CHAPTER

EMERGING PROTOCOL, CONTROLLER, AND APPLICATION MODELS

7

The landscape of SDN technologies is changing on a day-to-day basis. This makes keeping a book about SDN up to date more challenging than would be the case for a more established technology. In this chapter we explore new and emerging trends in SDN.

7.1 EXPANDED DEFINITIONS OF SDN

When the first edition of this book was published, the commercial SDN landscape was fluid as different vendors and startups sought to develop SDN solutions to customer issues. However, from a technological standpoint, SDN was fairly stable. SDN had emerged from universities in the years following the *Clean Slate* program we reviewed in Section 3.2.6. Most vendors planned to use OpenFlow in their SDN solutions. In the past few years, as major networking incumbents entered the SDN marketplace, we have begun to see changes. In the next sections we examine the impact these vendors have brought to bear on the SDN industry.

7.1.1 IMPACT OF MAJOR NEMS IN THE SDN ARENA

In the earlier days of SDN, the networking vendors most involved were those more interested in industry disruption. These were the vendors who would benefit most from a change to the status quo. They wanted to establish a beachhead in the various markets that were dominated by the larger vendors such as Cisco. Therefore, companies such as Hewlett-Packard, IBM, and NEC, as well as a number of startups, developed solutions oriented around Open SDN and the OpenFlow protocol.

When SDN began to take root and major vendors such as Cisco and Juniper took notice, this landscape and even the very definition of SDN began to shift. In Sections 4.6.1 and 6.2 we discussed the definition of *SDN via APIs*, which relied not on the new OpenFlow-based models of implementing SDN, but on more traditional models based on existing protocols. This contest has continued since the first edition of this book, with momentum swinging away from Open SDN toward API-based SDN solutions involving existing protocols.

For established vendors this achieved different goals, the priority of which depend on one's perspective. Two of these goals were:

- **Customer Protection**: One argument in favor of using traditional protocols is that it protects customer investments in current devices and technologies. Customers generally do not want to replace their networking equipment wholesale, and even a gradual evolution to a new paradigm carries potential risk and must be carefully evaluated.

Software Defined Networks. http://dx.doi.org/10.1016/B978-0-12-804555-8.00007-7

- **Vendor Protection**: Another argument is that using existing protocols and devices helps established vendors continue to dominate markets in which they are already the leader.

The emerging trends toward legacy protocols such as *Network Configuration Protocol* (NETCONF) and *Border Gateway Protocol* (BGP), and toward the SDN controllers that support these protocols are discussed in the sections that follow.

DISCUSSION QUESTION:

Discuss whether you believe that the influence on SDN by dominant networking vendors is a positive or negative influence on the advancement of networking technology.

7.1.2 NETWORK MANAGEMENT VERSUS SDN

One of the protocols that is being used for the application of SDN-based policies is NETCONF, which we examine in detail in the sections that follow. But the use of such a protocol, which was developed specifically as a means of improving the effectiveness of network management, raises the issue of where network management ends and SDN begins. Is this type of solution just an improved network management or is it really software defined networking?

In Chapter 6 we compared and contrasted three classes of SDN solutions: *Open SDN*, *SDN via APIs*, and *SDN via Overlays*. As that chapter illustrated, it is difficult to precisely circumscribe SDN. For the purposes of this discussion, since network management in general shares many of the same attributes as SDN (i.e., centralized control, network-wide views, network-wide policies), we consider such network management-based solutions to also fall under the larger SDN umbrella.

Fig. 7.1 shows the spectrum of solutions being promoted as SDN. On the right hand side of the picture is *reactive* OpenFlow, which involves packets getting forwarded to the controller via PACKET_IN messages. This type of SDN solution is the most dramatically different from traditional networking technologies, as highlighted in the figure. At the other end of the spectrum is network management, which is the most similar to what we see in traditional networks today. Between those extremes reside NETCONF, *Border Gateway Protocol Link State* (BGP-LS) and *Path Computation Element Protocol* (PCE-P), which we describe later in this chapter. Note that the term PCEP is frequently used as a substitute for PCE-P.

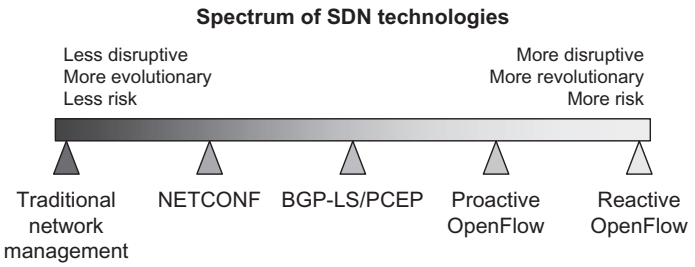

FIG. 7.1

SDN spectrum.

These SDN solutions all are considered valid approaches to SDN, based on the current and common use of the term, although they differ in their approach, their implementation and their suitability for various customer and domain needs. Each has benefits and limitations, which we examine next. The reader may wish to occasionally refer to Fig. 7.1 as our discussion below will generally follow the figure moving from left to right.

7.1.3 BENEFITS OF NETWORK MANAGEMENT-BASED SDN

We list here some benefits of SDN based on an evolved version of network management:

- **Least Disruptive**: This type of SDN is least disruptive to the current operation of the network because it operates with the current network infrastructure and capabilities of the networking staff.
- **Least Costly**: This type of SDN does not require new equipment, nor does it require a great deal of new training of IT personnel.
- **Least Risky**: This type of SDN introduces the least amount of risk into the network, since it runs on the same hardware and device software, with changes to networking behavior limited to what the SDN application is able to do through more traditional network management channels.

These advantages make network management-based SDN attractive to customers who want to eventually reach an SDN-based future through a gradual, more evolutionary path. Indeed, such solutions do move networking technology toward an SDN-based future.

7.1.4 LIMITATIONS OF NETWORK MANAGEMENT-BASED SDN

While there are benefits of network management-based SDN, there are limitations as well:

- **Limited Improvement**: Because it is restricted to using current devices, often without the capabilities for controlling forwarding and shaping of traffic, this type of SDN is limited in how much improvement it can provide in a network.
- **Limited Innovation**: Because this type of SDN is restricted to using currently configurable functionality in devices, it limits opportunities for truly innovative networking.
- **Limited Business Opportunity**: From an entrepreneurial standpoint, this type of SDN may not provide the opportunity to create disruptive and revolutionary new players in the networking device industry.

Customers with established networking environments may decide that the benefits of network management-based SDN outweigh the limitations, while others may opt for the more radical change offered by protocols such as OpenFlow.

DISCUSSION QUESTION:

Network management has been around for some time, and was never considered any type of "software defined networking." Now it is. Discuss whether you think that network management-based SDN should be considered "real" SDN or not, and defend your point of view.

7.2 ADDITIONAL SDN PROTOCOL MODELS

This book has predominantly focused on OpenFlow as the original and the most prominent SDN protocol being used in research, entrepreneurial efforts, and even in some established commercial environments (e.g., Google). However, current trends indicate that much SDN development today focuses on other protocols. We examine these protocols, controllers, and application development trends in the following sections.

7.2.1 USING EXISTING PROTOCOLS TO CREATE SDN SOLUTIONS

Assuming that one of the goals of emerging SDN solutions is to reduce risk and to make use of existing customer and vendor equipment, it is consistent that these solutions utilize existing protocols when feasible. Established protocols such as NETCONF, BGP, and *Multiprotocol Label Switching* (MPLS) are all potentially relevant here. These are mature protocols that are used in massively scaled production networks. Utilizing these protocols to implement SDN solutions makes sense for those interested in directing their SDN efforts along an evolutionary path utilizing existing technologies.

Fig. 7.2 shows a real-life example of existing protocols being used to create an SDN solution. The figure depicts some of the main components involved in the *OpenDaylight* (ODL) controller's BGP-LS/PCE-P plugin. Starting at the top of the figure, we see that there are three distinct protocols involved:

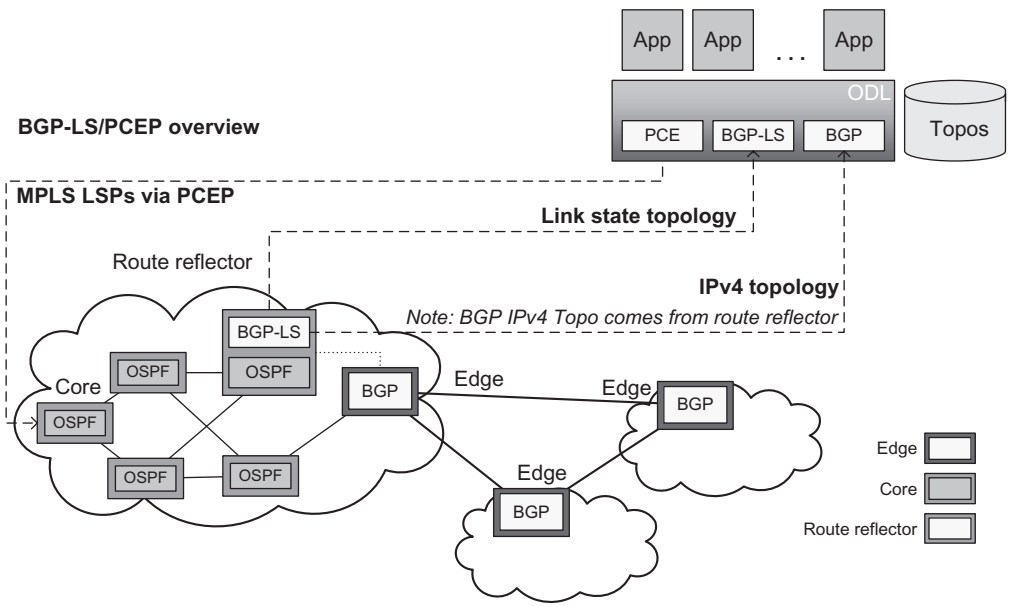

FIG. 7.2

BGP-LS/PCE-P overview.

- **BGP-LS**: The BGP-LS protocol is used by ODL to gather link state topology information from the routing protocols running in the clouds in the figure. This topology reflects routers and interconnecting links within the *Open Shortest Path First* (OSPF) or *Intermediate System to Intermediate System* (IS-IS) domains.
- **BGP**: The BGP protocol is used by ODL to gather IP *Exterior Gateway Protocol* (EGP) topology from the BGP routers connecting the clouds (domains) in the picture.
- **PCE-P**: The PCE-P protocol is used by ODL to configure MPLS *Label Switched Paths* (LSPs) for forwarding traffic across those networks.

The SDN solution in Fig. 7.2 will be discussed further in the following sections. The interested reader can find detailed information in [1]. In the next sections we will examine these protocols as well as NETCONF in order to understand their roles in SDN.

As we consider this use of existing protocols, a helpful perspective may be to look at the different control points shown in Fig. 7.3 that are managed and configured by the SDN application. These control points are *Config*, where general configuration is done, *Routing Information Base* (RIB), where routes (e.g., prefixes and next-hops) are set, and *Forwarding Information Base* (FIB), which is lower level and can be considered *flows*, where packet headers are matched and actions are taken.

The association between these control points and existing protocols is shown in Table 7.1. The table shows control points in general terms. NETCONF's role is for setting configuration parameters. BGP is

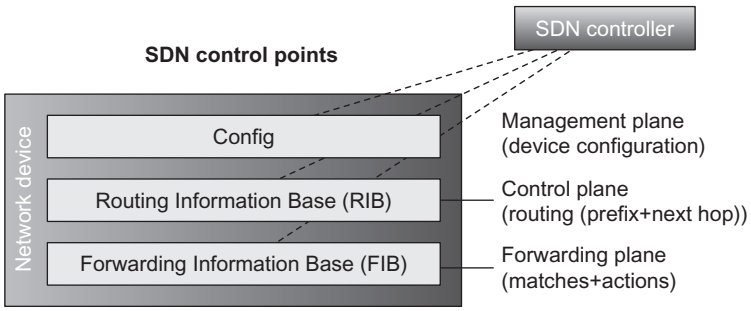

FIG. 7.3

SDN control points.

Table 7.1 Comparison of Existing Protocols for SDN

Protocol	Control Point	Details
NETCONF	Config	Interfaces, ACLs, Static routes
BGP-LS	-	Topology discovery is used to pass link-state IGP information about topology to ODL.
BGP	RIB	Topology discovery and setting RIB
PCE-P	MPLS	PCE to set MPLS LSPs. Used to transmit routing information from the PCE Server to the PCE Clients in the network.
BGP-FS	Flows	BGP-FlowSpec to set matches and actions

involved in setting RIB entries, and PCE-P is used for setting MPLS paths through the network. BGP-LS is used to gather topology information from the RIB. *BGP-FlowSpec* (BGP-FS) is employed to set matches and actions, similar to what is done with OpenFlow, using instead the BGP-FS functionality of the router. BGP-FS leverages the BGP *Route Reflection* infrastructure and can use BGP *Route Targets* to define which routers get which routes. Unlike OpenFlow, BGP-FS does not support layer 2 matches but only layer 3 and above.

7.2.2 USING THE NETCONF PROTOCOL FOR SDN

NETCONF is a protocol developed in an *Internet Engineering Task Force* (IETF) working group and became a standard in 2006, published in *Request for Comments* (RFC) 4741 [2] and later revised in 2011 and published in RFC 6241 [3]. The protocol was developed as a successor to the *Simple Network Management Protocol* (SNMP) and attempted to address some of SNMP's shortcomings. Some key attributes of NETCONF are:

- **Separation of configuration and state (operational) data.** Configuration data is set on the device to cause it to operate in a particular way. State (operational) data is set by the device as a result of dynamic changes on the device due to network events and activities.
- **Support for *Remote Procedure Call* (RPC)-like functionality.** Such functionality was not available in SNMP. With NETCONF, it is possible to invoke an operation on a device, passing parameters and receiving returned results, much like RPC calls in the programming paradigm.
- **Support for Notifications.** This capability is a general event mechanism, whereby the managed device can notify the management station of significant events. Within SNMP this concept is called a trap.
- **Support for transaction-based configurations.** This allows for the configuration of multiple devices to be initiated, but then rolled back in case of a failure at some point in the process.

NETCONF is a management protocol and as such it has the ability to configure only those capabilities which are exposed by the device. Fig. 7.4 illustrates the difference between a

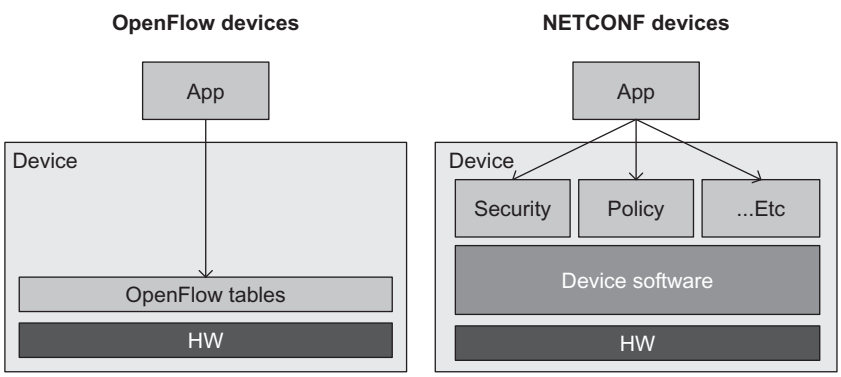

FIG. 7.4

NETCONF versus OpenFlow.

NETCONF-controlled and an OpenFlow-controlled device. We see in the figure that OpenFlow configures the lower levels of the networking device, that is, the ASIC containing the TCAM. This entails setting the matches and actions of the FIB.

NETCONF, on the other hand, performs traditional configuration of the device but via the NETCONF protocol rather than via the CLI or SNMP. The SDN application developer is limited by what the device exposes as configurable. If the device exposes NETCONF data models that allow for the configuration of *Access Control Lists* (ACLs), then the application can configure those. Similarly, if the device has data models for configuring *Quality of Service* (QoS) or static routes, then those will be possible as well.

The NETCONF programmer can learn which capabilities are exposed via the *Yet Another Next Generation data modeling language* (YANG) data models supported by the device.

NETCONF and YANG

NETCONF itself is a network management protocol, just as SNMP is a network management protocol. Such protocols become useful via the data models which convey information and requests to and from the device. With SNMP, the data models took the form of a *Management Information Base* (MIB), which we defined using *Structure of Management Information* (SMI). Contrasting the SNMP and NETCONF paradigms, SMI is analogous to YANG and the MIB is analogous to the YANG data model.

YANG provides a standardized way for devices to support and advertise their capabilities. One of the first operations that takes place between a NETCONF client on the controller and a NETCONF server running on the device is for the device to inform the client which data models are supported. This allows the SDN application running on the controller to know which operations are possible on each device. This granularity is key, since different devices will often vary in their capabilities. One of the current drawbacks of NETCONF and YANG is that different vendors often support different YANG models. This sometimes even occurs within different product families from the same vendor. Unlike the case of SNMP and standard MIBs (e.g., MIB-II, the Interfaces MIB, and the RMON MIB), there is currently no consistent set of YANG data models supported across the industry. It is currently necessary for applications to request and set data on different devices.

YANG models are still relatively new, and it is likely that standardized models will be defined in the near future. This will be facilitated by the fact that modern networking devices have better internal configuration schemas than in the early days of SNMP, so for most vendors it is relatively easy to auto-generate YANG data models that map onto those schemas. Note that in our discussions about NETCONF throughout this book we assume that YANG is used as its data modeling language.

NETCONF and RESTCONF

NETCONF uses the *Extensible Markup Language* (XML) to communicate with devices, which can be cumbersome. Fortunately, SDN controllers often support *REST-based NETCONF* (RESTCONF) as a mechanism for communicating between the controller and devices. RESTCONF works like a general REST API in that the application will use HTTP methods such as GET, POST, PUT, and DELETE in order to make NETCONF requests to the device.

As with normal REST communication, the URL specifies the specific resource that is being referenced in the request. The payload used for the request can be carried in either XML or *JavaScript Object Notation* (JSON).

Software developers with web programming backgrounds often find RESTCONF easier to work with than traditional use of NETCONF. This is due to the fact that REST APIs and JSON are generally more familiar to web developers than XML RPCs. Hence, using RESTCONF makes communication between the SDN controller and devices much simpler than it might be otherwise.

7.2.3 USING THE BGP PROTOCOL FOR SDN

Another protocol being promoted as a mechanism for SDN solutions is BGP. As we explained in Section 1.5.2, BGP is the EGP routing protocol used in the Internet. In addition to this traditional role, it is also used internally in some data centers. Consequently, the prospect of configuring BGP routes dynamically in a software defined manner is appealing. There are two major aspects of the BGP functionality currently used in ODL. These are:

- **IPv4 Topology**: The BGP plugin running inside ODL is implementing an actual BGP node, and as such it has access to topological information via the *Route Reflector* (RR). This information provides the topology between devices implementing the EGP, often referred to as the IPv4 topology. Fig. 7.5 shows an EGP network with routers supporting BGP, and an RR communicating topology information to the BGP node running inside the ODL controller. This information helps to provide the network-wide views characteristic of SDN solutions, and it can be used to dynamically configure intelligent routing paths throughout the network, via RIB configuration. This network-wide view is seen in Fig. 7.5 in the network topology to the right of the ODL controller. Note that while we specifically cite the IPv4 topology here, other topologies, such as the IPv6 topology, can be reported by the BGP plugin.

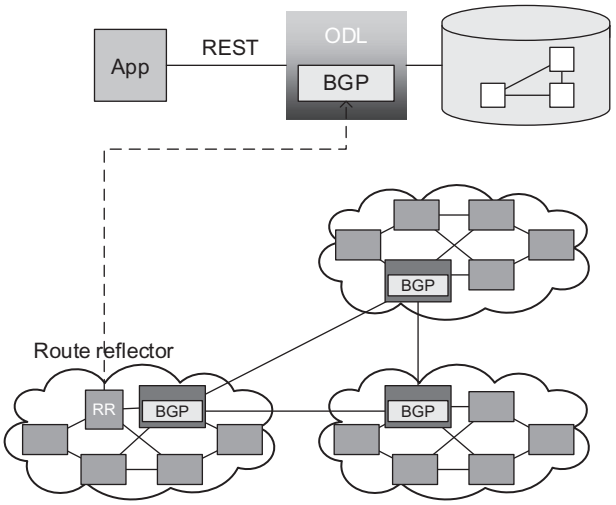

FIG. 7.5

SDN BGP topology.

- **RIB Configuration**: Within ODL there are APIs for creating a RIB application which can be used to inject routes into the network, based on the topology information, traffic statistics, congestion points to be avoided, as well as other possible relevant data.

A key aspect of this technique is that the ODL's controller's BGP plugin appears to the network as a normal BGP node. Note that the plugin does not advertise itself as a next hop for any of the routes. It will, however, effect changes on the adjacent nodes that will in turn propagate routing information throughout the network. In this way, an SDN application running on ODL can force RIB changes throughout the network, thereby achieving SDN-like agility and automatic network reconfiguration.

7.2.4 USING THE BGP-LS PROTOCOL FOR SDN

Figs. 7.6 and 7.7 depict the operation of the BGP-LS/PCE-P plugin on ODL.

- **BGP-LS** is used to pass link-state (OSPF or IS-IS) *Interior Gateway Protocol* (IGP) information about topology to ODL.
- **PCE-P** is used to transmit routing information from the PCE Server to the PCE clients in the network. A PCE client is also more simply known as a *Path Computation Client* (PCC).
- **MPLS** will be used to forward packets throughout the network, using the Label Switched Paths (LSPs) transmitted to head-end nodes via PCE-P.

Fig. 7.6 illustrates an IGP network of OSPF-supporting routers, sharing topology information with ODL. At a high level, BGP-LS is running on one of the OSPF (or another IGP) nodes in the network, and the IGP shares topology information with BGP-LS running on that node. That BGP-LS node in turn shares the topology information with the BGP-LS plugin running in ODL. That topology information is made available to the SDN application running on ODL, which can combine that knowledge with

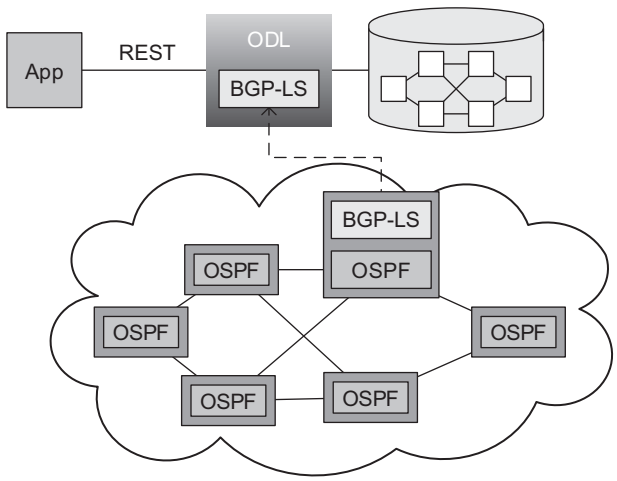

FIG. 7.6

SDN BGP-LS topology.

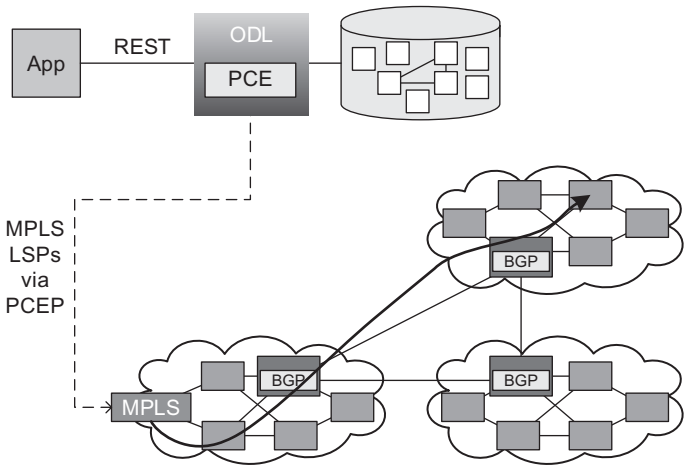

FIG. 7.7

SDN PCE-P and MPLS.

other knowledge about congestion, traffic, bandwidth, prioritization policies, and the like. This can be combined by the SDN application, which will determine optimal routing paths, and will communicate those MPLS paths to the PCCs in the network.

7.2.5 USING THE PCE-P PROTOCOL FOR SDN

PCE and its associated protocol PCE-P, have been in existence since roughly 2006 [4] and address the need to compute complex paths through IGP networks, as well as across *Autonomous System* (AS) boundaries via BGP. These paths are used in networks that support *MPLS Traffic Engineering* (MPLS-TE). The computation done by the PCE can be located in any compute node—in an MPLS head-end router, in the cloud, or on a dedicated server.

In Fig. 7.7, ODL (driven by an SDN application) sets MPLS *Label Switched Paths* (LSPs) using PCE-P. Communication is between the PCE server in ODL and the PCC on the MPLS router.

The PCC runs on the *head-end* of each LSP. Using these LSPs, the router is able to route traffic using MPLS through the network in an optimal manner. Using PCE-P in this fashion has advantages over a pre-SDN counterpart called *Constrained Shortest Path First* (CSPF). Like our PCE-P model described above, CSPF computes the LSPs but is limited to the topology of the IGP domains to which it belongs. Conversely, PCE-P can run across multiple IGP domains. Another advantage of PCE-P is that it can perform global optimization contrary to the CSPF model where each head-end router performs local optimization only.

7.2.6 USING THE MPLS PROTOCOL FOR SDN

MPLS will be used to forward packets throughout the network using the LSPs transmitted to head-end nodes via PCE-P. In the SDN solution described in the previous section the role of MPLS is to forward traffic according to the paths configured by the SDN application running on ODL. Configuration takes

place on the MPLS head-end router shown in Fig. 7.7. This router will receive the matching packets at the edge of the MPLS network, and packets will then be forwarded on the LSP that has been configured by PCE-P.

This solution does not require a new protocol such as OpenFlow in order to control the network's forwarding behavior, but rather uses these existing protocols (i.e., BGP, BGP-LS, PCE-P, and MPLS) to achieve intelligent routing of packets based on paths that have been configured by the centralized controller. This emerging solution holds promise for customers looking to utilize their existing infrastructure in a new, SDN-based manner.

DISCUSSION QUESTION:

Both the NETCONF and the BGP-LS/PCE-P southbound protocol plugins attempt to provide SDN capabilities using existing protocols. Which of these two seems to be more "SDN" and why? And what are some potential dangers in using BGP to set RIBs in the SDN controller?

7.3 ADDITIONAL SDN CONTROLLER MODELS

In this section we turn our attention to SDN controllers and controller technologies that have gained recent prominence in SDN. Hot topics in recent SDN controller innovation have included southbound protocol plugins, internal architectures, service provider solutions, scalability, and northbound interfaces to SDN applications.

As the dust begins to settle on SDN, two commercially viable controllers stand out: ODL and *Open Network Operating System* (ONOS). In the discussion that follows we are interested in the technologies used by these controllers. There are also business ramifications of the dominance that these controllers have asserted, but we defer our treatment of those until Chapter 14. Table 7.2 lists the primary areas of emphasis of each of these two controllers.

We caution the reader that the emphases denoted in Table 7.2 are just that—*emphases*. One should not infer that the other controller ignores these issues. For example, ONOS has projects for alternative southbound protocols like NETCONF and PCE-P, and ODL has projects and functionality related to service providers, scalability, and *intents*. We will see in the following sections that both controllers implement functionality in all of these areas.

7.3.1 CONTROLLERS WITH MULTIPLE SOUTHBOUND PLUGINS

Prior to the advent of ODL, almost every general-purpose SDN controller used the OpenFlow protocol as the sole southbound protocol. Other controllers were either not targeted at the open application development community or did not garner a sizeable community of developers. To gain traction,

Table 7.2 ODL and ONOS

Controller	Organization	Emphases
ODL	Linux Foundation	Multiple southbound, MD-SAL
ONOS	ON.Lab	Service providers, scalability, *intents*

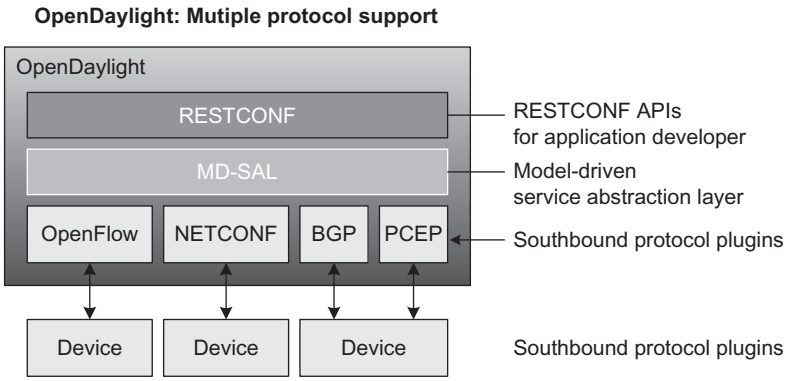

FIG. 7.8

ODL southbound protocol plugins.

an open controller effort requires a cadre of developers, some willing to write applications for that controller and others contributing to the controller's code base itself. OpenDaylight has emerged at the forefront of open SDN controller solutions.

OpenDaylight has enjoyed this success due to the support of the Linux Foundation and Cisco. In fact, the core of the ODL controller initially was Cisco's own XNC product. Another catalyst of its meteoric rise is the fact that ODL proposed and delivered on the promise of providing an SDN controller that supports not only OpenFlow, but also other suitable southbound protocol plugins. Consequently protocols such as NETCONF, SNMP, BGP, and PCE-P have gained traction as potential southbound protocols for SDN using the ODL controller.

Fig. 7.8 provides a high-level architectural view of ODL and its support for multiple southbound plugins. This affords customers an SDN controller that works with their legacy devices. The limitations of the protocol corresponding to the plugin are still present, but the architecture in Fig. 7.8 benefits from the SDN model of making networks software-centric and centrally programmed.

7.3.2 CONTROLLERS WITH MODEL-DRIVEN INTERNAL DESIGN

For some time, controllers and network management platforms have utilized *model-based* technologies for communicating with devices, particularly since object-oriented programming became prevalent in the industry. ODL, with significant implementation commencing in the Helium release and continuing with the Lithium release, builds upon this theme by making all internal ODL applications (i.e., modules) adhere to their *Model-Driven Service Abstraction Layer* (MD-SAL) architecture. The goals of MD-SAL are as follows:

- **Abstraction**: Earlier versions of ODL resulted in every southbound protocol having its own protocol-specific set of APIs. This clearly is not desirable for application developers, having to develop against varying APIs for multiple different protocols. MD-SAL permits communication to devices only through models, thus yielding an abstraction that provides protocol-independent APIs for application developers to use.

- **Standardized Communication**: When an MD-SAL module is created, APIs for the module are automatically generated by tools included within the MD-SAL environment. Thus, these APIs (both RESTful and internal) are more standardized than would otherwise be true if they were created ad hoc by human developers. Significantly, these APIs are generated *automatically* by the build framework, and thus require no extra design or implementation effort by the API developer. This facilitates easier intermodule and interapplication communication.
- **Microservices**: Microservices are a software architecture style in which complex applications are composed of small, independent processes communicating with each other using standardized APIs. MD-SAL facilitates such an environment in ODL through the use of YANG models to define every service. Hence every MD-SAL application is a service with its own auto-generated APIs.

7.3.3 CONTROLLERS TARGETING SERVICE PROVIDER SOLUTIONS

Since the early days of SDN, service providers have played a major role in defining standards and helping to make the new technology suitable for the largest of networks. Companies such as AT&T, Deutsche Telekom, and NTT were all key players in the *Open Networking Foundation*, (ONF) as well as delivering featured keynotes at various SDN conferences. The two dominant emerging controller technologies, ODL and ONOS have both targeted the service provider market and have developed their products accordingly.

ODL and service providers

The first release of ODL, Hydrogen, had a specific distribution of the controller geared toward service providers. Since that time, vendors such as Brocade and Cisco have openly courted service providers with their branded versions of ODL, the Brocade SDN Controller and the Cisco Open SDN Controller, respectively.

Although service providers have participated in the definition of OpenFlow, they also have massive networks of existing equipment upon which vast amounts of traffic and numerous organizations depend. Converting this established environment to OpenFlow, no matter how promising the potential benefits, is not a strategy these service providers will embrace overnight. Rather, most of these large organizations will pursue a more evolutionary approach, utilizing NETCONF configuring existing devices. Consider that Deutsche Telekom announced in 2013 [5] that it was selecting Tail-f as its SDN technology of choice (Tail-f provides solutions primarily involving NETCONF). Likewise, AT&T's Domain 2.0 project [6] uses NETCONF with ODL in order to reach their stated goal of being 75% SDN-based by 2020 [7].

Service providers also have very large deployments of IPv4 and IPv6 BGP. One way ODL supports those service providers is through the implementation of *BGP Monitoring Protocol* (BMP). Through this mechanism ODL can learn about best performing routes in the network, which can be useful for determining optimal paths and balancing traffic across the network.

ONOS and service providers

The ONOS controller primarily uses OpenFlow, but is built specifically for service providers. This service provider orientation is not related to the protocol used, but rather the controller's architecture. The ONOS controller is designed natively with a distributed core, meaning that ONOS scales to support

massive numbers of network devices [8]. We look more closely at scalability and the architectures of both ODL and ONOS in the next section.

7.3.4 CONTROLLERS BUILT FOR SCALABILITY

For service providers as well as for the majority of networks today, scalability is of paramount importance, especially as the demands on networking and bandwidth continue to grow. This is self-evident for service providers and data centers. Even enterprise networks, with their growing need for speed and dependence on the Internet, require a level of performance and scalability not seen before. Hence, any SDN solution will need to cope with rapidly expanding workloads. Networking environments today rely on three interrelated attributes, *Scalability*, *Performance*, and *High-Availability* (HA):

- **Scalability**: Technologies such as the cloud and streaming video, combined with the advent of the *Internet of Things* (IoT) demand that SDN controllers scale to handle the growth in traffic as well as the exploding number of network devices and hosts.
- **Performance**: This increased load requires commensurate improvement in the performance, speed, and efficiency of network devices and from the controllers that manage them. Therefore network solutions must be able to adapt dynamically to the frequent changes in network traffic loads and congestion. Overloaded or sluggish controllers will only exacerbate the issues faced by the network in such situations.
- **HA**: The critical role played by SDN controllers creates increasing dependence on the uninterrupted availability of the SDN controller. While certain SDN solutions place higher demands for HA, virtually every SDN solution requires some amount of resiliency. Whether it is achieved via clustering, teaming, or some other form of active-active or active-passive strategy, the imperative is that production SDN solutions must provide constant controller availability.

Different controllers employ varied strategies for scalability, performance, and HA. The specific methodologies are in flux as limitations are confronted and new technologies evolve. In the balance of this section, we look at the strategies employed by ODL and ONOS. For more detail, we refer the interested reader to [9] for ODL and [10] for ONOS.

ODL and scalability, performance, and high-availability

The very first release of ODL, Hydrogen, was more of a proof-of-concept and, hence, did not feature robust scalability, performance, and HA. With Helium, and later with the Lithium release, ODL adopted an Akka-based [11] solution for HA. Akka was developed originally for Erlang [12] and was adopted for use in Scala and Java environments. It is an open-source toolkit with the goal of simplifying the creation of applications that run in a distributed environment. Much of Akka involves improvements on concurrency, an area whose challenges are normally exacerbated when running on multiple systems.

Some of the tenets of Akka are: (a) concurrency is message-based and asynchronous; (b) components interact with each other in the same manner, regardless of whether they are running on the same or different hosts; and (c) components adhere to a hierarchical model, allowing supervision of child nodes to help prevent or resolve failures.

Another feature of ODL's controller clustering solution involves the use of *shards* [13], which are slices of functionality that are deployed in a distributed manner in order to balance the load across the systems in a cluster. As of this writing, shards are organized based on components, and so

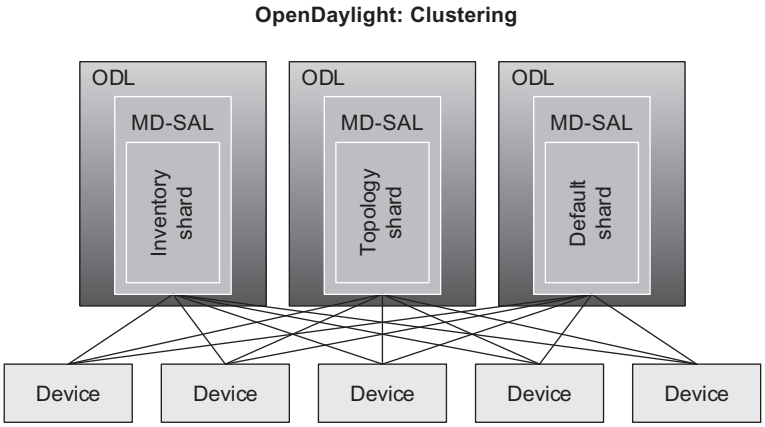

FIG. 7.9

ODL clustering.

components of the same functionality are typically co-located for performance, which is advantageous in the nonfailed state, but which requires some recovery time if a host in the cluster goes down. It is also possible to design your cluster such that shards are split across the nodes in the cluster, such that the functionality is distributed, with a designated leader, thus resulting in faster recovery times, but potentially slower performance.

Fig. 7.9 illustrates an example of ODL clustering, with shards implementing inventory and topology, and a default shard for other running services. The different components may interact with any of the controlled devices. ODL's clustering strategy has expanded over the course of successive ODL releases as demands from service providers and others help to ensure that their needs regarding performance, scalability, and high-availability are met.

ONOS and scalability, performance, and high-availability

The ONOS controller was conceived and built for scalability and performance. Consequently, its initial design was distributed, allowing it to control massively scaled environments. The controller is built around the distributed core shown in Fig. 7.10. According to the diagram, ONOS distributes its *adapter* (protocol) functionality onto all instances of the controller, but has a single distributed core functionality shared across all hosts running ONOS. Fig. 7.10 also depicts a northbound API which is based on intents, which is described in the next section.

DISCUSSION QUESTION:

Compare and contrast OpenDaylight and ONOS. Which do you believe is more "SDN"? Which do you believe has the greatest chance of being successful as the dominant controller solution?

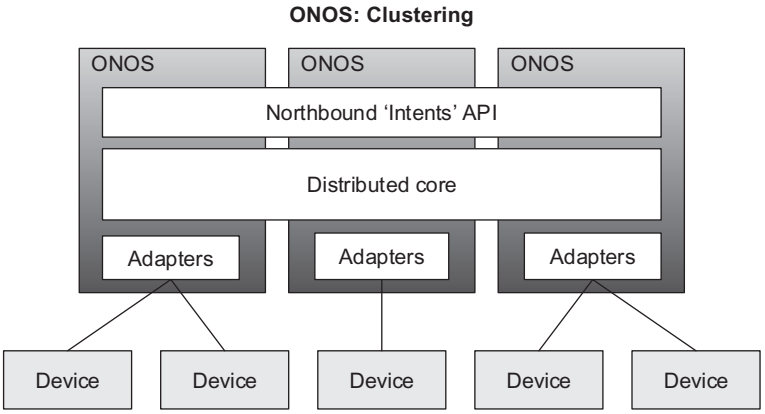

FIG. 7.10

ONOS clustering.

7.3.5 CONTROLLERS BUILT FOR INTENTS-BASED APPLICATIONS

In the early days of SDN, controllers were built with the sole purpose of creating a Java or REST API to allow SDN applications to program the OpenFlow tables resident in networking devices. This led to issues related to the fact that each SDN controller created its own API for setting OpenFlow rules. Each SDN application had to be developed as a function of the controller vendor.

This led to an attempt at standardization by the ONF. The ONF started a *Northbound Interface* (NBI) working group tasked with addressing this issue. Rather than define yet another northbound API, this working group defined an API at a level of abstraction above the base controller APIs, providing interface capabilities based on intents rather than on raw programming of OpenFlow tables.

As an example, an application wishing to provide some level of connectivity between two hosts A and B, using existing base controller APIs would program flows on switches all across the topology, from the switch connecting A to the network, to the switch connecting B to the network, including all switches along the path between the hosts. This clearly increases the complexity of the SDN application.

On the other hand, an intents-based API would allow the application to specify something more abstract, such as *the desire to allow host A to communicate with host B*. Invoking that API would result in the SDN controller performing flow programming of the switches on the path from host A to host B, assuming that OpenFlow was the SDN protocol in use. If some other type of protocol was being used, this would not perturb the application, since the controller would be responsible for making that connectivity happen with whatever southbound API was appropriate. Consider three features of this concept:

- **Abstraction**: The goal of an SDN controller, as with operating systems in general, is to *abstract* the details of the hardware below from the application running above.
- **Declarative**: Specifying *what* to do, rather than *how* to do it, is a characteristic of *declarative* systems.
- **Protocol-agnostic**: An abstract declarative interface hides details of how the network programming occurs, allowing different protocols to be used in different situations.

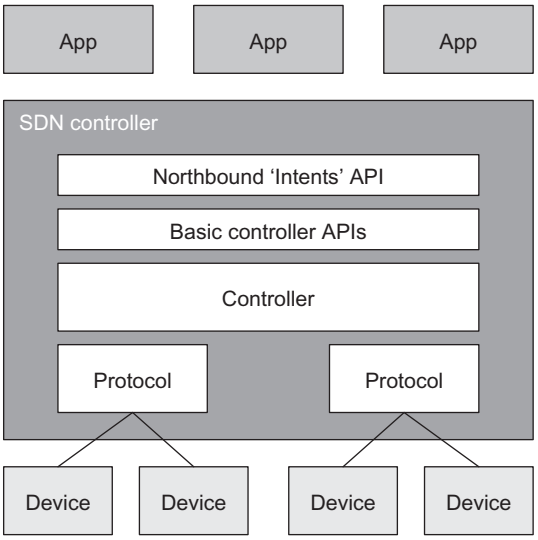

FIG. 7.11

Northbound intents-based interface.

Fig. 7.11 shows a controller with an intents-based API superimposed between the applications and the controller APIs. In [14] the reader will find a description of a functioning example of this concept. The intents-based API described in [14] actually runs on both ODL and ONOS. This is a step toward creating an API which frees SDN applications from having to know and understand the details of the devices in the physical network.

DISCUSSION QUESTION:

We discussed a number of important emerging attributes of SDN controllers. Comparing them to each other, which do you feel is the most important for SDN going forward? Which of them might be less important than the others?

7.4 ADDITIONAL APPLICATION MODELS

In addition to the introduction of new protocols for SDN, and evolving controller trends for SDN, there have also been shifts regarding SDN application models. We look at these in the next sections.

7.4.1 PROACTIVE APPLICATION FOCUS

Two fundamental classes of applications have emerged since the inception of SDN. Fig. 7.12 shows the two general designs of SDN applications, *proactive* and *reactive*. The figure reveals a number of differences, but for this discussion the most important difference between the two is that reactive

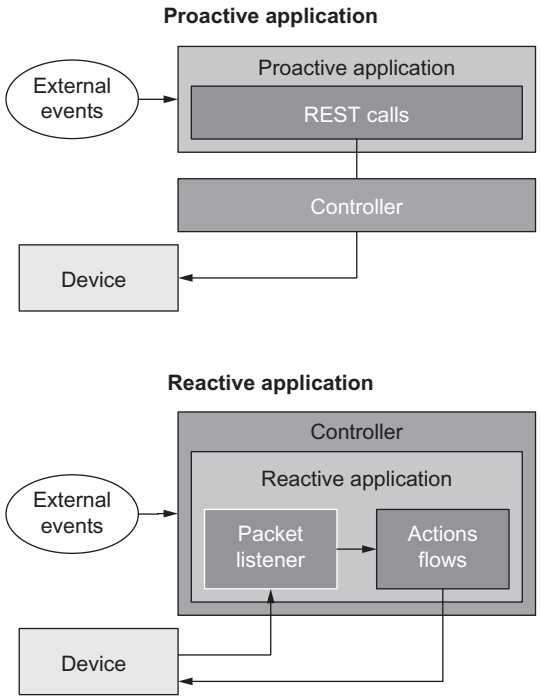

FIG. 7.12

SDN application types.

applications sometimes receive packets forwarded by the networking device. This is shown in the figure by the arrow that goes from the device to the *Packet Listener* component in the Reactive Application. We elaborate on the differences between these two applications classes in Chapter 12.

The idea that forwarding rules might not be programmed until related packets arrived was one of the more disruptive aspects of early SDN research. This concept fostered a wave of new innovation. However, reactive applications also engender concerns about scalability, reliability, and susceptibility to *Denial of Service* (DoS) attacks.

For these reasons, current emphasis on production applications focuses on proactive application designs. We defer detailed discussion of both classes of applications to Chapter 12. For this discussion, the main differentiators of a proactive application are: (1) setting of forwarding behavior a priori, rather than waiting for packets to be forwarded to the controller for processing, (2) some dynamism is achieved by responding to external events, but far less frequently than reactive applications, and (3) the ability to write applications that reside externally from the controller, which can be written in any programming language, since they often utilize the REST APIs of the controller.

The advantages of proactive applications involve simplicity, programming language flexibility, deployment flexibility, and security. We will delve more deeply into these advantages in Chapter 12.

7.4.2 DECLARATIVE APPLICATION FOCUS

As we discussed in Section 7.3.5, early SDN controller designs provided low-level APIs. Such a system, where the application developer dictates to each network device exactly how it was supposed to behave, is called an *imperative* system. In current programming and in systems development, the trend is toward *declarative* APIs and solutions. We distinguish imperative from declarative systems as follows:

- An **imperative** system requires that it be told exactly *how* to do something. The net result of all the *hows* is that the system will achieve the *what* that was intended in the first place.
- A **declarative** system needs only be told *what* needs to be done; the system itself figures out *how* to accomplish the goal.

An intents-based API clearly is more declarative in nature than the base APIs offered by controllers in the past. An internal architecture such as that provided by MD-SAL establishes the building blocks for creating declarative solutions. In fact, Brocade's SDN solution [15] has produced such an interface using MD-SAL, called the *Flow Manager* [16]. This MD-SAL application builds an intents-based API on top of ODL, yielding a more declarative API for application developers.

In addition to ODL and ONOS, other commercial SDN solutions such as Cisco's *Application Policy Infrastructure Controller—Data Center* (APIC-DC) and *Application Policy Infrastructure Controller—Enterprise Module* (APIC-EM), provide policy-level APIs for customers who want to write applications for those controllers. These APIs are oriented toward allowing developers to specify the *what*, allowing the system to decide *how* to make that happen.

7.4.3 EXTERNAL APPLICATION FOCUS

There is also an increased focus on creating external rather than internal SDN applications. The two types of applications are shown in Fig. 7.13. In most controllers, internal applications run inside the *Open Services Gateway initiative* (OSGi) [17] Java container in which the controller is running. Applications running internally must adhere to the standards defined by the internal controller environment. In general, there are more constraints when creating an internal application. External applications, however, have much more freedom:

- External SDN applications can be written in many different programming languages including Python, Ruby, PERL, and even scripting languages.
- External SDN applications can run on the same system as the controller, on a different but geographically local system, on a remote system, or even in the cloud.
- External SDN applications generally have a reduced impact on the operation of the controller (e.g., internal applications can cause failures or bottlenecks on the controller).

This relative ease of use and increased safety have been factors accelerating the migration toward external applications in SDN.

One drawback of external applications is the fact that REST APIs by themselves do not provide a mechanism for asynchronous notification back to the calling application. While this is true, some controller implementations do provide some mechanisms to achieve this goal. This is often done using some socket-based functionality, wherein the application opens a socket and listens for notifications, after *registering* itself with the controller. In this manner, the controller issues notifications to registered

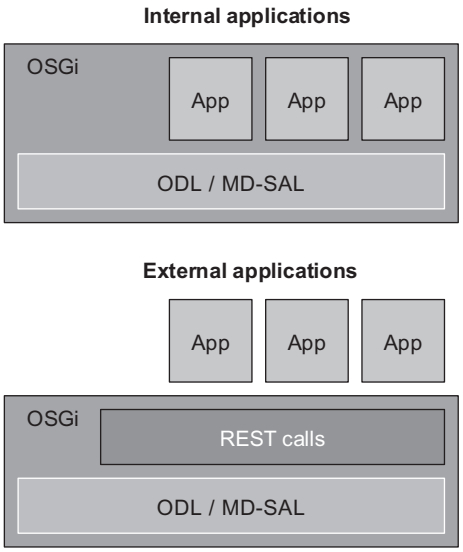

FIG. 7.13

External vs. internal SDN applications.

listeners for subscribed events (e.g., topology change notifications). These notifications are not as fast as would be the case with an internal application, so care must be taken regarding the design of the application, such that this does not become a bottleneck.

DISCUSSION QUESTION:

We have looked at a number of trends related to SDN application development. Which application type, proactive or reactive, will be the most important in the long term regarding SDN? Which in the short term? Do you believe there is a place for internal SDN applications in the future of SDN?

7.5 NEW APPROACHES TO SDN SECURITY

This chapter has described trends toward using existing protocols and using proactive external applications in SDN. These trends have ramifications on SDN security as well. Fortunately, these approaches have facilitated the task of securing an SDN environment. In the following sections we explore the impact of these trends on SDN security.

7.5.1 SECURITY ASPECTS OF REACTIVE APPLICATIONS

The model of reactive applications presumes that certain packets received by the OpenFlow device will be forwarded to the controller, exposing a number of security vulnerabilities. One such vulnerability

is the possibility of a DoS attack. In this type of attack, a rogue device connects to the controller and forwards massive numbers of packets to the controller, overwhelming it and its applications, thus bringing down the network. Admittedly, there are safeguards in place with OpenFlow to make it difficult for a rogue device to become trusted by the controller and, hence, some level of security in this regard is maintained; but the threat still exists. With the rise of proactive applications, the risk of this type of threat is diminished, as devices do not forward any user traffic to the controller. Even if this were to occur, in the proactive paradigm the SDN controller and applications will ignore and discard such packets.

7.5.2 SECURITY FOR NETWORK MANAGEMENT APPLICATIONS

We look now at the security aspects of implementing SDN using existing network management protocols, such as NETCONF. An SDN application making use of these protocols is essentially performing an enhanced style of network management. Such network management protocols and applications have already weathered many years of production use and have already been hardened to remove security vulnerabilities. Thus, SDN applications based on network management are already using existing, secure technologies. Since they are at the core of a network management application, they intervene with the device far less then a reactive, intensely dynamic SDN application. Consequently, networking configurations and policy changes are made far less frequently and are not as deep as those of an OpenFlow-based reactive application. We can illuminate what we mean by deep by looking back at the control points shown in Fig. 7.3. In that diagram we see OpenFlow applications set matches and actions directly at the lower level of devices, labeled FIB in the diagram. Conversely, network management-style SDN applications access the configuration layer of the device. That configuration layer is less *deep* in the sense that it only exposes configuration control points that ensure that the device cannot be compromised, rendering the device more secure.

7.5.3 SECURITY BENEFITS OF EXTERNAL APPLICATIONS

One final aspect of security is related to the trend toward developing external SDN applications. An application running externally has limited access to the internals of the SDN controller as it can only interact with the controller via RESTful APIs. Internal applications, on the other hand, are normally operating within the Java runtime environment (OSGi) and, hence, have access to the internals of the controller. While there may be safeguards in place to prevent the loading of malicious applications onto SDN controllers, should those safeguards be compromised, the danger for the SDN controller is very real.

To a certain extent, maintaining constant availability and high performance of the controller are also part of keeping an SDN system secure. As described in Section 7.4.3, an external application can fail or crash with limited effect on the SDN controller. However, an internal application that crashes can cause side effects which negatively impact the operation of the controller. If the internal application fails while it holds important resources, like shared threads or data locks, the impact on the rest of the controller can be significant, if not catastrophic. Similarly, a poorly performing external application will have limited effect on the controller, whereas a poorly performing internal application can have dreadful performance consequences, especially when using shared resources such as threads and data.

Since external applications are less likely to introduce the issues described here, their use contributes to a more secure and reliable SDN environment.

Before we leave this topic of security, we should point out to the reader that there are more aspects of SDN security than we can reasonably cover in this book. Two examples would be the issues of *integrity* and *confidentiality* between SDN components. We refer the interested reader to [18] for more information on these topics.

DISCUSSION QUESTION:

From our discussion and your own experience, do you believe that security is a major threat to the success of SDN? Do you believe that the emerging trends presented in this chapter are addressing those threats? Are there other threats that were not mentioned in the text?

7.6 THE P4 PROGRAMMING LANGUAGE

A novel SDN idea that is beginning to gain traction is a programming language specifically designed to tell networking devices how to handle incoming packets. P4 [19] is an example of such a purpose-built language. Some of the basic attributes of P4 are the following:

- **Language**: The language itself is declarative, and is syntactically similar to C.
- **Matches**: Matching tables define against what the incoming packets will be compared.
- **Actions**: Action tables define what should be done to the packet after a match has occurred.
- **Compile-time vs. Run-time**: The language is intended to be compiled into run-time format for efficient execution.

Today, the matches supported by P4 are those common for policy-based routing (e.g., ACLs, forwarding). This entails matching on the header fields specified by OpenFlow and other configuration protocols, and ultimately implemented on the device in TCAMs. The actions are also familiar from OpenFlow, such as *forwarding out a specific set of ports* or *modifying fields*. P4 represents an intriguing new frontier for SDN and it will be interesting to observe its impact in the coming years.

There are other academic research efforts aiming to provide high-level languages and abstractions for programming software defined networks. These include the work done in [20] using the *Pyretic* language and that using the *Procera* language in [21].

7.7 CONCLUSION

The SDN world is a fast-changing place. In this chapter, we have looked at trends toward using existing protocols for SDN applications, in particular as an alternative to OpenFlow. Our discussion presented two controllers that now dominate most of the SDN dialogue. We explained the growing emphasis on proactive and external applications. We now forge into the phalanx of data center racks where the need for SDN first emerged.

REFERENCES

[1] BGP LS PCEP:Main. OpenDaylight Wiki. Retrieved from: https://wiki.opendaylight.org/view/BGP_LS_PCEP:Main.
[2] NETCONF Configuration Protocol. IETF Proposed Standard, December 2006.
[3] Network Configuration Protocol (NETCONF). IETF Proposed Standard, June 2011.
[4] Path Computation Element (PCE) Communication Protocol (PCEP). IETF Proposed Standard, March 2009.
[5] Deutsche Telekom selects Tail-f as provider of SDN. Tail-f press release. Retrieved from: http://www.tail-f.com/deutsche-telekom-selects-tail-f-as-provider-of-software-defined-networking-sdn-in-terastream-project/.
[6] AT&T Domain 2.0 Vision White Paper. November 13, 2013. Retrieved from: https://www.att.com/Common/about_us/pdf/AT&TDomain 2.0 Vision White Paper.pdf.
[7] AT&T to virtualize 75% of its network by 2020. Wall Street Journal, December 16, 2014. Retrieved from: http://blogs.wsj.com/cio/2014/12/16/att-to-virtualize-75-of-its-network-by-2020/.
[8] Overview of ONOS architecture. ONOS Wiki. Retrieved from: https://wiki.onosproject.org/display/ONOS/Overview+of+ONOS+architecture.
[9] OpenDaylight Wiki. Retrieved from: https://wiki.opendaylight.org.
[10] ONOS Wiki. Retrieved from: https://wiki.onosproject.org/display/ONOS/Wiki+Home.
[11] akka. Retrieved from: http://akka.io.
[12] Erlang. Retrieved from: http://erlang.org.
[13] OpenDaylight Controller:MD-SAL:Architecture:Clusteringi. Retrieved from: https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Architecture:Clustering.
[14] Intent framework. ONOS Architecture Guide. Retrieved from: https://wiki.onosproject.org/display/ONOS/Intent+Framework.
[15] Brocade SDN controller. Retrieved from: http://www.brocade.com/en/products-services/software-networking/sdn-controllers-a
[16] Brocade flow manager. Retrieved from: http://www.brocade.com/content/brocade/en/products-services/software-networking/sdn-controllers-applications/flow-manager.html.
[17] OSGi: the dynamic module system for Java. Retrieved from: https://www.osgi.org.
[18] Silva AS, Smith P, Mauthe A, Schaeffer-Filho A. Resilience support in software-defined networking: a survey. Comput Netw 2015;92(1):189–207.
[19] Matsumoto C. P4 SDN language aims to take SDN beyond OpenFlow. SDXCentral, May 30, 2015. Retrieved from: https://www.sdxcentral.com/articles/news/p4-language-aims-to-take-sdn-beyond-openflow/2015/05/.
[20] Monsanto C, Reich J, Foster N, Rexford J, Walker D. Composing software-defined networks. In: Proceedings of the 10th USENIX conference on Networked systems design and implementation (NSDI'13). Berkeley, CA, USA: SENIX Association; 2013.
[21] Voellmy A, Kim H, Feamster N. Procera: a language for high-level reactive network control. In: Proceedings of the first workshop on Hot topics in software defined networks (HotSDN '12). New York, NY, USA: ACM; 2012.