ATM LAN, you need only send it to the BUS, which then forwards it on the multipoint VC.

It should be clear that ATM LAN Emulation is fairly complex, and the BUS in particular presents a scalability bottleneck. Perhaps as a result of these factors, plus the fact that ATM ultimately offered few real advantages over Ethernet, LANE is no longer widely used.

3.4 Implementation and Performance

So far, we have talked about what a switch must do without discussing how to do it. There is a very simple way to build a switch: Buy a general-purpose workstation and equip it with a number of network interfaces. Such a device, running suitable software, can receive packets on one of its interfaces, perform any of the switching functions described above, and send packets out another of its interfaces. This is, in fact, a popular way to build experimental switches when you want to be able to do things like develop new routing protocols because it offers extreme flexibility and a familiar programming environment. It is also not too far removed from the architecture of many low-end routers (which, as we will see in the next chapter, have much in common with switches).

Figure 3.24 shows a workstation with three network interfaces used as a switch. The figure shows a path that a packet might take from the time it arrives on interface 1 until it is output on interface 2. We have assumed here that the workstation has a mechanism to move data directly from an interface to its main memory without having to be directly copied by the CPU, that is, direct memory access (DMA) as described in Section 2.1.1. Once the packet is in memory, the CPU examines its header to determine which interface the packet should be sent out on. It then uses DMA to move the packet out to the appropriate interface. Note that Figure 3.24 does not show the packet going to the CPU because the CPU inspects only the header of the packet; it does not have to read every byte of data in the packet.

The main problem with using a workstation as a switch is that its performance is limited by the fact that all packets must pass through a single point of contention: In the example shown, each packet crosses the I/O bus twice and is written to and read from main memory once. The upper bound on aggregate throughput of such a device (the total sustainable data rate summed over all inputs) is, thus, either half the main memory bandwidth or half the I/O bus bandwidth, whichever is less. (Usually, it's the I/O bus bandwidth.) For example, a workstation with a 133MHz, 64-bit wide I/O bus can transmit data at a peak rate of a little over 8Gbps. Since forwarding a packet involves crossing the bus twice, the actual limit is 4Gbps—enough to build a switch with a fair number of 100Mbps Ethernet ports, for example, but hardly enough for a high-end router in the core of the Internet. (We'll return to the subject of router implementation in Section 4.2.6.)

Moreover, this upper bound also assumes that moving data is the only problem—a fair approximation for long packets but a bad one when packets are short. In the latter case, the cost of processing each packet—parsing its header and deciding which output link to transmit it on—is likely to dominate. Suppose, for example, that a workstation

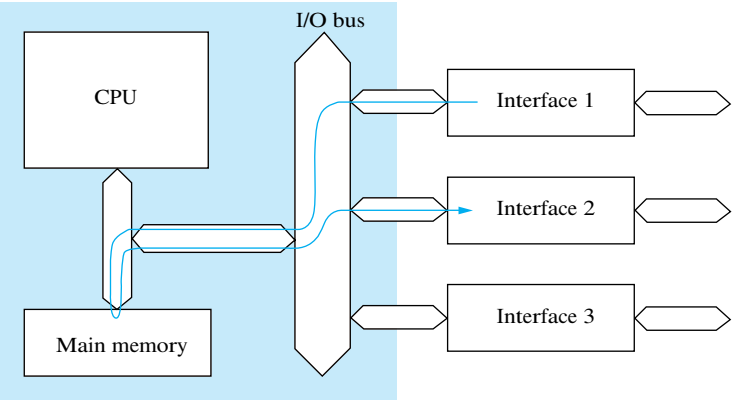


Figure 3.24: A workstation used as a packet switch.

can perform all the necessary processing to switch 1 million packets each second. This is sometimes called the packet per second (pps) rate. (This number is representative of what is achievable on today's high-end PCs.) If the average packet is short, say, 64 bytes, this would imply

$$\begin{aligned} \mathsf{Throughput} &= \mathsf{pps} \times (\mathsf{BitsPerPacket}) \\ &= 1 \times 10^6 \times 64 \times 8 \\ &= 512 \times 10^6 \end{aligned}$$

that is, a throughput of 512 Mbps—substantially below the range that users are demanding from their networks today. Bear in mind that this 512 Mbps would be shared by all users connected to the switch, just as the 10 Mbps of an Ethernet is shared among all users connected to the shared medium. Thus, for example, a 10-port switch with this aggregate throughput would only be able to cope with an average data rate of 51.2 Mbps on each port.

To address this problem, hardware designers have come up with a large array of switch designs that reduce the amount of contention and provide high aggregate throughput. Note that some contention is unavoidable: If every input has data to send to a single output, then they cannot all send it at once. However, if data destined for different outputs is arriving at different inputs, a well-designed switch will be able to move data from inputs to outputs in parallel, thus increasing the aggregate throughput.

> **Sidebar: Defining Throughput** It turns out to be difficult to define precisely the throughput of a switch. Intuitively, we might think that if a switch has n inputs that each support a link speed of s_i, then the throughput would just be the sum of all the s_i. This is actually the best possible throughput that such a switch could provide, but in practice almost no real switch can guarantee that level of performance. One reason for this is simple to understand. Suppose that, for some period of time, all the traffic arriving at the switch needed to be sent to the same output. As long as the

bandwidth of that output is less than the sum of the input bandwidths, then some of the traffic will need to be either buffered or dropped. With this particular traffic pattern, the switch could not provide a sustained throughput higher than the link speed of that one output. However, a switch might be able to handle traffic arriving at the full link speed on all inputs if it is distributed across all the outputs evenly; this would be considered optimal.

Another factor that affects the performance of switches is the size of packets arriving on the inputs. For an ATM switch, this is normally not an issue because all "packets" (cells) are the same length. But for Ethernet switches or IP routers, packets of widely varying sizes are possible. Some of the operations that a switch must perform have a constant overhead per packet, so a switch is likely to perform differently depending on whether all arriving packets are very short, very long, or mixed. For this reason, routers or switches that forward variable-length packets are often characterized by a *packet per second* (pps) rate as well as a throughput in bits per second. The pps rate is usually measured with minimum-sized packets.

The first thing to notice about this discussion is that the throughput of the switch is a function of the traffic to which it is subjected. One of the things that switch designers spend a lot of their time doing is trying to come up with traffic models that approximate the behavior of real data traffic. It turns out that it is extremely difficult to achieve accurate models. There are several elements to a traffic model. The main ones are (1) when do packets arrive, (2) what outputs are they destined for, and (3) how big are they.

Traffic modeling is a well-established science that has been extremely successful in the world of telephony, enabling telephone companies to engineer their networks to carry expected loads quite efficiently. This is partly because the way people use the phone network does not change that much over time: The frequency with which calls are placed, the amount of time taken for a call, and the tendency of everyone to make calls on Mother's Day have stayed fairly constant for many years.[6] By contrast, the rapid evolution of computer communications, where a new application like Napster can change the traffic patterns almost overnight, has made effective modeling of computer networks much more difficult. Nevertheless, there are some excellent books and articles on the subject that we list at the end of the chapter.

To give you a sense of the range of throughputs that designers need to be concerned about, a high-end router used in the Internet at the time of writing might support 10 OC-768 links for a throughput of approximately 400 Gbps. A 400-Gbps switch, if called upon to handle a steady stream of 64-byte packets, would need a packet per second rate of

$$400 \times 10^9 \div (64 \times 8) = 781 \times 10^6 \text{ pps}$$

[6]The advent of dial-up connections to the Internet did however cause a significant change in the average length of calls.

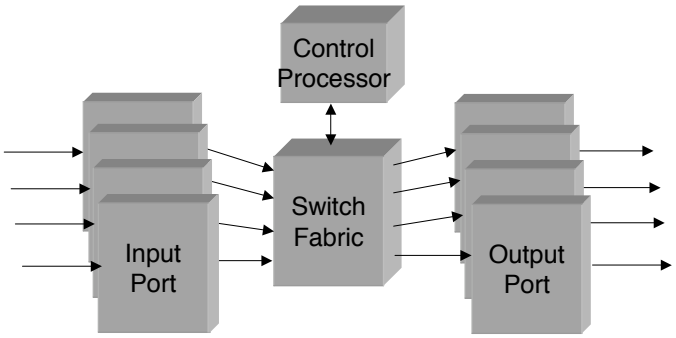


Figure 3.25: A 4 × 4 switch.

3.4.1 Ports

Most switches look conceptually similar to the one shown in Figure 3.25. They consist of a number of *input ports* and *output ports*, and a *fabric*. There is usually at least one control processor in charge of the whole switch that communicates with the ports either directly or, as shown here, via the switch fabric. The ports communicate with the outside world. They may contain fiber optic receivers and lasers, buffers to hold packets that are waiting to be switched or transmitted, and often a significant amount of other circuitry that enables the switch to function. The fabric has a very simple and well-defined job: When presented with a packet, deliver it to the right output port.

One of the jobs of the ports, then, is to deal with the complexity of the real world in such a way that the fabric can do its relatively simple job. For example, suppose that this switch is supporting a virtual circuit model of communication. In general, the virtual circuit mapping tables described in Section 3.1.2 are located in the ports. The ports maintain lists of virtual circuit identifiers that are currently in use, with information about what output a packet should be sent out on for each VCI and how the VCI needs to be remapped to ensure uniqueness on the outgoing link. Similarly, the ports of an Ethernet switch store tables that map between Ethernet addresses and output ports (bridge forwarding tables as described in Section 3.2). In general, when a packet is handed from an input port to the fabric, the port has figured out where the packet needs to go, and either the port sets up the fabric accordingly by communicating some control information to it, or it attaches enough information to the packet itself (e.g., an output port number) to allow the fabric to do its job automatically. Fabrics that switch packets by looking only at the information in the packet are referred to as "self-routing," since they require no external control to route packets. An example of a self-routing fabric is discussed below.

The input port is the first place to look for performance bottlenecks. The input port has to receive a steady stream of packets, analyze information in the header of each one to determine which output port (or ports) the packet must be sent to, and pass the packet on to the fabric. The type of header analysis that it performs can range from a simple

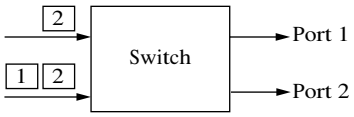


Figure 3.26: Simple illustration of head-of-line blocking.

table lookup on a VCI to complex matching algorithms that examine many fields in the header. This is the type of operation that sometimes becomes a problem when the average packet size is very small. Consider, for example, 64-byte packets arriving on a port connected to an OC-48 (2.48 Gbps) link. Such a port needs to process packets at a rate of

$$2.48 \times 10^9 \div (64 \times 8) = 4.83 \times 10^6 \text{ pps}$$

In other words, when small packets are arriving as fast as possible on this link (the worst-case scenario that most ports are engineered to handle), the input port has approximate 200 nanoseconds to process each packet.

Another key function of ports is buffering. Observe that buffering can happen in either the input or the output port; it can also happen within the fabric (sometimes called *internal buffering*). Simple input buffering has some serious limitations. Consider an input buffer implemented as a FIFO. As packets arrive at the switch, they are placed in the input buffer. The switch then tries to forward the packets at the front of each FIFO to their appropriate output port. However, if the packets at the front of several different input ports are destined for the same output port at the same time, then only one of them can be forwarded;[7] the rest must stay in their input buffers.

The drawback of this feature is that those packets left at the front of the input buffer prevent other packets further back in the buffer from getting a chance to go to their chosen outputs, even though there may be no contention for those outputs. This phenomenon is called *head-of-line blocking*. A simple example of head-of-line blocking is given in Figure 3.26, where we see a packet destined for port 1 blocked behind a packet contending for port 2. It can be shown that when traffic is uniformly distributed among outputs, head-of-line blocking limits the throughput of an input-buffered switch to 59% of the theoretical maximum (which is the sum of the link bandwidths for the switch). Thus, the majority of switches use either pure output buffering or a mixture of internal and output buffering. Those that do rely on input buffers use sophisticated buffer management schemes to avoid head-of-line blocking.

Buffers actually perform a more complex task than just holding onto packets that are waiting to be transmitted. Buffers are the main source of delay in a switch, and also the place where packets are most likely to get dropped due to lack of space to store them. The buffers therefore are the main place where the quality of service characteristics of a switch are determined. For example, if a certain packet has been sent along a VC that has a guaranteed delay, it cannot afford to sit in a buffer for very long. This means that the buffers, in general, must be managed using packet scheduling and dis-

[7] For a simple input-buffered switch, exactly one packet at a time can be sent to a given output port. It is possible to design switches that can forward more than one packet to the same output at once, at a cost of higher switch complexity, but there is always some upper limit on the number.

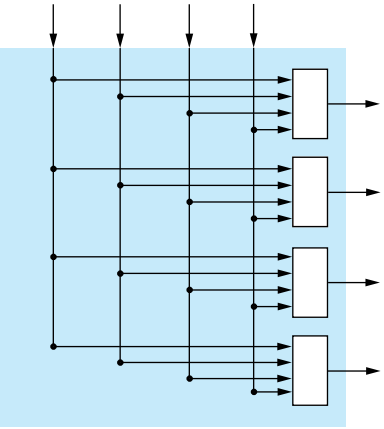


Figure 3.27: A 4 $\times$ 4 crossbar switch.

card algorithms that meet a wide range of QoS requirements. We talk more about these issues in Chapter 6.

3.4.2 Fabrics

While there has been an abundance of impressive research conducted on the design of efficient and scalable fabrics, it is sufficient for our purposes here to understand only the high level properties of a switch fabric. A switch fabric should be able to move packets from input ports to output ports with minimal delay and in a way that meets the throughput goals of the switch. That usually means that fabrics display some degree of parallelism. A high-performance fabric with n ports can often move one packet from each of its n ports to one of the output ports at the same time. A sample of fabric types includes the following:

- Shared Bus. This is the type of "fabric" found in a conventional workstation used as a switch, as described above. Because the bus bandwidth determines the throughput of the switch, high-performance switches usually have specially designed busses rather than the standard busses found in PCs.

- Shared Memory. In a shared memory switch, packets are written into a memory location by an input port and then read from memory by the output ports. Here it is the memory bandwidth that determines switch throughput, so wide and fast memory is typically used in this sort of design. A shared memory switch is similar in principle to the shared bus switch, except it usually uses a specially-designed, high-speed memory bus rather than an I/O bus.

- Crossbar. A crossbar switch is a matrix of pathways that can be configured to connect any input port to any output port. Figure 3.27 shows a 4×4 crossbar switch. The main problem with crossbars is that, in their simplest form, they require each output port to be able to accept packets from all inputs at once,

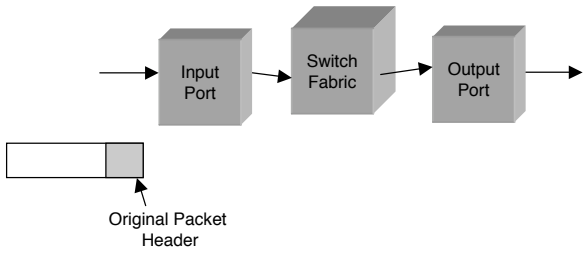
implying that each port would have a memory bandwidth equal to the total switch throughput. In reality, more complex designs are typically used to address this issue (see, for example, the Knockout switch and McKeown's virtual output-buffered approach in the Further Reading section.)

- Self-routing. As noted above, self-routing fabrics rely on some information in the packet header to direct each packet to its correct output. Usually a special "self-routing header" is appended to the packet by the input port after it has determined which output the packet needs to go to, as illustrated in Figure 3.28; this extra header is removed before the packet leaves the switch. Self-routing fabrics are often built from large numbers of very simple 2×2 switching elements interconnected in regular patterns, such as the *banyan* switching fabric shown in Figure 3.29. For some examples of self-routing fabric designs see the Further Reading section at the end of this chapter.

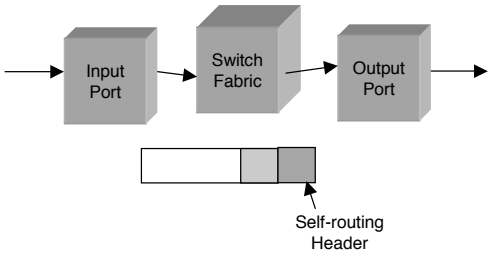
Self-routing fabrics are among the most scalable approaches to fabric design, and there has been a wealth of research on the topic, some of which is listed in the Further Reading section. Many self-routing fabrics resemble the one shown in Figure 3.29, consisting of regularly interconnected 2×2 switching elements. For example, the 2×2 switches in the banyan network perform a simple task: they look at 1 bit in each self-routing header and route packets toward the upper output if it is zero or toward the lower output if it is one. Obviously, if two packets arrive at a banyan element at the same time and both have the bit set to the same value, then they want to be routed to the same output and a collision will occur. Either preventing or dealing with these collisions is a main challenge for self-routing switch design. The banyan network is a clever arrangement of 2×2 switching elements that routes all packets to the correct output without collisions if the packets are presented in ascending order.

We can see how this works in an example, as shown in Figure 3.29, where the self-routing header contains the output port number encoded in binary. The switch elements in the first column look at the most significant bit of the output port number and route packets to the top if that bit is a 0 or the bottom if it is a 1. Switch elements in the second column look at the second bit in the header, and those in the last column look at the least significant bit. You can see from this example that the packets are routed to the correct destination port without collisions. Notice how the top outputs from the first column of switches all lead to the top half of the network, thus getting packets with port numbers 0–3 into the right half of the network. The next column gets packets to the right quarter of the network, and the final column gets them to the right output port. The clever part is the way switches are arranged to avoid collisions. Part of the arrangement includes the "perfect shuffle" wiring pattern at the start of the network. To build a complete switch fabric around a banyan network would require additional components to sort packets before they are presented to the banyan. The Batcher-banyan switch design is a notable example of such an approach. The Batcher network, which is also built from a regular interconnection of 2×2 switching elements, sorts packets into descending order. On leaving the Batcher network, the packets are then ready to be directed to the correct output, with no risk of collisions, by the banyan network.

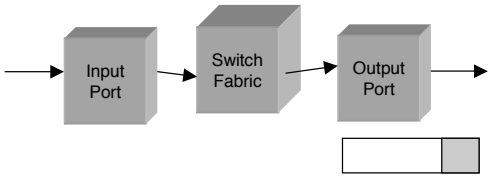
One of the interesting things about switch design is the wide range of different types of switch that can be built using the same basic technology. For example, both



(a) Packet arrives at input port



(b) Input port attaches self-routing header to direct packet to correct output



(c) Self-routing header is removed at output port before packet leaves switch

Figure 3.28: A self-routing header is applied to a packet at input to enable the fabric to send the packet to the correct output, where it is removed.

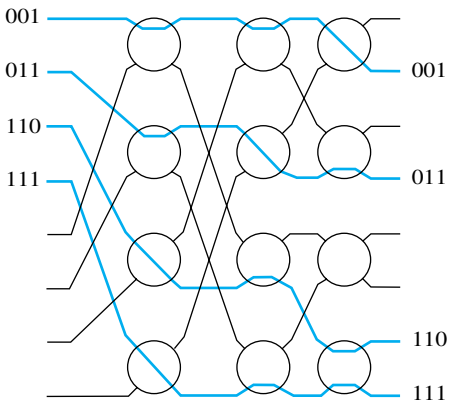


Figure 3.29: Routing packets through a banyan network. The 3-bit numbers represent values in the self-routing headers of four arriving packets.

the Ethernet switches and ATM switches discussed in this chapter, as well as Internet routers discussed in the next chapter, are all built using designs such as those outlined in this section.

3.5 Summary

This chapter has started to look at some of the issues involved in building large scalable networks by using switches, rather than just links, to interconnect hosts. There are several different ways to decide how to switch packets; the two main ones are the datagram (connectionless) model and the virtual circuit (connection-oriented) model.

An important application of switching is the interconnection of shared-media LANs. LAN switches, or bridges, use techniques such as source address learning to improve forwarding efficiency, and spanning tree algorithms to avoid looping. These switches are extensively used in data centers, campuses and corporate networks.

The most widespread uses of virtual circuit switching are in Frame Relay and ATM switches. ATM introduces some particular challenges through the use of cells—short, fixed-length packets. The availability of relatively high-throughput ATM switches has contributed to the acceptance of the technology, although it has certainly not swept all other technologies aside as some predicted. One of the main uses of ATM today is as a multiplexing technology in DSL access networks.

Independent of the specifics of the switching technology, switches need to forward packets from inputs to outputs at a high rate, and in some circumstances, switches need to grow to a large size to accommodate hundreds or thousands of ports. Building switches that both scale and offer high performance at acceptable cost is complicated by the problem of contention, and as a consequence, switches often employ special-purpose hardware rather than being built from general-purpose workstations.

In addition to the issues of contention discussed here, we observe that the related problem of congestion has come up throughout this chapter. We will postpone our dis-