Second Edition

Data Networks

DIMITRI BERTSEKAS

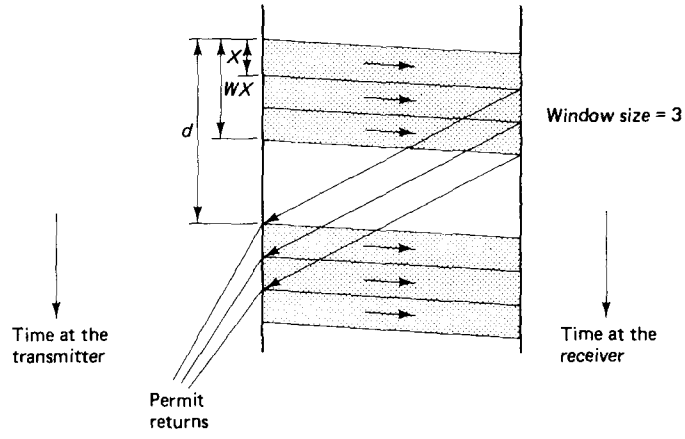
Massachusetts Institute of Technology

ROBERT GALLAGER

Massachusetts Institute of Technology

PRENTICE HALL, Englewood Cliffs, New Jersey 07632

6



Flow Control

6.1 INTRODUCTION

In most networks, there are circumstances in which the externally offered load is larger than can be handled even with optimal routing. Then, if no measures are taken to restrict the entrance of traffic into the network, queue sizes at bottleneck links will grow and packet delays will increase, possibly violating maximum delay specifications. Furthermore, as queue sizes grow indefinitely, the buffer space at some nodes may be exhausted. When this happens, some of the packets arriving at these nodes will have to be discarded and later retransmitted, thereby wasting communication resources. As a result, a phenomenon similar to a highway traffic jam may occur whereby, as the offered load increases, the actual network throughput decreases while packet delay becomes excessive. It is thus necessary at times to prevent some of the offered traffic from entering the network to avoid this type of congestion. This is one of the main functions of flow control.

Flow control is also sometimes necessary between two users for speed matching, that is, for ensuring that a fast transmitter does not overwhelm a slow receiver with more packets than the latter can handle. Some authors reserve the term "flow control" for this

type of speed matching and use the term "congestion control" for regulating the packet population within the subnetwork. We will not make this distinction in terminology; the type and objective of flow control being discussed will be clear from the context.

In this chapter we describe schemes currently used for flow control, explain their advantages and limitations, and discuss potential approaches for their improvement. In the remainder of this section we identify the principal means and objectives of flow control. In Sections 6.2 and 6.3 we describe the currently most popular flow control methods; window strategies and rate control schemes. In Section 6.4 we describe flow control in some representative networks. Section 6.5 is devoted to various algorithmic aspects of rate control schemes.

6.1.1 Means of Flow Control

Generally, a need for flow control arises whenever there is a constraint on the communication rate between two points due to limited capacity of the communication lines or the processing hardware. Thus, a flow control scheme may be required between two users at the transport layer, between a user and an entry point of the subnet (network layer), between two nodes of the subnet (network layer), or between two gateways of an interconnected network (internet layer). We will emphasize flow control issues within the subnet, since flow control in other contexts is in most respects similar.

The term "session" is used somewhat loosely in this chapter to mean any communication process to which flow control is applied. Thus a session could be a virtual circuit, a group of virtual circuits (such as all virtual circuits using the same path), or the entire packet flow originating at one node and destined for another node. Often, flow control is applied independently to individual sessions, but there is a strong interaction between its effects on different sessions because the sessions share the network's resources.

Note that different sessions may have radically different service requirements. For example, sessions transferring files may tolerate considerable delay but may require strict error control, while voice and video sessions may have strict minimum data rate and maximum end-to-end delay requirements, but may tolerate occasional packet losses.

There are many approaches to flow control, including the following:

1. *Call blocking.* Here a session is simply blocked from entering the network (its access request is denied). Such control is needed, for example, when the session requires a minimum guaranteed data rate that the network cannot provide due to limited uncommitted transmission capacity. A typical situation is that of the voice telephone network and, more generally, circuit switched networks, all of which use flow control of this type. However, a call blocking option is also necessary in integrated voice, video, and data networks, at least with respect to those sessions requiring guaranteed rate. In a more general view of call blocking one may admit a session only after a negotiation of some "service contract," for example, an agreement on some service parameters for the session's input traffic (maximum rate, minimum rate, maximum burst size, priority level, etc.)

2. *Packet discarding.* When a node with no available buffer space receives a packet, it has no alternative but to discard the packet. More generally, however, packets may be discarded while buffer space is still available if they belong to sessions that are using more than their fair share of some resource, are likely to cause congestion for higher-priority sessions, are likely to be discarded eventually along their path, and so on. (Note that if a packet has to be discarded anyway, it might as well be discarded as early as possible to avoid wasting additional network resources unnecessarily.) When some of a session's packets are discarded, the session may need to take some corrective action, depending on its service requirements. For sessions where all packets carry essential information (*e.g.*, file transfer sessions), discarded packets must be retransmitted by the source after a suitable timeout; such sessions require an acknowledgment mechanism to keep track of the packets that failed to reach their destination. On the other hand, for sessions such as voice or video, where delayed information is useless, there is nothing to be done about discarded packets. In such cases, packets may be assigned different levels of priority, and the network may undertake the obligation never to discard the highest-priority packets—these are the packets that are sufficient to support the minimum acceptable quality of service for the session. The data rate of the highest-priority packets (the minimum guaranteed rate) may then be negotiated between the network and the source when the session is established. This rate may also be adjusted in real time, depending on the congestion level in the network.

3. *Packet blocking.* When a packet is discarded at some node, the network resources that were used to get the packet to that node are wasted. It is thus preferable to restrict a session's packets from entering the network if after entering they are to be discarded. If the packets carry essential information, they must wait in a queue outside the network; otherwise, they are discarded at the source. In the latter case, however, the flow control scheme must honor any agreement on a minimum guaranteed rate that the session may have negotiated when it was first established.

4. *Packet scheduling.* In addition to discarding packets, a subnetwork node can exercise flow control by selectively expediting or delaying the transmission of the packets of various sessions. For example, a node may enforce a priority service discipline for transmitting packets of several different priorities on a given outgoing link. As another example, a node may use a (possibly weighted) round-robin scheduling strategy to ensure that various sessions can access transmission lines in a way that is consistent both with some fairness criterion and also with the minimum data rate required by the sessions. Finally, a node may receive information regarding congestion farther along the paths used by some sessions, in which case it may appropriately delay the transmission of the packets of those sessions.

In subsequent sections we discuss specific strategies for throttling sources and for restricting traffic access to the network.

6.1.2 Main Objectives of Flow Control

We look now at the main principles that guide the design of flow control algorithms. Our focus is on two objectives. First, *strike a good compromise between throttling sessions (subject to minimum data rate requirements) and keeping average delay and buffer overflow at a reasonable level.* Second, *maintain fairness between sessions in providing the requisite quality of service.*

Limiting delay and buffer overflow. We mentioned earlier that for important classes of sessions, such as voice and video, packets that are excessively delayed are useless. For such sessions, a limited delay is essential and should be one of the chief concerns of the flow control algorithm; for example, such sessions may be given high priority.

For other sessions, a small average delay per packet is desirable but it may not be crucial. For these sessions, network level flow control does not necessarily reduce delay; it simply shifts delay from the network layer to higher layers. That is, by restricting entrance to the subnet, flow control keeps packets waiting outside the subnet rather than in queues inside it. In this way, however, flow control avoids wasting subnet resources in packet retransmissions and helps prevent a disastrous traffic jam inside the subnet. Retransmissions can occur in two ways: first, the buildup of queues causes buffer overflow to occur and packets to be discarded; second, slow acknowledgments can cause the source node to retransmit some packets because it thinks mistakenly that they have been lost. Retransmissions waste network resources, effectively reduce network throughput, and cause congestion to spread. The following example (from [GeK80]) illustrates how buffer overflow and attendant retransmissions can cause congestion.

Example 6.1

Consider the five-node network shown in Fig. 6.1(a). There are two sessions, one from top to bottom with a Poisson input rate of 0.8, and the other from left to right with a Poisson input rate f. Assume that the central node has a large but finite buffer pool that is shared on a first-come first-serve basis by the two sessions. If the buffer pool is full, an incoming packet is rejected and then retransmitted by the sending node. For small f, the buffer rarely fills up and the total throughput of the system is $0.8 + f$. When f is close to unity (which is the capacity of the rightmost link), the buffer of the central node is almost always full, while the top and left nodes are busy most of the time retransmitting packets. Since the left node is transmitting 10 times faster than the top node, it has a 10-fold greater chance of capturing a buffer space at the central node, so the left-to-right throughput will be roughly 10 times larger than the top-to-bottom throughput. The left-to-right throughput will be roughly unity (the capacity of the rightmost link), so that the total throughput will be roughly 1.1. This is illustrated in more detail in Fig. 6.1(b), where it can be seen that the total throughput decreases toward 1.1 as the offered load f increases.

This example also illustrates how with buffer overflow, some sessions can capture almost all the buffers and nearly prevent other sessions from using the network. To avoid this, it is sometimes helpful to implement a *buffer management scheme*. In such a scheme, packets are divided in different classes based, for example, on origin, destination,

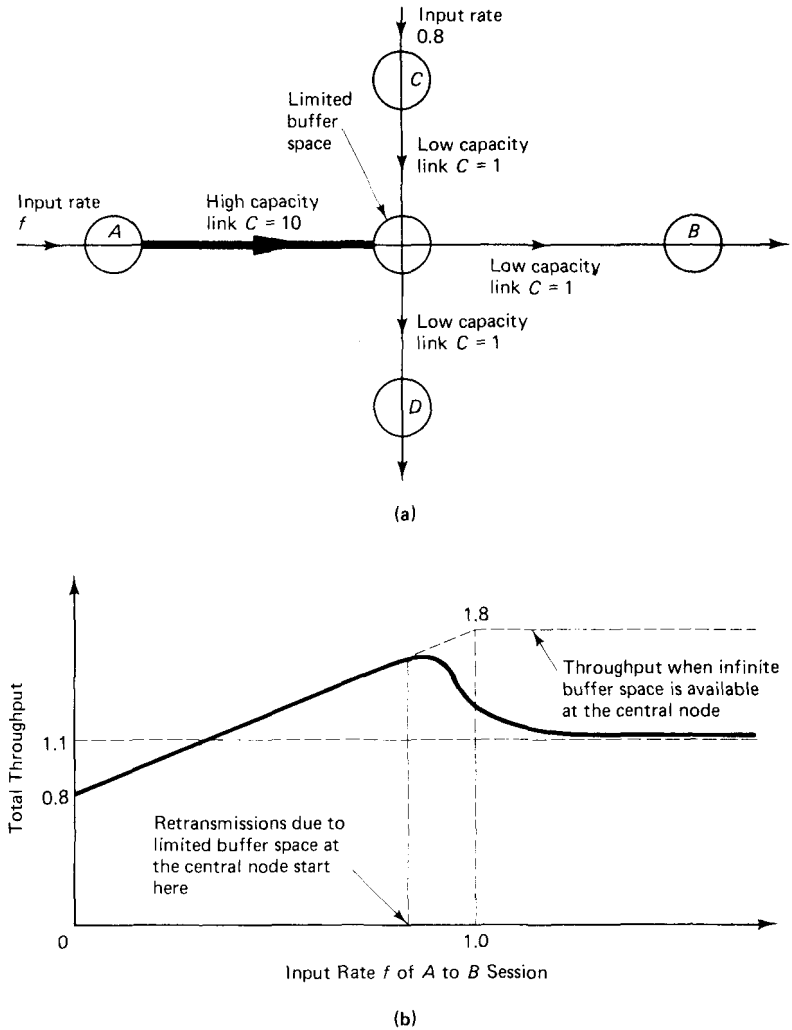


Figure 6.1 Example demonstrating throughput degradation due to retransmissions caused by buffer overflow. (a) For f approaching unity, the central node buffer is almost always full, thereby causing retransmissions. Because the A-to-B session uses a line 10 times faster than the C-to-D session, it has a 10-fold greater chance of capturing a free buffer and getting a packet accepted at the central node. As a result, the throughput of the A-to-B session approaches unity, while the throughput of the C-to-D session approaches 0.1. (b) Total throughput as a function of the input rate of the A-to-B session.

or number of hops traveled so far; at each node, separate buffer space is reserved for different classes, while some buffer space is shared by all classes.

Proper buffer management can also help avoid deadlocks due to buffer overflow. Such a deadlock can occur when two or more nodes are unable to move packets due to unavailable space at all potential receivers. The simplest example of this is two

nodes A and B routing packets directly to each other as shown in Fig. 6.2(a). If all the buffers of both A and B are full of packets destined for B and A, respectively, then the nodes are deadlocked into continuously retransmitting the same packets with no success, as there is no space to store the packets at the receiver. This problem can also occur in a more complex manner whereby more than two nodes arranged in a cycle are deadlocked because their buffers are full of packets destined for other nodes in the cycle [see Fig. 6.2(b)]. There are simple buffer management schemes that preclude this type of deadlock by organizing packets in priority classes and allocating extra buffers for packets of higher priority ([RaH76] and [Gop85]). A typical choice is to assign a level of priority to a packet equal to the number of links it has traversed in the network, as shown in Fig. 6.3. If packets are not allowed to loop, it is then possible to show that a deadlock of the type just described cannot occur.

We finally note that when offered load is large, limited delay and buffer overflow can be achieved only by lowering the input to the network. Thus, there is a natural trade-off between giving sessions free access to the network and keeping delay at a level low enough so that retransmissions or other inefficiencies do not degrade network performance. A somewhat oversimplified guideline is that, ideally, flow control should not be exercised at all when network delay is below some critical level, and, under heavy load conditions, should reject as much offered traffic as necessary to keep delay at the critical level. Unfortunately, this is easier said than done, since neither delay nor throughput can be represented meaningfully by single numbers in a flow control context.

Fairness. When offered traffic must be cut back, it is important to do so fairly. The notion of fairness is complicated, however, by the presence of different session priorities and service requirements. For example, some sessions need a minimum guaranteed rate and a strict upper bound on network delay. Thus, while it is appropriate to consider simple notions of fairness within a class of "similar" sessions, the notion of

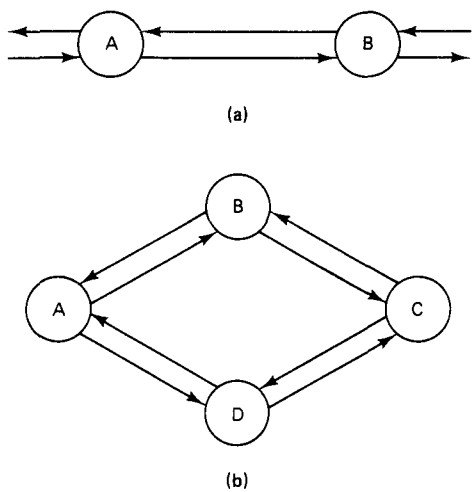


Figure 6.2 Deadlock due to buffer overflow. In (a), all buffers of A and B are full of packets destined for B and A, respectively. As a result, no packet can be accepted at either node. In (b), all buffers of A, B, C, and D are full of packets destined for C, D, A, and B, respectively.

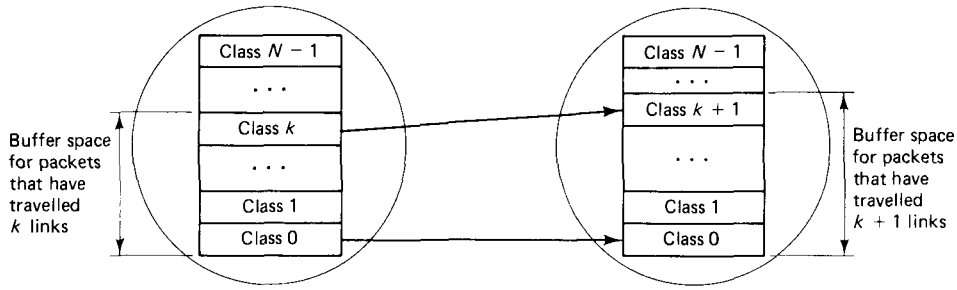


Figure 6.3 Organization of node memory in buffer classes to avoid deadlock due to buffer overflow. A packet that has traveled k links is accepted at a node only if there is an available buffer of class k or lower, where k ranges from 0 to $N - 1$ (where N is the number of nodes). Assuming that packets that travel more than $N - 1$ links are discarded as having traveled in a loop, no deadlock occurs. The proof consists of showing by induction (starting with $k = N - 1$) that at each node the buffers of class k cannot fill up permanently.

fairness between classes is complex and involves the requirements of those classes. Even within a class, there are several notions of fairness that one may wish to adopt. The example of Fig. 6.4 illustrates some of the contradictions inherent in choosing a fairness criterion. There are $n + 1$ sessions each offering 1 unit/sec of traffic along a sequence of n links with capacity of 1 unit/sec. One session's traffic goes over all n links, while the rest of the traffic goes over only one link. A maximum throughput of n units/sec can be achieved by accepting all the traffic of the single-link sessions while shutting off the n-link session. However, if our objective is to give equal rate to all sessions, the resulting throughput is only $(n + 1)/2$ units/sec. Alternatively, if our objective is to give equal resources to all sessions, the single link sessions should get a rate of $n/(n + 1)$ units/sec, while the n-link session should get a rate of $1/(n + 1)$ units/sec.

Generally, a compromise is required between equal access to a network resource and throttling those sessions that are most responsible for using up that resource. Achieving the proper balance, however, is a design decision that may be hard to quantify; often such decisions must be reached by trial and error.

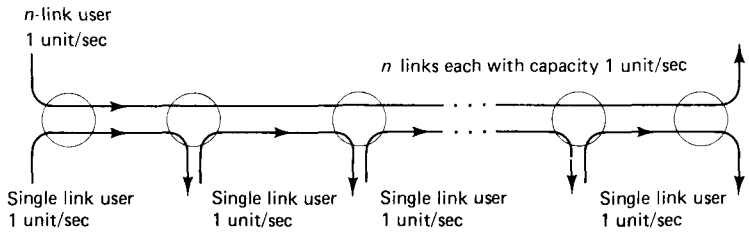


Figure 6.4 Example showing that maximizing total throughput may be incompatible with giving equal throughput to all sessions. A maximum throughput of n units/sec can be achieved if the n-link session is shut off completely. Giving equal rates of $1/2$ unit/sec to all sessions achieves a throughput of only $(n + 1)/2$ units/sec.

6.2 WINDOW FLOW CONTROL

In this section we describe the most frequently used class of flow control methods. In Sections 6.2.1 to 6.2.3, the main emphasis is on flow control within the communication subnet. Flow control outside the subnet, at the transport layer, is discussed briefly in Section 6.2.4.

A session between a transmitter A and a receiver B is said to be *window flow controlled* if there is an upper bound on the number of data units that have been transmitted by A and are not yet known by A to have been received by B (see Fig. 6.5). The upper bound (a positive integer) is called the *window size* or, simply, the *window*. The transmitter and receiver can be, for example, two nodes of the communication subnet, a user's machine and the entry node of the communication subnet, or the users' machines at the opposite ends of a session. Finally, the data units in a window can be messages, packets, or bytes, for example.

The receiver B notifies the transmitter A that it has disposed of a data unit by sending a special message to A, which is called a *permit* (other names in the literature are *acknowledgment, allocate message*, etc.). Upon receiving a permit, A is free to send one more data unit to B. Thus, a permit may be viewed as a form of passport that a data unit must obtain before entering the logical communication channel between A and B. The number of permits in use should not exceed the window size.

Permits are either contained in special control packets, or are piggybacked on regular data packets. They can be implemented in a number of ways; see the practical examples of Section 6.4 and the following discussion. Note also that a window flow control scheme for a given session may be combined with an error control scheme for the session, where the permits also play the role of acknowledgments; see Section 2.8.2 and the descriptions of the ARPANET and the Codex network in Section 6.4.

The general idea in the window strategy is that the input rate of the transmitter is reduced when permits return slowly. Therefore, if there is congestion along the communication path of the session, the attendant large delays of the permits cause a natural slowdown of the transmitter's data rate. However, the window strategy has an additional dimension, whereby the receiver may intentionally delay permits to restrict

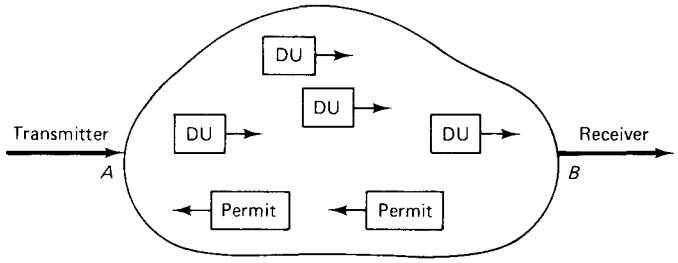


Figure 6.5 Window flow control between a transmitter and a receiver consists of an upper bound W_{AB} on the number of data units and permits in transit inside the network.

the transmission rate of the session. For example, the receiver may do so to avoid buffer overflow.

In the subsequent discussion, we consider two strategies, *end-to-end* and *node-by-node* windowing. The first strategy refers to flow control between the entry and exit subnet nodes of a session, while the second strategy refers to flow control between every pair of successive nodes along a virtual circuit's path.

6.2.1 End-to-End Windows

In the most common version of end-to-end window flow control, the window size is αW, where α and W are some positive numbers. Each time a new batch of α data units is received at the destination node, a permit is sent back to the source allocating a new batch of α data units. In a variation of this scheme, the destination node will send a new α-data unit permit upon reception of just the first of an α-data unit batch. (See the SNA pacing scheme description in Section 6.3.) To simplify the following exposition, we henceforth assume that $\alpha = 1$, but our conclusions are valid regardless of the value of α. Also, for concreteness, we talk in terms of packets, but the window maintained may consist of other data units such as bytes.

Usually, a numbering scheme for packets and permits is used so that permits can be associated with packets previously transmitted and loss of permits can be detected. One possibility is to use a sliding window protocol similar to those used for data link control, whereby a packet contains a sequence number and a request number. The latter number can serve as one or more permits for flow control purposes (see also the discussion in Section 2.8.2). For example, suppose that node A receives a packet from node B with request number k. Then A knows that B has disposed of all packets sent by A and numbered less than k, and therefore A is free to send those packets up to number $k+W-1$ that it has not sent yet, where W is the window size. In such a scheme, both the sequence number and the request number are represented modulo m, where $m \geq W+1$. One can show that if packet ordering is preserved between transmitter and receiver, this representation of numbers is adequate; the proof is similar to the corresponding proof for the goback n ARQ system. In some networks the end-to-end window scheme is combined with an end-to-end retransmission protocol, and a packet is retransmitted if following a suitable timeout, the corresponding permit has not returned to the source.

To simplify the subsequent presentation, the particular manner in which permits are implemented will be ignored. It will be assumed that the source node simply counts the number x of packets it has already transmitted but for which it has not yet received back a permit, and transmits new packets only as long as $x < W$.

Figure 6.6 shows the flow of packets for the case where the round-trip delay d, including round-trip propagation delay, packet transmission time, and permit delay is smaller than the time required to transmit the full window of W packets, that is,

$$d \leq WX$$

where X is the transmission time of a single packet. Then the source is capable of transmitting at the full speed of $1/X$ packets/sec, and flow control is not active. (To

simplify the following exposition, assume that all packets have equal transmission time and equal round-trip delay.)

The case where flow control is active is shown in Fig. 6.7. Here

$$d > WX$$

and the round-trip delay d is so large that the full allocation of W packets can be transmitted before the first permit returns. Assuming that the source always has a packet waiting in queue, the rate of transmission is W/d packets/sec.

If the results of Figs. 6.6 and 6.7 are combined, it is seen that the maximum rate of transmission corresponding to a round-trip delay d is given by

$$r = \min\left\{\frac{1}{X}, \frac{W}{d}\right\} \tag{6.1}$$

Figure 6.8 illustrates the flow control mechanism; the source transmission rate is reduced in response to congestion and the attendant large delay. Furthermore, assuming that W is relatively small, the window scheme reacts fast to congestion—within at most W packets' transmission time. This fast reaction, coupled with low overhead, is the major advantage of window strategies over other (nonwindow) schemes.

Limitations of end-to-end windows. One drawback of end-to-end windows is that they cannot guarantee a minimum communication rate for a session. Thus windows are inadequate for sessions that require a minimum guaranteed rate, such as voice and video sessions. Other drawbacks of the end-to-end window strategy have to do with window sizes. There is a basic trade-off here: One would like to make window sizes small to limit the number of packets in the subnet, thus avoiding large delays and congestion, and one would also like to make window sizes large to allow full-speed transmission and maximal throughput under light-to-moderate traffic conditions. Determining the proper window size and adjusting that size in response to congestion is not easy. This delay–throughput trade-off is particularly acute for high-speed networks, where because of high propagation delay relative to packet transmission time, window sizes should be large to allow high data rates [cf. Eq. (6.1) and Fig. 6.8]. One should remember, however, that the existence of a high-speed network does not necessarily make

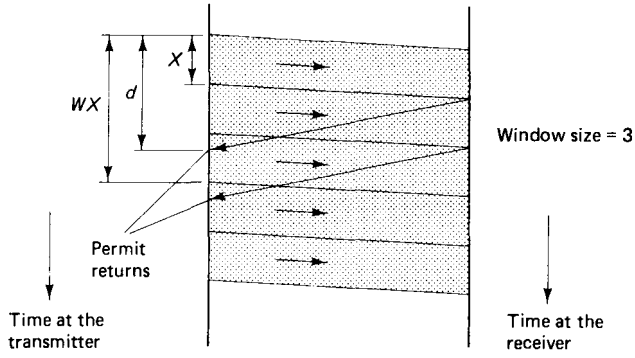


Figure 6.6 Example of full-speed transmission with a window size $W = 3$. The round-trip delay d is less than the time WX required to transmit the full window of W packets.

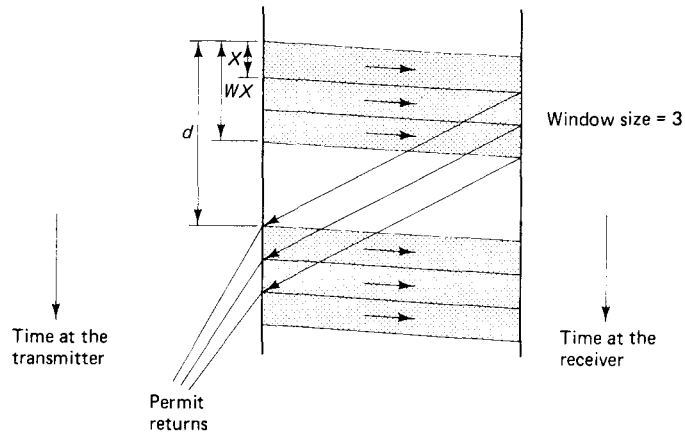


Figure 6.7 Example of delayed transmission with a window size $W = 3$. The round-trip delay d is more than the time WX required to transmit the full window of W packets. As a result, window flow control becomes active, restricting the input rate to W/d.

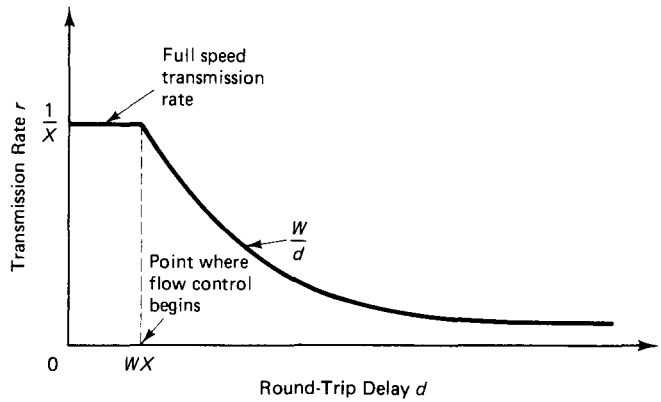


Figure 6.8 Transmission rate versus round-trip delay in a window flow control system. This oversimplified relationship assumes that all packets require equal transmission time at the source and have equal round-trip packet/permit delay.

it desirable for individual sessions to transmit at the full network speed. For high-speed networks, it is generally true that the window size of the session should be proportional to the round-trip propagation delay for the session. However, the window size should also be proportional to the session's maximum *desired* rate rather than the maximum rate allowed by the network; cf. Eq. (6.1).

End-to-end windows may also fail to provide adequte control of packet delay. To understand the relation between window size, delay, and throughput, suppose that there are n actively flow controlled sessions in the network with fixed window sizes $W_1, \ldots, W_n$. Then the total number of packets and permits traveling in the network is

$\sum_{i=1}^{n} W_i$. Focusing on packets (rather than permits), we see that their number in the network is $\sum_{i=1}^{n} \beta_i W_i$, where β_i is a factor between 0 and 1 that depends on the relative magnitude of return delay for the permit and the extent to which permits are piggybacked on regular packets. By Little's Theorem, the average delay per packet is given by

$$T = \frac{\sum_{i=1}^{n} \beta_i W_i}{\lambda}$$

where λ is the throughput (total accepted input rate of sessions). As the number of sessions increases, the throughput λ is limited by the link capacities and will approach a constant. (This constant will depend on the network, the location of sources and destinations, and the routing algorithm.) Therefore, the delay T will roughly increase proportionately to the number of sessions (more accurately their total window size) as shown in Fig. 6.9. Thus, if the number of sessions can become very large, the end-to-end window scheme may not be able to keep delay at a reasonable level and prevent congestion.

One may consider using small window sizes as a remedy to the problem of very large delays under high load conditions. Unfortunately, in many cases (particularly in low- and moderate-speed networks) one would like to allow sessions to transmit at maximum speed when there is no other interfering traffic, and this imposes a lower bound on window sizes. Indeed, if a session is using an n-link path with a packet transmission time X on each link, the round-trip packet and permit delay will be at least nX and considerably more if permits are not given high priority on the return channel. For example, if permits are piggybacked on return packets traveling on the same path in the opposite direction, the return time will be at least nX also. So from Fig. 6.8, we see that full-speed transmission will not be possible for that session even under light load conditions unless the window size exceeds the number of links n on the path (see Fig. 6.10). For this reason, recommended window sizes are typically between n and $3n$. This recommendation assumes that the transmission time on each link is much larger than the processing and propagation delay. When the propagation delay is much larger than

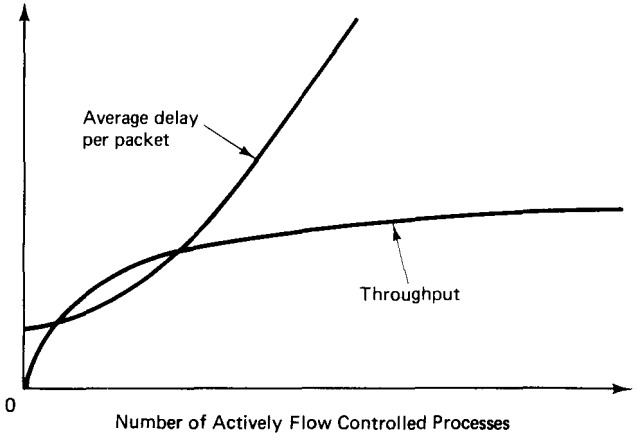


Figure 6.9 Average delay per packet and throughput as a function of the number of actively window flow controlled sessions in the network. When the network is heavily loaded, the average delay per packet increases approximately linearly with the number of active sessions, while the total throughput stays approximately constant. (This assumes that there are no retransmissions due to buffer overflow and/or large permit delays. In the presence of retransmissions, throughput may decrease as the number of active sessions increases.)

the transmission time, as in satellite links and some high-speed networks, the appropriate window size might be much larger. This is illustrated in the following example.

Example 6.2

Consider a transmission line with capacity of 1 gigabit per second (10^9 bits/sec) connecting two nodes which are 50 miles apart. The round-trip propagation delay can be roughly estimated as 1 millisecond. Assuming a packet size of 1000 bits, it is seen that a window size of at least 1000 packets is needed to sustain full-speed transmission; it is necessary to have 1000 packets and permits simultaneously propagating along the transmission line to "keep the pipeline full," with permits returning fast enough to allow unimpeded transmission of new packets. By extrapolation it is seen that Atlantic coast to Pacific coast end-to-end transmission over a distance of, say, 3000 miles requires a window size of at least 60,000 packets! Fortunately, for most sessions, such unimpeded transmission is neither required nor desirable.

Example 6.2 shows that windows should be used with care when high-speed transmission over a large distance is involved; they require excessive memory and they respond to congestion relatively slowly when the round-trip delay time is very long relative to the packet transmission time. For networks with relatively small propagation delays, end-to-end window flow control may be workable, particularly if there is no requirement for a minimum guaranteed rate. However, to achieve a good trade-off between delay and throughput, dynamic window adjustment is necessary. Under light load conditions, windows should be large and allow unimpeded transmission, while under heavy load conditions, windows should shrink somewhat, thereby not allowing delay to become excessive. This is not easy to do systematically, but some possibilities are examined in Section 6.2.5.

End-to-end windows can also perform poorly with respect to fairness. It was argued earlier that when propagation delay is relatively small, the proper window size of a session should be proportional to the number of links on its path. This means that at a heavily loaded link, long-path sessions can have many more packets awaiting transmission than short-path sessions, thereby obtaining a proportionately larger throughput. A typical situation is illustrated in Fig. 6.11. Here the windows of all sessions accumulate at the heavily loaded link. If packets are transmitted in the order of their arrival, the rate of transmission obtained by each session is roughly proportional to its window size, and this favors the long-path sessions.

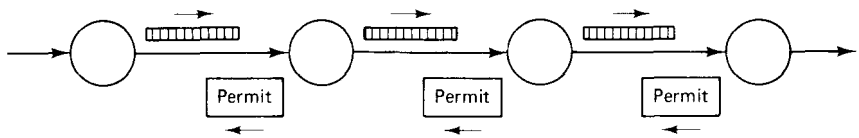


Figure 6.10 The window size must be at least equal to the number of links on the path to achieve full-speed transmission. (Assuming equal transmission time on each link, a packet should be transmitted at each link along the path simultaneously to achieve nonstop transmission.) If permit delay is comparable to the forward delay, the window size should be doubled. If the propagation delay is not negligible, an even larger window size is needed.

The fairness properties of end-to-end windows can be improved if flow-controlled sessions of the same priority class are served via a weighted round-robin scheme at each transmission queue. Such a scheme should take into account the priorities as well as the minimum guaranteed rate of different sessions. Using a round-robin scheme is conceptually straightforward when each session is a virtual circuit, but not in a datagram network, where it may not be possible to associate packets with particular flow-controlled sessions.

6.2.2 Node-by-Node Windows for Virtual Circuits

In this strategy, there is a separate window for every virtual circuit and pair of adjacent nodes along the path of the virtual circuit. Much of the discussion on end-to-end windows applies to this scheme as well. Since the path along which flow control is exercised is effectively one link long, the size of a window measured in packets is typically two or three for moderate-speed terrestrial links. For high-speed networks, the required window size might be much larger, thereby making the node-by-node strategy less attractive. For this reason, the following discussion assumes that a window size of about two is a reasonable choice.

Let us focus on a pair of successive nodes along a virtual circuit's path; we refer to them as the transmitter and the receiver. The main idea in the node-by-node scheme is that the receiver can avoid the accumulation of a large number of packets into its memory by slowing down the rate at which it returns permits to the transmitter. In the most common strategy, the receiver maintains a W-packet buffer for each virtual circuit and returns a permit to the transmitter as soon as it releases a packet from its W-packet buffer. A packet is considered to be released from the W-packet buffer once it is either delivered to a user outside the subnet or is entered in the data link control (DLC) unit leading to the subsequent node on the virtual circuit's path.

Consider now the interaction of the windows along three successive nodes ($i-1$, i, and $i+1$) on a virtual circuit's path. Suppose that the W-packet buffer of node i is full. Then node i will send a permit to node $i-1$ once it delivers an extra packet to the DLC of the $(i, i+1)$ link, which in turn will occur once a permit sent by node $(i+1)$ is received at node i. Thus, there is coupling of successive windows along the path of a virtual circuit. In particular, suppose that congestion develops at some link. Then the W-packet window

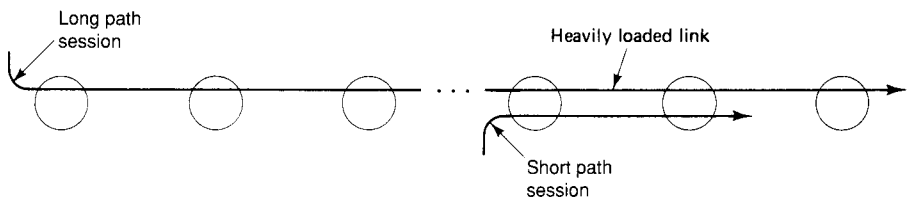


Figure 6.11 End-to-end windows discriminate in favor of long-path sessions. It is necessary to give a large window to a long-path session to achieve full-speed transmission. Therefore, a long-path session will typically have more packets waiting at a heavily loaded link than will a short-path session, and will receive proportionally larger service (assuming that packets are transmitted on a first-come first-serve basis).

at the start node of the congested link will fill up for each virtual circuit crossing the link. As a result, the W-packet windows of nodes lying upstream of the congested link will progressively fill up, including the windows of the origin nodes of the virtual circuits crossing the congested link. At that time, these virtual circuits will be actively flow controlled. The phenomenon whereby windows progressively fill up from the point of congestion toward the virtual circuit origins is known as *backpressure* and is illustrated in Fig. 6.12.

One attractive aspect of node-by-node windows can be seen from Fig. 6.12. In the worst case, where congestion develops on the last link (say, the nth) of a virtual circuit's path, the total number of packets inside the network for the virtual circuit will be approximately nW. If the virtual circuit were flow controlled via an end-to-end window, the total number of packets inside the network would be roughly comparable. (This assumes a window size of $W = 2$ in the node-by-node case, and of $W \simeq 2n$ in the end-to-end case based on the rule of thumb of using a window size that is twice the number of links of the path between transmitter and receiver.) The important point, however, is that these packets will be uniformly distributed along the virtual circuit's path in the node-by-node case, but will be concentrated at the congested link in the end-to-end case. Because of this the amount of memory required at each node to prevent buffer overflow may be much smaller for node-by-node windows than for end-to-end windows.

Distributing the packets of a virtual circuit uniformly along its path also alleviates the fairness problem, whereby large window sessions monopolize a congested link at the expense of small window sessions (cf. Fig. 6.11). This is particularly true when the window sizes of all virtual circuits are roughly equal as, for example, when the circuits involve only low-speed terrestrial links. A fairness problem, however, may still arise when satellite links (or other links with relatively large propagation delay) are involved. For such links, it is necessary to choose a large window size to achieve unimpeded transmission when traffic is light because of the large propagation delay. The difficulty arises at a node serving both virtual circuits with large window size that come over a satellite link and virtual circuits with small window size that come over a terrestrial link (see Fig. 6.13). If these circuits leave the node along the same transmission line, a fairness problem may develop when this line gets heavily loaded. A reasonable way to address this difficulty is to schedule transmissions of packets from different virtual circuits on a weighted round-robin basis (with the weights accounting for different priority classes).

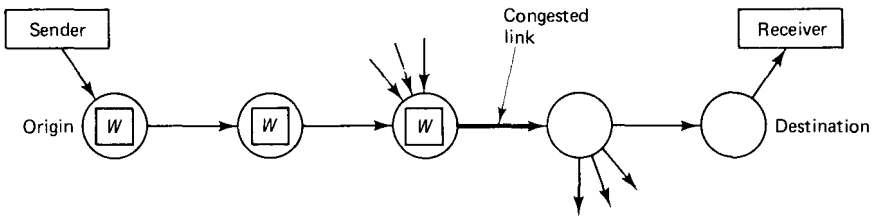


Figure 6.12 Backpressure effect in node-by-node flow control. Each node along a virtual circuit's path can store no more than W packets for that virtual circuit. The window storage space of each successive node lying upstream of the congested link fills up. Eventually, the window of the origin node fills, at which time transmission stops.

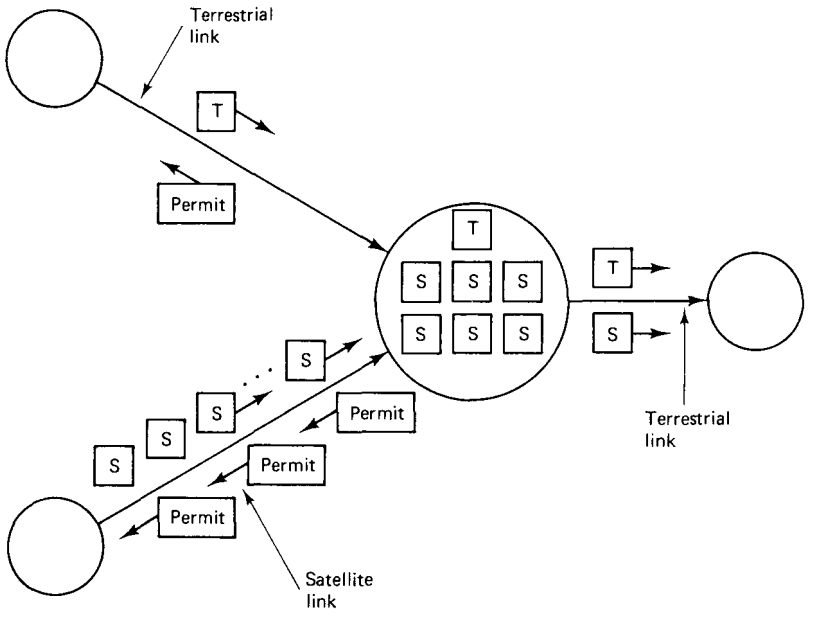


Figure 6.13 Potential fairness problem at a node serving virtual circuits that come over a satellite link (large window) and virtual circuits that come over a terrestrial link (small window). If virtual circuits are served on a first-come first-serve basis, the virtual circuits with large windows will receive better service on a subsequent transmission line. This problem can be alleviated by serving virtual circuits via a round-robin scheme.

6.2.3 The Isarithmic Method

The isarithmic method may be viewed as a version of window flow control whereby there is a single global window for the entire network. The idea here is to limit the total number of packets in the network by having a fixed number of permits circulating in the subnet. A packet enters the subnet after capturing one of these permits. It then releases the permit at its destination node. The total number of packets in the network is thus limited by the number of permits. This has the desirable effect of placing an upper bound on average packet delay that is independent of the number of sessions in the network. Unfortunately, the issues of fairness and congestion within the network depend on how the permits are distributed, which is not addressed by the isarithmic approach. There are no known sensible algorithms to control the location of the permits, and this is the main difficulty in making the scheme practical. There is also another difficulty, namely that permits can be destroyed through various malfunctions and there may be no easy way to keep track of how many permits are circulating in the network.

6.2.4 Window Flow Control at Higher Layers

Much of what has been described so far about window strategies is applicable to higher layer flow control of a session, possibly communicating across several interconnected

subnetworks. Figure 6.14 illustrates a typical situation involving a single subnetwork. A user sends data out of machine A to an entry node NA of the subnet, which is forwarded to the exit node NB and is then delivered to machine B. There is (network layer) flow control between the entry and exit nodes NA and NB (either end-to-end or node-by-node involving a sequence of nodes). There is also (network layer) window flow control between machine A and entry node NA, which keeps machine A from swamping node NA with more data than the latter can handle. Similarly, there is window flow control between exit node NB and machine B, which keeps NB from overwhelming B with too much data. Putting the pieces together we see that there is a network layer flow control system extending from machine A to machine B, which operates much like the node-by-node window flow control system of Section 6.2.2. In essence, we have a three-link path where the subnet between nodes NA and NB is viewed conceptually as the middle link.

It would appear that the system just described would be sufficient for flow control purposes, and in many cases it is. For example, it is the only one provided in the TYMNET and the Codex network described later on a virtual circuit basis. There may be a need, however, for additional flow control at the transport layer, whereby the input traffic rate of a user at machine A is controlled directly by the receiver at machine B. One reason is that the network layer window flow control from machine A to node NA may apply collectively to multiple user-pair sessions. These sessions could, for a variety of reasons, be multiplexed into a single traffic stream but have different individual flow control needs. For example, in SNA there is a transport layer flow control algorithm, which is the same as the one used in the network layer except that it is applied separately for each session rather than to a group of sessions. (See the description given in Section 6.4).

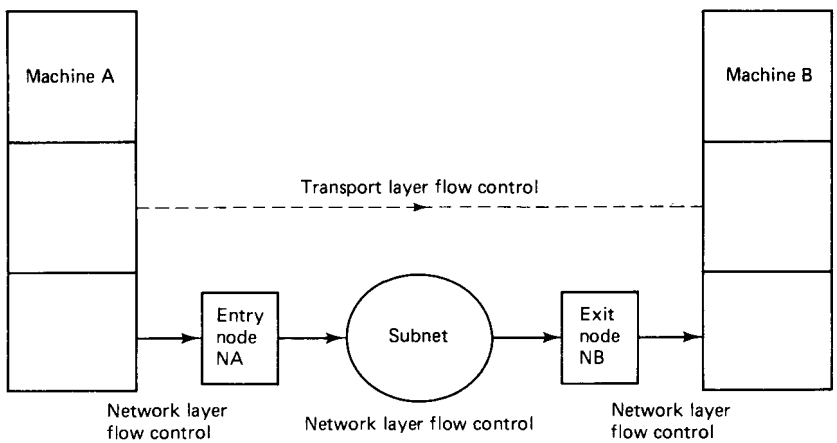


Figure 6.14 User-to-user flow control. A user sends data from machine A to a user in machine B via the subnet using the entry and exit nodes NA and NB. There is a conceptual node-by-node network layer flow control system along the connection A–NA–NB–B. There may also be direct user-to-user flow control at the transport layer or the internet sublayer, particularly if several users with different flow control needs are collectively flow controlled within the subnetwork.

In the case where several networks are interconnected with gateways, it may be useful to have windows for the gateway-to-gateway traffic of sessions that span several networks. If we take a higher-level view of the interconnected network where the gateways correspond to nodes and the networks correspond to links (Section 5.1.3), this amounts to using node-by-node windows for flow control in the internet sublayer. The gateway windows serve the purpose of distributing the total window of the internetwork sessions, thereby alleviating a potential congestion problem at a few gateways. However, the gateway-to-gateway windows may apply to multiple transport layer sessions, thereby necessitating transport layer flow control for each individual session.

6.2.5 Dynamic Window Size Adjustment

We mentioned earlier that it is necessary to adjust end-to-end windows dynamically, decreasing their size when congestion sets in. The most common way of doing this is through feedback from the point of congestion to the appropriate packet sources.

There are several ways by which this feedback can be obtained. One possibility is for nodes that sense congestion to send a special packet, sometimes called a *choke packet*, to the relevant sources. Sources that receive such a packet must reduce their windows. The sources can then attempt to increase their windows gradually following a suitable timeout. The method by which this is done is usually ad hoc in practice, and is arrived at by trial and error or simulation. The circumstances that will trigger the generation of choke packets may vary: for example, buffer space shortage or excessive queue length.

It is also possible to adjust window sizes by keeping track of permit delay or packet retransmissions. If permits are greatly delayed or if several retransmissions occur at a given source within a short period of time, this is likely to mean that packets are being excessively delayed or are getting lost due to buffer overflow. The source then reduces the relevant window sizes, and subsequently attempts to increase them gradually following a timeout.

Still another way to obtain feedback is to collect congestion information on regular packets as they traverse their route from origin to destination. This congestion information can be used by the destination to adjust the window size by withholding the return of some permits. A scheme of this type is used in SNA and is described in Section 6.4.

6.3 RATE CONTROL SCHEMES

We mentioned earlier that window flow control is not very well suited for high-speed sessions in high-speed wide area networks because the propagation delays are relatively large, thus necessitating large window sizes. An even more important reason is that windows do not regulate end-to-end packet delays well and do not guarantee a minimum data rate. Voice, video, and an increasing variety of data sessions require upper bounds on delay and lower bounds on rate. High-speed wide area networks increasingly carry such traffic, and many lower-speed networks also carry such traffic, making windows inappropriate.

An alternative form of flow control is based on giving each session a guaranteed data rate, which is commensurate to its needs. This rate should lie within certain limits that depend on the session type. For example, for a voice session, the rate should lie between the minimum needed for language intelligibility and a maximum beyond which the quality of voice cannot be further improved.

The main considerations in setting input session rates are:

1. *Delay–throughput trade-off.* Increasing throughput by setting the rates too high runs the risk of buffer overflow and excessive delay.
2. *Fairness.* If session rates must be reduced to accommodate some new sessions, the rate reduction must be done fairly, while obeying the minimum rate requirement of each session.

We will discuss various rate adjustment schemes focusing on these considerations in Section 6.5.

Given an algorithm that generates desired rates for various sessions, the question of implementing these rates arises. A strict implementation of a session rate of r packets/sec would be to admit 1 packet each $1/r$ seconds. This, however, amounts to a form of time-division multiplexing and tends to introduce large delays when the offered load of the sessions is bursty. A more appropriate implementation is to admit as many as W packets ($W > 1$) every W/r seconds. This allows a burst of as many as W packets into the network without delay, and is better suited for a dynamically changing load. There are several variations of this scheme. The following possibility is patterned after window flow control.

An allocation of W packets (a window) is given to each session, and a count x of the unused portion of this allocation is kept at the session origin. Packets from the session are admitted into the network as long as $x > 0$. Each time a packet is admitted, the count is decremented by 1, and W/r seconds later (r is the rate assigned to the session), the count is incremented by 1 as shown in Fig. 6.15. This scheme, called *time window flow control*, is very similar to window flow control with window size W except that the count is incremented W/r seconds after admitting a packet instead of after a round-trip delay when the corresponding permit returns.

A related method that regulates the burstiness of the transmitted traffic somewhat better is the so-called *leaky bucket scheme*. Here the count is incremented periodically, every $1/r$ seconds, up to a maximum of W packets. Another way to view this scheme is to imagine that for each session, there is a queue of packets without a permit and a bucket of permits at the session's source. The packet at the head of the packet queue obtains a permit once one is available in the permit bucket and then joins the set of packets with permits waiting to be transmitted (see Fig. 6.16). Permits are generated at the desired input rate r of the session (one permit each $1/r$ seconds) as long as the number in the permit bucket does not exceed a certain threshold W. The leaky bucket scheme is used in PARIS, an experimental high-speed network developed by IBM ([CiG88]; see Section 6.4). A variation, implemented in the ARPANET, is to allocate $W > 1$ permits initially to a session and subsequently restore the count back to W every W/r seconds, whether or not the session used any part of the allocation.

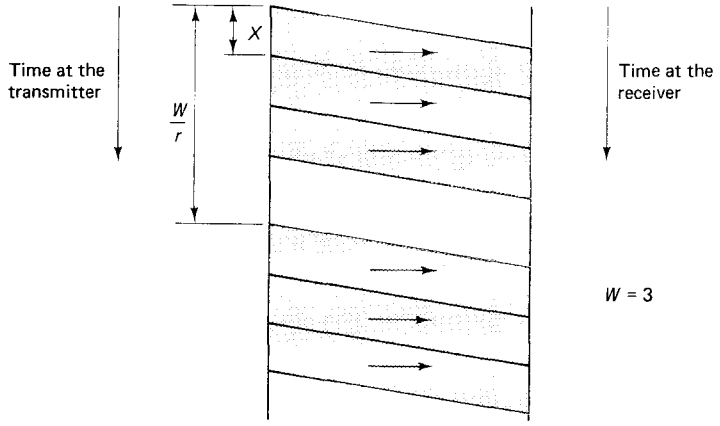


Figure 6.15 Time window flow control with $W = 3$. The count of packet allocation is decremented when a packet is transmitted and incremented W/r seconds later.

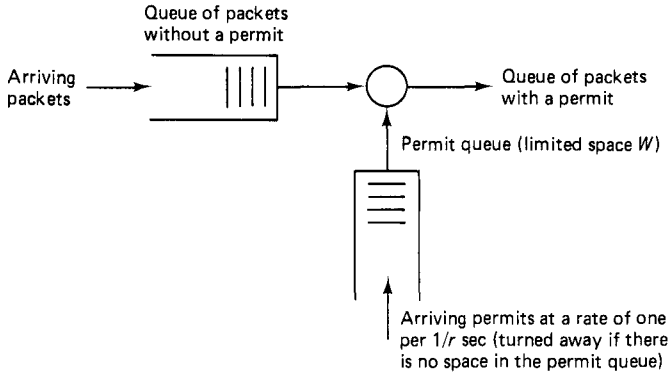


Figure 6.16 Leaky bucket scheme. To join the transmission queue, a packet must get a permit from the permit queue. A new permit is generated every $1/r$ seconds, where r is the desired input rate, as long as the number of permits does not exceed a given threshold.

The leaky bucket scheme does not necessarily preclude buffer overflow and does not guarantee an upper bound on packet delay in the absence of additional mechanisms to choose and implement the session rates inside the network; for some insight into the nature of such mechanisms, see Problem 6.19. However, with proper choice of the bucket parameters, buffer overflow and maximum packet delay can be reduced. In particular, the bucket size W is an important parameter for the performance of the leaky bucket scheme. If W is small, bursty traffic is delayed waiting for permits to become available ($W = 1$ resembles time-division multiplexing). If W is too large, long bursts of packets will be allowed into the network; these bursts may accumulate at a congested node downstream and cause buffer overflow.

The preceding discussion leads to the idea of dynamic adjustment of the bucket size. In particular, a congested node may send a special control message to the cor-

responding sources instructing them to shrink their bucket sizes. This is similar to the choke packet discussed in connection with dynamic adjustment of window sizes. The dynamic adjustment of bucket sizes may be combined with the dynamic adjustment of permit rates and merged into a single algorithm. Note, however, that in high-speed networks, the effectiveness of the feedback control messages from the congested nodes may be diminished because of relatively large propagation delays. Therefore, some predictive mechanism may be needed to issue control messages before congestion sets in.

Queueing analysis of the leaky bucket scheme. To provide some insight into the behavior of the leaky bucket scheme of Fig. 6.16, we give a queueing analysis. One should be very careful in drawing quantitative conclusions from this analysis, since some data sessions are much more bursty than is reflected in the following assumption of Poisson packet arrivals.

Let us assume the following:

1. Packets arrive according to a Poisson process with rate λ.
2. A permit arrives every $1/r$ seconds, but if the permit pool contains W permits, the arriving permit is discarded.

We view this system as a discrete-time Markov chain with states $0, 1, \ldots$. (For an alternative formulation, see Problem 3.13.) The states $i = 0, 1, \ldots, W$ correspond to $W - i$ permits available and no packets without permits waiting. The states $i = W+1, W+2, \ldots$, correspond to $i - W$ packets without permits waiting and no permits available. The state transitions occur at the times $0, 1/r, 2/r, \ldots$, just after a permit arrival. Let us consider the probabilities of k packet arrivals in $1/r$ seconds,

$$a_k = \frac{e^{-\lambda/r}(\lambda/r)^k}{k!}$$

It can be seen that the transition probabilities of the chain are

$$P_{0i} = \begin{cases} a_{i+1}, & \text{if } i \geq 1 \\ a_0 + a_1, & \text{if } i = 0 \end{cases}$$

and for $j \geq 1$,

$$P_{ji} = \begin{cases} a_{i-j+1}, & \text{if } j \leq i+1 \\ 0, & \text{otherwise} \end{cases}$$

(see Fig. 6.17). The global balance equations yield

$$p_0 = a_0 p_1 + (a_0 + a_1) p_0$$

$$p_i = \sum_{j=0}^{i+1} a_{i-j+1} p_j, \qquad i \geq 1$$

These equations can be solved recursively. In particular, we have

$$p_1 = a_2 p_0 + a_1 p_1 + a_0 p_2$$

so by using the equation $p_1 = (1 - a_0 - a_1)p_0/a_0$, we obtain

$$p_2 = \frac{p_0}{a_0}\left(\frac{(1 - a_0 - a_1)(1 - a_1)}{a_0} - a_2\right)$$

Similarly, we may use the global balance equation for p_2, and the computed expressions for p_1 and p_2, to express p_3 in terms of p_0, and so on.

The steady-state probabilities can now be obtained by noting that

$$p_0 = \frac{r - \lambda}{ra_0}$$

To see this, note that the permit generation rate averaged over all states is $(1 - p_0a_0)r$, while the packet arrival rate is λ. Equating the two rates, we obtain $p_0 = (r - \lambda)/(ra_0)$. The system is stable, that is, the packet queue stays bounded, if $\lambda < r$. The average delay for a packet to obtain a permit is

$$T = \frac{1}{r}\sum_{j=0}^{\infty} p_j \max\{0, j - W\} = \frac{1}{r}\sum_{j=W+1}^{\infty} p_j(j - W)$$

To obtain a closed-form expression for the average delay needed by a packet to get a permit and also to obtain a better model of the practical system, we modify the leaky bucket model slightly so that permits are generated on a per bit basis; this approximates the real situation where messages are broken up into small packets upon arrival at the source node. In particular, we assume that:

1. Credit for admission into the network is generated at a rate r bits/sec for transmission and the size of the bucket (*i.e.*, the maximum credit that can be saved) is W bits.
2. Messages arrive according to a Poisson process with rate λ, and the storage space for messages is infinite. Message lengths are independent and exponentially distributed with mean L bits.

Let $\mu = r/L$, so that $1/\mu$ is the mean time to transmit a message at the credit rate r. Also let $C = W/r$ be the time over which credit can be saved up. The state of the system can be described by the number of bits in the queue and the available amount of credit. At time t, let $X(t)$ be either the number of bits in the queue (if the queue is

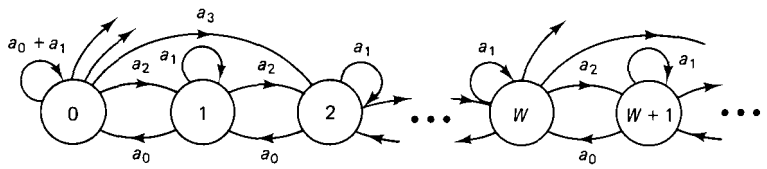


Figure 6.17 Transition probabilities of a discrete Markov chain model for the leaky bucket scheme. Here a_k is the probability of k packet arrivals in $1/r$ seconds. Note that this Markov chain is also the Markov chain for a slotted service $M/D/1$ queue.

nonempty) or minus the available credit. Thus, whenever a message consisting of x bits arrives, $X(t)$ increases by x [one of three things happens: the credit decreases by x; the queue increases by x; the credit decreases to 0 and the queue increases to $X(t)+x$ (this happens if $X(t) < 0$ and $X(t) + x > 0$)].

Letting $Y(t) = X(t) + C$, it can be seen that $Y(t)$ is the unfinished work in a fictitious $M/M/1$ queue with arrival rate λ messages/sec and service rate μ. An incoming bit is transmitted immediately if the size of the fictitious queue ahead of it is less than C and is transmitted C seconds earlier than in the fictitious queue otherwise. Focusing on the last bits of messages, we see that if T_i is the system delay in the fictitious queue for the i^{th} message, $\max\{0, T_i - C\}$ is the delay in the real queue. Using the theory of Section 3.3, it can be verified [a proof is outlined in Exercise 3.11(b)] that the steady-state distribution of the system time T_i at the fictitious queue is

$$P\{T_i \geq \tau\} = e^{-\tau(\mu-\lambda)}$$

Thus letting $T_i' = \max\{0, T_i - C\}$ be the delay of the i^{th} packet in the real queue, we obtain

$$P\{T_i' \geq \tau\} = \begin{cases} 1, & \text{if } \tau \leq 0 \\ e^{-(C+\tau)(\mu-\lambda)}, & \text{if } \tau > 0 \end{cases}$$

From this equation, the average delay of a packet in the real queue can be calculated as

$$T = \int_0^\infty P\{T_i' \geq \tau\}\, d\tau = \frac{1}{\mu - \lambda} e^{-C(\mu-\lambda)}$$

The preceding analysis can be generalized for the case where the packet lengths are independent but not exponentially distributed. In this case, the fictitious queue becomes an $M/G/1$ queue, and its system delay distribution can be estimated by using an exponential upper bound due to Kingman; see [Kle76], p. 45.

We finally note that the leaky bucket parameters affect not only the average packet delay to enter the network, which we have just analyzed. They also affect substantially the packet delay *after* the packet has entered the network. The relationship between this delay, the leaky bucket parameters, and the method for implementing the corresponding session rates at the network links is not well understood at present. For some interesting recent analyses and proposals see [Cru91a], [Cru91b], [PaG91a], [PaG91b], [Sas91], and Problem 6.19.

6.4 OVERVIEW OF FLOW CONTROL IN PRACTICE

In this section we discuss the flow control schemes of several existing networks.

Flow control in the ARPANET. Flow control in the ARPANET is based in part on end-to-end windows. The entire packet stream of each pair of machines (known as *hosts*) connected to the subnet is viewed as a "session" flowing on a logical pipe. For each such pipe there is a window of eight messages between the corresponding origin and destination subnet nodes. Each message consists of one or more packets up to a

maximum of eight. A transmitted message carries a number indicating its position in the corresponding window. Upon disposing of a message, the destination node sends back a special control packet (permit) to the origin, which in the ARPANET is called RFNM (ready for next message). The RFNM is also used as an end-to-end acknowledgment for error control purposes. Upon reception of an RFNM the origin node frees up a space in the corresponding window and is allowed to transmit an extra message. If an RFNM is not received after a specified time-out, the origin node sends a control packet asking the destination node whether the corresponding message was received. This protects against loss of an RFNM, and provides a mechanism for retransmission of lost messages.

There is an additional mechanism within the subnet for multipacket messages that ensures that there is enough memory space to reassemble these messages at their destination. (Packets in the ARPANET may arrive out of order at their destination.) Each multipacket message must reserve enough buffer space for reassembly at the receiver before it gets transmitted. This is done via a reservation message called REQALL (request for allocation) that is sent by the origin to the destination node. The reservation is granted when the destination node sends an ALL (allocate) message to the origin. When a long file is sent through the network, there is a long sequence of multipacket messages that must be transmitted. It would then be wasteful to obtain a separate reservation for each message. To resolve this problem, ALL messages are piggybacked on the returning RFNMs of multipacket messages, so that there is no reservation delay for messages after the first one in a file. If the reserved buffer space is not used by the origin node within a given timeout, it is returned to the destination via a special message. Single-packet messages do not need a reservation before getting transmitted. If, however, such a message finds the destination's buffers full, it is discarded and a copy is eventually retransmitted by the origin node after obtaining an explicit buffer reservation.

A number of improvements to the ARPANET scheme were implemented in late 1986 [Mal86]. First, the window size can be configured up to a maximum of 127; this allows efficient operation in the case where satellite links are used. Second, there can be multiple independent connections (up to 256) between two hosts, each with an independent window; this provides some flexibility in accommodating classes of traffic with different priorities and/or throughput needs. Third, an effort is made to improve the fairness properties of the current algorithm through a scheme that tries to allocate the available buffer space at each node fairly among all hosts. Finally, the reservation scheme for multipacket messages described above has been eliminated. Instead, the destination node simply reserves space for a multipacket message upon receiving the first packet of the message. If space is not available, the packet is discarded and is retransmitted after a time-out by the origin node.

The ARPANET flow control was supplemented by a rate adjustment scheme in 1989. Each node calculates an upper bound on flow rate for the origin-destination pairs routing traffic through it (the origin and the destination are subnetwork packet switches). This upper bound, also called a *ration*, is modified depending on the utilization of various critical resources of the node (processing power, transmission capacity of incident links, buffer space, etc.). The ration is adjusted up or down as the actual utilization of critical resources falls below or rises above a certain target utilization. The node rations are

broadcast to the entire network along with the routing update messages. Each origin then sets its flow rate to each destination to the minimum of the rations of the nodes traversed by the current route to the destination (these nodes become known to the origin through the shortest path routing algorithm). The rates are implemented by using a leaky bucket scheme as discussed in Section 6.3.

Flow control in the TYMNET. Flow control in the TYMNET is exercised separately for each virtual circuit via a sequence of node-by-node windows. There is one such window per virtual circuit and link on the path of the virtual circuit. Each window is measured in bytes, and its size varies with the expected peak data rate (or throughput class) of the virtual circuit. Flow control is activated via the backpressure effect discussed in Section 6.2.2. Fairness is enhanced by serving virtual circuits on a link via a round-robin scheme. This is accomplished by combining groups of bytes from several virtual circuits into data link control frames. The maximum number of bytes for each virtual circuit in a frame depends on the level of congestion on the link and the priority class of the virtual circuit. Flow control permits are piggybacked on data frames, and are highly encoded so that they do not require much bandwidth.

Flow control in SNA. We recall from Section 5.1.2 that the counterpart in SNA of the OSI architecture network layer is called the path control layer, and that it includes a flow control function called virtual route control. The corresponding algorithm, known as the virtual route pacing scheme, is based on an end-to-end window for each virtual circuit (or virtual route in SNA terminology). An interesting aspect of this scheme is that the window size (measured in packets) is dynamically adjusted depending on traffic conditions. The minimum window size is usually equal to the number of links on the path, and the maximum window size is three times as large. Each packet header contains two bits that are set to zero at the source. An intermediate node on the path that is "moderately congested" sets the first bit to 1. If the node is "badly congested," it sets both bits to 1. Otherwise, it does not change the bits. Upon arrival of the packet, the destination looks at the bits and increments the window size for no congestion, decrements the window size for moderate congestion, or sets the window size to the minimum for bad congestion.

Actually, the SNA scheme is a little different from the end-to-end window scheme focused on so far. In the main scheme discussed in Section 6.2.1, the window size is αW, and returning permits result in allocations of α packets each. We concentrated on the case where $\alpha = 1$. In SNA, however, $W = 1$ and α (rather than W) is adjusted between the minimum and maximum window size referred to earlier. Furthermore, the destination node can send a new α-packet allocation message (permit) upon reception of the first packet in an α-packet batch. Thus, full-speed transmission under light load conditions can be maintained even though $W = 1$.

In addition to virtual route control, SNA provides transport layer flow control on a session-by-session basis, which is known as session level pacing. This becomes necessary because a virtual route in SNA may contain several sessions that may have different flow control needs. The main idea here is to prevent the transmitting end of a session from sending data more quickly than the receiving end can process. Session-level

pacing is basically a window scheme whereby the transmitting end can introduce a new packet into the subnet upon receiving a permit (called a pacing response in SNA) from the other end. An interesting twist is that pacing responses can be delayed at the node through which the transmitting end of the session accesses the subnet. This provides the subnet with the means to control the rate at which it accepts data from external users.

Flow control in a Codex network. In one of the Codex networks, there is an end-to-end window associated with each virtual circuit. The window size is measured in bytes and is proportional to the number of links on the virtual circuit path and a nominal data rate for the virtual circuit. Returning permits are combined with end-to-end acknowledgments used for error control purposes, so the window scheme does not require additional communication overhead. Data link control (DLC) frames on each link are formed by concatenating groups of bytes from several virtual circuits that have traffic in queue. There is a maximum group size for each virtual circuit. Virtual circuits are served on a round-robin basis and this provides a natural mechanism for maintaining fairness.

There is also a rate control mechanism in the Codex network that is unrelated to the window scheme but plays a complementary role. The idea is to match the transmission rate of a virtual circuit along its incoming and outgoing links at each node on its path under heavy-load conditions. Without going into details (see [HSS86]), this is accomplished by adjusting the maximum number of bytes that a virtual circuit can insert in a DLC frame. As an example, suppose that a virtual circuit is slowed down on a given link due to heavy load. Then the start node of that link sends a special message to the preceding node along the virtual circuit's path, which proceeds to reduce the maximum number of bytes that a DLC frame can carry for this virtual circuit. One effect of this scheme is that when congestion develops downstream, the end-to-end window of a virtual circuit will not pile up entirely at the point of congestion, but rather will be spread more or less evenly along the virtual circuit path starting from the source node and ending at the point of congestion. This, combined with large memory space at the nodes, tends to make buffer overflow unlikely.

Flow control in the PARIS network. PARIS is an experimental high-speed packet switching system for integrated voice, video, and data communications. (PARIS is an acronym for Packetized Automatic Routing Integrated System.) PARIS uses virtual circuits and simple error control to expedite packet processing at the nodes, and achieve high packet throughput. Routes are calculated with an adaptive shortest path routing algorithm similar to the SPF algorithm of the ARPANET. Each link length is based on a measure of the load carried by the link, and is broadcast from time to time through the network using a spanning tree. Source routing is used; each packet carries the sequence of identities of the nodes that it has yet to cross.

Flow control is based on the leaky bucket scheme described in Section 6.3. A session requesting access to the network must provide the corresponding entry node with some information regarding its characteristics, such as average rate, peak rate, and average burst size. The entry node translates this information into an "equivalent capacity," which is a measure of bandwidth required by the session at each link along its path. The entry node then does a routing calculation to obtain a shortest path among

the paths that can accommodate the session. If no suitable path is found, the session is rejected. Otherwise the session is accepted and its leaky bucket parameters are determined based on its requirements and the current load of its path. The session may transmit more packets than the ones permitted by its leaky bucket. However, these extra packets are tagged as "red" and the network may discard them much more readily than other packets, which are called "green" and are given preferential access to buffer space. Even though the order of transmission of red and green packets is preserved, the algorithm is operated so that red packets have minimal effect on the loss rate of green packets. The leaky bucket parameters are kept constant during the lifetime of a session. It was found that dynamic adjustment of these parameters was of limited use because congestion information was largely outdated due to the high network speed and the relatively large propagation delays. We refer to [CiG88] and [CGK90] for further details.

Flow control in X.25. As discussed in Section 2.8.3, flow control at the X.25 packet level is implemented by means of a separate window for each virtual circuit. The default window size is 2, but it may be set as high as 7 or 127. Flow control is exercised in both directions [*i.e.*, from the user's machine (DTE) to the entry point of the network (DCE), and also from DCE to DTE]. The implementation of the window strategy is reminiscent of DLC protocols. Each data packet contains a three-bit sequence number and a three-bit request number. (If the window size is 127, these numbers are seven bits long.) The sequence number gives the position of the packet within the sender's window, while the request number is the number of the next packet expected to be received by the sender. Thus, the request number plays the role of a permit allowing the receiver to advance the corresponding window.

There is also a provision in X.25 for flow control between two DTEs communicating through the subnet. The X.25 packet format contains a special bit (called the D bit) that determines whether the piggyback number of a packet received by a DTE relates to the directly attached DCE ($D = 0$) or the remote DTE ($D = 1$). In the latter case, the request number serves as a permit for the advancement of a window maintained for flow control purposes between the two DTEs.

6.5 RATE ADJUSTMENT ALGORITHMS

In this section we look at two systematic formulations of the flow control problem to obtain algorithms for input rate adjustment. In the first approach (Section 6.5.1) we formulate an optimization problem that mathematically expresses the objective of striking a proper balance between maintaining high throughput and keeping average delay per packet at a reasonable level. In the second approach (Section 6.5.2) we emphasize fairness while maintaining average delay per packet at an acceptable level.

6.5.1 Combined Optimal Routing and Flow Control

We consider the possibility of combining routing and end-to-end flow control within the subnet by adjusting optimally *both* the routing variables and the origin–destination (OD)

pair input rates. A special case arises when routing is fixed and the only variables to be adjusted are the input rates—a problem of pure flow control.

We adopt a flow model similar to the one discussed in the context of optimal routing in Section 5.4, and we denote by r_w the input rate of an OD pair w. We first formulate a problem of adjusting routing variables together with the inputs r_w so as to minimize some "reasonable" cost function. We subsequently show that *this problem is mathematically equivalent to the optimal routing problem examined in Chapter 5* (in which r_w is fixed), and therefore the optimality conditions and algorithms given there are applicable.

If we minimize the cost function $\sum_{(i,j)} D_{ij}(F_{ij})$ of the routing problem with respect to both the path flows $\{x_p\}$ and the inputs $\{r_w\}$, we unhappily find that the optimal solution is $x_p = 0$ and $r_w = 0$ for all p and w. This indicates that the cost function should include a penalty for inputs r_w becoming too small and leads to the problem

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)} D_{ij}(F_{ij}) + \sum_{w \in W} e_w(r_w) \\ \text{subject to} \quad & \sum_{p \in P_w} x_p = r_w, \qquad \text{for all } w \in W \\ & x_p \geq 0, \qquad \text{for all } p \in P_w,\ w \in W \\ & 0 \leq r_w \leq \bar{r}_w, \qquad \text{for all } w \in W \end{aligned} \tag{6.2}$$

Here the minimization is to be carried out jointly with respect to $\{x_p\}$ and $\{r_w\}$. The given values $\bar{r}_w$ represent the desired input by OD pair w (*i.e.*, the offered load for w, defined as the input for w that would result if no flow control were exercised). As before, F_{ij} is the total flow on link (i, j) (*i.e.*, the sum of all path flows traversing the link). The functions e_w are of the form shown in Fig. 6.18 and provide a penalty for throttling the inputs r_w. They are monotonically decreasing on the set of positive numbers $(0, \infty)$, and tend to ∞ as r_w tends to zero. We assume that their first and second derivatives, e'_w and e''_w, exist on $(0, \infty)$ and are strictly negative and positive, respectively. An interesting class of functions e_w is specified by the following formula for their first derivative:

$$e'_w(r_w) = -\left(\frac{a_w}{r_w}\right)^{b_w}, \qquad \text{for } a_w \text{ and } b_w \text{ given positive constants} \tag{6.3}$$

As explained later in this section, the parameters a_w and b_w influence the optimal mag-

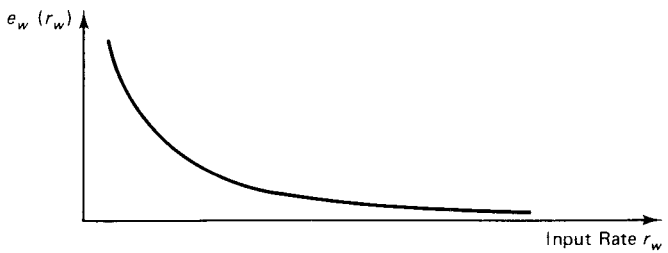


Figure 6.18 Typical form of penalty function for throttling the input rate r_w.

nitude of input r_w and the priority of OD pair w, respectively. The functions D_{ij} are defined and are monotonically increasing with positive second derivative in an interval $[0, C_{ij})$, where C_{ij} is either finite, representing link capacity, or is equal to ∞.

The value of the preceding formulation is enhanced if we adopt a broader view of w and consider it as *a class of sessions sharing the same set of paths* P_w. This allows different priorities (*i.e.*, different functions e_w) for different classes of sessions even if they share the same paths. A problem where P_w consists of a single path for each w can also be considered. This is a problem of pure flow control, namely, choosing the optimal fraction of the desired input flow of each session class that should be allowed into the network.

We now show that the combined routing and flow control problem of Eq. (6.2) is mathematically equivalent to a routing problem of the type considered in Section 5.4. Let us introduce a new variable y_w for each $w \in W$ via the equation

$$y_w = \overline{r}_w - r_w \tag{6.4}$$

We may view y_w as the *overflow* (the portion of $\overline{r}_w$ blocked out of the network), and consider it as a flow on an *overflow link* directly connecting the origin and destination nodes of w as shown in Fig. 6.19. If we define a new function E_w by

$$E_w(y_w) = e_w(\overline{r}_w - y_w) \tag{6.5}$$

the combined routing and flow control problem of Eq. (6.2) becomes, in view of the definition $y_w = \overline{r}_w - r_w$,

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)} D_{ij}(F_{ij}) + \sum_{w \in W} E_w(y_w) \\ \text{subject to} \quad & \sum_{p \in P_w} x_p + y_w = \overline{r}_w, \qquad \text{for all } w \in W \\ & x_p \geq 0, \qquad \text{for all } p \in P_w,\ w \in W \\ & y_w \geq 0, \qquad \text{for all } w \in W \end{aligned} \tag{6.6}$$

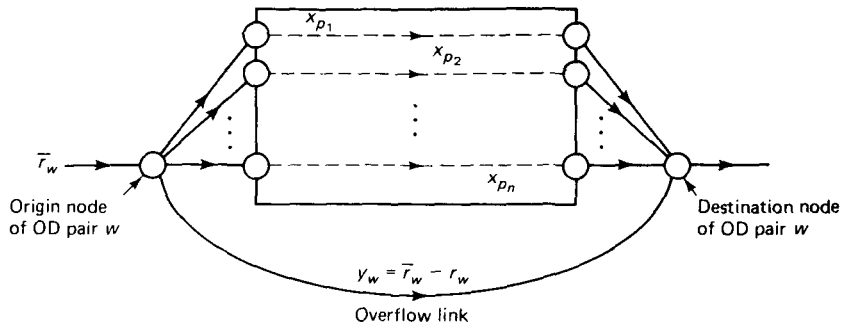


Figure 6.19 Mathematical equivalence of the flow control problem with an optimal routing problem based on the introduction of an artificial overflow link for each OD pair w. The overflow link carries the rejected traffic (*i.e.*, the difference between desired and accepted input flow, $\overline{r}_w - r_w$). The cost of the overflow link is obtained from the cost e_w for throttling the input by a change of variable.

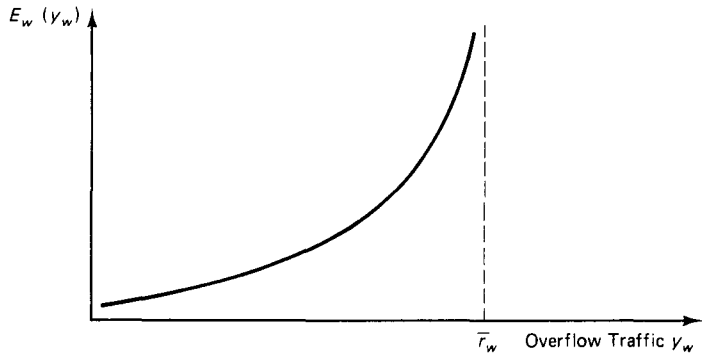


Figure 6.20 Cost function for the overflow link. The cost $E_w(y_w)$ for overflow traffic y_w equals the cost $e_w(r_w)$ for input traffic $r_w = \bar{r}_w - y_w$.

The form of the function E_w of Eq. (6.5) is shown in Fig. 6.20. If $e_w(r_w) \to \infty$ as $r_w \to 0$ (*i.e.*, there is "infinite penalty" for completely shutting off the class of sessions w), then $E_w(y_w) \to \infty$ as the overflow y_w approaches its maximum value—the maximum input $\bar{r}_w$. Thus, E_w may be viewed as a "delay" function for the overflow link, and $\bar{r}_w$ may be viewed as the "capacity" of the link.

It is now clear that the problem of Eq. (6.6) is of the optimal routing type considered in Section 5.4, and that the algorithms and optimality conditions given in Sections 5.5 to 5.7 apply. In particular, the application of the shortest path optimality condition of Section 5.5 (see also Problem 5.24) yields the following result:

A feasible set of path flows $\{x_p^*\}$ and inputs $\{r_w^*\}$ is optimal for the combined routing and flow control problem of Eq. (6.2) if and only if the following conditions hold for each $p \in P_w$ and $w \in W$:

$$x_p^* > 0 \quad \Rightarrow \quad d_p^* \leq d_{p'}^*, \qquad \text{for all } p' \in P_w, \text{ and } d_p^* \leq -e'_w(r_w^*) \tag{6.7a}$$

$$r_w^* < \bar{r}_w \quad \Rightarrow \quad -e'_w(r_w^*) \leq d_p^*, \qquad \text{for all } p \in P_w \tag{6.7b}$$

where d_p^* is the first derivative length of path p [$d_p^* = \sum_{(i,j)\in p} D'_{ij}(F_{ij}^*)$, and F_{ij}^* is the total flow of link (i,j) corresponding to $\{x_p^*\}$].

Note that the optimality conditions (6.7) depend only on the derivatives of the functions D_{ij} and e_w. This means that arbitrary constants could be added to D_{ij} and e_w without affecting the optimum. Note also from the optimality condition (6.7b) that the optimum point r_w^* is independent of $\bar{r}_w$ as long as $\bar{r}_w > r_w^*$. This is a desirable feature for a flow control strategy, preventing sessions that are being actively flow controlled from attempting to increase their share of the resources by increasing their demands. A simple example illustrates the optimality conditions (6.7).

Example 6.3

Consider the situation in Fig. 6.21 involving a single link connecting origin and destination. The cost function is

$$\frac{r}{C-r} + \frac{a}{r}$$

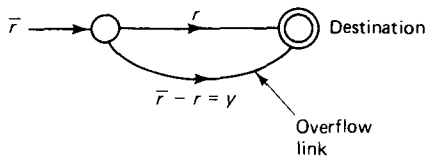


Figure 6.21 Example problem involving an origin and a destination connected by a single link.

where the first term represents a penalty due to large delay [cf. the term $D_{ij}(F_{ij})$ in Eq. (6.2)], and the second term represents a penalty for small throughput [cf. the term $e_w(r_w)$ in Eq. (6.2)]. The constant C is the capacity of the link, while the parameter a is a positive weighting factor. The equivalent routing problem [cf. Eq. (6.6)] is

$$\text{minimize } \frac{r}{C-r} + \frac{a}{\bar{r}-y}$$

$$\text{subject to } r + y = \bar{r}, \qquad r \geq 0,\ y \geq 0$$

where $y = \bar{r} - r$ represents the amount of offered load rejected by flow control (equivalently, the flow on the fictitious overflow link). The optimality conditions (6.7) show that there will be no flow control ($y = 0, r = \bar{r}$) if

$$\frac{C}{(C-\bar{r})^2} < \frac{a}{\bar{r}^2}$$

(*i.e.*, if the first derivative length of the overflow link exceeds the first derivative length of the regular link). Equivalently, there will be no flow control if

$$\bar{r} < C\frac{\sqrt{a}}{\sqrt{a}+\sqrt{C}}$$

According to the optimality condition [Eq. (6.7)], when there is flow control ($y > 0,\ r < \bar{r}$), the two first derivative lengths must be equal, that is,

$$\frac{C}{(C-r)^2} = \frac{a}{(\bar{r}-y)^2}$$

Substituting $y = \bar{r} - r$ and working out the result yields

$$r = C\frac{\sqrt{a}}{\sqrt{a}+\sqrt{C}}$$

The solution as a function of offered load is shown in Fig. 6.22. Note that the throughput is independent of the offered load beyond a certain point, as discussed earlier. The maximum throughput $C\sqrt{a}/(\sqrt{a}+\sqrt{C})$ can be regulated by adjustment of the parameter a and tends to the capacity C as $a \to \infty$.

The meaning of the parameters a_w and b_w in the cost function specified by the formula [cf. Eq. (6.3)]

$$e'_w(r_w) = -\left(\frac{a_w}{r_w}\right)^{b_w}$$

can now be clarified in the light of the optimality condition (6.7b). Consider two distinct classes of sessions, w_1 and w_2, sharing the same paths ($P_{w_1} = P_{w_2}$). Then the condition

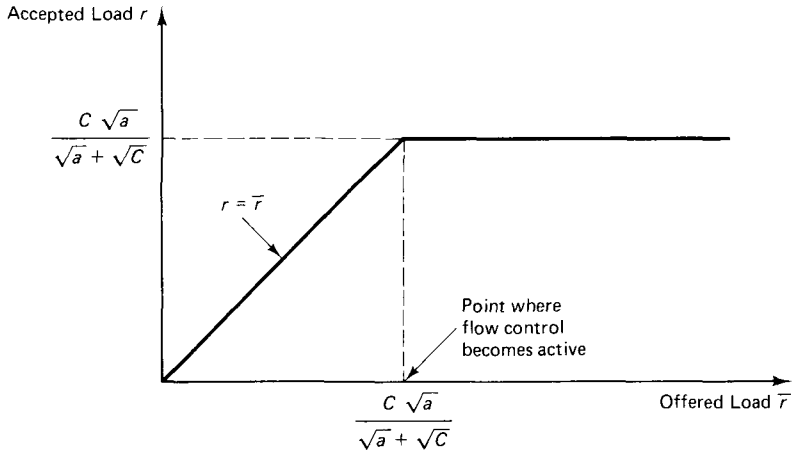


Figure 6.22 Optimal accepted load as a function of offered load in the flow control example. Flow control becomes active when the offered load exceeds a threshold level that depends on the weighting factor a.

(6.7b) implies that at an optimal solution in which both classes of sessions are throttled ($r^*_{w_1} < \bar{r}_{w_1}$, $r^*_{w_2} < \bar{r}_{w_2}$)

$$-e'_{w_1}(r^*_{w_1}) = -e'_{w_2}(r^*_{w_2}) = \min_{p \in P_{w_1}} \{d^*_p\} = \min_{p \in P_{w_2}} \{d^*_p\} \tag{6.8}$$

If e'_{w_1} and e'_{w_2} are specified by parameters a_{w_1}, b_{w_1} and a_{w_2}, b_{w_2} as in Eq. (6.3), it can be seen that:

1. If $b_{w_1} = b_{w_2}$, then

$$\frac{r^*_{w_1}}{r^*_{w_2}} = \frac{a_{w_1}}{a_{w_2}}$$

and it follows that the parameter a_w influences the optimal, relative input rate of the session class w.

2. If $a_{w_1} = a_{w_2} = a$ and $b_{w_1} < b_{w_2}$ (see Fig. 6.23), the condition (6.8) specifies that when the input flows must be made small ($r^*_{w_1}, r^*_{w_2} < a$), the session class w_2 (the one with higher parameter b_w) will be allowed a larger input. It follows that the parameter b_w influences the relative priority of the session class w under heavy load conditions.

6.5.2 Max-Min Flow Control

One of the most difficult aspects of flow control is treating all sessions fairly when it is necessary to turn traffic away from the network. Fairness can be defined in a number of different ways, but one intuitive notion of fairness is that any session is entitled to as much network use as is any other session. Figure 6.24 clarifies some of the ambiguities in this notion. One session flows through the tandem connection of all links, and each

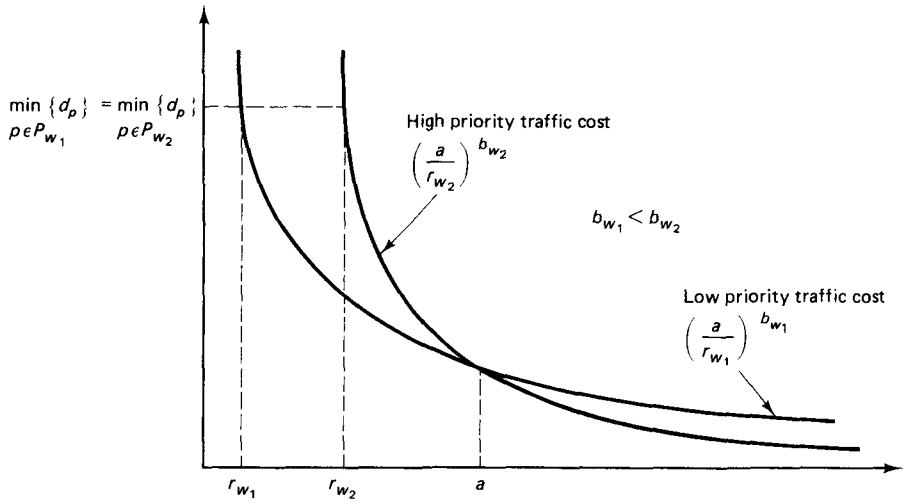


Figure 6.23 Incorporating priorities of session classes in the cost function of the flow control formulation. The session class with larger b_w will be throttled less under heavy load conditions.

other session goes through only one link. It is plausible to limit sessions 0, 1, and 2 to a rate of $1/3$ each, since this gives each of these sessions as much rate as the others. It would be rather pointless, however, to restrict session 3 to a rate of $1/3$. Session 3 might better be limited to $2/3$, since any lower limit would waste some of the capacity of the rightmost link without benefitting sessions 0, 1, or 2, and any higher limit would be unfair because it would further restrict session 0.

This example leads to the idea of maximizing the network use allocated to the sessions with the minimum allocation, thus giving rise to the term *max-min flow control*. After these most poorly treated sessions are given the greatest possible allocation, there might be considerable latitude left for choosing allocations for the other sessions. It is then reasonable to maximize the allocation for the most poorly treated of these other sessions, and so forth, until all allocations are specified. An alternative way to express this intuition, which turns out to be equivalent to the above, is to maximize the allocation of each session i subject to the constraint that an incremental increase in i's allocation does not cause a decrease in some other session's allocation that is already as small as i's or smaller.

We assume a directed graph $G = (\mathcal{N}, \mathcal{A})$ for the network and a set of sessions P using the network. Each session p has an associated fixed path in the network. We use p

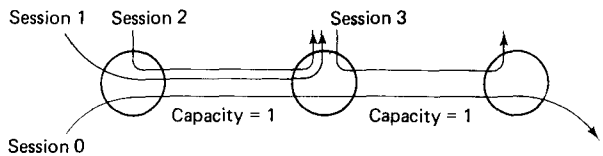


Figure 6.24 The fair solution is to give to sessions 0, 1, and 2 a rate of $1/3$ each and to give session 3 a rate of $2/3$ to avoid wasting the extra capacity available at the rightmost link.

both to refer to the session and to its path (if several sessions use the same path, several indices p refer to the same path). Thus, in our model we assume a fixed, single-path routing method.

We denote by r_p the allocated rate for session p. The allocated flow on link a of the network is then

$$F_a = \sum_{\substack{\text{all sessions } p \\ \text{crossing link } a}} r_p \tag{6.9}$$

Letting C_a be the capacity of link a, we have the following constraints on the vector $r = \{r_p \mid p \in P\}$ of allocated rates:

$$r_p \geq 0, \quad \text{for all } p \in P \tag{6.10a}$$

$$F_a \leq C_a, \quad \text{for all } a \in \mathcal{A} \tag{6.10b}$$

A vector r satisfying these constraints is said to be *feasible*.

A vector of rates r is said to be *max-min fair* if it is feasible and for each $p \in P$, r_p cannot be increased while maintaining feasibility without decreasing $r_{p'}$ for some session p' for which $r_{p'} \leq r_p$. (More formally, r is max-min fair if it is feasible, and for each $p \in P$ and feasible $\overline{r}$ for which $r_p < \overline{r}_p$, there is some p' with $r_p \geq r_{p'}$ and $r_{p'} > \overline{r}_{p'}$.) Our problem is to find a rate vector that is max-min fair.

Given a feasible rate vector r, we say that link a is a *bottleneck link* with respect to r for a session p crossing a if $F_a = C_a$ and $r_p \geq r_{p'}$ for all sessions p' crossing link a. Figure 6.25 provides an example of a max-min fair rate vector and illustrates the concept of a bottleneck link. In this example, every session has a bottleneck link. It turns out that this property holds in general as shown in the following proposition:

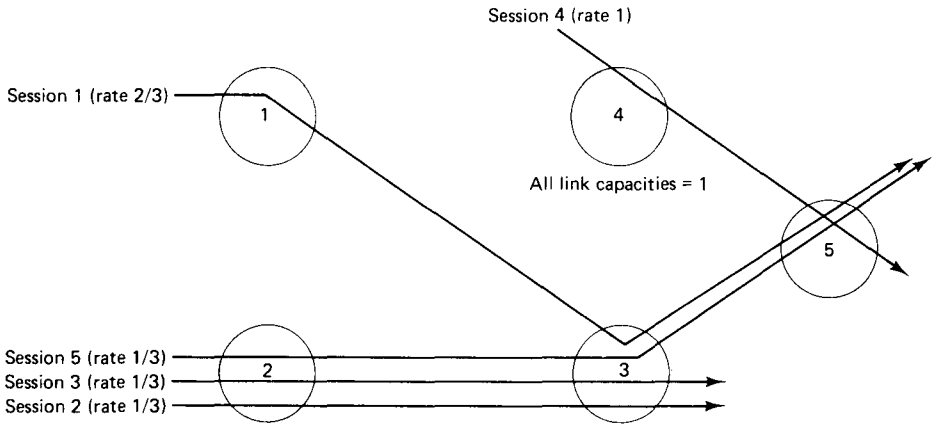


Figure 6.25 Max-min fair solution for an example network. The bottleneck links of sessions 1, 2, 3, 4, and 5 are (3,5), (2,3), (2,3), (4,5), and (2,3), respectively. Link (3,5) is not a bottleneck link for session 5 since sessions 1 and 5 share this link and session 1 has a larger rate than session 5. Link (1,3) is not a bottleneck link of any session since it has an excess capacity of 1/3 in the fair solution.

Proposition. A feasible rate vector r is max-min fair if and only if each session has a bottleneck link with respect to r.

Proof: Suppose that r is max-min fair and, to arrive at a contradiction, assume that there exists a session p with no bottleneck link. Then, for each link a crossed by p for which $F_a = C_a$, there must exist a session $p_a \neq p$ such that $r_{p_a} > r_p$; thus the quantity

$$\delta_a = \begin{cases} C_a - F_a, & \text{if } F_a < C_a \\ r_{p_a} - r_p, & \text{if } F_a = C_a \end{cases}$$

is positive. Therefore, by increasing r_p by the minimum δ_a over all links a crossed by p, while decreasing by the same amount the rates of the sessions r_{p_a} of the links a crossed by p with $F_a = C_a$, we maintain feasibility without decreasing the rate of any session p' with $r_{p'} \leq r_p$; this contradicts the max-min fairness property of r.

Conversely, assume that each session has a bottleneck link with respect to the feasible rate vector r. Then, to increase the rate of any session p while maintaining feasibility, we must decrease the rate of some session p' crossing the bottleneck link a of p (because we have $F_a = C_a$ by the definition of a bottleneck link). Since $r_{p'} \leq r_p$ for all p' crossing a (by the definition of a bottleneck link), the rate vector r satisfies the requirement for max-min fairness. **Q.E.D.**

Next, we give a simple algorithm for computing max-min fair rate vectors. The idea of the algorithm is to start with an all-zero rate vector and to increase the rates on all paths together until $F_a = C_a$ for one or more links a. At this point, each session using a saturated link (*i.e.*, a link with $F_a = C_a$) has the same rate at every other session using that link. Thus, these saturated links serve as bottleneck links for all sessions using them.

At the next step of the algorithm, all sessions not using the saturated links are incremented equally in rate until one or more new links become saturated. Note that the sessions using the previously saturated links might also be using these newly saturated links (at a lower rate). The newly saturated links serve as bottleneck links for those sessions that pass through them but do not use the previously saturated links. The algorithm continues from step to step, always equally incrementing all sessions not passing through any saturated link; when all sessions pass through at least one saturated link, the algorithm stops.

In the algorithm, as stated more precisely below, A^k denotes the set of links not saturated at the beginning of step k, and P^k denotes the set of sessions not passing through any saturated link at the beginning of step k. Also, n_a^k denotes the number of sessions that use link a and are in P^k. Note that this is the number of sessions that will share link a's yet unused capacity. Finally, $\tilde{r}^k$ denotes the increment of rate added to all of the sessions in P^k at the k^{th} step.

Initial conditions: $k = 1$, $F_a^0 = 0$, $r_p^0 = 0$, $P^1 = P$, and $A^1 = \mathcal{A}$.

1. n_a^k := number of sessions $p \in P^k$ crossing link a
2. $\tilde{r}^k := \min_{a \in A^k} \left(C_a - F_a^{k-1}\right)/n_a^k$

3. $r_p^k := \begin{cases} r_p^{k-1} + \tilde{r}^k & \text{for } p \in P^k \\ r_p^{k-1} & \text{otherwise} \end{cases}$
4. $F_a^k := \sum_{p \text{ crossing } a} r_p^k$
5. $A^{k+1} := \{a \mid C_a - F_a^k > 0\}$
6. $P^{k+1} := \{p \mid p \text{ does not cross any link } a \in A^{k+1}\}$
7. $k := k + 1$
8. If P^k is empty, then stop; else go to 1.

At each step k, an equal increment of rate is added to all sessions not yet passing through a saturated link, and thus at each step k, all sessions in P^k have the same rate. All sessions in P^k passing through a link that saturates in step k have at least as much rate as any other session on that link and hence are bottlenecked by that link. Thus upon termination of the algorithm, each session has a bottleneck link, and by the proposition shown earlier, the final rate vector is max-min fair.

Example 6.4

Consider the problem of max-min fair allocation for the five sessions and the network shown in Fig. 6.25. All links have a capacity of one.

Step 1: All sessions get a rate of $1/3$. Link (2,3) is saturated at this step, and the rate of the three sessions (2, 3, and 5) that go through it is fixed at $1/3$.

Step 2: Sessions 1 and 4 get an additional rate increment of $1/3$ for a total of $2/3$. Link (3,5) is saturated, and the rate of session 1 is fixed at $2/3$.

Step 3: Session 4 gets an additional rate increment of $1/3$ for a total of 1. Link (4,5) is saturated, and the rate of session 4 is fixed at 1. Since now all sessions go through at least one saturated link, the algorithm terminates with the max-min fair solution shown in Fig. 6.25.

Several generalizations can be made to the basic approach described above. First, to keep the flow on each link strictly below capacity, we can replace C_a in the algorithm with some fixed fraction of C_a. Next, we can consider ways to assign different priorities to different kinds of traffic and to make these priorities sensitive to traffic levels. If $b_p(r_p)$ is an increasing function representing the priority of p at rate r_p, the max-min fairness criterion can be modified as follows: For each p, maximize r_p subject to the constraint that any increase in r_p would cause a decrease of $r_{p'}$ for some p' satisfying $b_{p'}(r_{p'}) \leq b_p(r_p)$. It is easy to modify the algorithm above to calculate fair rates with such priorities. Another twist on the same theme is to require that each r_p be upper bounded by $C_a - F_a$ on each link used by path p; the rationale is to maintain enough spare capacity on each link to be able to add an extra session. The problem here, however, is that as the number of sessions on a link grow, the reserve capacity shrinks to zero and the buffer requirement grows with the number of sessions, just like the corresponding growth using windows. This difficulty can be bypassed by replacing the constraint $r_p \leq C_a - F_a$ by a constraint of the form $r_p \leq (C_a - F_a)q_a$, where q_a is a positive scalar factor depending on the number of sessions crossing link a (see Problem 6.18).

There has been a great deal of work on distributed algorithms that dynamically adjust the session rates to maintain max-min fairness as the sessions change. A repre-

sentative algorithm [Hay81] will help in understanding the situation. In this algorithm, v_a^k represents an estimate of the maximum allowable session rate on link a at the k^{th} iteration, F_a^k is the allocated flow on link a corresponding to the rates r_p^k, and n_a is the number of sessions using link a. The typical iteration of the algorithm is

$$v_a^{k+1} = v_a^k + \frac{C_a - F_a^k}{n_a}$$

$$r_p^{k+1} = \min_{\substack{\text{links } a \text{ on} \\ \text{path } p}} v_a^{k+1}$$

Each iteration can be implemented in a distributed way by first passing the values v_a^{k+1} from the links a to the sessions using those links, and then passing the values r_p^{k+1} from the sessions p to the links used by these sessions. The major problem is that the allocated rates can wander considerably before converging, and link flows can exceed capacity temporarily. There are other algorithms that do not suffer from this difficulty ([GaB84b] and [Mos84]). In the algorithm of [GaB84b], the session rates are iterated at the session sources according to

$$r_p^{k+1} = \min_{\substack{\text{links } a \text{ on} \\ \text{path } p}} \left\{ r_p^k + \frac{C_a - F_a^k - r_p^k}{1 + n_a} \right\}$$

This algorithm aims at solving the max-min fair flow control problem subject to the additional constraints $r_p \leq C_a - F_a$ for each session p and link a crossed by p (see Problem 6.18). By adding the relation

$$r_p^{k+1} \leq r_p^k + \frac{C_a - F_a^k - r_p^k}{1 + n_a}$$

over all sessions p crossing link a, it is straightforward to verify that

$$F_a^{k+1} \leq \frac{n_a C_a}{1 + n_a}$$

so the link flows are strictly less than the link capacities at all times. Extensions of this algorithm to accommodate session priorities are possible (see [GaB84b]).

Max-min fair rates may be implemented using leaky bucket schemes as discussed in Section 6.3. Another approach to implementing max-min fair rates has been explored in [HaG86]. This approach avoids both the problem of communication in a distributed algorithm and also the problem of the ambiguity in the meaning of rate for interactive traffic. The idea is very simple: Serve different sessions on each link in the network on a round-robin basis. This means that *if* a session always has a packet waiting at a node when its turn comes up on the outgoing link, that session gets as much service as any other session using that link. Thus, to achieve max-min fairness, it is only necessary that each session always has a packet waiting at its bottleneck link. In fact, it can be shown that by using node-by-node windowing with a large enough window, a session that always has something to send will always have a packet waiting at its bottleneck link (see [Hah86]).

SUMMARY

In this chapter we identified the major flow control objectives as limiting average delay and buffer overflow within the subnet, and treating sessions fairly. We reviewed the major flow control methods, and we saw that the dominant strategies in practice are based on windows and input rate control. Window strategies combine low overhead with fast reaction to congestion but have some limitations, particularly for networks with relatively large propagation delay. Window strategies are also unsuitable for sessions that require a minimum guaranteed rate. Rate adjustment schemes are usually implemented by means of leaky buckets. However, there remain a number of questions regarding the choice and adjustment of the leaky bucket parameters in response to traffic conditions. We also described two theoretical input rate adjustment schemes. The first scheme extends the optimal routing methodology of Chapter 5 to the flow control context, and combines routing and flow control into a single algorithm. The second scheme assumes fixed, single-path routing for each session, and focuses on maintaining flow control fairness.

NOTES, SOURCES, AND SUGGESTED READING

Section 6.1. Extensive discussions of flow control can be found in the April 1981 special issue of the IEEE Transactions on Communications. An informative survey is [GeK80]. A more recent survey [GeK89] discusses flow control in local area networks.

Section 6.2. The special difficulties of window flow control along satellite links are discussed in [GrB83].

Section 6.3. The leaky bucket scheme is discussed in [Tur86] and [Zha89]. Several rate control schemes are discussed in [Gol90a], [Gol90b], [ELL90a], and [ELL90b].

Section 6.4. Flow control in the ARPANET is described in several sources (*e.g.*, [Kle76] and [KlO77]). The more recent rate control scheme is discussed in [RFS91] and [ELS90]. The TYMNET flow control system is discussed in [Tym81]. For further discussion on SNA, see [Ahu79] and [GeY82], and for more on the Codex network, see [HSS86]. The PARIS flow control scheme and other PARIS protocols are discussed in [CiG88], and [CGK90].

Section 6.5. The combined optimal routing and flow control was formulated in [GaG80] and [Gol80]. For additional material on this subject, see [Ibe81] and [Gaf82]. A simulation together with a discussion of practical implementation schemes is given in [ThC86]. Flow control based on adjusting the rate of encoding of digitized voice has been considered in [BGS80]. The material on fair flow control is based on [Hay81]. For related work, see [Jaf81], [GaB84b], [Mos84], [Hah86], and [HaG86].

PROBLEMS

6.1 Consider a window flow controlled virtual circuit going over a satellite link. All packets have a transmission time of 5 msec. The round-trip processing and propagation delay is 0.5 sec. Find a lower bound on the window size for the virtual circuit to be able to achieve maximum speed transmission when there is no other traffic on the link.

6.2 Suppose that the virtual circuit in Problem 6.1 goes through a terrestrial link in addition to the satellite link. The transmission time on the terrestrial link is 20 msec, and the processing and propagation delay is negligible. What is the maximum transmission rate in packets/sec that can be attained for this virtual circuit assuming no flow control? Find a lower bound to an end-to-end window size that will allow maximum transmission rate assuming no other traffic on the links. Does it make a difference whether the terrestrial link is before or after the satellite link?

6.3 Suppose that node-by-node windows are used in the two-link system of Problem 6.2. Find lower bounds on the window size required along each link in order to achieve maximum speed transmission, assuming no other traffic on the links.

6.4 The three-node network of Fig. 6.26 contains only one virtual circuit from node 1 to 3, and uses node-by-node windows. Each packet transmission on link (1,2) takes 1 sec, and on link (2,3) takes 2 sec; processing and propagation delay is negligible. Permits require 1 sec to travel on each link. There is an inexhaustible supply of packets at node 1. The system starts at time 0 with W permits at node 1, W permits at node 2, and no packets stored at nodes 2 and 3. For $W = 1$, find the times, from 0 to 10 sec at which a packet transmission starts at node 1 and node 2. Repeat for $W = 2$.

6.5 In the discussion of node-by-node window flow control, it was assumed that node i can send a permit back to its predecessor $(i-1)$ once it releases a packet to the DLC of link $(i, i+1)$. The alternative is for node i to send the permit when it receives the DLC acknowledgment that the packet has been received correctly at node $i+1$. Discuss the relative merits of the two schemes. Which scheme requires more memory? What happens when link $(i, i+1)$ is a satellite link?

6.6 Consider a combined optimal routing and flow control problem involving the network of Fig. 6.27 (cf. Section 6.4.1). The cost function is

$$(F_{13})^2 + (F_{23})^2 + (F_{34})^2 + \frac{a}{r_1} + \frac{a}{r_2}$$

where a is a positive scalar parameter. Find the optimal values of the rates r_1 and r_2 for each value of a.

6.7 Consider six nodes arranged in a ring and connected with unit capacity bidirectional links $(i, i+1)$, $i = 1, 2, 3, 4, 5$, and (6,1). There are two sessions from nodes 1, 2, 3, 4, and 5 to node 6, one in the clockwise and one in the counterclockwise direction. Similarly, there are two sessions from nodes 2, 3, and 4 to node 5, and two sessions from node 3 to node 4. Find the max-min fair rates for these sessions.

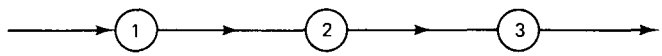


Figure 6.26

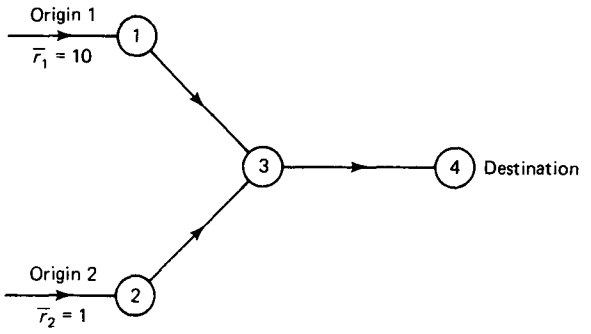


Figure 6.27

6.8 Suppose that the definition of a fair rate vector of Section 6.4.2 is modified so that, in addition to Eq. (6.10), there is a constraint $r_p \leq b_p$ that the rate of a session p must satisfy. Here b_p is the maximum rate at which session p is capable of transmitting. Modify the algorithm given in Section 6.4.2 to solve this problem. Find the fair rates for the example given in this section when $b_p = 1$, for $p = 2, 4, 5$ and $b_p = 1/4$, for $p = 1, 3$. *Hint:* Add a new link for each session.

6.9 The purpose of this problem is to illustrate how the relative throughputs of competing sessions can be affected by the priority rule used to serve them. Consider two sessions A and B sharing the first link L of their paths as shown in Fig. 6.28. Each session has an end-to-end window of two packets. Permits for packets of A and B arrive d_A and d_B seconds, respectively, after the end of transmission on link L. We assume that d_A is exponentially distributed with mean of unity, while (somewhat unrealistically) we assume that $d_B = 0$. Packets require a transmission time on L which is exponentially distributed with mean of unity. Packet transmission times and permit delays are all independent. We assume that a new packet for A (B) enters the transmission queue of L immediately upon receipt of a permit for A (B).

(a) Suppose that packets are transmitted on L on a first-come first-serve basis. Argue that the queue of L can be represented by a Markov chain with the 10 queue states BB, BBA, BAB, ABB, BBAA, BABA, BAAB, ABBA, ABAB, and AABB (each letter stands for a packet of the corresponding session). Show that all states have equal steady-state probability and that the steady-state throughputs of sessions A and B in packets/sec are 0.4 and 0.6, respectively.

(b) Now suppose that transmissions on link L are scheduled on a round-robin basis. Between successive packet transmissions for session B, session A transmits one packet if it has one waiting. Draw the state transition diagram of a five-state Markov chain which

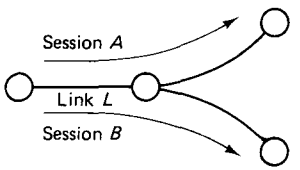


Figure 6.28

models the queue of L. Solve for the equilibrium state probabilities. What are the steady-state throughputs of sessions A and B?

(c) Finally, suppose that session A has nonpreemptive priority over session B at link L. Between successive packet transmissions for session B, session A transmits as many packets as it can. Session B regains control of the link only when A has nothing to send. Draw the state transition diagram of a five-state Markov chain which models L and its queue. Solve for the equilibrium state probabilities. What are the steady-state throughputs of sessions A and B?

6.10 Consider a window between an external source and the node by which that source is connected to the subnet. The source generates packets according to a Poisson process of rate λ. Each generated packet is accepted by the node if a permit is available. If no permit is available, the packet is discarded, never to return. When a packet from the given source enters the DLC unit of an outgoing link at the node, a new permit is instantaneously sent back to the source. The source initially has two permits and the window size is 2. Assume that the other traffic at the node imposes a random delay from the time a packet is accepted at the node to the time it enters the DLC unit. Specifically, assume that in any arbitrarily small interval δ, there is a probability $\mu\delta$ that a waiting packet from the source (or the first of two waiting packets) will enter the DLC; this event is independent of everything else.

(a) Construct a Markov chain for the number of permits at the source.

(b) Find the probability that a packet generated by the source is discarded.

(c) Explain whether the probability in part (b) would increase or decrease if the propagation and transmission delay from source to node and the reverse were taken into account.

(d) Suppose that a buffer of size k is provided at the source to save packets for which no permit is available; when a permit is received, one of the buffered packets is instantly sent to the network node. Find the new Markov chain describing the system, and find the probability that a generated packet finds the buffer full.

6.11 Consider a network with end-to-end window flow control applied to each virtual circuit. Assume that the data link control operates perfectly and that packets are never thrown away inside the network; thus, packets always arrive at the destination in the order sent, and all packets eventually arrive.

(a) Suppose that the destination sends permits in packets returning to the source; if no return packet is available for some time-out period, a special permit packet is sent back to the source. These permits consist of the number modulo m of the next packet awaited by the destination. What is the restriction on the window size W in terms of the modulus m? Why?

(b) Suppose next that the permits contain the number modulo m of *each* of the packets in the order received since the last acknowledgment was sent. Does this change your answer to part (a)? Explain.

(c) Is it permissible for the source to change the window size W without prior agreement from the destination? Explain.

(d) How can the destination reduce the effective window size below the window size used by the source without prior agreement from the source? (By effective window size we mean the maximum number of packets for the source–destination pair that can be in the network at one time.)

6.12 Consider a node-by-node window scheme. In an effort to reduce the required buffering, the designers associated the windows with destinations rather than with virtual circuits. Assume that all virtual circuit paths to a given destination j use a directed spanning tree so that each

node $i \neq j$ has only one outgoing link for traffic to that destination. Assume that each node $i \neq j$ originally has permits for two packets for j that can be sent over the outgoing link to j. Each time that node i releases a packet for j into its DLC unit on the outgoing link, it sends a new permit for one packet back over the incoming link over which that packet arrived. If the packet arrived from a source connected to i, the permit is sent back to the source (each source also originally has two permits for the given destination).

(a) How many packet buffers does node i have to reserve for packets going to destination j to guarantee that every arriving packet for j can be placed in a buffer?

(b) What are the pros and cons of this scheme compared with the conventional node-by-node window scheme on a virtual circuit basis?

6.13 Consider a network using node-by-node windows for each virtual circuit. Describe a strategy for sending and reclaiming permits so that buffer overflow never occurs regardless of how much memory is available for packet storage at each node and of how many virtual circuits are using each link. *Hint*: You need to worry about too many permits becoming available to the transmitting node of each link.

6.14 Consider a network using a node-by-node window for each session. Suppose that the transmission capacity of all links of the networks is increased by a factor K and that the number of sessions that can be served also increases by a factor K. Argue that in order for the network to allow for full-speed transmission for each session under light traffic conditions, the total window of the network should increase by a factor K if propagation delay is dominated by packet transmission time and by a factor K^2 if the reverse is true.

6.15 Consider the variation of the leaky bucket scheme where $W > 1$ permits are allocated initially to a session and the count is restored back to W every W/r seconds. Develop a Markov chain model for the number of packets waiting to get a permit. Assume a Poisson arrival process.

6.16 Describe how the gradient projection method for optimal routing can be used to solve in distributed fashion the combined optimal routing and flow control problem of Section 6.5.1.

6.17 Let r be a max-min fair rate vector corresponding to a given network and set of sessions.

(a) Suppose that some of the sessions are eliminated and let $\overline{r}$ be a corresponding max-min fair rate vector. Show by example that we may have $\overline{r}_p < r_p$ for some of the remaining sessions p.

(b) Suppose that some of the link capacities are increased and let $\overline{r}$ be a corresponding max-min fair rate vector. Show by example that we may have $\overline{r}_p < r_p$ for some sessions p.

6.18 *Alternative Formulation of Max-Min Fair Flow Control ([Jaf81] and [GaB84b]).* Consider the max-min flow control problem where the rate vector r is required, in addition, to satisfy $r_p \leq (C_a - F_a)q_a$ for each session p and link a crossed by session p. Here q_a are given positive scalars.

(a) Show that a max-min fair rate vector exists and is unique, and give an algorithm for calculating it.

(b) Show that for a max-min fair rate vector, the utilization factor $\rho_a = F_a/C_a$ of each link a satisfies

$$\rho_a \leq \frac{n_a q_a}{1 + n_a q_a}$$

where n_a is the number of sessions crossing link a.

(c) Show that additional constraints of the form $r_p \leq R_p$, where R_p is a given positive scalar for each session p, can be accommodated in this formulation by adding to the network one extra link per session.

6.19 *Guaranteed Delay Bounds Using Leaky Buckets [PaG91a],[PaG91b].* In this problem we show how guaranteed delay bounds for the sessions sharing a network can be obtained by appropriately choosing the sessions' leaky bucket parameters. We assume that each session i has a fixed route and is constrained by a leaky bucket with parameters W_i and r_i, as in the scheme of Fig. 6.16. We assume a fluid model for the traffic, *i.e.*, the packet sizes are infinitesimal. In particular, if $A_i(\tau, t)$ is the amount of session i traffic entering the network during an interval $[\tau, t]$,

$$A_i(\tau, t) \leq W_i + r_i(t - \tau).$$

Let $T_i^l(\tau, t)$ be the amount of session i traffic transmitted on link l in an interval $[\tau, t]$. Assume that each link l transmits at its maximum capacity C_l whenever it has traffic in queue and operates according to a priority discipline, called *Rate Proportional Processor Sharing*, whereby if two sessions i and j have traffic in queue throughout the interval $[\tau, t]$, then

$$\frac{T_i^l(\tau, t)}{T_j^l(\tau, t)} = \frac{r_i}{r_j} \tag{6.11}$$

We also assume that bits of the same session are transmitted in the order of their arrival. (For practical approximations of such an idealized scheme, see [PaG91a].) Let $S(l)$ be the set of sessions sharing link l and let

$$\rho(l) = \frac{\sum_{i \in S(l)} r_i}{C_l}$$

be the corresponding link utilization. Assume that $\rho(l) < 1$ for all links l.

(a) Let l_i be the first link crossed by session i. Show that the queue length at link l_i for session i is never more than W_i. Furthermore, at link l_i, each bit of session i waits in queue no more than $W_i \rho(l_i)/r_i$ time units, while for each interval $[\tau, t]$ throughout which session i has traffic in queue,

$$T_i^{l_i}(\tau, t) \geq \frac{r_i(t - \tau)}{\rho(l_i)}$$

Hint: The rate of transmission of session i in an interval throughout which session i has traffic in queue is at least $r_i/\rho(l_i)$. If $\overline{\tau}$ and $\overline{t}$ are the start and end of a busy period for session i, respectively, the queue length at times $t \in [\overline{\tau}, \overline{t}]$ is

$$Q_i(t) = A_i(\overline{\tau}, t) - T_i^{l_i}(\overline{\tau}, t) \leq W_i + \left(r_i - \frac{r_i}{\rho(l_i)}\right)(t - \overline{\tau})$$

(b) Let ρ_{max}^i be the maximum utilization over the links crossed by session i and assume that processing and propagation delay are negligible. Show that the amount of traffic of session i within the entire network never exceeds W_i. Furthermore, the time spent inside the network by a bit of session i is at most $W_i \rho_{max}^i / r_i$. *Hint*: Argue that while some link on session i's path has some session i traffic waiting in queue, the rate of departure of session i traffic from the network is at least r_i/ρ_{max}^i.

(c) Consider now a generalization of the preceding scheme called Generalized Processor Sharing. In particular, suppose that at each link l, instead of Eq. (6.11), the following relation holds for all sessions $i, j \in S(l)$

$$\frac{T_i^l(\tau, t)}{T_j^l(\tau, t)} = \frac{\phi_i^l}{\phi_j^l}$$

where ϕ_i^l, ϕ_j^l are given positive numbers. For any session i define

$$g_i = \min_{\substack{\text{all links } l \\ \text{crossed by } i}} \frac{1}{\rho(l)} \frac{\phi_i^l}{\sum_{j \in S(l)} \phi_j^l}$$

Show that if $g_i \geq r_i$, then the amount of traffic of session i within the entire network never exceeds

$$\frac{W_i r_i}{g_i}$$

Furthermore, the time spent by a bit of session i inside the network is at most W_i/g_i.