CHAPTER 8
Elements of Efficiency

Efficiency is a nebulous term. In general, it measures how thoroughly one manages to achieve some desired result as a function of the required resources. The biggest problems in implementing efficiency in a computer network are essentially matters of definition. What is the desired result for a network, and what resources are actually required?

With a relatively narrow view of the desired results in network design, it essentially comes down to the factors that I mentioned earlier when talking about network reliability. The network must deliver data to the destination. It must do so within the required application constraints. In most networks, the desired result is effectively quantified with just four parameters: latency, jitter, throughput, and dropped packets. I will define these terms when I come to talk about Quality of Service later in this chapter.

The hard part of describing efficiency is actually in defining the resources. Which resources should the definition include? Some resources are obvious. Everybody would agree that it's necessary to worry about CPU and memory utilization in their routers. The same is true for the bandwidth utilization on the various links that connect network devices. But some resources are harder to quantify or harder to see. How do you compare the relative importance of these resources? Do you want to save bandwidth, for example, at the expense of CPU load?

The ultimate resource for any organization comes down to money, and efficiency has to be defined for the entire organization. Suppose, for example, that you can save money by running a particular application on a particular type of server. The money you save has to be balanced against the extra money it costs to upgrade parts of the network. Perhaps the new implementation will turn out to be more expensive overall than the old way. But perhaps it will also allow the organization to win new business that will more than pay for the difference in cost. Doing such an upgrade is worthwhile for the organization. You just have to understand where you are drawing the line around what resources to include. Conversely, the network engineer may look at a Core router and see abnormally high CPU and memory utilization. If fixing

this problem means spending hundreds of thousands of dollars, the organization may not feel that the expense is justified.

Ultimately, efficiency is a matter of the global economics of the organization. This subject, however, is far beyond the scope of this book. The resources that I can reasonably talk about here are the ones that are specific to the network. I can discuss how to make the best use of the resources already in the network, so I look at the four parameters—latency, jitter, throughput, and dropped packets—that describe how well the network works. I also look at network resources such as bandwidth, CPU, and memory utilization. It is never possible to obtain perfect efficiency, though. Therefore, a network designer's most difficult job is to decide what tradeoffs will give the best design possible under the circumstances and will fit in best with larger corporate goals.

Using Equipment Features Effectively

There are usually several ways to configure any given piece of network equipment to achieve the same basic result. These different configurations usually do not use resources in the same way, and they often do not have exactly the same performance characteristics. For example, an inefficient configuration of a router might mean that it has to do too much processing on each packet. This extra processing increases the amount of time that each packet spends inside the router and probably also increases the memory utilization of the router, as it has to buffer large numbers of packets.

A typical example of this increase happens when an engineer fails to use special features of the equipment. For example, in many routers the ability to make most common routing decisions is delegated to logic circuits supporting the interface card. This approach works well because it means that the CPU is free to coordinate the activities of the different cards. The result is vastly improved net throughput; however, the advantage can be completely lost if this engineer implements a CPU-bound process that examines the contents of every packet.

For example, the engineer might turn on a feature that prioritizes or routes packets based on their contents. This sort of feature usually requires the CPU to examine the packet. So the packet cannot be processed solely by the interface. The same CPU loading happens if the router is configured to rewrite the contents of the packets, as in an address-translation feature systematically.

This has several implications for network design. Locally, it is important to ensure that each device does only what it needs to do and that it does so by the most efficient method. Globally, it means that network designers have to be careful about what network functions are performed where.

The local issue is really a matter for the network engineer who should sit down with the manuals for the network hardware and find the most efficient way to implement the functions of this device. If special optimizations are available, such as Cisco's

Fast Switching, then they should be used. Discussing the implementation details with the hardware vendor may also help, since the vendor should be aware of the features of the equipment.

The global issue, however, is for the network designer to resolve. A good example of this resolution comes up in prioritization and Quality of Service (QoS) implementation. The best place to decide on the priority of a packet is at the point where that packet enters the network. The entry-point router should examine the packet and mark its header with the appropriate priority value. Then, each subsequent device that handles the packet reads this priority value and treats the packet appropriately. The worst thing a designer can do is make every router look at the packet in detail and decide again what its priority should be.

Looking at a single byte in the header is easy for a router and can frequently be handled in highly optimized code in the hardware. However, looking at several bytes and making a decision about each packet takes a huge amount of extra resources. In most cases, it also increases the forwarding latency and reduces the number of packets that the router can handle.

Hop Counts

Efficiency means using the smallest amount of resources to accomplish the desired result. All other things being equal, heavily used network paths should have as few hops as possible.

Efficiency is actually a natural outcome of using a hierarchical network design. Building a network in a tree-like structure so that every leaf is no more than three hops away from the central Core of the network means that the greatest distance between any two leaves is six hops. Conversely, if the network has a relatively ad hoc structure, then the only effective upper limit to the number of hops between any two end points is the number of devices in the network.

Keeping hops counts low has several advantages. First, all routing protocols that were discussed in Chapters 6 and 7 (for IP and IPX, respectively) converge faster for lower hop counts. This convergence generally results in a more stable network. More specifically, it means that the network recovers more quickly after a failure. If a path is available to route traffic around the failure point, it can be found quickly and put into use.

Other advantages all have to do with the delivery of packets through the network. Every extra hop represents at least one additional queue and at least one additional link. Each additional queue introduces a random amount of latency. If the queue is relatively full, then the packet may spend extra time waiting to be processed. If the queue is realtively short, on the other hand, it may be processed immediately. If the network is extremely busy, the packet may be dropped rather than being kept until its data is no longer relevant.

This queuing delay means that every additional hop increases the net latency of the path. Latency is something that should be kept as low as possible in any network. Because this extra latency is random, the more hops that exist, the more variation there is in the latency.

This variation in latency is called *jitter*. Jitter is not a problem for bulk data transfers. But in any real-time applications such as audio or video, it is disastrous. These applications require that the time to deliver a packet from one point to another be as predictable as possible, or the resulting application will suffer from noticeable gaps. These gaps will appear as audible pops or skips and frozen or jumping video images.

Finally, there is the problem of what happens to a packet that passes through a highly congested device in the network. The device can do two things with a new packet entering its buffers. It can either put it into a queue to be forwarded at some future time, or it can decide that the queues are already too full and simply drop the packet. Clearly, the more hops in the path, the greater the probability of hitting one that is highly congested. Thus, a higher hop count means a greater chance of dropped packets.

This rule is true even if the network is rarely congested. A relatively short, random burst of data can temporarily exhaust the queues on a device at any time. Furthermore, the more hops there are in the path, the greater the probability of hitting a link that generates CRC errors. In most LAN media, the probability of CRC errors is relatively low, but this low probability is multiplied by the number of links in the path. Thus, the more hops there are the higher the probability that the packet will become corrupted and have to be dropped.

MTU Throughout the Network

The size of the largest data packet that can pass along any particular section of network is called the Maximum Transmission Unit (MTU). Suppose, for example, that a network contains both Ethernet and Token Ring segments. The default MTU for Ethernet is 1,500 bytes. For a 16Mbps Token Ring, the maximum MTU is 18,200 bytes. If a packet travels from one of these media to the other, the network will have to find a compromise.

There are two main ways to resolve MTU mismatch problems. The network can either fragment the large packets, or it can force everything to use the smaller value. In most cases, the network will fragment packets if it can and negotiate the greatest common MTU value only if it is not allowed to fragment. The efficiency issue is that both fragmentation and MTU negotiation consume network resources. However, fragmentation has to be done with every oversized packet, and MTU negotiation is done primarily during session establishment. MTU negotiation also happens if the path changes and the new path contains a leg with a lower MTU value.

In TCP sessions, the Path MTU Discovery process starts when a packet that has the Don't Fragment (DF) bit in the IP header set is sent. This bit literally instructs the network not to fragment the packet. Fragmentation is the default. Suppose a TCP packet passes through a network and it gets to a router that needs to break that packet to send it to the next hop on its path. If the DF bit in the IP header is not set, then the router simply breaks the packet into as many pieces as necessary and sends it along. When the fragments reach the ultimate destination, they are reassembled.

If the DF bit is set, then the router drops the packet and sends back a special ICMP packet explaining the situation. This packet tells the sender that the packet has been dropped because it could not be fragmented. It also tells the sender the largest packet it could have sent. Doing so allows the sender to shorten all future packets to this Path MTU value.

Note that it is more efficient in general to reassemble at the ultimate destination rather than at the other end of the link with a lower MTU. This is because it is possible that the packet will encounter another low MTU segment later in the path. Since there is significant overhead in both fragmentation and reassembly, if the network has to do it, it should do it only once.

Many protocols do not have a Path MTU Discovery mechanism. In particular, it is not possible to negotiate an end-to-end MTU for a UDP application. Thus, whenever a large UDP packet is sent through a network segment with a lower MTU value, it must be fragmented. Then the receiver has to carefully buffer and reassemble the pieces. However, most UDP applications deliberately keep their packets small to avoid fragmentation.

If the network is noisy or congested, it is possible to lose some fragments. This loss results in two efficiency problems. First, the device that reassembles the packet from the fragments must buffer the fragments and hold them in its memory until it decides it can no longer wait for the missing pieces. This is not only a resource issue on the device, but it also results in serious latency and jitter problems. The second problem can actually be more serious. If any fragment is lost, then the entire packet must be resent, including the fragments that were received properly. Data lost due to congestion problems will make the problem considerably worse.

Obviously, it is better if the network doesn't have to fragment packets. Thus, in a multiprotocol network it is often better to configure a common MTU manually throughout all end-device segments.

This configuration is not always practical for Token Ring segments that run IBM protocols. Suppose a tunneling protocol such as Data Link Switching (DLSw) connects two Token Ring segments through an Ethernet infrastructure. Generally, it is most efficient to use the greatest MTU possible. In this case, however, there is an important advantage. The DLSw protocol is TCP based and operates as a tunnel between two routers. These routers can discover a smaller Path MTU between them.

They can then simply hide the fragmentation and reassemble from the end devices. They will appear to pass full-sized Token Ring frames.

Even here, the routers suffer from additional memory utilization, and there will be latency and jitter issues on the end-to-end session. If at all possible, it is better to reduce the Token Ring MTU to match the lower Ethernet value.

Bottlenecks and Congestion

Avoiding bottlenecks in any large network is impossible, and it isn't always necessary or desirable to do so. One of the main efficiencies of scale in a large network is the ability to oversubscribe the Core links. Oversubscribing means that most designers deliberately aggregate more network segments than the network can support simultaneously. Then they hope that these segments don't all burst to their full capacity at once. This issue was discussed in Chapter 3 in the section, "Trunk capacity."

Just oversubscribing is not a problem. The network has a problem only when it cannot support the actual traffic flow. This is called congestion, and it results in increased latency and jitter if the application is lucky enough that the network can queue the packets. If it is not so lucky, the network has to drop packets.

A little bit of congestion is not a bad thing, provided it is handled gracefully. However, systematic congestion in which one or more network links cannot support typical traffic volumes is a serious issue. The network can handle intermittent congestion using the various QoS mechanisms discussed later in this chapter. For systematic congestion, however, the designer usually has to modify the network design to reduce the bottleneck.

By intermittent congestion, I mean congestion that never lasts very long. It is not uncommon for a link to fill up with traffic for short periods of time. This is particularly true when bursty applications use the network.

QoS mechanisms can readily handle short bursts of traffic. They can even handle longer periods of congestion when it is caused by low-priority applications such as file transfers. However, when a high volume of interactive traffic causes the congestion, it is usually considered a systematic problem. In general, QoS mechanisms are less expensive to implement than a redesign of the network, so it is usually best to try it first.

Another common method for handling intermittent congestion is using a Random Early Detection (RED) system on the router with the bottleneck. The RED algorithm deliberately drops some packets before the link is 100% congested. When the load rises above a certain predetermined threshold, the router begins to drop a few packets randomly in an attempt to coax the applications into backing off slightly. In this way, RED tries to avoid congestion before it becomes critical.

However, it is important to be careful about RED because not all applications and protocols respond well to it. It works very well in TCP applications, but in UDP applications, as well as Appletalk and IPX, RED does not achieve the desired results. These protocols cannot back off their sending rates in response to dropped packets.

There are essentially two different ways to handle a systematic and persistent congestion problem at a network bottleneck. You can either increase the bandwidth at the bottleneck point, or you can reroute the traffic so it doesn't all go through the same point.

Sometimes you can get a bottleneck because some redundant paths in a network are unused, forcing all of the traffic through a few congested links. Examining the link costs in the dynamic routing protocol can often provide a way to alleviate this problem.

In many protocols, such as OSPF, it is possible to specify the same cost for several different paths. This specification invokes equal-cost multipath routing. In some cases you may find that, despite equal costs, some of these paths are not used. This may be because the routers are configured to use only a small number (usually two) of these equal-cost paths simultaneously. Many routers offer the ability to increase this number. However, it is important to watch the router CPU and memory load if the number is increased because maintaining the additional paths may cause an additional strain on the device.

Ultimately, if a subtle rerouting cannot alleviate the bottleneck, it will be necessary to increase the bandwidth on the congested links. Doing so is not always easy. If the link is already the fastest available technology in this type, then you have to do something else.

Other options are usually available to you in these situations. You might migrate to a different high-speed link technology, such as ATM or 10 Gigabit Ethernet. Or, you may have the ability to *multiplex* several fast links together to make one super high-speed link. If even this is not possible, then it is probably best to start configuring new redundant paths through the network to share the load.

Filtering

One of the most important things a designer can do to improve how efficiently a network uses resources is to filter out ill-behaved or unwanted traffic. This is particularly true for chatty protocols that tend to transmit data that is not necessary. A good example of filtering for efficiency comes from IPX networking. In an IPX network, every device that has any sort of service to offer sends out Service Advertisement Packets (SAP). This information then circulates not just over the local segment, but throughout the entire network. Although unnecessary SAP information may not have a significant effect on the bandwidth used in the network, it can have a large impact

on the amount of memory that Core routers need to keep track of this information. Specifically, every printer sends at least one SAP; so does every Windows NT workstation.

In a large network, it is difficult enough to ensure that the SAP information regarding important servers is distributed correctly. If there are unneeded SAPs for every printer and workstation, then the amount of required memory can easily exceed the available resources. So this is a good example of the need for filtering. The first router that sees the unwanted SAP information simply discards it without passing it along. The information stays local to the LAN segment where it is used and does not use up key network resources.

Filtering can also restrict malicious, unwanted traffic. For example, some popular Internet-based attacks use certain types of ICMP or packets used in setting up TCP calls. If these packets are not eliminated, they may cause serious network problems. Thus, these specific types of packets can be filtered at the network boundaries.

I want to stress once again that connecting to an untrusted external network without a firewall is foolish. However, in some organizations, these same sorts of problems can arise either because of malicious employees or because of innocently executed malicious programs. In these cases, it may become necessary to filter the unwanted traffic at the user LAN segment level, just as I suggested eliminating unwanted IPX SAP information.

In many networks chatty little unnecessary applications (and network games!) can be easily filtered and prevented from crossing the network. The key is to remove the unwanted traffic as soon as possible. This usually means that the filtering should be applied at the edges of the network. If the network adjoins another network, then the border routers should perform this filtering before the unwanted traffic enters the network. Similarly, if the filtering is to restrict traffic from user LAN segments, then the best place to run the filter is on the default gateway routers for these segments.

Quality of Service and Traffic Shaping

As I mentioned before, four measurable parameters define how well a network performs: latency, jitter, bandwidth, and dropped packets.

Bandwidth is a term borrowed from transmission theory. If a signal has only one frequency that is broadcast perpetually, then it doesn't contain any information. If that carrier signal is modulated, then you can use it to carry a data signal. As soon as you do this, you introduce some fluctuation into the frequency of the carrier signal. Sometimes the carrier wave pulse comes a little bit earlier because of the modulation, and sometimes it comes a little late. If you draw a graph of the frequency, sometimes it's a little lower than the carrier, and sometimes it's a little higher. However, the average is equal to the carrier wave's frequency.

The width of this frequency curve is called the bandwidth, and it is a measure of how much information the signal carries. The width in frequencies is going to be a frequency itself, and frequencies are measured in Hz (cycles/s). If you can put one bit in a cycle of the wave, then it is exactly the same as bits per second. That's how the term originates. However, modern communications protocols use sophisticated compression and include extra overhead for error checking and so forth. Thus, using the term "bandwidth" to describe the throughput on a link is no longer accurate. The meanings of words migrate over time, and today people generally use the word bandwidth to mean the amount of data that a medium can transmit per unit time.

In any given network, several applications compete for the same bandwidth resources. Each application has a bandwidth requirement—a certain minimum amount of data that it has to send and receive. The network designer must balance these various requirements and find a way for the applications to all work well together. There are two ways to do this. You can either carve off a certain dedicated amount of bandwidth for each application, or you can make them share the total fairly. There are pros and cons to both approaches, as I describe next.

Latency is the amount of time it takes to get from one point in the network to another. Obviously, latency varies depending on the two points, how far apart they are, how many hops are between them, and the nature of the media. Three main factors affect latency: bandwidth, physical distance, and queuing time.

Bandwidth affects latency in a simple way. If the link can support a certain number of bits per second, then that is how many bits a device can inject per second into the link medium. It takes 10 times as long to inject a packet onto a 10Mbps Ethernet as it does onto a 100Mbps Fast Ethernet segment. There are exceptions to this rule for media that support carrying several bits at once in parallel. But parallel media are fairly uncommon.

Physical distance also affects latency in a simple way. The further the packet has to fly, the longer it takes. This time of flight component to latency is most relevant over WAN links, since it is governed by the speed of light in the transmission medium. The speed of light in fiber optic cable is governed by the refractive index of the medium. For glass, this index is about 1.5, so the speed of light in optical fiber is about 2/3 of its value in vacuum. In wire, the speed is somewhat slower than this, although the actual speed varies depending on the kind of cable. This effect may sound small, and usually it is for LAN- and campus-sized networks, but for WAN links it can be a large part of the total latency. The distance from New York to Los Angeles is about 4000 km. So the one-way time of flight for a signal through optical fiber is about 20 milliseconds. Signals sent around the world suffer significantly larger time-of-flight delays. Finally, queuing time introduces an additional random component to network latency, as I discussed earlier in the section "Hop Counts."

Jitter is the packet-by-packet variation in latency. Of the three components to latency that I just mentioned, only the queuing time is subject to change. So this is the main

factor in causing jitter. If the latency is changing very gradually, then it will not generally cause serious application problems. The most noticeable jitter issues happen when the latency of one packet is significantly different from the latency of the next packet following it in the same data stream. This is what causes skips, pops, and frozen frames in audio and video applications. So jitter is defined as the difference in latency between any two successive packets, as opposed to a general difference or standard deviation from the mean latency.

As I mentioned, the main cause of jitter is queuing. So devices need to be careful with how they handle queuing of data streams for jitter-sensitive applications. Basically, they should queue the packets as little as possible in these sensitive data streams.

Normally routers set up their queues so that whenever one of these jitter-sensitive packets arrives, it simply sends it to the front. Equivalently, they can give this application its own queue to ensure that other applications do not interfere. As long as the application doesn't send a sudden burst of packets to cause congestion within its own flow, the jitter should be minimal.

The final performance parameter is the number of *dropped packets*. Obviously the goal is to drop as few packets as possible. But there are times when the amount of data transmitted through the network is simply greater than what it can carry. Devices can only buffer for so long before they have to start throwing some data away.

Systematically dropping excess packets is also called *policing*. It is important that it be done fairly. Low-priority data should be dropped before high priority. But some high-priority data is extremely sensitive to jitter. In these cases, it may be better to drop the packet than to hold it in a buffer until it can be transmitted. Controlling how and when devices decide to drop packets is critical to maintaining any QoS criteria on a network.

QoS Basics

QoS implementations come in three functional flavors. Any real network implementing a QoS system generally uses more than one of these.

The first option is that the network can do nothing to discriminate between different applications. This is called Best Efforts Delivery. The second functional flavor is called Preferential Delivery. In this case, network devices define certain applications as more important than others and give them precedence whenever they encounter congestion. The final option, called Guaranteed Delivery, allows the network to reserve a guaranteed minimum bandwidth through the network for each important application.

In Best Efforts Service, packets are transmitted through the network if there is sufficient capacity. If congestion occurs along the path, then the packet may be dropped.

Note that Best Efforts is not necessarily the same thing as First In First Out (FIFO). FIFO is a queuing strategy in which the router deals with packets in the order in which they are received. There are several other possible queuing (sometimes called *scheduling*) algorithms. For example, many routers use Fair Queuing or Weighted Fair Queuing algorithms instead of FIFO.

Preferential Delivery requires the network engineer to make certain decisions about which applications are more important than others. For example, an FTP file transfer might be considered low priority, since it is effectively a batch-mode bulk transfer. An interactive business-critical application, on the other hand, would have a high priority.

Generically, Preferential Delivery means that if a device is dropping packets, it will drop low priority first. If it delivers packets, it delivers the high priority first. As I describe later in this chapter, the delivery priority could be different from the drop precedence.

Preferential Delivery does not mean that devices have to use any particular queuing strategy. Standard FIFO queuing is probably not going to provide a terribly effective way of implementing Preferential Delivery, but Weighted Fair Queuing is certainly a reasonable option. However, one can also implement a Preferential Delivery mechanism simply by sorting the various priority data streams into their own FIFO queues.

The Guaranteed Delivery service model means that each application is allocated a certain minimum amount of bandwidth through the network. There are different ways of implementing this bandwidth guarantee.

In some cases, the different applications have different reserved bandwidths through certain links. Whether an application uses its reserved minimum or not, that bandwidth is set aside for it. In other implementations, the specific applications have reserved bandwidths, but if they do not use it, other applications can borrow from the unused pool.

Some implementations allow each application a certain minimum bandwidth plus an option to burst above it if there is excess capacity. In this case, it is common to specify that packets sent using this burst capacity can be dropped if they encounter congestion.

One particularly interesting implementation of Guaranteed Delivery is the so-called Virtual Leased Line (VLL). In this case, the application is guaranteed a minimum and a maximum bandwidth with no congestion, no dropping, and minimal jitter. VLL is often implemented in conjunction with a tunnel, making the VLL look like a realistic dedicated link to the routers.

In general, Guaranteed Delivery allows the designer to specify not only bandwidth but also latency and jitter limitations. This specification is necessary for real-time interactive applications such as voice and video. In these applications, the data stream is usually almost constant, and jitter is intolerable.

The queuing mechanisms required to accomplish this are naturally more complex than the algorithms that I have discussed so far. They all involve setting up different queues for the different data streams and then servicing these queues appropriately. To minimize jitter, each queue has to be serviced on a timer instead of whenever the router gets around to it.

Layer 2 and Layer 3 QoS

So far, everything I discussed has left the actual implementation fairly generic. In principle, you can implement the QoS functionality at either Layer 2 or Layer 3.

The advantage to Layer 3 is, of course, that you can set a priority parameter in the Layer 3 packet header and have it visible at every hop through the network. This results in a good end-to-end QoS implementation and allows you to ensure consistent application behavior throughout the network.

Setting a parameter at Layer 3 tells the network very little about how it should actually handle this packet as it is routed from one media type to another.

There are also Layer 2 QoS features. Token Ring has the ability to send high-priority frames preferentially to lower priority frames. ATM has extremely sophisticated QoS functionality that allows you to specify sustained and burst rates directly, for example. Ethernet, on the other hand, has no native QoS functionality. However, Ethernet VLAN tags can specify a Class of Service value to affect how the frames in trunks are handled at each subsequent switch.

Over network regions that involve hopping from one segment to another via Layer 2 switches, you need a Layer 2 QoS implementation. This implementation allows you to specify how the switches handle the frames.

Meanwhile, a network needs Layer 3 QoS to allow consistent handling of packets as they pass through routers. Ideally, the Layer 3 information should be used to generate Layer 2 QoS behavior.

When a router receives a packet that has a high Layer 3–priority indication, it should use this information at Layer 2 in two ways. First, it should copy this information appropriately into the Layer 2 header so other Layer 2 devices can handle the packet properly. Second, it should select the appropriate Layer 2 QoS functionality when delivering the packet.

Buffering and Queuing

When a router receives a packet to pass along from one network to another, it often cannot transmit immediately. The medium may be in a busy state. For example, it could be an Ethernet segment on which another device is already talking. Or, the outbound port may already be busy sending another packet.

When a packet cannot be forwarded because of a temporary situation like this, it is usually best if the router holds onto it for a short time until it can send it along. This is called buffering. The packet is copied into the router's memory and placed in a queue to be transmitted as soon as possible.

There are several different kinds of queues. The simplest, which I have already mentioned earlier in this chapter, is a FIFO queue. A router using FIFO queuing simply puts all of the packets for a particular outbound physical interface in one place and sends them in the order they were received. FIFO queues are conceptually simple and may seem to treat all applications fairly, but in fact there are serious problems with FIFO queues when the network becomes busy.

Many bulk file-transfer applications, such as FTP or HTTP, have the property of sending data as fast as the network can accept it. When this data hits a bottleneck or congestion point in the network, it fills up the router's input queue until the router has to start dropping packets. Then the application backs off until it matches the available capacity of the network. Unfortunately, if other less-aggressive applications try to use the same network, their packets are also dropped when the queue fills up. Thus, FIFO queuing tends to favor the aggressive applications.

The worst part is that these aggressive applications are relatively time insensitive. The low-rate data flows that are choked off are often used for interactive real-time applications. Thus, FIFO queuing has the worst possible behavior in this situation. To get around this problem, other more sophisticated queuing algorithms have been developed. One of the most popular algorithms is called Fair Queuing.

In Fair Queuing, the router breaks up the incoming stream of packets into separate conversations and queues these conversations separately. Then the router takes packets from each queue in a simple rotation. It can take either a single packet at a time from each queue or, alternatively, a group of packets up to a certain predefined number of bytes.

Weighted Fair Queuing is a slight modification to this algorithm. Instead of picking equally (by number of packets or bytes) from each queue, the router assigns a *weight* to each queue. This weight can be based on any of a large number of different parameters such as the rate at which packets are received into the queue or the sizes of the packets. It can also be associated with formal priority markings such as IP Precedence or DSCP. In this way, Weighted Fair Queuing actually spans the gap between Best Efforts and Preferential Delivery service modes.

By breaking up the incoming stream of packets into individual conversations, Fair Queuing algorithms ensure that no one application can take all of the available bandwidth.

Returning to the FTP file-transfer example with Fair Queuing, the packets in this file transfer go into their own queue. If that queue fills up, then the router drops only FTP packets, but the other traffic streams are unaffected. When the FTP application

notices that packets have been dropped, it slows down the rate that it sends data. In this way, Fair Queuing and Weighted Fair Queuing prevent any one data stream (usually called a flow) from taking over the network.

Another Queuing option commonly used with Preferential Delivery is called Priority Queuing. This term means that each incoming packet is categorized by some rule and put into a queue. There will usually be a small number, perhaps as many as five of these queues, ranging in priority from high to low. The router services these different queues, taking packets preferentially from the high-priority queues.

This servicing is typically done by specifying a maximum number of bytes or packets to take in each pass from each queue, with the highest priority receiving the best service. In the most extreme version, the first priority queue is emptied before the second priority queue is considered. However, this process is usually just a recipe for ensuring that low priority traffic is not delivered at all.

You should be aware of three main problems with any Priority Queuing model. First, because every packet must be examined to determine its priority, high CPU loads on routers can occur. Care must be taken in limiting which routers need to do this examination and in making the test as simple as possible. Preferably, it should be based on just the IP TOS or DSCP field, which is described later in this chapter.

The second problem is that a straight-priority queue model allows different traffic flows within each priority grouping to interfere with one another. Effectively, each individual queue is a FIFO queue. Thus, it is important to understand how the application traffic flows work before selecting an implementation. If there is potential for one conversation within an application group to choke off the others in that group, then Priority Queuing is not appropriate.

The third problem happens when devices have too many different priorities. Each packet must be examined and compared to some criteria to find the appropriate priority. If there are many different priorities, then there are many different tests to perform on each packet, which results in high-router CPU load during peak traffic periods.

Also, using too many different priorities may divide the available bandwidth into too many pieces. This division then leaves each queue with a tiny amount of useful bandwidth, so it is always congested.

Suppose, for example, that a network has to support one extremely important application and eight less-important applications, all of which compete for a small amount of bandwidth. Each time the router services the high-priority queue, it grabs two packets. It then delivers one packet from each of the eight low-priority queues. In this example, the router winds up delivering four lower-priority packets for every high-priority packet. This situation clearly becomes worse the more queues each device has.

Furthermore, the more different queues the router has to service, the longer it takes to get back to the high-priority queue. This delay results in serious jitter problems, since there is a large random element in how long it will take to deliver any given packet. Thus, it is crucial to keep the number of different priorities as small as possible.

Integrated and Differentiated Services

The Internet Engineering Task Force (IETF) has specified two different standards for IP QoS. These standards are called Integrated Services (intserv) and Differentiated Services (diffserv).

The basic idea of intserv (also called IS in some documents) is to allow applications to request resources such as bandwidth or latency characteristics from the network. The network then keeps track of this individual conversation and ensures that it always has the reserved resources.

Although it is not required, the most common way to implement this resource request uses ReSerVation Protocol (RSVP). The end stations taking part in the user application use RSVP to request a specific performance characteristic from the network. RSVP is discussed later in this chapter.

The network then maintains state information about the individual conversations (called flows). This maintenance has an enormous overhead in a large network with thousands of simultaneous conversations. Therefore, it is not usually practical in the Core of a large network.

Integrated Services attempts to get around this scaling problem by allowing the network to aggregate the flows. In a complex network, allowing any-to-any communication, this aggregation poses further problems if it is done dynamically. In a hierarchical network, it should be possible to aggregate flows successfully at least on the in-bound direction to the Core of the network.

Differentiated Services takes a simpler approach to the same problem. By taking over the seldom-used TOS byte in the header of the IP packet, it defines an end-to-end priority. This priority value, called the Differentiated Services Control Point (DSCP), specifies how each router along the path will treat the packet.

Each router along the path reads this DSCP value. This step is easy for the routers because the information is stored in a single byte in the IP header. The DSCP value tells each router how to forward the packet, specifying a Per-Hop Behavior (PHB). There are standard PHB profiles that the router can follow. But the network engineer can configure the routers manually to interpret specific DSCP values differently.

Two standard flavors of PHB have been defined for Differentiated Services. These flavors are called Expedited Forwarding and Assured Forwarding, although the names are somewhat misleading. Assured Forwarding (AF) does not imply guaranteed

delivery as the name might suggest, but expedient delivery according to priority levels. Conversely, Expedited Forwarding (EF) is not merely expedient, as it does provide service assurances.

There are three main differences between the Integrated and Differentiated Services models for QoS:

- Integrated Services must maintain state information about individual traffic flows. Conversely, Differentiated Services combines all traffic of a particular type, which results in much better scaling properties for Differentiated Services in large networks.
- To set up a nondefault forwarding behavior, Integrated Services uses an external protocol such as RSVP reserve network resources. This is done on a per-conversation basis. It also works well with multicast data streams. Differentiated Services allows the end stations to define the way each individual packet is handled. This definition is done by setting the DSCP byte in the IP header of each packet. The network can optionally change this value if it is not appropriate.
- Because Differentiated Services defines the handling properties of each packet by referring to the DSCP byte in the header, it can handle path failure and path redundancy situations transparently. Integrated Services, on the other hand, needs the robust path-tracking features of RSVP to cope well with multiple paths or with changes in path routing through the network. Even with these capabilities, however, there is a significant probability of losing reserved resources when the path changes.

Assured Forwarding in Differentiated Services

The Assured Forwarding standard for Per-Hop Behavior Differentiated Services is defined in RFC 2597. In AF, two basic properties define how each packet will be forwarded. The standard defines four Classes and three different values for Drop Precedence.

The Class value is essentially a forwarding priority. Packets with the same Class value are all queued together. The standard requires that the packets of individual conversations be forwarded in the same order that they are received, as long as they are all of the same Class.

The most common way to implement AF is to give a separate queue to each Class. This allows the network to ensure that flows from different Classes do not interfere with one another. It also permits higher-priority Classes to receive more bandwidth from the network by increasing the amount of data taken from the more important queues each time the router takes packets from them.

In addition to the four Classes, AF defines three different types of Drop Precedence. This number simply tells the router which packets to drop first in case of congestion. When the Class queue fills up and the router needs to start dropping packets,

the ones with lower Drop Precedence values are protected. The router should scan through the queue and drop the packets with the highest Drop Precedence values first. If dropping the packets does not alleviate the congestion problem, then the router should drop all of the next-highest Drop Precedence packets before dropping the ones with the lowest Drop Precedence values.

In this way, AF can give important data streams better treatment as they pass through the network. Note, however, that AF does not necessarily guarantee a particular fraction of the total bandwidth for any one Class. It also doesn't give guaranteed end-to-end performance characteristics for specific data flows. Furthermore, it does not have the ability to give direct control over parameters such as jitter or bandwidth. It is merely a method for providing Preferential Delivery.

Expedited Forwarding in Differentiated Services

The Expedited Forwarding standard for PHB Differentiated Services is defined in RFC 2598. The basic goal of EF is to provide guaranteed service characteristics such as bandwidth, latency, and jitter.

One type of proposed EF implementation is the Virtual Leased Line (VLL). This implementation is essentially a reserved chunk of bandwidth through a network coupled with a queuing mechanism that restricts jitter. As with a real leased line, however, a VLL cannot handle any traffic in excess of its bandwidth limits. If an application tries to send packets too quickly, they will be dropped. Thus, EF is usually used in conjunction with some sort of Traffic Shaping.

IP TOS and Diffserv DSCP

The IP standards foresaw the need for specifying Quality of Service as long ago as 1981 in RFC 791. This document defines the current standard IP (IPv4) packet format and includes a byte called Type of Service (TOS). As QoS requirements and technology grew more sophisticated, this field has been replaced by the Distributed Services Control Point (DSCP), which includes significant backward compatibility with the older standard.

The TOS or DSCP value is typically set by the end devices. If an application knows that it needs special priority through the network, then it is able to set the appropriate value in each packet separately to affect how the network handles it. The network, however, is generally free to alter these values if they are not appropriate. If network devices change TOS or DSCP values, however, you should be careful about where it is done.

As I discussed elsewhere in this chapter, there is a lot of CPU overhead in categorizing and marking packets. Thus, the network should do it as little as possible. That usually means that it will mark the packets with the appropriate TOS or DSCP values as the packets enter the network. The first router they encounter should be the

only one making this change. Then the packets can traverse the network, enjoying the appropriate service level at each hop. If they leave this network and enter another, then they might be marked again with a different value.

The original standard for the format of the TOS field is defined in RFC 791. It breaks the 8-bit field into 2 3-bit sections. The first three bits specify the Precedence, and the second three specify a particular vision of PHB. The final two bits were designated as unused and set aside for future requirements. The approximate service types defined in Table 8-1 became the standard IP Precedence values.

Table 8-1. Standard IP Precedence values

IP Precedence	Decimal value	Bit pattern
Routine	0	000
Priority	1	001
Immediate	2	010
Flash	3	011
Flash Override	4	100
Critical	5	101
Internetwork Control	6	110
Network Control	7	111

The Internetwork Control value, 110, is reserved for network purposes such as routing protocol information. The highest-precedence value, Network Control, 111, is intended to remain confined within a network (or Autonomous System). Any of the other values can be freely assigned to specific user applications.

The third through sixth bits separately designate the required delay, throughput, and reliability characteristics, respectively. If the bit had a value of 0, then it could tolerate a high delay, low throughput, or low reliability. If the bit had a value of 1, then the packet needs a low delay, high throughput, or high reliability. The standard recommends setting only two of these parameters at a time, except in extreme situations.

In RFC 2474, these definitions were updated to allow them to work with Distributed Services. The TOS byte was renamed the DS byte. It was again broken into a 6-bit component, the DSCP, and two unused bits.

The 6-bit DSCP is broken into two 3-bit sections. The first three bits define the Class, and the last three define the PHB. This definition is done to help provide backward compatibility with networks that implement IP Precedence in the older TOS format. To create the four different Classes and three Drop Precedence values for Assured Forwarding, RFC 2597 defines the bit patterns as shown in Table 8-2.

Table 8-2. Assured Forwarding DSCP values

Drop Precedence	Class 1	Class 2	Class 3	Class 4
Lowest Drop Precedence	001010	010010	011010	100010
Medium Drop Precedence	001100	010100	011100	100100
Highest Drop Precedence	001110	010110	011110	100110

It is easy to see from Table 8-2 that the first three bits define the Class and the last three bits define the Drop Precedence. With three bits, it is possible to define several more Classes than the four defined in the standard. Specifically, the values 000, 101, 110, and 111 are all unused in Assured Forwarding. The Class values 110 and 111 are reserved for network purposes such as routing protocol information. Thus, these values are not available for general users. By default, any packet with a Class value of 000 is to be given a Best Efforts level of service. However, there is room for introduction of a new Class 5, if it is required. I use this value later when I talk about Expedited Forwarding.

This set of definitions for the AF DSCP is clearly compatible with the older TOS format. The only difference is that the older definitions of delay, throughput, and reliability are replaced with a new two-bit pattern indicating drop precedence. The last bit is always equal to zero.

There is only one defined DSCP value for EF. RFC 2598 recommends using the value 101110 for this purpose. Note that this is the obvious extension to the values in Table 8-2. Since EF offers service guarantees that are not available in AF, it is in some sense a higher priority. One additional Class value is available before reaching the reserved values—the value 101, which would be Class 5. At the same time, since packets designated for EF should not be dropped, they have the highest drop precedence value, 110. This value inherently means that only one type of EF is available.

A network can't have, for example, two flavors of EF—one with low and the other with high reserved bandwidth. If this separation is required, the best strategy is to define additional Control Point values and configure the routers to recognize them. In this case, it is better to fix the first three bits at 110 and use the second three bits to specify the different forwarding characteristics. However, it is important to remember that devices on other networks (such as the public Internet) will not recognize these parameters and may not handle the packet as delicately as you would like.

Traffic Shaping

Traffic Shaping is a system for controlling the rate of data flow into a network. Networks often use it in conjunction with other QoS mechanisms.

There are two main ways to control the rate of flow of traffic. A device can either throw away packets whenever the specified rate limit is reached, or it can buffer

packets and release them at the specified rate. The process of discarding packets that exceed a bandwidth parameter is usually called *policing*. Saving packets for future transmission is called *buffering*.

In any real-world application, of course, it is necessary to do both policing and buffering. If an application persistently sends data at twice the rate that the network can forward it, then it doesn't matter how many packets are put into the buffer because the network simply can't send them all along. If the traffic flow is characterized by a number of short bursts, then network devices can easily buffer the bursts to smooth them out—provided, of course, that the time average of the traffic rate is less than the available output bandwidth.

Traffic Shaping can be done either on an entire pipe of incoming data or on individual data flows. Usually, network designers implement Traffic Shaping only at network bottlenecks and at input points into the network.

EF is a good example of a place where Traffic Shaping needs to be used in conjunction with a QoS mechanism. The EF model specifies a certain sustained bandwidth level that the data flow is allowed to use. If an application exceeds this flow rate, then the excess packets are dropped. The best way to implement such a service is to ensure that the data stream entering the network is restricted to less than the reserved bandwidth. This data flow may enter the network from a user segment within the network, in which case the first router the traffic encounters does the traffic shaping.

Dropping packets, while undesirable, is not a completely bad thing in a network. Many protocols such as TCP have the ability to notice when they start losing packets because every packet has a sequence number. If packets do not follow in sequence then the end devices usually wait a short time to see if the missing packet will eventually arrive. When this time has elapsed, the receiving device sends a notification to the sending device to tell it about the missing packet.

When this happens, the sender assumes that the network has a reliability problem, and it reduces the number of packets it will send before it gets an acknowledgement (the TCP Window). Reducing the packets also reduces the amount of bandwidth that the application consumes.

By dropping TCP packets, the network can effectively control the rate that the application sends data. It tends to back off until it no longer sees dropped packets. This data-flow rate is exactly equal to the preset Traffic Shaping limit.

However, not all applications behave as well as TCP when they suffer from dropped packets. For example, UDP packets generally do not need to be acknowledged. Thus, UDP applications may not respond properly to traffic shaping. IPX has similar issues. The connection-oriented SPX protocol can respond to dropped packets by reducing its windowing, similar to TCP. But other IPX protocols are not so well behaved.

In general, it is a good idea to monitor applications that use heavily policed links to ensure that they behave well. If they do not, then you must increase the bandwidth to reduce the congestion.

Defining Traffic Types

Usually, traffic types are defined by some relatively simple parameters. Generally, looking at well-known fields within the IP packet header is fairly easy. Thus, these fields are the main factors used in identifying different traffic types.

In IP packets, five fields are typically used for classifying traffic. These fields are the source and destination IP addresses, the protocol type (primarily TCP, UDP, and ICMP), and the source and destination port numbers (for TCP and UDP).

Obviously, this amount of information is limited, but many applications can be easily identified with some combination of these parameters. Indeed, Fair Queuing applications use the set of all five fields to identify specific flows uniquely within a larger data stream.

For example, if a router needs to identify FTP file transfers, it needs only to look for a TCP protocol packet with either a source or destination port number of 20 or 21 (FTP uses 21 for control and 20 for actual data transfer). Similarly, if there is a large database server whose traffic needs to be protected, the router can simply look for its IP address in either the source or destination address field.

Note that in both of these examples the router looked in both the source and destination fields. This is because, in general, it is necessary to classify both sides of the conversation. If the router looks only at the destination address, then it will see the traffic going to the device with that address, but it will miss all of the traffic coming from it.

Similarly, a TCP session usually begins with a request on a well-known destination port from client to server. The client includes a dynamically assigned source port when it places this call. The server then uses this dynamic port number to identify the destination application when it talks back to the client.

In general, the router doesn't know which end is client and which end is server. When looking for a particular TCP port, the usual practice is to look in both the source and destination fields of a packet.

Some applications are not easily identified. For example, some applications use a dynamically generated port number. Using this port number can have important security and programming advantages, but it is extremely difficult for the network to give this session preferential treatment.

Conversely, some systems group many applications together. In some cases, such as with the Citrix system, the server passes only screen updates of applications running on a central server to the user workstation. Passing only screen updates makes it impossible to tell which packets correspond to which applications.

Citrix also includes the ability to run file transfers. In this case, however, the system's designers were thoughtful enough to include a batch-mode designation for these data streams and to put the flag specifying this mode in an easily accessible part of the packet. In this way, the network can at least distinguish between interactive and batch traffic. However, this is not always sufficient granularity.

The same problem occurs in many other network services. For example, it is sometimes necessary to give different Remote Procedure Call (RPC) applications different priorities. However, the fact that they all use the same basic Layer 4 architecture makes this difficult. In fact, this problem exists for any application built at a higher layer on the protocol stack. For programmers, building a new application on stock networking Application Program Interface (API) calls such as RPC can be extremely useful. If all of these applications wind up using the same TCP port numbers, it becomes hard to distinguish between them.

This distinction might be necessary for security reasons, as well as QoS reasons. For example, blocking Java code from being downloaded from certain web pages might be useful. However, blocking the code requires that the router distinguish between different types of URL information within a single web page. To the router, it all just looks like an HTTP connection.

One partial solution to this problem (there can never be a completely general solution because of the nature of the problem) is Cisco's proprietary Network-Based Application Recognition (NBAR) software. NBAR works with a set of specific Packet Description Language Module (PDLM) modules that tell the router how to find higher-layer information in the IP packet. When using NBAR to distinguish two applications that both use the same Layer 4 information, the router must have the appropriate PDLM module loaded. The PDLM modules then allow the router to distinguish between applications that use the same network layer information.

This information can then be applied to Access lists in the same way that Layer 3 information can be isolated. Once the information is accessible to an Access list, it is relatively easy to use it to set the DSCP or TOS bits in the IP header. The packet can then pass through the network with the appropriate QoS behavior.

The other parameter that is often used to define QoS classes is the size of the packet. Real-time applications such as packetized voice or video systems will often use a very small packet size. Small packets can usually be delivered with lower latency. If the data segment of a packet represents a constant amount of information, then it follows that a longer packet contains more data. Thus, a longer packet also represents a longer time period when capturing sound or video samples. If the application has to wait a longer time to fill up a packet before it is sent, then this clearly results in a higher latency.

Real-time applications often use shorter packets than low-priority batch-mode applications. For this reason, some networks give preferred treatment to smaller packets.

RSVP

ReSerVation Protocol (RSVP) is an IP protocol that allows end devices to request particular resource characteristics from the network. It is a control protocol similar in concept to ICMP, so it does not carry the data stream. Instead, it just reserves the resources.

The general concept is that an end device requiring certain network resources will send an RSVP packet through the network. This is an IP packet whose destination is the other end device taking part in the application conversation. The packet passes through the network, hop by hop. Each intermediate router reads the packet and allocates the appropriate resources, if possible.

If a router is unable to comply with the request, it responds back down the path with a packet indicating that the request has been refused. All intermediate routers again read the packet and release the resources. If the router is willing and able to reserve the resources for this application, it passes the packet along to the next device along the path.

If the RSVP request goes through the entire network, the end device responds with a message indicating that the request has been granted.

One clear problem with this model is that most good network designs don't have a single unique path between any two points. One of the main design principles is to use multiple-path redundancy.

RSVP includes elaborate methods for rerouting the reserved path in case of a network failure. When the routing table in a router in the middle of the network changes, it attempts to establish a new reserved path using the new routing information. Also, RSVP uses a periodic system to verify that the reserved resources are still available.

If a network failure forces a change in path, then the new path may refuse to grant the reservation request. In fact, this refusal is quite likely because the new path may suddenly find itself carrying a heavy additional load. Under these circumstances, it probably will not allow new reservation requests.

Thus, the application may suddenly lose its reserved resources without losing actual network connectivity. In a large network, this loss tends to result in considerably less-stable performance than the simpler Differentiated Service model.

Another problem arises because of multiple redundant paths through a network. There are two ways to handle redundant paths. If a router handling an RSVP request notices that it has more than one possible way to get to the destination, it could reserve bandwidth on both paths and forward the RSVP request to downstream next-hop devices. Or, it could select one of the paths and use it for the application.

The first case is clearly inefficient because the application reserves resources that it will not use. If the router shares the load among all possible paths, then reserving the full bandwidth requirement on each path individually is inefficient.

On the other hand, if the router deliberately selects only one of the possible paths for this traffic stream, then it loses one of the key advantages to a highly redundant design philosophy. Worse still, the highest level of fault-tolerant redundancy is used only for the lowest-priority traffic.

The only alternative is to have the RSVP protocol actively track all possible paths through the network. In doing so, it must have an accurate model for how effectively the network can share loads among these paths. This level of tracking is not practical in a large network.

Network-Design Considerations

The best QoS implementations readily break up into two functional parts. The first router a packet encounters upon entering the network should set its TOS or DSCP field. Then the rest of the devices in the network only need to look at this one field to know how to treat this packet.

There is a very simple reason for this division of labor. The process of reading and classifying packets can be extremely CPU intensive, so the network should do it only once.

Furthermore, when getting closer to the Core of a hierarchical network, one expects to see more traffic. The easiest place to do the classification is at the edge. In many cases, the edge router is in a unique position to do this classification. For example, if the edge router runs any sort of tunneling protocol, such as DLSw, then it can see application information in the packet before it is encapsulated into the tunnel protocol. This fact is even truer when the edge device encrypts the packet contents, as in a VPN architecture.

In this case, there is essentially no way to differentiate between applications after the packet is encrypted. The only practical place to do the classification is the edge router. Then, once the packet enters the network, it needs a design that permits treating the different traffic classes appropriately. In an Integrated Services implementation, the design must be built to respond to RSVP requests.

RSVP suffers from efficiency problems when many paths run through the network. Because it requires every router to keep track of all reserved data flows, it does not scale well to large networks. However, there is a relatively straightforward way of getting around this problem.

It is possible to use Integrated Services only at the edges of the network and build the Core with Differentiated Services. The key to making this possible is in the flow-aggregation properties of Integrated Services. These properties specify that the

network is allowed to group a set of flows together if they all have similar properties and then treat them all at once. That principle is good in theory, but Differentiated Services is usually limited to either Assured or Expedited Forwarding. Thus, you have to be careful about how you map specific RSVP requests to DSCP values and how you implement the Per-Hop Behavior.

An obvious way to make a gateway between an Integrated Services edge and a Differentiated Services Core is through EF. EF allows explicit reservation of bandwidth up to and including VLL implementations.

Note that this reservation implies that the network must aggregate an arbitrary number of reserved bandwidth flows. Thus, it is possible to oversubscribe the bandwidth that has been reserved in the Core. However, if oversubscription occurs, the router that acts as the gateway between the Integrated and Differentiated Services regions simply refuses any further RSVP requests.

For packets passing through Differentiated Services networks, there are many ways to implement the required traffic-flow characteristics. The simplest method is to use Weighted Fair Queuing on every router in the Core.

This method does not strictly meet the requirements of either EF or AF PHB models because it does not have the prescribed drop precedence characteristics. However, Weighted Fair Queuing does allow the different flows to be weighted according to the DSCP Class (or TOS IP Precedence, since they are compatible).

If a strict implementation of either EF or AF is not required, this implementation is much easier. If a strict AF model is required, then you must to consult the router vendor to find out how to turn on this style of queuing.

For EF implementations, on the other hand, you should define the different performance criteria carefully. How much bandwidth is reserved? What are the latency and jitter requirements? These parameters in turn define how the software that services the queues is configured. Most importantly for EF implementations, how are the different logical paths defined?

If many physical path possibilities exist between two end points (which is a design philosophy that I strongly advocate), then the designer has to be absolutely clear on how structures such as VLL will be implemented. Is the VLL only defined along one path, or is it configured through multiple paths?

In general, I prefer to keep network design as simple as possible. In almost all cases where QoS is required, I recommend the AF model of Distributed Services. Classification is to be done at the edges of the network. Then every other device in the network needs to implement only the appropriate PHB.

If congestion within the network is kept under control, it is rarely necessary to implement any real bandwidth reservation. For light congestion, there is little or no observable difference. However, if congestion becomes severe or sustained, then it is

usually easier to increase the bandwidth than it is to implement a more strict QoS system. If there is a serious congestion problem in the network, then implementing strict bandwidth reservation for one application only makes the congestion problem worse for every other application using the network.

In any QoS implementation, remember that bandwidth is a finite and limited resource. All you can with QoS is to allocate it a little more fairly. If there is simply not enough to go around, then QoS cannot solve the problem.