5.4.2 Traffic Shaping

Before the network can make QoS guarantees, it must know what traffic is being guaranteed. In the telephone network, this characterization is simple. For example, a voice call (in uncompressed format) needs 64 kbps and consists of one 8-bit sample every 125 μsec. However, traffic in data networks is **bursty**. It typically arrives at nonuniform rates as the traffic rate varies (e.g., videoconferencing with compression), users interact with applications (e.g., browsing a new Web page), and computers switch between tasks. Bursts of traffic are more difficult to handle than constant-rate traffic because they can fill buffers and cause packets to be lost.

Traffic shaping is a technique for regulating the average rate and burstiness of a flow of data that enters the network. The goal is to allow applications to transmit a wide variety of traffic that suits their needs, including some bursts, yet have a simple and useful way to describe the possible traffic patterns to the network. When a flow is set up, the user and the network (i.e., the customer and the provider) agree on a certain traffic pattern (i.e., shape) for that flow. In effect, the customer says to the provider "My transmission pattern will look like this; can you handle it?"

Sometimes this agreement is called an **SLA** (**Service Level Agreement**), especially when it is made over aggregate flows and long periods of time, such as all of the traffic for a given customer. As long as the customer fulfills her part of the bargain and only sends packets according to the agreed-on contract, the provider promises to deliver them all in a timely fashion.

Traffic shaping reduces congestion and thus helps the network live up to its promise. However, to make it work, there is also the issue of how the provider can tell if the customer is following the agreement and what to do if the customer is not. Packets in excess of the agreed pattern might be dropped by the network, or they might be marked as having lower priority. Monitoring a traffic flow is called **traffic policing**.

Shaping and policing are not so important for peer-to-peer and other transfers that will consume any and all available bandwidth, but they are of great importance for real-time data, such as audio and video connections, which have stringent quality-of-service requirements.

Leaky and Token Buckets

We have already seen one way to limit the amount of data an application sends: the sliding window, which uses one parameter to limit how much data is in transit at any given time, which indirectly limits the rate. Now we will look at a more general way to characterize traffic, with the leaky bucket and token bucket algorithms. The formulations are slightly different but give an equivalent result.

Try to imagine a bucket with a small hole in the bottom, as illustrated in Fig. 5-28(b). No matter the rate at which water enters the bucket, the outflow is at a constant rate, *R*, when there is any water in the bucket and zero when the bucket is empty. Also, once the bucket is full to capacity *B*, any additional water entering it spills over the sides and is lost.

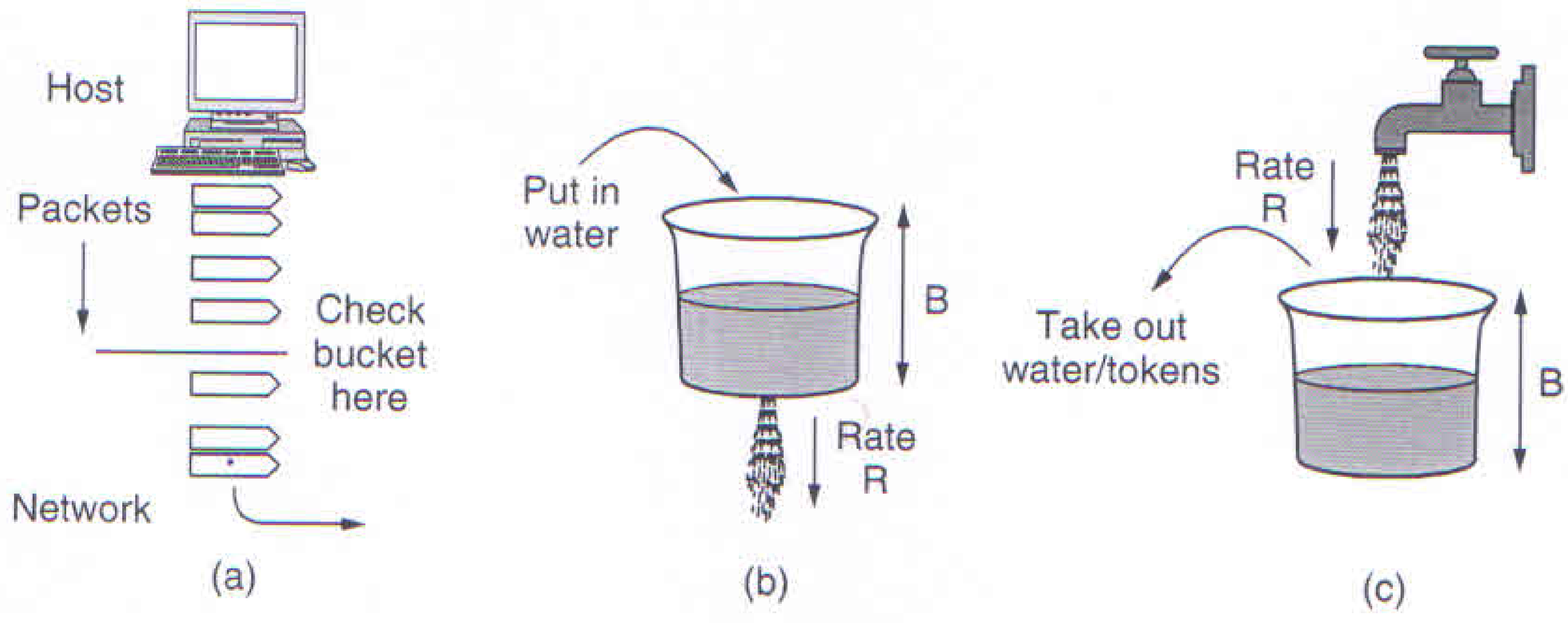

Figure 5-28. (a) Shaping packets. (b) A leaky bucket. (c) A token bucket.

This bucket can be used to shape or police packets entering the network, as shown in Fig. 5-28(a). Conceptually, each host is connected to the network by an interface containing a leaky bucket. To send a packet into the network, it must be possible to put more water into the bucket. If a packet arrives when the bucket is full, the packet must either be queued until enough water leaks out to hold it or be discarded. The former might happen at a host shaping its traffic for the network as part of the operating system. The latter might happen in hardware at a provider network interface that is policing traffic entering the network. This technique was proposed by Turner (1986) and is called the **leaky bucket algorithm**.

A different but equivalent formulation is to imagine the network interface as a bucket that is being filled, as shown in Fig. 5-28(c). The tap is running at rate *R* and the bucket has a capacity of *B*, as before. Now, to send a packet we must be able to take water, or tokens, as the contents are commonly called, out of the bucket (rather than putting water into the bucket). No more than a fixed number of tokens, *B*, can accumulate in the bucket, and if the bucket is empty, we must wait until more tokens arrive before we can send another packet. This algorithm is called the **token bucket algorithm**.

Leaky and token buckets limit the long-term rate of a flow but allow short-term bursts up to a maximum regulated length to pass through unaltered and without suffering any artificial delays. Large bursts will be smoothed by a leaky bucket traffic shaper to reduce congestion in the network. As an example, imagine that a computer can produce data at up to 1000 Mbps (125 million bytes/sec) and that the first link of the network also runs at this speed. The pattern of traffic the host generates is shown in Fig. 5-29(a). This pattern is bursty. The average rate over one second is 200 Mbps, even though the host sends a burst of 16,000 KB at the top speed of 1000 Mbps (for 1/8 of the second).

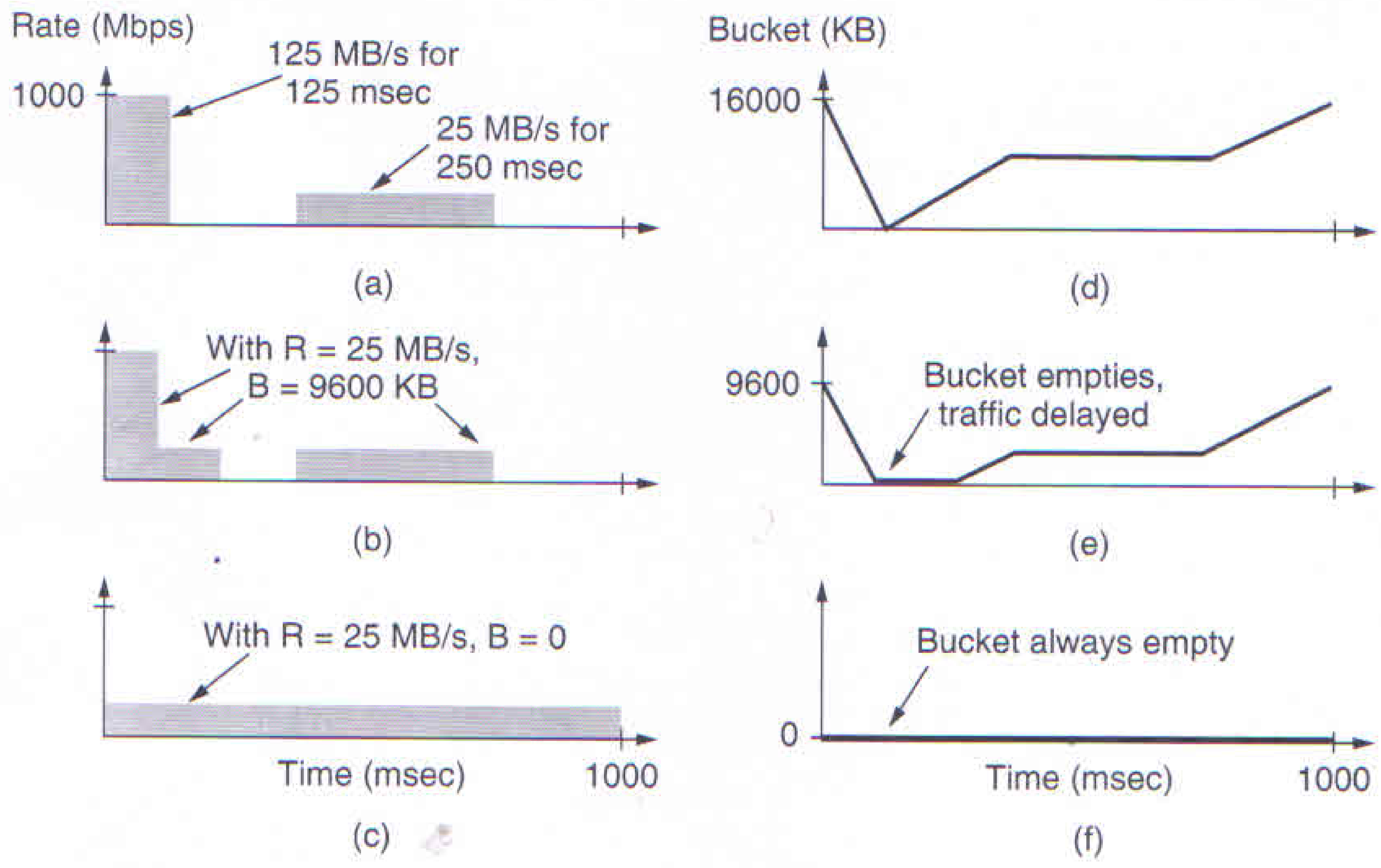

Figure 5-29. (a) Traffic from a host. Output shaped by a token bucket of rate 200 Mbps and capacity (b) 9600 KB and (c) 0 KB. Token bucket level for shaping with rate 200 Mbps and capacity (d) 16,000 KB, (e) 9600 KB, and (f) 0 KB.

Now suppose that the routers can accept data at the top speed only for short intervals, until their buffers fill up. The buffer size is 9600 KB, smaller than the traffic burst. For long intervals, the routers work best at rates not exceeding 200 Mbps (say, because this is all the bandwidth given to the customer). The implication is that if traffic is sent in this pattern, some of it will be dropped in the network because it does not fit into the buffers at routers.

To avoid this packet loss, we can shape the traffic at the host with a token bucket. If we use a rate, *R*, of 200 Mbps and a capacity, *B*, of 9600 KB, the traffic will fall within what the network can handle. The output of this token bucket is shown in Fig. 5-29(b). The host can send full throttle at 1000 Mbps for a short while until it has drained the bucket. Then it has to cut back to 200 Mbps until the burst has been sent. The effect is to spread out the burst over time because it was too large to handle all at once. The level of the token bucket is shown in Fig. 5-29(e). It starts off full and is depleted by the initial burst. When it reaches zero, new packets can be sent only at the rate at which the buffer is filling; there can be no more bursts until the bucket has recovered. The bucket fills when no traffic is being sent and stays flat when traffic is being sent at the fill rate.

We can also shape the traffic to be less bursty. Fig. 5-29(c) shows the output of a token bucket with *R* = 200 Mbps and a capacity of 0. This is the extreme case in which the traffic has been completely smoothed. No bursts are allowed, and the traffic enters the network at a steady rate. The corresponding bucket level, shown in Fig. 5-29(f), is always empty. Traffic is being queued on the host for release into the network and there is always a packet waiting to be sent when it is allowed.

Finally, Fig. 5-29(d) shows the bucket level for a token bucket with $R = 200$ Mbps and a capacity of $B = 16{,}000$ KB. This is the smallest token bucket through which the traffic passes unaltered. It might be used at a router in the network to police the traffic that the host sends. If the host is sending traffic that conforms to the token bucket on which it has agreed with the network, the traffic will fit through that same token bucket run at the router at the edge of the network. If the host sends at a faster or burstier rate, the token bucket will run out of water. If this happens, a traffic policer will know that the traffic is not as described. It will then either drop the excess packets or lower their priority, depending on the design of the network. In our example, the bucket empties only momentarily, at the end of the initial burst, then recovers enough for the next burst.

Leaky and token buckets are easy to implement. We will now describe the operation of a token bucket. Even though we have described water flowing continuously into and out of the bucket, real implementations must work with discrete quantities. A token bucket is implemented with a counter for the level of the bucket. The counter is advanced by $R/\Delta T$ units at every clock tick of ΔT seconds. This would be 200 Kbit every 1 msec in our example above. Every time a unit of traffic is sent into the network, the counter is decremented, and traffic may be sent until the counter reaches zero.

When the packets are all the same size, the bucket level can just be counted in packets (e.g., 200 Mbit is 20 packets of 1250 bytes). However, often variable-sized packets are being used. In this case, the bucket level is counted in bytes. If the residual byte count is too low to send a large packet, the packet must wait until the next tick (or even longer, if the fill rate is small).

Calculating the length of the maximum burst (until the bucket empties) is slightly tricky. It is longer than just 9600 KB divided by 125 MB/sec because while the burst is being output, more tokens arrive. If we call the burst length S sec., the maximum output rate M bytes/sec, the token bucket capacity B bytes, and the token arrival rate R bytes/sec, we can see that an output burst contains a maximum of $B + RS$ bytes. We also know that the number of bytes in a maximum-speed burst of length S seconds is MS. Hence, we have

$$B + RS = MS$$

We can solve this equation to get $S = B/(M - R)$. For our parameters of $B = 9600$ KB, $M = 125$ MB/sec, and $R = 25$ MB/sec, we get a burst time of about 94 msec.

A potential problem with the token bucket algorithm is that it reduces large bursts down to the long-term rate R. It is frequently desirable to reduce the peak rate, but without going down to the long-term rate (and also without raising the long-term rate to allow more traffic into the network). One way to get smoother traffic is to insert a second token bucket after the first one. The rate of the second bucket should be much higher than the first one. Basically, the first bucket characterizes the traffic, fixing its average rate but allowing some bursts. The second bucket reduces the peak rate at which the bursts are sent into the network. For example, if the rate of the second token bucket is set to be 500 Mbps and the capacity is set to 0, the initial burst will enter the network at a peak rate of 500 Mbps, which is lower than the 1000 Mbps rate we had previously.

Using all of these buckets can be a bit tricky. When token buckets are used for traffic shaping at hosts, packets are queued and delayed until the buckets permit them to be sent. When token buckets are used for traffic policing at routers in the network, the algorithm is simulated to make sure that no more packets are sent than permitted. Nevertheless, these tools provide ways to shape the network traffic into more manageable forms to assist in meeting quality-of-service requirements.

5.4.3 Packet Scheduling

Being able to regulate the shape of the offered traffic is a good start. However, to provide a performance guarantee, we must reserve sufficient resources along the route that the packets take through the network. To do this, we are assuming that the packets of a flow follow the same route. Spraying them over routers at random makes it hard to guarantee anything. As a consequence, something similar to a virtual circuit has to be set up from the source to the destination, and all the packets that belong to the flow must follow this route.

Algorithms that allocate router resources among the packets of a flow and between competing flows are called **packet scheduling algorithms**. Three different kinds of resources can potentially be reserved for different flows:

1. Bandwidth.
2. Buffer space.
3. CPU cycles.

The first one, bandwidth, is the most obvious. If a flow requires 1 Mbps and the outgoing line has a capacity of 2 Mbps, trying to direct three flows through that line is not going to work. Thus, reserving bandwidth means not oversubscribing any output line.

A second resource that is often in short supply is buffer space. When a packet arrives, it is buffered inside the router until it can be transmitted on the chosen outgoing line. The purpose of the buffer is to absorb small bursts of traffic as the flows contend with each other. If no buffer is available, the packet has to be discarded since there is no place to put it. For good quality of service, some buffers might be reserved for a specific flow so that flow does not have to compete for