request will be broadcast until a route with a higher sequence number is found. Intermediate nodes store the routes that have a higher sequence number, or the fewest hops for the current sequence number.

In the spirit of an on demand protocol, intermediate nodes only store the routes that are in use. Other route information learned during broadcasts is timed out after a short delay. Discovering and storing only the routes that are used helps to save bandwidth and battery life compared to a standard distance vector protocol that periodically broadcasts updates.

So far, we have considered only a single route, from *A* to *I*. To further save resources, route discovery and maintenance are shared when routes overlap. For instance, if *B* also wants to send packets to *I*, it will perform route discovery. However, in this case the request will first reach *D*, which already has a route to *I*. Node *D* can then generate a reply to tell *B* the route without any additional work being required.

There are many other ad hoc routing schemes. Another well-known on demand scheme is DSR (Dynamic Source Routing) (Johnson et al., 2001). A different strategy based on geography is explored by GPSR (Greedy Perimeter Stateless Routing) (Karp and Kung, 2000). If all nodes know their geographic positions, forwarding to a destination can proceed without route computation by simply heading in the right direction and circling back to escape any dead ends. Which protocols win out will depend on the kinds of ad hoc networks that prove useful in practice.

5.3 CONGESTION CONTROL ALGORITHMS

Too many packets present in (a part of) the network causes packet delay and loss that degrades performance. This situation is called **congestion**. The network and transport layers share the responsibility for handling congestion. Since congestion occurs within the network, it is the network layer that directly experiences it and must ultimately determine what to do with the excess packets. However, the most effective way to control congestion is to reduce the load that the transport layer is placing on the network. This requires the network and transport layers to work together. In this chapter we will look at the network aspects of congestion. In Chap. 6, we will complete the topic by covering the transport aspects of congestion.

Figure 5-21 depicts the onset of congestion. When the number of packets hosts send into the network is well within its carrying capacity, the number delivered is proportional to the number sent. If twice as many are sent, twice as many are delivered. However, as the offered load approaches the carrying capacity, bursts of traffic occasionally fill up the buffers inside routers and some packets are lost. These lost packets consume some of the capacity, so the number of delivered packets falls below the ideal curve. The network is now congested.

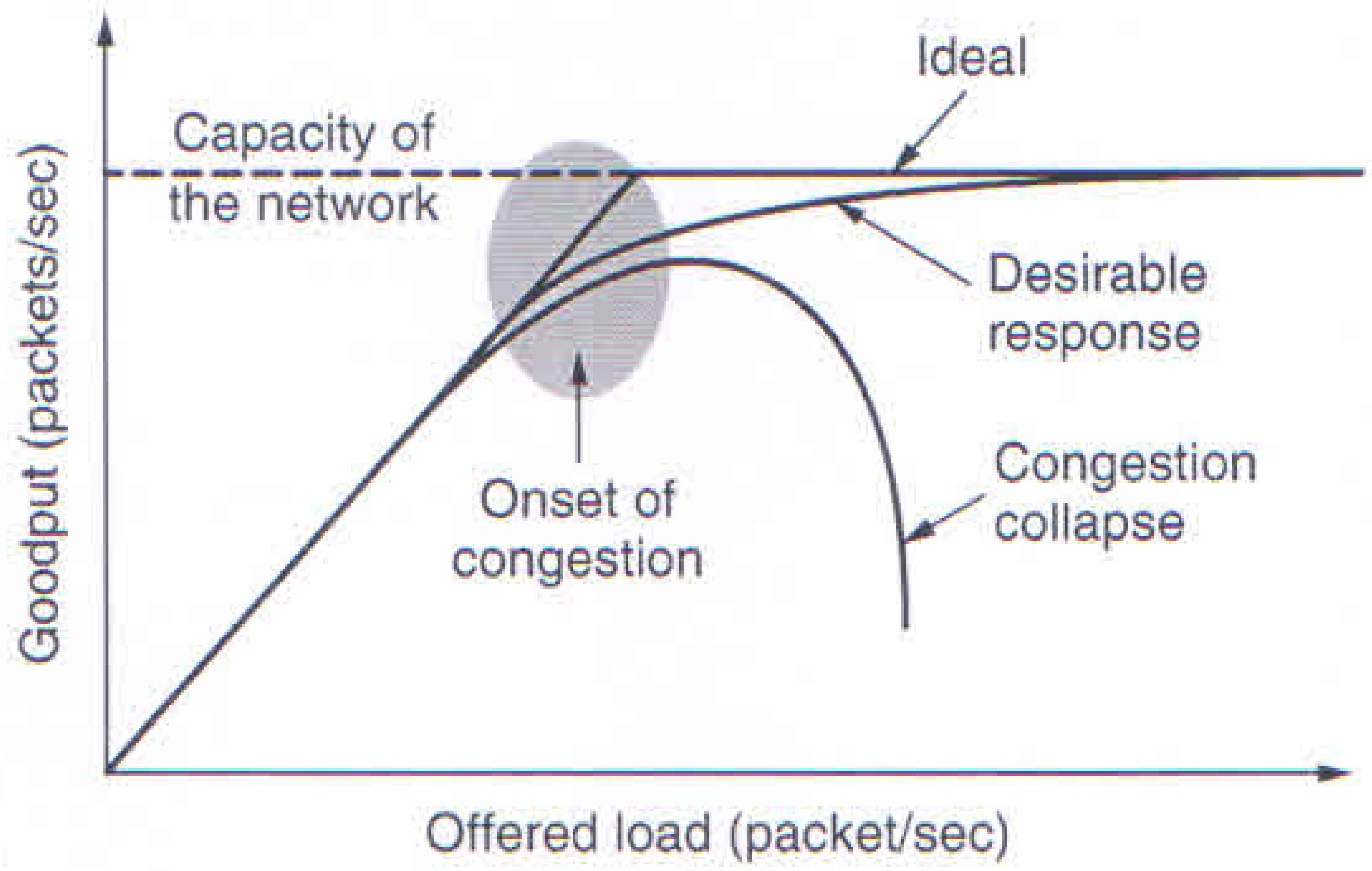

Figure 5-21. With too much traffic, performance drops sharply.

Unless the network is well designed, it may experience a **congestion collapse**, in which performance plummets as the offered load increases beyond the capacity. This can happen because packets can be sufficiently delayed inside the network that they are no longer useful when they leave the network. For example, in the early Internet, the time a packet spent waiting for a backlog of packets ahead of it to be sent over a slow 56-kbps link could reach the maximum time it was allowed to remain in the network. It then had to be thrown away. A different failure mode occurs when senders retransmit packets that are greatly delayed, thinking that they have been lost. In this case, copies of the same packet will be delivered by the network, again wasting its capacity. To capture these factors, the y-axis of Fig. 5-21 is given as **goodput**, which is the rate at which *useful* packets are delivered by the network.

We would like to design networks that avoid congestion where possible and do not suffer from congestion collapse if they do become congested. Unfortunately, congestion cannot wholly be avoided. If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up. If there is insufficient memory to hold all of them, packets will be lost. Adding more memory may help up to a point, but Nagle (1987) realized that if routers have an infinite amount of memory, congestion gets worse, not better. This is because by the time packets get to the front of the queue, they have already timed out (repeatedly) and duplicates have been sent. This makes matters worse, not better—it leads to congestion collapse.

Low-bandwidth links or routers that process packets more slowly than the line rate can also become congested. In this case, the situation can be improved by directing some of the traffic away from the bottleneck to other parts of the network. Eventually, however, all regions of the network will be congested. In this situation, there is no alternative but to shed load or build a faster network.

It is worth pointing out the difference between congestion control and flow control, as the relationship is a very subtle one. Congestion control has to do with making sure the network is able to carry the offered traffic. It is a global issue, involving the behavior of all the hosts and routers. Flow control, in contrast, relates to the traffic between a particular sender and a particular receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver is able to absorb it.

To see the difference between these two concepts, consider a network made up of 100-Gbps fiber optic links on which a supercomputer is trying to force feed a large file to a personal computer that is capable of handling only 1 Gbps. Although there is no congestion (the network itself is not in trouble), flow control is needed to force the supercomputer to stop frequently to give the personal computer a chance to breathe.

At the other extreme, consider a network with 1-Mbps lines and 1000 large computers, half of which are trying to transfer files at 100 kbps to the other half. Here, the problem is not that of fast senders overpowering slow receivers, but that the total offered traffic exceeds what the network can handle.

The reason congestion control and flow control are often confused is that the best way to handle both problems is to get the host to slow down. Thus, a host can get a "slow down" message either because the receiver cannot handle the load or because the network cannot handle it. We will come back to this point in Chap. 6.

We will start our study of congestion control by looking at the approaches that can be used at different time scales. Then we will look at approaches to preventing congestion from occurring in the first place, followed by approaches for coping with it once it has set in.

5.3.1 Approaches to Congestion Control

The presence of congestion means that the load is (temporarily) greater than the resources (in a part of the network) can handle. Two solutions come to mind: increase the resources or decrease the load. As shown in Fig. 5-22, these solutions are usually applied on different time scales to either prevent congestion or react to it once it has occurred.

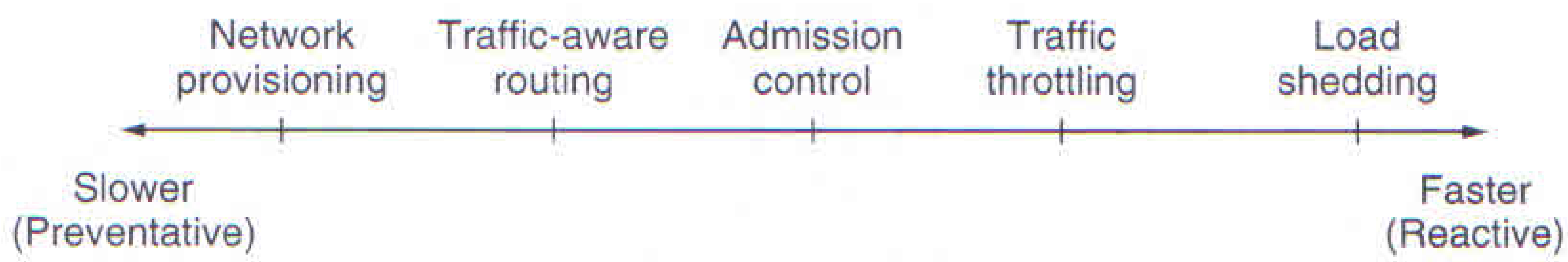

Figure 5-22. Timescales of approaches to congestion control.

The most basic way to avoid congestion is to build a network that is well matched to the traffic that it carries. If there is a low-bandwidth link on the path along which most traffic is directed, congestion is likely. Sometimes resources can be added dynamically when there is serious congestion, for example, turning on spare routers or enabling lines that are normally used only as backups (to make the system fault tolerant) or purchasing bandwidth on the open market. More often, links and routers that are regularly heavily utilized are upgraded at the earliest opportunity. This is called **provisioning** and happens on a time scale of months, driven by long-term traffic trends.

To make the most of the existing network capacity, routes can be tailored to traffic patterns that change during the day as network users wake and sleep in different time zones. For example, routes may be changed to shift traffic away from heavily used paths by changing the shortest path weights. Some local radio stations have helicopters flying around their cities to report on road congestion to make it possible for their mobile listeners to route their packets (cars) around hotspots. This is called **traffic-aware routing**. Splitting traffic across multiple paths is also helpful.

However, sometimes it is not possible to increase capacity. The only way then to beat back the congestion is to decrease the load. In a virtual-circuit network, new connections can be refused if they would cause the network to become congested. This is called **admission control**.

At a finer granularity, when congestion is imminent the network can deliver feedback to the sources whose traffic flows are responsible for the problem. The network can request these sources to throttle their traffic, or it can slow down the traffic itself.

Two difficulties with this approach are how to identify the onset of congestion, and how to inform the source that needs to slow down. To tackle the first issue, routers can monitor the average load, queueing delay, or packet loss. In all cases, rising numbers indicate growing congestion.

To tackle the second issue, routers must participate in a feedback loop with the sources. For a scheme to work correctly, the time scale must be adjusted carefully. If every time two packets arrive in a row, a router yells STOP and every time a router is idle for 20 μsec, it yells GO, the system will oscillate wildly and never converge. On the other hand, if it waits 30 minutes to make sure before saying anything, the congestion-control mechanism will react too sluggishly to be of any use. Delivering timely feedback is a nontrivial matter. An added concern is having routers send more messages when the network is already congested.

Finally, when all else fails, the network is forced to discard packets that it cannot deliver. The general name for this is **load shedding**. A good policy for choosing which packets to discard can help to prevent congestion collapse.

5.3.2 Traffic-Aware Routing

The first approach we will examine is traffic-aware routing. The routing schemes we looked at in Sec 5.2 used fixed link weights. These schemes adapted to changes in topology, but not to changes in load. The goal in taking load into account when computing routes is to shift traffic away from hotspots that will be the first places in the network to experience congestion.

The most direct way to do this is to set the link weight to be a function of the (fixed) link bandwidth and propagation delay plus the (variable) measured load or average queuing delay. Least-weight paths will then favor paths that are more lightly loaded, all else being equal.

Traffic-aware routing was used in the early Internet according to this model (Khanna and Zinky, 1989). However, there is a peril. Consider the network of Fig. 5-23, which is divided into two parts, East and West, connected by two links, *CF* and *EI*. Suppose that most of the traffic between East and West is using link *CF*, and, as a result, this link is heavily loaded with long delays. Including queueing delay in the weight used for the shortest path calculation will make *EI* more attractive. After the new routing tables have been installed, most of the East-West traffic will now go over *EI*, loading this link. Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems.

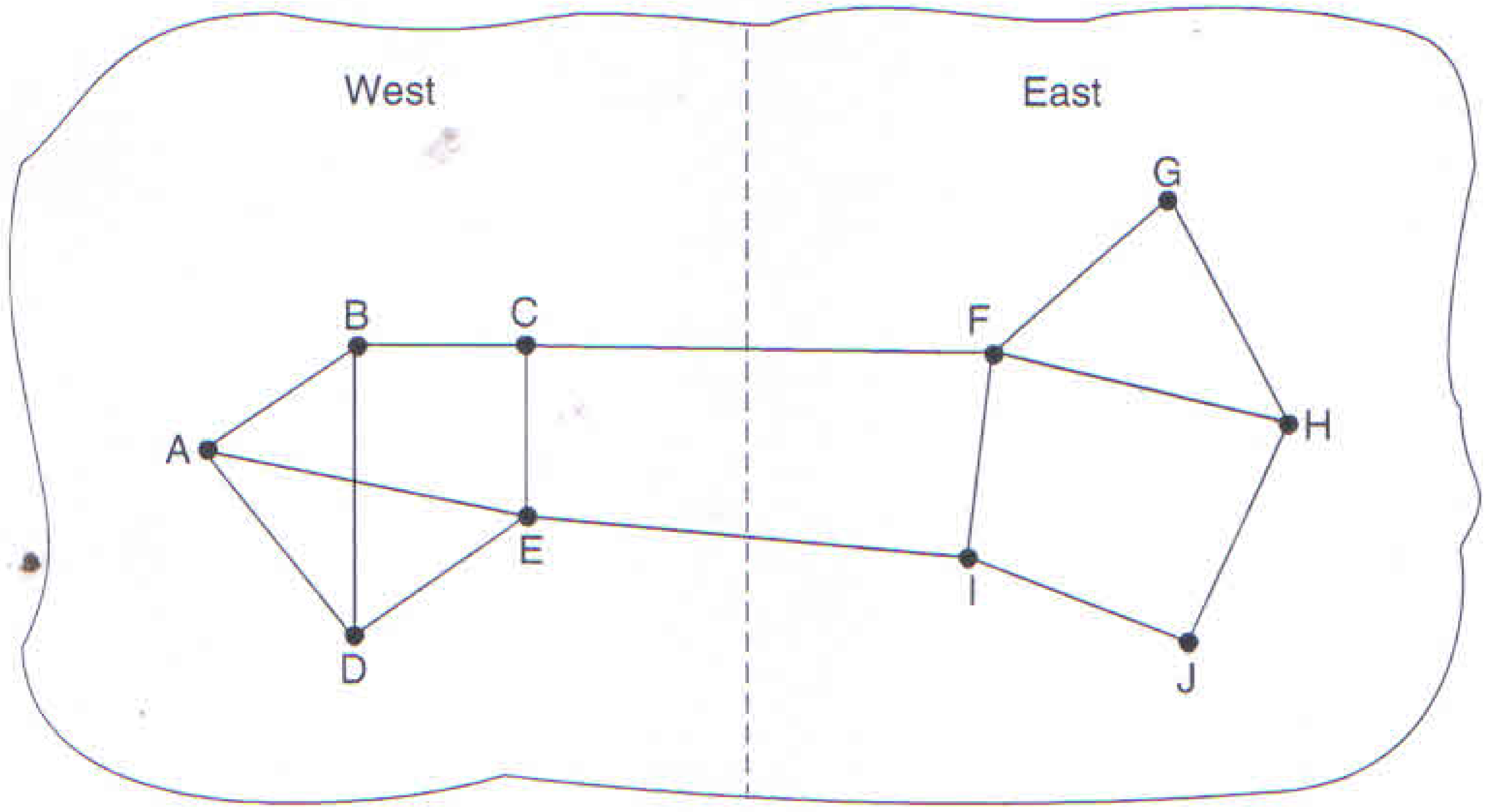

Figure 5-23. A network in which the East and West parts are connected by two links.

If load is ignored and only bandwidth and propagation delay are considered, this problem does not occur. Attempts to include load but change weights within a narrow range only slow down routing oscillations. Two techniques can contribute to a successful solution. The first is multipath routing, in which there can be multiple paths from a source to a destination. In our example this means that the traffic can be spread across both of the East to West links. The second one is for the routing scheme to shift traffic across routes slowly enough that it is able to converge, as in the scheme of Gallagher (1977).

Given these difficulties, in the Internet routing protocols do not generally adjust their routes depending on the load. Instead, adjustments are made outside the routing protocol by slowly changing its inputs. This is called **traffic engineering**.

5.3.3 Admission Control

One technique that is widely used in virtual-circuit networks to keep congestion at bay is **admission control**. The idea is simple: do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested. Thus, attempts to set up a virtual circuit may fail. This is better than the alternative, as letting more people in when the network is busy just makes matters worse. By analogy, in the telephone system, when a switch gets overloaded it practices admission control by not giving dial tones.

The trick with this approach is working out when a new virtual circuit will lead to congestion. The task is straightforward in the telephone network because of the fixed bandwidth of calls (64 kbps for uncompressed audio). However, virtual circuits in computer networks come in all shapes and sizes. Thus, the circuit must come with some characterization of its traffic if we are to apply admission control.

Traffic is often described in terms of its rate and shape. The problem of how to describe it in a simple yet meaningful way is difficult because traffic is typically bursty—the average rate is only half the story. For example, traffic that varies while browsing the Web is more difficult to handle than a streaming movie with the same long-term throughput because the bursts of Web traffic are more likely to congest routers in the network. A commonly used descriptor that captures this effect is the **leaky bucket** or **token bucket**. A leaky bucket has two parameters that bound the average rate and the instantaneous burst size of traffic. Since leaky buckets are widely used for quality of service, we will go over them in detail in Sec. 5.4.

Armed with traffic descriptions, the network can decide whether to admit the new virtual circuit. One possibility is for the network to reserve enough capacity along the paths of each of its virtual circuits that congestion will not occur. In this case, the traffic description is a service agreement for what the network will guarantee its users. We have prevented congestion but veered into the related topic of quality of service a little too early; we will return to it in the next section.

Even without making guarantees, the network can use traffic descriptions for admission control. The task is then to estimate how many circuits will fit within the carrying capacity of the network without congestion. Suppose that virtual circuits that may blast traffic at rates up to 10 Mbps all pass through the same 100-Mbps physical link. How many circuits should be admitted? Clearly, 10 circuits can be admitted without risking congestion, but this is wasteful in the normal case since it may rarely happen that all 10 are transmitting full blast at the same time. In real networks, measurements of past behavior that capture the statistics of transmissions can be used to estimate the number of circuits to admit, to trade better performance for acceptable risk.

Admission control can also be combined with traffic-aware routing by considering routes around traffic hotspots as part of the setup procedure. For example, consider the network illustrated in Fig. 5-24(a), in which two routers are congested, as indicated.

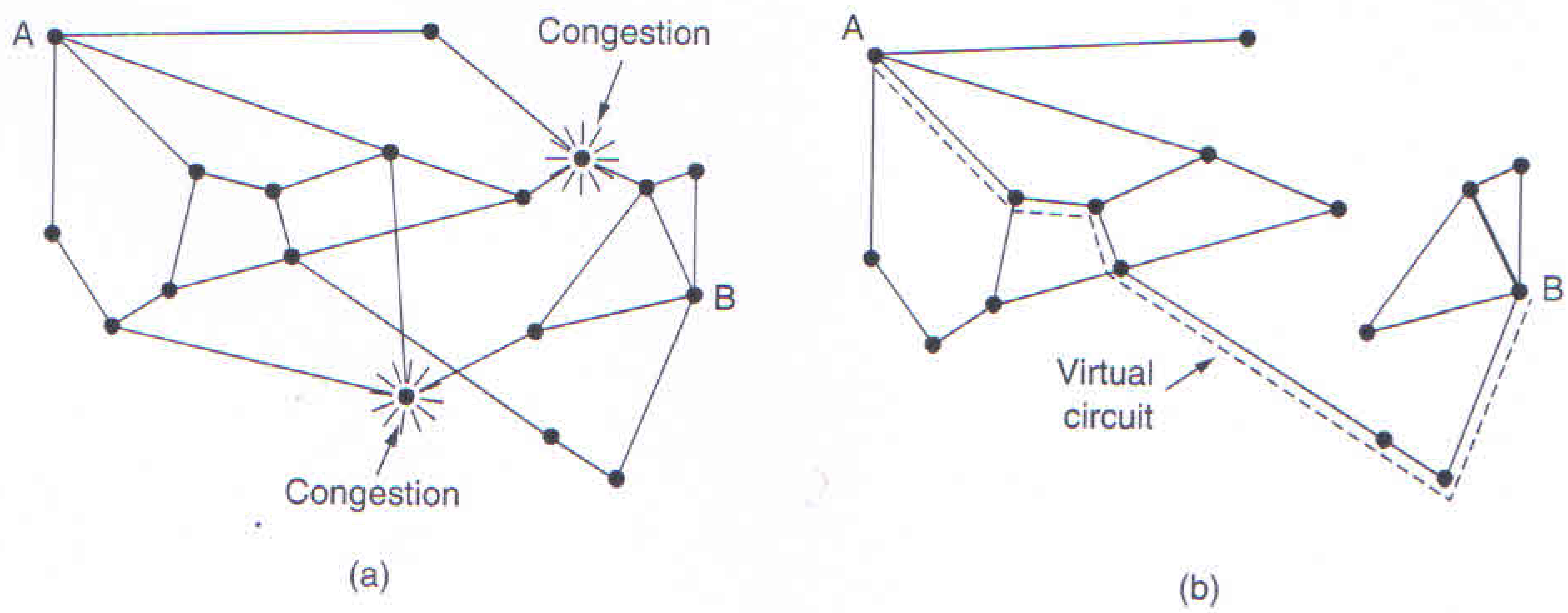

Figure 5-24. (a) A congested network. (b) The portion of the network that is not congested. A virtual circuit from A to B is also shown.

Suppose that a host attached to router A wants to set up a connection to a host attached to router B. Normally, this connection would pass through one of the congested routers. To avoid this situation, we can redraw the network as shown in Fig. 5-24(b), omitting the congested routers and all of their lines. The dashed line shows a possible route for the virtual circuit that avoids the congested routers. Shaikh et al. (1999) give a design for this kind of load-sensitive routing.

5.3.4 Traffic Throttling

In the Internet and many other computer networks, senders adjust their transmissions to send as much traffic as the network can readily deliver. In this setting, the network aims to operate just before the onset of congestion. When congestion is imminent, it must tell the senders to throttle back their transmissions and slow down. This feedback is business as usual rather than an exceptional situation. The term **congestion avoidance** is sometimes used to contrast this operating point with the one in which the network has become (overly) congested.

Let us now look at some approaches to throttling traffic that can be used in both datagram networks and virtual-circuit networks. Each approach must solve two problems. First, routers must determine when congestion is approaching, ideally before it has arrived. To do so, each router can continuously monitor the resources it is using. Three possibilities are the utilization of the output links, the buffering of queued packets inside the router, and the number of packets that are lost due to insufficient buffering. Of these possibilities, the second one is the most useful. Averages of utilization do not directly account for the burstiness of most traffic—a utilization of 50% may be low for smooth traffic and too high for highly variable traffic. Counts of packet losses come too late. Congestion has already set in by the time that packets are lost.

The queueing delay inside routers directly captures any congestion experienced by packets. It should be low most of time, but will jump when there is a burst of traffic that generates a backlog. To maintain a good estimate of the queueing delay, d, a sample of the instantaneous queue length, s, can be made periodically and d updated according to

$$d_{\text{new}} = \alpha d_{\text{old}} + (1 - \alpha)s$$

where the constant α determines how fast the router forgets recent history. This is called an **EWMA** (**Exponentially Weighted Moving Average**). It smoothes out fluctuations and is equivalent to a low-pass filter. Whenever d moves above the threshold, the router notes the onset of congestion.

The second problem is that routers must deliver timely feedback to the senders that are causing the congestion. Congestion is experienced in the network, but relieving congestion requires action on behalf of the senders that are using the network. To deliver feedback, the router must identify the appropriate senders. It must then warn them carefully, without sending many more packets into the already congested network. Different schemes use different feedback mechanisms, as we will now describe.

Choke Packets

The most direct way to notify a sender of congestion is to tell it directly. In this approach, the router selects a congested packet and sends a **choke packet** back to the source host, giving it the destination found in the packet. The original packet may be tagged (a header bit is turned on) so that it will not generate any more choke packets farther along the path and then forwarded in the usual way. To avoid increasing load on the network during a time of congestion, the router may only send choke packets at a low rate.

When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination, for example, by 50%. In a datagram network, simply picking packets at random when there is congestion is likely to cause choke packets to be sent to fast senders, because they will have the most packets in the queue. The feedback implicit in this protocol can help prevent congestion yet not throttle any sender unless it causes trouble. For the same reason, it is likely that multiple choke packets will be sent to a given host and destination. The host should ignore these additional chokes for the fixed time interval until its reduction in traffic takes effect. After that period, further choke packets indicate that the network is still congested.

An example of a choke packet used in the early Internet is the SOURCE-QUENCH message (Postel, 1981). It never caught on, though, partly because the circumstances in which it was generated and the effect it had were not clearly specified. The modern Internet uses an alternative notification design that we will describe next.

Explicit Congestion Notification

Instead of generating additional packets to warn of congestion, a router can tag any packet it forwards (by setting a bit in the packet's header) to signal that it is experiencing congestion. When the network delivers the packet, the destination can note that there is congestion and inform the sender when it sends a reply packet. The sender can then throttle its transmissions as before.

This design is called **ECN** (**Explicit Congestion Notification**) and is used in the Internet (Ramakrishnan et al., 2001). It is a refinement of early congestion signaling protocols, notably the binary feedback scheme of Ramakrishnan and Jain (1988) that was used in the DECNET architecture. Two bits in the IP packet header are used to record whether the packet has experienced congestion. Packets are unmarked when they are sent, as illustrated in Fig. 5-25. If any of the routers they pass through is congested, that router will then mark the packet as having experienced congestion as it is forwarded. The destination will then echo any marks back to the sender as an explicit congestion signal in its next reply packet. This is shown with a dashed line in the figure to indicate that it happens above the IP level (e.g., in TCP). The sender must then throttle its transmissions, as in the case of choke packets.

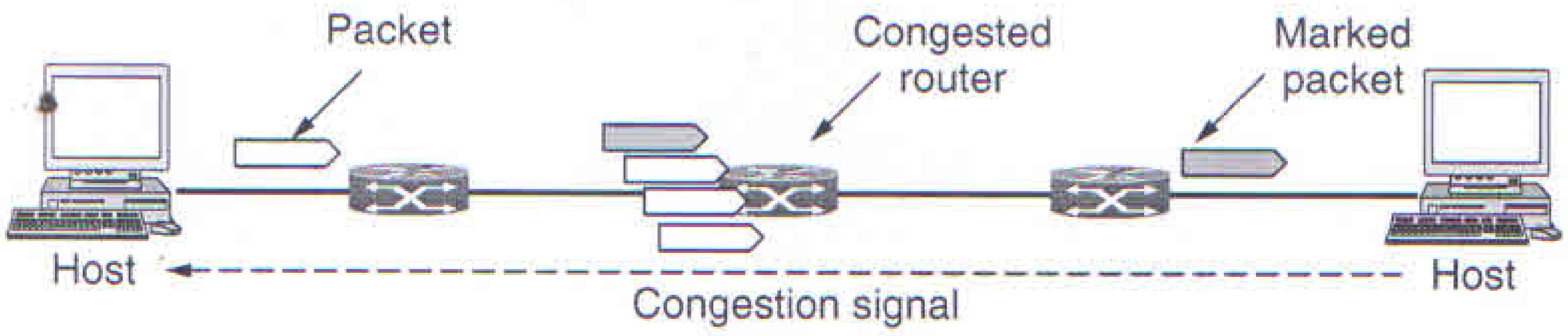

Figure 5-25. Explicit congestion notification

Hop-by-Hop Backpressure

At high speeds or over long distances, many new packets may be transmitted after congestion has been signaled because of the delay before the signal takes effect. Consider, for example, a host in San Francisco (router *A* in Fig. 5-26) that is sending traffic to a host in New York (router *D* in Fig. 5-26) at the OC-3 speed of 155 Mbps. If the New York host begins to run out of buffers, it will take about 40 msec for a choke packet to get back to San Francisco to tell it to slow down. An ECN indication will take even longer because it is delivered via the destination. Choke packet propagation is illustrated as the second, third, and fourth steps in

Fig. 5-26(a). In those 40 msec, another 6.2 megabits will have been sent. Even if the host in San Francisco completely shuts down immediately, the 6.2 megabits in the pipe will continue to pour in and have to be dealt with. Only in the seventh diagram in Fig. 5-26(a) will the New York router notice a slower flow.

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Fig. 5-26(b). Here, as soon as the choke packet reaches *F*, *F* is required to reduce the flow to *D*. Doing so will require *F* to devote more buffers to the connection, since the source is still sending away at full blast, but it gives *D* immediate relief, like a headache remedy in a television commercial. In the next step, the choke packet reaches *E*, which tells *E* to reduce the flow to *F*. This action puts a greater demand on *E*'s buffers but gives *F* immediate relief. Finally, the choke packet reaches *A* and the flow genuinely slows down.

The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion, at the price of using up more buffers upstream. In this way, congestion can be nipped in the bud without losing any packets. The idea is discussed in detail by Mishra et al. (1996).

5.3.5 Load Shedding

When none of the above methods make the congestion disappear, routers can bring out the heavy artillery: load shedding. **Load shedding** is a fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away. The term comes from the world of electrical power generation, where it refers to the practice of utilities intentionally blacking out certain areas to save the entire grid from collapsing on hot summer days when the demand for electricity greatly exceeds the supply.

The key question for a router drowning in packets is which packets to drop. The preferred choice may depend on the type of applications that use the network. For a file transfer, an old packet is worth more than a new one. This is because dropping packet 6 and keeping packets 7 through 10, for example, will only force the receiver to do more work to buffer data that it cannot yet use. In contrast, for real-time media, a new packet is worth more than an old one. This is because packets become useless if they are delayed and miss the time at which they must be played out to the user.

The former policy (old is better than new) is often called **wine** and the latter (new is better than old) is often called **milk** because most people would rather drink new milk and old wine than the alternative.

More intelligent load shedding requires cooperation from the senders. An example is packets that carry routing information. These packets are more important than regular data packets because they establish routes; if they are lost, the network may lose connectivity. Another example is that algorithms for compressing video, like MPEG, periodically transmit an entire frame and then send subsequent

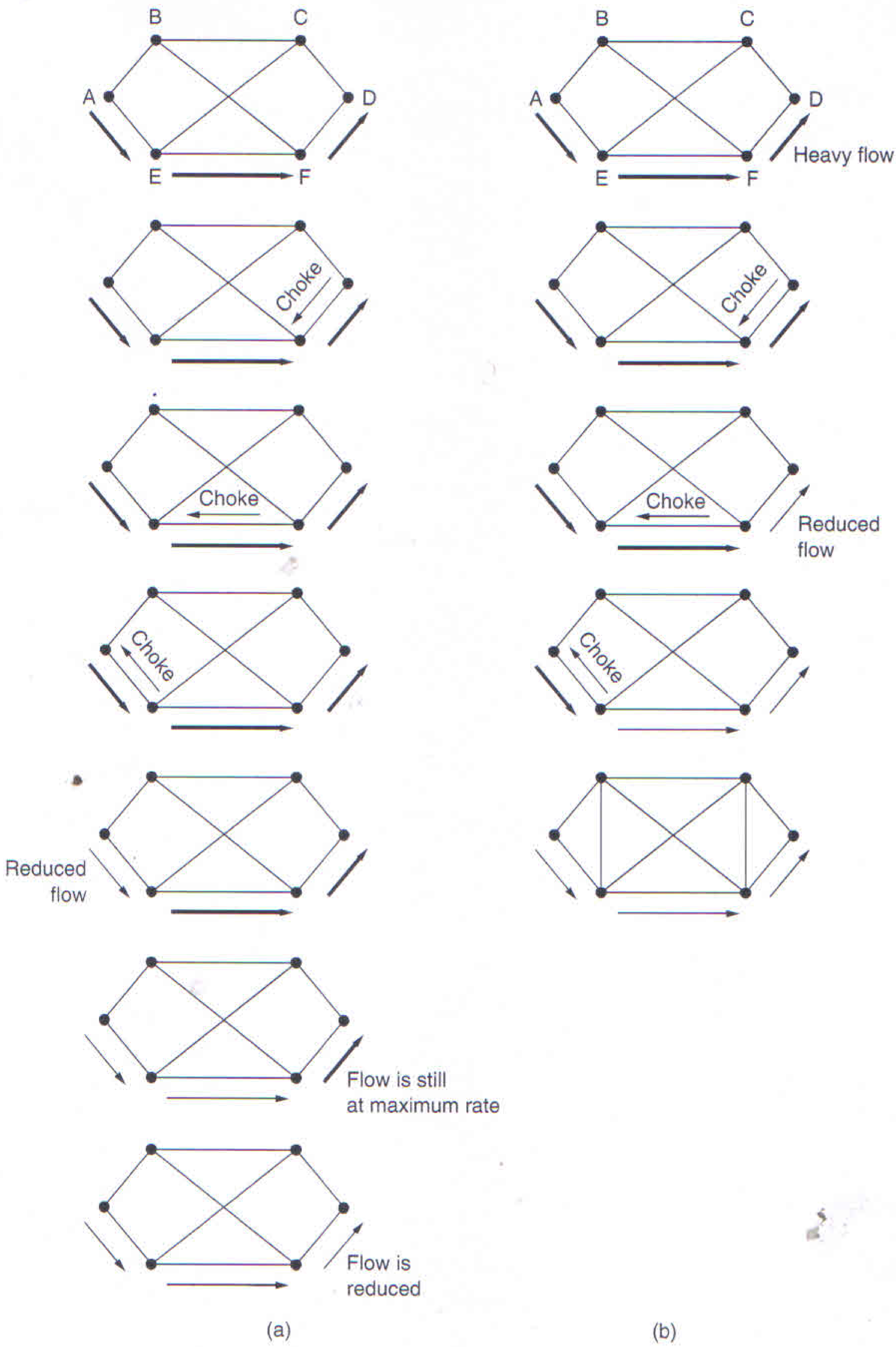

Figure 5-26. (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

frames as differences from the last full frame. In this case, dropping a packet that is part of a difference is preferable to dropping one that is part of a full frame because future packets depend on the full frame.

To implement an intelligent discard policy, applications must mark their packets to indicate to the network how important they are. Then, when packets have to be discarded, routers can first drop packets from the least important class, then the next most important class, and so on.

Of course, unless there is some significant incentive to avoid marking every packet as VERY IMPORTANT—NEVER, EVER DISCARD, nobody will do it. Often accounting and money are used to discourage frivolous marking. For example, the network might let senders send faster than the service they purchased allows if they mark excess packets as low priority. Such a strategy is actually not a bad idea because it makes more efficient use of idle resources, allowing hosts to use them as long as nobody else is interested, but without establishing a right to them when times get tough.

Random Early Detection

Dealing with congestion when it first starts is more effective than letting it gum up the works and then trying to deal with it. This observation leads to an interesting twist on load shedding, which is to discard packets before all the buffer space is really exhausted.

The motivation for this idea is that most Internet hosts do not yet get congestion signals from routers in the form of ECN. Instead, the only reliable indication of congestion that hosts get from the network is packet loss. After all, it is difficult to build a router that does not drop packets when it is overloaded. Transport protocols such as TCP are thus hardwired to react to loss as congestion, slowing down the source in response. The reasoning behind this logic is that TCP was designed for wired networks and wired networks are very reliable, so lost packets are mostly due to buffer overruns rather than transmission errors. Wireless links must recover transmission errors at the link layer (so they are not seen at the network layer) to work well with TCP.

This situation can be exploited to help reduce congestion. By having routers drop packets early, before the situation has become hopeless, there is time for the source to take action before it is too late. A popular algorithm for doing this is called **RED** (**Random Early Detection**) (Floyd and Jacobson, 1993). To determine when to start discarding, routers maintain a running average of their queue lengths. When the average queue length on some link exceeds a threshold, the link is said to be congested and a small fraction of the packets are dropped at random. Picking packets at random makes it more likely that the fastest senders will see a packet drop; this is the best option since the router cannot tell which source is causing the most trouble in a datagram network. The affected sender will notice the loss when there is no acknowledgement, and then the transport protocol will slow down. The lost packet is thus delivering the same message as a choke packet, but implicitly, without the router sending any explicit signal.

RED routers improve performance compared to routers that drop packets only when their buffers are full, though they may require tuning to work well. For example, the ideal number of packets to drop depends on how many senders need to be notified of congestion. However, ECN is the preferred option if it is available. It works in exactly the same manner, but delivers a congestion signal explicitly rather than as a loss; RED is used when hosts cannot receive explicit signals.

5.4 QUALITY OF SERVICE

The techniques we looked at in the previous sections are designed to reduce congestion and improve network performance. However, there are applications (and customers) that demand stronger performance guarantees from the network than "the best that could be done under the circumstances." Multimedia applications in particular, often need a minimum throughput and maximum latency to work. In this section, we will continue our study of network performance, but now with a sharper focus on ways to provide quality of service that is matched to application needs. This is an area in which the Internet is undergoing a long-term upgrade.

An easy solution to provide good quality of service is to build a network with enough capacity for whatever traffic will be thrown at it. The name for this solution is **overprovisioning**. The resulting network will carry application traffic without significant loss and, assuming a decent routing scheme, will deliver packets with low latency. Performance doesn't get any better than this. To some extent, the telephone system is overprovisioned because it is rare to pick up a telephone and not get a dial tone instantly. There is simply so much capacity available that demand can almost always be met.

The trouble with this solution is that it is expensive. It is basically solving a problem by throwing money at it. Quality of service mechanisms let a network with less capacity meet application requirements just as well at a lower cost. Moreover, overprovisioning is based on expected traffic. All bets are off if the traffic pattern changes too much. With quality of service mechanisms, the network can honor the performance guarantees that it makes even when traffic spikes, at the cost of turning down some requests.

Four issues must be addressed to ensure quality of service:

1. What applications need from the network.
2. How to regulate the traffic that enters the network.
3. How to reserve resources at routers to guarantee performance.
4. Whether the network can safely accept more traffic.