that the audio packets tend to be fairly small. For interactive video streaming, we break the interactive video streaming session into two main pieces (i) the tailoring mechanism, and (ii) the adaptation mechanism. The former provides the video-specific adaptation, while the latter provides the dynamic targeting of the bandwidth to encode at. In the remainder section, we will describe basic tailoring mechanisms for interactive video streaming.

8.2.3.1 . *Tailoring Mechanisms*

There are a large number of techniques that could be employed to reduce the size of a compressed interactive streaming session. Fortunately, the video in interactive streaming sessions is being captured, compressed, and transmitted as quickly as possible. This means that after capture, the video is in the form that is most amenable to change. That is, uncompressed RGB or YUV data. Any manipulations occur on the actual data before the compression process. The types of data reduction that can be used include:

- Temporal scaling – the encoder can reduce the frame rate of the video. This can be accomplished by reducing the number of frames captured at the camera or by dropping the frames before compression.
- Quality scaling – as we saw in the video compression chapter, one of the primary mechanisms employed by encoders is the output buffer feedback mechanism. For MPEG-1 video, the data is taken out of the buffer at 1.5 Mbps. The quality is then adjusted so that the buffer does not overflow or underflow. This is perhaps the easiest mechanism to implement because it is part of most encoders already.
- Resolution scaling – as an alternative to quality scaling, one could encode a smaller resolution video stream. As an example, the encoder could capture 160x120 pixel video, compress, and transmit it to the receiver. Upon receipt, the system could upscale the video to 320x240 resolution.
- Amplitude scaling – amplitude scaling reduces the pixel depth of each color channel. For example, rather than using 8-bit samples, the system could use 4-bit samples. Amplitude scaling by itself does not typically work that well because the sacrifice in quality is far greater than the reduction in bandwidth requirements.
- Color space scaling – color space scaling can be used to reduce the overall number of colors. This is typically used in conjunction with indexed color spaces such as in the GIF image format. As with the amplitude scaling, color space scaling is not generally usable for video, given its large color space needs.

Almost all systems implemented today employ quality scaling as the primary tailoring mechanism. The primary reason for this is that the video coder already has a bit-rate target buffer mechanism built into it. Thus, the only thing that needs to be done is that the adaptation mechanism needs to provide a target bit-rate for the tailoring mechanism to target.

Another alternative that can be employed is the notion of *two-dimensional* media scaling. In this approach, the frame rate and the frame quality are linked together. For example, suppose that the quantization scale range ranges from 1 to 10, where 1 represents the highest quality and 10 represents the poorest quality. Then, when the quality is 1 the video is encoded at the full frame rate. If the quality is 10, then the video is encoded at a small frame rate (e.g. 1 frame per second). Using such an approach, provides a greater range of bandwidth adaptation and a smoother degradation in the quality of the video.

8.2.4 Adaptive Interactive Streaming Components

With the tailoring mechanism in place, we now have a way to generate the video streams that we require. Fortunately, given the small latency requirements, there are not a lot of options to do other things. Otherwise, our lives would be more complicated (as we will see in the stored video streaming chapter). The *adaptation mechanism* is responsible for

implementing the rest of the interactive video streaming session. Some of the design issues for the adaptation mechanism include what transport layer mechanism to use, how to deal with packet loss, and how to deal with congestion control. In the remainder of this section, we will put the rest of the video adaptation system together. We will be creating a

Most interactive streaming algorithms will use a combination of TCP and UDP for the transport protocol. UDP is used to deliver the video to the destination with low-latency. The TCP channel is used by the receiver to transmit feedback to the sender so that it can adjust the "prototypical" interactive streaming architecture.

8.2.4.1 Interactive Streaming Transport Protocol

There are two options for transport protocols: the user datagram protocol (UDP) and the transmission control protocol (TCP).

The TCP protocol provides a reliable, congestion-sensitive, connection-oriented transmission stream. Reliability is provided via retransmissions (done at the network layer and not the application layer). Congestion-sensitivity is provided through a additive-increase, multiplicative-decrease algorithm. This means that if *n* streams are sharing a common network link that, over time, each of the streams will use approximately 1/*n*th the bandwidth. Because of this, it makes networking systems much more scalable and avoids congestion collapse[4]. The main disadvantage of such congestion-sensitivity is that using TCP means that the bandwidth will follow what is known as the *TCP sawtooth*, bandwidth that fluctuates continually over time. This makes it difficult to provide an accurate bandwidth target for the tailoring mechanism to encode to. Finally, TCP is connection-oriented which means that all the packets will arrive to the application in order.

Overall, the main advantage of using TCP as the transport layer is that it makes it extremely easy to implement. The user simply needs to issue write calls to the network sock. The main disadvantage of TCP, particular for interactive streaming, are the bandwidth fluctuations due to the TCP sawtooth, and the extra latency introduced when retransmissions are necessary. Finally, because the operating system buffers TCP packets, the latency is extremely hard to control under TCP.

The UDP protocol provides a simple "address the packet and send it" mechanism. This has a number of ramifications. First, there is no reliability built into the protocol. If the application wants to retransmit missing data, it has to do it. Second, there is no congestion sensitivity. This means that the application can send an arbitrary amount of data that is being sent by the application. Finally, the application needs to deal with packet duplicates and out-of-order delivery of packets to the application.

Overall, the main advantages of using UDP for interactive streaming are (i) the low-latency, minimal overhead nature of UDP and (ii) targeting a particular bit-rate is relatively easy (just send what you want). The main disadvantages are the havoc that an overly large stream can cause on the network and the fact that the application has to more indirectly discover the bit-rate at which to send at. As an example of the latter, suppose that we have a home networking scenario with a PC connected to a network address translation (NAT)

[4] Congestion collapse is analogous to thrashing in an operating system. Very little usable packets are actually delivered with a large percentage of them being dropped within the routers along the way.

device, which in turn is connected to the user's Internet Service Provider (ISP). In this scenario, the PC will be connected via a high-speed link to the NAT (e.g. 100 Mbps). The NAT is connected to the ISP at a pre-agreed upon rate (e.g. 384 kbps). Suppose the user is generating video at 1 Mbps. Here, the video is being captured, compressed, and transmitted easily to the NAT box. The NAT box, however, is being inundated with packets. It can only transmit packets at 384 kbps but is receiving data at 1Mbps. Here, the NAT box will simply drop any packets that do not fit into its buffer. Thus, 2/3rd of the data will never make it out of the user's home! The only way to determine that the bit-rate needs to be reduced is that the application writer needs to determine what the rate is through packet-loss measurement.

8.2.4.2 Dealing with Packet Loss / Corruption

Assuming that UDP is being used as the transport protocol, the application will have to deal with packets that were lost in transmission. There are three broad techniques that can be employed. The first and simplest is to simply ignore the fact that that a packet was dropped in the network and to make the receiver deal with the fact that it will not receive the lost packet. *Reactive* approaches wait for packet loss to occur and then respond to such event by resending the missing packet, if so desired by the adaptation mechanism. *Proactive* approaches introduce redundancy into the stream so that receiving side has extra information in the hopes of recovering missing data.

Reactive approaches: Conventional wisdom says that retransmissions for interactive video streaming applications are a waste of time. The reason for this is that it requires one complete round trip in order to retrieve the missing packet. For a low-latency application, this means that, at a minimum, a full round-trip time of delay must be added to the system so that the video can be continuous. Retransmissions, however, are not entirely useless. For reference frames, the retransmission can be used to clear up other frames that depend on it. Retransmission of missing data can be accomplished through traditional mechanisms such as a negative acknowledgement (NACK).

Proactive approaches: Proactive approaches to packet loss add redundant forward error correction (FEC) data to the stream so that the receiver can potentially recover missing data. A really simple FEC mechanism can be sending a redundant copy of all packets over the network. This comes at the expense of being able to send only half the data rate! For proactive approaches, there is a distinction between the types of data that need to be recovered: packet loss vs. bit-errors. Packet losses are much more difficult to recover from than bit-errors. In general, it is my opinion that FEC for packet loss should only be employed when the round-trip time is extremely large. Otherwise, FEC should be used for bit-error recovery, as commonly found in wireless network transmission. More advanced FEC algorithms such as Reed-Solomon codes are commonly used for such approaches.

8.2.4.3 Congestion Control

Buffering, retransmission, and FEC limit the effects of packet loss and variation in end-to-end delay on the continuity of the video stream. However, the do not really address what the root cause is. That is, network congestion. Packets in the Internet get dropped when there are too many packets at a given time for a router to handle. There are numerous causes including (i) the stream we are generating is too large a bit-rate or (ii) there are too many

streams sharing too little bandwidth. In any case, the solution is clear. We need to reduce the amount of data being transmitted!

The goal that we are seeking is that we want to make the best use of the time-varying bandwidth that is available to our stream. This requires us to answer the following questions:

- What is the bandwidth available?
- How do we track the network bandwidth changes over time?
- How do we match the codec to bandwidth available?

For UDP-based interactive streaming, typically the bandwidth and target bit-rate are determined based upon watching the packet loss characteristics. The basic observations are as follows:

- Significant packet loss indicates that the video data (along with other traffic in the network) are overwhelming a network router somewhere between the sender and the receiver.
- A small amount of packet loss indicates that the network is operating near capacity and sometime over, which causes the small amount of packet loss.
- No packet loss, while good for the application, may indicate that

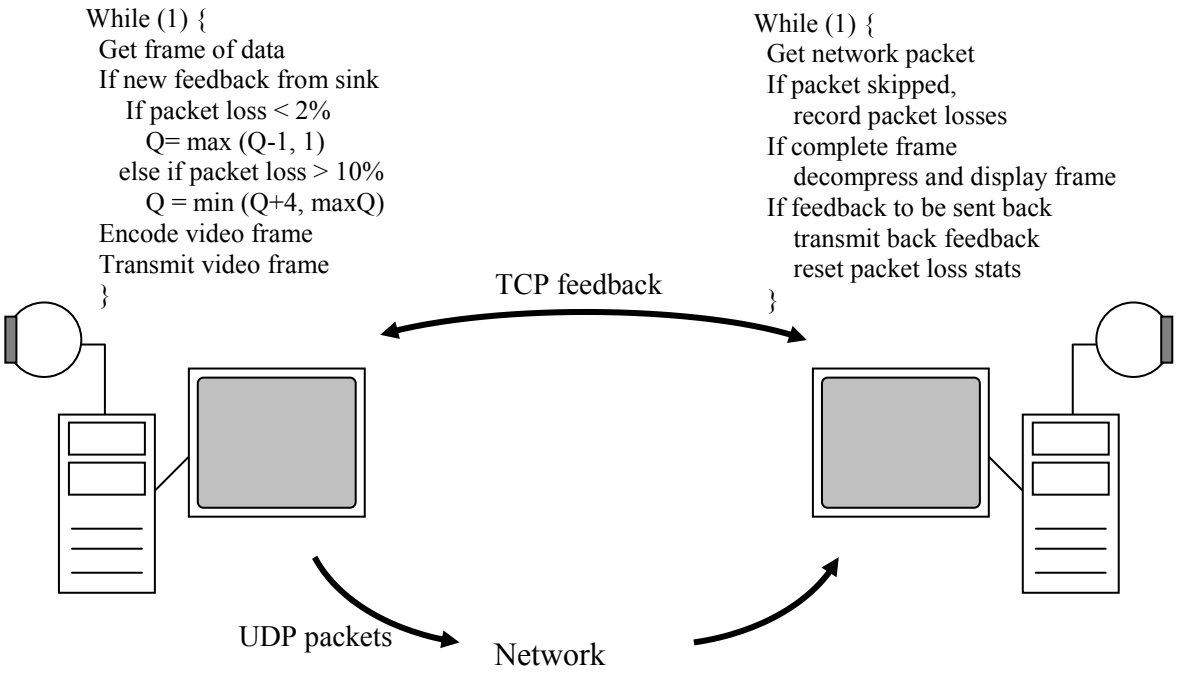
From this set of bullet points, we see that no packet loss indicates that we could be delivering a higher quality video stream to the user while a larger amount of packet loss indicates that we're overwhelming the network. Thus, our solution is in the middle: targeting a small amount of packet loss. For most interactive systems, the loss rate is typically targeted to approximately 5% to 10%.

One last question remains, however, what is the target bit-rate to target? From this perspective, we can take a lesson from the congestion control algorithms from TCP. If the measured packet loss is greater than 10%, the quantization value can be increased a larger amount in order to quickly reduce the congestion. If the measured packet loss is less than a few percent, than the quantization value can be slowly decreased in order to slowly increase the quality of the video. The result is that the video quality is reduced quickly in order to respond to a congestion event and slowly increased in order to get close to the maximum quality video attainable.

Some algorithms that have been implemented do both incremental increase and decrease of the quantization value. The idea here is to keep the video quality more constant over time.

8.2.5 A Prototype Interactive Streaming Application

We now have the necessary components to build a complete interactive video conferencing system. As shown in the figure below, we have diagramed a simple streaming application.



At the source (left), the algorithm captures, compresses, and transmits video to the client. In the middle, the source periodically receives feedback from the receiver of the data. It updates the quantization value accordingly. Here, we've implemented a fairly primitive increase and decrease algorithm. Note that it is linear (and not multiplicative) in this example. On the receiver side, the receiver receives packets. If there are sufficient packets to complete a frame, the frame is decompressed and displayed to the user. The algorithm will also keep track of the packet losses. Every so often (e.g. 1 second), the receiver sends the packet loss statistics back to the sender so that it can update its quantization algorithm.

To implement the full system, a parallel system for the video going the other way is implemented.

8.3 Bringing Interactive Streaming Sessions Together

We have now designed a simple adaptive, interactive video streaming system. One question remains though. How do clients connect to each other without having to call on the phone to set the system parameters up accordingly? Obviously, a standard protocol by which to interoperate is necessary. For example, http is *the* protocol for exchanging web information.

To bring interactive video conferencing systems together, there are two main *umbrella* standards: H.323 and the Session Initiation Protocol (SIP).

8.3.1 H.323

H.323 is the most widely deployed protocol for video conferencing and IP telephony. H.323 was developed by the ITU in 1996. Its main purpose is to provide a session layer protocol that allows end user machines to exchange the necessary information to negotiate

parameters for the video conference. This allows a diversity of machines to talk with each other to determine what the setup is on the other side and to find a common way to communicate. In order to accomplish this, H.323 has a number of mandatory and optional components. Any H.323-compliant end station must have an H.261 video compressor and decompressor as well as a G.711 encoder and decoder for the audio. H.323 also specifies a number of optional components such as H.263 and H.264. If two stations have newer H.264 codecs available, then the session startup protocol will determine the availability and commonality of the codecs between the machines and take advantage of them. It is important to note that H.323 is *not* a video or audio compression standard; it relies on the fact that audio and video formats such as G.711, H.261, etc exist to do the actual transmission.

In addition to the video specific setup, H.323 specifies a number of components that can be implemented.

- Terminals: Terminals are the end devices that the users use for the interactive session. Terminals can include voice only telephones, video-capable phones, standalone video conferencing units (e.g. Polycom's), and PCs.
- Multi-point Control Units: Multi-point control units (MCUs) are expensive devices that bring together multiple sessions together, allowing more than two people to participate in the interactive session. An MCU is analogous to a bridge in the telephony world. MCUs are expensive devices because they need to bring a number of compressed streams together and render a new video stream that has all the participants in a single video, all in real-time!
- Gateways: Gateways are specified to allow an H.323 compliant system to talk with a non-H.323 device. For example, suppose a user only has access to a motion-JPEG codec. The gateway would be responsible for re-coding the video so that the two-sides have video that they can interpret. For video, this means translating motion JPEG to H.261 video in one direction and converting H.261 to motion JPEG in the other.
- Gatekeepers: Gatekeepers provide session layer services such as address translation and network access control for the terminals, MCUs, and gateways. They can also provide services such a bandwidth management and accounting.

8.3.2 Session Initiation Protocol

The session initiation protocol (SIP) is a peer-to-peer architecture to support multimedia conferencing over IP-based networks. SIP's primary purpose is the establishment, maintenance, and termination of calls between two or more hosts. SIP was designed to be an application-layer text-based protocol (message format similar to HTTP). SIP provides a number of capabilities including:

- Determining the location of the target host
- Determining the audio / video capabilities of the target host
- Determine the availability of the target host
- Establish a session between the source and target host
- Handle the transfer and termination of calls.

Much like H.323, SIP is meant to be an umbrella protocol and is defined in the IETF RFC 2543. Like H.323, SIP specifies a number of components: terminals and gateways. Their

functions are the same as in H.323's terminals and gateways. In the establishment of sessions, modern SIP-based audio / video applications use SDP (the session description protocol) to describe streaming media initialization parameters such as media type, bandwidth, encryption and other parameters relevant to the media stream.

8.4 Video Conferencing Case Studies

We conclude this chapter with a discussion of implementations of video conferencing systems in the real-world. Two of the more popular interactive video conferencing systems include Microsoft's NetMeeting and Skype.

8.4.1 Microsoft NetMeeting

Microsoft NetMeeting was introduced by the Microsoft Corporation as an add-on to its Windows 95 operating system with audio capability. In 1998, NetMeeting was introduced with video calling capability. NetMeeting leveraged the, then new, H.323 standard for audio / video connections and the ITU T.120 standard for data sharing (which included a shared whiteboard, application sharing, and desktop sharing). As an H.323 compliant application, NetMeeting used G.711 or G723.1 for the audio format and H.261 and H.263 for video compression. Because of the design towards interoperability, NetMeeting could dial into multipoint control units (MCUs) that were able to mosaic several H.323 clients into one screen as well as dial stand alone devices such as Polycom video conferencing system. The figure below shows an example of an MCU's output (left) and a picture of a Polycom system (right)[5].

Thus, the big advantage this brought to Microsoft was the ability to interoperate with any H.323 device. While H.323 and SIP systems are still used today in enterprise video conferencing, it failed to gain traction for the common consumer. There were two primary reasons for this: directory services and the ability traverse NATs.

H.323 systems required user to enter the destination machine or IP address, and thus, was missing an easy-to-use directory service to connect users. Microsoft tried to manage this by running an Internet Lookup Service for users to find other user. In 1999, Microsoft switched over to MSN Messenger Instant Messenger for its people lookup. Thus, by 2000, Microsoft

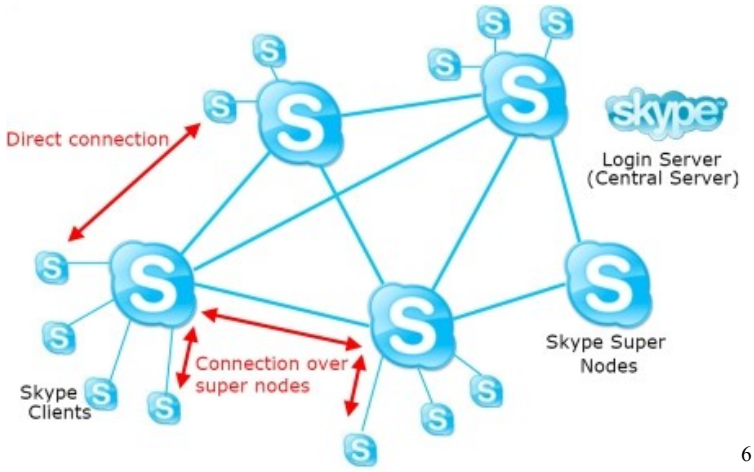
[5] http://telepresenceoptions.com/2010/06/telepresence_and_visual_collab_1/
http://ciscorussia.files.wordpress.com/2011/08/polycom_hdx90001.jpg

had all the advantages of interoperability by using H.323 and had a directory service. Unfortunately, the ability to configure NetMeeting to work with NAT devices limited its acceptance and use in the consumer marketplace.

8.5 Skype

Skype was introduced by the writers of the Kazaa peer-to-peer file sharing applications in 2003. Skype solved the two main shortcomings of NetMeeting. In particular, it provided a way for Skype clients to login to the Skype directory service, letting the user connect easily to other Skype clients that users had put in their contact lists. Furthermore, Skype worked seamless across NAT devices.

Skype uses a proprietary set of protocols, and thus, will not interoperate with any other systems other than Skype clients. For audio, Skype uses ITU G.729 or one of several Skype proprietary codecs. For video, Skype uses VP7 or VP8 video, proprietary video compression formats from On2 Technologies. Skype consists of three main components: Skype client applications, the Skype Directory service, and Skype Supernodes. The relation between these components is shown in the figure below:



6

Login into Skype: Upon starting the Skype application, users authenticated themselves to the central (replicated) login server (shown on the right). With the Skype applications connected to the Login Servers, users then were able to initiate calls to other Skype clients.

Initiating calls: Once a Skype user initiates a call, the Skype central server determines the reachability of the Skype clients. As the actual calls within Skype are peer-to peer (i.e., not through the central Skype server), one of the hosts needs to be able to reach the other. If it is possible, the Skype clients start the call with a direct connection (as shown with the "Direct connection" line in the figure above. If they are behind NATs and not reachable, Skype creates the call by connecting the clients through Skype Supernodes. This is primary mechanism by which Skype clients are able to set up a call when behind NAT devices.

The primary drawback of Skype from a security standpoint is the use of supernodes. While the supernodes provide a mechanism by which clients behind NATs can connect to each other, all the video data for those sessions are routed through the supernode. As a result, the

[6] http://watchfriendsonline.info/www.letsbytecode.com/wp-content/uploads/2011/05/skype4.jpg

supernode has access to all audio and video data. Unfortunately, it has been purported that Skype used any node with IP address reachability, good bandwidth, and running Skype to serve as supernodes. Thus, it was unclear who was actually eavesdropping onto unsuspecting Skype users.

8.6 References

Guide to Cisco Systems' VoIP Infrastructure Solution, 2000, http://www.cisco.com