3.1 The Evolution of Networking Technology

In the earliest days of computing there was no networking; each individual computer occupied an entire room or even an entire floor. As mainframes emerged and began to find their way into our technological consciousness, these computing powerhouses still operated as islands, and any data sharing took place using physical media such as magnetic tapes.

3.1.1 Mainframe Networking: Remote Terminals

Even in the age of mainframes, remote connectivity to the mainframe was needed. This was provided in the form of remote terminal controllers and card readers, which operated as subservient devices known as peripherals, with control residing entirely in the central mainframe. Network connections in this case were simple point-to-point or point-to-multipoint links. Communication was solely between a few connected entities, in this case the mainframe and a small set of remote terminal controllers or card readers. This communication was entirely synchronous, with the peripherals allowed to transmit only when polled by the mainframe.
3.1.2 Peer-to-Peer, Point-to-Point Connections

As computer technology began to move from solely mainframes to the addition of minicomputers, these machines had a greater need to share information in a quick and efficient manner. Computer manufacturers began to create protocols for sharing data between two peer machines. The network in this case was also point-to-point, although the nature of the connection was peer-to-peer in that the two machines (e.g., minicomputers) would communicate with each other and share data as relative equals, at least compared to the earlier mainframe-to-terminal-controller type of connections.

Of course, in these point-to-point connections, the network was trivial, with only the two parties communicating with each other. Control for this communication resided not in any networking device but in the individual computers participating in this one-to-one communication.

3.1.3 Local Area Networks

Eventually, with further evolution of computing toward smaller, independent systems, the need arose for a way to connect these devices in order to allow them to share information and collaborate in a manner that wasn’t required when everything ran on one large mainframe or even on a powerful minicomputer. Hence, local area networking (LAN) technology arrived, with various battles being fought between technologies (e.g., Ethernet/IEEE 802.3 versus Token Ring/IEEE802.5). Notably, these early LANs were running on shared media, so all traffic was seen by every device attached to the network.

IEEE 802.3 emerged as the more popular of these technologies. It uses Carrier-Sense Multiple-Access/Collision Detect (CSMA/CD) technology, which exhibits poor aggregate throughput when the number of active devices reaches a certain level. The exact number of devices where this would occur is dependent on the amount of data each device attempts to transmit. This decrease in performance resulted from each device backing off and waiting to transmit when a collision occurs, as stipulated by CSMA/CD. The number of collisions reaches a critical point as a function of the number of nodes and their respective transmission activity. Once this critical point is reached, the network becomes very inefficient, with too much time spent either in the midst of or recovering from collisions.

These flat, shared-media networks were quite simple, and the repeater devices provided physical extensions to the shared media by merely forwarding all frames to the extended medium. The greatest impact of these early devices was a new topology created by wire concentration. This made layer two networks more deployable and reliable than the former method of snaking a coaxial cable from one node to the next. This early networking required minimum control plane intelligence, if any. Simple repeaters basically just did one thing: They forwarded to everybody. More advanced, managed repeaters did have limited control planes where segmentation and error monitoring occurred. Managed repeaters did perform functions such as removing erroneous traffic from the network, as well as removing isolated ports that were causing problems. Segmentable repeaters were able to split themselves into multiple repeaters via configuration. Different groups of ports could be configured to reside in different collision domains. These separate collision domains would be connected by bridges. This feature provided more control over the size of the collision domains to optimize performance.

3.1.4 Bridged Networks

Eventually these shared-media networks needed to scale in physical extent as well as number of nodes. As explained in the previous section, shared-media networks do not scale well as the number of hosts grows.
It became desirable to split the shared-media network into separate segments. In so doing, and since not all the nodes are transmitting all the time, spatial reuse occurs and the aggregate available bandwidth actually increases due to the locality of transmissions in each segment. The first devices to perform this functionality were called bridges, forerunners of today’s switches but much simpler. They typically had only two ports connecting two shared domains. These bridges possessed an incipient control plane in that they were able to actually learn the location and MAC address of all devices and create forwarding tables that allowed them to make decisions about what to do with incoming packets. As we mentioned in the previous chapter, these forwarding tables were implemented entirely in software.

Furthermore, these bridges were implemented in such a way that each device was able to operate independently and autonomously, without requiring any centralized intelligence. The goal was to facilitate expansion of the network without a lot of coordination or interruption across all the devices in the network. One of the first manifestations of this distributed intelligence paradigm was the Spanning Tree Protocol (STP), which allowed a group of bridges to interact with each other and converge on a topology decision (the spanning tree), which eliminated loops and superimposed a hierarchical loop-free structure on the network.

The important point here is that this greater scale in terms of the number of nodes as well as physical extent drove the need for this new model of individually autonomous devices, with distributed protocols implementing the required control functionality. If we translate this period’s realities into today’s terminology, there was no centralized controller, merely a collector of statistics. Policies, if indeed one can use that term, were administered by setting specific parameters on each device in the network. We need to keep in mind that at the time, networks in question were small, and this solution was entirely acceptable.

3.1.5 Routed Networks

In the same manner that bridged and switched networks dealt with layer two domains with distributed protocols and intelligence, similar strategies were employed for layer three routing. Routers were directly connected locally to layer two domains and interconnected over large distances with point-to-point WAN links. Distributed routing protocols were developed to allow groups of interconnected routers to share information about those networks to which they were directly connected. By sharing this information among all the routers, each was able to construct a routing table, allowing it to route packets to remote layer three IP subnets using the optimal forwarding ports. This was another application of autonomous devices, utilizing distributed protocols to allow each to make appropriate forwarding decisions.

This sequence has led to the current state of affairs, with networking intelligence distributed in the networking devices themselves. During this evolution, however, the growth of network size and complexity was unrelenting. The size of MAC forwarding tables grew, control plane protocols became more complex, and network overlays and tunneling technology became more prevalent. Making major changes to these implementations was a continual challenge. Since the devices were designed to operate independently, centrally administered, large-scale upgrades were challenging. In addition, the fact that the actual control plane implementations were from many different sources and not perfectly matched created a sort of lowest common denominator effect, where only those features that were perfectly aligned between the varied implementations could truly be relied upon. In short, existing solutions were not scaling well with this growth and complexity. This situation led network engineers and researchers to question whether this evolution was headed in the right direction. In the following section we describe some of the innovations and research that resulted.
3.2 Forerunners of SDN

Prior to OpenFlow, and certainly prior to the birth of the term Software Defined Networking, forward-thinking researchers and technologists were considering fundamental changes to today’s world of autonomous, independent devices and distributed networking intelligence and control. This section considers some of those early explorations of SDN-like technology. There is a steady progression of ideas around advancing networking technology toward what we now know as SDN. Table 3.1 shows this progression, which is discussed in more detail in the subsections that follow. The timeframes shown in the table represent approximate time periods when these respective technologies were developed or applied in the manner described.

3.2.1 Early Efforts

A good survey of early programmable networks that were stepping-stones on the path to SDN can be found in [12]. Some of the earliest work in programmable networks began not with Internet routers and switches but with ATM switches. Two notable examples were Devolved Control of ATM Networks (DCAN) and open signaling.

As its name indicates, DCAN [8] prescribed the separation of the control and management of the ATM switches from the switches themselves. This control would be assumed by an external device that is similar to the role of the controller in SDN networks.

<table>
<thead>
<tr>
<th>Table 3.1 Precursors of SDN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project</strong></td>
</tr>
<tr>
<td>Open signaling</td>
</tr>
<tr>
<td>Active networking</td>
</tr>
<tr>
<td>DCAN</td>
</tr>
<tr>
<td>IP switching</td>
</tr>
<tr>
<td>MPLS</td>
</tr>
<tr>
<td>RADIUS, COPS Orchestration</td>
</tr>
<tr>
<td>Virtualization Manager</td>
</tr>
<tr>
<td>ForCES</td>
</tr>
<tr>
<td>4D</td>
</tr>
<tr>
<td>Ethane</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Open signaling [6] proposed a set of open, programmable interfaces to the ATM switching hardware. The key concept was to separate the control software from the switching hardware. This work led to the IETF effort that resulted in the creation of the General Switch Management Protocol (GSMP) [7]. In GSMP, a centralized controller is able to establish and release connections on an ATM switch as well as a multitude of other functions that may otherwise be achieved via distributed protocols on a traditional router. Tag switching was Cisco’s version of label switching, and both of these terms refer to the technology that ultimately became known as Multiprotocol Label Switching (MPLS). Indeed, MPLS and related technologies are a deviation from the autonomous, distributed forwarding decisions characteristic of the traditional Internet router, and in that sense they were a small step toward a more SDN-like Internet switching paradigm. In the late 1990s, Ipsilon Networks utilized the GSMP protocol to set up and tear down ATM connections internally in the company’s IP Switch product. The Ipsilon IP Switch [13] presented normal Internet router interfaces externally, but its internal switching fabric could utilize ATM switches for persistent flows. Flows were defined as a relatively persistent set of packets between the same two endpoints, where an endpoint was determined by IP address and TCP/UDP port number. Since there was some overhead in establishing the ATM connection that would carry that flow, this additional effort would be expended only if the IP Switch believed that a relatively large number of packets would be exchanged between the endpoints in a short period of time. This definition of flow is somewhat consistent with the notion of flow within SDN networks.

The active networking project [10, 11] also included the concept of switches that could be programmed by out-of-band management protocols. In the case of active networking, the switching hardware was not ATM switches but Internet routers. Active networking also included a very novel proposal for small downloadable programs called capsules that would travel in packets to the routers and could reprogram the router’s behavior on the fly. These programs could be so fine-grained as to prescribe the forwarding rules for a single packet, even possibly the payload of the packet that contained the program.

### 3.2.2 Network Access Control

Network access control (NAC) products control access to a network based on policies established by the network administration. The most basic control is to determine whether or not to admit the user onto the network. This decision is usually accomplished by some exchange of credentials between the user and the network. If the user is admitted, his rights of access will also be determined and restricted in accordance with those policies. In some early efforts, network policy beyond admission control was dynamically provisioned via NAC methods, such as Remote Authentication Dial-In User Service (RADIUS) [16] and Common Open Policy Service (COPS) [14].

RADIUS has been used to provide the automatic reconfiguration of the network. This solution was somewhat forward-looking in that RADIUS can be viewed as a precursor to SDN. The idea is that via RADIUS, networking attributes would change based on the identity of the compute resource that had just appeared and required connectivity to the network. Whereas RADIUS was originally designed for the authentication, authorization, and accounting (AAA) processes related to granting a user access to a network, that original paradigm maps well to our network reconfiguration problem. The identity of the resource connecting to the network serves to identify the resource to the RADIUS database, and the authorization attributes returned from that database could be used to change the networking attributes described above. Solutions of this nature achieved automatic reconfiguration of the network but never
CHAPTER 3

The Genesis of SDN

FIGURE 3.1

Early attempts at SDN: RADIUS.
gained the full trust of IT administrators and, thus, never became mainstream. Although this RADIUS solution did an adequate job of automatically reconfiguring the edge of the network, the static, manually configured core of the network remained the same. This problem still awaited a solution.

Figure 3.1 shows an overview of the layout and operation of a RADIUS-based solution for automating the process of creating network connectivity for a virtual server. In the figure, the process would be that a VM is moved from physical server A to physical server B, and as a result the RADIUS server becomes aware of the presence of this VM at a new location. The RADIUS server is then able to automatically configure the network based on this information, using standard RADIUS mechanisms.

3.2.3 Orchestration

Early attempts at automation involved applications that were commonly labeled orchestrators [17]. Just as a conductor can make a harmonious whole out of the divergent instruments of an orchestra, such applications could take a generalized command or goal and apply it across a wide range of often heterogeneous devices. These orchestration applications would typically utilize common device application programmer interfaces (APIs) such as the Command-Line Interface (CLI) or Simple Network Management Protocol (SNMP).

Figure 3.2 shows an overview of the layout and operation of an orchestration management solution for automating the process of creating network connectivity for a virtual server. The figure shows
SNMP/CLI plug-ins for each vendor’s specific type of equipment; an orchestration solution can then have certain higher-level policies that are in turn executed at lower levels by the appropriate plugins. The vendor-specific plugins are used to convert the higher-level policy requests into the corresponding native SNMP or CLI request specific to each vendor.

Such orchestration solutions alleviated the task of updating device configurations. But since they were really limited to providing a more convenient interface to existing capabilities, and since no capability existed in the legacy equipment for network-wide coordination of more complex policies such as security and virtual network management, tasks such as configuring VLANs remained hard. Consequently, orchestration management applications continued to be useful primarily for tasks such as software and firmware updates but not for tasks that would be necessary in order to automate today’s data centers.

3.2.4 Virtualization Manager Network Plugins

The concept of virtualization manager network plugins builds on the notion of orchestration and attempts to automate the network updates that are required in a virtualization environment. Tools specifically targeted at the data center would often involve virtualization manager plugins (e.g., plugins for VMware’s vCenter [18]), which would be configured to take action in the event of a server change, such as a vMotion [19]. The plugins would then take the appropriate actions on the networking devices they controlled in order to make the network follow the server and storage changes with changes of its own. Generally, the mechanism for making changes to the network devices would be SNMP or CLI commands. These plugins can be made to work, but since they must use the static configuration capabilities of SNMP and CLI, they suffer from being difficult to manage and prone to error.

Figure 3.3 shows an overview of the layout and operation of a virtualization manager plugin solution for automating the process of creating network connectivity for a virtual server. This figure reflects a similar use of SNMP/CLI plugins to that which we saw in Figure 3.2. The most notable difference is that this application starts from a base that supports virtualization. The nature of the reconfiguration managed by this application is oriented to support the networking of virtual machines via VLANs and tunnels.

3.2.5 ForCES: Separation of Forwarding and Control Planes

The forwarding and control element separation (ForCES) [9] work produced in the IETF began around 2003. ForCES was one of the original proposals recommending the decoupling of forwarding and control planes. The general idea of ForCES was to provide simple hardware-based forwarding entities at the foundation of a network device and software-based control elements above. These simple hardware forwarders were constructed using cell-switching or tag-switching technology. The software-based control had responsibility for the broader tasks, often involving coordination between multiple network devices (e.g., BGP routing updates).

The functional components of ForCES are as follows:

- **Forwarding Element.** The forwarding element (FE) would be typically implemented in hardware and located in the network. The FE is responsible for enforcement of the forwarding and filtering rules that it receives from the controller.
- **Control Element.** The control element (CE) is concerned with the coordination between the individual devices in the network and for communication forwarding and routing information to the FEs below.
3.2 Forerunners of SDN

- **Network Element.** The Network Element (NE) is the actual network device that consists of one or more FEs and one or more CEs.
- **ForCES protocol.** The ForCES protocol is used to communicate information back and forth between FEs and CEs.
ForCES proposes the separation of the forwarding plane from the control plane, and it suggests two different embodiments of this architecture. In one of these embodiments, both the forwarding and control elements are located within the networking device. The other embodiment规格ulates that it would be possible to actually move the control element(s) off the device and to locate them on an entirely different system. Although the suggestion of a separate controller thus exists in ForCES, the emphasis is on the communication between CE and FE over a switch backplane, as shown in Figure 3.4.

3.2.6 4D: Centralized Network Control

Seminal work on the topic of moving networking technology from distributed networking elements into a centralized controller appeared in the 4D proposal [2] A Clean Slate 4D Approach to Network Control and Management. 4D, named after the architecture’s four planes—decision, dissemination, discovery, and data—proposes a complete refactoring of networking away from autonomous devices and toward the idea of concentrating control plane operation in a separate and independent system dedicated to that purpose.

4D argues that the state of networking today is fragile and, therefore, often teeters on the edge of failure because of its current design based on distributed, autonomous systems. Such systems exhibit a defining characteristic of unstable, complex systems: a small local event such as a misconfiguration of a routing protocol can have a severe, global impact. The proposal argues that the root cause is the fact that the control plane is running on the network elements themselves.

4D centers around three design principles:

- Network-level objectives. In short, the goals and objectives for the network system should be stated in network-level terms based on the entire network domain, separate from the network elements, rather than in terms related to individual network device performance.
3.2 Forerunners of SDN

Network-wide view. There should be a comprehensive understanding of the whole network. Topology, traffic, and events from the entire system should form the basis on which decisions are made and actions are taken.

Direct control. The control and management systems should be able to exert direct control over the networking elements, with the ability to program the forwarding tables for each device rather than only being able to manipulate some remote and individual configuration parameters, as is the case today.

Figure 3.5 shows the general architecture of a 4D solution, with centralized network control via the control and management system.

One aspect of 4D that is actually stated in the title of the paper is the concept of a clean slate, meaning the abandoning of the current manner of networking in favor of this new method, as described by the three aforementioned principles. A quote from the 4D proposal states that “We hope that exploring an extreme design point (the clean-slate approach) will help focus the attention of the research and industrial communities on this crucially important and intellectually challenging area.”

The 4D proposal delineates some of the challenges faced by the proposed centralized architecture. These challenges continue to be relevant today in SDN. We list a few of them here:

- **Latency.** Having a centralized controller means that a certain (hopefully small) number of decisions will suffer nontrivial round-trip latency as the networking element requests policy directions from the controller. The way this delay impacts the operation of the network, and to what extent, remains to be determined. Furthermore, with the central controller providing policy advice for a number of network devices, it is unknown whether the conventional servers on which the controller runs will be able to service these requests at sufficient speed to have minimal or no impact on network operation.
• **Scale.** Having a centralized controller means that responsibility for the topological organization of the network, determination of optimal paths, and responses to changes must be handled by the controller. As has been argued, this is the appropriate location for this functionality; however, as more and more network devices are added to the network, questions arise of scale and the ability of a single controller to handle all those devices. It is difficult to know how well a centralized system can handle hundreds, thousands, or tens of thousands of network devices and to know what is the solution when the number of network devices outgrows the capacity of the controller to handle them. If we attempt to scale by adding controllers, how do they communicate, and who orchestrates coordination among the controllers? Section 6.1.3 addresses these questions.

• **High availability (HA).** The centralized controller must not constitute a *single point of failure* for the network. This implies the need for redundancy schemes in a number of areas. First, there must be redundant controllers such that processing power is available in the event of failure of a single controller. Second, the actual data used by the set of controllers needs to be mirrored such that the controllers can program the network devices in a consistent fashion. Third, the communications paths to the various controllers need to be redundant to ensure that there is always a functioning communications path between a switch and at least one controller. We further discuss high availability in the context of a modern SDN network in Section 6.1.2.

• **Security.** Having a centralized controller means that security attacks are able to focus on that one point of failure, and thus the possibility exists that this type of solution is more vulnerable to attack than a more distributed system. It is important to consider what extra steps must be taken to protect both the centralized controller and the communication channels between it and the networking devices.

ForCES and 4D contributed greatly to the evolution of the concepts that underlie SDN: separation of forwarding and control planes (ForCES) and having a centralized controller responsible for overall routing and forwarding decisions (4D). However, both of these proposals suffer from a lack of actual implementations. Ethane, examined in the following section, benefited from the experience that can only come from a real-life implementation.

### 3.2.7 Ethane: Controller-Based Network Policy

Ethane was introduced in a paper titled *Rethinking Enterprise Network Control* [1] in 2007. Ethane is a policy-based solution that allows network administrators to define policies pertaining to network-level access for users, which includes authentication and quarantine for misbehaving users. Ethane was taken beyond the proposal phase. Multiple instances have been implemented and shown to behave as suggested in [1].

Ethane is built around three fundamental principles:

1. **The network should be governed by high-level policies.** Similar to 4D, Ethane espouses the idea that the network be governed by policies defined at high levels rather than on a per-device basis. These policies should be at the level of services and users and the machines through which users can connect to the network.

2. **Routing for the network should be aware of these policies.** Paths that packets take through the network are to be dictated by the higher-level policies described in the previous bullet point rather than as in the case of today’s networks, in which paths are chosen based on lower-level directives.
For example, guest packets may be required to pass through a filter of some sort, or certain types of traffic may require routing across lightly loaded paths. Some traffic may be highly sensitive to packet loss; other traffic (e.g., Voice over IP, or VoIP) may tolerate dropped packets but not latency and delay. These higher-level policies are more powerful guidelines for organizing and directing traffic flows than are low-level and device-specific rules.

3. The network should enforce binding between packets and their origin. If policy decisions rely on higher-level concepts such as the concept of a user, then the packets circulating in the network must be traceable back to their point of origin (i.e., the user or machine that is the actual source of the packet).

Figure 3.6 illustrates the basics of the Ethane solution. As will become apparent in the following chapters, there are many similarities between this solution and OpenFlow.

To test Ethane, the researchers themselves had to develop switches in order to have them implement the protocol and the behavior of such a device—that is, a device that allows control plane functionality to be determined by an external entity (the controller) and that communicates with that external entity via a protocol that allows flow entries to be configured into its local flow tables, which then perform the forwarding functions as packets arrive.

The behavior of the Ethane switches is generally the same as today’s OpenFlow switches, which forward and filter packets based on the flow tables that have been configured on the device. If the switch does not know what to do with the packet, it forwards it to the controller and awaits further instructions.
In short, Ethane is basically a Software Defined Networking technology, and its components are the antecedents of OpenFlow, which we describe in detail in Chapter 5.

### 3.3 Software Defined Networking is Born

#### 3.3.1 The Birth of OpenFlow

Just as the previous sections presented standards and proposals that were precursors to SDN, seeing SDN through a gestation period, the arrival of OpenFlow is the point at which SDN was actually born. In reality, the term SDN did not come into use until a year after OpenFlow made its appearance on the scene in 2008, but the existence and adoption of OpenFlow by research communities and networking vendors marked a sea change in networking, one that we are still witnessing even now. Indeed, though the term SDN was in use in the research community as early as 2009, SDN did not begin to make a big impact in the broader networking industry until 2011.

For reasons identified in the previous chapter, OpenFlow was developed and designed to allow researchers to experiment and innovate with new protocols in everyday networks. The OpenFlow specification encouraged vendors to implement and enable OpenFlow in their switching products for deployment in college campus networks. Many network vendors have implemented OpenFlow in their products.

The OpenFlow specification delineates both the protocol to be used between the controller and the switch as well as the behavior expected of the switch. Figure 3.7 illustrates the simple architecture of an OpenFlow solution.

![General OpenFlow design](image_url)

**FIGURE 3.7**

General OpenFlow design.
The following list describes the basic operation of an OpenFlow solution:

- The controller populates the switch with flow table entries.
- The switch evaluates incoming packets and finds a matching flow, then performs the associated action.
- If no match is found, the switch forwards the packet to the controller for instructions on how to deal with the packet.
- Typically the controller will update the switch with new flow entries as new packet patterns are received, so that the switch can deal with them locally. It is also possible that the controller will program wildcard rules that will govern many flows at once.

OpenFlow is examined in detail in Chapter 5. For now, though, the reader should understand that OpenFlow has been adopted by both the research community and by a number of networking vendors. This has resulted in a significant number of network devices supporting OpenFlow on which researchers can experiment and test new ideas.

### 3.3.2 Open Networking Foundation

OpenFlow began with the publication of the original proposal in 2008. By 2011 OpenFlow had gathered enough momentum that the responsibility for the standard itself moved to the Open Networking Foundation (ONF). The ONF was established in 2011 by Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo! It is now the guardian of the OpenFlow standard and consists of a number of working groups, many of which are listed in Table 3.2.

<table>
<thead>
<tr>
<th>Workgroup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensibility</td>
<td>Ongoing creation of new OpenFlow versions to support more features and implementations.</td>
</tr>
<tr>
<td>Architecture and Framework</td>
<td>Defining the scope of SDN that the ONF will attempt to standardize.</td>
</tr>
<tr>
<td>Forwarding Abstractions</td>
<td>Looking at how the standard interacts with actual hardware implementation components.</td>
</tr>
<tr>
<td>Testing and Interoperability</td>
<td>Defining test and interoperability criteria, certification, etc.</td>
</tr>
<tr>
<td>Configuration and Management</td>
<td>Defining a protocol for configuring non-flow-related OpenFlow parameters (e.g., setup, controller, etc.)</td>
</tr>
<tr>
<td>Migration</td>
<td>Study migration of existing networks towards an SDN network based on OpenFlow.</td>
</tr>
<tr>
<td>Optical Transport</td>
<td>Responsible for identifying areas in which the OpenFlow Standard can be applied to optical networks.</td>
</tr>
<tr>
<td>Market Education</td>
<td>Responsible for educating the SDN community on the value of OpenFlow and for channeling market feedback within the ONF.</td>
</tr>
<tr>
<td>Wireless &amp; Mobile</td>
<td>Study methods by which OpenFlow can be used to control wireless radio area networks (RANs) and core networks.</td>
</tr>
<tr>
<td>Northbound Interface</td>
<td>Developing concrete requirements, architecture, and working code for northbound interfaces.</td>
</tr>
</tbody>
</table>
One novel aspect of the ONF is that its board of directors is entirely made up of major network operators, not the networking vendors themselves. As of this writing, the ONF board is composed of chief technology officers (CTOs), technical directors, and fellows from companies such as Google, Facebook, Deutsche Telekom, Verizon, Microsoft, and NTT, among others. This helps prevent the ONF from supporting the interests of one major networking vendor over another. It also helps provide a real-world perspective on what should be investigated and standardized. Conversely, it runs the risk of defining a specification that is difficult for NEMs to implement. Our previous comments about NEMs being locked into their status quo notwithstanding, it is also true that there is a wealth of engineering experience resident in the NEMs regarding how to actually design and build high-performance, high-reliability switches. Though the real-world experience of the users is indeed indispensable, it is imperative that the ONF seek input from the vendors to ensure that the specifications they produce are in fact implementable.

### 3.4 Sustaining SDN Interoperability

At the current point in the evolution of SDN, we have a standard that has been accepted and adopted by academia and industry alike, and we have an independent standards body to shepherd OpenFlow forward and to ensure that it remains independent and untethered to any specific institution or organization. It is now important to ensure that the implementations of the various players in the OpenFlow space adhere to the standards as they are defined, clarify and expand the standards where they are found to be incomplete or imprecise, and in general guarantee interoperability among OpenFlow implementations. This goal can be achieved in a few ways:

- **Plugfests.** Plugfests, normally staged at conferences, summits, and congresses, are environments in which vendors can bring their devices and software in order to test them with devices and software from other vendors. These are rich opportunities to determine where implementations may be lacking or where the standard itself is unclear and needs to be made more precise and specific.

- **Interoperability labs.** Certain institutions have built dedicated test labs for the purpose of testing the interoperability of equipment from various vendors and organizations. One such lab is the *Indiana Center for Network Translational Research and Education* (InCNTRE) at Indiana University, which hosts a large collection of vendor devices and controllers as well as experimental devices and controllers from open source contributors. We discuss open source contributions to SDN in the next section.

- **Certification programs.** There is a need for certification of switches so that buyers can know they are getting a switch that is certified to support a particular version or versions of OpenFlow. The ONF has now implemented such a program [5].

- **Education and consulting.** A complex, game-changing technological shift such as that represented by SDN will not easily permeate a large industry without the existence of an infrastructure to train and advise networking staff about the migration. It is important that a cadre of highly qualified yet vendor-neutral organizations address this need.

Initially, many SDN interoperability tests revealed dissimilarities and issues. Existing implementations of OpenFlow are increasingly interoperable, but challenges remain. For example, as of this writing, it remains difficult for an OpenFlow controller to work consistently across multiple, varied
switch implementations of OpenFlow. This is largely due to limitations in existing ASICs that lead to switches only supporting different subsets of OpenFlow consistent with their particular hardware. Other problems emanate from the fact that the OpenFlow 1.0 specification is vague in terms of specifying which features are required versus which are optional. As we discuss in Chapter 5, the OpenFlow standard is not static, and the goalposts of interoperability are moved with each new release.

3.5 Open Source Contributions

One of the basic rationales for SDN is that innovation and advancement have been stifled as a result of the closed and inflexible environment that exists in networking today. Thus, the openness that results from the creation of standards such as OpenFlow should encourage researchers to dissect old networking methods and should usher in a new dawn of network operation, management, and control. This section examines ways in which open source contributes to this process.

3.5.1 The Power of the Collective

Technological advancement sometimes arises from the efforts of corporations and major organizations, quite often due to the fact that they are the only ones in the position to make contributions in their domains. In the world of software, however, it is occasionally possible for small players to develop technology and make it freely available to the general public. Some examples are:

- **Operating systems.** The Linux operating system was developed as open source and is used today to control countless devices that we use every day, from digital video recorders (DVRs) to smartphones.
- **Databases.** Many of the websites we visit for news reports or to purchase products store their information and product data in databases that were developed, at least initially, by the open source community. MySQL is an example of such an open source database.
- **Servers.** When we access locations on the Internet, many of those servers are running application server software and using tools that have been developed over time by the open source community. For example, the open source Apache Web Server is used in countless applications worldwide.
- **Security.** Applying the open source model is also often considered in order to deliver more secure environments [20]. Open source can be more secure because of the peer review and white-box evaluation that naturally occur in the open source development paradigm. Proprietary protocols may be less secure because they are not open and evaluated. Many security products providing antivirus protection and maintaining lists of malicious sites and programs are running open source software to accomplish their tasks. OpenSSL is probably the foremost example of a widely used open source encryption toolkit.
- **File sharing.** The BitTorrent protocol is an example of a hugely successful [3] open protocol used for file sharing. BitTorrent works as a P2P/overlay network that achieves high-bandwidth file downloading by performing the download of a file piecemeal and in parallel from multiple servers.

These are just a few examples. Imagine our world today without those items. Would private institutions have eventually implemented the software solutions required in order to provide that functionality? Most likely. Would these advancements in technology have occurred at the velocity that we have witnessed in the past 10 years, without current and ongoing contributions from open source? Almost certainly not.
3.5.2 The Danger of the Collective

Of course, with an endeavor being driven by individuals who are governed not only by their own desire to contribute but also by their free will, whims, and other interests, there is bound to be some risk. The areas of risk include quality, security, timeliness, and support. Here we explain how these risks can be mitigated.

Open source software must undergo tests and scrutiny by even larger numbers of individuals than its commercial counterpart. This is due to the fact that an entire world of individuals has access to and can test those contributions. For any given feature being added or any problem being solved, the open source community may offer a number of competing approaches to accomplish the task. Even open source initiatives are subject to release cycles, and there are key individuals involved in deciding what code will make its way into the next release. Just because an open source developer creates a body of code for an open source product does not mean that code will make it into a release. To keep quality high, competing approaches may be assessed by the community, and admission into a release is actually controlled by key individuals associated with the open source effort. These same factors serve to minimize the risk of security threats. Since the source code is open for all to view, it is more difficult to hide malicious threats such as back doors.

Timeliness is certainly an issue since there are no committed schedules, especially since open source contributors are often doing so in their spare time. The prudent approach is not to depend on future deliverables, but to use existing functionality.

There seem to be two fundamental solutions to the issue of support. If you are a business intending to utilize open source in a product you are developing, you need to feel either that you have the resources to support it on your own or that there is such a large community that has been using that code for a long enough time that you simply trust that the bad bugs have already surfaced and that you are using a stable code base.

It is important to remember that open source licensing models differ greatly from one model to the next. Some models severely restrict the way contributions can be made and are used. Section 11.2 discusses in more detail some of the different major open source licenses and the issues with them.

3.5.3 Open Source Contributions to SDN

Based on the previous discussions, it is easy to see the potential value of open source contributions in the drive toward SDN. Huge advances in SDN technology are attributable to open source projects. Multiple open source implementations of SDN switches, controllers, and applications are available. In Chapter 11 we provide details of the open source projects that have been specifically targeted to accelerate innovation in SDN. In that chapter we also discuss other open source efforts that, though not as directly related to SDN, are nonetheless influential on the ever-growing acceptance of the SDN paradigm.

3.6 Legacy Mechanisms Evolve Toward SDN

The urgent need for SDN we described in Chapter 2 could not wait for a completely new network paradigm to be fleshed out through years of research and experimentation. It is not surprising, then, that there were early attempts to achieve some SDN-like functionality within the traditional
The capabilities of legacy switches were sometimes extended to support detailed policy configuration related to security, QoS, and other areas. Old APIs were extended to allow centralized programming of these features. Some SDN providers have based their entire SDN solution on a rich family of extended APIs on legacy switches, orchestrated by a centralized controller. In Chapter 6 we examine how these alternative solutions work and whether or not they genuinely constitute an SDN solution.

### 3.7 Network Virtualization

In Chapter 2 we discussed how network virtualization lagged behind its compute and storage counterparts and how this has resulted in a strong demand for network virtualization in the data center. Network virtualization, in essence, provides a network service that is decoupled from the physical hardware below that offers a feature set identical to the behavior of its physical counterpart. An important and early approach to such network virtualization was the virtual local area network (VLAN). VLANs permitted multiple virtual local area networks to co-reside on the same layer two physical network in total isolation from one another. Although this technical concept is very sound, the provisioning of VLANs is not particularly dynamic, and they scale only to the extent of a layer two topology. Layer three counterparts based on tunneling scale better than VLANs to larger topologies. Complex systems have evolved to use both VLAN as well as tunneling technologies to provide network virtualization solutions.

One of the most successful commercial endeavors in this space was Nicira, now part of VMware. Early on, Nicira claimed that there were seven properties of network virtualization [4]:

1. Independence from network hardware
2. Faithful reproduction of the physical network service model
3. Following an operational model of compute virtualization
4. Compatibility with any hypervisor platform
5. Secure isolation among virtual networks, the physical networks, and the control plane
6. Cloud performance and scale
7. Programmatic network provisioning and control

Several of these characteristics closely resemble what we have said is required from an SDN solution. SDN promises to provide a mechanism for automating the network and abstracting the physical hardware below from the Software Defined Network above. Network virtualization for data centers has undoubtedly been the largest commercial driver behind SDN. This momentum has become so strong that to some, network virtualization has become synonymous with SDN. Indeed, VMware’s (Nicira) standpoint [15] on this issue is that SDN is simply about abstracting control plane from data plane, and therefore network virtualization is SDN. Well, is it SDN or not?

### 3.8 May I Please Call My Network SDN?

If one were to ask four different attendees at a 2013 networking conference what they thought qualified a network to be called SDN, they would likely have provided divergent answers. Based on the genesis of SDN as presented in Section 3.3, in this book we define an SDN network as characterized by five
fundamental traits: plane separation, a simplified device, centralized control, network automation and virtualization, and openness. We call an SDN solution possessing these five traits an Open SDN technology. We acknowledge that there are many competing technologies offered today that claim that their solution is an SDN solution. Some of these technologies have had larger economic impact in terms of the real-life deployments and dollars spent by customers than those that meet all five of our criteria. In some respects they may address customers’ needs better than Open SDN. For example, a network virtualization vendor such as Nicira has had huge economic success and widespread installations in data centers but does this without simplified devices. We define these five essential criteria not to pass judgment on these other SDN solutions but in acknowledgment of what the SDN pioneers had in mind when they coined the term SDN in 2009 to refer to their work on OpenFlow. We provide details about each of these five fundamental traits in Section 4.1 and compare and contrast competing SDN technologies against these five as well as other criteria in Chapter 6.

3.9 Conclusion

With the research and open source communities clamoring for an open environment for expanded research and experimentation, as well as the urgent needs of data centers for increased agility and virtualization, networking vendors have been forced into the SDN world. Some have moved readily into that world; others have dragged their feet or have attempted to redefine SDN. In the next chapter we examine what SDN is and how it actually works.

References