- **Integrated security:** Network applications must integrate seamless security as a core service instead of an add-on solution.
- **On-demand scaling:** Implementations must have the ability to scale up or scale down the network and its services to support on-demand requests.

SDN Architecture

An analogy can be drawn between the way in which computing evolved from closed, vertically integrated, proprietary systems into an open approach to computing and the evolution coming with SDN (see Figure 3.1). In the early decades of computing, vendors such as IBM and DEC provided a fully integrated product, with a proprietary processor hardware, unique assembly language, unique operating system (OS), and the bulk if not all of the application software. In this environment, customers, especially large customers, tended to be locked in to one vendor, dependent primarily on the applications offered by that vendor. Migration to another vendor's hardware platform resulted in major upheaval at the application level.

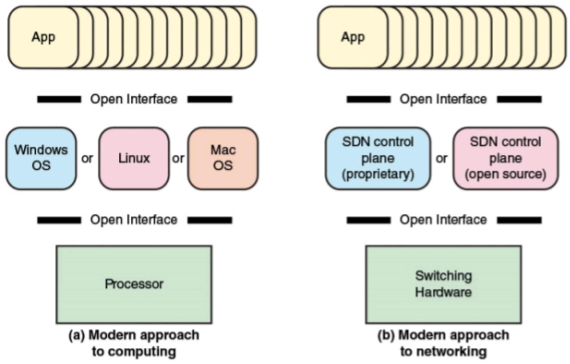


FIGURE 3.1 The Modern Approach to Computing and Networking

Today, the computing environment is characterized by extreme openness and great customer flexibility. The bulk of computing hardware consists of x86 and x86-compatible processors for standalone systems and ARM processors for embedded systems. This makes it easy to port operating systems implemented in C, C++, Java, and the like. Even proprietary hardware architectures, such as IBM's zEnterprise line, provide standardized compilers and programming environments and so can easily run open sources operating systems such as Linux. Therefore, applications written for Linux or other open

operating systems can easily be moved from one vendor platform to another. Even proprietary systems such as Windows and Mac OS provide programming environments to make porting of applications an easy matter. It also enables the development of virtual machines that can be moved from one server to another across hardware platforms and operating systems.

The networking environment today faces some of the same limitations faced in the pre-open era of computing. Here the issue is not developing applications that can run on multiple platforms. Rather, the difficulty is the lack of integration between applications and network infrastructure. As demonstrated in the preceding section, traditional network architectures are inadequate to meet the demands of the growing volume and variety of traffic.

The central concept behind SDN is to enable developers and network managers to have the same type of control over network equipment that they have had over x86 servers. As discussed in Section 2.6 in Chapter 2, the SDN approach splits the switching function between a data plane and a control plane that are on separate devices (see Figure 3.2). The data plane is simply responsible for forwarding packets, whereas the control plane provides the "intelligence" in designing routes, setting priority and routing policy parameters to meet QoS and QoE requirements and to cope with the shifting traffic patterns. Open interfaces are defined so that the switching hardware presents a uniform interface regardless of the details of internal implementation. Similarly, open interfaces are defined to enable networking applications to communicate with the SDN controllers.

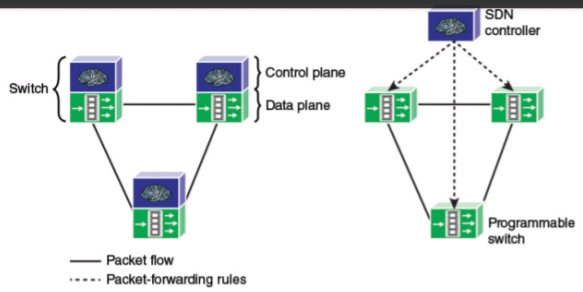


FIGURE 3.2 Control and Data Planes

← *See Figure 2.15, Software Defined Networking*

Figure 3.3 elaborates on the structure shown in Figure 2.15, showing more detail of the SDN approach. The data plane consists of physical switches and virtual switches. In both cases, the switches are responsible for forwarding packets. The internal implementation of buffers, priority parameters, and other data structures related to forwarding can be vendor dependent. However, each switch must implement a model, or abstraction, of packet forwarding that is uniform and open to the SDN controllers. This model is defined in terms of an open application programming interface (API) between the control plane and the data plane (southbound API). The most prominent example of such an open API is OpenFlow, discussed in Chapter 4, "SDN Data Plane and OpenFlow." As Chapter 4 explains, the OpenFlow specification defines both a protocol between the control and data planes and an API by which the control plane can invoke the OpenFlow protocol.

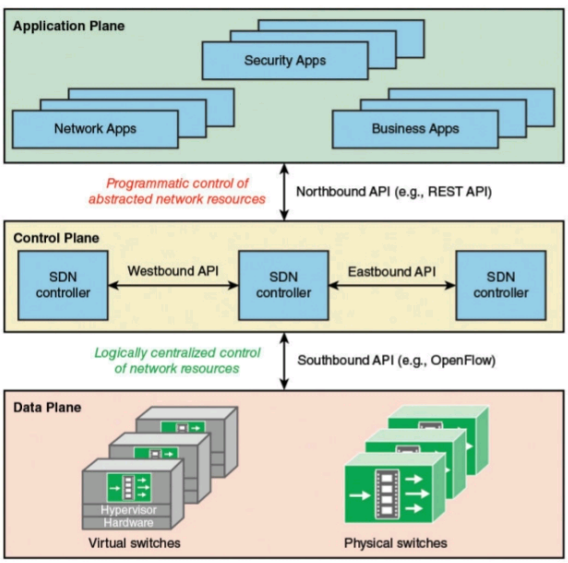


FIGURE 3.3 Software-Defined Architecture

A language and message format used by an application program to communicate with the operating system or some other control program such as a database management system (DBMS) or communications protocol. APIs are implemented by writing function calls in the program, which provide the linkage to the required subroutine for execution. An open or standardized API can ensure the portability of the application code and the vendor independence of the called service.

FIGURE 3.3 Software-Defined Architecture

A language and message format used by an application program to communicate with the operating system or some other control program such as a database management system (DBMS) or communications protocol. APIs are implemented by writing function calls in the program, which provide the linkage to the required subroutine for execution. An open or standardized API can ensure the portability of the application code and the vendor independence of the called service.

SDN controllers can be implemented directly on a server or on a virtual server. OpenFlow or some other open API is used to control the s
the data plane. In addition, controllers use information about ca
demand obtained from the networking equipment through which
flows. SDN controllers also expose northbound APIs, which allow developers and network managers to deploy a wide range of off-the-shelf and custom-built network applications, many of which were not feasible before the advent of SDN. As yet there is no standardized northbound API nor a consensus on an open northbound API. A number of vendors offer a REpresentational State Transfer (REST)-based API to provide a programmable interface to their SDN controller.

143 / 800

➔ *See Chapter 5, "SDN Control Plane"*

Also envisioned but not yet defined are horizontal APIs (east/westbound), which would enable communication and cooperation among groups or federations of controllers to synchronize state for high availability.

At the application plane are a variety of applications that interact with SDN controllers. SDN applications are programs that may use an abstract view of the network for their decision-making goals. These applications convey their network requirements and desired network behavior to the SDN controller via a northbound API. Examples of applications are energy-efficient networking, security monitoring, access control, and network management.

Characteristics of Software-Defined Networking

Putting it all together, the key characteristics of SDN are as follows:

- The control plane is separated from the data plane. Data plane devices become simple packet-forwarding devices (refer back to Figure 3.2).
- The control plane is implemented in a centralized controller or set of coordinated centralized controllers. The SDN controller has a centralized view of the network or networks under its control. The controller is portable software that can run on commodity servers and is capable of programming the forwarding devices based on a centralized view of the network.
- Open interfaces are defined between the devices in the control plane (controllers) and those in the data plane.
- The network is programmable by applications running on top of the SDN controllers. The SDN controllers present an abstract view of network resources to the applications.

3.3 SDN- and NFV-Related Standards

Unlike some technology areas, such as Wi-Fi, there is no single standards body responsible for developing open standards for SDN and NFV. Rather, there is a large and evolving collection of standards-developing organizations (SDOs), industrial consortia, and open development initiatives involved in creating standards and guidelines for SDN and NFV. Table 3.1 lists the main SDOs and other organizations involved in the effort and the main outcomes so far produced. This section covers some of the most prominent efforts.

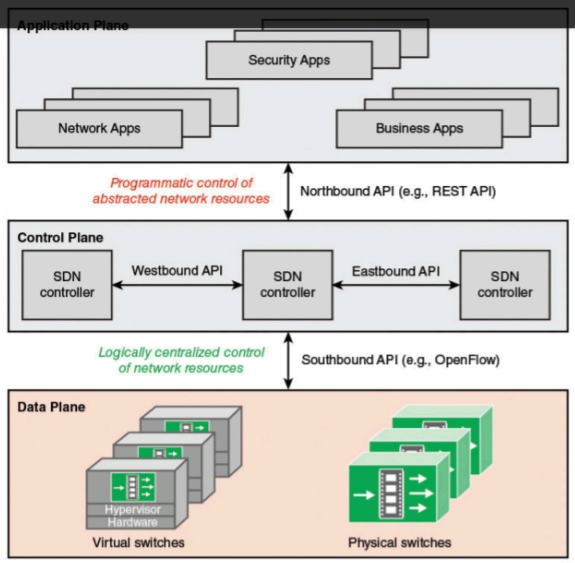


FIGURE 4.1 SDN Architecture

4.1 SDN Data Plane

The SDN data plane, referred to as the resource layer in ITU-T Y.3300 and also often referred to as the infrastructure layer, is where network forwarding devices perform the transport and processing of data according to decisions made by the SDN control plane. The important characteristic of the network devices in an SDN network is that these devices perform a simple forwarding function, without embedded software to make autonomous decisions.

Data Plane Functions

Figure 4.2 illustrates the functions performed by the data plane network devices (also called data plane network elements or switches). The principal functions of the network device are the following:

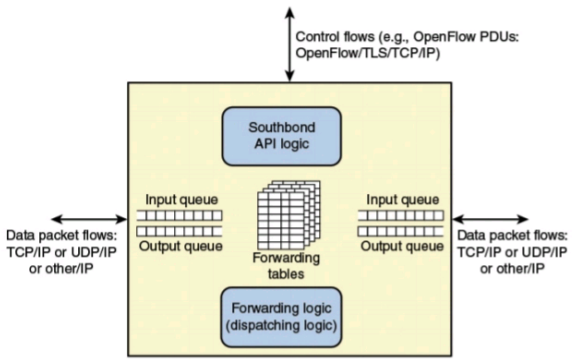


FIGURE 4.2 Data Plane Network Device

- **Control support function:** Interacts with the SDN control layer to support programmability via resource-control interfaces. The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.
- **Data forwarding function:** Accepts incoming data flows from other network devices and end systems and forwards them along the data forwarding paths that have been computed and established according to the rules defined by the SDN applications.

These forwarding rules used by the network device are embodied in forwarding tables that indicate for given categories of packets what the next hop in the route should be. In addition to simple forwarding of a packet, the

The SDN data plane, referred to as the resource layer in ITU-T Y.3300 and also often referred to as the infrastructure layer, is where network forwarding devices perform the transport and processing of data according to decisions made by the SDN control plane. The important characteristic of the network devices in an SDN network is that these devices perform a simple forwarding function, without embedded software to make autonomous decisions.

Data Plane Functions

Figure 4.2 illustrates the functions performed by the data plane network devices (also called data plane network elements or switches). The principal functions of the network device are the following:

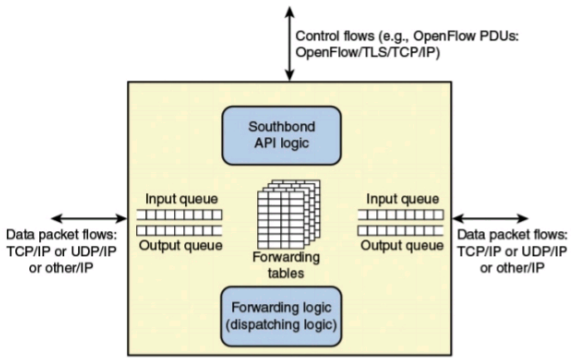


FIGURE 4.2 Data Plane Network Device

- **Control support function:** Interacts with the SDN control layer to support programmability via resource-control interfaces. The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.
- **Data forwarding function:** Accepts incoming data flows from other network devices and end systems and forwards them along the data forwarding paths that have been computed and established according to the rules defined by the SDN applications.

These forwarding rules used by the network device are embodied in forwarding tables that indicate for given categories of packets what the next hop in the route should be. In addition to simple forwarding of a packet, the network device can alter the packet header before forwarding, or discard the packet. As shown, arriving packets may be placed in an input queue, awaiting processing by the network device, and forwarded packets are generally placed in an output queue, awaiting transmission.

The network device in Figure 4.2 is shown with three I/O ports: one providing control communication with an SDN controller, and two for the input and output of data packets. This is a simple example. The network device may have multiple ports to communicate with multiple SDN controllers, and may have more than two I/O ports for packet flows into and out of the device.

Data Plane Protocols

Figure 4.2 suggests the protocols supported by the network device. Data packet flows consist of streams of IP packets. It may be necessary for the forwarding table to define entries based on fields in upper-level protocol headers, such as TCP, UDP, or some other transport or application protocol. The network device examines the IP header and possibly other headers in each packet and makes a forwarding decision.

The other important flow of traffic is via the southbound application programming interface (API), consisting of OpenFlow protocol data units (PDUs) or some similar southbound API protocol traffic.

4.2 OpenFlow Logical Network Device

To turn the concept of SDN into practical implementation, two requirements must be met:

- There must be a common logical architecture in all switches, routers, and other network devices to be managed by an SDN controller. This logical architecture may be implemented in different ways on different vendor equipment and in different types of network devices, as long as the SDN controller sees a uniform logical switch functionality.
- A standard, secure protocol is needed between the SDN controller and

OpenDaylight Architecture

Figure 5.7 provides a top-level view of the OpenDaylight architecture. It consists of five logical layers, as further described in the list that follows.

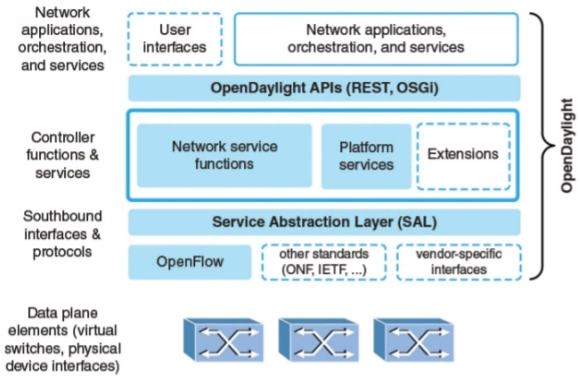


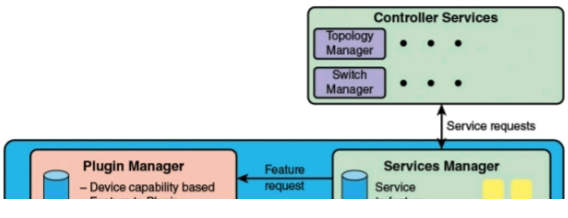
FIGURE 5.7 OpenDaylight Architecture

- **Network applications, orchestration, and services:** Consists of business and network logic applications that control and monitor network behavior. These applications use the controller to gather network intelligence, run algorithms to perform analytics, and then use the controller to orchestrate the new rules, if any, throughout the network.
- **APIs:** A set of common interfaces to OpenDaylight controller functions. OpenDaylight supports the Open Service Gateway Initiative (OSGi) framework and bidirectional REST for the northbound API. The OSGi framework is used for applications that will run in the same address space as the controller, while the REST (web-based) API is used for applications that do not run in the same address space (or even necessarily on the same machine) as the controller.
- **Controller functions and services:** SDN control plane functions and services.
- **Service abstraction layer (SAL):** Provides a uniform view of data plane resources, so that control plane functions can be implemented independent of the specific southbound interface and protocol.
- **Southbound interfaces and protocols:** Supports OpenFlow, other standard southbound protocols, and vendor-specific interfaces.

There are several noteworthy aspects to the OpenDaylight architecture. First, OpenDaylight encompasses both control plane and application plane functionality. Thus, OpenDaylight is more than just an SDN controller implementation. This enables enterprise and telecommunications network managers to host open source software on their own servers to construct an SDN configuration. Vendors can use this software to create products with value-added additional application plane functions and services.

A second significant aspect of the OpenDaylight design is that it is not tied to OpenFlow or any other specific southbound interface. This provides greater flexibility in constructing SDN network configurations. The key element in this design is the SAL, which enables the controller to support multiple protocols on the southbound interface and provide consistent services for controller functions and for SDN applications. Figure 5.8 illustrates the operation of the SAL. The OSGi framework provides for dynamically linking

plug-ins for the available southbound protocols. The capabilities of these protocols are abstracted to a collection of features that can be invoked by control plane services via a services manager in the SAL. The services manager maintains a registry that maps service requests to feature requests. Based on the service request, the SAL maps to the appropriate plug-in and thus uses the most appropriate southbound protocol to interact with a given network device.



plane resources, so that control plane functions can be implemented independent of the specific southbound interface and protocol.

standard southbound protocols, and vendor-specific interfaces.

There are several noteworthy aspects to the OpenDaylight architecture. First, OpenDaylight encompasses both control plane and application plane functionality. Thus, OpenDaylight is more than just an SDN controller implementation. This enables enterprise and telecommunications network managers to host open source software on their own servers to construct an SDN configuration. Vendors can use this software to create products with value-added additional application plane functions and services.

A second significant aspect of the OpenDaylight design is that it is not tied to OpenFlow or any other specific southbound interface. This provides greater flexibility in constructing SDN network configurations. The key element in this design is the SAL, which enables the controller to support multiple protocols on the southbound interface and provide consistent services for controller functions and for SDN applications. Figure 5.8 illustrates the operation of the SAL. The OSGi framework provides for dynamically linking

plug-ins for the available southbound protocols. The capabilities of these protocols are abstracted to a collection of features that can be invoked by control plane services via a services manager in the SAL. The services manager maintains a registry that maps service requests to feature requests. Based on the service request, the SAL maps to the appropriate plug-in and thus uses the most appropriate southbound protocol to interact with a given network device.

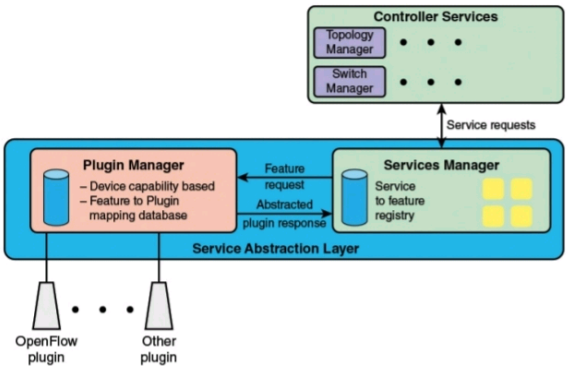


FIGURE 5.8 Service Abstraction Layer Model

The emphasis in the OpenDaylight project is that the software suite be modular, pluggable, and flexible. All of the code is implemented in Java and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

OpenDaylight Helium

At the time of this writing, the most recent release of OpenDaylight is the Helium release, illustrated in Figure 5.9. The controller platform (exclusive of applications, which may also run on the controller) consists of a growing

collection of dynamically pluggable modules, each of which performs one or more SDN-related functions and services. Five modules are considered base network service functions, likely to be included in any OpenDaylight implementation, as described in the list that follows.