CHAPTER

INTRODUCTION

1

It is not often that an author of a technology text gets to read about his subject matter in a major story in a current issue of a leading news magazine. The tempest surrounding *Software Defined Networking* (SDN) is indeed intense enough to make mainstream news [1]. The modern computer network has evolved into a complex beast that is challenging to manage and which struggles to scale to the requirements of some of today's environments. SDN represents a new approach to computer networking that attempts to address these weaknesses of the current paradigm. SDN is a fundamentally novel way to program the switches utilized in modern data networks. SDN's move to a highly scalable and centralized network control architecture is better suited to the extremely large networks prevalent in today's mega-scale data centers. Rather than trying to crowbar application-specific forwarding into legacy architectures ill-suited to the task, SDN is designed from the outset to perform fine-grained traffic forwarding decisions. Interest in SDN goes far beyond the research and engineering communities intrigued by this new Internet switching technology. Another of the early drivers of SDN was to facilitate experimentation with and the development of novel protocols and applications designed to address problems related to an Internet bogged down by the weight of three decades of incremental technological fixes—a problem sometimes called *Internet Ossification*. If SDN's technological promise is realized, this will represent nothing short of a tectonic shift in the networking industry, as long-term industry incumbents may be unseated and costs to consumers may plummet. Along with this anticipation, though, surely comes a degree of overhype, and it is important that we understand not only the potentials of this new networking model, but also its limitations. In this work we will endeavor to provide a technical explanation of how SDN works, an overview of those networking applications for which it is well-suited and those for which it is not, a tutorial on building custom applications on top of this technology, and a discussion of the many ramifications it has on the networking business itself.

This introductory chapter provides background on the fundamental concepts underlying current state-of-the-art Internet switches, where data plane, control plane and management plane will be defined and discussed. These concepts are key to understanding how SDN implements these core functions in a substantially different manner than the traditional switch architecture. We will also present how forwarding decisions are made in current implementations and the limited flexibility this offers network administrators to tune the network to varying conditions. At a high level, we provide examples of how more flexible forwarding decisions could greatly enhance the business versatility of existing switches. We illustrate how breaking the control plane out of the switch itself into a separate, open-platform controller can provide this greater flexibility. We conclude by drawing parallels between how the Linux operating system has enjoyed rapid growth by leveraging the open source development community and how the same efficiencies can be applied to the control plane on Internet switches.

Software Defined Networks. http://dx.doi.org/10.1016/B978-0-12-804555-8.00001-6

We next look at some basic packet switching terminology that will be used throughout the text, and following that we provide a brief history of the field of packet switching and its evolution.

1.1 BASIC PACKET SWITCHING TERMINOLOGY

This section defines much of the basic packet switching terminology used throughout the book. Our convention is to italicize a new term on its first use. For more specialized concepts that are not defined in this section, they will be defined on their first use. Many packet switching terms and phrases have several and varied meanings to different groups. Throughout the book we try to use the most-accepted definition for terms and phrases. Acronyms are also defined and emphasized on their first use, and the appendix on acronyms provides an alphabetized list of all of the acronyms used in this work. An advanced reader may decide to skip over this section. Others may want to skim this material, and later look back to refer to specific concepts.

This terminology is an important frame of reference as we explain how SDN differs from traditional packet switching. To some degree, though, SDN does away with some of these historic concepts or changes their meaning in a fundamental way. Throughout this book, we encourage the reader to look back at these definitions and consider when the term's meaning is unchanged in SDN, when SDN requires a nuanced definition, and when a discussion of SDN requires entirely new vocabulary.

A *Wide Area Network* (WAN) is a network that covers a broad geographical area, usually larger than a single metropolitan area.

A *Local Area Network* (LAN) is a network that covers a limited geographical area, usually not more than a few thousand square meters in area.

A *Metropolitan Area Network* (MAN) is a network that fills the gap between LANs and WANs. This term came into use because LANs and WANs were originally distinguished not only by their geographical areas of coverage, but also by the transmission technologies and speeds that they used. With the advent of technologies resembling LANs in terms of speed and access control, but with the capability of serving a large portion of a city, the term MAN came into use to distinguish these networks as a new entity distinct from large LANs and small WANs.

A *Wireless Local Area Network* (*WLAN*) is a LAN in which the transmission medium is air. The typical maximum distance between any two devices in a wireless network is on the order of 50 m. While it is possible to use transmission media other than air for wireless communication, we will not consider these in our use of this term in this work.

The *Physical Layer* is the lowest layer of the seven layer *Open Systems Interconnection* (OSI) model of computer networking [2]. It consists of the basic hardware transmission technology to move bits of data on a network.

The *Data Link Layer* is the second lowest layer of the OSI model. This is the layer that provides the capability to transfer data from one device to another on a single network segment. For clarity, here we equate a LAN network segment with a collision domain. A strict definition of LAN network segment is an electrical or optical connection between network devices. For our definition of data link layer we will consider multiple segments linked by repeaters as a single LAN segment. Examples of network segments are a single LAN, such as an Ethernet, or a point-to-point communications link between adjacent nodes in a WAN. The link layer includes: (1) mechanisms to detect sequencing errors or bit-errors that may occur during transmission, (2) some mechanism of flow control between the sender and receiver across that network segment, and (3) a multiplexing ability that allows multiple network protocols to use the same communications medium. These three functions are considered to be part of the *logical link control* (LLC) component of the data link layer. The remaining functions of the data link layer are part of the *Media Access Control* (MAC) component, described separately below.

The *MAC* layer is the part of the data link layer that controls when a shared medium may be accessed and provides addressing in the case that multiple receivers will receive the data yet only one should process it. For our purposes in this book, we will not distinguish between data link layer and MAC layer.

The *Network Layer* provides the functions and processes that allow data to be transmitted from sender to receiver across multiple intermediate networks. To transit each intermediate network involves the data link layer processes described above. The network layer is responsible for stitching together those discrete processes such that the data correctly makes its way from the sender to the intended receiver.

Layer one is the same as the physical layer defined above.

Layer two is the same as the data link layer defined above. We will also use the term *L2* synonymously with layer two.

Layer three is the same as the network layer defined above. *L3* will be used interchangeably with layer three in this work.

A *port* is a connection to a single communications medium, including the set of data link layer and physical layer mechanisms necessary to correctly transmit and receive data over that link. This link may be of any feasible media type. We will use the term *interface* interchangeably with port throughout this text. Since this book will also deal with virtual switches, the definition of port will be extended to include virtual interfaces, which are the endpoints of tunnels.

A *frame* is the unit of data transferred over a layer two network.

A *packet* is the unit of data transferred over a layer three network. Sometimes this term is used more generally to refer to the units of data transferred over either a layer two network (frames) as well, without distinguishing between layers two and three. When the distinction is important, a packet is always the payload of a frame.

A *MAC address* is a unique value that globally identifies a piece of networking equipment. While these addresses are globally unique, they serve as layer two addresses, identifying a device on a layer two network topology.

An *IP Address* is a nominally unique value assigned to each host in a computer network that uses the Internet Protocol for layer three addressing.

An *IPv4 Address* is an IP address that is a 32-bit integer value conforming to the rules of Internet Protocol Version 4. This 32-bit integer is frequently represented in *dotted* notation, with each of the 4 bytes comprising the address represented by a decimal number from 0 to 255, separated by periods (e.g., 192.168.1.2).

An *IPv6 Address* is an IP address that is a 128-bit integer conforming to the rules of Internet Protocol Version 6, introducing a much larger address space than IPv4.

A *switch* is a device that receives information on one of its ports and transmits that information out one or more of its other ports, directing this information to a specified destination.

A *circuit switch* is a switch where contextual information specifying where to forward the data belonging to a circuit (i.e., connection) is maintained in the switch for a prescribed duration, which may span lapses of time when no data belonging to that connection is being processed. This context is established either by configuration or by some *call set-up* or *connection set-up* procedure specific to the type of circuit switch.

A *packet switch* is a switch where the data comprising the communication between two or more entities is treated as individual packets that each make their way independently through the network toward the destination. Packet switches may be of the connection-oriented or connectionless type.

The *connection-oriented* model is when data transits a network where there is some context information residing in each intermediate switch that allows the switch to forward the data toward its destination. The circuit switch described above is a good example of the connection-oriented paradigm.

The *connectionless* model is when data transits a network and there is sufficient data in each packet such that each intermediate switch can forward the data toward its destination without any a priori context having been established about that data.

A *router* is a packet switch used to separate subnets. A subnet is a network consisting of a set of hosts that share the same network prefix. A network prefix consists of the most significant bits of the IP address. The prefix may be of varying lengths. Usually all of the hosts on a subnet reside on the same LAN. The term router is now often used interchangeably with *layer three switch*. A home wireless access point typically combines the functionality of WiFi, layer two switch, and router into a single box.

To *flood* a packet is to transmit it on all ports of a switch except for the port on which it was received.

To *broadcast* a packet is the same as flooding it.

Line rate refers to the bandwidth of the communications medium connected to a port on a switch. On modern switches this bandwidth is normally measured in *megabits per second* (Mbps) or *gigabits per second* (Gbps). When we say that a switch handles packets at line rate, this means it is capable of handling a continuous stream of packets arriving on that port at that bandwidth.

WiFi is common name for wireless communications systems that are based on the IEEE 802.11 standard.

1.2 HISTORICAL BACKGROUND

The major communications networks around the world in the first half of the 20th century were the telephone networks. These networks were universally circuit switched networks. Communication between endpoints involved the establishment of a communications path for that dialogue and the tearing down of that path at the dialogue's conclusion. The path on which the conversation traveled was static during the call. This type of communications is also referred to as connection-oriented. In addition to being based on circuit switching, the world's telephone networks were quite centralized, with large volumes of end-users connected to large switching centers. Paul Baran, a Polish immigrant who became a researcher working at Rand Corporation in the United States in the 1960s, argued that in the event of enemy attack networks like the telephone network were very easy to disrupt [3,4]. The networks had poor survivability characteristics in that the loss of a single switching center could remove phone capability from a large swath of the country. Mr. Baran's proposed solution was to transmit the voice signals of the phone conversations in packets of data that could travel autonomously through the network, finding their own way toward their destination. This concept included the notion that if part of the path being used for a given conversation was destroyed by enemy attack, the communication would *survive* by automatically rerouting over alternative paths to the same destination. He demonstrated that the national voice communications system could still function even if fifty per cent of the forwarding switches were destroyed, greatly reducing the vulnerability characteristics of the centralized, circuit switching architecture prevalent at the time.

When Mr. Baran did his work at Rand, he never could have envisioned the dominance his idea would ultimately achieve in the area of data networking. While certainly not the universally accepted networking paradigm at the time, the history that followed is now part of Internet folklore. Mr. Baran's ideas became embodied in the *Department of Defense's* (DOD's) experimental ARPANET network that began to operate in 1969. The ARPANET connected academic research institutions, military departments and defense contractors. This decentralized, connectionless network grew over the years until bursting upon the commercial landscape around 1990 in the form of the Internet known and loved around the world today.

For decades after the emergence of the ARPANET, networking professionals waged battles over the advantages of connection-based vs. connectionless architectures and centralized vs. distributed architectures. The tides in this struggle seemed to turn one way, only to reverse some years later. The explosive growth of the Internet in the 1990s seemed to provide a conclusion to these arguments, at least as far as computer networking was concerned. The Internet was in ascendance, and was unequivocally a distributed, connectionless architecture. Older connection-oriented protocols like X.25 [5] seemed destined to become a thing of the past. Any overly centralized design was considered too vulnerable, whether to malicious attacks or to simple acts of nature. Even the new kid on the block, *Asynchronous Transfer Mode* (ATM) [5] hyped in the mid-1990s to be capable of handling line rates greater than the Internet could ever handle, would eventually be largely displaced by the ever-flexible Internet that somehow managed to handle line rates in the tens of gigabits per second range, once thought only to be possible using cell-switching technology like ATM.

DISCUSSION QUESTION:

Describe why Paul Baran's notion of a connectionless network had potential appeal to the United States Department of Defense.

1.3 THE MODERN DATA CENTER

The Internet came to dominate computer networking to a degree rarely seen in the history of technology, and the Internet was at its core connectionless and distributed. During this period of ascendance, the *World Wide Web* (WWW), an offspring of the Internet, spawned by Sir Tim Berners-Lee of the United Kingdom, gave rise to ever-growing data centers, hosting ever more complex and more heavily subscribed web services. These data centers served as environmentally protected warehouses housing large numbers of computers that served as compute and storage servers. The warehouses themselves were protected against environmental entropy as much as possible by being situated in disaster-unlikely geographies, with redundant power systems, and ultimately with duplicate capacity at disparate geographical locations.

Because of the large numbers of these servers, they were physically arranged into highly organized rows of racks of servers. Over time, the demand for efficiency drove the migration of individual server computers into server *blades* in densely packed racks. Racks of compute-servers were hierarchically

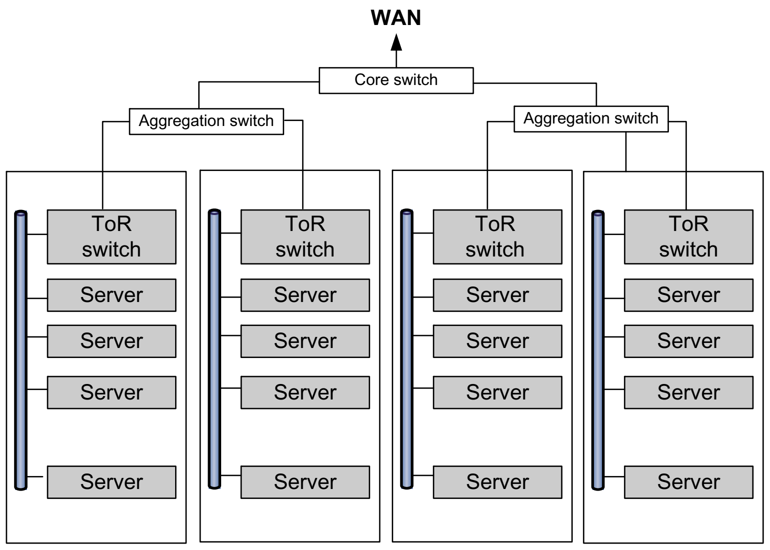

FIG. 1.1

Typical data center network topology.

organized such that *Top-of-Rack* (ToR) switches provided the networking within the rack and the inter-rack interface capability. We depict the rigid hierarchy typical in today's data center in Fig. 1.1.

During this evolution, the sheer numbers of computers in the data center grew rapidly. Data centers are being built now that will accommodate over 120,000 physical servers [6]. State-of-the-art physical servers can conceivably host twenty *virtual machines* (VMs) per physical server. This means that the internal network in such a data center would interconnect 2,400,000 hosts! This growing number of host computers within a data center all communicate with each other via a set of communications protocols and networking devices that were optimized to work over a large, disparate geographical area with unreliable communications links. Obviously, tens of thousands of computers sitting side by side in an environmental cocoon with highly reliable communications links is a different nature of network altogether. It gradually emerged that the goals of survivability in the face of lost communications links or a bombed-out network control center were not highly relevant in this emerging reality of the data center. Indeed, an enormous amount of complexity invented to survive precisely those situations was making the operation of the ever-larger and more complex data center untenable.

Beyond the obvious difference in the stability of the network topology, the sheer scale of these mega-data centers in terms of nodes and links creates a network management challenge different than those encountered previously. Network management systems designed for carrier public networks or large corporate intranets simply cannot scale to these numbers. A new network management paradigm was needed.

Furthermore, while these data centers exist in order to support interaction with the turbulent world outside of their walls, studies [7,8] indicate that the majority of the traffic in current data centers is *East-West* traffic. East-West traffic is composed of packets sent by one host in a data center to another host in that same data center. Analogously, *North-South* traffic is traffic entering (leaving) the data center from (to) the outside world. For example, a user's web browser's query about the weather might be processed by a web server (North-South) which retrieves data from a storage server in the same data center (East-West) before responding to the user (North-South). The protocols designed to achieve robustness in the geographically dispersed wide-area Internet today require that routers spend more than thirty percent of their CPU cycles [6] rediscovering and recalculating routes for a network topology in the data center that is highly static and only changed under strict centralized control. This increasing preponderance of East-West traffic does not benefit from the overhead and complexities that have evolved in traditional network switches to provide just the de-centralized survivability that Mr. Baran so wisely envisioned for the WANs of the past.

The mega-data centers discussed in this section differ from prior networks in a number of ways: stability of topology, traffic patterns, and sheer scale. In addition, since the services provided by these data centers require frequent reconfiguration, these networks demand a level of agility not required in the past. Traditional networking methods are simply insufficiently dynamic to scale to the levels being required. SDN is a technology designed explicitly to work well with this new breed of network, and represents a fundamental transformation from traditional Internet switching. In order to understand how SDN does differ, in Sections 1.4 and 1.5 we review how legacy Internet switches work to establish a baseline for comparison with SDN.

Before continuing, we should emphasize that while the modern data center is the premier driver behind the current SDN fervor, by no means is SDN only applicable to the data center. When we review SDN applications in Chapter 12 we will see that SDN technology can bring important innovations in domains such as mobility that have little to do with the data center.

DISCUSSION QUESTION:

Provide three examples of east-west traffic and three examples of north-south traffic in a modern data center.

1.4 TRADITIONAL SWITCH ARCHITECTURE

We will now look at what the traditional Internet switch looks like from an architectural perspective, and how and why it has evolved to be the complex beast it is today. The various switching functions are traditionally segregated into three separate categories. Since each category may be capable of *horizontal* communication with peer elements in adjacent entities in a topology, and also capable of *vertical* communication with the other categories, it has become common to represent each of these categories as a layer or *plane*. Peer communications occur in the same plane, and cross-category messaging occurs in the third dimension, between planes.

1.4.1 DATA, CONTROL, AND MANAGEMENT PLANES

We show the control, management, and data planes of the traditional switch in Fig. 1.2. The vast majority of packets handled by the switch are only touched by the *data plane*. The data plane consists of the various ports that are used for the reception and transmission of packets and a forwarding table with

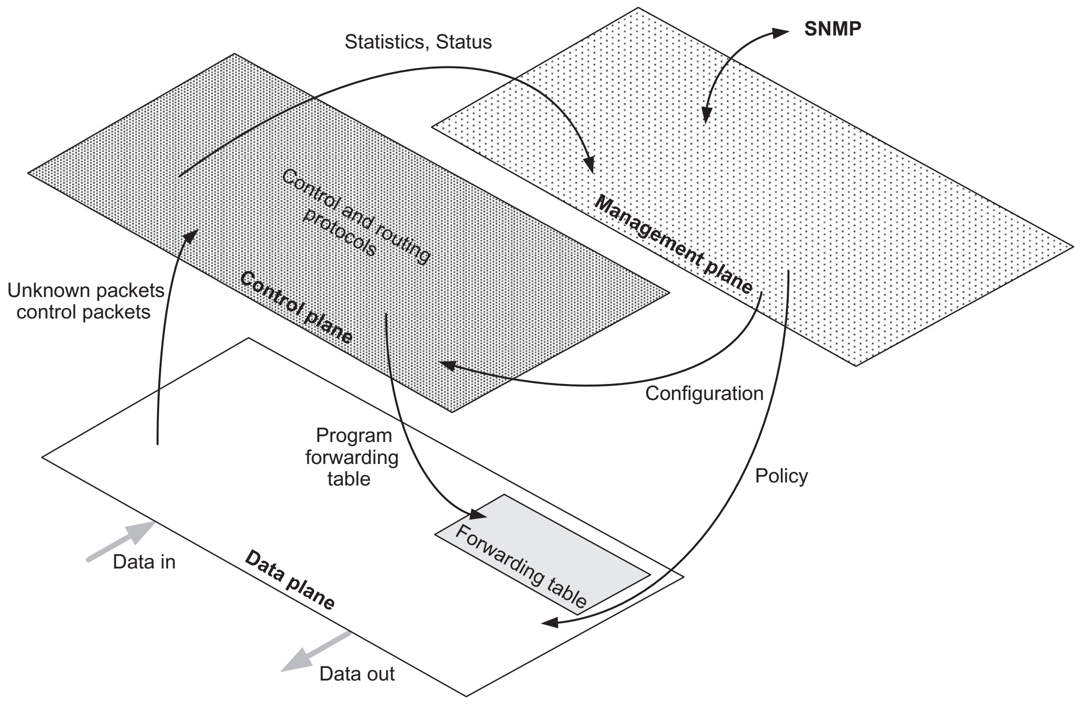

FIG. 1.2

Roles of the control, management, and data planes.

its associated logic. The data plane assumes responsibility for packet buffering, packet scheduling, header modification, and forwarding. If an arriving data packet's header information is found in the forwarding table it may be subject to some header field modifications and then will be forwarded without any intervention of the other two planes. Not all packets can be handled in that way, sometimes simply because their information is not yet entered into the table, or because they belong to a control protocol that must be processed by the *control plane*. The control plane, as shown in Fig. 1.2, is involved in many activities. Its principal role is to keep current the information in the forwarding table so that the data plane can independently handle as high a percentage of the traffic as possible. The control plane is responsible for processing a number of different control protocols that may affect the forwarding table, depending on the configuration and type of switch. These control protocols are jointly responsible for managing the active topology of the network. We review some of these control protocols below in Section 1.5. These control protocols are sufficiently complex as to require the use of general purpose microprocessors and accompanying software in the control plane, whereas we will see in Section 1.4.3 that the data plane logic can today be incarnated entirely in silicon.

The third plane depicted in Fig. 1.2 is the management plane. Network administrators configure and monitor the switch through this plane, which in turn extracts information from or modifies data in the control and data planes as appropriate. The network administrators use some form of network management system to communicate with the management plane in a switch.

1.4.2 SOFTWARE-BASED ROUTING AND BRIDGING

In the late 1980s the commercial networking world was undergoing the cathartic process of the emergence of the Internet as a viable commercial network. New start-ups such as Cisco Systems and Wellfleet Communications had large teams of engineers building special-purpose, commercial versions of the *routers* that had previously been relegated to use in research and military networks. While the battle between connection-oriented packet switching, such as X.25 and ATM, and the connectionless Internet architecture would continue to be waged for a number of years, ultimately the connectionless router would dominate the packet switching world.

Before this time, most routers were just general purpose Unix computers running software that inspected a packet that arrived on one interface and looked up the destination IP address in some efficiently searched data type such as a hierarchical tree structure. This data structure was called the *routing table*. Based on the entry found during this search, the packet would be transmitted on the indicated outbound interface. Control packets would be shunted to the appropriate control processes on the Unix system rather than being processed through that routing table.

In addition to layer three routers, many companies marketed layer two *bridges*. Bridges create a bridged LAN which is a topology of interconnected LAN segments. As there were multiple competing layer two technologies prevalent, including Token Ring, *Fiber Distributed Data Interface* (FDDI), and different speeds and physical media versions of the original Ethernet, these bridges also served as a mechanism to interface between disparate layer two technologies. The then-current terminology was to use the terms *bridging* of *frames* for layer two and *routing* of *packets* for layer three. We will see in this book that as the technologies have evolved, these terms have become blended such that a modern networking device capable of handling both layer two and three is commonly referred to simply as a *packet switch*.

Demand for increased speed brought about concomitant advances in the performance of these network devices. In particular, rapid evolution occurred both in optical and twisted copper pair physical media. Ten Mbps Ethernet interfaces were commonplace by 1990 and one hundred Mbps fiber and ultimately twisted pair were on the horizon. Such increases in media speed made it necessary to continually increase the speed of the router and bridge platforms. At first, such performance increases were achieved by distributing the processing over ever-more parallel implementations involving multiple *blades*, each running state-of-the-art microprocessors with independent access to a distributed forwarding table. Ultimately, though, the speed of the interfaces reached a point where performing the header inspection and routing table look-up in software simply could not keep up.

1.4.3 HARDWARE LOOK-UP OF FORWARDING TABLES

The first major use of hardware acceleration in packet switching was via the use of *Application-Specific Integrated Circuits* (ASICs) to perform high-speed hashing functions for table look-ups. In the mid-1990s advances in *Content-Addressable Memory* (CAM) technology made it possible to perform very high speed look-up using destination address fields to find the output interface for high-speed packet forwarding. The networking application of this technology made its commercial debut in *Ethernet switches*.

At that time, the term *switch* became used to distinguish a hardware-look-up-capable layer two bridge from the legacy software-look-up devices discussed above in Section 1.4.2. The term router still referred to a layer three device, and initially these were still based upon software-driven address look-up. Routers were still software-based for a number of reasons. First, the packet header modifications required for layer three switching were beyond the capability of the ASICs used at the time. Also, the address look-up in layer two switching was based on the somewhat more straightforward task of looking up a 48-bit *MAC* address. Layer three address look-up is more complicated since the devices look up the *closest* match on a network address, where the match may only be on the most significant bits of a network address. Any of the destination addresses matching that network address will be forwarded out the same output interface to the same *next-hop* address. The capabilities of CAMs steadily improved, however, and within a few years there were layer three routers that were capable of hardware look-up of the destination layer three address. At this point, the distinction between router and switch began to blur, and the terms *layer two switch* and *layer three switch* came into common use. Today, where the same device has the capability to simultaneously act as both a layer two and layer three switch, the use of the simple term switch has become commonplace.

1.4.4 GENERICALLY PROGRAMMABLE FORWARDING RULES

In reality there are many packets that need to be processed by a switch that need more complicated treatment than simply being forwarded on another interface different than the one on which it arrived. For example, the packet may belong to one of the control protocols covered in Sections 1.5.1 and 1.5.2, in which case it needs to be processed by the control plane of the switch. The brute-force means of handling these exceptions by passing all exception cases to the control plane for processing rapidly became unrealistic as forwarding rates increased. The solution was to push greater intelligence down into the forwarding table such that as much of the packet processing as possible can take place at line rates in the hardware *forwarding engine* itself.

Early routers only had to perform limited packet header field modifications. This included decrementing the *Time-To-Live* (TTL) field and swapping the MAC header to the source-destination MAC headers for the next hop. In the case of multicast support, they had to replicate the packet for transmission out multiple output ports. As the switch's features grew to support advanced capabilities like *Virtual Local Area Networks* (VLANs) and *Multiprotocol Label Switching* (MPLS), more packet header fields needed ever more complex manipulation. To this end, the idea of *programmable rules* came into being whereby some of this more complex processing could be encoded in rules and carried out directly in the forwarding engines at line rate. Such rules could be communicated to the switch by network managers using a *Command Line Interface* (CLI) or some other primitive configuration mechanism. While most control protocol packets still needed to be shunted to the control plane processor for handling, this capability allowed the evolving switches to implement ever more complex protocols and features at ever increasing line rates. This very programmability of the hardware was one of the early seeds that gave life to the concept of SDN.

While there are many variations in hardware architecture among modern switches, we attempt to depict an idealized state-of-the-art switch in Fig. 1.3. The figure shows the major functional blocks that a packet will transit as it is being forwarded by the switch. In the figure we see that the packet may transit the *packet receive*, *ingress filter*, *packet translation*, *egress filter*, and *packet transmit* functions or be consumed by the *Switch OS*. We can see that the forwarding database is influenced by the arrival

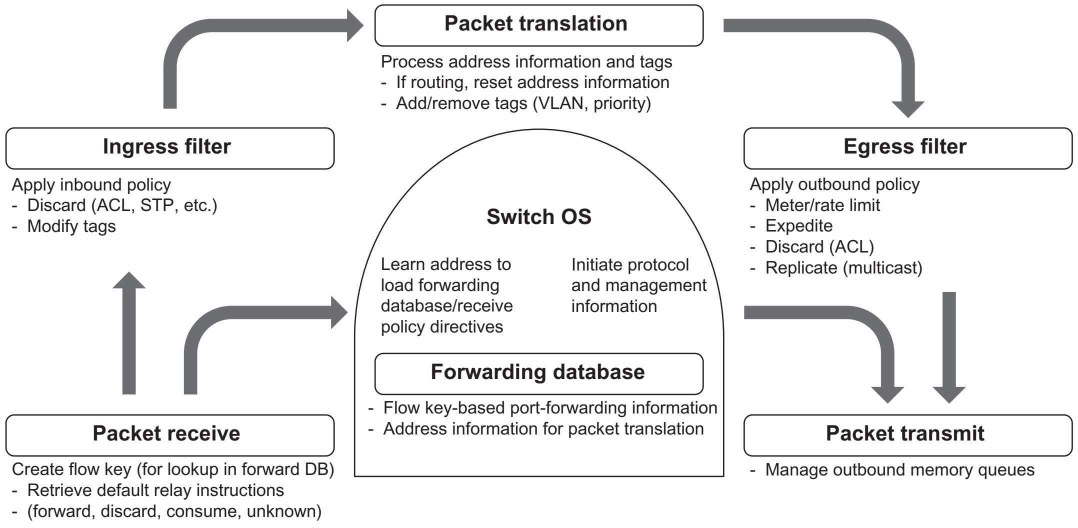
Packet translation
Process address information and tags
- If routing, reset address information
- Add/remove tags (VLAN, priority)
Ingress filter
Apply inbound policy
- Discard (ACL, STP, etc.)
- Modify tags
Switch OS
Learn address to load forwarding database/receive policy directives
Initiate protocol and management information
Egress filter
Apply outbound policy
- Meter/rate limit
- Expedite
- Discard (ACL)
- Replicate (multicast)
Forwarding database
- Flow key-based port-forwarding information
- Address information for packet translation
Packet receive
Create flow key (for lookup in forward DB)
- Retrieve default relay instructions
- (forward, discard, consume, unknown)
Packet transmit
- Manage outbound memory queues

FIG. 1.3

A packet's journey through switching hardware.

of certain packets, such as when a new address is learned. We also see that the hardware generates a flow key to serve as a look-up key into the forwarding database. The forwarding database will provide basic forwarding instructions as well as policy directives, if relevant in this switch. The ingress filter applies policy, perhaps resulting in the packet being dropped. Packet header modifications take place both at the ingress filter as well as in the packet translation function. Outbound policy is implemented at the egress filter, where again the packet may be dropped, if, for example, established *Quality of Service* (QoS) rates are being exceeded. Finally, the packet passes through the packet transmit function where additional priority queue processing occurs prior to its actual transmission on the communications medium.

DISCUSSION QUESTION:

Discuss two major reasons why hardware-based packet forwarding was achievable in layer two bridges before layer three routers.

1.5 AUTONOMOUS AND DYNAMIC FORWARDING TABLES

In order to respond to unstable communications links and the possible disappearance of an entire switching center, protocols were developed so that bridges and routers could dynamically and autonomously build and update their forwarding tables. In this section we provide background on some of the major protocols that were developed to provide this capability.

1.5.1 LAYER TWO CONTROL

A layer two forwarding table is a look-up table indexed by destination MAC address, where the table entry indicates which port on the switch should be used to forward the packet. In the case of early, stand-alone bridges, when a frame was first received from a device on a given interface, the switch could *learn* the sender's MAC address and location and populate its forwarding table with this new information. When a frame arrived destined for that address, it could then use the forwarding table to send that frame out the correct port. In the case that an *unknown* destination address was received, the bridge could flood this out all interfaces, knowing that it would be dropped on all attached networks except the one where the destination actually resided. This approach no longer worked once bridges were interconnected into networks of bridges as there were multiple paths to reach that destination, and unless a single one was chosen predictably, an infinite loop could be created. While a static mapping of MAC addresses to ports was an alternative, protocols were developed that allowed the bridge to learn MAC addresses dynamically, and to learn how to assign these to ports in such a way as to automatically avoid creating switching loops yet predictably be able to reach the destination MAC address, even when that required transiting multiple intermediate bridges. We discuss some of these MAC-learning protocols below to illustrate ways that the legacy layer two switch's forwarding table can be built up automatically.

As we explained earlier, the Ethernet switch's role is to bridge multiple layer two networks. The most simplistic concept of such a device includes only the ability to maintain a forwarding table of MAC addresses such that when a frame reaches the switch it understands on which attached layer two network that destination MAC address exists, and to forward it out on that layer two network. Clearly, if the frame arrived on a given port, and if the destination is reachable via that same port, the Ethernet switch need not be involved in forwarding that frame to its destination. In this case, the frame was dropped. The task of the Ethernet switch is considerably more complex when multiple Ethernet switches are connected into complicated topologies containing loops. Without taking care to prevent such situations, it was possible to accidentally construct forwarding loops where a frame is sent back to a switch that had forwarded it earlier ad infinitum. The performance impact of such switching loops is more exaggerated in layer two switching than layer three, since the MAC header has no analog to the layer three *TTL* field, which limits the number of times the packet can transit routers before it is dropped. Thus, an unintentional loop in a layer two network can literally bring the network to a halt, where the switch appears frozen, but is actually one hundred percent occupied forwarding the same packet over and over through the loop.

The prevalence of broadcast frames in the layer two network exacerbates this situation. Since many layer two control frames must be broadcast to all MAC addresses on that layer two topology, it was very easy for a broadcast frame to return to a switch that had already broadcast that frame. This latter problem is known as *broadcast radiation*. The *Spanning Tree Protocol* (STP), specified in IEEE 802.1D [9], was designed specifically to prevent loops and thus addresses the problem of broadcast radiation. The protocol achieves this by having all switches in the topology collectively compute a *spanning tree*, a computer science concept whereby from a single root (source MAC address) there is exactly one path to every leaf (destination MAC address) on the tree. STP has been superceded, from a standards perspective at least, by protocols offering improvements on the original protocol's functionality, most recently the *Shortest Path Bridging* (SPB) [10] and the *Transparent Interconnection of Lots of Links* (TRILL) protocols [11]. In practice, though, STP is still prevalent in networks today.

1.5.2 LAYER THREE CONTROL

The router's most fundamental task when it receives a packet is to determine if the packet should be forwarded out one of its interfaces or if the packet needs some exception processing. In the former, normal case, the layer three destination address in the packet header is used to determine over which output port the packet should be forwarded. Unless the destination is directly attached to this router, the router does not need to know exactly where in the network the host is located, it only needs to know the next-hop router to which it should forward the packet. There may be large numbers of hosts that reside on the same destination network, and all of these hosts will share the same entry in the router's *routing table*. This is a table of layer three *networks* and sometimes layer three host addresses, not a table of just layer two destination host addresses as is the case with the layer two forwarding table. Thus, the look-up that occurs in a router is to match the destination address in the packet with one of the networks in the forwarding table. Finding a match provides the next-hop router, which maps to a specific forwarding port, and the forwarding process can continue. In a traditional router, this routing table is built primarily through the use of *routing protocols*. While there is a wide variety of these protocols and they are specialized for different situations, they all serve the common purpose of allowing the router to autonomously construct a layer three forwarding table that can automatically and dynamically change in the face of changes elsewhere in the network. Since these protocols are the basic mechanism by which the traditional layer three switch builds its forwarding table, we provide a brief overview of them here.

The *Routing Information Protocol* (RIP) [12] is a comparatively simple routing protocol that was widely deployed in small networks in the 1990s. Each router in the RIP-controlled routing domain periodically broadcasts its entire routing table on all of its interfaces. These broadcasts include the hop count from the broadcasting router to each reachable network. The weight of each hop can be tailored to accommodate relevant differences between the hops, such as link speed. Neighboring routers can integrate this reachability and hop count information into their own routing tables, which will in turn be propagated to their neighbors. This propagation pervades the entire routing domain or *autonomous system* (AS). The routers in an AS share detailed routing information allowing each member router in the AS to compute the shortest path to any destination within that AS. In a stable network, eventually all of the routing tables in that routing domain will converge, and the hop count information can be used to determine a least-cost route to reach any network in that domain. RIP is a distance-vector protocol, where each router uses the weighted hop count over one interface as its distance to that destination network, and is able to select the best next-hop router as the one presenting the least total distance to the destination. RIP does not need to have a complete picture of the network topology in order to make this distance-vector routing computation. This differs from the next two protocols, *Open Shortest Path First* (OSPF) [13] and *Intermediate System to Intermediate System*, (IS-IS) [14] both of which maintain a complete view of the network topology in order to compute best routing paths. OSPF is an Internet Engineering Task Force project, whereas IS-IS has its origins in OSI.

Both OSPF and IS-IS are link-state dynamic routing protocols. This name derives from the fact that they both maintain a complete and current view of the state of each link in the AS and can thus determine its topology. The fact that each router knows the topology permits them to compute the best routing paths. Since both protocols know the topology, they can represent all reachable networks as leaf nodes on a graph. Internal nodes represent the routers comprising that autonomous system. The edges on the graph represent the communication links joining the nodes. The *cost* of edges may be assigned by any metric, but an obvious one is bandwidth. The standard shortest-path algorithm, known as *Dijkstra's algorithm* is used to compute the shortest path to each reachable network from each router's perspective. For this class of algorithm, the shortest path is defined as the one traversed with the smallest total cost. If a link goes down, this information is propagated via the respective protocol, and each router in the AS will shortly converge to have identical pictures of the network topology. OSPF is currently widely used in large enterprise networks. Large service provider networks tend more to use IS-IS for this purpose. This is primarily due to IS-IS having been a more mature and thus more viable alternative than OSPF in the early service provider implementations. Although OSPF and IS-IS are essentially equally capable at this point in time, there is little incentive for the service providers to change.

While the three routing protocols discussed above are *Interior Gateway Protocols* (IGPs) and therefore concerned with optimizing routing within an AS, the Internet is comprised of a large set of interconnected autonomous systems, and it is obviously important to be able to determine the best path transiting these ASs to reach the destination address. This role is filled by an *Exterior Gateway Protocol* (EGP). The only EGP used in the modern Internet is the *Border Gateway Protocol* (BGP) [15]. BGP routers primarily reside at the periphery of an AS and learn about networks that are reachable via peer edge routers in adjacent ASs. This network routing information can be shared with other routers in the same AS either by using the BGP protocol between routers in the same AS, which is referred to as Internal BGP, or by redistributing the external network routing information into the IGP routing protocol. The most common application of BGP is to provide the routing *glue* between ASs run by different service providers, allowing communication from a host in one to transit through other providers' networks to reach the ultimate destination. Certain private networks may also require the use of BGP to interconnect multiple internal ASs that cannot grow larger due to reaching the maximum scale of the IGP they are using. BGP is called a *path vector protocol*, which is similar to a distance vector protocol like RIP discussed above, in that it does not maintain a complete picture of the topology of the network for which it provides routing information. Unlike RIP, the metric is not a simple distance (i.e., hop count), but involves parameters such as network policies and rules as well as the distance to the destination network. The policies are particularly important when used by service providers as there may be business agreements in place that need to dictate routing choices that may trump simple routing cost computations.

1.5.3 PROTOCOL SOUP OR (S)WITCH'S BREW?

The control protocols listed by name above are only a small part of a long list of protocols that must be implemented in a modern layer two or layer three switch in order to make it a fully functional participant in a state-of-the-art Internet. In addition to those we have mentioned thus far, a modern router will likely also support LDP [16] for exchanging information about MPLS labels, as well as IGMP [17], MSDP [18], and PIM [19] for multicast routing. We see in Fig. 1.4 how the control plane of the switch is bombarded with a constant barrage of control protocol traffic. Indeed, Ref. [6] indicates that in large scale data center networks *thirty percent* of the router's control plane capacity today is spent tracking network topology. We have explained above that the Internet was conceived as and continues to evolve as an organism that is extremely robust in the face of flapping communications links and switches that may experience disruption in their operation. The Internet-family protocols were also designed to gracefully handle routers joining and leaving the topology whether intentionally or otherwise.

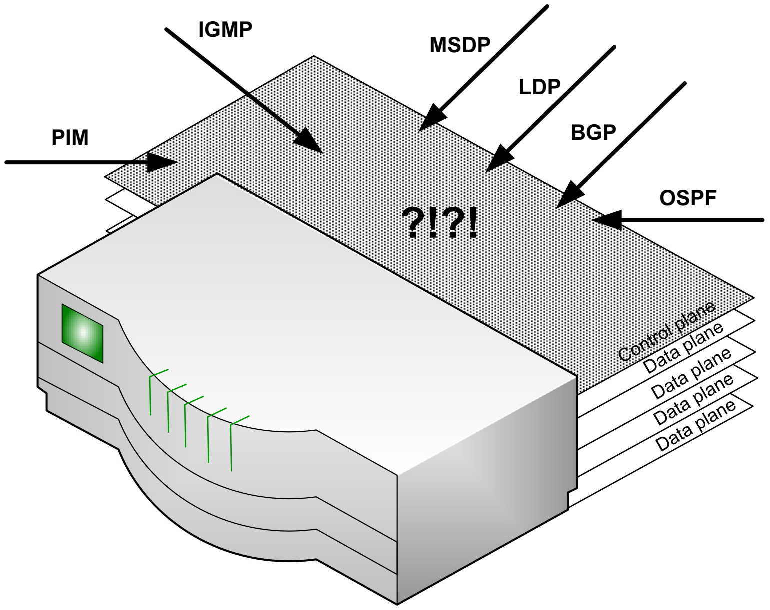

FIG. 1.4

Control plane consternation in the switch.

Such chaotic conditions still exist to varying degrees in the geographically diverse Internet around the globe, though switches and links are generally much more predictable than at the Internet's inception. In particular, the data center is distinct in that it has virtually no unscheduled downtime. While there are links and nodes in the data center that fail occasionally, the base topology is centrally provisioned. Physical servers and VMs do not magically appear or move around in the data center; rather, they only change when the central orchestration software dictates so. Despite the fact that the data center's topology is static and the links are stable, the individual router's travails shown in Fig. 1.4 multiply at data center scale, as depicted in Fig. 1.5. Indeed, since these control protocols are actually distributed protocols, when something is intentionally changed, such as a server being added or a communications link taken out of service, these distributed protocols take time to converge to a set of consistent forwarding tables, wreaking temporary havoc inside the data center. Thus, the overhead experienced by switches shown in Fig. 1.5 is not merely that of four independent units experiencing the problems of the switch shown in Fig. 1.4. Rather, the overhead is exacerbated by the added processing of this convergence. Thus, we reach a situation where most topology change is done programmatically and intentionally, reducing the benefit of the autonomous and distributed protocols, and the scale of the networks is larger than these protocols were designed for, making convergence times unacceptably long.

So, while this array of control protocols was necessary to develop autonomous switches that can dynamically and autonomously respond to rapidly changing network conditions, these conditions do not exist in the modern data center and, by using them in this overkill fashion, we have created a network management nightmare that is entirely avoidable. Instead of drinking this *(s)witch's brew of*

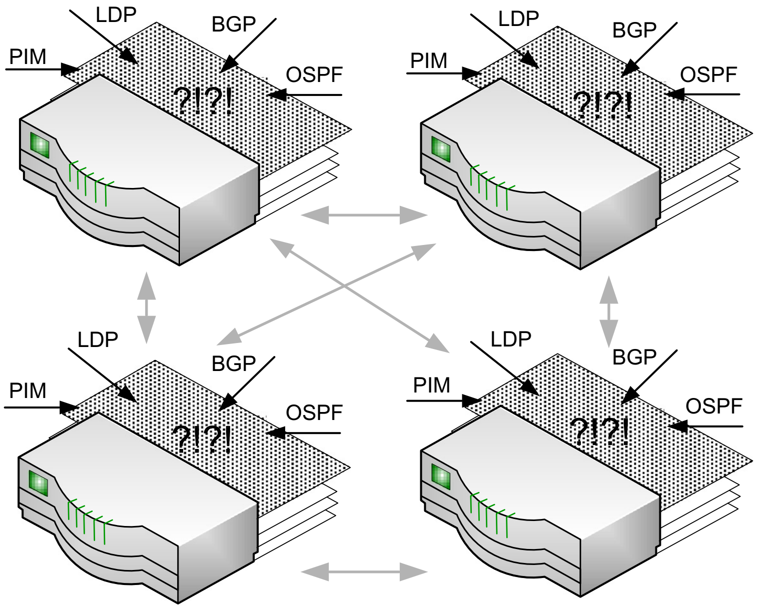

FIG. 1.5

Overhead of dynamic distributed route computation.

protocols, is there a more simple approach? Is there hope for network sobriety by conceiving of a less distributed, less autonomous, but simpler approach of removing the control plane from the switches to a centralized control plane, serving the entire network? This centralized control plane would program the forwarding tables in all the switches in the data center. The simplicity of this concept arises from the facts that (1) the topology inside the data center is quite stable and under strict local administrative control, (2) knowledge of the topology is already centralized and controlled by the same administrators, and (3) when there are node or link failures, this global topology knowledge can be used by the centralized control to quickly reprovision a consistent set of forwarding tables. Thus, it should follow that a framework for programming the switches' forwarding tables like that depicted in Fig. 1.6 might be feasible. This more simple, centralized approach is one of the primary drivers behind SDN. This approach is expected to help address the issue of instability in the control plane which results in long convergence times when changes occur in the network.

There are other important justifications for the migration to SDN. For example, there is a strong desire to drive down the cost of networking equipment. SDN also opens up the possibility of experimentation and innovation by allowing researchers and engineers to build custom protocols in software, which can be easily deployed in the network. We will review these and other arguments underpinning SDN in the next chapter. First, though, we will consider a more fine-grained sort of packet switching that has been pursued for a number of years, and for which SDN has been specifically tailored.

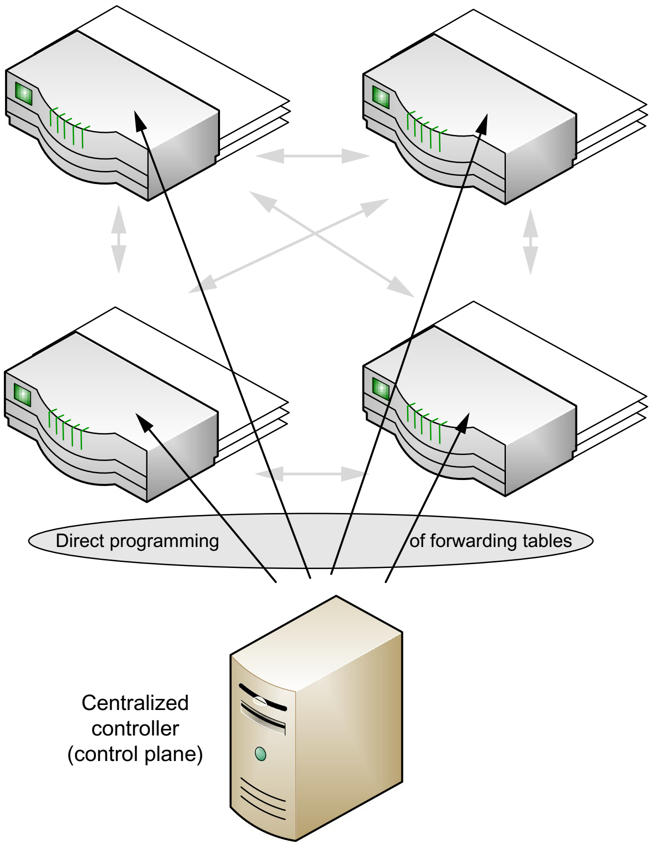

FIG. 1.6

Centralized programming of forwarding tables.

DISCUSSION QUESTION:

A premise of SDN is that the plethora of control plane protocols depicted in Fig. 1.4 creates complexity that is not necessary in all networks. Give an example of a network where the complexity is unnecessary and explain why this is so.

1.6 CAN WE INCREASE THE PACKET FORWARDING IQ?

Whether centrally programmed or driven by distributed protocols, the forwarding tables we have discussed thus far have contained either layer two or layer three addresses. For many years, now, network designers have implemented special processes and equipment that permit more fine-grained forwarding decisions. For example, it may be desirable to segregate traffic destined to different TCP ports on the same IP address such that they travel over separate paths. We may wish to route different QoS classes, such as data, voice, and video, over different paths even if they are directed to the same destination host. Once we leave the realm of routing on just the destination address, we enter a world where many different fields in the packet may be used for forwarding decisions. In general, this kind of routing is called *policy-based routing* (PBR). At this point, we will introduce a new term, *flow*, to describe a particular set of application traffic between two endpoints that receives the same forwarding treatment. On legacy commercial routers, PBR is normally implemented by using *Access Control Lists* (ACLs) to define a set of criteria that determines if an incoming packet corresponds to a particular flow. Thus, if, for example, we decide to forward all video traffic to a given IP address via one port and all email traffic via another port, the forwarding table will treat the traffic as two different flows with distinct forwarding rules for each. While PBR has been in use for a number of years, this has only been possible by configuring these special forwarding rules through the management plane. Such more sophisticated routing has not been dynamically updated via the control plane in the way that simple layer two and three destinations have been. The fact that PBR has long been implemented via central configuration provides further underpinning for SDN's centralized approach of programming forwarding tables.

At the end of Section 1.5.3 we suggested that the SDN model of breaking the control plane out of the switch itself into a separate, centralized controller can provide greater simplicity and flexibility in programming the forwarding tables of the switches in a large and tightly controlled network, such as the modern data center. While Fig. 1.6 hints at an architecture that might allow us such programming capability, the reality is that it becomes possible to do more than just replace the distributed routing algorithms with a cleaner alternative. In particular, we can identify particular user application traffic flows and give them different treatment, routing their traffic along different paths in some cases, even if they are destined for the same network endpoint. In this manner, we can incorporate PBR at the ground level of SDN. Many of the specialized network appliances, such as firewalls, load balancers, and *Intrusion Detection Systems* (IDS), have long relied on performing deeper packet inspection than just the destination address. By looking at source and destination address, and source and destination TCP port number, these devices have been able to segregate and quarantine traffic in a highly useful manner. If we consider that the network's forwarding tables are to be programmed from a centralized controller with global network knowledge, it follows naturally that the kind of information utilized by these special appliances could be programmed directly into the switches themselves, possibly obviating the need for the separate network appliances and simplifying the network topology. If the switches' forwarding tables have the capability to filter, segregate, and quarantine flows, the power of the network itself could grow exponentially. From its birth, SDN has been based on the notion of constructing forwarding tables defining actions to take on flows, rather than having forwarding tables merely map destination address to output port.

SDN purports to incorporate a richer set of information in its processing of individual flows, including fields like the TCP port number which provide clues as to which application that flow corresponds. Nonetheless, we emphasize that the SDN intelligence added at the switching layers will not require any modifications to user applications. SDN is intended to affect how layer two and layer three forwarding decisions are made. As currently defined, SDN can inspect and potentially modify layer two, three, and four protocol header information, but is not intended to make decisions on, nor modify, higher layer protocol fields.

DISCUSSION QUESTION:

In what way is policy-based routing a precursor of SDN?

1.7 OPEN SOURCE AND TECHNOLOGICAL SHIFTS

The SDN paradigm that this book examines represents a major technological deviation from the legacy network protocols and architectures that we have reviewed in this chapter. Those legacy technologies required countless man-years of engineering effort to develop and refine. A paradigm-shifting technology such as SDN will also require a huge engineering effort. While some of this effort will come from commercial enterprises that expect to follow the SDN star to financial success, it is widely expected that SDN will be able to rapidly provide matching functionality to legacy systems by counting on the open source community to develop much of the SDN control plane software.

The open source model has revolutionized the way that software is developed and delivered. Functionality that used to be reinvented in every organization is now often readily available and being incorporated into enterprise-class applications. Often software development organizations no longer have to devote time and energy to implement features that have already been developed.

Probably the most famous and influential body of open source code today is the Linux operating system. This complex and high-quality body of code is still developed by an amorphous and changing team of volunteer software developers around the world. It has changed the markets of commercial operating system vendors and improved quality of both open source and competitive retail products. Numerous businesses have been spawned by innovating on top of this body of work. Small technology start-ups with minimal funding have had free source access to a world-class operating system, allowing them to innovate without incurring onerous licensing fees that may have prevented them from ever escaping their garage.

Another important example of open source is the OpenSSL implementation [20]. Encryption software is notoriously complex, yet is required in more and more products in today's security-conscious world. This very complete body of asymmetrical and symmetrical encryption protocols and key algorithms has been a boon to many researchers and technology start-ups that could never afford to develop or license this technology, yet need it to begin pursuing their security-related ideas.

In the networking world, open-source implementations of routing protocols such as BGP, OSPF, RIP, and switching functionality such as Spanning Tree, have been available for many years now. Unlike the PC world, where a common *Wintel* hardware platform has made it possible to develop software trivially ported to numerous PC platforms, the hardware platforms from different *Network Equipment Manufacturers* (NEMs) remain quite divergent. Thus, porting the open source implementations of these complex protocols to the proprietary hardware platforms of network switches has always been labor-intensive. Moving the control plane and its associated complex protocols off of the proprietary switch hardware and onto a centralized server built on a PC platform makes the use of open source control plane software much more feasible.

It is not possible to fully grasp the origins of SDN without understanding that the early experimentation with SDN was done in a highly collaborative and open environment. Specifications and software were exchanged freely between participating university groups. An impressive amount of pioneering engineering work was performed without any explicit commercial funding, all based on the principle that something was being created for the greater good. It truly was and is a kind of network-world reflection of the Linux phenomenon. With few exceptions, the most vocal supporters of SDN have been individuals and institutions that did not intend to extract financial profit from the sales of the technology. This made a natural marriage with open source possible. We will see in later chapters of this book that SDN has begun to have such a large commercial impact that an increasing number of proprietary implementations of SDN-related technology are reaching the market. This is an unsurprising companion on almost any journey where large sums of money can be made. Nevertheless, there remains a core SDN community that believes that openness should remain an abiding characteristic of SDN.

DISCUSSION QUESTION:

While open source implementations of many control plane protocols exist today, it remains challenging to port these to traditional switches. What characteristic(s) of SDN could potentially make open source control plane software more ubiquitous?

1.8 ORGANIZATION OF THE BOOK

The first two chapters of the book define terminology about packet switching and provide a historical context for evolution of the data, control, and management planes extant in today's switches. We describe some of the limitations of the current technology and explain how these motivated the development of SDN technology. In Chapter 3 we explain the historical origins of SDN. Chapter 4 presents the key interfaces and protocols of SDN and explains how the traditional control plane function is implemented in a novel way. Up to this point in the book our references to SDN are fairly generic. Also in Chapter 4 we begin to distinguish the original or *Open* SDN from proprietary alternatives that also fall under the SDN umbrella. When we use the term *Open* SDN in this work, we refer to SDN as it was developed and defined in research institutions starting around 2008. Open SDN is tightly coupled with the *OpenFlow* protocol presented in Chapter 5 and also strongly committed to the openness of the protocols and specifications of SDN. Alternative definitions of SDN exist that share some but not all of the basic precepts of open SDN. In Chapter 6 we provide more details about these alternative SDN implementations, as well as a discussion about limitations and potential drawbacks of using SDN. Since this book was originally published, commonly accepted definitions of what constitutes SDN have been stretched considerably. SDN control protocols and controller models that formerly lay on the sidelines have become mainstream. We discuss the major new protocols and controllers in Chapter 7. In Chapter 8 we present use cases related to the earliest driver of interest in SDN, the exploding networking demands of the data center. In Chapter 9 we discuss additional use cases that contribute to the current frenzy of interest in this new technology, most notably the application of SDN in wide-area networks. One of the most significant aspects of these use cases is that of *Network Functions Virtualization* (NFV) wherein physical appliances such as *IDS* can be virtualized via SDN. This particular class of SDN use cases is significant enough to merit treatment separately in Chapter 10. The book provides in Chapter 11 an inventory of major enterprises and universities developing SDN products today. Chapter 12 presents several real-world SDN applications, including a tutorial on writing one's own SDN application. Chapter 13 provides a survey of open source initiatives that will contribute to the standardization and dissemination of SDN. In Chapter 14 we examine the impact that this new technology is already making in the business world, and consider what likely future technological and business consequences may be imminent. The final chapter delves into current SDN research directions and considers several novel applications of SDN. There is an appendix of SDN-related acronyms, source code from a sample SDN application and a comprehensive user-friendly index.

REFERENCES

[1] Network effect. The Economist 2012;405(8815):67–8.

[2] Bertsekas D, Gallager R. Data networks. Englewood Cliffs, NJ: Prentice Hall; 1992.

[3] Internet Hall of Fame. Paul Baran. Retrieved from: http://internethalloffame.org/inductees/paul-baran.

[4] Christy P. OpenFlow and Open Networking: an introduction and overview. Ethernet Technology Summit, February 2012, San Jose, CA; 2012.

[5] de Prycker M. Asynchronous transfer mode—solution for broadband ISDN. London: Ellis Horwood; 1992.

[6] Gahsinsky I. Warehouse scale datacenters: the case for a new approach to networking. Open Networking Summit, October 2011. Stanford University; 2011.

[7] Guis I. Enterprise Data Center Networks. Open Networking Summit, April 2012, Santa Clara, CA; 2012.

[8] Kerravala Z. Arista launches new security feature to cover growing East-to-West data center traffic. Network World, October 6; 2015. Retrieved from: http://www.networkworld.com/article/2989761/cisco-subnet/arista-launches-new-security-feature-to-cover-growing-east-to-west-data-center-traffic.html.

[9] Information technology—telecommunications and information exchange between systems—local and metropolitan area networks—common specifications—Part 3: Media Access Control (MAC) Bridges. IEEE P802.1D/D17, May 25, 1998.

[10] Draft Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks—Amendment XX: Shortest Path Bridging. IEEE P802.1aq/D4.6, February 10, 2012.

[11] Perlman R, Eastlake D, Dutt D, Gai S, Ghanwani A. Routing Bridges (RBridges): Base Protocol Specification, RFC 6325. Internet Engineering Task Force; 2011.

[12] Hendrick C. Routing Information Protocol, RFC 1058. Internet Engineering Task Force; 1988.

[13] Moy J. OSPF Version 2, RFC 2328. Internet Engineering Task Force; 1998.

[14] Oran D. OSI IS-IS Intra-domain Routing Protocol, RFC 1142. Internet Engineering Task Force; 1990.

[15] Meyer D, Patel K. BGP-4 protocol analysis, RFC 4274. Internet Engineering Task Force; 2006.

[16] Andersson L, Minei I, Thomas B. LDP specification, RFC 3036. Internet Engineering Task Force; 2007.

[17] Cain B, Deering S, Kouvelas I, Fenner B, Thyagarajan A. Internet Group Management Protocol, Version 3, RFC 3376. Internet Engineering Task Force; 2002.

[18] Fenner B, Meyer D. Multicast Source Discovery Protocol (MSDP), RFC 3618. Internet Engineering Task Force; 2003.

[19] Fenner B, Handley M, Holbrook H, Kouvelas I. Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised), RFC 2362. Internet Engineering Task Force; 2006.

[20] OpenSSL. Retrieved from: https://www.openssl.org.