packet destined for x arriving at y or z as of t will bounce back and forth between these two nodes forever (or until the forwarding tables are changed). 2. Since node y has computed a new minimum cost to x, it informs z of its new distance vector at time t . 3. Sometime after t , z receives y's new distance vector, which indicates that y's minimum cost to x is 6. z knows it can get to y with a cost of 1 and hence computes a new least cost to x of Since z's least cost to x has increased, it then informs y of its new distance vector at t . 4. In a similar manner, after receiving z's new distance vector, y determines and sends z its distance vector. z then determines and sends y its distance vector, and so on. How long will the process continue? You should convince yourself that the loop will persist for 44 iterations (message exchanges between y and z)—until z eventually computes the cost of its path via y to be greater than 50. At this point, z will (finally!) determine that its least-cost path to x is via its direct connection to x. y will then route to x via z. The result of the bad news about the increase in link cost has indeed traveled slowly! What would have happened if the link cost c(y, x) had changed from 4 to 10,000 and the cost c(z, x) had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the count-to-infinity problem. Distance-Vector Algorithm: Adding Poisoned Reverse The specific looping scenario just described can be avoided using a technique known as poisoned reverse. The idea is simple—if z routes through y to get to destination x, then z will advertise to y that its distance to x is infinity, that is, z will advertise to y that (even though z knows in truth). z will continue telling this little white lie to y as long as it routes to x via y. Since y believes that z has no path to x, y will never attempt to route to x via z, as long as z continues to route to x via y (and lies about doing so). Let's now see how poisoned reverse solves the particular looping problem we encountered before in Figure 5.5(b). As a result of the poisoned reverse, y's distance table indicates When the cost of the (x, y) link changes from 4 to 60 at time t , y updates its table and continues to route directly to x, albeit $D_y(x)=\min\{c(y,x)+D_x(x), c(y,z)+D_z(x)\}=\min\{60+0,1+5\}=6$ 1 1 1 1 $D_z(x)=\min\{50+0,1+6\}=7$. 2 $D_y(x)=8$ $D_z(x)=9$ $D_z(x)=\infty$ $D_z(x)=5$ $D_z(x)=\infty$. 0 at a higher cost of 60, and informs z of its new cost to x, that is, After receiving the update at t , z immediately shifts its route to x to be via the direct (z, x) link at a cost of 50. Since this is a new least-cost path to x, and since the path no longer passes through y, z now informs y that at t . After receiving the update from z, y updates its distance table with Also, since z is now on y's leastcost path to x, y poisons the reverse path from z to x by informing z at time t that (even though y knows that in truth). Does poisoned reverse solve the general count-to-infinity problem? It does not. You should convince yourself that loops involving three or more nodes (rather than simply two immediately neighboring nodes) will not be detected by the poisoned reverse technique. A Comparison of LS and DV Routing Algorithms The DV and LS algorithms take complementary approaches toward computing routing. In the DV algorithm, each node talks to only its directly connected neighbors, but it provides its neighbors with leastcost estimates from itself to all the nodes (that it knows about) in the network. The LS algorithm requires global information. Consequently, when implemented in each and every router, e.g., as in Figure 4.2 and 5.1, each node would need to communicate with all other nodes (via broadcast), but it tells them only the costs of its directly connected links. Let's conclude our study of LS and DV algorithms with a quick comparison of some of their attributes. Recall that N is the set of nodes (routers) and E is the set of edges (links). Message complexity. We have seen that LS requires each node to know the cost of each link in the network. This requires O(|N| |E|) messages to be sent. Also, whenever a link cost changes, the new link cost must be sent to all nodes. The DV algorithm requires message exchanges between directly connected neighbors at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link. Speed of convergence. We have seen that our implementation of LS is an O(|N|) algorithm requiring O(|N| |E|)) messages. The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem. Robustness. What can happen if a router fails, misbehaves, or is sabotaged? Under LS, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any packets it received as part of an LS broadcast. But an LS node is computing only its own forwarding tables; other nodes are performing similar calculations for themselves. This means route calculations are somewhat separated under LS, providing a degree of robustness. Under DV, a node can advertise incorrect least-cost paths to any or all destinations. (Indeed, in 1997, a malfunctioning router in a small ISP provided national backbone routers with erroneous routing information. This caused other routers to flood the malfunctioning router with traffic and caused large portions of the Dy(x)=60. 1 Dz(x)=50 2 Dy(x)=51. 3 Dy(x)=∞ Dy(x)=51 2 Internet to become disconnected for up to several hours [Neumann 1997].) More generally, we note that, at each iteration, a node's calculation in DV is passed on to its neighbor and then indirectly to its neighbor's neighbor on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV. In the end, neither algorithm is an obvious winner over the other; indeed, both algorithms are used in the Internet. 5.3 Intra-AS Routing in the Internet: OSPF In our study of routing algorithms so far, we've viewed the network simply as a collection of interconnected routers. One router was indistinguishable from another in the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is simplistic for two important reasons: Scale. As the number of routers becomes large, the overhead involved in communicating, computing, and storing routing information becomes prohibitive. Today's Internet consists of hundreds of millions of routers. Storing routing information for possible destinations at each of these routers would clearly require enormous amounts of memory. The overhead required to broadcast connectivity and link cost updates among all of the routers would be huge! A distance-vector algorithm that iterated among such a large number of routers would surely never converge. Clearly, something must be done to reduce the complexity of route computation in a network as large as the Internet. Administrative autonomy. As described in Section 1.3, the Internet is a network of ISPs, with each ISP consisting of its own network of routers. An ISP generally desires to operate its network as it pleases (for example, to run whatever routing algorithm it chooses within its network) or to hide aspects of its network's internal organization from the outside. Ideally, an organization should be able to operate and administer its network as it wishes, while still being able to connect its network to other outside networks. Both of these problems can be solved by organizing routers into autonomous systems (ASs), with each AS consisting of a group of routers that are under the same administrative control. Often the routers in an ISP, and the links that interconnect them, constitute a single AS. Some ISPs, however, partition their network into multiple ASs. In particular, some tier-1 ISPs use one gigantic AS for their entire network, whereas others break up their ISP into tens of interconnected ASs. An autonomous system is identified by its globally unique autonomous system number (ASN) [RFC 1930]. AS numbers, like IP addresses, are assigned by ICANN regional registries [ICANN 2016]. Routers within the same AS all run the same routing algorithm and have information about each other. The routing algorithm running within an autonomous system is called an intra-autonomous system routing protocol. Open Shortest Path First (OSPF) OSPF routing and its closely related cousin, IS-IS, are widely used for intra-AS routing in the Internet. The Open in OSPF indicates that the routing protocol specification is publicly available (for example, as opposed to Cisco's EIGRP protocol, which was only recently became open [Savage 2015], after roughly 20 years as a Cisco-proprietary protocol). The most recent version of OSPF, version 2, is defined in [RFC 2328], a public document. OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra's least-cost path algorithm. With OSPF, each router constructs a complete topological map (that is, a graph) of the entire autonomous system. Each router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as the root node. Individual link costs are configured by the network administrator (see sidebar, Principles and Practice: Setting OSPF Weights). The administrator might choose to set all link costs to 1, PRINCIPLES IN PRACTICE SETTING OSPF LINK WEIGHTS Our discussion of link-state routing has implicitly assumed that link weights are set, a routing algorithm such as OSPF is run, and traffic flows according to the routing tables computed by the LS algorithm. In terms of cause and effect, the link weights are given (i.e., they come first) and result (via Dijkstra's algorithm) in routing paths that minimize overall cost. In this viewpoint, link weights reflect the cost of using a link (e.g., if link weights are inversely proportional to capacity, then the use of high-capacity links would have smaller weight and thus be more attractive from a routing standpoint) and Dijsktra's algorithm serves to minimize overall cost. In practice, the cause and effect relationship between link weights and routing paths may be reversed, with network operators configuring link weights in order to obtain routing paths that achieve certain traffic engineering goals [Fortz 2000, Fortz 2002]. For example, suppose a network operator has an estimate of traffic flow entering the network at each ingress point and destined for each egress point. The operator may then want to put in place a specific routing of ingress-to-egress flows that minimizes the maximum utilization over all of the network's links. But with a routing algorithm such as OSPF, the operator's main "knobs" for tuning the routing of flows through the network are the link weights. Thus, in order to achieve the goal of minimizing the maximum link utilization, the operator must find the set of link weights that achieves this goal. This is a reversal of the cause and effect relationship—the desired routing of flows is known, and the OSPF link weights must be found such that the OSPF routing algorithm results in this desired routing of flows. thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links. OSPF does not mandate a policy for how link weights are set (that is the job of the network administrator), but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights. With OSPF, a router broadcasts routing information to all other routers in the autonomous system, not just to its neighboring routers. A router broadcasts link-state information whenever there is a change in a link's state (for example, a change in cost or a change in up/down status). It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed. RFC 2328 notes that "this periodic updating of link state advertisements adds robustness to the link state algorithm." OSPF advertisements are contained in OSPF messages that are carried directly by IP, with an upper-layer protocol of 89 for OSPF. Thus, the OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router's database of network-wide link state. Some of the advances embodied in OSPF include the following: Security. Exchanges between OSPF routers (for example, link-state updates) can be authenticated. With authentication, only trusted routers can participate in the OSPF protocol within an AS, thus preventing malicious intruders (or networking students taking their newfound knowledge out for a joyride) from injecting incorrect information into router tables. By default, OSPF packets between routers are not authenticated and could be forged. Two types of authentication can be configured— simple and MD5 (see Chapter 8 for a discussion on MD5 and authentication in general). With simple authentication, the same password is configured on each router. When a router sends an OSPF packet, it includes the password in plaintext. Clearly, simple authentication is not very secure. MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key. (See the discussion of message authentication codes in Chapter 8.) Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks. Multiple same-cost paths. When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist). Integrated support for unicast and multicast routing. Multicast OSPF (MOSPF) [RFC 1584] provides simple extensions to OSPF to provide for multicast routing. MOSPF uses the existing OSPF link database and adds a new type of link-state advertisement to the existing OSPF link-state broadcast mechanism. Support for hierarchy within a single AS. An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area. Lastly, exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non-border routers as well. Inter-area routing within the AS requires that the packet be first routed to an area border router (intra-area routing), then routed through the backbone to the area border router that is in the destination area, and then routed to the final destination. OSPF is a relatively complex protocol, and our coverage here has been necessarily brief; [Huitema 1998; Moy 1998; RFC 2328] provide additional details. 5.4 Routing Among the ISPs: BGP We just learned that OSPF is an example of an intra-AS routing protocol. When routing a packet between a source and destination within the same AS, the route the packet follows is entirely determined by the intra-AS routing protocol. However, to route a packet across multiple ASs, say from a smartphone in Timbuktu to a server in a datacenter in Silicon Valley, we need an inter-autonomous system routing protocol. Since an inter-AS routing protocol involves coordination among multiple ASs, communicating ASs must run the same inter-AS routing protocol. In fact, in the Internet, all ASs run the same inter-AS routing protocol, called the Border Gateway Protocol, more commonly known as BGP [RFC 4271; Stewart 1999]. BGP is arguably the most important of all the Internet protocols (the only other contender would be the IP protocol that we studied in Section 4.3), as it is the protocol that glues the thousands of ISPs in the Internet together. As we will soon see, BGP is a decentralized and asynchronous protocol in the vein of distance-vector routing described in Section 5.2.2. Although BGP is a complex and challenging protocol, to understand the Internet on a deep level, we need to become familiar with its underpinnings and operation. The time we devote to learning BGP will be well worth the effort. 5.4.1 The Role of BGP To understand the responsibilities of BGP, consider an AS and an arbitrary router in that AS. Recall that every router has a forwarding table, which plays the central role in the process of forwarding arriving packets to outbound router links. As we have learned, for destinations that are within the same AS, the entries in the router's forwarding table are determined by the AS's intra-AS routing protocol. But what about destinations that are

outside of the AS? This is precisely where BGP comes to the rescue. In BGP, packets are not routed to a specific destination address, but instead to CIDRized prefixes, with each prefix representing a subnet or a collection of subnets. In the world of BGP, a destination may take the form 138.16.68/22, which for this example includes 1,024 IP addresses. Thus, a router's forwarding table will have entries of the form (x, I), where x is a prefix (such as 138.16.68/22) and I is an interface number for one of the router's interfaces. As an inter-AS routing protocol, BGP provides each router a means to: 1. Obtain prefix reachability information from neighboring ASs. In particular, BGP allows each subnet to advertise its existence to the rest of the Internet. A subnet screams, "I exist and I am here," and BGP makes sure that all the routers in the Internet know about this subnet. If it weren't for BGP, each subnet would be an isolated island—alone, unknown and unreachable by the rest of the Internet. 2. Determine the "best" routes to the prefixes. A router may learn about two or more different routes to a specific prefix. To determine the best route, the router will locally run a BGP routeselection procedure (using the prefix reachability information it obtained via neighboring routers). The best route will be determined based on policy as well as the reachability information. Let us now delve into how BGP carries out these two tasks.
++ 5.4.2 Advertising BGP Route Information Consider the network shown in Figure 5.8. As we can see, this simple network has three autonomous systems: AS1, AS2, and AS3. As shown, AS3 includes a subnet with prefix x. For each AS, each router is either a gateway router or an internal router. A gateway router is a router on the edge of an AS that directly connects to one or more routers in other ASs. An internal router connects only to hosts and routers within its own AS. In AS1, for example, router 1c is a gateway router; routers 1a, 1b, and 1d are internal routers. Let's consider the task of advertising reachability information for prefix x to all of the routers shown in Figure 5.8. At a high level, this is straightforward. First, AS3 sends a BGP message to AS2, saying that x exists and is in AS3; let's denote this message as "AS3 x". Then AS2 sends a BGP message to AS1, saying that x exists and that you can get to x by first passing through AS2 and then going to AS3; let's denote that message as "AS2 AS3 x". In this manner, each of the autonomous systems will not only learn about the existence of x, but also learn about a path of autonomous systems that leads to x. Although the discussion in the above paragraph about advertising BGP reachability information should get the general idea across, it is not precise in the sense that autonomous systems do not actually send messages to each other, but instead routers do. To understand this, let's now re-examine the example in Figure 5.8. In BGP, Figure 5.8 Network with three autonomous systems. AS3 includes a subnet with prefix x pairs of routers exchange routing information over semi-permanent TCP connections using port 179. Each such TCP connection, along with all the BGP messages sent over the connection, is called a BGP connection. Furthermore, a BGP connection that spans two ASs is called an external BGP (eBGP) connection, and a BGP session between routers in the same AS is called an internal BGP (iBGP) connection. Examples of BGP connections for the network in Figure 5.8 are shown in Figure 5.9. There is typically one eBGP connection for each link that directly connects gateway routers in different ASs; thus, in Figure 5.9, there is an eBGP connection between gateway routers 1c and 2a and an eBGP connection between gateway routers 2c and 3a. There are also iBGP connections between routers within each of the ASs. In particular, Figure 5.9 displays a common configuration of one BGP connection for each pair of routers internal to an AS, creating a mesh of TCP connections within each AS. In Figure 5.9, the eBGP connections are shown with the long dashes; the iBGP connections are shown with the short dashes. Note that iBGP connections do not always correspond to physical links. In order to propagate the reachability information, both iBGP and eBGP sessions are used. Consider again advertising

the reachability information for prefix x to all routers in AS1 and AS2. In this process, gateway router 3a first sends an eBGP message “AS3 x” to gateway router 2c. Gateway router 2c then sends the iBGP message “AS3 x” to all of the other routers in AS2, including to gateway router 2a. Gateway router 2a then sends the eBGP message “AS2 AS3 x” to gateway router 1c. Figure 5.9 eBGP and iBGP connections Finally, gateway router 1c uses iBGP to send the message “AS2 AS3 x” to all the routers in AS1. After this process is complete, each router in AS1 and AS2 is aware of the existence of x and is also aware of an AS path that leads to x. Of course, in a real network, from a given router there may be many different paths to a given destination, each through a different sequence of ASs. For example, consider the network in Figure 5.10, which is the original network in Figure 5.8, with an additional physical link from router 1d to router 3d. In this case, there are two paths from AS1 to x: the path “AS2 AS3 x” via router 1c; and the new path “AS3 x” via the router 1d. 5.4.3 Determining the Best Routes As we have just learned, there may be many paths from a given router to a destination subnet. In fact, in the Internet, routers often receive reachability information about dozens of different possible paths. How does a router choose among these paths (and then configure its forwarding table accordingly)? Before addressing this critical question, we need to introduce a little more BGP terminology. When a router advertises a prefix across a BGP connection, it includes with the prefix several BGP attributes. In BGP jargon, a prefix along with its attributes is called a route. Two of the more important attributes are AS-PATH and NEXT-HOP. The AS-PATH attribute contains the list of ASs through which the Figure 5.10 Network augmented with peering link between AS1 and AS3 advertisement has passed, as we’ve seen in our examples above. To generate the AS-PATH value, when a prefix is passed to an AS, the AS adds its ASN to the existing list in the AS-PATH. For example, in Figure 5.10, there are two routes from AS1 to subnet x: one which uses the AS-PATH “AS2 AS3”; and another that uses the AS-PATH “A3”. BGP routers also use the AS-PATH attribute to detect and prevent looping advertisements; specifically, if a router sees that its own AS is contained in the path list, it will reject the advertisement. Providing the critical link between the inter-AS and intra-AS routing protocols, the NEXT-HOP attribute has a subtle but important use. The NEXT-HOP is the IP address of the router interface that begins the AS-PATH. To gain insight into this attribute, let’s again refer to Figure 5.10. As indicated in Figure 5.10, the NEXT-HOP attribute for the route “AS2 AS3 x” from AS1 to x that passes through AS2 is the IP address of the left interface on router 2a. The NEXT-HOP attribute for the route “AS3 x” from AS1 to x that bypasses AS2 is the IP address of the leftmost interface of router 3d. In summary, in this toy example, each router in AS1 becomes aware of two BGP routes to prefix x: IP address of leftmost interface for router 2a; AS2 AS3; x IP address of leftmost interface of router 3d; AS3; x Here, each BGP route is written as a list with three components: NEXT-HOP; AS-PATH; destination prefix. In practice, a BGP route includes additional attributes, which we will ignore for the time being. Note that the NEXT-HOP attribute is an IP address of a router that does not belong to AS1; however, the subnet that contains this IP address directly attaches to AS1. Hot Potato Routing We are now finally in position to talk about BGP routing algorithms in a precise manner. We will begin with one of the simplest routing algorithms, namely, hot potato routing. Consider router 1b in the network in Figure 5.10. As just described, this router will learn about two possible BGP routes to prefix x. In hot potato routing, the route chosen (from among all possible routes) is that route with the least cost to the NEXT-HOP router beginning that route. In this example, router 1b will consult its intra-AS routing information to find the least-cost intra-AS path to NEXT-HOP router 2a and the least-cost intra-AS path to NEXT-HOP router 3d, and then select the route with the smallest of these least-cost paths. For example, suppose that cost is defined as the number of links traversed. Then the least cost from router 1b to router 2a is 2, the least cost from router 1b to router 2d is 3, and router 2a would therefore be selected.

Router 1b would then consult its forwarding table (configured by its intra-AS algorithm) and find the interface I that is on the least-cost path to router 2a. It then adds (x, I) to its forwarding table. The steps for adding an outside-AS prefix in a router's forwarding table for hot potato routing are summarized in Figure 5.11. It is important to note that when adding an outside-AS prefix into a forwarding table, both the inter-AS routing protocol (BGP) and the intra-AS routing protocol (e.g., OSPF) are used. The idea behind hot-potato routing is for router 1b to get packets out of its AS as quickly as possible (more specifically, with the least cost possible) without worrying about the cost of the remaining portions of the path outside of its AS to the destination. In the name "hot potato routing," a packet is analogous to a hot potato that is burning in your hands. Because it is burning hot, you want to pass it off to another person (another AS) as quickly as possible. Hot potato routing is thus Figure 5.11 Steps in adding outside-AS destination in a router's forwarding table a selfish algorithm—it tries to reduce the cost in its own AS while ignoring the other components of the end-to-end costs outside its AS. Note that with hot potato routing, two routers in the same AS may choose two different AS paths to the same prefix. For example, we just saw that router 1b would send packets through AS2 to reach x. However, router 1d would bypass AS2 and send packets directly to AS3 to reach x. Route-Selection Algorithm In practice, BGP uses an algorithm that is more complicated than hot potato routing, but nevertheless incorporates hot potato routing. For any given destination prefix, the input into BGP's route-selection algorithm is the set of all routes to that prefix that have been learned and accepted by the router. If there is only one such route, then BGP obviously selects that route. If there are two or more routes to the same prefix, then BGP sequentially invokes the following elimination rules until one route remains: 1. A route is assigned a local preference value as one of its attributes (in addition to the AS-PATH and NEXT-HOP attributes). The local preference of a route could have been set by the router or could have been learned from another router in the same AS. The value of the local preference attribute is a policy decision that is left entirely up to the AS's network administrator. (We will shortly discuss BGP policy issues in some detail.) The routes with the highest local preference values are selected. 2. From the remaining routes (all with the same highest local preference value), the route with the shortest AS-PATH is selected. If this rule were the only rule for route selection, then BGP would be using a DV algorithm for path determination, where the distance metric uses the number of AS hops rather than the number of router hops. 3. From the remaining routes (all with the same highest local preference value and the same ASPATH length), hot potato routing is used, that is, the route with the closest NEXT-HOP router is selected. 4. If more than one route still remains, the router uses BGP identifiers to select the route; see [Stewart 1999]. As an example, let's again consider router 1b in Figure 5.10. Recall that there are exactly two BGP routes to prefix x, one that passes through AS2 and one that bypasses AS2. Also recall that if hot potato routing on its own were used, then BGP would route packets through AS2 to prefix x. But in the above route-selection algorithm, rule 2 is applied before rule 3, causing BGP to select the route that bypasses AS2, since that route has a shorter AS PATH. So we see that with the above route-selection algorithm, BGP is no longer a selfish algorithm—it first looks for routes with short AS paths (thereby likely reducing end-to-end delay). As noted above, BGP is the de facto standard for inter-AS routing for the Internet. To see the contents of various BGP routing tables (large!) extracted from routers in tier-1 ISPs, see http:// www.routeviews.org. BGP routing tables often contain over half a million routes (that is, prefixes and corresponding attributes). Statistics about the size and characteristics of BGP routing tables are presented in [Potaroo 2016]. 5.4.4 IP-Anycast In addition to being the Internet's inter-AS routing protocol, BGP is often used to implement the IPanycast service [RFC 1546, RFC 7094], which is commonly used in DNS. To motivate IP-anycast, consider that in many applications, we are interested in (1) replicating the same content on different servers in many different dispersed geographical locations, and (2) having each user access the content from the server that is closest. For example, a CDN may replicate videos and other objects on servers in different countries. Similarly, the DNS system can replicate DNS records on DNS servers throughout the world. When a user wants to access this replicated content, it is desirable to point the user to the “nearest” server with the replicated content. BGP’s route-selection algorithm provides an easy and natural mechanism for doing so. To make our discussion concrete, let’s describe how a CDN might use IP-anycast. As shown in Figure 5.12, during the IP-anycast configuration stage, the CDN company assigns the same IP address to each of its servers, and uses standard BGP to advertise this IP address from each of the servers. When a BGP router receives multiple route advertisements for this IP address, it treats these advertisements as providing different paths to the same physical location (when, in fact, the advertisements are for different paths to different physical locations). When configuring its routing table, each router will locally use the BGP route-selection algorithm to pick the “best” (for example, closest, as determined by AS-hop counts) route to that IP address. For example, if one BGP route (corresponding to one location) is only one AS hop away from the router, and all other BGP routes (corresponding to other locations) are two or more AS hops away, then the BGP router would choose to route packets to the location that is one hop away. After this initial BGP address-advertisement phase, the CDN can do its main job of distributing content. When a client requests the video, the CDN returns to the client the common IP address used by the geographically dispersed servers, no matter where the client is located. When the client sends a request to that IP address, Internet routers then forward the request packet to the “closest” server, as defined by the BGP route-selection algorithm. Although the above CDN example nicely illustrates how IP-anycast can be used, in practice CDNs generally choose not to use IP-anycast because BGP routing changes can result in different packets of the same TCP connection arriving at different instances of the Web server. But IP-anycast is extensively used by the DNS system to direct DNS queries to the closest root DNS server. Recall from Section 2.4, there are currently 13 IP addresses for root DNS servers. But corresponding Figure 5.12 Using IP-anycast to bring users to the closest CDN server to each of these addresses, there are multiple DNS root servers, with some of these addresses having over 100 DNS root servers scattered over all corners of the world. When a DNS query is sent to one of these 13 IP addresses, IP anycast is used to route the query to the nearest of the DNS root servers that is responsible for that address. 5.4.5 Routing Policy When a router selects a route to a destination, the AS routing policy can trump all other considerations, such as shortest AS path or hot potato routing. Indeed, in the route-selection algorithm, routes are first selected according to the local-preference attribute, whose value is fixed by the policy of the local AS. Let’s illustrate some of the basic concepts of BGP routing policy with a simple example. Figure 5.13 shows six interconnected autonomous systems: A, B, C, W, X, and Y. It is important to note that A, B, C, W, X, and Y are ASs, not routers. Let’s Figure 5.13 A simple BGP policy scenario assume that autonomous systems W, X, and Y are access ISPs and that A, B, and C are backbone provider networks. We’ll also assume that A, B, and C, directly send traffic to each other, and provide full BGP information to their customer networks. All traffic entering an ISP access network must be destined for that network, and all traffic leaving an ISP access network must have originated in that network. W and Y are clearly access ISPs. X is a multi-homed access ISP, since it is connected to the rest of the network via two different providers (a scenario that is becoming increasingly common in practice). However, like W and Y, X itself must be the source/destination of all traffic leaving/entering X. But how will this stub network behavior be implemented and enforced? How will X be prevented from forwarding traffic between B and C? This can easily be accomplished by controlling the manner in which BGP routes are advertised. In particular X will function as an access ISP network if it advertises (to its neighbors B and C) that it has no paths to any other destinations except itself. That is, even though X may know of a path, say XCY, that reaches network Y, it will not advertise this path to B. Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X. This simple example illustrates how a selective route advertisement policy can be used to implement customer/provider routing relationships. Let's next focus on a provider network, say AS B. Suppose that B has learned (from A) that A has a path AW to W. B can thus install the route AW into its routing information base. Clearly, B also wants to advertise the path BAW to its customer, X, so that X knows that it can route to W via B. But should B advertise the path BAW to C? If it does so, then C could route traffic to W via BAW. If A, B, and C are all backbone providers, than B might rightly feel that it should not have to shoulder the burden (and cost!) of carrying transit traffic between A and C. B might rightly feel that it is A's and C's job (and cost!) to make sure that C can route to/from A's customers via a direct connection between A and C. There are currently no official standards that govern how backbone ISPs route among themselves. However, a rule of thumb followed by commercial ISPs is that any traffic flowing across an ISP's backbone network must have either a source or a destination (or both) in a network that is a customer of that ISP; otherwise the traffic would be getting a free ride on the ISP's network. Individual peering agreements (that would govern questions such as PRINCIPLES IN PRACTICE WHY ARE THERE DIFFERENT INTER-AS AND INTRA-AS ROUTING PROTOCOLS? Having now studied the details of specific inter-AS and intra-AS routing protocols deployed in today's Internet, let's conclude by considering perhaps the most fundamental question we could ask about these protocols in the first place (hopefully, you have been wondering this all along, and have not lost the forest for the trees!): Why are different inter-AS and intra-AS routing protocols used? The answer to this question gets at the heart of the differences between the goals of routing within an AS and among ASs: Policy. Among ASs, policy issues dominate. It may well be important that traffic originating in a given AS not be able to pass through another specific AS. Similarly, a given AS may well want to control what transit traffic it carries between other ASs. We have seen that BGP carries path attributes and provides for controlled distribution of routing information so that such policy-based routing decisions can be made. Within an AS, everything is nominally under the same administrative control, and thus policy issues play a much less important role in choosing routes within the AS. Scale. The ability of a routing algorithm and its data structures to scale to handle routing to/among large numbers of networks is a critical issue in inter-AS routing. Within an AS, scalability is less of a concern. For one thing, if a single ISP becomes too large, it is always possible to divide it into two ASs and perform inter-AS routing between the two new ASs. (Recall that OSPF allows such a hierarchy to be built by splitting an AS into areas.) Performance. Because inter-AS routing is so policy oriented, the quality (for example, performance) of the routes used is often of secondary concern (that is, a longer or more costly route that satisfies certain policy criteria may well be taken over a route that is shorter but does not meet that criteria). Indeed, we saw that among ASs, there is not even the notion of cost (other than AS hop count) associated with routes. Within a single AS, however, such policy concerns are of less importance, allowing routing to focus more on the level of performance realized on a route. those raised above) are typically negotiated between pairs of ISPs and are often confidential; [Huston 1999a] provides an interesting discussion of peering agreements. For a detailed description of how routing policy reflects commercial relationships among ISPs, see [Gao 2001; Dmitiropoulos 2007]. For a discussion of BGP routing polices from an ISP standpoint, see [Caesar 2005b]. This completes our brief introduction to BGP. Understanding BGP is important because it plays a central role in the Internet. We encourage you to see the references [Griffin 2012; Stewart 1999;

Labovitz 1997; Halabi 2000; Huitema 1998; Gao 2001; Feamster 2004; Caesar 2005b; Li 2007] to learn more about BGP. 5.4.6 Putting the Pieces Together: Obtaining Internet Presence Although this subsection is not about BGP per se, it brings together many of the protocols and concepts we've seen thus far, including IP addressing, DNS, and BGP. Suppose you have just created a small company that has a number of servers, including a public Web server that describes your company's products and services, a mail server from which your employees obtain their e-mail messages, and a DNS server. Naturally, you would like the entire world to be able to visit your Web site in order to learn about your exciting products and services. Moreover, you would like your employees to be able to send and receive e-mail to potential customers throughout the world. To meet these goals, you first need to obtain Internet connectivity, which is done by contracting with, and connecting to, a local ISP. Your company will have a gateway router, which will be connected to a router in your local ISP. This connection might be a DSL connection through the existing telephone infrastructure, a leased line to the ISP's router, or one of the many other access solutions described in Chapter 1. Your local ISP will also provide you with an IP address range, e.g., a /24 address range consisting of 256 addresses. Once you have your physical connectivity and your IP address range, you will assign one of the IP addresses (in your address range) to your Web server, one to your mail server, one to your DNS server, one to your gateway router, and other IP addresses to other servers and networking devices in your company's network. In addition to contracting with an ISP, you will also need to contract with an Internet registrar to obtain a domain name for your company, as described in Chapter 2. For example, if your company's name is, say, Xanadu Inc., you will naturally try to obtain the domain name xanadu.com. Your company must also obtain presence in the DNS system. Specifically, because outsiders will want to contact your DNS server to obtain the IP addresses of your servers, you will also need to provide your registrar with the IP address of your DNS server. Your registrar will then put an entry for your DNS server (domain name and corresponding IP address) in the .com top-level-domain servers, as described in Chapter 2. After this step is completed, any user who knows your domain name (e.g., xanadu.com) will be able to obtain the IP address of your DNS server via the DNS system. So that people can discover the IP addresses of your Web server, in your DNS server you will need to include entries that map the host name of your Web server (e.g., www.xanadu.com) to its IP address. You will want to have similar entries for other publicly available servers in your company, including your mail server. In this manner, if Alice wants to browse your Web server, the DNS system will contact your DNS server, find the IP address of your Web server, and give it to Alice. Alice can then establish a TCP connection directly with your Web server. However, there still remains one other necessary and crucial step to allow outsiders from around the world to access your Web server. Consider what happens when Alice, who knows the IP address of your Web server, sends an IP datagram (e.g., a TCP SYN segment) to that IP address. This datagram will be routed through the Internet, visiting a series of routers in many different ASs, and eventually reach your Web server. When any one of the routers receives the datagram, it is going to look for an entry in its forwarding table to determine on which outgoing port it should forward the datagram. Therefore, each of the routers needs to know about the existence of your company's /24 prefix (or some aggregate entry). How does a router become aware of your company's prefix? As we have just seen, it becomes aware of it from BGP! Specifically, when your company contracts with a local ISP and gets assigned a prefix (i.e., an address range), your local ISP will use BGP to advertise your prefix to the ISPs to which it connects. Those ISPs will then, in turn, use BGP to propagate the advertisement. Eventually, all Internet routers will know about your prefix (or about some aggregate that includes your prefix) and thus be able to appropriately forward datagrams destined to your Web and mail servers. 5.5 The SDN Control

Plane In this section, we'll dive into the SDN control plane—the network-wide logic that controls packet forwarding among a network's SDN-enabled devices, as well as the configuration and management of these devices and their services. Our study here builds on our earlier discussion of generalized SDN forwarding in Section 4.4, so you might want to first review that section, as well as Section 5.1 of this chapter, before continuing on. As in Section 4.4, we'll again adopt the terminology used in the SDN literature and refer to the network's forwarding devices as "packet switches" (or just switches, with "packet" being understood), since forwarding decisions can be made on the basis of network-layer source/destination addresses, link-layer source/destination addresses, as well as many other values in transport-, network-, and link-layer packet-header fields. Four key characteristics of an SDN architecture can be identified [Kreutz 2015]: Flow-based forwarding. Packet forwarding by SDN-controlled switches can be based on any number of header field values in the transport-layer, network-layer, or link-layer header. We saw in Section 4.4 that the OpenFlow1.0 abstraction allows forwarding based on eleven different header field values. This contrasts sharply with the traditional approach to router-based forwarding that we studied in Sections 5.2–5.4, where forwarding of IP datagrams was based solely on a datagram's destination IP address. Recall from Figure 5.2 that packet forwarding rules are specified in a switch's flow table; it is the job of the SDN control plane to compute, manage and install flow table entries in all of the network's switches. Separation of data plane and control plane. This separation is shown clearly in Figures 5.2 and 5.14. The data plane consists of the network's switches— relatively simple (but fast) devices that execute the "match plus action" rules in their flow tables. The control plane consists of servers and software that determine and manage the switches' flow tables. Network control functions: external to data-plane switches. Given that the "S" in SDN is for "software," it's perhaps not surprising that the SDN control plane is implemented in software. Unlike traditional routers, however, this software executes on servers that are both distinct and remote from the network's switches. As shown in Figure 5.14, the control plane itself consists of two components —an SDN controller (or network operating system [Gude 2008]) and a set of network-control applications. The controller maintains accurate network state information (e.g., the state of remote links, switches, and hosts); provides this information to the network-control applications running in the control plane; and provides the means through which these applications can monitor, program, and control the underlying network devices. Although the controller in Figure 5.14 is shown as a single central server, in practice the controller is only logically centralized; it is typically implemented on several servers that provide coordinated, scalable performance and high availability. A programmable network. The network is programmable through the network-control applications running in the control plane. These applications represent the "brains" of the SDN control plane, using the APIs provided by the SDN controller to specify and control the data plane in the network devices. For example, a routing network-control application might determine the end-end paths between sources and destinations (e.g., by executing Dijkstra's algorithm using the node-state and link-state information maintained by the SDN controller). Another network application might perform access control, i.e., determine which packets are to be blocked at a switch, as in our third example in Section 4.4.3. Yet another application might forward packets in a manner that performs server load balancing (the second example we considered in Section 4.4.3). From this discussion, we can see that SDN represents a significant "unbundling" of network functionality —data plane switches, SDN controllers, and network-control applications are separate entities that may each be provided by different vendors and organizations. This contrasts with the pre-SDN model in which a switch/router (together with its embedded control plane software and protocol implementations) was monolithic, vertically integrated, and sold by a single vendor.

This unbundling of network functionality in SDN has been likened to the earlier evolution from mainframe computers (where hardware, system software, and applications were provided by a single vendor) to personal computers (with their separate hardware, operating systems, and applications). The unbundling of computing hardware, system software, and applications has arguably led to a rich, open ecosystem driven by innovation in all three of these areas; one hope for SDN is that it too will lead to a such rich innovation. Given our understanding of the SDN architecture of Figure 5.14, many questions naturally arise. How and where are the flow tables actually computed? How are these tables updated in response to events at SDN-controlled devices (e.g., an attached link going up/down)? And how are the flow table entries at multiple switches coordinated in such a way as to result in orchestrated and consistent network-wide functionality (e.g., end-to-end paths for forwarding packets from sources to destinations, or coordinated distributed firewalls)? It is the role of the SDN control plane to provide these, and many other, capabilities. Figure 5.14 Components of the SDN architecture: SDN-controlled switches, the SDN controller, network-control applications 5.5.2 The SDN Control Plane: SDN Controller and SDN Network-control Applications Let's begin our discussion of the SDN control plane in the abstract, by considering the generic capabilities that the control plane must provide. As we'll see, this abstract, "first principles" approach will lead us to an overall architecture that reflects how SDN control planes have been implemented in practice. As noted above, the SDN control plane divides broadly into two components—the SDN controller and the SDN network-control applications. Let's explore the controller first. Many SDN controllers have been developed since the earliest SDN controller [Gude 2008]; see [Kreutz 2015] for an extremely thorough and up-to-date survey. Figure 5.15 provides a more detailed view of a generic SDN controller. A controller's functionality can be broadly organized into three layers. Let's consider these layers in an uncharacteristically bottom-up fashion: A communication layer: communicating between the SDN controller and controlled network devices. Clearly, if an SDN controller is going to control the operation of a remote SDN-enabled switch, host, or other device, a protocol is needed to transfer information between the controller and that device. In addition, a device must be able to communicate locally-observed events to the controller (e.g., a message indicating that an attached link has gone up or down, that a device has just joined the network, or a heartbeat indicating that a device is up and operational). These events provide the SDN controller with an up-to-date view of the network's state. This protocol constitutes the lowest layer of the controller architecture, as shown in Figure 5.15. The communication between the controller and the controlled devices cross what has come to be known as the controller's "southbound" interface. In Section 5.5.2, we'll study OpenFlow—a specific protocol that provides this communication functionality. OpenFlow is implemented in most, if not all, SDN controllers. A network-wide state-management layer. The ultimate control decisions made by the SDN control plane—e.g., configuring flow tables in all switches to achieve the desired end-end forwarding, to implement load balancing, or to implement a particular firewalling capability—will require that the controller have up-to-date information about state of the networks' hosts, links, switches, and other SDN-controlled devices. A switch's flow table contains counters whose values might also be profitably used by network-control applications; these values should thus be available to the applications. Since the ultimate aim of the control plane is to determine flow tables for the various controlled devices, a controller might also maintain a copy of these tables. These pieces of information all constitute examples of the network-wide "state" maintained by the SDN controller. The interface to the network-control application layer. The controller interacts with networkcontrol applications through its "northbound" interface. This API Figure 5.15 Components of an SDN controller allows network-control applications to read/write network state and flow tables within the statemanagement layer. Applications can register to be notified when state-change events occur, so that they can take actions in response to network event notifications sent from SDN-controlled devices. Different types of APIs may be provided; we'll see that two popular SDN controllers communicate with their applications using a REST [Fielding 2000] request-response interface. We have noted several times that an SDN controller can be considered to be "logically centralized," i.e., that the controller may be viewed externally (e.g., from the point of view of SDN-controlled devices and external network-control applications) as a single, monolithic service. However, these services and the databases used to hold state information are implemented in practice by a distributed set of servers for fault tolerance, high availability, or for performance reasons. With controller functions being implemented by a set of servers, the semantics of the controller's internal operations (e.g., maintaining logical time ordering of events, consistency, consensus, and more) must be considered [Panda 2013]. Such concerns are common across many different distributed systems; see [Lamport 1989, Lampson 1996] for elegant solutions to these challenges. Modern controllers such as OpenDaylight [OpenDaylight Lithium 2016] and ONOS [ONOS 2016] (see sidebar) have placed considerable emphasis on architecting a logically centralized but physically distributed controller platform that provides scalable services and high availability to the controlled devices and network-control applications alike. The architecture depicted in Figure 5.15 closely resembles the architecture of the originally proposed NOX controller in 2008 [Gude 2008], as well as that of today's OpenDaylight [OpenDaylight Lithium 2016] and ONOS [ONOS 2016] SDN controllers (see sidebar). We'll cover an example of controller operation in Section 5.5.3. First, however, let's examine the OpenFlow protocol, which lies in the controller's communication layer. 5.5.2 OpenFlow Protocol The OpenFlow protocol [OpenFlow 2009, ONF 2016] operates between an SDN controller and an SDN-controlled switch or other device implementing the OpenFlow API that we studied earlier in Section 4.4. The OpenFlow protocol operates over TCP, with a default port number of 6653. Among the important messages flowing from the controller to the controlled switch are the following: Configuration. This message allows the controller to query and set a switch's configuration parameters. Modify-State. This message is used by a controller to add/delete or modify entries in the switch's flow table, and to set switch port properties. Read-State. This message is used by a controller to collect statistics and counter values from the switch's flow table and ports. Send-Packet. This message is used by the controller to send a specific packet out of a specified port at the controlled switch. The message itself contains the packet to be sent in its payload. Among the messages flowing from the SDN-controlled switch to the controller are the following: Flow-Removed. This message informs the controller that a flow table entry has been removed, for example by a timeout or as the result of a received modify-state message. Port-status. This message is used by a switch to inform the controller of a change in port status. Packet-in. Recall from Section 4.4 that a packet arriving at a switch port and not matching any flow table entry is sent to the controller for additional processing. Matched packets may also be sent to the controller, as an action to be taken on a match. The packet-in message is used to send such packets to the controller. Additional OpenFlow messages are defined in [OpenFlow 2009, ONF 2016]. Principles in Practice Google's Software-Defined Global Network Recall from the case study in Section 2.6 that Google deploys a dedicated wide-area network (WAN) that interconnects its data centers and server clusters (in IXPs and ISPs). This network, called B4, has a Google-designed SDN control plane built on OpenFlow. Google's network is able to drive WAN links at near 70% utilization over the long run (a two to three fold increase over typical link utilizations) and split application flows among multiple paths based on application priority and existing flow demands [Jain 2013]. The Google B4 network is particularly it well-suited for SDN: (i) Google controls all devices from the edge servers in IXPs and ISPs to routers in their network core; (ii) the most bandwidthintensive applications are large-scale data copies between sites that can defer to higher-priority interactive applications during times of resource congestion; (iii) with only a few dozen data centers being connected, centralized control is feasible. Google's B4 network uses custom-built switches, each implementing a slightly extended version of OpenFlow, with a local Open Flow Agent (OFA) that is similar in spirit to the control agent we encountered in Figure 5.2. Each OFA in turn connects to an Open Flow Controller (OFC) in the network control server (NCS), using a separate "out of band" network, distinct from the network that carries data-center traffic between data centers. The OFC thus provides the services used by the NCS to communicate with its controlled switches, similar in spirit to the lowest layer in the SDN architecture shown in Figure 5.15. In B4, the OFC also performs state management functions, keeping node and link status in a Network Information Base (NIB). Google's implementation of the OFC is based on the ONIX SDN controller [Koponen 2010]. Two routing protocols, BGP (for routing between the data centers) and IS-IS (a close relative of OSPF, for routing within a data center), are implemented. Paxos [Chandra 2007] is used to execute hot replicas of NCS components to protect against failure. A traffic engineering network-control application, sitting logically above the set of network control servers, interacts with these servers to provide global, network-wide bandwidth provisioning for groups of application flows. With B4, SDN made an important leap forward into the operational networks of a global network provider. See [Jain 2013] for a detailed description of B4. 5.5.3 Data and Control Plane Interaction: An Example In order to solidify our understanding of the interaction between SDN-controlled switches and the SDN controller, let's consider the example shown in Figure 5.16, in which Dijkstra's algorithm (which we studied in Section 5.2) is used to determine shortest path routes. The SDN scenario in Figure 5.16 has two important differences from the earlier per-router-control scenario of Sections 5.2.1 and 5.3, where Dijkstra's algorithm was implemented in each and every router and link-state updates were flooded among all network routers: Dijkstra's algorithm is executed as a separate application, outside of the packet switches. Packet switches send link updates to the SDN controller and not to each other. In this example, let's assume that the link between switch s1 and s2 goes down; that shortest path routing is implemented, and consequently and that incoming and outgoing flow forwarding rules at s1, s3, and s4 are affected, but that s2's Figure 5.16 SDN controller scenario: Link-state change operation is unchanged. Let's also assume that OpenFlow is used as the communication layer protocol, and that the control plane performs no other function other than link-state routing. 1. Switch s1, experiencing a link failure between itself and s2, notifies the SDN controller of the link-state change using the OpenFlow port-status message. 2. The SDN controller receives the OpenFlow message indicating the link-state change, and notifies the link-state manager, which updates a link-state database. 3. The network-control application that implements Dijkstra's link-state routing has previously registered to be notified when link state changes. That application receives the notification of the link-state change. 4. The link-state routing application interacts with the link-state manager to get updated link state; it might also consult other components in the state-management layer. It then computes the new least-cost paths. 5. The link-state routing application then interacts with the flow table manager, which determines the flow tables to be updated. 6. The flow table manager then uses the OpenFlow protocol to update flow table entries at affected switches—s1 (which will now route packets destined to s2 via s4), s2 (which will now begin receiving packets from s1 via intermediate switch s4), and s4 (which must now forward packets from s1 destined to s2). This example is simple but illustrates how the SDN control plane provides control-plane services (in this case network-layer routing) that had been previously implemented with per-router control exercised in each and every network router. One can now easily appreciate how an SDN-enabled ISP could easily switch from least-cost path routing to a more hand-tailored approach to routing. Indeed, since the controller can tailor the flow tables as it pleases, it can implement any form of forwarding that it pleases —simply by changing its application-control software. This ease of change should be contrasted to the case of a traditional per-router control plane, where software in all routers (which might be provided to the ISP by multiple independent vendors) must be changed. 5.5.4 SDN: Past and Future Although the intense interest in SDN is a relatively recent phenomenon, the technical roots of SDN, and the separation of the data and control planes in particular, go back considerably further. In 2004, [Feamster 2004, Lakshman 2004, RFC 3746] all argued for the separation of the network's data and control planes. [van der Merwe 1998] describes a control framework for ATM networks [Black 1995] with multiple controllers, each controlling a number of ATM switches. The Ethane project [Casado 2007] pioneered the notion of a network of simple flow-based Ethernet switches with match-plus-action flow tables, a centralized controller that managed flow admission and routing, and the forwarding of unmatched packets from the switch to the controller. A network of more than 300 Ethane switches was operational in 2007. Ethane quickly evolved into the OpenFlow project, and the rest (as the saying goes) is history! Numerous research efforts are aimed at developing future SDN architectures and capabilities. As we have seen, the SDN revolution is leading to the disruptive replacement of dedicated monolithic switches and routers (with both data and control planes) by simple commodity switching hardware and a sophisticated software control plane. A generalization of SDN known as network functions virtualization (NFV) similarly aims at disruptive replacement of sophisticated middleboxes (such as middleboxes with dedicated hardware and proprietary software for media caching/service) with simple commodity servers, switching, and storage [Gember-Jacobson 2014]. A second area of important research seeks to extend SDN concepts from the intra-AS setting to the inter-AS setting [Gupta 2014]. PRINCIPLES IN PRACTICE SDN Controller Case Studies: The OpenDaylight and ONOS Controllers In the earliest days of SDN, there was a single SDN protocol (OpenFlow [McKeown 2008; OpenFlow 2009]) and a single SDN controller (NOX [Gude 2008]). Since then, the number of SDN controllers in particular has grown significantly [Kreutz 2015]. Some SDN controllers are company-specific and proprietary, e.g., ONIX [Koponen 2010], Juniper Networks Contrail [Juniper Contrail 2016], and Google's controller [Jain 2013] for its B4 wide-area network. But many more controllers are open-source and implemented in a variety of programming languages [Erickson 2013]. Most recently, the OpenDaylight controller [OpenDaylight Lithium 2016] and the ONOS controller [ONOS 2016] have found considerable industry support. They are both open-source and are being developed in partnership with the Linux Foundation. The OpenDaylight Controller Figure 5.17 presents a simplified view of the OpenDaylight Lithium SDN controller platform [OpenDaylight Lithium 2016]. ODL's main set of controller components correspond closely to those we developed in Figure 5.15. Network-Service Applications are the applications that determine how data-plane forwarding and other services, such as firewalling and load balancing, are accomplished in the controlled switches. Unlike the canonical controller in Figure 5.15, the ODL controller has two interfaces through which applications may communicate with native controller services and each other: external applications communicate with controller modules using a REST request-response API running over HTTP. Internal applications communicate with each other via the Service Abstraction Layer (SAL). The choice as to whether a controller application is implemented externally or internally is up to the application designer; Figure 5.17 The OpenDaylight controller the particular configuration of applications shown in Figure 5.17 is only meant as an example. ODL's Basic Network-Service Functions are at the heart of the controller, and they correspond closely to the network-wide state management capabilities that we encountered in Figure 5.15. The SAL is the controller's nerve center, allowing controller components and applications to invoke each other's services and to subscribe to events they generate. It also provides a uniform abstract interface to the specific underlying communications protocols in the communication layer, including OpenFlow and SNMP (the Simple Network Management Protocol—a network management protocol that we will cover in Section 5.7). OVSDB is a protocol used to manage data center switching, an important application area for SDN technology. We'll introduce data center networking in Chapter 6. Figure 5.18 ONOS controller architecture The ONOS Controller Figure 5.18 presents a simplified view of the ONOS controller ONOS 2016]. Similar to the canonical controller in Figure 5.15, three layers can be identified in the ONOS controller: Northbound abstractions and protocols. A unique feature of ONOS is its intent framework, which allows an application to request a high-level service (e.g., to setup a connection between host A and Host B, or conversely to not allow Host A and host B to communicate) without having to know the details of how this service is performed. State information is provided to network-control applications across the northbound API either synchronously (via query) or asynchronously (via listener callbacks, e.g., when network state changes). Distributed core. The state of the network's links, hosts, and devices is maintained in ONOS's distributed core. ONOS is deployed as a service on a set of interconnected servers, with each server running an identical copy of the ONOS software; an increased number of servers offers an increased service capacity. The ONOS core provides the mechanisms for service replication and coordination among instances, providing the applications above and the network devices below with the abstraction of logically centralized core services. Southbound abstractions and protocols. The southbound abstractions mask the heterogeneity of the underlying hosts, links, switches, and protocols, allowing the distributed core to be both device and protocol agnostic. Because of this abstraction, the southbound interface below the distributed core is logically higher than in our canonical controller in Figure 5.14 or the ODL controller in Figure 5.17. 5.6 ICMP: The Internet Control Message Protocol The Internet Control Message Protocol (ICMP), specified in [RFC 792], is used by hosts and routers to communicate network-layer information to each other. The most typical use of ICMP is for error reporting. For example, when running an HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your HTTP request. That router created and sent an ICMP message to your host indicating the error. ICMP is often considered part of IP, but architecturally it lies just above IP, as ICMP messages are carried inside IP datagrams. That is, ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP payload. Similarly, when a host receives an IP datagram with ICMP specified as the upper-layer protocol (an upper-layer protocol number of 1), it demultiplexes the datagram's contents to ICMP, just as it would demultiplex a datagram's content to TCP or UDP. ICMP messages have a type and a code field, and contain the header and the first 8 bytes of the IP datagram that caused the ICMP message to be generated in the first place (so that the sender can determine the datagram that caused the error). Selected ICMP message types are shown in Figure 5.19. Note that ICMP messages are used not only for signaling error conditions. The well-known ping program sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request, sends back a type 0 code 0 ICMP echo reply. Most TCP/IP implementations support the ping server directly in the operating system; that is, the server is not a process. Chapter 11 of [Stevens 1990] provides the source code for the ping client program. Note that the client program needs to be able to instruct the operating system to generate an ICMP message of type 8 code 0. Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control—to allow a congested router to send an ICMP source quench message to a host to force Figure 5.19 ICMP message types that host to reduce its transmission rate. We have seen in Chapter 3 that TCP has its own congestioncontrol mechanism that operates at the transport layer, without the use of network-layer feedback such as the ICMP source quench message. In Chapter 1 we introduced the Traceroute program, which allows us to trace a route from a host to any other host in the world. Interestingly, Traceroute is implemented with ICMP messages. To determine the names and addresses of the routers between source and destination, Traceroute in the source sends a series of ordinary IP datagrams to the destination. Each of these datagrams carries a UDP segment with an unlikely UDP port number. The first of these datagrams has a TTL of 1, the second of 2, the third of 3, and so on. The source also starts timers for each of the datagrams. When the nth datagram arrives at the nth router, the nth router observes that the TTL of the datagram has just expired. According to the rules of the IP protocol, the router discards the datagram and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When this ICMP message arrives back at the source, the source obtains the round-trip time from the timer and the name and IP address of the nth router from the ICMP message. How does a Traceroute source know when to stop sending UDP segments? Recall that the source increments the TTL field for each datagram it sends. Thus, one of the datagrams will eventually make it all the way to the destination host. Because this datagram contains a UDP segment with an unlikely port number, the destination host sends a port unreachable ICMP message (type 3 code 3) back to the source. When the source host receives this particular ICMP message, it knows it does not need to send additional probe packets. (The standard Traceroute program actually sends sets of three packets with the same TTL; thus the Traceroute output provides three results for each TTL.) In this manner, the source host learns the number and the identities of routers that lie between it and the destination host and the round-trip time between the two hosts. Note that the Traceroute client program must be able to instruct the operating system to generate UDP datagrams with specific TTL values and must also be able to be notified by its operating system when ICMP messages arrive. Now that you understand how Traceroute works, you may want to go back and play with it some more. A new version of ICMP has been defined for IPv6 in RFC 4443. In addition to reorganizing the existing ICMP type and code definitions, ICMPv6 also added new types and codes required by the new IPv6 functionality. These include the “Packet Too Big” type and an “unrecognized IPv6 options” error code. 5.7 Network Management and SNMP Having now made our way to the end of our study of the network layer, with only the link-layer before us, we’re well aware that a network consists of many complex, interacting pieces of hardware and software —from the links, switches, routers, hosts, and other devices that comprise the physical components of the network to the many protocols that control and coordinate these devices. When hundreds or thousands of such components are brought together by an organization to form a network, the job of the network administrator to keep the network “up and running” is surely a challenge. We saw in Section 5.5 that the logically centralized controller can help with this process in an SDN context. But the challenge of network management has been around long before SDN, with a rich set of network management tools and approaches that help the network administrator monitor, manage, and control the network. We’ll study these tools and techniques in this section. An often-asked question is “What is network management?” A well-conceived, single-sentence (albeit a rather long run-on sentence) definition of network management from [Saydam 1996] is: Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost. Given this broad definition, we'll cover only the rudiments of network management in this section—the architecture, protocols, and information base used by a network administrator in performing their task. We'll not cover the administrator's decision-making processes, where topics such as fault identification [Labovitz 1997; Steinder 2002; Feamster 2005; Wu 2005; Teixeira 2006], anomaly detection [Lakhina 2005; Barford 2009], network design/engineering to meet contracted Service Level Agreements (SLA's) [Huston 1999a], and more come into consideration. Our focus is thus purposefully narrow; the interested reader should consult these references, the excellent network-management text by Subramanian [Subramanian 2000], and the more detailed treatment of network management available on the Web site for this text. 5.7.1 The Network Management Framework Figure 5.20 shows the key components of network management: The managing server is an application, typically with a human in the loop, running in a centralized network management station in the network operations center (NOC). The managing server is the locus of activity for network management; it controls the collection, processing, analysis, and/or display of network management information. It is here that actions are initiated to control network behavior and here that the human network administrator interacts with the network's devices. A managed device is a piece of network equipment (including its software) that resides on a managed network. A managed device might be a host, router, switch, middlebox, modem, thermometer, or other network-connected device. There may be several so-called managed objects within a managed device. These managed objects are the actual pieces of hardware within the managed device (for example, a network interface card is but one component of a host or router), and configuration parameters for these hardware and software components (for example, an intraAS routing protocol such as OSPF). Each managed object within a managed device associated information that is collected into a Management Information Base (MIB); we'll see that the values of these pieces of information are available to (and in many cases able to be set by) the managing server. A MIB object might be a counter, such as the number of IP datagrams discarded at a router due to errors in an IP datagram header, or the number of UDP segments received at a host; descriptive information such as the version of the software running on a DNS server; status information such as whether a particular device is functioning correctly; or protocol-specific information such as a routing path to a destination. MIB objects are specified in a data description language known as SMI (Structure of Management Information) [RFC 2578; RFC 2579; RFC 2580]. A formal definition language is used to ensure that the syntax and semantics of the network management data are well defined and unambiguous. Related MIB objects are gathered into MIB modules. As of mid-2015, there were nearly 400 MIB modules defined by RFCs, and a much larger number of vendor-specific (private) MIB modules. Also resident in each managed device is a network management agent, a process running in the managed device that communicates with the managing server, Figure 5.20 Elements of network management: Managing server, managed devices, MIB data, remote agents, SNMP taking local actions at the managed device under the command and control of the managing server. The network management agent is similar to the routing agent that we saw in Figure 5.2. The final component of a network management framework is the network management protocol. The protocol runs between the managing server and the managed devices, allowing the managing server to query the status of managed devices and indirectly take actions at these devices via its agents. Agents can use the network management protocol to inform the managing server of exceptional events (for example, component failures or violation of performance thresholds). It's important to note that the network management protocol does not itself manage the network. Instead, it provides capabilities that a network administrator can use to manage (“monitor, test, poll, configure, analyze, evaluate, and control”) the network. This is a subtle, but important, distinction. In the following section, we’ll cover the Internet’s SNMP (Simple Network Management Protocol) protocol. 5.7.2 The Simple Network Management Protocol (SNMP) The Simple Network Management Protocol version 2 (SNMPv2) [RFC 3416] is an application-layer protocol used to convey network-management control and information messages between a managing server and an agent executing on behalf of that managing server. The most common usage of SNMP is in a request-response mode in which an SNMP managing server sends a request to an SNMP agent, who receives the request, performs some action, and sends a reply to the request. Typically, a request will be used to query (retrieve) or modify (set) MIB object values associated with a managed device. A second common usage of SNMP is for an agent to send an unsolicited message, known as a trap message, to a managing server. Trap messages are used to notify a managing server of an exceptional situation (e.g., a link interface going up or down) that has resulted in changes to MIB object values. SNMPv2 defines seven types of messages, known generically as protocol data units—PDUs—as shown in Table 5.2 and described below. The format of the PDU is shown in Figure 5.21. The GetRequest , GetNextRequest, and GetBulkRequest PDUs are all sent from a managing server to an agent to request the value of one or more MIB objects at the agent’s managed device. The MIB objects whose values are being Table 5.2 SNMPv2 PDU types SNMPv2 PDU Type Sender-receiver Description GetRequest manager-toagent get value of one or more MIB object instances GetNextRequest manager-toagent get value of next MIB object instance in list or table GetBulkRequest manager-toagent get values in large block of data, for example, values in a large table InformRequest manager-tomanager inform remote managing entity of MIB values remote to its access SetRequest manager-toagent set value of one or more MIB object instances Response agent-tomanager or generated in response to manager-tomanager GetRequest, GetNextRequest, GetBulkRequest, SetRequest PDU, or InformRequest SNMPv2-Trap agent-tomanager inform manager of an exceptional event # Figure 5.21 SNMP PDU format requested are specified in the variable binding portion of the PDU. GetRequest , GetNextRequest , and GetBulkRequest differ in the granularity of their data requests. GetRequest can request an arbitrary set of MIB values; multiple GetNextRequest s can be used to sequence through a list or table of MIB objects; GetBulkRequest allows a large block of data to be returned, avoiding the overhead incurred if multiple GetRequest or GetNextRequest messages were to be sent. In all three cases, the agent responds with a Response PDU containing the object identifiers and their associated values. The SetRequest PDU is used by a managing server to set the value of one or more MIB objects in a managed device. An agent replies with a Response PDU with the “noError” error status to confirm that the value has indeed been set. The InformRequest PDU is used by a managing server to notify another managing server of MIB information that is remote to the receiving server. The Response PDU is typically sent from a managed device to the managing server in response to a request message from that server, returning the requested information. The final type of SNMPv2 PDU is the trap message. Trap messages are generated asynchronously; that is, they are not generated in response to a received request but rather in response to an event for which the managing server requires notification. RFC 3418 defines well-known trap types that include a cold or warm start by a device, a link going up or down, the loss of a neighbor, or an authentication failure event. A received trap request has no required response from a managing server. Given the request-response nature of SNMP, it is worth noting here that although SNMP PDUs can be carried via many different transport protocols, the SNMP PDU is typically carried in the payload of a UDP datagram. Indeed, RFC 3417 states that UDP is “the preferred transport mapping.” However, since UDP is an unreliable transport protocol, there is no guarantee that a request, or its response, will be received at the intended destination. The request ID field of the PDU (see Figure 5.21) is used by the managing server to number its requests to an agent; the agent's response takes its request ID from that of the received request. Thus, the request ID field can be used by the managing server to detect lost requests or replies. It is up to the managing server to decide whether to retransmit a request if no corresponding response is received after a given amount of time. In particular, the SNMP standard does not mandate any particular procedure for retransmission, or even if retransmission is to be done in the first place. It only requires that the managing server "needs to act responsibly in respect to the frequency and duration of retransmissions." This, of course, leads one to wonder how a "responsible" protocol should act! SNMP has evolved through three versions. The designers of SNMPv3 have said that "SNMPv3 can be thought of as SNMPv2 with additional security and administration capabilities" [RFC 3410]. Certainly, there are changes in SNMPv3 over SNMPv2, but nowhere are those changes more evident than in the area of administration and security. The central role of security in SNMPv3 was particularly important, since the lack of adequate security resulted in SNMP being used primarily for monitoring rather than control (for example, SetRequest is rarely used in SNMPv1). Once again, we see that security—a topic we'll cover in detail in Chapter 8 — is of critical concern, but once again a concern whose importance had been realized perhaps a bit late and only then "added on." 5.7 Summary We have now completed our two-chapter journey into the network core—a journey that began with our study of the network layer's data plane in Chapter 4 and finished here with our study of the network layer's control plane. We learned that the control plane is the network-wide logic that controls not only how a datagram is forwarded among routers along an end-to-end path from the source host to the destination host, but also how network-layer components and services are configured and managed. We learned that there are two broad approaches towards building a control plane: traditional per-router control (where a routing algorithm runs in each and every router and the routing component in the router communicates with the routing components in other routers) and software-defined networking (SDN) control (where a logically centralized controller computes and distributes the forwarding tables to be used by each and every router). We studied two fundamental routing algorithms for computing least cost paths in a graph—link-state routing and distance-vector routing—in Section 5.2; these algorithms find application in both per-router control and in SDN control. These algorithms are the basis for two widelydeployed Internet routing protocols, OSPF and BGP, that we covered in Sections 5.3 and 5.4. We covered the SDN approach to the network-layer control plane in Section 5.5, investigating SDN network-control applications, the SDN controller, and the OpenFlow protocol for communicating between the controller and SDN-controlled devices. In Sections 5.6 and 5.7, we covered some of the nuts and bolts of managing an IP network: ICMP (the Internet Control Message Protocol) and SNMP (the Simple Network Management Protocol). Having completed our study of the network layer, our journey now takes us one step further down the protocol stack, namely, to the link layer. Like the network layer, the link layer is part of each and every network-connected device. But we will see in the next chapter that the link layer has the much more localized task of moving packets between nodes on the same link or LAN. Although this task may appear on the surface to be rather simple compared with that of the network layer's tasks, we will see that the link layer involves a number of important and fascinating issues that can keep us busy for a long time. Homework Problems and Questions Chapter 5 Review Questions SECTION 5.1 SECTION 5.2 SECTIONS 5.3–5.4 R1. What is meant by a control plane that is based on per-router control? In such cases, when we say the network control and data planes are implemented "monolithically," what do we mean? R2. What is meant by a control plane that is based on logically centralized control? In such cases, are the data plane and the