control plane implemented within the same device or in separate devices? Explain. R3. Compare and contrast the properties of a centralized and a distributed routing algorithm. Give an example of a routing protocol that takes a centralized and a decentralized approach. R4. Compare and contrast link-state and distance-vector routing algorithms. R5. What is the “count to infinity” problem in distance vector routing? R6. Is it necessary that every autonomous system use the same intra-AS routing algorithm? Why or why not? R7. Why are different inter-AS and intra-AS protocols used in the Internet? R8. True or false: When an OSPF route sends its link state information, it is sent only to those nodes directly attached neighbors. Explain. R9. What is meant by an area in an OSPF autonomous system? Why was the concept of an area introduced? R10. Define and contrast the following terms: subnet, prefix, and BGP route. R11. How does BGP use the NEXT-HOP attribute? How does it use the AS-PATH attribute? R12. Describe how a network administrator of an upper-tier ISP can implement policy when configuring BGP. R13. True or false: When a BGP router receives an advertised path from its neighbor, it must add its own identity to the received path and then send that new path on to all of its neighbors. SECTION 5.5 SECTIONS 5.6–5.7 Problems Explain. R14. Describe the main role of the communication layer, the network-wide state-management layer, and the network-control application layer in an SDN controller. R15. Suppose you wanted to implement a new routing protocol in the SDN control plane. At which layer would you implement that protocol? Explain. R16. What types of messages flow across an SDN controller’s northbound and southbound APIs? Who is the recipient of these messages sent from the controller across the southbound interface, and who sends messages to the controller across the northbound interface? R17. Describe the purpose of two types of OpenFlow messages (of your choosing) that are sent from a controlled device to the controller. Describe the purpose of two types of Openflow messages (of your choosing) that are send from the controller to a controlled device. R18. What is the purpose of the service abstraction layer in the OpenDaylight SDN controller? R19. Names four different types of ICMP messages R20. What two types of ICMP messages are received at the sending host executing the Traceroute program? R21. Define the following terms in the context of SNMP: managing server, managed device, network management agent and MIB. R22. What are the purposes of the SNMP GetRequest and SetRequest messages? R23. What is the purpose of the SNMP trap message? P1. Looking at Figure 5.3 , enumerate the paths from y to u that do not contain any loops. P2. Repeat Problem P1 for paths from x to z, z to u, and z to w. P3. Consider the following network. With the indicated link costs, use Dijkstra’s shortest-path algorithm to compute the shortest path from x to all network nodes. Show how the algorithm works by computing a table similar to Table 5.1 . Dijkstra’s algorithm: discussion and example P4. Consider the network shown in Problem P3. Using Dijkstra’s algorithm, and showing your work using a table similar to Table 5.1 , do the following: a. Compute the shortest path from t to all network nodes. b. Compute the shortest path from u to all network nodes. c. Compute the shortest path from v to all network nodes. d. Compute the shortest path from w to all network nodes. e. Compute the shortest path from y to all network nodes. f. Compute the shortest path from z to all network nodes. P5. Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance-vector algorithm and show the distance table entries at node z. P6. Consider a general topology (that is, not the specific network shown above) and a synchronous version of the distance-vector algorithm. Suppose that at each iteration, a node exchanges its distance vectors with its neighbors and receives their distance vectors. Assuming that the algorithm begins with each node knowing only the costs to its immediate neighbors, what is the maximum number of iterations required before the distributed algorithm converges? Justify your answer. P7. Consider the network fragment shown below. x has only two attached neighbors, w and y. w has a minimum-cost path to destination u (not shown) of 5, and y has a minimum-cost path to u of 6. The complete paths from w and y to u (and between w and y) are not shown. All link costs in the network have strictly positive integer values. a. Give x's distance vector for destinations w, y, and u. b. Give a link-cost change for either c(x, w) or c(x, y) such that x will inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm. c. Give a link-cost change for either c(x, w) or c(x, y) such that x will not inform its neighbors of a new minimum-cost path to u as a result of executing the distance-vector algorithm. P8. Consider the three-node topology shown in Figure 5.6 . Rather than having the link costs shown in Figure 5.6 , the link costs are Compute the distance tables after the initialization step and after each iteration of a synchronous version of the distancevector algorithm (as we did in our earlier discussion of Figure 5.6). P9. Consider the count-to-infinity problem in the distance vector routing. Will the count-to-infinity problem occur if we decrease the cost of a link? Why? How about if we connect two nodes which do not have a link? P10. Argue that for the distance-vector algorithm in Figure 5.6 , each value in the distance vector D(x) is non-increasing and will eventually stabilize in a finite number of steps. P11. Consider Figure 5.7. Suppose there is another router w, connected to router y and z. The costs of all links are given as follows: Suppose that poisoned reverse is used in the distance-vector routing algorithm. a. When the distance vector routing is stabilized, router w, y, and z inform their distances to x to each other. What distance values do they tell each other? b. Now suppose that the link cost between x and y increases to 60. Will there be a count-toinfinity problem even if poisoned reverse is used? Why or why not? If there is a count-toinfinity problem, then how many iterations are needed for the distance-vector routing to c(x,y)=3, c(y,z)=6, c(z,x)=4. c(x,y)=4, c(x,z)=50, c(y,w)=1, c(z,w)=1, c(y,z)=3. reach a stable state again? Justify your answer. c. How do you modify c(y, z) such that there is no count-to-infinity problem at all if c(y,x) changes from 4 to 60? P12. Describe how loops in paths can be detected in BGP. P13. Will a BGP router always choose the loop-free route with the shortest ASpath length? Justify your answer. P14. Consider the network shown below. Suppose AS3 and AS2 are running OSPF for their intra-AS routing protocol. Suppose AS1 and AS4 are running RIP for their intra-AS routing protocol. Suppose eBGP and iBGP are used for the inter-AS routing protocol. Initially suppose there is no physical link between AS2 and AS4. a. Router 3c learns about prefix x from which routing protocol: OSPF, RIP, eBGP, or iBGP? b. Router 3a learns about x from which routing protocol? c. Router 1c learns about x from which routing protocol? d. Router 1d learns about x from which routing protocol? P15. Referring to the previous problem, once router 1d learns about x it will put an entry (x, I) in its forwarding table. a. Will I be equal to I or I for this entry? Explain why in one sentence. b. Now suppose that there is a physical link between AS2 and AS4, shown by the dotted line. Suppose router 1d learns that x is accessible via AS2 as well as via AS3. Will I be set to I or I ? Explain why in one sentence. c. Now suppose there is another AS, called AS5, which lies on the path between AS2 and AS4 (not shown in diagram). Suppose router 1d learns that x is accessible via AS2 AS5 AS4 as well as via AS3 AS4. Will I be set to I or I ? Explain why in one sentence. 1 2 1 2 1 2 P16. Consider the following network. ISP B provides national backbone service to regional ISP A. ISP C provides national backbone service to regional ISP D. Each ISP consists of one AS. B and C peer with each other in two places using BGP. Consider traffic going from A to D. B would prefer to hand that traffic over to C on the West Coast (so that C would have to absorb the cost of carrying the traffic cross-country), while C would prefer to get the traffic via its East Coast peering point with B (so that B would have carried the traffic across the country). What BGP mechanism might C use, so that B would hand over A-to-D traffic at its East Coast peering point? To answer this question, you will need to dig into the BGP specification. P17. In Figure 5.13 , consider the path information that reaches stub networks W, X, and Y. Based on the information available at W and X, what are their respective views of the network topology? Justify your answer. The topology view at Y is shown below. P18. Consider Figure 5.13 . B would never forward traffic destined to Y via X based on BGP routing. But there are some very popular applications for which data packets go to X first and then flow to Y. Identify one such application, and describe how data packets follow a path not given by BGP routing. Socket Programming Assignment At the end of Chapter 2, there are four socket programming assignments. Below, you will find a fifth assignment which employs ICMP, a protocol discussed in this chapter. Assignment 5: ICMP Ping Ping is a popular networking application used to test from a remote location whether a particular host is up and reachable. It is also often used to measure latency between the client host and the target host. It works by sending ICMP “echo request” packets (i.e., ping packets) to the target host and listening for ICMP “echo response” replies (i.e., pong packets). Ping measures the RRT, records packet loss, and calculates a statistical summary of multiple ping-pong exchanges (the minimum, mean, max, and standard deviation of the round-trip times). In this lab, you will write your own Ping application in Python. Your application will use ICMP. But in order to keep your program simple, you will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems. You can find full details of this assignment, as well as important snippets of the Python code, at the Web site http://www.pearsonhighered.com/csresources. Programming Assignment P19. In Figure 5.13 , suppose that there is another stub network V that is a customer of ISP A. Suppose that B and C have a peering relationship, and A is a customer of both B and C. Suppose that A would like to have the traffic destined to W to come from B only, and the traffic destined to V from either B or C. How should A advertise its routes to B and C? What AS routes does C receive? P20. Suppose ASs X and Z are not directly connected but instead are connected by AS Y. Further suppose that X has a peering agreement with Y, and that Y has a peering agreement with Z. Finally, suppose that Z wants to transit all of Y’s traffic but does not want to transit X’s traffic. Does BGP allow Z to implement this policy? P21. Consider the two ways in which communication occurs between a managing entity and a managed device: request-response mode and trapping. What are the pros and cons of these two approaches, in terms of (1) overhead, (2) notification time when exceptional events occur, and (3) robustness with respect to lost messages between the managing entity and the device? P22. In Section 5.7 we saw that it was preferable to transport SNMP messages in unreliable UDP datagrams. Why do you think the designers of SNMP chose UDP rather than TCP as the transport protocol of choice for SNMP? In this programming assignment, you will be writing a “distributed” set of procedures that implements a distributed asynchronous distance-vector routing for the network shown below. You are to write the following routines that will “execute” asynchronously within the emulated environment provided for this assignment. For node 0, you will write the routines: rtinit0(). This routine will be called once at the beginning of the emulation. rtinit0() has no arguments. It should initialize your distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In the figure above, all links are bidirectional and the costs in both directions are identical. After initializing the distance table and any other data structures needed by your node 0 routines, it should then send its directly connected neighbors (in this case, 1, 2, and 3) the cost of its minimum-cost paths to all other network nodes. This minimum-cost information is sent to neighboring nodes in a routing update packet by calling the routine tolayer2(), as described in the full assignment. The format of the routing update packet is also described in the full assignment. rtupdate0(struct rtpkt *rcvdpkt). This routine will be called when node 0 receives a routing packet that was sent to it by one of its directly connected neighbors. The parameter *rcvdpkt is a pointer to the packet that was received. rtupdate0() is the “heart” of the distance-vector algorithm. The values it receives in a routing update packet from some other node i contain i’s current shortest-path costs to all other network nodes. rtupdate0() uses these received values to update its own distance table (as specified by the distance-vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change in minimum cost by sending them a routing packet. Recall that in the distance-vector algorithm, only directly connected nodes will exchange routing packets. Thus, nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other. Similar routines are defined for nodes 1, 2, and 3. Thus, you will write eight procedures in all: rtinit0(), rtinit1(), rtinit2(), rtinit3(), rtupdate0(), rtupdate1(), rtupdate2(), and rtupdate3(). These routines will together implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in the figure on the preceding page. You can find the full details of the programming assignment, as well as C code that you will need to create the simulated hardware/software environment, at http://www.pearsonhighered.com/cs-resource. A Java version of the assignment is also available. Wireshark Lab In the Web site for this textbook, www.pearsonhighered.com/cs-resources, you’ll find a Wireshark lab assignment that examines the use of the ICMP protocol in the ping and traceroute commands. An Interview With… Jennifer Rexford Jennifer Rexford is a Professor in the Computer Science department at Princeton University. Her research has the broad goal of making computer networks easier to design and manage, with particular emphasis on routing protocols. From 1996–2004, she was a member of the Network Management and Performance department at AT&T Labs–Research. While at AT&T, she designed techniques and tools for network measurement, traffic engineering, and router configuration that were deployed in AT&T’s backbone network. Jennifer is co-author of the book “Web Protocols and Practice: Networking Protocols, Caching, and Traffic Measurement,” published by Addison-Wesley in May 2001. She served as the chair of ACM SIGCOMM from 2003 to 2007. She received her BSE degree in electrical engineering from Princeton University in 1991, and her PhD degree in electrical engineering and computer science from the University of Michigan in 1996. In 2004, Jennifer was the winner of ACM’s Grace Murray Hopper Award for outstanding young computer professional and appeared on the MIT TR-100 list of top innovators under the age of 35. Please describe one or two of the most exciting projects you have worked on during your career. What were the biggest challenges? When I was a researcher at AT&T, a group of us designed a new way to manage routing in Internet Service Provider backbone networks. Traditionally, network operators configure each router individually, and these routers run distributed protocols to compute paths through the network. We believed that network management would be simpler and more flexible if network operators could exercise direct control over how routers forward traffic based on a network-wide view of the topology and traffic. The Routing Control Platform (RCP) we designed and built could compute the routes for all of AT&T’s backbone on a single commodity computer, and could control legacy routers without modification. To me, this project was exciting because we had a provocative idea, a working system, and ultimately a real deployment in an operational network. Fast forward a few years, and software-defined networking (SDN) has become a mainstream technology, and standard protocols (like OpenFlow) have made it much easier to tell the underlying switches what to do. How do you think software-defined networking should evolve in the future? In a major break from the past, control-plane software can be created by many different programmers, not just at companies selling network equipment. Yet, unlike the applications running on a server or a smart phone, controller apps must work together to handle the same traffic. Network operators do not want to perform load balancing on some traffic and routing on other traffic; instead, they want to perform load balancing and routing, together, on the same traffic. Future SDN controller platforms should offer good programming abstractions for composing independently written multiple controller applications together. More broadly, good programming abstractions can make it easier to create controller applications, without having to worry about low-level details like flow table entries, traffic counters, bit patterns in packet headers, and so on. Also, while an SDN controller is logically centralized, the network still consists of a distributed collection of devices. Future controllers should offer good abstractions for updating the flow tables across the network, so apps can reason about what happens to packets in flight while the devices are updated. Programming abstractions for control-plane software is an exciting area for interdisciplinary research between computer networking, distributed systems, and programming languages, with a real chance for practical impact in the years ahead. Where do you see the future of networking and the Internet? Networking is an exciting field because the applications and the underlying technologies change all the time. We are always reinventing ourselves! Who would have predicted even ten years ago the dominance of smart phones, allowing mobile users to access existing applications as well as new location-based services? The emergence of cloud computing is fundamentally changing the relationship between users and the applications they run, and networked sensors and actuators (the “Internet of Things”) are enabling a wealth of new applications (and security vulnerabilities!). The pace of innovation is truly inspiring. The underlying network is a crucial component in all of these innovations. Yet, the network is notoriously “in the way”—limiting performance, compromising reliability, constraining applications, and complicating the deployment and management of services. We should strive to make the network of the future as invisible as the air we breathe, so it never stands in the way of new ideas and valuable services. To do this, we need to raise the level of abstraction above individual network devices and protocols (and their attendant acronyms!), so we can reason about the network and the user’s high-level goals as a whole. What people inspired you professionally? I’ve long been inspired by Sally Floyd at the International Computer Science Institute. Her research is always purposeful, focusing on the important challenges facing the Internet. She digs deeply into hard questions until she understands the problem and the space of solutions completely, and she devotes serious energy into “making things happen,” such as pushing her ideas into protocol standards and network equipment. Also, she gives back to the community, through professional service in numerous standards and research organizations and by creating tools (such as the widely used ns-2 and ns-3 simulators) that enable other researchers to succeed. She retired in 2009 but her influence on the field will be felt for years to come. What are your recommendations for students who want careers in computer science and networking? Networking is an inherently interdisciplinary field. Applying techniques from other disciplines breakthroughs in networking come from such diverse areas as queuing theory, game theory, control theory, distributed systems, network optimization, programming languages, machine learning, algorithms, data structures, and so on. I think that becoming conversant in a related field, or collaborating closely with experts in those fields, is a wonderful way to put networking on a stronger foundation, so we can learn how to build networks that are worthy of society’s trust. Beyond the theoretical disciplines, networking is exciting because we create real artifacts that real people use. Mastering how to design and build systems—by gaining experience in operating systems, computer architecture, and so on—is another fantastic way to amplify your knowledge of networking to help make the world a better place. Chapter 6 The Link Layer and LANs In the previous two chapters we learned that the network layer provides a communication service between any two network hosts. Between the two hosts, datagrams travel over a series of communication links, some wired and some wireless, starting at the source host, passing through a series of packet switches (switches and routers) and ending at the destination host. As we continue down the protocol stack, from the network layer to the link layer, we naturally wonder how packets are sent across the individual links that make up the end-to-end communication path. How are the networklayer datagrams encapsulated in the link-layer frames for transmission over a single link? Are different link-layer protocols used in the different links along the communication path? How are transmission conflicts in broadcast links resolved? Is there addressing at the link layer and, if so, how does the linklayer addressing operate with the network-layer addressing we learned about in Chapter 4? And what exactly is the difference between a switch and a router? We'll answer these and other important questions in this chapter. In discussing the link layer, we'll see that there are two fundamentally different types of link-layer channels. The first type are broadcast channels, which connect multiple hosts in wireless LANs, satellite networks, and hybrid fiber-coaxial cable (HFC) access networks. Since many hosts are connected to the same broadcast communication channel, a so-called medium access protocol is needed to coordinate frame transmission. In some cases, a central controller may be used to coordinate transmissions; in other cases, the hosts themselves coordinate transmissions. The second type of link-layer channel is the point-to-point communication link, such as that often found between two routers connected by a long-distance link, or between a user's office computer and the nearby Ethernet switch to which it is connected. Coordinating access to a point-to-point link is simpler; the reference material on this book's Web site has a detailed discussion of the Point-to-Point Protocol (PPP), which is used in settings ranging from dial-up service over a telephone line to high-speed point-to-point frame transport over fiber-optic links. We'll explore several important link-layer concepts and technologies in this chapter. We'll dive deeper into error detection and correction, a topic we touched on briefly in Chapter 3. We'll consider multiple access networks and switched LANs, including Ethernet—by far the most prevalent wired LAN technology. We'll also look at virtual LANs, and data center networks. Although WiFi, and more generally wireless LANs, are link-layer topics, we'll postpone our study of these important topics until Chapter 7. 6.1 Introduction to the Link Layer Let's begin with some important terminology. We'll find it convenient in this chapter to refer to any device that runs a link-layer (i.e., layer 2) protocol as a node. Nodes include hosts, routers, switches, and WiFi access points (discussed in Chapter 7). We will also refer to the communication channels that connect adjacent nodes along the communication path as links. In order for a datagram to be transferred from source host to destination host, it must be moved over each of the individual links in the end-to-end path. As an example, in the company network shown at the bottom of Figure 6.1, consider sending a datagram from one of the wireless hosts to one of the servers. This datagram will actually pass through six links: a WiFi link between sending host and WiFi access point, an Ethernet link between the access point and a link-layer switch; a link between the link-layer switch and the router, a link between the two routers; an Ethernet link between the router and a link-layer switch; and finally an Ethernet link between the switch and the server. Over a given link, a transmitting node encapsulates the datagram in a linklayer frame and transmits the frame into the link. In order to gain further insight into the link layer and how it relates to the network layer, let's consider a transportation analogy. Consider a travel agent who is planning a trip for a tourist traveling from Princeton, New Jersey, to Lausanne, Switzerland. The travel agent decides that it is most convenient for the tourist to take a limousine from Princeton to JFK airport, then a plane from JFK airport to Geneva's airport, and finally a train from Geneva's airport to Lausanne's train station. Once the travel agent makes the three reservations, it is the responsibility of the Princeton limousine company to get the tourist from Princeton to JFK; it is the responsibility of the airline company to get the tourist from JFK to Geneva; and it is the responsibility Figure 6.1 Six link-layer hops between wireless host and server of the Swiss train service to get the tourist from Geneva to Lausanne. Each of the three segments of the trip is "direct" between two "adjacent" locations. Note that the three transportation segments are managed by different companies and use entirely different transportation modes (limousine, plane, and train). Although the transportation modes are different, they each provide the basic service of moving passengers from one location to an adjacent location. In this transportation analogy, the tourist is a datagram, each transportation segment is a link, the transportation mode is a link-layer protocol, and the travel agent is a routing protocol. 6.1.1 The Services Provided by the Link Layer Although the basic service of any link layer is to move a datagram from one node to an adjacent node over a single communication link, the details of the provided service can vary from one link-layer protocol to the next. Possible services that can be offered by a link-layer protocol include: Framing. Almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission over the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields. The structure of the frame is specified by the link-layer protocol. We'll see several different frame formats when we examine specific link-layer protocols in the second half of this chapter. Link access. A medium access control (MAC) protocol specifies the rules by which a frame is transmitted onto the link. For point-to-point links that have a single sender at one end of the link and a single receiver at the other end of the link, the MAC protocol is simple (or nonexistent)—the sender can send a frame whenever the link is idle. The more interesting case is when multiple nodes share a single broadcast link—the so-called multiple access problem. Here, the MAC protocol serves to coordinate the frame transmissions of the many nodes. Reliable delivery. When a link-layer protocol provides reliable delivery service, it guarantees to move each network-layer datagram across the link without error. Recall that certain transport-layer protocols (such as TCP) also provide a reliable delivery service. Similar to a transport-layer reliable delivery service, a link-layer reliable delivery service can be achieved with acknowledgments and retransmissions (see Section 3.4). A link-layer reliable delivery service is often used for links that are prone to high error rates, such as a wireless link, with the goal of correcting an error locally—on the link where the error occurs—rather than forcing an end-to-end retransmission of the data by a transport- or application-layer protocol. However, link-layer reliable delivery can be considered an unnecessary overhead for low bit-error links, including fiber, coax, and many twisted-pair copper links. For this reason, many wired link-layer protocols do not provide a reliable delivery service. Error detection and correction. The link-layer hardware in a receiving node can incorrectly decide that a bit in a frame is zero when it was transmitted as a one, and vice versa. Such bit errors are introduced by signal attenuation and electromagnetic noise. Because there is no need to forward a datagram that has an error, many link-layer protocols provide a mechanism to detect such bit errors. This is done by having the transmitting node include error-detection bits in the frame, and having the receiving node perform an error check. Recall from Chapters 3 and 4 that the Internet's transport layer and network layer also provide a limited form of error detection—the Internet checksum. Error detection in the link layer is usually more sophisticated and is implemented in hardware. Error correction is similar to error detection, except that a receiver not only detects when bit errors have occurred in the frame but also determines exactly where in the frame the errors have occurred (and then corrects these errors). 6.1.2 Where Is the Link Layer Implemented? Before diving into our detailed study of the link layer, let's conclude this introduction by considering the question of where the link layer is implemented. We'll focus here on an end system, since we learned in Chapter 4 that the link layer is implemented in a router's line card. Is a host's link layer implemented in hardware or software? Is it implemented on a separate card or chip, and how does it interface with the rest of a host's hardware and operating system components?

Figure 6.2 shows a typical host architecture. For the most part, the link layer is implemented in a network adapter, also sometimes known as a network interface card (NIC). At the heart of the network adapter is the link-layer controller, usually a single, special-purpose chip that implements many of the link-layer services (framing, link access, error detection, and so on). Thus, much of a link-layer controller's functionality is implemented in hardware. For example, Intel's 710 adapter [Intel 2016] implements the Ethernet protocols we'll study in Section 6.5; the Atheros AR5006 [Atheros 2016] controller implements the 802.11 WiFi protocols we'll study in Chapter 7. Until the late 1990s, most network adapters were physically separate cards (such as a PCMCIA card or a plug-in card fitting into a PC's PCI card slot) but increasingly, network adapters are being integrated onto the host's motherboard —a so-called LAN-on-motherboard configuration. On the sending side, the controller takes a datagram that has been created and stored in host memory by the higher layers of the protocol stack, encapsulates the datagram in a link-layer frame (filling in the frame's various fields), and then transmits the frame into the communication link, following the linkaccess protocol. On the receiving side, a controller receives the entire frame, and extracts the networklayer datagram. If the link layer performs error detection, then it is the sending controller that sets the error-detection bits in the frame header and it is the receiving controller that performs error detection. Figure 6.2 shows a network adapter attaching to a host's bus (e.g., a PCI or PCI-X bus), where it looks much like any other I/O device to the other host Figure 6.2 Network adapter: Its relationship to other host components and to protocol stack functionality components. Figure 6.2 also shows that while most of the link layer is implemented in hardware, part of the link layer is implemented in software that runs on the host's CPU. The software components of the link layer implement higher-level link-layer functionality such as assembling link-layer addressing information and activating the controller hardware. On the receiving side, link-layer software responds to controller interrupts (e.g., due to the receipt of one or more frames), handling error conditions and passing a datagram up to the network layer. Thus, the link layer is a combination of hardware and software—the place in the protocol stack where software meets hardware. [Intel 2016] provides a readable overview (as well as a detailed description) of the XL710 controller from a softwareprogramming point of view. 6.2 Error-Detection and -Correction Techniques In the previous section, we noted that bit-level error detection and correction—detecting and correcting the corruption of bits in a link-layer frame sent from one node to another physically connected neighboring node—are two services often provided by the link layer. We saw in Chapter 3 that errordetection and -correction services are also often offered at the transport layer as well. In this section, we'll examine a few of the simplest techniques that can be used to detect and, in some cases, correct such bit errors. A full treatment of the theory and implementation of this topic is itself the topic of many textbooks (for example, [Schwartz 1980] or [Bertsekas 1991]), and our treatment here is necessarily brief. Our goal here is to develop an intuitive feel for the capabilities that error-detection and -correction techniques provide and to see how a few simple techniques work and are used in practice in the link layer. Figure 6.3 illustrates the setting for our study. At the sending node, data, D, to be protected against bit errors is augmented with error-detection and -correction bits (EDC). Typically, the data to be protected includes not only the datagram passed down from the network layer for transmission across the link, but also link-level addressing information, sequence numbers, and other fields in the link frame header. Both D and EDC are sent to the receiving node in a link-level frame. At the receiving node, a sequence of bits, D' and EDC' is received. Note that D' and EDC' may differ from the original D and EDC as a result of in-transit bit flips. The receiver's challenge is to determine whether or not D' is the same as the original D, given that it has only received D' and EDC'. The exact wording of the receiver's decision in Figure 6.3 (we ask whether an error is detected, not whether an error has occurred!) is important. Error-detection and -correction techniques allow the receiver to sometimes, but not always, detect that bit errors have occurred. Even with the use of error-detection bits there still may be undetected bit errors; that is, the receiver may be unaware that the received information contains bit errors. As a Figure 6.3 Error-detection and -correction scenario consequence, the receiver might deliver a corrupted datagram to the network layer, or be unaware that the contents of a field in the frame's header has been corrupted. We thus want to choose an errordetection scheme that keeps the probability of such occurrences small. Generally, more sophisticated error-detection and-correction techniques (that is, those that have a smaller probability of allowing undetected bit errors) incur a larger overhead—more computation is needed to compute and transmit a larger number of error-detection and -correction bits. Let's now examine three techniques for detecting errors in the transmitted data—parity checks (to illustrate the basic ideas behind error detection and correction), checksumming methods (which are more typically used in the transport layer), and cyclic redundancy checks (which are more typically used in the link layer in an adapter). 6.2.1 Parity Checks Perhaps the simplest form of error detection is the use of a single parity bit. Suppose that the information to be sent, D in Figure 6.4, has d bits. In an even parity scheme, the sender simply includes one additional bit and chooses its value such that the total number of 1s in the bits (the original information plus a parity bit) is even. For odd parity schemes, the parity bit value is chosen such that there is an odd number of 1s. Figure 6.4 illustrates an even parity scheme, with the single parity bit being stored in a separate field. d+1 Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1s in the received bits. If an odd number of 1-valued bits are found with an even parity scheme, the receiver knows that at least one bit error has occurred. More precisely, it knows that some odd number of bit errors have occurred. But what happens if an even number of bit errors occur? You should convince yourself that this would result in an undetected error. If the probability of bit errors is small and errors can be assumed to occur independently from one bit to the next, the probability of multiple bit errors in a packet would be extremely small. In this case, a single parity bit might suffice. However, measurements have shown that, rather than occurring independently, errors are often clustered together in "bursts." Under burst error conditions, the probability of undetected errors in a frame protected by single-bit parity can approach 50 percent [Spragins 1991]. Clearly, a more robust error-detection scheme is needed (and, fortunately, is used in practice!). But before examining error-detection schemes that are used in practice, let's consider a simple Figure 6.4 One-bit even parity generalization of one-bit parity that will provide us with insight into error-correction techniques. Figure 6.5 shows a two-dimensional generalization of the single-bit parity scheme. Here, the d bits in D are divided into i rows and j columns. A parity value is computed for each row and for each column. The resulting parity bits comprise the link-layer frame's error-detection bits. Suppose now that a single bit error occurs in the original d bits of information. With this twodimensional parity scheme, the parity of both the column and the row containing the flipped bit will be in error. The receiver can thus not only detect the fact that a single bit error has occurred, but can use the column and row indices of the column and row with parity errors to actually identify the bit that was corrupted and correct that error! Figure 6.5 shows an example in which the 1-valued bit in position (2,2) is corrupted and switched to a 0—an error that is both detectable and correctable at the receiver. Although our discussion has focused on the original d bits of information, a single error in the parity bits themselves is also detectable and correctable. Two-dimensional parity can also detect (but not correct!) any combination of two errors in a packet. Other properties of the two-dimensional parity scheme are explored in the problems at the end of the chapter. d+1 i+j+1 Figure 6.5 Two-dimensional even parity The ability of the receiver to both detect and correct errors is known as forward error correction (FEC). These techniques are commonly used in audio storage and playback devices such as audio CDs. In a network setting, FEC techniques can be used by themselves, or in conjunction with link-layer ARQ techniques similar to those we examined in Chapter 3. FEC techniques are valuable because they can decrease the number of sender retransmissions required. Perhaps more important, they allow for immediate correction of errors at the receiver. This avoids having to wait for the round-trip propagation delay needed for the sender to receive a NAK packet and for the retransmitted packet to propagate back to the receiver—a potentially important advantage for real-time network applications [Rubenstein 1998] or links (such as deep-space links) with long propagation delays. Research examining the use of FEC in error-control protocols includes [Biersack 1992; Nonnenmacher 1998; Byers 1998; Shacham 1990]. 6.2.2 Checksumming Methods In checksumming techniques, the d bits of data in Figure 6.4 are treated as a sequence of k-bit integers. One simple checksumming method is to simply sum these k-bit integers and use the resulting sum as the error-detection bits. The Internet checksum is based on this approach—bytes of data are treated as 16-bit integers and summed. The 1s complement of this sum then forms the Internet checksum that is carried in the segment header. As discussed in Section 3.3, the receiver checks the checksum by taking the 1s complement of the sum of the received data (including the checksum) and checking whether the result is all 1 bits. If any of the bits are 0, an error is indicated. RFC 1071 discusses the Internet checksum algorithm and its implementation in detail. In the TCP and UDP protocols, the Internet checksum is computed over all fields (header and data fields included). In IP the checksum is computed over the IP header (since the UDP or TCP segment has its own checksum). In other protocols, for example, XTP [Strayer 1992], one checksum is computed over the header and another checksum is computed over the entire packet. Checksumming methods require relatively little packet overhead. For example, the checksums in TCP and UDP use only 16 bits. However, they provide relatively weak protection against errors as compared with cyclic redundancy check, which is discussed below and which is often used in the link layer. A natural question at this point is, Why is checksumming used at the transport layer and cyclic redundancy check used at the link layer? Recall that the transport layer is typically implemented in software in a host as part of the host's operating system. Because transport-layer error detection is implemented in software, it is important to have a simple and fast error-detection scheme such as checksumming. On the other hand, error detection at the link layer is implemented in dedicated hardware in adapters, which can rapidly perform the more complex CRC operations. Feldmeier [Feldmeier 1995] presents fast software implementation techniques for not only weighted checksum codes, but CRC (see below) and other codes as well. 6.2.3 Cyclic Redundancy Check (CRC) An error-detection technique used widely in today's computer networks is based on cyclic redundancy check (CRC) codes. CRC codes are also known as polynomial codes, since it is possible to view the bit string to be sent as a polynomial whose coefficients are the 0 and 1 values in the bit string, with operations on the bit string interpreted as polynomial arithmetic. CRC codes operate as follows. Consider the d-bit piece of data, D, that the sending node wants to send to the receiving node. The sender and receiver must first agree on an bit pattern, known as a generator, which we will denote as G. We will require that the most significant (leftmost) bit of G be a 1. The key idea behind CRC codes is shown in Figure 6.6. For a given piece of data, D, the sender will choose r additional bits, R, and append them to D such that the resulting bit pattern (interpreted as a binary number) is exactly divisible by G (i.e., has no remainder) using modulo-2 arithmetic. The process of error checking with CRCs is thus simple: The receiver divides the received bits by G. If the remainder is nonzero, the receiver knows that an error has occurred; otherwise the data is accepted as being correct. r+1 d+r d+r All CRC calculations are done in modulo-2 arithmetic without carries in addition or borrows in subtraction. This means that addition and subtraction are identical, and both are equivalent to the bitwise exclusive-or (XOR) of the operands. Thus, for example, 1011 XOR 0101 = 1110 1001 XOR 1101 = 0100 Also, we similarly have 1011 - 0101 = 1110 1001 - 1101 = 0100 Multiplication and division are the same as in base-2 arithmetic, except that any required addition or subtraction is done without carries or borrows. As in regular Figure 6.6 CRC binary arithmetic, multiplication by 2 left shifts a bit pattern by k places. Thus, given D and R, the quantity yields the bit pattern shown in Figure 6.6. We'll use this algebraic characterization of the bit pattern from Figure 6.6 in our discussion below. Let us now turn to the crucial question of how the sender computes R. Recall that we want to find R such that there is an n such that That is, we want to choose R such that G divides into without remainder. If we XOR (that is, add modulo-2, without carry) R to both sides of the above equation, we get k D·2rXOR R d+r d+r D·2rXOR R=nG D·2rXOR R This equation tells us that if we divide by G, the value of the remainder is precisely R. In other words, we can calculate R as Figure 6.7 illustrates this calculation for the case of and . The 9 bits transmitted in this case are 101 110 011. You should check these calculations for yourself and also check that indeed . Figure 6.7 A sample CRC calculation International standards have been defined for 8-, 12-, 16-, and 32-bit generators, G. The CRC-32 32-bit standard, which has been adopted in a number of link-level IEEE protocols, uses a generator of Each of the CRC standards can detect burst errors of fewer than bits. (This means that all consecutive bit errors of r bits or fewer will be detected.) Furthermore, under appropriate assumptions, a burst of length greater than bits is detected with probability . Also, each of the CRC standards can detect any odd number of bit errors. See [Williams 1993] for a discussion of implementing CRC checks. The theory behind CRC codes and even more powerful codes is beyond the scope of this text. The text [Schwartz 1980] provides an excellent introduction to this topic. D·2r=nG XOR R D·2r R=remainderD·2rG D=101110, d=6, G=1001, r=3 D·2r=101011·G XOR R GCRC-32=100000100110000010001110110110111 r+1 r+1 1−0.5r 6.3 Multiple Access Links and Protocols In the introduction to this chapter, we noted that there are two types of network links: point-to-point links and broadcast links. A point-to-point link consists of a single sender at one end of the link and a single receiver at the other end of the link. Many link-layer protocols have been designed for point-to-point links; the point-to-point protocol (PPP) and high-level data link control (HDLC) are two such protocols. The second type of link, a broadcast link, can have multiple sending and receiving nodes all connected to the same, single, shared broadcast channel. The term broadcast is used here because when any one node transmits a frame, the channel broadcasts the frame and each of the other nodes receives a copy. Ethernet and wireless LANs are examples of broadcast link-layer technologies. In this section we'll take a step back from specific link-layer protocols and first examine a problem of central importance to the link layer: how to coordinate the access of multiple sending and receiving nodes to a shared broadcast channel—the multiple access problem. Broadcast channels are often used in LANs, networks that are geographically concentrated in a single building (or on a corporate or university campus). Thus, we'll look at how multiple access channels are used in LANs at the end of this section. We are all familiar with the notion of broadcasting—television has been using it since its invention. But traditional television is a one-way broadcast (that is, one fixed node transmitting to many receiving nodes), while nodes on a computer network broadcast channel can both send and receive. Perhaps a more apt human analogy for a broadcast channel is a cocktail party, where many people gather in a large room (the air providing the broadcast medium) to talk and listen. A second good analogy is something many readers will be familiar with—a classroom—where teacher(s) and student(s) similarly share the same, single, broadcast medium. A central problem in both scenarios is that of determining who gets to talk (that is, transmit into the channel) and when. As humans, we've evolved an elaborate set of protocols for sharing the broadcast channel: "Give everyone a chance to speak." "Don't speak until you are spoken to." "Don't monopolize the conversation." "Raise your hand if you have a question." "Don't interrupt when someone is speaking." "Don't fall asleep when someone is talking." Computer networks similarly have protocols—so-called multiple access protocols—by which nodes regulate their transmission into the shared broadcast channel. As shown in Figure 6.8, multiple access protocols are needed in a wide variety of network settings, including both wired and wireless access networks, and satellite networks. Although technically each node accesses the broadcast channel through its adapter, in this section we will refer to the node as the sending and Figure 6.8 Various multiple access channels receiving device. In practice, hundreds or even thousands of nodes can directly communicate over a broadcast channel. Because all nodes are capable of transmitting frames, more than two nodes can transmit frames at the same time. When this happens, all of the nodes receive multiple frames at the same time; that is, the transmitted frames collide at all of the receivers. Typically, when there is a collision, none of the receiving nodes can make any sense of any of the frames that were transmitted; in a sense, the signals of the colliding frames become inextricably tangled together. Thus, all the frames involved in the collision are lost, and the broadcast channel is wasted during the collision interval. Clearly, if many nodes want to transmit frames frequently, many transmissions will result in collisions, and much of the bandwidth of the broadcast channel will be wasted. In order to ensure that the broadcast channel performs useful work when multiple nodes are active, it is necessary to somehow coordinate the transmissions of the active nodes. This coordination job is the responsibility of the multiple access protocol. Over the past 40 years, thousands of papers and hundreds of PhD dissertations have been written on multiple access protocols; a comprehensive survey of the first 20 years of this body of work is [Rom 1990]. Furthermore, active research in multiple access protocols continues due to the continued emergence of new types of links, particularly new wireless links. Over the years, dozens of multiple access protocols have been implemented in a variety of link-layer technologies. Nevertheless, we can classify just about any multiple access protocol as belonging to one of three categories: channel partitioning protocols, random access protocols, and taking-turns protocols. We'll cover these categories of multiple access protocols in the following three subsections. Let's conclude this overview by noting that, ideally, a multiple access protocol for a broadcast channel of rate R bits per second should have the following desirable characteristics: 1. When only one node has data to send, that node has a throughput of R bps. 2. When M nodes have data to send, each of these nodes has a throughput of R/M bps. This need not necessarily imply that each of the M nodes always has an instantaneous rate of R/M, but rather that each node should have an average transmission rate of R/M over some suitably defined interval of time. 3. The protocol is decentralized; that is, there is no master node that represents a single point of failure for the network. 4. The protocol is simple, so that it is inexpensive to implement. 6.3.1 Channel Partitioning Protocols Recall from our early discussion back in Section 1.3 that time-division multiplexing (TDM) and frequency-division multiplexing (FDM) are two techniques that can Figure 6.9 A four-node TDM and FDM example be used to partition a broadcast channel's bandwidth among all nodes sharing that channel. As an example, suppose the channel supports N nodes and that the transmission rate of the channel is R bps. TDM divides time into time frames and further divides each time frame into N time slots. (The TDM time frame should not be confused with the link-layer unit of data exchanged between sending and receiving adapters, which is also called a frame. In order to reduce confusion, in this subsection we'll refer to the link-layer unit of data exchanged as a packet.) Each time slot is then assigned to one of the N nodes. Whenever a node has a packet to send, it transmits the packet's bits during its assigned time slot in the revolving TDM frame. Typically, slot sizes are chosen so that a single packet can be transmitted during a slot time. Figure 6.9 shows a simple four-node TDM example. Returning to our cocktail party analogy, a TDM-regulated cocktail party would allow one partygoer to speak for a fixed period of time, then allow another partygoer to speak for the same amount of time, and so on. Once everyone had had a chance to talk, the pattern would repeat. TDM is appealing because it eliminates collisions and is perfectly fair: Each node gets a dedicated transmission rate of R/N bps during each frame time. However, it has two major drawbacks. First, a node is limited to an average rate of R/N bps even when it is the only node with packets to send. A second drawback is that a node must always wait for its turn in the transmission sequence—again, even when it is the only node with a frame to send. Imagine the partygoer who is the only one with anything to say (and imagine that this is the even rarer circumstance where everyone wants to hear what that one person has to say). Clearly, TDM would be a poor choice for a multiple access protocol for this particular party. While TDM shares the broadcast channel in time, FDM divides the R bps channel into different frequencies (each with a bandwidth of R/N) and assigns each frequency to one of the N nodes. FDM thus creates N smaller channels of R/N bps out of the single, larger R bps channel. FDM shares both the advantages and drawbacks of TDM. It avoids collisions and divides the bandwidth fairly among the N nodes. However, FDM also shares a principal disadvantage with TDM—a node is limited to a bandwidth of R/N, even when it is the only node with packets to send. A third channel partitioning protocol is code division multiple access (CDMA). While TDM and FDM assign time slots and frequencies, respectively, to the nodes, CDMA assigns a different code to each node. Each node then uses its unique code to encode the data bits it sends. If the codes are chosen carefully, CDMA networks have the wonderful property that different nodes can transmit simultaneously and yet have their respective receivers correctly receive a sender's encoded data bits (assuming the receiver knows the sender's code) in spite of interfering transmissions by other nodes. CDMA has been used in military systems for some time (due to its anti-jamming properties) and now has widespread civilian use, particularly in cellular telephony. Because CDMA's use is so tightly tied to wireless channels, we'll save our discussion of the technical details of CDMA until Chapter 7. For now, it will suffice to know that CDMA codes, like time slots in TDM and frequencies in FDM, can be allocated to the multiple access channel users. 6.3.2 Random Access Protocols The second broad class of multiple access protocols are random access protocols. In a random access protocol, a transmitting node always transmits at the full rate of the channel, namely, R bps. When there is a collision, each node involved in the collision repeatedly retransmits its frame (that is, packet) until its frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. Instead it waits a random delay before retransmitting the frame. Each node involved in a collision chooses independent random delays. Because the random delays are independently chosen, it is possible that one of the nodes will pick a delay that is sufficiently less than the delays of the other colliding nodes and will therefore be able to sneak its frame into the channel without a collision. There are dozens if not hundreds of random access protocols described in the literature [Rom 1990; Bertsekas 1991]. In this section we'll describe a few of the most commonly used random access protocols—the ALOHA protocols [Abramson 1970; Abramson 1985; Abramson 2009] and the carrier sense multiple access (CSMA) protocols [Kleinrock 1975b]. Ethernet [Metcalfe 1976] is a popular and widely deployed CSMA protocol. Slotted ALOHA Let's begin our study of random access protocols with one of the simplest random access protocols, the slotted ALOHA protocol. In our description of slotted ALOHA, we assume the following: All frames consist of exactly L bits. Time is divided into slots of size L/R seconds (that is, a slot equals the time to transmit one frame). Nodes start to transmit frames only at the beginnings of slots. The nodes are synchronized so that each node knows when the slots begin. If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends. Let p be a probability, that is, a number between 0 and 1. The operation of slotted ALOHA in each node is simple: When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot. If there isn't a collision, the node has successfully transmitted its frame and thus need not consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.) If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability p until the frame is transmitted without a collision. By retransmitting with probability p, we mean that the node effectively tosses a biased coin; the event heads corresponds to "retransmit," which occurs with probability p. The event tails corresponds to "skip the slot and toss the coin again in the next slot"; this occurs with probability . All nodes involved in the collision toss their coins independently. Slotted ALOHA would appear to have many advantages. Unlike channel partitioning, slotted ALOHA allows a node to transmit continuously at the full rate, R, when that node is the only active node. (A node is said to be active if it has frames to send.) Slotted ALOHA is also highly decentralized, because each node detects collisions and independently decides when to retransmit. (Slotted ALOHA does, however, require the slots to be synchronized in the nodes; shortly we'll discuss an unslotted version of the ALOHA protocol, as well as CSMA protocols, none of which require such synchronization.) Slotted ALOHA is also an extremely simple protocol. Slotted ALOHA works well when there is only one active node, but how efficient is it when there are multiple active nodes? There are two possible efficiency $(1-p)$ Figure 6.10 Nodes 1, 2, and 3 collide in the first slot. Node 2 finally succeeds in the fourth slot, node 1 in the eighth slot, and node 3 in the ninth slot concerns here. First, as shown in Figure 6.10, when there are multiple active nodes, a certain fraction of the slots will have collisions and will therefore be "wasted." The second concern is that another fraction of the slots will be empty because all active nodes refrain from transmitting as a result of the probabilistic transmission policy. The only "unwasted" slots will be those in which exactly one node transmits. A slot in which exactly one node transmits is said to be a successful slot. The efficiency of a slotted multiple access protocol is defined to be the long-run fraction of successful slots in the case when there are a large number of active nodes, each always having a large number of frames to send. Note that if no form of access control were used, and each node were to immediately retransmit after each collision, the efficiency would be zero. Slotted ALOHA clearly increases the efficiency beyond zero, but by how much? We now proceed to outline the derivation of the maximum efficiency of slotted ALOHA. To keep this derivation simple, let's modify the protocol a little and assume that each node attempts to transmit a frame in each slot with probability p. (That is, we assume that each node always has a frame to send and that the node transmits with probability p for a fresh frame as well as for a frame that has already suffered a collision.) Suppose there are N nodes. Then the probability that a given slot is a successful slot is the probability that one of the nodes transmits and that the remaining nodes do not transmit. The probability that a given node transmits is p; the probability that the remaining nodes do not transmit is . Therefore the probability a given node has a success is . Because there are N nodes, the probability that any one of the N nodes has a success is . Thus, when there are N active nodes, the efficiency of slotted ALOHA is . To obtain the maximum efficiency for N active nodes, we have to find the p* that maximizes this expression. (See the N−1 (1−p)N−1 p(1−p)N−1 Np(1−p)N−1 Np(1−p)N−1 homework problems for a general outline of this derivation.) And to obtain the maximum efficiency for a large number of active nodes, we take the limit of as N approaches infinity. (Again, see the homework problems.) After performing these calculations, we'll find that the maximum efficiency of the protocol is given by . That is, when a large number of nodes have many frames to transmit, then (at best) only 37 percent of the slots do useful work. Thus the effective transmission rate of the channel is not R bps but only 0.37 R bps! A similar analysis also shows that 37 percent of the slots go empty and 26 percent of slots have collisions. Imagine the poor network administrator who has purchased a 100-Mbps slotted ALOHA system, expecting to be able to use the network to transmit data among a large number of users at an aggregate rate of, say, 80 Mbps! Although the channel is capable of transmitting a given frame at the full channel rate of 100 Mbps, in the long run, the successful throughput of this channel will be less than 37 Mbps. ALOHA The slotted ALOHA protocol required that all nodes synchronize their transmissions to start at the beginning of a slot. The first ALOHA protocol [Abramson 1970] was actually an unslotted, fully decentralized protocol. In pure ALOHA, when a frame first arrives (that is, a network-layer datagram is passed down from the network layer at the sending node), the node immediately transmits the frame in its entirety into the broadcast channel. If a transmitted frame experiences a collision with one or more other transmissions, the node will then immediately (after completely transmitting its collided frame) retransmit the frame with probability p. Otherwise, the node waits for a frame transmission time. After this wait, it then transmits the frame with probability p, or waits (remaining idle) for another frame time with probability 1 – p. To determine the maximum efficiency of pure ALOHA, we focus on an individual node. We'll make the same assumptions as in our slotted ALOHA analysis and take the frame transmission time to be the unit of time. At any given time, the probability that a node is transmitting a frame is p. Suppose this frame begins transmission at time t . As shown in Figure 6.11, in order for this frame to be successfully transmitted, no other nodes can begin their transmission in the interval of time . Such a transmission would overlap with the beginning of the transmission of node i's frame. The probability that all other nodes do not begin a transmission in this interval is . Similarly, no other node can begin a transmission while node i is transmitting, as such a transmission would overlap with the latter part of node i's transmission. The probability that all other nodes do not begin a transmission in this interval is also . Thus, the probability that a given node has a successful transmission is . By taking limits as in the slotted ALOHA case, we find that the maximum efficiency of the pure ALOHA protocol is only 1/(2e)—exactly half that of slotted ALOHA. This then is the price to be paid for a fully decentralized ALOHA protocol. Np*(1−p*)N−1 1/e=0.37 0 [t0−1,t0] (1−p)N−1 (1−p)N−1 p(1−p)2(N−1) Figure 6.11 Interfering transmissions in pure ALOHA Carrier Sense Multiple Access (CSMA) In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. In our cocktail party analogy, ALOHA protocols are quite like a boorish partygoer who continues to chatter away regardless of whether other people are talking. As humans, we have human protocols that allow us not only to behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and, consequently, to increase the amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation: Listen before speaking. If someone else is speaking, wait until they are finished. In the networking world, this is called carrier sensing—a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits until it detects no transmissions for a short amount of time and then begins transmission. If someone else begins talking at the same time, stop talking. In the networking world, this is called collision detection—a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering frame, it stops transmitting and waits a random amount of time before repeating the sense-and-transmit-when-idle cycle. These two rules are embodied in the family of carrier sense multiple access (CSMA) and CSMA with collision detection (CSMA/CD) protocols [Kleinrock 1975b; Metcalfe 1976; Lam 1980; Rom 1990]. Many variations on CSMA and CASE HISTORY NORM ABRAMSON AND ALOHANET Norm Abramson, a PhD engineer, had a passion for surfing and an interest in packet switching. This combination of interests brought him to the University of Hawaii in 1969. Hawaii consists of many mountainous islands, making it difficult to install and operate land-based networks. When not surfing, Abramson thought about how to design a network that does packet switching over radio. The network he designed had one central host and several secondary nodes scattered over the Hawaiian Islands. The network had two channels, each using a different frequency band. The downlink channel broadcasted packets from the central host to the secondary hosts; and the upstream channel sent packets from the secondary hosts to the central host. In addition to sending informational packets, the central host also sent on the downstream channel an acknowledgment for each packet successfully received from the secondary hosts. Because the secondary hosts transmitted packets in a decentralized fashion, collisions on the upstream channel inevitably occurred. This observation led Abramson to devise the pure ALOHA protocol, as described in this chapter. In 1970, with continued funding from ARPA, Abramson connected his ALOHAnet to the ARPAnet. Abramson's work is important not only because it was the first example of a radio packet network, but also because it inspired Bob Metcalfe. A few years later, Metcalfe modified the ALOHA protocol to create the CSMA/CD protocol and the Ethernet LAN. CSMA/CD have been proposed. Here, we'll consider a few of the most important, and fundamental, characteristics of CSMA and CSMA/CD. The first question that you might ask about CSMA is why, if all nodes perform carrier sensing, do collisions occur in the first place? After all, a node will refrain from transmitting whenever it senses that another node is transmitting. The answer to the question can best be illustrated using space-time diagrams [Molle 1987]. Figure 6.12 shows a space-time diagram of four nodes (A, B, C, D) attached to a linear broadcast bus. The horizontal axis shows the position of each node in space; the vertical axis represents time. At time t , node B senses the channel is idle, as no other nodes are currently transmitting. Node B thus begins transmitting, with its bits propagating in both directions along the broadcast medium. The downward propagation of B's bits in Figure 6.12 with increasing time indicates that a nonzero amount of time is needed for B's bits actually to propagate (albeit at near the speed of light) along the broadcast medium. At time , node D has a frame to send. Although node B is currently transmitting at time t , the bits being transmitted by B have yet to reach D, and thus D senses 0 t1(t1>t0) 1 Figure 6.12 Space-time diagram of two CSMA nodes with colliding transmissions the channel idle at t . In accordance with the CSMA protocol, D thus begins transmitting its frame. A short time later, B's transmission begins to interfere with D's transmission at D. From Figure 6.12, it is evident that the end-to-end channel propagation delay of a broadcast channel—the time it takes for a signal to propagate from one of the nodes to another—will play a crucial role in determining its performance. The longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network. Carrier Sense Multiple Access with Collision Dection (CSMA/CD) In Figure 6.12, nodes do not perform collision detection; both B and D continue to transmit their frames in their entirety even though a collision has occurred. When a node performs collision detection, it ceases transmission as soon as it detects a collision. Figure 6.13 shows the same scenario as in Figure 6.12, except that the two 1 Figure 6.13 CSMA with collision detection nodes each abort their transmission a short time after detecting a collision. Clearly, adding collision detection to a multiple access protocol will help protocol performance by not transmitting a useless, damaged (by interference with a frame from another node) frame in its entirety. Before analyzing the CSMA/CD protocol, let us now summarize its operation from the perspective of an adapter (in a node) attached to a broadcast channel: 1. The adapter obtains a datagram from the network layer, prepares a link-layer frame, and puts the frame adapter buffer. 2. If the adapter senses that the channel is idle (that is, there is no signal energy entering the adapter from the channel), it starts to transmit the frame. If, on the other hand, the adapter senses that the channel is busy, it waits until it senses no signal energy and then starts to transmit the frame. 3. While transmitting, the adapter monitors for the presence of signal energy coming from other adapters using the broadcast channel. 4. If the adapter transmits the entire frame without detecting signal energy from other adapters, the adapter is finished with the frame. If, on the other hand, the adapter detects signal energy from other adapters while transmitting, it aborts the transmission (that is, it stops transmitting its frame). 5. After aborting, the adapter waits a random amount of time and then returns to step 2. The need to wait a random (rather than fixed) amount of time is hopefully clear—if two nodes transmitted frames at the same time and then both waited the same fixed amount of time, they'd continue colliding forever. But what is a good interval of time from which to choose the random backoff time? If the interval is large and the number of colliding nodes is small, nodes are likely to wait a large amount of time (with the channel remaining idle) before repeating the sense-and-transmit-when-idle step. On the other hand, if the interval is small and the number of colliding nodes is large, it's likely that the chosen random values will be nearly the same, and transmitting nodes will again collide. What we'd like is an interval that is short when the number of colliding nodes is small, and long when the number of colliding nodes is large. The binary exponential backoff algorithm, used in Ethernet as well as in DOCSIS cable network multiple access protocols [DOCSIS 2011], elegantly solves this problem. Specifically, when transmitting a frame that has already experienced n collisions, a node chooses the value of K at random from . Thus, the more collisions experienced by a frame, the larger the interval from which K is chosen. For Ethernet, the actual amount of time a node waits is bit times (i.e., K times the amount of time needed to send 512 bits into the Ethernet) and the maximum value that n can take is capped at 10. Let's look at an example. Suppose that a node attempts to transmit a frame for the first time and while transmitting it detects a collision. The node then chooses with probability 0.5 or chooses with probability 0.5. If the node chooses , then it immediately begins sensing the channel. If the node chooses , it waits 512 bit times (e.g., 5.12 microseconds for a 100 Mbps Ethernet) before beginning the sense-and-transmit-when-idle cycle. After a second collision, K is chosen with equal probability from {0,1,2,3}. After three collisions, K is chosen with equal probability from {0,1,2,3,4,5,6,7}. After 10 or more collisions, K is chosen with equal probability from {0,1,2,..., 1023}. Thus, the size of the sets from which K is chosen grows exponentially with the number of collisions; for this reason this algorithm is referred to as binary exponential backoff. We also note here that each time a node prepares a new frame for transmission, it runs the CSMA/CD algorithm, not taking into account any collisions that may have occurred in the recent past. So it is possible that a node with a new frame will immediately be able to sneak in a successful transmission while several other nodes are in the exponential backoff state. CSMA/CD Efficiency $\{0,1,2,\ldots 2^n-1\}$ $K \cdot 512$ $K=0$ $K=1$ $K=0$ $K=1$ When only one node has a frame to send, the node can transmit at the full channel rate (e.g., for Ethernet typical rates are 10 Mbps, 100 Mbps, or 1 Gbps). However, if many nodes have frames to transmit, the effective transmission rate of the channel can be much less. We define the efficiency of CSMA/CD to be the long-run fraction of time during which frames are being transmitted on the channel without collisions when there is a large number of active nodes, with each node having a large number of frames to send. In order to present a closed-form approximation of the efficiency of Ethernet, let d denote the maximum time it takes signal energy to propagate between any two adapters. Let d be the time to transmit a maximum-size frame (approximately 1.2 msecs for a 10 Mbps Ethernet). A derivation of the efficiency of CSMA/CD is beyond the scope of this book (see [Lam 1980] and [Bertsekas 1991]). Here we simply state the following approximation: We see from this formula that as d approaches 0, the efficiency approaches 1. This matches our intuition that if the propagation delay is zero, colliding nodes will abort immediately without wasting the channel. Also, as d becomes very large, efficiency approaches 1. This is also intuitive because when a frame grabs the channel, it will hold on to the channel for a very long time; thus, the channel will be doing productive work most of the time. 6.3.3 Taking-Turns Protocols Recall that two desirable properties of a multiple access protocol are (1) when only one node is active, the active node has a throughput of R bps, and (2) when M nodes are active, then each active node has a throughput of nearly R/M bps. The ALOHA and CSMA protocols have this first property but not the second. This has motivated researchers to create another class of protocols—the taking-turns protocols. As with random access protocols, there are dozens of taking-turns protocols, and each one of these protocols has many variations. We'll discuss two of the more important protocols here. The first one is the polling protocol. The polling protocol requires one of the nodes to be designated as a master node. The master node polls each of the nodes in a round-robin fashion. In particular, the master node first sends a message to node 1, saying that it (node 1) can transmit up to some maximum number of frames. After node 1 transmits some frames, the master node tells node 2 it (node 2) can transmit up to the maximum number of frames. (The master node can determine when a node has finished sending its frames by observing the lack of a signal on the channel.) The procedure continues in this manner, with the master node polling each of the nodes in a cyclic manner. The polling protocol eliminates the collisions and empty slots that plague random access protocols. This allows polling to achieve a much higher efficiency. But it also has a few drawbacks. The first drawback is that the protocol introduces a polling delay—the amount of time required to notify a node that it can prop trans Efficiency=11+5dprop/dtrans prop trans transmit. If, for example, only one node is active, then the node will transmit at a rate less than R bps, as the master node must poll each of the inactive nodes in turn each time the active node has sent its maximum number of frames. The second drawback, which is potentially more serious, is that if the master node fails, the entire channel becomes inoperative. The 802.15 protocol and the Bluetooth protocol we will study in Section 6.3 are examples of polling protocols. The second taking-turns protocol is the token-passing protocol. In this protocol there is no master node. A small, special-purpose frame known as a token is exchanged among the nodes in some fixed order. For example, node 1 might always send the token to node 2, node 2 might always send the token to node 3, and node N might always send the token to node 1. When a node receives a token, it holds onto the token only if it has some frames to transmit; otherwise, it immediately forwards the token to the next node. If a node does have frames to transmit when it receives the token, it sends up to a maximum number of frames and then forwards the token to the next node. Token passing is decentralized and highly efficient. But it has its problems as well. For example, the failure of one node can crash the entire channel. Or if a node accidentally neglects to release the token, then some recovery procedure must be invoked to get the token back in circulation. Over the years many token-passing protocols have been developed, including the fiber distributed data interface (FDDI) protocol [Jain 1994] and the IEEE 802.5 token ring protocol [IEEE 802.5 2012], and each one had to address these as well as other sticky issues. 6.3.4 DOCSIS: The Link-Layer Protocol for Cable Internet Access In the previous three subsections, we've learned about three broad classes of multiple access protocols: channel partitioning protocols, random access protocols, and taking turns protocols. A cable access network will make for an excellent case study here, as we'll find aspects of each of these three classes of multiple access protocols with the cable access network! Recall from Section 1.2.1 that a cable access network typically connects several thousand residential cable modems to a cable modem termination system (CMTS) at the cable network headend. The DataOver-Cable Service Interface Specifications (DOCSIS) [DOCSIS 2011] specifies the cable data network architecture and its protocols. DOCSIS uses FDM to divide the downstream (CMTS to modem) and upstream (modem to CMTS) network segments into multiple frequency channels. Each downstream channel is 6 MHz wide, with a maximum throughput of approximately 40 Mbps per channel (although this data rate is seldom seen at a cable modem in practice); each upstream channel has a maximum channel width of 6.4 MHz, and a maximum upstream throughput of approximately 30 Mbps. Each upstream and Figure 6.14 Upstream and downstream channels between CMTS and cable modems downstream channel is a broadcast channel. Frames transmitted on the downstream channel by the CMTS are received by all cable modems receiving that channel; since there is just a single CMTS transmitting into the downstream channel, however, there is no multiple access problem. The upstream direction, however, is more interesting and technically challenging, since multiple cable modems share the same upstream channel (frequency) to the CMTS, and thus collisions can potentially occur. As illustrated in Figure 6.14, each upstream channel is divided into intervals of time (TDM-like), each containing a sequence of mini-slots during which cable modems can transmit to the CMTS. The CMTS explicitly grants permission to individual cable modems to transmit during specific mini-slots. The CMTS accomplishes this by sending a control message known as a MAP message on a downstream channel to specify which cable modem (with data to send) can transmit during which mini-slot for the interval of time specified in the control message. Since mini-slots are explicitly allocated to cable modems, the CMTS can ensure there are no colliding transmissions during a mini-slot. But how does the CMTS know which cable modems have data to send in the first place? This is accomplished by having cable modems send mini-slot-request frames to the CMTS during a special set of interval mini-slots that are dedicated for this purpose, as shown in Figure 6.14. These mini-slotrequest frames are transmitted in a random access manner and so may collide with each other. A cable modem can neither sense whether the upstream channel is busy nor detect collisions. Instead, the cable modem infers that its mini-slot-request frame experienced a collision if it does not receive a response to the requested allocation in the next downstream control message. When a collision is inferred, a cable modem uses binary exponential backoff to defer the retransmission of its mini-slot-request frame to a future time slot. When there is little traffic on the upstream channel, a cable modem may actually transmit data frames during slots nominally assigned for mini-slot-request frames (and thus avoid having to wait for a mini-slot assignment). A cable access network thus serves as a terrific example of multiple access protocols in action—FDM, TDM, random access, and centrally allocated time slots all within one network! 6.4 Switched Local Area Networks Having covered broadcast networks and multiple access protocols in the previous section, let's turn our attention next to switched local networks. Figure 6.15 shows a switched local network connecting three departments, two servers and a router with four switches. Because these switches operate at the link layer, they switch link-layer frames (rather than network-layer datagrams), don't recognize network-layer addresses, and don't use routing algorithms like RIP or OSPF to determine Figure 6.15 An institutional network connected together by four switches paths through the network of layer-2 switches. Instead of using IP addresses, we will soon see that they use link-layer addresses to forward link-layer frames through the network of switches. We'll begin our study of switched

LANs by first covering link-layer addressing (Section 6.4.1). We then examine the celebrated Ethernet protocol (Section 6.5.2). After examining link-layer addressing and Ethernet, we'll look at how link-layer switches operate (Section 6.4.3), and then see (Section 6.4.4) how these switches are often used to build large-scale LANs. 6.4.1 Link-Layer Addressing and ARP Hosts and routers have link-layer addresses. Now you might find this surprising, recalling from Chapter 4 that hosts and routers have network-layer addresses as well. You might be asking, why in the world do we need to have addresses at both the network and link layers? In addition to describing the syntax and function of the link-layer addresses, in this section we hope to shed some light on why the two layers of addresses are useful and, in fact, indispensable. We'll also cover the Address Resolution Protocol (ARP), which provides a mechanism to translate IP addresses to link-layer addresses. MAC Addresses In truth, it is not hosts and routers that have link-layer addresses but rather their adapters (that is, network interfaces) that have link-layer addresses. A host or router with multiple network interfaces will thus have multiple link-layer addresses associated with it, just as it would also have multiple IP addresses associated with it. It's important to note, however, that link-layer switches do not have linklayer addresses associated with their interfaces that connect to hosts and routers. This is because the job of the link-layer switch is to carry datagrams between hosts and routers; a switch does this job transparently, that is, without the host or router having to explicitly address the frame to the intervening switch. This is illustrated in Figure 6.16. A link-layer address is variously called a LAN address, a physical address, or a MAC address. Because MAC address seems to be the most popular term, we'll henceforth refer to link-layer addresses as MAC addresses. For most LANs (including Ethernet and 802.11 wireless LANs), the MAC address is 6 bytes long, giving 2 possible MAC addresses. As shown in Figure 6.16, these 6-byte addresses are typically expressed in hexadecimal notation, with each byte of the address expressed as a pair of hexadecimal numbers. Although MAC addresses were designed to be permanent, it is now possible to change an adapter's MAC address via software. For the rest of this section, however, we'll assume that an adapter's MAC address is fixed. One interesting property of MAC addresses is that no two adapters have the same address. This might seem surprising given that adapters are manufactured in many countries by many companies. How does a company manufacturing adapters in Taiwan make sure that it is using different addresses from a company manufacturing 48 Figure 6.16 Each interface connected to a LAN has a unique MAC address adapters in Belgium? The answer is that the IEEE manages the MAC address space. In particular, when a company wants to manufacture adapters, it purchases a chunk of the address space consisting of 2 addresses for a nominal fee. IEEE allocates the chunk of 2 addresses by fixing the first 24 bits of a MAC address and letting the company create unique combinations of the last 24 bits for each adapter. An adapter's MAC address has a flat structure (as opposed to a hierarchical structure) and doesn't change no matter where the adapter goes. A laptop with an Ethernet interface always has the same MAC address, no matter where the computer goes. A smartphone with an 802.11 interface always has the same MAC address, no matter where the smartphone goes. Recall that, in contrast, IP addresses have a hierarchical structure (that is, a network part and a host part), and a host's IP addresses needs to be changed when the host moves, i.e., changes the network to which it is attached. An adapter's MAC address is analogous to a person's social security number, which also has a flat addressing structure and which doesn't change no matter where the person goes. An IP address is analogous to a person's postal address, which is hierarchical and which must be changed whenever a person moves. Just as a person may find it useful to have both a postal address and a social security number, it is useful for a host and router interfaces to have both a network-layer address and a MAC address. When an adapter wants to send a frame to some