The Network Layer: Data Plane

4.1 Overview of Network Layer

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.

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.
- *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. Routing is implemented in the control plane of the network layer.

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, and this 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, and as we will see is often implemented in software.

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.

Control Plane: The Traditional

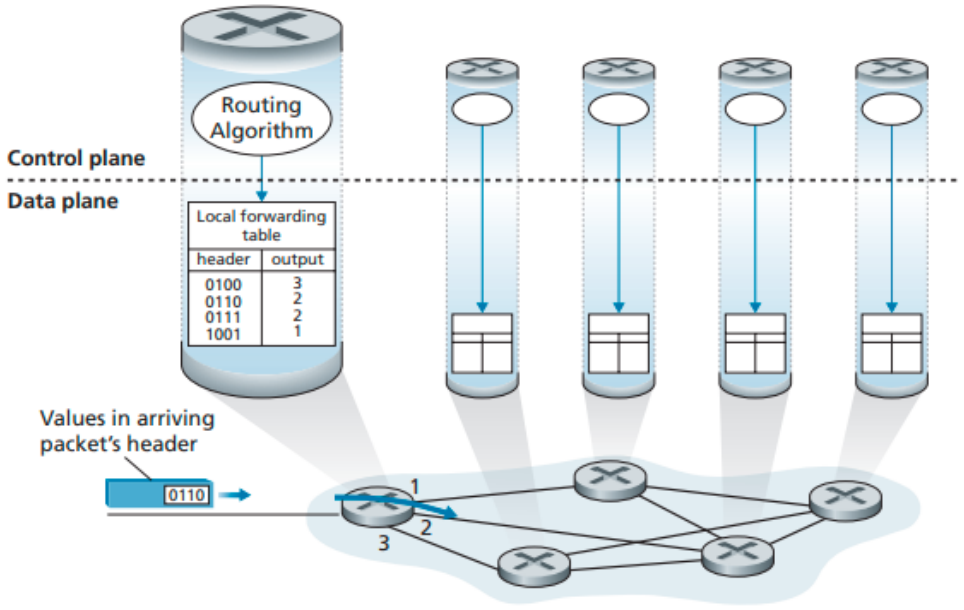

Figure 4.2 ♦ Routing algorithms determine values in forward tables

The routing algorithm determines the contents of the routers' forwarding tables. 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 messages containing routing information according to a routing protocol!

The distinct and different purposes of the forwarding and routing functions can be further illustrated by considering the hypothetical 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.

Control Plane: The SDN Approach

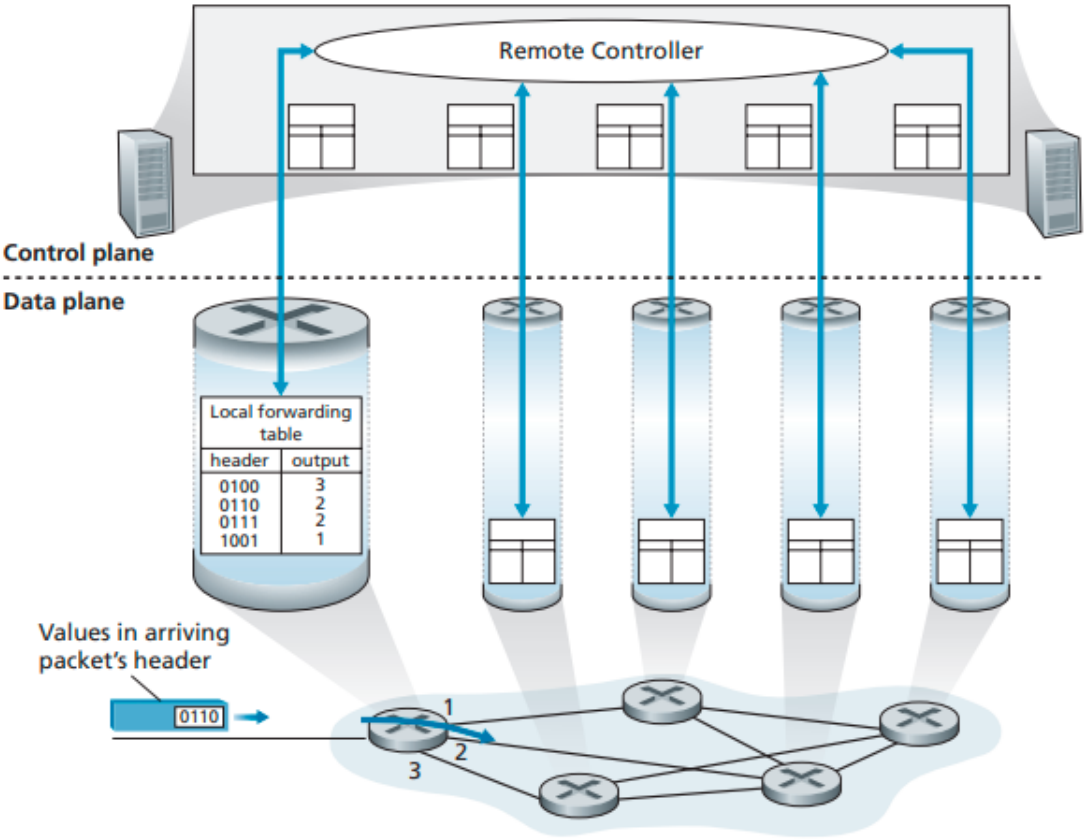

Figure 4.3 ♦ A remote controller determines and distributes values in forwarding tables

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. However, control-plane routing functionality is separated from the 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.

4.1.2 Network Service Model

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.
- 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 between sending and receiving hosts. As long as the sending host transmits bits 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.

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.

4.2 What’s Inside a Router?

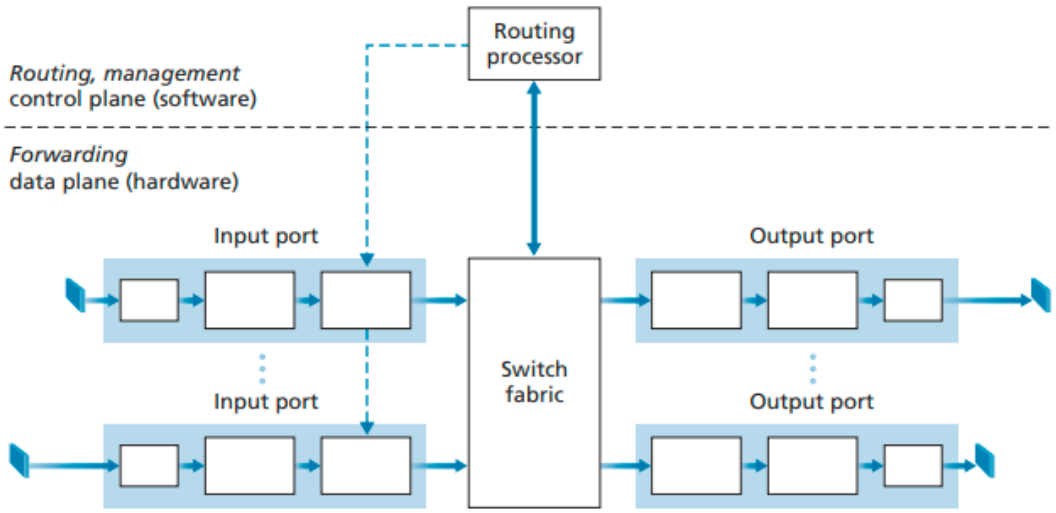

Figure 4.4 ♦ Router architecture

- Input ports.
- Switching fabric. The switching fabric connects the router's input ports to its output ports.
- 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-functions.
- Routing processor. The routing processor performs control-plane functions. 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.

A router's input ports, output ports, and switching fabric are almost always implemented in hardware.

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)

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. An attendant at the entry station looks up the final destination, determines the roundabout exit that lead 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.

We can easily recognize the principal router components 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; the roundabout exit corresponds to the output port.

4.2.1 Input Port Processing and Destination-Based Forwarding

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.

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. 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.

Once a packet's output 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. 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 must be checked and the latter two fields rewritten; and (3) counters used for network management must be updated.

In link-layer switches, 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—devices that filter out selected incoming packets—an incoming packet whose header matches a given criteria may be dropped. In a network address translator (NAT), an incoming packet whose transport-layer port number matches a given value will have its port number rewritten before forwarding.

4.2.2 Switching

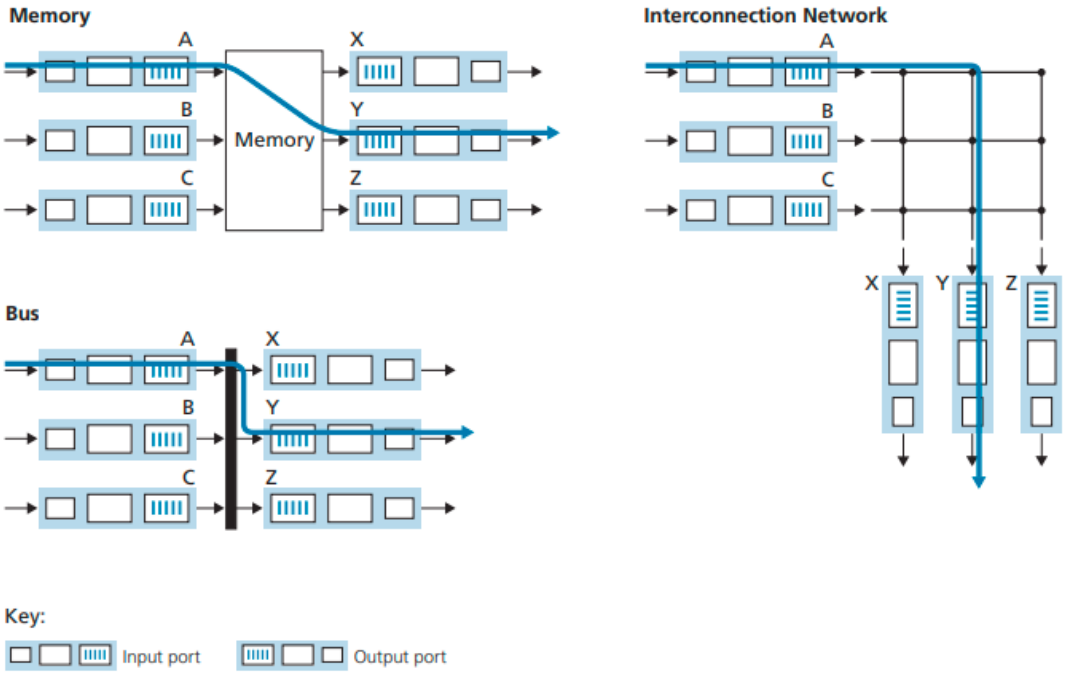

Figure 4.6 ♦ Three switching techniques

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

1. **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). 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. Note also that two packets cannot be forwarded 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.
2. **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.

3. **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. 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.

4.2.3 Output Port Processing

Output port processing 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?

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.

Input Queuing

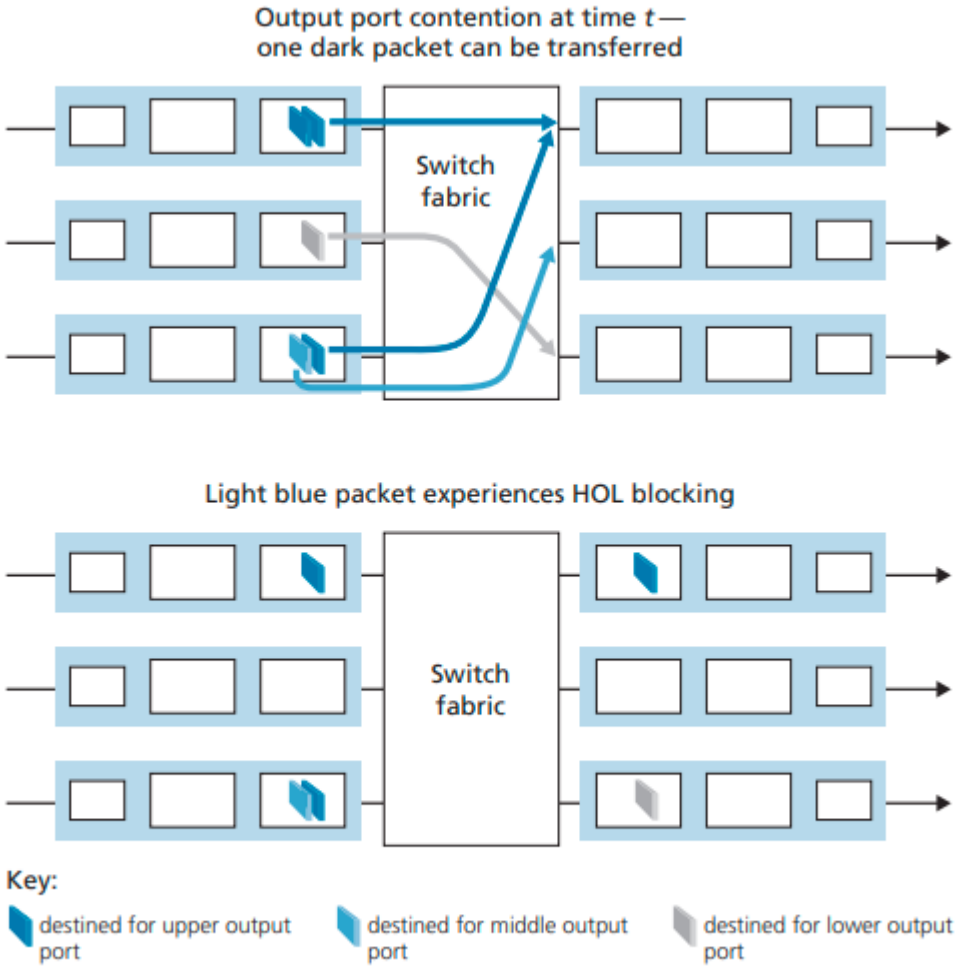

Figure 4.8 ♦ HOL blocking at and input-queued switch

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.

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 because it is blocked by another packet at the head of the line. It has been shown 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.

Output Queuing

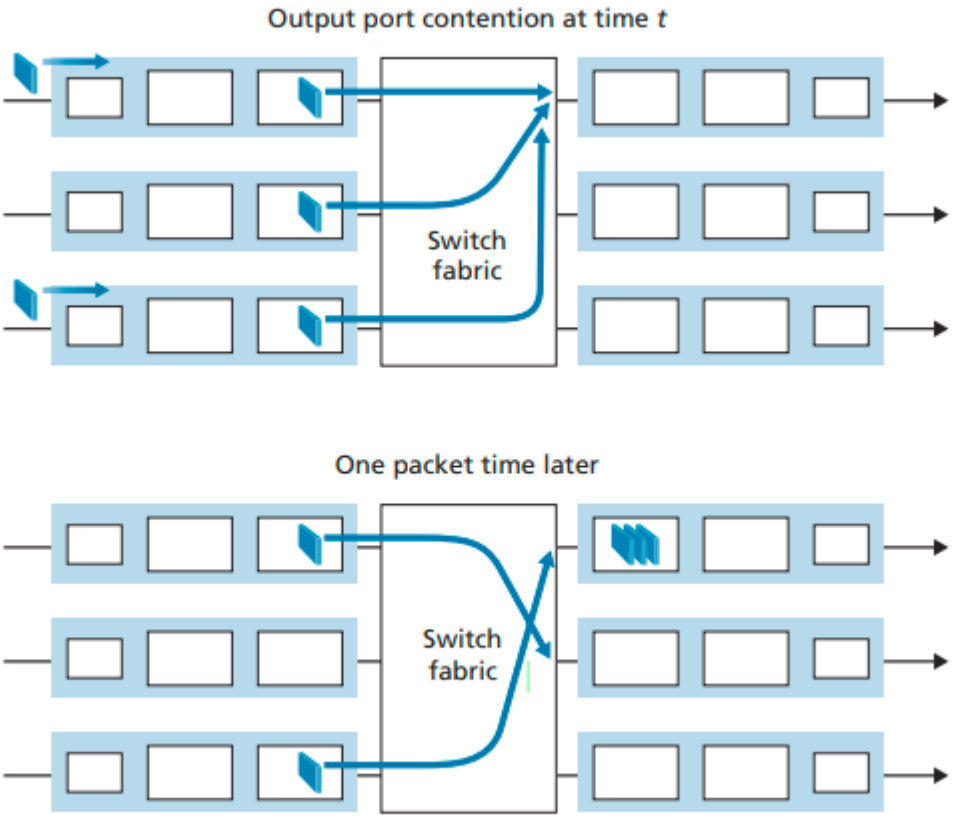

Figure 4.9 ♦ Output port queuing

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.

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

4.2.5 Packet Scheduling

First-In First-Out (FIFO)

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.

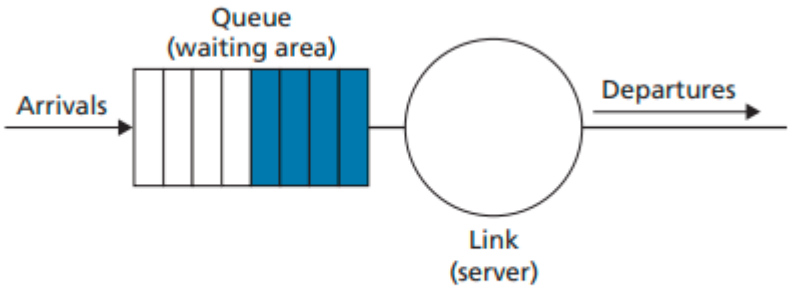

Figure 4.11 ♦ FIFO queuing abstraction

Priority Queuing

Under priority queuing, packets arriving at the output link are classified into priority classes upon arrival at the queue. When choosing a packet to transmit, the priority 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.

Under a non-preemptive priority queuing discipline, the transmission of a packet is not interrupted once it has begun.

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. 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.

A generalized form of round robin queuing that has been widely implemented in routers is the so-called weighted fair queuing (WFQ) discipline. WFQ is also a work-conserving queuing discipline and thus will immediately move on to the next class in the service sequence when it finds an empty class queue.

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

4.3.1 IPv4 Datagram Format

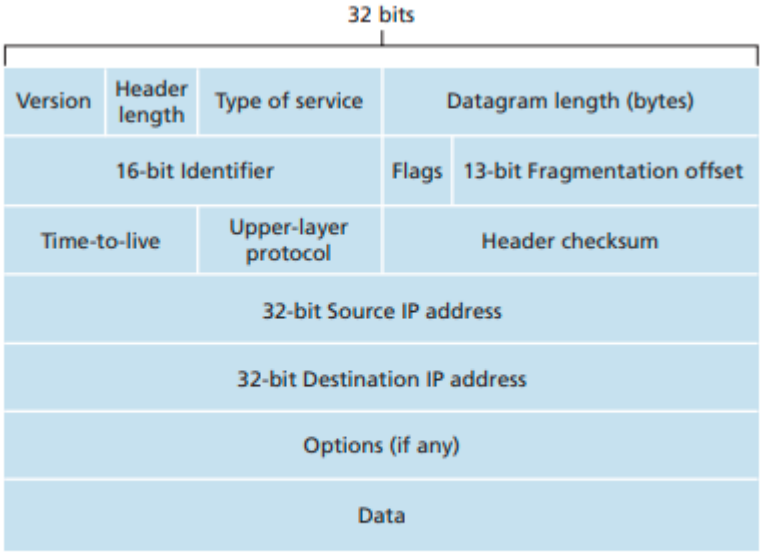

Figure 4.17 ♦ IPv4 datagram format

Recall that the Internet's network-layer packet is referred to as a datagram. The key fields in the IPv4 datagram are the following:

- Version number. These 4 bits specify the IP protocol version of the datagram. By looking at the version number, the router can determine how to interpret the remainder of the IP datagram. Different versions of IP use different datagram formats.
- Header length. Because an IPv4 datagram can contain a variable number of options, these 4 bits are needed to determine where in the IP datagram the payload actually begins.
- Type of service. The type of service (TOS) bits were included in the IPv4 header to allow different types of IP datagrams to be distinguished from each other.
- Datagram length. This is the total length of the IP datagram (header plus data), measured in bytes.
- Identifier, flags, fragmentation offset. These three fields have to do with so-called IP fragmentation, when a large IP datagram is broken into several smaller IP datagrams which are then forwarded independently to the destination, where they are reassembled before their payload data is passed up to the transport layer at the destination host. Interestingly, the new version of IP, IPv6, does not allow for fragmentation.
- Time-to-live. The time-to-live (TTL) field is included to ensure that datagrams do not circulate forever in the network. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, a router must drop that datagram.
- Protocol. This field is typically used only when an IP datagram reaches its final destination. The value of this field indicates the specific transport-layer protocol to which the data portion of this IP datagram should be passed.
- Header checksum. The header checksum aids a router in detecting bit errors in a received IP datagram. Routers typically discard datagrams for which an error has been detected.
- Source and destination IP addresses. When a source creates a datagram, it inserts its IP address into the source IP address field and inserts the address of the ultimate destination into the destination IP address field.
- Options. The options fields allow an IP header to be extended.
- Data (payload). The data field of the IP P datagram contains the transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages

4.3.2 IPv4 Addressing

A host typically has only a single link into the network; when IP in the host wants to send a datagram, it does so over this link. The boundary between the host and the physical link is called an interface. The boundary between the router and any one of its links is also called an interface. A router thus has multiple interfaces, one for each of its links. Because every host and router is capable of sending and receiving IP datagrams, IP requires each host and router interface to have its own IP address. Thus, an IP address is technically associated with an interface, rather than with the host or router containing that interface.

Each interface on every host and router in the global Internet must have an IP address that is globally unique (except for interfaces behind NATs). These addresses cannot be chosen in a willy-nilly manner, however. A portion of an interface's IP address will be determined by the subnet to which it is connected.

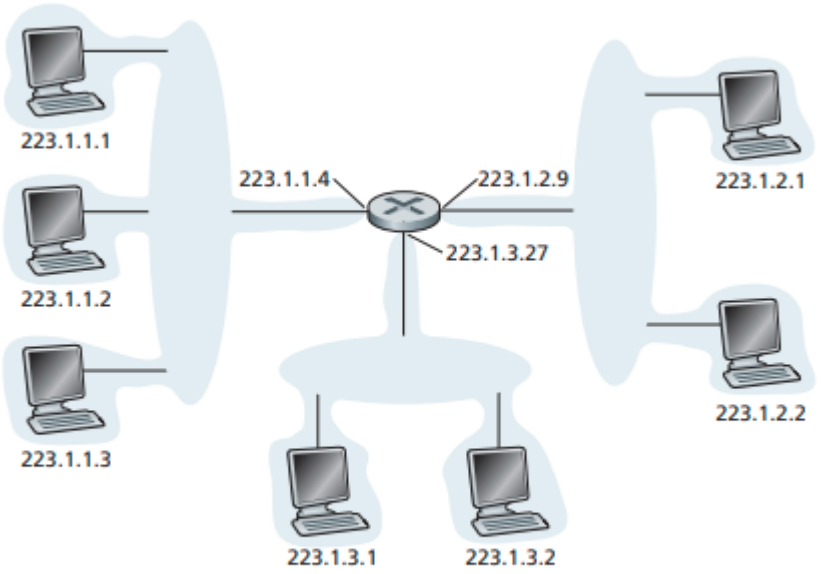

Figure 4.18 ♦ Interface addresses and subnets

In this figure, one router (with three interfaces) is used to interconnect seven hosts. Take a close look at the IP addresses assigned to the host and router interfaces, as there are several things to notice. The three hosts in the upper-left portion and the router interface to which they are connected, all have an IP address of the form 223.1.1.xxx. That is, they all have the same leftmost 24 bits in their IP address. These four interfaces are also interconnected to each other by a network that

contains no routers. This network could be interconnected by an Ethernet LAN, in which case the interfaces would be interconnected by an Ethernet switch, or by a wireless access point. We'll represent this routerless network connecting these hosts as a cloud for now.

In IP terms, this network interconnecting three host interfaces and one router interface forms a subnet. (A subnet is also called an IP network or simply a network in the Internet literature.) IP addressing assigns an address to this subnet: 223.1.1.0/24, where the /24 ("slash-24") notation, sometimes known as a subnet mask, indicates that the leftmost 24 bits of the 32-bit quantity define the subnet address.

The IP definition of a subnet is not restricted to Ethernet segments that connect multiple hosts to a router interface.

To determine the subnets, detach each interface from its host or router, creating islands of isolated networks, with interfaces terminating the end points of the isolated networks. Each of these isolated networks is called a subnet.

With multiple Ethernet segments and point-to-point links will have multiple subnets, with all of the devices on a given subnet having the same subnet address.

The Internet's address assignment strategy is known as Classless Interdomain Routing. CIDR generalizes the notion of subnet addressing. As with subnet addressing, the 32-bit IP address is divided into two parts and again has the dotted-decimal form a.b.c.d/x, where x indicates the number of bits in the first part of the address.

The x most significant bits of an address of the form a.b.c.d/x constitute the network portion of the IP address, and are often referred to as the prefix (or network prefix) of the address. An organization is typically assigned a block of contiguous addresses, that is, a range of addresses with a common prefix. In this case, the IP addresses of devices within the organization will share the common prefix. We'll see that only these x leading prefix bits are considered by routers outside the organization's network. That is, when a router outside the organization forwards a datagram whose destination address is inside the organization, only the leading x bits of the address need be considered. This considerably reduces the size of the forwarding table in these routers, since a single entry of the form a.b.c.d/x will be sufficient to forward packets to any destination within the organization.

The remaining '32 - x' bits of an address can be thought of as distinguishing among the devices within the organization, all of which have the same network prefix.

Before CIDR was adopted, the network portions of an IP address were constrained to be 8, 16, or 24 bits in length, an addressing scheme known as classful addressing, since subnets with 8-, 16-, and 24-bit subnet addresses were known as class A, B, and C networks, respectively.

We would be remiss if we did not mention yet another type of IP address, the IP broadcast address 255.255.255.255. When a host sends a datagram with destination address 255.255.255.255, the message is delivered to all hosts on the same subnet. Routers optionally forward the message into neighboring subnets as well.

Obtaining a Block of Addresses

In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP.

While obtaining a set of addresses from an ISP is one way to get a block of addresses, it is not the only way. Clearly, there must also be a way for the ISP itself to get a block of addresses. Is there a global authority that has ultimate responsibility for managing the IP address space and allocating address blocks to ISPs and other organizations? Indeed there is! IP addresses are managed under the authority of the Internet Corporation for Assigned Names and Numbers (ICANN). The role of the nonprofit ICANN organization is not only to allocate IP addresses, but also to manage the DNS root servers. It also has the very contentious job of assigning domain names and resolving domain name disputes. The ICANN allocates addresses to regional Internet registries, and handle the allocation/management of addresses within their regions.

Obtaining a Host Address: The Dynamic Host Configuration Protocol

A system administrator will typically manually configure the IP addresses into the router (often remotely, with a network management tool). **Host addresses can also be configured manually**, but typically this is done using the Dynamic Host Configuration Protocol (DHCP). DHCP allows a host to obtain (be allocated) an IP address automatically. A network administrator can configure DHCP so that a given host receives the same IP address each time it connects to the network, or a host may be assigned a temporary IP address that will be different each time the host

connects to the network. In addition to host IP address assignment, DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router (often called the default gateway), and the address of its local DNS server.

Because of DHCP's ability to automate the network-related aspects of connecting a host into a network, it is often referred to as a plug-and-play or zeroconf (zero-configuration) protocol. This capability makes it very attractive to the network administrator who would otherwise have to perform these tasks manually!

DHCP is a client-server protocol. A client is typically a newly arriving host wanting to obtain network configuration information, including an IP address for itself. In the simplest case, each subnet will have a DHCP server. If no server is present on the subnet, a DHCP relay agent (typically a router) that knows the address of a DHCP server for that network is needed.

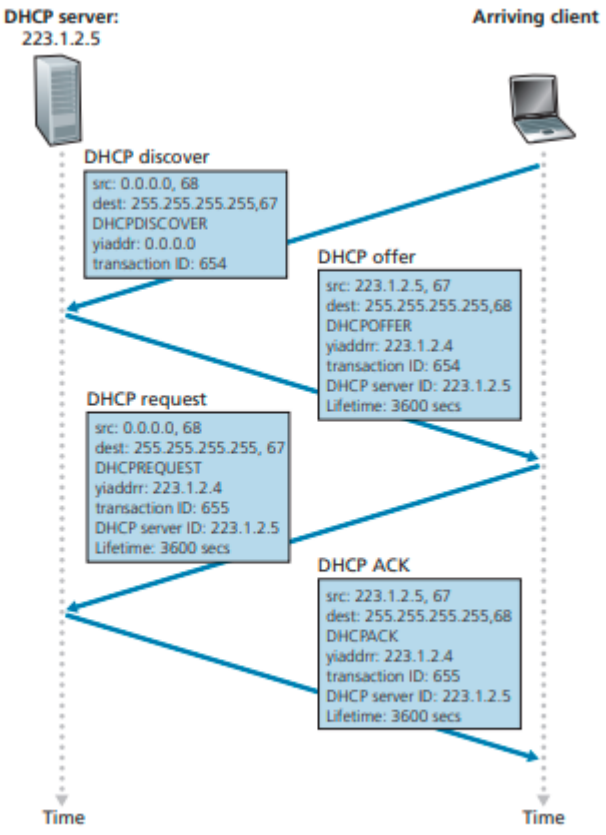

Figure 4.24 ♦ DHCP client-server interaction

For a newly arriving host, the DHCP protocol is a four-step process. In this figure, yiaddr (as in "your Internet address") indicates the address being allocated to the newly arriving client. The four steps are:

1. **DHCP server discovery.** The first task of a newly arriving host is to find a DHCP server with which to interact. This is done using a DHCP discover message, which a client sends within a UDP packet to port 67. The UDP packet is encapsulated in an IP datagram. Given this, the DHCP client creates an IP datagram containing its DHCP discover message along with the broadcast destination IP address of 255.255.255.255 and a “this host” source IP address of 0.0.0.0. The DHCP client passes the IP datagram to the link layer, which then broadcasts this frame to all nodes attached to the subnet.
2. **DHCP server offer(s).** A DHCP server receiving a DHCP discover message responds to the client with a DHCP offer message that is broadcast to all nodes on the subnet, again using the IP broadcast address of 255.255.255.255. (You might want to think about why this server reply must also be broadcast). Since several DHCP servers can be present on the subnet, the client may find itself in the enviable position of being able to choose from among several offers. Each server offer message contains the transaction ID of the received discover message, the proposed IP address for the client, the network mask, and an IP address lease time—the amount of time for which the IP address will be valid.
3. **DHCP request.** The newly arriving client will choose from among one or more server offers and respond to its selected offer with a DHCP request message, echoing back the configuration parameters.
4. **DHCP ACK.** The server responds to the DHCP request message with a DHCP ACK message, confirming the requested parameters.

Once the client receives the DHCP ACK, the interaction is complete and the client can use the DHCP-allocated IP address for the lease duration. Since a client may want to use its address beyond the lease’s expiration, DHCP also provides a mechanism that allows a client to renew its lease on an IP address.

4.3.3 Network Address Translation (NAT)

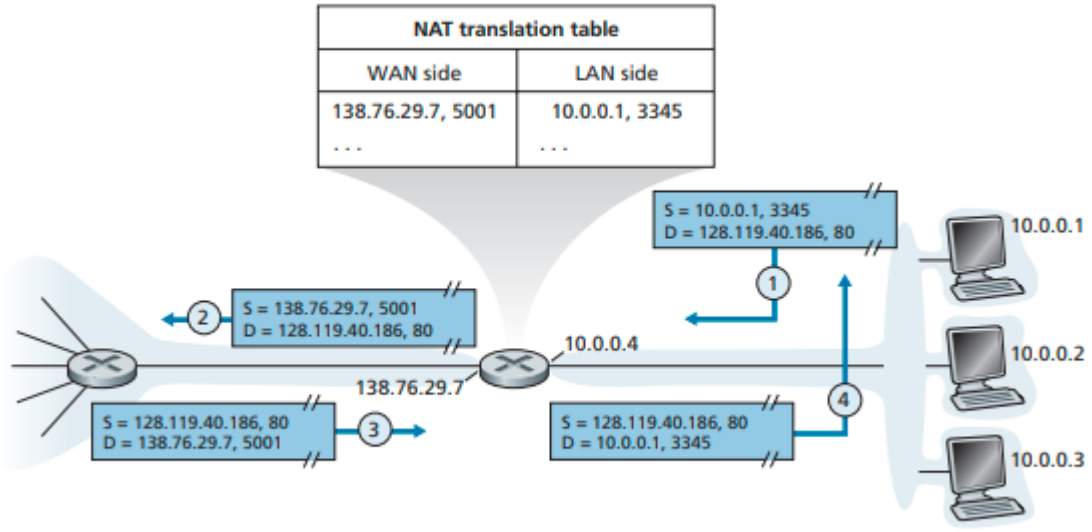

Figure 4.25 ♦ Network address translation

A realm with private addresses refers to a network whose addresses only have meaning to devices within that network. However, packets forwarded beyond the home network into the larger global Internet clearly cannot use these addresses (as either
a source or a destination address) because there are hundreds of thousands of networks using this block of addresses. That is, the 10.0.0.0/24 addresses can only have meaning within the given home network.

The NAT-enabled router does not look like a router to the outside world. Instead the NAT router behaves to the outside world as a single device with a single IP address. In Figure 4.25, all traffic leaving the home router for the larger Internet has a source IP address of 138.76.29.7, and all traffic entering the home router must have a destination address of 138.76.29.7. In essence, the NAT-enabled router is hiding the details of the home network from the outside world. Often, the answer is the same—DHCP! The router gets its address from the ISP's DHCP server, and the router runs a DHCP server to provide addresses to computers within the NAT-DHCP-router-controlled home network's address space.)

If all datagrams arriving at the NAT router from the WAN have the same destination IP address (specifically, that of the WAN-side interface of the NAT router), then how does the router know the internal host to which it should forward a given datagram? The trick is to use a NAT translation table at the NAT router, and to include port numbers as well as IP addresses in the table entries.

NAT has enjoyed widespread deployment in recent years. But NAT is not without detractors. First, one might argue that, port numbers are meant to be used for addressing processes, not for addressing hosts. This violation can indeed cause problems for servers running on the home network, since, as we have seen in Chapter 2, server processes wait for incoming requests at well-known port numbers and peers in a P2P protocol need to accept incoming connections when acting as servers. How can one peer connect to another peer that is behind a NAT server, and has a DHCP-provided NAT address? Technical solutions to these problems include NAT traversal tools.

More “philosophical” arguments have also been raised against NAT by architectural purists. Here, the concern is that routers are meant to be layer 3 (i.e., network-layer) devices, and should process packets only up to the network layer. NAT violates this principle that hosts should be talking directly with each other, without interfering nodes modifying IP addresses, much less port numbers.

4.3.4 IPv6

A prime motivation for this effort was the realization that the 32-bit IPv4 address space was beginning to be used up, with new subnets and IP nodes being attached to the Internet (and being allocated unique IP addresses) at a breathtaking rate. To respond to this need for a large IP address space, a new IP protocol, IPv6, was developed.

In February 2011, IANA allocated out the last remaining pool of unassigned IPv4 addresses to a regional registry. While these registries still have available IPv4 addresses within their pool, once these addresses are exhausted, there are no more available address blocks that can be allocated from a central pool.

IPv6 Datagram Format

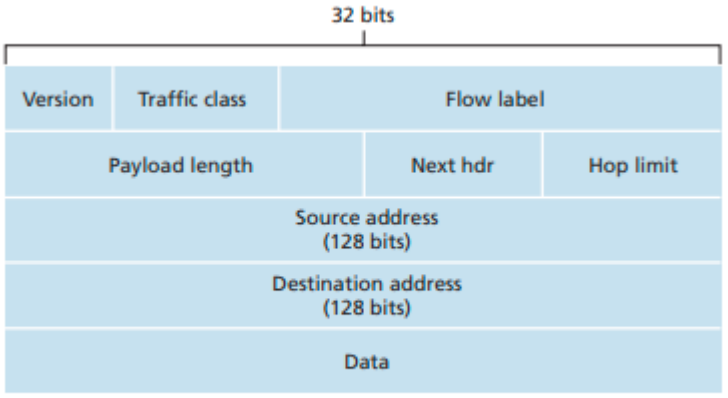

Figure 4.26 ♦ IPv6 datagram format

- **Expanded addressing capabilities.** IPv6 increases the size of the IP address from 32 to 128 bits. This ensures that the world won't run out of IP addresses. In addition to unicast and
  multicast addresses, IPv6 has introduced a new type of address, called an anycast address, that allows a datagram to be delivered to any one of a group of hosts.
- **A streamlined 40-byte header.** A number of IPv4 fields have been dropped or made optional. The resulting 40-byte fixed-length header allows for faster processing of the IP datagram by a router. A new encoding of options allows for more flexible options processing.
- **Flow labeling.** IPv6 has an elusive definition of a flow. RFC 2460 states that this allows "labeling of packets belonging to particular flows for which the sender requests special handling, such as a non-default quality of service or real-time service." What is clear, however, is that the designers of IPv6 foresaw the eventual need to be able to differentiate among the flows, even if the exact meaning of a flow had yet to be determined.

The following fields are defined in IPv6:

- **Version.** This 4-bit field identifies the IP version number.
- **Traffic class.** The 8-bit traffic class field, like the TOS field in IPv4, can be used to give priority to certain datagrams within a flow, or it can be used to give priority to datagrams from certain applications.
- **Flow label.** As discussed above, this 20-bit field is used to identify a flow of datagrams.

- **Payload length.** This 16-bit value is treated as an unsigned integer giving the number of bytes in the IPv6 datagram following the fixed-length, 40-byte datagram header.
- **Next header.** This field identifies the protocol to which the contents (data field) of this datagram will be delivered.
- Hop limit. The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, a router must discard that datagram.
- Source and destination addresses.
- Data. This is the payload portion of the IPv6 datagram. When the datagram reaches its destination, the payload will be removed from the IP datagram and passed on to the protocol specified in the next header field.

We notice that several fields appearing in the IPv4 datagram are no longer present in the IPv6 datagram:

- **Fragmentation/reassembly.** IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a “Packet Too Big” ICMP error message back to the sender. Fragmentation and reassembly is a time-consuming operation; removing this functionality from the routers and placing it squarely in the end systems considerably speeds up IP forwarding within the network.
- **Header checksum.** Because the transport-layer and link-layer protocols in the Internet layers perform checksumming, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed.
- **Options.** An options field is no longer a part of the standard IP header. However, it has not gone away. Instead, the options field is one of the possible next headers pointed to from within the IPv6 header.

Transitioning from IPv4 to IPv6

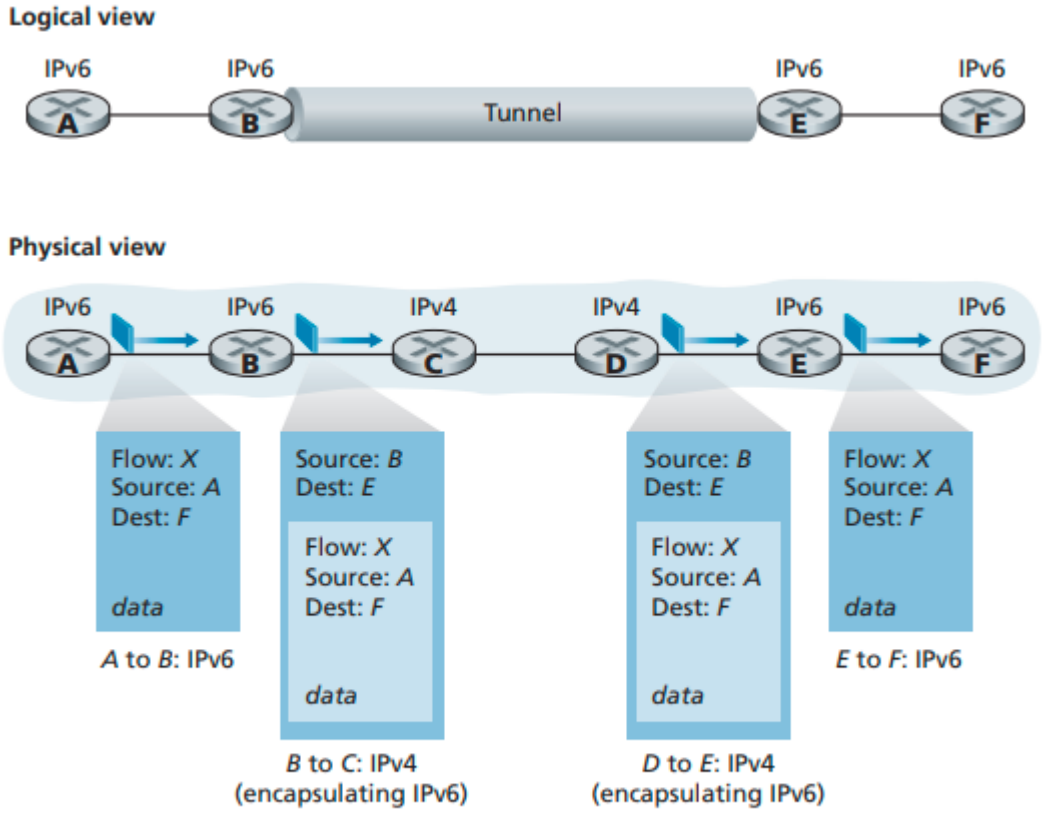

Figure 4.27 ♦ Tunneling

The problem is that while new IPv6-capable systems can be made backward-compatible, that is, can send, route, and receive IPv4 datagrams, already deployed IPv4-capable systems are not capable of handling IPv6 datagrams. Several options are possible.

One option would be to declare a flag day—a given time and date when all Internet machines would be turned off and upgraded from IPv4 to IPv6. A flag day involving billions of devices is even more unthinkable today.

The approach to IPv4-to-IPv6 transition that has been most widely adopted in practice involves **tunneling**. The basic idea behind tunneling is the following. Suppose two IPv6 nodes (in this example, B and E in Figure 4.27) want to interoperate using IPv6 datagrams but are connected to each other by intervening IPv4 routers. We refer to the intervening set of IPv4 routers between two IPv6 routers as a tunnel. With tunneling, the IPv6 node on the sending side of the tunnel (in this example, B) takes the entire IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram. This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel (in this example, E) and sent to the first node in the tunnel (in this example, C). The intervening IPv4 routers in the tunnel route this IPv4

datagram among themselves, just as they would any other datagram, blissfully unaware that the IPv4 datagram itself contains a complete IPv6 datagram. The IPv6 node on the receiving side of the tunnel eventually receives the IPv4 datagram, determines that the IPv4 datagram contains an IPv6 datagram (by observing that the protocol number field in the IPv4 datagram is 41 [RFC 4213], indicating that the IPv4 payload is a IPv6 datagram), extracts the IPv6 datagram, and then routes the IPv6 datagram exactly as it would if it had received the IPv6 datagram from a directly connected IPv6 neighbor.

One important lesson that we can learn from the IPv6 experience is that it is enormously difficult to change network-layer protocols. Indeed, introducing new protocols into the network layer is like replacing the foundation of a house. Introducing new application-layer protocols is like adding a new layer of paint to a house—it is relatively easy to do, and if you choose an attractive color, others in the neighborhood will copy you.

4.4 Generalized Forwarding and SDN

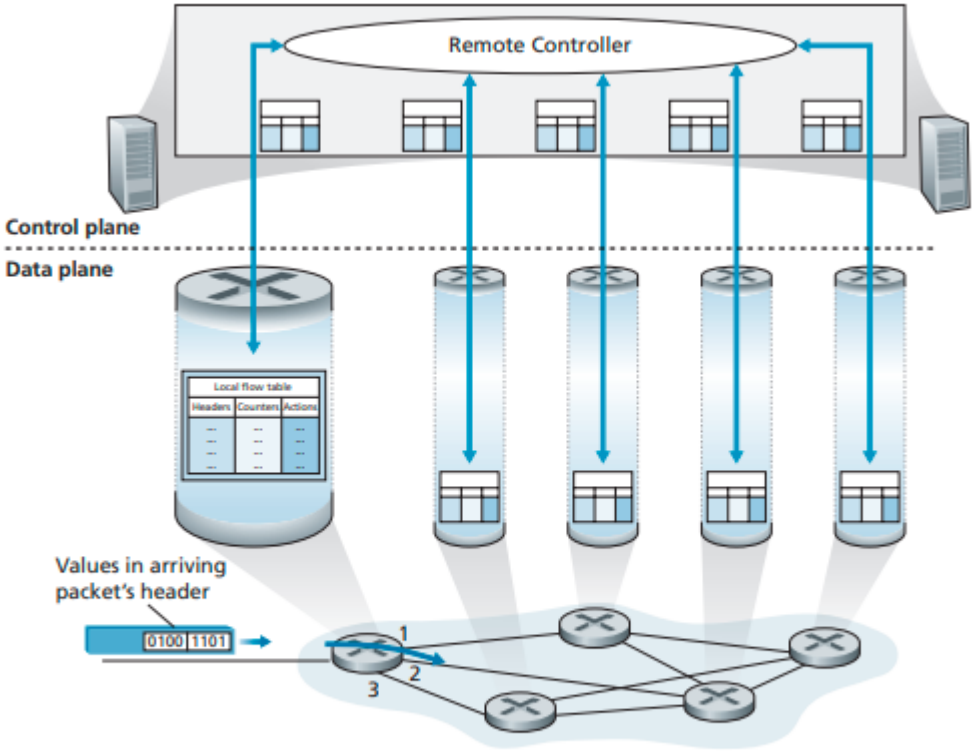

Figure 4.28 ♦ Generalized forwarding: Each packet switch contains a match-plus-action table that is computed and distributed by a remote controller

Let's now consider a significantly more general "match-plus-action" paradigm, where the "match" can be made over multiple header fields associated with different protocols at different layers in the protocol stack. The "action" can include forwarding the packet to one or more output ports (as in destination-based forwarding), load balancing packets across multiple outgoing interfaces that lead to a service (as in load balancing), rewriting header values (as in NAT), purposefully blocking/dropping a packet (as in a firewall), sending a packet to a special server for further processing and action (as in DPI), and more.

In generalized forwarding, a match-plus-action table generalizes the notion of the destination-based forwarding table.

Each entry in the match-plus-action forwarding table, known as a flow table in OpenFlow, includes:

- **A set of header field values** to which an incoming packet will be matched. As in the case of destination-based forwarding, hardware-based matching is most rapidly performed in TCAM memory. A packet that matches no flow table entry can be dropped or sent to the remote controller for more processing.
- **A set of counters** that are updated as packets are matched to flow table entries. These counters might include the number of packets that have been matched by that table entry, and the time since the table entry was last updated.
- **A set of actions to be taken** when a packet matches a flow table entry. These actions might be to forward the packet to a given output port, to drop the packet, makes copies of the packet and sent them to multiple output ports, and/or to rewrite selected header fields.

4.4.1 Match

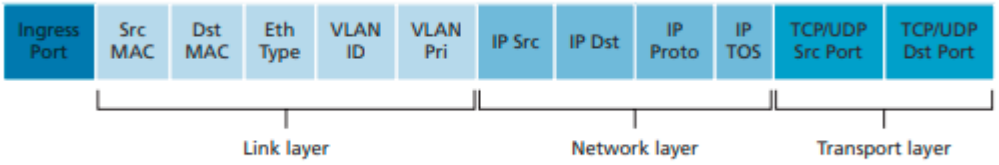

Figure 4.29 ♦ Packet matching fields, OpenFlow 1.0 flow table

Figure 4.29 shows the 11 packet-header fields and the incoming port ID that can be matched in an OpenFlow 1.0 match-plus-action rule. The first observation we make is that OpenFlow's match abstraction allows for a match to be made on selected

fields from three layers of protocol headers. Since we've not yet covered the link-layer, suffice it to say that the source and destination MAC addresses shown in Figure 4.29 are the link-layer addresses associated with the frame's sending and receiving interfaces; by forwarding on the basis of Ethernet addresses rather than IP addresses, we can see that an OpenFlow-enabled device can equally perform as a router (layer-3 device) forwarding datagrams as well as a switch (layer-2 device) forwarding frames.

The ingress port refers to the input port at the packet switch on which a packet is received. The packet's IP source address, IP destination address, IP protocol field, and IP type of service fields were discussed earlier in Section 4.3.1. The transport-layer source and destination port number fields can also be matched.

For example, an IP address of 128.119.. in a flow table will match the corresponding address field of any datagram that has 128.119 as the first 16 bits of its address. Each flow table entry also has an associated priority. If a packet matches multiple flow table entries, the selected match and corresponding action will be that of the highest priority entry with which the packet matches.

Lastly, we observe that not all fields in an IP header can be matched. Why are some fields allowed for matching, while others are not? Undoubtedly, the answer has to do with the tradeoff between functionality and complexity. The "art" in choosing an abstraction is to provide for enough functionality to accomplish a task without over-burdening the abstraction with so much detail
and generality that it becomes bloated and unusable.

> Do one thing at a time, and do it well. Aa interface should capture the minimum essentials of an abstraction. Don't generalize; generalizations are generally wrong.

4.4.2 Action

Each flow table entry has a list of zero or more actions that determine the processing that is to be applied to a packet that matches a flow table entry. If there are multiple actions, they are performed in the order specified in the list.

Among the most important possible actions are:

- **Forwarding.** An incoming packet may be forwarded to a particular physical output port, broadcast over all ports (except the port on which it arrived) or

multicast over a selected set of ports. The packet may be encapsulated and sent to the remote controller for this device. That controller then may (or may not) take some action on that packet, including installing new flow table entries, and may return the packet to the device for forwarding under the updated set of flow table rules.

- **Dropping.** A flow table entry with no action indicates that a matched packet should be dropped.
- **Modify-field.** The values in 10 packet-header fields may be re-written before the packet is forwarded to the chosen output port.

4.4.3 OpenFlow Examples of Match-plus-action in Action

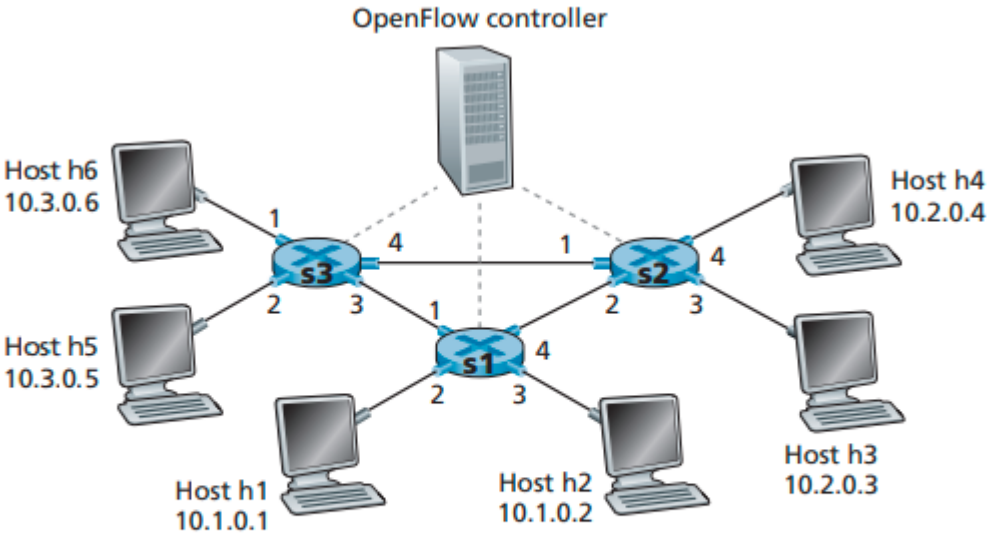

Figure 4.30 ♦ OpenFlow match-plus-action network with three packet switches, 6 hosts, and an OpenFlow controller

A First Example: Simple Forwarding

As a very simple example, suppose that the desired forwarding behavior is that packets from h5 or h6 destined to h3 or h4 are to be forwarded from s3 to s1, and then from s1 to s2 (thus completely avoiding the use of the link between s3 and s2). The flow table entry in s1 would be:

s1 Flow Table (Example 1)	
Match	Action
Ingress Port = 1 ; IP Src = 10.3.*.* ; IP Dst = 10.2.*.*	Forward(4)
…	…

Of course, we'll also need a flow table entry in s3 so that datagrams sent from h5 or h6 are forwarded to s1 over outgoing interface 3:

s3 Flow Table (Example 1)	
Match	Action
IP Src = 10.3.*.* ; IP Dst = 10.2.*.*	Forward(3)
…	…

Lastly, we'll also need a flow table entry in s2 to complete this first example, so that datagrams arriving from s1 are forwarded to their destination, either host h3 or h4:

s2 Flow Table (Example 1)	
Match	Action
Ingress port = 2 ; IP Dst = 10.2.0.3	Forward(3)
Ingress port = 2 ; IP Dst = 10.2.0.4	Forward(4)
…	…

A Second Example: Firewalling

As a third example, let's consider a firewall scenario in which s2 wants only to receive (on any of its interfaces) traffic sent from hosts attached to s3.

s2 Flow Table (Example 3)	
Match	Action
IP Src = 10.3.*.* IP Dst = 10.2.0.3	Forward(3)
IP Src = 10.3.*.* IP Dst = 10.2.0.4	Forward(4)
…	…

If there were no other entries in s2's flow table, then only traffic from 10.3.. would be forwarded to the hosts attached to s2.

The match-plus-action flow tables that we've seen in this section are actually a limited form of programmability, specifying how a router should forward and manipulate (e.g., change a header field) a datagram, based on the match between the datagram's header values and the matching conditions.