address of 00:22:6B:45:1F:1B (the gateway router) and sends the frame to the switch. 19. The routers in the school network, Comcast's network, and Google's network forward the datagram containing the TCP SYN toward www.google.com, using the forwarding table in each router, as in steps 14–16 above. Recall that the router forwarding table entries governing forwarding of packets over the inter-domain link between the Comcast and Google networks are determined by the BGP protocol (Chapter 5). 20. Eventually, the datagram containing the TCP SYN arrives at www.google.com. The TCP SYN message is extracted from the datagram and demultiplexed to the welcome socket associated with port 80. A connection socket (Section 2.7) is created for the TCP connection between the Google HTTP server and Bob's laptop. A TCP SYNACK (Section 3.5.6) segment is generated, placed inside a datagram addressed to Bob's laptop, and finally placed inside a link-layer frame appropriate for the link connecting www.google.com to its first-hop router. 21. The datagram containing the TCP SYNACK segment is forwarded through the Google, Comcast, and school networks, eventually arriving at the Ethernet card in Bob's laptop. The datagram is demultiplexed within the operating system to the TCP socket created in step 18, which enters the connected state. 22. With the socket on Bob's laptop now (finally!) ready to send bytes to www.google.com, Bob's browser creates the HTTP GET message (Section 2.2.3) containing the URL to be fetched. The HTTP GET message is then written into the socket, with the GET message becoming the payload of a TCP segment. The TCP segment is placed in a datagram and sent and delivered to www.google.com as in steps 18–20 above. 23. The HTTP server at www.google.com reads the HTTP GET message from the TCP socket, creates an HTTP response message (Section 2.2), places the requested Web page content in the body of the HTTP response message, and sends the message into the TCP socket. 24. The datagram containing the HTTP reply message is forwarded through the Google, Comcast, and school networks, and arrives at Bob's laptop. Bob's Web browser program reads the HTTP response from the socket, extracts the html for the Web page from the body of the HTTP response, and finally (finally!) displays the Web page! Our scenario above has covered a lot of networking ground! If you've understood most or all of the above example, then you've also covered a lot of ground since you first read Section 1.1, where we wrote "much of this book is concerned with computer network protocols" and you may have wondered what a protocol actually was! As detailed as the above example might seem, we've omitted a number of possible additional protocols (e.g., NAT running in the school's gateway router, wireless access to the school's network, security protocols for accessing the school network or encrypting segments or datagrams, network management protocols), and considerations (Web caching, the DNS hierarchy) that one would encounter in the public Internet. We'll cover a number of these topics and more in the second part of this book. Lastly, we note that our example above was an integrated and holistic, but also very "nuts and bolts," view of many of the protocols that we've studied in the first part of this book. The example focused more on the "how" than the "why." For a broader, more reflective view on the design of network protocols in general, see [Clark 1988, RFC 5218]. 6.8 Summary In this chapter, we've examined the link layer—its services, the principles underlying its operation, and a number of important specific protocols that use these principles in implementing link-layer services. We saw that the basic service of the link layer is to move a network-layer datagram from one node (host, switch, router, WiFi access point) to an adjacent node. We saw that all link-layer protocols operate by encapsulating a network-layer datagram within a link-layer frame before transmitting the frame over the link to the adjacent node. Beyond this common framing function, however, we learned that different link-layer protocols provide very different link access, delivery, and transmission services. These differences are due in part to the wide variety of link types over which link-layer protocols must operate. A simple point-to-point link has a single sender and receiver communicating over a single “wire.” A multiple access link is shared among many senders and receivers; consequently, the link-layer protocol for a multiple access channel has a protocol (its multiple access protocol) for coordinating link access. In the case of MPLS, the “link” connecting two adjacent nodes (for example, two IP routers that are adjacent in an IP sense—that they are next-hop IP routers toward some destination) may actually be a network in and of itself. In one sense, the idea of a network being considered as a link should not seem odd. A telephone link connecting a home modem/computer to a remote modem/router, for example, is actually a path through a sophisticated and complex telephone network. Among the principles underlying link-layer communication, we examined error-detection and -correction techniques, multiple access protocols, link-layer addressing, virtualization (VLANs), and the construction of extended switched LANs and data center networks. Much of the focus today at the link layer is on these switched networks. In the case of error detection/correction, we examined how it is possible to add additional bits to a frame’s header in order to detect, and in some cases correct, bit-flip errors that might occur when the frame is transmitted over the link. We covered simple parity and checksumming schemes, as well as the more robust cyclic redundancy check. We then moved on to the topic of multiple access protocols. We identified and studied three broad approaches for coordinating access to a broadcast channel: channel partitioning approaches (TDM, FDM), random access approaches (the ALOHA protocols and CSMA protocols), and taking-turns approaches (polling and token passing). We studied the cable access network and found that it uses many of these multiple access methods. We saw that a consequence of having multiple nodes share a single broadcast channel was the need to provide node addresses at the link layer. We learned that link-layer addresses were quite different from network-layer addresses and that, in the case of the Internet, a special protocol (ARP—the Address Resolution Protocol) is used to translate between these two forms of addressing and studied the hugely successful Ethernet protocol in detail. We then examined how nodes sharing a broadcast channel form a LAN and how multiple LANs can be connected together to form larger LANs—all without the intervention of network-layer routing to interconnect these local nodes. We also learned how multiple virtual LANs can be created on a single physical LAN infrastructure. We ended our study of the link layer by focusing on how MPLS networks provide link-layer services when they interconnect IP routers and an overview of the network designs for today’s massive data centers. We wrapped up this chapter (and indeed the first five chapters) by identifying the many protocols that are needed to fetch a simple Web page. Having covered the link layer, our journey down the protocol stack is now over! Certainly, the physical layer lies below the link layer, but the details of the physical layer are probably best left for another course (for example, in communication theory, rather than computer networking). We have, however, touched upon several aspects of the physical layer in this chapter and in Chapter 1 (our discussion of physical media in Section 1.2). We’ll consider the physical layer again when we study wireless link characteristics in the next chapter. Although our journey down the protocol stack is over, our study of computer networking is not yet at an end. In the following three chapters we cover wireless networking, network security, and multimedia networking. These four topics do not fit conveniently into any one layer; indeed, each topic crosscuts many layers. Understanding these topics (billed as advanced topics in some networking texts) thus requires a firm foundation in all layers of the protocol stack—a foundation that our study of the link layer has now completed! Homework Problems and Questions Chapter 6 Review Questions SECTIONS 6.1–6.2 SECTION 6.3 SECTION 6.4 R1. Consider the transportation analogy in Section 6.1.1 . If the passenger is analagous to a datagram, what is analogous to the link layer frame? R2. If all the links in the Internet were to provide reliable delivery service, would the TCP reliable delivery service be redundant? Why or why not? R3. What are some of the possible services that a link-layer protocol can offer to the network layer? Which of these link-layer services have corresponding services in IP? In TCP? R4. Suppose two nodes start to transmit at the same time a packet of length L over a broadcast channel of rate R. Denote the propagation delay between the two nodes as d . Will there be a collision if ? Why or why not? R5. In Section 6.3 , we listed four desirable characteristics of a broadcast channel. Which of these characteristics does slotted ALOHA have? Which of these characteristics does token passing have? R6. In CSMA/CD, after the fifth collision, what is the probability that a node chooses ? The result corresponds to a delay of how many seconds on a 10 Mbps Ethernet? R7. Describe polling and token-passing protocols using the analogy of cocktail party interactions. R8. Why would the token-ring protocol be inefficient if a LAN had a very large perimeter? prop dprop 1023 80 any allow outside of 222.22/16 222.22/16 TCP 80 > 1023 ACK allow 222.22/16 outside of 222.22/16 UDP > 1023 53 — allow outside of 222.22/16 222.22/16 UDP 53 > 1023 — deny all all all all all all Recall from Section 3.5 that the first segment in every TCP connection has the ACK bit set to 0, whereas all the other segments in the connection have the ACK bit set to 1. Thus, if an organization wants to prevent external clients from initiating connections to internal servers, it simply filters all incoming segments with the ACK bit set to 0. This policy kills all TCP connections originating from the outside, but permits connections originating internally. Firewall rules are implemented in routers with access control lists, with each router interface having its own list. An example of an access control list for an organization 222.22/16 is shown in Table 8.6. This access control list is for an interface that connects the router to the organization's external ISPs. Rules are applied to each datagram that passes through the interface from top to bottom. The first two rules together allow internal users to surf the Web: The first rule allows any TCP packet with destination port 80 to leave the organization's network; the second rule allows any TCP packet with source port 80 and the ACK bit set to enter the organization's network. Note that if an external source attempts to establish a TCP connection with an internal host, the connection will be blocked, even if the source or destination port is 80. The second two rules together allow DNS packets to enter and leave the organization's network. In summary, this rather restrictive access control list blocks all traffic except Web traffic initiated from within the organization and DNS traffic. [CERT Filtering 2012] provides a list of recommended port/protocol packet filterings to avoid a number of well-known security holes in existing network applications. Stateful Packet Filters In a traditional packet filter, filtering decisions are made on each packet in isolation. Stateful filters actually track TCP connections, and use this knowledge to make filtering decisions. Table 8.7 Connection table for stateful filter source address dest address source port dest port 222.22.1.7 37.96.87.123 12699 80 222.22.93.2 199.1.205.23 37654 80 222.22.65.143 203.77.240.43 48712 80 To understand stateful filters, let's reexamine the access control list in Table 8.6. Although rather restrictive, the access control list in Table 8.6 nevertheless allows any packet arriving from the outside with ACK = 1 and source port 80 to get through the filter. Such packets could be used by attackers in attempts to crash internal systems with malformed packets, carry out denial-of-service attacks, or map the internal network. The naive solution is to block TCP ACK packets as well, but such an approach would prevent the organization's internal users from surfing the Web. Stateful filters solve this problem by tracking all ongoing TCP connections in a connection table. This is possible because the firewall can observe the beginning of a new connection by observing a three-way handshake (SYN, SYNACK, and ACK); and it can observe the end of a connection when it sees a FIN packet for the connection. The firewall can also (conservatively) assume that the connection is over when it hasn't seen any activity over the connection for, say, 60 seconds. An example connection table for a firewall is shown in Table 8.7. This connection table indicates that there are currently three ongoing TCP connections, all of which have been initiated from within the organization. Additionally, the stateful filter includes a new column, “check connection,” in its access control list, as shown in Table 8.8. Note that Table 8.8 is identical to the access control list in Table 8.6, except now it indicates that the connection should be checked for two of the rules. Let’s walk through some examples to see how the connection table and the extended access control list work hand-in-hand. Suppose an attacker attempts to send a malformed packet into the organization’s network by sending a datagram with TCP source port 80 and with the ACK flag set. Further suppose that this packet has source port number 12543 and source IP address 150.23.23.155. When this packet reaches the firewall, the firewall checks the access control list in Table 8.7, which indicates that the connection table must also be checked before permitting this packet to enter the organization’s network. The firewall duly checks the connection table, sees that this packet is not part of an ongoing TCP connection, and rejects the packet. As a second example, suppose that an internal user wants to surf an external Web site. Because this user first sends a TCP SYN segment, the user’s TCP connection gets recorded in the connection table. When Table 8.8 Access control list for stateful filter action source address dest address protocol source port dest port flag bit check conxion allow 222.22/16 outside of 222.22/16 TCP > 1023 80 any allow outside of 222.22/16 222.22/16 TCP 80 > 1023 ACK X allow 222.22/16 outside of 222.22/16 UDP > 1023 53 — allow outside of 222.22/16 222.22/16 UDP 53 > 1023 — X deny all all all all all all the Web server sends back packets (with the ACK bit necessarily set), the firewall checks the table and sees that a corresponding connection is in progress. The firewall will thus let these packets pass, thereby not interfering with the internal user’s Web surfing activity. Application Gateway In the examples above, we have seen that packet-level filtering allows an organization to perform coarse-grain filtering on the basis of the contents of IP and TCP/UDP headers, including IP addresses, port numbers, and acknowledgment bits. But what if an organization wants to provide a Telnet service to a restricted set of internal users (as opposed to IP addresses)? And what if the organization wants such privileged users to authenticate themselves first before being allowed to create Telnet sessions to the outside world? Such tasks are beyond the capabilities of traditional and stateful filters. Indeed, information about the identity of the internal users is application-layer data and is not included in the IP/TCP/UDP headers. To have finer-level security, firewalls must combine packet filters with application gateways. Application gateways look beyond the IP/TCP/UDP headers and make policy decisions based on application data. An application gateway is an application-specific server through which all application data (inbound and outbound) must pass. Multiple application gateways can run on the same host, but each gateway is a separate server with its own processes. To get some insight into application gateways, let’s design a firewall that allows only a restricted set of internal users to Telnet outside and prevents all external clients from Telneting inside. Such a policy can be accomplished by implementing Figure 8.34 Firewall consisting of an application gateway and a filter a combination of a packet filter (in a router) and a Telnet application gateway, as shown in Figure 8.34. The router’s filter is configured to block all Telnet connections except those that originate from the IP address of the application gateway. Such a filter configuration forces all outbound Telnet connections to pass through the application gateway. Consider now an internal user who wants to Telnet to the outside world. The user must first set up a Telnet session with the application gateway. An application running in the gateway, which listens for incoming Telnet sessions, prompts the user for a user ID and password. When the user supplies this information, the application gateway checks to see if the user has permission to Telnet to the outside world. If not, the Telnet connection from the internal user to the gateway is terminated by the gateway. If the user has permission, then the gateway (1) prompts the user for the host name of the external host to which the user wants to connect, (2) sets up a Telnet session between the gateway and the external host, and (3) relays to the external host all data arriving from the user, and relays to the user all data arriving from the external host. Thus, the Telnet application gateway not only performs user authorization but also acts as a Telnet server and a Telnet client, relaying information between the user and the remote Telnet server. Note that the filter will permit step 2 because the gateway initiates the Telnet connection to the outside world. CASE HISTORY ANONYMITY AND PRIVACY Suppose you want to visit a controversial Web site (for example, a political activist site) and you (1) don't want to reveal your IP address to the Web site, (2) don't want your local ISP (which may be your home or office ISP) to know that you are visiting the site, and (3) don't want your local ISP to see the data you are exchanging with the site. If you use the traditional approach of connecting directly to the Web site without any encryption, you fail on all three counts. Even if you use SSL, you fail on the first two counts: Your source IP address is presented to the Web site in every datagram you send; and the destination address of every packet you send can easily be sniffed by your local ISP. To obtain privacy and anonymity, you can instead use a combination of a trusted proxy server and SSL, as shown in Figure 8.35. With this approach, you first make an SSL connection to the trusted proxy. You then send, into this SSL connection, an HTTP request for a page at the desired site. When the proxy receives the SSL-encrypted HTTP request, it decrypts the request and forwards the cleartext HTTP request to the Web site. The Web site then responds to the proxy, which in turn forwards the response to you over SSL. Because the Web site only sees the IP address of the proxy, and not of your client's address, you are indeed obtaining anonymous access to the Web site. And because all traffic between you and the proxy is encrypted, your local ISP cannot invade your privacy by logging the site you visited or recording the data you are exchanging. Many companies today (such as proxify .com) make available such proxy services. Of course, in this solution, your proxy knows everything: It knows your IP address and the IP address of the site you're surfing; and it can see all the traffic in cleartext exchanged between you and the Web site. Such a solution, therefore, is only as good as the trustworthiness of the proxy. A more robust approach, taken by the TOR anonymizing and privacy service, is to route your traffic through a series of non-colluding proxy servers [TOR 2016]. In particular, TOR allows independent individuals to contribute proxies to its proxy pool. When a user connects to a server using TOR, TOR randomly chooses (from its proxy pool) a chain of three proxies and routes all traffic between client and server over the chain. In this manner, assuming the proxies do not collude, no one knows that communication took place between your IP address and the target Web site. Furthermore, although cleartext is sent between the last proxy and the server, the last proxy doesn't know what IP address is sending and receiving the cleartext. Figure 8.35 Providing anonymity and privacy with a proxy Internal networks often have multiple application gateways, for example, gateways for Telnet, HTTP, FTP, and e-mail. In fact, an organization's mail server (see Section 2.3) and Web cache are application gateways. Application gateways do not come without their disadvantages. First, a different application gateway is needed for each application. Second, there is a performance penalty to be paid, since all data will be relayed via the gateway. This becomes a concern particularly when multiple users or applications are using the same gateway machine. Finally, the client software must know how to contact the gateway when the user makes a request, and must know how to tell the application gateway what external server to connect to. 8.9.2 Intrusion Detection Systems We've just seen that a packet filter (traditional and stateful) inspects IP, TCP, UDP, and ICMP header fields when deciding which packets to let pass through the firewall. However, to detect many attack types, we need to perform deep packet inspection, that is, look beyond the header fields and into the actual application data that the packets carry. As we saw in Section 8.9.1, application gateways often do deep packet inspection. But an application gateway only does this for a specific application. Clearly, there is a niche for yet another device—a device that not only examines the headers of all packets passing through it (like a packet filter), but also performs deep packet inspection (unlike a packet filter). When such a device observes a suspicious packet, or a suspicious series of packets, it could prevent those packets from entering the organizational network. Or, because the activity is only deemed as suspicious, the device could let the packets pass, but send alerts to a network administrator, who can then take a closer look at the traffic and take appropriate actions. A device that generates alerts when it observes potentially malicious traffic is called an intrusion detection system (IDS). A device that filters out suspicious traffic is called an intrusion prevention system (IPS). In this section we study both systems—IDS and IPS—together, since the most interesting technical aspect of these systems is how they detect suspicious traffic (and not whether they send alerts or drop packets). We will henceforth collectively refer to IDS systems and IPS systems as IDS systems. An IDS can be used to detect a wide range of attacks, including network mapping (emanating, for example, from nmap), port scans, TCP stack scans, DoS bandwidth-flooding attacks, worms and viruses, OS vulnerability attacks, and application vulnerability attacks. (See Section 1.6 for a survey of network attacks.) Today, thousands of organizations employ IDS systems. Many of these deployed systems are proprietary, marketed by Cisco, Check Point, and other security equipment vendors. But many of the deployed IDS systems are public-domain systems, such as the immensely popular Snort IDS system (which we'll discuss shortly). An organization may deploy one or more IDS sensors in its organizational network. Figure 8.36 shows an organization that has three IDS sensors. When multiple sensors are deployed, they typically work in concert, sending information about Figure 8.36 An organization deploying a filter, an application gateway, and IDS sensors suspicious traffic activity to a central IDS processor, which collects and integrates the information and sends alarms to network administrators when deemed appropriate. In Figure 8.36, the organization has partitioned its network into two regions: a high-security region, protected by a packet filter and an application gateway and monitored by IDS sensors; and a lower-security region—referred to as the demilitarized zone (DMZ)—which is protected only by the packet filter, but also monitored by IDS sensors. Note that the DMZ includes the organization's servers that need to communicate with the outside world, such as its public Web server and its authoritative DNS server. You may be wondering at this stage, why multiple IDS sensors? Why not just place one IDS sensor just behind the packet filter (or even integrated with the packet filter) in Figure 8.36? We will soon see that an IDS not only needs to do deep packet inspection, but must also compare each passing packet with tens of thousands of "signatures"; this can be a significant amount of processing, particularly if the organization receives gigabits/sec of traffic from the Internet. By placing the IDS sensors further downstream, each sensor sees only a fraction of the organization's traffic, and can more easily keep up. Nevertheless, high-performance IDS and IPS systems are available today, and many organizations can actually get by with just one sensor located near its access router. IDS systems are broadly classified as either signature-based systems or anomaly-based systems. A signature-based IDS maintains an extensive database of attack signatures. Each signature is a set of rules pertaining to an intrusion activity. A signature may simply be a list of characteristics about a single packet (e.g., source and destination port numbers, protocol type, and a specific string of bits in the packet payload), or may relate to a series of packets. The signatures are normally created by skilled network security engineers who research known attacks. An organization's network administrator can customize the signatures or add its own to the database. Operationally, a signature-based IDS sniffs every packet passing by it, comparing each sniffed packet with the signatures in its database. If a packet (or series of packets) matches a signature in the database, the IDS generates an alert. The alert could be sent to the network administrator in an e-mail message, could be sent to the network management system, or could simply be logged for future inspection. Signature-based IDS systems, although widely deployed, have a number of limitations. Most importantly, they require previous knowledge of the attack to generate an accurate signature. In other words, a signature-based IDS is completely blind to new attacks that have yet to be recorded. Another disadvantage is that even if a signature is matched, it may not be the result of an attack, so that a false alarm is generated. Finally, because every packet must be compared with an extensive collection of signatures, the IDS can become overwhelmed with processing and actually fail to detect many malicious packets. An anomaly-based IDS creates a traffic profile as it observes traffic in normal operation. It then looks for packet streams that are statistically unusual, for example, an inordinate percentage of ICMP packets or a sudden exponential growth in port scans and ping sweeps. The great thing about anomaly-based IDS systems is that they don't rely on previous knowledge about existing attacks—that is, they can potentially detect new, undocumented attacks. On the other hand, it is an extremely challenging problem to distinguish between normal traffic and statistically unusual traffic. To date, most IDS deployments are primarily signature-based, although some include some anomaly-based features. Snort Snort is a public-domain, open source IDS with hundreds of thousands of existing deployments [Snort 2012; Koziol 2003]. It can run on Linux, UNIX, and Windows platforms. It uses the generic sniffing interface libpcap, which is also used by Wireshark and many other packet sniffers. It can easily handle 100 Mbps of traffic; for installations with gibabit/sec traffic rates, multiple Snort sensors may be needed. To gain some insight into Snort, let's take a look at an example of a Snort signature: alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING NMAP"; dsize: 0; itype: 8;) This signature is matched by any ICMP packet that enters the organization's network ($HOME_NET) from the outside ($EXTERNAL_NET), is of type 8 (ICMP ping), and has an empty payload (dsize = 0). Since nmap (see Section 1.6) generates ping packets with these specific characteristics, this signature is designed to detect nmap ping sweeps. When a packet matches this signature, Snort generates an alert that includes the message "ICMP PING NMAP" . Perhaps what is most impressive about Snort is the vast community of users and security experts that maintain its signature database. Typically within a few hours of a new attack, the Snort community writes and releases an attack signature, which is then downloaded by the hundreds of thousands of Snort deployments distributed around the world. Moreover, using the Snort signature syntax, network administrators can tailor the signatures to their own organization's needs by either modifying existing signatures or creating entirely new ones. 8.10 Summary In this chapter, we've examined the various mechanisms that our secret lovers, Bob and Alice, can use to communicate securely. We've seen that Bob and Alice are interested in confidentiality (so they alone are able to understand the contents of a transmitted message), end-point authentication (so they are sure that they are talking with each other), and message integrity (so they are sure that their messages are not altered in transit). Of course, the need for secure communication is not confined to secret lovers. Indeed, we saw in Sections 8.5 through 8.8 that security can be used in various layers in a network architecture to protect against bad guys who have a large arsenal of possible attacks at hand. The first part of this chapter presented various principles underlying secure communication. In Section 8.2, we covered cryptographic techniques for encrypting and decrypting data, including symmetric key cryptography and public key cryptography. DES and RSA were examined as specific case studies of these two major classes of cryptographic techniques in use in today's networks. In Section 8.3, we examined two approaches for providing message integrity: message authentication codes (MACs) and digital signatures. The two approaches have a number of parallels. Both use cryptographic hash functions and both techniques enable us to verify the source of the message as well as the integrity of the message itself. One important difference is that MACs do not rely on encryption whereas digital signatures require a public key infrastructure. Both techniques are extensively used in practice, as we saw in Sections 8.5 through 8.8. Furthermore, digital signatures are used to create digital certificates, which are important for verifying the validity of public keys. In Section 8.4, we examined endpoint authentication and introduced nonces to defend against the replay attack. In Sections 8.5 through 8.8 we examined several security networking protocols that enjoy extensive use in practice. We saw that symmetric key cryptography is at the core of PGP, SSL, IPsec, and wireless security. We saw that public key cryptography is crucial for both PGP and SSL. We saw that PGP uses digital signatures for message integrity, whereas SSL and IPsec use MACs. Having now an understanding of the basic principles of cryptography, and having studied how these principles are actually used, you are now in position to design your own secure network protocols! Armed with the techniques covered in Sections 8.2 through 8.8, Bob and Alice can communicate securely. (One can only hope that they are networking students who have learned this material and can thus avoid having their tryst uncovered by Trudy!) But confidentiality is only a small part of the network security picture. As we learned in Section 8.9, increasingly, the focus in network security has been on securing the network infrastructure against a potential onslaught by the bad guys. In the latter part of this chapter, we thus covered firewalls and IDS systems which inspect packets entering and leaving an organization's network. This chapter has covered a lot of ground, while focusing on the most important topics in modern network security. Readers who desire to dig deeper are encouraged to investigate the references cited in this chapter. In particular, we recommend [Skoudis 2006] for attacks and operational security, [Kaufman 1995] for cryptography and how it applies to network security, [Rescorla 2001] for an in-depth but readable treatment of SSL, and [Edney 2003] for a thorough discussion of 802.11 security, including an insightful investigation into WEP and its flaws. Homework Problems and Questions Chapter 8 Review Problems SECTION 8.1 SECTION 8.2 SECTIONS 8.3–8.4 R1. What are the differences between message confidentiality and message integrity? Can you have confidentiality without integrity? Can you have integrity without confidentiality? Justify your answer. R2. Internet entities (routers, switches, DNS servers, Web servers, user end systems, and so on) often need to communicate securely. Give three specific example pairs of Internet entities that may want secure communication. R3. From a service perspective, what is an important difference between a symmetric-key system and a public-key system? R4. Suppose that an intruder has an encrypted message as well as the decrypted version of that message. Can the intruder mount a ciphertext-only attack, a known-plaintext attack, or a chosenplaintext attack? R5. Consider an 8-block cipher. How many possible input blocks does this cipher have? How many possible mappings are there? If we view each mapping as a key, then how many possible keys does this cipher have? R6. Suppose N people want to communicate with each of other people using symmetric key encryption. All communication between any two people, i and j, is visible to all other people in this group of N, and no other person in this group should be able to decode their communication. How many keys are required in the system as a whole? Now suppose that public key encryption is used. How many keys are required in this case? R7. Suppose , and Use an identity of modular arithmetic to calculate in your head . R8. Suppose you want to encrypt the message 10101111 by encrypting the decimal number that corresponds to the message. What is the decimal number? $N-1$ $n=10{,}000$, $a=10{,}023$ $b=10{,}004$. $(a\cdot b) \bmod n$ SECTIONS 8.5–8.8 R9. In what way does a hash provide a better message integrity check than a checksum (such as the Internet checksum)? R10. Can you "decrypt" a hash of a message to get the original message? Explain your answer. R11. Consider a variation of the MAC algorithm (Figure 8.9) where the sender sends where is the concatenation of H(m) and s. Is this variation flawed? Why or why not? R12. What does it mean for a signed document to be verifiable and nonforgeable? R13. In what way does the public-key encrypted message hash provide a better digital signature than the public-key encrypted message? R14. Suppose certifier.com creates a certificate for foo.com. Typically, the entire certificate would be encrypted with certifier.com's public key. True or false? R15. Suppose Alice has a message that she is ready to send to anyone who asks. Thousands of people want to obtain Alice's message, but each wants to be sure of the integrity of the message. In this context, do you think a MAC-based or a digital-signature-based integrity scheme is more suitable? Why? R16. What is the purpose of a nonce in an end-point authentication protocol? R17. What does it mean to say that a nonce is a once-in-a-lifetime value? In whose lifetime? R18. Is the message integrity scheme based on HMAC susceptible to playback attacks? If so, how can a nonce be incorporated into the scheme to remove this susceptibility? (m, H(m)+s), H(m)+s R19. Suppose that Bob receives a PGP message from Alice. How does Bob know for sure that Alice created the message (rather than, say, Trudy)? Does PGP use a MAC for message integrity? R20. In the SSL record, there is a field for SSL sequence numbers. True or false? R21. What is the purpose of the random nonces in the SSL handshake? R22. Suppose an SSL session employs a block cipher with CBC. True or false: The server sends to the client the IV in the clear. R23. Suppose Bob initiates a TCP connection to Trudy who is pretending to be Alice. During the handshake, Trudy sends Bob Alice's certificate. In what step of the SSL handshake algorithm will Bob discover that he is not communicating with Alice? R24. Consider sending a stream of packets from Host A to Host B using IPsec. Typically, a new SA will be established for each packet sent in the stream. True or false? R25. Suppose that TCP is being run over IPsec between headquarters and the branch office in Figure 8.28 . If TCP retransmits the same packet, then the two corresponding packets sent by R1 packets will have the same sequence number in the ESP header. True or false? R26. An IKE SA and an IPsec SA are the same thing. True or false? R27. Consider WEP for 802.11. Suppose that the data is 10101100 and the keystream is 1111000. What is the resulting ciphertext? SECTION 8.9 Problems R28. In WEP, an IV is sent in the clear in every frame. True or false? R29. Stateful packet filters maintain two data structures. Name them and briefly describe what they do. R30. Consider a traditional (stateless) packet filter. This packet filter may filter packets based on TCP flag bits as well as other header fields. True or false? R31. In a traditional packet filter, each interface can have its own access control list. True or false? R32. Why must an application gateway work in conjunction with a router filter to be effective? R33. Signature-based IDSs and IPSs inspect into the payloads of TCP and UDP segments. True or false? P1. Using the monoalphabetic cipher in Figure 8.3 , encode the message "This is an easy problem." Decode the message "rmij'u uamu xyj." P2. Show that Trudy's known-plaintext attack, in which she knows the (ciphertext, plaintext) translation pairs for seven letters, reduces the number of possible substitutions to be checked in the example in Section 8.2.1 by approximately 109. P3. Consider the polyalphabetic system shown in Figure 8.4 . Will a chosen-plaintext attack that is able to get the plaintext encoding of the message "The quick brown fox jumps over the lazy dog." be sufficient to decode all messages? Why or why not? P4. Consider the block cipher in Figure 8.5 . Suppose that each block cipher T simply reverses the order of the eight input bits (so that, for example, 11110000 becomes 00001111). Further suppose that the 64-bit scrambler does not modify any bits (so that the output value of the mth bit is equal to the input value of the mth bit). (a) With and the original 64-bit input equal to 10100000 repeated eight times, what is the value of the output? (b) Repeat part (a) but now change the last bit of the original 64-bit input from a 0 to a 1. (c) Repeat parts (a) and (b) but now suppose that the 64-bit scrambler inverses the order of the 64 bits. P5. Consider the block cipher in Figure 8.5 . For a given "key"

Alice and Bob would need to keep eight tables, each 8 bits by 8 bits. For Alice (or Bob) to store all eight tables, how many bits of storage are necessary? How does this number compare with the number of bits required for a full-table 64-bit block cipher? P6. Consider the 3-bit block cipher in Table 8.1 . Suppose the plaintext is 100100100. (a) Initially assume that CBC is not used. What is the resulting ciphertext? (b) Suppose Trudy sniffs the ciphertext. Assuming she knows that a 3-bit block cipher without CBC is being employed (but doesn't know the specific cipher), what can she surmise? (c) Now suppose that CBC is used i n=3 with . What is the resulting ciphertext? P7. (a) Using RSA, choose and , and encode the word "dog" by encrypting each letter separately. Apply the decryption algorithm to the encrypted version to recover the original plaintext message. (b) Repeat part (a) but now encrypt "dog" as one message m. P8. Consider RSA with and . a. What are n and z? b. Let e be 3. Why is this an acceptable choice for e? c. Find d such that (mod z) and . d. Encrypt the message using the key (n, e). Let c denote the corresponding ciphertext. Show all work. Hint: To simplify the calculations, use the fact: P9. In this problem, we explore the Diffie-Hellman (DH) public-key encryption algorithm, which allows two entities to agree on a shared key. The DH algorithm makes use of a large prime number p and another large number g less than p. Both p and g are made public (so that an attacker would know them). In DH, Alice and Bob each independently choose secret keys, S and S , respectively. Alice then computes her public key, T , by raising g to S and then taking mod p. Bob similarly computes his own public key T by raising g to S and then taking mod p. Alice and Bob then exchange their public keys over the Internet. Alice then calculates the shared secret key S by raising T to S and then taking mod p. Similarly, Bob calculates the shared key S' by raising T to S and then taking mod p. a. Prove that, in general, Alice and Bob obtain the same symmetric key, that is, prove . b. With p = 11 and g = 2, suppose Alice and Bob choose private keys and , respectively. Calculate Alice's and Bob's public keys, T and T . Show all work. c. Following up on part (b), now calculate S as the shared symmetric key. Show all work. d. Provide a timing diagram that shows how Diffie-Hellman can be attacked by a man-inthe-middle. The timing diagram should have three vertical lines, one for Alice, one for Bob, and one for the attacker Trudy. P10. Suppose Alice wants to communicate with Bob using symmetric key cryptography using a session key K . In Section 8.2 , we learned how public-key cryptography can be used to distribute the session key from Alice to Bob. In this problem, we explore how the session key can be distributed—without public key cryptography—using a key distribution center (KDC). The KDC is a server that shares a unique secret symmetric key with each registered user. For Alice and Bob, denote these keys by K and K . Design a scheme that uses the KDC to distribute K to Alice and Bob. Your scheme should use three messages to distribute the session key: a message from Alice to the KDC; a message from the KDC to Alice; and finally a message from Alice to Bob. The first message is K (A, B). Using the notation, K , K , S, A, and B answer the following questions. IV=111 p=3 q=11 p=5 q=11 de=1 dr. p x−r f 0 t>t0+B/r video, each compressed at a different rate. DASH is discussed in detail in Section 2.6.2. CDNs are often used to distribute stored and live video. CDNs are discussed in detail in Section 2.6.3. 9.3 Voice-over-IP Real-time conversational voice over the Internet is often referred to as Internet telephony, since, from the user's perspective, it is similar to the traditional circuit-switched telephone service. It is also commonly called Voice-over-IP (VoIP). In this section we describe the principles and protocols underlying VoIP. Conversational video is similar in many respects to VoIP, except that it includes the video of the participants as well as their voices. To keep the discussion focused and concrete, we focus here only on voice in this section rather than combined voice and video. 9.3.1 Limitations of the Best-Effort IP Service The Internet's network-layer protocol, IP, provides best-effort service. That is to say the service makes its best effort to move each datagram from source to destination as quickly as possible but makes no promises whatsoever about getting the packet to the destination within some delay bound or about a limit on the percentage of packets lost. The lack of such guarantees poses significant challenges to the design of real-time conversational applications, which are acutely sensitive to packet delay, jitter, and loss. In this section, we'll cover several ways in which the performance of VoIP over a best-effort network can be enhanced. Our focus will be on application-layer techniques, that is, approaches that do not require any changes in the network core or even in the transport layer at the end hosts. To keep the discussion concrete, we'll discuss the limitations of best-effort IP service in the context of a specific VoIP example. The sender generates bytes at a rate of 8,000 bytes per second; every 20 msecs the sender gathers these bytes into a chunk. A chunk and a special header (discussed below) are encapsulated in a UDP segment, via a call to the socket interface. Thus, the number of bytes in a chunk is and a UDP segment is sent every 20 msecs. If each packet makes it to the receiver with a constant end-to-end delay, then packets arrive at the receiver periodically every 20 msecs. In these ideal conditions, the receiver can simply play back each chunk as soon as it arrives. But unfortunately, some packets can be lost and most packets will not have the same end-to-end delay, even in a lightly congested Internet. For this reason, the receiver must take more care in determining (1) when to play back a chunk, and (2) what to do with a missing chunk. Packet Loss (20 msecs)·(8,000 bytes/sec)=160 bytes, Consider one of the UDP segments generated by our VoIP application. The UDP segment is encapsulated in an IP datagram. As the datagram wanders through the network, it passes through router buffers (that is, queues) while waiting for transmission on outbound links. It is possible that one or more of the buffers in the path from sender to receiver is full, in which case the arriving IP datagram may be discarded, never to arrive at the receiving application. Loss could be eliminated by sending the packets over TCP (which provides for reliable data transfer) rather than over UDP. However, retransmission mechanisms are often considered unacceptable for conversational real-time audio applications such as VoIP, because they increase end-to-end delay [Bolot 1996]. Furthermore, due to TCP congestion control, packet loss may result in a reduction of the TCP sender's transmission rate to a rate that is lower than the receiver's drain rate, possibly leading to buffer starvation. This can have a severe impact on voice intelligibility at the receiver. For these reasons, most existing VoIP applications run over UDP by default. [Baset 2006] reports that UDP is used by Skype unless a user is behind a NAT or firewall that blocks UDP segments (in which case TCP is used). But losing packets is not necessarily as disastrous as one might think. Indeed, packet loss rates between 1 and 20 percent can be tolerated, depending on how voice is encoded and transmitted, and on how the loss is concealed at the receiver. For example, forward error correction (FEC) can help conceal packet loss. We'll see below that with FEC, redundant information is transmitted along with the original information so that some of the lost original data can be recovered from the redundant information. Nevertheless, if one or more of the links between sender and receiver is severely congested, and packet loss exceeds 10 to 20 percent (for example, on a wireless link), then there is really nothing that can be done to achieve acceptable audio quality. Clearly, best-effort service has its limitations. End-to-End Delay End-to-end delay is the accumulation of transmission, processing, and queuing delays in routers; propagation delays in links; and end-system processing delays. For real-time conversational applications, such as VoIP, end-to-end delays smaller than 150 msecs are not perceived by a human listener; delays between 150 and 400 msecs can be acceptable but are not ideal; and delays exceeding 400 msecs can seriously hinder the interactivity in voice conversations. The receiving side of a VoIP application will typically disregard any packets that are delayed more than a certain threshold, for example, more than 400 msecs. Thus, packets that are delayed by more than the threshold are effectively lost. Packet Jitter A crucial component of end-to-end delay is the varying queuing delays that a packet experiences in the network's routers. Because of these varying delays, the time from when a packet is generated at the source until it is received at the receiver can fluctuate from packet to packet, as shown in Figure 9.1. This phenomenon is called jitter. As an example, consider two consecutive packets in our VoIP application. The sender sends the second packet 20 msecs after sending the first packet. But at the receiver, the spacing between these packets can become greater than 20 msecs. To see this, suppose the first packet arrives at a nearly empty queue at a router, but just before the second packet arrives at the queue a large number of packets from other sources arrive at the same queue. Because the first packet experiences a small queuing delay and the second packet suffers a large queuing delay at this router, the first and second packets become spaced by more than 20 msecs. The spacing between consecutive packets can also become less than 20 msecs. To see this, again consider two consecutive packets. Suppose the first packet joins the end of a queue with a large number of packets, and the second packet arrives at the queue before this first packet is transmitted and before any packets from other sources arrive at the queue. In this case, our two packets find themselves one right after the other in the queue. If the time it takes to transmit a packet on the router's outbound link is less than 20 msecs, then the spacing between first and second packets becomes less than 20 msecs. The situation is analogous to driving cars on roads. Suppose you and your friend are each driving in your own cars from San Diego to Phoenix. Suppose you and your friend have similar driving styles, and that you both drive at 100 km/hour, traffic permitting. If your friend starts out one hour before you, depending on intervening traffic, you may arrive at Phoenix more or less than one hour after your friend. If the receiver ignores the presence of jitter and plays out chunks as soon as they arrive, then the resulting audio quality can easily become unintelligible at the receiver. Fortunately, jitter can often be removed by using sequence numbers, timestamps, and a playout delay, as discussed below. 9.3.2 Removing Jitter at the Receiver for Audio For our VoIP application, where packets are being generated periodically, the receiver should attempt to provide periodic playout of voice chunks in the presence of random network jitter. This is typically done by combining the following two mechanisms: Prepending each chunk with a timestamp. The sender stamps each chunk with the time at which the chunk was generated. Delaying playout of chunks at the receiver. As we saw in our earlier discussion of Figure 9.1, the playout delay of the received audio chunks must be long enough so that most of the packets are received before their scheduled playout times. This playout delay can either be fixed throughout the duration of the audio session or vary adaptively during the audio session lifetime. We now discuss how these three mechanisms, when combined, can alleviate or even eliminate the effects of jitter. We examine two playback strategies: fixed playout delay and adaptive playout delay. Fixed Playout Delay With the fixed-delay strategy, the receiver attempts to play out each chunk exactly q msecs after the chunk is generated. So if a chunk is timestamped at the sender at time t, the receiver plays out the chunk at time assuming the chunk has arrived by that time. Packets that arrive after their scheduled playout times are discarded and considered lost. What is a good choice for q? VoIP can support delays up to about 400 msecs, although a more satisfying conversational experience is achieved with smaller values of q. On the other hand, if q is made much smaller than 400 msecs, then many packets may miss their scheduled playback times due to the network-induced packet jitter. Roughly speaking, if large variations in end-to-end delay are typical, it is preferable to use a large q; on the other hand, if delay is small and variations in delay are also small, it is preferable to use a small q, perhaps less than 150 msecs. The trade-off between the playback delay and packet loss is illustrated in Figure 9.4. The figure shows the times at which packets are generated and played Figure 9.4 Packet loss for different fixed playout delays out for a single talk spurt. Two distinct initial playout delays are considered. As shown by the leftmost staircase, the sender generates packets at regular intervals—say, every 20 msecs. The first packet in this talk spurt is received at time r. As shown in the figure, the arrivals of subsequent packets are not evenly spaced due to the network jitter. For the first playout schedule, the fixed initial playout delay is set to With this schedule, the fourth t+q, p−r. packet does not arrive by its scheduled playout time, and the receiver considers it lost. For the second playout schedule, the fixed initial playout delay is set to For this schedule, all packets arrive before their scheduled playout times, and there is therefore no loss. Adaptive Playout Delay The previous example demonstrates an important delay-loss trade-off that arises when designing a playout strategy with fixed playout delays. By making the initial playout delay large, most packets will make their deadlines and there will therefore be negligible loss; however, for conversational services such as VoIP, long delays can become bothersome if not intolerable. Ideally, we would like the playout delay to be minimized subject to the constraint that the loss be below a few percent. The natural way to deal with this trade-off is to estimate the network delay and the variance of the network delay, and to adjust the playout delay accordingly at the beginning of each talk spurt. This adaptive adjustment of playout delays at the beginning of the talk spurts will cause the sender's silent periods to be compressed and elongated; however, compression and elongation of silence by a small amount is not noticeable in speech. Following [Ramjee 1994], we now describe a generic algorithm that the receiver can use to adaptively adjust its playout delays. To this end, let the timestamp of the ith packet the time the packet was generated by the sender the time packet i is received by receiver the time packet i is played at receiver The end-to-end network delay of the ith packet is Due to network jitter, this delay will vary from packet to packet. Let d denote an estimate of the average network delay upon reception of the ith packet. This estimate is constructed from the timestamps as follows: where u is a fixed constant (for example,). Thus d is a smoothed average of the observed network delays The estimate places more weight on the recently observed network delays than on the observed network delays of the distant past. This form of estimate should not be completely unfamiliar; a similar idea is used to estimate round-trip times in TCP, as discussed in Chapter 3. Let v denote an estimate of the average deviation of the delay from the estimated average delay. This estimate is also constructed from the timestamps: p'−r. ti= = ri= pi= ri−ti. i di=(1−u)di−1+u(ri−ti) u=0.01 i r1−t1,...,ri−ti. i vi=(1−u)vi−1+u|ri−ti−di| The estimates d and v are calculated for every packet received, although they are used only to determine the playout point for the first packet in any talk spurt. Once having calculated these estimates, the receiver employs the following algorithm for the playout of packets. If packet i is the first packet of a talk spurt, its playout time, p, is computed as: where K is a positive constant (for example,). The purpose of the Kv term is to set the playout time far enough into the future so that only a small fraction of the arriving packets in the talk spurt will be lost due to late arrivals. The playout point for any subsequent packet in a talk spurt is computed as an offset from the point in time when the first packet in the talk spurt was played out. In particular, let be the length of time from when the first packet in the talk spurt is generated until it is played out. If packet j also belongs to this talk spurt, it is played out at time The algorithm just described makes perfect sense assuming that the receiver can tell whether a packet is the first packet in the talk spurt. This can be done by examining the signal energy in each received packet. 9.3.3 Recovering from Packet Loss We have discussed in some detail how a VoIP application can deal with packet jitter. We now briefly describe several schemes that attempt to preserve acceptable audio quality in the presence of packet loss. Such schemes are called loss recovery schemes. Here we define packet loss in a broad sense: A packet is lost either if it never arrives at the receiver or if it arrives after its scheduled playout time. Our VoIP example will again serve as a context for describing loss recovery schemes. As mentioned at the beginning of this section, retransmitting lost packets may not be feasible in a realtime conversational application such as VoIP. Indeed, retransmitting a packet that has missed its playout deadline serves absolutely no purpose. And retransmitting a packet that overflowed a router queue cannot normally be accomplished quickly enough. Because of these considerations, VoIP applications often use some type of loss anticipation scheme. Two types of loss anticipation schemes are forward error correction (FEC) and interleaving. i i i pi=ti+di+Kvi K=4 i qi=pi−ti pj=tj+qi Forward Error Correction (FEC) The basic idea of FEC is to add redundant information to the original packet stream. For the cost of marginally increasing the transmission rate, the redundant information can be used to reconstruct approximations or exact versions of some of the lost packets. Following [Bolot 1996] and [Perkins 1998], we now outline two simple FEC mechanisms. The first mechanism sends a redundant encoded chunk after every n chunks. The redundant chunk is obtained by exclusive OR-ing the n original chunks [Shacham 1990]. In this manner if any one packet of the group of packets is lost, the receiver can fully reconstruct the lost packet. But if two or more packets in a group are lost, the receiver cannot reconstruct the lost packets. By keeping , the group size, small, a large fraction of the lost packets can be recovered when loss is not excessive. However, the smaller the group size, the greater the relative increase of the transmission rate. In particular, the transmission rate increases by a factor of 1/n, so that, if then the transmission rate increases by 33 percent. Furthermore, this simple scheme increases the playout delay, as the receiver must wait to receive the entire group of packets before it can begin playout. For more practical details about how FEC works for multimedia transport see [RFC 5109]. The second FEC mechanism is to send a lower-resolution audio stream as the redundant information. For example, the sender might create a nominal audio stream and a corresponding low-resolution, lowbit rate audio stream. (The nominal stream could be a PCM encoding at 64 kbps, and the lower-quality stream could be a GSM encoding at 13 kbps.) The low-bit rate stream is referred to as the redundant stream. As shown in Figure 9.5, the sender constructs the nth packet by taking the nth chunk from the nominal stream and appending to it the st chunk from the redundant stream. In this manner, whenever there is nonconsecutive packet loss, the receiver can conceal the loss by playing out the lowbit rate encoded chunk that arrives with the subsequent packet. Of course, low-bit rate chunks give lower quality than the nominal chunks. However, a stream of mostly high-quality chunks, occasional lowquality chunks, and no missing chunks gives good overall audio quality. Note that in this scheme, the receiver only has to receive two packets before playback, so that the increased playout delay is small. Furthermore, if the low-bit rate encoding is much less than the nominal encoding, then the marginal increase in the transmission rate will be small. In order to cope with consecutive loss, we can use a simple variation. Instead of appending just the st low-bit rate chunk to the nth nominal chunk, the sender can append the st and nd lowbit rate chunk, or append the st and rd low-bit rate chunk, and so on. By appending more lowbit rate chunks to each nominal chunk, the audio quality at the receiver becomes acceptable for a wider variety of harsh best-effort environments. On the other hand, the additional chunks increase the transmission bandwidth and the playout delay. n+1 n+1 n=3, (n−1) (n−1) (n−1) (n−2) (n−1) (n−3) Figure 9.5 Piggybacking lower-quality redundant information Interleaving As an alternative to redundant transmission, a VoIP application can send interleaved audio. As shown in Figure 9.6, the sender resequences units of audio data before transmission, so that originally adjacent units are separated by a certain distance in the transmitted stream. Interleaving can mitigate the effect of packet losses. If, for example, units are 5 msecs in length and chunks are 20 msecs (that is, four units per chunk), then the first chunk could contain units 1, 5, 9, and 13; the second chunk could contain units 2, 6, 10, and 14; and so on. Figure 9.6 shows that the loss of a single packet from an interleaved stream results in multiple small gaps in the reconstructed stream, as opposed to the single large gap that would occur in a noninterleaved stream. Interleaving can significantly improve the perceived quality of an audio stream [Perkins 1998]. It also has low overhead. The obvious disadvantage of interleaving is that it increases latency. This limits its use for conversational applications such as VoIP, although it can perform well for streaming stored audio. A major advantage of interleaving is that it does not increase the bandwidth requirements of a stream. Error Concealment Error concealment schemes attempt to produce a replacement for a lost packet that is similar to the original. As discussed in [Perkins 1998], this is possible since audio Figure 9.6 Sending interleaved audio signals, and in particular speech, exhibit large amounts of short-term self-similarity. As such, these techniques work for relatively small loss rates (less than 15 percent), and for small packets (4–40 msecs). When the loss length approaches the length of a phoneme (5–100 msecs) these techniques break down, since whole phonemes may be missed by the listener. Perhaps the simplest form of receiver-based recovery is packet repetition. Packet repetition replaces lost packets with copies of the packets that arrived immediately before the loss. It has low computational complexity and performs reasonably well. Another form of receiver-based recovery is interpolation, which uses audio before and after the loss to interpolate a suitable packet to cover the loss. Interpolation performs somewhat better than packet repetition but is significantly more computationally intensive [Perkins 1998]. 9.3.4 Case Study: VoIP with Skype Skype is an immensely popular VoIP application with over 50 million accounts active on a daily basis. In addition to providing host-to-host VoIP service, Skype offers host-to-phone services, phone-to-host services, and multi-party host-to-host video conferencing services. (Here, a host is again any Internet connected IP device, including PCs, tablets, and smartphones.) Skype was acquired by Microsoft in 2011. Because the Skype protocol is proprietary, and because all Skype's control and media packets are encrypted, it is difficult to precisely determine how Skype operates. Nevertheless, from the Skype Web site and several measurement studies, researchers have learned how Skype generally works [Baset 2006; Guha 2006; Chen 2006; Suh 2006; Ren 2006; Zhang X 2012]. For both voice and video, the Skype clients have at their disposal many different codecs, which are capable of encoding the media at a wide range of rates and qualities. For example, video rates for Skype have been measured to be as low as 30 kbps for a low-quality session up to almost 1 Mbps for a high quality session [Zhang X 2012]. Typically, Skype's audio quality is better than the "POTS" (Plain Old Telephone Service) quality provided by the wire-line phone system. (Skype codecs typically sample voice at 16,000 samples/sec or higher, which provides richer tones than POTS, which samples at 8,000/sec.) By default, Skype sends audio and video packets over UDP. However, control packets are sent over TCP, and media packets are also sent over TCP when firewalls block UDP streams. Skype uses FEC for loss recovery for both voice and video streams sent over UDP. The Skype client also adapts the audio and video streams it sends to current network conditions, by changing video quality and FEC overhead [Zhang X 2012]. Skype uses P2P techniques in a number of innovative ways, nicely illustrating how P2P can be used in applications that go beyond content distribution and file sharing. As with instant messaging, host-to-host Internet telephony is inherently P2P since, at the heart of the application, pairs of users (that is, peers) communicate with each other in real time. But Skype also employs P2P techniques for two other important functions, namely, for user location and for NAT traversal. Figure 9.7 Skype peers As shown in Figure 9.7, the peers (hosts) in Skype are organized into a hierarchical overlay network, with each peer classified as a super peer or an ordinary peer. Skype maintains an index that maps Skype usernames to current IP addresses (and port numbers). This index is distributed over the super peers. When Alice wants to call Bob, her Skype client searches the distributed index to determine Bob's current IP address. Because the Skype protocol is proprietary, it is currently not known how the index mappings are organized across the super peers, although some form of DHT organization is very possible. P2P techniques are also used in Skype relays, which are useful for establishing calls between hosts in home networks. Many home network configurations provide access to the Internet through NATs, as discussed in Chapter 4. Recall that a NAT prevents a host from outside the home network from initiating a connection to a host within the home network. If both Skype callers have NATs, then there is a problem—neither can accept a call initiated by the other, making a call seemingly impossible. The clever use of super peers and relays nicely solves this problem. Suppose that when Alice signs in, she is assigned to a non-NATed super peer and initiates a session to that super peer. (Since Alice is initiating the session, her NAT permits this session.) This session allows Alice and her super peer to exchange control messages. The same happens for Bob when he signs in. Now, when Alice wants to call Bob, she informs her super peer, who in turn informs Bob's super peer, who in turn informs Bob of Alice's incoming call. If Bob accepts the call, the two super peers select a third non-NATed super peer—the relay peer—whose job will be to relay data between Alice and Bob. Alice's and Bob's super peers then instruct Alice and Bob respectively to initiate a session with the relay. As shown in Figure 9.7, Alice then sends voice packets to the relay over the Alice-to-relay connection (which was initiated by Alice), and the relay then forwards these packets over the relay-to-Bob connection (which was initiated by Bob); packets from Bob to Alice flow over these same two relay connections in reverse. And voila!—Bob and Alice have an end-to-end connection even though neither can accept a session originating from outside. Up to now, our discussion on Skype has focused on calls involving two persons. Now let's examine multi-party audio conference calls. With participants, if each user were to send a copy of its audio stream to each of the other users, then a total of audio streams would need to be sent into the network to support the audio conference. To reduce this bandwidth usage, Skype employs a clever distribution technique. Specifically, each user sends its audio stream to the conference initiator. The conference initiator combines the audio streams into one stream (basically by adding all the audio signals together) and then sends a copy of each combined stream to each of the other participants. In this manner, the number of streams is reduced to For ordinary two-person video conversations, Skype routes the call peer-to-peer, unless NAT traversal is required, in which case the call is relayed through a non-NATed peer, as described earlier. For a video conference call involving participants, due to the nature of the video medium, Skype does not combine the call into one $N>2$ $N-1$ $N(N-1)$ $N-1$ $2(N-1)$. $N>2$ stream at one location and then redistribute the stream to all the participants, as it does for voice calls. Instead, each participant's video stream is routed to a server cluster (located in Estonia as of 2011), which in turn relays to each participant the streams of the other participants [Zhang X 2012]. You may be wondering why each participant sends a copy to a server rather than directly sending a copy of its video stream to each of the other participants? Indeed, for both approaches, video streams are being collectively received by the N participants in the conference. The reason is, because upstream link bandwidths are significantly lower than downstream link bandwidths in most access links, the upstream links may not be able to support the streams with the P2P approach. VoIP systems such as Skype, WeChat, and Google Talk introduce new privacy concerns. Specifically, when Alice and Bob communicate over VoIP, Alice can sniff Bob's IP address and then use geo-location services [MaxMind 2016; Quova 2016] to determine Bob's current location and ISP (for example, his work or home ISP). In fact, with Skype it is possible for Alice to block the transmission of certain packets during call establishment so that she obtains Bob's current IP address, say every hour, without Bob knowing that he is being tracked and without being on Bob's contact list. Furthermore, the IP address discovered from Skype can be correlated with IP addresses found in BitTorrent, so that Alice can determine the files that Bob is downloading [LeBlond 2011]. Moreover, it is possible to partially decrypt a Skype call by doing a traffic analysis of the packet sizes in a stream [White 2011]. N−1 N−1 N−1 N(N−1) N−1 9.4 Protocols for Real-Time Conversational Applications Real-time conversational applications, including VoIP and video conferencing, are compelling and very popular. It is therefore not surprising that standards bodies, such as the IETF and ITU, have been busy for many years (and continue to be busy!) at hammering out standards for this class of applications. With the appropriate standards in place for real-time conversational applications, independent companies are creating new products that interoperate with each other. In this section we examine RTP and SIP for real-time conversational applications. Both standards are enjoying widespread implementation in industry products. 9.4.1 RTP In the previous section, we learned that the sender side of a VoIP application appends header fields to the audio chunks before passing them to the transport layer. These header fields include sequence numbers and timestamps. Since most multimedia networking applications can make use of sequence numbers and timestamps, it is convenient to have a standardized packet structure that includes fields for audio/video data, sequence number, and timestamp, as well as other potentially useful fields. RTP, defined in RFC 3550, is such a standard. RTP can be used for transporting common formats such as PCM, ACC, and MP3 for sound and MPEG and H.263 for video. It can also be used for transporting proprietary sound and video formats. Today, RTP enjoys widespread implementation in many products and research prototypes. It is also complementary to other important real-time interactive protocols, such as SIP. In this section, we provide an introduction to RTP. We also encourage you to visit Henning Schulzrinne's RTP site [Schulzrinne-RTP 2012], which provides a wealth of information on the subject. Also, you may want to visit the RAT site [RAT 2012], which documents VoIP application that uses RTP. RTP Basics RTP typically runs on top of UDP. The sending side encapsulates a media chunk within an RTP packet, then encapsulates the packet in a UDP segment, and then hands the segment to IP. The receiving side extracts the RTP packet from the UDP segment, then extracts the media chunk from the RTP packet, and then passes the chunk to the media player for decoding and rendering. As an example, consider the use of RTP to transport voice. Suppose the voice source is PCM-encoded (that is, sampled, quantized, and digitized) at 64 kbps. Further suppose that the application collects the encoded data in 20-msec chunks, that is, 160 bytes in a chunk. The sending side precedes each chunk of the audio data with an RTP header that includes the type of audio encoding, a sequence number, and a timestamp. The RTP header is normally 12 bytes. The audio chunk along with the RTP header form the RTP packet. The RTP packet is then sent into the UDP socket interface. At the receiver side, the application receives the RTP packet from its socket interface. The application extracts the audio chunk from the RTP packet and uses the header fields of the RTP packet to properly decode and play back the audio chunk. If an application incorporates RTP—instead of a proprietary scheme to provide payload type, sequence numbers, or timestamps—then the application will more easily interoperate with other networked multimedia applications. For example, if two different companies develop VoIP software and they both incorporate RTP into their product, there may be some hope that a user using one of the VoIP products will be able to communicate with a user using the other VoIP product. In Section 9.4.2, we'll see that RTP is often used in conjunction with SIP, an important standard for Internet telephony. It should be emphasized that RTP does not provide any mechanism to ensure timely delivery of data or provide other quality-of-service (QoS) guarantees; it does not even guarantee delivery of packets or prevent out-of-order delivery of packets. Indeed, RTP encapsulation is seen only at the end systems. Routers do not distinguish between IP datagrams that carry RTP packets and IP datagrams that don't. RTP allows each source (for example, a camera or a microphone) to be assigned its own independent RTP stream of packets. For example, for a video conference between two participants, four RTP streams could be opened—two streams for transmitting the audio (one in each direction) and two streams for transmitting the video (again, one in each direction). However, many popular encoding techniques—including MPEG 1 and MPEG 2—bundle the audio and video into a single stream during the encoding process. When the audio and video are bundled by the encoder, then only one RTP stream is generated in each direction. RTP packets are not limited to unicast applications. They can also be sent over one-to-many and manyto-many multicast trees. For a many-to-many multicast session, all of the session's senders and sources typically use the same multicast group for sending their RTP streams. RTP multicast streams belonging together, such as audio and video streams emanating from multiple senders in a video conference application, belong to an RTP session. Figure 9.8 RTP header fields RTP Packet Header Fields As shown in Figure 9.8, the four main RTP packet header fields are the payload type, sequence number, timestamp, and source identifier fields. The payload type field in the RTP packet is 7 bits long. For an audio stream, the payload type field is used to indicate the type of audio encoding (for example, PCM, adaptive delta modulation, linear predictive encoding) that is being used. If a sender decides to change the encoding in the middle of a session, the sender can inform the receiver of the change through this payload type field. The sender may want to change the encoding in order to increase the audio quality or to decrease the RTP stream bit rate. Table 9.2 lists some of the audio payload types currently supported by RTP. For a video stream, the payload type is used to indicate the type of video encoding (for example, motion JPEG, MPEG 1, MPEG 2, H.261). Again, the sender can change video encoding on the fly during a session. Table 9.3 lists some of the video payload types currently supported by RTP. The other important fields are the following: Sequence number field. The sequence number field is 16 bits long. The sequence number increments by one for each RTP packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. For example, if the receiver side of the application receives a stream of RTP packets with a gap between sequence numbers 86 and 89, then the receiver knows that packets 87 and 88 are missing. The receiver can then attempt to conceal the lost data. Timestamp field. The timestamp field is 32 bits long. It reflects the sampling instant of the first byte in the RTP data packet. As we saw in the preceding section, the receiver can use timestamps to remove packet jitter introduced in the network and to provide synchronous playout at the receiver. The timestamp is derived from a sampling clock at the sender. As an example, for audio the timestamp clock increments by one for each sampling period (for example, each 125 μsec for an 8 kHz sampling clock); if the audio application generates chunks consisting of 160 encoded samples, then the timestamp increases by 160 for each RTP packet when the source is active. The timestamp clock continues to increase at a constant rate even if the source is inactive. Synchronization source identifier (SSRC). The SSRC field is 32 bits long. It identifies the source of the RTP stream. Typically, each stream in an RTP session has a distinct SSRC. The SSRC is not the IP address of the sender, but instead is a number that the source assigns randomly when the new stream is started. The probability that two streams get assigned the same SSRC is very small. Should this happen, the two sources pick a new SSRC value. Table 9.2 Audio payload types supported by RTP Payload-Type Number Audio Format Sampling Rate Rate 0 PCM μ-law 8 kHz 64 kbps 1 1016 8 kHz 4.8 kbps 3 GSM 8 kHz 13 kbps 7 LPC 8 kHz 2.4 kbps 9 G.722 16 kHz 48–64 kbps 14 MPEG Audio 90 kHz — 15 G.728 8 kHz 16 kbps Table 9.3 Some video payload types supported by RTP Payload-Type Number Video Format 26 Motion JPEG 31 H.261 32 MPEG 1 video 33 MPEG 2 video 9.4.2 SIP The Session Initiation Protocol (SIP), defined in [RFC 3261; RFC 5411], is an open and lightweight protocol that does the following: It provides mechanisms for establishing calls between a caller and a callee over an IP network. It allows the caller to notify the callee that it wants to start a call. It allows the participants to agree on media encodings. It also allows participants to end calls. It provides mechanisms for the caller to determine the current IP address of the callee. Users do not have a single, fixed IP address because they may be assigned addresses dynamically (using DHCP) and because they may have multiple IP devices, each with a different IP address. It provides mechanisms for call management, such as adding new media streams during the call, changing the encoding during the call, inviting new participants during the call, call transfer, and call holding. Setting Up a Call to a Known IP Address To understand the essence of SIP, it is best to take a look at a concrete example. In this example, Alice is at her PC and she wants to call Bob, who is also working at his PC. Alice's and Bob's PCs are both equipped with SIP-based software for making and receiving phone calls. In this initial example, we'll assume that Alice knows the IP address of Bob's PC. Figure 9.9 illustrates the SIP call-establishment process. In Figure 9.9, we see that an SIP session begins when Alice sends Bob an INVITE message, which resembles an HTTP request message. This INVITE message is sent over UDP to the well-known port 5060 for SIP. (SIP messages can also be sent over TCP.) The INVITE message includes an identifier for Bob (bob@193.64.210.89), an indication of Alice's current IP address, an indication that Alice desires to receive audio, which is to be encoded in format AVP 0 (PCM encoded μ-law) and Figure 9.9 SIP call establishment when Alice knows Bob's IP address encapsulated in RTP, and an indication that she wants to receive the RTP packets on port 38060. After receiving Alice's INVITE message, Bob sends an SIP response message, which resembles an HTTP response message. This response SIP message is also sent to the SIP port 5060. Bob's response includes a 200 OK as well as an indication of his IP address, his desired encoding and packetization for reception, and his port number to which the audio packets should be sent. Note that in this example Alice and Bob are going to use different audio-encoding mechanisms: Alice is asked to encode her audio with GSM whereas Bob is asked to encode his audio with PCM μ-law. After receiving Bob's response, Alice sends Bob an SIP acknowledgment message. After this SIP transaction, Bob and Alice can talk. (For visual convenience, Figure 9.9 shows Alice talking after Bob, but in truth they would normally talk at the same time.) Bob will encode and packetize the audio as requested and send the audio packets to port number 38060 at IP address 167.180.112.24. Alice will also encode and packetize the audio as requested and send the audio packets to port number 48753 at IP address 193.64.210.89. From this simple example, we have learned a number of key characteristics of SIP. First, SIP is an outof-band protocol: The SIP messages are sent and received in sockets that are different from those used for sending and receiving the media data. Second, the SIP messages themselves are ASCII-readable and resemble HTTP messages. Third, SIP requires all messages to be acknowledged, so it can run over UDP or TCP. In this example, let's consider what would happen if Bob does not have a PCM μ-law codec for encoding audio. In this case, instead of responding with 200 OK, Bob would likely respond with a 606 Not Acceptable and list in the message all the codecs he can use. Alice would then choose one of the listed codecs and send another INVITE message, this time advertising the chosen codec. Bob could also simply reject the call by sending one of many possible rejection reply codes. (There are many such codes, including "busy," "gone," "payment required," and "forbidden.") SIP Addresses In the previous example, Bob's SIP address is sip:bob@193.64.210.89. However, we expect many—if not most—SIP addresses to resemble e-mail addresses. For example, Bob's address might be sip:bob@domain.com. When Alice's SIP device sends an INVITE message, the message would include this e-mail-like address; the SIP infrastructure would then route the message to the IP device that Bob is currently using (as we'll discuss below). Other possible forms for the SIP address could be Bob's legacy phone number or simply Bob's first/middle/last name (assuming it is unique). An interesting feature of SIP addresses is that they can be included in Web pages, just as people's email addresses are included in Web pages with the mailto URL. For example, suppose Bob has a personal homepage, and he wants to provide a means for visitors to the homepage to call him. He could then simply include the URL sip:bob@domain.com. When the visitor clicks on the URL, the SIP application in the visitor's device is launched and an INVITE message is sent to Bob. SIP Messages In this short introduction to SIP, we'll not cover all SIP message types and headers. Instead, we'll take a brief look at the SIP INVITE message, along with a few common header lines. Let us again suppose that Alice wants to initiate a VoIP call to Bob, and this time Alice knows only Bob's SIP address, bob@domain.com, and does not know the IP address of the device that Bob is currently using. Then her message might look something like this: INVITE sip:bob@domain.com SIP/2.0 Via: SIP/2.0/UDP 167.180.112.24 From: sip:alice@hereway.com To: sip:bob@domain.com Call-ID: a2e3a@pigeon.hereway.com Content-Type: application/sdp Content-Length: 885 c=IN IP4 167.180.112.24 m=audio 38060 RTP/AVP 0 The INVITE line includes the SIP version, as does an HTTP request message. Whenever an SIP message passes through an SIP device (including the device that originates the message), it attaches a Via header, which indicates the IP address of the device. (We'll see soon that the typical INVITE message passes through many SIP devices before reaching the callee's SIP application.) Similar to an e-mail message, the SIP message includes a From header line and a To header line. The message includes a Call-ID, which uniquely identifies the call (similar to the message-ID in e-mail). It includes a Content-Type header line, which defines the format used to describe the content contained in the SIP message. It also includes a Content-Length header line, which provides the length in bytes of the content in the message. Finally, after a carriage return and line feed, the message contains the content. In this case, the content provides information about Alice's IP address and how Alice wants to receive the audio. Name Translation and User Location In the example in Figure 9.9, we assumed that Alice's SIP device knew the IP address where Bob could be contacted. But this assumption is quite unrealistic, not only because IP addresses are often dynamically assigned with DHCP, but also because Bob may have multiple IP devices (for example, different devices for his home, work, and car). So now let us suppose that Alice knows only Bob's e-mail address, bob@domain.com, and that this same address is used for SIP-based calls. In this case, Alice needs to obtain the IP address of the device that the user bob@domain.com is currently using. To find this out, Alice creates an INVITE message that begins with INVITE bob@domain.com SIP/2.0 and sends this message to an SIP proxy. The proxy will respond with an SIP reply that might include the IP address of the device that bob@domain.com is currently using. Alternatively, the reply might include the IP address of Bob's voicemail box, or it might include a URL of a Web page (that says "Bob is sleeping. Leave me alone!"). Also, the result returned by the proxy might depend on the caller: If the call is from Bob's wife, he might accept the call and supply his IP address; if the call is from Bob's mother-inlaw, he might respond with the URL that points to the I-am-sleeping Web page! Now, you are probably wondering, how can the proxy server determine the current IP address for bob@domain.com? To answer this question, we need to say a few words about another SIP device, the SIP registrar. Every SIP user has an associated registrar. Whenever a user launches an SIP application on a device, the application sends an SIP register message to the registrar, informing the registrar of its current IP address. For example, when Bob launches his SIP application on his PDA, the application would send a message along the lines of: REGISTER sip:domain.com SIP/2.0 Via: SIP/2.0/UDP 193.64.210.89 From: sip:bob@domain.com To: sip:bob@domain.com Expires: 3600 Bob's registrar keeps track of Bob's current IP address. Whenever Bob switches to a new SIP device, the new device sends a new register message,