

among routers along an end-to-end path from the source host to the destination host. We'll cover routing algorithms, as well as routing protocols, such as OSPF and BGP, that are in widespread use in today's Internet. Traditionally, these control-plane routing protocols and data-plane forwarding functions have been implemented together, monolithically, within a router. Software-defined networking (SDN) explicitly separates the data plane and control plane by implementing these control plane functions as a separate service, typically in a remote "controller." We'll also cover SDN controllers in Chapter 5.

This distinction between data-plane and control-plane functions in the network layer is an important concept to keep in mind as you learn about the network layer—it will help structure your thinking about the network layer and reflects a modern view of the network layer's role in computer networking.

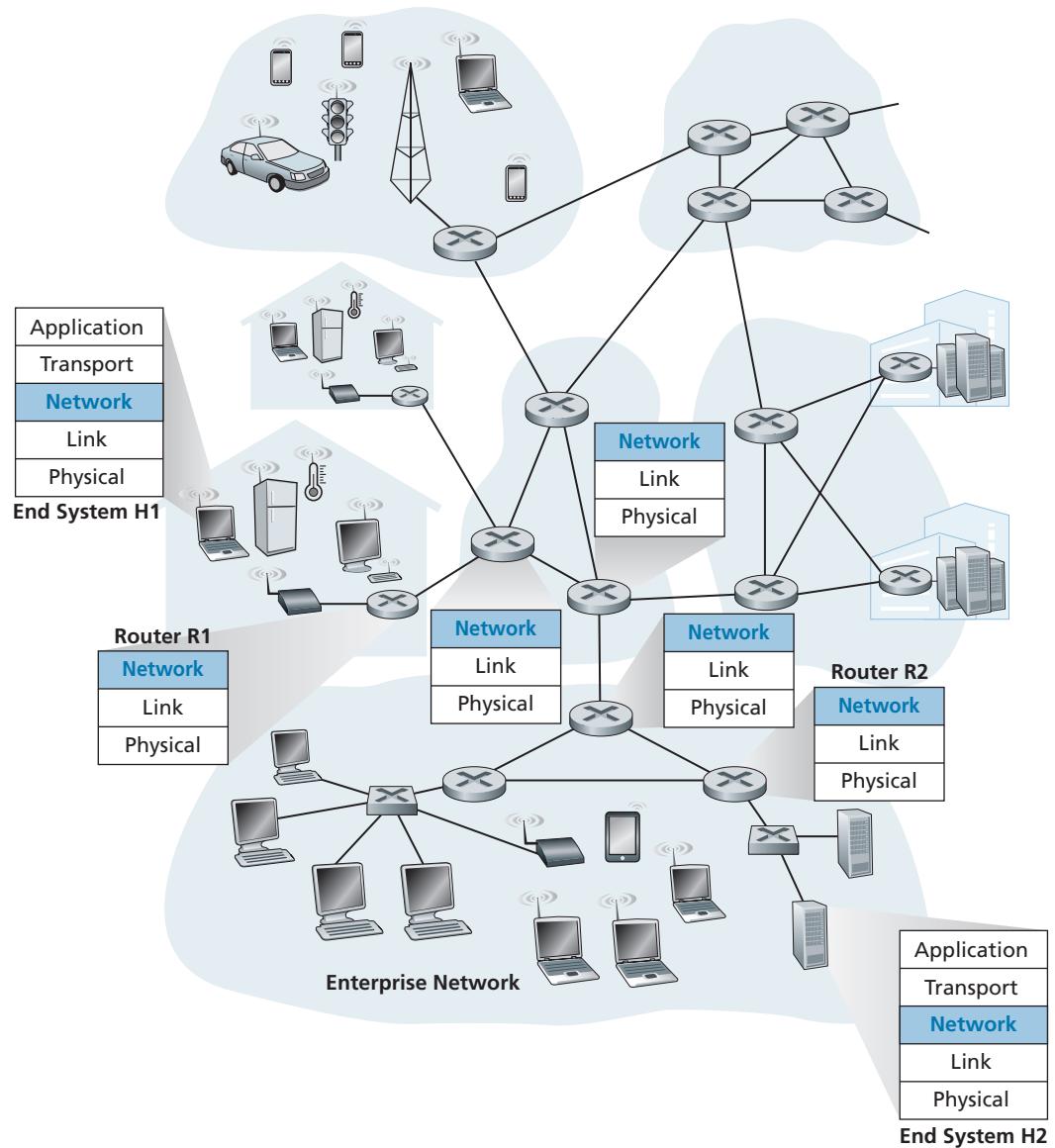
## 4.1 Overview of Network Layer

Figure 4.1 shows a simple network with two hosts, H1 and H2, and several routers on the path between H1 and H2. Let's suppose that H1 is sending information to H2, and consider the role of the network layer in these hosts and in the intervening routers. The network layer in H1 takes segments from the transport layer in H1, encapsulates each segment into a datagram, and then sends the datagrams to its nearby router, R1. At the receiving host, H2, the network layer receives the datagrams from its nearby router R2, extracts the transport-layer segments, and delivers the segments up to the transport layer at H2. The primary data-plane role of each router is to forward datagrams from its input links to its output links; the primary role of the network control plane is to coordinate these local, per-router forwarding actions so that datagrams are ultimately transferred end-to-end, along paths of routers between source and destination hosts. Note that the routers in Figure 4.1 are shown with a truncated protocol stack, that is, with no upper layers above the network layer, because routers do not run application- and transport-layer protocols such as those we examined in Chapters 2 and 3.

### 4.1.1 Forwarding and Routing: The Data and Control Planes

The primary role of the network layer is deceptively simple—to move packets from a sending host to a receiving host. To do so, two important network-layer functions can be identified:

- *Forwarding.* When a packet arrives at a router's input link, the router must move the packet to the appropriate output link. For example, a packet arriving from Host H1 to Router R1 in Figure 4.1 must be forwarded to the next router on a path to H2. As we will see, forwarding is but one function (albeit the most



**Figure 4.1** ♦ The network layer

common and important one!) implemented in the data plane. In the more general case, which we'll cover in Section 4.4, a packet might also be blocked from exiting a router (for example, if the packet originated at a known malicious sending host, or if the packet were destined to a forbidden destination host), or might be duplicated and sent over multiple outgoing links.

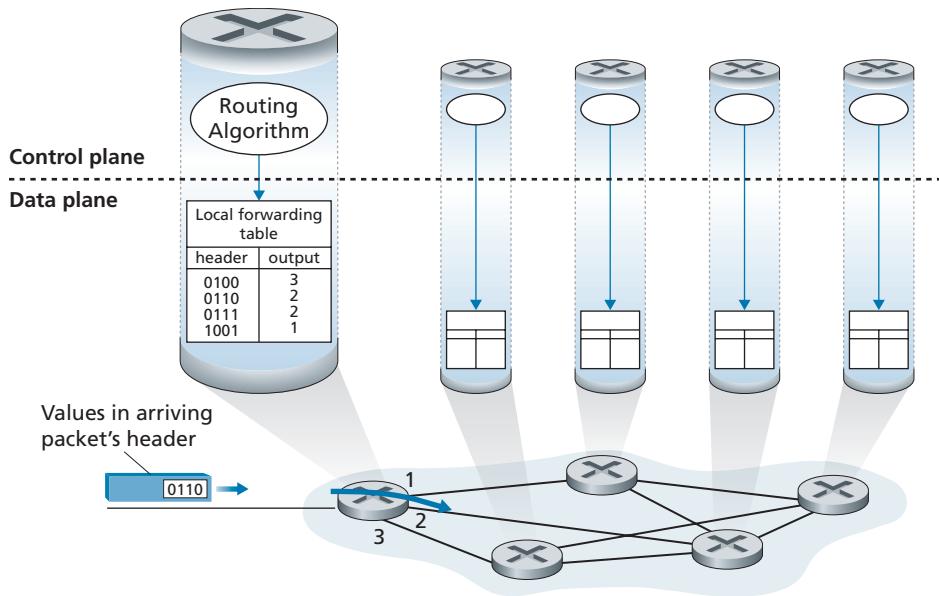
- *Routing*. The network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as **routing algorithms**. A routing algorithm would determine, for example, the path along which packets flow from H1 to H2 in Figure 4.1. Routing is implemented in the control plane of the network layer.

The terms *forwarding* and *routing* are often used interchangeably by authors discussing the network layer. We'll use these terms much more precisely in this book. **Forwarding** refers to the router-local action of transferring a packet from an input link interface to the appropriate output link interface. Forwarding takes place at very short timescales (typically a few nanoseconds), and thus is typically implemented in hardware. **Routing** refers to the network-wide process that determines the end-to-end paths that packets take from source to destination. Routing takes place on much longer timescales (typically seconds), and as we will see is often implemented in software. Using our driving analogy, consider the trip from Pennsylvania to Florida undertaken by our traveler back in Section 1.3.1. During this trip, our driver passes through many interchanges en route to Florida. We can think of forwarding as the process of getting through a single interchange: A car enters the interchange from one road and determines which road it should take to leave the interchange. We can think of routing as the process of planning the trip from Pennsylvania to Florida: Before embarking on the trip, the driver has consulted a map and chosen one of many paths possible, with each path consisting of a series of road segments connected at interchanges.

A key element in every network router is its **forwarding table**. A router forwards a packet by examining the value of one or more fields in the arriving packet's header, and then using these header values to index into its forwarding table. The value stored in the forwarding table entry for those values indicates the outgoing link interface at that router to which that packet is to be forwarded. For example, in Figure 4.2, a packet with header field value of 0110 arrives to a router. The router indexes into its forwarding table and determines that the output link interface for this packet is interface 2. The router then internally forwards the packet to interface 2. In Section 4.2, we'll look inside a router and examine the forwarding function in much greater detail. Forwarding is the key function performed by the data-plane functionality of the network layer.

## Control Plane: The Traditional Approach

But now you are undoubtedly wondering how a router's forwarding tables are configured in the first place. This is a crucial issue, one that exposes the important interplay between forwarding (in data plane) and routing (in control plane). As shown



**Figure 4.2** ♦ Routing algorithms determine values in forward tables

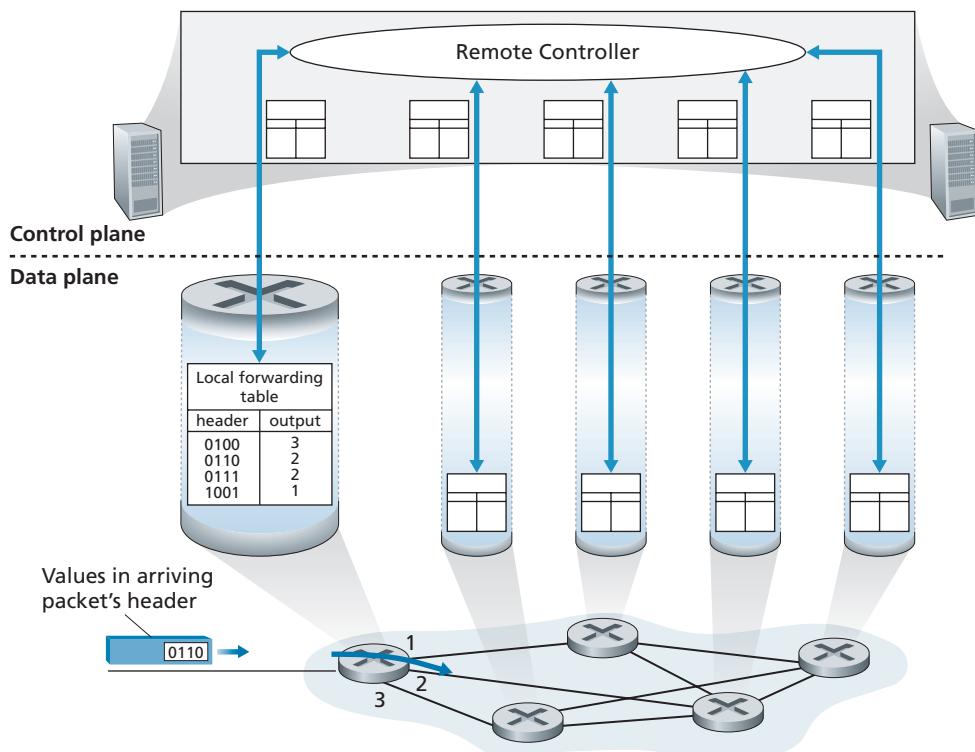
in Figure 4.2, the routing algorithm determines the contents of the routers' forwarding tables. In this example, a routing algorithm runs in each and every router and both forwarding and routing functions are contained within a router. As we'll see in Sections 5.3 and 5.4, the routing algorithm function in one router communicates with the routing algorithm function in other routers to compute the values for its forwarding table. How is this communication performed? By exchanging routing information according to a routing protocol! We'll cover routing algorithms and protocols in Sections 5.2 through 5.4.

The distinct and different purposes of the forwarding and routing functions can be further illustrated by considering the hypothetical (and unrealistic, but technically feasible) case of a network in which all forwarding tables are configured directly by human network operators physically present at the routers. In this case, *no* routing protocols would be required! Of course, the human operators would need to interact with each other to ensure that the forwarding tables were configured in such a way that packets reached their intended destinations. It's also likely that human configuration would be more error-prone and much slower to respond to changes in the network topology than a routing protocol. We're thus fortunate that all networks have both a forwarding *and* a routing function!

## Control Plane: The SDN Approach

The approach to implementing routing functionality shown in Figure 4.2—with each router having a routing component that communicates with the routing component of other routers—has been the traditional approach adopted by routing vendors in their products, at least until recently. Our observation that humans could manually configure forwarding tables does suggest, however, that there may be other ways for control-plane functionality to determine the contents of the data-plane forwarding tables.

Figure 4.3 shows an alternative approach in which a physically separate, remote controller computes and distributes the forwarding tables to be used by each and every router. Note that the data plane components of Figures 4.2 and 4.3 are identical. In Figure 4.3; however, control-plane routing functionality is separated from the



**Figure 4.3** ♦ A remote controller determines and distributes values in forwarding tables

physical router—the routing device performs forwarding only, while the remote controller computes and distributes forwarding tables. The remote controller might be implemented in a remote data center with high reliability and redundancy, and might be managed by the ISP or some third party. How might the routers and the remote controller communicate? By exchanging messages containing forwarding tables and other pieces of routing information. The control-plane approach shown in Figure 4.3 is at the heart of **software-defined networking (SDN)**, where the network is “software-defined” because the controller that computes forwarding tables and interacts with routers is implemented in software. Increasingly, these software implementations are also open, that is, similar to Linux OS code, the code is publically available, allowing ISPs (and networking researchers and students!) to innovate and propose changes to the software that controls network-layer functionality. We will cover the SDN control plane in Section 5.5.

### 4.1.2 Network Service Model

Before delving into the network layer’s data plane, let’s wrap up our introduction by taking the broader view and consider the different types of service that might be offered by the network layer. When the transport layer at a sending host transmits a packet into the network (that is, passes it down to the network layer at the sending host), can the transport layer rely on the network layer to deliver the packet to the destination? When multiple packets are sent, will they be delivered to the transport layer in the receiving host in the order in which they were sent? Will the amount of time between the sending of two sequential packet transmissions be the same as the amount of time between their reception? Will the network provide any feedback about congestion in the network? The answers to these questions and others are determined by the service model provided by the network layer. The **network service model** defines the characteristics of end-to-end delivery of packets between sending and receiving hosts.

Let’s now consider some possible services that the network layer could provide. These services could include:

- *Guaranteed delivery.* This service guarantees that a packet sent by a source host will eventually arrive at the destination host.
- *Guaranteed delivery with bounded delay.* This service not only guarantees delivery of the packet, but delivery within a specified host-to-host delay bound (for example, within 100 msec).
- *In-order packet delivery.* This service guarantees that packets arrive at the destination in the order that they were sent.
- *Guaranteed minimal bandwidth.* This network-layer service emulates the behavior of a transmission link of a specified bit rate (for example, 1 Mbps) between sending and receiving hosts. As long as the sending host transmits bits (as part

of packets) at a rate below the specified bit rate, then all packets are eventually delivered to the destination host.

- *Security.* The network layer could encrypt all datagrams at the source and decrypt them at the destination, thereby providing confidentiality to all transport-layer segments.

This is only a partial list of services that a network layer could provide—there are countless variations possible.

The Internet’s network layer provides a single service, known as **best-effort service**. With best-effort service, packets are neither guaranteed to be received in the order in which they were sent, nor is their eventual delivery even guaranteed. There is no guarantee on the end-to-end delay nor is there a minimal bandwidth guarantee. It might appear that *best-effort service* is a euphemism for *no service at all*—a network that delivered *no* packets to the destination would satisfy the definition of best-effort delivery service! Other network architectures have defined and implemented service models that go beyond the Internet’s best-effort service. For example, the ATM network architecture [Black 1995] provides for guaranteed in-order delay, bounded delay, and guaranteed minimal bandwidth. There have also been proposed service model extensions to the Internet architecture; for example, the Intserv architecture [RFC 1633] aims to provide end-end delay guarantees and congestion-free communication. Interestingly, in spite of these well-developed alternatives, the Internet’s basic best-effort service model combined with adequate bandwidth provisioning and bandwidth-adaptive application-level protocols such as the DASH protocol we encountered in Section 2.6.2 have arguably proven to be more than “good enough” to enable an amazing range of applications, including streaming video services such as Netflix and video-over-IP, real-time conferencing applications such as Skype and Facetime.

## An Overview of Chapter 4

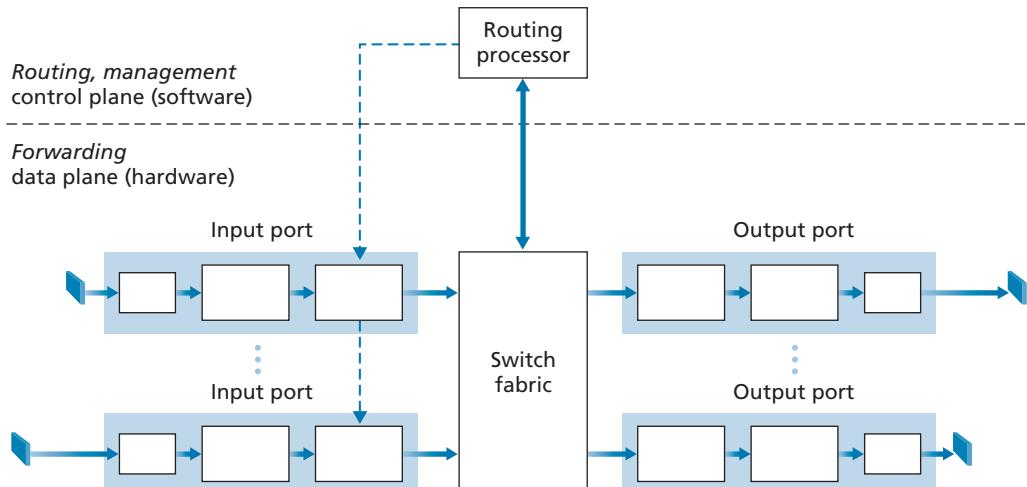
Having now provided an overview of the network layer, we’ll cover the data-plane component of the network layer in the following sections in this chapter. In Section 4.2, we’ll dive down into the internal hardware operations of a router, including input and output packet processing, the router’s internal switching mechanism, and packet queuing and scheduling. In Section 4.3, we’ll take a look at traditional IP forwarding, in which packets are forwarded to output ports based on their destination IP addresses. We’ll encounter IP addressing, the celebrated IPv4 and IPv6 protocols and more. In Section 4.4, we’ll cover more generalized forwarding, where packets may be forwarded to output ports based on a large number of header values (i.e., not only based on destination IP address). Packets may be blocked or duplicated at the router, or may have certain header field values rewritten—all under software control. This more generalized form of packet forwarding is a key component of a modern network data plane, including the data plane in software-defined networks (SDN). In Section 4.5, we’ll learn about “middleboxes” that can perform functions in addition to forwarding.

We mention here in passing that the terms *forwarding* and *switching* are often used interchangeably by computer-networking researchers and practitioners; we'll use both terms interchangeably in this textbook as well. While we're on the topic of terminology, it's also worth mentioning two other terms that are often used interchangeably, but that we will use more carefully. We'll reserve the term *packet switch* to mean a general packet-switching device that transfers a packet from input link interface to output link interface, according to values in a packet's header fields. Some packet switches, called **link-layer switches** (examined in Chapter 6), base their forwarding decision on values in the fields of the link-layer frame; switches are thus referred to as link-layer (layer 2) devices. Other packet switches, called **routers**, base their forwarding decision on header field values in the network-layer datagram. Routers are thus network-layer (layer 3) devices. (To fully appreciate this important distinction, you might want to review Section 1.5.2, where we discuss network-layer datagrams and link-layer frames and their relationship.) Since our focus in this chapter is on the network layer, we'll mostly use the term *router* in place of *packet switch*.

## 4.2 What's Inside a Router?

Now that we've overviewed the data and control planes within the network layer, the important distinction between forwarding and routing, and the services and functions of the network layer, let's turn our attention to its forwarding function—the actual transfer of packets from a router's incoming links to the appropriate outgoing links at that router.

A high-level view of a generic router architecture is shown in Figure 4.4. Four router components can be identified:



**Figure 4.4** ♦ Router architecture

- *Input ports.* An **input port** performs several key functions. It performs the physical layer function of terminating an incoming physical link at a router; this is shown in the leftmost box of an input port and the rightmost box of an output port in Figure 4.4. An input port also performs link-layer functions needed to interoperate with the link layer at the other side of the incoming link; this is represented by the middle boxes in the input and output ports. Perhaps most crucially, a lookup function is also performed at the input port; this will occur in the rightmost box of the input port. It is here that the forwarding table is consulted to determine the router output port to which an arriving packet will be forwarded via the switching fabric. Control packets (for example, packets carrying routing protocol information) are forwarded from an input port to the routing processor. Note that the term “port” here—referring to the physical input and output router interfaces—is distinctly different from the software ports associated with network applications and sockets discussed in Chapters 2 and 3. In practice, the number of ports supported by a router can range from a relatively small number in enterprise routers, to hundreds of 10 Gbps ports in a router at an ISP’s edge, where the number of incoming lines tends to be the greatest. The Juniper MX2020, edge router, for example, supports up to 800 100 Gbps Ethernet ports, with an overall router system capacity of 800 Tbps [Juniper MX 2020 2020].
- *Switching fabric.* The switching fabric connects the router’s input ports to its output ports. This switching fabric is completely contained within the router—a network inside of a network router!
- *Output ports.* An **output port** stores packets received from the switching fabric and transmits these packets on the outgoing link by performing the necessary link-layer and physical-layer functions. When a link is bidirectional (that is, carries traffic in both directions), an output port will typically be paired with the input port for that link on the same line card.
- *Routing processor.* The routing processor performs control-plane functions. In traditional routers, it executes the routing protocols (which we’ll study in Sections 5.3 and 5.4), maintains routing tables and attached link state information, and computes the forwarding table for the router. In SDN routers, the routing processor is responsible for communicating with the remote controller in order to (among other activities) receive forwarding table entries computed by the remote controller, and install these entries in the router’s input ports. The routing processor also performs the network management functions that we’ll study in Section 5.7.

A router’s input ports, output ports, and switching fabric are almost always implemented in hardware, as shown in Figure 4.4. To appreciate why a hardware implementation is needed, consider that with a 100 Gbps input link and a 64-byte IP datagram, the input port has only 5.12 ns to process the datagram before another datagram may arrive. If  $N$  ports are combined on a line card (as is often done in practice), the datagram-processing pipeline must operate  $N$  times faster—far too

fast for software implementation. Forwarding hardware can be implemented either using a router vendor's own hardware designs, or constructed using purchased merchant-silicon chips (for example, as sold by companies such as Intel and Broadcom).

While the data plane operates at the nanosecond time scale, a router's control functions—executing the routing protocols, responding to attached links that go up or down, communicating with the remote controller (in the SDN case) and performing management functions—operate at the millisecond or second timescale. These **control plane** functions are thus usually implemented in software and execute on the routing processor (typically a traditional CPU).

Before delving into the details of router internals, let's return to our analogy from the beginning of this chapter, where packet forwarding was compared to cars entering and leaving an interchange. Let's suppose that the interchange is a roundabout, and that as a car enters the roundabout, a bit of processing is required. Let's consider what information is required for this processing:

- *Destination-based forwarding.* Suppose the car stops at an entry station and indicates its final destination (not at the local roundabout, but the ultimate destination of its journey). An attendant at the entry station looks up the final destination, determines the roundabout exit that leads to that final destination, and tells the driver which roundabout exit to take.
- *Generalized forwarding.* The attendant could also determine the car's exit ramp on the basis of many other factors besides the destination. For example, the selected exit ramp might depend on the car's origin, for example the state that issued the car's license plate. Cars from a certain set of states might be directed to use one exit ramp (that leads to the destination via a slow road), while cars from other states might be directed to use a different exit ramp (that leads to the destination via super-highway). The same decision might be made based on the model, make and year of the car. Or a car not deemed roadworthy might be blocked and not be allowed to pass through the roundabout. In the case of generalized forwarding, any number of factors may contribute to the attendant's choice of the exit ramp for a given car.

Once the car enters the roundabout (which may be filled with other cars entering from other input roads and heading to other roundabout exits), it eventually leaves at the prescribed roundabout exit ramp, where it may encounter other cars leaving the roundabout at that exit.

We can easily recognize the principal router components in Figure 4.4 in this analogy—the entry road and entry station correspond to the input port (with a lookup function to determine to local outgoing port); the roundabout corresponds to the switch fabric; and the roundabout exit road corresponds to the output port. With this analogy, it's instructive to consider where bottlenecks might occur. What happens if cars arrive blazingly fast (for example, the roundabout is in Germany or Italy!) but the station attendant is slow? How fast must the attendant work to ensure there's no backup on an entry road? Even with a blazingly fast attendant, what happens if cars

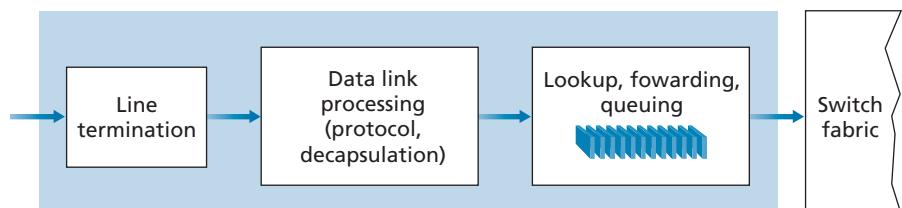
traverse the roundabout slowly—can backups still occur? And what happens if most of the cars entering at all of the roundabout’s entrance ramps all want to leave the roundabout at the same exit ramp—can backups occur at the exit ramp or elsewhere? How should the roundabout operate if we want to assign priorities to different cars, or block certain cars from entering the roundabout in the first place? These are all analogous to critical questions faced by router and switch designers.

In the following subsections, we’ll look at router functions in more detail. [Turner 1988; McKeown 1997a; Partridge 1998; Iyer 2008; Serpanos 2011; Zilberman 2019] provide a discussion of specific router architectures. For concreteness and simplicity, we’ll initially assume in this section that forwarding decisions are based only on the packet’s destination address, rather than on a generalized set of packet header fields. We will cover the case of more generalized packet forwarding in Section 4.4.

### 4.2.1 Input Port Processing and Destination-Based Forwarding

A more detailed view of input processing is shown in Figure 4.5. As just discussed, the input port’s line-termination function and link-layer processing implement the physical and link layers for that individual input link. The lookup performed in the input port is central to the router’s operation—it is here that the router uses the forwarding table to look up the output port to which an arriving packet will be forwarded via the switching fabric. The forwarding table is either computed and updated by the routing processor (using a routing protocol to interact with the routing processors in other network routers) or is received from a remote SDN controller. The forwarding table is copied from the routing processor to the line cards over a separate bus (e.g., a PCI bus) indicated by the dashed line from the routing processor to the input line cards in Figure 4.4. With such a shadow copy at each line card, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor on a per-packet basis and thus avoiding a centralized processing bottleneck.

Let’s now consider the “simplest” case that the output port to which an incoming packet is to be switched is based on the packet’s destination address. In the case of 32-bit IP addresses, a brute-force implementation of the forwarding table would have one entry for every possible destination address. Since there are more than 4 billion possible addresses, this option is totally out of the question.



**Figure 4.5** ♦ Input port processing

As an example of how this issue of scale can be handled, let's suppose that our router has four links, numbered 0 through 3, and that packets are to be forwarded to the link interfaces as follows:

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	0
11001000 00010111 00010111 11111111	
11001000 00010111 00011000 00000000 through	1
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through	2
11001000 00010111 00011111 11111111	
Otherwise	3

Clearly, for this example, it is not necessary to have 4 billion entries in the router's forwarding table. We could, for example, have the following forwarding table with just four entries:

Prefix	Link Interface
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
Otherwise	3

With this style of forwarding table, the router matches a **prefix** of the packet's destination address with the entries in the table; if there's a match, the router forwards the packet to a link associated with the match. For example, suppose the packet's destination address is 11001000 00010111 00010110 10100001; because the 21-bit prefix of this address matches the first entry in the table, the router forwards the packet to link interface 0. If a prefix doesn't match any of the first three entries, then the router forwards the packet to the default interface 3. Although this sounds simple enough, there's a very important subtlety here. You may have noticed that it is possible for a destination address to match more than one entry. For example, the first 24 bits of the address 11001000 00010111 00011000 10101010 match the second entry in the table, and the first 21 bits of the address match the third entry in the table. When there are multiple matches, the router uses the **longest prefix matching rule**; that is, it finds the longest matching entry in the table and forwards the packet to the link interface associated with the longest prefix match. We'll see exactly *why* this

longest prefix-matching rule is used when we study Internet addressing in more detail in Section 4.3.

Given the existence of a forwarding table, lookup is conceptually simple—hardware logic just searches through the forwarding table looking for the longest prefix match. But at Gigabit transmission rates, this lookup must be performed in nanoseconds (recall our earlier example of a 10 Gbps link and a 64-byte IP datagram). Thus, not only must lookup be performed in hardware, but techniques beyond a simple linear search through a large table are needed; surveys of fast lookup algorithms can be found in [Gupta 2001, Ruiz-Sanchez 2001]. Special attention must also be paid to memory access times, resulting in designs with embedded on-chip DRAM and faster SRAM (used as a DRAM cache) memories. In practice, Ternary Content Addressable Memories (TCAMs) are also often used for lookup [Yu 2004]. With a TCAM, a 32-bit IP address is presented to the memory, which returns the content of the forwarding table entry for that address in essentially constant time. The Cisco Catalyst 6500 and 7600 Series routers and switches can hold upwards of a million TCAM forwarding table entries [Cisco TCAM 2014].

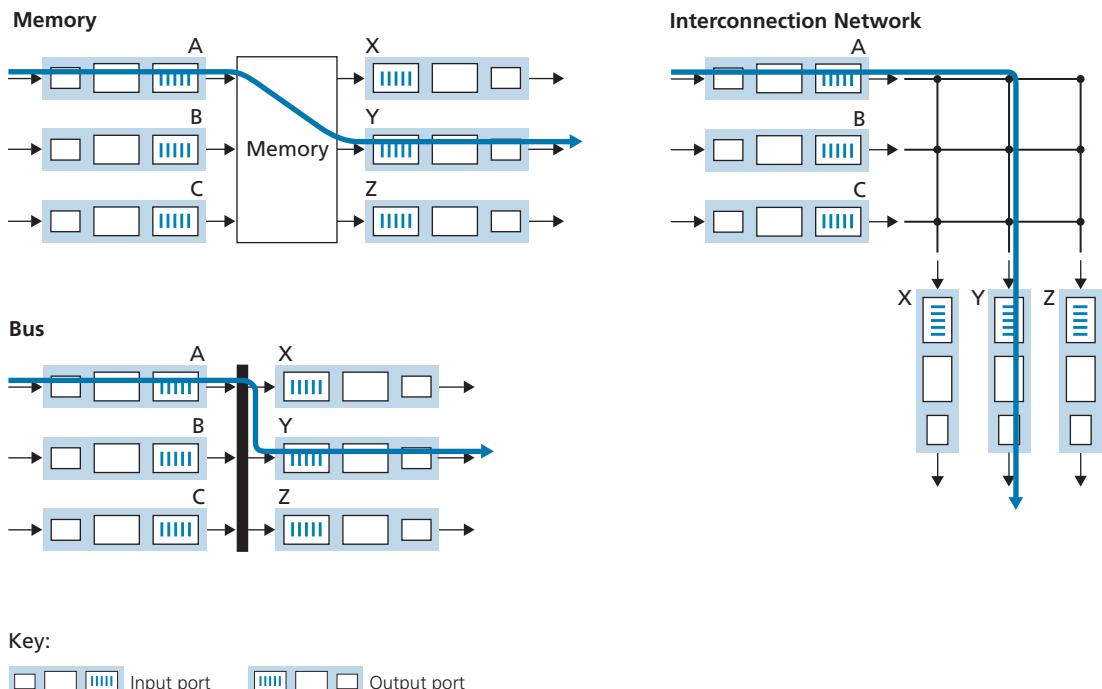
Once a packet’s output port has been determined via the lookup, the packet can be sent into the switching fabric. In some designs, a packet may be temporarily blocked from entering the switching fabric if packets from other input ports are currently using the fabric. A blocked packet will be queued at the input port and then scheduled to cross the fabric at a later point in time. We’ll take a closer look at the blocking, queuing, and scheduling of packets (at both input ports and output ports) shortly. Although “lookup” is arguably the most important action in input port processing, many other actions must be taken: (1) physical- and link-layer processing must occur, as discussed previously; (2) the packet’s version number, checksum and time-to-live field—all of which we’ll study in Section 4.3—must be checked and the latter two fields rewritten; and (3) counters used for network management (such as the number of IP datagrams received) must be updated.

Let’s close our discussion of input port processing by noting that the input port steps of looking up a destination IP address (“match”) and then sending the packet into the switching fabric to the specified output port (“action”) is a specific case of a more general “match plus action” abstraction that is performed in many networked devices, not just routers. In link-layer switches (covered in Chapter 6), link-layer destination addresses are looked up and several actions may be taken in addition to sending the frame into the switching fabric towards the output port. In firewalls (covered in Chapter 8)—devices that filter out selected incoming packets—an incoming packet whose header matches a given criteria (e.g., a combination of source/destination IP addresses and transport-layer port numbers) may be dropped (action). In a network address translator (NAT, covered in Section 4.3), an incoming packet whose transport-layer port number matches a given value will have its port number rewritten before forwarding (action). Indeed, the “match plus action” abstraction [Bosschart 2013] is both powerful and prevalent in network devices today, and is central to the notion of generalized forwarding that we’ll study in Section 4.4.

### 4.2.2 Switching

The switching fabric is at the very heart of a router, as it is through this fabric that the packets are actually switched (that is, forwarded) from an input port to an output port. Switching can be accomplished in a number of ways, as shown in Figure 4.6:

- *Switching via memory.* The simplest, earliest routers were traditional computers, with switching between input and output ports being done under direct control of the CPU (routing processor). Input and output ports functioned as traditional I/O devices in a traditional operating system. An input port with an arriving packet first signaled the routing processor via an interrupt. The packet was then copied from the input port into processor memory. The routing processor then extracted the destination address from the header, looked up the appropriate output port in the forwarding table, and copied the packet to the output port's buffers. In this scenario, if the memory bandwidth is such that a maximum of  $B$  packets per second can be written into, or read from, memory, then the overall forwarding throughput (the total rate at which packets are transferred from input ports to output ports) must be less than  $B/2$ . Note also that two packets cannot be forwarded



**Figure 4.6** ♦ Three switching techniques

at the same time, even if they have different destination ports, since only one memory read/write can be done at a time over the shared system bus.

Some modern routers switch via memory. A major difference from early routers, however, is that the lookup of the destination address and the storing of the packet into the appropriate memory location are performed by processing on the input line cards. In some ways, routers that switch via memory look very much like shared-memory multiprocessors, with the processing on a line card switching (writing) packets into the memory of the appropriate output port. Cisco's Catalyst 8500 series switches [Cisco 8500 2020] internally switches packets via a shared memory.

- *Switching via a bus.* In this approach, an input port transfers a packet directly to the output port over a shared bus, without intervention by the routing processor. This is typically done by having the input port pre-pend a switch-internal label (header) to the packet indicating the local output port to which this packet is being transferred and transmitting the packet onto the bus. All output ports receive the packet, but only the port that matches the label will keep the packet. The label is then removed at the output port, as this label is only used within the switch to cross the bus. If multiple packets arrive to the router at the same time, each at a different input port, all but one must wait since only one packet can cross the bus at a time. Because every packet must cross the single bus, the switching speed of the router is limited to the bus speed; in our roundabout analogy, this is as if the roundabout could only contain one car at a time. Nonetheless, switching via a bus is often sufficient for routers that operate in small local area and enterprise networks. The Cisco 6500 router [Cisco 6500 2020] internally switches packets over a 32-Gbps-backplane bus.
- *Switching via an interconnection network.* One way to overcome the bandwidth limitation of a single, shared bus is to use a more sophisticated interconnection network, such as those that have been used in the past to interconnect processors in a multiprocessor computer architecture. A crossbar switch is an interconnection network consisting of  $2N$  buses that connect  $N$  input ports to  $N$  output ports, as shown in Figure 4.6. Each vertical bus intersects each horizontal bus at a crosspoint, which can be opened or closed at any time by the switch fabric controller (whose logic is part of the switching fabric itself). When a packet arrives from port A and needs to be forwarded to port Y, the switch controller closes the crosspoint at the intersection of busses A and Y, and port A then sends the packet onto its bus, which is picked up (only) by bus Y. Note that a packet from port B can be forwarded to port X at the same time, since the A-to-Y and B-to-X packets use different input and output busses. Thus, unlike the previous two switching approaches, crossbar switches are capable of forwarding multiple packets in parallel. A crossbar switch is **non-blocking**—a packet being forwarded to an output port will not be blocked from reaching that output port as long as no other packet is currently being forwarded to that output port. However, if two packets from two different input ports are destined to that same output port, then one will have to wait at the input, since only one packet can be sent over any given bus at a time. Cisco 12000 series

switches [Cisco 12000 2020] use a crossbar switching network; the Cisco 7600 series can be configured to use either a bus or crossbar switch [Cisco 7600 2020].

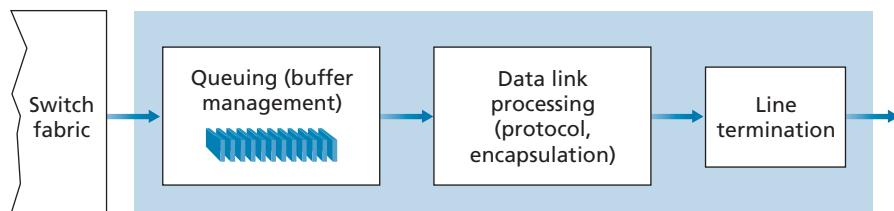
More sophisticated interconnection networks use multiple stages of switching elements to allow packets from different input ports to proceed towards the same output port at the same time through the multi-stage switching fabric. See [Tobagi 1990] for a survey of switch architectures. The Cisco CRS employs a three-stage non-blocking switching strategy. A router's switching capacity can also be scaled by running multiple switching fabrics in parallel. In this approach, input ports and output ports are connected to  $N$  switching fabrics that operate in parallel. An input port breaks a packet into  $K$  smaller chunks, and sends ("sprays") the chunks through  $K$  of these  $N$  switching fabrics to the selected output port, which reassembles the  $K$  chunks back into the original packet.

### 4.2.3 Output Port Processing

Output port processing, shown in Figure 4.7, takes packets that have been stored in the output port's memory and transmits them over the output link. This includes selecting (i.e., scheduling) and de-queuing packets for transmission, and performing the needed link-layer and physical-layer transmission functions.

### 4.2.4 Where Does Queuing Occur?

If we consider input and output port functionality and the configurations shown in Figure 4.6, it's clear that packet queues may form at both the input ports *and* the output ports, just as we identified cases where cars may wait at the inputs and outputs of the traffic intersection in our roundabout analogy. The location and extent of queuing (either at the input port queues or the output port queues) will depend on the traffic load, the relative speed of the switching fabric, and the line speed. Let's now consider these queues in a bit more detail, since as these queues grow large, the router's memory can eventually be exhausted and **packet loss** will occur when no memory is available to store arriving packets. Recall that in our earlier discussions, we said that packets were "lost within the network" or "dropped at a router." *It is here, at these queues within a router, where such packets are actually dropped and lost.*



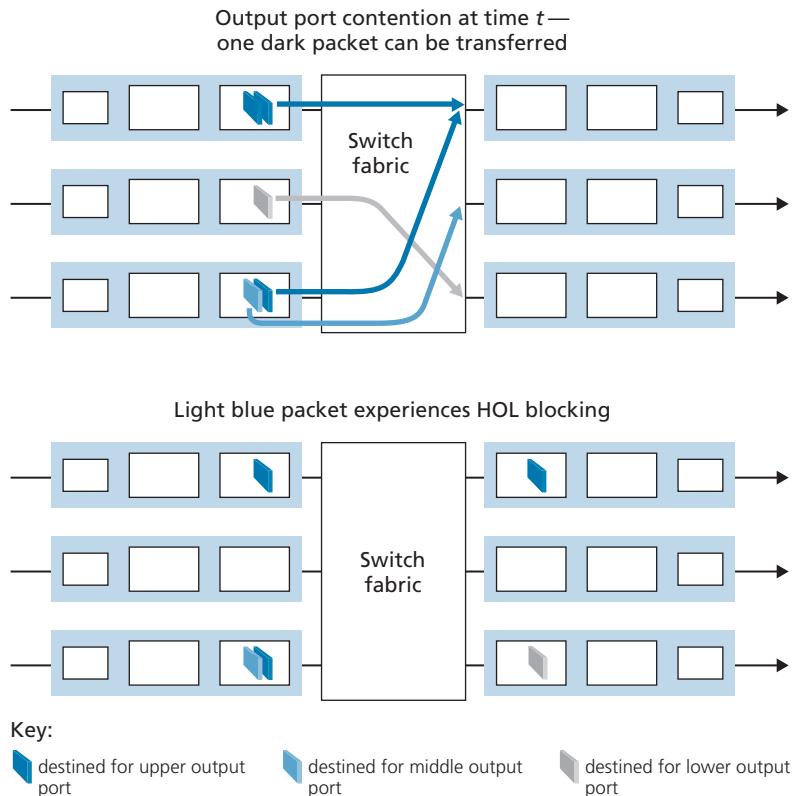
**Figure 4.7** ♦ Output port processing

Suppose that the input and output line speeds (transmission rates) all have an identical transmission rate of  $R_{\text{line}}$  packets per second, and that there are  $N$  input ports and  $N$  output ports. To further simplify the discussion, let's assume that all packets have the same fixed length, and that packets arrive to input ports in a synchronous manner. That is, the time to send a packet on any link is equal to the time to receive a packet on any link, and during such an interval of time, either zero or one packets can arrive on an input link. Define the switching fabric transfer rate  $R_{\text{switch}}$  as the rate at which packets can be moved from input port to output port. If  $R_{\text{switch}}$  is  $N$  times faster than  $R_{\text{line}}$ , then only negligible queuing will occur at the input ports. This is because even in the worst case, where all  $N$  input lines are receiving packets, and all packets are to be forwarded to the same output port, each batch of  $N$  packets (one packet per input port) can be cleared through the switch fabric before the next batch arrives.

### Input Queuing

But what happens if the switch fabric is not fast enough (relative to the input line speeds) to transfer *all* arriving packets through the fabric without delay? In this case, packet queuing can also occur at the input ports, as packets must join input port queues to wait their turn to be transferred through the switching fabric to the output port. To illustrate an important consequence of this queuing, consider a crossbar switching fabric and suppose that (1) all link speeds are identical, (2) that one packet can be transferred from any one input port to a given output port in the same amount of time it takes for a packet to be received on an input link, and (3) packets are moved from a given input queue to their desired output queue in an FCFS manner. Multiple packets can be transferred in parallel, as long as their output ports are different. However, if two packets at the front of two input queues are destined for the same output queue, then one of the packets will be blocked and must wait at the input queue—the switching fabric can transfer only one packet to a given output port at a time.

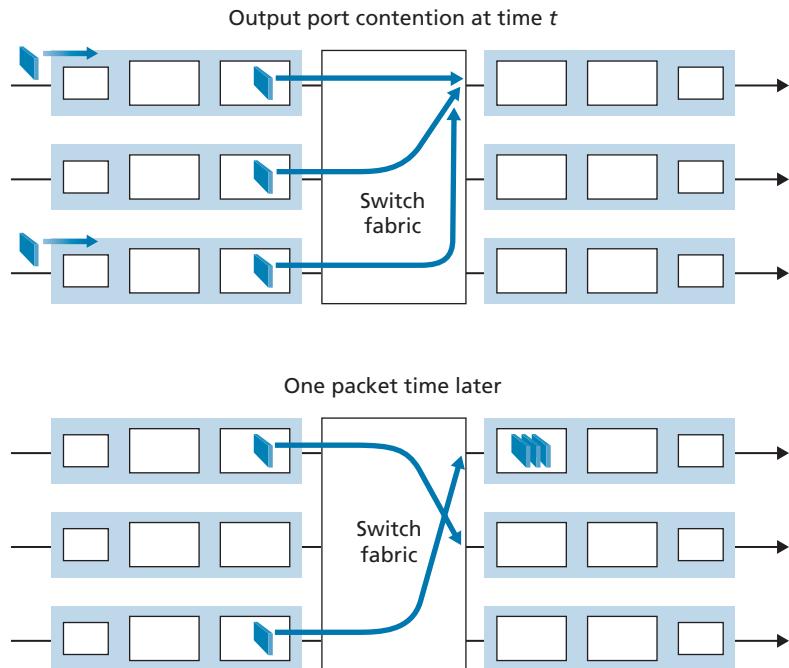
Figure 4.8 shows an example in which two packets (darkly shaded) at the front of their input queues are destined for the same upper-right output port. Suppose that the switch fabric chooses to transfer the packet from the front of the upper-left queue. In this case, the darkly shaded packet in the lower-left queue must wait. But not only must this darkly shaded packet wait, so too must the lightly shaded packet that is queued behind that packet in the lower-left queue, even though there is *no* contention for the middle-right output port (the destination for the lightly shaded packet). This phenomenon is known as **head-of-the-line (HOL) blocking** in an input-queued switch—a queued packet in an input queue must wait for transfer through the fabric (even though its output port is free) because it is blocked by another packet at the head of the line. [Karol 1987] shows that due to HOL blocking, the input queue will grow to unbounded length (informally, this is equivalent to saying that significant packet loss will occur) under certain assumptions as soon as the packet arrival rate on the input links reaches only 58 percent of their capacity. A number of solutions to HOL blocking are discussed in [McKeown 1997].



**Figure 4.8** ♦ HOL blocking at and input-queued switch

## Output Queuing

Let's next consider whether queuing can occur at a switch's output ports. Suppose that  $R_{\text{switch}}$  is again  $N$  times faster than  $R_{\text{line}}$  and that packets arriving at each of the  $N$  input ports are destined to the same output port. In this case, in the time it takes to send a single packet onto the outgoing link,  $N$  new packets will arrive at this output port (one from each of the  $N$  input ports). Since the output port can transmit only a single packet in a unit of time (the packet transmission time), the  $N$  arriving packets will have to queue (wait) for transmission over the outgoing link. Then  $N$  more packets can possibly arrive in the time it takes to transmit just one of the  $N$  packets that had just previously been queued. And so on. Thus, packet queues can form at the output ports even when the switching fabric is  $N$  times faster than the port line speeds. Eventually, the number of queued packets can grow large enough to exhaust available memory at the output port.



**Figure 4.9** ♦ Output port queuing

When there is not enough memory to buffer an incoming packet, a decision must be made to either drop the arriving packet (a policy known as **drop-tail**) or remove one or more already-queued packets to make room for the newly arrived packet. In some cases, it may be advantageous to drop (or mark the header of) a packet *before* the buffer is full in order to provide a congestion signal to the sender. This marking could be done using the Explicit Congestion Notification bits that we studied in Section 3.7.2. A number of proactive packet-dropping and -marking policies (which collectively have become known as **active queue management (AQM)** algorithms) have been proposed and analyzed [Labrador 1999, Hollot 2002]. One of the most widely studied and implemented AQM algorithms is the **Random Early Detection (RED)** algorithm [Christiansen 2001]. More recent AQM policies include PIE (the Proportional Integral controller Enhanced [RFC 8033]), and CoDel [Nichols 2012].

Output port queuing is illustrated in Figure 4.9. At time  $t$ , a packet has arrived at each of the incoming input ports, each destined for the uppermost outgoing port. Assuming identical line speeds and a switch operating at three times the line speed, one time unit later (that is, in the time needed to receive or send a packet), all three original packets have been transferred to the outgoing port and are queued awaiting transmission. In the next time unit, one of these three packets will have been transmitted over the outgoing link. In our example, two *new* packets have arrived at the incoming side of the

switch; one of these packets is destined for this uppermost output port. A consequence of such queuing is that a **packet scheduler** at the output port must choose one packet, among those queued, for transmission—a topic we'll cover in the following section.

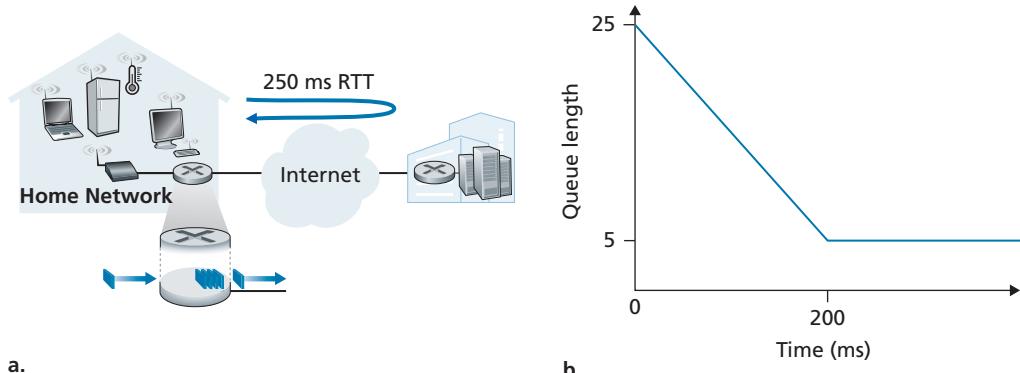
### How Much Buffering Is “Enough?”

Our study above has shown how a packet queue forms when bursts of packets arrive at a router's input or (more likely) output port, and the packet arrival rate temporarily exceeds the rate at which packets can be forwarded. The longer the amount of time that this mismatch persists, the longer the queue will grow, until eventually a port's buffers become full and packets are dropped. One natural question is how *much* buffering should be provisioned at a port. It turns out the answer to this question is much more complicated than one might imagine and can teach us quite a bit about the subtle interaction among congestion-aware senders at the network's edge and the network core!

For many years, the rule of thumb [RFC 3439] for buffer sizing was that the amount of buffering (**B**) should be equal to an average round-trip time (**RTT**, say 250 msec) times the link capacity (**C**). Thus, a 10-Gbps link with an RTT of 250 msec would need an amount of buffering equal to  $B = RTT \cdot C = 2.5$  Gbits of buffers. This result was based on an analysis of the queuing dynamics of a relatively small number of TCP flows [Villamizar 1994]. More recent theoretical and experimental efforts [Appenzeller 2004], however, suggest that when a large number of *independent* TCP flows (**N**) pass through a link, the amount of buffering needed is  $B = RTT \cdot C / \sqrt{N}$ . In core networks, where a large number of TCP flows typically pass through large backbone router links, the value of **N** can be large, with the decrease in needed buffer size becoming quite significant. [Appenzeller 2004; Wischik 2005; Beheshti 2008] provide very readable discussions of the buffer-sizing problem from a theoretical, implementation, and operational standpoint.

It's tempting to think that more buffering *must* be better—larger buffers would allow a router to absorb larger fluctuations in the packet arrival rate, thereby decreasing the router's packet loss rate. But larger buffers also mean potentially longer queuing delays. For gamers and for interactive teleconferencing users, tens of milliseconds count. Increasing the amount of per-hop buffer by a factor of 10 to decrease packet loss could increase the end-end delay by a factor of 10! Increased RTTs also make TCP senders less responsive and slower to respond to incipient congestion and/or packet loss. These delay-based considerations show that buffering is a double-edged sword—buffering can be used to absorb short-term statistical fluctuations in traffic but can also lead to increased delay and the attendant concerns. Buffering is a bit like salt—just the right amount of salt makes food better, but too much makes it inedible!

In the discussion above, we've implicitly assumed that many independent senders are competing for bandwidth and buffers at a congested link. While this is probably an excellent assumption for routers within the network core, at the network edge



**Figure 4.10** ♦ Bufferbloat: persistent queues

this may not hold. Figure 4.10(a) shows a home router sending TCP segments to a remote game server. Following [Nichols 2012], suppose that it takes 20 ms to transmit a packet (containing a gamer's TCP segment), that there are negligible queuing delays elsewhere on the path to the game server, and that the RTT is 200 ms. As shown in Figure 4.10(b), suppose that at time  $t = 0$ , a burst of 25 packets arrives to the queue. One of these queued packets is then transmitted once every 20 ms, so that at  $t = 200$  msec, the first ACK arrives, just as the 21st packet is being transmitted. This ACK arrival causes the TCP sender to send another packet, which is queued at the outgoing link of the home router. At  $t = 220$ , the next ACK arrives, and another TCP segment is released by the gamer and is queued, as the 22nd packet is being transmitted, and so on. You should convince yourself that in this scenario, ACK clocking results in a new packet arriving at the queue every time a queued packet is sent, resulting in queue size at the home router's outgoing link that is *always* five packets! That is, the end-end-pipe is full (delivering packets to the destination at the path bottleneck rate of one packet every 20 ms), but the amount of queuing delay is constant and *persistent*. As a result, the gamer is unhappy with the delay, and the parent (who even knows wireshark!) is confused because he or she doesn't understand why delays are persistent and excessively long, even when there is no other traffic on the home network.

This scenario above of long delay due to persistent buffering is known as **bufferbloat** and illustrates that not only is throughput important, but also minimal delay is important as well [Kleinrock 2018], and that the interaction among senders at the network edge and queues within the network can indeed be complex and subtle. The DOCSIS 3.1 standard for cable networks that we will study in Chapter 6, recently added a specific AQM mechanism [RFC 8033, RFC 8034] to combat bufferbloat, while preserving bulk throughput performance.

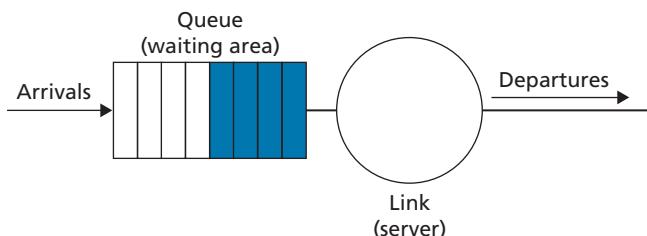
## 4.2.5 Packet Scheduling

Let's now return to the question of determining the order in which queued packets are transmitted over an outgoing link. Since you yourself have undoubtedly had to wait in long lines on many occasions and observed how waiting customers are served, you're no doubt familiar with many of the queuing disciplines commonly used in routers. There is first-come-first-served (FCFS, also known as first-in-first-out, FIFO). The British are famous for patient and orderly FCFS queuing at bus stops and in the marketplace ("Oh, are you queuing?"). Other countries operate on a priority basis, with one class of waiting customers given priority service over other waiting customers. There is also round-robin queuing, where customers are again divided into classes (as in priority queuing) but each class of customer is given service in turn.

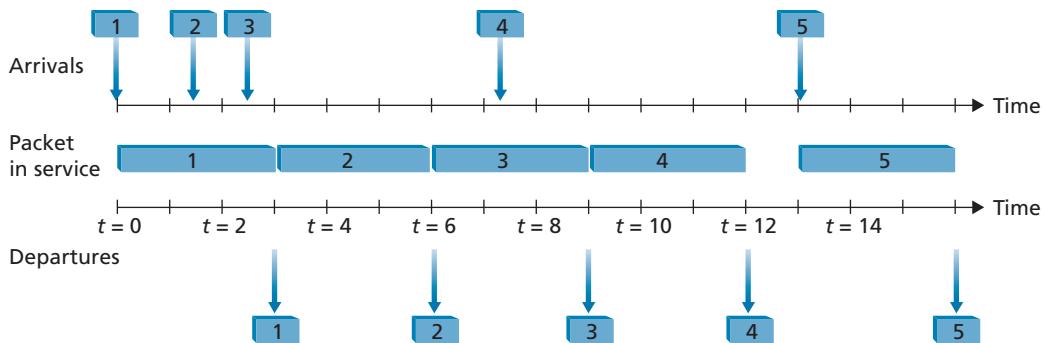
### First-in-First-Out (FIFO)

Figure 4.11 shows the queuing model abstraction for the FIFO link-scheduling discipline. Packets arriving at the link output queue wait for transmission if the link is currently busy transmitting another packet. If there is not sufficient buffering space to hold the arriving packet, the queue's packet-dropping policy then determines whether the packet will be dropped (lost) or whether other packets will be removed from the queue to make space for the arriving packet, as discussed above. In our discussion below, we'll ignore packet discard. When a packet is completely transmitted over the outgoing link (that is, receives service) it is removed from the queue.

The FIFO (also known as first-come-first-served, or FCFS) scheduling discipline selects packets for link transmission in the same order in which they arrived at the output link queue. We're all familiar with FIFO queuing from service centers, where arriving customers join the back of the single waiting line, remain in order, and are then served when they reach the front of the line. Figure 4.12 shows the FIFO queue in operation. Packet arrivals are indicated by numbered arrows above the upper timeline, with the number indicating the order in which the packet arrived. Individual packet departures are shown below the lower timeline. The time that a packet spends in service (being transmitted) is indicated by the shaded rectangle between the two timelines. In



**Figure 4.11** ♦ FIFO queuing abstraction

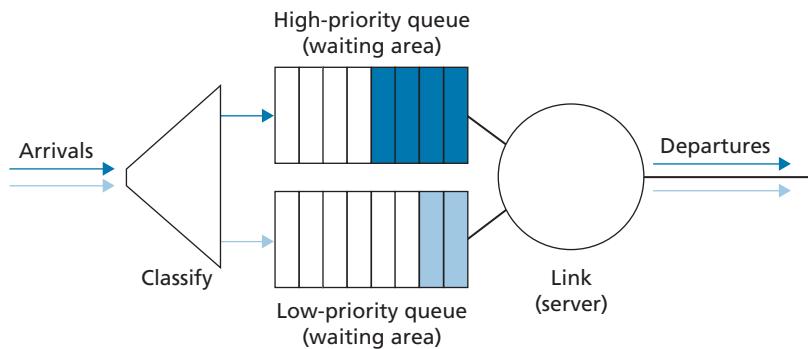


**Figure 4.12** ♦ The FIFO queue in operation

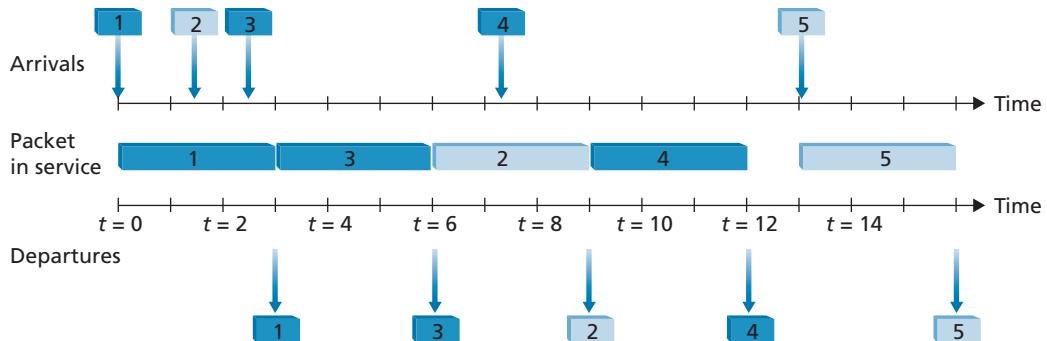
our examples here, let's assume that each packet takes three units of time to be transmitted. Under the FIFO discipline, packets leave in the same order in which they arrived. Note that after the departure of packet 4, the link remains idle (since packets 1 through 4 have been transmitted and removed from the queue) until the arrival of packet 5.

### Priority Queuing

Under priority queuing, packets arriving at the output link are classified into priority classes upon arrival at the queue, as shown in Figure 4.13. In practice, a network operator may configure a queue so that packets carrying network management information (for example, as indicated by the source or destination TCP/UDP port number) receive priority over user traffic; additionally, real-time voice-over-IP packets might receive priority over non-real-time traffic such e-mail packets. Each priority class typically has its own queue. When choosing a packet to transmit, the priority



**Figure 4.13** ♦ The priority queuing model



**Figure 4.14** ♦ The priority queue in operation

queuing discipline will transmit a packet from the highest priority class that has a nonempty queue (that is, has packets waiting for transmission). The choice among packets in the same priority class is typically done in a FIFO manner.

Figure 4.14 illustrates the operation of a priority queue with two priority classes. Packets 1, 3, and 4 belong to the high-priority class, and packets 2 and 5 belong to the low-priority class. Packet 1 arrives and, finding the link idle, begins transmission. During the transmission of packet 1, packets 2 and 3 arrive and are queued in the low- and high-priority queues, respectively. After the transmission of packet 1, packet 3 (a high-priority packet) is selected for transmission over packet 2 (which, even though it arrived earlier, is a low-priority packet). At the end of the transmission of packet 3, packet 2 then begins transmission. Packet 4 (a high-priority packet) arrives during the transmission of packet 2 (a low-priority packet). Under a **non-preemptive priority queuing** discipline, the transmission of a packet is not interrupted once it



## PRINCIPLES IN PRACTICE

### NET NEUTRALITY

We've seen that packet scheduling mechanisms (e.g., priority traffic scheduling disciplines such as strict priority, and WFQ) can be used to provide different levels of service to different "classes" of traffic. The definition of what precisely constitutes a "class" of traffic is up to an ISP to decide, but could be potentially based on any set of fields in the IP datagram header. For example, the port field in the IP datagram header could be used to classify datagrams according to the "well-known service" associated with that port: SNMP network management datagram (port 161) might be assigned to a higher priority class than an IMAP e-mail protocol (ports 143, or 993) datagram and therefore receive better service. An ISP could also potentially use a datagram's source IP address to provide priority to datagrams being sent by certain companies (who have presumably paid the ISP for this privilege) over datagrams being sent from other companies (who have not paid); an ISP

could even block traffic with a source IP address in a given company, or country. There are many *mechanisms* that would allow an ISP to provide different levels of service to different classes of traffic. The real question is what *policies* and *laws* determine what an ISP can actually do. Of course, these laws will vary by country; see [Smithsonian 2017] for a brief survey. Here, we'll briefly consider US policy on what has come to be known as "net neutrality."

The term "net neutrality" doesn't have a precise definition, but the March 2015 *Order on Protecting and Promoting an Open Internet* [FCC 2015] by the US Federal Communications Commission provides three "clear, bright line" rules that are now often associated with net neutrality:

- **"No Blocking.** . . . A person engaged in the provision of broadband Internet access service, . . . shall not block lawful content, applications, services, or non-harmful devices, subject to reasonable network management."
- **"No Throttling.** . . . A person engaged in the provision of broadband Internet access service, . . . shall not impair or degrade lawful Internet traffic on the basis of Internet content, application, or service, or use of a non-harmful device, subject to reasonable network management."
- **"No Paid Prioritization.** . . . A person engaged in the provision of broadband Internet access service, . . . shall not engage in paid prioritization. "Paid prioritization" refers to the management of a broadband provider's network to directly or indirectly favor some traffic over other traffic, including through use of techniques such as traffic shaping, prioritization, resource reservation, or other forms of preferential traffic management, . . ."

Quite interestingly, before the Order, ISP behaviors violating the first two of these rules had been observed [Faulhaber 2012]. In 2005, an ISP in North Carolina agreed to stop its practice of blocking its customers from using Vonage, a voice-over-IP service that competed with its own telephone service. In 2007, Comcast was judged to be interfering with BitTorrent P2P traffic by internally creating and sending TCP RST packets to BitTorrent senders and receivers, which caused them to close their BitTorrent connection [FCC 2008].

Both sides of the net neutrality debate have been argued strenuously, mostly focused on the extent to which net neutrality provides benefits to customers, while at the same time promoting innovation. See [Peha 2006, Faulhaber 2012, Economides 2017, Madhyastha 2017].

The 2015 FCC *Order on Protecting and Promoting an Open Internet*, which banned ISPs from blocking, throttling, or providing paid prioritizing, was superseded by the 2017 *FCC Restoring Internet Freedom Order*, [FCC 2017] which rolled back these prohibitions and focused instead on ISP transparency. With so much interest and so many changes, it's probably safe to say we aren't close to having seen the final chapter written on net neutrality in the United States, or elsewhere.

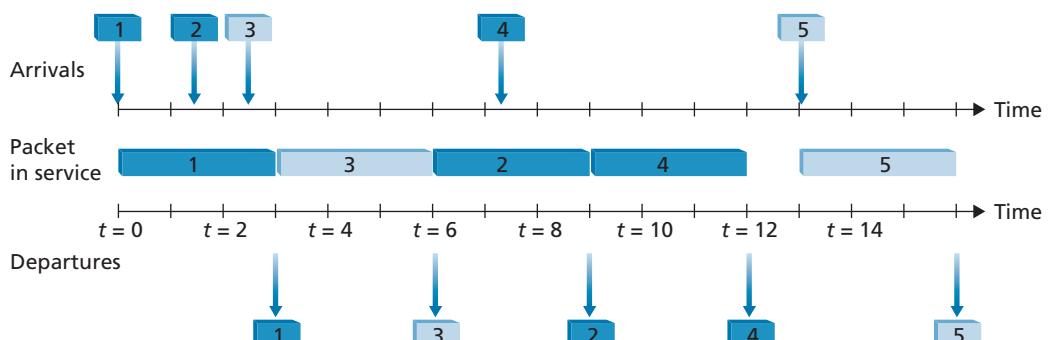
has begun. In this case, packet 4 queues for transmission and begins being transmitted after the transmission of packet 2 is completed.

### Round Robin and Weighted Fair Queuing (WFQ)

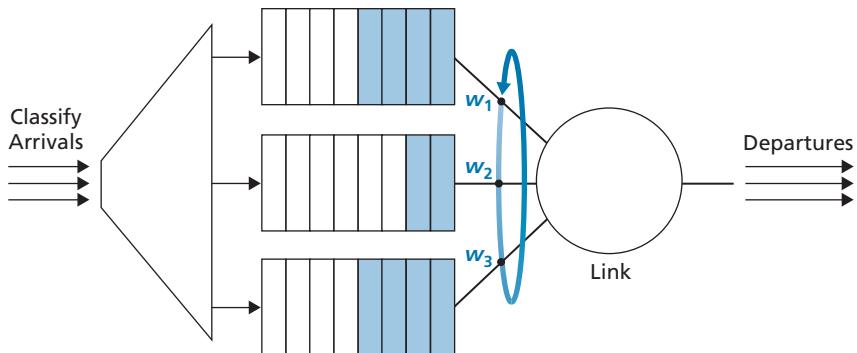
Under the round robin queuing discipline, packets are sorted into classes as with priority queuing. However, rather than there being a strict service priority among classes, a round robin scheduler alternates service among the classes. In the simplest form of round robin scheduling, a class 1 packet is transmitted, followed by a class 2 packet, followed by a class 1 packet, followed by a class 2 packet, and so on. A so-called **work-conserving queuing** discipline will never allow the link to remain idle whenever there are packets (of any class) queued for transmission. A work-conserving round robin discipline that looks for a packet of a given class but finds none will immediately check the next class in the round robin sequence.

Figure 4.15 illustrates the operation of a two-class round robin queue. In this example, packets 1, 2, and 4 belong to class 1, and packets 3 and 5 belong to the second class. Packet 1 begins transmission immediately upon arrival at the output queue. Packets 2 and 3 arrive during the transmission of packet 1 and thus queue for transmission. After the transmission of packet 1, the link scheduler looks for a class 2 packet and thus transmits packet 3. After the transmission of packet 3, the scheduler looks for a class 1 packet and thus transmits packet 2. After the transmission of packet 2, packet 4 is the only queued packet; it is thus transmitted immediately after packet 2.

A generalized form of round robin queuing that has been widely implemented in routers is the so-called **weighted fair queuing (WFQ) discipline** [Demers 1990; Parekh 1993]. WFQ is illustrated in Figure 4.16. Here, arriving packets are classified and queued in the appropriate per-class waiting area. As in round robin scheduling, a WFQ scheduler will serve classes in a circular manner—first serving class 1, then serving class 2, then serving class 3, and then (assuming there are three classes) repeating the service pattern. WFQ is also a work-conserving queuing discipline and



**Figure 4.15** ♦ The two-class robin queue in operation



**Figure 4.16** • Weighted fair queuing

thus will immediately move on to the next class in the service sequence when it finds an empty class queue.

WFQ differs from round robin in that each class may receive a differential amount of service in any interval of time. Specifically, each class,  $i$ , is assigned a weight,  $w_i$ . Under WFQ, during any interval of time during which there are class  $i$  packets to send, class  $i$  will then be guaranteed to receive a fraction of service equal to  $w_i / (\sum w_j)$ , where the sum in the denominator is taken over all classes that also have packets queued for transmission. In the worst case, even if all classes have queued packets, class  $i$  will still be guaranteed to receive a fraction  $w_i / (\sum w_j)$  of the bandwidth, where in this worst case the sum in the denominator is over *all* classes. Thus, for a link with transmission rate  $R$ , class  $i$  will always achieve a throughput of at least  $R \cdot w_i / (\sum w_j)$ . Our description of WFQ has been idealized, as we have not considered the fact that packets are discrete and a packet's transmission will not be interrupted to begin transmission of another packet; [Demers 1990; Parekh 1993] discuss this packetization issue.

### 4.3 The Internet Protocol (IP): IPv4, Addressing, IPv6, and More

Our study of the network layer thus far in Chapter 4—the notion of the data and control plane component of the network layer, our distinction between forwarding and routing, the identification of various network service models, and our look inside a router—have often been without reference to any specific computer network architecture or protocol. In this section, we'll focus on key aspects of the network layer on today's Internet and the celebrated Internet Protocol (IP).

There are two versions of IP in use today. We'll first examine the widely deployed IP protocol version 4, which is usually referred to simply as IPv4 [RFC 791] in Section 4.3.1. We'll examine IP version 6 [RFC 2460; RFC 4291], which has