# Chapter 10. Congestion Control

* The Need for Congestion Control
* Congestion Control on the Internet
* Implications for Multimedia
* Congestion Control for Multimedia

Until now we have assumed that it is possible to send a media stream at its natural rate, and that attempts to exceed the natural rate result in packet loss that we must conceal and correct. In the real world, however, our multimedia stream most likely shares the network with other traffic. We must consider the effects we are having on that traffic, and how to be good network citizens.

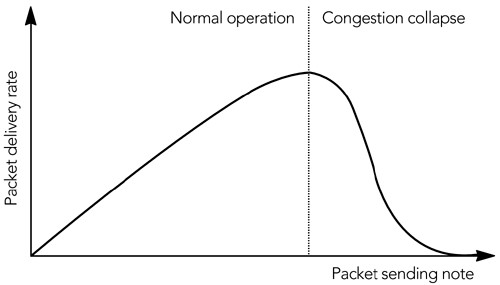
## The Need for Congestion Control

Before we discuss the congestion control mechanisms employed on the Internet, and the implications they have for multimedia traffic, it is important to understand what congestion control is, why it is needed, and what happens if applications are not congestion-controlled.

As we discussed in HTUChapter 2UTH, Voice and Video Communication over Packet Networks, an IP network provides a best-effort packet-switched service. One of the defining characteristics of such a network is that there is no admission control. The network layer accepts all packets and makes its best effort to deliver them. However, it makes no guarantees of delivery, and it will discard excess packets if links become congested. This works well, provided that the higher-layer protocols note packet drops and reduce their sending rate when congestion occurs. If they do not, there is a potential for congestion collapse of the network.

T*Congestion collapse*T is defined as the situation in which an increase in the network load results in a decrease in the number of packets delivered usefully by the network. HTUFigure 10.1UTH shows the effect, with a sudden drop in delivery rate occurring when the congestion collapse threshold is exceeded. Congestion collapse arises when capacity is wasted sending packets through the network that are dropped before they reach their destination—for example, because of congestion at intermediate nodes.

### **Figure 10.1. Congestion Collapse**



Consider a source sending a packet that is dropped because of congestion. That source then retransmits the packet, which again is dropped. If the source continues to resend the packet, and the other network flows remain stable, the process can continue with the network being fully loaded yet no useful data being delivered. This is congestion collapse. It can be avoided if sources detect the onset of congestion and use it as a signal to reduce their sending rate, allowing the congestion to ease.

Congestion collapse is not only a theoretical problem. In the early days of the Internet, TCP did not have an effective congestion control algorithm, and the result was networkwide congestion collapse on several occasions in the mid-1980s.HTPU3UTPH To prevent such collapse, the TCP protocol was augmented with the mechanisms developed by Van Jacobson,HTPU81UTPH which are described in the next section, HTUCongestion Control on the InternetUTH.

With the increase in non-congestion-controlled multimedia traffic, the potential for congestion collapse is again becoming a concern. Various mechanisms have been proposed by which routers can detect flows that do not respond to congestionHTPU73UTPH and penalize those flows with higher loss rates than those that are responsive. Although these mechanisms are not widely deployed at present, it is likely that in the future the network will be more actively policed for compliance with the principles of congestion control.

In addition to congestion collapse, there is another reason for employing congestion control: fairness. The policy for sharing network capacity among flows is not defined by IP, but by the congestion control algorithms of the transport protocols that run above it. The model adopted by TCP is to share the available capacity approximately evenly among all flows. Multimedia flows that use different congestion control algorithms upset this fair sharing of resources, usually to the detriment of TCP (which is squeezed out by the more aggressive multimedia traffic).

Although it may be desirable to give multimedia traffic priority in some cases, it is not clear that multimedia traffic is always more important than TCP traffic, and that it should always get a larger share of the network capacity. Mechanisms such as

Differentiated ServicesHTPU23UTPHP,HPTPU24UTPH and Integrated Services/RSVPHTPU11UTPH allow for some flows to be given priority in a controlled way, based on application needs, rather than in an undifferentiated manner based on the transport protocol they employ. However, these service prioritization mechanisms are not widely deployed on the public Internet.

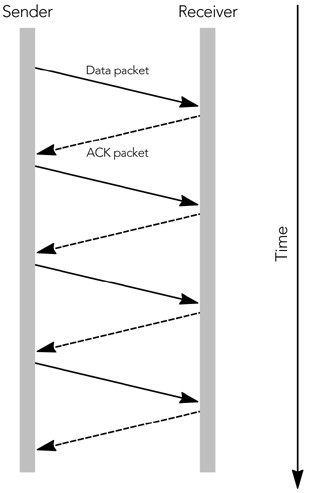
## Congestion Control on the Internet

Congestion control on the Internet is implemented by the transport protocols layered above IP. It is a simple matter to explain the congestion control functions of UDP—there are none unless they are implemented by the applications layered above UDP—but TCP has an extensive set of algorithms for congestion

control.HPU29UPHP,HU81UPHP,HU112UPH

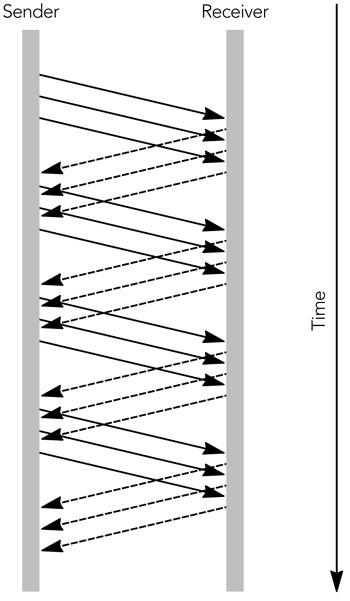
TCP is an example of a sliding window protocol. The source includes sequence numbers with each data packet it sends, and these are echoed back to it in ACK (positive acknowledgment) packets from the receiver. The simplest sliding window protocol requires each packet to be acknowledged immediately, before the next can be sent, as shown in HUFigure 10.2UH. This is known as a *stop-and-wait protocol* because the sender must wait for the acknowledgment before it can send the next packet. Clearly, a stop-and-wait protocol provides flow control—preventing the sender from overrunning the receiver—but it also hinders performance, especially if the round-trip time between sender and receiver is long.

### **Figure 10.2. A Simple Stop-and-Wait Protocol**



Use of a larger window allows more than one packet to be sent before an ACK is received. Each TCP ACK packet includes a receiver window, indicating to the source how many octets of data can be accepted at any time, and the highest continuous sequence number from the data packets. The source is allowed to send enough packets to fill the receive window, before it receives an ACK. As ACKs come in, the receive window slides along, allowing more data to be sent. This process is shown in HUFigure 10.3UH, in which the window allows three outstanding packets. The larger receive window improves performance compared to a simple stop-and-wait protocol.

### **Figure 10.3. Use of a Sliding Receiver Window**

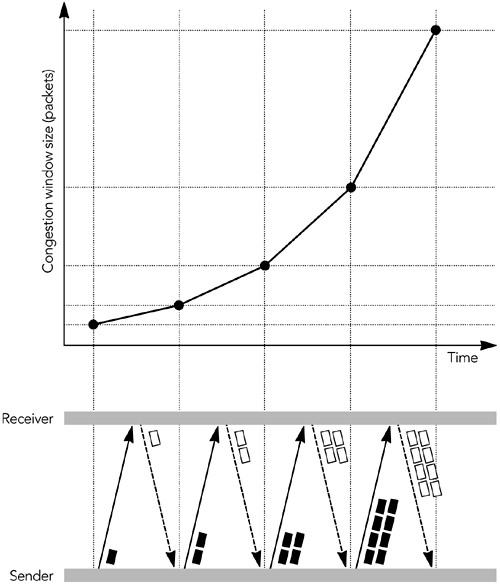


In addition to the receive window, TCP senders implement a congestion window. The congestion window is sized according to an estimate of the network capacity, and it prevents the sender from overrunning the network (that is, it provides congestion control). At any time, the sender can send enough packets to fill the minimum of the congestion window and the receive window.

The congestion window starts at the size of one packet. It increases according to either the slow-start algorithm or the congestion avoidance algorithm, as long as no loss occurs. New connections initially use slow start and then transition to congestion avoidance.

In slow-start mode, the congestion window increases by the size of a packet with each ACK received. As a result, the sender gradually builds up to the full rate the network can handle, as shown in HUFigure 10.4UH. Slow start continues until a packet is lost, or until the slow-start threshold is exceeded.

### **Figure 10.4. TCP Slow Start**



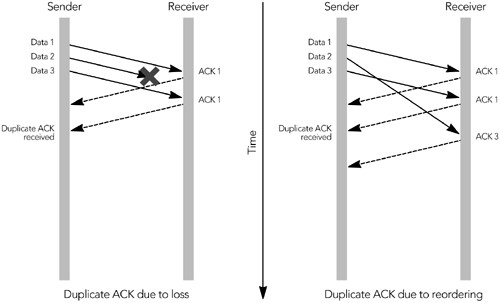
In congestion avoidance mode, the congestion window is increased by the size of one packet per round-trip time (instead of one per ACK). The result is linear growth of the congestion window, an additive increase in the sending rate.

TCP connections increase their sending rate according to one of these two algorithms until a packet is lost. Loss of a packet indicates that the network capacity has been reached and momentary congestion has occurred. The sender can detect congestion in two ways: by a timeout or by receipt of a triple duplicate ACK.

If no ACK is received for an extended time after data has been sent, data is assumed to have been lost. This is a timeout situation: it can occur when the last packet in the receive window is lost, or if there is a (temporary) failure that prevents packets from reaching their destination. When a timeout occurs, the sender sets the slow-start threshold to be one-half the number of packets in flight, or two packets, whichever is larger. It then sets the congestion window to the size of one packet, and enters slow start. The slow-start process continues until the slow-start threshold is exceeded, when the sender enters congestion avoidance mode. Prolonged timeout will cause the sender to give up, breaking the connection.

The other way a sender can detect congestion is by the presence of duplicate ACK packets, which occur when a data packet is lost or reordered (see HUFigure 10.5UH). The ACK packets contain the highest continuous sequence number received, so if a data packet is lost, the following ACK packets will contain the sequence number before the loss, until the missing packet is retransmitted. A duplicate ACK will also be generated if a packet is reordered, but in this case the ACK sequence returns to normal when the reordered packet finally arrives.

### **Figure 10.5. Generation of Duplicate ACK Packets**

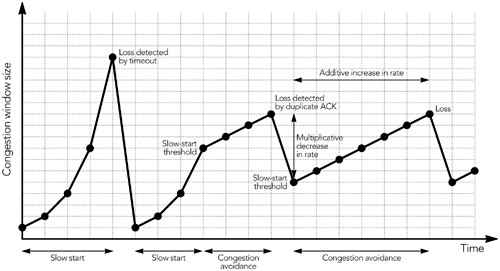


If the sender receives three duplicate ACK packets, it assumes that a packet was lost because of congestion. The sender responds by setting its congestion window and slow-start threshold to one-half the number of packets in flight, or two packets, whichever is larger. It then retransmits the lost packet and enters congestion avoidance mode.

The combination of these algorithms gives TCP connection throughput as shown in

HUFigure 10.6UH. The characteristic is an additive-increase, multiplicative-decrease (AIMD) sawtooth pattern, with large variations in throughput over short time intervals.

### **Figure 10.6. Variation in the TCP Sending Rate**



This rapid variation in throughput is actually the key behavior that makes a system using TCP stable. The multiplicative decrease ensures a rapid response to congestion, preventing congestion collapse; additive increase, probing for the maximum possible throughput, ensures that the network capacity is fully utilized.

There have been many studies of the variation in TCP throughput, and of the

behavior of competing TCP flows.HPU73UPHP,HU81UPHP,HU94UPHP,HU95UPHP,HU124UPH These studies have shown that competing flows get approximately equal shares of the capacity on average, although the jagged throughput profile means that their instantaneous share of the bandwidth is unlikely to be fair.

There is one systematic unfairness in TCP: Because the algorithm responds to feedback, connections with lower round-trip times return feedback faster and therefore work better. Thus, connections with longer network round-trip time systematically receive a lower average share of the network capacity.

### **Implications for Multimedia**

As noted in HTUChapter 2UTH, Voice and Video Communication over Packet Networks, the vast majority of traffic—over 95% of octets transported—uses TCP as its transport protocol. Because it is desirable for multimedia traffic to coexist peacefully with other traffic on the network, the multimedia traffic should employ a congestion control algorithm that is fair to that of TCP.

As we have seen, TCP adapts to network congestion on very short timescales, and it gives a somewhat higher share of the network bandwidth to connections with short network round-trip times. Also the evolution of TCP has involved several variants in the congestion control algorithms, each having slightly different behavior. These factors make it difficult to define fairness for TCP: Over what timescale is fairness measured—instantaneous or long-term average? Is it a problem that long-distance connections get less bandwidth than local connections? What about changes in the protocol that affect behavior in some conditions?

The more studies that are undertaken, the more it becomes clear that TCP is not entirely fair. Over the short term, one flow will always win out over another. Over the long term, these variations mostly average out, except that connections with longer round-trip times generally achieve lower throughput on average. The different variants of TCP also have an effect—for example, SACK-TCP gets better throughput with certain loss patterns. At best, TCP is fair within a factor of 2 or 3, over the long term.

The variation in TCP behavior actually makes our job, as designers of congestion control for multimedia applications, easier. We don't have to worry too much about being exactly fair to any particular type of TCP, because TCP is not exactly fair to itself. We have to implement some form of congestion control—to avoid the danger of congestion collapse—but as long as it is approximately fair to TCP, that's the best that can be expected.

Some have argued that a multimedia application doesn't want to be fair to TCP, and that it is acceptable for such traffic to take more than its fair share of the bandwidth. After all, multimedia traffic has strict timing requirements that are not shared by the more elastic TCP flows. To some extent this argument has merit, but it is important to understand the problems that can arise from such unfairness.

For example, consider a user browsing the Web while listening to an online radio station. In many cases it might be argued that the correct behavior is for the streaming audio to be more aggressive than the TCP flow so that it steals capacity from the Web browser. The result is music that does not skip, but less responsive Web downloads. In many cases this is the desired behavior. Similarly, a user sending e-mail while making a voice-over-IP telephone call usually prefers having the e-mail delivered more slowly, rather than having the audio break up.

This prioritization of multimedia traffic over TCP traffic cannot be guaranteed, though. Perhaps the Web page the user is submitting is a bid in an online auction, or a stock-trading request for which it's vital that the request be acted on immediately. In these cases, users may be less than pleased if their online radio station delays the TCP traffic.

**If your application needs higher priority than normal traffic gets, this requirement should be signaled to the network rather than being implied by a particular transport protocol. Such communication allows the network to make an intelligent choice between whether to permit or deny the higher priority on the basis of the available capacity and the user's preferences. Protocols such as RSVP/Integrated**

**Services**HTPU**11**UTPH **and Differentiated Services**HTPU**24**UTPH **provide signaling for this purpose. Support for these services is extremely limited at the time of this writing; they are available on some private networks but not the public Internet. Applications intended for the public Internet should consider some form of TCP-friendly congestion control.**

### **Congestion Control for Multimedia**

At the time of this writing, there are no standards for congestion control of audio/video streams on the Internet. It is possible either to use TCP directly or to emulate its behavior, as discussed in the next section, HTUTCP-Like Rate ControlUTH, although mimicking TCP has various problems in practice. There is also work in progress in the IETF to define a standard for TCP-friendly rate control (see the section titled HTUTCP-Friendly Rate ControlUTH) that will likely be more suitable for unicast multimedia applications. The state of the art for multicast congestion control is less clear, but the layered coding techniques discussed later in this chapter have, perhaps, the most promise.

#### TCP-Like Rate Control

The obvious congestion control technique for audio/video applications is either to use TCP or to emulate the TCP congestion control algorithm.

As discussed in HTUChapter 2UTH, Voice and Video Communication over Packet Networks, TCP has several properties that make it unsuitable for real-time applications, in particular the emphasis on reliability over timeliness. Nevertheless, some multimedia applications do use TCP, and an RTP-over-TCP encapsulation is defined for use with RTSP (Real-Time Streaming Protocol).HTPU14UTPH

Instead of using TCP directly, it might also be possible to emulate the congestion control algorithm of TCP without the reliability mechanisms. Although no standards exist yet, there have been several attempts to produce such a protocol, with perhaps the most complete being the Rate Adaptation Protocol (RAP), by Rejaie et al.HTPU99UTPH Much like TCP, a RAP source sends data packets containing sequence numbers, which are acknowledged by the receiver. Using the acknowledgment feedback from the receiver, a sender can detect loss and maintain a smoothed average of the round-trip time.

A RAP sender adjusts its transmission rate using an additive-increase, multiplicative-decrease (AIMD) algorithm, in much the same manner as a TCP sender, although since it is rate-based, it exhibits somewhat smoother variation than TCP. Unlike TCP, the congestion control in RAP is separate from the reliability mechanisms. When loss is detected, a RAP sender must reduce its transmission rate but is under no obligation to resend the lost packet. Indeed, the most likely response would be to adapt the codec output to match the new rate and continue without recovering the lost data.

Protocols, like RAP, that emulate—to some degree—the behavior of TCP congestion control exhibit behavior that is most fair to existing traffic. They also give an application more flexibility than it would have with standard TCP, allowing it to send data in any order or format desired, rather than being stuck with the reliable, in-order delivery provided by TCP.

The downside of using TCP, or a TCP-like protocol, is that the application has to adapt its sending rate rapidly, to match the rate of adaptation of TCP traffic. It also has to follow the AIMD model of TCP, with the sudden rate changes that that implies. This is problematic for most audio/video applications because few codecs can adapt quickly and over such large ranges, and because rapid changes in picture or sound quality have been found to be disturbing to the viewer.

These problems do not necessarily mean that TCP, or TCP-like, behavior is inappropriate for all audio/video applications, merely that care must be taken to determine its applicability. The main problem with these congestion control algorithms is the rapid rate changes that are implied. To some extent you can insulate the application from these changes by buffering the output, hiding the short-term variation in rate, and feeding back a smoothed average rate to the codec. This can work well for noninteractive applications, which can tolerate the increased end-to-end delay implied by the buffering, but it is not suitable for interactive use.

Ongoing research into protocols combines TCP-like congestion control with unreliable delivery. If one of these is found suitable for use with RTP, it is expected to be possible to extend RTP to support the necessary feedback (using, for example, the RTCP extensions described in HTUChapter 9UTH, Error CorrectionHTPU44UTPH). The difficulty remains in the design of a suitable congestion control algorithm.

At the time of this writing, none of these new protocols are complete. Applications that want to use TCP-like congestion control are probably best suited to the direct use of TCP.

#### TCP-Friendly Rate Control

The main problem that makes TCP, or TCP-like, congestion control unsuitable for interactive audio/video transport is the large rate changes that can occur over short periods. Many audio codecs are nonadaptive and operate at a single fixed rate (for example, GSM, G.711), or can adapt only between a fixed set of rates (for example, AMR). Video codecs generally have more scope for rate adaptation because both the frame rate and the compression ratio can be adjusted, but the rate at which they can adapt is often low. Even when the media codec can adapt rapidly, it is unclear that doing so is necessarily appropriate: Studies have shown that users prefer stable quality, even if the variable-quality stream has a higher average quality.

Various T*TCP-friendly*T rate control algorithms have been devised that attempt to smooth the short-term variation in sending rate,HTPU72UTPHP,HPTPU125UTPH resulting in an algorithm more suitable for audio/video applications. These algorithms achieve fairness with TCP when averaged over intervals of several seconds but are potentially unfair in the short term. They have considerable potential for use with unicast audio/video applications, and there is work in progress in the IETF to define a standard

mechanism.HTPU78UTPH

TCP-friendly rate control is based on emulation of the steady-state response function for TCP, derived by Padhye et al.HTPU94UTPH The response function is a mathematical model for the throughput of a TCP connection, a predication of the average throughput, given the loss rate and round-trip time of the network. The derivation of the response function is somewhat complex, but Padhye has shown that the average throughput of a TCP connection, T*T*T, under steady conditions can be modeled in this way:



In this formula, T*s*T is the packet size in octets, T*R*T is the round-trip time between sender and receiver in seconds, T*p*T is the loss event rate (which is not quite the same as the fraction of packets lost; see the following discussion), and T*T*TBrtoB is the TCP retransmit time out in seconds.

This equation looks complex, but the parameters are relatively simple to measure. An RTP-based application knows the size of the data packets it is sending, the round-trip time may be obtained from the information in RTCP SR and RR packets, and an approximation of the loss event rate is reported in RTCP RR packets. This leaves only the TCP retransmit timeout, T*T*TBrtoB, for which a satisfactory approximationHTPU78UTPH is four times the round-trip time, T*T*TBrtoB = 4T*R*T.

Having measured these parameters, a sender can calculate the average throughput that a TCP connection would achieve over a similar network path, in the steady state—that is, the throughout averaged over several seconds, assuming that the loss rate is constant. This data can then be used as part of a congestion control scheme. If the application is sending at a rate higher than that calculated for TCP, it should reduce the rate of transmission to match the calculated value, or it risks congesting the network. If it is sending at a lower rate, it may increase its rate to match the rate that TCP would achieve. The application operates a feedback loop: change the transmission rate, measure the loss event rate, change the transmission rate to match, measure the loss event rate, repeat. For applications using RTP, this feedback loop can be driven by the arrival of RTCP reception report packets. These reports cause the application to reevaluate and possibly change its sending rate, which will have an effect measured in the next reception report.

For example, if the reported round-trip time is 100 milliseconds, the application is sending PCM µ-law audio with 20-millisecond packets (T*s*T = 200, including RTP/UDP/IP headers), and the loss event rate is 1% (T*p*T = 0.01), the TCP equivalent throughput will be T*T*T = 22,466 octets per second (21.9 Kbps).

Because this is less than the actual data rate of a 64-kilobit PCM audio stream, the sender knows that it is causing congestion and must reduce its transmission rate. It can do this by switching to a lower-rate codec—for example, GSM.

This seems to be a simple matter, but in practice there are issues to be resolved. The most critical matter is how the loss rate is measured and averaged, but there are secondary issues with packet sizing, slow start, and noncontinuous transmission:

• The most important issue is the algorithm for calculating the loss event rate, to be fed back to the sender. RTP applications do not directly measure the loss event rate, but instead count the number of packets lost over each RTCP reporting interval and include that number in the RTCP reception report packets as a loss fraction. It is not clear that this is the correct measurement for a TCP-friendly rate control algorithm.

The loss fraction may not be a sufficient measurement for two reasons: First, TCP responds primarily to loss events, rather than to the actual number of lost packets, with most implementations halving their congestion window only once in response to any number of losses within a single round-trip time. A measure of loss events that treats multiple consecutive lost packets within a round-trip time as a single event, rather than counting individual lost packets, should compete more equally with TCP. The loss event rate could be reported via an extension to RTCP receiver reports, but there is no standard solution at present.

The second reason is that the reported loss fraction during any particular interval is not necessarily a reflection of the underlying loss rate, because there can be sudden changes in the loss fraction as a result of unrepresentative bursts of loss. This is a problem because oscillatory behavior can result if a sender uses the loss fraction to compute its sending rate directly. To some extent, such behavior is unavoidable—AIMD algorithms are inherently oscillatoryHTPU68UTPH—but the oscillations should be reduced as much as possible.

The solution for reducing oscillations is to average loss reports over a particular time period, but there is no clear consensus among researchers on the correct averaging algorithm. Sisalem and Schulzrinne suggested the use of an exponentially weighted moving average of the loss fraction, which is better than the raw loss rate but can still give rise to oscillations in some cases.HTPU125UTPH Handley et al. suggest the use of a weighted average of the loss event rate, modified to exclude the most recent loss event unless that would increase the average loss event rate.HTPU72UTPHP,HPTPU78UTPH This modified weighted average prevents isolated, unrepresentative loss from corrupting the loss estimate and hence reduces the chance of oscillations.

Although it is not optimal, it may be sufficient to approximate the loss event rate using an exponentially weighted moving average of the loss fraction reported in RTCP reception reports. The goal is not to be exactly fair to TCP connections, but to be somewhat fair, not cause congestion, and still be usable for audio/video applications. A more complete implementation will extend RTCP reception reports, to return the loss event rate directly.

* The TCP-friendly rate control algorithm assumes that the packet size, T*s*T, is fixed while the transmission rate, T*R*T, is varied. A fixed T*s*T and varied T*R*T are easy to achieve with some codecs, but difficult with others. The effect on fairness of codec changes that affect both packet size and rate is the subject of ongoing research.

Likewise, the application probably is using a codec with a limited range of adaptation, and may be unable to send at the rate specified by the TCP-friendly algorithm. The safe solution is to send at the next lower possible rate; and if no lower rate is possible, the transmission may have to be stopped.

* As noted previously, TCP implements a slow-start algorithm when it starts transmission. This slow start allows a sender to probe the network capacity in a gradual manner, rather than starting with a burst that might cause congestion. Slow start is simple to implement for TCP because there is immediate feedback when loss occurs, but it is more difficult with a rate-based protocol that sends feedback at longer intervals.

The solution adapted for TCP-friendly rate control is for the receiver to send feedback on the reception rate once per round-trip time during slow start. The sender starts at a low rate (one packet per second is suggested) and doubles its transmission rate each time a feedback packet arrives, until the first loss is detected. After loss occurs, the system begins normal operation, starting from the rate used immediately before the loss. This gives a gradual increase up to a "reasonable" value, at which point the rate control algorithm takes over.

This solution can feasibly be adapted for use with RTP, with the receiver sending acknowledgments according to the extended RTCP feedback profile

(see HTUChapter 9UTH, Error Correction, and Ott et al. 2003HTPU44UTPH) and the sender doubling its rate once per round-trip time until loss occurs. The receiver then reverts to normal RTCP operation, with the sender following the TCP-friendly rate.

Some applications may not be able to perform this rate-doubling algorithm because of limitations in the set of codecs they support. Such applications might consider sending dummy data initially, and then switching to their most appropriate codec after the sustainable transmission rate is known.

• The final problem is that of discontinuous transmission. If a source stops sending for some time, no feedback on the correct rate will be received. The available capacity might change during the pause, perhaps if another source starts up, so the last-used transmission rate might not be appropriate when the sender resumes. Little can be done to solve this problem, except perhaps

to begin again from zero—or from another reduced rate—with the slow-start algorithm, until the correct rate has been determined.

If these problems can be solved, TCP-friendly rate control has the potential to become the standard approach for congestion control of unicast audio/video applications. It is strongly recommended that all unicast RTP implementations include some form of TCP-friendly congestion control.

Implementations should, at least, observe the loss fraction reported in RTCP RR packets, and compare their sending rate with the TCP-friendly rate derived from that loss fraction. If an implementation finds that it is sending significantly faster than the TCP-friendly rate, it should either switch to a lower-rate codec or cease transmission if a lower rate is not possible. These measures prevent congestion collapse and ensure correct functioning of the network.

Implementing the full TCP-friendly rate control algorithm will let an application optimize its transmission to match the network, giving the user the best possible quality. In the process, it will also be fair to other traffic, so as not to disrupt other applications that the user is running. If the application has a suitable codec, or set of codecs, it is strongly recommended that rate control be used—not just to reduce the rate in times of network congestion, but to allow an application to increase its quality when the network is lightly loaded.

#### Layered Coding

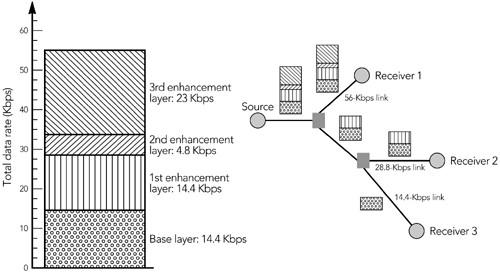
Multicast makes the problem of congestion control significantly more difficult: A sender is required to adapt its transmission to suit many receivers simultaneously, a requirement that seems impossible at first glance. The advantage of multicast is that it allows a sender to efficiently deliver identical data to a group of receivers, yet congestion control requires each receiver to get a media stream that is adapted to its particular network environment. The two requirements seem to be fundamentally at odds with each other.

The solution comes from layered coding, in which the sender splits its transmission across multiple multicast groups, and the receivers join only a subset of the available groups. The burden of congestion control is moved from the source, which is unable to satisfy the conflicting demands of each receiver, to the receivers that can adapt to their individual circumstances.HTPU86UTPH

Layered coding requires a media codec that can encode a signal into multiple layers that can be incrementally combined to provide progressively higher quality. A receiver that receives only the base layer will get a low-fidelity signal, and one that receives the base and one additional layer will get higher quality, with each additional layer increasing the fidelity of the received signal. With the exception of the base, layers are not usable on their own: They merely refine the signal provided by the sum of the lower layers.

The simplest use of layered coding gives each receiver a static subscription to one or more layers. For example, the sender could generate layers arranged as shown in HTUFigure 10.7UTH, in which the base layer corresponds to the capacity of a 14.4-Kbps modem, the combination of base layer and first enhancement layer matches that of a 28.8-Kbps modem, the combination of base and first two enhancement layers matches a 33.6-Kbps modem, and so on. Each layer is sent on a separate multicast group, with the receivers joining the appropriate set of groups so that they receive only the layers of interest. The multicast-capable routers within the network ensure that traffic flows only on links that lead to interested receivers, placing the burden of adaptation on the receivers and the network.

##### Figure 10.7. Layered Coding



A related solution involves the use of simulcast, in which the sender generates several complete media streams, adapted to different rates, and receivers join the single most appropriate group. This solution uses more bandwidth at the sender—the sum of the possible rates, rather than the highest possible rate—but it is simpler to implement. It doesn't solve the issues due to transient congestion, but it does provide a good solution to the rate selection problem.

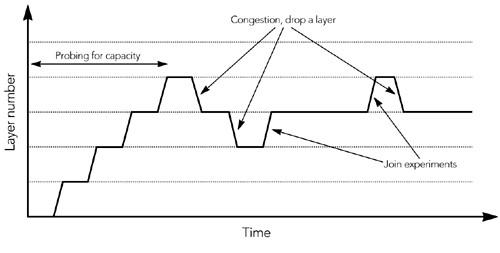
Although static assignment of layers solves the rate selection problem by adapting a media stream to serve many receivers, it doesn't respond to transient congestion due to cross-traffic. It is clear, though, that allowing receivers to dynamically change their layer subscription in response to congestion might provide a solution for multicast congestion control. The basic idea is for each receiver to run a simple control loop:

* If there is congestion, drop one or more layers.
* If there is spare capacity, add a layer.

If the layers are chosen appropriately, the receivers search for the optimal level of subscription, changing their received bandwidth in much the same way that a TCP source probes the network capacity during the slow-start phase. The receivers join layers until congestion is observed, then back off to a lower subscription level.

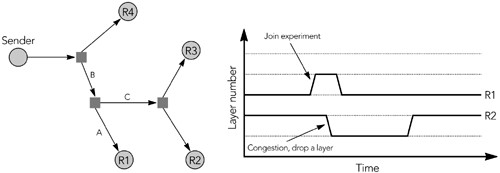
To drive the adaptation, receivers must determine whether they are at too high or too low a subscription level. It is easy to detect over-subscription because congestion will occur and the receiver will see packet loss. Undersubscription is harder to detect because there is no signal to indicate that the network can support a higher rate. Instead, a receiver must try to join an additional layer and immediately leave that layer if it causes congestion, a process known as a T*join experiment*T.HTPU86UTPH The result looks as shown in HTUFigure 10.8UTH, with the subscription level varying according to network congestion.

##### Figure 10.8. Adaptation by Varying Subscription Level



The difficulty with join experiments is in trying to achieve shared learning. Consider the network shown in HTUFigure 10.9UTH, in which receiver R1 performs a join experiment but R2 and R3 do not. If the bottleneck link between the source and R1 is link A, everything will work correctly. If the bottleneck is link B, however, a join experiment performed by R1 will cause R2 and R3 to see congestion because they share the capacity of the bottleneck link. If R2 and R3 do not know that R1 is performing a join experiment, they will treat congestion as a signal to drop a layer—which is not the desired outcome!

##### Figure 10.9. Difficulties with Join Experiments



There is a second problem also. If link C is the bottleneck and R2 leaves a layer, the traffic flowing through the bottleneck will not be affected unless R3 also leaves a layer. Because R2 still sees congestion, it will leave another layer, a process that will repeat until either R2 leaves the session in disgust or R3 also drops a layer.

The solution to both problems is to synchronize receiver join experiments. This synchronization can be achieved if each receiver notifies all others that it is about to join or leave a layer, but such notification is difficult to implement. A better solution

is for the sender to include synchronization points—specially marked packets—within the data stream, telling receivers when to perform join

experiments.HTPU104UTPH

Other issues relate to the operation of multicast routing. Although multicast joins are fast, processing of a leave request often takes some time. Receivers must allow time for processing of leave requests before they treat the continuing presence of congestion as a signal to leave additional layers. Furthermore, rapid joins or leaves can cause large amounts of routing-control traffic, and this may be problematic.

If these issues can be resolved, and with appropriate choice of bandwidth for each layer, it may be possible to achieve TCP-friendly congestion control with layered coding. The difficulty in applying this sort of congestion control to audio/video applications would then be in finding a codec that could generate cumulative layers with the appropriate bandwidth.

Layered coding is the most promising solution for multicast congestion control, allowing each receiver to choose an appropriate rate without burdening the sender. The Reliable Multicast Transport working group in the IETF is developing a standard for layered congestion control, and it is likely that this work will form the basis for a future congestion control standard for multicast audio/video.

## Summary

Congestion control is a necessary evil. As an application designer, you can ignore it, and your application may even appear to work better if you do. However, you may be affecting other traffic on the network—perhaps other instances of your own application—in adverse ways, and you have no way of knowing the relative importance of that traffic. For this reason, and to avoid the possibility of causing congestion collapse, applications should implement some form of congestion control. It is also desirable that the chosen algorithm be approximately fair to TCP traffic, allowing unfairness—priority—to be introduced into the network in a controlled manner.

The standards for congestion control are still evolving, so it is difficult to provide a detailed specification for implementers. Nonetheless, it is strongly recommended that unicast applications implement a TCP-friendly rate control algorithm, because these are the best developed. For multicast applications, the choice of congestion control is less clear, but layered coding seems to be the best alternative at present.