4

Techniques for Performance Engineering

4.1 Introduction

In this chapter we present general techniques for performance engineering data networks. We begin by discussing a collection of techniques: load engineering, application characterization, application discrimination methods and collection and traffic monitoring. We end this chapter with a section that ties these various techniques and tools together. The context we choose is a discussion of the process to follow when implementing a new application in an existing data network. The discussion of load engineering in the first section of this chapter builds on our knowledge of queuing systems, their stability, and general load versus delay characterization. This knowledge is leveraged to define measurable component thresholds in order to maintain satisfactory application level performance over the enterprise network.

We next classify applications into two broad categories: *latency sensitive* and *bandwidth sensitive*. Even this broad classification of application types is useful when engineering data networks for application-level performance objectives. This discussion is followed by an extensive taxonomy of techniques to minimize cross-application interference effects in multiprotocol, enterprise networks. Techniques discussed here include type of service routing, priority queuing, processor sharing, adaptive controls, and selective discards.

This is followed up by a section on network management and data collection tools. These are useful in several areas including (1) data collection and application-level characterization and (2) traffic monitoring and forecasting. The general techniques we consider in this chapter are used in one way or another for performance engineering data networks. To reinforce this concept, we conclude the chapter with a step-by-step identification of the process to follow when deploying a new application over an existing corporate network.

4.2 Load Engineering

In Chapter 3 we developed various queuing models relating component delays to their utilization. We also know that queuing systems are stable, that is, they have finite queuing time and queue lengths, when their load remains less than 100%. The goal of load engineering is to ensure that the load on the system remains less than an identified threshold, above which performance degradation occurs. This is accomplished through (1) defining appropriate target component utilization on critical components within the network, (2) constant monitoring of the load on system components, and (3) making necessary adjustments to component load in the event that it becomes too high or too low, based on a developed action plan.

Within this section, we concentrate on a simple queuing model akin to a packet buffer system feeding a private line facility. In Section 5.4 in the following chapter, we discuss load engineering of a more complex frame relay interface and its associated virtual circuits.

As discussed in Chapter 3, the queuing time is a function of T_s(average), the average service time, and U, the utilization. The component utilization is a product of the arrival rate, L, and T_s(average), that is, $U = L \times T_s$(average). In fact, the queuing delay increases when either (or both) the service time or the arrival rate on the system increases. As a rule of thumb, queuing delays in a system should not overly dominate the service times; say, queuing time should never be more than twice the service time.

Looking at the curve for the queuing time in an M/M/1 system, shown in Figure 4.1 (and discussed in Chapter 3), this implies that the load on the system should remain less than roughly 70%. In the figure, the queuing time is measured in units of T_s(average). This point can be thought of as the location in the curve above which small increases in load will result in large increases in queuing times. If instead we wanted to keep the queuing delays less than or roughly equal to the insertion times, then from the figure we see that the utilization should be maintained at less than or equal to 50%. Again, these estimates are based on the M/M/1 queuing model and results will vary depending on the specific queuing model used.

From Chapter 3, we know that the queuing delay, Q, as a function of utilization, U, is:

$$Q = [U / (1 - U)] \times T_s\text{(average)}$$

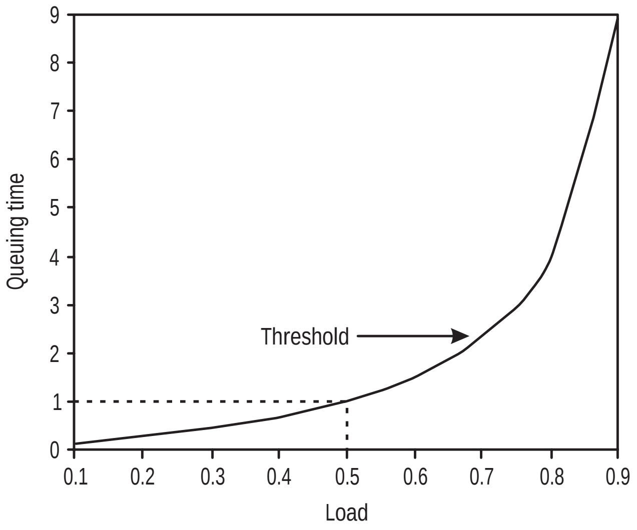

Figure 4.1 The expected queuing time as a function of load for the M/M/1 queuing.

For the queuing delay to be equal to T_s(average), we see that the utilization should be equal to 50%, that is,

$$Q = [0.5 / (1 - 0.5)] \times T_s(\text{average}) = T_s(\text{average})$$

For the queuing delay to be equal to 2 × T_s(average), we see that the utilization is equal to 66%:

$$Q = [0.66 / (1 - 0.66)] \times T_s(\text{average}) = 2 \times T_s(\text{average})$$

This is consistent with our reading of the delay versus load curve shown in Figure 4.1.

A more general queuing model discussed in Appendix A is the M/G/1 model. This model incorporates the variability of the packet sizes in its expression for the queuing delay. The queuing delay expression from the M/G/1 model is:

$$Q = [(U \times V) / (1 - U)] \times T_s(\text{average})$$

where V is a measure of the variability of the packet transmission time, that is, $V = (1 + C_V{**}2) / 2$, and C_V is the ratio of the standard deviation of the packet transmission time to the average packet transmission time. As we discuss in Appendix A, $V = 1$ for the M/M/1 queuing model. The M/G/1 expression simply states that the greater the variability in the packet sizes, the greater the queuing delay experienced on the transmission facility. So, the higher the variability in packet sizes, the more conservative one must be about loading the connection. To make this a little more quantitative, say we choose to try to maintain queuing delays to be no more than twice the average packet insertion time on the transmission facilities. Then the M/G/1 queuing model yields:

$$Q = 2 \times T_s(\text{average}) = [(U \times V) / (1 - U)] \times T_s(\text{average})$$

Eliminating T_s(average) on both sides of this equation yields

$$2 = [(U \times V) / (1 - U)]$$

Rearranging this expression and solving for U we get:

$$U = 2 / (2 + V)$$

If $V = 1$, as it is assumed to be in the M/M/1 model, we get that $U = 66\%$ as a target threshold (which is roughly our 70% threshold in the previous paragraph). However, if the variability in the packet sizes is higher, say, $V = 2$, then we get $U = 50\%$ as a target threshold. This validates our claim that the higher the variability in packet sizes, the more conservative one must be about loading the connection.

One last consideration in defining thresholds is the period of time over which the loads are measured. Or, in other words, should the thresholds (and their corresponding averaging) apply to 1-, 5-, or 15-min intervals? And should the engineer focus on the average results over a number of sampling intervals or the peak result? Further, at what point should the engineer decide to make capacity management decisions to increase the necessary bandwidth? If the chosen threshold is exceeded once a day for one month, does this constitute the point at which more capacity is ordered? Or should more capacity be ordered only when 20% of all measurements during the 8-hour work day exceed the chosen threshold?

Unfortunately there are no clear and definitive answers to these questions. Much depends on the nature and the time sensitivity of the applications running over the network and on the amount of data the engineer wishes to store and analyze. For example:

- *Fifteen-minute averages.* In most situations it is probably reasonable to tract 15-min intervals and monitor the average utilization threshold over this period. Fifteen-minute intervals are short enough to catch the hourly variations in the daily behavior of corporate employee activities. However, it is not so short as to cause the data collection and monitoring systems to have to store large amounts of data.
- *Five-minute averages.* In some situations, for example, reservation systems where end customers of the corporation are immediately affected by the system delays, network engineers should consider shorter time periods over which to tract thresholds. This would give the engineer greater visibility into the nature of the variations in the network loading, while not requiring the collection and storage of too much data.
- *Five-minute averages with hourly peaks.* In some situations it is more important to keep track of the extremes in the variation of the utilization. In these cases, taking relatively short averaging periods, for example, 5-min, and monitoring the hourly peak 5-min period is appropriate. Here, the thresholds are set for these 5-min, hourly peaks and capacity management decisions are based on these peak values.

Of course, there are an infinite number of possibilities and combinations. Note that the shorter the time interval over which the component values are averaged, the greater the amount of data to be collected and the greater the storage requirement on the data collection system.

Further, for a fixed threshold, for example, 70% utilization on a private line facility, the shorter the time measurement interval, the greater the bandwidth requirement to maintain the chosen component levels. This is due to the fact that short measurement intervals will show a greater variation in the averaged values and in turn increase the likelihood of seeing component thresholds exceeded.

At one level it would be nice to be able to answer all of these questions by relating all of this back to end-to-end performance objectives. But this is rarely feasible. At another level, just by going through the thought process of setting thresholds and measurement periods (and collecting and analyzing them), the network engineer has accomplished the majority of the work discussed in this section, independent of the specific periods, peaks, thresholds, and so on.

Once the component thresholds are identified and the measurement intervals chosen, then methods to monitor the component metrics must be determined. Monitoring tools is one of the topics discussed in Section 4.5. However, before moving on, let us first mention several ways to modify the load on a queuing system in the event that it is determined through monitoring that the component threshold is consistently being exceeded. These include:

- Changing the number of servers in the system; for example, we can add servers in a client/server environment when the server is being overloaded, or we can add parallel communications facilities between two locations in the event that the transmission line is being overloaded.
- Changing the rate at which the servers operate; for example, we can increase the speed of the transmission facility that is being overloaded.
- Providing preferential treatment to the more important applications, hence reducing their effective utilization (see Section 4.4).

To summarize, the keys to successful load engineering are:

- To clearly define thresholds for each component that is critical to the end-to-end performance of the network;
- To ensure that the load on the particular components in question is constantly monitored; and
- To develop a strategy to actively adjust the load in the event that the specified threshold is either being exceeded or the load is too small in comparison to the threshold.

To accomplish these tasks, measurement and monitoring equipment need to be utilized. These are discussed in Section 4.5. But first we wish to discuss some techniques to characterize applications and to discriminate between application types in network deployments.

4.3 Latency-Sensitive and Bandwidth-Sensitive Applications

The first section of this chapter outlined a methodology for load engineering components within a data network. The methodology consisted of component level thresholds from load engineering models and monitoring of the components to maintain loads less than or equal to the desired thresholds. Measurement systems are deployed in order to maintain the component level thresholds and to support capacity management tools and methods. But how do these component thresholds relate to the end-to-end performance desired by the end user or application running over the data network? To answer this question, it is necessary to understand the applications running over the data network and their characteristics.

This section discusses various application types or profiles. We use these profiles to better understand how to engineer the underlying network components in order to achieve desirable end-to-end performance. We identify two categories of applications, *latency-sensitive* and *bandwidth-sensitive* applications.[1] In this section we identify and define these application types. In the next section, we discuss issues associated with supporting both application types within a common packet network infrastructure.

Latency-sensitive applications are those whose performance critically depends on the delay of the underlying network. In Chapter 3, we distinguished between latency and delay as a way to reinforce the notion that some delays are immutable, that is, cannot be changed by modifying the underlying network infrastructure. For instance, propagation delay is ultimately bounded by a fundamental law of physics, that is, the finite speed of light.

Latency-sensitive applications do not perform well over a network with significant delays. These applications are often very "chatty" in nature. A single user transaction, for example, a request for a piece of information, may be composed of numerous, elemental exchanges between the requesting client and the responding server. This is illustrated in Figure 4.2. Here a user makes a request for some information from the remote server. The application invokes numerous elemental exchanges (for example, multiple SQL queries to a database server) of information between the client and the server before sending the user the specific information requested. In this instance, the user-perceived delay between the issuance of the request and the receipt of information is proportional to $\sim N \times$ (Network round-trip delay). Therefore, the network round-trip delay is magnified by a factor of N in the eyes of the end user. The factor of N can be as large as several hundred for some applications (see Chapters 9 and 11 for examples of such applications).

This tends not to be a problem if the underlying network is a LAN with round-trip delays on the order of a few milliseconds. However, if the application runs over a WAN where the round-trip delays are typically in the hundreds of milliseconds, the impact on the end user performance would be devastating.

Applications such as these abound in corporate data networks. Often, the application developers do not give WAN performance issues adequate attention until after full-scale deployment. When deployed over the WAN, the

1. Many applications have the combined characteristics of latency and bandwidth sensitivity. These are treated in more detail in Chapter 9.

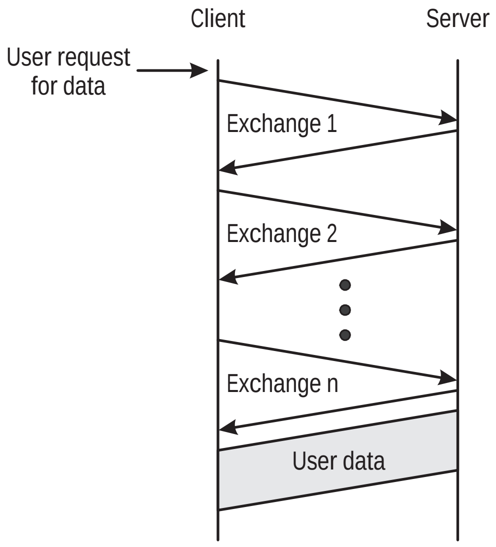

Figure 4.2 An example trace of a latency-sensitive application.

frailties of these latency-sensitive applications are exposed. Several approaches can be taken to alleviate this situation, including migrating to three-tier architectures or relying on the use of thin-client solutions. This topic is discussed in Chapter 9.

In contrast to latency-sensitive applications are bandwidth-sensitive applications. Bandwidth-sensitive applications typically involve the transfer of high volumes of data with relatively few elemental exchanges between the client and the remote server machines. This is illustrated in Figure 4.3. Here the user perception of application delay is dominated by the connection bandwidth, and is proportional to ~(Volume of data) / (Connection throughput). Bandwidth-sensitive applications show little to no dependence on the round-trip delay. These applications are written to require few elemental exchanges, in contrast to latency-sensitive applications.

4.4 Methods to Discriminate Traffic in a Multiprotocol Network

Some latency-sensitive applications tend to exchange a high number of relatively small packet transactions, such as many client/server applications. Other latency-sensitive applications, such as Winframe and telnet, rely on the network to essentially echoplex packets off of remote servers before presenting the

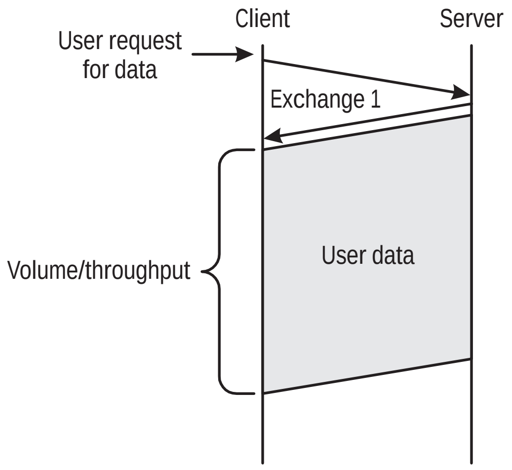

Figure 4.3 An example trace of a bandwidth-sensitive application.

typed characters or the mouse actions at the user's terminal screen. For good user performance, these packet exchanges must not encounter excessive delays.

In contrast, bandwidth-sensitive applications tend to transmit large volumes of traffic within a short period of time. If care is not taken in engineering, these applications may cause temporary network congestion. If the network is simultaneously carrying both types of applications, then the presence of the bandwidth-sensitive applications may adversely degrade the user perceived performance of the latency-sensitive applications. The several techniques available to mitigate these cross-application effects are the topic of this section. In the following chapter, we revisit this discussion within the context of frame relay networking.

Consider the case where we are mixing the two types of applications simultaneously carried over the reference connection in Figure 4.4. We first consider the impact the underlying window size associated with the bandwidth-sensitive application has on the performance of the latency-sensitive application. The result of this inspection will lead us to the conclusion that we should not rely on network buffers to store our end system's data. We follow this analysis with a discussion of the techniques available to network designers and engineers to discriminate between traffic types within the network.

4.4.1 Window Size Tuning

In this section, we analyze the effect the bandwidth-sensitive application window size has on latency-sensitive applications. The result of this analysis is that

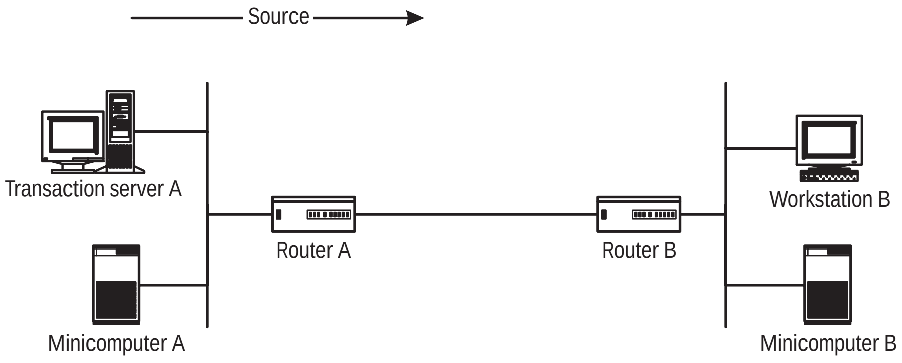

Figure 4.4 Reference connection #1.

windows should be tuned large enough that the file transfers can reasonably fill the facilities along the data path, but not be so large that they cause a large amount of queuing within the data buffers of the intermediate network routers and switches.

Although we find this statement to be true, it is often the case that the engineer has little control over the specifics of the transport window sizes. We discuss this in more detail later.

Now that we know the conclusion, let's see how this conclusion comes about. Consider the performance of reference connection #1 (RC#1, shown in Figure 4.4), when the transaction and the file transfer servers are simultaneously active. What is the impact of the file transfer traffic on the observed performance of the database transaction application? Assume for the moment that the file transfer has been ongoing for some time and look at the timing diagrams in Figure 4.5.

Figure 4.5 shows the timing diagrams for the file transfer traffic in "steady state" for two different window sizes, windows of size three and seven. These timing diagrams are similar to those we have shown earlier with respect to file transfers, only they are somewhat more cluttered. However, this clutter is necessary to observe the effect we want to analyze.

Notice that for the case of the smaller window size (that is, the timing diagram on the left-hand side of the figure), the private line facility is fairly highly utilized yet there is little or no queuing delay experienced by the file transfer data in the WAN router. The upper part of this timing diagram roughly represents a steady-state behavior of the file traffic over RC#1. The window is smoothly rotating and the WAN facility shows a fairly high utilization. Then, about halfway down the timing diagram, the transaction application kicks off and transmits a transaction packet onto the LAN segment. This transaction

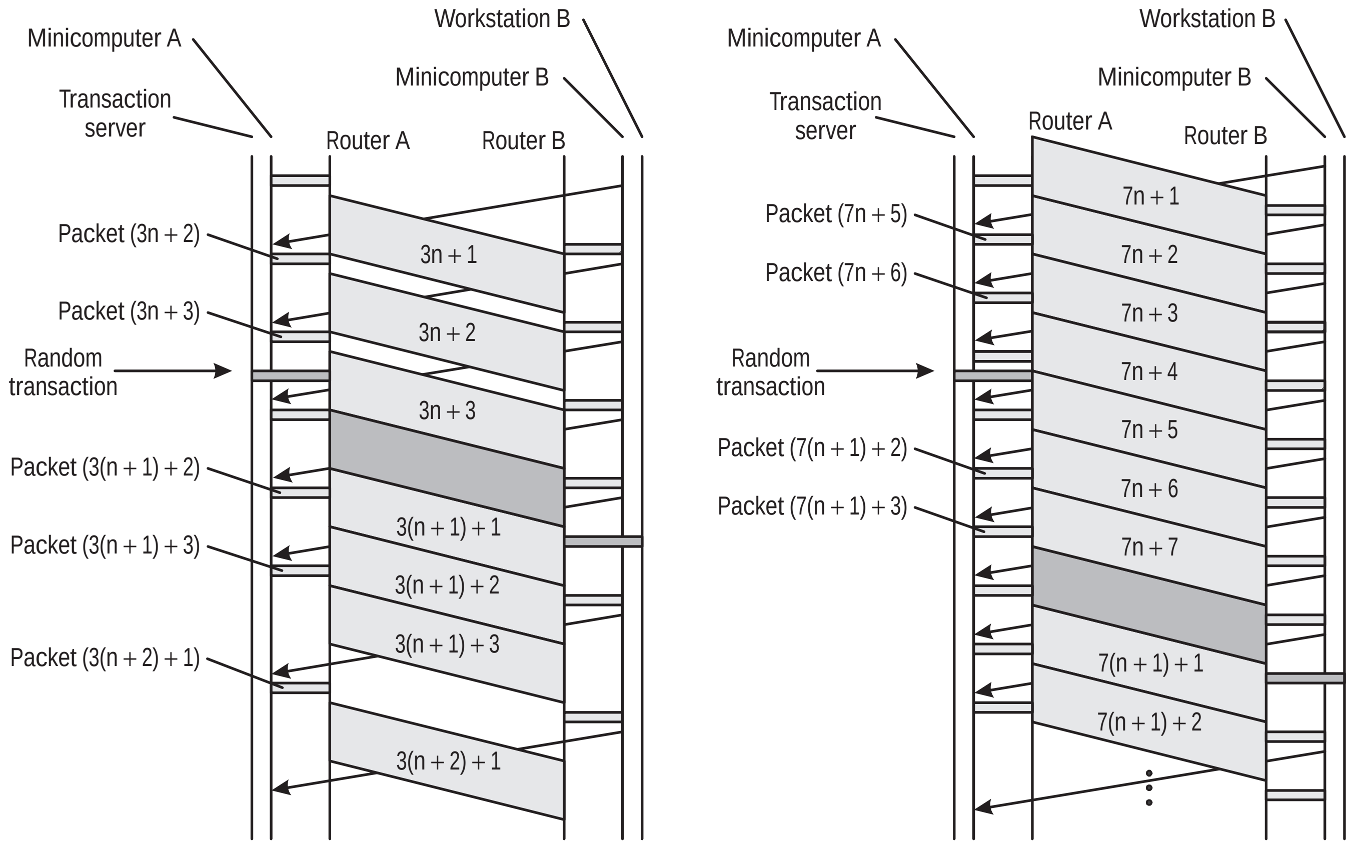

Figure 4.5 Timing diagrams showing cross-application effects.

packet gets interleaved between packets of the ongoing file transfer. Due to the relatively high speed of the LAN segment, gaps exist between the ongoing file transfer packets, and the transaction packet has little effect on the file packets. Once the transaction packet enters the router, the interesting question is how long does the transaction packet have to wait before it gets transmitted onto the relatively slower WAN private line segment. In the case of the timing diagram on the left-hand side of the figure for the case of the smaller file transfer window size, the wait for the transaction packet is small. The window is tuned to keep the WAN facility at a little less than 100% utilization (as is evident by the existence of small idle periods between file transfer packet transmissions). Hence, the transaction packet has a queuing delay of (on average) one-half the insertion delay of a file transfer packet (onto the private line facility).

As an example, for a WAN private line facility running at 56 Kbps and a file transfer packet size of 560 bytes (roughly the size of an off-LAN IP packet carrying FTP data), one-half the insertion delay is roughly 40 msec ($560 \times 8/56{,}000$ sec divided by 2). This would not be noticeable to the end user of the transaction application.

The timing diagram on the right-hand side of Figure 4.5 shows comparable behavior for the case in which the window size of the file transfer application is larger, that is, seven packets. As we now know, the main effect of increasing the window size in these example reference connections is to place more of a burden on the buffers in the intermediate routers to queue the file traffic packets. Increasing the window beyond a given size does not improve the performance of the file transfer application.

This point is again evident in Figure 4.5. In this example, the router is roughly buffering from two to three file transfer data packets while queuing for transmission onto the relatively slow WAN facility. So now what do we expect to happen when the transaction application packet is transmitted? As before, it is interleaved with the file transfer packets on the LAN segment to the router. It must now wait its turn for transmission onto the WAN facility and hence it is queued. But instead of only having to wait on average for half a packet insertion time, it has to wait for roughly two and a half file transfer packet insertion delays.

Using the same example conditions as in the previous paragraphs, the transaction packet will experience a queuing delay of roughly 200 msec. This is within the perception of an end user and can be exacerbated by larger window sizes or a greater number of simultaneous file transfers or a greater number of transaction packets.

For the conditions assumed in drawing the timing diagram in Figure 4.5, anything larger than roughly a window size of three will cause larger and larger delays for the transaction applications. Further, increasing the window from one to two to three will improve the end-to-end performance of the file transfer traffic. But increasing the window size greater than three will not improve the end-to-end performance of the file application and it will only degrade the performance of the transaction application. From this example, we can then conclude that the "optimal" window size for the file transfer is three. This example demonstrates the necessity of choosing a window size large enough to support sufficient file transfer throughput but not so large as to degrade the performance of other applications sharing the same network resources. However, while it is important to understand the dynamics of this situation, window size tuning may be appropriate only on special occasions. It cannot be recommended as a general method for traffic discrimination because (1) dissimilar TCP stacks deployed within an enterprise will have different configuration capabilities and characteristics, (2) it is hard to implement/configure options on all clients in an enterprise, and (3) it is not always clear what to configure the options to because of the heterogeneous nature of the enterprise network. Therefore, other methods of traffic discrimination are usually relied on.

We are not quite ready to leave the example in Figure 4.5. We now want to ask another question: What can be tuned or implemented in this example to improve the relative performance of the file and transaction traffic when mixed on a common network segment? We have already discussed in some detail one way to improve the relative performance of these interacting applications, that is, optimal tuning of the file transfer window size. A number of other techniques have been discussed in the industry and have been implemented in various data products, in networking protocols, and in data services, including the following:

- Type of service routing;
- Priority queuing;
- Bandwidth or processor sharing (also referred to as weighted fair queuing);
- Adaptive controls; and
- Selective discards.

We now discuss each of these other techniques in turn as they apply to our first example of mixing various types of applications onto a common multiprotocol data network.

4.4.2 Type of Service Routing

Within most data networks, there are often several different routes to a given destination. This is by design, usually to provide a high level of reliability within the overall network. These different paths to the same destination may ride over similar types of facilities, but these facilities may have different engineering rules applied to their capacity management or have different bandwidths associated with each, or they may ride over different types of facilities. For example, one route may travel over satellite facilities while another route over terrestrial facilities.

In any event, these different routes may have different performance characteristics associated with them. The path utilizing satellite facilities may be characterized as a high-throughput yet high-delay route. The path utilizing terrestrial facilities may be characterized as a low-throughput, low-delay route. In this case, our transaction traffic would be perceived to perform better by the end user if it were carried over the path utilizing the terrestrial facilities. The file transfer application would be perceived to perform better by the end user if it were carried (or routed) over the path utilizing the satellite facilities. This is the essence of type of service (TOS) routing.

Because different applications impose different performance requirements on the underlying network, it seems reasonable to route the various applications over the "optimal path" within the network for the particular application in question. In practice, this capability is described by first defining the set of performance metrics necessary to specify all reasonable application requirements. The metrics typically include delay, delay variation, throughput, reliability, and cost. The requested values for each of the metrics and allowable groupings are defined. Then their encoding is specified. Often only a rather crude set of values is specified: high, medium, or low delay; high, medium, and low throughput; and so on. Typical groupings may be high throughput with high-delay service or low throughput with low-delay service. These groupings are termed the type of service.

To implement TOS routing, several capabilities are necessary. For virtual circuit networks the requested TOS values for the connection are indicated in the call setup message. In datagram networks, the requested TOS values must be carried in every datagram header. For TOS routing, separate routing tables must be maintained for each possible TOS supported within the given network. Also, the network should have developed separate capacity management capabilities and guidelines for each TOS supported in the given network. When strict TOS routing is desired, some negotiation capabilities should be supported. In practice, all of these are extremely complex to develop, implement, and maintain.

The most common implementations of TOS routing support throughput metrics (often referred to as *throughput classes*). These implementations are found in X.25 (referred to as throughput class), frame relay (referred to as *committed information rate*), and ATM (referred to as *sustainable cell rate*) networks supporting SVC signaling capabilities or PVC manual provisioning capabilities. All of these technologies support the routing of a VC, which can sustain the throughput defined by these terms, for example, throughput class, CIR, and SCR.

Finally, to be truly useful in our examples, the end applications need some capability of communicating the necessary TOS to the routing entity in the network. This is rarely available in layered networks due to the lack of consistent standards in this area.

So how would TOS routing improve the relative performance of our file transfer and transaction-based application traffic in our reference network? Figure 4.6 shows an expansion of RC#1 with multiple routes providing different TOS paths. The satellite path provides a high-throughput and high-delay TOS, and the terrestrial path provides a low-throughput and low-delay TOS. One strategy would be to route the file transfer traffic over the satellite path and the transaction traffic over the terrestrial path; this essentially isolates the cross-application effects. Specifically, the transaction traffic does not get caught in a large queue behind the file transfer packets, while queuing for insertion onto the same communications facility.

4.4.3 Priority Queuing

In the examples given in the last section, the file transfer and transaction application data essentially collide at the gateway buffer onto the WAN facility.

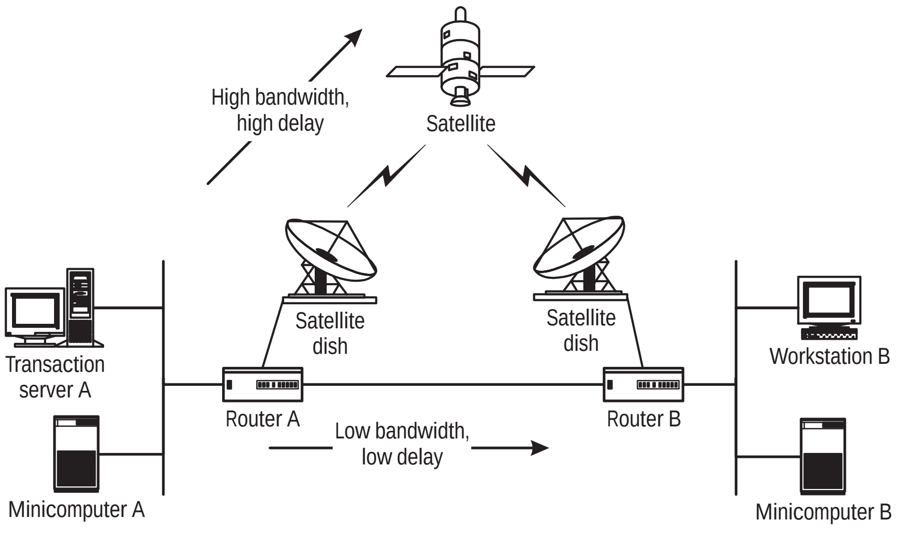

Figure 4.6 Reference connection showing multiple paths with different TOS.

Window size tuning minimizes to some extent the impact of the collision. TOS routing avoids the collision altogether by routing the different applications' data onto different buffers and WAN facilities out of the LAN gateway. As discussed, the TOS routing capability requires the routers to identify the different TOS requests and routes these requests appropriately. Another mechanism, which has the potential to greatly minimize the collision of the various applications traffic, is *priority queuing.*

Like TOS routing, suppose that a mechanism exists for communicating to the gateway the type of application data being carried within a given data packet. It would then be possible for the router to favor one application's data over the other by moving it "to the head of the line" so to speak. If the transaction application data were placed at the front of the queue holding the file transfer traffic, then the transaction data would only have to wait for the insertion of the current file transfer packet onto the facility to complete before it was transmitted onto the same WAN facility.

In this case (that of a single transaction packet), the maximum queuing delay would be a single file transfer packet insertion delay. And the average queuing delay would be one-half the file transfer packet insertion delay.[2] This mechanism is termed *priority queuing*.

As an example, for a WAN facility running at 56 Kbps and a file transfer packet size of 560 bytes, one-half the insertion delay of the file transfer packet is roughly 40 msec. This would not be noticeable to the end user of the transaction application.

Strictly speaking, it is not necessary to communicate the nature or type of application data being carried within the data packet. It is only necessary to communicate the *priority level* of the data. One could easily imagine a need for several priority levels. One example would be to have high-, medium-, and low-delay TOS priority levels. Packet level headers would then only be required to indicate the priority level of the packet.

A typical implementation would allocate separate data buffers for each interface on a gateway, router, or switch; there would be one queue per priority level. When the router has to insert the next data packet onto the facility, it takes the first packet from the highest priority queue containing data. This type

2. If the transaction packet arrives to the router buffer just prior to the completion of the current file transfer packet insertion onto the facility, its queuing time will be essentially zero. If it arrives just following the start of the current file transfer insertion onto the facility, its queuing time will be essentially a full file transfer packet insertion time. Then, assuming that the transaction packet arrives independently of the state of the file transfer packet insertion, its average queuing time will be one-half the file transfer packet insertion time.

of priority queuing mechanism is referred to as *nonpreemptive priority queuing*. This is shown in Figure 4.7.

Conversely, a *preemptive priority queuing* implementation allows for the server to halt (or suspend) the insertion of the lower priority packet on the arrival of a higher priority packet. Once the higher priority packet (and all other subsequent higher priority packets) is serviced, then the service of the preempted packet can start over (or resume).

In practice, the majority of the priority queuing implementations are of the nonpreemptive priority case. However, there are cases, especially on low-speed serial links where routers will support a form of preemptive priority, based on a packet fragmentation scheme. This is necessary when the high-priority traffic has very stringent delays and/or delay variation requirements and is being run over relatively slow links. One such example would be when attempting to carry packet voice over an enterprise data network.

As for the case of implementing a TOS routing capability, priority queuing schemes require that information on the nature (e.g., the priority level) of the application be passed down to the entity building the packet level data header. It is the packet header that must carry the indication of the priority level of the application. In a single, end-to-end, networking protocol

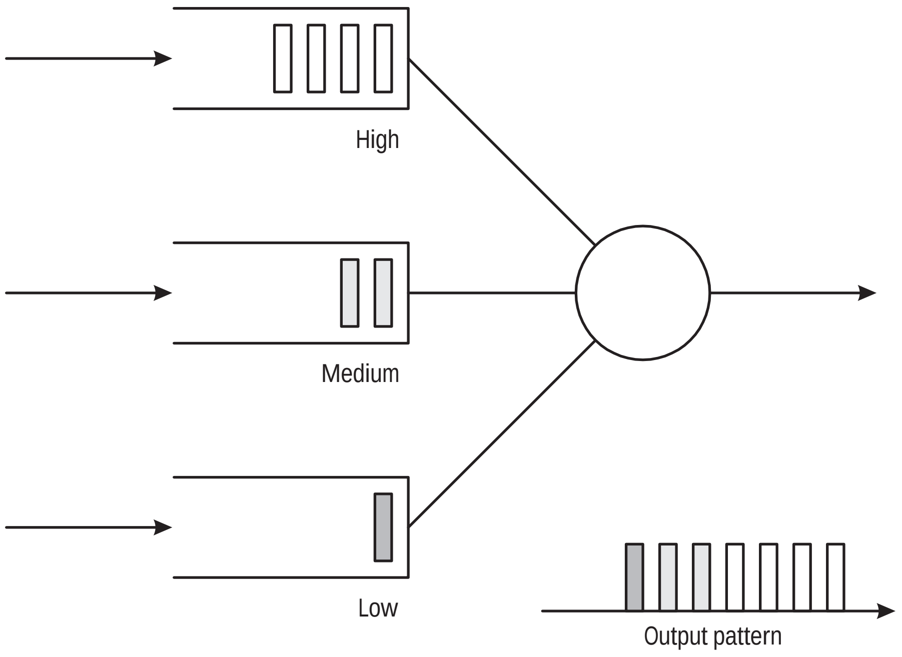

Figure 4.7 A three-level priority queuing system.

environment this could be indicated through an application programming interface (API) and then down to the appropriate protocol level.

However, in most of the implementations of priority queuing in multiprotocol networks, the priority level of the data packets is usually inferred based on some higher level indication of the application. This is usually satisfactory, but it sometimes may not be granular enough for the needs of the application and end users.

The priority queuing schemes generally base the decision of the priority level of the data packets on one of the indicators discussed next (the specific implementation depends on the particular type and manufacturer of the networking equipment).

Priority Level ID

Some packet level protocol headers carry explicit priority level bits. For example, the IPv4 packet headers carry a TOS field indicating special packet handling. The first three bits were designated as precedence bits and the next five indicated the type of service requested by the hosts (low delay, high throughput, and so on).

Until recently, this field has usually been ignored, however its use is beginning to gain popularity within the differential services architecture being proposed within the IETF. In this architecture, the TOS field is referred to as the DS octet and the first five bits are used to indicate priority packet handling at the network edge, with the sixth bit indicating whether this packet is conforming to a traffic contract or not. The rest of the bits are not used at this time. This, however, still requires that an entity (in this case the IP packet header processing entity) infer or be told the priority level to set. This can be done by other methods discussed later or could have been indicated through an API.

This method, where the priority level is directly indicated in the packet header, offers simplicity of protocol processing for the intermediate routers along the data path. Some of the schemes identified later require that the queuing entity search far into the higher level protocol headers to infer a priority level, and this can have a detrimental impact on the protocol processing entity (increase the processing delays at this location in the network).

Protocol ID

Packet level headers contain a field to indicate the higher level protocol carried within it. For example, in an IP header this is the protocol ID indicating TCP, UDP, ICMP, and so on. This works for our purpose in some situations, but it relies on a rather low-level protocol type in order to base priority queuing decisions for particular applications. For example, if it is desirable to give telnet terminal traffic a higher priority than file transfer protocol (FTP) traffic, then this scheme will not work because both of these higher level protocols are carried over a TCP transport-level connection. Hence, both telnet and FTP would receive the same priority level treatment under this method.

High-Level Protocol Identifier

Switches and routers can be programmed to look deeper into the protocol headers within the packet. One popular approach is to configure a router view the TCP or UDP port numbers, which identify the protocols riding over these transport-level protocols. Recently, this has been referred to as level 4 switching, to indicate the protocol level at which the routers are required to process in order to make switching or filtering decisions.

TCP port numbers indicate whether the data are from a telnet application or an FTP application. The router can then rely on the port numbers as an indication of the type of priority to be given to the data packet. This is valuable in that the higher level protocol indications are "closer" in some sense to the application and thus they give a better capability to discriminate file applications from transaction applications.

The price for this type of approach is that it is relatively more difficult for most high-speed switches or routers to search this deep into the protocol headers. This consumes more processing capabilities in the router and will, in turn, negatively impact the overall packet throughput on the router.

Protocol Encapsulation Type

Devices that operate at the link level, for example, FRADs or ATM switches, could conceivably rely on the protocol encapsulation type indication within the link level encapsulation protocols. An example is the encapsulation for frame relay networks defined in RFC 1490 (and discussed in Chapter 2). This would base the prioritization on the NLPID. Basing priority levels on the protocol encapsulation type would allow switches to prioritize based on the type of protocol. This is useful when, for example, you want to give all of your Novell NetWare Inter-Packet eXchange (IPX) traffic priority over TCP/IP applications (primarily carrying e-mail). The downside of this method is that you cannot prioritize, for example, telnet traffic over FTP transfers. Although this technique is occasionally discussed, it is not implemented in practice. One strong criticism of this approach is that it would violate the requirement that virtual circuits do not reorder the packets on the individual circuits.

Incoming Port or Interface

Suppose a router has two LAN interfaces, such as an Ethernet and a token ring. The token ring LAN carries primarily SNA traffic, and the Ethernet carries primarily native, non-real-time TCP/IP applications such as e-mail. Then, it would be useful for the router to give all packets incoming from the token ring LAN a high priority level and all packets incoming from the Ethernet LAN a low priority level.

However, like the case of basing priority levels on encapsulation types, this is a relatively nondiscriminating method for assigning priority levels. Also, unless there is some method of indicating the priority level on the packet, this method provides only local priority service; this does not extend across to the rest of the network.

Source/Destination Address Pair

Often traffic from a given source or traffic between a given source/destination pair is to be given priority handling. This can be accomplished in the routers by examining, for example, the IP addresses of the source and the destination indicated on the IP packet header. This is a fairly flexible form of identifying high-priority traffic in router networks.

Circuit ID

Virtual circuit switches base switching decisions on a circuit ID. They could just as well base a priority-level decision on a circuit ID. If the access device (for example, a FRAD) applied one of the other methods for assigning priority levels to packets and ran multiple virtual circuits to all other end points, then it could place the higher priority packets on one virtual circuit and the lower priority packets on another virtual circuit. This would extend the priority treatment of the data packet beyond the access facility and across the entire virtual circuit network.

Packet Length

Some systems have been known to assign priority levels based on packet size. The theory here is that terminal traffic is comprised of generally small packets while file transfer traffic is comprised of larger packets. This is not always true. Further, extreme care must be taken to ensure that the wrong packets are not reordered along the network connection. This could cause high levels of end-to-end retransmissions and greatly degrade the perceived performance of the network. In general, avoid these techniques for “guessing” at the priority level of the traffic.

The methods discussed for determining priority can be categorized by the type of discrimination, that is, explicit versus implicit, and the level in the protocol stack at which the discrimination occurs, for example, layer 2, layer 3, and so on. We summarize these in the Table 4.1.

As a rule of thumb, priority queuing methods are useful when accessing relatively low-speed facilities, for example, analog lines or 56-Kbps to fractional-T1 digital lines. At high speeds the relative benefits of priority

Table 4.1
Categories of the Priority Discrimination Schemes

Priority Determination	Discrimination Type	Discrimination Layer	Comments
Packet length	Implicit	Not applicable	Rarely implemented (thank goodness)
Port or interface	Implicit	Layer 1	Can prioritize one token ring versus Ethernet, or one department versus another
Source/destination address pair	Explicit	Layer 1	Prioritize all traffic from a given source or between a source/destination pair
Encapsulation indicator	Explicit	Layer 2	Can prioritize, for example, SNA over IP, however causes packet reordering on a VC, not implemented (thank goodness)
Protocol ID	Explicit	Layer 3	Can prioritize, for example, IPX over IP
High-level protocol ID	Explicit	Layer 4 (plus)	Can prioritize, for example, telnet over FTP

queuing are diminished (and sometimes are detrimental if the processing overhead incurred by enabling priority queuing greatly reduces the overall throughput of the networking device). Also, attempts should be made to ensure that the total amount of high-priority traffic is small relative to the amount of low-priority traffic. A good rule is to keep the high-priority traffic to less than 10% of the low-priority traffic. After all, if all the traffic were treated as high priority then the system would degenerate to a simple, single queue system.

Low-priority queue "starvation" can also occur if the higher priority traffic load becomes too great. This may cause retransmit timers associated with low-priority traffic to expire and greatly degrade the performance of these applications.

4.4.4 Processor Sharing

Suppose that routers were to put packets into different buckets (depending on some criteria), and then transmit one packet at a time from each bucket onto the transmission facility. If a given bucket were empty, then it would simply jump to the next bucket containing a data packet. We refer to this type of packet transmission system as a *processor sharing* system. (A related system is referred to as a *weighted fair queuing* algorithm; see the discussion in the following paragraphs.)

A processor sharing system is fundamentally different from a priority queuing system in that a processor sharing system treats all types of packets (or traffic) equally in some sense. A priority queuing system, however, explicitly favors one type of packet (or traffic) over other, lower priority traffic type.[3] This is shown in Figure 4.8. Notice the predicted output pattern for this processor sharing system and compare it to the predicted output pattern in Figure 4.7 for the priority queuing system.

Here, the processor sharing server effectively interleaves the packets from the various packet buffers. In this sense it behaves differently from a priority queuing system. The processor sharing system treats all queues equally by transmitting a packet from each occupied queue in a round-robin fashion. The

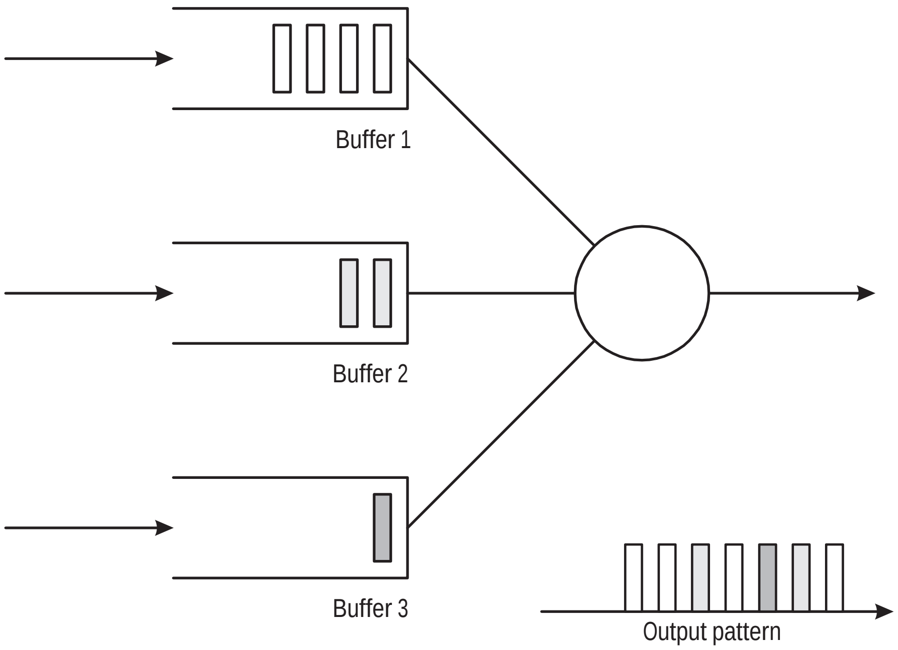

Figure 4.8 A three-bucket, packet-level processor sharing system.

3. Other types of processor sharing systems exist where the data from each bucket are essentially transmitted a byte at a time. These are referred to as byte-interleaving, processor sharing schemes, as opposed to the packet-interleaving, processor sharing schemes discussed earlier.

priority queuing system explicitly favors those packets in the queue designated high priority.

Some processor sharing schemes allow for a weight to be assigned to specific flows. This gives a larger proportion of the bandwidth to some flows over others. For example, flow A can be assigned twice the bandwidth as the other flows by serving two packets from flow A's queue versus one packet for each visit to the other queues. Various schemes like these exist, for example, proportional round-robin, weighted processor sharing, or weighted fair queuing algorithms, which are variations on the processor sharing algorithm.

Assume that a single transaction packet arrives at a processor sharing system and is placed into an empty bucket.[4] In this case (that of a single transaction packet), the maximum queuing delay would be a file transfer packet insertion delay of $N - 1$ packets, where N is the number of active, separate queues in the processor system.

In contrast, the maximal delay in the priority queuing system would be a single file transfer packet insertion delay. The factor of $N - 1$ arises because the processor sharing system treats all queues equally. This maximal delay for active queues would occur if all the other queues were occupied and the transaction packet is queued just following its "turn" in the processor cycle. If the transaction packet arrived to its queue just prior to its turn, then its delay would be roughly zero. Hence, the average delay would be one-half the maximum delay: $(N - 1) \times (\text{Insertion delay}) / 2$.

As an example, for a WAN facility running at 56 Kbps and a single active file transfer with a packet size of 560 bytes, the average delay, under processor sharing, for a packet belonging to an interactive traffic stream that is queued separately is roughly 40 msec (that is, half of 80 msec, the insertion delay of a 560-byte packet on a 56-Kbps link). This would not be noticeable to the end user of the transaction application. For two simultaneously active file transfers this delay would double; for three, it would triple; and so on. On the other hand, if interactive traffic is explicitly given highest priority, then the maximum queuing delay (due to packets from other traffic streams) for an interactive packet will be half the full insertion delay, *no matter how many queues are maintained and active.*

Although processor sharing is not as effective at reducing the transaction delays as a priority queuing scheme, it does have several advantages. Like priority queuing schemes, a processor sharing scheme can be implemented based on very low-level indications, for example, virtual circuit identifiers, packet level

4. In the case where the transaction arrives at a nonempty queue, the delay bounds discussed in this paragraph are increased by the number of transaction packets in queue ahead of the transaction in question times $(N - 1)$ packet insertion delays.

flow identifiers, or packet destination or source/destination pairs. Processor sharing is effective at sharing bandwidth across all active circuits or flows, and avoids “starvation” effects, which can occur in priority queuing systems. Also, by assigning unequal weights to the different buffers, a form of favoritism can be assigned to “higher priority” traffic.

4.4.5 Adaptive Controls

Adaptive controls attempt to minimize the length of the queues at the resource in contention by monitoring, either implicitly or explicitly, the state of the buffers. In our example, where a large file transfer and a small transaction are contending for a common communication facility, a large queue develops when the file source is sending too much data into the network at any given time. These data end up sitting in the buffers for access to the common network facility, which causes an excessive delay for a small, delay-sensitive transaction. By minimizing the length of this queue, an adaptive control scheme can improve the transaction delays.

There are basically two mechanisms that adaptive controls can apply to manage queue buildup: dynamically adjusting the rate at which the source transmits into the network (commonly referred to as *traffic shaping*) or dynamically adjusting the transmission window size of the source. When the queue is determined to be building up too large, then the rate or the window can be decreased. In the event that the queue is determined to be extremely small, then the rate or the window of the transmitter can be increased. By constantly monitoring and adjusting the rate or window size, the system can achieve a reasonable trade-off between high throughputs and low delays.

The adaptive source requires some form of feedback on the state of the network path in order to make the appropriate changes to the transmitter. The form of the feedback can vary, being either explicit or implicit. This is shown in Figure 4.9.

The lower path shows an explicit congestion indication being sent to the source. Receipt of this message would then allow the transmitter to either decrease its transmission rate (either by spacing out its packets or decreasing its transport window) or increase it depending on the nature of the message.

Several methods exist to implement this explicit notification. One method would have the congested resource generate a signaling message and transmit it to the source of the traffic. Another method would have the congested resource setting a bit (or bits) in the protocol header of the data packets as they traverse the congested resource. The receiver would then pass this back to the transmitter through a similar indication on the packet headers.

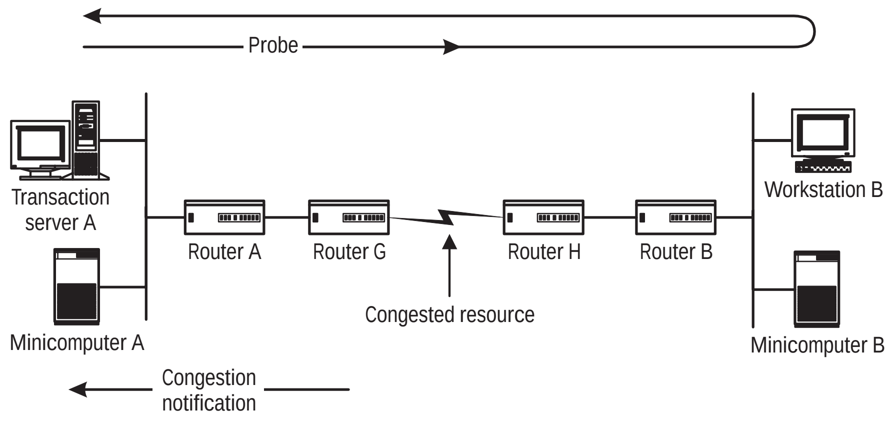

Figure 4.9 Implicit and explicit feedback on network congestion.

The upper loop refers to an implicit method of sensing network congestion. One example of this method is to base adaptive decisions on a round-trip time measurement. Here the "probe" could be a single data packet in the forward direction and the acknowledgment in the reverse direction. Then, the onset of congestion in the data path would cause delays in queuing and increase the measured round-trip time. The transmitter could then slow down its transmission rate or decrease the size of its transport window. This would help to alleviate the congested resource. Another method of implicitly sensing network problems is to rely on packet loss to indicate congestion.

An example is the TCP slow start adaptive algorithm (see the discussion in [1]). Here, the transmitter would sense a lost packet through the windowing acknowledgments (or lack thereof) and would infer that the packet was dropped due to congestion. The transmitter would then decrease its transmission rate (through one of several methods already discussed). This is not a direct measure of network congestion because packet loss can occur in networks due to other reasons, for example, bit errors on transmission facilities or misrouted packets.

By adapting the transmission rate based on some form of feedback from the network on its congestion state, transmitters attempt to keep packet buffers from growing too large. As we have discussed, this will help in reducing the network delays for all data including transaction-oriented application data. This will help to reduce the negative, cross impact of file transfer traffic on transaction data. We have already discussed similar algorithms in Chapter 2 on frame relay congestion control methods.

4.4.6 Selective Discards

Selective discard mechanisms discriminate among various sources of traffic during periods of extreme resource congestion by discarding packets from select sources while not discarding from other sources. It is argued that this is a method of reserving critical network resources during congestion periods for time-sensitive or high-priority applications.

This reservation mechanism implicitly reserves resources for critical applications by explicitly denying access to noncritical applications. This scheme relies on the same methods to identify those applications deemed noncritical as the other schemes discussed in the previous sections. Often noncritical applications are considered to be those utilizing certain TCP port numbers, for example, FTP is deemed noncritical, while telnet is deemed critical.

Some implementations of selective discard rely on the specific switching equipment to look deep into the packet headers to determine the nature of the applications and to determine which packets to discard during resource overload. Other schemes have end systems tagging the packets as discard eligible through some network protocol-specific indication. Others tag packets based on a bandwidth contract as measured at the access control point into a network. One such implementation, known as *discard eligibility*, is found within the frame relay network standards as discussed in Chapter 2.

The selective discard strategy is fundamentally different than the other methods discussed within this chapter. The trigger for the selective discard action to begin is usually a buffer congestion threshold. These thresholds are usually set to a significant fraction of the total buffer space allocated to the resource. Otherwise, the majority of this buffer space is wasted. Once the threshold is reached, the selective discard mechanism is initiated until the buffer utilization drops below a lower level congestion threshold (in order to prevent thrashing). What is different with this scheme is that it is responding to large buffer utilization, instead of responding to delay and bandwidth considerations. Long before the time the buffers have reached their threshold, the delays for the time-sensitive applications have exceeded reasonable limits. Therefore, this should not be considered a useful mechanism to ensure good transaction application performance in multiprotocol networks.

Often heard in discussions of packet discard techniques is the random early detection (RED) strategy implemented in many IP-based router networks. This is a technique in which routers attempt to mitigate the onset of router congestion by initiating a limited level of random packet discards based on buffer threshold settings. By performing a low level of packet discards, the router is relying on throttling back TCP slow start implementations and damping the load somewhat on the router. This has been shown to be a very effective technique in maintaining high network utilization while mitigating the effects of congestion. From an Internet provider perspective, RED provides significant benefits to their overall customer base. However, from a critical end-applications perspective (which is the focus of this section and, in fact, this book), RED does not significantly benefit particular applications by providing significant differentiated service. These are fundamentally different goals.

4.5 Data Collection

We have discussed application characteristics, load engineering and capacity management, and methods to give preferential treatment of one application type over another when designing and implementing data networks. However, to deliver on a capacity management plan and to maintain the desired performance engineering, one needs to collect data on various aspects of the network. Data collection techniques are discussed in this section. Here we identify various types of data collection tools.

To help accomplish the tasks that arise only occasionally, specialized monitoring equipment, such as LAN/WAN analyzers or protocol traces, can be utilized. However, for ongoing tasks, such as capacity management, one needs to rely primarily on the existing measurement capabilities of the networking equipment and network management systems employed. We briefly discuss three types of collection tools: LAN/WAN analyzers, SNMP management tools, and RMON2 probes. We end this section with a brief classification of commercially available tools that are useful in this context.

4.5.1 LAN/WAN Analyzers

LAN/WAN analyzers are devices that can tap into the LAN/WAN and capture the data being transmitted over the LAN/WAN segment. These devices store some aspects of the data onto disk to be retrieved at a later date. Sniffers capture all the data, but usually one applies filters to look at some particular aspect of the information of use in characterizing the traffic load and in developing an understanding of the protocol models required for trouble shooting. LAN/WAN analyzers typically store the link level (and can be configured to capture higher level protocol information such as the IP or IPX) packet formats, the byte count of the data fields, whether the packets were received in error, and time stamps showing the time at which the packet data were captured.

From this, the analyst can compute the interesting statistics associated with the data traffic, for example, average and standard deviation of the packet sizes, the traffic loads, and even the point-to-point requirements by mapping end-system addresses in the packet headers to the destination locations.

Another important use of LAN/WAN analyzers is traffic characterization of specific applications, particularly client/server applications. This is of great utility as discussed in Section 4.3.

This equipment tends to be rather specialized and therefore relatively expensive. However, it is not necessary to widely deploy this equipment, and it is mostly used on an individual case basis for specific reasons. These are typically used to help resolve problems, to help build an understanding of traffic load over a given, finite period of time, or to characterize specific applications prior to generally deploying the application on the enterprise network.

4.5.2 Network Management Systems and RMON Probes

Here we focus on those aspects of management systems that capture and store traffic data in networks on an ongoing basis. This is a distributed functionality, in that individual network components are required to collect local information, then periodically transmit this information to a centralized network management system that can process this information into quantities of utility to the network analysts and administrators.

Local information typically collected on the individual network components includes the following:

- *Serial trunk interfaces:* the number of bytes and frames transmitted and received, the number of errored frames transmitted and received, the number of frames discarded due to buffer overflows, and so on;
- *Multiplexed interfaces* (such as frame relay or ATM): the number of bytes and frames transmitted and received, the number of errored frames transmitted and received, the number of frames discarded due to buffer overflow on an individual virtual connection basis, the number of established switched connections on an interface basis, and so on;
- *Switch processors:* the number of frames or cells forwarded over the switch fabric, the utilization of call processors, the size of the run queue in the processor, and so on;
- *Routers:* the number of packets forwarded as a function of the protocol type, for example, IP, IPX, AppleTalk, the utilization of the routing processor, the size of the routing tables, and so on; and
- *Network servers:* the utilization of the processor and disk, the number of simultaneous sessions established, and so on.

These parameters are stored locally and are periodically (in periods ranging from 5 min to 1 h) transmitted to a centralized management system when prompted by the management station. The management system will process and summarize the information and archive the results for trend analyses by the network analysts. Useful trend analyses might include the weekly variations in the utilization of a trunk interface into the WAN or the monthly increase on the peak utilization or the variation in the utilization of a network server, as discussed in Section 4.2. Useful troubleshooting information includes the number of frames or packets received in error or the number of frames discarded due to buffer overflows on a particular hour of a given day.

The trend today is to try to standardize the type of information to be collected and stored by the network management systems. The type of information to be collected is found in an SNMP management information base (MIB) for the particular device or technology in question. Many of the MIB definitions are standard, but many vendors have developed proprietary extensions that are specific to their particular equipment.

The types of information identified in the preceding list are local to particular devices comprising the network. As mentioned in the preceding paragraph, this type of information is captured within the MIB for the device. A standard MIB common across all SNMP manageable devices is defined and referred to as MIB II [2]. Because this type of information is local to a specific device, it is up to a central management system to correlate these individual pictures into a view across a specific subnet or LAN segment.

To eliminate this burden on a central management system, and to provide a richer set of subnet-based statistics, the RMON MIB was developed. The IETF has developed a set of recommendations for a standard remote monitoring capability. Hardware devices that implement these recommendations for a standard set of subnetwork monitors are referred to as *RMON probes.* RMON2 extends the monitoring capabilities of the RMON above the MAC layer.

We prefer to think of these additional RMON2 capabilities in terms of three different levels of traffic analysis. Level 1 of the traffic analysis is overall byte count and average packet sizes. Level 2 of the traffic analysis is a breakdown via layer 3 protocols, for example, IP versus IPX, DECnet, AppleTalk. Level 3 of the traffic analysis is a detailed analysis of IP traffic into TCP and UDP components along with further breakdown of TCP traffic according to port numbers and so on. This information, collected from networks and stored within the probes can be communicated to a central management system through the SNMP [3].

For these reasons, RMON2 probes offer a rich set of collection capabilities, which are extremely useful in capacity engineering. For more information on RMON2 probes see [2].

It is the job of the analyst to develop a capacity management strategy that relies primarily on the information types and statistics collected as identified in the MIBs available in the components deployed within their data network. As such, the MIBs must necessarily contain the required local information that the analyst uses to develop an understanding of the important end-to-end performance requirements of the applications riding over the multiprotocol network.

The capabilities provided by these activities in the IETF and afforded to a network management tool by RMON2 (or similar function) probes have led to the development of a host of monitoring, trending, and performance management tools. These tools can help the analyst by delivering much of the capabilities required in their capacity management needs. These are discussed next.

For an excellent discussion of IP management standards and MIBs, refer to [3], and [2] for RMON.

4.5.3 A Taxonomy of Commercially Available Tools for Performance Engineering Data Networks

As mentioned in this section on data collection, several commercially available tools cover all aspects of data networking: configuration management, equipment inventory, capacity planning, application analysis, performance modeling, troubleshooting, and traffic generation.

Although all of these tools are important in their own right, we focus below on a set of tools that we think is indispensable in performance engineering data networks, especially WANs. The objective is to provide the reader with a high-level view of the tool's capabilities and strengths. In this process, it is likely that some important aspects of the tools may not be mentioned. For more details, the reader is referred to vendors' Web sites.

We classify tools for performance engineering as follows:

- Sniffers;
- Capacity management tools;
- Application analysis tools;
- Predictive modeling tools.

We discuss each type separately next and give some examples of each.

Sniffers

Sniffers are passive devices that can be placed on a LAN or WAN segment to collect traffic statistics and packet-level information on the segment. They can be set up with various filters to selectively collect data. For instance, one can isolate specific conversations between a client and a server by filtering on their MAC or IP addresses.

Sniffers are indispensable for performance engineering for two reasons: troubleshooting and application characterization. Many examples in the later chapters of this book contain descriptions of how sniffer protocol traces were used to troubleshoot performance problems and to characterize application behavior over WANs. Some tools even have the capability of directly reading sniffer protocol traces for performance modeling purposes.

The flagship tools in this category are the Network Associates Sniffer® and Wandel & Goltermann's Domino®.

Capacity Management Tools

Tools in this category are primarily software based. They rely on network components (routers, bridges, hubs, CSU/DSUs, servers, and so on) to collect measurement data in accordance with the IETF's SNMP/RMON/RMON2 standards and MIBs. Centralized network management stations poll the network components for information on the MIB variables and report on all aspects of network capacity, network congestion, application usage, and so on.

Examples of such tools are Concord Communications' Network/Router Health/Traffic Accountant®, INS's E-Pro®, Cabletron's Spectrum®, and NetScout Systems RMON2 probes and NetScout Manager.

Some tools from vendors such as Visual Networks and Paradyne can be used for capacity planning and troubleshooting. However, they are hardware and software based. The hardware is in the form of a "smart" DSU interfacing with the WAN.

Application Analysis Tools

These tools use self-generated measurements and probes to estimate application-level performance. They rely on various application profiles that capture the essence of the application-level traffic flows to measure expected application performance.

These tools require a centralized management server as well as the deployment of distributed software clients at the remote locations from which performance measurements are desired. Examples of this type of tool include Ganymede's Pegasus® tool.

Predictive Modeling Tools

These tools are either simulation tools or analytical modeling tools. Simulation tools build a logical model of the network and the applications in question, and run a simulation collecting the performance metrics of interest. Examples of these tools include MIL 3's OPNET® and IT Decision Guru®. Analytical modeling builds reference connections in the network and, along with an application profile input, provides answers to "what if" scenarios such as increased bandwidth and protocol tuning. A prime example of analytical tools is the Application Expert from Optimal® Networks.

Other predictive tools have or build application profiles and deploy these profiles on remote software clients. These profiles are then run from the remote client over the existing network and performance metrics of interest are collected. Ganymede's Chariot® tool is a good example of such a tool.

4.6 An Example: Deploying New Applications

We end this chapter by providing a high-level discussion of the necessary steps that one should take before deploying a new critical application over an existing WAN. This discussion ties together all of the techniques presented within this chapter.

The analysis and design of data networks and their performance evaluations are only as good as the inputs to the design and the ongoing monitoring of the implemented network. Similarly, when developing the engineering rules for the data network to support the rollout of a new application, the effectiveness of the engineering rules is only as good as the input to the engineering models.

We recommend the following four steps in the context of deploying new applications over a WAN.

Application Characterization

Characterize the new application to be carried over the network and its performance requirements, for example, delay and throughputs. When entering into the process of redesigning a network to support a new application, it is important to have first built up an understanding of the nature of the specific application in question.

We are strong proponents of a top-down approach, as far as possible, to design, engineer, and maintain data networks. This begins with time spent in defining the performance requirements of this application. Networks are built to support a variety of applications and application types.

Applications range from highly transactional, or latency-sensitive, applications where delay is critical, to bulk data transfer, or bandwidth-sensitive, applications where throughput is critical, or hybrid applications, which share the characteristics of sensitivity to latency and bandwidth. Various tools are available to the engineer to help in characterizing the behavior of the application. These include sniffers, product literature, and vendor support.

Ideally, at the conclusion of this first phase, the network engineer should be able to draw a timing diagram for the important network applications. This level of detail will aid in (1) the redesign phase in determining if the design will meet the application-level performance requirements and (2) the capacity management phase in developing a component-level performance allocation. However, it is not always possible to develop this level of detail, depending on the specific application in question.

Traffic Matrix

Identify the traffic loads and develop a point-to-point traffic matrix related to the new application. Once the application characterization is complete, the next stage is the characterization of the traffic, including its arrival patterns, its point-to-point flows, and its offered load.

This information should be presented in terms of the point-to-point traffic loads between the data sources and the data sinks. This is best built up from the volume of traffic a given application transaction or file transfer produces and the frequency of the transmissions of these transactions between all sources and destinations during the busiest period of the day. Often these traffic volumes must be estimated based on issues such as (1) assumptions regarding number of users at a given location, (2) assumptions regarding usage concurrence, and (3) best case and worst case bounds.

The sum total of this information is the traffic matrix. The traffic matrix is used to determine the additional network connectivity and bandwidth requirements during the network redesign phase.

Network Redesign

Redesign the network and identify the affected network components to support the new application characteristics and traffic. The first step in the network redesign phase is to determine the specific network components that will be affected by the deployment of the new application. At a minimum, the engineer must identify the components whose capacity must be increased to support the bandwidth and connectivity requirements specified in the new traffic matrix.

At this point, the engineer should have a network layout, including the expected loads on each of the network components, for example, links and PVCs. The engineer can now develop the expected delays and throughputs for the dominant applications to be carried over the design. This is accomplished by developing the appropriate timing diagrams for these applications over typical reference connections within the network layout. From the timing diagrams, one can determine the realizable end-to-end delays and throughputs. If these realizable delay and throughput estimates meet the initial set of performance requirements for the applications, then the initial design is finalized. If the realizable performance estimates do not meet the initial requirements, then the design must be modified until the performance objectives are met.

If the new applications are characterized as latency sensitive, and must compete for network resources with various bandwidth-sensitive applications, then the engineer should consider the various methods of traffic discrimination, which were discussed in Section 4.4.

Capacity Management

Modify the existing capacity management tactics to maintain acceptable network performance for the existing and new applications. Redesigning a network in this a priori fashion is useful but is never sufficient. New users and servers are constantly added to existing networks. This has the effect of changing the nature of the traffic flows over the network and hence changing the network performance characteristics. Also often it is impossible to fully characterize the new application being deployed to the level of detail suggested earlier. Therefore the network analyst/network administrator must design a strategy to continuously monitor the network and plan its growth in order to maintain the desired level of performance as changes to traffic occur or as an understanding of the nature of the traffic improves.

Ideally, the capacity management strategy is devised by reverse engineering the desired end-to-end delays or throughputs into the individual network component metrics that can be monitored effectively. This is accomplished in two parts.

First, the end-to-end delays must be mapped into the individual component delays. We refer to this as *component allocation*. Next, the component delays must be mapped to component metrics, which are measurable. It is generally not possible to directly measure component delays. Therefore, these must be mapped to directly measurable quantities, such as the component utilization. This relies on load engineering and associated load engineering models.

Consider these four steps as a recipe for performance engineering when deploying a new application. Each step is as important as the next and none should be skipped when designing an enterprise network from scratch or when deploying a new application onto an existing network. However, it must be remembered that these guidelines cannot be fully realized for all applications being deployed. Consider the rollout of a company-wide Microsoft Exchange® e-mail platform, or an IBM/Lotus Notes® deployment, or an Intranet Web-based application. For these applications, it will be very hard, if not impossible, to obtain traffic matrices, usage patterns, and performance requirements prior to their deployment. For other applications, for example, the deployment of client/server applications, this approach may be more fully realized.

4.7 Summary

In this chapter, we covered several topics regarding general performance engineering techniques. The first topic was load engineering where we discussed the definition of directly measurable component thresholds and their relationship to maintaining acceptable network performance. We next discussed latency- and bandwidth-sensitive applications and the problems of simultaneously supporting both on a common enterprise network. This led us to list the various traffic discrimination techniques available to minimize negative cross-application effects. Following this we presented a section on tools for maintaining data networks. This touched on tools for monitoring, capacity management, network planning, and troubleshooting. We ended this section with a brief discussion of a method to follow for the deployment of a new application over an existing data network.

References

[1] Stevens, W. R., *TCP/IP Illustrated, Volume 1: The Protocols*, Reading, MA: Addison-Wesley, 1994.

[2] Stallings, W., *SNMP, SNMPv2 and RMON*, Reading, MA: Addison-Wesley, 1996.

[3] Rose, M. T., *The Simple Book: An Introduction to Management of TCP/IP-Based Internets*, Englewood Cliffs, NJ: Prentice Hall, 1989.